การประมวลผลแบบกริด

GRID COMPUTING

นาย มรกต หงษ์วิทยากร
นางสาว มัทนา บุญรอด
นาย วรพงศ์ สมงาม

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2547

การประมวลผลแบบกริด

GRID COMPUTING

โดย

นาย มรกต หงษ์วิทยากร

นางสาว มัทนา บุญรอด

นาย วรพงศ์ สมงาม

อาจารย์ที่ปรึกษา

ดร.วรวัฒน์ ลิ้มโภคา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

ปริญญานิพนธ์ ปีการศึกษา 2547

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การประมวลผลแบบกริด

GRID COMPUTING

คณะผู้จัดทำ	นาย มรกต หงษ์วิทยากร	รหัส 44010374
	นางสาว มัทนา บุญรอด	รหัส 44010375
	นาย วรพงศ์ สมงาม	รหัส 44010411

..... อาจารย์ที่ปรึกษา

(ดร.วรวัฒน์ ลิ้มโภคา)

การประมวลผลแบบกริด

นาย มรกต หงษ์วิทยากร 44010374
นางสาว มัทนา บุญรอด 44010375
นาย วรพงศ์ สมงาม 44010411
ดร.วรวัฒน์ ลิ้มโภคา อาจารย์ที่ปรึกษา
ปีการศึกษา 2547

บทคัดย่อ

โครงงานนี้เป็นการศึกษาและพัฒนาระบบกริด ซึ่งมีลักษณะเป็นระบบที่มีการทำงานแบบกระจาย โดยสามารถแจกจ่ายงานไปยังเครื่องคอมพิวเตอร์อื่นๆที่อยู่ในระบบ ซึ่งอาจเป็นเครื่องคอมพิวเตอร์ส่วนบุคคลธรรมดา ให้สามารถประมวลผลร่วมกันได้ ทำให้สามารถเพิ่มประสิทธิภาพของคอมพิวเตอร์ โดยใช้เทคโนโลยีของการติดต่อผ่านเครือข่ายเน็ตเวิร์กเข้ามาช่วย ทำให้สามารถนอกจากนี้ยังได้พัฒนาโปรแกรมให้สามารถใช้งานในระบบที่สร้างขึ้นได้ โดยเลือกพัฒนาโปรแกรมการคำนวณเมตริกซึ่งง่ายต่อการทำความเข้าใจ มีโครงสร้างที่สามารถแจกแจงออกเป็นงานย่อยและสามารถกระจายไปทำยังเครื่องอื่นๆได้ อีกทั้งยังสามารถใช้เป็นพื้นฐานของการพัฒนาโปรแกรมที่มีความซับซ้อนมากขึ้นได้อีกด้วย

GRID PROCESSING

Mr. Morakot Hongvittayakorn 44010374
Miss. Matthana Boonrawd 44010375
Mr. Worapong Somngam 44010411
Dr. Woravat Limpoka Advisor
Academic Year 2004

ABSTACT

This project is for studying and developing grid system which working as the distributed system that can distribute the processes to others computers which can be personal computers in the system This will increase the efficient of computers by using connecting through network technology. We also developed program that can work in this system by choosing the matrix processing program because it is easy to understand, have a distributable structure and can distribute works to others computers. This also use as the basic for developing the program that has more complex in the future.

กิตติกรรมประกาศ

ปริญญานิพนธ์นี้สำเร็จลุล่วงได้อย่างดีด้วยคำแนะนำ คำปรึกษาและคอยดูแลจากหลายๆฝ่ายด้วยกัน โดยเฉพาะอาจารย์ที่ปรึกษาที่ให้โอกาสให้ข้าพเจ้าได้ทำปริญญานิพนธ์ฉบับนี้ คอยให้ความเอาใจใส่ แนะนำ และให้ความช่วยเหลือเสมอมา คือ ดร. วรวัฒน์ ลิ้มโภคา ซึ่งต้องขอขอบพระคุณเป็นอย่างสูง

ขอขอบคุณภาควิชาวิศวกรรมคอมพิวเตอร์ที่ได้จัดเตรียมสิ่งอำนวยความสะดวก เพื่อให้การวิจัยและพัฒนาระบบเป็นไปได้ด้วยความสะดวกและรวดเร็ว รวมทั้งยังมีอินเทอร์เน็ตความเร็วสูงให้บริการ สำหรับการค้นคว้าหาข้อมูลต่างๆ ซึ่งท้ายที่สุดแล้วจึงประกอบเป็นส่วนหนึ่งของโครงงานนี้

ขอขอบคุณ พี่ๆ เพื่อนๆ น้องๆ ในห้องปฏิบัติการโอลาล้าที่คอยให้คำปรึกษาส่วนหนึ่งและเป็นเพื่อนให้ความครื้นเครงทั้งกลางวันและกลางคืน อีกทั้งยังเป็นที่อาศัย พักผ่อนอีกด้วย

และสุดท้ายนี้ต้องขอขอบคุณ บิดา มารดาและบุคคลในครอบครัว อันเป็นที่เคารพรัก คอยสั่งสอนข้าพเจ้ามาเป็นอย่างดี พร้อมให้โอกาสในการศึกษาอย่างเต็มที่ และยังให้กำลังใจเสมอมา

นาย มรกต หงษ์วิทยากร
นางสาว มัทนา บุญรอด
นาย วรพงศ์ สมงาม

สารบัญ

หน้าที่

สารบัญตาราง

หน้าที่

สารบัญรูปภาพ

หน้าที่

บทที่ 1
บทนำ

1.1 หลักการและเหตุผล

ปัจจุบันเทคโนโลยีด้านการสื่อสารผ่านระบบเครือข่ายเน็ตเวิร์คได้พัฒนาขึ้นกว่าเดิมมาก รวมถึงความสามารถของคอมพิวเตอร์ส่วนบุคคลได้พัฒนาไปอย่างรวดเร็วทั้งในเรื่องของความเร็วและจำนวนพื้นที่ในการเก็บข้อมูล จึงเป็นโอกาสที่ดีที่จะนำคอมพิวเตอร์เหล่านี้มาใช้ประโยชน์แทนที่ ซุปเปอร์คอมพิวเตอร์(super computer) ที่มีราคาสูงและยากแก่การบำรุงรักษา นักวิชาการจึงพัฒนาเทคโนโลยีใหม่ที่เรียกว่า "กริด (Grid)" ขึ้นมาเพื่อใช้ในการประมวลผลงานใหญ่ๆ ได้นำเอาข้อดีของคลัสเตอร์ และ Peer-to-Peer รวมเข้าไว้ด้วยกัน กล่าวคือ คอมพิวเตอร์แต่ละตัวสามารถร่วมกันประมวลผลงานชิ้นเดียวกัน ซึ่งคล้ายกับคลัสเตอร์ แต่ไม่ต้องจำเป็นต้องมีศูนย์กลางที่ใช้ในการแจกจ่ายงาน ซึ่งคล้ายกับ Peer-to-Peer ทำให้สามารถใช้งานได้อย่างยืดหยุ่น

จากข้อดีของระบบกริดที่กล่าวมา ทำให้คณะผู้จัดทำโครงงานเกิดแนวคิดที่จะออกแบบและสร้างระบบกริดขึ้นมาใช้งาน เพื่อเรียนรู้และทดสอบความสามารถของระบบกริด โดยเลือกใช้ Globus Toolkit 3.2.1 เป็น middleware ในการพัฒนาระบบ และทดลองพัฒนาโปรแกรมที่สามารถใช้งานบนกริดได้ โดยเลือกพัฒนาโปรแกรมการคำนวณเมตริก เนื่องจากสามารถทำความเข้าใจการทำงานได้ง่าย สามารถออกแบบรูปแบบของการกระจายงานได้ และสามารถนำไปเป็นพื้นฐานของการพัฒนาโปรแกรมอื่นๆที่มีความซับซ้อนมากๆได้

1.2 วัตถุประสงค์ของโครงงาน

1. เพื่อศึกษาแนวคิดและกระบวนการทำงานของกริด
2. ศึกษาการสร้างระบบในการใช้งานจริง
3. ศึกษาเครื่องมือที่จะนำมาใช้เป็น middleware
4. ศึกษาการเขียนโปรแกรมที่สามารถใช้งานในระบบกริดได้
5. ทดลองสร้างระบบกริดให้สามารถ
6. ทำการทดลอง และศึกษาถึงประสิทธิภาพการทำงาน วัดผล และเปรียบเทียบระหว่างการทำงานบนเครื่องเดียว กับการทำงานแบบกระจาย

1.3 ขอบเขตของการพัฒนา

1. สร้างระบบกริดโดยใช้ Globus Toolkit 3.2.1 บนระบบปฏิบัติการ Linux Redhat 9
2. สามารถเขียนโปรแกรมการคำนวณเมตริกให้สามารถใช้งานได้บนระบบกริดที่สร้างขึ้น
3. สามารถเขียนโปรแกรมการคำนวณเมตริกให้สามารถทำงานแบบกระจายได้

ในการพัฒนาโครงงานชิ้นนี้ได้มีการศึกษาถึงการออกแบบและทดลองสร้างระบบกริดให้สามารถใช้งานได้จริง รวมถึงสามารถพัฒนาในส่วนของการเขียนโปรแกรมให้สามารถใช้งานบนระบบที่สร้างขึ้นได้ โดย

ไม่ได้เน้นในส่วนของการรักษาความปลอดภัยของระบบ โดยจะใช้เพียงแค่ระบบการรักษาความปลอดภัยเท่าที่ Globus Toolkit มีมาให้เท่านั้น ซึ่งถ้าผู้สนใจจะนำระบบที่ผู้จัดทำไปใช้งานจริงอาจเจอปัญหาในเรื่องของความปลอดภัยบ้าง แต่ก็สามารถเพิ่มเติมส่วนของระบบความปลอดภัยเพิ่มเข้าไปได้ ซึ่งในส่วนนี้ทางคณะผู้จัดทำจะไม่กล่าวถึง

1.4 วิธีการดำเนินงาน

1. ศึกษาแนวคิดและรูปแบบของการใช้งานกริดในปัจจุบัน
2. ศึกษาเทคโนโลยีที่เกี่ยวข้องในการสร้างระบบกริด
3. ทดลองสร้างระบบกริด และเรียนรู้วิธีการใช้งาน
4. ศึกษารูปแบบของโปรแกรมที่ใช้งานบนกริด
5. ศึกษาวิธีการเขียนโปรแกรมที่สามารถใช้งานบนกริดได้
6. ออกแบบโปรแกรมการคำนวณเมตริก ให้สามารถใช้งานในระบบกริดที่สร้างขึ้นได้
7. ทดสอบและวิเคราห์ผลที่ได้จากการทดลองใช้งานโปรแกรมที่เขียนขึ้น

บทที่ 2
ระบบการประมวลผลแบบกริด

2.1 ความเป็นมาของระบบการประมวลผลแบบกริด

ระบบการประมวลผลแบบกริด กำลังดำเนินรอยตาม SETI@home, Beowulf, Condor และ Unicore เพื่อสร้างรีซอร์สที่ใช้ในการประมลผลขนาดใหญ่ โดยอาศัยหน่วยประมวลผล และ เครือข่ายสำหรับส่งข้อมูล มาช่วยทำให้เครือข่ายกลายเป็นเครื่องคอมพิวเตอร์ขนาดยักษ์เพียงเครื่องเดียว ที่ทุกคนสามารถเข้ามาใช้งานได้ นักวิจัยหวังว่าในท้ายที่สุดผู้ใช้งานที่เชื่อมเข้ามายังกริด ไม่จำเป็นต้องทราบเลยว่าข้อมูลจะถูกประมวลผลที่ไหนและอย่างไร คล้ายกับระบบไฟฟ้าที่เราใช้ภายในบ้าน ซึ่งเราก็ไม่จำเป็นต้องรู้ว่าไฟฟ้ามาจากแหล่งจ่ายที่ไหน

การประมวลผลแบบกริดได้รับความสนใจอย่างมากในบริษัทที่ทำธุรกิจเกี่ยวกับ IT แม้ว่าจะนำไปใช้ในด้านการศึกษาหรือวิทยาศาสตร์เป็นส่วนใหญ่ก็ตาม ทำให้เกิดมาตรฐานและเครื่องมือที่ใช้สร้างระบบกริดเพิ่มมากขึ้น และ ผลิตภัณฑ์ที่สร้างขึ้นเหล่านี้ ได้ถูกนำมาใช้ในวงการธุรกิจเพื่ออาศัยข้อดีต่างๆของการประมวลผลแบบกริดมาใช้ให้เป็นประโยชน์อย่างสูงสุด เครื่องมือหรือมิดเดิ้ลแวร์ที่เป็นที่นิยมมากที่สุดเพื่อใช้ในการสร้างและจัดการเกี่ยวกับกริดก็คือ Globus Toolkit

โดยสรุปคือ ระบบกริด เป็นกลุ่มของทรัพยากรที่ใช้คำนวณที่กระจายอยู่ตามที่ต่างๆบนระบบเครือข่าย ไม่ว่าจะเป็นแบบภายในหรือภายนอก ที่แสดงให้ผู้ใช้งานเห็นเสมือนเป็น ระบบการคำนวณขนาดใหญ่ 1 ระบบ โดยมีจุดประสงค์ที่จะสร้างระบบที่ใช้คำนวณที่ปลอดภัย มีการใช้งานทรัพยากรร่วมกัน ซึ่งการคำนวณแบบกริดนั้นพยายามที่จะเข้าใกล้การคำนวณแบบกระจายที่รวบรวมทรัพยากรต่างๆเพื่อให้ผู้ใช้เข้าถึงได้อย่างไม่จำกัด

2.2 แรงจูงใจในการพัฒนากริด

เมื่อพูดถึงแรงจูงใจในการพัฒนากริดแล้ว จะขอยกตัวอย่างจาก การพยากรณ์อากาศ ซึ่งต้องใช้ข้อมูลจำนวนมหาศาลจากทั่วโลกในการทำแบบจำลองให้แม่นยำ

ข้อมูลจะต้องถูกป้อนเข้าอยู่ตลอดเวลาเนื่องจากสภาพอากาศก็มีการเปลี่ยนแปลงอยู่เสมอ ดังนั้นเพื่อให้ครอบคลุมสภาพอากาศทั้งหมด จึงต้องมีเครือข่ายเซ็นเซอร์วางกระจายอยู่เป็นบริเวณกว้าง ด้วยเหตุนี้ จึงต้องใช้นักวิทยาศาสตร์หลายคนช่วยกันจัดการระบบ และจากการที่ขั้นตอนการคำนวณรวมถึงข้อมูลที่ใช้ มีความซับซ้อนมากยิ่งขึ้น มีขนาดใหญ่ขึ้น กระจายอยู่บริเวณต่างๆ และต้องถูกใช้งานจากคนหลายๆคน ดังนั้นจึงเกิดความต้องการระบบที่สามารถปฏิบัติการ ในแต่ละตำแหน่งที่เซ็นเซอร์กระจายตัวอยู่ผ่านทางเครือข่ายได้

เมื่อเวลาผ่านไป คนเราก็มีความรู้ความเข้าใจ สติปัญญา ความสามารถเพิ่มมากขึ้น นำไปสู่การพัฒนาทุกๆขั้นตอนในการพยากรณ์อากาศ ในขณะที่

- เทคโนโลยีหลายๆแขนงมีการพัฒนาไปมาก ทำให้นักวิทยาศาสตร์ต้องประสานงานกัน
- ความเร็วของคอมพิวเตอร์และระบบเครือข่ายสูงขึ้นในขณะที่มีราคาถูกลง

- ซุปเปอร์คอมพิวเตอร์มีราคาแพง สามารถใช้งานและควบคุมได้ในแผนกงานเดียว
- เมื่อมีระบบเครือข่ายเกิดขึ้นมา ทำให้คอมพิวเตอร์สามารถติดต่อสื่อสารกันได้
- เมื่อมีระบบอินเตอร์เน็ตเกิดขึ้นมา ทำให้แต่ละองค์กรสามารถติดต่อสื่อสารกันได้

ด้วยปัจจัยต่างๆเหล่านี้ทำให้เกิดการเปลี่ยนแปลงจากต้องมีศูนย์กลางการคำนวณ เป็นระบบที่ไม่จำเป็นต้องมีศูนย์กลาง ทำให้สามารถรวบรวมทรัพยากรที่กระจายอยู่ทั่วโลกมาใช้ประโยชน์ได้ แม้ว่าทรัพยากรเหล่านั้นจะอยู่ในคอมพิวเตอร์ที่มีโครงสร้างแตกต่างกัน

2.3 ประเภทของระบบกริด

การประมวลผลแบบกริด สามารถนำไปใช้ประโยชน์ได้หลายทางตามความต้องการของแอพพลิเคชั่นที่ผู้ใช้เรียกใช้ ซึ่งสามารถแบ่งกริด ออกเป็น 3 ประเภทตามวิธีการแก้ปัญหา

2.3.1 Computational Grid

Computational Grid จะเน้นไปที่การกำหนดทรัพยากร ที่จะใช้ในการคำนวณโดยเฉพาะเจาะจง ซึ่งกริด ประเภทนี้มักจะเป็นพวกเครื่องที่มีสมรรถนะสูงที่ใช้ในการคำนวณด้านต่างๆ

2.3.2 Scavenging Grid

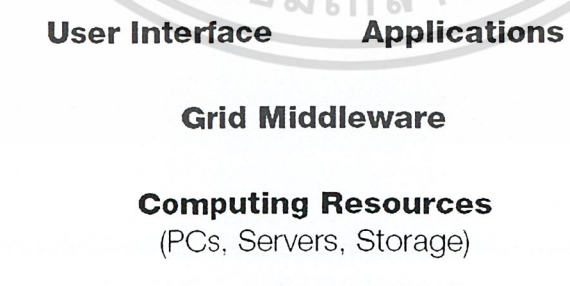
Scavenging Grid เป็นกริดที่สร้างขึ้นด้วยการประกอบกันของ เครื่องคอมพิวเตอร์ (desktop) จำนวนมาก โดยแต่ละเครื่องจะใช้ CPU cycle และ ทรัพยากรที่มีอยู่ ซึ่งจะมีเครื่องคอมพิวเตอร์ 1 เครื่องที่ทำหน้าที่ควบคุมทรัพยากรที่มีอยู่ในกริดให้กับเครื่องคอมพิวเตอร์เครื่องอื่นๆที่ทำงานร่วมกัน

2.3.3 Data Grid

Data Grid เป็นกริดที่รับผิดชอบ ภาระหน้าที่ที่เกี่ยวกับการเข้าถึงข้อมูลที่ผ่านการรวบรวมหลายรูปแบบ ทำให้ผู้ใช้ไม่ต้องกังวลเกี่ยวกับการเข้าถึงข้อมูลว่าข้อมูลที่ต้องการนั้นอยู่ที่จุดใด Data Grid จะช่วยให้การเข้าถึงข้อมูลให้ทำได้ง่ายขึ้น รวมถึงยังทำหน้าที่จัดการเรื่องความปลอดภัยของข้อมูลด้วย

2.4 การสร้างระบบกริด

ในการสร้างระบบกริดขึ้นมานั้น จำเป็นต้องมีโครงสร้าง ดังต่อไปนี้



รูปที่ 2-1 โครงสร้างของระบบการประมวลผลแบบกริด

โครงสร้างของกริดนั้น จะประกอบไปด้วย

- User Interface & Application เป็นแอพพลิเคชั่นที่ติดต่อกับผู้ใช้งานระบบกริด เช่น ส่วนที่ใช้จัดการควบคุมดูแลกริด ส่วนที่ใช้จัดหาเซอร์วิส เป็นต้น
- Grid Middleware เป็นเครื่องมือที่ช่วยในการสร้างระบบกริด เปรียบเสมือนส่วนที่เชื่อมต่อระหว่างผู้ใช้งานกับทรัพยากรที่ใช้งานในกริด มีหลายเครื่องมือที่ใช้เป็นมิดเดิ้ลแวร์ เช่น Globus Toolkit, Rocks Cluster & Roll Grid เป็นต้น
- Computing Resources เป็นทรัพยากรที่ใช้ในการคำนวณ ประกอบไปด้วย คอมพิวเตอร์ แหล่งเก็บข้อมูล แหล่งประมวลผลต่างๆ เป็นต้น
- Grid Network ระบบเครือข่ายที่ใช้ในกริด ซึ่งรวมถึงระบบเครือข่ายภายใน (Local Area Network) และรวมถึงระบบเครือข่ายภายใน (Wide Area Network)

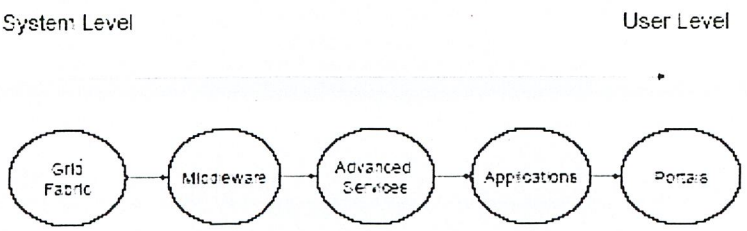
ส่วนที่สำคัญที่สุดที่จะต้องกล่าวถึงเป็นพิเศษก็คือ มิดเดิ้ลแวร์ เนื่องจากเป็นเสมือนส่วนที่ทำให้ระบบกริด สามารถใช้งานได้จริง ซึ่งมิดเดิ้ลแวร์ที่นิยมใช้ คือ Globus Toolkit ซึ่งมีข้อดีก็คือ ได้นำเทคโนโลยีเว็บเซอร์วิสมาผนวกรวมเข้ากับมาตรฐานที่ใช้กำหนดระบบกริด (Open Grid Service Architecture หรือ OGSA) ซึ่งจะกล่าวถึงในบทต่อๆไป และยังเป็นเครื่องมือที่ผู้คิดค้นกริดที่รวมตัวเป็นกลุ่มๆหนึ่ง ที่เรียกว่า Global Grid Forum หรือ GFF พัฒนาขึ้นมาโดยตรงด้วย

ข้อดีของการนำกริดมาอ้างอิงบนพื้นฐานของ เว็บเซอร์วิส คือมันจะทำให้ได้เฟรมเวิร์ค ที่สามารถขยายได้เพื่อการเข้าถึง ทรัพยากรของกริด โดยใช้มาตรฐานต่างๆที่มีอยู่แล้ว เช่น SOAP, XML และ WS-Security, TCP/IP และ มาตรฐาน IETF อื่นๆ ทำให้สามารถติดต่อสื่อสารแบบ interoperable ท่ามกลางระบบที่แตกต่างกันได้ในอินเตอร์เน็ตและเครือข่ายที่ใช้งาน

จากการพิจารณาผลงานวิจัยของกริดแสดงให้เห็นว่า มีสิ่งที่แตกต่างกันได้ชัดเจนให้สังเกตได้ 3 สิ่ง กริดแสดงให้เห็นลักษณะที่แตกต่างของทางสถาปัตยกรรมที่แสดงให้เห็นทิศทางที่ควรจะเป็นไปของกริด และพบว่าสถาปัตยกรรมของกริดเป็นลักษณะของ Multi-faceted แบ่งได้เป็น 3 ประเภทคือ

2.4.1 สถาปัตยกรรมกริดแบบลำดับชั้น (N-Tier Grid Architecture)

สถาปัตยกรรมกริดแบบลำดับชั้น เป็นลักษณะของแบบจำลองเพื่อให้การพัฒนากริดได้รับความยืดหยุ่นและสามารถ นำแอพพลิเคชั่นที่เคยสร้างไว้กลับมาพัฒนา และใช้งานซ้ำได้ การแบ่งแยกกริดออกเป็นชั้นๆ ทำให้ผู้พัฒนาสามารถทำการเปลี่ยนแปลง และเพิ่มเลเยอร์ที่ต้องการได้ จึงทำให้นักพัฒนาเลือกเพิ่มเติมหรือแก้ไขได้อย่างจำเพาะเจาะจง แทนที่จะต้องทำการสร้างแอพพลิเคชั่นขึ้นมาใหม่ ซึ่งเปรียบได้กับ ชั้นที่ 7 ของ OSI model จึงเป็นที่นิยมใช้กันโดยทั่วไป

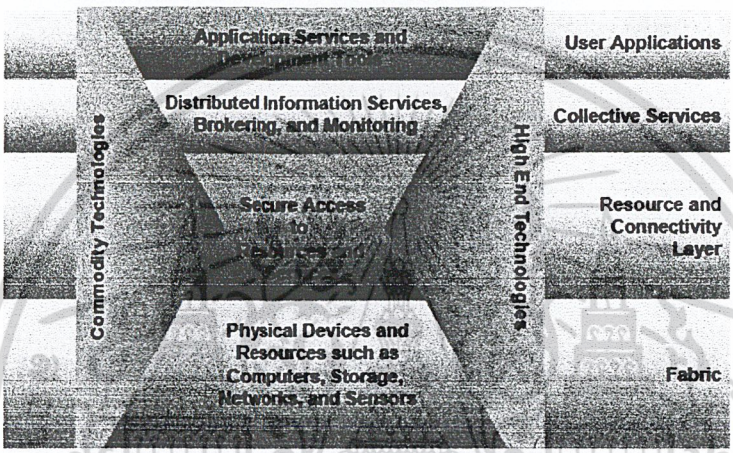


รูปที่ 2-2 รูปสถาปัตยกรรมกริดแบบลำดับชั้น

2.4.2 สถาปัตยกรรมกริดที่ขึ้นกับหน้าที่ (Role-Based Grid Architecture)

การเข้าถึงกลุ่มของทรัพยากรทางกายภาพได้อย่างปลอดภัย จะต้องใช้ สถาปัตยกรรมแบบที่ขึ้นกับหน้าที่ ซึ่งสถาปัตยกรรมแบบนี้ ง่ายที่จะบ่งบอกส่วนประกอบต่างๆของระบบ ที่ทำการติดต่อกับภายนอกซึ่งเปรียบเสมือนเป็นโปรโตคอลที่แยกออกเป็นประเภทๆตามหน้าที่ โดยร่วมใช้ข้อมูลชุดเดียวกัน

สถาปัตยกรรมแบบนี้แบ่งออกได้เป็น 5 ชั้น คือ fabric, connectivity, resource, collective, and application layer และใช้มาตรฐานเหล่านี้เพื่อช่วยให้การแลกเปลี่ยนข้อมูลเป็นไปด้วยความปลอดภัย และเสมือนเป็นทรัพยากรจากแหล่งเดียวกัน ทรัพยากรเหล่านี้ถูกควบคุมและดูแลโดย collective services ซึ่งทำให้มองเสมือนทรัพยากรเหล่านั้นคือทรัพยากรจากแหล่งเดียวกันเพื่อให้ง่ายต่อการออกแบบแอพพลิเคชั่นและง่ายสำหรับการเรียกใช้ของผู้ใช้งาน



รูปที่ 2-3 รูปสถาปัตยกรรมกริดที่ขึ้นกับหน้าที่

ในสถาปัตยกรรมแบบที่ขึ้นกับหน้าที่แบ่งเป็นเลเยอร์ต่างๆดังนี้

- fabric layer ประกอบไปด้วยโปรโตคอลต่างๆที่ทำการติดต่อกับ แอพพลิเคชั่นและเครื่องมือที่ทำให้เกิดการพัฒนาการบริการต่างๆ รวมไปถึงควบคุมส่วนต่างๆที่ทำการเข้าถึงทรัพยากร เช่น เครื่องคอมพิวเตอร์ , แหล่งเก็บทรัพยากร , ระบบเครือข่าย และ เซ็นเซอร์ เป็นต้น

- connectivity layer เป็นส่วนที่รับผิดชอบหน้าที่ในการติดต่อ และ รับรองการเชื่อมต่อของเครือข่ายเพื่อทำให้เกิดความปลอดภัยในระหว่างที่ทำการติดต่อกับทรัพยากรของกริด เป็นโปรโตคอลและบริการที่มีหน้าที่แลก เปลี่ยนข้อความที่มีความปลอดภัยถึงกัน , การรับรองสิทธิ ,และการให้อำนาจในการติดต่อ

- resource layer ประกอบไปด้วยโปรโตคอลที่ทำหน้าที่ดูแลและสร้างความปลอดภัยให้กับทรัพยากร

- collective layer รับผิดชอบในส่วนของการร่วมกันทำงานของทรัพยากรจากหลายแหล่ง และบ่งบอกว่าอยู่ที่ใดในส่วนของ virtual organization

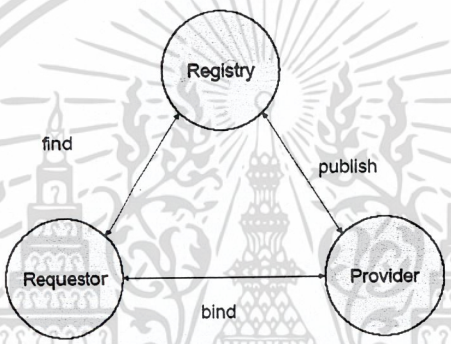
- application layer ทำหน้าที่ติดต่อกับผู้ใช้โดยตรงโดยผ่านแอพพลิเคชั่นซึ่งจะติดต่อผ่านทาง virtual organization อีกทีหนึ่ง

2.4.3 สถาปัตยกรรมกริดที่ขึ้นกับเซอร์วิส (Service-Based Grid Architecture)

ปัจจุบันเราพบว่าแนวทางของเทคโนโลยีจะเน้นไปทางแนวคิดที่แต่ละส่วนของบริการใดๆ เป็นอิสระต่อกัน และภาพรวมของกริดพบว่าลักษณะซอฟท์แวร์ที่เราต้องการคือไม่ขึ้นอยู่กับแพลตฟอร์มใดๆ บริการที่เราต้องการจะสามารถเรียกถึงได้ง่าย ซึ่งกระบวนการนี้เรียกว่า การค้นหารีซอร์ส (resource discovery)

ประโยชน์ของสถาปัตยกรรมกริดที่ขึ้นกับเซอร์วิส เปรียบได้กับกลุ่มของคอมพิวเตอร์ที่ทำงานตามรูปแบบตารางเวลา เริ่มจาก หากลุ่มรีซอร์สที่สามารถใช้งานได้ ต่อมาก็เลือกรีซอร์สที่ใช้คำนวณจากกลุ่มที่ได้ในขั้นแรกเพื่อจัดตารางงาน โดยหลักการเลือกรีซอร์สนั้น อาจเลือกจากอัตราการโหลดของแต่ละรีซอร์ส หรือจากค่าต่างๆ ตามแต่จะพิจารณาว่าสมควร

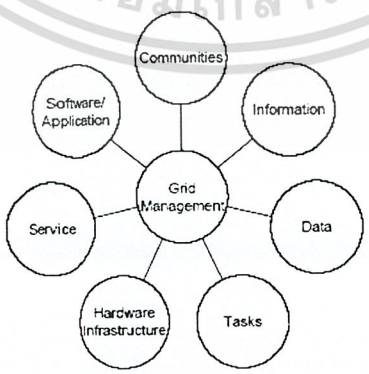
โดยสรุปแล้ว โครงสร้างนี้สามารถพัฒนาเป็นการทำงานที่เชื่อมต่อกันระหว่างเซอร์วิสได้ เพราะรูปแบบที่ขึ้นกับเซอร์วิสนี้ เป็นเซอร์วิสที่ไม่ได้เกิดขึ้นในเวลาเดียวกัน ตามรูปแบบ asynchronous service นั่นเอง



รูปที่ 2-4 รูปสถาปัตยกรรมกริดที่ขึ้นกับเซอร์วิส

2.5 ความท้าทายในการพัฒนากริด

ด้วยเหตุผลที่ว่าเราต้องการให้เกิดการทำงานร่วมกัน แต่ต้องเป็นการทำงานที่มีความปลอดภัยมาก ดังนั้น สิ่งที่กริดต้องการคือความน่าเชื่อถือของระบบการประมวลผล และ อิทธิพลของความไม่น่าเชื่อถือว่ามีมากน้อยอย่างไร รวมถึงการเข้าถึงข้อมูลต่างๆของระบบ ดังนั้นจะต้องมีการพัฒนาเฟรมเวิร์คและเครื่องมือที่จะทำหน้าที่จัดการสิ่งต่างๆเหล่านี้



รูปที่ 2-5 การจัดการด้านต่างๆที่จำเป็นในกริด

2.6 การจัดการเรื่องความปลอดภัยภายในกริด

จากที่ต้องมีการติดต่อรวบรวมและกระจายทรัพยากรออกไป ทำให้เป้าหมายที่สำคัญอย่างหนึ่งที่กริดต้องมีคือกระบวนการจัดการด้านความปลอดภัยในระบบซึ่งบริการเหล่านั้นได้แก่ Authentication, Authorization, Encryption, และ Nonrepudiation

การรับรองตัวตน (Authentication) ทำหน้าที่เป็นตัวบ่งบอกว่าสามารถเข้าถึงกริดได้หรือไม่ คือทำการแสดงตัวตนของผู้ใช้กริดนั่นเอง

การให้อำนาจ (Authorization) หลังจากที่มีผ่านการรับรองตัวตนเสร็จเรียบร้อยแล้ว ส่วนการให้อำนาจ จะเป็นส่วนที่ทำหน้าที่ควบคุมการกระทำที่ผู้ใช้กริดสามารถกระทำได้ จึงต้องมีการกำหนดนโยบายเพื่อที่จะตัดสินความสามารถของการกระทำใดๆเป็นสิ่งที่ผู้ใช้นั้นๆสามารถทำได้

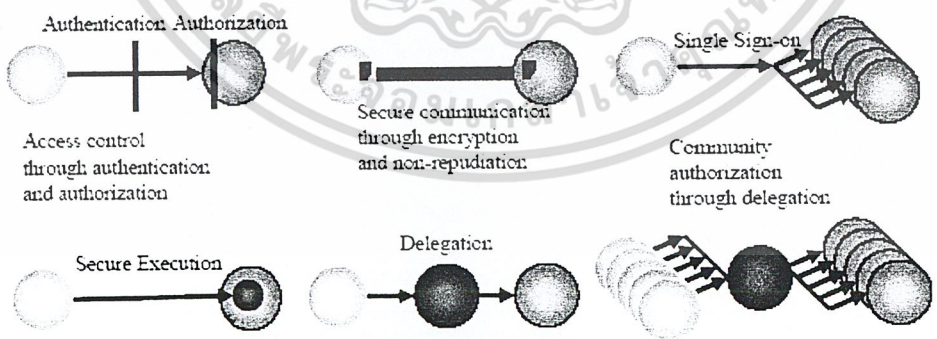
การเข้ารหัส (Encryption) เป็นกลไกของการป้องกันข้อความต่างๆ เพื่อให้ยากลำบากยิ่งขึ้นในการขโมยข้อมูลไประหว่างทาง

Nonrepudiation ทำการรวบรวมข้อมูลเข้าด้วยกัน และพิสูจน์ได้ว่าข้อมูลเหล่านั้นไม่ได้ถูกเปลี่ยนแปลงไปจากเดิมระหว่างที่ทำการส่งข้อมูล นอกจากนั้นสิ่งที่ต้องการด้วยก็คือ รหัสผ่าน (password) ในการเข้าถึง

Single sign-on เป็นกลไกที่ทำหน้าที่รองรับการเข้าถึงในเรื่องของการรับรองตัวตน เนื่องจากกริดมีทรัพยากรขนาดใหญ่ที่ต้องใช้ร่วมกัน ซึ่งจะใช้ proxy service และ secure proxy เข้ามาช่วยในการรับรองตัวตนเพื่อต้องการทรัพยากรที่ไม่ได้อยู่ในที่เดียวกัน ซึ่งอาจจะมีกลไกของการรักษาความปลอดภัยที่ต่างกันด้วย ดังนั้นกระบวนการที่จะใช้นี้จะต้องสามารถ mapping ความหมายเหล่านั้นได้

Delegation เป็นกระบวนการแสดงตัวแทนซึ่งกัน ซึ่งในการทำ Delegation จะต้องมีความระมัดระวังเป็นพิเศษเพราะว่ามีความเป็นไปได้ที่จะเกิดการทำ delegation ต่อกันไปเรื่อยๆไม่จบ

Community authorization ทำหน้าที่จัดการกลไกของ virtual organization เพื่อที่จะทำการกำหนดนโยบายของกลุ่มของผู้ใช้ ที่สามารถเข้าถึงทรัพยากรเหล่านั้น



รูปที่ 2-6 แนวคิดเกี่ยวกับการจัดการด้านความปลอดภัยในกริด

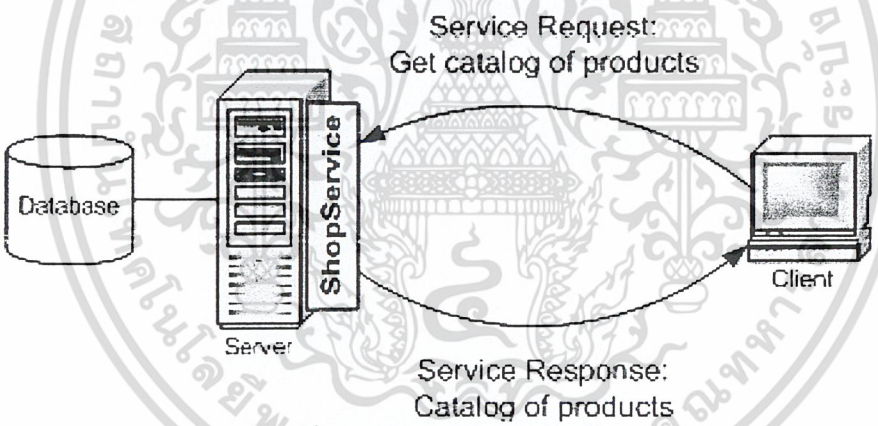
บทที่ 3
กริดเซอร์วิส

ในการคำนวณแบบกริดนั้น เราจะมองทุกๆอย่างในกริดเป็น กริดเซอร์วิส ที่จะประกอบไปด้วยกริดเซอร์วิสต่างๆ ที่ทำงานร่วมกันจนเป็นระบบกริด และเนื่องจากกริดเซอร์วิสนั้น มีพื้นฐานมาจาก เว็บเซอร์วิส ดังนั้น เราจึงต้องมีพื้นฐานความรู้เกี่ยวกับ เว็บเซอร์วิส อย่างน้อยตามที่จะกล่าวให้ทราบในบทนี้

3.1 เว็บเซอร์วิส

เป็นเทคโนโลยีการคำนวณแบบกระจาย ที่ให้เราสร้างแอพพลิเคชั่นใน client/server เช่น สมมติว่า เราต้องพัฒนาแอพพลิเคชั่นสำหรับโซ่คลังสินค้าที่มีอยู่ทั่วประเทศ แต่รายชื่อสินค้าอยู่ในฐานข้อมูลที่สำนักงานกลางเท่านั้น ดังนั้นซอฟท์แวร์ที่คลังสินค้าเหล่านั้นต้องสามารถเข้าถึงรายชื่อสินค้าที่สำนักงานกลางได้ ทำได้โดยผ่านเว็บเซอร์วิสที่ชื่อ ShopService (ไม่ควรให้ใช้บนเว็บไซต์ทั่วไป เพราะทุกคนสามารถเห็นสิ่งเหล่านั้นได้ ซึ่งต่างจากการใช้ เว็บเซอร์วิสเพราะเป็นการเรียกใช้ผ่านซอฟท์แวร์)

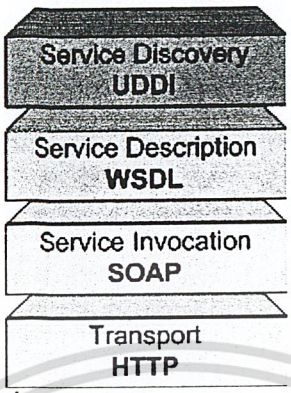
Clients ที่ต้องการใช้ เว็บเซอร์วิส จะส่งการร้องขอไปยัง server เพื่อขอรายการต่างๆ จากนั้น server ก็จะส่งรายการนั้นมาให้เป็นการตอบรับของเซอร์วิส ดังภาพด้านล่าง



รูปที่ 3-1 การตอบรับของเซอร์วิส

ข้อดีของเว็บเซอร์วิส ประการแรกคือ ไม่ขึ้นกับแพลตฟอร์ม และ ไม่ขึ้นกับภาษาที่ใช้เขียน เนื่องจากใช้ภาษามาตรฐาน XML ซึ่งหมายความว่าเว็บเซอร์วิสที่เขียนด้วยภาษา JAVA บน Linux Server สามารถทำงานบนเครื่อง Windows ที่มีโปรแกรมภาษา C++ ประการที่สองคือ เว็บเซอร์วิส เกือบทั้งหมดใช้ HTTP ในการรับ-ส่งข้อมูล (เช่น การร้องขอ และ ตอบกลับของเซอร์วิส) เป็นผลดีต่อการทำแอพพลิเคชั่นที่ใช้งานบนอินเตอร์เน็ต เนื่องจาก HTTP ไม่มีปัญหากับ พร็อกซี่ และ ไฟร์วอล

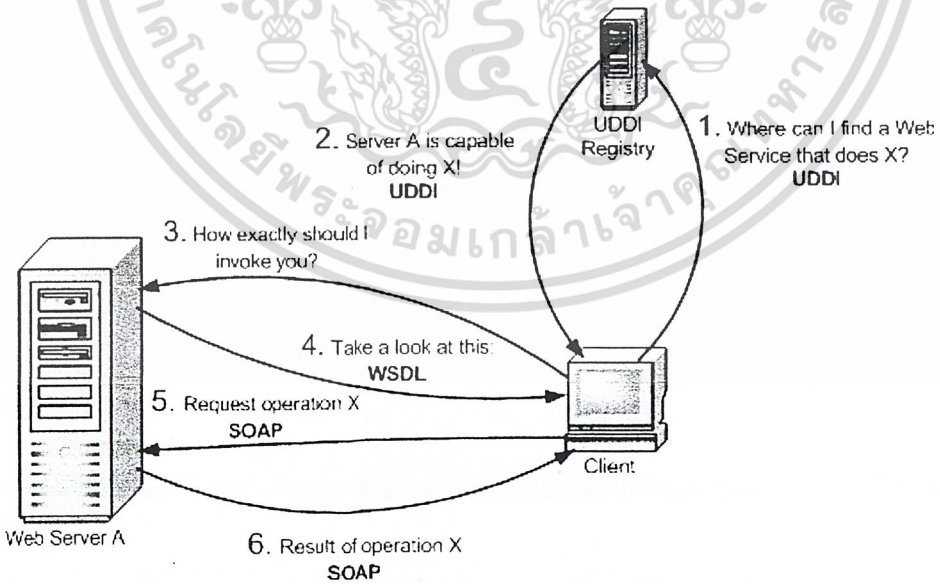
3.1.1 โครงสร้างของเว็บเซอร์วิส



รูปที่ 3-2 โครงสร้างของเว็บเซอร์วิส

- Service Discovery : ในส่วนนี้ช่วยให้เราสามารถรู้ เว็บเซอร์วิส ที่เหมาะกับความต้องการโดยใช้ UDDI
- Service Description : เมื่อรู้ว่าจะติดต่อ เว็บเซอร์วิส ได้ที่ไหนแล้ว เราก็สามารถขอให้ เว็บเซอร์วิส นั้น อธิบายอธิบายตัวเองในแง่ของแอพพลิเคชั่นที่ใช้, คำสั่งที่มี, วิธีเรียกใช้ โดย WSDL
- Service Invocation : การเรียกใช้ เว็บเซอร์วิส รวมไปถึงการส่งข้อความระหว่าง client และ server มักจะใช้ SOAP ในการกำหนดรูปแบบที่ใช้
- Transport : ใช้ HTTP ในการรับ-ส่ง ระหว่าง client กับ server

3.1.2 การเรียกใช้ เว็บเซอร์วิส



รูปที่ 3-3 แสดงลำดับการเรียกใช้เว็บเซอร์วิส

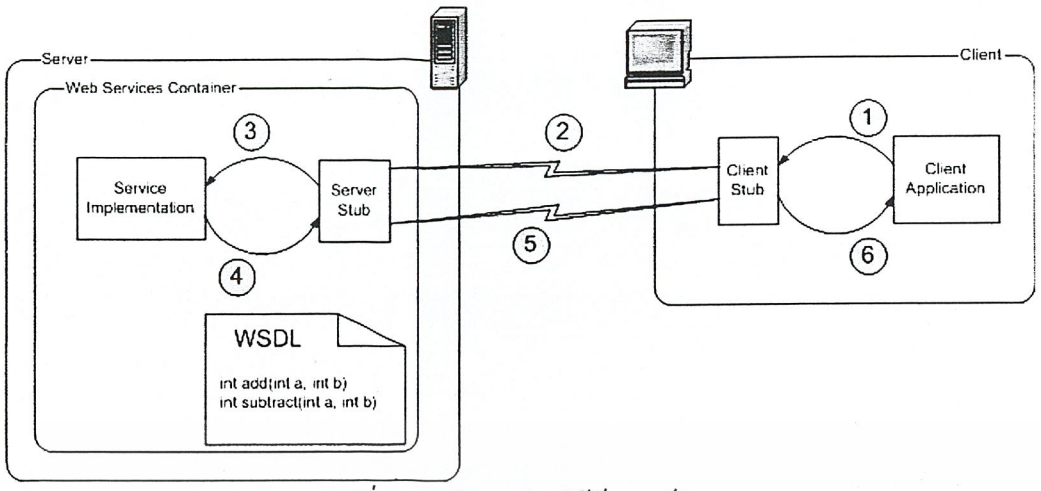
1. client ไม่จำเป็นต้องมีความรู้เกี่ยวกับ เว็บเซอร์วิส ที่ตัวเองกำลังจะเรียกใช้มาก่อน โดยจะเริ่มต้นจากค้นหา เว็บเซอร์วิส ที่เหมาะสมกับความต้องการของเราโดยติดต่อไปยัง UDDI registry
2. UDDI จะตอบกลับมาว่า server ไหนมีเซอร์วิสที่เราต้องการ
3. เมื่อรู้ เว็บเซอร์วิส แล้ว แต่ก็ยังไม่รู้ว่าจะเรียกใช้ยังไง ก็ต้องถามไปยัง เว็บเซอร์วิส นั้นให้อธิบายวิธีเรียกใช้
4. เว็บเซอร์วิส ก็จะส่ง WSDL มาให้
5. เมื่อรู้ เว็บเซอร์วิส และวิธีเรียกใช้ ก็จะเกิดการเรียกใช้ขึ้นเองโดยภาษา SOAP ส่งการร้องขอไปยัง เว็บเซอร์วิส
6. เว็บเซอร์วิส ตอบกลับมาด้วย SOAP response พร้อมสิ่งที่เราต้องการ หรืออาจจะเป็นข้อความแสดงความผิดพลาด ถ้า SOAP ที่ส่งไปนั้นผิด

3.1.3 แอพพลิเคชั่นของเว็บเซอร์วิส

เมื่อ client ต้องการเรียกใช้ เว็บเซอร์วิส เราสามารถสร้าง ทาสก์ (task) แทรกไว้ใน ซอฟท์แวร์เรียกว่า client stub สำหรับเรียกใช้ เว็บเซอร์วิส แทนที่จะเขียน SOAP หรือ WSDL ทีละบรรทัด ซึ่งหลายๆ เครื่องมือในปัจจุบันมีการสร้าง client stub ให้อย่างอัตโนมัติ โดยยึดเอา WSDL เป็นพื้นฐาน ดังนั้น ขั้นตอนที่ถูกต้องในการเรียกใช้ เว็บเซอร์วิส เป็นดังนี้

1. เรารู้ที่ตั้งของ เว็บเซอร์วิส ที่เหมาะสมกับความต้องการผ่าน UDDI
2. เราได้รับ WSDL ของ เว็บเซอร์วิส นั้น
3. เราสร้าง stub ขึ้นมาแทรกไว้ในแอพพลิเคชั่น
4. แอพพลิเคชั่นจะใช้ stub นั้นในแต่ละครั้งที่ต้องการเรียกใช้เว็บเซอร์วิส

เราสามารถสร้างฟังก์ชั่นทั้งหมดของ เว็บเซอร์วิส ขึ้นมา แล้วสร้าง server stub จาก WSDL เพื่อใช้แปลการร้องขอและส่งต่อให้เซอร์วิสที่สร้างขึ้นมา และเมื่อเซอร์วิสนั้นได้ผลลัพธ์ก็จะส่งไปให้ server stub ซึ่งจะทำการสร้าง SOAP response ที่เหมาะสม ในการจัดการกับเซอร์วิสและ server stub จะใช้ ผ่านซอฟท์แวร์ที่เรียกว่า เว็บเซอร์วิสคอนเทนเนอร์ (Web Service container) เพื่อให้แน่ใจว่า HTTP request ที่จะเข้ามายัง เว็บเซอร์วิส ได้เข้ามาที่ server stub ได้อย่างถูกต้อง



รูปที่ 3-4 ลำดับการเรียกใช้เว็บเซอร์วิส

สมมติว่า เรารู้ที่ตั้ง เว็บเซอร์วิส แล้ว และสร้าง client stubs จาก WSDL และ โปรแกรมทางฝั่ง server ก็สร้าง server stubs แล้ว สามารถอธิบายขั้นตอนการเรียกใช้ เว็บเซอร์วิส ได้ดังนี้

1. เมื่อ client ต้องการเรียกใช้ เว็บเซอร์วิส ก็จะทำการเรียก client stub ซึ่งจะทำการแปลงคำขอให้เป็น SOAP request เรียกขั้นตอนนี้ว่า marshaling หรือ serializing process
2. SOAP request ถูกส่งผ่านเครือข่ายโดยใช้ผ่านโปรโตคอล HTTP จากนั้นเว็บเซอร์วิสคอนเทนเนอร์ ก็จะได้รับ SOAP request และส่งให้ server stub ซึ่งจะทำการเปลี่ยน SOAP request ให้อยู่ในรูปที่เซอร์วิสสามารถเข้าใจได้ เรียกขั้นตอนนี้ว่า unmarshaling หรือ deserializing
3. เซอร์วิสได้รับ การร้องขอ จาก server stub และทำงานตามการร้องขอนั้น
4. ผลลัพธ์ที่ได้ถูกส่งไปยัง server stub และถูกแปลงเป็น SOAP response
5. SOAP response ถูกส่งผ่านเครือข่ายผ่านโปรโตคอล HTTP เมื่อ client stub ได้รับก็จะทำการแปลง SOAP response นั้นให้อยู่ในรูปที่แอพพลิเคชั่นเข้าใจได้
6. แอพพลิเคชั่นได้รับผลลัพธ์ที่ต้องการจาก เว็บเซอร์วิส ไปใช้

3.2 กริดเซอร์วิส

กริดเซอร์วิส เป็นส่วนขยายของเว็บเซอร์วิสที่ขยายขึ้นตามมาตรฐานที่กำหนดไว้ใน Open Grid Service Infrastructure เพื่อนำมาใช้งานในการสร้างระบบกริด ในหัวข้อต่อไปนี้กล่าวถึงมาตรฐานที่ใช้กำหนดกริดเซอร์วิส ให้ทราบก่อน

3.2.1 OGSA (Open Grid Service Architecture)

เป็นสิ่งที่กำหนดโดย The Global Grid Forum เพื่อใช้กำหนดมาตรฐานแอพพลิเคชั่นของกริดและเพื่อให้สามารถใช้งานได้จริง เนื่องจาก กริดเซอร์วิส เป็นเทคโนโลยีการคำนวณแบบกระจายภายใต้ OGSA จึงกล่าวได้ว่า OGSA เป็นรากฐานของ กริดเซอร์วิส

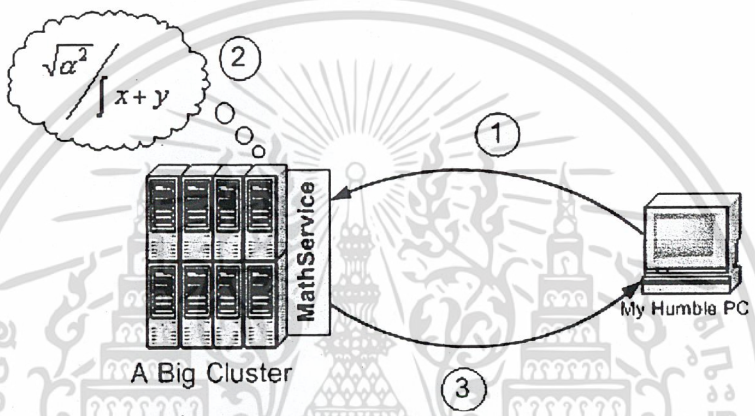
3.2.2 OGSI (Open Grid Service Infrastructure)

เนื่องจาก OGSA เป็นเพียงโครงร่างของ กริดเซอร์วิส แต่ไม่ได้อธิบายรายละเอียดถึงขั้นการใช้งานจริง จึงได้พัฒนาให้มี OGSI ที่ใช้ระบุวิธีการทำงานของ กริดเซอร์วิส โดยละเอียด

3.2.3 คุณสมบัติหลักที่ได้รับการพัฒนาใน OGSI

- Stateful and potentially transient services

เป็นลักษณะสำคัญอย่างหนึ่งที่เกี่ยวข้องกับ เว็บเซอร์วิส ยกตัวอย่างเพื่อความเข้าใจดังนี้ สมมติถ้าเรามีกลุ่มคลัสเตอร์หรือคอมพิวเตอร์ ขนาดใหญ่ที่สามารถประมวลผลงานยากๆได้ ตั้งอยู่ที่สำนักงานในกรุงเทพ และต้องการให้พนักงานที่เชียงใหม่, ขอนแก่น และชลบุรี สามารถใช้งาน คลัสเตอร์นี้ได้อย่างสะดวก

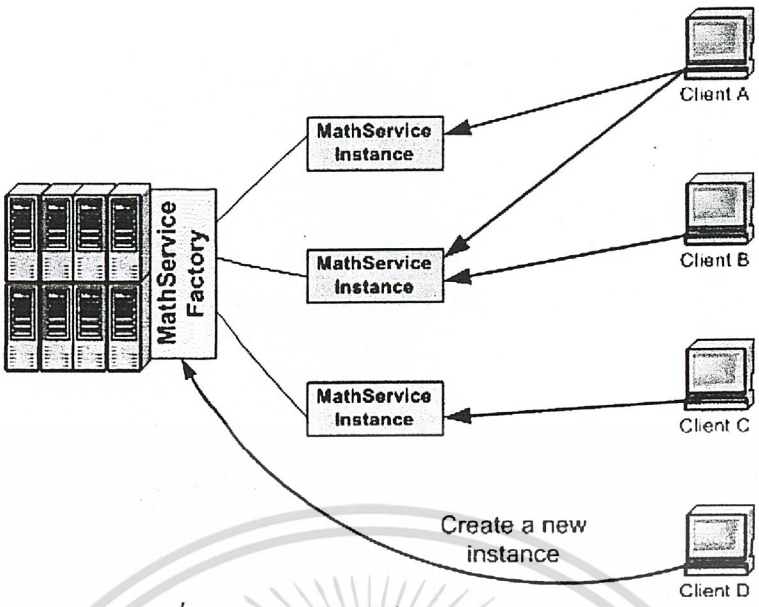


รูปที่ 3-5 ลำดับการทำงานแบบมีสเตทของ OGSI

โดยที่เราอาจจะสร้าง เว็บเซอร์วิสที่ใช้คำนวณทางคณิตศาสตร์ขึ้นมา เรียกว่า MathService ซึ่งมีคำสั่ง SolveReallyBigSystem(), SolveFermatsLastTeorem() และอื่นๆให้ใช้งาน การทำงานของเซอร์วิสนี้จะเป็นตามขั้นตอนดังนี้

1. เรียกใช้ MathService โดยขอใช้คำสั่งต่างๆของเซอร์วิสนั้น
2. MathService จะสั่งให้กลุ่มคลัสเตอร์ทำงานตามคำสั่งที่ขอไว้
3. MathService จะคืนผลลัพธ์ที่ได้กลับมาให้

อย่างไรก็ตาม หากว่ากันตามจริงแล้ว เราอาจต้องการให้คลัสเตอร์ทำงานให้เราหลายคำสั่ง ซึ่งมีความเกี่ยวข้องกันในคราวเดียว ถ้าเป็น เว็บเซอร์วิส ซึ่งเป็น stateless แล้วล่ะก็ จะไม่สามารถจดจำได้ว่าการเรียกใช้แต่ละคำสั่งได้ ว่ามีความสัมพันธ์กันอย่างไร แต่ในกริดเซอร์วิสนั้น จะแก้จุดนั้นโดยใช้แฟคทอรี่ (factory) ที่เมื่อ client ต้องการสร้างหรือลบอินสแตนซ์ (instance) ต้องบอกผ่านแฟคทอรี่



รูปที่ 3-6 แสดงการเรียกใช้ MathService Instance

อินสแตนซ์ มีอายุการใช้งานชั่วคราว เท่าที่ client ต้องการ เพื่อจะได้ไม่ต้องขึ้นกับอายุการใช้งานของกริดเซอร์วิสคอนเทนเนอร์ แต่จะขึ้นอยู่กับแอพพลิเคชั่น แต่ใช่ว่าทุก กริดเซอร์วิส จะต้องใช้ แฟคทอรี่ หรือ อินสแตนซ์ เท่านั้น แต่กริดเซอร์วิส สามารถรอให้บริการตลอดเวลาเหมือน เว็บเซอร์วิส ทั่วไปได้ ตามแต่สมควรกับแอพพลิเคชั่นนั้นๆ

- Lifecycle management

  การจัดการกับวัฏจักรชีวิตของเซอร์วิสโดยตรงนั้นเป็นเรื่องยาก (ถ้าใช้ผ่านแฟคทอรี่หรืออินสแตนซ์ เนื่องจาก อินสแตนซ์ จะถูกสร้างและลบทิ้งเมื่อไหร่ก็ได้) จึงต้องมี lifecycle management มาช่วยจัดการในเกี่ยวกับวัฏจักรชีวิตของกริดเซอร์วิส

- Service Data

  Service Data มีความเป็น stateful และใช้งานได้แบบชั่วคราว เป็นสิ่งสำคัญที่ กริดเซอร์วิส มีเพิ่มจากเว็บเซอร์วิส เพื่อช่วยให้เราสามารถเพิ่มโครงสร้างข้อมูล เข้าไปในทุกๆเซอร์วิส ทำให้สามารถเข้าถึงคำสั่งต่างๆได้โดยตรงผ่านอินเตอร์เฟสของมันเอง โดยข้อมูลที่เพิ่มเข้าไปมักเป็น

  - State information : ให้ข้อมูลสเตทปัจจุบันของเซอร์วิส
  - Service metadata : ข้อมูลเกี่ยวกับตัวเซอร์วิสเอง

- Notifications

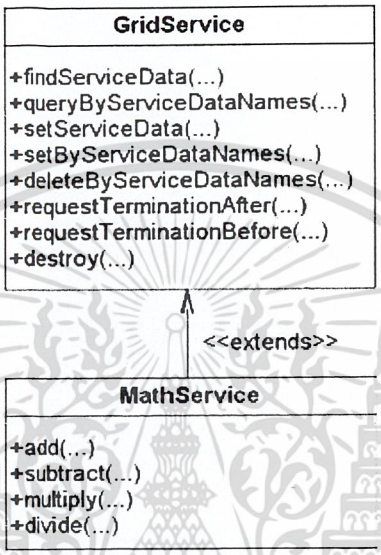
  สามารถใช้ กริดเซอร์วิส เป็นแหล่งประกาศ (notification) และ client เป็นผู้รับประกาศนั้นเมื่อมีการเปลี่ยนแปลงบางอย่างเกิดขึ้นกับ กริดเซอร์วิส จะมีการแจ้งไปยัง client ทุกตัว (แจ้งเฉพาะการเปลี่ยนแปลงบางอย่างตามแต่ผู้เขียนโปรแกรมของกริดเซอร์วิสต้องการ)

- Service Groups

เราสามารถนำเซอร์วิส ต่างๆมาจับกลุ่มเข้าไว้ด้วยกัน แล้วเรียกใช้ได้ในครั้งเดียว โดยสามารถเพิ่มหรือลบเซอร์วิสออกจากกลุ่มได้

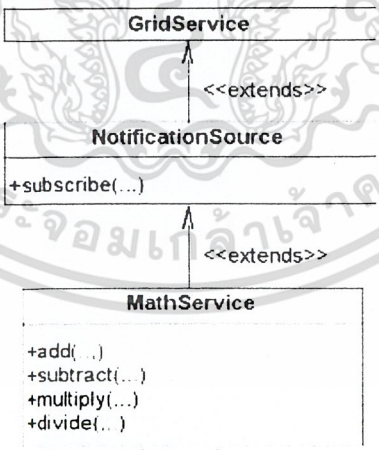
- portType Extension

เรียกอินเตอร์เฟสของ เว็บเซอร์วิส ว่า portType ซึ่งโดยทั่วไปมักจะมีเพียง 1 portType ต่อ 1 เว็บเซอร์วิส แต่ใน กริดเซอร์วิส สามารถมีได้หลายๆ portType ต่อ 1 เซอร์วิส ซึ่ง portType ที่เพิ่มขึ้นมานั้นต้องขยายมาจาก portType ที่มีอยู่ก่อน



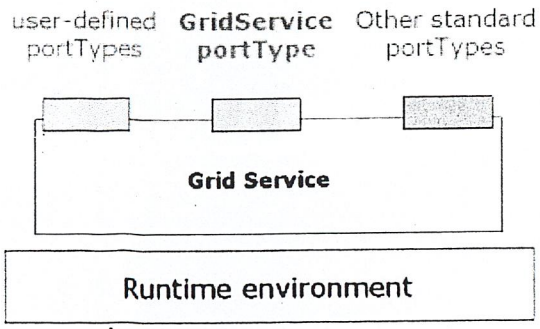
รูปที่ 3-7 แสดงการขยาย portType ของกริดเซอร์วิส

OGSI ออกมาตรฐาน portType อื่นๆที่สามารถขยายได้โดยการเพิ่มฟังก์ชั่น เข้าไปใน กริดเซอร์วิส



รูปที่ 3-8 แสดงการเพิ่มฟังก์ชั่นเข้าไปในกริดเซอร์วิส

โดยทั่วไปแล้ว จะพบว่า กริดเซอร์วิส มี portType อยู่ 3 อย่าง



รูปที่ 3-9 แสดง portType ของกริดเซอร์วิส

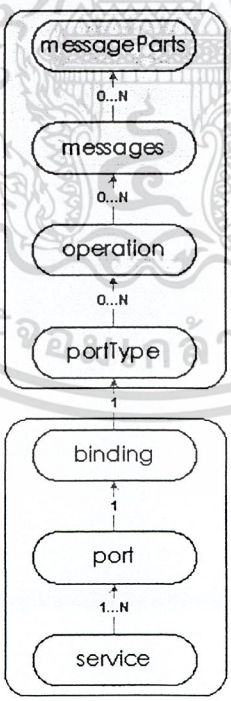
- GSH & GSR

  กริดเซอร์วิส ถูกระบุตำแหน่งด้วย URI เช่นเดียวกับ เว็บเซอร์วิส แต่เรียก กริดเซอร์วิส URI ว่า Grid Service Handle (GSH) เช่นเดียวกับ เว็บเซอร์วิส ที่ต้องมี WSDL ใช้บอกวิธีติดต่อกับ เว็บเซอร์วิส นั้นๆ กริดเซอร์วิส ก็ต้องมี Grid Service Reference (GSR)

3.3 ความแตกต่างของกริดเซอร์วิสและเว็บเซอร์วิส

3.3.1 Web Services Description Language (WSDL)

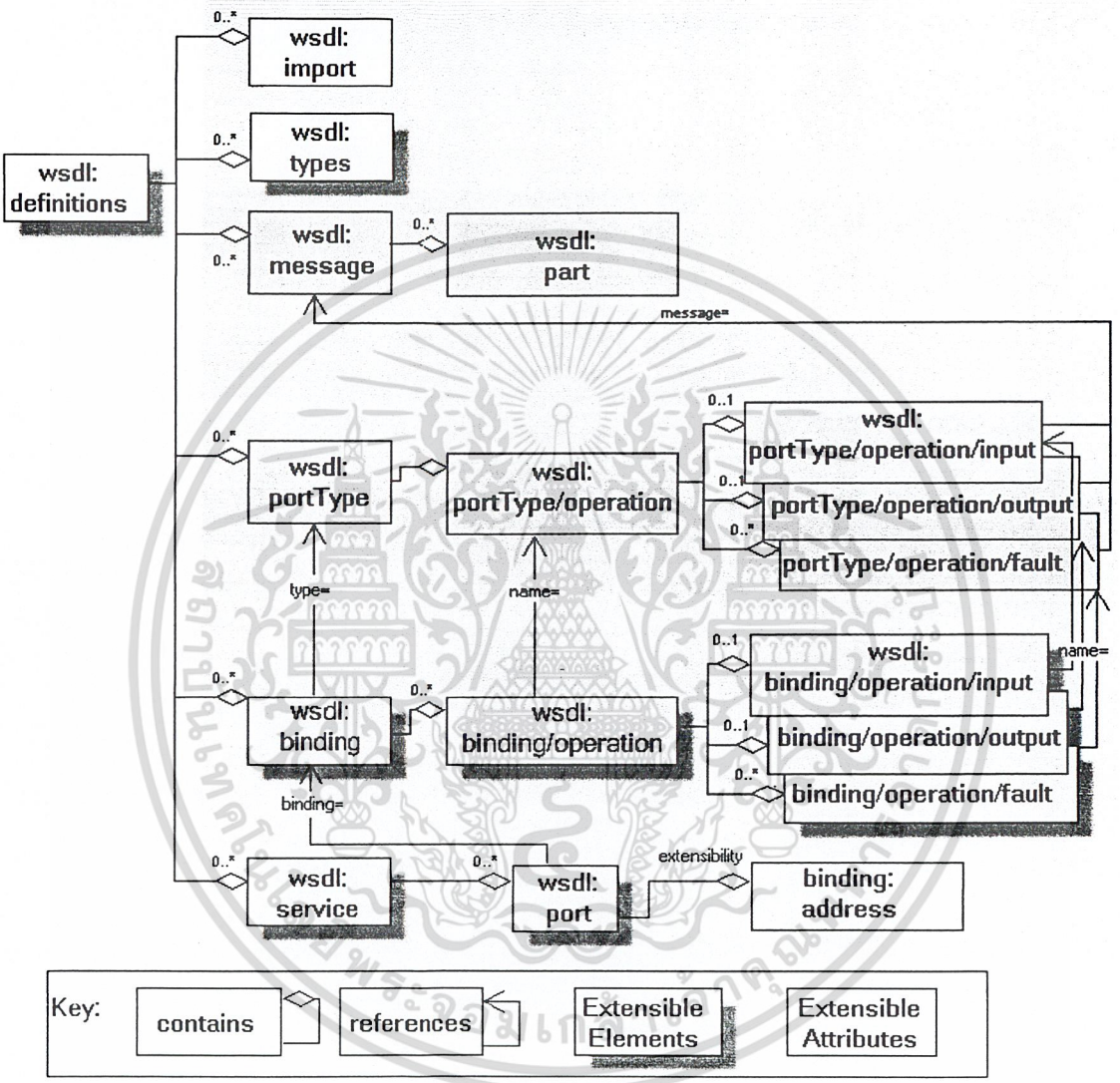
WSDL เป็น XML document เพื่ออธิบายข้อตกลงระหว่าง Web Service และผู้บริโภคเกี่ยวกับรูปแบบของข้อความว่าสามารถเปลี่ยนเป็นรูปแบบใดได้บ้างที่ เซอร์วิสจะรองรับ จากรูป 3-10 เป็น outlines และความสัมพันธ์ของโครงสร้างในส่วนหลักของ WSDL document



รูปที่ 3-10 โครงสร้างเอกสาร WSDL

องค์ประกอบของ document ถูกแบ่งออกเป็น 2 ส่วน:

- ส่วนบน ('message', 'portType') เป็นส่วนสำคัญของ service interface, operations, parameters ซึ่งหมายความว่าผู้บริโภคสามารถทำอะไรได้บ้าง
- ส่วนล่าง ระบุ binding(s) ของวิธีการทำให้ message formats เป็นรูปธรรมขึ้นมา, โปรโตคอล และ ที่อยู่ ปลายทาง ที่สามารถเรียกใช้เซอร์วิสได้



รูปที่ 3-11 ค่าต่างๆของ WSDL

จากรูป 3-11 อธิบายโครงสร้างของ WSDL document ละเอียดขึ้น เริ่มจาก ส่วนประกอบ XML ที่เรียกว่า wsdl:definitions อาจประกอบด้วยส่วนย่อยๆ เช่น wsdl:import และ wsdl:types

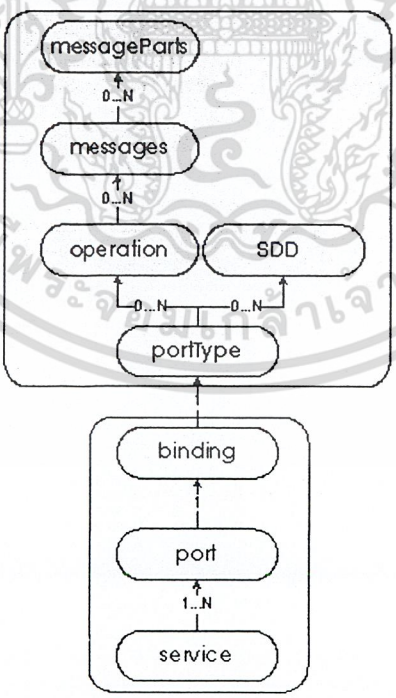
หัวใจของ WSDL คือ portType ซึ่งระบุกลุ่ม input, output และ fault message ที่เซอร์วิส เตรียมรับมือไว้แล้ว บางครั้งก็เทียบง่ายๆได้ว่า portType เป็น JAVA interface หรือ C++ class ส่วน message ใน portType อาจประกอบด้วยหลายๆส่วนซึ่งแต่ละส่วนอาจต่างชนิดกัน หรืออาจเป็น input และ output parameters ที่ message รวมไว้ด้วยกัน

ชนิดของ message ถูกระบุด้วยชนิดของส่วนประกอบใน wsdl:definition ซึ่งสามารถขยายออกไปได้จากการเข้าไปสร้างส่วนประกอบย่อยเพื่อสร้างชนิดของข้อมูล โดยชนิดโดยปกติใน WSDL คือ XML Schema

ส่วนสำคัญอื่นๆของ WSDL คือการนำมาผูกกัน (binding) เป็นการสร้าง message ซึ่งก็คือ data encoding, messaging protocol, และ underlying communication protocol ส่วนประกอบ XML ที่ใช้จะเป็น operations ที่ใช้ชื่ออ้างอิงถึง message ต่างๆ ลักษณะสำคัญของ WSDL คือความจุที่สามารถขยายได้โดยการเพิ่มส่วนประกอบย่อยๆเข้าไป WSDL ยอมให้บรรยาย data encoding, message และ communication protocol ใดๆได้ ประโยชน์ของการบรรยายส่วนต่างๆขึ้นกับระบบ client และ server สามารถแปล WSDL descriptions เพื่อ encode/decode message

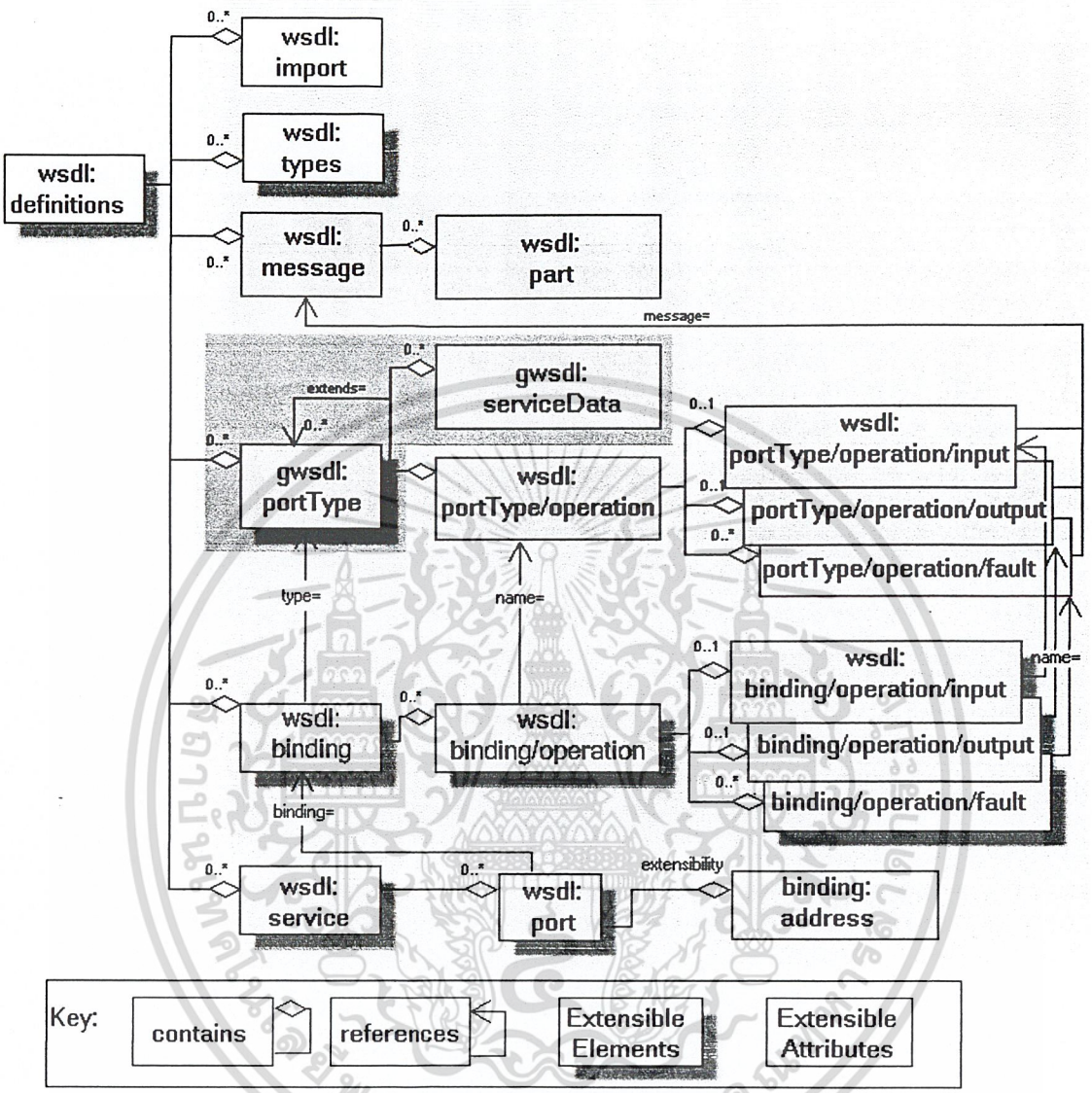
3.3.2 Grid Services Description Language (GWSDL)

OGSI เพิ่ม features ของกริดเซอร์วิสเข้าไปในเว็บเซอร์วิสโดยทำ WSDL portType element ซ้ำ โดยกริดเซอร์วิส จะบรรยายในรูปแบบของ WSDL ที่เพิ่มขึ้น ที่เรียกว่า GWSDL รูป 3-12 แสดงโครงสร้างหลักของ GWSDL ซึ่งคล้ายกับโครงสร้างของ WSDL



รูปที่ 3-12 โครงสร้างเอกสาร GWSDL

ข้อแตกต่างระหว่าง GWSDL กับ WSDL แสดงอยู่ในรูปต่อไปนี้



รูปที่ 3-13 Markup language for Grid Services

- ใน OGSI, portType หนึ่งๆ ถูกสร้างโดยการอ้างถึง portType หรือ portTypes ที่มีอยู่กับ attribute ที่เพิ่มขึ้น และการกำหนดความหมายเพิ่มเข้าไปเท่าที่ต้องการ
- ขณะที่ WSDL portType มีแค่ operations, OGSI portType ก็มี Service Data Declarations (SDDs) ซึ่งช่วยบรรยาย identity และ interfaces ของ service รวมไปถึงสถานะที่ client สามารถดูได้

3.3.3 ตัวอย่างโปรแกรม 'Counter'

จะเปรียบเทียบให้เห็นความแตกต่างระหว่างกริดเซอร์วิส และ เว็บเซอร์วิสโดยอาศัยโปรแกรม Counter ช่วยในการเปรียบเทียบ

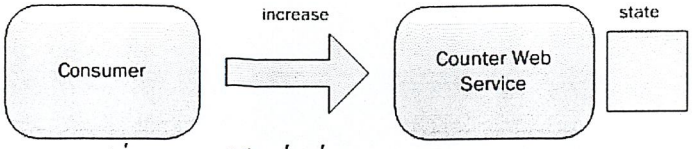
3.3.3.1 Counter Web Service

WSDL สำหรับ Counter Service ทั่วไปที่มี 2 operations (increase และ getValue) แสดงดังรูป 3-14

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:counter="http://www.gridforum.org/namespaces/
2003/05/ogsiprimer/counter/webservice"
        targetNamespace="http://www.gridforum.org/namespaces/
2003/05/ogsiprimer/counter/webservice">
  <wsdl:types>
    <xs:schema/>
  </wsdl:types>
  <wsdl:message name="increaseMsg">
    <wsdl:part name="value" type="xs:positiveInteger"/>
  </wsdl:message>
  <wsdl:message name="getValueMsg">
    <wsdl:part name="value" type="xs:positiveInteger"/>
  </wsdl:message>
  <wsdl:portType name="counterPortType">
    <wsdl:operation name="increase">
      <wsdl:input message="increaseMsg"/>
    </wsdl:operation>
    <wsdl:operation name="getValue">
      <wsdl:output message="getValueMsg"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

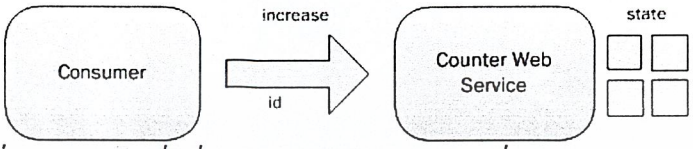
รูปที่ 3-14 เอกสาร WSDL Counter Web Service

ในโลกของ Web Services, Counter Web Service คือ agent ที่คอยแสดงผลเป็น WSDL และเป็นตัวรับ operation requests



รูปที่ 3-15 การใช้คำสั่งเพิ่มค่าใน Counter Web Service

ถ้าทำ Counter Web Service ให้เป็น multiple counter ต้องมี id เพิ่มเข้ามาเพื่อใช้ระบุ counter ที่กำลังถูกใช้งาน ตามที่แสดงในรูป 3-16



รูปที่ 3-16 การใช้คำสั่งเพิ่มค่าใน Counter Web Service ที่รองรับหลายๆ Counter

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:counter="http://www.gridforum.org/namespaces/
2003/05/ogsiprimer/counter/basic "
        targetNamespace="http://www.gridforum.org/namespaces/
2003/05/ogsiprimer/counter/basic ">
  <wsdl:types>
    <xs:schema/>
  </wsdl:types>
  <wsdl:message name="increaseMsg">
    <wsdl:part name="counterId" type="xs:positiveInteger"/>
    <wsdl:part name="value" type="xs:positiveInteger"/>
  </wsdl:message>
  <wsdl:message name="getValueRequestMsg">
    <wsdl:part name="counterId" type="xs:positiveInteger"/>
  </wsdl:message>
  <wsdl:message name="getValueResponseMsg">
    <wsdl:part name="value" type="xs:positiveInteger"/>
  </wsdl:message>
  <wsdl:portType name="counterPortType">
    <wsdl:operation name="increase">
```

```
    <wsdl:input message="increaseMsg"/>
  </wsdl:operation>
  <wsdl:operation name="getValue">
    <wsdl:input  message="getValueRequestMsg"/>
    <wsdl:output message="getValueResponseMsg"/>
  </wsdl:operation>
 </wsdl:portType>
</wsdl:definitions>
```

รูปที่ 3-17 เวอร์ชั่นแก้ไขของ Counter Web Services WSDL ที่รองรับหลายๆ counters

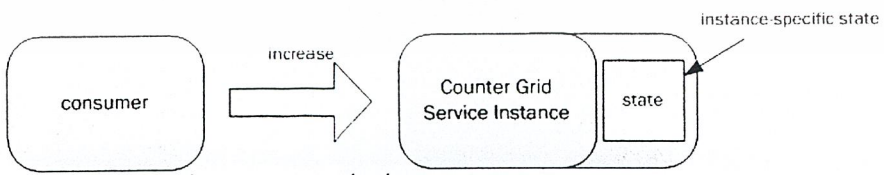
3.3.3.2 Counter Grid Service

ใน OGSI-terms, กริดเซอร์วิส ใช้อ้างอิงถึงแม่แบบอย่างเป็นทางการ เป็นข้อตกลงที่ว่า interface ที่แสดงเป็น GWSDL ต้องมีกริดเซอร์วิสอินสแตนซ์ติดมาด้วย แสดงถึงลักษณะส่วนประกอบของ แอพพลิเคชั่นของกริด (ไม่นับรวมถึง agent ที่สามารถรับ operation request ได้) หมายถึง กริดเซอร์วิสอินสแตนซ์ เป็นตัวรับ logical ของ operation requests

ในมาตรฐานของ OGSI, จำเป็นต้องมี กริดเซอร์วิสอินสแตนซ์ ที่กำหนด Grid Service Handle (GSH) ซึ่งถูกระบุขึ้นมาเป็นพิเศษ และมีการลงทะเบียน แล้วจึงให้ กริดเซอร์วิสอินสแตนซ์ อื่นๆหาเจอได้ ขณะที่ GSH มีลักษณะพิเศษ เพื่อสร้าง Grid Service Reference (GSR) ขึ้นมาเพื่อให้ได้ endpoint-related information ซึ่งจำเป็นต้องใช้ในการเข้าถึงกริดเซอร์วิสอินสแตนซ์ที่ระบุไว้

ครั้งหนึ่ง ผู้ใช้สามารถใช้งาน GSR เพื่อทำให้กริดเซอร์วิสอินสแตนซ์ทำงานได้เหมือนกับ Web Service counterpart อย่างไรก็ตาม ลักษณะโดยเฉพาะของ OGSI ของกริดเซอร์วิสอินสแตนซ์นั้นสามารถใช้ประโยชน์ได้มากกว่านี้

Counter GSI (Counter Grid Service Instance) ดังแสดงในรูป 3-18 เหมือนกับ Counter Web Service ที่ได้แสดงไปแล้วข้างต้น ผู้ใช้งานเซอร์วิสหลายคนสามารถขอคำสั่งบน Counter GSI เดียวกันได้ เพื่อเข้าถึงหรือแก้ไขสเตทของ counter นี่เป็นข้อแตกต่างอย่างหนึ่งจาก Web Service ที่ต้องติดต่อเป็น logic ไปยัง อินสแตนซ์



รูปที่ 3-18 การใช้คำสั่งเพิ่มค่า Counter Grid Service Instance

ด้วยความแตกต่างจาก Counter Web Service ทำให้ Counter Grid Service Instance ต้องมี คุณสมบัติในการจัดการไลฟ์ไทม์ ซึ่งใช้จัดการกับวัฏจักรชีวิตของ counter

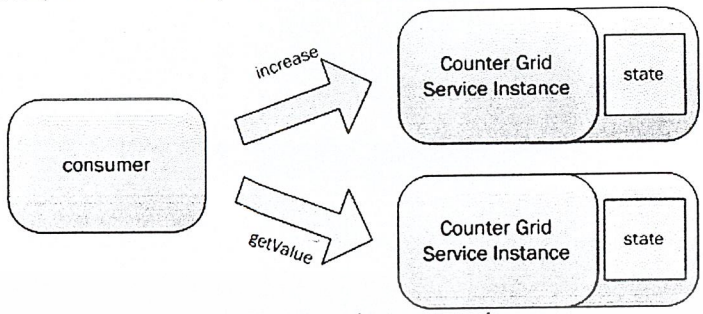
GWSDL interface ของ Counter Grid Service ดูคล้ายกับ เว็บเซอร์วิส นอกจากเปลี่ยน namespace อย่างเห็นได้ชัดแล้ว ข้อแตกต่างอื่นๆดังรูปด้านล่าง แสดงถึง GWSDL processors ที่ counterPortType ขยายการใช้งานออกไปด้วย ogsi:GridService portType

```
<wsdl:definitions xmlns:ogsi="http://www.gridforum.org/namespaces/2003/03/OGSI"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:counter="http://www.gridforum.org/namespaces/
2003/05/ogsiprimer/counter/basic"
        targetNamespace="http://www.gridforum.org/namespaces/
2003/05/ogsiprimer/counter/basic">
  <wsdl:types>
    <xs:schema/>
  </wsdl:types>
  <wsdl:message name="increaseMsg">
    <wsdl:part name="value" type="xs:positiveInteger"/>
  </wsdl:message>
  <wsdl:message name="getValueMsg">
    <wsdl:part name="value" type="xs:positiveInteger"/>
  </wsdl:message>
  <ogsi:portType name="counterPortType" extends="ogsi:GridService">
    <wsdl:operation name="increase">
      <wsdl:input message="increaseMsg"/>
    </wsdl:operation>
    <wsdl:operation name="getValue">
      <wsdl:output message="getValueMsg"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

รูปที่ 3-19 เอกสาร GWSDL ของ Counter Grid Service

แน่นอนว่าความสัมพันธ์ระหว่างกริดเซอร์วิสอินสแตนซ์และสเตทของมันยอมให้เราสำเนาการใช้งานของ Multiple-Counter Web Service โดยไม่ส่งผลถึง interface ทั้งหมดนี้ กล่าวคือ กริดเซอร์วิสอินสแตนซ์เป็นรูปแบบใหม่จาก Web Service ดังรูป 3-18 เวลาต้องการใช้งาน counter หลายครั้ง ค่าเดิมจะยังคงอยู่ใน

interface เดิม อย่างไรก็ตาม OGSI ก็สนับสนุนให้ กริดเซอร์วิสอินสแตนซ์ แสดงสเตทได้ชั่วคราว ดังนั้น จึงเป็นธรรมดาที่จะแสดงแต่ละ counter ด้วยแต่ละ กริดเซอร์วิสอินสแตนซ์ กล่าวคือ มีความสัมพันธ์แบบ 1 ต่อ 1 ระหว่างทรัพยากรหนึ่ง(ในที่นี้คือ counter) กับกริดเซอร์วิสอินสแตนซ์



รูปที่ 3-20 การใช้ 2 Counter GSI ที่แตกต่างกันที่ใช้ในแต่ละ counter

นอกจากการจัดการสเตทแล้ว กริดเซอร์วิสอินสแตนซ์ สามารถเข้าถึงข้อมูลผ่านโครงสร้างที่ เรียกว่า Service Data Elements (SDEs) ที่ประกาศใน interface ของกริดเซอร์วิสผ่านทาง คำสั่งที่ระบุโดย GridService portType ของ OGSI

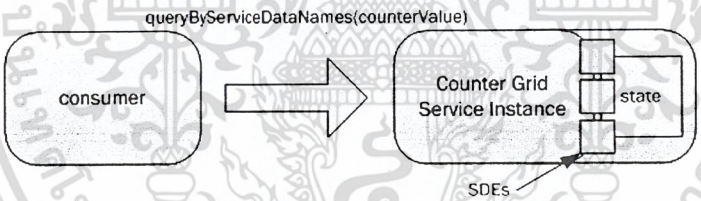
Counter Grid Service อาจเผยค่าของ counter ผ่าน SDE โดยไม่ต้องพึ่งคำสั่ง getValue รูป 3-21แสดง GWSDL ของ Counter Grid Service ที่มี SDE

```
<wsdl:definitions xmlns:ogsi="http://www.gridforum.org/namespaces/2003/03/OGSI"
        xmlns:sd="http://www.gridforum.org/namespaces
/2003/03/serviceData"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:counter="http://www.gridforum.org/namespaces/
2003/05/ogsiprimer/counter/basic"
        targetNamespace="http://www.gridforum.org/namespaces/
2003/05/ogsiprimer/counter/basic">
  <wsdl:types>
    <xs:schema/>
  </wsdl:types>
  <wsdl:message name="increaseMsg">
    <wsdl:part name="value" type="xs:positiveInteger"/>
  </wsdl:message>
    <wsdl:message name="getValueMsg">
    <wsdl:part name="value" type="xs:positiveInteger"/>
  </wsdl:message>
```

```
<ogsi:portType name="counterPortType" extends="ogsi:GridService">
  <wsdl:operation name="increase">
    <wsdl:input message="increaseMsg"/>
  </wsdl:operation>
  <sd:serviceData name="counterValue"
                  type="xs:positiveInteger"
                  minOccurs="1"
                  maxOccurs="1"
                  mutability="mutable">
    <sd:documentation>
      The value of the counter.
    </sd:documentation>
  </sd:serviceData>
 </ogsi:portType>
</wsdl:definitions>
```

รูปที่ 3-21 GWSDL สำหรับ Counter Grid Service ที่ใช้ค่า SDE

ตอนนี้จึงเป็นไปได้ที่จะเข้าถึง counterValue SDE ผ่าน operations ที่กำหนดโดย GridService portType (รูป 3-22)



รูปที่ 3-22 การร้องขอค่า counter จากลูกค้าผ่าน counterValue SDE

บทที่ 4

OGSA & OGSI

ในการพัฒนาระบบกริดนั้นได้มีผู้พัฒนามากมาย จากหลายๆที่ ซึ่งแต่ละที่แม้จะมีแนวความคิดที่เหมือนกัน แต่ถ้าจะนำมาใช้แล้ว จะสามารถใช้งานเข้าด้วยกันได้หรือไม่ ด้วยเหตุนี้เองทำให้ต้องมีการกำหนดมาตรฐานขึ้นมา เพื่อให้สามารถใช้งานระบบกริดร่วมกันได้ขึ้นมา โดยกำหนดออกมาเป็น Open Grid Service Architecture (OGSA)

OGSA เปรียบเสมือนพิมพ์เขียวในการที่จะสร้างมิดเดิ้ลแวร์ขึ้นมา หรือสามารถกล่าวได้ว่า OGSA เป็นโปรโตคอลพื้นฐานที่กำหนดวิธีการการติดต่อ, การส่งข้อมูล, การจัดเตรียมรีซอร์ส, การแชร์รีซอร์ส, การทำตารางเวลา และอื่นๆที่เกี่ยวข้อง โดยอาศัยเทคโนโลยีอยู่ 2 อย่างในการ สร้างมันขึ้นมา คือ เครื่องมือ และเว็บเซอร์วิส โดยสร้างขึ้นมาเป็น Open Grid Service Infrastructure (OGSI) ที่จะนำมาใช้งานจริง

OGSI จะเป็นวิธีการในการสร้างกริดเซอร์วิส ซึ่งจะหมายถึงการจัดการ การแลกเปลี่ยนข้อมูลระหว่างผู้ใช้งานกริด และ เซอร์วิส อื่นๆ หรืออาจพูดได้ว่า เป็น เว็บเซอร์วิสที่จะช่วยเพิ่มความสะดวกให้กับการทำระบบกริด

4.1 Open Grid Service Architecture

ภายในโครงสร้างไอทีที่ใช้ในธุรกิจนั้น จะใช้ ส่วนจัดเตรียมเซอร์วิส (Service Provider) เป็นส่วนช่วยในการเพิ่มประสิทธิภาพให้โครงสร้างของระบบไอทีและ การคำนวณแบบกริดจากหลายๆที่ โดยในการคำนวณนั้นจะเกี่ยวข้องกับการสร้าง การจัดการ และ แอพพลิเคชั่น ที่เกิดจากการนำมารวมกันของ รีซอร์ส และเซอร์วิส (และ ผู้ใช้งาน) ที่เรียกว่า Virtual Organizations ซึ่ง VO นั้นจะเป็นได้ทั้งแบบ เล็ก ใหญ่ มีอายุสั้น อายุยาว ใช้แบบเดี่ยวใช้แบบหลาย และเป็นเนื้อเดียวกัน (homogeneous) หรือ เป็นแตกต่างกัน (heterogeneous)

Open Grid Service Infrastructure นั้นเป็นโครงสร้างที่ถูกกำหนดให้รองรับการสร้าง การดูแลรักษา และให้เป็น แอพพลิเคชั่นของเซอร์วิสที่ดูแลโดย VOs

4.1.1 Service Orientation and Virtualization

เมื่อกล่าวถึง VO จะมองรวมไปที่ รีซอร์สทางกายภาพ นั้นถูกใช้งานร่วมกันอยู่ หรือ อาจจะเป็นเซอร์วิส นั้นจะได้รับความช่วยเหลือจาก resource เหล่านั้น โดยใน OGSA ได้ให้คำจำกัดความของ เซอร์วิส ว่า รีซอร์สที่ใช้ในการคำนวณ, รีซอร์สที่เป็นอุปกรณ์เก็บข้อมูล, ระบบเครือข่าย, โปรแกรม, ฐานข้อมูล และทุกสิ่งที่คล้ายๆกับที่กล่าวมาว่าเป็นเซอร์วิส

Service-oriented จะทำให้สามารถกำหนดสิ่งที่ต้องการทั้งหมดเพื่อกำหนดมาตรฐานใน กลไกการทำงาน (mechanisms),ลักษณะการใช้งานแบบ local หรือ remote (local/remote transparency), การดัดแปลงให้เหมาะสมกับ OS services (adaptation to local OS services) และ ทำเซอร์วิสให้อยู่ในรูปแบบเดียวกัน (uniform service semantics) ซึ่ง Service-oriented นั้น จะทำให้ Virtualization แก้ไขได้ง่าย ซึ่งเป็นความสามารถที่

encapsulation ภายใต้การ implement ในรูปแบบต่างๆ โดยที่ Virtualization ของ กริดเซอร์วิส จะทำให้สามารถ map เซอร์วิสพื้นฐานได้สะดวกบนแต่ละ แพลทฟอร์ม

Virtualization จะง่าย ถ้า ฟังก์ชั่นของเซอร์วิสสามารถแสดงได้ในรูปแบบมาตรฐาน ดังนั้น ในการสร้างเซอร์วิส ควรจะถูก invoke ด้วยวิธีเดียวกัน โดย WSDL ถูกนำมาใช้ในกรณีนี้ โดย WSDL จะให้ทำ multiple bindings สำหรับรูปแบบเดี่ยวได้ โดยรวมถึงการกระจายโปรโตคอลสื่อสาร (เช่น HTTP) ได้ดี เช่นเดียวกับ locally optimized binding (เช่น local IPC) สำหรับการตอบโต้กันระหว่างการร้องขอและกระบวนการต่างๆของเซอร์วิสบนเครื่องเดียวกัน

4.1.2 Service Semantics: The Grid Services

ในการที่จะทำให้เกิดมาตรฐานในการตอบโต้ของ เซอร์วิสได้นั้น จะต้องมี เซอร์วิส ที่ต่างกันและใช้ในการแจ้งความผิดพลาดขึ้น ซึ่ง OGSA จะเป็นตัวที่กำหนด กริดเซอร์วิส ขึ้น ซึ่ง กริดเซอร์วิส นั้น คือ เว็บเซอร์วิส ซึ่งจัดเตรียม กลุ่มของรูปแบบที่กำหนดไว้ และ ทำตามแบบแผนโดยเฉพาะ โดยที่รูปแบบนี้จะทำให้ค้นหาที่อยู่, สร้างเซอร์วิสแบบไดนามิค, จัดการไลฟ์ไทม์, ตรวจสอบ และควบคุมได้ง่าย

รูปแบบและแบบแผนที่กำหนด กริดเซอร์วิส นั้นย่อมต้องเกี่ยวข้องกัน โดยเฉพาะส่วนที่เกี่ยวข้องกันกับการจัดการ เซอร์วิสอินสแตนซ์แบบชั่วคราว ยกตัวอย่างเช่น เซอร์วิสอินสแตนซ์ อาจจะเป็นส่วนที่ คิวรี่ฐานข้อมูล, ทำเหมืองข้อมูล, จองแบนด์วิธของระบบเครือข่าย, ขนย้ายข้อมูล หรือ รองรับความสามารถในการประมวลผล เป็นต้น โดยแบบชั่วคราวนั้นจะเป็นส่วนที่แสดงว่า เซอร์วิสจะถูก จัดการ, มีชื่อ, ค้นหา และใช้งานได้อย่างไร

4.1.2.1 Upgradeability Conventions and Transport Protocols

เซอร์วิสภายในระบบนี้ต้องมีความสามารถที่จะอัพเกรดได้ โดยที่ไม่ทำให้ client ที่ทำงานอยู่เกิดปัญหาขึ้น เช่น การ อัพเกรดเครื่องที่ใช้ทำระบบนั้น อาจทำให้โปรโตคอลของระบบเครือข่ายเปลี่ยน โดย OGSA ได้กำหนดโครงสร้างที่จะทำการรีเฟรชให้ client รู้เกี่ยวกับเซอร์วิส ที่มีการอัพเกรดขึ้น เช่น บอกว่างานใดที่รองรับ หรือ บอกว่า โปรโตคอลเครือข่ายใดที่รองรับกับเซอร์วิสนั้นๆ เป็นต้น โดยที่เซอร์วิสนั้น จะต้องมี 2 คุณสมบัติที่สำคัญ ได้แก่

- Reliable service invocation คือ เซอร์วิสจะตอบโต้กับเซอร์วิสอื่นๆโดยการแลกเปลี่ยน ข้อความในระบบแบบกระจายนั้น จะต้องมีความแน่นอนว่า ข้อความนั้นส่งไปถึงที่หมายได้จริงหรือไม่

- Authentication แต่ละเซอร์วิสนั้นจะต้องเป็นไปตามนโยบายที่กำหนดไว้ เช่น มีการตรวจสอบ client หรือ ตรวจสอบเซอร์วิสอินสแตนซ์ก่อนใช้งาย

4.1.2.2 Standard Interfaces

Interfaces ที่ใช้กำหนด กริดเซอร์วิส นั้น ได้ถูกแสดงไว้ดังตารางด้านล่าง

PortType	Operation	Description
GridService	FindServiceData	Query a variety of information about the Grid service instance, including basic introspection information (handle, reference, primary key, home handleMap: terms to be defined), richer per-interface information, and service-specific information (e.g., service instances known to a registry). Extensible support for various query languages.
	SetTerminationTime	Set (and get) termination time for Grid service instance
	Destroy	Terminate Grid service instance
Notification-Source	SubscribeTo-NotificationTopic	Subscribe to notifications of service-related events, based on message type and interest statement. Allows for delivery via third party messaging services.
Notification-Sink	DeliverNotification	Carry out asynchronous delivery of notification messages
Registry	RegisterService	Conduct soft-state registration of Grid service handles
	UnregisterService	Deregister a Grid service handle
Factory	CreateService	Create new Grid service instance
HandleMap	FindByHandle	Return Grid Service Reference currently associated with supplied Grid Service Handle

ตารางที่ 4-1 อินเตอร์เฟสมาตรฐานของกริดเซอร์วิส

แอพพลิเคชั่นนั้นจะต้องการกลไกที่จะใช้ค้นหาเซอร์วิสที่ใช้งานได้ และสำหรับตัดสินใจเลือกลักษณะพิเศษของเซอร์วิส ที่จะใช้งานที่จะทำงานได้ตามที่ร้องขอ ตามนี้

- มาตรฐานที่ใช้แสดงข้อมูลของเซอร์วิส เป็นการแสดงข้อมูลเกี่ยวกับ กริดเซอร์วิสอินสแตนซ์ (Grid Services Instances) ซึ่ง มีโครงสร้างเป็นกลุ่มของชื่อและอยู่ในรูปแบบ XML ซึ่งเรียกว่า Service data elements ซึ่ง encapsulate เป็นรูปแบบมาตรฐานไว้

- มาตรฐานของคำสั่งต่างๆ เช่น FindServiceData ที่ใช้สำหรับนำข้อมูลจาก กริดเซอร์วิสอินสแตนซ์ กลับมาใช้งาน

- มาตรฐานของการลงทะเบียนข่าวสารเกี่ยวกับ กริดเซอร์วิส ด้วย Registry Services และทำการ map จาก handle ให้เป็น reference จะกล่าวในหัวข้อต่อๆไป

4.1.3 The Role of Hosting Environment

OGSA ได้กำหนดเกี่ยวกับ กริดเซอร์วิสอินสแตนซ์ ไว้ กล่าวคือ กำหนดว่า สร้างขึ้นมาอย่างไร ให้ชื่อได้อย่างไร มีวัฏจักรชีวิตนานเท่าไหร่ ติดต่อกับส่วนใดบ้าง เป็นต้น อย่างไรก็ตาม OGSA บอกในส่วนที่เป็นพฤติกรรมเบื้องต้น มันไม่ได้บอกว่าเซอร์วิสทำอะไร และไม่ได้บอกว่าทำเซอร์วิสขึ้นมาอย่างไร เพราะ OGSA ไม่ได้บ่งบอกเกี่ยวกับ Programming Model, Programming Language, Implementation Tool หรือ Execution Environment เลย แต่ในปัจจุบันก็มีการใช้ภาษาหลากหลายรูปแบบในการสร้างเซอร์วิสเช่น C, C++, Java or Fortran

4.1.4 Using OGSA Mechanisms to Build VO Structures

แอพพลิเคชั่นและผู้ใช้งาน ต้องสามารถที่จะสร้างเซอร์วิสชั่วคราว และ ค้นหาและเลือกใช้ตามคุณสมบัติได้ ใน OGSA จะมี Factory, Registry, GridService and HandleMap ที่เป็นอินเตอร์เฟสที่รองรับการสร้างเซอร์วิสอินสแตนซ์ชั่วคราวและ ค้นหาและเลือกเซอร์วิสอินสแตนซ์ที่เกี่ยวข้องกับ VO ได้

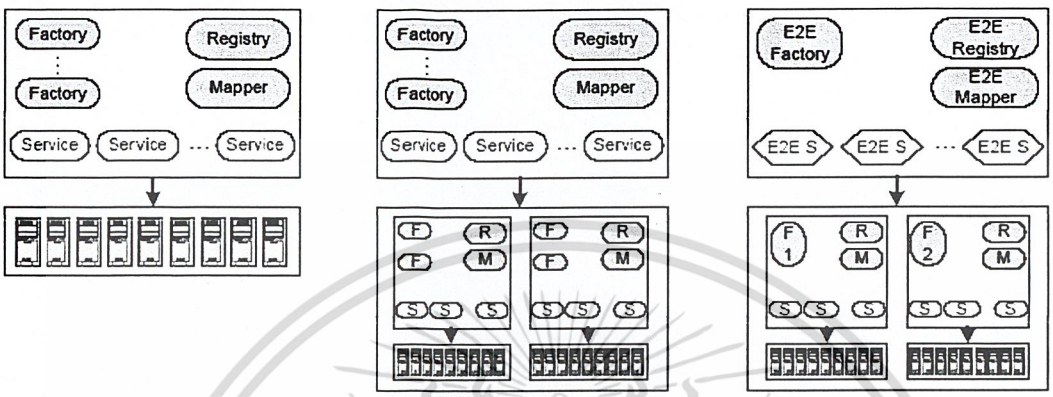


Figure 2: Three different VO structures, as described in the text. From left to right: simple hosting environment, virtual hosting environment, and collective services.

รูปที่ 4-1 ตัวอย่าง 3 รูปแบบที่ใช้งาน

1. Simple Hosting Environment เป็น set ของ รีซอร์ส ที่ตั้งใน single administrative domain และรองรับการจัดการเซอร์วิสโดยแต่ละ factory จะถูกบันทึกในรูปแบบ registry เพื่อให้ client สามารถค้นพบ factories ที่ใช้งานได้ เมื่อมี factory ที่ถูก client ร้องขอเพื่อสร้าง กริดเซอร์วิสอินสแตนซ์ แล้ว factory จะ invoke host-environment ให้สร้างอินสแตนซ์ใหม่ขึ้น และใส่ค่า handle และ register instance นั้น และสร้าง handle ที่ใช้งานได้ลงใน handleMap โดยลักษณะแบบนี้กล่าวได้อีกอย่างว่าเป็น Local Operations

2. Virtual Hosting Environment เช่นเดียวกับแบบแรก แต่จะเป็นการกระจายไปสู่หลายๆ hosting environment โดยที่ client ที่จะใช้งาน จะต้อง register ใน VO ก่อน เพื่อใช้ในการค้นหาและเรียกใช้งาน เป็นการจำลองการทำงานของ 2 environment ที่ต่างกัน แต่ใช้ทำงานร่วมกัน

3. Collective Services เปรียบเสมือนกับการสร้าง Virtual Hosting Environment หลายๆอัน โดยที่แต่ละส่วนนั้นจะเป็นเซอร์วิสอินสแตนซ์ระดับสูง (High Level Service Instance) ต่างกับแบบที่ 2 ที่จะเป็นเพียงเซอร์วิสอินสแตนซ์ แบบธรรมดา โดยเซอร์วิสอินสแตนซ์ระดับสูงนั้น จะเปรียบเสมือนเซอร์วิสอินสแตนซ์หลายๆอันที่รวมกันอยู่

4.2 รายละเอียดทางเทคนิคของ OGSA

ในส่วนนี้จะอธิบายรายละเอียดเกี่ยวกับ กริดเซอร์วิส แบบลึกๆ และความเกี่ยวข้องกับรูปแบบตามข้อตกลง

4.2.1 OGSA Service Model

ในส่วนของ OGSA นั้นกล่าวได้ว่าเป็นส่วนที่กล่าวถึงเซอร์วิสซึ่งเป็น entity ทาง network ที่จัดเตรียมความสามารถในการแลกเปลี่ยนข้อความอันได้แก่ รีซอร์สที่ใช้ในการคำนวณ, รีซอร์สทางกายภาพ, ระบบเครือข่าย, โปรแกรม, ฐานข้อมูล และ เซอร์วิสทั้งหมด โดยที่ OGSA จะแสดงทุกๆอย่างที่กล่าวมาให้เป็น กริดเซอร์วิส

กริดเซอร์วิส เป็นการแสดงลักษณะที่มันสามารถทำได้ ซึ่งจะ implement ออกมาเป็น 1 หรือ หลายๆ อินเตอร์เฟส โดยที่แต่ละอินเตอร์เฟสนั้นจะกำหนดกลุ่มของกระบวนการที่จะ invoke โดยการแลกเปลี่ยนการกำหนดลำดับของข้อความ ซึ่ง กริดเซอร์วิส จะตอบโต้กันผ่านทาง portTypes ของ WSDL ซึ่งใน set ของแต่ละ portTypes นั้นจะถูกรองรับด้วย กริดเซอร์วิส

กริดเซอร์วิส จะมีการดูแลสเตทภายในของมัน สำหรับการจัดการไลฟ์ไทม์ ของ service ซึ่งช่วงที่คงอยู่ของสเตทนั้นจะ แบ่งแยกออกเป็น 1 อินสแตนซ์ของ เซอร์วิสซึ่งจะใช้ กริดเซอร์วิสอินสแตนซ์ เป็นส่วนอ้างอิงถึง กริดเซอร์วิส

การผูกโปรโตคอลที่เกี่ยวข้องกับ เซอร์วิส ที่ส่งออกไปนั้น เช่น เซอร์วิสมีการตอบโต้กับอีกเซอร์วิสโดยการแลกเปลี่ยน message ทั้งที่ไม่ได้ต้องการให้ ในระบบการคำนวณแบบกระจายนั้น จะถือว่าเป็นการล้มเหลว อย่างไรก็ตามส่วนนั้นไม่สามารถรับรองได้ว่าได้ถูกส่งจริง แต่จะใช้การคงอยู่ของสเตทภายในของมันเป็นตัวที่รับรองว่าเซอร์วิสได้รับข้อความหรือยัง แม้ว่าจะต้องใช้การส่งใหม่อีกครั้งถ้ายังไม่ได้รับก็ตาม

OGSA เซอร์วิสจะถูกสร้างหรือถูกทำลายแบบ dynamic ซึ่งบางที เซอร์วิสอาจจะถูกทำลายจริง หรือไม่สามารถเข้าถึงระบบได้เนื่องจากเหตุการณ์หลายๆกรณี ไม่ว่าจะเป็นกรณีผลลัพธ์ของระบบผิดพลาด, ระบบปฏิบัติการล่ม หรือ ระบบเครือข่ายล่ม ทำให้ต้องมี อินเตอร์เฟส ที่จะกำหนดการจัดการเกี่ยวกับไลฟ์ไทม์

เนื่องจาก กริดเซอร์วิส เป็นแบบ dynamic และ stateful ทำให้มันจะต้องมีทางที่จะแยกการสร้างเซอร์วิสอินสแตนซ์ จากอันอื่นๆที่สามารถสร้างขึ้นมาได้เหมือนกัน ในทุกๆ กริดเซอร์วิสอินสแตนซ์ จะต้องมี Global Unique Name ที่เรียกว่า Grid Service Handle (GSH) ที่จะใช้แยก กริดเซอร์วิสอินสแตนซ์ จากอันอื่นๆที่ยังคงอยู่ ไม่ว่าในปัจจุบัน หรือในอนาคตที่กำลังจะมีอันใหม่เพิ่มขึ้นมาอีก (ในกรณีที่ล้มเหลว แล้วมีการสร้างอินสแตนซ์อันเดิม ขึ้นมาใหม่ จะใช้ GSH อันเดิมที่เคยใช้งาน)

ซึ่ง กริดเซอร์วิส นั้นจะสามารถอัพเกรด ได้ในระหว่างที่ยังมีชีวิตอยู่ เช่น เมื่อมีการใช้โปรโตคอลเวอร์ชั่นใหม่ หรือ เพิ่มโปรโตคอล อื่นๆเข้าไป ด้วยเหตุนี้ทำให้ GSH จะไม่อยู่บนปรโตคอล หรือ อินสแตนส์จะไม่มี ข้อมูลที่เจาะจง เช่น ที่อยู่บนเครือข่าย หรือ โปรโตคอล ที่รองรับ โดยข้อมูลนี้จะถูก encapsulate กับอินสแตนซ์ อื่นๆที่ใช้ระบุเจาะจงเกี่ยวกับการตอบโต้กับ เซอร์วิสอินสแตนซ์ ที่เรียกว่า Grid Service Reference (GSR) ที่แตกต่างกับ GSH โดย กริดเซอร์วิสอินสแตนซ์ จะเปลี่ยนได้กระทั่ง Service Lifetime เนื่องจาก GSR จะมีเวลาหมดอายุอย่างตรงๆ หรือแม้กระทั่งจะยกเลิกได้แม้กระทั่งในไลฟ์ไทม์ และ OGSA จะกำหนดการ mapping สำหรับการอัพเดต GSR

อย่างที่บอกว่าใน OGSA นั้นทุกๆอย่าง ถือเป็น กริดเซอร์วิส ซึ่งทำให้จะต้องมี กริดเซอร์วิส ที่ใช้ในการสร้าง กริดเซอร์วิส, handle และ reference ที่ได้กำหนดไว้ใน OGSA model

4.2.2 Creating Transient Services: Factories

OGSA ได้กำหนด class ของ กริดเซอร์วิส ที่ใช้สร้าง กริดเซอร์วิสอินสแตนซ์ ซึ่งเราเรียกว่า Factory ซึ่งใน Factory interface's CreateService จะเป็น operation ที่ใช้สร้างการร้องขอ กริดเซอร์วิส และส่งค่า GSH และค่าเริ่มต้นของ GSR ไปให้เซอร์วิสอินสแตนซ์ ใหม่

ใน Factory interface นั้น ไม่ได้ระบุว่า เซอร์วิสอินสแตนซ์ สร้างขึ้นมาได้อย่างไร เนื่องจาก factory interface จะถูก implement บางส่วนจาก hosting environment (เช่น .NET หรือ J2EE) ที่มีกลวิธีในการสร้าง Service instance ใหม่ๆ ซึ่ง hosting environment จะกำหนดว่า เซอร์วิสจะสร้างได้อย่างไร (เช่น ใช้ภาษาอะไรเขียน เป็นต้น) โดยที่การสร้าง factory ที่มีระดับสูงขึ้นไปนั้น อาจจะใช้การสร้างจากการรวมกันของหลายๆ factory (ตามที่อธิบายไว้ในหัวข้อ 4.4 เรื่อง Using OGSA Mechanisms to Build VO Structures ในรูปที่ 3)

4.2.3 Service Lifetime Management

ในการจัดการไลฟ์ไทม์ นั้น มีอยู่ 2 กรณี คือ

- Client ทราบ หรือ เป็นตัวตัดสินใจที่จะให้ กริดเซอร์วิส นั้นถูกยกเลิกไป ส่วนนี้เป็นส่วนที่จะทำให้ client แน่ใจที่จะได้รับข่าวสารว่า เซอร์วิสอินสแตนซ์ นั้นได้ถูกยกเลิกไปแล้วและ รีซอร์ส ที่ใช้ไปได้กลับคืนมาแล้ว แม้ว่าจะเกิดจากกรณีที่ระบบล้มเหลว (เช่น เกิดจาก การล่มของ server, network หรือ client) ซึ่ง client จะต้องทราบสถานะสุดท้ายของ เซอร์วิสอินสแตนซ์ หรือ การร้องขอนั้น ถ้าเป็นในกรณีที่ระบบล่มจะต้องทราบว่ามันจะต้องติดต่อ เซอร์วิส ไหนต่อหลังจากที่มันยกเลิกไปแล้ว โดยที่ resource ที่เกี่ยวข้องกับ เซอร์วิสนั้นๆจะถูกปลดปล่อยหลังจากเวลานั้น และอย่างน้อย client จะต้องสามารถขยาย ไลฟ์ไทม์ ได้ด้วย
- Hosting Environment เป็นตัวรับประกันว่ารีซอร์สนั้นเป็นส่วนที่สิ้นเปลือง แม้ว่าจะเกิดจากกรณีการผิดพลาดนอกการควบคุม ก็ตาม ถ้าเวลาที่ควรยกเลิกเซอร์วิสมาถึง hosting environment นั้นจะต้องสามารถที่จะเรียกรีซอร์สที่เกี่ยวข้องคืน

ซึ่งกลวิธีของมันจะแบ่งออกเป็นรอบๆ ตามนี้

- Negotiation an initial lifetime เป็นช่วงที่มีการร้องขอที่จะสร้าง กริดเซอร์วิสอินสแตนซ์ ใหม่ผ่าน factory client จะต้องเป็นตัวแสดงเวลาที่น้อยที่สุดและมากที่สุดที่จะยอมรับได้ในการตั้ง initial lifetime และ factory จะเป็นตัวเลือกค่า initial lifetime นั้นส่งกลับไปให้ client

- Request a lifetime extension เป็นช่วงที่ client จะร้องขอให้ขยายไลฟ์ไทม์ ผ่านข้อความ ที่ชื่อว่า SetTerminalTime เข้าไปให้ กริดเซอร์วิสอินสแตนซ์ซึ่งจะระบุเวลาที่ยอมรับได้ที่น้อยและมากที่สุดไปให้

4.2.4 Managing Handles and Reference

จากที่กล่าวไปในช่วงต้น การตอบของ Factory นั้น จะตอบเป็น GSH และ GSR โดยที่ GSH เป็นการรับรองการอ้างถึงการสร้างเซอร์วิสอินสแตนซ์ และ GSR จะถูกสร้างภายในช่วงเวลาจำกัด และจะเปลี่ยนได้ใน

ระยะเวลาของ service lifetime การที่ GSR มันสามารถเปลี่ยนแปลงได้นั้นอาจทำให้ผิดพลาดได้ ซึ่งจะต้องมีวิธีการที่จะใช้เพียง GSH เพื่อนำมาใช้หาค่า GSR ที่ถูกต้องได้

วิธีการนั้นก็คือ HandleMap ซึ่งเป็นกระบวนการที่จะรับค่า GSH และตอบกลับเป็น GSR ที่ถูกต้อง กระบวนการ map ของ HandleMap นั้น จะเก็บ track ว่า กริดเซอร์วิสอินสแตนซ์ แท้จริงแล้วยังมีชีวิตอยู่มั้ย และไม่ตอบรับค่าอินสแตนซ์ ว่ามันต้องยกเลิกอย่างไร อย่างไรก็ตาม การมีของ GSR ที่ถูกต้อง ไม่ได้เป็นการแน่นอนว่า กริดเซอร์วิสอินสแตนซ์ นั้นจะติดต่ออย่างไร และ เซอร์วิสบางทีจะล่ม หรือ ยกเลิกระหว่างเวลาที่ GSR ไม่ได้ให้ออกมาว่าถูกใช้อยู่ เกี่ยวกับ HandleMap Interface นั้นจะมีปัญหาที่เกี่ยวกับบรรจุ GSR เพื่อให้เป็นเซอร์วิสอยู่ 2 ปัญหา ดังนี้

1) การบอก handleMap service ว่ามันบรรจุการ map จาก GSH ที่เจาะจง

2) การติดต่อกับ handleMap เพื่อให้บรรจุค่า GSR ที่ต้องการเข้าไป

ใน 2 ปัญหานี้ เป็นปัญหาที่เกี่ยวกับการ map GSH เป็น GSR ซึ่งจำเป็นต้องให้ทุกๆกริดเซอร์วิสอินสแตนซ์ จะต้องถูกลงทะเบียนอย่างน้อย 1 ครั้งจาก handleMap ซึ่งเรียกว่า home handleMap โดยโครงสร้างแล้ว GSH จะรวม home handleMap's identity เข้าไปด้วย โดยการติดต่อให้ handleMap บรรจุค่า GSR จากค่า GSH ที่ให้ไป สามารถระบุและกำหนดได้โดยง่ายเนื่องจากว่า มีชื่อที่เป็นเอกเทศในแต่ละ local นั้นๆ เพื่อหลีกเลี่ยงการซ้ำกันของ service ที่ถูกสร้างขึ้นมาจากศูนย์กลาง แม้ว่าจะอาศัย Domain Name System อย่างไรก็ตามทุกๆ GSH นั้นจะต้องมี 1 home handleMap

ในการสร้าง HandleMap นั้น เป็นการใช้กริดเซอร์วิสสร้าง และมันจะต้องมี GSH ให้ จึงใช้ค่า GSH นี้ในการสร้าง ซึ่งจะทำได้ย่อมต้องการให้ home handleMap ถูกกำหนดค่าโดย URL และรองรับ bootstrapping operation ที่จะทำให้เป็นหนึ่งเดียว โดยใช้โปรโตคอลที่รู้จักกันดีในการให้ชื่อ เช่น HTTP (หรือ HTTPS) เนื่องจากการใช้ GSR เพื่อบอกว่า โปรโตคอลอะไรที่ควรจะติดต่อกับ handleMap service ซึ่งการใช้ HTTP GET operation ที่ถูกใช้บน URL ที่จุดนั้นเพื่อให้เป็น home handleMap และ GSR สำหรับ handleMap ในรูปแบบ WSDL เพื่อตอบกลับไป

ความสัมพันธ์ระหว่างเซอร์วิส ที่ implement HandleMap และ Factory interface โดยเฉพาะ GSH จะตอบกลับโดย factory request ที่ต้องบรรจุ URL ของ home handleMap และ GSH/GSR เพื่อ mapping จะต้องเข้าไปและปรับปรุงค่าได้ใน handleMap service ในการ implement factory จะต้องตัดสินใจว่าจะให้ service อะไรใช้ใน home handleMap ที่จริงแล้วเพียง service เดียวบางทีจะสามารถสร้างได้ทั้ง Factory และ HandleMap Interface

4.2.5 Service Data และ Service Discovery

ส่วนที่เกี่ยวข้องกับกริดเซอร์วิสอินสแตนซ์นั้นเป็นกลุ่มของข้อมูลที่เรียกว่า service data ซึ่งเป็นกลุ่มของค่า XML ที่ฝังอยู่ในค่าของข้อมูลของเซอร์วิส ซึ่งในแต่ละค่านั้นจะมีชื่อของกริดเซอร์วิสอินสแตนซ์นั้น, ชนิด, เวลาที่ยังมีชีวิตอยู่ ซึ่งเป็นข้อมูลที่ใช้ในกระบวนการจัดการไลฟ์ไทม์

รูปแบบของกริดเซอร์วิสนั้นจะกำหนดด้วยคำสั่ง WSDL โดยใช้ FindServiceData สำหรับการคิวรี่ และเลือกเอามาจากข้อมูลของเซอร์วิสนั้น ซึ่งคำสั่งนี้จะใช้ชื่อในการคิวรี่ และอาจขยายไปรูปแบบอื่นๆ เช่น Xquery ที่ใช้กับกรณีพิเศษๆบางกรณี

คุณสมบัติของกริดเซอร์วิสนั้นจะต้องประกอบไปด้วยค่า service data อย่างน้อย 1 ค่าที่รองรับโดยกริดเซอร์วิสอินสแตนซ์ใดๆ ค่าที่ใช้ เช่น GSH,GSR, คีย์ชั้นปฐมภูมิ และค่า handleMap เป็นต้น

ใน 1 แอพพลิเคชั่นของกริดเซอร์วิสนั้นจะต้องมี FindService ที่ใช้ค้นหาเซอร์วิส ซึ่งจะใช้ค่า GSH ในการค้นหา และจะพิจารณาจากค่า การจัดเตรียม, จำนวนที่ร้องขอใช้เซอร์วิส, ค่าโหลดของเซอร์วิส, หรือ นโยบายที่ใช้จัดการ มาใช้เลือกเซอร์วิสที่เหมาะสม

กริดเซอร์วิสที่รองรับการค้นหานั้นจะถูกเรียกโดยค่า registry ซึ่งใช้กำหนดอยู่ 2 อย่าง คือ รูปแบบการลงทะเบียน ที่จะกำหนดว่า GSH ใดที่จะให้ลงทะเบียนด้วยเซอร์วิสที่ใช้ลงทะเบียน และ ใช้กำหนดค่าที่เกี่ยวข้องกับการลงทะเบียนของ GSH

การลงทะเบียนนั้นเป็นการยอมให้ GSH ที่ลงทะเบียนไว้ด้วยเซอร์วิสที่ใช้ลงทะเบียนเพื่อคัดเลือกกลุ่มย่อยให้ได้อย่างถูกต้อง ตรงตามความต้องการ เพื่อสะดวกต่อการค้นหาตามที่ร้องขอ ซึ่งค่าที่ลงทะเบียนของ GSH นั้นจะต้องรีเฟรชเป็นระยะๆ เพื่อให้เซอร์วิสที่ใช้ค้นหาสามารถเลือกเซอร์วิสที่เหมาะสมที่สุดได้

4.2.6 Notification

ใน OGSA ส่วนเฟรมเวิร์คของ notification นั้นจะเป็นส่วนที่ยอมให้ client จะลงทะเบียนประกาศเฉพาะทางที่สำคัญได้ (ในรูปแบบ NotificationSource) เพื่อจะส่งไปให้ผู้รับตามที่ต้องการ (NotificationSink) ถ้าเซอร์วิสใดๆต้องการให้รองรับข้อความ notification แล้ว มันจะต้องรองรับรูปแบบของ NotificationSource ด้วย ซึ่งเซอร์วิสที่ต้องการข้อความ notification จะต้องสร้าง รูปแบบ NotificationSink ขึ้นมา ซึ่งใช้ในการส่งข้อความนั้นๆ ซึ่งขั้นตอนของมันจะเริ่มจากต้องรับรูปแบบของ แหล่งข้อมูลก่อน แล้วให้ค่า GSH ของ sink ไปเพื่อใช้ส่งข้อความ โดยที่ sink จะต้องส่งจ้อความที่บ่งบอกว่าสนใจข้อความ notification นั้นไปด้วย

สิ่งสำคัญในรูปแบบของ notification นั้นคือ ต้องปิดการรวมตัวกันด้วยข้อมูลของเซอร์วิส โดยมีคำสั่งพิเศษคือ “push”ที่ใช้ส่งข้อมูลตามเงื่อนพิเศษ (โดยการเรียกคำสั่ง FindServiceData เพื่อเตรียมที่จะ “pull”) ซึ่งเฟรมเวิร์คที่ใช้นี้จะยอมให้มีการส่งข้อความตรงๆจากเซอร์วิสไปยังเซอร์วิส รวมถึงการส่งไปยังเซอร์วิสกลุ่มที่ 3 ด้วย เช่น การส่งข้อความที่สำคัญทางธุรกิจไปให้กลุ่มที่ต้องการ เป็นต้น โดยต้องมีโปรโตคอลที่รองรับการส่งแบบมัลติคาสท์ด้วย

4.2.7 การควบคุมการเปลี่ยนแปลง

เพื่อรองรับการส่งและการควบคุมการเปลี่ยนแปลงของกริดเซอร์วิสได้อย่างมีประสิทธิภาพ รูปแบบของกริดเซอร์วิสนั้นจะต้องเป็นแบบที่ทั่วถึงกัน (globally) และมีชื่อเป็นเอกเทศกัน (uniquely) ซึ่งใน WSDL นั้นจะกำหนดไว้ใน portType ด้วย qname โดยการเปลี่ยนแปลงใดๆจะต้องสร้างคำจำกัดความของเซอร์วิสที่เปลี่ยนแปลงนั้น ซึ่งการเปลี่ยนแปลงต้องรองรับกับรูปแบบเดิมด้วย โดยส่วนนี้จะเป็นส่วนที่ยอมให้ client ที่

ต้องการกริดเซอร์วิสนั้นๆด้วยคุณสมบัติที่พิเศษที่เหมาะสมกับงานของตน เพื่อให้เซอร์วิสนั้นๆใช้ได้อย่างมีประสิทธิภาพ

4.2.8 การผูกกับโปรโตคอลเครือข่าย

ในเฟรมเวิร์คของเว็บเซอร์วิสนั้นจะใช้โปรโตคอลที่แตกต่างกันมาร่วมกันทำงาน เช่น ใช้ SOAP+HTTP พร้อมด้วย TLS สำหรับด้านความปลอดภัย ส่วนนี้จะบ่งบอกว่า OGSA ใช้อะไรบ้าง ซึ่งจะแบ่งเป็น 4 ส่วนที่มีความสำคัญดังนี้

- Reliable transport: อย่างที่กล่าวไปแล้วว่ากริดเซอร์วิสนั้น จำเป็นจะต้องมีการส่งเซอร์วิสไปได้อย่างน่าเชื่อถือ ซึ่งโปรโตคอลที่รองรับด้านนี้ก็เช่น HTTP-R
- Authentication and delegation: จากที่กริดเซอร์วิสนั้นจำเป็นที่จะต้องรองรับการติดต่อสื่อสารไปยังที่อื่นๆ ซึ่งต้องมีการรับรองด้วย ทางหนึ่งที่ใช้กันคือ ใช้การสื่อสารแบบทางเดียว เช่น ใช้ TLS ช่วยรองรับด้านนี้
- Ubiquiry: เมื่อเป้าหมายของกริดคือเพื่อให้เป็นแหล่งรวมของ VO จากรีซอร์สที่กระจายตามส่วนต่างๆ จึงจำเป็นต้องมีการเลือกเซอร์วิสที่เหมาะสมที่จะใช้ตอบโต้กัน
- GSR Format: ในการเรียก GSR นั้น จะต้องมีการผูกกันด้วยรูปแบบพิเศษ 1 ในรูปแบบที่ใช้ก็คือรูปแบบเอกสาร WSDL แต่ก็มีทางเลือกอื่น เช่น CORBA IOR เป็นต้น

4.2.9 Higher-Level Services

ส่วนทฤษฎีและเซอร์วิสที่กล่าวในหัวข้อนี้ จะบอกถึงการเตรียมสิ่งที่จะใช้สร้างกริดเซอร์วิสระดับสูงขึ้นไป ซึ่งเอาไว้ใช้ในด้าน ธุรกิจ หรือ วิทยาศาสตร์ ซึ่งจะอาศัยส่วนต่างๆ ต่อไปนี้

- Distributed data management service: เป็นส่วนที่รองรับการเข้าถึงและการย้ายของข้อมูลที่กระจายกัน ไม่ว่าจะเป็นฐานข้อมูลหรือแฟ้มข้อมูล ซึ่งเซอร์วิสที่รวมอยู่ในนี้จะมีการเข้าถึงฐานข้อมูล, การแปลงข้อมูล, การจำลองการจัดการ, การจำลองตำแหน่งที่เก็บ และ ทรานแซคชั่น

- Workflow service: เป็นส่วนที่รองรับการทำงานของหลายๆแอพพลิเคชั่นบนรีซอร์สที่กระจายกันอยู่

- Auditing service: เป็นส่วนที่รองรับการจำข้อมูล ทำให้ที่เก็บข้อมูลมีความปลอดภัย วิเคราห์ถึงข้อมูลที่มีจุดประสงค์ไม่ดีหรือบุกรุกเข้ามา

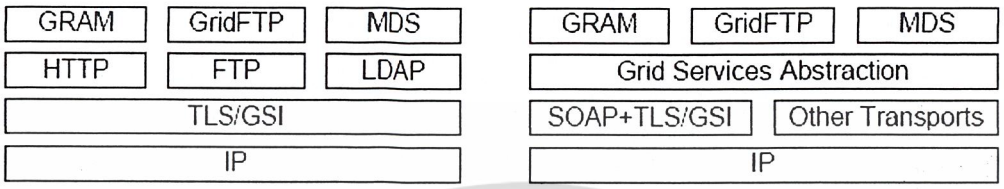
- Instrumentation and monitoring service: รองรับการค้นหาในสภาพแวดล้อมแบบกระจาย โดยจะอาศัย เซ็นเซอร์ช่วย ซึ่งจะมีหน้าที่รวบรวมข่าวสารและวิเคราะห์ เพื่อใช้เตือนสิ่งที่ผิดปกติด้านต่างๆ

- Problem determination service for distributed computing: รวมหลายๆวิธีการ เช่น dump,trace และ log ด้วยการบันทึกเหตุการณ์และด้วยความสามารถในการเข้าคู่กัน

- Security protocol mapping services: จัดเตรียมโปรโตคอลที่ปลอดภัยสำหรับการกระจายส่วนต่างๆ โดยจะ map เข้ากับรูปแบบทั่วไป โดยมีการรับรองความปลอดภัยในการกระจายและการควบคุมสิทธิการเข้าถึงส่วนต่างๆ

ความยืดหยุ่นของเฟรมเวิร์คนี้ก็คือ สามารถสร้างและรวมได้จากหลากหลายวิธี เช่น รวมเซอร์วิสจากส่วนที่จองไว้และใช้หลายๆรีซอร์สมาช่วยในการคำนวณ หรือ อาจจะเชื่อมกับแอพพลิเคชั่นที่เป็นไลบรารี่ หรือ รวมกับเซอร์วิสระดับสูง เป็นต้น

ดังนั้น เพื่อความสะดวกในการจัดการรีซอร์ส, การขนส่งข้อมูล, การใช้โปรโตคอลของเซอร์วิส ใน Globus Toolkit นั้นจึงสร้างเครื่องมือในด้านต่างๆไว้ ดังภาพด้านล่างนี้



รูปที่ 4-2 ภาพทางซ้ายเป็นโปรโตคอลของ Globus Toolkit 3 ส่วนภาพทางขวาเป็นแนวคิดของ OGSA

4.3 Open Grid Service Infrastructure

OGSI (Open Grid Services Infrastructures) นั้นเป็นส่วนที่คิดขึ้นมาเพื่อใช้กำหนดคุณสมบัติของระบบกริด ซึ่งจะกล่าวถึง ความสัมพันธ์ระหว่างโครงสร้างของกริด, ระบบการคำนวณแบบกระจาย, ความเกี่ยวพันกับเว็บเซอร์วิส เฟรมเวิร์ค และความเกี่ยวพันระหว่าง client กับ hosting environment

4.3.1 ความสัมพันธ์กับระบบการคำนวณแบบกระจาย

กริดเซอร์วิส (Grid Service) นั้นจะสร้างขึ้นมาแบบฝังไว้ภายในแต่ละอินสแตนซ์ (instance) ซึ่งมีส่วนที่ใช้อธิบายอินสแตนซ์ ก็คือ WSDL portType โดยหลายๆ Grid Service นั้นจะรวมตัวกัน เรียกว่า Grid Services Factory ซึ่งใช้สร้างอินสแตนซ์ สำหรับแต่ละ portType โดยที่แต่ละอินสแตนซ์ จะเป็นส่วนที่ใช้ติดต่อกันระหว่าง 2 กริดเซอร์วิส ในการทำงานแต่ละครั้งก็จะติดต่อไปที่อินสแตนซ์ ให้ส่งค่าที่ต้องการกลับมา

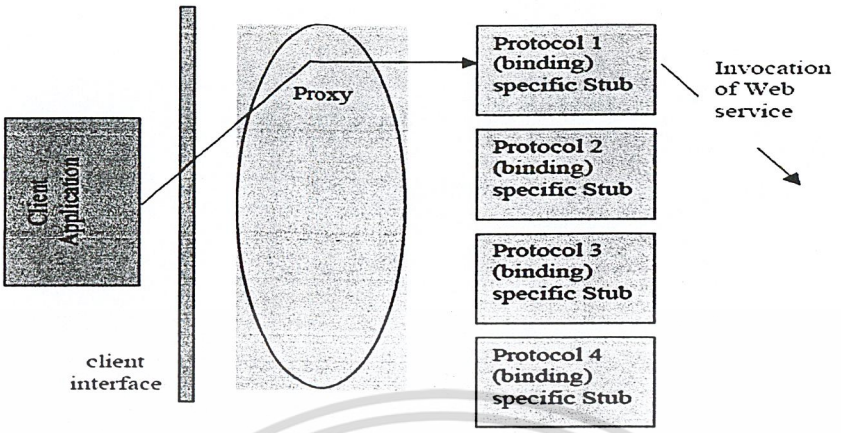
กริดเซอร์วิสอินสแตนซ์นั้น ใช้เพื่อให้เข้าถึงแอพพลิเคชั่นของ client โดยอาศัย Grid Service Handle และ Grid Service Reference เพื่อส่งรีเควสต์ตรงไปที่แต่ละอินสแตนซ์ที่เป็นเป้าหมาย ซึ่งจะได้รับคำตอบเป็นเครื่องเป้าหมายที่จะส่งงานไปให้คำนวณ แสดงออกมาเป็นค่า Grid Service Reference ที่ใช้ในการติดต่อกันจริง

โดยสรุปแล้วสิ่งที่เรียกว่าระบบการคำนวณแบบกระจายในกริดนั้น จะอาศัย GSR ในการกำหนดว่าควรจะส่งงานไปให้ที่ส่วนไหนทำงาน โดยอาศัยตัวจัดการที่เรียกว่าตัวจัดเตรียมเซอร์วิส (Service Provider) ที่เก็บรวบรวมข้อมูลว่า แต่ละโฮสต์ในกริด นั้นมีเซอร์วิสที่มี client ต้องการเรียกใช้หรือไม่ ถ้าเครื่องไหนมีและพบว่ามีความพร้อมที่สุด ก็จะส่งไปเครื่องนั้นๆทำงาน

4.3.2 แนวทางการเขียนโปรแกรมฝั่ง client

เนื่องจากกริดในยุคหลังนั้น ได้นำเทคโนโลยีของเว็บเซอร์วิส มาช่วยในการพัฒนาส่วนต่างๆ โดยส่วนที่ OGSI นำ Web Service ใช้นั้น ส่วนหนึ่งก็คือ WSDL ที่ใช้อธิบายว่า ผูกกับโปรโตคอลอย่างไร, เอนโค้ดอยู่ในรูปแบบใด, ส่งข้อความแบบใด (ระหว่าง RPC หรือ document-oriented) และใช้เว็บเซอร์วิส อย่างไร ซึ่ง

ในกริด จะใช้ Web Service Invocation Framework ร่วมกับ Java API for XML RPC ในหลายๆส่วนของซอฟท์แวร์ ที่พัฒนาขึ้น

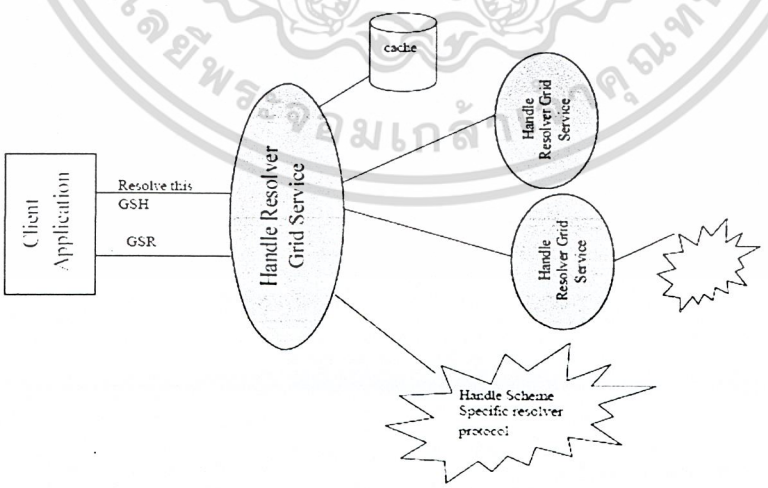


รูปที่ 4-3 แสดงโครงสร้างการทำงานฝั่ง client

4.3.3 การใช้ Grid Services Handles และ References ของ Client

การที่ Client จะเข้าถึงกริดเซอร์วิสอินสแตนซ์ได้นั้น จะต้องผ่าน GSH (Grid Service Handle) และ GSR (Grid Service Reference) โดยที่ GSH นั้นจะเปรียบเสมือน Network Pointer ที่จะชี้ไปที่กริดเซอร์วิสอินสแตนซ์ แต่ GSH ไม่ใช่ส่วนที่จัดเตรียมข้อมูลที่จะให้ client เข้าถึงเซอร์วิสอินสแตนซ์ ได้ การที่จะเข้าถึงกริดเซอร์วิสอินสแตนซ์ได้นั้น client จะต้องผ่านวิธีการแปลง GSH ให้เป็น GSR (Grid Service Reference) ซึ่ง GSR จะเป็นส่วนเก็บข้อมูลที่จำเป็นสำหรับการเข้าถึงกริดเซอร์วิสอินสแตนซ์ ไว้ โดยที่ GSR ไม่ใช่ Network Pointer ที่ชี้ไปที่กริดเซอร์วิสอินสแตนซ์ เนื่องจาก GSR อาจทำให้ผิดพลาดได้ในบางกรณี เช่น กริดเซอร์วิสอินสแตนซ์ นั้นย้ายไปที่ Server ตัวอื่นแล้ว เป็นต้น

OGSI นั้นได้จัดเตรียมวิธีการที่จะทำ HandleResolver เพื่อรองรับให้ client สามารถแปลง GSH ให้เป็น GSR ได้ ในรูปด้านล่างจะแสดงวิธีการ ให้ดู

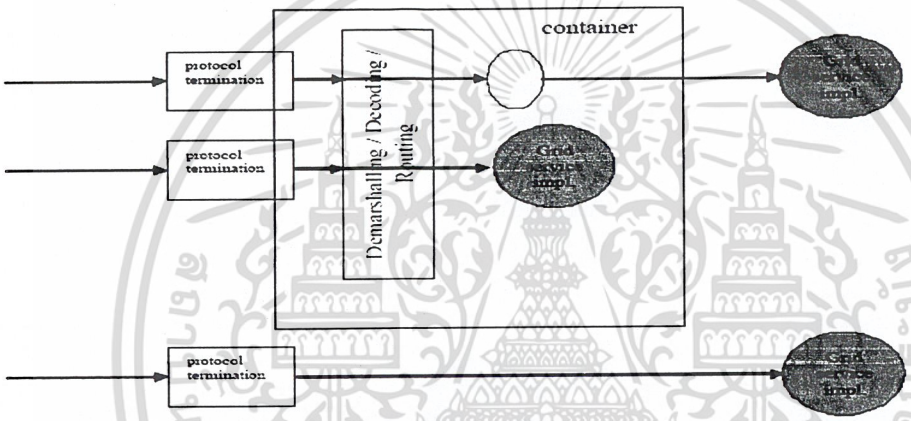


รูปที่ 4-4 แสดงวิธีการ resolve GSH

Client นั้นจะแปลงจาก GSH เป็น GSR โดยผ่านการ invoke ที่ HandleResolver กริดเซอร์วิสอินสแตนซ์โดยที่ HandleResolver จะมีค่า GSR เก็บไว้ที่แคช ภายในของตัวมัน แต่ถ้าไม่มีเก็บไว้ที่ แคชมันก็ต้องขอ HandleResolver ตัวอื่นๆให้ช่วยติดต่อไปที่ GSH ที่ต้องการให้ โดยอาจจะผ่านโปรโตคอลที่นิยมใช้กันทั่วๆไป เช่น ใช้ HTTP เพื่อรับค่า URL แล้วจึงค่อยเอนโค้ดจากค่า GSH ให้เป็น GSR เป็นต้น

4.3.4 ความสัมพันธ์ระหว่าง Client กับ Hosting Environment

OGSI นั้นไม่ได้ทำส่วนในฝั่งที่เป็นตัวจัดเตรียมเซอร์วิส (Service Provider) แต่จะใช้โมเดลคล้ายๆกับของฝั่ง Server ที่ใช้ใน J2EE กล่าวคือ ทุกๆส่วนในมาตรฐานของกริดเซอร์วิส (เช่น ส่วนการร้องขอ, ส่วนจัดการไลฟ์ไทม์, ส่วนที่เกี่ยวกับการลงทะเบียน เป็นต้น) จะถูกซ่อนไว้ภายใน ยูสเซอร์โปรเซส เช่น การเชื่อมต่อกับไลบรารี่มาตรฐาน เป็นต้น



รูปที่ 4-5 แสดงลักษณะการทำงานของฝั่ง hosting environment

จากรูป กล่าวคือ กริดเซอร์วิสอินสแตนซ์นั้น จะฝังอยู่ในคอนเทนเนอร์ ที่ข้อความต่างๆ จะผ่านภายในคอมโพเนนท์เหล่านี้ ในรูปแบบต่างๆ เช่น XML หรือ SOAP เป็นต้น

โดยส่วนมากแล้ว ในคอนเทนเนอร์นั้นมักจะสร้างขึ้นมาให้ใช้ในเรื่องต่างๆ เช่น จัดเตรียมการจัดการเวลาชีวิต, หรือการตรวจสอบสิทธิและอนุญาตแบบอัตโนมัติ, การร้องขอเพื่อเข้าสู่ระบบ, การยกเลิกเซอร์วิสอินสแตนซ์เมื่อหมดไลฟ์ไทม์ และ ยกเลิกการร้องขอแบบตรงๆ

4.3.5 คุณสมบัติของกริดเซอร์วิส

จากที่กล่าวไปบทก่อนๆแล้ว ว่าทุกอย่างที่เป็นกริดเซอร์วิสก็คือเว็บเซอร์วิสแม้ว่าลักษณะมันจะไม่เหมือนกันทุกอย่างก็ตาม ในส่วนนี้จะระบุคุณสมบัติของกริดเซอร์วิสไว้

- ใช้ส่วนของ WSDL ตามที่เว็บเซอร์วิสใช้ ตั้งแต่ WSDL 1.2 ขึ้นไป
- กำหนด เซอร์วิสเดต้า (service data) ซึ่งเป็นตัวมาตรฐานที่มาจากแสดงหรือซักถามจากเซอร์วิสอินสแตนซ์
- แนะนำเกี่ยวกับแกนในคุณสมบัติของกริดเซอร์วิส ที่ต้องมี

- กำหนดกริดเซอร์วิสเดสคริปชั่น(Grid service description) และกริดเซอร์วิสอินสแตนซ์จากลักษณะการใช้งานจริงๆ
- กำหนดว่า OGSI มีรูปแบบเกี่ยวด้านเวลาอย่างไร
- กำหนดการสร้าง GSH และ GSR ซึ่งใช้ในการอ้างกริดเซอร์วิสอินสแตนซ์
- กำหนดวัฏจักรชีวิตของกริดเซอร์วิสอินสแตนซ์

4.4 เซอร์วิสเดต้า (Service Data)

การที่กริดเข้าไปใกล้การเป็นเว็บเซอร์วิสที่ทำงานเป็นสเตท ทำให้มันจำเป็นต้องมีกลไกที่จะใช้อธิบายข้อมูลในสเตทของเซอร์วิสอินสแตนซ์ที่ผู้ร้องขอคิวรี่, อัพเดต, เปลี่ยนการประกาศ อย่างที่เว็บเซอร์วิสทำ ตั้งแต่ที่ใช้เว็บเซอร์วิสรวมสิ่งที่ใช้ภายนอกซึ่งใช้บ่งบอกข้อมูลสเตทที่เรียกว่า เซอร์วิสเดต้า

ในการเตรียมส่วนที่ใช้อธิบายรายละเอียดของเว็บเซอร์วิสที่ทำงานเป็นเสตท (เช่น กริดเซอร์วิส) มันจำเป็นที่จะต้องอธิบายสเตทที่ใช้กับส่วนภายนอกอย่างชัดเจน ซึ่งก็คือสเตทที่เซอร์วิสอินสแตนซ์ถูกแสดงเมื่อผู้ใช้สร้างเซอร์วิสขึ้นมานั่นเอง สิ่งที่จำเป็นสำหรับการประกาศเซอร์วิสเดต้าก็คือแนวความคิดที่จะประกาศคุณสมบัติในตัวมัน ซึ่งอธิบายได้จาก Interface Definition Language (IDL) โดยที่เซอร์วิสเดต้านั้นจะสามารถใช้ได้จากการอ่าน, อัพเดต หรือ ลงท้ายในข้อมูล

ซึ่ง WSDL นั้นได้กำหนด กระบวนการและข้อความสำหรับ portTypes ที่สเตทที่ถูกประกาศนั้นต้องมีการเข้าถึงได้จากภายนอกผ่านกระบวนการของเซอร์วิสที่ใช้กำหนดเซอร์วิสอินเตอร์เฟสเท่านั้น เพื่อหลีกเลี่ยงความต้องการที่จะกำหนด กระบวนการพิเศษของเซอร์วิสเดต้าสำหรับแต่ละเซอร์วิสเดต้า โดยที่กริดเซอร์วิสนั้น จะมี portType ที่ใช้จัดกระบวนการที่ใช้ทำเซอร์วิสเดต้าด้วยตัวเอง

ตัวอย่าง มีอินเตอร์เฟสที่มีกระบวนการ op1, op2, และ op3 อยู่ โดยสมมติให้อินเตอร์เฟสนี้มีข้อมูลที่ให้สาธารณะเข้าถึงได้คือ de1, de2, de3 ซึ่งผู้ใช้ได้กำหนดรายละเอียดของอินเตอร์เฟสนี้ด้วย WSDL ซึ่ง serviceData ใน OGSI นี้ จะสร้างสืบเนื่องมาจาก WSDL ซึ่งผู้ออกแบบจะสามารถเพิ่มรายละเอียดอินเตอร์เฟสนี้ได้โดยกำหนดส่วนที่เกี่ยวกับการยอมให้เข้าถึงแบบสาธารณะของแต่ละสเตทสำหรับ de1, de2, de3 โดยการประกาศนี้จะทำให้สะดวกต่อการใช้งานของกระบวนการต่างๆบนเซอร์วิสเดต้าของเซอร์วิสอินสแตนซ์แบบมีสเตทซึ่งใช้พัฒนาอินเตอร์เฟสนี้ขึ้นมา

4.4.1 การเปรียบเทียบกับ Java Bean

ในคุณสมบัติของ serviceData ที่ OGSI ได้กำหนดสเป็คให้เป็นส่วนที่เข้าถึงข้อมูลสเตทของเว็บเซอร์วิส โดยแนวคิดของ serviceData นั้น คล้ายคลึงกับของ public instance variable หรือ field ใน object-oriented programming language เช่น Java, Smalltalk หรือ C++

ServiceData นั้นคล้ายคลึงกับคุณสมบัติของ JavaBean ซึ่งโมเดลของ JavaBean นั้นได้กำหนดรูปแบบสำหรับการทำ method signatures (getXXX/setXXX) เพื่อเข้าถึงคุณสมบัติ และผู้ช่วยในการสร้างคลาส (BeanInfo) เพื่อทำคุณสมบัติของเอกสาร โดย OGSI ได้ใช้ส่วนของ serviceData และ XML สำหรับการทำในสิ่งที่คล้ายๆกันกับของ JavaBean

โดยในสเป็คของ OGSI นั้นไม่ได้เลือกใช้วิธี getXXX และ setXXX เพื่อใช้เป็นกระบวนการของ WSDL ที่นำมาใช้สำหรับแต่ละ serviceData แม้ว่าส่วนที่ใช้พัฒนาเซอร์วิสจะเลือกวิธี get และ set ด้วยตัวมันเองก็ตาม โดย OGSI ได้เลือกใช้กระบวนการ query แทนการ get และใช้วิธีการ update แทนการ set และได้ใช้ subscribe to notification สำหรับแก้ไขค่าใน serviceData ซึ่งสิ่งที่ OGSI ต้องการนั้นอย่างน้อยต้องรองรับวิธีการเหล่านี้ ซึ่งเป็นสิ่งที่อนุญาตให้เข้าถึง serviceData ในชื่อของมันของแต่ละเซอร์วิสอินสแตนซ์ อย่างไรก็ตามกระบวนการบางส่วนของ OGSI ก็นำบางส่วนมาจากอินเตอร์เฟสอื่นๆ เพื่อรองรับการคิวรี่, อัพเดต และลงท้าย แบบที่ยุ่งยากขึ้น เช่นการ คิวรี่หลายๆ serviceData ใน 1 เซอร์วิสอินสแตนซ์

โดยค่า serviceDataName ใน GridService portType นั้น ได้กำหนดไว้ตามรูปแบบเดียวกับของ BeanInfo ใน JavaBeans อย่างไรก็ตาม OGSI ได้เลือก XML (WSDL) สำหรับจัดเตรียมข้อมูลข่าวสารเกี่ยวกับ serviceData แทนที่การใช้โมเดลของ BeanInfo แบบตรงๆ

4.4.2 การแทนค่า portType ด้วย serviceData

ServiceData ที่ใช้กำหนดค่าของ portType นั้นจะเรียกว่า ServiceData Elements (SDEs) ซึ่งใน SDE จะใช้กำหนดการอ้างจาก ServiceData Declarations (SDDs) โดยค่าเริ่มต้นของ SDE จะกำหนดได้จากค่า staticServiceDataValues ภายใน portType โดยค่าใดๆของ SDE จะถูกกำหนดใน portType หรืออาจจะถูกใส่ค่าจากเว็บเซอร์วิสอินสแตนซ์

ส่วนนี้เป็นตัวอย่างการใช้ gwsdl:portType ซึ่งยอมให้มีค่าปรากฏจาก namespaces อื่นๆ

```
<gwsdl:portType name="NCName"> *
   <wsdl:documentation … /> ?
   <wsdl:operation name="NCName"> … </wsdl:operation> ?
…
   <sd:serviceData name="NCName" … /> *
   <sd:staticServiceDataValues>?
      <some element>*
   </sd:staticServiceDataValues>
…
</gwsdl:portType>
```

สำหรับตัวอย่างนี้ จะเป็นการประกาศ 2 SDEs ของ portType ด้วยชื่อ “tns:sdl” and “tns:sdl2” ซึ่งไม่ว่าเซอร์วิสอินสแตนซ์ใดๆที่สร้างจาก portType นี้จะต้องมีค่า 2 ค่านี้

```
<wsdl:definitions xmlns:tns="xxx" targetNamespace="xxx">
   <gwsdl:portType name="exampleSDUse"> *
      <wsdl:operation name=…> … </wsdl:operation>
…
      <sd:serviceData name="sd1" type="xsd:String"
                      mutability="static"/>
      <sd:serviceData name="sd2" type="tns:SomeComplexType"/>
…
      <sd:staticServiceDataValues>
         <tns:sdl>initValue</tns:sdl>
      </sd:staticServiceDataValues>
   </gwsdl:portType>
…
</wsdl:definitions>
```

4.4.2.1 โครงสำหรับการประกาศ serviceData

จากที่ sd:serviceData ใช้กำหนดค่าใน XMML ซึ่งปรากฏใน sd:serviceDataValue และ sd:serviceData ที่ได้ออกแบบไว้หลังจากกำหนด xsd:element ของ XML ซึ่งค่าใน sd:serviceData จะใช้ 5 แอททริบิวท์เหมือนกันอย่างที่แสดงใน xsd:element: ประกอบด้วย name, type, minOccurs, maxOccurs, และ nillable ซึ่ง xsd:element อื่นๆจะห้ามใช้ภายใน sd:serviceData ซึ่งค่า sd:serviceData จะอนุญาตให้ 2 ค่าพิเศษนี้เพิ่มเติมเข้าไป ก็คือค่า mutability และค่า modifiable

โครง XML ที่ใช้กำหนด sd:serviceData นั้น ดูได้ตามนี้

```
…
   targetNamespace =
     "http://www.gridforum.org/namespaces/2003/serviceData"
   xmlns:sd =
     "http://www.gridforum.org/namespaces/2003/03/serviceData"
…

<attributeGroup name="occurs">
   <attribute name="minOccurs"
              type="nonNegativeInteger"
              use="optional"
              default="1"/>
   <attribute name="maxOccurs">
      <simpleType>
         <union memberTypes="nonNegativeInteger">
            <simpleType>
               <restriction base="NMTOKEN">
                  <enumeration value="unbounded"/>
               </restriction>
            </simpleType>
         </union>
      </simpleType>
   </attribute>
</attributeGroup>

<complexType name="ServiceDataType">
   <sequence>
      <any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>
   </sequence>
   <attribute name="name" type="NCName"/>
   <attribute name="type" type="QName"/>
   <attribute name="nillable"
              type="boolean"
              use="optional"
              default="false"/>
   <attributeGroup ref="sd:occurs"/>
   <attribute name="mutability" use="optional" default="extendable">
      <simpleType>
         <restriction base="string">
            <enumeration value="static"/>
            <enumeration value="constant"/>
            <enumeration value="extendable"/>
            <enumeration value="mutable"/>
         </restriction>
      </simpleType>
   </attribute>
   <attribute name="modifiable" type="boolean" default="false"/>
   <anyAttribute namespace="##other" processContents="lax"/>
</complexType>
```

```
<element name="serviceData" type="sd:ServiceDataType"/>

<xsd:complexType name="ServiceDataValuesType">
   <xsd:sequence>
      <xsd:any namespace="##any" minOccurs="0"
                                 maxOccurs="unbounded" />
   </xsd:sequence>
</xsd:complexType>

<element name="serviceDataValues"
         type="sd:ServiceDataValuesType"/>

<element name="staticServiceDataValues"
         type="sd:ServiceDataValuesType"/>
```

โดยค่าของ แอททริบิวท์แต่ละส่วน serviceData คือตามนี้

- maxOccurs = (nonNegativeInteger | unbounded) : default to 1
  - ค่านี้แสดงค่าตัวเลขที่มากที่สุดของ serviceData ที่สามารถปรากฏได้ในค่า serviceDataValues ของเซอร์วิสอินสแตนซ์ หรือ ค่า staticServiceDataValues ของ portType
- minOccurs = nonNegativeInteger : default to 1
  - ค่านี้แสดงค่าตัวเลขที่น้อยที่สุดของ serviceData ที่สามารถปรากฏได้ในค่า serviceDataValues ของเซอร์วิสอินสแตนซ์ หรือ ค่า staticServiceDataValues ของ portType
  - ถ้าค่านี้เป็น 0 แสดงว่าค่าของ serviceData นี้เป็นแบบ optional
- name = NCName and {target namespace}
  - ชื่อของ serviceData ต้องเป็นเอกเทศกันระหว่าง sd:serviceData และ xsd:element ทั้งหมดที่ใช้ประกาศใน namespace เป้าหมายของ wsdl:definition element
  - การรวมกันของชื่อใน serviceData และ targetNamespace ของ wsdl:element จาก QName นั้นจะยอมให้มีการอ้างเอกเทศของค่า serviceData นี้
- nillable = boolean : default to false
  - ค่านี้จะแสดงว่า serviceData มีค่าที่เป็น nil (ซึ่งเป็นค่าที่มี xsi:nil เป็นค่า true)

  ตัวอย่าง

  <serviceDataElement name="foo" type="xsd:string"

  nillable=true" />

  <foo xsi:nil="true"/>
- type = QName
  - ค่านี้ใช้กำหนดชนิดของ XML ของ serviceData
- modifiable = "boolean" : default to false
  - ถ้าค่านี้เป็นจริง จะหมายความว่าเป็นผู้ร้องขอโดยตรงเพื่อขออัพเดตค่า serviceData ผ่านคำสั่ง serServiceData ซึ่งเป็นการสั่งแบบฝืนค่า cardinality (minOccurs, maxOccurs) และ mutability ถ้าค่านี้เป็นเท็จ จะหมายถึงค่าของ serviceData ควรจะถูก

พิจารณาแบบ read only เท่านั้น โดยผู้ร้องขอ แม้ว่าค่านั้นจะเปลี่ยนผลลัพธ์ของกระบวนการอื่นๆบนอินเตอร์เฟสนี้ก็ตาม

- mutability = “static” | “constant” |“extendable” | “mutable” : default to extendable
  - ค่านี้แสดงว่า serviceData จะสามารถเปลี่ยนได้อย่างไร

4.4.2.2 การใช้ serviceData โดยใช้ตัวอย่างจาก GridService portType

เพื่อแสดงว่า serviceData ถูกใช้อย่างไร จะอธิบายได้โดยโค้ดส่วนนี้

```
<wsdl:definitions …

<gwsdl:portType name="GridService" …>
   <wsdl:operation name=…> … </wsdl:operation>
   …

   <sd:serviceData name="interface" type="xsd:QName"
        minOccurs="1" maxOccurs="unbounded"
        mutability="constant"/>
   <sd:serviceData name="serviceDataName" type="xsd:QName"
        minOccurs="0" maxOccurs="unbounded"
        mutability="mutable" nillable="false"/>
   <sd:serviceData name="factoryHandle"
        type="ogsi:HandleType"
        minOccurs="1" maxOccurs="1"
        mutability="constant" nillable="true"/>
   <sd:serviceData name="gridServiceHandle"
        type="ogsi:HandleType"
        minOccurs="0" maxOccurs="unbounded"
        mutability="extendable"/>
   <sd:serviceData name="gridServiceReference"
        type="ogsi:ReferenceType"
        minOccurs="0" maxOccurs="unbounded"
        mutability="mutable"/>
   <sd:serviceData name="findServiceDataExtensibility"
        type="ogsi:OperationExtensibilityType"
        minOccurs="1" maxOccurs="unbounded"
        mutability="static"/>
   <sd:serviceData name="terminationTime" type="ogsi:terminationTime"
        minOccurs="1" maxOccurs="1"
        mutability="mutable"/>
   <sd:serviceData name="setServiceDataExtensibility"
        type="ogsi:OperationExtensibilityType"
        minOccurs="2" maxOccurs="unbounded"
        mutability="static" />
…
```

ส่วนนี้คือตัวอย่างการเซ็ตค่า serviceData สำหรับกริดเซอร์วิสอินสแตนซ์

```
...
    xmlns:crm="http://gridforum.org/namespaces/2002/11/crm"
    xmlns:tns="http://example.com/exampleNS"
    xnlns="http://example.com/exampleNS">

<sd:serviceDataValues>
    <ogsi:interface>crm:GenericOSPT</ogsi:interface>
    <ogsi:interface>ogsi:GridService</ogsi:interface>

    <ogsi:serviceDataName>ogsi:interface
    </ogsi:serviceDataName>
    <ogsi:serviceDataName>ogsi:serviceDataName
    </ogsi:serviceDataName>
    <ogsi:serviceDataName>ogsi:factoryHandle
    </ogsi:serviceDataName>
    <ogsi:serviceDataName>ogsi:gridServiceHandle
    </ogsi:serviceDataName>
    <ogsi:serviceDataName>ogsi:gridServiceReference
    </ogsi:serviceDataName>
    <ogsi:serviceDataName>ogsi:findServiceDataExtensibility
    </ogsi:serviceDataName>
    <ogsi:serviceDataName>ogsi:terminationTime
    </ogsi:serviceDataName>
    <ogsi:serviceDataName>ogsi:setServiceDataExtensibility
    </ogsi:serviceDataName>

    <ogsi:factoryHandle>someURI</ogsi:factoryHandle>

    <ogsi:gridServiceHandle>someURI</ogsi:gridServiceHandle>
    <ogsi:gridServiceHandle>someOtherURI</ogsi:gridServiceHandle>

    <ogsi:gridServiceReference>…</ogsi:gridServiceReference>
    <ogsi:gridServiceReference>…</ogsi:gridServiceReference>

    <ogsi:findServiceDataExtensibility
          inputElement="ogsi:queryByServiceDataNames" />

    <ogsi:terminationTime after="2002-11-01T11:22:33"
                          before="2002-12-09T11:22:33"/>

    <ogsi:setServiceDataExtensibility
          inputElement="ogsi:setByServiceDataNames" />
    <ogsi:setServiceDataExtensibility
          inputElement="ogsi:deleteByServiceDataNames" />

</sd:serviceDataValues>
```

4.4.3 Mutability

ค่า mutability บน serviceData นั้นจะเป็นส่วนที่แสดงว่าค่าของ serviceData นั้นจะเปลี่ยนได้อย่างไรในไลฟ์ไทม์

mutability = “static” แสดงว่า ค่า SDE ถูกแทนให้ด้วยการประกาศของ WSDL (staticServiceDataValue) และต้องคงมีค่า instance อื่นๆของ portType นั้น โดยค่า “static” นั้น เปรียบเสมือนค่าตัวแปรที่เป็นสมาชิกของคลาสในภาษาโปรแกรมมิ่ง

mutability = “constant” แสดงว่า SDE ถูกแทนให้โดยการสร้างกริดเซอร์วิสอินสแตนซ์และต้องไม่มีการเปลี่ยนแปลงระหว่างช่วงที่อยู่ในเวลาชีวิตของกริดเซอร์วิสอินสแตนซ์นั้นจนกระทั่งมีการเซ็ตค่า

mutability = “extendable” แสดงว่าอย่างน้อย 1 ค่าของ SDE นั้นเป็นส่วนหนึ่งของไลฟ์ไทม์ของกริดเซอร์วิสอินสแตนซ์ ซึ่งมีค่าใหม่เพิ่มขึ้นมาได้ แต่ค่าเหล่านั้นต้องไม่ถูกลบออกไป

mutability = “mutable” แสดงว่าค่าใดๆใน SDE อาจถูกลบไปบางช่วงเวลาและบางทีอาจถูกเพิ่มขึ้นมา

4.4.4 serviceDataValues

แต่ละเซอร์วิสอินสแตนซ์จะเกี่ยวข้องกับกลุ่มของ serviceData ซึ่งค่าใน serviceData จะถูกกำหนดโดยหลายๆ portType จากอินเตอร์เฟสของเซอร์วิสนั้น และ บางทีอาจจะมีการเพิ่มค่าระหว่างช่วงเวลาที่รันอยู่ ซึ่งเรียกกลุ่มของ serviceData ที่เกี่ยวข้องกับเซอรีวิสอินสแตนซ์นี้ว่า “serviceData set” ซึ่งบางที จะอ้างอิงกับกลุ่มของ serviceData ที่ ไปรวมกับค่าของ serviceData ที่ถูกประกาศไว้ด้วย อินเตอร์เฟสของ portType

แต่ละเซอร์วิสอินสแตนซ์นั้นต้องมีเอกสาร XML ที่เป็นแบบลอจิค พร้อมด้วยค่า serviceDataValues ที่บรรจุ serviceData ไว้ ซึ่งการเขียนเซอร์วิสนั้นจะอิสระที่จะเลือกใช้ว่าค่า SDE นั้นถูกเก็บไว้แบบไหน เช่น บางทีอาจจะไม่ได้เก็บไว้ในรูป XML แต่เก็บเป็นตัวแปรอินสแตนซ์ซึ่งอาจเปลี่ยนรูปจาก XML หรือ วิธีการเอนโค้ดแบบอื่นๆเท่าที่จำเป็น

4.4.5 การกำหนดค่าเริ่มต้นของ SDE

ในการใช้ค่าของ staticServiceDataValues นั้น portType จะต้องมีการประกาศค่าเริ่มต้นสำหรับ serviceData ใดๆก็ตามด้วย mutability ที่เซ็ตไว้แบบ “static” เท่านั้น แม้ว่า serviceData จะถูกกำหนดไว้แบบ local หรือต่อเนื่องมาจาก portTypes ค่าเริ่มต้นนั้นจะต้องกำหนดในหลายๆ portTypes ข้างในอินเตอร์เฟสนั้นๆ ตราบที่ค่าเริ่มต้นนั้นยังไม่เกินค่า maxOccurs ที่ระบุไว้ ตัวอย่างนี้จะเป็นการกำหนดค่าเริ่มต้น 2 ค่าสำหรับ tns:otherSD ให้เป็น “1” และ “2”

```
<wsdl:definitions xmlns:tns="xxx" targetNamespace="xxx">
   <gwsdl:portType name="otherPT">
      <wsdl:operation name=…> … </wsdl:operation>
…
      <sd:serviceData name="otherSD" type="xsd:String"
                  mutability="static" maxOccurs="unbounded"/>

      <sd:staticServiceDataValues>
         <tns:otherSD>initial value 1</tns:otherSD>
      </sd:staticServiceDataValues>
…
   </gwsdl:portType>

   <gwsdl:portType name="exampleSDUse" extends="tns:otherPT">
      <wsdl:operation name=…> … </wsdl:operation>
…
      <sd:serviceData name="sd1" type="xsd:String"
                  mutability="static" />
      <sd:serviceData name="sd2" type="tns:SomeComplexType"/>

      <sd:staticServiceDataValues>
         <tns:sd1>an initial value</tns:sd1>
         <tns:otherSD>initial value 2</tns:otherSD>
      </sd:staticServiceDataValues>
…
   </gwsdl:portType>
…
</wsdl:definitions>
```

4.4.6 การรวมกันของ SDE ภายในโครงสร้างของ portType

WSDL 1.2 นั้นจะแนะนำเกี่ยวกับหลายๆ portType และการออกแบบภายในขอบเขตของ gwsdl ซึ่ง portType จะสามารถสืบมาจาก 0 หรือมากกว่านี้จาก portTypes อื่นๆ โดยที่ไม่ใช่ความสัมพันธ์ตรงๆระหว่าง wsdl:Service และ portType ที่ถูกรับรองโดยโมเดลของเซอร์วิสในรูปแบบของ WSDL ซึ่งกลุ่มของ portType นั้นจะสร้างโดยเซอร์วิสที่สืบทอดมาจากค่าลูกของเซอร์วิสและค่าที่อ้างจากค่า port โดยค่า กลุ่มของ portTypes และ portTypes จะถูกกำหนดด้วยเซอร์วิสอินเตอร์เฟสที่สมบูรณ์

serviceData ที่ถูกกำหนดโดยเซอร์วิสนั้นเป็นเซ็ตที่รวมกันของ serviceData ที่รวมกันของแต่ละ portType ในอินเตอร์เฟสที่สมบูรณ์ที่สร้างมาจากเซอร์วิสอินสแตนซ์ เพราะว่าค่า serviceData จะเป็นเอกเทศจาก QName ซึ่งเป็นกลุ่มจะปรากฏขึ้นเพียงครั้งเดียวในเซ็ตของ serviceData ยกตัวอย่างเช่น ถ้า portType มีชื่อ “pt1” และ “pt2” ซึ่งทั้งคู่กำหนดโดย serviceData ที่ชื่อ “tns:sd1” และมี portType ที่ชื่อ “pt3” ซึ่งสืบทอดมาจาก “pt1” และ “pt2” โดยที่มีแค่ 1 serviceData เท่านั้นที่ชื่อ “tns:sd1”

พิจารณาได้จากตัวอย่างนี้

```
<gwsdl:portType name="pt1">
   <sd:serviceData name="sd1" … />
</gwsdl:portType>

<gwsdl:portType name="pt2" extends="pt1">
   <sd:serviceData name="sd2" … />
</gwsdl:portType>

<gwsdl:portType name="pt3" extends="pt1">
   <sd:serviceData name="sd3" … />
</gwsdl:portType>

<gwsdl:portType name="pt4" extends="pt2 pt3">
   <sd:serviceData name="sd4" … />
</gwsdl:portType>
```

ซึ่งกลุ่มของ serviceData ได้กำหนด 4 portTypes ดังที่แสดงไว้ดังตารางข้างล่าง

if a service implements…	its serviceData set contains…
Pt1	sd1
Pt2	sd1. sd2
Pt3	sd1. sd3
Pt4	sd1. sd2. sd3. sd4

ตารางที่ 4-2 แสดงกลุ่มของ serviceData

4.4.6.1 ค่าเริ่มต้นของ SDE แบบ static ภายในอินเตอร์เฟสของ portType

ค่าเริ่มต้นของ SDE จะสามารถรวมกันลงไปที่อินเตอร์เฟสของ portType ได้ อย่างไรก็ตาม ค่า cardinality ที่ต้องการ (minOccurs และ maxOccurs) จะต้องถูกกันไว้ เช่น ในตัวอย่าง เซอร์วิสอินสแตนซ์ที่สร้าง pt1 จะมีค่า <sd1> 1 </sd1> สำหรับ SDE ที่ชื่อ sd1

```
<gwsdl:portType name="pt1">
   <sd:serviceData name="sd1" minOccurs="1" maxOccurs="1"
                           mutability="static"/>
   <sd:staticServiceDataValues>
      <sd1>1</sd1>
   </sd:staticServiceDataValues>

</gwsdl:portType>
```

ต่อจากนี้จะเป็นการสร้าง pt2 โดยสืบทอดค่า <sd1> 1 </sd1> สำหรับ SDE ที่ชื่อ sd1 และจะมีค่า <sd2> 2 </sd> สำหรับ SDE ที่ชื่อ sd2

```
<gwsdl:portType name="pt2" extends="pt1">
   <sd:serviceData name="sd2" minOccurs="1" maxOccurs="1"
                           mutability="static"/>
   <sd:staticServiceDataValues>
      <sd2>2</sd2>
   </sd:staticServiceDataValues>
</gwsdl:portType>
```

ค่าเซอร์วิสอินสแตนซ์ที่สร้าง pt3 ก็จะมี 2 ค่าคือ <sd3> 3a </sd3> และ <sd3>3b</sd3> สำหรับ SDE ที่ชื่อ sd3 ซึ่งควรจะสืบทอดค่าจาก SDE ที่ชื่อ sd1 ด้วย

```
<gwsdl:portType name="pt3" extends="pt1">
   <sd:serviceData name="sd3" minOccurs="1" maxOccurs="unbounded"
                           mutability="static"/>
   <sd:staticServiceDataValues>
      <sd3>3a</sd3>
      <sd3>3b</sd3>
   </sd:staticServiceDataValues>
</gwsdl:portType>
```

ค่าเซอร์วิสอินสแตนซ์ที่สร้าง pt4 จะสืบทอดจากค่า sd1 ที่กำหนดโดย pt1 แต่การขาดไปของค่า staticServiceDataValues ที่แสดงให้เห็น โดยไม่มีค่าเริ่มต้นสำหรับ sd4 (แม้ว่ากำหนดจาก portType ที่สืบทอดมาจาก pt4)

```
<gwsdl:portType name="pt4" extends="pt1">
   <sd:serviceData name="sd4" minOccurs="0" maxOccurs="unbounded"
                           mutability="static"/>
</gwsdl:portType>
```

เซอร์วิสอินสแตนซ์ที่สร้าง pt5 จะไม่สามารถสร้างได้ ตั้งแต่ที่ไม่มีค่าเริ่มต้นสำหรับ sd5 และตั้งแต่ที่ค่า minOccurs มีค่ามากกว่า 0 โดยจะมีความผิดพลาดเกิดขึ้นและแสดงออกผ่านการสร้างอินสแตนซ์ โดย portType ชนิดนี้จะพบผ่านการกำหนด “abstract” portType ภายใน portTypes ที่มาจากการกำหนดค่าสำหรับ SDEs ด้วย minOccurs ที่มากกว่า 0

```
<gwsdl:portType name="pt5" extends="pt1">
   <sd:serviceData name="sd5" minOccurs="1" maxOccurs="unbounded"
                          mutability="static"/>
</gwsdl:portType>
```

เซอร์วิสอินสแตนซ์ที่สร้าง pt6 ก็ไม่สามารถสร้างได้ ตั้งแต่ที่ portType กำหนดค่าเพิ่มต้นของ SDE ที่ชื่อ sd1 (เรียกใช้ค่าที่สืบเนื่องมาจาก pt1) ซึ่งมีค่ามากกว่า maxOccurs ของ SDE ที่ชื่อ sd1 โดยจะมีความผิดพลาดแจ้งให้ทราบผ่านการสร้างอินสแตนซ์เช่นกัน

```
<gwsdl:portType name="pt6" extends="pt1">
   </sd:staticServiceDataValues>
      <sd1>6</sd1>
   <sd:staticServiceDataValues>
</gwsdl:portType>
```

เซอร์วิสอินสแตนซ์ที่สร้าง pt7 จะมีเซ็ตที่น่าสนใจของ serviceData อยู่ อย่างแรกคือ เป็นค่าเดี่ยว <sd1>1</sd1> สำหรับ SDE ชื่อ sd1 แม้ว่าจะมีการสืบทอด pt1 ผ่าน pt2 และ pt3 ซึ่งค่าเริ่มต้นของ sd1 จะไม่ซ้ำกัน โดยค่า <sd2>2</sd2> เป็นค่าเดียวสำหรับ SDE ที่ชื่อ sd2 ซึ่งสืบทอดมาจาก pt2 และ SDE ที่ชื่อ pt3 จะมีค่า 3 ค่า คือ <sd3>3a</sd3>, <sd3>3b</sd3> (สืบทอดมาจาก pt3) และ <sd3>7</sd3> ซึ่งกำหนดแบบ local นอกจากนี้ยังมีค่าแบบ local ที่กำหนดโดย SDE ที่ชื่อ sd7 คือ <sd7>7</sd7>

```
<gwsdl:portType name="pt7" extends="pt2 pt3">
   <sd:serviceData name="sd7" minOccurs="1" maxOccurs="1"
                          mutability="static"/>
   <sd:staticServiceDataValues>
      <sd7>7</sd7>
      <sd3>7</sd3>
   </sd:staticServiceDataValues>
</gwsdl:portType>
```

โดยสรุปแล้ว ค่า SDE จะรวมอินเตอร์เฟสของ portType ถ้าผลของ SDE ไปกระทบกับ cardinality (ค่าที่น้อยกว่า minOccurs หรือ มากกว่า maxOccurs) ของ SDE แล้ว จะมีการแจ้งความผิดพลาดขึ้นเมื่อเว็บเซอร์วิสนั้นได้ถูกสร้างขึ้น

4.4.7 ค่า serviceData แบบ dynamic

แม้ว่าหลายๆ serviceData จะถูกกำหนดโดยรูปแบบของเซอรีวิสแล้ว บางสถานการณ์ก็อาจจะใช้การเพิ่มหรือการลบ serviceData แบบ dynamic หรือ จากอินสแตนซ์มาช่วย นี่หมายความว่าบางการอัพเดตเป็นจะเป็นการสร้างแบบเจาะจง ตัวอย่างเช่น เซอร์วิสอินสแตนซ์จะมีกระบวนการสร้างค่าเซอร์วิสเคต้าใหม่ๆเพิ่มขึ้นมาได้

โดย GridService portType ที่ใช้กับ SDEs แบบ dynamic นั้น จะประกอบไปด้วยค่า serviceData ที่ชื่อ “serviceDataName” ที่ใช้แสดง serviceData ที่กำหนดในปัจจุบัน คุณสมบัติของอินสแตนซ์นี้จะตอบค่าซุปเปอร์เซ็ตของเซอร์วิสเคต้าที่ประกาศโดย GWSDL กลับไป โดยกำหนดรูปแบบของเซอร์วิส ที่ยอมให้ผู้ร้อง

ขอ ใช้ กระบวนการsubscribe ที่ใช้เปลี่ยน serviceDataSet และ ใช้กระบวนการ findServiceData ที่ใช้เลือกค่าปัจจุบันของ serviceDataSet

4.5 คุณสมบัติสำหรับแก่นของกริดเซอร์วิส

จะแบ่งออกเป็นกลุ่มๆของทุกๆกริดเซอร์วิส ดังนี้

1. Service Description และ Service Instance
2. รูปแบบของเวลาที่ใช้ใน OGSI
3. ค่าไลฟ์ไทม์ของ XML
4. รูปแบบการให้ชื่อและการจัดการการเปลี่ยนแปลง
5. การให้ชื่อกริดเซอร์วิสอินสแตนซ์
6. วัฏจักรชีวิตของกริดเซอร์วิส
7. การจัดการเมื่อเกิดกระบวนการผิดพลาด
8. กระบวนการที่ใช้สืบทอด

4.5.1 Service Description และ Service Instance

ใน OGSI นั้นได้แบ่งแยกระหว่าง description กับ instance ของกริดเซอร์วิส ดังนี้

- Grid Service Description จะเป็นส่วนที่ใช้บ่งบอกว่า client จะตอบโต้กับเซอร์วิสอินสแตนซ์ได้อย่างไร โดยที่ description นี้จะเป็นอิสระต่ออินสแตนซ์อื่นๆ ภายในเอกสารของ WSDL นั้น Grid Service Description จะมาจากการสืบทอดจาก portType (เช่น portType ที่อ้างอิงจากค่าพอร์ทลูกของ wsdl:service) ของอินสแตนซ์ พร้อมด้วย portType ที่เกี่ยวข้อง (รวมถึงการประกาศ SDE) , การผูก, ข้อความ และ การกำหนดชนิดต่างๆ

- Grid Service Description ที่ถูกใช้ในเวลาเดียวกันโดย Grid Service Instance นั้น จะมีลักษณะดังต่อไปนี้

1. เกิดเป็นรูปร่างจากที่มาจากการ description ของเซอร์วิสที่บ่งบอกว่าจะตอบโต้อย่างไร
2. มี Grid Service Handles อยู่ 1 ค่าหรือ มากกว่านั้น
3. มี Grid Service Reference อยู่ 1 ค่าหรือ มากกว่านั้นที่ใช้อ้างถึง

โดย service description จะใช้ในเบื้องต้นอยู่ 2 อย่าง คือ ใช้เพื่ออธิบายเกี่ยวกับเซอร์วิสซึ่งส่วนนี้จะสามารถใช้เครื่องมือช่วยในการสร้างขึ้นมาได้อย่างอัตโนมัติ และใช้เพื่อค้นหาเซอร์วิส เช่น เพื่อหาเซอร์วิสอินสแตนซ์ที่สร้างขึ้นมาโดยเฉพาะจาก service description หรือหา factory ที่สามารถสร้างอินสแตนซ์โดยเฉพาะจาก service description

โดยการบ่งบอกถึง service description นั้น ต้องมี 2 ส่วนคือ syntax และ semantics โดยที่ syntax จะอธิบายได้ด้วย WSDL portType และ semantics จะใช้แสดงผ่านชื่อที่ใช้ใน portType เช่น เมื่อกำหนดกริกเซอร์วิสขึ้นมา โดยใช้ชื่อที่เป็นเอกเทศขึ้นมาหลายๆชื่อ semantic จะพยายามที่จะเกี่ยวข้องกับแต่ละ ชื่อนั้นๆ โดยชื่อเหล่านี้จะสามารถใช้โดย client เพื่อค้นหาเซอร์วิสโดยความต้องการของ semantics โดยจะค้นหา

เซอร์วิสอินสแตนซ์และแฟคทอรี่ด้วยชื่อพิเศษนั้นๆ การใช้ขอบเขตของชื่อเหล่านี้จะเป็นการเตรียมสำหรับการใช้ชื่อที่จะเป็นเอกเทศต่อกัน

4.5.2 รูปแบบของเวลาที่ใช้ใน OGSI

ส่วนที่สำคัญส่วนหนึ่งที่เกิดขึ้นกับการใช้งานของหลายๆกลุ่มในการกระจายของกริดนั้น ก็คือ การแสดงเวลานั่นเอง ยกตัวอย่างเช่น มีข้อมูลที่ต้องส่งด้วย timestamps ตามลำดับ เพื่อให้ข้อมูลนั้นจะมีประโยชน์กับผู้ที่นำไปใช้ เพราะบางที client อาจจะจำเป็นต้องใช้เซอร์วิสอินสแตนซ์นั้นในช่วงเวลาที่ยังควรใช้งานอยู่ และหลายๆเซอร์วิสก็ต้องอาศัยความเข้าใจเรื่องลำดับให้ถูกต้องเพื่อให้ client สามารถจัดการและตอบโต้กลับมันได้ทันที

เวลามาตรฐานแบบ GMT นั้น ถูกนำมาใช้ในกริดเซอร์วิส ที่จะยอมให้เวลาโดยรวมกลมกลืนกัน อย่างไรก็ตามการใช้เวลา GMT เป็นมาตรฐานนั้นจะไม่ทำให้เกิดการ synchronization กับทุกๆเครื่องได้ เพราะเวลาของแต่ละเครื่องย่อมไม่ตรงกันทั้งหมด ซึ่งส่วนนี้จำเป็นต้องใช้วิธีพิเศษที่จะทำให้เวลานั้น synchronize กัน เพื่อคุณภาพของงาน

โดยกริดเซอร์วิสนั้นทั้งฝั่ง hosting environments และฝั่ง client จะต้องใช้เวลาร่วมกันผ่าน Network Time Protocol (NTP) หรือ อาจจะใช้ฟังก์ชั่นพิเศษที่จะทำให้เวลานั้นตรงตาม GMT อยู่เสมอ อย่างไรก็ตาม client และ service ต้องมีการยอมรับข้อความที่บรรจุค่าเกี่ยวกับเวลาไว้แล้วเกินขอบเขตบ้าง เพราะ การ synchronize นั้นไม่เพียงพอ ซึ่งอาจทำให้ข้อมูลที่สำคัญส่งไม่ถึงได้

ในบางกรณีก็อาจจะต้องการเวลาที่เป็น 0 หรือเป็นอนันต์ก็มี โดยเวลาที่เป็น 0 นั้นจะใช้แสดงแทนเวลาที่ผ่านไปแล้วในอดีต ส่วนเวลาที่เป็นค่าอนันต์จะใช้สำหรับแสดงความคิดเกี่ยวกับเวลา โดยในขอบเขตของชื่อใน ogsi นั้นจะแทนที่ค่าเวลาพิเศษนั้นๆใน xsd:dateTime ดังตัวอย่าง

```
… targetNamespace =
    "http://www.gridforum.org/namespaces/2003/03/OGSI"

<simpleType name="ExtendedDateTimeType">
   <union memberTypes="ogsi:InfinityType xsd:dateTime"/>
</simpleType>

<simpleType name="InfinityType">
   <restriction base="string">
      <enumeration value="infinity"/>
   </restriction>
</simpleType>
```

4.5.3 ค่าไลฟ์ไทม์ของ XML

ตั้งแต่ที่ serviceData จะต้องแทนไดนามิคสเตทของเซอร์วิสอินสแตนซ์ จึงเป็นเรื่องจำเป็นที่จะต้องเข้าใจเกี่ยวกับไลฟ์ไทม์ที่ถูกต้อง โดย client จะต้องใช้ข้อมูลที่เกี่ยวพันกับเวลาสำหรับการใช้ในเรื่องต่างๆ จนกระทั่งถึงการอิสระที่จะปฏิเสธข้อมูลนั้นๆด้วย

ซึ่งจะใช้ 3 แอททริบิวท์ของ XML ที่เป็นข้อมูลที่เกี่ยวข้องกับค่าไลฟ์ไทม์ คือ

- ogsi:goodFrom: ใช้กำหนดเวลาจากที่ซึ่งข้อมูลถูกต้อง ส่วนนี้ใช้เวลาตอนที่เริ่มสร้างมาใช้

- ogsi:goodUntil: ใช้กำหนดเวลาจนกระทั่งข้อมูลยังถูกต้อง ถ้าใช้ส่วนนี้จะมีคุณสมบัติที่ดีกว่าหรืออย่างน้อยเทียบเท่ากับแบบ goodForm

- ogsi:availableUntil: ใช้กำหนดเวลาจนกระทั่งถึงช่วงที่ค่าที่คาดหวังยังคงใช้ได้ บางทีอาจจะมีการอัพเดตค่านั้นด้วย ซึ่งเวลาแบบนี้นั้น client จะต้องได้รับค่าที่อัพเดตนี้ด้วย หลังจากนั้นแล้ว client ไม่ควรที่จะได้รับค่านี้อีกที เนื่องจากอาจมีปัญหากับกฎหรือสิ่งที่สำคัญอื่นๆ คุณสมบัติแบบนี้ต้องดีกว่าหรือเทียบเท่ากับแบบ goodFrom

ตัวอย่างการใช้ XML สำหรับแอททริบิวท์เหล่านั้น

```
targetNamespace = "http://www.gridforum.org/namespaces/2003/03/OGSI"

<xsd:attribute name="goodFrom" type="ogsi:ExtendedDateTimeType"/>
<xsd:attribute name="goodUntil" type="ogsi:ExtendedDateTimeType"/>
<xsd:attribute name="availableUntil" type="ogsi:ExtendedDateTimeType"/>

<xsd:attributeGroup name="LifeTimePropertiesGroup">
   <xsd:attribute ref="ogsi:goodFrom" use="optional"/>
   <xsd:attribute ref="ogsi:goodUntil" use="optional"/>
   <xsd:attribute ref="ogsi:availableUntil" use="optional"/>
</xsd:attributeGroup>
```

ซึ่งจะใช้ serviceData ค่านี้เพิ่มกำหนดไลฟ์ไทม์ตามนี้

```
<wsdl:definitions
   targetNamespace="http://example.com/ns"
   xmlns:n1="http://example.com/ns"
   … >

   <wsdl:types>
      <xsd:schema …
         "targetNamespace=http://example.com/ns"
         …>
         <xsd:complexType name="MyType">
            <xsd:sequence>
               <xsd:element name="e1" type="xsd:string" minOccurs="1"/>
               <xsd:element name="e2" type="xsd:string" minOccurs="1"/>
               <xsd:element name="e3" type="xsd:string" minOccurs="0"/>
            </xsd:sequence>
         <anyAttribute namespace="##any"/>
         </xsd:complexType>
      </xsd:schema>
   </wsdl:types>
   …
   <gwsdl:portType name="MyPortType">
      …
      <sd:serviceData name="mySDE" type= "n1:MyType"
                      minOccurs="1" maxOccurs="1"
                      mutability="mutable"/>
      …
   </gwsdl:portType>
   …
</wsdl:definitions>
```

และภายในเซอร์วิสอินสแตนซ์นั้นจะต้องมีค่า serviceDataValues ดังนี้

```
<sd:serviceDataValues
   <n1:mySDE
         goodFrom="2002-04-27T10:20:00.000-06:00"
         goodUntil="2002-04-27T11:20:00.000-06:00"
         availableUntil="2002-04-28T10:20:00.000-06:00">
      <n1:e1>
         abc
      </n1:e1>
      <n1:e2 ogsi:goodUntil="2002-04-27T10:30:00.000-06:00">
         def
      </n1:e2>
      <n1:e3 ogsi:availableUntil="2002-04-27T20:20:00.000-06:00">
         ghi
      </n1:e3>
   </n1:mySDE>
</sd:serviceDataValues>
```

โดยการใช้ goodFrom และ goodUntil ของ n1:mySDE นั้นจะอ้างถึงค่าลูกทั้งหมด ที่แอททริบิวท์เหล่านั้นกำหนดให้ใน SDE เป็นค่าไลฟ์ไทม์ที่คาดหวังไว้สำหรับค่านั้น ซึ่งในตัวอย่างนี้จะใช้จาก 10.20 am ไปจนถึง 11.20 am ในวันที่ 27 เดือนเมษายน ปี 2002 โดยผู้ที่นำไปใช้จะได้รับ SDE นี้ที่เวลา 10.20 am ดังนั้นแม้ว่าผู้ที่ใช้งานจะได้รับช้ากว่านั้นอีก 1 ชม. ก็จะสามารถใช้งานได้ถูกต้อง โดยที่ client บางทีอาจจะคิวรี่เซอร์วิสอินสแตนซ์นี้อีกครั้งเมื่อเวลา 11.20 เพื่อรับค่าใหม่ก็ได้

4.5.4 รูปแบบการให้ชื่อและการจัดการการเปลี่ยนแปลง

ส่วนสำคัญของระบบแบบกระจายนั้นก็คือต้องยอมรับค่าที่อ่จเปลี่ยนแปลงตลอดเวลาได้ เพื่อให้ได้สิ่งนี้แล้ว client จึงต้องการที่จะเลือกเซอรวิสที่เปลี่ยนแปลงไปได้เพื่อที่จะนำใช้ในการทำงานของเขา ส่วนนี้จะเป็นส่วนที่แสดงถึงการจัดการส่วนนี้ของ OGSI

4.5.4.1 ปัญหาของการจัดการการเปลี่ยนแปลง

ในกริดเซอร์วิสอินสแตนซ์นั้นจะมีการจำกัดความได้อยู่ 2 ลักษณะคือ

1. เป็นรูปแบบคุณสมบัติของมัน ซึ่งแบบนี้ กริดเซอร์วิสอินสแตนซ์จะกำหนดโดย service description ซึ่งประกอบไปด้วย portTypes, operations, serviceData, messages และ types

2. เป็นรูปแบบการทำงาน ส่วนนี้อาจจะแสดงมาจากแบบแรกด้วย โดยจะบอกว่าแท้จริงแล้วมันให้กริดเซอร์วิสอินสแตนซ์ได้อย่างไร ซึ่งรูปแบบการทำหรือความผิดพลาดจะแสดงผลออกมาที่เซอร์วิสอินสแตนซ์

สำหรับ client เพื่อที่จะค้นหาและใช้งานกริดเซอร์วิสอินสแตนซ์ได้อย่างถูกต้องนั้น client จะต้องเลือกจากคำจำกัดความของ 2 ชนิดที่กล่าวมา แต่เมื่อลองพิจารณาดูการที่ Grid service description บ่งบอกค่าออกมาเป็น portType นั้นเป็นสิ่งที่ client ต้องการจริงมั้ย? และก็ส่วนที่เป็นโครงของการทำงานนั้นจะไม่มีความผิดพลาดจนต้องแก้ไขเลยหรือ?

ต่อมา grid service description นั้นจำเป็นจะต้องมีออกมาอยู่ตลอดเวลา ถ้ากริดเซอร์วิสอินสแตนซ์นั้นสืบทอดมาจากส่วนข้างหลัง แล้ว client ต้องการส่วนนั้น ก็ควรต้องรองรับส่วนที่มันเป็นส่วนที่สืบทอดมาได้

ด้วย เช่น เพิ่ม operation หรือ service description ใหม่ๆขึ้นมา ซึ่งบางทีอาจเกิดจากการเปลี่ยนแปลงที่จำเป็น เช่น แก้ไขบั๊ก, ทำไว้ให้แก้ไขความผิดพลาดที่เกิดขึ้น เป็นต้น

อย่างไรก็ตาม client ต้องสามารถค้นพบ grid service description ที่เปลี่ยนแปลงที่ไม่ใช่ backward-compatible ได้ ซึ่ง backward-compatible นั้นจะทำให้เกิดการเปลี่ยนรูปแบบหรือการทำของกริดเซอร์วิสได้

4.5.4.2 ระเบียบการให้ชื่อของ Grid Service Description

ใน WSDL แต่ละ portType จะใช้ถูกใช้ทั่วไปและมีชื่อที่เป็นเอกเทศกันผ่านเงื่อนไขการมีชื่อของมัน ซึ่งจะมีการรวมกันของขอบเขตของชื่อที่บอกด้วย portType และมีค่าชื่อที่เป็นเอกเทศกันในแต่ละ ค่า portType ภายในขอบเขตของชื่อนั้นๆ ใน OGSI จะใช้การควบคุมการจัดการ โดยอาศัยค่าทั้งหมดของ grid service description ที่เปลี่ยนแปลงไม่ได้ โดยใช้ QName ของ Grid service prtType, operation, message, serviceData และ underlying type จะต้องกำหนดจาก WSDL/XSD ถ้าเปลี่ยนค่านี้ไปแล้ว portType ใหม่นั้นจะต้องกำหนดค่า QName ใหม่ด้วย นั้นหมายความว่าจะมีชื่อภายในใหม่ และมีขอบเขตชื่อใหม่ด้วย

ระหว่างพัฒนาระบบนั้น grid service instance จะสามารถเปลี่ยนแปลงได้ทั้งรูปแบบ, portType หรือแม้กระทั่งระดับการ implement ทำให้ผู้พัฒนาระบบต้องเลือกที่จะใช้งานมันอย่างระมัดระวัง ซึ่งก็ไม่น่ากังวลนักเพราะว่าจะการเปลี่ยนแปลงจะต้องผ่านการแสดงค่าของ portType ใหม่ๆนั้นอยู่แล้ว

4.5.5 การให้ชื่อกริดเซอร์วิสอินสแตนซ์

แต่ละกริดเซอร์วิสอินสแตนซ์นั้นจะมี อย่างน้อย 1 Grid Service Handles (GSH) ดังที่กล่าวไปช่วงต้น ซึ่ง GHS นั้นจะใช้ชื่อที่อยู่ในรูปแบบของ URI และไม่ได้เป็นส่วนที่เก็บข้อมูลว่าจะยอมให้ client ติดต่อตรงมาที่เซอร์วิสอินสแตนซ์อย่างไร โดยที่ client นั้นจะทราบว่าจะต้องติดต่อตรงกับเซอร์วิสอินสแตนซ์ไหนนั้นจะต้องผ่านการ resolve ค่า GSH ให้เป็น Grid Service Reference (GSR) ก่อน โดยที่ GSR จะเป็นส่วนที่เก็บข้อมูลที่จำเป็นที่จะใช้ในการติดต่อกับเซอร์วิสอินสแตนซ์ผ่านการผูกของโปรโตคอลในเครือข่ายนั้นๆ

เหมือนกับ URI โดย GSH จะเกิดจาก scheme โดยตามด้วยสตริงที่บรรจุข้อมูลพิเศษ โดย scheme นั้นจะแสดงว่ามันจะแสดงค่าอะไรบ้าง และ resolve จาก GSH ให้เป็น GSR ได้ยังไง โดย client จะเลือกที่จะ resolve GSH ด้วยตนเองหรืออาจจะเลือกโดยอาศัยแหล่งภายนอกก็ได้ เช่น แก้ไขวิธีการทำ HandleResolver ที่ portType เป็นต้น

รูปแบบของ GSR จะเป็นลักษณะพิเศษโดยใช้ผ่าน client เพื่อติดต่อสื่อสารกับกริดเซอร์วิสอินสแตนซ์ เช่น RMI/IIOP จะใช้มัน FSR จะเลือกรูปแบบของ IOR แต่ถ้าใช้ SOAP แล้ว GSR ก็จะเลือกรูปแบบของ WSDL มาใช้

ขณะที่ GSH จะถูกต้องในไลฟ์ไทม์ของกริดเซอร์วิสอินสแตนซ์นั้น ถ้า GSR มีการผิดพลาดจะทำให้ client ต้องการที่จะ resolve ใหม่เพื่อให้ได้ค่า GSR ที่ถูกต้อง

4.5.5.1 Grid Service Reference (GSR)

กริดเซอร์วิสอินสแตนซ์นั้นจะสร้างการเข้าถึงกับแอพพลิเคชั่นของ client ได้ผ่านการใช้ GSR โดยที่ GSR จะเป็น network-wide pointer เพื่อเข้าสู่กริดเซอร์วิสอินสแตนซ์ที่อยู่ในโฮสต์ที่พร้อมจะทำงาน โดยแอพพลิเคชั่นของ client นั้นจะสามารถใช้ GSR เพื่อส่งการร้องขอตรงไปที่อินสแตนซ์ที่กำหนดโดย GSR ซึ่ง GSR จะรองรับการเขียนโปรแกรมผ่านกริดเซอร์วิสอินสแตนซ์ที่ชื่อ "by reference" ซึ่ง GSR จะบรรจุข้อมูลที่จำเป็นสำหรับการที่จะเข้าถึงกริดเซอร์วิสอินสแตนซ์ในโฮสต์ได้ผ่านหลายๆ โปรโตคอล

การเอนโค้ดของ GSR นั้น บางทีจะได้รับจากหลายๆรูปแบบในระบบ เหมือนกับกระบวนการอื่นๆในระบบ โดยการเอนโค้ดที่แท้จริงนั้นจะผ่านรูปแบบของ GSR ที่เปรียบเสมือนการผูกผ่านเว็บเซอร์วิสด้วยกริดเซอร์วิสอินสแตนซ์ อย่างที่กำหนดไว้ในรูปแบบการเอนโค้ด WSDL ของ GSR ที่ใช้สำหรับการผูกในบางครั้ง

GSR จะใช้รูปแบบ "on the wire" จากการสร้าง GSR ในกริดเซอร์วิสโฮสต์ เมื่อส่วนที่ใช้อ้างได้ส่งไปที่พารามิเตอร์ที่เป็นของกระบวนการที่ใช้กำหนด WSDL แล้ว แอพพลิเคชั่นของ client จะส่งผ่านการร้องขอข้อความจาก WSDL-defined operation ซึ่งควรจะรวมทั้งหมดของที่อยู่ที่มีเซอร์วิสที่ต้องการจะสื่อสารนั้นผ่านการสร้างเซอร์วิสอินสแตนซ์ด้วย

ค่าใดๆของ GSR นั้นจะต้องคงอยู่ในระบบ โดยที่วัฏจักรชีวิตของ GSR จะต้องเป็นอิสระกับความเกี่ยวข้องของกริดเซอร์วิสอินสแตนซ์ ซึ่ง GSR จะยังคงถูกต้องตราบเท่าที่ เซอร์วิสอินสแตนซ์ยังคงสามารถใช้ได้ผ่าน GSR นี้

เมื่อ GSR ถูกพบว่าถูกต้องและได้ถูกสร้างจากกริดเซอร์วิสอินสแตนซ์ที่ยังคงอยู่ client จะได้รับค่า GSR ใหม่นั้นผ่านการใช้ GSH ของกริดเซอร์วิสอินสแตนซ์ที่เกี่ยวข้อง โดยที่ GSH จะบรรจุค่า binding-specific ที่ใช้สร้าง GSR ไว้ โดยที่ binding-specific ของ GSR นั้นจะรวมทั้ง เวลาที่หมดอายุด้วยที่จะประกาศเมื่อ client ต้องการ GSR นั้น ซึ่ง GSR จะต้องถูกต้องตลอดถึงช่วงเวลานั้นและควรจะผิดพลาดเมื่อเลยค่าเวลานั้น

ตัวอย่างโครง XML ที่ใช้กำหนด GSR ตามนี้

```
targetNamespace = "http://www.gridforum.org/namespaces/2003/03/OGSI"

<xsd:element name="reference" type="ogsi:ReferenceType"/>

<xsd:complexType name="ReferenceType" abstract="true">
   <xsd:attribute ref="ogsi:goodFrom" use="optional"/>
   <xsd:attribute ref="ogsi:goodUntil" use="optional"/>
</xsd:complexType>
```

4.5.5.1.1 WSDL ที่ใช้เอนโค้ด GSR

เป็นสิ่งที่จำเป็นที่ WSDL จะต้องเอนโค้ด GSR ที่บรรจุข้อมูลที่จำเป็นน้อยที่สุดสำหรับว่ามันจะต้องถึงกริดเซอร์วิสอินสแตนซ์นั้นได้อย่างไร โดย WSDL GSR จะมีค่า wsdl:definition ซึ่งบรรจุค่า wsdl:service ไว้และอาจจะมีค่าอื่นๆด้วย ดูได้จากโค้ดนี้

```
targetNamespace = http://www.gridforum.org/namespaces/2003/03/OGSI"

<xsd:complexType name="WSDLReferenceType">
   <xsd:complexContent>
      <xsd:extension base="ogsi:ReferenceType">
         <xsd:sequence>
            <xsd:any namespace="http://schemas.xmlsoap.org/wsdl/"
                 minOccurs="1" maxOccurs="1" processContents="lax"/>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
```

4.5.5.2 Grid Service Handles (GSH)

ต้องมีคุณสมบัติตามนี้

1. GSH ต้องมีค่า URI ที่ถูกต้อง
2. GSH จะต้องใช้ได้อย่างทั่วถึงเมื่อเกิดจากการอ้างกริดเซอร์วิสอินสแตนซ์เดียวกัน และ GSH ต้องไม่อ้างมากกว่า 1 กริดเซอร์วิสอินสแตนซ์
3. กริดเซอร์วิสอินสแตนซ์นั้นจะต้องมีอย่างน้อย 1 GSH
4. กริดเซอร์วิสอินสแตนซ์อาจจะมีหลายๆ GSH ได้ถ้าใช้ URI ที่ไม่เหมือนกัน
5. กริดเซอร์วิสอินสแตนซ์จะต้องไม่มีหลายๆ GSH ถ้าใช้ URI ที่เหมือนกัน
6. ค่าของ gridServiceHandle ของกริดเซอร์วิสอินสแตนซ์จะต้องมีค่าเฉพาะค่า SGH ที่อ้างถึงอินสแตนซ์นั้นเท่านั้น
7. อาจจะมีหลายๆ GSRs ที่อ้างไปที่กริดเซอร์วิสอินสแตนซ์อันเดียวกัน
8. ผลจาก GSH อันเดียวกัน อาจทำให้เกิด GSR ที่ต่างกันได้ โดย resolver จะตอบค่า GSR ที่ต่างกันผ่านค่า GSH เดียวกันที่เวลาต่างกัน และอาจจะส่งค่า GSR ที่ต่างกันไปให้ client ที่ต่างกันด้วย
9. GSH อาจจะไม่สามารถ resolve GSR ได้ ไม่ว่าจะก่อน,ในระหว่าง หรือหลังจากที่กริดเซอร์วิสอินสแตนซ์นั้นยังคงมีชีวิตอยู่ อย่างไรก็ตามโอกาสแบบนี้จะเกิดขึ้นไม่บ่อยนัก เนื่องจากรูปแบบของ URI ที่ใช้จะมีคุณภาพที่ค่อนข้างดีพอ
10. ผลจากการ resolve จาก GSH เป็น GSR นั้น บางทีอาจจะเชื่อถือได้หรืออาจเชื่อถือไม่ได้ก็เป็นได้ ขึ้นอยู่กับหลายๆอย่าง เช่น ค่าต่างๆที่ client เซ็ตไว้ หรือการจำกัดของโปรโตคอลที่ใช้ resolve

ตัวอย่าง XML ที่ใช้กำหนด GSH

```
targetNamespace = "http://www.gridforum.org/namespaces/2003/03/OGSI"

<xsd:element name="handle" type="ogsi:HandleType"/>

<xsd:simpleType name="HandleType">
   <xsd:restriction base="xsd:anyURI"/>
</xsd:simpleType>
```

4.5.5.3 เซอร์วิสโลเคเตอร์

เซอร์วิสโลเคเตอร์นั้นเป็นโครงสร้างที่บรรจุค่า GSH, ค่า GSR และค่าอินเตอร์เฟสของ QNames (portType) ไว้อย่างน้อย 0 หรือมากกว่านั้น โดยค่า GSHs และ GSRs ทั้งหมดนั้นจะอ้างอิงไปที่กริดเซอร์วิส

อินสแตนซ์เดียวกัน และอินเตอร์เฟสทั้งหมดนั้นต้องถูกพัฒนาขึ้นจากกริดเซอร์วิสอินสแตนซ์นั้น โดยโลเคเตอร์จะใช้เพื่อยอมรับ GSH หรือ GSR นั้น โดยการกำหนด XML สำหรับโลเคเตอร์ ทำตามนี้

```
targetNamespace = "http://www.gridforum.org/namespaces/2003/03/OGSI"

<xsd:element name="locator" type="ogsi:LocatorType"/>

<xsd:complexType name="LocatorType">
   <xsd:sequence>
      <xsd:element ref="ogsi:handle"
                   minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="ogsi:reference"
                   minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="interface" type="QName"
                   minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
```

4.5.6 วัฏจักรชีวิตของกริดเซอร์วิส

วัฏจักรชีวิตของทุกๆกริดเซอร์วิสอินสแตนซ์นั้นจะกำหนดโดยการสร้างและการทำลายของเซอร์วิสอินสแตนซ์นั้นๆ โดยวิธีที่แท้จริงของกริดเซอร์วิสอินสแตนซ์จะถูกสร้างหรือถูกทำลายโดยคุณสมบัติขั้นพื้นฐานของกลุ่มโฮสต์นั้นๆ แต่กระนั้นก็ยังมีกลุ่มของ portTypes ที่เกี่ยวข้องกันได้กำหนดว่า client จะติดต่อกับเหตุการณ์ในวัฏจักรชีวิตนี้ได้อย่างไร โดยจะแบ่งเป็นเหตุการณ์ย่อยๆดังนี้

- client จะร้องขอเพื่อสร้าง (creation) กริดเซอร์วิสอินสแตนซ์โดยการเรียก createService ของเซอร์วิสอินสแตนซ์ที่จะสร้างขึ้นโดย Factory portType หรือ จากวิธีพิเศษที่ใช้กำหนด เซอร์วิสใหม่ๆ เช่น NotificationSource portType ซึ่งเป็นเซอร์วิสของ factory ที่จะกล่าวถึงต่อไป
- client จะร้องขอเพื่อทำลาย (destruction) กริดเซอร์วิสอินสแตนซ์ ผ่าน client ใดๆโดย explicit destruction operation ที่รองรับโดย GridService portType ซึ่ง client ที่ลงทะเบียนนั้นจะสนใจกับกริดเซอร์วิสอินสแตนซ์เพียงช่วงเวลาหนึ่ง จนกระทั่งเวลานั้นหมดไป โดยที่เซอร์วิสอินสแตนซ์นั้นจะต้องถูกทำลายโดยอัตโนมัติ

กริดเซอร์วิสอินสแตนซ์จะรองรับการจัดการไลฟ์ไทม์แบบซอฟท์สเตท (Soft state) ในกรณีที่ client จะเริ่มค่าไลฟ์ไทม์นั้นเมื่อกริดเซอร์วิสอินสแตนซ์ได้ถูกสร้างขึ้นผ่าน factory และได้รับการยืนยันจากข้อความ requestTerminationBefore/After (“keepalive”) เพื่อร้องขอที่จะรับไลฟ์ไทม์ของเซอร์วิสนั้น ถ้าถึงเวลาสิ้นสุดแล้ว ฝั่ง server ที่เก็บเซอร์วิสอินสแตนซ์นั้นก็จะทำลาย และรีซอร์สที่เกี่ยวข้องทั้งหมดทิ้งไป

ซึ่งเวลาสิ้นสุดนั้นจะเปลี่ยนอย่างไม่ซ้ำกัน การที่ client ร้องขอเวลาสิ้นสุดนั้นจะเร็วกว่าขอเวลาในตอนปัจจุบัน นั่นคือ ถ้ามีการร้องขอเวลาสิ้นสุดโดยที่เวลานั้นเป็นก่อนเวลาปัจจุบัน นั่นคือจำเป็นต้องแสดงสิทธิเพื่อยกเลิกในครั้งนั้นๆเป็นพิเศษ

เวลาสิ้นสุดที่แสดงผ่าน OGSI ดูได้ ดังตัวอย่างนี้

```
targetNamespace = "http://www.gridforum.org/namespaces/2003/03/OGSI"
```

```
<xsd:complexType name="TerminationTimeType">
   <xsd:attribute name="after" type="ogsi:ExtendedDateTimeType"
                  use="optional"/>
   <xsd:attribute name="before" type="ogsi:ExtendedDateTimeType"
                  use="optional"/>
   <xsd:attribute name="timestamp" type="xsd:dateTime"
                  use="optional"/>
</xsd:complexType>

<xsd:simpleType name="ExtendedDateTimeType">
   <xsd:union memberTypes="ogsi:InfinityType xsd:dateTime"/>
</xsd:simpleType>

<xsd:simpleType name="InfinityType">
   <xsd:restriction base="string">
      <xsd:enumeration value="infinity"/>
   </xsd:restriction>
</xsd:simpleType>
```

โดยค่า after ของ TerminationTimeType นั้นจะบรรจุเวลาแรกสุดที่เซอร์วิสอินสแตนซ์วางไว้ว่าจะยกเลิก โดยมีค่าพิเศษคือ ค่าอนันต์ ที่ทำให้เซอร์วิสอินสแตนซ์นั้นจะคงอยู่อย่างไม่มีกำหนด

โดยค่า before ของ TerminationTimeType นั้นจะบรรจุเวลาสุดท้ายที่เซอร์วิสอินสแตนซ์วางไว้ว่าจะยกเลิก โดยมีค่าพิเศษคือ ค่าอนันต์ ที่ทำให้เซอร์วิสอินสแตนซ์นั้นจะคงอยู่โดยไม่มีการยกเลิก

โดยค่า timestamp ของ TerminationTimeType จะบรรจุเวลาที่เซอร์วิสอินสแตนซ์ได้เลือกไว้ว่าเวลาที่หลังจากและก่อนหน้าที่กำหนดไว้จะยังคงถูกต้องอยู่ โดย timestamp นั้นจะใช้ใน TerminationTimeType เพื่อใช้เป็นเกณฑ์ของเวลาที่จะเกี่ยวข้องกับ client หรือแหล่งเวลาอื่นๆ

4.5.7 การจัดการเมื่อเกิดกระบวนการผิดพลาด

OGSI จะกำหนด XSD ไว้สำหรับความผิดพลาดทั้งหมดที่กริดเซอร์วิสส่งกลับมาเพื่อใช้แก้ไขปัญหาต่างๆ โดยบรรจุข้อความที่ผิดพลาดต่างๆไว้ โดย ogsi:FaultType XSD ดังนี้

```
targetNamespace = "http://www.gridforum.org/namespaces/2003/03/OGSI"

<xsd:element name="fault" type="FaultType">

<xsd:complexType name="FaultType">
   <xsd:sequence>
      <xsd:element name="description"
                   type="xsd:string"
                   minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="originator"
```

```
                    type="ogsi:LocatorType"
                    minOccurs="1" maxOccurs="1"/>
      <xsd:element name="timestamp"
                    type="xsd:dateTime"
                    minOccurs="1" maxOccurs="1"/>
      <xsd:element name="faultcause"
                    type="ogsi:FaultType"
                    minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="faultcode"
                    type="ogsi:FaultCodeType"
                    minOccurs="0" maxOccurs="1"/>
      <xsd:element name="extension"
                    type="ogsi:ExtensibilityType"
                    minOccurs="0" maxOccurs="1"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="FaultCodeType">
   <xsd:simpleContent>
      <xsd:extension base="xsd:string">
         <xsd:attribute name="faultscheme" type="anyURI"
                        use="required"/>
      </xsd:extension>
   </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="ExtensibilityType">
   <xsd:sequence>
      <xsd:any namespace="##any"/>
   </xsd:sequence>
</xsd:complexType>
```

โดยค่าความผิดพลาดทั้งหมดจากกริดเซอร์วิสอินสแตนซ์นั้นจะใช้ FaultType ที่แก้ไขต่างๆกัน โดยค่าต่างๆที่ใช้ใน OGSI portTypes จะใช้ดังนี้

ค่า description จะบรรจุรายละเอียดความผิดพลาดเป็นภาษาตรงๆ กล่าวคือ เป็นส่วนอธิบายความผิดพลาดกับผู้ใช้งาน ค่านี้เป็นแบบ optional ไม่จำเป็นต้องมีก็ได้

ค่า originator เป็นตัวบอกตำแหน่งของเซอร์วิสอินสแตนซ์ที่ผิดพลาด

ค่า timestamp เป็นค่าเวลาที่ความผิดพลาดเกิดขึ้น

ค่า faultcause เป็นค่าใน ogsi:FaultType ใช้อธิบายผลลัพธ์ที่ผิดพลาด ใช้ร่วมกับ xsi:type ใช้เพื่อให้ทราบรายละเอียดเหตุผลของความผิดพลาด

ค่า faultcode ใช้ในการรายงานความผิดพลาดให้ระบบ โดยอาศัย faultscheme ที่แสดงเป็น URI

ค่า extension ใช้ระบุข้อความพิเศษที่เกี่ยวกับความผิดพลาดในรูปแบบขอบเขตชื่อของ XML ซึ่งบางทีจะเป็นค่าตัวเลขของค่าที่สืบเนื่องมา

โดยค่าความผิดพลาดทั้งหมดนั้นจะตอบกลับในรูป ogsi:fault ในการแจ้งความผิดพลาดดังนี้

```
<wsdl:definitions …>
   <types>
      <xsd:schema …>
         <xsd:complexType name="MyFaultType">
            <xsd:complexContent>
               <xsd:extension base="ogsi:FaultType/>
            </xsd:complexContent>
         </xsd:complexType>
         <xsd:element name="myFault" type="tns:MyFaultType"/>
      </xsd:schema>
   </types>

   <message name=myFaultMessage">
      <part name="fault" element="tns:myFault"/>
   </message>

   <gwsdl:portType …>
      <wsdl:operation …>
         <input …>
         <output …>
         <fault name="myFault" message="tns:myFaultMessage"/>
         <fault name="fault" message="ogsi:faultMessage"/>
      </wsdl:operation>
    </gwsdl:portType>
</wsdl:definitions>
```

4.5.8 กระบวนการที่ใช้ขยาย

หลายๆกระบวนการใน OGSI จะรับค่าที่ไม่มีรูปแบบสืบทอดกันมา เช่น รับค่าที่ให้ client ค้นหากระบวนการที่ถูกต้องเป็นต้น ดูได้จากส่วนที่เป็น NotificationSource::subscribe ที่ใช้เพื่อให้ client ถามหาเซอร์วิสอินสแตนซ์ที่ต้องการใช้ ของ serviceDataValues ที่เปลี่ยน อย่างไรก็ตามเซอร์วิสที่ใช้ทำ portType ที่สืบทอดมาจาก NotificationSource นั้นจะแสดงกำหนดโดยรูปแบบการคิวรี่ซึ่งมีประสิทธิภาพมากกว่า โดยส่วนนี้จะอธิบายถึง client จะเลือกการลงท้ายจาก NotificationSource portType ได้อย่างไร ดูได้จากตัวอย่างข้างล่างนี้

```
targetNamespace = "http://www.gridforum.org/namespaces/2003/03/OGSI"

<xsd:complexType name="OperationExtensibilityType">
  <xsd:attribute name="inputElement" type="QName" use="optional"/>
</xsd:complexType>
```

แต่ละ portType จะมีการกำหนด serviceData ของ OperationExtensibilityType และมีค่า mutability= “static” โดยค่า static ของ SDE นี้จะกำหนดด้วยค่าที่สืบทอดอย่างถูกต้อง ดังตัวอย่าง เช่น มี portType ชื่อ myPT ซึ่งมี myOperation อยู่ ซึ่งจะสร้างวิธีเรียกค่าอนพุทได้ 2 แบบ คือ myop:myOption1 และ myop:myOption2 ที่ผ่านเข้าไปสู่ส่วนที่สืบทอดจาก myOperation ได้ โดยการกำหนด myOperation ทำได้ตามนี้

```
<gwsdl:portType name="myPT">
   …
   <sd:serviceData name="myOperationExtensiblity"
           type="ogsi:OperationExtensiblityType"
           minOccurs=0 maxOccurs="unbounded"
           mutability="static"
           modifiability="false"
           nillable="false" />
   …
   <sd:staticServiceDataValues>
      <myOperationExtensibility inputElement="m1:myOption1"/>
      <myOperationExtensibility inputElement="m1:myOption2"/>
   </sd:staticServiceDataValues>
   …
</gwsdl:portType>
```

โดยค่า inputElement ของ SDE จะเป็นค่า QName ที่เป็นเอกเทศกัน โดยมาจากการประกาศของค่า XSD ที่กำหนดที่ค่าที่ถูกต้อง โดยคุณสมบัติทั้งหมดจะถูกแสดงโดย inputElement ของ OperationExtensibilityType เช่น inputElement ที่แสดงโดยค่าพารามิเตอร์จากกระบวนการที่แสดงว่ากระบวนนั้นจะรับค่า inputElement นั้นได้อย่างไร

ถ้า inputElement นั้นถูกละเว้นจากค่า SDE แล้ว มันจะต้องถูกแสดงด้วยการขยายค่าที่ผ่านการกระบวนการเรียกนั้น

ตัวอย่างกระบวนการที่รวมค่า portTypes ที่ขยายจากการประกาศ serviceData เช่น มี portType ชื่อ myPT2 ที่ขยาย myPT จะกำหนดค่าที่ถูกต้องให้กับ myOperation ตามนี้

```
<gwsdl:portType name="myPT2" extends="m1:myPT>
   …
   <sd:staticServiceDataValues>
      <myOperationExtensibility inputElement="m2:myOption3"/>
      <myOperationExtensibility/>
   </sd:staticServiceDataValues>
   …
</gwsdl:portType>
```

ในตัวอย่างนี้ เซอร์วิสอินสแตนซ์ที่ใช้ทำ myPT2 จะรองรับการขยายอินพุทของ myOperation: m1:myOption1, m1:myOption2, m2:myOption3 และ ไม่มีค่าที่ทั้งหมด โดยแต่ละกระบวนการจะเกี่ยวข้องกันกับ inputElement

4.6 อินเตอร์เฟสของกริดเซอร์วิส

ส่วนนี้เป็นส่วนที่ระบุเกี่ยวกับการรวมตัวของ WSDL portTypes และพฤติกรรมที่เกี่ยวข้องทั้งหมด โดยเป็นกลุ่มฟังก์ชั่นพื้นฐานของแนวทางการคิดแบบกระจาย โดยจะอธิบายในตารางด้านล่าง และจะแยกออกเป็นหัวข้อย่อย ในส่วนถัดไป

portType Name	Description
GridService	เป็นส่วนที่แสดงรูปแบบของเซอร์วิสนั้นๆ
HandleResolver	ทำการ โยงจาก GSH ให้เป็น GSR
NotificationSource	ยอมให้ client ลงท้ายข้อความประกาศ
NotificationSubscription	กำหนดความสัมพันธ์ระหว่างคู่ NotificationSource กับ NotificationSink
NotificationSink	กำหนดกระบวนการส่งข้อความประกาศไปที่เซอร์วิสอินสแตนซ์
Factory	เป็นกระบวนการมาตรฐานที่ใช้สร้าง กริดเซอร์วิสอินสแตนซ์
ServiceGroup	ยอมให้ client ดูแลกลุ่มของเซอร์วิส
ServiceGroupRegistration	ยอมให้กริดเซอร์วิสเพิ่มหรือลดออกจาก ServiceGroup
ServiceGroupEntry	กำหนดความสัมพันธ์ระหว่างกริดเซอร์วิสอินสแตนซ์กับสมาชิกภายใน ServiceGroup

ตารางที่ 4-3 แสดงอินเตอร์เฟสของกริดเซอร์วิส

ซึ่ง OGSI portType ทั้งหมดนี้กำหนดในขอบเขตชื่อของ OGSI

4.6.1 GridService portType

GridService portType จะเป็นส่วนที่ใช้สร้างกริดเซอร์วิสทั้งหมดและใช้จำกัดความพื้นฐานใน OGSI โดยลักษณะของการใช้ portType นั้นจะคล้ายคลึงกันกับการใช้อ็อบเจ็คคลาส เหมือนใน object-oriented programming language เช่น smalltalk หรือ Java ซึ่งจะซ่อนส่วนต่างๆเป็นองค์ประกอบย่อยๆ โดย GridService portType นั้นจะใช้ในการ คิวรี่และอัพเดต serviceData ของกริดเซอร์วิสอินสแตนซ์และใช้จัดการการยกเลิกอินสแตนซ์

ในรูปแบบของเว็บเซอร์วิสนั้น จะเลือกวิธีอย่างใดอย่างหนึ่งระหว่าง ใช้รูปแบบการส่งเอกสาร หรือ ใช้ remote procedure call (RPC) แต่กริดเซอร์วิสนั้นได้ถูกออกแบบให้อิสระ สามารถใช้งานทั้ง 2 รูปแบบผสมผสานกันได้

4.6.2 HandleResolver PortType

HandleResolver portType นั้นเป็นส่วนที่ใช้ resolve ค่า GSH ให้เป็น GSR โดยเป็นอิสระกับ URI ใดๆของ GSH ซึ่งค่าเซอร์วิสอินสแตนซ์นั้นจะทำผ่าน HandleResolver portType ที่เรียกว่า handle resolver

แต่ละ handle resolver นั้นจะมีวิธีการที่ต่างกันเกี่ยวกับการแม็พจาก GSH ให้เป็น GSR โดยบาง handle resolver นั้นจะใช้เซอร์วิสของส่วนที่จัดการเกี่ยวกับไลฟ์ไทม์ของ hosting environment มาใช้โดยตรง เช่น การสร้างและการทำลายอินสแตนซ์แบบอัตโนมัติ หรือ การเพิ่มและลบการแม็พ เมื่อเซอร์วิสอินสแตนซ์นั้นลงทะเบียนว่าคงอยู่กับ resolver แล้ว resolver จะคิวรี่ค่า gridServiceHandle และค่า gridServiceReference ของอินสแตนซ์นั้นเพื่อแม็พลงฐานข้อมูล

portType นี้จะสืบเนื่องมาจาก GridService portType

4.6.3 Notification

จุดประสงค์ของ notification คือส่งข้อความที่น่าสนใจจาก notification sourceไปที่ notification sink โดยจะอธิบายส่วนต่างๆไว้ตามนี้

- notification source คือ กริดเซอร์วิสอินสแตนซ์ที่ใช้ทำ NotificationSource portType และเป็นผู้ส่ง notification messages โดยที่แหล่งข้อมูลนั้นอาจจะส่ง notification message ไปที่ sink ใดๆก็ได้
- notification sink เป็นกริดเซอร์วิสอินสแตนซ์ที่ได้รับ notification message จากแหล่งข้อมูลใดๆ โดย sink จะใช้ทำ NotificationSink portType ซึ่งยอมให้รับค่า notification message
- notification message เป็นค่า XML ที่ส่งจาก notification source ไปยัง notification sink โดยค่า XML นี้จะถูกเลือกโดย subscription expression
- subscription expression เป็นค่า XML ที่ใช้อธิบายว่าข้อความนั้นควรจะส่งจาก notification source ไปยัง notification sink โดยขึ้นอยู่กับค่า serviceDataValues ของเซอร์วิสอินสแตนซ์
- ในการหาค่าว่า notification message อะไรจะต้องถูกส่งไปที่ไหนนั้น จะใช้การร้องขอค่า subscription ไปที่แหล่งข้อมูล ประกอบไปด้วย subscription expression, locator ของ notification sink ที่ notification message จะส่งไป และค่าเริ่มต้นของเวลาชีวิตสำหรับ subscription
- การร้องขอ subscription นั้นจะทำให้เกิดการสร้าง กริดเซอร์วิสอินสแตนซ์ ที่เรียกว่า subscription ที่ใช้สร้าง NotificationSubscription portType โดย portType นี้จะถูกใช้โดย client เพื่อจัดการไลฟ์ไทม์ของ subscription และค้นหาคุณสมบัติของค่า subscription

4.6.3.1 NotificationSource PortType

Notification portType จะยอมให้ client เพื่อลงท้าย notification message จากกริดเซอร์วิสอินสแตนซ์ที่ใช้ทำ portTypr นี้ โดยที่ NotificationSource portType จะขยายมาจาก GridService portType

4.6.3.2 NotificationSink portType

NotificationSink portType นั้นกำหนดโดยคำสั่งเดียวที่ใช้สำหรับส่ง notification message ไปให้เซอร์วิสอินสแตนซ์ที่ใช้ทำคำสั่งนั้น

4.6.4 Factory PortType

factory เป็นรูปแบบที่ตรงตามกริดเซอร์วิสที่ client สร้างขึ้นโดยกริดเซอร์วิสอินสแตนซ์อื่นๆ โดยที่ client จะเรียกสร้างคำสั่งบน factory และได้รับผลตอบรับเป็น locator ของเซอร์วิทอินสแตนซ์ใหม่ที่ถูกสร้างขึ้น โดย factory อาจเป็นกริดเซอร์วิสอินสแตนซ์ที่ใช้สร้าง Factory portType หรืออาจเป็นกริดเซอร์วิสอินสแตนซ์ที่สร้างคำสั่งของ factory โดยเฉพาะ เช่น NotificationSource::subscribe เป็นคำสั่งที่อธิบายไปในเรื่องที่แล้ว

บนการสร้าง factory นั้น กริดเซอร์วิสอินสแตนซ์ที่ถูกสร้างขึ้นใหม่นั้นจะถูกลงทะเบียนและได้รับค่า GSH ซึ่งเป็น handle resolution service ที่ใช้ในการลงทะเบียนกับ hosting environment

ซึ่ง Factory portType จะต้องสืบเนื่องมาจาก GridService portType

4.6.5 ServiceGroup portType

ServiceGroup เป็นกริดเซอร์วิสอินสแตนซ์ที่ดูแลข้อมูลข่าวสารเกี่ยวกับกลุ่มของกริดเซอร์วิสอื่นๆ เช่น กริดเซอร์วิสที่ใช้ลงทะเบียนจะกำหนดโดย portType ที่สืบเนื่องมาจากพฤติกรรมพื้นฐานจาก ServiceGroup ซึ่งมี 3 portType ที่ดูแลรูปแบบของกลุ่มของเซอรีวิสนี้ คือ ServiceGroup, ServiceGroupEntry และ ServiceGroupRegistration

ServiceGroup portType เป็นส่วนที่ใช้จัดเตรียมรูปแบบการแทน service group โดยมีอย่างน้อย 0 เซอร์วิสที่เป็นสมาชิกในกลุ่ม โดยค่า SDE ของ ServiceGroup นั้นจะประกอบไปด้วยค่าสำหรับแต่ละกริดเซอร์วิสที่เป็นสมาชิกในกลุ่ม โดยมีค่า ServiceGroupEntryLocator ใช้อ้างที่เซอร์วิสอินสแตนซ์ที่ใช้ทำ ServiceGroupEntry portType ซึ่งเตรียมฟังก์ชั่นควบคุมการดูแลไว้ โดยจะมีค่าที่เป็นเอกเทศไว้ใช้เป็นคีย์คือ GSH สำหรับแต่ละค่า

คุณสมบัติต่อไปนี้จะเป็นคุณสมบัติที่จำเป็นของ ServiceGroup, ServiceGroupEntry, ServiceGroupRegistration และ สมาชิกของกริดเซอร์วิสอินสแตนซ์ของ ServiceGroup

- กริดเซอร์วิสอินสแตนซ์จะต้องมีสมาชิกของหลายๆ ServiceGroups
- สมาชิกของกริดเซอร์วิสอินสแตนซ์ของ ServiceGroup จะต้องทำในหลายๆ portType ที่ต่างกัน
- ค่า ServiceGroupEntry จะต้องถอดออกจาก ServiceGroup โดยการจัดการไลฟ์ไทม์ของ ServiceGroupEntry
- เมื่อ ServiceGroup ถูกทำลาย client จะต้องไม่ได้รับผลกระทบจากการทำลายเซอร์วิส ServiceGroupEntry นั้น
- เมื่อ ServiceGroup ถูกทำลาย client จะต้องสามารถทราบเกี่ยวกับการไม่อยู่ของ ServiceGroupEntry หรือรายละเอียดที่ถูกต้องของมัน (เช่น คุณสมบัติของไลฟ์ไทม์)
- ServiceGroupEntry จะต้องเป็นของหนึ่ง ServiceGroup
- ถ้า GridService รวมค่าการยกเลิก ServiceGroup ไว้ด้วยแล้ว ServiceGroup จะต้องไม่สนใจมัน
- สมาชิกทั้งหมดของกริดเซอร์วิสอินสแตนซ์ของ ServiceGroup จะต้องทำตามกันอย่างน้อย 1 portType ที่เป็นสมาชิกใน membershipContentRule ของ ServiceGroup
- กริดเซอร์วิสอินสแตนซ์ที่เป็นสมาชิกจะต้องรวมอยู่ใน ServiceGroup โดยที่ ServiceGroup เปรียบเสมือนถุงที่รวบรวมค่าต่างๆไว้ โดยผู้ออกแบบได้ใช้สืบเนื่องมาจาก ServiceGroup ซึ่งเป็นเซ็ทของกลุ่มต่างๆ

บทที่ 5

Globus Toolkit 3

5.1 GT3 คืออะไร

GT3 (Globus Toolkit 3) นั้นเป็นเครื่องมือที่พัฒนาโดยภาษา Java ที่นำคุณสมบัติของ OGSI มาใช้ในการพัฒนา กล่าวคือ GT3 เปรียบเสมือนเฟรมเวิร์คที่ใช้ในการสร้างกริดเซอร์วิส

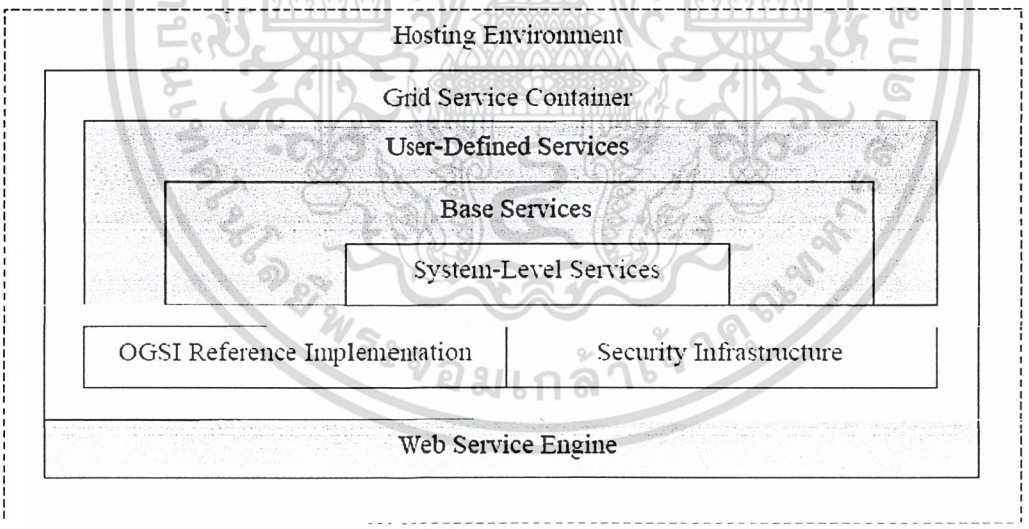
5.2 เซอร์วิสพื้นฐานที่จัดเตรียมโดย GT3

GT3 นั้นได้จัดเตรียมเซอร์วิสพื้นฐานไว้มากมาย ซึ่งแบ่งออกเป็นส่วนๆตามหน้าที่ ได้ดังต่อไปนี้

- แกน (Core)
- ด้านความปลอดภัย (GSI และ CAS)
- การจัดการข้อมูล (GridFTP, RLS, RFT, XIO)
- การจัดการรีซอร์ส (GRAM)
- ข้อมูลเซอร์วิส (MDS)

5.2.1 แกน (Core)

ภาพด้านล่างนี้ เป็นภาพโครงสร้างของ GT3 ซึ่ง องค์ประกอบของ Core จะถูกแทนในช่องสีขาว



รูปที่ 5-1 โครงสร้างของ GT3

5.2.1.1 Hosting Environment

คือ กลุ่มโฮสต์ที่ใช้ในการสร้างระบบกริด ได้จำแนกไว้ 4 ประเภทที่แตกต่างกัน คือ

1. Embedded โดยจะมีประโยชน์กับ client หรือ server ขนาดเล็กที่ให้ใช้งานกริดเซอร์วิส

2. Standalone มักใช้กับโปรแกรมขนาดเล็ก
3. Servlet ใช้พื้นฐานของ Java Servlet Engine มาเป็นมาตรฐานของ Container
4. EJB (Enterprise Java Bean) ใช้ EJB application server มาช่วยในการทำระบบกริด

ซึ่งทุกประเภทของ Hosting environment ที่กล่าวไปข้างต้น สามารถรองรับการทำกริดเซอร์วิสได้ โดยแบบที่ 1-3 นั้นจะเป็นการสร้างกริดเซอร์วิสแบบชั่วคราวจริงๆ แต่แบบที่ 4 จะชั่วคราวที่การทำงานในส่วนของ EJB

5.2.1.2 Grid Service Container

เป็นส่วนที่จะใช้เก็บเซอร์วิสต่างๆ ได้แบ่งการทำงานออกเป็น 3 ประเภท ดังนี้

1. คิดเกี่ยวกับเซอร์วิสขนาดเล็กๆ โดยใช้การค้นหา 2 แบบ คือ pull และ push
2. ควบคุมซอฟท์สเตทของเซอร์วิสอินสแตนซ์ที่ทำงานเป็นสเตทอย่างเต็มรูปแบบซึ่งอ้างได้อย่างทั่วถึง
3. ขนส่งส่วนที่เป็นอิสระกับโครงสร้างการทำงานที่เน้นความปลอดภัยตาม Grid Service Infrastructure โดยใช้ delegation, message signing และ encryption มาใช้ในการอนุญาต

5.2.1.3 OGSI Reference Implementation

เป็นส่วนที่ทำมาจาก OGSI ซึ่งเป็นนำคุณสมบัติทั้งหมดตามที่กำหนดไว้ใน OGSI มาช่วยในการสร้างระบบ เช่น GridService, Factory, Notification (Source/Sink/Subscription), HandleResolver, ServiceGroup (Entry/Registration) ตามที่กล่าวไปแล้วในบท OGSI

5.2.1.4 Security Infrastructure

เป็นส่วนที่ใช้เกี่ยวกับความปลอดภัยในระดับการส่งข้อความ โดยจะมีองค์ประกอบของความปลอดภัยระดับข้อความ, การรับรองสิทธิ และ การอนุญาตขึ้นกับกริดแม็พ ซึ่งสิ่งที่นำมาช่วยในข้อนี้ก็มี WS-Security, XML-Signature และ XML-Encryption

5.2.1.5 System Level Services

เป็นกริดเซอร์วิสที่ทำตาม OGSI ซึ่งสามารถใช้งานได้โดยโดยกริดเซอร์วิสตัวอื่นๆทั้งหมด ซึ่ง GT3 ในตอนนี้ ได้แบ่งเซอร์วิสส่วนนี้ออกเป็น 3 ระดับ

- Ping service: ใช้ในการ "ping" ไปยัง hosting environment เพื่อเช็คว่าเข้าถึงได้หรือไม่
- Logging Management Service: ยอมให้แก้ไขการกรอง log และใช้สร้าง log ที่คงอยู่เพื่อช่วยในการจัดการเวลาทำงาน
- Management Service: เป็นส่วนจัดเตรียมการมอนิเตอร์สถานะและการโหลดของกริดเซอร์วิสคอนเทนเนอร์ และเพื่อใช้ปิดคอนเทนเนอร์ และยังเป็นส่วนที่ยอมให้สั่งเปิดและปิดเซอร์วิสอินสแตนซ์

5.2.2 ด้านความปลอดภัย (GSI และ CAS)

5.2.2.1 GSI

Grid Security Infrastructure (GSI) เป็นการเข้ารหัสคีย์สาธารณะ เพื่อใช้ประโยชน์ในด้านความปลอดภัยต่างๆ โดยจุดประสงค์เบื้องต้นที่ทำให้เกิด GSI ขึ้นนั้น คือ

- ความต้องการการสื่อสารที่ปลอดภัย รับรองได้ บางทีต้องเป็นความลับระหว่างการสื่อสารที่ใช้ระบบกริด
- ความต้องการที่จะรองรับการใช้งานข้ามเขตของแต่ละองค์กร ภายใต้ระบบความปลอดภัยที่ควบคุมจากศูนย์กลาง
- ความต้องการที่จะรองรับ “single sign-on” สำหรับผู้ใช้งานกริด โดยที่การสื่อสารนั้น จะครอบคลุมไปถึงการใช้งานหลายๆรีซอร์สหรือจากหลายๆไซต์ได้

5.2.2.1.1 การเข้ารหัสคีย์สาธารณะ

ความสำคัญของการเข้ารหัสกับคีย์สาธารณะนั้นคือ มันไม่เหมือนกับระบบที่เป็นความลับอื่นๆ มันไม่ได้ขึ้นอยู่กับเพียงคีย์ๆหนึ่ง เหมือนกับ พาสเวิร์ดหรือโค้ดลับ แต่มันขึ้นกับคีย์ 2 คีย์ ที่ใช้ระหว่างการการเข้ารหัสข้อมูลและใช้กับการถอดรหัสข้อมูล ซึ่งเทคโนโลยีในตอนนี้ยังเป็นไปไม่ได้ที่จะได้ค่าคีย์อันที่ 2 ที่ใช้ในการถอดรหัสจากคีย์อันแรก ซึ่งทำให้จำเป็นต้องมีคีย์อันหนึ่งที่ใช้ได้กับสาธารณะ และมีคีย์อีกอันหนึ่งที่ใช้แบบส่วนตัว กล่าวคือถ้าข้อความนั้นสามารถถอดได้จากคีย์สาธารณะแล้ว จะต้องมีการใช้คีย์ส่วนตัวในการเข้ารหัสข้อความ

ประเด็นสำคัญก็คือ คีย์ส่วนตัวนั้นต้องเก็บเป็นการส่วนตัวจริงๆ เนื่องจากผู้ที่ทราบคีย์ส่วนตัวนั้นสามารถแสดงตนเป็นเจ้าของข้อความนั้นๆได้

5.2.2.1.2 ลายมือชื่อดิจิตัล

การใช้งานคีย์สาธารณะที่เข้ารหัสนั้น จำเป็นต้องใช้มีชื่อดิจิตัลเป็นส่วนหนึ่งของข้อมูลนั้น เพื่อทำให้ผู้รับข่าวสารนั้นวางใจได้ว่าข่าวสารนั้นไม่ได้ถูกที่อื่นทราบก่อนที่ผู้รับจะได้รับมัน

ในการใส่ชื่อในข้อความนั้นๆ ช่วงแรกได้ใช้หลักการแฮช (hash) ข้อมูลนั้น โดยเป็นค่าส่วนย่อของข้อความ ซึ่งอัลกอริทึมสำหรับการแฮชนั้นผู้รับจะต้องทราบมันด้วย (แต่ไม่ใช่ความลับ) การใช้คีย์ส่วนตัวและการใส่ข้อความแฮชนั้น จะทำให้แน่ใจว่าผู้รับมีคีย์สาธารณะของผู้ส่งด้วย

เพื่อพิสูจน์ว่าข้อความที่ลงชื่อนั้นเป็นสิ่งแท้จริง ผู้รับข้อความจะต้องมีการคำนวณข้อความแฮชโดยใช้อัลกอริธึมเดียวกับที่ผู้ส่งใช้ เพื่อจะถอดข้อความแฮชนั้นออกมาจากข้อความที่ได้รับ ถ้าการแฮชครั้งใหม่นั้นเข้าคู่กันกับข้อความที่ถอดจากการแฮช ก็แสดงว่าผู้รับได้รับข้อความนั้นโดยที่ไม่ได้ถูกเปลี่ยนแปลงแก้ไขระหว่างทางนั่นเอง

5.2.2.1.3 ใบรับรอง (Certificates)

แนวความคิดที่สำคัญในการรับรองสิทธิที่ใช้ใน GSI ก็คือ ใบรับรอง (Certificate) โดยผู้ใช้งานทุกคนและเซอร์วิสทุกๆอันจะต้องผ่านการพิสูจน์จากใบรับรอง ซึ่งบรรจุด้วยข้อมูลข่าวสารที่สำคัญที่ใช้รับรองผู้ใช้และเซอร์วิสต่างๆ

ซึ่ง ใบรับรอง ใน GSI นั้นจะมีข้อมูลเบื้องต้นอยู่ 4 ข้อมูล ดังนี้

- ชื่อวัตถุ ที่ใช้กำหนดบุคคลหรือสิ่งที่ใบรับรองนั้นแสดง
- คีย์สาธารณะที่เป็นของวัตถุนั้นๆ
- ค่าของ Certificate Authority (CA) ที่ใช้รับรองว่าค่าคีย์สาธารณะและค่า CA ทั้งคู่เป็นของวัตถุนั้น
- ลายมือชื่อดิจิตัลของชื่อ CA นั้นๆ

ซึ่งค่า CA จะใช้รับรองการเชื่อมต่อระหว่างคีย์สาธารณะและวัตถุของใบรับรองนั้น เพื่อที่จะเชื่อถือใบรับรองได้นั้น ค่า CA จะต้องเป็นแบบที่เชื่อถือได้ โดย GSI ได้ใช้การเข้ารหัสในรูปแบบ X.509 ซึ่งเป็นรูปแบบมาตรฐานสำหรับที่กำหนดโดย Internet Engineering Task Force (IETF) ซึ่งค่าใบรับรอง นั้นจะใช้มาตรฐานร่วมกับซอฟท์แวร์ที่ขึ้นอยู่กับคีย์สาธารณะ รวมถึงเว็บเบราส์เซอร์ที่ใช้ในทางธุรกิจ เช่น ของไมโครซอฟท์ หรือ เน็ทส์เคป เป็นต้น

5.2.2.1.4 การรับรองทั้ง 2 ฝ่าย (Mutual Authentication)

ถ้า 2 กลุ่มที่มีใบรับรองและถ้า 2 กลุ่มนั้นเชื่อ CA ที่มีชื่อของแต่ละใบรับรองนั้นๆแล้ว 2 กลุ่มนั้นจะสามารถพิสูจน์อีกกลุ่มได้ว่า เป็นใคร ทำอะไรได้บ้าง ซึ่งวิธีนี้เรียกว่า mutual authentication หรือการรับรองทั้ง 2 ฝ่าย โดยที่ GSI จะใช้ Secure Socket Layer (SSL) สำหรับการรับรองทั้ง 2 ฝ่ายนั้น

ก่อนการรับรองทั้ง 2 ฝ่ายจะเกิดขึ้น กลุ่มที่เกี่ยวข้องนั้นจะต้องเชื่อถือ CA ที่ใช้อ้างกับแต่ละ ใบรับรองก่อน อาจกล่าวได้ว่า พวกเขาจะต้องคัดลอกค่าใบรับรองของ CA นั้นๆไว้ ซึ่งมีค่าคีย์สาธารณะและต้องเชื่อถือใบรับรองนั้นๆซึ่งแท้จริงแล้วเป็นของ CA

ต่อไปนี้จะแสดงขั้นตอนของการรับรองทั้ง 2 ฝ่าย

- เพื่อเริ่มการรับรองทั้ง 2 ฝ่าย ฝ่ายบุคคลแรก (A) จะสร้างการเชื่อมต่อกับบุคคลที่สอง (B)
- เพื่อเริ่มกระบวนการรับรองซึ่งกันและกัน A จะส่งใบรับรองให้ B
- ใบรับรองจะบอก B เกี่ยวกับผู้ซึ่ง A เกี่ยวข้องด้วย, คีย์สาธารณะของ A และค่า CA ที่เคยถูกใช้เพื่อรับรองใบรับรองนั้นๆ
- B จะเริ่มเชื่อว่าใบรับรองนั้นถูกต้องโดยตรวจสอบรายมือชื่อดิจิตัลของ CA นั้น เพื่อให้มั่นใจว่าเป็น CA จริงที่ใช้รับรองและไม่เคยถูกเปลี่ยนแปลงค่าระหว่างทาง
- เมื่อ B ตรวจสอบใบรับรองของ A เรียบร้อยแล้ว B ต้องสร้างความมั่นใจว่า A เป็นบุคคลที่ใช้กำหนดผ่านใบรับรองนั้นจริงๆ
- B จะสร้างข้อความที่สุ่มออกมาและส่งไปให้ A เพื่อให้ A เข้ารหัสมัน
- A เข้ารหัสข้อความนั้นโดยใช้คีย์ส่วนตัวของมัน และส่งกลับไปยัง B

- B จะถอดรหัสข้อความนั้นโดยใช้คีย์สาธารณะของ A
- ด้วยผลลัพธ์ของข้อความสุ่มนั้น ทำให้ B จะรู้ว่า A คือบุคคลที่เขาพูดด้วย
- ตอนนี้ B จะเชื่อค่าของ A และจะต้องมีการทำงานที่กลับกันกับส่วนนี้ที่ทำให้ A เชื่อ B เกิดขึ้น
- B ส่งใบรับรองให้ A โดย A จะตรวจสอบความถูกต้องของใบรับรองและส่งข้อความชวนให้เข้ารหัสให้
- B จะเข้ารหัสข้อความและส่งกลับไปยัง A และ A จะถอดรหัสเพื่อนำมาเปรียบเทียบกับของดั้งเดิม
- ถ้าเข้าคู่กันแล้ว A จะทราบว่า B เป็นบุคคลที่เขาพูดด้วย
- ที่จุดนี้ A และ B จะเริ่มสร้างการเชื่อมต่อระหว่างกันและกัน และจะแน่ใจว่าพวกเขาจะรู้ค่าของกันและกัน

5.2.2.1.5 การสื่อสารที่เป็นความลับ

โดยปกติแล้ว GSI จะไม่สร้างการสื่อสารแบบลับระหว่างกลุ่ม เพราะเมื่อเริ่มการรับรองทั้ง 2 ทางแล้ว การสื่อสารจะเชื่อถือได้ว่าไม่เกิดความผิดพลาดระหว่างค่าคงที่สำหรับการเข้ารหัสและการถอดรหัส

GSI สามารถสร้างคีย์ร่วมกันสำหรับการเข้ารหัสลับได้โดยง่ายถ้าต้องการ ซึ่งที่ใช้ตอนนี้จะรวมการสื่อสารที่เข้ารหัสเป็นมาตรฐานของการสื่อสารพื้นฐานของ GSI แล้ว

ความเกี่ยวข้องกับส่วนหนึ่งความปลอดภัยคือความมั่นคงของการสื่อสาร ซึ่งหมายความว่าแม้จะถูกผู้อื่นอ่านการสื่อสารนั้น แต่จะไม่สามารถแก้ไขการสื่อสารนั้นๆ ได้ โดย GSI จัดเตรียมด้านความมั่นคงของการสื่อสารไว้ตามปกติอยู่แล้ว (สามารถปิดมันได้ถ้าต้องการ) ซึ่งความมั่นคงด้านการสื่อสารย่อมทำให้โอเวอร์เฮดของการสื่อสารเพิ่มขึ้นแต่ก็ไม่ได้ใหญ่เกินเท่าการเข้ารหัส

5.2.2.1.6 คีย์ส่วนตัวที่ปลอดภัย

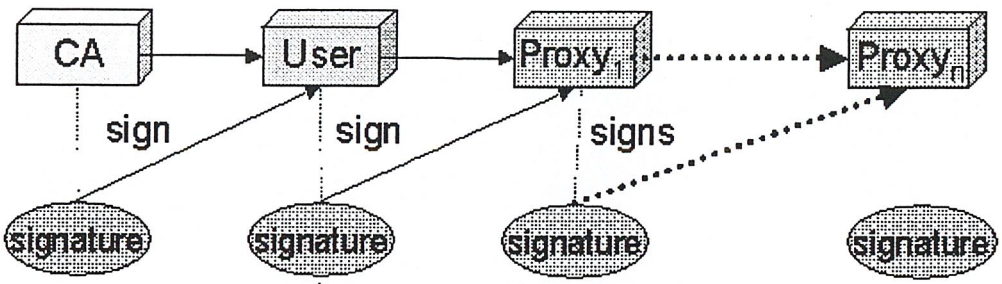
ซอฟท์แวร์ส่วนสำคัญของ GSI ที่ถูกจัดเตรียมโดย Globus Toolkit นั้น จะคาดหวังไว้ว่าคีย์ส่วนตัวของผู้ใช้นั้นจะต้องเก็บในไฟล์ที่อยู่ในแหล่งเก็บข้อมูลที่เป็นโลคอล เพื่อป้องกันผู้ใช้อื่นๆขโมยคีย์ส่วนตัวนั้นไป ซึ่งไฟล์นั้นจะต้องประกอบด้วยคีย์ที่เข้ารหัสผ่านพาสเวิร์ด (เรียกว่า passphrase) เพื่อที่จะใช้ใน GSI แล้วผู้ใช้ต้องใส่ค่า passphase ที่ต้องการเข้ารหัสกับไฟล์ที่เก็บคีย์ส่วนตัว

5.2.2.1.7 Delegation และ Single Sign-On

GSI ได้จัดเตรียมความสามารถด้านการ delegation ไว้ ซึ่งได้ขยายมาจากโปรโตคอลมาตรฐาน SSL เพื่อลดจำนวนเวลาที่ผู้ใช้จะต้องใส่ passphrase ถ้าการคิดแบบกริดนั้นต้องการหลายรีซอร์สที่เคยใช้ (ที่ต้องการในการรับรองทั้ง 2 ทาง) หรือต้องการผู้แทน (local หรือ remote) ที่ร้องขอเซอร์วิสที่เป็นประโยชน์กับผู้ใช้ ความจำเป็นที่จะให้ใส่ passphrase ของผู้ใช้อีกครั้งนั้น สามารถหลีกเลี่ยงได้ด้วยการสร้างพร็อกซี่ (proxy)

พร็อกซี่ เกิดขึ้นจากใบรับรองใหม่(ที่มีคีย์สาธารณะข้างในมัน) และมีคีย์ส่วนตัวใหม่ โดยที่ใบรับรองใหม่อันนั้นจะเป็นค่าของเจ้าของ โดยส่วนที่ดัดแปลงให้ต่างออกไปจากเดิมนั้นคือ พร็อกซี่ ซึ่งใบรับรองใหม่จะ

ถูกลงชื่อโดยเจ้าของ มากกว่าจาก CA (ดูได้จากรูปด้านล่าง) ใบรับรองจะรวมค่าเวลาซึ่ง พร็อกซี่ไม่ควรจะถูกยอมรับโดยส่วนอื่นๆด้วย กล่าวคือ พร็อกซี่นั้นต้องมีไลฟ์ไทม์ที่จำกัด



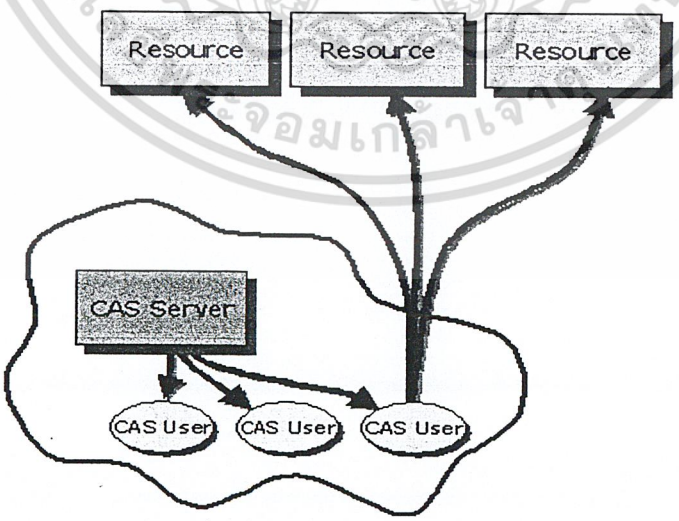
รูปที่ 5-2 แสดงวิธีการ Delegation และ Single Sign-On

คีย์ส่วนตัวของพร็อกซี่ต้องเก็บเป็นความลับ แต่เพราะพร็อกซี่ไม่ได้ถูกต้องในเวลาที่นานมากนัก ทำให้มันไม่ต้องเก็บให้ปลอดภัยเท่ากับค่าคีย์ส่วนตัวของเจ้าของ ซึ่งเป็นไปได้ที่จะเก็บค่าคีย์ส่วนตัวของ พร็อกซี่บนแหล่งเก็บข้อมูลแบบโลคอลโดยปราศจากการเข้ารหัส เท่าที่การอนุญาตของไฟล์จะต้องการปกป้องเพื่อไม่ให้หาได้โดยง่าย เมื่อพร็อกซี่สร้างและเก็บไว้แล้ว ผู้ใช้จะใช้พร็อกซี่สำหรับรับรองการสื่อสารทั้ง 2 ทางโดยปราศจากการใส่พาสเวิร์ด

เมื่อพร็อกซี่ถูกใช้ กลุ่มที่อยู่ระยะไกลนั้นจะไม่ได้รับเพียงแค่ใบรับรองของพร็อกซี่ แต่จะได้รับใบรับรองของเจ้าของด้วย ระหว่างการรับรองทั้ง 2 ทางนั้น คีย์สาธารณะของเจ้าของจะถูกใช้เพื่อตรวจสอบลายมือชื่อของใบรับรองพร็อกซี่ โดยคีย์สาธารณะของ CA จะถูกใช้เพื่อตรวจสอบใบรับรองของเจ้าของ สิ่งนี้จะสร้างโซ่แห่งความเชื่อมั่นสำหรับพร็อกซี่ผ่านไปยังเจ้าของได้

5.2.2.2 CAS

CAS สร้างขึ้นโดย Globus Toolkit ใช้ใน GSI ซึ่ง CAS เป็นส่วนที่ยอมให้ส่วนที่จัดเตรียมรีซอร์ส ยอมให้มีการสื่อสาร โดยควบคุมด้วยนโยบายสำหรับการเข้าถึงมัน ซึ่งจะใช้หลักของ delegation ในการควบคุมการจัดการในส่วนนี้



รูปที่ 5-3 แสดงการทำงานของ CAS

การทำงานของ CAS

1. CAS Server จะเริ่มสร้างช่องทางการสื่อสาร โดยจะได้มาซึ่ง GSI ที่เตรียมไว้สำหรับให้ CAS server กำหนดการสื่อสาร
2. ตัวจัดเตรียมรีซอร์สจะได้สิทธิสำหรับการสื่อสารนั้นๆ โดยแต่ละตัวจัดเตรียมนั้นจะเปรียบเสมือนผู้ถือครองการสื่อสารและมีนโยบายการสื่อสารเป็นของตนเอง เมื่อสร้างการเชื่อมต่อที่เชื่อถือได้แล้ว ตัวจัดเตรียมรีซอร์สก็จะยอมให้มีหลักฐานสำหรับการสื่อสารนั้น โดยใช้วิธีการภายในของมัน (เช่น ใช้ กริดแม็พไฟล์และดิสก์โควต้า, การยินยอมจากระบบไฟล์ เป็นต้น)
3. ตัวแทนการสื่อสารนั้นจะใช้ CAS เพื่อจัดการความเชื่อถือได้ของการสื่อสารนั้น และยอมให้ได้รับสิทธิการเข้าถึงรีซอร์สนั้น โดย CAS server จะมีนโยบายของมันที่ใช้ในการจัดการ เช่น สมาชิกของการสื่อสารต้องการสิทธิเพื่อเพิ่มสมาชิกของการสื่อสารนั้น หรือ ขอสิทธิสำหรับเข้าถึงรีซอร์สทั้งหมดของการสื่อสารนั้น
4. เมื่อผู้ใช้งานต้องการที่จะเข้าถึงรีซอร์สได้ร้องขอไปยัง CAS server แล้ว ถ้าฐานข้อมูลของ CAS server ยอมให้ผู้ใช้นั้นใช้งาน CAS จะแจก GSI restricted proxy เพื่อให้ผู้ใช้สามารถทำได้ตามที่ร้องขอไว้
5. เมื่อผู้ใช้ได้รับการเชื่อมต่อกับรีซอร์สของ CAS ด้วยเครื่องมือปกติของ Globus แล้ว (เช่น GridFTP) รีซอร์สที่ผ่านการยอมรับก็จะส่งไปยังผู้ที่ได้รับการยินยอมให้เข้าถึง และได้รับสิทธิการเข้าถึงตามที่จำกัดไว้ตามนโยบายของ CAS โดยจะมีข้อจำกัดตามแต่สิทธิของผู้ใช้แต่ละคนซึ่ง CAS ยินยอมให้ตัวจัดเตรียมรีซอร์สสื่อสารด้วย

5.2.3 การจัดการข้อมูล (GridFTP, RLS, RFT, XIO)

5.2.3.1 GridFTP

GridFTP เป็นโปรโตคอลที่ใช้ในการส่งข้อมูลไปบน wide-area networks ที่มีประสิทธิภาพ, ปลอดภัย และเชื่อถือได้ ซึ่ง GridFTP นั้นมีพื้นฐานมาจาก FTP ซึ่งเป็นโปรโตคอลที่นิยมใช้ในการส่งข้อมูลในระบบอินเตอร์เน็ต ซึ่งผู้พัฒนาได้เพิ่มและขยายส่วนต่างๆที่จำเป็นสำหรับการใช้ในระบบกริด

ซึ่งฟีเจอร์ที่ GridFTP จัดเตรียมไว้ในระบบกริด มีดังต่อไปนี้

- ทำงานร่วมกับ GSI security เพื่อควบคุมช่องทางข้อมูลให้ปลอดภัย
- ส่งข้อมูลแบบหลายๆทางแบบขนานไปพร้อมๆกัน
- เลือกส่งไฟล์บางส่วน
- ส่งไปยังกลุ่มที่ 3 (ส่งตรงจาก server ไปยัง server)
- รับรองช่องทางข้อมูล
- ใช้ช่องทางข้อมูลใหม่
- ทำไปป์ไลน์ของคำสั่งได้

5.2.3.2 RLS

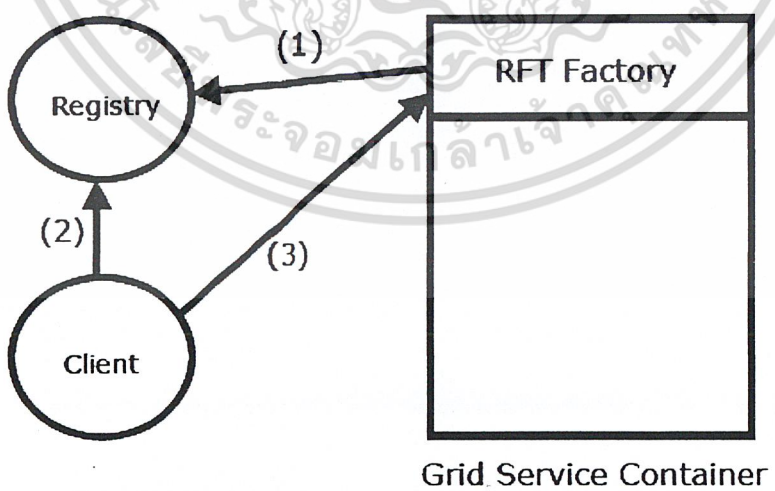
Replica Location Service (RLS) เป็นส่วนที่ดูแลและจัดเตรียมการเข้าสู่การแม็ฟจากข้อมูลที่เป็นลอจิคให้เป็นค่าข้อมูลที่จะใช้ที่เป้าหมาย ซึ่งส่วนนี้ได้พัฒนาโดยทีมที่ทำเรื่อง DataGrid ซึ่งจะช่วยลดเวลาในการเข้าถึงข้อมูล เพิ่มการวางตำแหน่งที่ตั้งของข้อมูล และเพิ่มความแข็งแรง, ประสิทธิภาพของแอพพลิเคชั่นแบบกระจาย

RLS นั้นจะมีหน้าที่ในด้านต่อไปนี้

- Consistent local state maintained in Local Replica Catalogs (LRCs): จะใช้ดูแลการแม็พระหว่าง arbitrary logical file names (LFNs) และ psysical file names (PFNs) โดยเกี่ยวข้องกับระบบของอุปกรณ์การเก็บข้อมูล
- Collective state with relaxed consistency mainained in Replica Location Indices (RLIs): แต่ละ RLI จะประกอบไปด้วยกลุ่มของการแม็พจาก LFNs เป็น LRCs
- Soft state maintenance of RLI state: LRC จะส่งข้อมูลที่เกี่ยวกับสเตทของ RLIs โดยใช้ซอฟท์สเตทโปรโตคอล ซึ่งค่าข้อมูลสเตทนั้นจะต้องรีเฟรทอย่างสม่ำเสมอ
- Compression of state update: ใช้รวมเนื้อหาทั้งหมดของ Local Replica Catalog ก่อนที่จะส่งไปยังการอัพเดตแบบซอฟท์สเตทที่ Replica Location Index Node
- Membership and partitioning information maintenance: RLS จะใช้ดูแลข้อมูลที่เป็นแบบ static ที่เกี่ยวกับ LRCs และ RLIs ในระบบแบบกระจาย ซึ่งจะใช้กลไกของ OGSA สำหรับการลงทะเบียนเซอร์วิสและใช้จัดการช่วงเวลาชีวิตของเซอร์วิสนั้น

5.2.3.3 RFT

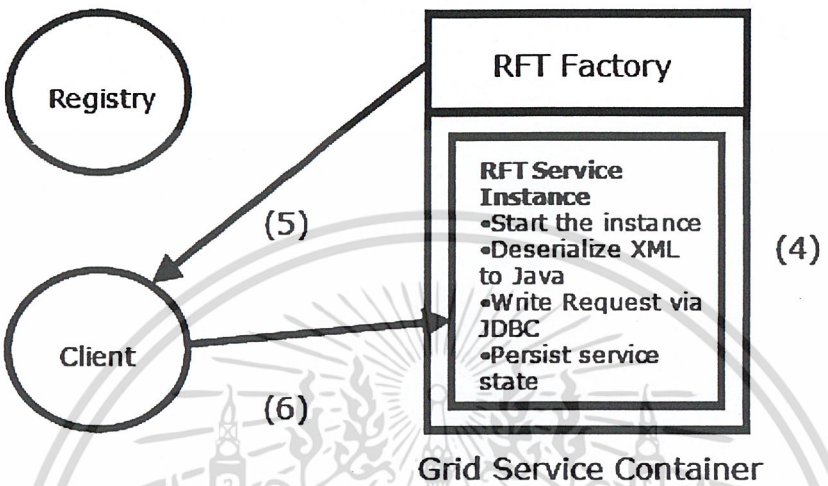
Reliable File Tranfer (RFT) เป็นเซอร์วิสพื้นฐานของ OGSA ที่ใช้สำหรับควบคุมดูแลการถ่ายโอนข้อมูลระหว่างผู้ใช้งานกลุ่มที่ 3 ที่ใช้ GridFTP ซึ่งมีหลักการทำงานดังภาพ



รูปที่ 5-4 แสดงขั้นตอนการทำงานของ RFT

การทำงานเริ่มเมื่อกริดเซอร์วิสคอนเทนเนอร์เริ่มทำงาน โดยบรรจุ RFT factory ไว้

1. RFT Factory นั้นจะเริ่มลงทะเบียนตัวของมันให้เป็นค่ารีจิสตรีที่เป็นที่รู้จักกัน
2. Client ค้นเจอ RFT Factory นั้น โดยการคิวรี่ serviceData ของรีจิสตรีนั้น
3. Client เรียกคำสั่ง createService บนแฟคทอรี่นั้นและส่งไปยังที่ที่ร้องขอ



รูปที่ 5-5 แสดงขั้นตอนการทำงานของ RFT (ต่อ)

4. เซอร์วิสอินสแตนซ์นั้นเริ่มทำงาน
5. RFT factory จะส่งค่า locator กลับไปยัง client
6. Client จะเรียก Start() และส่งค่าไปยังส่วน notifications หรืออื่นๆ

5.2.3.4 XIO

Globus XIO เป็นส่วนที่ใช้สำหรับขยายอินพุท/เอาท์พุท ไลบรารี่สำหรับ Globus Toolkit ที่ใช้จัดเตรียม API พื้นฐาน (เช่น open/close/read/write) เพื่อนำไปใช้ใน IO ตามต้องการ ซึ่ง Globus XIO นั้นมีจุดหมายสำคัญอยู่ 2 ประการ คือ

- เตรียม API สำหรับผู้ใช้แบบเดี่ยวให้กับโปรโตคอลที่เกี่ยวข้องกับ IO ของกริด ส่วนนี้จะใช้การส่งข้อความสำหรับใช้งานในโปรโตคอลที่แตกต่างกัน ซึ่งผู้เขียนโปรแกรมจำเป็นต้องเขียนให้อยู่ในรูปแบบที่กำหนดไว้ใน API นั้นๆ ซึ่งโปรโตคอลที่สร้างให้ Globus XIO นั้นจะเรียกว่า "drivers" ซึ่งเมื่อใช้งานจะเป็นส่วนที่ซ่อนจากผู้พัฒนา โดยจำเป็นต้องเขียน API นี้ในครั้งแรกเมื่อจะใช้แอพพลิเคชั่นนั้นๆในครั้งต่อๆไป จะสามารถทำได้โดยไม่ต้องเปลี่ยนซอร์สโค้ดเลย
- ลดเวลาในการสร้างหรือทดลองโปรโตคอลใหม่ๆ โดย Globus XIO นั้นจะจัดเตรียม "driver" ดังที่กล่าวไปแล้ว ทำให้ไม่จำเป็นต้องยุ่งมากนักในการใช้โปรโตคอลใหม่ๆ และยังเป็นส่วนที่ทำให้เกิดการนำซอร์สโค้ดเก่ามาใช้งานใหม่ด้วย และยังสามารถนำไปรวมกับ driver อื่นๆได้

Globus XIO นั้นมีฟีเจอร์ที่เกี่ยวข้องกับระบบการทำงาน โดยได้เตรียม API ที่ใช้งานได้ให้กับผู้ใช้งาน ทำให้ผู้ใช้งานไม่ต้องเขียนฟังก์ชั่นยากๆด้วยตัวเองอย่างเดียว โดยฟีเจอร์ของ XIO นั้น มีดังนี้

- Simple API ใช้สำหรับงานพื้นฐาน เช่น open, close, read, write
- Extensibility with drivers เนื่องจาก Globus XIO นั้นออกแบบให้ขยายได้ง่าย ทำให้สามารถเพิ่มฟีเจอร์ใหม่ๆ เข้าไปได้
- Effeciency เนื่องจาก Globus XIO ได้ถูกออกแบบให้มีประสิทธิภาพ มีไลบรารี่ที่สามารถใช้งานได้หลากหลาย
- Timeouts ทำให้ผู้ใช้งานไม่ต้องกำหนด timeout เอง เพราะ globus xio จะมีค่า timeout โดยปกติของมันอยู่แล้ว และสามารถปิดตัวเองเมื่อทราบว่าคำสั่งนั้นไม่ต้องทำงานแล้วด้วย
- Data Descriptions มีบัฟเฟอร์ที่ให้ผู้ใช้สามารถเขียนหรืออ่านได้ โดย Globus XIO ได้ยอมให้ผู้ใช้ผ่านเข้าไปจัดการส่วนนี้ได้ แม้ว่า IO นั้นจะทำงานสมบูรณ์ไปแล้ว

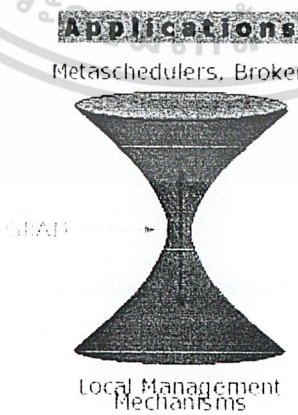
5.2.4 การจัดการรีซอร์ส (GRAM)

5.2.4.1 GRAM

Globus Toolkit ได้ใช้ GRAM (Grid Resource Allocation and Management) ในการจัดการระบบในระยะไกลโดยเตรียมรูปแบบสำหรับการร้องขอและใช้ระบบจากระยะไกลสำหรับการทำงาน โดยส่วนที่ใช้มากที่สุดก็คือ ใช้สำหรับจ่ายและควบคุมงานจากระยะไกล

GRAM ได้ออกแบบโดยใช้โปรโตคอลพื้นฐานและ API สำหรับการร้องขอและใช้รีซอร์สจากระบบในระยะไกล โดยอาศัยรูปแบบที่อ้างจากตารางงาน (Job Schedule) โดยอาศัย GSI มาช่วยเสริมในด้านความปลอดภัย

GRAM จะใช้หลากหลายวิธีในการควบคุมงาน คือใช้ ตารางงาน, ระบบคิวรี่, ระบบสำรอง และรูปแบบการควบคุม ซึ่งทำให้ผู้พัฒนาสามารถเรียนรู้มันและนำไปช่วยในการพัฒนาระบบได้ ซึ่งถ้าเปรียบแล้ว GRAM จะทำงานเหมือนกับคอของนาฬิกาทรายที่ใช้แอพพลิเคชั่นและเซอร์วิสระดับสูง (เช่น resource broker) มาช่วยในการให้สิทธิการเข้าถึงนั่นเอง



รูปที่ 5-6 ลักษณะการทำงานของ GRAM

5.2.5 ข้อมูลเซอร์วิส (MDS)

5.2.5.1 MDS3

Monitoring and Discovery Services (MDS3) เป็นส่วนที่จัดการเกี่ยวกับข้อมูลเซอร์วิส สำหรับ Globus Toolkit 3 ซึ่งจะใช้จัดการกับพวก serviceData และ Index Service โดยมีวิธีการคิวรี่ผ่าน XML รูปแบบหนึ่งที่เรียกว่า XPath

serviceData ที่ใช้ใน GT3 ก็คือ ข้อมูล XML ที่ถูกสร้างโดยทุกๆกริดเซอร์วิสที่แทนในสเตทภายใน ซึ่งแต่ละส่วนของ serviceData นั้นจะถูกเรียกว่า serviceData Element (SDE) ดังที่กล่าวไปแล้วในบท OGSI ซึ่งสเตทของโฮสท์นั้นๆจะถูกแสดงโดย SDE แบบเดี่ยว (ด้วย GRAM) หรือจากสถานะของงาน

วิธีการคิวรี่แบบ XPath ที่ใช้ใน MDS นั้น สร้างขึ้นมาจาก XATAN XPath library ซึ่งเป็นส่วนหนึ่งของภาษาคิวรี่แบบ W3C XML โดยที่เซอร์วิสใดๆที่ถูกสร้างขึ้นจาก GT3 OGSA แล้วจะต้องรองรับการทำงานของฟังก์ชั่นนี้ โดย Globus ได้กำหนดรูปแบบของการคิวรี่อินพุท โดยใช้ค่า SDE และ XPath โดยที่เอาท์พุทที่ได้เป็นผลลัพธ์จากการคำนวณของ Xpath คิวรี่นั้นกับกลุ่มของ SDE ที่พิจารณา

บทที่ 6
การเขียนกริดเซอร์วิส

ในส่วนนี้จะแสดงขั้นตอนการเขียนกริดเซอร์วิสซึ่งได้แบ่งเป็น 5 ขั้นตอน โดยในตัวอย่างนี้จะเป็นกริดเซอร์วิสทางคณิตศาสตร์อย่างง่าย คือ จะมีคำสั่งบวกและลบ

1. กำหนดรูปแบบของเซอร์วิส ส่วนนี้จะทำโดย GWSDL
2. สร้างเซอร์วิส ส่วนนี้ทำโดยใช้ภาษา Java
3. กำหนดค่าส่วนที่จะนำมาใช้งาน ส่วนนี้จะทำโดย WSDD
4. สร้างไฟล์ชนิด GAR จากการคอมไพล์ทุกส่วนแล้วนำมารวมกัน ส่วนนี้จะใช้ Ant ช่วย
5. นำเซอร์วิสไปไว้ในกริดเซอร์วิสคอนเทนเนอร์ ส่วนนี้จะใช้ Ant ช่วย

6.1 ขั้นตอนที่ 1: กำหนดรูปแบบของเซอร์วิส

ขั้นตอนแรกจะเป็นการเขียนกริดเซอร์วิส หรือ เว็บเซอร์วิส ที่ใช้กำหนดรูปแบบของเซอร์วิสนั้น โดยจะเป็นการกำหนดว่าจะให้เซอร์วิสนี้มีความสามารถอะไรบ้างที่จะให้ผู้อื่นใช้งาน โดยที่ส่วนนี้จะไม่สนใจว่าจะใช้อัลกอริธึมยังไงในการเขียน จะต้องใช้งานไฟล์ไหนบ้าง แต่จะสนใจเพียงว่ามีคำสั่งอะไรบ้างให้ใช้งาน โดยรูปแบบการใช้งานในกริดเซอร์วิสหรือเว็บเซอร์วิสจะใช้งานผ่านอินเตอร์เฟสที่เรียกว่า portType ซึ่งส่วนนี้จะกำหนดโดย Web Service Description Language (WSDL)

วิธีการที่จะกำหนด WSDL ที่จะให้ใช้งานในส่วนนี้มีวิธีทำได้อยู่ 2 วิธี ดังต่อไปนี้

- เขียน WSDL ด้วยตนเอง จะสามารถกำหนดทุกอย่างได้ตามที่ตนเองต้องการ แต่ผู้ใช้มักจะไม่คุ้นเคยกับส่วนนี้ เนื่องจาก WSDL เป็นภาษาที่เขียนหลายส่วนมากเกินความจำเป็น
- สร้าง WSDL ขึ้นมาตามรูปแบบของ Java ซึ่งจะสร้างออกมาได้สะดวก แต่อาจไม่ครอบคลุมทุกอย่าง มักมีปัญหาในการทำงานบางอย่าง

โดยตัวอย่างการกำหนดส่วนนี้ทำได้ดังโค้ดตัวอย่างด้านล่างนี้

```
public interface Math
{
 public void add(int a);
 public void subtract(int a);
 public int getValue();
}
```

คราวนี้จะทำให้อยู่ในรูปแบบที่อธิบายด้วย WSDL ที่แม้ว่าจะเข้าใจยากกว่ารูปแบบของ Java แต่การทำงานของมันจะเหมาะสมกับกริดมากกว่า ซึ่งกริดนั้นจะเปลี่ยนแปลงบางส่วนของ WSDL ให้เป็น GWSDL ซึ่งกำหนดรูปแบบมาจาก OGSI และจาก Globus Toolkit โดย GWSDL นั้นเป็นการขยายส่วนของ WSDL ให้

รองรับการอธิบายเกี่ยวกับกริดเซอร์วิส ที่ไม่สามารถอธิบายได้ด้วย WSDL โดยรูปแบบของ GSWDL ที่เหมือนกับที่ได้กำหนดในรูปแบบของ Java ข้างต้นนั้น แสดงได้ดังโค้ดด้านล่างนี้

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
name="MathService"
 targetNamespace="http://www.globus.org/namespaces/2004/02/progtutorial/MathService"
 xmlns:tns="http://www.globus.org/namespaces/2004/02/progtutorial/MathService"
 xmlns:ogsi="http://www.gridforum.org/namespaces/2003/03/OGSI"
 xmlns:gwsdl="http://www.gridforum.org/namespaces/2003/03/gridWSDLExtensions"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns="http://schemas.xmlsoap.org/wsdl/">

<import location="../../ogsi/ogsi.gwsdl"
 namespace="http://www.gridforum.org/namespaces/2003/03/OGSI"/>

<types>
<xsd:schema
targetNamespace="http://www.globus.org/namespaces/2004/02/progtutorial/MathService"
 attributeFormDefault="qualified"
 elementFormDefault="qualified"
 xmlns="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="add">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="value" type="xsd:int"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
 <xsd:element name="addResponse">
  <xsd:complexType/>
 </xsd:element>
 <xsd:element name="subtract">
  <xsd:complexType>
```

```
    <xsd:sequence>
     <xsd:element name="value" type="xsd:int"/>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
  <xsd:element name="subtractResponse">
   <xsd:complexType/>
  </xsd:element>
  <xsd:element name="getValue">
   <xsd:complexType/>
  </xsd:element>
  <xsd:element name="getValueResponse">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:element name="value" type="xsd:int"/>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
 </xsd:schema>
</types>

<message name="AddInputMessage">
 <part name="parameters" element="tns:add"/>
</message>
<message name="AddOutputMessage">
 <part name="parameters" element="tns:addResponse"/>
</message>
<message name="SubtractInputMessage">
 <part name="parameters" element="tns:subtract"/>
</message>
<message name="SubtractOutputMessage">
 <part name="parameters" element="tns:subtractResponse"/>
</message>
<message name="GetValueInputMessage">
```

```
  <part name="parameters" element="tns:getValue"/>
</message>
<message name="GetValueOutputMessage">
  <part name="parameters" element="tns:getValueResponse"/>
</message>

<gwsdl:portType name="MathPortType"
extends="ogsi:GridService">
  <operation name="add">
    <input message="tns:AddInputMessage"/>
    <output message="tns:AddOutputMessage"/>
    <fault name="Fault" message="ogsi:FaultMessage"/>
  </operation>
  <operation name="subtract">
    <input message="tns:SubtractInputMessage"/>
    <output message="tns:SubtractOutputMessage"/>
    <fault name="Fault" message="ogsi:FaultMessage"/>
  </operation>
  <operation name="getValue">
    <input message="tns:GetValueInputMessage"/>
    <output message="tns:GetValueOutputMessage"/>
    <fault name="Fault" message="ogsi:FaultMessage"/>
  </operation>
</gwsdl:portType>
</definitions>
```

หมายเหตุ: โค้ดส่วนที่ทำเน้นตัวอักษรไว้ เป็นส่วนที่สำคัญที่จะนำมาอธิบายเพิ่มเติม ดังต่อไปนี้

ส่วนแรกเป็นเป้าหมายของเนมสเปซจะเป็นอย่างไร

http://www.globus.org/namespaces/2004/02/progtutorial/MathService

ส่วนที่ 2 เป็นการประกาศเนมสเปซของ OGSI

xmlns:ogsi="http://www.gridforum.org/namespaces/2003/03/OGSI"

xmlns:gwsdl="http://www.gridforum.org/namespaces/2003/03/gridWSDLExtensions"

ส่วนที่ 3 เป็นการอิมพอร์ท GWSDL และกำหนดชนิด ข้อความ และ portType ของ OGSI

```
<import location="../../ogsi/ogsi.gwsdl"
namespace="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
```

6.1.1 ส่วนที่แตกต่างกันระหว่าง GWSDL และ WSDL

เมื่อพิจารณาจากโค้ดนี้จะแสดงให้เห็นว่าส่วนที่ขยายขึ้นมาจาก WSDL นั้น คืออะไรบ้าง ดังนี้

```
<gwsdl:portType name="MathPortType" extends="ogsi:GridService"> </gwsdl:portType>
```

ส่วนแรกที่สำคัญคือ แท็กของ GWSDL ที่แสดงด้วย namespace: <gwsdl:portType> ส่วนนี้จะทำให้สามารถกำหนด portType ให้ขยายได้จาก 1 portType ที่ขยายมาจากหลายๆ portType

ส่วนที่สองที่สำคัญคือ ความเกี่ยวข้องกับ ข้อมูลเซอร์วิส ซึ่งจะแสดงให้เห็นในเรื่องต่อๆไป

6.2 ขั้นตอนที่ 2: สร้างเซอร์วิส

จากขั้นตอนแรกที่กล่าวไปเป็นการบอกว่า เซอร์วิสจะทำอะไร ส่วนต่อไปนี้จะเป็นการแสดงว่า จะทำอย่างที่กำหนดไว้ในขั้นตอนแรกอย่างไร กล่าวคือ เขียนโปรแกรมให้ทำตามที่กำหนดไว้นั่นเอง ซึ่งใช้ Java ในการเขียนโค้ดในส่วนนี้ โดยจะอธิบายแยกเป็นส่วนๆดังต่อไปนี้

ส่วนแรก เป็นการอิมพอร์ทแพ็คเกจที่ต้องการใช้งาน

```
package org.globus.progtutorial.services.core.first.impl;

import org.globus.ogsa.impl.ogsi.GridServiceImpl;
import org.globus.progtutorial.stubs.MathService.MathPortType;
import java.rmi.RemoteException;
```

ส่วนนี้จะเป็นการกำหนดคลาสที่ชื่อว่า MathImpl ที่จะใช้สร้างเซอร์วิสที่กำหนดไว้

```
public class MathImpl extends GridServiceImpl implements MathPortType
```

ส่วนนี้จะเป็นการสร้าง คอนสตัคเตอร์ ของกริดเซอร์วิส

```
public MathImpl()
{
    super("Simple Math Service");
}
```

ส่วนนี้จะเป็นการกำหนดวิธีการที่จะให้ได้ผลลัพธ์ดังที่กำหนดไว้จากขั้นตอนแรก

```
public void add(int a) throws RemoteException
{
 value = value + a;
```

```
}

public void subtract(int a) throws RemoteException
{
 value = value - a;
}

public int getValue() throws RemoteException
{
 return value;
}
```

เมื่อนำโค้ดแต่ละส่วนมารวมกันแล้ว จะได้โค้ดดังต่อไปนี้

```
package org.globus.progtutorial.services.core.first.impl;

import org.globus.ogsa.impl.ogsi.GridServiceImpl;
import org.globus.progtutorial.stubs.MathService.MathPortType;
import java.rmi.RemoteException;

public class MathImpl extends GridServiceImpl implements MathPortType
{
 private int value = 0;

 public MathImpl()
 {
  super("Simple MathService");
 }

 public void add(int a) throws RemoteException
 {
  value = value + a;
 }
```

```
public void subtract(int a) throws RemoteException
{
 value = value - a;
}

public int getValue() throws RemoteException
{
 return value;
}
}
```

6.3 ขั้นตอนที่ 3: กำหนดค่าส่วนที่จะนำมาใช้งาน

เมื่อมาถึงขั้นตอนนี้ เราได้ทำส่วนที่สำคัญก็คือ กำหนดรูปแบบของเซอร์วิส และ สร้างเซอร์วิส มาแล้ว ในขั้นตอนนี้จะนำ 2 ส่วนแรกมารวมเข้าด้วยกัน และทำให้มันเป็นกริดเซอร์วิส ที่จะใช้งานบน Server 1 ได้ ในขั้นนี้มักจะเรียกว่าการ ดีพลอย กริดเซอร์วิส (deployment grid service)

ส่วนที่สำคัญในขั้นตอนนี้คือการกำหนดรายละเอียดการดีพลอยที่เรียกว่า deployment descriptor ซึ่งเป็นไฟล์ที่จะบอก server ว่าจะสร้างกริดเซอร์วิสขึ้นมาอย่างไร (เช่น บอกว่า GSH ของเราจะเป็นอะไร) โดยค่ารายละเอียดการดีพลอยนั้น จะเขียนในรูปแบบของ WSDD (Web Service Deployment Descriptor) ซึ่งมีตัวอย่างโค้ดให้ดู ดังนี้

```
<?xml version="1.0"?>
<deployment name="defaultServerConfig" xmlns="http://xml.apache.org/axis/wsdd/"
 xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

 <service name="progtutorial/core/first/MathService" provider="Handler" style="wrapped">
   <parameter name="name" value="MathService"/>
   <parameter name="className"
value="org.globus.progtutorial.stubs.MathService.MathPortType"/>

   <parameter name="baseClassName"
value="org.globus.progtutorial.services.core.first.impl.MathImpl"/>
   <parameter name="schemaPath"
value="schema/progtutorial/MathService/Math_service.wsdl"/>

   <!-- Start common parameters -->
```

```
    <parameter name="allowedMethods" value="*"/>
    <parameter name="persistent" value="true"/>
    <parameter name="handlerClass" value="org.globus.ogsa.handlers.RPCURIProvider"/>
   </service>
  </deployment>
```

จะอธิบายส่วนสำคัญแต่ละส่วน ต่อไปนี้

6.3.1 ชื่อเซอร์วิส

คือโค้ดส่วนนี้

<service name="progtutorial/core/first/MathService" provider="Handler"

style="wrapped">

เป็นส่วนที่กำหนดว่า กริดเซอร์วิสของเราจะเจอที่ตำแหน่งไหน เราอาจจะนำส่วนนี้มารวมกับที่อยู่ของกริดเซอร์วิสคอนเทนเนอร์ก็ได้ โดยเราจะได้ค่า GSH แบบเต็ม จากกริดเซอร์วิส เช่น ถ้าเราใช้ GT3 ที่มีคอนเทนเนอร์เป็นแบบสแตนด์อโลน จะทำให้มีที่อยู่ที่เป็นฐานคือ http://localhost:8080/ogsa/services เมื่อรวมกับใน GSH ของเซอร์วิสที่เรากำหนดไว้แล้ว จะเป็นดังนี้

http://localhost:8080/ogsa/services/progtutorial/core/first/MathService

6.3.2 ชื่อเซอร์วิสอีกส่วน

คือโค้ดส่วนนี้

<parameter name="name" value="MathService"/>

ส่วนนี้จะเป็นส่วนที่บ่งบอกว่าเซอร์วิสนั้นจะมีค่าทั้งเป็น ชื่อแอททิบิวท์ อย่างที่แสดงไว้ส่วนแรก และส่วนนี้จะบ่งบอกถึงพารามิเตอร์ของเซอร์วิส ในโค้ดจะใช้เป็น MathService

6.3.3 className และ baseClassName

คือโค้ดส่วนนี้

<parameter name="className"

value="org.globus.progtutorial.stubs.MathService.MathPortType"/>

<parameter name="baseClassName"

value="org.globus.progtutorial.services.core.first.impl.MathImpl"/>

ใน WSDD นั้น ค่านี้จะบอกว่าคลาสใดที่ใช้สร้างเซอร์วิส (ในตัวอย่างนี้จะใช้ MathImpl จากส่วนก่อนๆ) อย่างไรก็ตามควรมีการแจ้งค่านี้ว่าเป็น Math-PortType stub ที่กำหนดต่อเนื่องมาจากหน้าก่อนๆ

เมื่อกริดเซอร์วิสได้กำหนดให้ยืดหยุ่นมากกว่าเว็บเซอร์วิส จึงจำเป็นต้องกำหนดความแตกต่างกันของ className และ baseClassName โดย className จะอ้างรูปแบบที่แสดงฟังก์ชั่นของกริดเซอร์วิส (MathPortType) และ baseClassName จะเป็นคลาสที่จัดเตรียมการสร้างกริดเซอร์วิส ในกรณีนี้ baseClassName จะเป็น MathImpl ที่สร้างมาจากส่วนก่อนๆ ในขั้นตอนต่อไป จะแสดงให้เห็นว่า baseClassName ไม่ได้ต้องการรวมส่วนที่สร้างขึ้นมาทั้งหมดของสิ่งที่แสดงใน classNameก

6.3.4 ไฟล์ WSDL

คือโค้ดส่วนนี้

```
<parameter name="schemaPath"
value="schema/progtutorial/MathService/Math_service.wsdl"/>
```

เป็น schemaPath ที่บอกกริดเซอร์วิสคอนเทนเนอร์ว่า WSDL ไฟล์นั้นจะสามารถเจอได้ที่ไหน ที่ไม่ได้บอกถึง GWSDL เนื่องจาก GWSDL ไม่ใช่มาตรฐาน แต่เป็นส่วนที่ขยายมาจาก WSDL โดยขั้นแรกจะต้องแปลงให้เป็น WSDL เพื่อใช้บอกว่ามีการใช้งานกับเทคโนโลยีเว็บเซอร์วิสที่มีอยู่จริง ซึ่งไฟล์ WSDL นั้นจะสร้างโดยอัตโนมัติด้วย GT3 เมื่อเราคอมไพล์เซอร์วิส

6.3.5 พารามิเตอร์ทั่วไป

คือโค้ดส่วนนี้ ซึ่งเป็นค่าพารามิเตอร์ที่ใช้ในโปรแกรมตัวอย่าง

```
<!-- Start common parameters -->
<parameter name="allowedMethods" value="*"/>
<parameter name="persistent" value="true"/>
<parameter name="handlerClass" value="org.globus.ogsa.handlers.RPCURIProvider"/>
```

6.4 ขั้นตอนที่ 4: สร้างไฟล์ชนิด GAR จากการคอมไพล์ทุกส่วนแล้วนำมารวมกัน

เมื่อมาถึงขั้นตอนนี้เราจะได้ รูปแบบของเซอร์วิสจากขั้นที่ 1 ได้เซอร์วิสที่สร้างขึ้นจากขั้นที่ 2 และได้ส่วนที่บอกกริดเซอร์วิสคอนเทนเนอร์ว่าจะแสดงส่วนที่ได้จากขั้นที่ 1 และ 2 ไปให้ภายนอกเห็นอย่างไรบ้าง แต่ส่วนที่ได้มานั้นยังไม่สามารถใช้งานได้จริงในคอนเทนเนอร์ เนื่องจากไฟล์ Java ก็ยังไม่ได้คอมไพล์และไฟล์อื่นๆก็ยังไม่รู้ว่าต้องนำไปวางที่ส่วนไหน

ดังนั้นในขั้นนี้เราจะรวมทุกส่วนที่กล่าวมาเข้าด้วยกัน เพื่อให้นำไปใช้งานจริงได้ โดยเราจะสร้างไฟล์ Grid Archive หรือไฟล์ประเภท GAR ขึ้นมา โดยที่ไฟล์ Gar นั้นจะเป็นไฟล์เดี่ยวที่บรรจุไฟล์ทั้งหมดและข้อมูลที่กริดเซอร์วิสคอนเทนเนอร์ต้องการทั้งหมดไว้เพื่อที่จะดีพลอยไปให้แก่เซอร์วิสที่เราสร้างขึ้นให้ข้างนอกมาใช้งาน ต่อมาเราจะแสดงวิธีที่จะทำไฟล์ GAR และดีพลอยมัน โดยขั้นตอนนั้นจะแบ่งออกได้ดังนี้

- แปลงจาก GWSDL ให้เข้าสู่รูปแบบมาตรฐานคือ WSDL

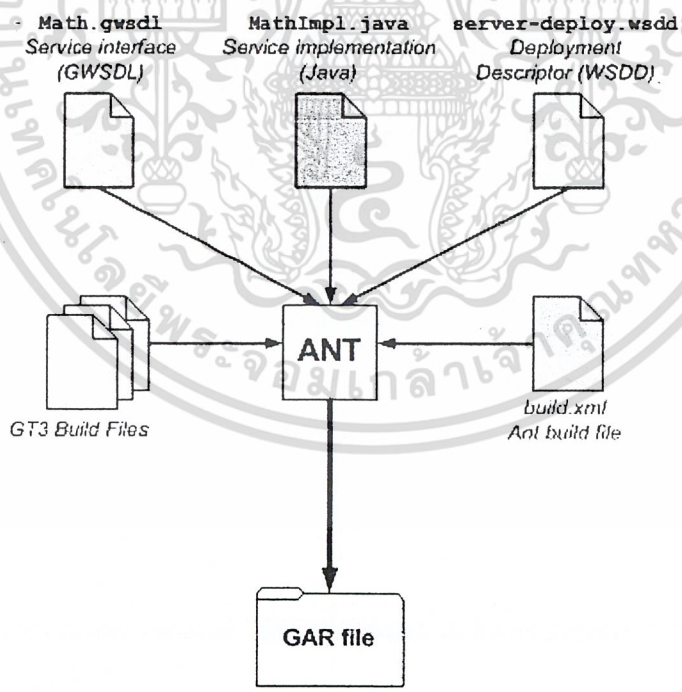
- สร้างสตับคลาสจาก WSDL
- คอมไพล์สตับคลาสนั้นๆ
- คอมไพล์เซอร์วิสที่เราสร้างขึ้น
- รวบรวมไฟล์ทั้งหมดเข้าสู่โครงสร้างที่จะใช้โดยเฉพาะ

ขั้นตอนเหล่านี้ เมื่อกล่าวขึ้นมาอาจจะฟังดูยาก แต่ Globus ก็ได้มีเครื่องมือที่ช่วยในขั้นตอนนี้ นั่นก็คือ Ant ที่สามารถทำขั้นตอนที่กล่าวมาได้ในขั้นตอนเดียว

6.4.1 Ant

Ant เป็น ซอฟท์แวร์ของ Apache ที่เป็นเครื่องมือสร้าง Java โดยอาศัยหลักการคล้ายคำสั่ง make ของ UNIX ซึ่งทำให้คนเขียนโปรแกรมสามารถข้ามขั้นตอนที่ยุ่งยากเกี่ยวกับการนำไฟล์เอ็กคิวท์จากไฟล์ต้นทาง ซึ่งส่วนนี้จะทำโดย Ant ที่จะช่วยลดขั้นตอนในส่วนนี้ให้เหลือขั้นเดียว ซึ่งด้วย Ant นี้จะทำให้ปัญหาของการทำกริดเซอร์วิสนั้นมาอยู่ที่การกำหนดรูปแบบและการสร้างมันขึ้นมา

ซึ่ง Ant ต้องการไฟล์ที่ใช้สร้างอยู่ 2 ไฟล์ คือ ไฟล์ที่ใช้สร้างที่เป็นส่วนหนึ่งของ GT3 และอีกส่วนคือจะเป็นส่วนที่สำคัญที่เราจะต้องเขียนขึ้นมาเอง (ตามขั้นตอนที่สร้างโค้ด WSDL และสร้างสตับ) เมื่อสร้างไฟล์นี้ขึ้นมาแล้วอาจจะใช้ได้กับหลายๆกริดเซอร์วิสโดยไม่ต้องสร้างขึ้นมาใหม่ ซึ่งส่วนนี้ Ant ก็ได้ทำไว้ให้แล้ว แต่ถ้าต้องการงานที่ยืดหยุ่นมากขึ้นก็คงต้องสร้างไฟล์นี้ขึ้นมาใหม่ โดยเราจะแสดงองค์ประกอบของส่วนนี้ให้เห็นดังรูปต่อไปนี้



รูป 6-1: แสดงองค์ประกอบในการใช้งาน Ant

สิ่งที่จำเป็นต้องทำในขั้นตอนนี้

จำเป็นต้องใส่ไดเรคทอรี่ที่เราลง GT3 ไว้ ดังในตัวอย่างข้างล่างนี้

ogsa.root=*/usr/local/gt3*

สร้างเซอร์วิสที่เป็น GAR (ตัวอย่างนี้จะสร้าง MathService) โดยใส่ สคริปท์ ตามนี้

./tutorial_build.sh *<service base directory> <service's GWSDL file>*

โดยที่ <service base directory> นั้นเป็นไดเรคทอรี่ที่เราวาง server-deploy.wsdd ไว้และกำหนดที่ที่จะเจอ MathImpl.java ดังตัวอย่างข้างล่างนี้

**./tutorial_build.sh \ org/globus/progtutorial/services/core/first **

schema/progtutorial/MathService/Math.gwsdl

ถ้าทุกขั้นตอนนั้นถูกต้องไฟล์ GAR นั้นจะอยู่ในตำแหน่งของ $TUTORIAL_DIR/build/lib. ดังในตัวอย่างนี้

$TUTORIAL_DIR/build/lib/org_globus_progtutorial_services_core_first.gar

6.5 ขั้นตอนที่ 5: นำเซอร์วิสไปไว้ในกริดเซอร์วิสคอนเทนเนอร์

ไฟล์ GAR จากขั้นตอนที่แล้วนั้นจะบรรจุไฟล์ที่จำเป็นที่จะใช้ในกริดเซอร์วิสแล้ว ขั้นนี้เราดีพลอยมันโดย Ant ซึ่งจะนำไฟล์เหล่านี้ คือ WSDL, สคับที่คอมไพล์แล้ว, โค้ดที่สร้างขึ้นคอมไพล์แล้ว และก็ WSDD ไปไว้ที่ตำแหน่งที่ให้ใช้งานที่กำหนดไว้ใน GT3 ซึ่งคำสั่งเกี่ยวกับการดีพลอยนั้นเราจะต้องทำที่ root ของส่วนที่เราลง GT3 โดยใช้คำสั่งตามรูปแบบนี้

ant deploy -Dgar.name=*<full path of GAR file>*

สำหรับในตัวอย่างนี้จะได้โค้ดดังนี้

ant deploy \ -Dgar.name=$TUTORIAL_DIR/build/lib/

org_globus_progtutorial_services_core_first.gar

การดีพลอยในขั้นตอนสุดท้ายนี้ก็ทำแต่เพียงเท่านี้ ซึ่งเมื่อมาถึงขั้นนี้เราจะได้ 5 ขั้นตอนที่จำเป็นของการทำกริดเซอร์วิสแล้ว ต่อมาเราจะมาแสดงขั้นตอนในการเขียนแอพพลิเคชั่นในฝั่ง client

6.6 แอพพลิเคชั่นฝั่ง client

เมื่อเราต้องการทดสอบกริดเซอร์วิสที่เขียนขึ้นด้วยคอมมานด์-ไลน์ ซึ่งจะใช้วิธี getValue ในการทดสอบ MathService นี้ สิ่งที่ client จะต้องได้รับจากคอมมานด์-ไลน์มี 2 ค่าคือ

1. GSH

2. ค่าที่เพิ่มขึ้นตามการทำงานที่กำหนดไว้ในเซอร์วิส

เราใช้โค้ดนี้ในการทดสอบการทำงานของ MathService

```
package org.globus.progtutorial.clients.MathService;

import org.globus.progtutorial.stubs.MathService.service.MathServiceGridLocator;
import org.globus.progtutorial.stubs.MathService.MathPortType;

import java.net.URL;

public class Client
{
 public static void main(String[] args)
 {
  try
  {
   // Get command-line arguments
   URL GSH = new java.net.URL(args[0]);
   int a = Integer.parseInt(args[1]);


   // Get a reference to the MathService instance
   MathServiceGridLocator mathServiceLocator = new MathServiceGridLocator();
   MathPortType math = mathServiceLocator.getMathServicePort(GSH);

   // Call remote method 'add'
   math.add(a);
   System.out.println("Added " + a);

   // Get current value through remote method 'getValue'
   int value = math.getValue();
   System.out.println("Current value: " + value);
  }catch(Exception e)
  {
```

```
            System.out.println("ERROR!");
            e.printStackTrace();
        }
    }
}
```

แต่ก่อนที่เราจะรันคอมไพเลอร์นี้ สิ่งที่ต้องรันก่อน มีดังต่อไปนี้

source $GLOBUS_LOCATION/etc/globus-devel-env.sh

เพื่อที่จะคอมไพล์ client จะต้องทำตามนี้

**javac \ -classpath ./build/classes/:$CLASSPATH **

org/globus/progtutorial/clients/MathService/Client.java

โดยก่อนที่จะให้มันทำงานเราต้องเริ่มการทำงานของคอนเทนเนอร์ก่อน โดยรันคำสั่งนี้ที่ root

globus-start-container

เมื่อคอนเทนเนอร์ทำงานแล้ว เราจะเห็นรายการของ GSH ที่มีขึ้นมา ซึ่งเราต้องเลือก MathService ตามนี้

http://127.0.0.1:8080/ogsa/services/progtutorial/core/first/MathService

คราวนี้จะเป็นการสรุปคำสั่งเต็มๆที่จะให้ client ทำงาน

**java **

**-classpath ./build/classes/:$CLASSPATH **

**org.globus.progtutorial.clients.MathService.Client **

**http://127.0.0.1:8080/ogsa/services/progtutorial/core/first/MathService **

5

ถ้าคำสั่งนี้ผ่านเราจะเห็นค่าต่อไปนี้

Added 5

Current value: 5

ซึ่งแสดงว่าคำสั่งบวกของเราใช้งานได้แล้ว โดยจะเริ่มบวกค่าจากค่าเริ่มต้นที่เป็น 0 ทำให้ได้ค่า 5 ขึ้นมา เมื่อเรารันคำสั่งนั้นอีกครั้ง เราจะได้ค่าดังต่อไปนี้

Added 5

Current value: 10

ซึ่งแสดงว่าสเตทภายในของมันนั้นยังเก็บค่า 5 ซึ่งเป็นค่าเดิมไว้อยู่ ทำให้จะได้ค่าใหม่ออกมาเป็น 10 ซึ่งเพิ่มจากค่าเดิมนั่นเอง

บทที่ 7
การออกแบบ

การออกแบบจะแบ่งออกเป็น 2 ส่วนหลักๆคือ การออกแบบโครงสร้างของระบบกริด และการออกแบบโปรแกรมที่ใช้ในการคำนวณเมตริก

7.1. โครงสร้างและการทำงานของระบบ

การทดลองนี้ จะเป็นการสร้างระบบกริดโดยใช้เครื่องคอมพิวเตอร์ 3 เครื่อง อยู่ในวงแลนเดียวกัน โดยกำหนดรายละเอียดของแต่ละเครื่องดังนี้

- x1 (x1.kmitl.com), IP = 161.246.6.102 , CPU = Pentiun4 2.4 GHz , Ram = 512 Mb
- x2 (x2.kmitl.com), IP = 161.246.6.124, CPU = AMD Athlon 1.8GHz , Ram = 512 Mb
- x3 (x3.kmitl.com), IP = 161.246.6.93 , CPU = Pentiun4 1.6 GHz , Ram = 512 Mb

โดยกำหนดให้เครื่อง x1 เป็น Authentication Server มีหน้าที่การทำ Certificate Authentication ให้กับเครื่องอื่นๆที่ต้องการเข้ามาใช้งานระบบ

เครื่อง x2, x3, เป็น Grid Client ที่สามารถใช้งานระบบกริดได้โดยที่ทุกเครื่อง (รวมทั้ง x1) จะมีเซอร์วิสของการคูณเมตริกอยู่ สามารถเรียกใช้งานผ่านเครือข่ายได้

ซอฟท์แวร์ที่ใช้ในการเซ็ตอัพระบบมีดังนี้

- Linux Redhat 9
- Globus Toolkit (gt3.2.1-all-linux-rh9-installer.tar.gz)
- Java J2SE (j2sdk-1_4_2_04-linux-i586-rpm.bin)
- Apache Ant (apache-ant-1.6.1-bin.tar.gz)
- Junit (junit3.8.1.zip)

7.2. การออกแบบโปรแกรม

จุดมุ่งหมายของปริญญานิพนธ์ฉบับนี้ เป็นการเขียนโปรแกรมการคำนวณเมตริกให้สามารถใช้งานในระบบกริด ซึ่งจากทฤษฎีที่กล่าวมาข้างต้น การใช้งานโปรแกรมภายในระบบกริดของ gt3.2.1 จะเป็นการใช้งานในรูปแบบของกริดเซอร์วิส การออกแบบจึงใช้รูปแบบการเขียนของกริดเซอร์วิสของการคูณเมตริก ซึ่งแบ่งออกเป็น 2 ส่วนคือ ส่วนของ Matrix Service และส่วนของ Matrix Client

- Matrix Service เป็น เซอร์วิสที่ใช้ในการคูณเมตริก โดยรับตัวแปรเป็น array of integer ผ่านทาง Matrix portType ซึ่งเป็น GWSDL file โดยรับข้อมูลมา 2 ชุด ชุดแรกเป็นตัวตั้ง และอีกชุดเป็นตัวคูณ ผลลัพธ์จะส่งกลับเป็น array of integer เช่นกัน
- Matrix Client เป็นโปรแกรมที่ใช้ในการจัดการข้อมูลที่จะใช้ในการคูณเมตริก โดยจะรับข้อมูลเข้ามาเป็นไฟล์ของเมตริก 2 ไฟล์(text file) และทำการแปลงไฟล์ให้ออกมาอยู่ในรูปของ array of

integer เพื่อใช้ในการคำนวณ จากนั้นจะทำการค้นหาเครื่องที่มีเซอร์วิสของการคูณเมตริก (Matrix Service) โดยผ่านทาง Index Service แบ่งงานและกระจายงานไปยังเครื่องดังกล่าวให้แยกกันคำนวณ รวบรวมผลลัพธ์ แล้วส่งค่ากลับไปยัง user โดยการเขียนไฟล์ผลลัพธ์ ไปยัง directory ที่กำหนด

7.2.1 การออกแบบ Matrix Service

Matrix Service จะอยู่ในรูปแบบของกริดเซอร์วิส มีส่วนประกอบดังนี้

- Matrix portType เป็น GWSDL file (Matrix.gwsdl) ใช้ในการกำหนดรูปแบบของการรับส่งข้อมูลของ Matrix Service ประกอบด้วยของมูลที่ต้องส่ง 2 ชุด ชุดแรก เป็นข้อมูลที่จะนำไปคำนวณ มีข้อมูล 3 อย่างคือ อะเรย์ตัวตั้ง อะเรย์ตัวคูณ และขนาดของอะเรย์ ชุดที่ 2 เป็นส่วนของการรีเทิร์นค่า
- Java file ที่ระบุวิธีการคำนวณ (Matrix.java) ประกอบด้วยฟังก์ชั่นของการคูณเมตริก และฟังก์ชั่นของการรีเทิร์นค่า
- Sever-deploy.wsdd ใช้เป็นไฟล์ในการกำหนดวิธีการของการทำ deployment

การคอมไพล์และการทำ deployment ใช้ apache ant

7.2.2 การออกแบบ Matrix Client

เป็น Java file ใช้เป็นส่วนที่ติดต่อระหว่าง user กับเซอร์วิส ซึ่งจะประกอบด้วยส่วนประกอบหลักๆ ดังนี้

- ส่วนของการรับไฟล์ รับมาเป็น text file แล้วแปลงให้อยู่ในรูปของอาเรย์ 2 มิติ
- ส่วนของการเรียกใช้ Matrix Service โดยเรียกใช้ผ่าน service instance
- ส่วนของการติดต่อกับ Matrix Service เพื่อส่งค่าและรับผลลัพธ์กลับมา
- ส่วนของการรวบรวมผลลัพธ์ที่ได้จากการกระจายงาน แล้วแสดงผลลัพธ์กลับไปยัง user โดยการเขียนไฟล์ (text file)

บทที่ 8

การทดลอง

การทดลองจะเป็นการทดสอบการใช้งานได้ของระบบกริดที่สร้างขึ้นและทดสอบการใช้งานได้ของโปรแกรมที่เขียนขึ้นมาโดยจะเน้นในเรื่องของประสิทธิภาพการเรียกใช้งานเซอร์วิสในรูปแบบต่างๆ เช่นการทดลองรันบนเครื่องเดียว หรือหลายๆเครื่อง แล้วสังเกตว่าผลที่ได้มีความแตกต่างกันอย่างไร และทดลองเรียกใช้เซอร์วิสจากเครื่องที่มีความเร็วต่างกันว่าให้ผลเป็นอย่างไร

8.1 อุปกรณ์ที่ใช้

ในการทดลอง จะใช้ คอมพิวเตอร์ 3 เครื่องในการสร้างระบบโดยมีสเปคที่ต่างกันดังนี้

ชื่อเครื่อง	IP	CPU	Ram
x1 (x1.kmitl.com)	161.246.6.102	Pentium 4 2.4 GHz	512
x2 (x2.kmitl.com)	161.246.6.124	Amd Samptron 2500+	512
x3 (x3.kmitl.com)	161.246.6.93	Pentium 4 1.6 GHz	512

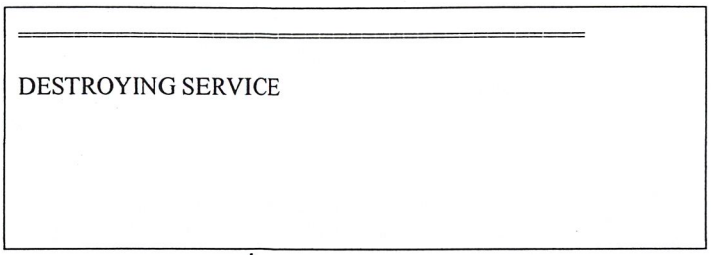
ตารางที่ 8-1 อุปกรณ์ที่ใช้ในการทดลอง

8.2 ทดสอบการใช้กริดโดยการเรียกใช้งาน test service ซึ่งมีอยู่แล้วใน gt3.2.1 โดยให้ทุกเครื่องลองเรียกใช้เซอร์วิสของกันและกัน

เช่น ให้ x1 ลองเรียกใช้เซอร์วิสของ x2

```
$ managed-job-globusrun –factory \
http://x2:8080/ogsa/services/base/gram/MasterForkManagedJobFactoryService -file \
$GLOBUS_LOCATION/schema/base/gram/example/test.xml
```

```
========== Status Notification ==========
Job Status: StageIn
==========================================
========== Status Notification ==========
Job Status: active
==========================================
========== Status Notification ==========
Job Status: done
```



รูปที่ 8-1 ผลลัพธ์จากการรัน test service

ปรากฏว่าทุกเครื่องสามารถรันแล้วได้ผลดังรูป สามารถสรุปได้ว่าทุกเครื่องสามารถใช้งานระบบกริดที่สร้างขึ้นได้

8.3 ทดสอบการใช้งานของเซอร์วิสที่เขียนขึ้นในรูปแบบต่างๆ

8.2.1 ทดสอบการใช้งาน โดยใช้ x2 กระจายงาน

1. ทดสอบการรันบนเครื่องเดียว โดยให้รันบน x2

	เวลา (วินาที)
เมตริก 10*10	26
	24
	22
	23
	23
ค่าเฉลี่ย	**23.6**
เมตริก 25*25	187
	179
	188
	168
	182
ค่าเฉลี่ย	**180.8**

ตารางที่ 8-2 ผลการรันบน x2

2. ทดสอบการรันบน 2 เครื่อง โดยให้รันบน x1 และ x2 ใช้ x2 เป็นตัวกระจายงาน

	เวลา (วินาที)
เมตริก 10*10	19
	17
	17
	19
	17
ค่าเฉลี่ย	**17.8**
เมตริก 25*25	110
	124
	122
	113
	119
ค่าเฉลี่ย	**117.6**

ตารางที่ 8-3 ผลการรันบน x1 และ x2 โดยให้ x2 เป็นตัวกระจายงาน

3. ทดสอบการรันบน 3 เครื่อง โดยรันบน x1, x2 และ x3 ใช้ x2 เป็นตัวกระจายงาน

	เวลา (วินาที)
เมตริก 10*10	18
	17
	16
	15
	16
ค่าเฉลี่ย	**16.4**
เมตริก 25*25	78
	81
	70
	63
	72
ค่าเฉลี่ย	**72.8**

ตารางที่ 8-4 ผลการรันบน x1 ,x2 และ x3 โดยให้ x2 เป็นตัวกระจายงาน

8.2.2 ทดสอบการใช้งาน โดยใช้ x3 กระจายงาน

1. ทดสอบการรันบนเครื่องเดียว โดยให้รันบน x3

	เวลา (วินาที)
เมตริก 10*10	28
	30
	26
	28
	27
ค่าเฉลี่ย	**27.8**
เมตริก 25*25	216
	246
	199
	223
	212
ค่าเฉลี่ย	**219.2**

ตารางที่ 8-5 ผลการรันบน x2

2. ทดสอบการรันบน 2 เครื่อง โดยให้รันบน x1 และ x3 ใช้ x3 เป็นตัวกระจายงาน

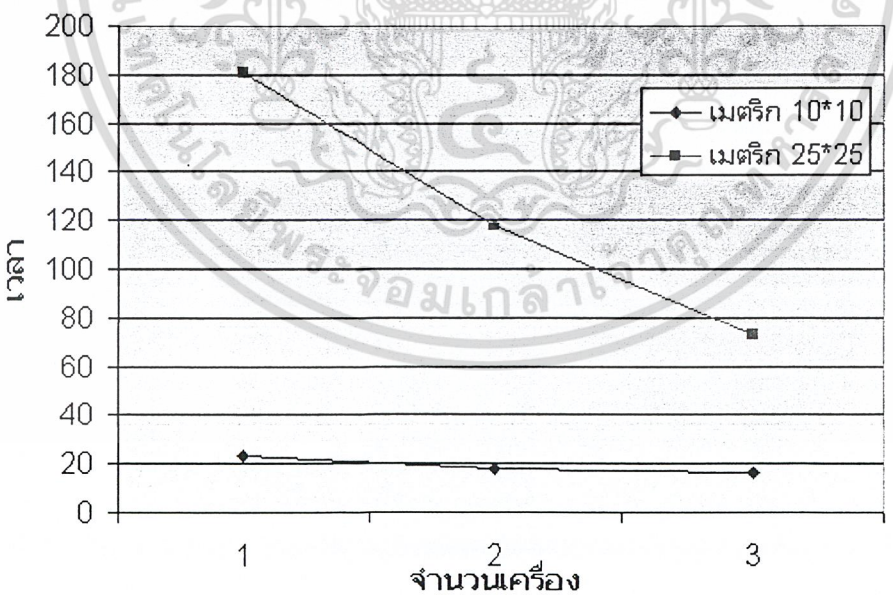
	เวลา (วินาที)
เมตริก 10*10	23
	21
	22
	24
	21
ค่าเฉลี่ย	**22.2**
เมตริก 25*25	129
	134
	122
	124
	119
ค่าเฉลี่ย	**125.6**

ตารางที่ 8-6 ผลการรันบน x1 และ x3 โดยให้ x3 เป็นตัวกระจายงาน

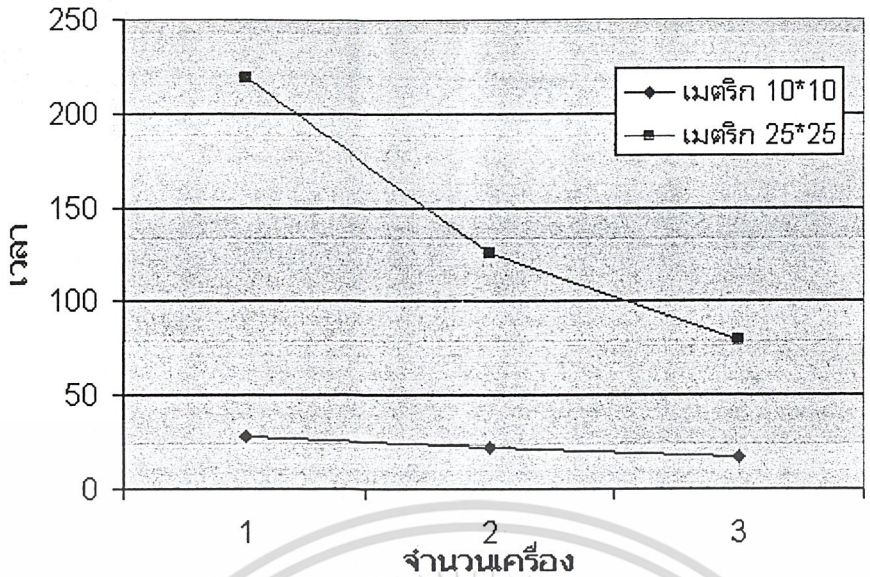
3. ทดสอบการรันบน 3 เครื่อง โดยรันบน x1, x2 และ x3 ใช้ x2 เป็นตัวกระจายงาน

	เวลา (วินาที)
เมตริก 10*10	17
	19
	17
	17
	17
ค่าเฉลี่ย	**17.4**
เมตริก 25*25	91
	79
	75
	82
	72
ค่าเฉลี่ย	**79.8**

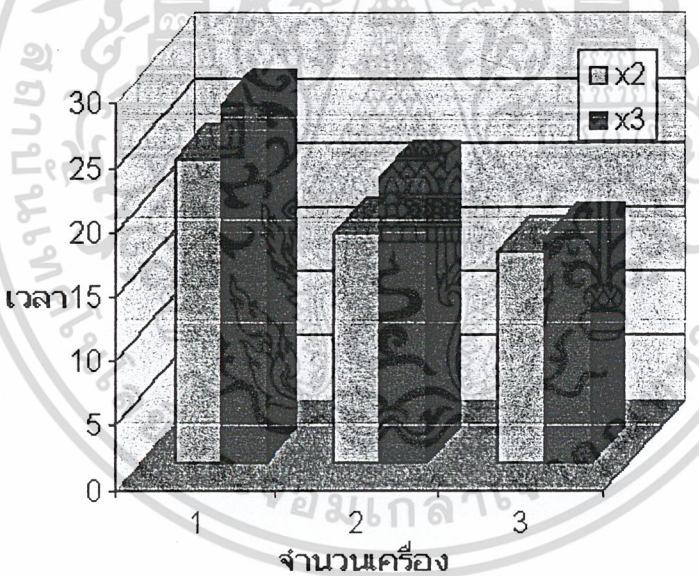
ตารางที่ 8-7 ผลการรันบน x1 ,x2 และ x3 โดยให้ x3 เป็นตัวกระจายงาน



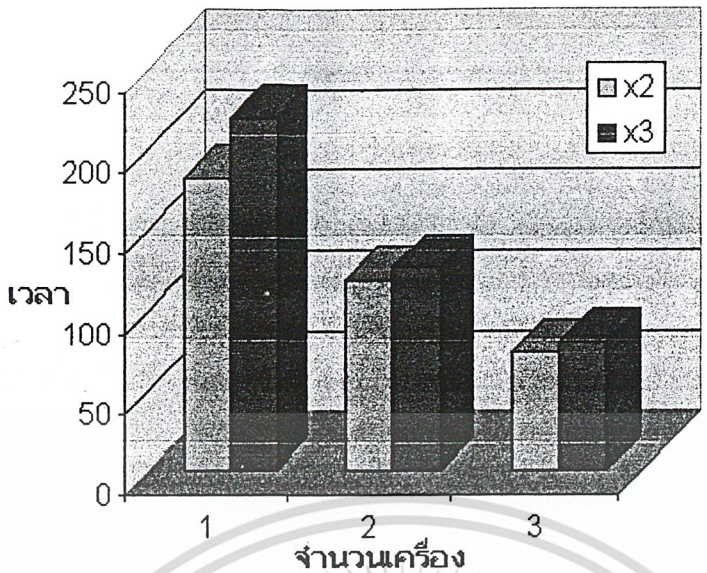
รูปที่ 8-2 กราฟแสดงความสัมพันธ์ของการจ่ายงานโดย x2



รูปที่ 8-3 กราฟแสดงความสัมพันธ์ของการจ่ายงานโดย x3



*รูปที่ 8-4 แผนภูมิเปรียบเทียบผลของการคำนวณเมตริก 10*10 โดยใช้ x2 และ x3 กระจายงาน*



*รูปที่ 8-5 แผนภูมิเปรียบเทียบผลของการคำนวณเมตริก 25*25 โดยใช้ x2 และ x3 กระจายงาน*

บทที่ 9
บทวิจารณ์และสรุป

จากการทดลองใช้งาน กริดเซอร์วิสบนระบบกริดที่สร้างขึ้น ทำให้สังเกตได้ว่าระบบกริดเป็นระบบที่มีความยืดหยุ่นมากในแง่ของการเป็น distributed system คือสามารถกระจายงานไปในหลายๆเครื่องได้โดยไม่จำกัดการกระจายงานว่าจะต้องใช้กี่เครื่อง และยังสามารถลดเวลาของการคำนวณตามจำนวนเครื่องที่สามารถเรียกใช้เซอร์วิสได้ โดยที่เมื่อมีเครื่องที่สามารถรับงานย่อยไปใช้ประมวลผลหลายๆเครื่อง จะทำให้ใช้เวลาในการคำนวณโดยรวมน้อยลง และเครื่องที่รับงานไปช่วยประมวลผลก็ยังสามารถใช้งานได้ตามปกติ

9.1 สรุปผลการทดลอง

จากผลการทดลองสามารถสรุปได้ดังนี้

- การใช้จำนวนเครื่องคอมพิวเตอร์หลายๆเครื่องร่วมกันคำนวณ จะทำให้ความเร็วโดยรวมเร็วกว่าการประมวลผลในเครื่องเดียว โดยอาศัยวิธีการแบ่งงานออกเป็นงานย่อยแล้วกระจายไปยังเครื่องอื่นให้ช่วยกันประมวลผล
- การใช้เครื่องที่มีความเร็วของ CPU มากกว่าเป็นตัวกระจายงาน จะทำให้ใช้เวลาโดยรวมน้อยกว่าการใช้เครื่องที่มี CPU ต่ำ
- การใช้งานกับการคำนวณใหญ่ๆ จะเห็นผลลัพธ์จากการใช้งานระบบกริดได้ชัดเจนกว่า เช่น การคำนวณเมตริกขนาด 25*25 จะเห็นความแตกต่างระหว่างการคำนวณบนเครื่องเดียวกับการกระจายไปหลายๆเครื่อง ได้ชัดเจนกว่าการคำนวณมตริกขนาด 10*10

9.2 ข้อจำกัดของการใช้งานระบบ

- การแบ่งงานที่ย่อยเกินไป จะทำให้เสียโอเวอร์เฮดในการส่งข้อมูลมาก ดังนั้นจึงไม่เหมาะสมที่จะใช้ในการคำนวณที่มีรายละเอียดน้อย
- หากงานย่อยแต่ละงานไม่เกี่ยวข้องกันหรือไม่ต้องรอผลลัพธ์ของกันและกันจะทำให้ประสิทธิภาพโดยรวมของการใช้งานระบบมีมากขึ้น
- เวลาที่ใช้ในการคำนวณแต่ละงานอาจไม่คงที่ เนื่องจากความเร็วจะขึ้นอยู่กับความคับคั่งของการใช้งานเน็ตเวิร์ค และโหลดของการใช้งานเครื่องที่เป็น service host
- หากมีการใช้งานเครื่องที่เป็น service host พร้อมกันด้วย จะทำให้ความเร็วในการคำนวณโดยรวมตกลง
- อาจปัญหาเกี่ยวกับพื้นที่ swap เนื่องจากการกระจายงานใช้การแบ่ง thread ซึ่งจำเป็นต้องใช้หน่วยความจำชั่วคราวเยอะ หากมีการแบ่ง thread ที่มากเกินไปอาจทำให้เกิดปัญหา stack overflow ได้

- การกระจายงาน จะใช้หลักการของการเขียนโปรแกรมแบบขนานซึ่งยังไม่แพร่หลายนัก อาจเกิดปัญหาในการนำไปพัฒนาต่อได้

9.3 แนวทางในการพัฒนาต่อ

ในการพัฒนาโปรแกรมการคำนวณเมตริกนี้ ยังขาดการใช้งานความสามารถบางอย่าง ในการใช้งานระบบกริดอยู่ เช่น การทำโหลดบาลานซ์ ซึ่งจะช่วยทำให้ประสิทธิภาพของการใช้งานดีขึ้น เนื่องจากในการพัฒนาชุดนี้ ใช้เพียงการกระจายงานให้ทุกเครื่องในระบบทำงานเท่าๆกัน ทำให้ในบางครั้ง เครื่องที่มีจำนวนโหลดมากๆ ซึ่งอาจเกิดจากการใช้งานของผู้ใช้งานเครื่องนั้นๆ คำนวณงานทั้งหมดที่ได้รับได้ช้ากว่าเครื่องอื่น หรือถ้าหากมีการนำไปใช้ในเครื่องที่มีความเร็ว CPU ต่างกันมากๆ ก็อาจเกิดปัญหาที่ต้องรอการประมวลผลจากเครื่องที่มีความเร็วต่ำๆได้ ทำให้ประสิทธิภาพของการใช้งานระบบลดลง นอกจากนี้ยังขาดโมดูลของการตรวจเช็คเครื่องที่เข้ามาใช้งานระบบให้เป็นแบบไดนามิก คือถ้าหากว่ามีเครื่องใดๆเข้ามาเพิ่มในระบบ หรือออกจากระบบ จะไม่สามารถตรวจเช็คได้อัตโนมัติ และสุดท้ายคือเรื่องของระบบการรักษาความปลอดภัย เนื่องจากการพัฒนาชุดนี้ได้ใช้ระบบการรักษาความปลอดภัยเพียงแค่การใช้งาน SimpleCA เท่านั้น ซึ่งเป็นเพียงการพิสูจน์ตนในการเข้ามาใช้งานระบบกริด เมื่อเข้ามาได้แล้วก็จะไม่มีการควบคุมอย่างอื่น ทำให้อาจเกิดปัญหาเรื่องความปลอดภัยขึ้นได้

ภาคผนวก

ภาคผนวก ก.ขั้นตอนการเซ็ตอัพระบบกริด

1. ซอฟท์แวร์ที่ใช้

1. Linux Redhat 9
2. Globus Toolkit 3.2.1 (gt3.2.1-all-linux-rh9-installer.tar.gz)
3. Apache-ant 1.6.1 (apache-ant-1.6.1-bin.tar.gz -> ant.apache.org)
4. Java J2SE 1.4.2 (j2sdk-1_4_2_04-linux-i586-rpm.bin -> java.sun.com)
5. Junit 3.8.1 (junit3.8.1.zip –> www.junit.org)

2. รูปแบบของระบบ

1. ทุกเครื่องจะต้องทำการติดตั้ง gt3.2.1 เพื่อให้สามารถใช้งาน globus container ได้ โดยจะแบ่งรูปแบบของ user เป็นดังนี้

- root ใช้ในการจัดการเกี่ยวกับ host ของระบบ
- user globus เป็นคนจัดการเกี่ยวกับการใช้งาน Globus Toolkit เช่นการ install การ execute ต่างๆ
- other user เป็นคนที่เรียกใช้งาน service ต่างๆของ globus สามารถเขียนเซอร์วิสเพิ่มให้กับระบบได้ โดยการเขียนเซอร์วิสแล้วคอมไฟล์ให้เป็นไฟล์ .gar แล้วส่งให้ user globus ทำการ deploy service

2. เครื่อง Authentication Server ใน 1 ระบบจะมีเพียงเครื่องเดียว ทำหน้าที่ในการทำ certificate ให้กับ host และ user ที่จะเข้ามาใช้งานระบบ

3. Grid Client เป็นเครื่องที่อยู่ในระบบกริด ซึ่งผ่านการทำ Certificate แล้ว ซึ่งทุกเครื่องสามารถเป็นได้ทั้ง master และ slave ของเซอร์วิสใดๆ ขึ้นอยู่กับว่าเครื่องใดจะเป็นเครื่องที่สั่งกระจายงาน และเครื่องใดมีเซอร์วิสให้เรียกใช้งานได้ โดยที่เครื่อง Authentication Server ในช่วงเวลาที่ไม่ได้ทำหน้าที่เกี่ยวกับการทำ certificate ก็จะถือว่าเป็น Grid Client เช่นกัน

3. ขั้นตอนการติดตั้ง gt3.2.1

1. ติดตั้ง Linux RedHat 9 แบบ work station
2. ใช้ root ในการลง java

```
# rpm -ivh j2sdk-1_4_2_04-linux-i586-rpm.bin
```

3. ลง apache ที่โฟลเดอร์ /usr/local

```
# tar -xzvf apache-ant-1.6.1-bin.tar.gz
```

4. ลง junit ที่โฟลเดอร์ /usr/local

```
# unzip junit3.8.1.zip
```

5. ก็อปปี้ไฟล์ junit.jar จาก junit มาที่ apache

```
# cp junit3.8.1/junit.jar apache-ant-1.6.1/lib/
```

6. เพิ่ม user globus

```
# adduser globus
```

```
# passwd globus
```

7. สร้างโฟลเดอร์ globus และกำหนดให้สิทธิ์เป็นของ user globus

```
# mkdir /usr/local/globus
```

```
# chown globus:globus /usr/local/globus
```

8. กำหนด path ของ java , apache , และ globus โดยแก้ไขไฟล์ /etc/profile

```
# vi /etc/profile
```

```
if [ -z "$INPUTRC" -a ! -f "$HOME/.inputrc" ]; then
    INPUTRC=/etc/inputrc
fi

# ------------ Edit Here -------------
JAVA_HOME=/usr/java/j2sdk1.4.2_04
ANT_HOME=/usr/local/apache-ant-1.6.1
GLOBUS_LOCATION=/usr/local/globus
PATH=$PATH:$JAVA_HOME/bin:$ANT_HOME/bin:$GLOBUS_LOCATION/bin
export JAVA_HOME
export ANT_HOME
export GLOBUS_LOCATION
#-------------------------------------

export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE INPUTRC
```

รูปที่ 1 การแก้ไข /etc/profile

9. ใช้ user globus สร้างโฟลเดอร์ /home/globus/install

```
$ mkdir /home/globus/install
```

10. แตกแพ็คเกจ gt3.2.1-all-linux-rh9-installer.tar.gz ในโฟลเดอร์ /home/globus/install

```
$ tar –xzvf gt3.2.1-all-linux-rh9-installer.tar.gz
```

11. source ไฟล์ /etc/profile

```
$ . /etc/profile
```

11. ใช้ user globus ติดตั้ง gt3.2.1 ในโฟลเดอร์ $GLOBUS_LOCATION

```
$ ./install-gt3-bin /usr/local/globus | tee installgt3.log
```

4. การทำ Certification Authentication

4.1 ขั้นตอนการเซ็ต Authentication Server

ใช้ simpleCA ซึ่งเป็นแพ็คเกจมีอยู่ใน gt3.2.1-all-linux-rh9-installer.tar.gz อยู่แล้วในการเซ็ตอัพโดยกำหนดให้ครื่อง x1 เป็นเครื่อง Authentication Server ซึ่งขั้นตอนในการทำ Authentication จะใช้ user globus เป็นคนจัดการทั้งหมดโดยมีขั้นตอนดังนี้

1. ใช้ user globus ทำการ source $GLOBUS_LOCATION/etc/globus-user-env.sh

$. $GLOBUS_LOCATION/etc/globus-user-env.sh

2. รันสคริปต์ $GLOBUS_LOCATION/setup/globus/setup-simple-ca

```
[globus@x1 globus]$ $GLOBUS_LOCATION/setup/globus/setup-simple-ca


   C e r t i f i c a t e   A u t h o r i t y   S e t u p

This script will setup a Certificate Authority for signing Globus
users certificates.  It will also generate a simple CA package
that can be distributed to the users of the CA.

The CA information about the certificates it distrubtes will
be kept in:

/home/globus/.globus/simpleCA/

The unique subject name for this CA is:

cn=Globus Simple CA, ou=simpleCA-localhost.localdomain, ou=GlobusTest, o=Grid

Do you want to keep this as the CA subject (y/n) [y]:
```

กด y เพื่อเลือกใช้ default CA

```
Enter the email of the CA (this is the email where certificate
requests will be sent to be signed by the CA):
```

เนื่องจากการติดต่อระหว่าง CA Server และ บุคคลอื่นที่ต้องการร้องขอเพื่อทำ CA จะติดต่อกันผ่านทาง E-mail จึงต้องมีการใส่ E-mail ของ CA server เอาไว้ แต่ในการทดลองนี้ ทุกเครื่องอยู่ในวงแลนเดียวกัน จึงสามารถติดต่อกันผ่านทาง nfs(network file system) ข้อมูลตรงนี้จึงไม่มีความจำเป็น สามารถใส่ค่าเป็น E-mail อะไรก็ได้ ในที่นี้ใช้ webmaster@x1

```
The CA certificate has an expiration date. Keep in mind that
once the CA certificate has expired, all the certificates
signed by that CA become invalid.  A CA should regenerate
the CA certificate and start re-issuing ca-setup packages
before the actual CA certificate expires.  This can be done
by re-running this setup script.  Enter the number of DAYS
the CA certificate should last before it expires.
[default: 5 years (1825 days)]:
```

กด enter เพื่อใช้ค่า default

```
Generating a 1024 bit RSA private key
.............................++++++
...................++++++
writing new private key to '/home/globus/.globus/simpleCA//private/cakey.pem'
Enter PEM pass phrase:
```

ใส่ password ที่จะใช้ในการ sign CA จานั้นระบบจำทำการสร้างแพ็คเกจที่ใช้ในการร้องขอ CA และแสดงผลดังนี้

```
A self-signed certificate has been generated
for the Certificate Authority with the subject:

/O=Grid/OU=GlobusTest/OU=simpleCA-localhost.localdomain/CN=Globus Simple CA

If this is invalid, rerun this script

/usr/local/globus/setup/globus/setup-simple-ca

and enter the appropriate fields.

-----------------------------------------------------------------

The private key of the CA is stored in /home/globus/.globus/simpleCA//private/ca
key.pem
The public CA certificate is stored in /home/globus/.globus/simpleCA//cacert.pem

The distribution package built for this CA is stored in

/home/globus/.globus/simpleCA//globus_simple_ca_c7881362_setup-0.17.tar.gz

--More--
```

โดยที่ จะมีการสร้างแพ็คเกจที่ใช้ในการแจกกระจายให้ผู้ใช้งานคนอื่นหรือเครื่องอื่นเอาไปใช้สร้างแพ็คเกจเพื่อร้องขอในการทำ CA โดยจะมีชื่อไฟล์เป็น globus_simple_ca_*HashNumber*_setup-0.17.tar.gz ซึ่ง *HashNumber* ในการ generate ในแต่ละครั้งจะไม่ตรงกัน จากนั้นกดคีย์ใดๆเพื่อให้โปรแกรมทำงานต่อ

```
********************************************************************************

Note: To complete setup of the GSI software you need to run the
following script as root to configure your security configuration
directory:

/usr/local/globus/setup/globus_simple_ca_c7881362_setup/setup-gsi

For further information on using the setup-gsi script, use the -help
option.  The -default option sets this security configuration to be
the default, and -nonroot can be used on systems where root access is
not available.

********************************************************************************

setup-ssl-utils: Complete

[globus@x1 globus]$
```

เมื่อ install เสร็จ โปรแกรมจะมีข้อความดังตัวอย่าง ให้ใช้ root รันสคริปต์ โดยเพิ่มออปชั่น –default ไปด้วย

$ /usr/local/globus/setup/globus_simple_ca_c7881362_setup/setup-gsi –default

```
setup-gsi: Configuring GSI security
Making /etc/grid-security...
mkdir /etc/grid-security
Making trusted certs directory: /etc/grid-security/certificates/
mkdir /etc/grid-security/certificates/
Installing /etc/grid-security/certificates//grid-security.conf.c7881362...
Running grid-security-config...
Installing Globus CA certificate into trusted CA certificate directory...
Installing Globus CA signing policy into trusted CA certificate directory...
setup-gsi: Complete
```

4.2 การเซ็ต Client ในการทำ certificate authentication

เมื่อเครื่อง client ได้รับไฟล์ globus_simple_ca_*HashNumber*_serup-1.07.tar.gz จาก Authentication Server แล้ว ให้ใช้ user globus รันสคริปต์ $GLOBUS_LOCATION/sbin/gpt-build

$ $GLOBUS_LOCATION/sbin/gpt-build globus_simple_ca_*HashNumber*_serup-1.07.tar.gz

จากนั้นใช้ root ในการรันสคริปต์ดังนี้

$GLOBUS_LOCATION/setup/globus_simple_ca_HashNumber_setup/setup-gsi -default

4.3 การร้องขอ host certification

ขั้นตอนนี้ต้องใช้ root ของแต่ละ host ที่ต้องการเข้ามาใช้งานระบบกริด เป็นคนการจัดการดังนี้

1. รันสคริปต์ grid-cert-request –host 'hostname' โดยที่ 'hostname' ให้ใช้ชื่อเต็มของ host เช่น

   $ grid-cert-request –host x1.kmil.com

   โปรแกรมจะทำการสร้างไฟล์มา 3 ไฟล์คือ

   /etc/grid-security/hostkey.pem

   /etc/grid-security/hostcert_request.pem

   /etc/grid-security/hostcert.pem (ไฟล์เปล่า)

2. ส่งไฟล์ /etc/grid-security/hostcert_request.pem ไปยัง Authentication Server เพื่อร้องขอ CA
3. ที่ Authentication Server ใช้ user glubus ในการsign CA โดยรันสคริปต์ grid-ca-sign

   $ grid-ca-sign –in hostcet_request.pem –out hostcert.pem

   จะมีการสร้างไฟล์ hostcert.pem ขึ้นมา ซึ่งเป็นไฟล์ที่ใช้ยืนยันการทำ host certification จากนั้น ส่งไฟล์นี้กลับไปยัง host ที่ร้องขอเข้ามา
4. ที่ host เมื่อได้รับไฟล์มาจาก Authentication Server แล้ว ให้ก็อบปี้ไฟล์ hostcert.pem ที่ได้รับมาไปแทนที่ /etc/grid-security/hostcert.pem เก่าซึ่งเป็นไฟล์เปล่า

4.4 การร้องขอ user certification

เป็นการทำ certificate ของ user ทุกคนที่จะเข้ามาใช้งานระบบกริด มีขั้นตอนดังนี้

1. รันสคริปต์ grid-cert-request

   $ grid-cert-request

   โปรแกรมจะมีการสร้างไฟล์มา 3 ไฟล์คือ

   /home/$USER/.globus/usercert.pem (ไฟล์เปล่า)

   /home/$USER/.globus/userkey.pem

   /home/$USER/.globus/usercert_request.pem
2. ส่งไฟล์ /home/$USER/.globus/usercert_request.pem ไปยัง Authentication Server เพื่อร้องขอ CA
3. ที่ Authentication Server ใช้ user globus ในการ sign CA โดยรันสคริปต์ grid-ca-sign

$ grid-ca-sign –in usercert_request.pem –out usercert.pem

โปรแกรมจะทำการสร้างไฟล์ usercert.pem ซึ่งเป็นไฟล์ที่ใช้ยืนยันการทำ user certification จากนั้นก็ส่งไฟล์นี้กลับไปหา user ที่ร้องขอ

4. เมื่อ user ได้รับไฟล์จาก Authentication Server แล้ว ให้ก๊อปปี้ไฟล์ที่ได้มาแทนที่ไฟล์ usercert.pem เดิม
5. ทดสอบว่า การทำ CA สำเร็จหรือไม่ โดยการรันสคริปต์ grid-proxy-init –debug –verify

   $ grid-proxy-init –debug –verify

   ซึ่งจะแสดงข้อมูลของการทำ certificate

```
User Cert File: /home/user/.globus/usercert.pem
User Key File: /home/user/.globus/userkey.pem

Trusted CA Cert Dir: /etc/grid-security/certificates
Output File: /tmp/x509up_u1817
Your identity: /O=Grid/OU=GlobusTest/OU=simpleCA-
x1.kmitl.com/OU=kmitl.com/CN=User1
Enter GRID pass phrase for this identity:
```

จากนั้นใส่ password หากถูกต้องจะมีข้อความว่า

```
Creating proxy .....++++++++++++
.....++++++++++++
Done Proxy Verify OK
Your proxy is valid until: Sat March 10 16:01:46 2010
```

ซึ่งวันที่ก็จะขึ้นอยู่กับ ตั้งค่า expire date ในขั้นตอนการเซ็ต Authentication Server

6. ใช้ root รันสคริปต์ $GLOBUS_LOCATION/bin/setperms.sh
7. ใช้ root สร้าง /etc/grid-security/grid-mapfile ในการเพิ่ม user ในการใช้งานกริด คยที่รายละเอียดจะได้จากการใช้ user นั้นๆ รันสคริปต์ grid-proxy-info –subject เช่น

   user1$ grid-cert-info -subject

   /O=Grid/OU=GlobusTest/OU=simpleCA-x1.kmitl.com/OU=kmitl.com/CN=user1

   user1$ whoami

   user1

   ข้อความที่จะเพิ่มใน grid-mapfile ก็จะเป็น

"/O=Grid/OU=GlobusTest/OU=simpleCA-x1.kmitl.com/OU=kmitl.com/CN=user1" user1

5. การทดสอบการใช้งานระบบกริดที่สร้างขึ้น

1.ใช้ user globus รันคำสั่ง globus-start-container

$ globus-start-container

ซึ่งจะมีลิสต์ของเซอร์วิสโชว์ขึ้นมาหลายอัน โดยหนึ่งในนั้นจะมี

http://161.246.6.93:8080/ogsa/services/base/gram/MasterForkManagedJobFactoryService \

2. ใช้ user ที่ทำ certificate เรียบร้อยแล้ว รันสคริปต์ดังนี้

$ managed-job-globusrun -factory \

http://161.246.6.93:8080/ogsa/services/base/gram/MasterForkManagedJobFactoryService \

-file schema/base/gram/examples/test.xml

ถ้าระบบสามารถทำงานได้ถูกต้องจะมีข้อความบอกว่า

```
========== Status Notification ==========
Job Status: StageIn
==========================================
========== Status Notification ==========
Job Status: active
==========================================
========== Status Notification ==========
Job Status: done
==========================================
DESTROYING SERVICE
```

จบขั้นตอนนี้ถือว่าการเซ็ตอัพระบบ กริดโดยใช้ gt3.2.1 เสร็จสิ้น แต่จะยังไม่ได้คอนฟิกในส่วนของเซอร์วิสพื้นฐานอื่นๆ เช่น gridFTP , RFT, Index Service ซึ่งไม่จำเป็นต้องคอนฟิกทุกเซอร์วิส สามารถคอนฟิกแค่เซอร์วิสที่ต้องการใช้งานก็พอ รายละเอียดเพิ่มเติมสามารถดูได้จาก http://www-unix.globus.org/toolkit/docs/3.2/index.html

ภาคผนวก ข. วิธีการเขียน GWSDL

ส่วนนี้เป็นการแสดงวิธีการเขียน GWSDL อย่างง่าย ที่จะใช้อธิบายรายละเอียดของ PortType แม้ว่าส่วนนี้จะเป็นส่วนที่ง่ายต่อการเขียน แต่ก็เป็นส่วนที่ค่อนข้างสำคัญส่วนหนึ่ง ที่จะทำให้ PortTypes ที่จะใช้งานมีความยืดหยุ่นมากขึ้น

การเริ่มต้นเขียน GWSDL จำเป็นต้องรู้รูปแบบการเขียน Java ขั้นต้นก่อน โดยดูได้จากโค้ดนี้

```
public interface Math
{
  public void add(int a);
  public void subtract(int a);
  public int getValue();
}
```

ซึ่งนี่เป็นรูปแบบพื้นฐานที่ใช้งานกันทั่วไป
ขั้นแรกจะต้องเขียนค่า root ของ GWSDL ก่อน โดยกำหนดตามนี้

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MathService"
  targetNamespace="http://www.globus.org/namespaces/2004/02/progtutorial/MathService"
  xmlns:tns="http://www.globus.org/namespaces/2004/02/progtutorial/MathService"
  xmlns:ogsi="http://www.gridforum.org/namespaces/2003/03/OGSI"
  xmlns:gwsdl="http://www.gridforum.org/namespaces/2003/03/gridWSDLExtensions"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
</definitions>
```

โดยส่วนนี้มีค่าที่สำคัญ 2 ค่า คือ

- name: เป็นชื่อของไฟล์ GWSDL โดยไม่ได้เกี่ยวข้องกับชื่อของ PortType
- targetNamespace: เป็นเนมสเปซเป้าหมายของไฟล์ GWSDL ซึ่งหมายถึง PortTypes และคำสั่งทั้งหมดที่กำหนดในไฟล์ GWSDL ที่อยู่ในขอบเขตเนมสเปซนี้ ในกรณีที่ไม่คุ้นเคยกับการเนมสเปซ ของ XML ควรจะรวมกลุ่มที่คล้ายกันเป็นกลุ่มเดียว ซึ่งเนมสเปซของ XML จะมีค่า

URI ที่ถูกต้องแต่ไม่จำเป็นต้องมีจริง (เช่น เจอ URI นั้น แต่จะได้ค่า Page Not Found error เป็นต้น)

โดยค่า root นั้นจะใช้ประกาศเนมสเปซทั้งหมดที่กำลังจะนำไปใช้งาน ซึ่งต่อไปนี้จะใช้ tns แทน Target NameSpace

ขั้นต่อมาจะเป็นการอิมพอร์ทไฟล์ OGSI GWSDL ที่กำลังจะใช้งานต่อไป

```
<import location="../../ogsi/ogsi.gwsdl"
namespace="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
```

ขั้นนี้จะกำหนด PortType ที่ใช้แท็ก <gwsdl:portType> (ถ้าเป็น WSDL ตามต้นฉบับเดิม จะใช้แท็ก <portType> แทน) ต่อมาจะกำหนดเนมสเปซของ gwsdl ในค่าของ root

```
<definitions ... ">
<gwsdl:portType name="MathPortType" extends="ogsi:GridService">
 <operation name="add">
  <input message="tns:AddInputMessage"/>
  <output message="tns:AddOutputMessage"/>
  <fault name="Fault" message="ogsi:FaultMessage"/>
 </operation>
 <operation name="subtract">
  <input message="tns:SubtractInputMessage"/>
  <output message="tns:SubtractOutputMessage"/>
  <fault name="Fault" message="ogsi:FaultMessage"/>
 </operation>
 <operation name="getValue">
  <input message="tns:GetValueInputMessage"/>
  <output message="tns:GetValueOutputMessage"/>
  <fault name="Fault" message="ogsi:FaultMessage"/>
 </operation>
</gwsdl:portType>
</definitions>
```

ซึ่งแท็ก <gwsdl:portType> จะมีค่าที่สำคัญ 2 ค่า

- name: เป็นชื่อของ PortType
- extends: ค่านี้จะยอมให้กำหนด PortType ที่เป็นการขยายจาก PortType ที่คงอยู่ ในกรณีตัวอย่างนี้จะขยายจาก ogsi:GridService PortType ซึ่งกริดเซอร์วิสทุกเซอร์วิสต้องขยายจาก PortType นี้

ใน <gwsdl:portType> จะมีแท็ก <operation> ที่ใช้สำหรับกำหนด method ของ PortType: เช่น add, subtract, getValue เป็นต้น ซึ่งทั้งหมดจะกำหนดในรูปแบบคล้ายๆกัน โดยพิจารณาที่แท็ก <operation>

```
<operation name="add">
  <input message="tns:AddInputMessage"/>
  <output message="tns:AddOutputMessage"/>
  <fault name="Fault" message="ogsi:FaultMessage"/>
</operation>
```

ในแท็ก <operation> จะมีแท็ก <input>, <output> และ <fault> ซึ่งทั้ง 3 แท็กจะมีข้อความ ที่ใช้ระบุว่าข้อความนั้นจะส่งผ่านเมื่อคำสั่งถูกร้องขอ (จาก input message) เมื่อส่งค่ากลับแล้ว (เป็น output message) หรือเป็นข้อความแจ้งความผิดพลาด (เป็น fault message) โดยที่ fault message จะกำหนดใน OGSI GWSDL ที่มีอยู่แล้ว อย่างไรก็ตาม ก็ยังจำเป็นต้องกำหนดข้อความของคำสั่งที่เราสร้างด้วย ส่วนนี้จะเป็นข้อความที่ใช้กับคำสั่ง add (ถ้าใช้กับคำสั่ง subtract ก็แค่เปลี่ยนจาก ‘add’ เป็น ‘substract’)

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions ... >
<message name="AddInputMessage">
 <part name="parameters" element="tns:add"/>
</message>
<message name="AddOutputMessage">
 <part name="parameters" element="tns:addResponse"/>
</message>
&lt!-- PortType -->
</definitions>
```

การกำหนดค่าส่วนนี้จะใช้โครงของ XML ข้างในแท็กใหม่ คือ แท็ก <types> ส่วนต่อไปนี้จะกำหนด add และค่า addResponse

```
<types>
<xsd:schema
targetNamespace="http://www.globus.org/namespaces/2004/02/progtutorial/MathService"
 attributeFormDefault="qualified"
 elementFormDefault="qualified"
 xmlns="http://www.w3.org/2001/XMLSchema">

 <xsd:element name="add">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="value" type="xsd:int"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>

 <xsd:element name="addResponse">
  <xsd:complexType/>
 </xsd:element>

</xsd:schema>
</types>
&lt!-- Messages -->
&lt!-- PortType -->
</definitions>
```

โดยแท็ก <types> จะมีแท็ก <xsd:schema> อยู่ ค่าของ <xsd:schema> ควรจะก๊อปปี้ตรงตามที่กำหนดไว้ ยกเว้นแค่ targetNamespace ซึ่งควรจะเป็นค่าเดียวกับเนมสเปซเป้าหมายของเอกสาร GWSDL

ค่าบวก จะประกอบไปด้วยค่าที่ถูกเรียก ซึ่งเป็นค่าของคำสั่งบวกนั้น (ค่า type นั้นจะเท่ากันกับ xsd:int ซึ่งเป็นค่า integer ในโครงของ XML)

ค่า addResponse นั้นจะไม่มีค่าทั้งหมด เนื่องจากคำสั่งบวกนั้นไม่ได้ส่งค่าอะไรกลับมา

ส่วน <messages> ที่เหลือ และค่าทั้งหมดที่กำหนดในส่วนต่อไปนี้ เป็นไฟล์ GWSDL อย่างที่ควรจะเป็น

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MathService"
 targetNamespace="http://www.globus.org/namespaces/2004/02/progtutorial/MathService"
 xmlns:tns="http://www.globus.org/namespaces/2004/02/progtutorial/MathService"
 xmlns:ogsi="http://www.gridforum.org/namespaces/2003/03/OGSI"
 xmlns:gwsdl="http://www.gridforum.org/namespaces/2003/03/gridWSDLExtensions"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns="http://schemas.xmlsoap.org/wsdl/">

<import location="../../ogsi/ogsi.gwsdl"
 namespace="http://www.gridforum.org/namespaces/2003/03/OGSI"/>

<types>
<xsd:schema
targetNamespace="http://www.globus.org/namespaces/2004/02/progtutorial/MathService"
 attributeFormDefault="qualified"
 elementFormDefault="qualified"
 xmlns="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="add">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="value" type="xsd:int"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
 <xsd:element name="addResponse">
  <xsd:complexType/>
 </xsd:element>
 <xsd:element name="subtract">
  <xsd:complexType>
```

```
    <xsd:sequence>
     <xsd:element name="value" type="xsd:int"/>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
  <xsd:element name="subtractResponse">
   <xsd:complexType/>
  </xsd:element>
  <xsd:element name="getValue">
   <xsd:complexType/>
  </xsd:element>
  <xsd:element name="getValueResponse">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:element name="value" type="xsd:int"/>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
 </xsd:schema>
 </types>

 <message name="AddInputMessage">
  <part name="parameters" element="tns:add"/>
 </message>
 <message name="AddOutputMessage">
  <part name="parameters" element="tns:addResponse"/>
 </message>
 <message name="SubtractInputMessage">
  <part name="parameters" element="tns:subtract"/>
 </message>
 <message name="SubtractOutputMessage">
  <part name="parameters" element="tns:subtractResponse"/>
 </message>
 <message name="GetValueInputMessage">
```

```
  <part name="parameters" element="tns:getValue"/>
 </message>
 <message name="GetValueOutputMessage">
  <part name="parameters" element="tns:getValueResponse"/>
 </message>

 <gwsdl:portType name="MathPortType" extends="ogsi:GridService">
  <operation name="add">
   <input message="tns:AddInputMessage"/>
   <output message="tns:AddOutputMessage"/>
   <fault name="Fault" message="ogsi:FaultMessage"/>
  </operation>
  <operation name="subtract">
   <input message="tns:SubtractInputMessage"/>
   <output message="tns:SubtractOutputMessage"/>
   <fault name="Fault" message="ogsi:FaultMessage"/>
  </operation>
  <operation name="getValue">
   <input message="tns:GetValueInputMessage"/>
   <output message="tns:GetValueOutputMessage"/>
   <fault name="Fault" message="ogsi:FaultMessage"/>
  </operation>
 </gwsdl:portType>

 </definitions>
```

สรุปแล้ว ขั้นตอนที่จะสร้างไฟล์ GWSDL นั้น จะเป็นไปตามนี้

เขียน root element <definitions>

เขียน <gwsdl:PortType>

เขียน อินพุท และ เอาท์พุท <message> สำหรับแต่ละคำสั่งใน PortType

เขียน <types>

ซึ่งขั้นตอนที่กล่าวไปนี้เป็นเพียง 1 ในวิธีการเขียน GWSDL เท่านั้น ยังมีวิธีการเขียนได้อีกหลากหลายรูปแบบ ที่จะใช้ความรู้ของ WSDL และ XML มากขึ้นกว่านี้ เพื่อความยืดหยุ่นของงานที่จะทำ

ภาคผนวก ค. ความปลอดภัยในระบบกริด

1. ระบบความปลอดภัยพื้นฐานในกริด

เนื่องจากระบบกริด เป็นการกระจายรีซอร์สไปในระบบเครือข่าย ทำให้ระบบกริดนั้นจำเป็นจะต้องมีการสื่อสารที่ปลอดภัย ข้อมูลที่ส่งไปจะต้องเชื่อถือได้ มีความถูกต้อง จึงจะทำให้ระบบกริดใช้งานได้อย่างมีประโยชน์จริงๆ

2. การสื่อสารที่ปลอดภัย

การสื่อสารที่ปลอดภัยนั้น ต้องมีคุณสมบัติ 3 อย่าง คือ ความเป็นส่วนตัว (privacy) มีความมั่นคง (integrity) และมีการรับรองสิทธิ (authentication) แต่การสื่อสารบางกรณีนั้นก็ไม่จำเป็นต้องมีครบทั้ง 3 ข้อ ตามแต่กรณีไป เช่น ต้องการแต่ความเป็นส่วนตัว เป็นต้น

2.1 ความเป็นส่วนตัว (Privacy)

การสื่อสารที่มีความเป็นส่วนตัวนั้น กล่าวคือ มีเพียงผู้รับและผู้ส่งเท่านั้นที่สามารถเข้าใจการสื่อสารนั้นได้ ถ้ามีผู้ดักฟังข้อมูลจะต้องไม่เข้าใจการสื่อสารนั้น ส่วนนี้โดยทั่วไปจะใช้การเข้ารหัสและการถอดรหัสมาช่วยในการจัดการ (Encryption/Decryption)

ยกตัวอย่างเช่น เราต้องการส่งข้อความว่า "INVOKE METHOD ADD" และต้องการให้บุคคลที่ 3 ที่ดักข้อมูลไป (เช่น ใช้สนิฟเฟอร์ดักข้อมูล) ไม่สามารถเข้าใจบทความนี้ได้ จะต้องใช้กลวิธีในการเข้ารหัสเข้าช่วย เช่น เปลี่ยนตัวอักษรเป็นอัลฟาเบ็ทตัวอื่นๆ ตามแต่อัลกอริธึมเป็นต้น โดยการส่ง "INVOKE METHOD ADD" อาจจะส่งเป็น "JOWPLFANFUIPEABEE" เป็นต้น (จะเห็นว่าตัวอย่างนี้จะบวก 1 เข้าไปแต่ละอัลฟาเบ็ท โดยมี A ที่ใช้แทนช่องว่าง) บุคคลที่ 3 ที่ดักข้อมูลไปจะได้รับเป็นภาษาที่ไม่สามารถเข้าใจได้ แต่ฝ่ายผู้รับนั้นจำเป็นต้องรู้อัลกอริธึมในการถอดรหัสกลับมาด้วย ในการเข้ารหัสและถอดรหัสนั้นจะมีวิธีมากมายหลายวิธี ซึ่งจะกล่าวถึงในส่วนต่อๆไป

2.2 ความมั่นคง (integrity)

การสื่อสารที่มีความมั่นคงนั้น กล่าวคือ ผู้รับนั้นจะต้องแน่ใจได้ว่า ข้อมูลที่ได้รับนั้นเป็นข้อมูลที่จะถูกส่งมาให้เขาจริงๆ โดยไม่ผ่านการเปลี่ยนแปลง แก้ไข เพิ่มเติมโดยบุคคลที่ 3

ส่วนนี้นั้นมีความสำคัญมาก เช่น ในกรณีที่มีการเข้ารหัสข้อมูล แต่บุคคลที่ 3 ที่ประสงค์ร้ายนั้น แม้จะไม่ทราบวิธีการถอดรหัสข้อมูล แต่อาจเปลี่ยนแปลงแก้ไขข้อมูลบางส่วนได้ ซึ่งอาจทำให้ผู้รับข้อมูลได้รับข้อมูลที่ผิดพลาดได้ ส่วนนี้จะใช้การเข้ารหัสที่ใช้คีย์สาธารณะเข้าช่วยในการป้องกันการโจมตีชนิดนี้ โดยผู้รับจะสามารถทราบได้ว่าข้อมูลไม่ได้ผ่านการแก้ไขระหว่างทาง

2.3 การรับรอง (authentication)

การสื่อสารที่มีการรับรองนั้น จะต้องแน่ใจว่าผู้ที่สื่อสารจะต้องมีสิทธิในการสื่อสารนั้น กล่าวคือ จะต้องป้องกันการปลอมตัวเข้ามาอยู่ในกลุ่มของการสื่อสารนั้นๆได้ ซึ่งเป็นวิธีที่โจมตีได้ง่ายๆ สามารถใช้เครื่องมือเช่น สนิฟเฟอร์ ได้ ซึ่งจะใช้วิธีการเข้ารหัสในการป้องกันการโจมตีประเภทนี้

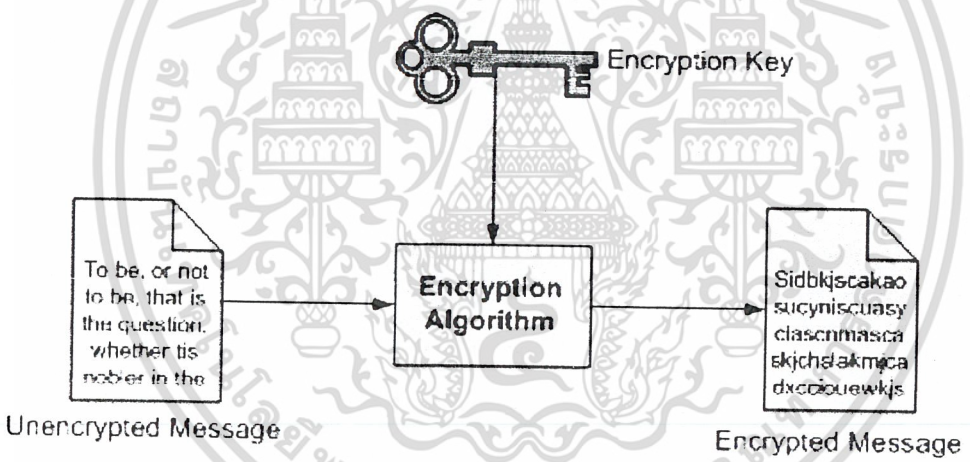
การรับรองนั้นจะเกี่ยวข้องกับการให้อนุญาต (authorization) ด้วย จึงจะได้รับสิทธิในการสื่อสารนั้นๆ

3. หนังสือที่เขียนด้วยอักษรลับ (Cryptography)

Cryptography โดยรากศัพท์เป็นศาสตร์การเขียนด้วยอักษรลับ กล่าวคือ เป็นการเข้ารหัสข้อความปกติที่ใช้ส่งด้วยตัวอักษรที่เป็นความลับ (ที่เรียกกันว่า "encrypted message") โดยการถอดรหัสจะเป็นการนำตัวอักษรลับเหล่านั้นมาเปลี่ยนให้อยู่ในรูปข้อความที่ตีความได้ (ที่เรียกกันว่า "unencrypted message") ซึ่งการใช้หนังสือลับนี้จะทำให้ได้คุณสมบัติของการสื่อสารที่ปลอดภัยครบทั้ง 3 คุณสมบัติ

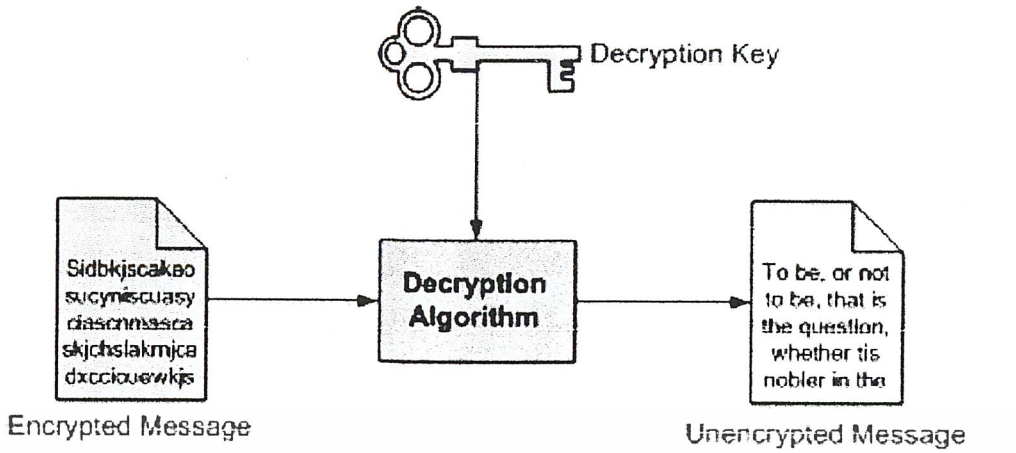
3.1 อัลกอริธึมที่ใช้ในการทำหนังสือที่เขียนด้วยอักษรลับ

จะใช้ คีย์ เข้าช่วยในการทำหนังสือที่เขียนด้วยอักษรลับ โดยจะใช้คีย์เข้ารหัส (Encryption Key) ซึ่งหมายความว่า การเข้ารหัสนั้นจะไม่ได้ใช้เพียงข้อความ แต่ใช้คีย์จัดการทั้งหมด ตามรูป



รูปที่ 1 แสดงการใช้คีย์เข้ารหัสข้อความ

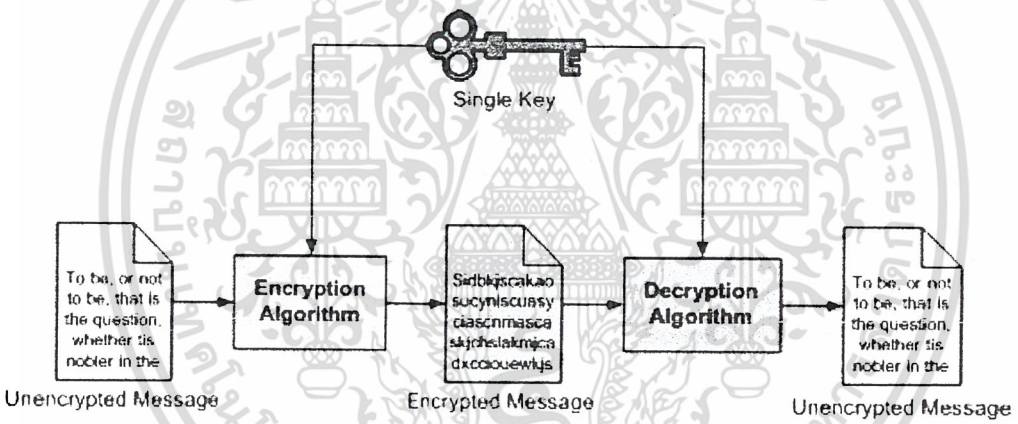
โดยที่ผู้รับข้อความนั้น จะใช้คีย์ถอดรหัส (Decryption Key) ในการถอดรหัสข้อความ ซึ่งหมายความว่า ไม่ได้ใช้อัลกอริธึมในการถอดรหัส แต่ใช้คีย์



รูปที่ 2 แสดงการใช้คีย์ถอดรหัสข้อความ

3.2 อัลกอริธึมที่ขึ้นกับคีย์แบบสมมาตรและไม่สมมาตร

อัลกอริธึมที่ขึ้นกับคีย์แบบสมมาตร เป็นการใช้คีย์เดียวกันในการเข้ารหัสและถอดรหัส ดังรูป



รูปที่ 3 แสดงอัลกอริธึมที่ขึ้นกับคีย์แบบสมมาตร

แม้ว่าอัลกอริธึมนี้จะใช้รวดเร็วและง่ายต่อการนำไปใช้งาน แต่มีข้อจำกัดหลายอย่าง อย่างแรกที่สำคัญ คือ สามารถรับรองความเป็นส่วนตัวได้อย่างเพียงอย่างเดียว (ความมั่นคงและการรับรองสิทธิต้องใช้วิธีอื่นเข้าช่วย) ส่วนข้อจำกัดอื่นๆ เช่น ทั้งผู้รับและผู้ส่งจะต้องยอมรับคีย์ที่ใช้เพื่อสร้างความปลอดภัยนั้นด้วย (แต่ไม่ใช่ปัญหาที่สำคัญนัก)

ระบบความปลอดภัยที่ใช้งานในปัจจุบันนั้น ส่วนมากจะใช้อัลกอริธึมที่ขึ้นกับคีย์แบบไม่สมมาตร ซึ่งใช้คีย์ที่ต่างกันในการเข้ารหัสและถอดรหัส โดยใช้อัลกอริธึมที่ใช้คีย์สาธารณะ ที่จะแนะนำในหัวข้อต่อไปนี้

3.3 หนังสือลับที่ใช้คีย์สาธารณะ (Public key cryptography)

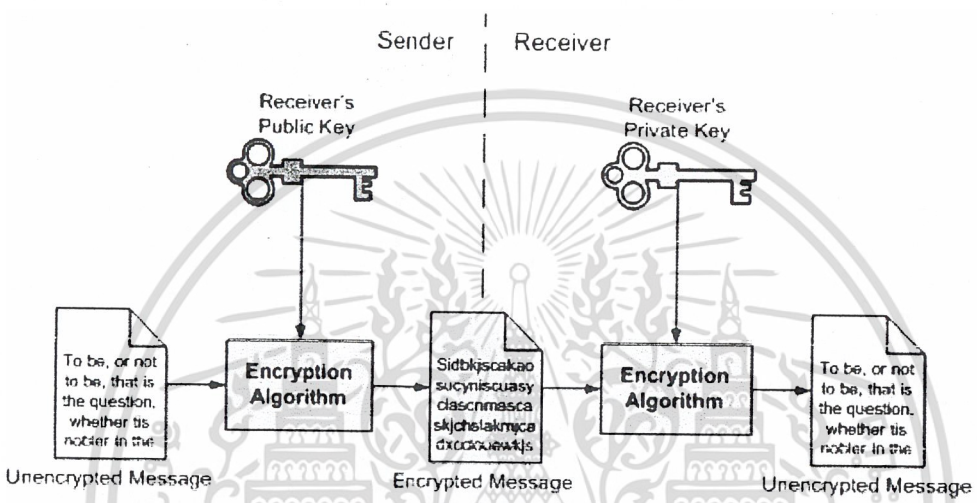
เป็นอัลกอริธึมที่ขึ้นกับคีย์แบบไม่สมมาตร โดยใช้คีย์ที่ต่างกันที่เรียกว่า คีย์ส่วนตัว (Private key) และคีย์สาธารณะ (Public key) ซึ่ง

คีย์ส่วนตัว จะเป็นคีย์ที่ทราบเฉพาะเจ้าของข้อความนั้น

คีย์สาธารณะ จะเป็นคีย์ที่ทุกคนทราบ

ความสัมพันธ์ระหว่างคีย์ทั้งสองนั้น จะใช้ตัวหนึ่งเป็นตัวเข้ารหัส และใช้อีกตัวเป็นตัวถอดรหัส เช่นผู้ส่งเข้ารหัสด้วยคีย์สาธารณะ ผู้รับก็จะต้องถอดรหัสด้วยคีย์ส่วนตัว เป็นต้น

ซึ่งจะแสดงวิธีการให้ดูตามรูปนี้

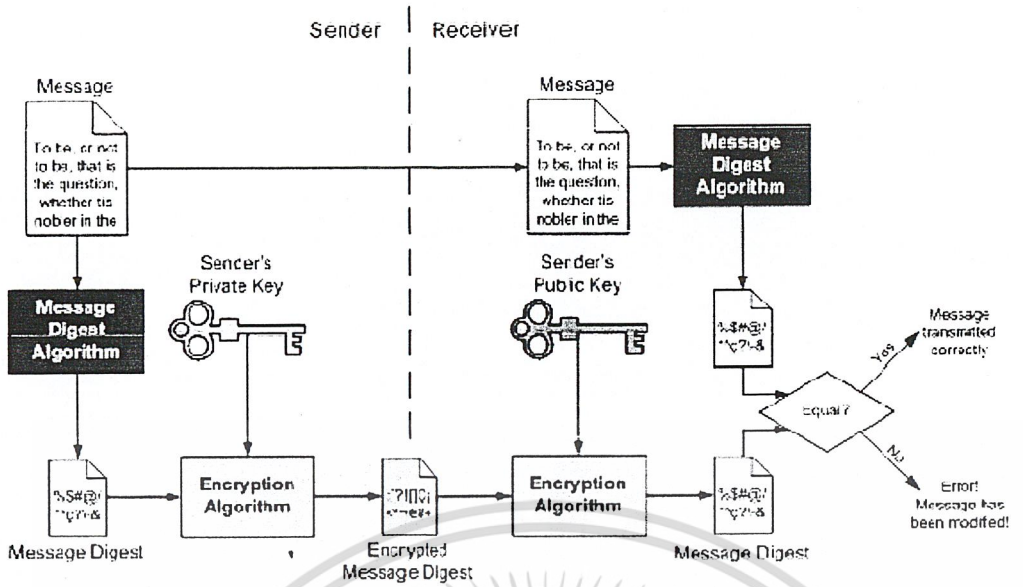


รูปที่ 4 แสดงการใช้คีย์ส่วนตัวและคีย์สาธารณะ

ซึ่งการใช้งานรูปแบบนี้จะค่อนข้างปลอดภัย ตราบเท่าที่ผู้รับข้อความยังเก็บคีย์ส่วนตัวไว้เป็นความลับได้อยู่ เนื่องจากการคำนวณหาคีย์ส่วนตัวจากคีย์สาธารณะที่ทราบนั้น ทำได้ค่อนข้างยาก ใช้เวลานานในการคำนวณออกมา ซึ่งบางอัลกอริธึมนั้นอาจใช้เวลานานถึง หลายๆเดือน หรือ อาจจะใช้เวลาเป็นปีในการคำนวณซึ่งข้อความนั้น คงไม่มีอายุถึงขนาดนั้นในระบบเครือข่าย

4. การใช้งานลายมือชื่อดิจิตัล (digital signature) รวมกับระบบคีย์สาธารณะ

ในการใช้งานลายมือชื่อดิจิตัลรวมกับคีย์สาธารณะนั้น จะช่วยเพิ่มความปลอดภัยมากขึ้น โดยลายมือชื่อดิจิตัลนั้น คือ ชิ้นส่วนของข้อมูลที่เพิ่มเข้าไปในข้อความและใช้ในการตรวจสอบว่าข้อมูลนั้นถูกใช้ระหว่างทางหรือไม่ (อาจเกิดจากผู้ประสงค์ร้าย เป็นต้น)



รูปที่ 5 การใช้งานลายมือชื่อดิจิตัลรวมกับคีย์สาธารณะ

ส่วนที่เกี่ยวข้องกับลายมือชื่อดิจิตัลในฝั่งผู้ส่งนั้น จะมี 2 ขั้นตอน กล่าวคือ

1. ข้อความย่อย (Message Digest) จะถูกสร้างขึ้น โดยข้อความย่อยนั้นที่ใช้ส่ง จะมี 2 คุณสมบัติที่สำคัญ คือ จะต้องมีขนาดเล็กกว่าข้อความนั้นๆ และ มีการเปลี่ยนแปลงน้อยที่สุดในการสร้างข้อความของแต่ละข้อความย่อยที่แตกต่างกัน โดยข้อความย่อยนั้นอาจจะสร้างด้วยอัลกอริธึมแฮชชิ่ง (hashing)
2. ข้อความย่อยนั้นจะเข้ารหัส โดยคีย์ส่วนตัวของผู้ส่ง ซึ่งจะให้ผลลัพธ์เป็น ลายมือชื่อดิจิตัล

ลายมือชื่อดิจิตัลที่ถูกใช้ในข้อความ และส่งไปให้ผู้รับนั้น ฝั่งผู้รับจะทำตามขั้นตอนดังนี้

1. ใช้คีย์สาธารณะของผู้ส่งในการถอดรหัสลายมือชื่อดิจิตัลนั้นเพื่อรับข้อความย่อยที่สร้างโดยฝั่งผู้ส่ง
2. ใช้อัลกอริธึมข้อความย่อยอันเดียวกันที่ใช้ในฝั่งผู้ส่งเพื่อสร้างข้อความย่อยของข้อความที่รับมา
3. เปรียบเทียบข้อความย่อยทั้ง 2 (จากฝั่งผู้ส่งและจากผู้รับเอง) ถ้าข้อความนั้นไม่ตรงกัน แสดงว่าข้อความนั้นถูกแก้ไขโดยบุคคลที่ 3 ซึ่งผู้รับมั่นใจได้ว่าลายมือชื่อดิจิตัลนั้นส่งมาจากผู้ส่งจริง เนื่องจากมีเพียงคีย์สาธารณะของผู้ส่งเท่านั้นที่จะถอดลายมือชื่อดิจิตัลออกมาได้ตรงกัน ยกเว้นกรณีที่คีย์สาธารณะจะสร้างข้อความย่อยที่ผิดพลาดออกมาเอง แต่มันไม่ใช่ประเด็นที่ใช้ในการส่งข้อความผ่านเครือข่าย

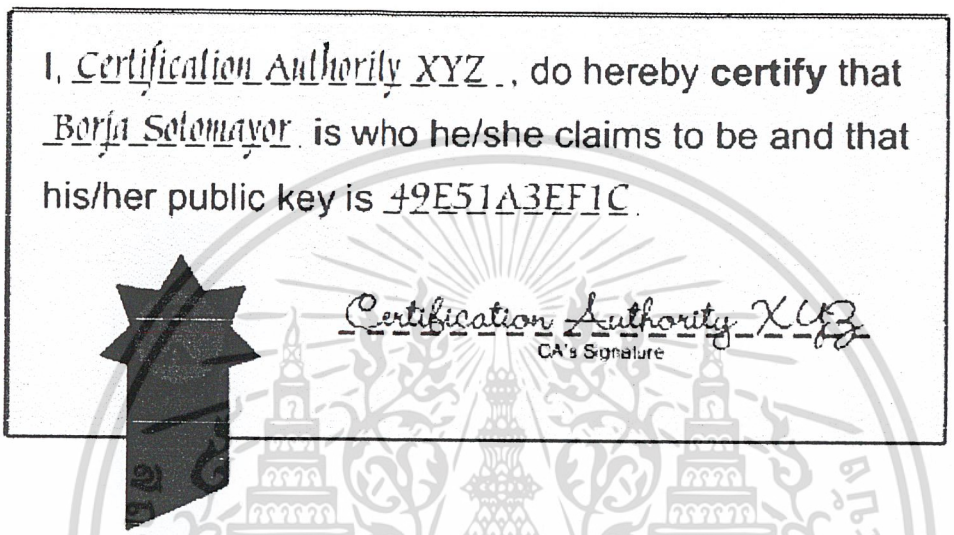
5. การรับรองสิทธิในระบบสาธารณะ

จากตัวอย่างก่อนหน้านี้ จะรับรองได้ว่าผู้ส่งเป็นคนที่มีสิทธิจริงๆ ได้เพียงกรณีที่ฝั่งผู้ส่งใช้คีย์สาธารณะในการถอดรหัสค่าลายมือชื่อดิจิตัล (ถูกเข้ารหัสโดยคีย์ส่วนตัว) อย่างไรก็ตาม ส่วนเดียวที่รับรองได้ คือ รับรองว่าผู้ส่งมีคีย์ส่วนตัวที่ตรงกันกับคีย์สาธารณะ เท่านั้น แม้ว่าคีย์สาธารณะนั้นจะเป็นของผู้ส่งซึ่งอาจจะเป็นใครก็ได้ ซึ่งจะแน่ใจได้อย่างไรว่าผู้ส่งเป็นผู้ไม่ประสงค์ร้าย หรือ ไม่ใช่ผู้ส่งตัวปลอม

ด้วยเหตุดังที่กล่าวซึ่งแสดงให้เห็นว่ามีจุดอ่อนในด้านการมีสิทธินั้นๆ จึงจำเป็นต้องใช้วิธีการที่เพิ่มขึ้นเพื่อความปลอดภัยที่เรียกว่า ใบรับรองดิจิตัล (digital certificate) ซึ่งจะอธิบายในหัวข้อต่อๆไป

5.1 ใบรับรองและการให้อำนาจใบรับรอง (Certificates and Certificate Authorities)

ใบรับรองดิจิตัล เป็นเอกสารดิจิตัลที่รับรองว่าคีย์สาธารณะนั้นเป็นของเป็นของผู้ใช้นั้นๆหรือไม่ โดยเอกสารนี้จะลงชื่อโดยบุคคลที่ 3 เรียกว่าใบรับรองสิทธิ (certificate authority) หรือ CA ซึ่งภาพด้านล่างนี้จะทำให้มองเห็นลักษณะของใบรับรองดิจิตัล



รูปที่ 6 ตัวอย่างใบรับรองดิจิตัล

แน่นอนว่าใบรับรองจะถูกเอนโค้ดในรูปแบบดิจิตัล ซึ่งสิ่งสำคัญที่ต้องจำไว้ก็คือ ใบรับรองจะถูกลงชื่อโดยบุคคลที่ 3 ซึ่งเกี่ยวข้องกับสื่อสารที่ปลอดภัยนั้นๆ ซึ่งการลงชื่อนั้นอาจใช้ผ่านลายมือชื่อดิจิตัลที่สร้างโดยคีย์ส่วนตัวของ CA

5.2 รูปแบบใบรับรอง X.509

รูปแบบใบรับรอง X.509 นั้น เป็นมาตรฐาน ที่ใช้ในการเอนโค้ดใบรับรองดิจิตัล โดยมีรูปแบบเป็นเท็กซ์ไฟล์แบบง่ายๆที่มีข้อมูลที่จำเป็นมากมายที่จำเป็นต้องใช้ ซึ่งได้แบ่งออกเป็น 4 ส่วนที่สำคัญ ดังนี้

Subject: ส่วนนี้เป็นชื่อของผู้ใช้ จะถูกเอนโค้ดเป็นชื่อที่แบ่งแยกได้

Subject's public key: ส่วนนี้รวมทั้งคีย์สาธารณะที่ใช้และอัลกอริธึมที่ใช้สร้างมันด้วย

Issuer's subject: เป็นชื่อที่ใช้แบ่งแยกของ CA

Digital signature: เป็นใบรับรับรองที่รวมลายมือชื่อดิจิตัลของข้อมูลทั้งหมดของใบรับรอง ส่วนนี้จะสร้างโดยคีย์ส่วนตัวของ CA และเพื่อใช้ตรวจสอบลายมือชื่อดิจิตัลจะต้องมีคีย์สาธารณะของ CA อีกอันด้วย

ซึ่งรายละเอียดทั้งหมดที่กล่าวมา ในรูปใบรับรองรูปก่อนหน้านี้ ได้แสดงไว้ครบทุกค่าแล้ว

5.3 ชื่อที่ใช้แบ่งแยก (Distinguished name)

ชื่อในรูปแบบของใบรับรอง X.509 นั้นจะไม่ได้เอนโค้ดในรูปแบบชื่อตามปกติ อย่างเช่น "Borja Satomayor" หรือ "Certificate Authority XYZ" หรือ "System Administrator" แต่มันจะเอนโค้ดในรูปแบบที่เรียกว่า ชื่อที่ใช้แบ่งแยก (distinguished name) ซึ่งจะมีคู่ของชื่อในรูปแบบที่แตกต่างกัน เช่นด้านล่างนี้

O=University of Deusto, OU=Department of Software Engineering, CN=Borja Satomay

ซึ่ง "O", "OU" และ "CN" จะมีความหมายตามนี้

O: Organization

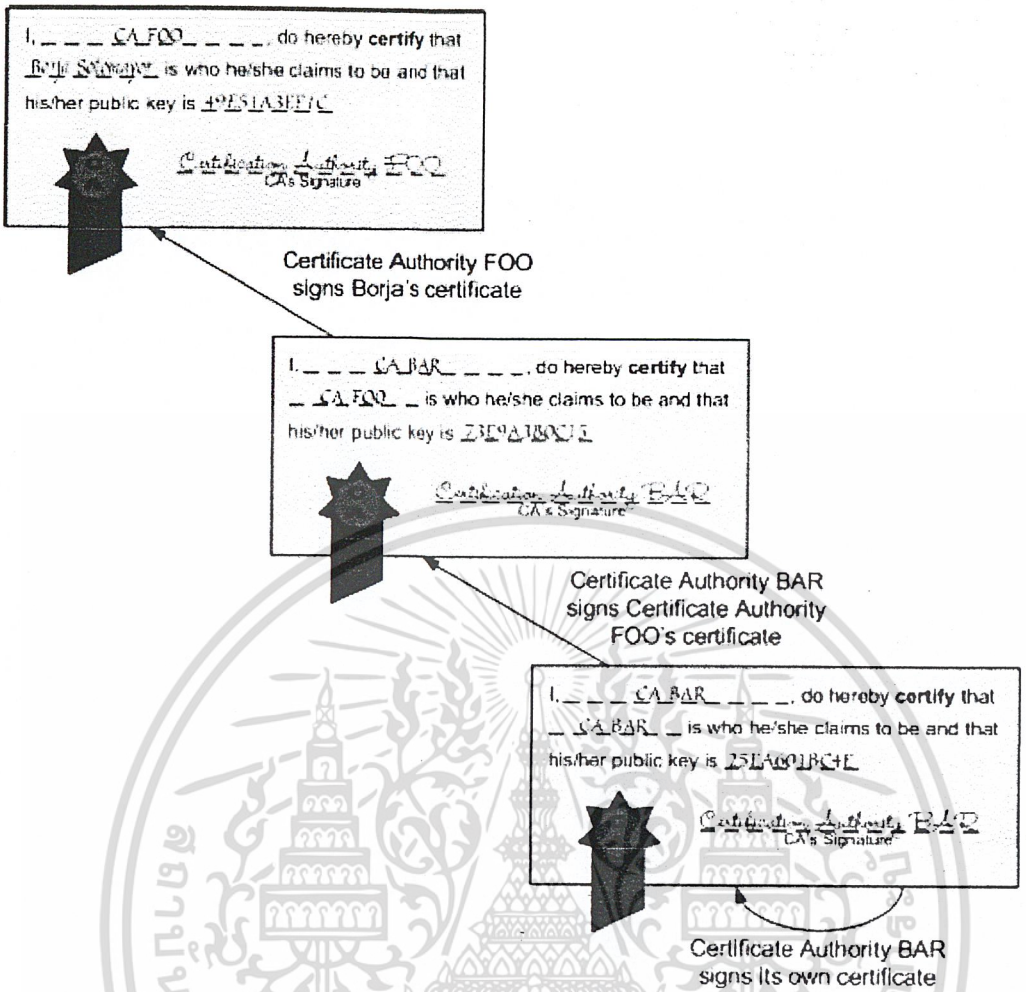
OU: Organizational Unit

CN: Common Name (เช่น ชื่อผู้ใช้)

C: Country

5.4 ลำดับชั้นของ CA

ในการใช้งาน CA นั้น ผู้ใช้จะต้องมี กลุ่ม CA ที่เชื่อถือ (trusted CA list) ได้ที่ CA ทั้งหมดนั้นจะถูกตัดสินว่าเชื่อถือได้ ซึ่งการตัดสินใจย่อมขึ้นอยู่กับผู้ใช้งาน และผู้ที่ลงชื่อให้ใบรับรองของ CA นั้นๆ ซึ่งก็คือ CA อันอื่นนั่นเอง ซึ่งทำให้เกิดเป็นลำดับชั้นของ CA ที่ถูกสร้างขึ้น ซึ่งผู้ใช้งานจะต้องเชื่อถือ CA ที่อยู่ระดับสูงกว่าที่ลงชื่อให้ใบรับรองนั้น (ซึ่งทำให้ CA ที่อยู่ระดับที่ต่ำกว่ามีความน่าเชื่อถือน้อยลงไปเรื่อยๆ ตามลำดับชั้น) ซึ่งรูปต่อไปนี้ จะทำให้เห็นภาพได้ชัดเจนขึ้น



รูปที่ 7 แสดงตัวอย่างลำดับชั้นของ CA

ในภาพนั้นจะแสดงให้เห็นว่ามี มีใบรับรองที่ลงชื่อโดย CA FOO ซึ่งใบรับรองของ FOO จะถูกลงชื่อโดย CA ที่ชื่อว่า BAR และใบรับรอง BAR จะลงชื่อโดยตัวมันเอง

ถ้าผู้รับได้รับใบรับรองอันหนึ่งนั้น และไม่เชื่อถือ CA FOO นั่นไม่ได้หมายความว่า ใบรับรองของผู้ส่งไม่สามารถเชื่อถือได้โดยอัตโนมัติ ผู้รับต้องพิจารณาว่า CA FOO นั้นเชื่อถือได้หรือไม่ แต่ถ้าปรากฏว่า CA BAR นั้นอยู่ในรายการที่เชื่อถือได้ นั่นหมายความว่าใบรับรองนั้นเชื่อถือได้โดยปริยาย

อย่างไรก็ตาม ถ้า CA ชั้นที่สูงกว่า (BAR) ลงชื่อโดยตัวมันเอง นี่ไม่ใช่กรณีตามปกติ แต่จะเรียกว่า "self-signed certificate" ซึ่ง CA ที่เป็นตามกรณีนี้จะเรียกว่า root CA เพราะว่า ไม่มีใครที่สูงกว่ามันแล้ว การจะถูกเชื่อถือโดย CA นี้ หมายความว่า จะต้องอยู่ในรายการที่เชื่อถือได้ของมันเท่านั้น

บรรณานุกรม

[1] Globus Alliance, http://www.globus.org

[2] Java, http://java.sun.com

[3] Apache Ant, http://ant.apache.org

[4] Junit, http://www.junit.org

[5] What is the Grid, http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf

[6] Gestalt of the Grid, http://www.mcs.anl.gov/~gregor/ papers/vonLaszewski--gestalt.pdf

[7] The Anatomy of the Grid: Enabling Scalable Virtual Organizations, http://www.globus.org/research/papers/anatomy.pdf

[8] The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, http://www.globus.org/research/papers/ogsa.pdf

[9] Gt3 Core, http://www-unix.globus.org/toolkit/3.0/ogsa/docs/gt3_core.pdf

[10] Globus Toolkit 3 Quick Start, http://www.redbooks.ibm.com/redpapers/pdfs/redp3697.pdf

[11] The Globus Toolkit 3 Programmer's Tutorial, http://www.casa-sotomayor.net/gt3-tutorial/

[12] How to install Glubus and write a hello world service, http://www.dutchgrid.nl/install/gang/ HowTo-GlobusInstallationExperience.pdf

[13] Implementing a Distributed Master/Slave Grid Service with Globus Toolkit 3 (GT3), http://dps.uibk.ac.at/~gregor/mandel.pdf

[14] Ian Foster, http://www-fp.mcs.anl.gov/~foster/

[15] IBM Redbook, http://www.ibm.com/redbooks/