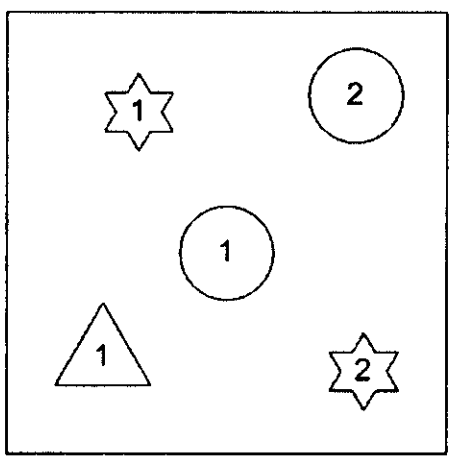
บทที่ 4

การออกแบบและสร้างโมดูล

ในบทนี้จะบรรยายถึงแนวคิดสำหรับการออกแบบระบบการจำลองเพื่อนำไปสู่การสร้างโมดูลของ HLA เพิ่มเติมลงในโปรแกรม ns ซึ่งจะครอบคลุมทั้ง Object Management, Data Distribution Management และ Time Management

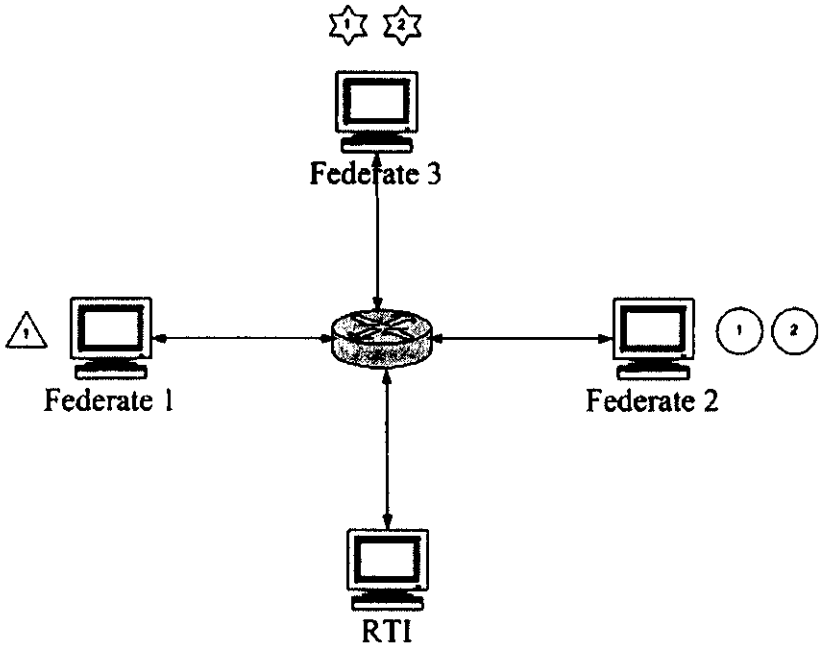
4.1 แนวคิดเบื้องต้นสำหรับการออกแบบ

สำหรับในส่วนแรกนี้จะอธิบายถึงแนวคิดเบื้องต้นที่จะใช้เป็นรากฐานสำหรับการออกแบบส่วนของโปรแกรม เพื่อสะดวกแก่การทำความเข้าใจในเบื้องต้นจะใช้การอธิบายโดยยกตัวอย่างเชิงเปรียบเทียบเพื่อให้เกิดแนวคิดในเชิงกว้างก่อน ส่วนในรายละเอียดจะกล่าวถึงในส่วนต่อไป



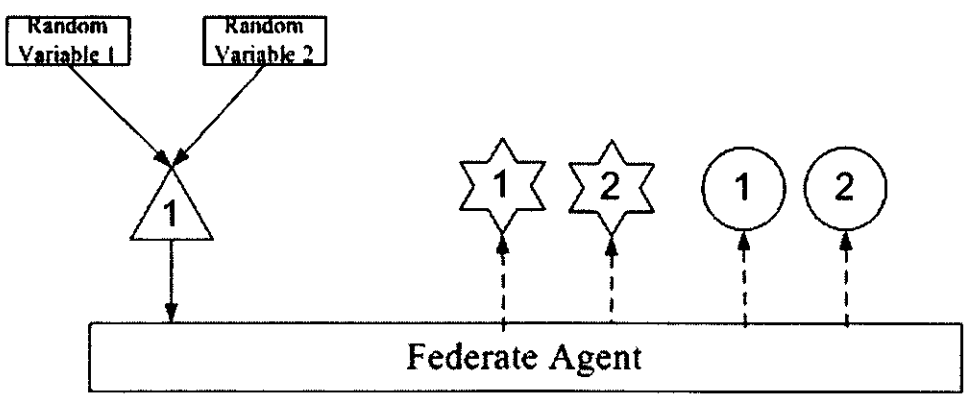
รูปที่ 4.1 แสดงโลกเสมือนซึ่งประกอบด้วยอ็อบเจกต์ 3 ชนิด

สมมติให้ในโลกเสมือน (Virtual World) ของการจำลองใดๆ ชนิดหนึ่ง ซึ่งประกอบด้วยคลาส 3 ชนิด คือ คลาสของวงกลม สามเหลี่ยม และ ดาว สำหรับวงกลมและดาวประกอบด้วยอ็อบเจกต์ 2 ชนิด คือที่มีหมายเลข 1 และ 2 กำกับอยู่ ส่วนคลาสรูปดาวมีเพียงอ็อบเจกต์เดียว สำหรับแต่ละคลาสจะถูกกระจายอยู่บนคอมพิวเตอร์ทั้ง 3 เครื่อง ดังแสดงลักษณะการทำงานจริงในรูปแบบของเครือข่ายคอมพิวเตอร์ในรูปที่ 4.2



รูปที่ 4.2 แสดงการกระจายอ็อบเจกต์ให้แก่แต่ละ Federate

การทำงานจริง HLA Federation อาจมีลักษณะหนึ่งที่เป็นไปได้ ซึ่งแต่ละ Federate จะทำงานอยู่บนเครื่องคอมพิวเตอร์หนึ่งเครื่อง และแต่ละเครื่องเชื่อมต่อผ่านทางเราเตอร์ตัวหนึ่ง ดังแสดงในรูปที่ 4.2 แต่อย่างไรก็ตาม HLA ไม่ได้กำหนดให้แต่ละ Federate ให้ต้องอยู่ในคอมพิวเตอร์ 1 เครื่อง ซึ่งหมายความว่า ในแต่ละเครื่องคอมพิวเตอร์อาจมี Federate มากว่า 1 Federate ก็เป็นไปได้ แต่เพื่อความสะดวกจะกำหนดให้แต่ละคอมพิวเตอร์ รับผิดชอบเพียง 1 Federate เท่านั้น โดยแต่ละ Federate จะรับผิดชอบการจำลองของแต่ละคลาส และจะทำงานประสานกันผ่านทาง RTI

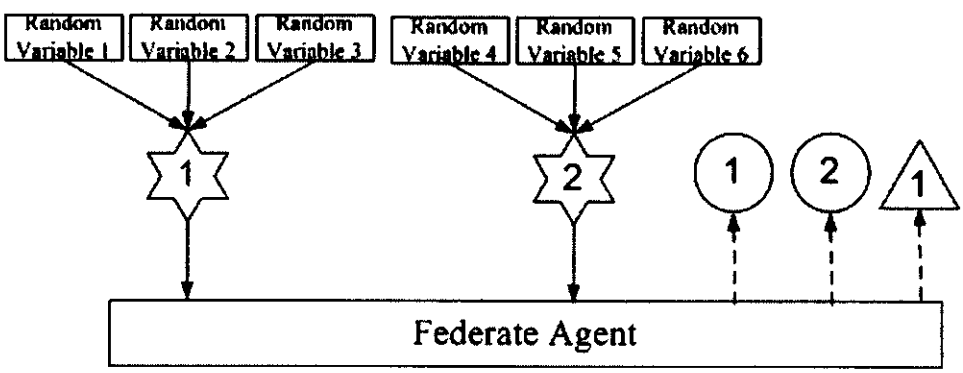


รูปที่ 4.3 แสดงโครงสร้างการทำงานของ Federate 1

หากพิจารณาที่ Federate 1 จะมีลักษณะดังรูป 4.3 ซึ่งแสดงให้เห็นการจำลองของอ็อบเจกต์เพียง 1 อ็อบเจกต์ คือ อ็อบเจกต์สามเหลี่ยม และจะต้องส่งข้อมูลที่เป็นผลลัพธ์จากการคำนวณออกไปสู่ RTI ผ่านทางส่วนการทำงานที่เรียกว่า Federate Agent ซึ่งจะอธิบายรายละเอียดการทำงานในภายหลัง แต่ในขณะเดียวกัน Federate 1 ก็ต้องสามารถมองเห็นอ็อบเจกต์วงกลมและดาว อย่างละ 2 อ็อบเจกต์ ซึ่งจะรับผิดชอบการคำนวณโดย Federate 2 และ 3 ซึ่งเมื่อทำการคำนวณเสร็จก็จะต้องส่งผลลัพธ์มายัง Federate 1 ผ่านทาง RTI การที่ Federate 1 สามารถมองเห็นสำหรับการทำงานของ Federate 3 ซึ่งก็สามารถพิจารณาได้ในทำนองเดียวกันกับ Federate 1 โดยใช้รูปที่ 4.4 ประกอบ และสำหรับ Federate 2 ให้พิจารณาลักษณะเดียวกับ Federate 3 เพียงแต่สลับระหว่างดาวและวงกลมในทุกตำแหน่ง

สำหรับรูป 4.3 เราจะจัดอ็อบเจกต์สามเหลี่ยมเป็น Simulation Object เพราะเป็นอ็อบเจกต์ที่ Federate 1 ทำการคำนวณค่าเอง และจะจัดรูปดาวและวงกลมเป็น Virtual Object เพราะเป็นอ็อบเจกต์เสมือนที่การคำนวณทำที่อื่นแต่ส่งข้อมูลกลับมาให้ Federate 1 ได้เห็นเสมือนกับได้ทำการคำนวณที่ Federate 1 เอง

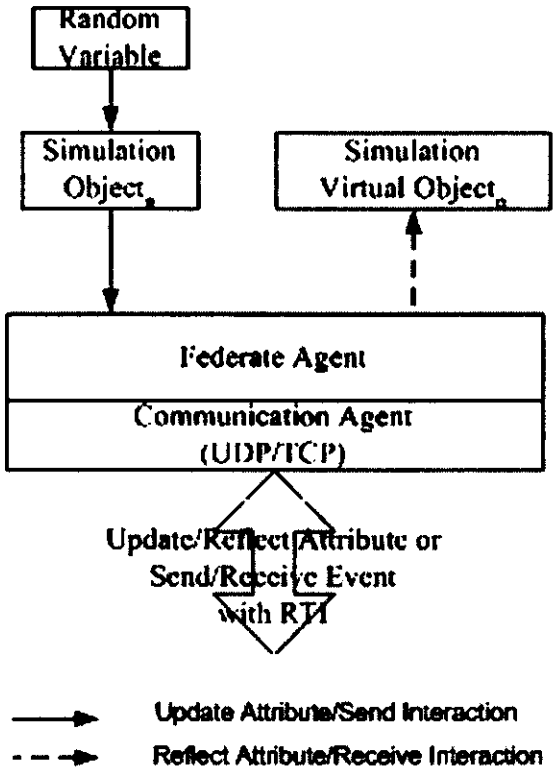
สำหรับตัวแปรสุ่มที่ปรากฏอยู่ในรูปที่ 4.3 และ 4.4 นั้นเพื่อแสดงให้เห็นว่าอ็อบเจกต์ใดๆที่ถูกคำนวณจะสามารถถูกกำหนดพารามิเตอร์ได้ด้วยตัวแปรสุ่มชนิดต่างๆ และยังได้แสดงให้เห็นว่าการออกแบบอ็อบเจกต์ลงใน ns ยังจะต้องสนับสนุนการทำงานร่วมกับส่วนของโปรแกรมตัวแปรสุ่มด้วยโดยไม่จำกัดจำนวน



รูปที่ 4.4 แสดงโครงสร้างการทำงานของ Federate 3

ดังนั้นจึงสรุปแนวคิดการออกแบบโดยรวมได้ โดยแสดงในรูป 4.5 ซึ่งส่วนของโปรแกรมที่จะต้องสร้างขึ้นใหม่คือ ส่วนของคลาส Simulation Object, Simulation Virtual Object, และ

Federate Agent รวมทั้งส่วนของโปรแกรมที่จำลองการทำงานของ RTI ทั้งนี้ในส่วนของโปรแกรม Random Variable และ Communication Agent เป็นส่วนที่มีอยู่แล้วใน ns



รูปที่ 4.5 แสดงการทำงานร่วมกันเพื่อจำลองการทำงานของ Federate

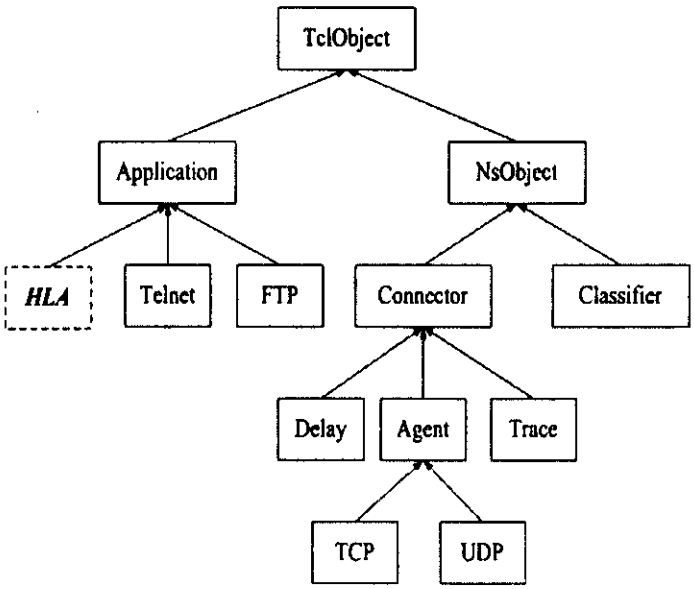
4.2 แนวคิดการออกแบบในเชิงโครงสร้าง

ในส่วนนี้จะกล่าวถึงโครงสร้างของระบบในระดับส่วนของโปรแกรม ซึ่งจะอธิบายให้เห็นถึงความสัมพันธ์ระหว่างส่วนของโปรแกรมต่างๆในระบบ

ก่อนอื่นขอเสนอโครงสร้างในภาพรวมของโมดูลใน ns ดังแสดงในรูปที่ 4.6 โดย ns ได้ออกแบบให้โปรโตคอลในชั้นโปรแกรมประยุกต์ทั้งหมดสืบทอดจากคลาส Application เช่น โมดูลการจำลองของ FTP หรือ Telnet และเช่นเดียวกับโมดูล HLA ซึ่งผู้วิจัยได้ออกแบบให้อยู่ในชั้นโปรแกรมประยุกต์ จึงต้องสืบทอดจากคลาส Application ด้วยเช่นเดียวกัน

จากในหัวข้อที่ 4.1 ในรูป 4.5 จะเห็นได้อย่างชัดเจนว่าส่วนของโปรแกรมที่จะต้องสร้างขึ้นมาใหม่เพื่อทำการจำลองการทำงานของ HLA ซึ่งประกอบด้วย ส่วนของคลาส Simulation Object, Simulation Virtual Object, และ Federate Agent รวมทั้งส่วนของโปรแกรมที่จำลองการ

ทำงานของ RTI นั้น ล้วนแล้วแต่ต้องอยู่เหนือชั้น Communication Agent ซึ่งทำหน้าที่แทน Transport Layer ซึ่งเป็นองค์ประกอบที่มีอยู่แล้วใน ns นั่นคือ ส่วนของโปรแกรมที่สร้างขึ้นใหม่จึงจัดให้อยู่ใน Application Layer

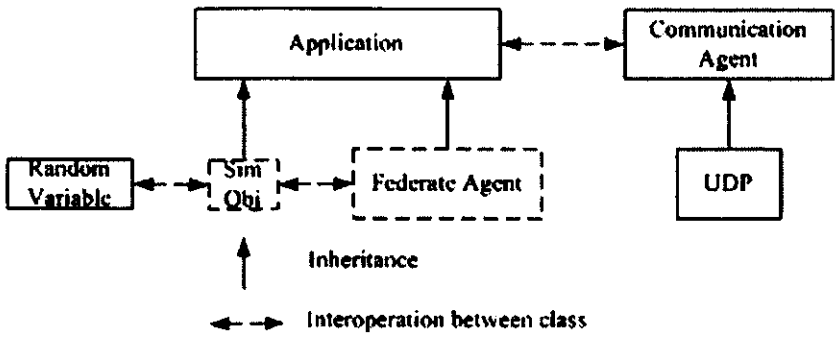


รูปที่ 4.6 แสดงโครงสร้างโดยรวมของโมดูลใน ns

จากรูปที่ 4.7 และ 4.8 รูปกล่องที่เป็นเส้นประเป็นคลาสที่ได้สร้างขึ้นเพิ่มเติม ส่วนคลาสที่แสดงด้วยเส้นทึบนั้นเป็นคลาสที่มีอยู่แล้วใน ns

คลาส Federate Agent จะทำหน้าที่แทน Federate Ambassador และ RTI Ambassador จะทำการสืบทอด (Inheritance) จากคลาสแม่คือ คลาส Application เพราะต้องการให้คลาส Federate Agent สามารถทำงานประสานกับ Communication Agent ได้ โดยเฉพาะในส่วนของ Attribute Distribution ซึ่งแสดงในรูป 4.7 นั้น Federate Agent จะต้องทำงานร่วมกับ UDP (User Datagram Protocol) ซึ่งเป็นคลาสลูกของคลาส Communication Agent

ตาม Service Specification (IEEE 1516.1-2000) ได้กำหนดให้โปรโตคอลสำหรับ Attribute Distribution เป็นแบบ Best Effort โดยยกตัวอย่างโปรโตคอลที่เป็นไปได้ คือ UDP แต่ในทางปฏิบัติใน RTI-NG และ pRTI ซึ่งเป็น RTI ที่ได้รับความนิยมล้วนใช้ UDP สำหรับ Attribute Distribution (Sjöström, Johansson and Nyberg, 2000) แต่สำหรับ Interaction Distribution ตาม Service Specification ได้กำหนดไว้แต่เพียงให้เป็น Reliable โดยยกตัวอย่างโปรโตคอลที่เป็นไปได้ คือ TCP (Transmission Control Protocol) แต่ในทางปฏิบัติก็ใช้ TCP (Sjöström, Johansson and Nyberg, 2000)

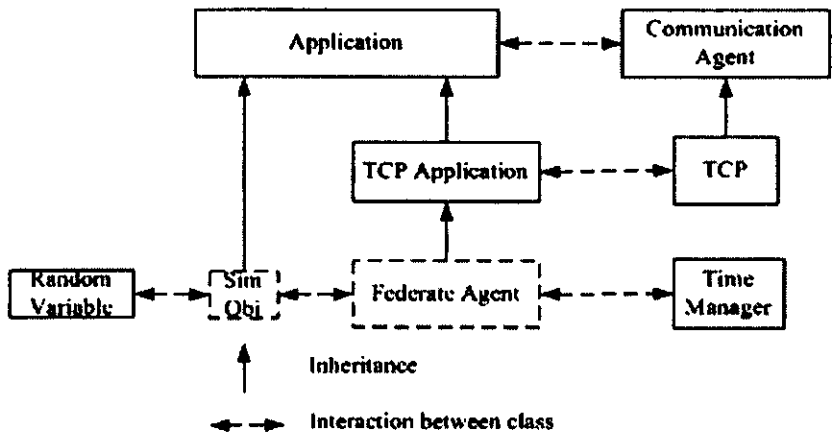


รูปที่ 4.7 แสดงการสร้างส่วนของโปรแกรมเพื่อทำงานส่วน Attribute Distribution

คลาส Simulation Object ทำหน้าที่แทน Simulation Object จริงๆ ซึ่งเป็นผู้ผลิตข้อมูลหรือแหล่งกำเนิดข้อมูล โดยข้อมูลจะลักษณะทางสถิติเป็นเช่นไร เช่น ค่าเฉลี่ยหรือช่วงของการสุ่ม ก็ขึ้นอยู่กับธรรมชาติของตัวแปรสุ่มที่ผู้ใช้กำหนดให้ คลาส Simulation Object จะสืบทอดจาก คลาส Application เช่นเดียวกัน เพื่อให้สามารถทำงานร่วมกับคลาส Random Variable ได้

การทำงานร่วมกันได้ซึ่งแสดงไว้ในรูปที่ 4.7 และ 4.8 ด้วยรูปลูกศรเส้นประนั้นแทนความหมายที่ว่า คลาสต่างๆนั้นสามารถเรียกใช้ส่วนของโปรแกรมภายในคลาสซึ่งกันและกันได้ เช่น ระหว่างคลาส Random Variable และ คลาส Simulation Object มีลูกศรเส้นประอยู่แสดงว่าคลาสทั้งสองสามารถเรียกใช้ส่วนของโปรแกรมภายในซึ่งกันและกันได้เหตุที่ทำเช่นได้ก็เพราะอาศัยคุณสมบัติประการหนึ่งของการโปรแกรมเชิงอ็อบเจกต์ที่เรียกว่า ภาวะหลายรูปแบบ ทั้งนี้เพราะว่า Random Variable มีหลายชนิดและในอนาคตอาจมีชนิดใหม่เกิดขึ้น ซึ่งถ้าหากคลาส Random Variable ชนิดใหม่นี้ยังคงสืบทอดคุณสมบัติจากคลาสแม่ Random Variable ก็สามารถมั่นใจได้ว่าจะยังทำงานร่วมกับคลาส Simulation Object ได้อย่างปกติ โดยแนวคิดที่เกี่ยวกับภาวะหลายรูปแบบนี้ได้กล่าวถึงแล้วในบทที่ 3

สำหรับในส่วน Interaction Distribution การสืบทอดของคลาสแสดงดังรูป 4.8 ซึ่งใช้ TCP ในการขนส่งข้อความในการจัดการเวลาคือ TAR, NER ,TAG และ ข้อความที่เป็นการระบุการเกิดขึ้นของเหตุการณ์ (Event) (Sjöström, Johansson and Nyberg, 2000) ทั้งนี้เพราะข้อความเหล่านี้มีอาจสูญหายได้ และจะต้องทำงานร่วมกับคลาส Time Manger ซึ่งจะทำการจัดการเวลาเพื่อการเข้าจังหวะ (Synchronization) ระหว่าง Federate ด้วยการใช้การหลักการคำนวณ Lower Bound on the Time Stamp (LBTS) สำหรับรายละเอียดของคลาส Time Manager จะอธิบายในส่วนของ Time Management ในหัวข้อ 4.5



รูปที่ 4.8 แสดงการสร้างส่วนของโปรแกรมเพื่อทำงานส่วน Interaction Distribution

นอกจากนี้ปัญหาใหญ่ประการหนึ่งของ ns ก็คือการที่ถูกออกแบบมาให้ทำการจำลองด้วยข้อมูลเสมือน คือการส่งข้อมูลระหว่างกันไม่สามารถระบุข้อความที่จำเพาะเจาะได้ ทำให้ Federate ไม่สามารถส่งข้อความ TAR หรือ NER จริงๆได้ นั่นคือ RTI จะรับรู้แค่ว่า Federate ได้ส่งข้อมูลมาจำนวนหนึ่ง แต่ไม่สามารถแยกแยะได้ว่า เป็น TAR หรือ NER ซึ่งก็ทำให้ RTI ไม่สามารถตอบ TAG กลับไปได้เช่นกัน

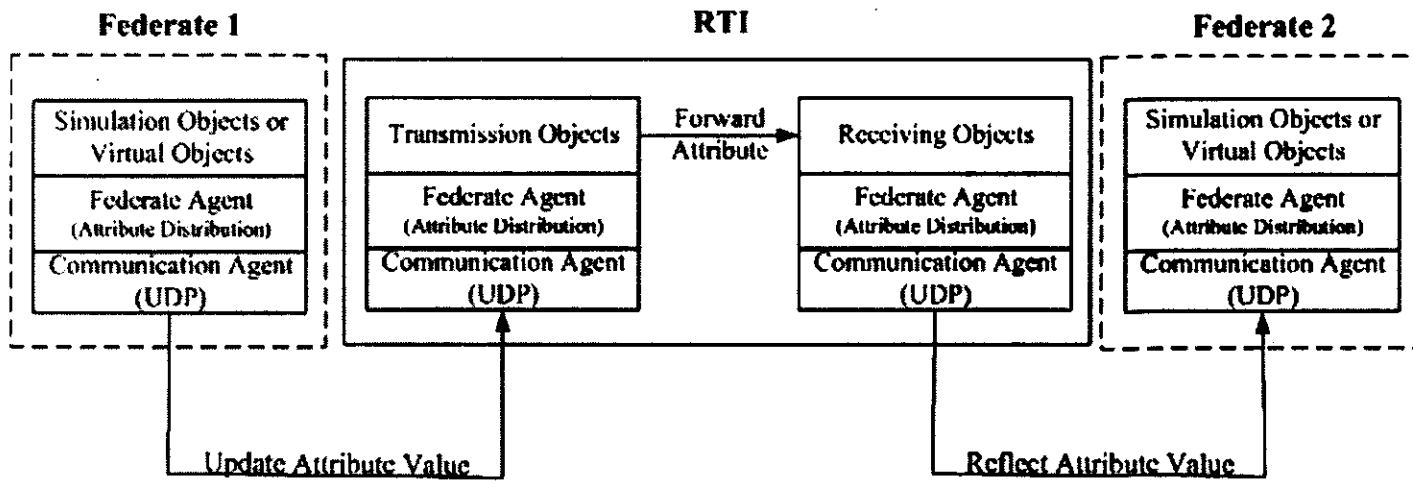
จากความจริงดังกล่าวทำให้คลาส Federate Agent สำหรับ Interaction Distribution นี้ต้องสืบทอดจาก TCP Application แทน ซึ่งทำให้ ns สามารถรับและส่งข้อมูลที่มีทั้งขนาดและข้อความพิเศษได้ ดังนั้น จึงทำให้ลักษณะการสืบทอดคลาสของ Attribute Distribution และ Interaction Distribution แตกต่างกัน สำหรับการทำงานของ Tcp Application นั้นได้แสดงไว้ในบทที่ 3 แล้ว

4.3 การออกแบบส่วน Object Management

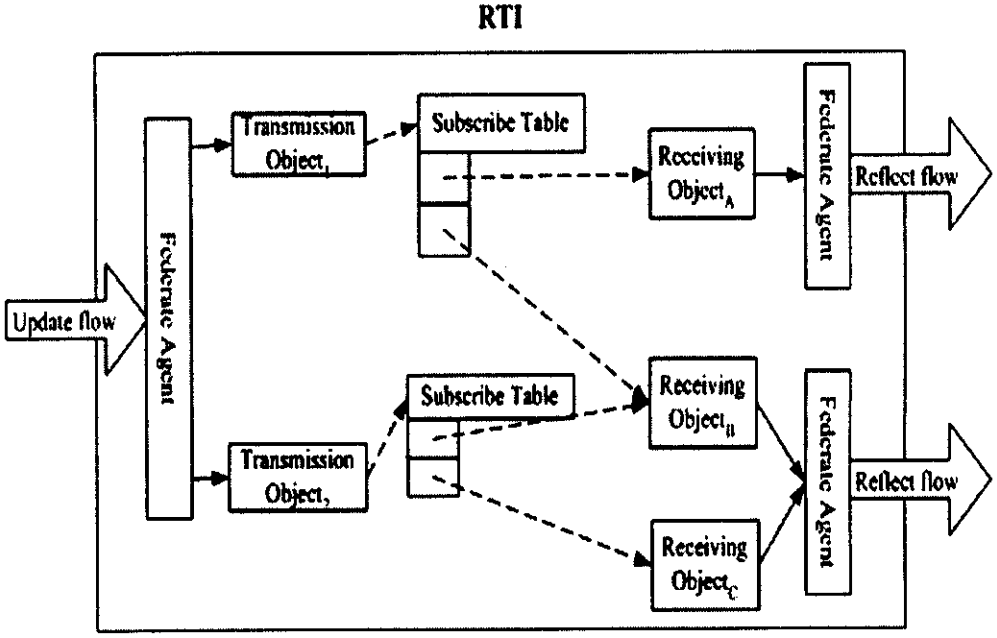
Object Management จะเป็นเรื่องที่ว่าด้วยกระบวนการกระจายลักษณะเฉพาะ (Attribute Distribution) และการกระจายเหตุการณ์ (Event Distribution หรือ Interaction Distribution) ในส่วนนี้จะเน้นตรง Attribute Distribution เป็นหลัก ทั้งนี้เพราะ Interaction Distribution นั้นยังเกี่ยวเนื่องกับส่วนของ Time Management ด้วย ซึ่งจะขอกล่าวภายหลังในหัวข้อ 4.6

กระบวนการของ Attribute Distribution ประกอบด้วยกระบวนการย่อย 3 ขั้นตอน คือ Update Attribute Value, กระบวนการส่งผ่านข้อมูลของ RTI, Reflect Attribute Value ซึ่งโครงสร้างการทำงานได้แสดงอยู่ในรูป 4.9 ซึ่งจะกล่าวในรายละเอียดได้ดังนี้

- Update Attribute Value เป็นกระบวนการที่ Simulation Object ซึ่งอยู่ในฐานะของผู้ผลิตข้อมูลทำการผลิตและส่งข้อมูลออกสู่ RTI เป็นกระบวนการที่เกิดขึ้นทางด้านซ้ายของรูป 4.9 โดยจะเริ่มที่ Simulation Object ที่ Federate 1 และสิ้นสุดที่ Transmission Objects ที่ RTI
- กระบวนการส่งผ่านข้อมูลของ RTI โดยได้ออกแบบให้ RTI สามารถส่งผ่านข้อมูลจาก Simulation Object ไปสู่ Simulation Virtual Object ที่ถูกต้องได้ เป็นกระบวนการที่เกิดขึ้นที่ส่วน RTI ของรูป 4.9 กระบวนการจะเริ่มจากเมื่อ Transmission Object ได้รับข้อมูลและจะทำการค้นหาเป้าหมายในตารางการสมัครสมาชิก (Subscribe Table) ตารางการสมัครสมาชิกเกิดจากการสมัครสมาชิก เช่น จากรูปที่ 4.10 ตารางการสมัครสมาชิกของ Transmission Object $_1$ เกิดจากการสมัครสมาชิกของ Receiving Object $_A$ และ Receiving Object $_B$ ซึ่งทำให้ เมื่อข้อมูลเดินทางมาถึง Transmission Object $_1$ ก็จะมองดูในตารางการสมัครสมาชิก และมุ่งสู่เป้าหมาย Receiving Object $_A$ และ Receiving Object $_B$ ได้
- Reflect Attribute Value คือกระบวนการที่ Receiving Object $_A$ และ Receiving Object $_B$ นำข้อมูลส่งต่อไปยัง Virtual Object ซึ่งคือ กระบวนการทางขวาของรูป 4.9



รูปที่ 4.9 แสดงลักษณะการเชื่อมต่อโมดูลเพื่อการจำลองในส่วน OM



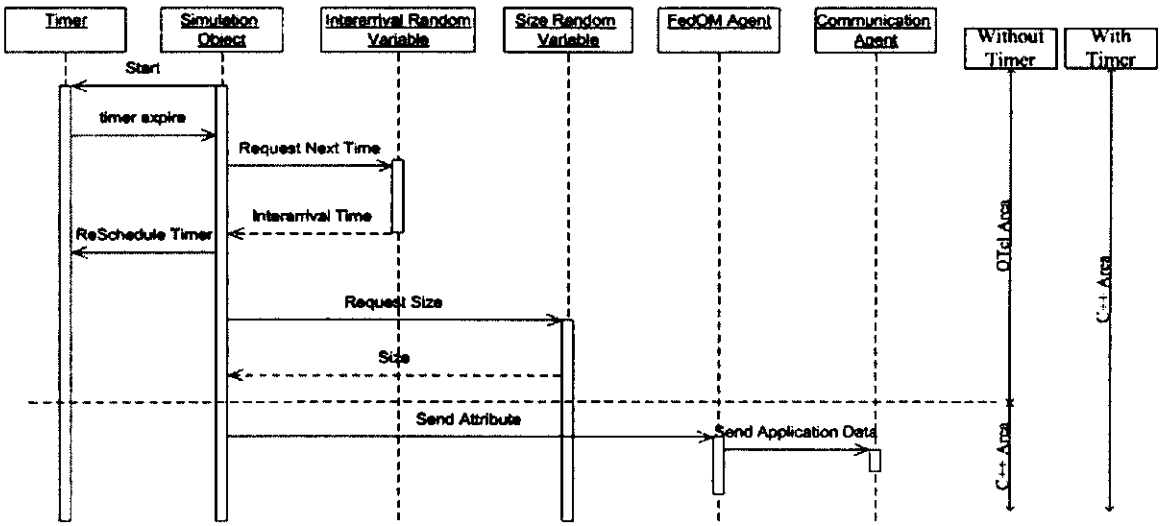
รูปที่ 4.10 แสดงกลไกการส่งผ่านข้อมูลของ RTI

4.4 การสร้างโมดูล Object Management

ในหัวข้อนี้จะกล่าวถึงคลาสต่างๆ ที่มีปฏิสัมพันธ์ต่อกันกลายเป็นโมดูลสำหรับ Object Management ในรูปที่ 4.11ได้แสดงผังการทำงานของคลาสต่างๆ ที่ฝั่งส่งข้อมูล ซึ่งเป็นที่กำเนิดของกระบวนการของการปรับปรุงข้อมูล โดยกระบวนการสามารถอธิบายเป็นลำดับดังนี้

- เริ่มต้นที่คลาส Simulation Object ซึ่งทำงานผสานกับคลาส Timer และ Interarrival Time Random Variable เพื่อใช้สำหรับกำหนดจังหวะเวลาของการส่งออกข้อมูล หรืออัตราการปรับปรุงข้อมูลให้มีรูปแบบตามที่ต้องการ ในบทที่ 5 จะได้แสดงการใช้ตัวแปรสุ่มที่ให้การแจกแจงแบบเอ็กโปเนนเชียล เพื่อทดสอบความถูกต้องของระบบ อย่างไรก็ตาม เนื่องจาก Interarrival Time Random Variable เป็นคลาสที่สืบทอดมาจากคลาส Random Variable ดังนั้น จึงสามารถกำหนดให้ Interarrival Time Random Variable นั้นเป็นชนิดใดก็ได้
- เมื่อระบบสามารถกำหนดจังหวะหรือความถี่ของการปรับปรุงข้อมูลได้แล้ว สิ่งต่อไปคือการกำหนดขนาดของข้อมูลในที่นี้ใช้การทำงานส่วนของ Size Random Variable ซึ่งสามารถกำหนดให้ขนาดข้อมูลมีการแจกแจงแบบใดก็ได้

- เมื่อระบบกำหนดความถี่และขนาดของข้อมูลได้แล้วก็จะส่งข้อมูลนั้นออกจาก Simulation Object สู่ FedOM Agent ซึ่งเป็นคลาสที่สืบทอดจากของ Federate Agent ที่ทำหน้าที่เป็นตัวแทนของ Federate



รูปที่ 4.11 แสดงผังการทำงานของ Attribute Distribution ที่ฝั่งส่ง

- และสุดท้ายคือคลาส FedOM Agent ซึ่งจะทำหน้าที่ส่งผ่านข้อมูลต่อให้กับ Communication Agent

ในรูปที่ 4.11 ทางขวาได้แสดงให้เห็นว่าเราสามารถประยุกต์ใช้โมดูลการทำงานส่วนการปรับปรุงข้อมูลโดยอาศัยคลาส Timer ร่วมด้วยหรือไม่ก็ได้ โดยการเรียกใช้งานจะแตกต่างกันออกไปดังแสดงในรูปที่ 4.12

proc sendpacket {} {
set now [$ns now]
$ns at [expr $now + [$expo0 value]] "sendpacket"
set bytes [expr round ([$expo1 value])]
$hla_obj1 update $bytes
}
$ns at 0.1 "sendpacket"
	$ns at 0.1 "$hla_obj1 start-update"

รูปที่ 4.12 แสดงการเปรียบเทียบจำนวนคำสั่งภาษา OTcl สำหรับการ Update สำหรับ Simulation Object ที่ใช้และไม่ใช้ Timer

ทางด้านซ้ายของรูป 4.12 นั้นได้แสดงวิธีเรียกใช้งาน Random Variable ผ่านทาง OTcl สคริปต์ซึ่งจะไม่มีการเรียกใช้คลาส Timer โดยตรงทำให้การกำหนดความถี่ของการปรับปรุงและขนาดของข้อมูลต้องกำหนดผ่าน OTcl สคริปต์ และทางขวาการเรียกใช้งานทั้งหมดอาศัย OTcl สคริปต์เพียงคำสั่งเดียวและอาศัยคลาส Timer ในการกำหนดความถี่และขนาดของข้อมูล

การทำงานทั้ง 2 รูปแบบสามารถทำงานได้ดี โดยการทำงานทางด้านซ้ายมือเป็นการทำงานที่ต้องอาศัยการแปลความหมายของสคริปต์โดย OTcl interpreter ซึ่งทำให้การจำลองทำงานได้ช้ากว่าแต่การทำงานทางด้านขวาจะทำได้เร็วกว่าเพราะอาศัยการทำงานของคลาส Timer ซึ่งเป็นคลาสของระบบเขียนขึ้นด้วยภาษา C++ จึงไม่ต้องผ่านการแปลคำสั่งของ OTcl interpreter

ในรูปที่ 4.13 ได้แสดงส่วนการทำงานที่ใช้เริ่มต้นคลาส Simulation Object ซึ่งจะต้องตั้งเวลาให้กับคลาส Timer ด้วยคำสั่ง Send_.resched(NextSend_) และเมื่อเวลาที่ได้ตั้งไว้หมด ก็จะเกิดเหตุการณ์ expire ซึ่งภายในมีส่วนของการส่งข้อมูลอยู่ ดังแสดงในรูปที่ 4.14 โดยการส่งข้อมูลในที่นี้คือการบอกให้คลาส Simulation Object รับข้อมูลนั่นเอง

```
SendTimer Send_;
void SimObj::start(){
        NextSend_ = ( Scheduler::instance().clock() + interarrival_->value());
        Send_.resched(NextSend_);
}
```

รูปที่ 4.13 แสดงส่วนการทำงานสำหรับการตั้งเวลา Timer

```
void SendTimer::expire(Event* e)
{
        NextSend_ = ( Scheduler::instance().clock() + interarrival_->value());
        Send_.resched(NextSend_);
        SimObj_->recv(size_->value());
}
```

รูปที่ 4.14 แสดงส่วนการทำงานเมื่อหมดเวลาและมีการส่งข้อมูลออก

ในส่วนต่อไปจะแสดงวิธีการเชื่อมต่อคลาส Simulation Object เข้ากับคลาส Random Variable โดยอาศัยส่วนการทำงานดังรูปที่ 4.15 โดยจะต้องเชื่อต่อทั้งกับ Size Random Variable และ Interarrival Time Random Variable และในรูปที่ 4.16 ได้แสดงให้เห็นถึงส่วนการแปลคำสั่ง OTcl ซึ่งส่วนนี้จะต้องเขียนอยู่ในฟังก์ชัน command ซึ่งจะทำให้ผู้ใช้สามารถกำหนดให้ Simulation Object ทำงานร่วมกับ Random Variable ด้วยคำสั่ง เช่น $SimObj attachsize $SizeRanVar สำหรับการกำหนดขนาด และ $SimObj attachinterarrivaltime $InterarrivalRanVar สำหรับการกำหนดความถี่การปรับปรุงข้อมูล

```
void SimObj::attachsize(RandomVariable* rng){ size_ = rng;}
void SimObj::attachinterarrival(RandomVariable* rng){ interarrival_ = rng;}
```

รูปที่ 4.15 แสดงส่วนการทำงานที่เกี่ยวกับการเชื่อมต่อระหว่างคลาส Simulation Object และคลาส Random Variable

```
if (strcmp(argv[1],"attachsize") == 0){
        RandomVariable*  ranvar = (RandomVariable *)TclObject::lookup(argv[2]);
        if (ranvar){
                attachsize (ranvar);
                return (TCL_OK);
        }else{
                printf("can not attach RNG\n");
                return(TCL_ERROR);
        }
}
```

รูปที่ 4.16 แสดงส่วนการทำงานสำหรับการแปลคำสั่ง OTcl ของคำสั่ง attachsize

สำหรับการเชื่อมต่อคลาส Simulation Object เข้ากับ Federate Agent ซึ่งในที่นี้อยู่ในรูปของคลาส FedOM ก็ทำได้ในทำนองเดียวกันโดยอาศัยส่วนคำสั่งดังรูปที่ 4.17 ซึ่งทำให้แต่ละ Federate สามารถมีได้หลาย Simulation Object สำหรับคำสั่งที่ใช้สำหรับการเชื่อมต่อคือ attachsimobj ซึ่ง

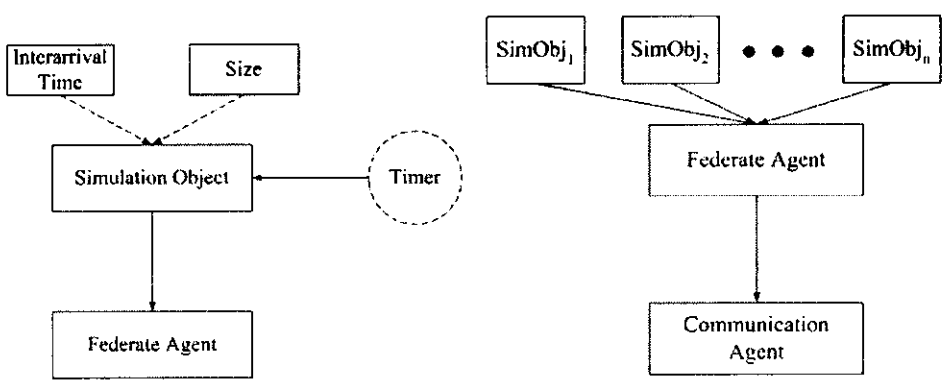
สามารถใช้งานได้ เช่น $fed1 attachsimobj $simobj1 ก็จะทำให้ Simulation Object สามารถส่งข้อมูลให้กับ Federate Agent ได้ โดยส่วนการแปลคำสั่งแสดงดังรูปที่ 4.18

```
void FedOM::attachobj(SimObj* obj){
        objindex_++;
        tableobj_[objindex_] = (SimObj*)obj;
}
```

รูปที่ 4.17 แสดงส่วนการทำงานที่เกี่ยวกับการเชื่อมต่อระหว่างคลาส FedOM และคลาส Simulation Object

```
if (strcmp(argv[1],"attachsimobj") == 0){
        SimObj*  simobj = (SimObj *)TclObject::lookup(argv[2]);
        if (simobj) {
                attachobj(simobj);
                return(TCL_OK);
        }else {
                printf("attach obj error\n");
                return(TCL_ERROR);
        }
}
```

รูปที่ 4.18 แสดงส่วนการทำงานสำหรับการแปลคำสั่ง OTcl ของคำสั่ง attachsimobj



รูปที่4.19 แสดงลักษณะการเชื่อมต่อของ Simulation Object, Federate Agent และ Communication Agent

ดังนั้นโดยสรุปสำหรับฝั่งการส่งข้อมูลนั้นจะมีลักษณะดังรูปที่ 4.19 คือ การทำงานของ Simulation Object นั้นจะต้องทำงานร่วมกับ Random Variable โดยอาศัยคลาส Timer เป็นโครงสร้างเสริมสำหรับการทำงานที่เร็วขึ้น และทางขวานั้นแสดงการเชื่อมต่อ Simulation Object เข้ากับ Federate Agent ซึ่งแต่ละ Federate สามารถมีได้หลาย Simulation Object

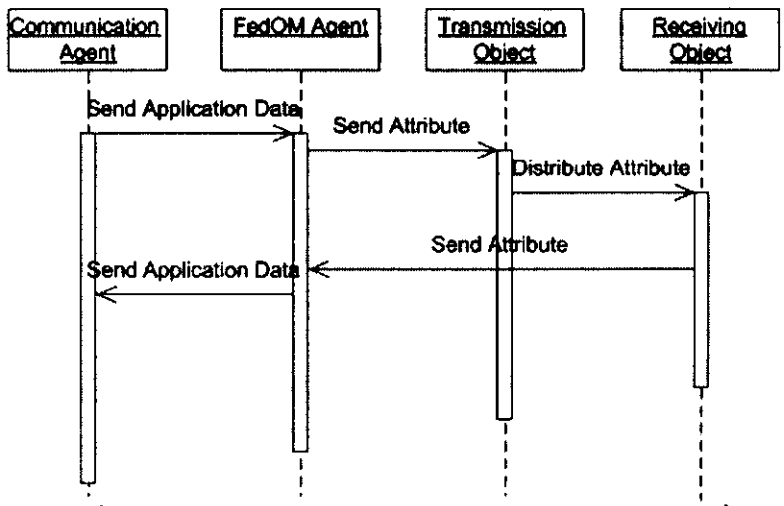
เมื่อเกิดการเชื่อมต่อกันได้สำเร็จการส่งผ่านข้อมูลระหว่างคลาสก็สามารถทำได้ โดยในรูปที่ 4.20 ได้แสดงโครงร่างการรับและส่งผ่านข้อมูลของคลาส Simulation Object ซึ่งในฟังก์ชันเหล่านี้สามารถแทรกชุดคำสั่งสำหรับการบันทึกค่าการทดลองลงไปได้ เช่น ถ้าหากต้องการทราบจำนวนของ ข้อมูลทั้งหมดที่ถูกส่งโดย Federate Agent ก็ต้องแทรกไว้ในส่วนฟังก์ชัน FedOM::send และสำหรับฟังก์ชันอื่นๆ ก็เป็นไปในลักษณะเดียวกัน ส่วนการส่งผ่านข้อมูลระหว่างคลาสนี้เป็นส่วนฟังก์ชันมาตรฐานที่กำหนดไว้ในคลาส Application ซึ่งคลาสลูกจะต้องนำมาเขียนใหม่ให้ได้ลักษณะการทำงานตามต้องการ

```
void SimObj::send(int size){
        //trace number of attribute send
        fed_->recv(size);
}
void SimObj::recv(int size){
        //trace number of attribute produced
        this->send(size);
}
void FedOM::send(int size){
        //trace number of ADU send
        agent_->send(size);
}
void FedOM::recv(int size){
        //trace number of ADU produced
        this->send(send);
}
```

รูปที่ 4.20 แสดงส่วนการทำงานของการส่งผ่านข้อมูลของ Simulation Object และ Federate Agent

ต่อไปจะอธิบายถึงส่วนการทำงานของ Object Management ที่ RTI ซึ่งจะต้องทำการส่งผ่านข้อมูลจากผู้ปรับปรุงข้อมูลสู่ผู้สมัครสมาชิกดังได้อธิบายโครงร่างการทำงานในรูปที่ 4.10 แต่ในส่วนนี้จะแสดงให้เห็นรายละเอียดการทำงานภายในมากขึ้น โดยแสดงผังการทำงานในรูปที่ 4.12 ได้ดังนี้

- เมื่อคลาส Communication Agent ได้รับข้อมูลสมบูรณ์ก็จะส่งขึ้นสู่คลาส FedOM แล้วส่งต่อให้ Transmission Object
- ที่ Transmissin Object จะต้องมีตารางการสมัครสมาชิกซึ่งจะอธิบายต่อไปถึงคลาสที่เกี่ยวข้องกับตารางสมัครสมาชิกและวิธีการสมัครสมาชิก โดย Transmission Object จะตรวจสอบในตารางดังกล่าวหาก Receiving Object ใดสมัครสมาชิกก็จะสะท้อนข้อมูลให้
- เมื่อ Receiving Object ที่สมัครสมาชิกได้รับการสะท้อนข้อมูลก็จะส่งต่อให้คลาส FedOM ซึ่งจะนำส่งข้อมูลสู่ปลายทาง



รูปที่ 4.21 แสดงผังการทำงานของ Attribute Distribution ที่ RTI

ดังนั้นในคลาส Transmission Object จึงต้องมีตารางสมาชิกเป็นส่วนประกอบ โดยตารางดังกล่าวในวิทยานิพนธ์นี้ได้สร้างไว้เป็นคลาสหนึ่ง ซึ่งสามารถเพิ่มสมาชิกได้ด้วยตัวเอง ดังแสดงการประกาศไว้ในรูปที่ 4.22 ซึ่งคลาส Transmission Object นั้นเป็นคลาสลูกของ Simulation Object โดยได้เพิ่มคลาส Sub ไว้ภายใน

สำหรับคลาส Sub ได้แสดงไว้ในรูปที่ 4.23 ซึ่งมีฟังก์ชันที่สำคัญ 2 ฟังก์ชันคือ

- void AddFwdTable(SimObj *) สำหรับการเพิ่มสมาชิกเข้าสู่ตารางการสมัครสมาชิก โดยฟังก์ชันนี้จะทำงานเมื่อมีคำสั่ง subscribe-by โดยการแปลคำสั่งดังปรากฏในรูปที่ 4.25 โดยลักษณะการใช้คำสั่ง เช่น $simobj1 subscribe-by $simobj2 หรือ อาจใช้ในลักษณะ $simobj2 subscribe $simobj1 ก็ได้

- void Fwd(int) ฟังก์ชันนี้สร้างไว้เพื่อส่งผ่านข้อมูลจาก Transmission Object ไปสู่ Receiving Object ดังรายละเอียดการทำงานดังแสดงในรูปที่ 4.26 กล่าวคือเมื่อมีข้อมูลที่ผ่านมาจากกระบวนการปรับปรุงข้อมูลก็จะทำการตรวจสอบตารางทั้งหมด และทำการส่งให้กับผู้สมัครสมาชิก

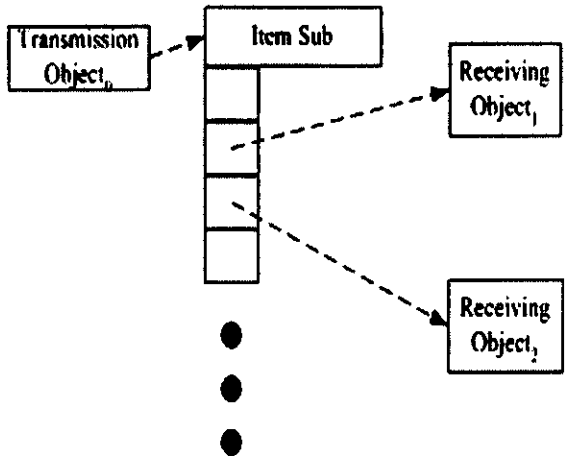
```
class TransObj:public SimObj{
public:
        Sub* sub_;
};
```

รูปที่ 4.22 แสดงการขยายความสามารถของ Simulation Object เพื่อทำงานเป็น Transmission Object

```
class Sub {
private:
        SimObj * ItemSub[Conf::iMaxFed];
public:
        Sub();
        void AddFwdTable(SimObj *);
        void Fwd(int);
};
```

รูปที่ 4.23 แสดงการประกาศคลาส Subscribe

โดยสรุป ตารางการสมัครสมาชิกสามารถแสดงได้ด้วยรูปที่ 4.24 เมื่อเกิดการใช้คำสั่ง subscribe ก็จะสั่งให้ตารางชี้ไปยังผู้สมัคร และเมื่อมีข้อมูลที่ต้องการสะท้อนก็จะถูกส่งผ่านไปยัง Receiving Object ได้



รูปที่ 4.24 แสดงโครงสร้างการสมัครสมาชิกและนำไปใช้สำหรับการ Update

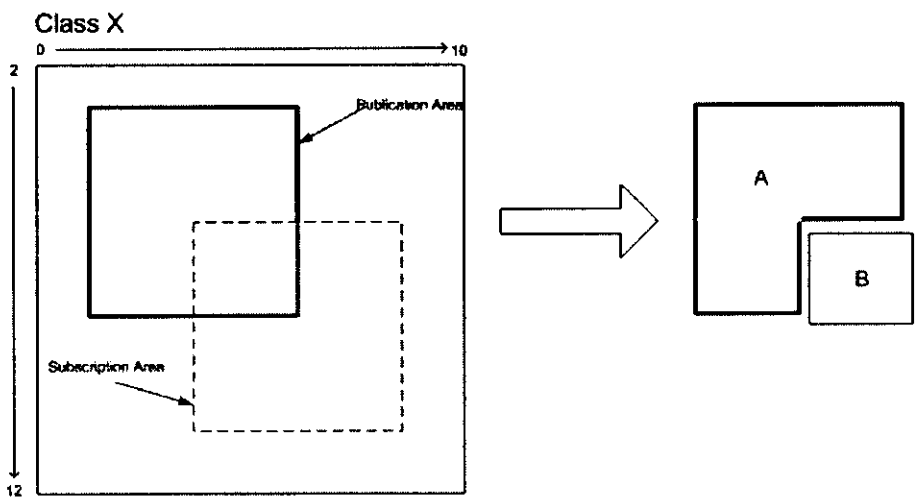
```
if (strcmp(argv[1],"subscribe-by") == 0){
        SimObj*  simobj = (SimObj *)TclObject::lookup(argv[2]);
        if (simobj) {
                sub_->AddFwdTable(simobj);
                return(TCL_OK);
        }else {
                printf("subscribe error\n");
                return(TCL_ERROR);
        }
```

รูปที่ 4.25 แสดงส่วนการทำงานสำหรับแปลคำสั่ง OTcl ของคำสั่ง subscribe-by

```
void Sub::Fwd(int size)
{
int i=0;
        for(i=0;i<Conf::iMaxFed;i++){
                if(ItemSub[i] != NULL)
                {
                        ItemSub[i]->send(size);
                }
        }
}
```

รูปที่ 4.26 แสดงการสะท้อนข้อมูลไปยังสมาชิก

4.5 การออกแบบส่วน Data Distribution Management



รูปที่ 4.27 แสดงหลักการการแยกแยะกระแสข้อมูล

ในส่วนของ Data Distribution Management นั้นมีความซับซ้อนกว่าส่วนของ Object Management เนื่องจากต้องมีการพิจารณาค่าข้อมูลเชิงค่า โดยเฉพาะอย่างยิ่งส่วนของ RTI ซึ่งตาม

หลักการ Data Distribution Management จำเป็นต้องอาศัยการกรองค่า เพื่อทำให้การส่งผ่านข้อมูลเชิงค่าประสบความสำเร็จ แต่เราก็สามารถประยุกต์โครงสร้างของ RTI ของ Object Management ดังแสดงในรูป 4.27 เพื่อแทนการทำงานของ Data Distribution Management ได้ ด้วยวิธีการแยกแยะกระแสของข้อมูลดังจะแสดงแนวความคิดโดยละเอียดในรูป 4.12 ซึ่งมีรายละเอียดดังนี้

กำหนดให้

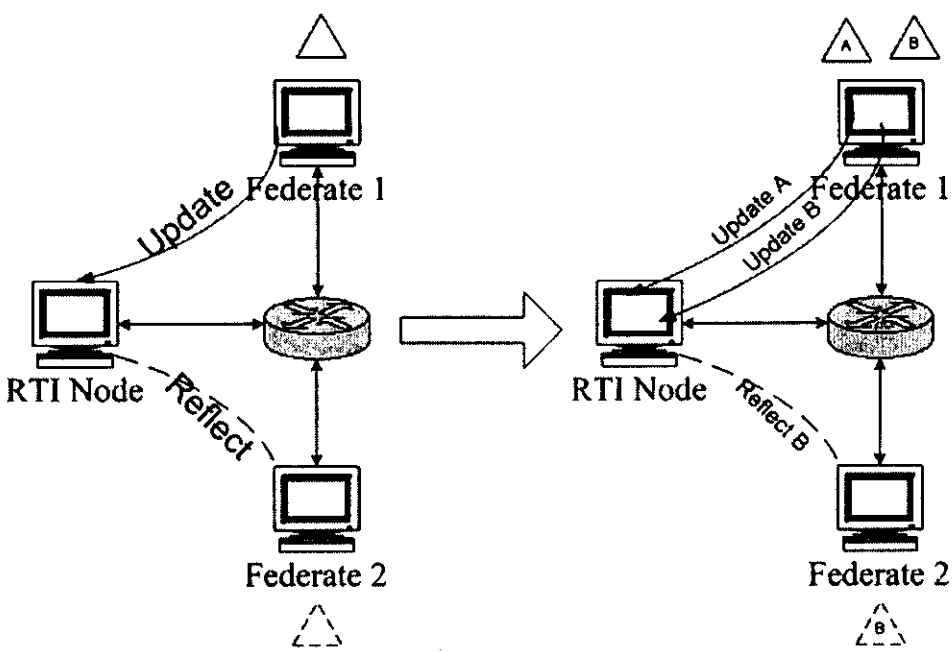
P แทน Publication Area

S แทน Subscription Area

ดังนั้น จะได้

$$B = P \cap S \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (1)$$

$$A = P - B \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (2)$$



รูปที่ 4.28 แสดงหลักการแยกแยะกระแสข้อมูลขณะใช้งานจริง

การแยกแยะกระแสข้อมูลแสดงได้อย่างเป็นรูปธรรมดังรูป 4.28 ทางซ้ายมือคือกระแสข้อมูลที่เกิดขึ้นจริงในระบบ โดยกระแสการปรับปรุงข้อมูลนั้นผ่านการคัดกรองเชิงค่าโดย RTI แล้ว สำหรับการประยุกต์ใช้โครงสร้างของ Object Management มาใช้สามารถทำได้โดยการแยกแยะกระแสข้อมูล โดยแนวคิดแสดงในรูปขวากล่าวคือจะสมมติให้มีอ็อบเจกต์ 2 ชนิด คือ อ็อบเจกต์ A และอ็อบเจกต์ B อยู่ที่ Federate 1 และกำหนดให้ มีอ็อบเจกต์เสมือน B ที่ Federate 2 โดยอ็อบเจกต์

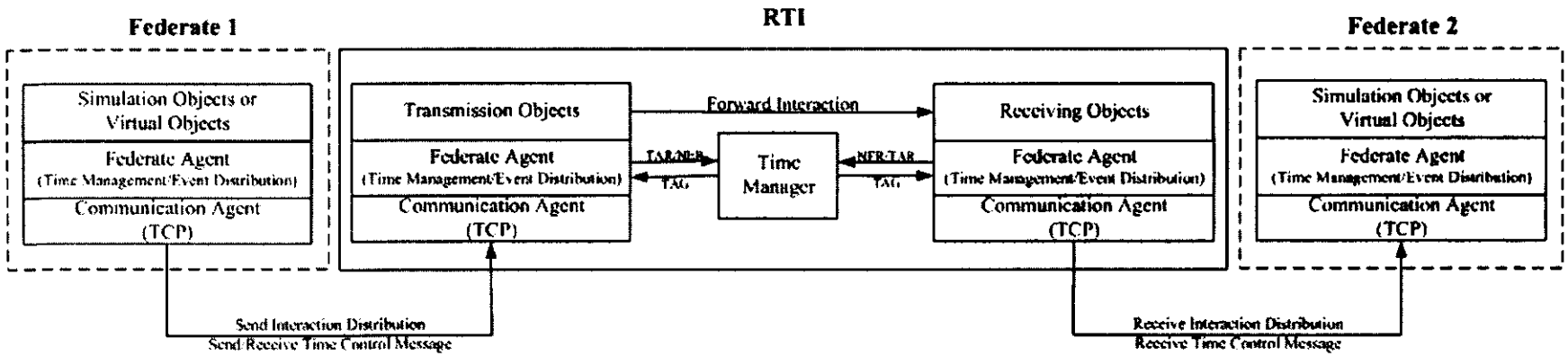
เสมือน B นี้เกิดจากการสมัครสมาชิกต่ออ็อบเจกต์ B นั่นเอง และอ็อบเจกต์ A จะส่งผ่านข้อมูลเชิงคลาสคือ Update A ซึ่งก็คือ สมการ (2) โดยอาจมองได้ว่าเป็นการส่งผ่านข้อมูลเชิงคลาสที่ไม่มีการสมัครสมาชิกเลย ทำให้กระแส Update A ไม่มีการสะท้อนข้อมูลออกหรือไม่มี Reflect A แต่ขณะเดียวกันอ็อบเจกต์ B จะส่งผ่านข้อมูลเชิงคลาส ซึ่งสามารถมองเป็นกระแสของ Update B ซึ่งคือการส่งผ่านข้อมูลเชิงคลาสที่มีการสมัครสมาชิกโดยอ็อบเจกต์เสมือน B ทำให้มีการสะท้อนข้อมูลออก ซึ่งที่นี้คือ Reflect B ซึ่งเป็นไปตามสมการ (1)

4.6 การออกแบบส่วน Time Management

ในหัวข้อนี้จะกล่าวถึงการออกแบบคลาสเพื่อทำงานในส่วนการจัดการเวลา หรือ Time Management ซึ่งเป็นบริการหนึ่งซึ่ง HLA กำหนดให้ระบบการจำลองต้องรองรับ ทั้งนี้เพื่อให้ Federate ทุกตัวทำงานประสานกันได้ ทั้งนี้โดยอาศัย RTI เป็นผู้ควบคุมการเคลื่อนที่ของเวลาของระบบทั้งหมด

กระบวนการทำงานของ Time Management ทั้งหมดประกอบด้วย 3 ส่วน สำหรับรายละเอียดกระบวนการ วิธีการ ศึกษาได้จากบทที่ 2 ในส่วนนี้จะไม่กล่าวอย่างละเอียดเพียงแต่จะบ่งบอกว่ากระบวนการจะเกิดตรงส่วนประกอบใดของระบบ ซึ่งหากพิจารณาโดยอาศัยรูป 4.29 ประกอบจะอธิบายได้ดังนี้

1. การส่ง และ รับข้อความร้องขอการเลื่อนเวลา คือ TAR(t) โดยกระบวนการจะเริ่มจากการที่ Federate1 ต้องการเคลื่อนเวลาไปสู่เวลา t ดังนั้นจึงส่งข้อความ TAR(t) เมื่อ TAR(t) มาถึงผู้รับข้อความคือ Transmission Object ที่ RTI ก็จะส่งต่อให้ Time Manager ต่อไป ส่วนข้อความ NER ก็จะพิจารณาได้ในทำนองเดียวกัน
2. การคำนวณ LBTS จะทำโดย Time Manager ซึ่งเป็นคลาสหนึ่งทำงานอยู่ภายใน RTI ซึ่งต้องคอยทำการคำนวณ LBTS ทุกครั้งที่มีการร้องขอการเคลื่อนเวลาไม่ว่าจะอยู่ในรูปของข้อความ TAR หรือ NER ก็ตาม
3. การส่ง และ รับข้อความอนุญาตการเคลื่อนเวลา คือ TAG(t) จะถูกส่งโดย Time Manager สู่ Federate Agent ผู้ที่เคยส่ง TAG(t) ออกมา และ ผ่านการคำนวณ LBTS แล้ว และหาก Federate Agent ใดผ่านการพิจารณา Time Manager ยังได้ส่งสัญญาณให้กับ Transmission Object เพื่อส่งเหตุการณ์หรือข้อความที่มีเวลากำกับ และค่าเวลากำกับนั้น น้อยกว่าหรือเท่ากับ



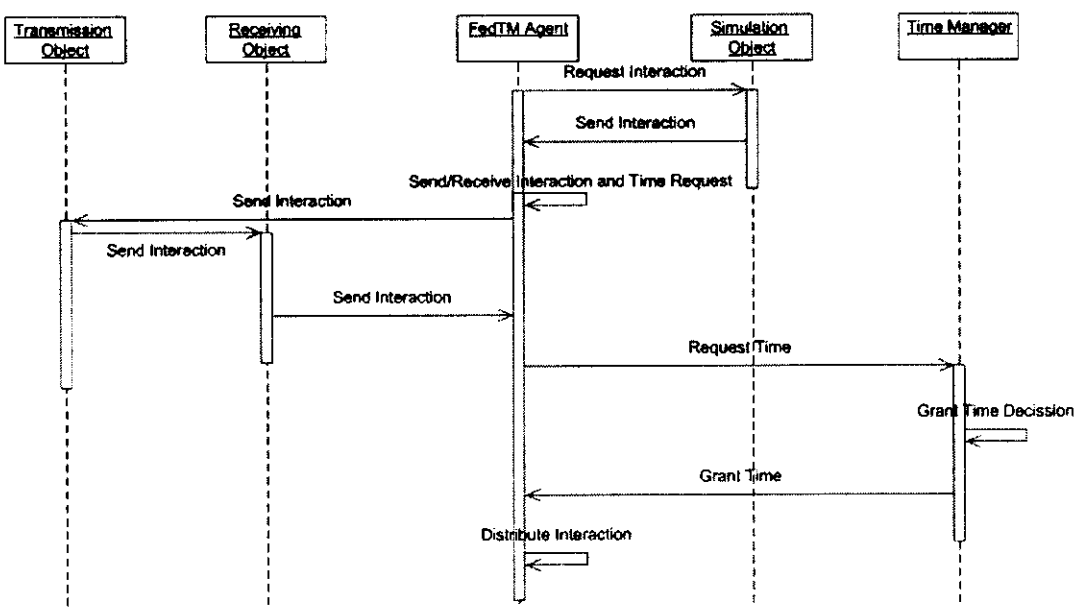
รูปที่ 4.29 แสดงการเชื่อมต่อโมดูลเพื่อการจำลองในส่วน TM

t ออกไปด้วย กระบวนการนี้เป็นสิ่งที่จะรับประกันได้ว่า จะไม่ทำให้ Federate ได้รับเหตุการณ์หรือข้อความที่มีเวลากำกับ ที่มีความเสี่ยงต่อการประมวลผล

การทำงานที่เกี่ยวข้องกับเหตุการณ์หรือข้อความที่มีเวลากำกับ จะใช้โปรโตคอลเป็น TCP ทั้งนี้เพราะต้องการรับประกันความถูกต้องของเหตุการณ์ เพื่อมิให้เกิดความผิดพลาดของการจำลองอันเกิดจากเหตุการณ์บางเหตุการณ์ในระบบสูญหาย

4.7 การสร้างโมดูล Time Management

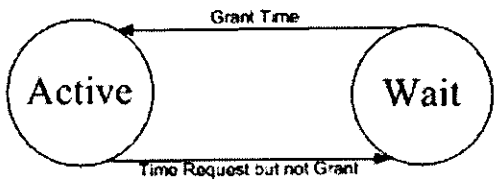
โมดูลที่เกี่ยวข้องกับ Time Management ประกอบด้วย 5 ส่วนหลัก คือ Simulation Object, Federate Agent, Transmission Object, Receiving Object และ Time Manager โดยสำหรับ Simulation Object, Transmission Object และ Receiving Object ที่ใช้ในโมดูล Time Management นั้นคือคลาสเดียวกันกับที่ใช้ใน Object Management โดยมีลักษณะการทำงานคล้ายกัน ทั้งในส่วนการเชื่อมต่อกับ Federate Agent และการส่งข้อมูล ดังนั้นจึงจะกล่าวถึงในลักษณะเด่นที่เกี่ยวข้องกับ Time Management เท่านั้น สำหรับในหัวข้อนี้จะกล่าวในรายละเอียดของการทำงานของโมดูล FedTM และ Time Manager โดยจะเริ่มต้นจากภาพรวมของระบบโดยผ่านรูปที่ 4.30 ซึ่งจะแสดงปฏิสัมพันธ์ระหว่างคลาสต่างๆในระบบ จากนั้นจะกล่าวถึงส่วนที่เป็นหัวใจหลักของ Time Management นั่นคือ Time Manger ซึ่งจะเป็นผู้ทำหน้าที่ในการประสานเวลา และสุดท้ายจะเป็นเรื่องคลาส FedTM



รูปที่ 4.30 แสดงผังการทำงานของคลาสที่ใช้สำหรับ Time Management

ดังแสดงในรูป 4.30 ซึ่งเป็นผังการทำงานของระบบทั้งหมดที่เกี่ยวข้องกับ Time Management ซึ่งจะใช้อธิบายภายรวมของระบบได้ โดยจะอธิบายเป็นลำดับได้ดังนี้ คือ

- กระบวนการส่งข้อมูลของ Time Management จะเกี่ยวข้องกับข้อความหลายชนิด ซึ่งประกอบด้วยทั้งข้อมูลของเหตุการณ์และข้อมูลของข้อความร้องขอการเคลื่อนเวลา โดยการส่งเหตุการณ์นั้นจะไม่สามารถส่งได้อย่างอิสระเหมือนดังใน Object Management โดยจะถูกจำกัดด้วยเงื่อนไขของเวลาด้วย ดังนั้นจึงได้ออกแบบให้คลาส FedTM เป็นผู้ร้องของเหตุการณ์จากคลาส Simulation Object เพื่อให้ระบบสามารถควบคุมการส่งข้อมูลเหตุการณ์ได้ ดังแสดงในผังการทำงานที่เริ่มต้นด้วย Request Interaction และ Simulation Object ตอบกลับมาด้วยการส่ง Interaction มาให้
- ต่อมา FedTM ก็จะจัดส่งข้อมูลทั้งหมด โดยอาศัยการจัดส่งของคลาส TcpApp ซึ่งทำให้สามารถส่งข้อมูลจริงๆ ได้ ดังที่เราได้ทราบมาแล้วเกี่ยวกับการส่งข้อมูลใน Object Management นั้น กลุ่มข้อมูลมีเพียงแต่ขนาด แต่ไม่ได้นำข้อมูลนั้นมาแปลความหมายใดๆ (เพราะสืบทอดมาจากคลาส Application) แตกต่างกับ Time Management ที่จะต้องมีการแยกกลุ่มข้อมูลระหว่างเหตุการณ์และข้อความร้องขอการเคลื่อนเวลา หากไม่ส่งข้อมูลจริงๆ ก็ไม่สามารถบ่งบอกได้ว่ากลุ่มขนาดข้อมูลที่มาถึง RTI นั้นมีความแตกต่างกันอย่างไร คือเหตุการณ์หรือข้อความร้องขอการเคลื่อนเวลา
- เมื่อ FedTM ที่ RTI ได้รับข้อมูลก็จะทำการแยกแยะความแตกต่าง หากเป็นเหตุการณ์ ก็จะจัดส่งให้ Transmission Object แต่หากเป็นข้อความร้องขอการเคลื่อนเวลาก็จะส่งข้อความนั้นให้ Time Manager เพื่อใช้ในการประสานเวลา
- สำหรับ Transmission Object นั้น เมื่อได้รับเหตุการณ์ก็จะทำการสำรวจในตารางสมาชิกว่ามี Receiving Object ใดบ้างที่สมัครเป็นสมาชิกและทำการจัดส่งเหตุการณ์นั้นให้ และสำหรับข้อความร้องขอการเคลื่อนเวลานั้น FedTM จะส่งผ่านให้ Time Manager สำหรับคำนวณ LBTS หาก Federate ใดได้รับการอนุมัติให้เคลื่อนเวลาก็จะได้รับข้อความ TAG แต่ถ้าไม่ผ่านก็จะเข้าสู่สถานะรอคอย ดังแสดงในรูปที่ 4.31

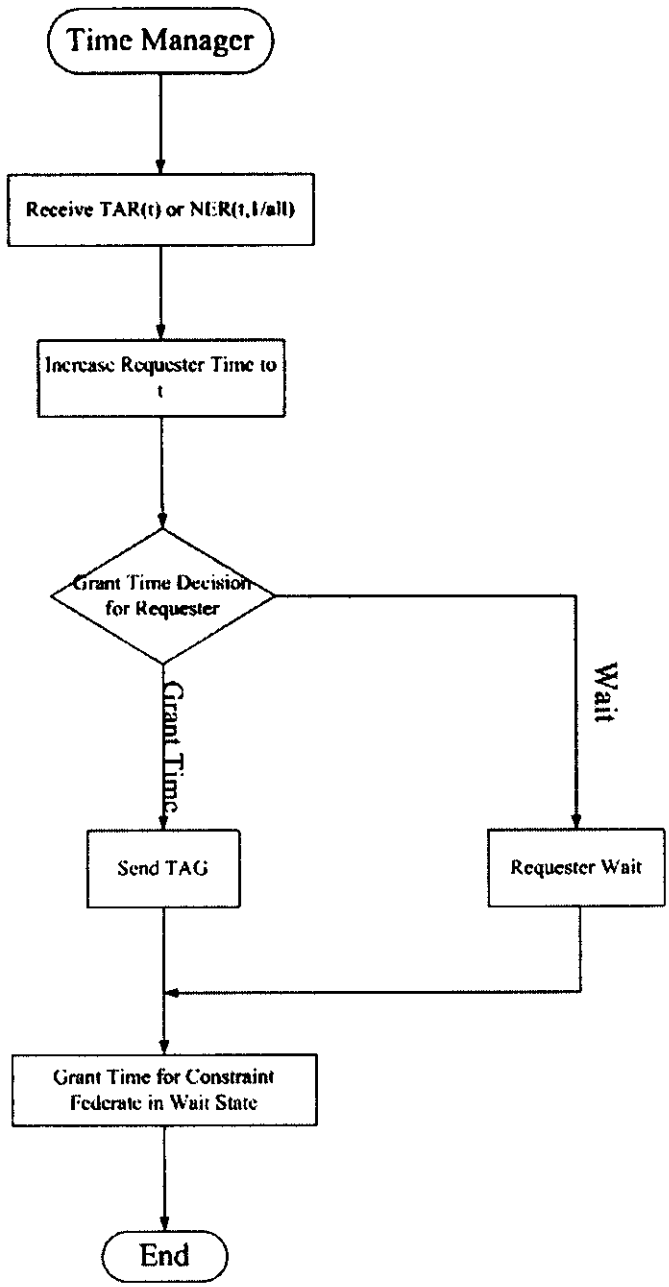


รูปที่ 4.31 แสดงสถานะที่เป็นไปได้ของ Federate

- เมื่อได้รับ TAG FedTM ก็จะทำการส่งเหตุการณ์พร้อมกับข้อความ TAG ให้กับผู้ได้รับการอนุมัติรับทราบ

ทั้งหมดที่ได้อธิบายตามผังการทำงานไปแล้วนั้น เป็นภาพรวมของการทำงานของระบบทั้งหมด อีกหลายส่วนจะได้อธิบายเพิ่มเติมในส่วนต่อไปของหัวข้อนี้ โดยต่อไปจะกล่าวถึงการทำงานของคลาส Time Manager ดังแสดงด้วยแผนภูมิสายงาน (Flow Chart) ดังรูปที่ 4.32 โดยในเบื้องต้นจะต้องมีโครงสร้างข้อมูลที่เป็นตารางบันทึกว่า Federate หนึ่งๆนั้นเป็น Regulating Federate ของ Federate ใดบ้าง รวมทั้งเป็น Constrained Federate ของ Federate ใดบ้าง โครงสร้างนี้จะใช้สำหรับการคำนวณ LBTS โดยจะอธิบายแทรกอยู่ในขั้นตอนการทำงาน ดังนี้

- โดยเริ่มต้นที่ Time Manager จะต้องทำการรับข้อความร้องขอการเคลื่อนเวลาไม่ว่าจะเป็น TAR(t) หรือ NER(t,1/all) และทำการแยกเอาเฉพาะพารามิเตอร์ t ไปใช้
- นำพารามิเตอร์ t ไปปรับปรุงเวลาตรรกะของ Federate ผู้ขอ
- ทำการคำนวณ LBTS โดยใช้ข้อมูลเข้าเป็นเวลาตรรกะของ Regulating Federate ทั้งหมด (ตรวจสอบได้จากตาราง Regulating Federate) ถ้าได้รับการอนุมัติจะทำส่วน TAG แต่หากไม่อนุมัติจะเข้าสู่สถานะรอคอย
- หลังจากนั้น Time Manager จะมองว่า Federate ผู้ขอการเคลื่อนเวลานี้เป็น Regulating Federate ของ Federate ที่อยู่ในสถานะรอคอยใดบ้าง (Federate ที่ต้องเข้าสู่สถานะรอคอยเพราะไม่ผ่านการคำนวณ LBTS เพราะเวลาตรรกะของ Federate ผู้ขอ) ก็จะต้องนำ Federate เหล่านั้นมาคำนวณ LBTS ใหม่ ซึ่งอาจจะได้รับอนุมัติหรือไม่ก็ได้ หากอนุมัติก็จะได้รับ TAG แต่หากไม่อนุมัติก็จะอยู่ในสถานะรอคอยต่อไป



รูปที่ 4.32 แสดงแผนภูมิสายงานของ Time Manager

ต่อไปจะเข้าสู่รายละเอียดของการคำนวณ LBTS ซึ่งจะเป็นขั้นตอนวิธีอันสำคัญ เป็นหัวใจของการประสานเวลาแบบอนุรักษ์นิยม ดังแสดงฟังก์ชันการทำงานในรูปที่ 4.33 โดยสาระสำคัญของการคำนวณ LBTS นั้นก็คือการหาค่าน้อยที่สุดในกลุ่มของจำนวนหนึ่งๆ โดยกลุ่มหรือเซ็ตของจำนวนเหล่านั้นคือค่าเวลาตรรกะบวกด้วยค่า Lookahead ของ Regulating Federate ทั้งหลาย โดยในบรรทัดที่ 7 เป็นการตรวจสอบว่า Federate ใดบ้างที่เป็น Regulating Federate และในบรรทัดที่ 8-12

เป็นการหาค่าน้อยสุดซึ่งใช้ตัวแปร min เป็นตัวบันทึกค่าน้อยสุด ค่าที่ฟังชัน LBTS คืนกลับมาให้นี้คือค่าน้อยสุดในบรรดา Regulating Federate ทั้งหมด

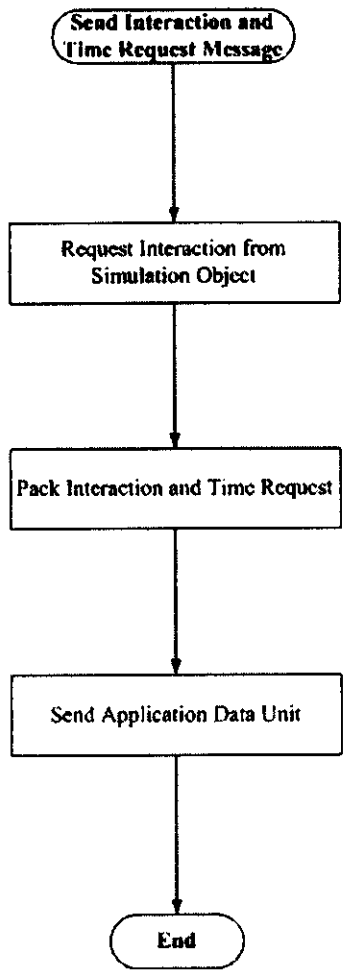
```
1. int LbtsArray::LbtsCal(FedAgent* FedAgent){
2. int min;
3. int i;
4.        min = Conf::iMaxLgt;
5.        for(i=0;i<Conf::iMaxFed;i++)
6.        {
7.                if( GetCon(FedAgent->iGetFedAgentId(),i ) == 1){
8.                        if( ((Rti_->GetFedAgent(i))->iGetReqTime() +
9.                            (Rti_->GetFedAgent(i))->iGetLookAhead()) < min){
10.                               min =  (Rti_->GetFedAgent(i))->iGetReqTime() +
11.                               (Rti_->GetFedAgent(i))->iGetLookAhead();
12.                       }
13.               }
14.       }//end for
15.       return min;
16. }
```

รูปที่ 4.33 แสดงส่วนการทำงานที่ใช้คำนวณ LBTS

ในลำดับต่อไปจะกล่าวถึงคลาส FedTM ซึ่งรับผิดชอบในการส่ง รับ รวมกลุ่มและแยกกลุ่มข้อมูล โดยขั้นตอนของการส่งเหตุการณ์และข้อความร้องขอการเคลื่อนเวลาดังแสดงในรูปที่ 4.34 ซึ่งอธิบายได้เป็นขั้นตอนดังนี้

- โดย FedTM จะเป็นผู้ร้องขอต่อ Simulation Object ก่อนจะได้มาซึ่งเหตุการณ์ ความจำเป็นในเรื่องนี้ก็เพราะว่า FedTM ไม่อาจยอมให้ Simulation Object ส่งเหตุการณ์ได้อย่างเป็นอิสระเนื่องจากมีเงื่อนไขด้านเวลาเข้ามาเกี่ยวข้องด้วย

- เมื่อได้เหตุการณ์มาแล้วก็จะทำการจัดส่งทั้งนี้จะอาศัยการขนส่งข้อมูลจริงผ่านทางคลาส TcpApp



รูปที่ 4.34 แสดงลำดับการทำงานของการส่งเหตุการณ์และข้อความร้องขอการเคลื่อนเวลา

และในส่วนการรับข้อมูลแสดงในรูปที่ 4.35 โดยการรับข้อมูลนี้เป็นความรับผิดชอบของ Federate Agent ที่อยู่ที RTI เป็นหลัก โดยมีขั้นตอนดังต่อไปนี้

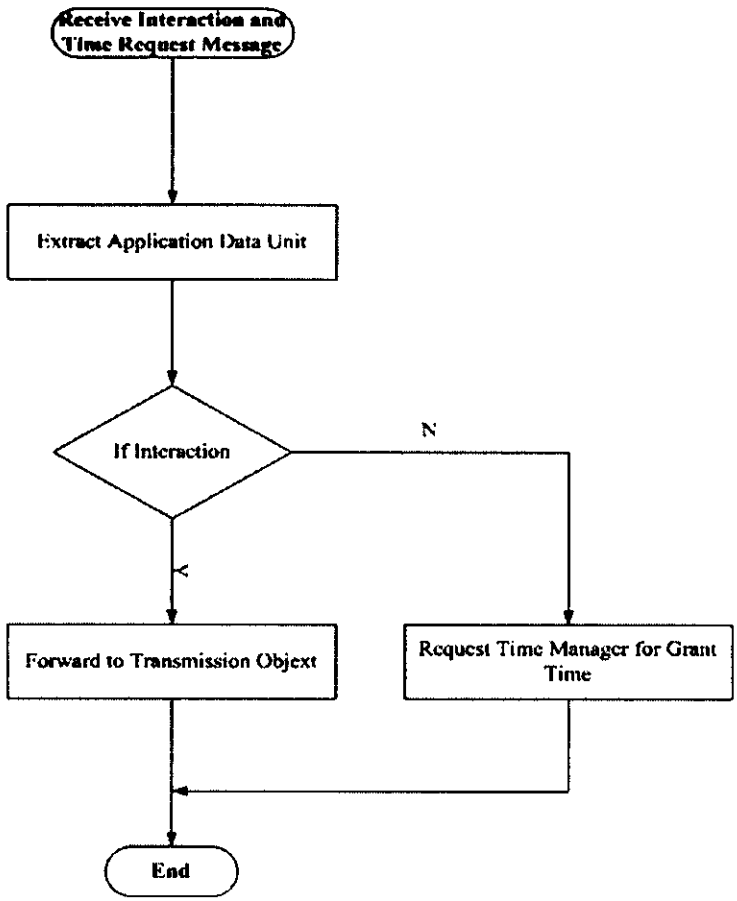
- เมื่อรับกลุ่มข้อมูลเข้ามาจะทำการแยกเหตุการณ์และข้อความร้องขอการเคลื่อนเวลาชนิดต่างๆ ออกจากกัน การแยกแยะกลุ่มข้อมูลนี้ทำในฟังก์ชัน virtual void process_data(int*,AppData*) ซึ่งประกาศไว้ในคลาส TcpApp การที่ฟังก์ชันนี้ถูกประกาศเป็น virtual ฟังก์ชัน ทำให้คลาสลูกสามารถเขียนโปรแกรมใหม่ได้ ให้ตรงกับความต้องการ สำหรับข้อมูลที่รับเข้ามานั้นจะอยู่ในรูปของ AppData* ซึ่งจะต้องนำมาแยกอีกครั้งหนึ่ง

- ถ้าเป็นเหตุการณ์ก็จะส่งต่อให้ Transmission Object และจะทำการส่งต่อให้กับ Receiving Object ผู้ซึ่งทำการสมัครสมาชิกแล้ว ทั้งนี้คลาส Transmission Object จะใช้คลาส Sub เพื่อใช้เป็นตารางสมัครสมาชิก
- แต่ถ้าเป็นข้อความการร้องขอการเคลื่อนเวลาก็จะทำการประสานกับ Time Manager ในการจัดการการเคลื่อนเวลา ดังที่ได้อธิบายแล้วในเรื่องคลาส Time Manager

โดยสรุปสำหรับการทำงานของโมดูล Time Management นั้นผู้ใช้ OTcl สคริปต์สามารถใช้คำสั่งต่อไปนี้ได้ คือ

- การสมัครสมาชิกโดยผ่านฟังชัน subscribe หรือ subscribe-byโดยทำงานเหมือนใน Object Management ทั้งนี้เพราะอาศัยการทำงานของคลาส Sub คลาสเดียวกัน
- การกำหนดค่า Lookahead ผ่านทางฟังก์ชัน setlookahead โดยสามารถใช้คำสั่งได้ เช่น $fed setlookahed 2 ก็คือการกำหนดให้มีค่า Lookahead เป็น 2
- การกำหนดชนิดของการขอการเคลื่อนที่ของเวลา โดยจะกำหนดได้ เช่น set fed [Application/FedTMTar] หรือ set fed [Application/FedTMNer]
- การกำหนดค่าช่วงการเคลื่อนเวลา ผ่านทางฟังก์ชัน settimesize โดยสามารถใช้คำสั่งได้ เช่น $fed settimesize 1

ทั้งนี้การกำหนดค่าต่างๆของผู้ใช้ผ่านทาง OTcl สคริปต์สามารถทำได้ตลอดเวลาช่วงการทำงานทำให้สามารถเปลี่ยนแปลงการกำหนดค่าต่างๆได้ตลอด



รูปที่ 4.35 แสดงลำดับการทำงานของการส่ง Interaction และข้อความร้องขอการเคลื่อนเวลา

4.8 สรุป

ในบทนี้ได้กล่าวถึงการออกแบบระบบและการสร้างโมดูลทั้งหมด ซึ่งประกอบด้วยโมดูลหลัก 2 ส่วนคือ Object Management และ Time Management โดยได้อธิบายตั้งแต่แนวคิดเบื้องต้นสำหรับการออกแบบ จนถึงโครงสร้างของโมดูลที่เกี่ยวข้องทั้งหมด โดยตลอดทั้งบทได้พยายามชี้ให้เห็นถึงความสมนัยระหว่างระบบที่ได้ออกแบบและข้อกำหนดมาตรฐาน เพื่อให้แน่ใจว่าระบบที่ได้ออกแบบและสร้างทั้งหมดสามารถใช้จำลองการทำงานของ Federation ที่ใช้ HLA ได้ ในหัวข้อ 4.5 ยังได้กล่าวถึงวิธีการประยุกต์ใช้โมดูล Object Management มาใช้เป็น Data Distribution Management ด้วย