บทที่ 2

ทฤษฎีและหลักการทำงาน

2.1 ระบบจำลอง (Simulation System)

ระบบจำลองเป็นระบบการทำงานที่เลียนแบบการทำงานของระบบจริงหรือระบบต้นแบบ (Fujimoto, 2000) ตัวอย่างระบบจริงที่ใช้งาน เช่น การทำงานของธนาคารที่มีลักษณะการทำงานแบบวันต่อวัน, การทำงานอย่างเป็นระบบของโรงงานอุตสาหกรรม, เจ้าหน้าที่ที่ทำงานภายในโรงพยาบาล หรือ ระบบรักษาความปลอดภัยของบริษัท เป็นต้น

การจำลองบนคอมพิวเตอร์ (Computer Simulation) เป็นการทำงานของแบบจำลอง ซึ่งจะถูกแทนด้วยโปรแกรมคอมพิวเตอร์ สามารถทำงานได้อย่างละเอียด, เก็บรวบรวมผลการทำงานและนำกลับมาใช้งานใหม่ได้ไม่จำกัดจำนวนครั้ง

เทคโนโลยีแบบจำลองกระจายศูนย์ (Distributed Simulation Technology) คือ เทคโนโลยีที่ทำให้โปรแกรมแบบจำลองต่างๆ สามารถทำงานได้ภายใต้ระบบคอมพิวเตอร์แบบกระจายการประมวลผล (Distributed computer system) ซึ่งระบบเหล่านี้ประกอบไปด้วยเครื่องคอมพิวเตอร์หลายเครื่องติดต่อกันและทำงานพร้อมกันผ่านระบบเครือข่าย โดยแบ่งการทำงานออกเป็น 2 ส่วน คือ

ก. โปรแกรมแบบจำลอง คือโปรแกรมที่มีการคำนวณการทำงานตามโมเดลแบบจำลองที่ได้มีการออกแบบเอาไว้ ซึ่งโมเดลแบบจำลองนั้น อาจจะเลียนแบบจากระบบจริงและสร้างขึ้นจากข้อมูลหรือสมมติฐานที่มีอยู่ ซึ่งปัจจุบันระบบจำลองเหล่านี้มีการใช้งานอย่างแพร่หลายเพื่อวิเคราะห์พฤติกรรมของระบบต่างๆ เช่น แบบจำลองการขับเครื่องบิน

ข. เทคโนโลยีแบบจำลองแบบกระจายศูนย์ ที่ทำให้โปรแกรมที่สร้างขึ้นสามารถทำงานภายใต้ระบบเครือข่ายคอมพิวเตอร์ที่ทำงานแบบกระจายการประมวลผล

การจำลองแบบกระจายศูนย์มีข้อดีดังนี้

ก. ลดเวลาในการจำลองลง เนื่องจากสามารถแบ่งระบบจำลองขนาดใหญ่เป็นแบบจำลองย่อยๆหลายๆแบบจำลองและแยกประมวลผลได้พร้อมกัน

ข. สามารถกระจายความรับผิดชอบได้ ซึ่งเป็นผลสืบเนื่องมากจากการแบ่งย่อยแบบจำลอง โดยแต่แบบจำลองย่อยจะมีพื้นที่รับผิดชอบในระบบเสมือน (Virtual Work) ตัวอย่างเช่น แบบจำลองการจราจรทางอากาศ สามารถแบ่งความรับผิดชอบในการจำลองให้แก่

เครื่องคอมพิวเตอร์แต่ละเครื่องรับผิดชอบ คือ ให้คอมพิวเตอร์เครื่องหนึ่งจำลองการจราจรทางอากาศในจังหวัดสงขลา ส่วนเครื่องคอมพิวเตอร์อีกเครื่องจำลองการจราจรทางอากาศในจังหวัดภูเก็ต เป็นต้น

ค. สามารถรวมโปรแกรมหรือแบบจำลองหลายๆชนิด ที่สร้างขึ้นโดยนักพัฒนาที่แตกต่างกัน แต่อยู่บนเทคโนโลยีเดียวกัน ให้สามารถทำงานร่วมกัน ภายใต้สภาวะแวดล้อมเดียวกัน (Virtual Environment) ได้

ง. มีความทนทานต่อความผิดพลาดหรือความล้มเหลวของระบบสูง เนื่องจากการจำลองแบบกระจายศูนย์หากมีความผิดพลาดเกิดขึ้นที่แบบจำลองใดแบบจำลองหนึ่ง จะไม่ส่งต่อแบบกระทบต่อแบบจำลองอื่นมากนัก คือ แบบจำลองอื่นๆสามารถทำงานต่อไปได้ แต่อาจจะขาดข้อมูลจากแบบจำลองที่เกิดความผิดพลาด ซึ่งในบางระบบจริงการขาดผลการจำลองย่อยบางส่วนไม่ส่งผลต่อระบบโดยรวม

2.2 High Level Architecture (HLA)

สถาปัตยกรรมชั้นสูงหรือ HLA ได้ถูกออกแบบเพื่อให้เป็นสถาปัตยกรรมมาตรฐานสำหรับระบบการจำลองทั้งหมดของกระทรวงกลาโหม ประเทศสหรัฐอเมริกา และต้องสามารถทำงานร่วมกับโมเดลระบบจำลองที่ถูกสร้างขึ้นมากับระบบ Command, Control, Communications, Computers and Intelligence หรือ C41 (K. Coat, 2000) นอกจากนั้นจะต้องสะดวกต่อการนำส่วนประกอบต่างๆของโมเดลที่สร้างไว้กลับมาใช้งานใหม่ได้อีกด้วย โดยทางกระทรวงกลาโหมของสหรัฐอเมริกาคาดหวังว่ามาตรฐานดังกล่าวจะเข้ามาแทนที่ระบบจำลองแบบกระจายศูนย์แบบเก่า อย่างเช่น Distributed Interaction Simulation หรือ DIS (Jenese, Kuijpers and Dumay, 1996), Aggregate Level Simulation Protocol หรือ ALSP (Steffen StraBburger, 2000) เป็นต้น และได้ประกาศให้เป็นมาตรฐานของ Institute of Electrical and Electronic Engineering หรือ IEEE รวมไปถึงให้องค์กรต่างๆและบุคคลทั่วไปทั้งในและนอกประเทศสหรัฐอเมริกา สามารถใช้งานสถาปัตยกรรมชั้นสูงได้ฟรี จนกระทั่งปีค.ศ. 2002 กระทรวงกลาโหมของสหรัฐ ก็ได้แบ่งลิขสิทธิ์ของมาตรฐานดังกล่าวให้บริษัทของเอกชน เข้ามามีส่วนร่วมในการพัฒนาหรือเริ่มต้นการพัฒนาเพื่อการค้านั้นเอง

จุดเด่นของมาตรฐานสถาปัตยกรรมชั้นสูง (Devis and Moeller, 1999 ; Kuhl, Weatherly and Dahman, 1999)

ก. ความสามารถในการนำกลับมาใช้งานใหม่ (Reusability) เทคโนโลยีของสถาปัตยกรรมชั้นสูง สามารถนำเอาแบบจำลองที่ได้มีการออกแบบไว้ก่อนหน้านี้ (ก่อนที่จะมีมาตรฐานนี้) ทั้งแบบจำลองหรือบางส่วนของแบบจำลองมาใช้งานร่วมกับแบบจำลองที่สร้างภายใต้สถาปัตยกรรมชั้นสูงได้โดยไม่ต้องเขียนโปรแกรมจำลองใหม่ทั้งหมด ซึ่งทำให้แบบจำลองที่ใช้เทคโนโลยีแบบเก่าสามารถนำมาพัฒนาร่วมกับเทคโนโลยีสถาปัตยกรรมชั้นสูงและสร้างแบบจำลองที่ซับซ้อนยิ่งขึ้นได้

ข. การก้าวข้ามข้อจำกัดของระบบปฏิบัติการ (Interoperation) แบบจำลองแบบกระจายศูนย์มีปัญหาที่พบได้บ่อยคือปัญหาในเรื่องของระบบปฏิบัติการที่ต่างกันไม่สามารถทำงานร่วมกันได้ และรูปแบบการจัดการเวลาที่แตกต่างกัน (Time Management) ซึ่งปัญหาเหล่านี้สามารถแก้ไขได้โดยการใช้เทคโนโลยีของสถาปัตยกรรมชั้นสูง เนื่องจากสถาปัตยกรรมชั้นสูงสามารถทำงานร่วมกับระบบปฏิบัติการต่างๆได้ทั้ง Windows 98, Windows NT, Windows2000, Linux, และ Unix นอกจากนี้สถาปัตยกรรมชั้นสูงยังสามารถตอบสนองต่อการจัดการเวลาได้ทั้งแบบเวลาต่อเนื่อง (Continuous Simulation) และแบบขึ้นกับเหตุการณ์ (Discrete Events Simulation)

ข้อกำหนดการทำงานตามมาตรฐานของสถาปัตยกรรมชั้นสูง ได้แบ่งส่วนประกอบหลักออกเป็น 3 ส่วน คือ

ก. HLA Rules ในส่วนนี้ได้อธิบายกฎหรือสิ่งที่ต้องทำเพื่อสร้าง Federate, Federation และการสร้างความสัมพันธ์กับ RTI (Run-Time Infrastructure) (Dahman, Fujimoto and Weatherly, 1998 ; Kuhl, Weatherly and Dahman, 1999) ซึ่งจะอธิบายรายละเอียดความสัมพันธ์ในหัวข้อที่ 2.3 กฎในส่วนนี้มีทั้งหมด 10 ข้อ แบ่งเป็น กฎสำหรับ Federate 5 ข้อ และ Federation 5 ข้อ

กฎสำหรับ Federation หรือ Federation Rules

(1) แต่ละ Federation จะต้องมี HLA Federation Object Model หรือ FOM โดยมีรูปแบบที่สามารถสร้างขึ้นได้ตาม OMT

(2) ภายใน Federation ทุกๆวัตถุที่มีอยู่ใน FOM นั้นจะต้องอยู่ใน Federate ด้วย

(3) ในระหว่างการดำเนินการของ Federation แต่ละ Federate จะต้องติดต่อกับ RTI ตามที่ระบุไว้ใน HLA Interface Specification

(4) ในระหว่างการดำเนินการของ Federation ค่าคุณลักษณะหรือ Attribute ของแต่ละวัตถุจะต้องมี Federate ที่เป็นเจ้าของเดียว ณ.เวลาหนึ่งๆเท่านั้น แต่เราสามารถถ่ายโอนความเป็นเจ้าของระหว่าง Federate ได้

(5) ในระหว่างการดำเนินการของ Federation การแลกเปลี่ยนข้อมูลตาม FOM ระหว่าง Federate จะต้องกระทำผ่าน RTI

กฎสำหรับ Federate หรือ Federate Rules

(1) แต่ละ Federate จะต้องมี HLA Simulation Object Model หรือ SOM ที่ได้ออกแบบไว้ตามที่ระบุไว้ใน OMT

(2) แต่ละ Federate ต้องสามารถเปลี่ยนแปลงค่าและรับค่าคุณลักษณะของวัตถุใดๆ ที่ระบุไว้ตาม SOM ได้ นอกจากนี้ต้องสามารถส่งหรือรับข้อความที่ส่งถึงกันที่ระบุไว้ใน SOM ได้อีกด้วย

(3) แต่ละ Federate ต้องสามารถส่งผ่านหรือรับความเป็นเจ้าของคุณลักษณะของวัตถุหนึ่งได้ แม้ว่า Federation กำลังดำเนินการอยู่

(4) แต่ละ Federate ต้องสามารถเปลี่ยนแปลงค่าสถานะต่างๆได้ในระหว่างการเปลี่ยนแปลงค่าข้อมูลของวัตถุ

(5) แต่ละ Federate นั้นจะต้องสามารถจัดการเวลาของตัวเองได้ เพื่อให้สามารถทำงานสำหกรับการแลกเปลี่ยนข้อมูลกับสมาชิกหรือ Federate อื่นใน Federation ได้

ข. Interface Specification เป็นการระบุการเชื่อมต่อระหว่าง Federate กับ RTI ซึ่งได้นิยามเอาไว้ในรูปแบบของการให้บริการของแบบจำลองหรือของ RTI ในระหว่างที่ federation กำลังปฏิบัติการ ซึ่งจะอธิบายรายละเอียดในหัวข้อถัดไป

ค. OMT เป็นรูปแบบทั่วไปที่ทำหน้าที่ในการอธิบายข้อมูลของความสนใจโดยทั่วไปของแบบจำลองใน Federation (Kuhl, Weatherly and Dahmann, 1999 ; Defense Modeling and Simulation office, U.S. Department of Defense) ซึ่ง OMT มีส่วนประกอบดังนี้คือ

(1) Object model specification table เป็นการเก็บข้อมูลที่มีความสำคัญกับ HLA object model

(2) Object class structure table เป็นการบันทึกค่า namespace ของ federate ทั้งหมด หรือ Federation class object และอธิบาย class-subclass relationships

(3) Interaction structure class table เป็นการบันทึกค่า namespace ของ federate ทั้งหมด หรือ federation class object และอธิบาย class-subclass relationships

(4) Attribute table เป็นการอธิบายลักษณะของ object attribute ใน federate หรือ federation

(5) Parameter table เป็นการอธิบายลักษณะของพารามิเตอร์แบบข้อความที่ส่งถึงกันของ federate หรือ federation

(6) Dimension table เป็นการระบุขนาดสำหรับลักษณะประจำของอ็อบเจกต์และข้อความที่ส่งถึงกัน

(7) Time representation table เป็นตารางเก็บค่าของเวลา

(8) User-supplied tag table เป็นการเก็บค่าของ tags ที่ถูกใช้ในการให้บริการของ HLA

(9) Synchronization table เป็นการเก็บค่ารายการและชนิดของข้อมูลที่ใช้ในการให้บริการแบบ Synchronization

(10) Transportation type table เป็นการอธิบายถึงกลไกที่ใช้ในการขนย้ายข้อมูล

(11) Switches table เป็นการกำหนดค่าของพารามิเตอร์เริ่มต้นที่ถูกใช้งานโดย RTI

(12) Datatype table เป็นการอธิบายรายละเอียดของค่าที่จะแสดงใน Object model

(13) Notes table เป็นการอธิบายเพิ่มเติมสำหรับ OMT table อื่นๆ

(14) FOM/SOM lexicon เป็นการนิยามอ็อบเจกต์, ลักษณะประจำ, ข้อความที่ส่งถึงกัน และพารามิเตอร์ ที่ถูกใช้งานใน HLA object model

2.3 Run-Time Infrastructure (RTI)

Runt-Time Infrastructure หรือ RTI เป็นโปรแกรมซึ่งสำคัญที่สุดสำหรับการติดต่อระหว่าง Federate ภายใน Federation (Kuhl, Wheatherly and Dahman, 1999) เพราะทำหน้าที่ให้บริการเกี่ยวกับระบบแบบจำลองทั้งหมด เพื่อให้แต่ละ Federate สามารถติดต่อแลกเปลี่ยนข้อมูลระหว่างกันได้ โดยหน้าที่หลักๆของ RTI สามารถแบ่งออกได้เป็นดังนี้

ก. เพื่อให้สามารถแยกระหว่างการจำลองและการติดต่อสื่อสารออกจากกันภายในระบบสถาปัตยกรรมชั้นสูง

ข. เพื่อปรับปรุงประสิทธิภาพการทำงานของมาตรฐานเก่าๆ อย่างเช่น DIS หรือ ALSP เป็นต้น

ค. เพื่ออำนวยความสะดวกในการสร้างและทำลาย Federation

ง. สนับสนุนการประกาศอ็อบเจกต์ (Object Declaration) และการจัดการระหว่างแต่ละ Federate เป็นตัวกลางในการช่วยการจัดการกับเวลาทั้งหมด

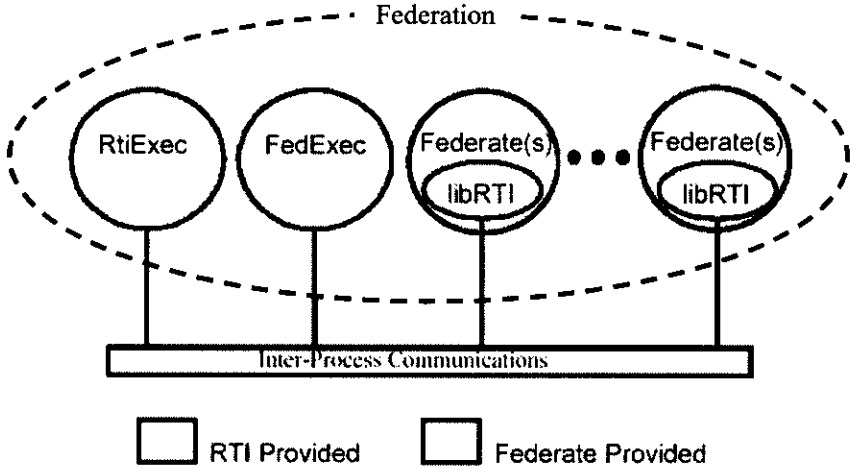
จ. ให้การติดต่อที่มีประสิทธิภาพสำหรับทุกๆ Federate

ปัจจุบัน RTI ประกอบด้วยส่วนประกอบหลัก 3 ส่วน ดังภาพประกอบ 2-1 ได้แก่

ก. RTI Execution Process (RtiExec)

ข. Federation Execution Process (FedExec)

ค. LibRTI library



ภาพประกอบ 2-1 แสดงส่วนประกอบการทำงานผ่าน RTI (Defense Modeling and Simulation office, U.S. Department of Defense. 2000)

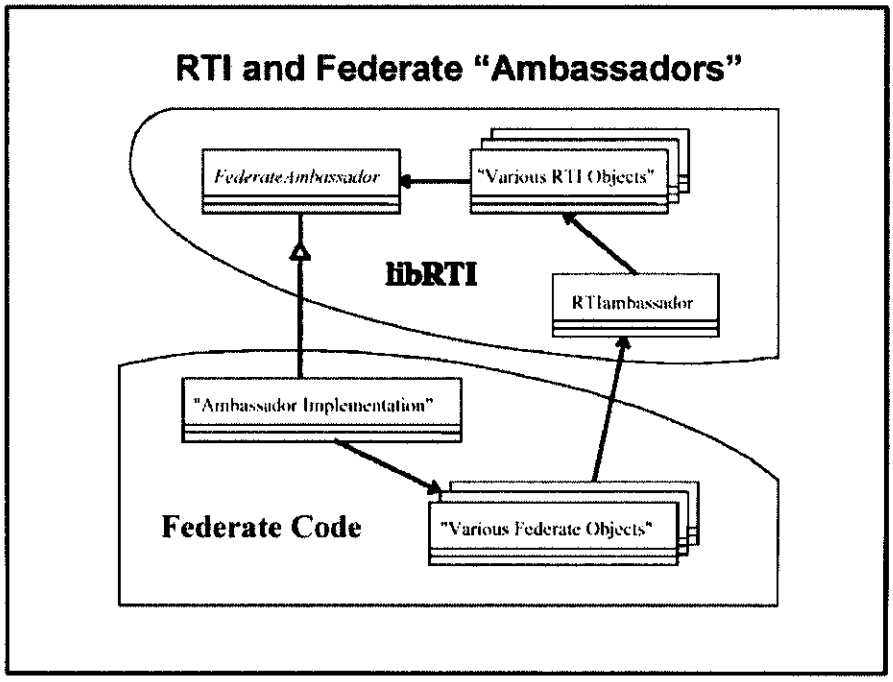
จากภาพประกอบ 2-1 แต่ละ Federate จะต้องประกอบไปด้วย libRTI เพื่อใช้ติดต่อกับ RTI และในเวลาเดียวกัน Federate หนึ่งสามารถทำงานร่วมกับ Federation ได้หลายๆ Federation พร้อมกัน

โปรแกรม RTI สามารถทำงานได้ทั้งบนเครื่องเดียวและทำงานในระบบเครือข่าย แต่อย่างไรก็ตามในระบบเครือข่ายเดียวกัน (Local Area Network เดียวกัน) สามารถมีโปรแกรม RTI ที่ดำเนินงานอยู่ได้เพียงตัวเดียวเท่านั้น โดยแต่ละส่วนประกอบของ RTI มีหน้าที่ดังนี้

ก. RtiExec รับผิดชอบในการสร้างและทำลายการดำเนินการของ Federation โดยแต่ละ Federate นั้น เริ่มต้นจะต้องติดต่อกับ RtiExec ก่อนจึงจะสามารถเข้าร่วมใน Federation ที่ต้องการได้ ดังนั้นจุดประสงค์หลักของ RtiExec คือสร้างและทำลาย FedExec นั้นเอง

ข. FedExec จะเป็นตัวแทนของแต่ละ Federation คือแต่ละ Federation จะมี FedExec ซึ่งรับผิดชอบในการจัดการกับ Federation นั้นๆ โดย FedExec จะยอมให้ Federate ที่อยู่ภายใน Federation นั้นสามารถเข้าร่วมหรือสิ้นสุดการเข้าร่วมได้ รวมไปถึงการอำนวยความสะดวกในการแลกเปลี่ยนข้อมูลระหว่าง Federate อีกด้วย

ค. libRTI เป็นคลังโปรแกรมซึ่งมีทั้งสำหรับ Java และ C++ เพื่อเก็บการให้บริการสำหรับ RTI ตามที่ระบุไว้ใน Interface Specification ให้กับนักพัฒนาเพื่อสร้าง Federate ที่สามารถติดต่อกับ RTI ได้ โครงสร้างของคลาสที่ติดต่อกับ RTI และโปรแกรมการทำงานของแบบจำลองแสดงไว้ในภาพประกอบ 2-2

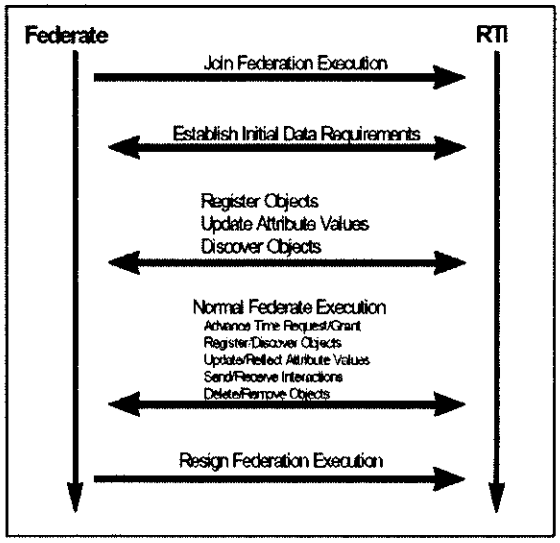


ภาพประกอบ 2-2 แสดงการเขียนโปรแกรมเพื่อเชื่อมต่อแบบจำลองกับ RTI (Defense Modeling and Simulation office, U.S. Department of Defense. 2000)

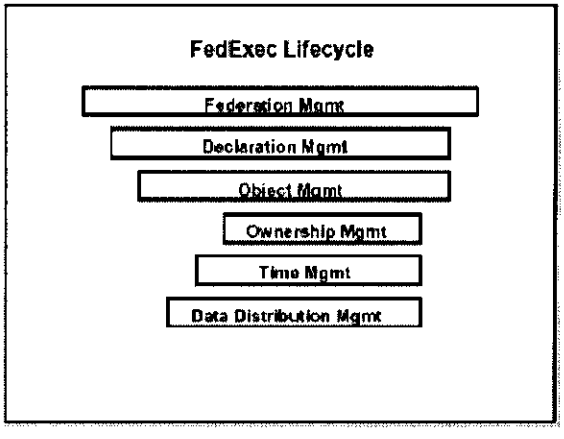
HLA Interface Specification ได้แบ่ง libRTI ออกเป็น 2 ส่วนหลัก ได้แก่ ส่วนที่ให้บริการเกี่ยวกับฟังก์ชันการติดต่อกับ RTI และ Federate อื่นทั้งหมด ซึ่งก็คือ RTI Ambassador และส่วนของฟังก์ชัน Call back ที่ RTI ใช้เพื่อติดต่อกับ Federate นั้น และการส่งข้อมูลทั้งหมดที่ Federate นั้นต้องการก็ต้องผ่านฟังก์ชัน Call back นี้ด้วย ซึ่งก็คือ Federate Ambassador โดย RTI Ambassador จะเป็นคลาสที่สมบูรณ์สามารถเรียกใช้งานได้ทันที แต่ Federate Ambassador จะเป็นคลาสอย่างย่อหรือ Abstract Class ซึ่งนักพัฒนาสามารถเพิ่มเติมโปรแกรมภายในฟังก์ชันเพื่อให้เหมาะสมกับ Federate ที่ตนเองสร้างขึ้น

2.4 การทำงานของ HLA Interface Specification

การทำงานร่วมกันระหว่าง Federate และ Federation สามารถแสดงได้ดังภาพประกอบ 2-3 ซึ่งใน HLA Interface Specification ได้แบ่งการทำงานร่วมกันระหว่าง Federate และ Federation ออกเป็นวัฏจักรการจัดการ 6 ส่วน (Defense Modeling and Simulation office, U.S. Department of Defense. 2000) ดังภาพประกอบ 2-4



ภาพประกอบ 2-3 แสดงการเชื่อมต่อระหว่างแบบจำลองกับ RTI ภายใต้ HLA Interface Specification (Defense Modeling and Simulation office, U.S. Department of Defense. 2000)



ภาพประกอบ 2-4 แสดงวัฏจักรการทำงานของ FedExec (Defense Modeling and Simulation office, U.S. Department of Defense. 2000)

2.4.1 Federation Management

รับผิดชอบการจัดการ Federation, Name space, Transportation, Ordering default, routing space ซึ่งรวมไปถึงงานหลักที่มีการใช้งานบ่อยๆได้แก่

ก. การสร้าง Federation ขึ้นมา โดยเรียกใช้งานฟังก์ชัน

RTIambassador::createFederateionExecution()

ข. การเข้าร่วมใน Federation นั้น โดยเรียกใช้ฟังก์ชัน

RTIambassador::joinFedaertionExecution()

ค. การออกจาก Federation นั้นโดยเรียกใช้ฟังก์ชัน

RTIambassador::resignFederationExecution()

ง. การทำลาย Federation นั้น โดยเรียกใช้ฟังก์ชัน

RTIambassador::detroyFederationExecution

2.4.2 Declaration Management

หน้าที่จัดการกับระบบชนิดของข้อมูลที่ Federate จะส่งและรับมา นอกจากนั้นยังควบคุมข้อมูลที่ต้องการซึ่งขึ้นอยู่กับความต้องการของแต่ละ Federate การทำงานที่มีการใช้งานบ่อยๆได้แก่

ก. การประกาศข้อมูลของอ็อบเจกต์และข้อความที่ส่งถึงกันที่ Federate สามารผลิตได้

RTIambassador::publishObjectClass()

RTIambassador::publishInteractionClass()

ข. การประกาศข้อมูลของอ็อบเจกต์และข้อความที่ส่งถึงกันที่ Federate นั้นต้องการจาก Federate อื่นๆ

RTIambassador::subacribeObjectClassAttribute()

RTIambassador::subscribeInteractionClass()

2.4.3 Object ManageMent

ทำหน้าที่จัดการข้อมูลของตัวของอ็อบเจกต์ เพื่อแจ้งให้ Federate ภายใน Federation ทราบ การทำงานที่มีการใช้งานบ่อยๆได้แก่

2.4.3.1 การสร้างเปลี่ยนแปลง หรือลบอ็อบเจกต์หรือข้อความที่ส่งถึงกันใดๆ การทำงานที่มีการใช้งานบ่อยๆได้แก่

ก. การสร้างอ็อบเจกต์

RTIambassador::registerObjectInstance()

ข. การลบอ็อบเจกต์

RTIambassador::deleteObjectInstance()

ค. การเปลี่ยนแปลงค่าของอ็อบเจกต์

RTIambassador::updateAttributeValues()

ง. การรับข้อมูลที่มีการเปลี่ยนแปลงของอ็อบเจกต์จาก Federate อื่นๆ

Federateambassador::reflectAttributeValues()

จ. การรับและส่งข้อความที่ส่งถึงกัน

RTIambassador::sendInteraction()

Federateambassador::receiveInteraction()

2.4.3.2 จัดการเกี่ยวกับการแยกแยะอ็อบเจกต์ต่างๆ

2.4.3.3 อำนวยความสะดวกในเรื่องของการลงทะเบียนวัตถุและการกระจายข้อมูล

2.4.3.4 จัดการการแลกเปลี่ยนข้อมูลร่วมกันของแต่ละ Federate

2.4.3.5 สนับสนุนการส่งผ่านข้อมูลและการจัดการเวลาที่แตกต่างกัน

2.4.4 Ownership Management

โดยปกติ RTI ยอมให้มีการกระจายความรับผิดชอบในการเปลี่ยนแปลงข้อมูลของวัตถุและลบวัตถุนั้นออกไปได้โดยมีข้อจำกัดเพียงเล็กน้อยเท่านั้น ซึ่งในกรณีนี้หน้าที่ความรับผิดชอบการเปลี่ยนแปลงข้อมูลของวัตถุทั้งหมดรวมไปถึงการลบวัตถุเมื่อไม่มีการใช้งานแล้วจะเป็นของ Federate ทั้งหมด ที่เป็นเจ้าของอยู่ และมีความเป็นไปได้ที่มี Federate มากกว่า 2 Federate ที่มีหน้าที่รับผิดชอบการเปลี่ยนแปลงข้อมูลสำหรับวัตถุเดียวกัน แต่จากกฎจะมีเพียง Federate เดียวเท่านั้นที่สามารถเปลี่ยนแปลงข้อมูลของวัตถุหนึ่งๆได้ ณ.เวลาหนึ่งๆ ดังนั้นในส่วนนี้จึงทำหน้าที่จัดการควบคุมการเป็นเจ้าของวัตถุโดยการถ่ายโอนวัตถุจาก Federate หนึ่งไปยังอีก Federate หนึ่งได้ เพื่อให้ Federate ที่ต้องการเปลี่ยนแปลงค่าของอ็อบเจกต์เดียวกันสามารถเปลี่ยนแปลงค่าของข้อมูลได้ที่เวลาแตกต่างกัน

2.4.5 Time Management

ส่วนนี้มีหน้าที่รับผิดชอบเกี่ยวกับกลไกการจัดการเวลา และการเปลี่ยนแปลงเวลาบนแกนเวลาของสถาปัตยกรรมชั้นสูงซึ่งมีหน้าที่หลักๆที่สำคัญคือ

2.4.5.1 เริ่มเหตุการณ์ (Event) และเข้าร่วมกับแกนเวลาของสถาปัตยกรรมชั้นสูงซึ่งแต่ละ Federate จะได้รับ Federate time สำหรับแต่ละเหตุการณ์ส่งผ่านข้อมูลของอ็อบเจกต์และตัวแทนของอ็อบเจกต์ซึ่งจะสัมพันธ์กับรูปแบบของเวลาของ Federate นั้น

2.4.5.2 สนับสนุนพฤติกรรมต่างๆของ Federate ภายใน Federation นั้น

2.4.5.3 สนับสนุนการเกิด Interaction ระหว่าง Federate ที่มีการใช้รูปแบบของเวลาที่แตกต่างกัน

2.4.5.4 ฟังก์ชันเกี่ยวกับ Time Management ที่มีการเรียกใช้งานบ่อยได้แก่

ก. เข้าสู่และยกเลิกสถานะ Regulation

RTIambassador::enableTimeRegulation()

RTIambassador::disableTimeRegulation()

ข. เข้าสู่และยกเลิกสถานะ Constrained

RTIambassador::enableTimeConstrained()

RTIambassador::disableTimeConstrained()

ค. เปลี่ยนแปลงค่าของเวลาให้เป็นเวลาตามที่กำหนดไว้

RTIambassador::timeAdvanceRequest()

ง. เปลี่ยนแปลงค่าเวลาให้เป็นไปตามเหตุการณ์ล่าสุดที่รับมาได้

RTIambassador::nextEventRequest()

จ. ได้รับการยอมรับให้มีการเปลี่ยนแปลงเวลาจาก RTI

Federateambassador::timeAdvanceGrant()

2.4.6 Data Distribution Management (DDM)

Data Distribution Management เป็นกลไกที่ทำหน้าที่ช่วยให้เกิดความสะดวกในการขยายขอบเขตสำหรับการ Publication และ Subscription โดยใช้ความสามารถในการค้นหาเส้นทาง (Routing) สำหรับการรับและส่งข้อมูล โดยใช้เทคโนโลยี Multicast และการจัดเก็บตารางลักษณะประจำ, อ็อบเจกต์ และข้อความที่ส่งถึงกัน

2.5 เวลา

การสร้างแบบจำลองแบบกระจายศูนย์สิ่งที่สำคัญอย่างหนึ่งคือการจัดการเวลาในระหว่างที่แบบจำลองทำงานร่วมกัน ซึ่ง RTI สนับสนุนการจัดการเวลาได้หลายรูปแบบ โดยการจัดการเวลาที่กล่าวถึงนี้เป็นเพียงรูปแบบหนึ่งเท่านั้น

2.5.1 เวลาพื้นฐาน

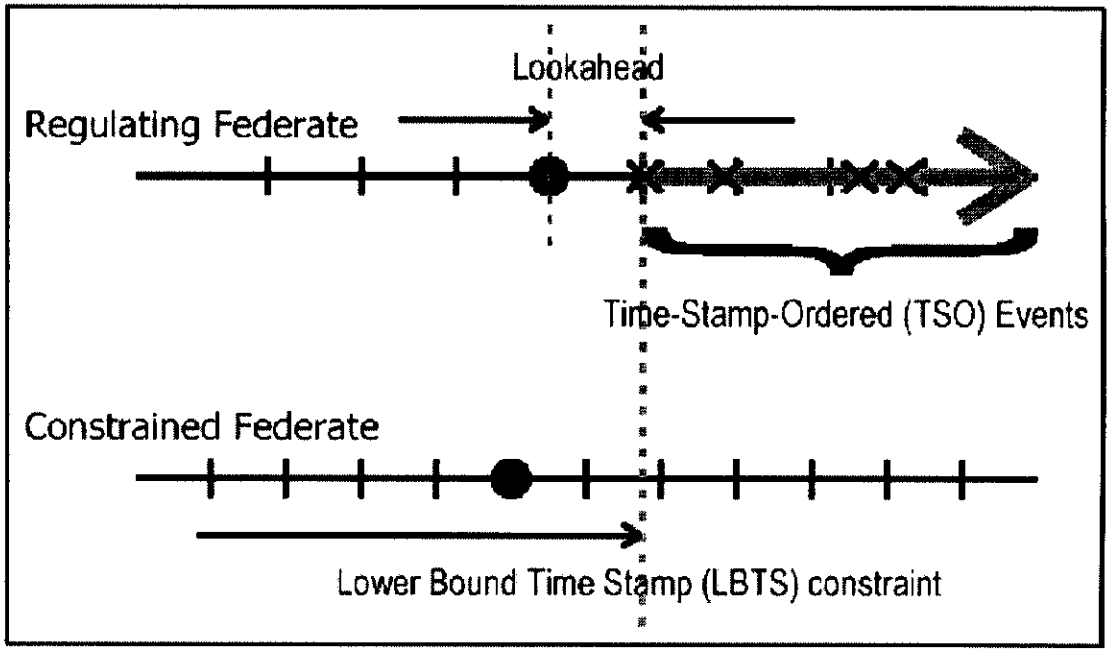
การสร้างแบบจำลองจะมีการพิจารณาเวลาพื้นฐานที่สำคัญเป็น 3 ชนิดคือ เวลาในระบบต้นแบบ (Physical time), เวลาของแบบจำลอง (Simulation time) และเวลาที่ใช้ในการจำลอง (Wallclock time) เช่น การจำลองการเกิดน้ำท่วมในวันที่ 1 พฤศจิกายน ระหว่างเวลา 18.00 – 20.00 น. เวลาที่ใช้ในส่วนนี้จะเป็นเวลาในระบบต้นแบบ เมื่อนำมาใช้ในระบบจำลอง เวลาที่ใช้ในแบบจำลองจะมีค่าตั้งแต่ 18.0 ถึง 20.0 เพื่อแทนค่าของเวลาในระบบต้นแบบจากเวลา 18.00 – 20.00 น. และแบบจำลองดังกล่าวใช้เวลาในการจำลอง 5 นาทีคือจากเวลา 18.00 – 18.05 น. (Fujimoto, 2000)

จากรูปแบบของเวลาที่ใช้ในการสร้างแบบจำลองทำให้เราสามารถแบ่งชนิดของแบบจำลองออกได้เป็น 3 ชนิด คือ

ก. As-fast-as-possible คือแบบจำลองที่ทำงานให้เร็วที่สุดเท่าที่จะทำได้ โดยไม่สนใจความสัมพันธ์ระหว่างเวลาของแบบจำลองกับเวลาที่ใช้ในการจำลอง

ข. Real-time execution คือแบบจำลองที่ใช้เวลาของแบบจำลองต่อเวลาที่ใช้ในการจำลองเป็นอัตราหนึ่งต่อหนึ่ง หรือเวลาของแบบจำลองเท่ากับเวลาที่ใช้ในการจำลองนั้นเอง

ค. Scale real-time execution คือแบบจำลองที่ใช้เวลาของแบบจำลองเป็นอัตราส่วนต่อกันกับเวลาที่ใช้ในการจำลอง ซึ่งการจำลองแบบนี้สามารถจำลองระบบต้นแบบให้เร็วกว่าหรือช้ากว่าเวลาจริงได้ ทำให้เกิดความเหมาะสมในการนำแบบจำลองดังกล่าวมาใช้ในการพิจารณาเพื่อหาความสัมพันธ์กับระบบจริง



ภาพประกอบ 2-5 แสดงการจัดการเวลาของแบบจำลองแบบ Regulating และ Constrained (Defense Modeling and Simulation office, U.S. Department of Defense. 2000)

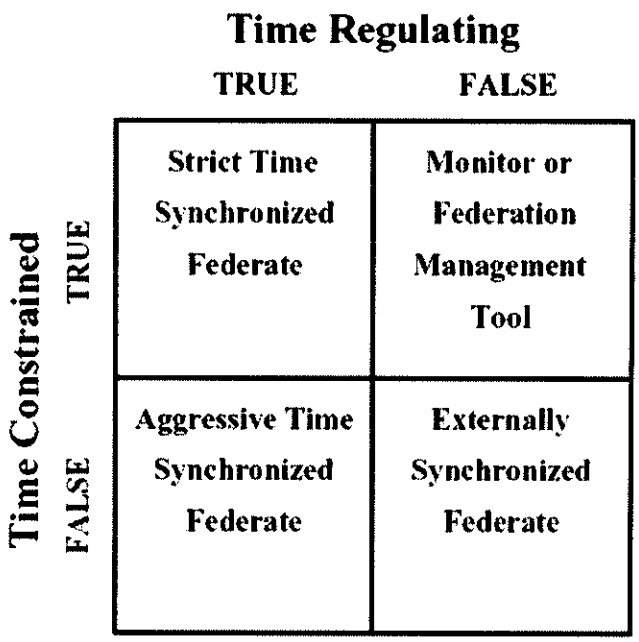
2.5.2 Regulating Federate และ Constrained Federate

สถาปัตยกรรมชั้นสูงสามารถแสดงความสัมพันธ์ระหว่างค่าต่างๆที่มีการใช้งานเมื่อ Federate อยู่ในสถานะ Regulating และ Constrained ได้ดังภาพประกอบ 2-5

Regulating Federate คือ Federate ที่สามารถสร้างเหตุการณ์ที่ลงเวลาได้ (Time-stamp ordered event หรือ TSO event) โดย TSO event นั้นจะเกิดขึ้นที่เวลาใดเวลาหนึ่งบนแกนเวลา ส่วน

Federate ที่ไม่ใช่ Regulating ก็สามารถสร้างเหตุการณ์ได้แต่จะไม่มีความสัมพันธ์กับเวลาหรือขึ้นกับแกนของเวลา โดย Regulating Federate จะเปลี่ยนแปลงค่าของเวลาร่วมกับ Local RTI Component (LRC) และแต่ละ Federate นั้นสามารถเปลี่ยนแปลงการทำงานเกี่ยวกับเวลาได้เรื่อยๆ เช่น เปลี่ยนจาก Regulating Federate เป็น Non-regulating Federate เป็นต้น

Lookahead เป็นตัวแปรที่แต่ละ regulating federate ต้องมีอยู่ภายในเพื่อบอกให้ RTI ทราบว่า federate นั้นจะสร้างเหตุการณ์แบบ TSO event ที่มีการลงเวลาเท่ากับหรือไม่น้อยกว่า "$t_{current} + t_{lookahead}$" ซึ่งหมายความว่า Regulating Federate สามารถสร้าง TSO event ที่มีความเร็วที่สุดที่เป็นไปได้ไม่น้อยกว่าหรือก่อนเวลา "$t_{current} + t_{lookahead}$". โดยเทียบกับเวลาปัจจุบัน Regulating federate ต้องระบุค่า Lookahead นี้ตั้งแต่ตอนเริ่มต้นการทำงาน แต่สามารถเปลี่ยนแปลงค่าได้ในขณะที่ทำงานอยู่ โดย Lookahead นั้นเป็นลักษณะของ Conservative algorithm ที่ทำให้ระบบสามารถคำนวณผลเหตุการณ์ที่มีค่าบันทึกเวลา (Time stamp) มากกว่าเวลาปัจจุบันได้โดยปลอดภัย



ภาพประกอบ 2-6 แสดงการจัดแบ่งรูปแบบของแบบจำลองตามการทำงานแบบ Regulating และ Constrained (Carothers, et al., 1997)

TSO event เป็นเหตุการณ์ที่มีการลง Time-stamp ให้ทำงานเป็นลำดับตามเวลา และมีเพียง Regulating Federate เท่านั้นที่สามารถสร้าง TSO event ได้ ซึ่ง Regulating Federate สามารถสร้างได้ทั้ง TSO event และ Non-TSO event แต่ทุกๆ TSO event จะต้องเกิดขึ้นที่เวลา "$t_{current} + t_{lookahead}$" หรือมากกว่า และเป็นไปได้ที่ Regulating Federate สามารถสร้าง TSO event ที่เวลา "$t_{current}$

+ $t_{lookahead}$+5" ก่อนที่จะสร้าง TSO event ที่เวลา "$t_{current}$ + $t_{lookahead}$+2" และจะเป็นหน้าที่ของ LRC ของ Constrained Federate ที่ต้องเรียงลำดับเหตุการณ์เหล่านี้

Constrained Federate คือแบบจำลองที่สามารถรับ TSO event ได้ ส่วนแบบจำลองที่ไม่ใช่ Constrained Federate สามารถรับ TSO event ได้เช่นกัน แต่จำเป็นต้องดำเนินการไม่เป็นไปตามลำดับเวลาหรือไม่มีข้อมูลเกี่ยวกับเวลา

Lower bound time stamp (LBTS) เป็นตัวแปรที่ใช้ภายใน Constrained Federate โดยค่า LBTS จะเป็นตัวบอกให้ RTI ทราบว่าแบบจำลองสามารถรับ TSO event ได้เร็วที่สุดที่เวลาไหน ค่านี้จะถูกกำหนดโดย TSO event ที่จะมาถึงเร็วที่สุดที่อาจเกิดขึ้นได้สำหรับทุกๆ Regulating Federate ซึ่ง Constrained Federate ไม่สามารถที่จะเปลี่ยนแปลงเวลาให้เกิน LBTS ได้ เนื่องจาก RTI ไม่รับประกันว่าจะไม่มี Packet ที่ส่งผ่านมายังแบบจำลองก่อนหน้าเวลานี้

จากรูปแบบการทำงานของแบบจำลองแบบ Regulating และ Constrained สามารถจัดแบ่งแบบจำลองออกได้เป็น 4 ประเภท ดังภาพประกอบ 2-6

2.5.3 การเปลี่ยนแปลงค่าเวลา (Advancing Time)

จากภาพประกอบ 2-7 เป็นแผนภาพประกอบแทนแกนเวลาของแต่ละ Federate ภายใน Federation โดยแต่ละ Federate ใช้นโยบายเกี่ยวกับเวลาของตัวเองในการจัดการเวลา ภายใน Federation นี้มี 6 Federate โดยมี 5 Federate ได้เข้าร่วมภายใน Federation นี้แล้วแต่มี Federate #6 ยังไม่ได้เข้าร่วมจะเรียกว่า late arriving ส่วนที่เป็นจุดวงกลมคือเวลาปัจจุบันของแต่ละ Federate ซึ่งต้องเข้าใจว่าไม่ใช่เวลาของ Federation หรือ Federation time แต่ละ Federate มีอิสระที่จะเปลี่ยนแปลงค่าเวลา

ในส่วนที่เป็นสีเทาก็คือค่า Lookahead ที่ระบุไว้ใน Regulating Federae ซึ่งจะเห็นได้ว่า Federate #1 มีค่า Lookahead เป็น 2 ช่วงเวลา ส่วน Federate #2 และ #3 มีค่า Lookahead เป็น 1 ช่วงเวลา แสดงว่าค่า Lookahead นี้ไม่จำเป็นต้องสัมพันธ์กับช่วงเวลาของ Federate

จากภาพประกอบ 2-7 จะเห็นเวลาของแต่ละ Federate เป็นดังนี้

Federate #1 เวลาปัจจุบันคือ 17

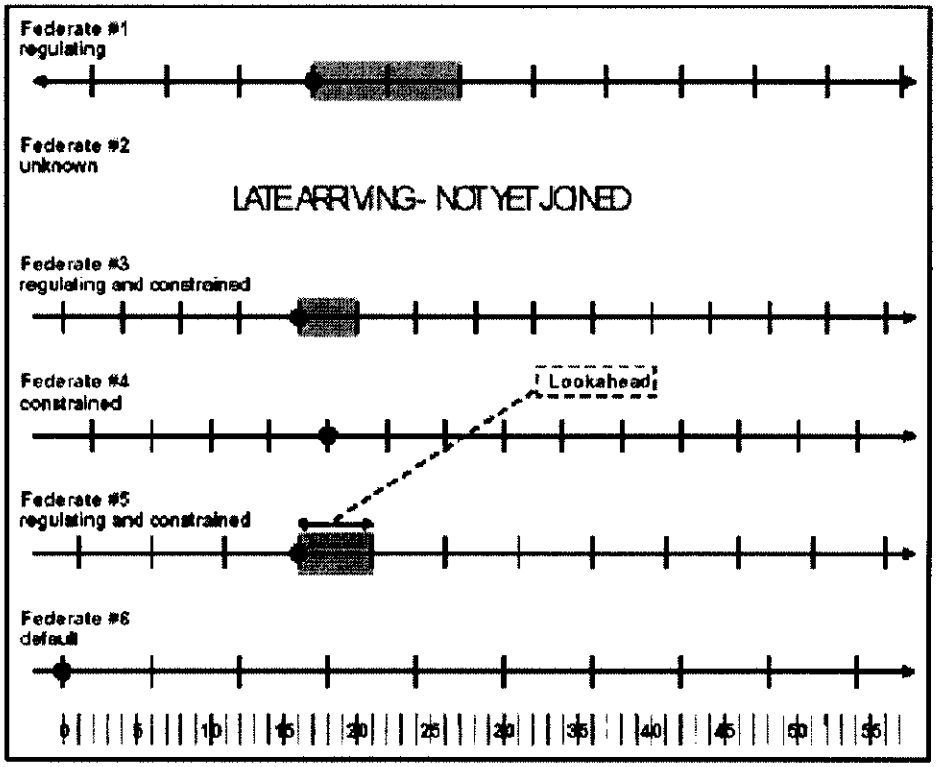
Federate #2 ยังไม่เข้าร่วมกับ Federation

Federate #3 เวลาปัจจุบันคือ 16

Federate #4 เวลาปัจจุบันคือ 18

Federate #5 เวลาปัจจุบันคือ 16

Federate #6 เวลาปัจจุบันคือ 0



ภาพประกอบ 2-7 แสดงลักษณะการเลื่อนเวลาของแบบจำลอง (Defense Modeling and Simulation office, U.S. Department of Defense. 2000)

โดยทั่วไปแล้ว Unconstrained Federate จะมีอิสระในการเพิ่มค่าเวลา ไม่ต้องร้องขอเปลี่ยนแปลงค่าเวลากับ RTI

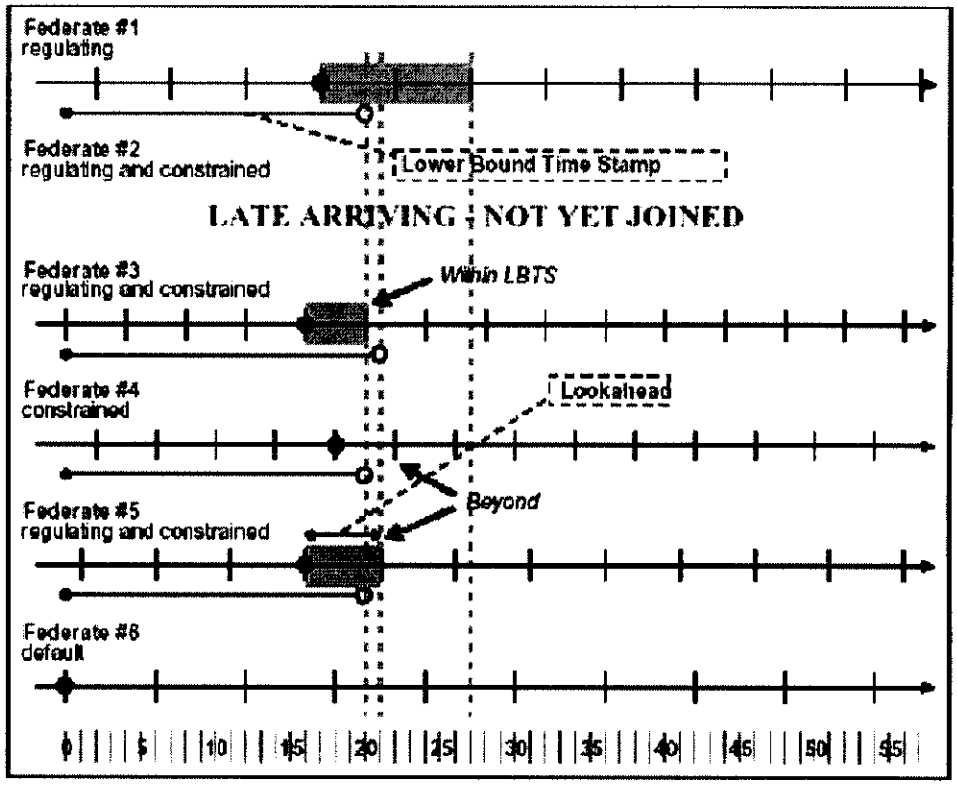
2.5.3.1 LBTS Constrained

Constrained Federate นั้นไม่สามารถเปลี่ยนแปลงค่าเวลาเกินกว่าค่า LBTS ปัจจุบัน ค่า LBTS จะถูกคำนวณโดยเป็นค่าเวลาที่เร็วที่สุดของข้อมูล (message) ที่ถูกส่งมาให้โดย Federate อื่นๆ ดังนั้นเมื่อ Regulating Federate เปลี่ยนแปลงจะทำให้ LBTS ของ Constrained Federate เพิ่มขึ้นมาด้วย

จากภาพประกอบ 2-8 เส้นประแนวตั้งแทน TSO message ที่เร็วที่สุดที่เป็นไปได้ของแต่ละ Regulating Federate ซึ่งก็คือค่าเวลาปัจจุบัน (Current time) บวกกับค่า Lookahead และจะสังเกตเห็นเส้นแนวนอนของแต่ละ Federate แทนเวลาตั้งแต่ t = 0 จนถึงค่า LBTS สังเกตเห็นว่าเวลาปัจจุบันของแต่ละ Federate จะขึ้นกับค่า LBTS

Constrained Federate จะเป็นอิสระในการเปลี่ยนแปลงเวลาในระหว่างที่ไม่เกินค่า LBTS จากภาพประกอบ สังเกตเห็นว่า Federate #3 สามารถเลื่อนค่าจนถึงช่วงเวลาถัดไปบน

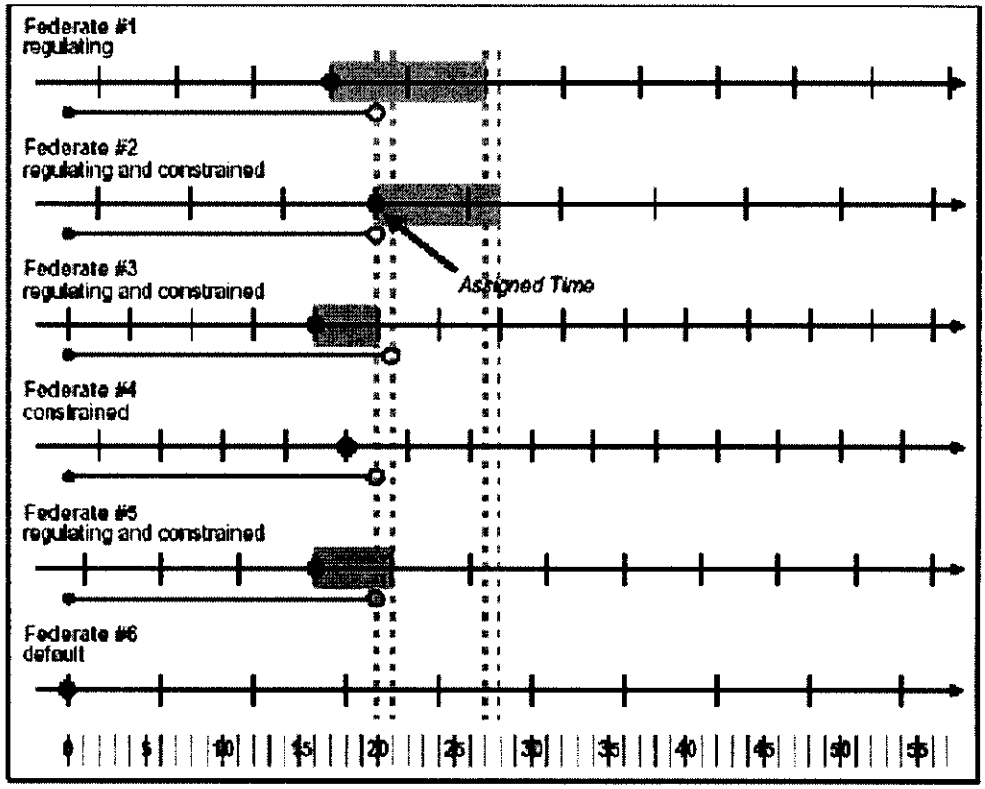
แกนเวลา เนื่องจากช่วงเวลาถัดไปยังอยู่ภายใน LBTS แต่อย่างไรก็ตาม Federate #3 ก็ไม่สามารถเปลี่ยนแปลงเวลาไปยังช่วงเวลาถัดไปได้เนื่องจากเกินค่า LBTS



ภาพประกอบ 2-8 แสดงค่า LBTS ของแบบจำลองแบบ Constrained (Defense Modeling and Simulation office, U.S. Department of Defense. 2000)

2.5.3.2 Late arriving federate

กรณีนี้เป็นกรณีมี Federate ที่เข้าร่วม Federation หลังจากเริ่มดำเนินการไปแล้ว จากภาพประกอบ 2-9 Federate #2 ยังไม่เข้าร่วม Federation ถ้า Federate #2 เข้าร่วมใน Federation โดยมีสถานะเป็นทั้ง Regulating Federate และ Constrained Federate เริ่มแรกเมื่อ Federate #2 เข้าร่วมใน Federation ค่า LBTS ของ Federate นั้นจะถูกคำนวณแล้ว ดังนั้น Federate #2 จะต้องไม่สร้าง TSO message ก่อนค่า LBTS นี้ ซึ่งจากภาพค่าดังกล่าวถูกกำหนดให้เป็น 20



ภาพประกอบ 2-9 แสดงการเข้าร่วมแบบจำลองเมื่อเริ่มดำเนินการจำลองแล้ว (Defense Modeling and Simulation office, U.S. Department of Defense. 2000)

2.6 การติดต่อกันระหว่างแบบจำลอง

การติดต่อกันระหว่างแบบจำลองภายใต้สถาปัตยกรรมชั้นสูง ในส่วนนี้จะหมายถึงรูปแบบของข้อมูลที่มีการแลกเปลี่ยนกันระหว่างแบบจำลอง ซึ่งมี 2 รูปแบบหลัก คือ ข้อมูลของวัตถุ (Object Instance) และข้อความที่ส่งถึงกัน (Interaction)

2.6.1 ข้อมูลของวัตถุ (Object Instance)

ข้อมูลที่ส่งถึงกันในส่วนนี้เป็นข้อมูลที่ใช้งานกันโดยปกติในแบบจำลองที่สร้างตามหลักของสถาปัตยกรรมชั้นสูง โดยเริ่มต้นนักพัฒนาจะต้องสร้างโครงสร้างของแบบจำลองให้อยู่ในรูปแบบของวัตถุก่อน (Object class) เนื่องจากการแลกเปลี่ยนข้อมูลในส่วนนี้จะเป็นการแลกเปลี่ยนข้อมูลลักษณะประจำของวัตถุที่เวลาต่างๆหลังจากที่มีการสร้างวัตถุนั้นๆขึ้นมาใช้งาน โดยการแลกเปลี่ยนข้อมูลจะใช้รูปแบบของผู้ผลิตข้อมูลและความสนใจในข้อมูล (Publish and Subscript) ซึ่งมีจุดเด่นในเรื่องของการเลือกแลกเปลี่ยนข้อมูลที่แบบจำลองนั้นๆต้องการได้ ทำให้ลดปริมาณ

การส่งข้อมูลในเครือข่ายโดยไม่จำเป็นและสามารถกระจายข้อมูลไปยังกลุ่มหรือแบบจำลองเป้าหมายได้พร้อมๆกัน

การรับส่งข้อมูลภายใต้สถาปัตยกรรมชั้นสูง นักพัฒนาจะต้องสร้างแฟ้มข้อมูลตามหลักการของ OMT ก่อน เรียกว่า "Fed File" ซึ่งแฟ้มข้อมูลในส่วนนี้จะนำมาใช้ในการสร้าง FedExec เมื่อดำเนินการจำลอง ภายในแฟ้มข้อมูลนี้จะมีการระบุข้อมูลที่จำเป็นในการสร้างวัตถุและข้อความที่ส่งถึงกันด้วย

สำหรับกรณีการสร้างวัตถุ ภายในแฟ้มข้อมูลจะมีการกำหนดโครงสร้างของคลาสของวัตถุและระบุรายละเอียดของแต่ละวัตถุว่ามีลักษณะประจำเป็นอย่างไรบ้าง เมื่อดำเนินการจำลองทุกๆแบบจำลองสามารถที่จะเลือกสร้างวัตถุที่มีลักษณะประจำแบบใดก็ได้และเลือกสนใจในลักษณะประจำใดๆของวัตถุนั้นๆก็ได้ตามความต้องการของนักพัฒนา

ในการดำเนินการจำลอง การเปลี่ยนแปลงข้อมูลของวัตถุจะสามารถเปลี่ยนแปลงและแจ้งให้แบบจำลองอื่นทราบ จะต้องทำตามขั้นตอนดังนี้คือ

ก. เมื่อแบบจำลองจะเชื่อมต่อกับ RTI แล้ว จากนั้นแบบจำลองแจ้งให้ RTI ทราบว่ามีข้อมูลของวัตถุใดบ้างที่สามารถเปลี่ยนแปลงข้อมูลได้ โดยการใช้คำสั่ง publicObjectClass

ข. เมื่อมีการสร้างวัตถุขึ้นมาใช้งานต้องลงทะเบียนวัตถุนั้นก่อน เพื่อให้ RTI ทราบว่ามีวัตถุใดบ้างอยู่ในระบบจำลอง โดยใช้คำสั่ง registerObjectInstance

ค. เมื่อมีการเปลี่ยนแปลงข้อมูลวัตถุ แบบจำลองต้องแจ้งให้ RTI ทราบ โดยใช้คำสั่ง updateAttributeValues

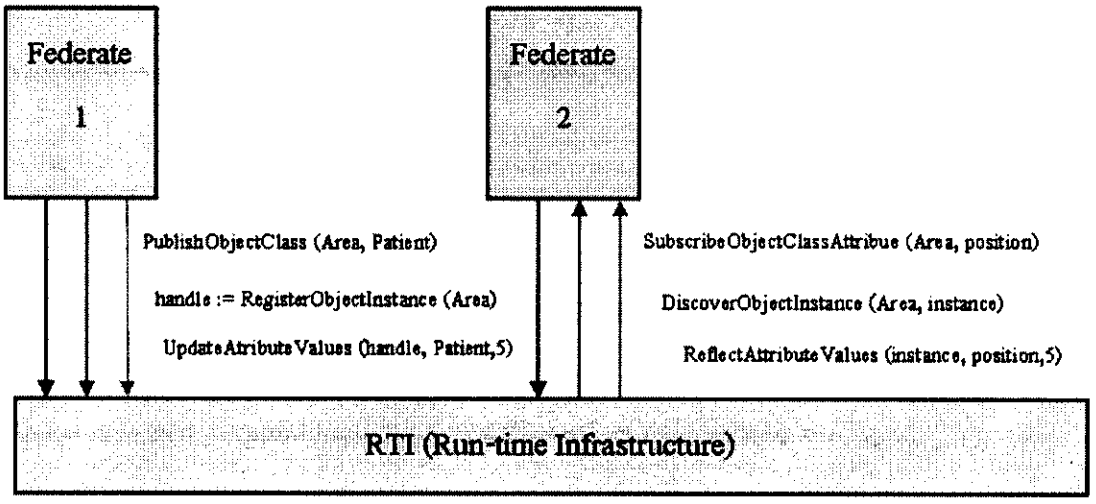
ง. และแบบจำลองจะสามารถรับทราบการเปลี่ยนแปลงข้อมูลของวัตถุที่สนใจได้ จะมีขั้นตอนการทำงานดังนี้

จ. เมื่อแบบจำลองเชื่อมต่อกับ RTI แล้ว จากนั้นแบบจำลองแจ้งให้ทราบว่าต้องการข้อมูลของวัตถุใดบ้าง โดยใช้คำสั่ง subscibeObjectClassAttribute

ฉ. เมื่อมีการลงทะเบียนของวัตถุจากแบบจำลองอื่นๆมายัง RTI และวัตถุนั้นเป็นวัตถุที่แบบจำลองสนใจ RTI จะแจ้งให้แบบจำลองทราบ ผ่านคำสั่ง discoverObjectInstance

ช. เมื่อมีการเปลี่ยนแปลงข้อมูลของวัตถุใดๆที่มีการลงทะเบียนแล้ว และเป็นวัตถุที่แบบจำลองสนใจ RTI จะแจ้งการเปลี่ยนแปลงข้อมูล,ให้ทราบ โดยผ่านคำสั่ง reflectAttributeValues

ตัวอย่างการทำงานในส่วนนี้แสดงไว้ในภาพประกอบ 2-10



ภาพประกอบ 2-10 แสดงลำดับการทำงานในการแลกเปลี่ยนข้อมูลของวัตถุผ่าน RTI

2.6.2 ข้อความที่ส่งถึงกัน (Interaction)

ข้อมูลที่ส่งถึงกันโดยวิธีนี้จะแตกต่างกับวิธีแรก เนื่องจากข้อความจะมีการส่งออกไปเมื่อเกิดเหตุการณ์ใดเหตุการณ์หนึ่ง จึงมีลักษณะการทำงานจึงแตกต่างกับข้อมูลแบบแรกซึ่งมีความสม่ำเสมอในการทำงานหรือการแจ้งการเปลี่ยนแปลงข้อมูล

การรับส่งข้อมูลแบบนี้ นักพัฒนาจะระบุรายละเอียดของข้อมูลที่ทำการส่งไว้ใน Fed File เช่นกัน โดยจะประกอบด้วย 2 ส่วนที่สำคัญคือ คลาสของข้อความที่ส่งถึงกันซึ่งมีลักษณะใกล้เคียงกับคลาสของวัตถุ แต่สมาชิกภายในคลาสแบบนี้จะเป็นพารามิเตอร์แทน

แบบจำลองที่เลือกใช้ข้อความที่ส่งถึงกันจะต้องสร้างหรือรับค่าพารามิเตอร์ทั้งหมดของข้อความนั้นด้วย ซึ่งแตกต่างกับข้อมูลของวัตถุที่สามารถเลือกเฉพาะคุณลักษณะที่ต้องการได้ ไม่จำเป็นต้องรับข้อมูลทั้งหมด

ในการดำเนินการการจำลอง แบบจำลองจะสามารถแจ้งข้อความให้ RTI ทราบและส่งต่อไปยังแบบจำลองเป้าหมายได้ จะต้องทำตามขั้นตอนดังนี้

ก. เมื่อแบบจำลองเชื่อมต่อกับ RTI แล้ว แบบจำลองจะต้องแจ้งให้ RTI ทราบว่าสามารถผลิตข้อความแบบใดได้บ้าง โดยใช้คำสั่ง publicInteractionClass

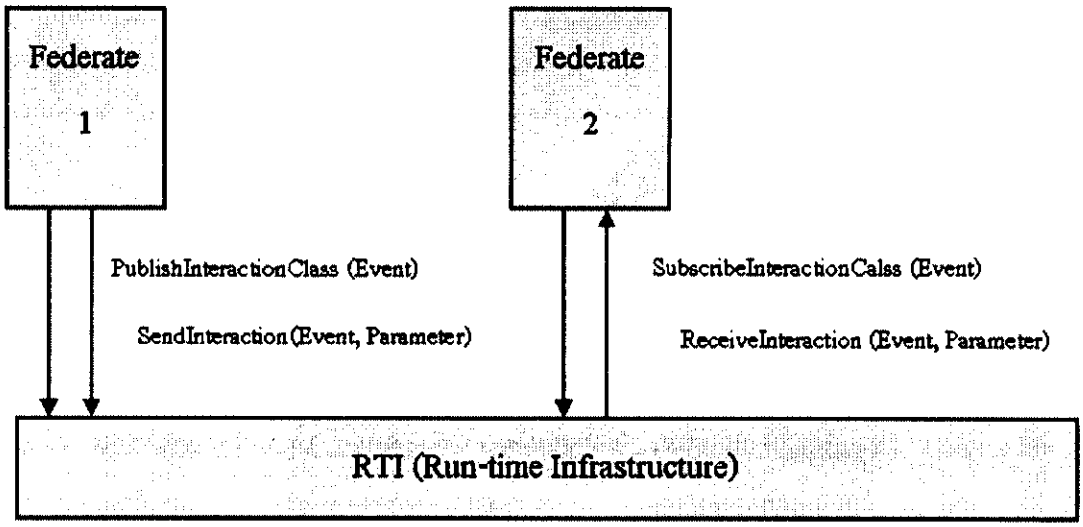
ข. เมื่อแบบจำลองต้องการส่งข้อความไปยังแบบจำลองอื่น สามารถแจ้งหรือส่งข้อความนั้นไปยัง RTI โดยใช้คำสั่ง sendInteraction

และแบบจำลองจะสามารถรับข้อความที่ถูกสร้างโดยแบบจำลองอื่นได้ จะต้องทำตามขั้นตอนดังนี้

ก. เมื่อแบบจำลองเชื่อมต่อกับ RTI แล้ว แบบจำลองจะต้องแจ้งให้ RTI ทราบว่าต้องการข้อความที่ส่งถึงกันแบบใดบ้าง โดยใช้คำสั่ง subscribeInteractionClass

ข. เมื่อมีแบบจำลองใดสร้างข้อความส่งถึงกัน และส่งมายัง RTI แล้ว RTI จะตรวจสอบว่าข้อความดังกล่าวเป็นข้อความในรูปแบบที่แบบจำลองต้องการหรือไม่ ถ้าใช่จะส่งข้อความดังกล่าวมายังแบบจำลองโดยใช้คำสั่ง receiveInteraction

ตัวอย่างการทำงานในส่วนนี้แสดงไว้ในภาพประกอบ 2-11



ภาพประกอบ 2-11 แสดงลำดับการทำงานในการส่งข้อความถึงกันผ่าน RTI

2.7 สรุป

ระบบจำลองเป็นระบบการทำงานที่เลียนแบบระบบจริงหรือระบบต้นแบบ ซึ่งปัจจุบันนิยมทำนิยมสร้างแบบจำลองโดยใช้คอมพิวเตอร์ ซึ่งใช้โปรแกรมคอมพิวเตอร์แทนแบบจำลอง ดังนั้นจึงเรียกว่าแบบจำลองคอมพิวเตอร์ และเพื่อความสะดวกและรวดเร็วในการทำงาน นักพัฒนาจึงพัฒนาให้โปรแกรมคอมพิวเตอร์สามารถแบ่งและประสานการประมวลผลการทำงานโดยใช้คอมพิวเตอร์หลายๆเครื่องที่เป็นเอกเทศต่อกันได้ จึงเกิดเทคโนโลยีการจำลองแบบกระจายศูนย์ขึ้น

เทคโนโลยีการจำลองแบบกระจายศูนย์ที่ใช้สำหรับการวิจัยนี้คือ สถาปัตยกรรมชั้นสูง ซึ่งเป็นเทคโนโลยีที่พัฒนาโดยกระทรวงกลาโหม ประเทศสหรัฐอเมริกา โดยเทคโนโลยีนี้สามารถสนับสนุนการทำงานของเทคโนโลยีก่อนหน้านี้ได้ ดังนั้นจึงสามารถนำแบบจำลองที่มีอยู่แล้วกลับมาใช้งานใหม่ได้ รวมทั้งเทคโนโลยีนี้ไม่ขึ้นอยู่กับระบบปฏิบัติการหรือแพลตฟอร์ม ดังนั้นจึงสะดวกในการพัฒนาแบบจำลอง

โครงสร้างของเทคโนโลยีสถาปัตยกรรมชั้นสูงจะมีองค์ประกอบที่สำคัญ 3 ส่วนได้แก่ HLA Rules, HLA Interface Specification และ OMT โดยมีสื่อกลางหรือโปรแกรมกลางที่ทำหน้าที่ตรวจสอบองค์ประกอบทั้งสามส่วนของแบบจำลองชื่อว่า Run-time Infrastructure (RTI) แบบจำลองที่พัฒนาขึ้นมาทั้งหมดภายใต้สถาปัตยกรรมชั้นสูงจะเชื่อมต่อกันผ่านโปรแกรม RTI ซึ่งจะให้บริการที่ใช้สำหรับการเชื่อมต่อแบบจำลองเข้าด้วยกัน รวมทั้งสนับสนุนการรับส่งข้อความแบบต่างๆเพื่อแก้ไขหรือรายงานสถานะข้อมูลระหว่างแบบจำลองและสนับสนุนการจัดการกับเวลาในรูปแบบที่แตกต่างกันอีกด้วย