การพัฒนาบรรณาธิกรณ์สำหรับกำหนดพฤติกรรมของวัตถุพร้อมทำงานแบบวิชวล

นางสาว จันทร์พร ลักษพร

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมศาสตร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2542

ISBN 974-333-636-2

A DEVELOPMENT OF AN EDITOR FOR VISUALLY SPECIFYING THE BEHAVIOR OF ACTIVE OBJECTS

MISS CHANPORN LAPPAYAPORN

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science in Computer Science

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic Year 1999

ISBN 974-333-636-2

หัวข้อวิทยานิพนธ์	การพัฒนาบรรณาธิกรณ์สำหรับกำหนดพฤติกรรมของวัตถุพร้อมทำงานแบบวิชวล
โดย	นางสาวจันทร์พร ลัภยพร
ภาควิชา	วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษา	อาจารย์ ดร.พรศิริ หมื่นไชยศรี

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

.....คณบดีคณะวิศวกรรมศาสตร์
(รองศาสตราจารย์ ดร.ธัชชัย สุมิตร)

คณะกรรมการสอบวิทยานิพนธ์

.....ประธานกรรมการ
(รองศาสตราจารย์ ดร.วันชัย ริวไพบูลย์)

.....อาจารย์ที่ปรึกษา
(อาจารย์ ดร.พรศิริ หมื่นไชยศรี)

.....กรรมการ
(อาจารย์ ดร.ธาราทิพย์ สุวรรณศาสตร์)

.....กรรมการ
(อาจารย์ วิวัฒน์ วัฒนาวุฒิ)

จันทร์พร ลัภยพร : การพัฒนาบรรณาธิกรณ์สำหรับกำหนดพฤติกรรมของวัตถุพร้อมทำงานแบบวิชวล (A DEVELOPMENT OF AN EDITOR FOR VISUALLY SPECIFYING THE BEHAVIOR OF ACTIVE OBJECTS) อาจารย์ที่ปรึกษา : อาจารย์ ดร. พรศิริ หมื่นไชยศรี. 106 หน้า.
ISBN 974-333-636-2

วิทยานิพนธ์นี้ มีวัตถุประสงค์เพื่อพัฒนาบรรณาธิกรณ์สำหรับกำหนดพฤติกรรมของวัตถุพร้อมทำงานให้ผู้ใช้งานสามารถกำหนดพฤติกรรมของวัตถุพร้อมทำงานผ่านการติดต่อกับผู้ใช้งานแบบกราฟิก และสามารถสร้างโครงร่างของโปรแกรมเป็นชุดคำสั่งภาษาจาวาที่รองรับกลไกของพฤติกรรมตามที่กำหนดได้โดยอัตโนมัติ พฤติกรรมของวัตถุพร้อมทำงานที่กำหนดได้จากบรรณาธิกรณ์ได้แก่ กฎการเปลี่ยนแปลง สมการกำหนดค่า และ การกระทำตามเหตุการณ์

บรรณาธิกรณ์ที่พัฒนาขึ้นนี้จะถูกใช้งานเป็นส่วนหนึ่งในสิ่งแวดล้อมสำหรับการพัฒนาโปรแกรมด้วยแผนภาพเอนทิตีและความสัมพันธ์ซึ่งเป็นเครื่องมือที่ใช้แนวความคิดใหม่ในการเชื่อมต่อองค์ประกอบต่างๆเข้าด้วยกัน โดยใช้แผนภาพเอนทิตีและความสัมพันธ์เป็นแนวทางในการเชื่อมต่อองค์ประกอบ โปรแกรมจะสามารถประมวลผลได้ทันทีที่เชื่อมต่อเอนทิตีต่างๆเข้าด้วยกันเนื่องจากเอนทิตีแต่ละตัวมีคุณสมบัติเป็นวัตถุพร้อมทำงาน เมื่อเพิ่มเติมความสามารถในการกำหนดพฤติกรรมของวัตถุพร้อมทำงานเหล่านี้ จะเป็นการช่วยอำนวยความสะดวกในการสร้างโปรแกรมในขอบเขตเรื่องราวใหม่ๆ และทำให้มีชุดคำสั่งที่ต้องเขียนด้วยตัวเองน้อยลง

เมื่อทดลองกำหนดพฤติกรรมให้กับชนิดของเอนทิตีต่างๆในโปรแกรม 3 โปรแกรมได้แก่ ระบบแถวคอย ระบบแท็งค์ และระบบเครือข่าย แล้วทำการสร้างชุดคำสั่งอัตโนมัติ ผลปรากฏว่าระบบสามารถสร้างคำสั่งที่รองรับกลไกของพฤติกรรมแบบต่างๆตามที่กำหนดได้อย่างถูกต้อง โดยมีร้อยละของจำนวนบรรทัดคำสั่งที่สร้างได้โดยอัตโนมัติเทียบกับจำนวนบรรทัดคำสั่งในโปรแกรมที่ใช้งานได้จริงของระบบแถวคอยเท่ากับ 76.0 ระบบแท็งค์เท่ากับ 52.7 และระบบเครือข่ายเท่ากับ 48.7

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....
สาขาวิชา วิทยาศาสตร์คอมพิวเตอร์ ลายมือชื่ออาจารย์ที่ปรึกษา .....
ปีการศึกษา 2542 ลายมือชื่ออาจารย์ที่ปรึกษาร่วม .....

##4170250221 : MAJOR COMPUTER SCIENCE
KEY WORD : ACTIVE OBJECT / SOURCE CODE GENERATION / VISUAL TOOL
CHANPORN LAPPAYAPORN : A DEVELOPMENT OF AN EDITOR FOR VISUALLY SPECIFYING THE BEHAVIOR OF ACTIVE OBJECTS. THESIS ADVISOR : DR. PORNSIRI MUENCHAISRI. 106 pp. ISBN 974-333-636-2

This thesis aimed to develop an editor for visually specifying the behavior of active objects. Users can specify the behavior of active objects via graphic user interface and generate a set of Java source code that enables the specified behavior. The behavior supported by this editor are transition rules, equational assignments, and event routines.

This editor was merged with The *Entity-Relationship Software Development Environment (ERSDE)*, a visual tool which demonstrated a new software composition approach. With ERSDE, an application can be constructed by using an *extended entity-relationship diagram (EERD)* as a connection guideline in a specific domain. The application is executed immediately once the entities (components) are interconnected since each entity is implemented as an active object. With the enhancement of ERSDE capability for specifying the behavior of the active object and behavioral code generation, this tool will facilitate user to build various domains of applications with less manual coding time and effort.

The editor was tested by regenerating the skeleton code for all entities in three applications, the queuing system, the tank system and the local area network system. The active behavioral code was created correctly as specified in the behavior editor. The percentage of line of code between the regenerated code and the executable code for queuing system is 76.0 , for the tank system is 52.7 and for the LAN system is 48.7.

ภาควิชา วิศวกรรมคอมพิวเตอร์ ..... ลายมือชื่อนิสิต .....
สาขาวิชา วิทยาศาสตร์คอมพิวเตอร์ ..... ลายมือชื่ออาจารย์ที่ปรึกษา .....
ปีการศึกษา 2542 ..... ลายมือชื่ออาจารย์ที่ปรึกษาร่วม .....

กิตติกรรมประกาศ

ข้าพเจ้าใคร่ขอกราบขอบพระคุณอาจารย์ ดร. พรศิริ หมื่นไชยศรี อาจารย์ที่ปรึกษาวิทยานิพนธ์ของข้าพเจ้า ที่ท่านเป็นผู้แนะนำให้ความรู้ คำปรึกษา ความช่วยเหลือต่างๆ ตลอดจนคอยดูแลการทำวิจัยของข้าพเจ้าอย่างดียิ่งจนสำเร็จลุล่วงลงได้ด้วยดี

ขอกราบขอบพระคุณอาจารย์ในห้องปฏิบัติบัติการวิศวกรรมซอฟต์แวร์ รองศาตราจารย์ ดร. วันชัย ริ้วไพบูลย์ อาจารย์ ดร. ธาราทิพย์ สุวรรณศาสตร์ และ อาจารย์ วิวัฒน์ วัฒนาวุฒิ ที่ได้ให้คำแนะนำ และข้อคิดเห็นต่างๆในการทำวิทยานิพนธ์ ซึ่งข้าพเจ้ามิอาจลืมเลือนได้ อนึ่งช่วงเวลาที่ข้าพเจ้าทำวิทยานิพนธ์ในห้องปฏิบัติการ ข้าพเจ้าได้รับความสะดวกสบายจากสิ่งแวดล้อมรวมทั้งเครื่องมือต่างๆ ซึ่งทำให้ข้าพเจ้าทำงานได้อย่างมีประสิทธิภาพ

และขอขอบคุณบรรดาเพื่อนๆที่ได้ให้คำแนะนำ และความช่วยเหลือแก่ข้าพเจ้าตลอดเวลาที่ศึกษาในภาควิชาวิศวกรรมคอมพิวเตอร์แห่งนี้ ขอขอบคุณ คุณโชคชัย สุทธิธรรมจิต ที่เอื้อเฟื้ออนุเคราะห์เครื่องคอมพิวเตอร์ที่ใช้ในการทำวิจัยครั้งนี้

ท้ายที่สุด ข้าพเจ้าใคร่ขอกราบขอบพระคุณบิดา มารดา และพี่ น้อง ที่ได้ให้โอกาสและสนับสนุนในด้านการเงินและกำลังใจแก่ข้าพเจ้าเสมอมา

จันทร์พร ลัภยพร

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

หน้า

สารบัญรูป

หน้า

สารบัญรูป (ต่อ)

หน้า

สารบัญรูป (ต่อ)

หน้า

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

การเขียนโปรแกรมเชิงวัตถุ (object-oriented programming) กำลังมีบทบาทที่สำคัญต่อการเปลี่ยนแปลงแนวทางในการพัฒนาซอฟต์แวร์ คุณลักษณะต่างๆของการเขียนโปรแกรมเชิงวัตถุ ไม่ว่าจะเป็นการสืบทอดคุณสมบัติ (inheritance) เอ็นแคปซูเลชั่น (encapsulation) และ โพลีมอร์ฟิซึม (polymorphism) ล้วนเป็นปัจจัยส่งเสริมให้เกิดการนำองค์ประกอบซอฟต์แวร์ต่างๆมาใช้ใหม่ (reuse) ผู้พัฒนาซอฟต์แวร์สามารถเลือกใช้องค์ประกอบย่อยๆที่มีผู้พัฒนาไว้แล้วและได้รับการทดสอบอย่างดีมาใช้โดยไม่ต้องออกแบบและพัฒนาตั้งแต่เริ่มแรกด้วยตัวเอง ทำให้สามารถลดระยะเวลาที่ใช้ในการพัฒนาได้มาก

Toshimi Minoura และ Sungwoon Choi ได้เสนอ ระบบวัตถุพร้อมทำงานแบบโครงสร้าง (Structural Active Object System :SAOS) [1] ซึ่งเป็นแนวความคิดใหม่สำหรับการพัฒนาระบบเชิงวัตถุที่ประกอบด้วยกลุ่มของวัตถุที่ทำงานพร้อมๆกัน (concurrent) ในระบบดังกล่าว วัตถุแต่ละตัวจะต้องมีคุณสมบัติเป็นวัตถุพร้อมทำงาน กล่าวคือสามารถริเริ่มการกระทำใดๆได้ด้วยตัวเอง โดยมิต้องรอคำขอ (message) จากวัตถุอื่นๆ ทำให้วัตถุเหล่านี้มีการควบคุมในตัวเอง สามารถนำไปใช้ใหม่ได้ดีขึ้นและเข้าใจได้ง่ายขึ้น ผลของการวิจัยนี้ทำให้เกิดโครงร่าง (framework) สำหรับพัฒนาโปรแกรมประยุกต์ที่ประกอบด้วยวัตถุพร้อมทำงานที่ทำงานพร้อมๆกัน เช่น ระบบการจำลองเหตุการณ์ ระบบติดต่อกับผู้ใช้แบบ การจำลองโปรโตคอลของระบบเครือข่าย การจำลองการทำงานของอัลกอริทึม ในการพัฒนาโปรแกรมเหล่านี้ ผู้พัฒนาระบบไม่จำเป็นต้องเขียนส่วนควบคุมของระบบทั้งหมดด้วยตัวเอง สามารถนำคลาสต่างๆที่ได้ออกแบบและเขียนไว้แล้วไปใช้ เพียงแต่เพิ่มเติมคำสั่งต่างๆเกี่ยวกับพฤติกรรมและวิธีการทำงานของวัตถุพร้อมทำงานแต่ละตัวเท่านั้น ส่งผลให้โปรแกรมประยุกต์มีขนาดของชุดคำสั่งที่เล็กและสามารถพัฒนาได้ในระยะเวลาอันสั้น

ต่อมา พรศิริ หมื่นไชยศรี ได้เสนอวิธีการเชื่อมองค์ประกอบเข้าด้วยกันโดยการใช้แผนภาพเอนทิตีและความสัมพันธ์ (Extended Entity-Relationship Diagram : EERD) เป็นแผ่นแบบ(template) และรายการเลือก (menu) สำหรับสร้างโปรแกรมประยุกต์ ในงานวิจัยดังกล่าว ผู้วิจัยได้พัฒนาเครื่องมือชื่อว่า "สิ่งแวดล้อมสำหรับพัฒนาโปรแกรมด้วยแผนภาพเอนทิตีและความสัมพันธ์" (Entity-Relationship Software Development Environment : ERSDE) [2] ซึ่งเป็นเครื่องมือเพื่อให้ผู้ใช้งานสามารถสร้างโปรแกรมประยุกต์สำหรับแต่ละขอบเขตในเรื่องราว (application domain) ที่ต้องการ มีการใช้แผนภาพเอนทิตีและความสัมพันธ์เพื่อแสดงชนิดของเอนทิตีและความสัมพันธ์ระหว่างชนิดของเอนทิตีเหล่านั้น ผู้ใช้สามารถสร้างเอนทิตีต่างๆโดยการเลือกจากชนิดของเอนทิตีที่แสดงในแผนภาพ หลังจากนั้นจึงเชื่อมต่อเอนทิตีเข้าด้วยกันให้สอดคล้องกับความสัมพันธ์ที่แสดงในแผนภาพที่ได้กำหนดไว้แล้ว โปรแกรมจะสามารถประมวลผลได้ทันทีและเป็นไปในลักษณะเชิงโต้ตอบ

ได้ (interactive) จุดเด่นและคุณประโยชน์ที่สำคัญของงานวิจัยนี้คือ ผู้ใช้งานสามารถเชื่อมต่อองค์ประกอบโดยมีแผนภาพเอนทิตีและความสัมพันธ์เป็นแนวทางในการเชื่อมต่อ มิต้องลองผิดลองถูกว่าแต่ละองค์ประกอบจะสามารถทำงานเข้ากันได้หรือไม่ เมื่อทำการเชื่อมองค์ประกอบเหล่านี้เข้าด้วยกันแล้ว ระบบสามารถแสดงผลการทำงานได้ทันที

ระบบวัตถุพร้อมทำงานและสิ่งแวดล้อมสำหรับพัฒนาโปรแกรมด้วยแผนภาพเอนทิตีและความสัมพันธ์เป็นระบบที่ช่วยอำนวยความสะดวกสำหรับผู้พัฒนา สนับสนุนแนวความคิดเกี่ยวกับองค์ประกอบซอฟต์แวร์และการนำกลับมาใช้ใหม่ ทำให้สามารถลดระยะเวลาในการพัฒนาโปรแกรมประยุกต์ได้เป็นอย่างมาก อย่างไรก็ตามระบบนี้ยังไม่ถูกนำไปใช้อย่างกว้างขวาง เนื่องจากเป็นแนวความคิดใหม่ในการเขียนโปรแกรม ผู้พัฒนาจะต้องศึกษาทำความเข้าใจเพื่อให้ทราบถึงขั้นตอนและคำสั่งต่างๆที่ต้องใช้เพื่อกำหนดพฤติกรรมของวัตถุพร้อมทำงานและให้วัตถุเหล่านั้นสามารถทำงานประสานกับโครงร่างที่ออกแบบไว้ได้ ทำให้ผู้พัฒนาระบบที่ไม่มีประสบการณ์หรือไม่ได้ศึกษาอย่างลึกซึ้งไม่สามารถนำแนวความคิดดังกล่าวไปใช้ได้อย่างมีประสิทธิภาพ ถึงแม้ว่าเครื่องมือที่พัฒนาขึ้นในงานวิจัยของ พรศิริ หมื่นไชยศรี จะสามารถอำนวยความสะดวกให้กับผู้ใช้งานโดยสามารถสร้างชุดคำสั่งได้ในบางส่วน แต่ยังไม่สามารถสร้างชุดคำสั่งที่เกี่ยวข้องกับพฤติกรรมของวัตถุได้ ดังนั้นหลังจากที่ชุดคำสั่งได้ถูกสร้างขึ้นแล้ว ผู้ใช้งานยังต้องเพิ่มเติมชุดคำสั่งด้วยตัวเองอีกมาก โดยเฉพาะคำสั่งที่เกี่ยวข้องกับการทำให้แต่ละเอนทิตีมีคุณสมบัติเป็นวัตถุที่พร้อมทำงานและคำสั่งเกี่ยวกับพฤติกรรมของวัตถุอันเป็นหัวใจที่จะทำให้ระบบสามารถทำงานได้

ดังนั้นวิทยานิพนธ์นี้จึงมีความมุ่งหมายเพื่อแก้ไขปรับปรุงสิ่งแวดล้อมสำหรับพัฒนาโปรแกรมด้วยแผนภาพเอนทิตีและความสัมพันธ์ เพื่อให้มีความสามารถเพิ่มเติมในการสร้างชุดคำสั่งเกี่ยวกับพฤติกรรมของวัตถุพร้อมทำงาน โดยพัฒนาบรรณาธิกรณ์สำหรับให้ผู้พัฒนาโปรแกรมประยุกต์กำหนดพฤติกรรมของวัตถุผ่านการติดต่อกับผู้ใช้งานแบบกราฟิก (Graphic User Interface) พร้อมทั้งปรับปรุงหน่วยสร้างชุดคำสั่งอัตโนมัติซึ่งจะรับผิดชอบการสร้างคำสั่งต่างๆตามกฎเกณฑ์และสร้างฟังก์ชันที่จำเป็น ทำให้ผู้พัฒนาไม่ต้องจดจำกฎเกณฑ์และรูปแบบของฟังก์ชันที่ซับซ้อน สามารถใช้เวลากับคำสั่งในส่วนอื่นๆได้มากขึ้น และผู้พัฒนาที่เริ่มศึกษาใหม่ไม่ต้องเสียเวลาศึกษากฎเกณฑ์ต่างๆเหล่านี้มากนัก เมื่อนำมาใช้กับระบบบรรณาธิกรณ์อื่นๆที่มีอยู่แล้วในสิ่งแวดล้อมสำหรับพัฒนาโปรแกรมด้วยแผนภาพเอนทิตีและความสัมพันธ์จะทำให้สามารถอำนวยความสะดวกในการสร้างโปรแกรม มีส่วนที่ต้องเขียนคำสั่งด้วยตัวเองน้อยลง อันจะส่งผลให้สามารถพัฒนาโปรแกรมประยุกต์ได้ง่ายและรวดเร็ว

1.2 วัตถุประสงค์

พัฒนาบรรณาธิกรณ์สำหรับกำหนดพฤติกรรมของวัตถุพร้อมทำงานผ่านการติดต่อกับผู้ใช้งานแบบกราฟิกให้สามารถสร้างชุดคำสั่งพฤติกรรมของวัตถุพร้อมทำงานได้โดยอัตโนมัติ

1.3 ขอบเขตของงานวิจัย

1) พัฒนาบรรณาธิกรณ์สำหรับกำหนดพฤติกรรมของวัตถุพร้อมทำงานโดยมีคุณลักษณะดังนี้
   - สามารถกำหนดพฤติกรรมครอบคลุมประโยคการเปลี่ยนแปลงทั้งสามแบบได้แก่ กฎการเปลี่ยนแปลง สมการคำหนดค่า และ การกระทำตามเหตุการณ์
   - แสดงเมธอด และ แอตทริบิวต์ของแต่ละชนิดของเอนทิตีที่ได้กำหนดไว้แล้วจากบรรณาธิกรณ์สำหรับสร้างชนิดของเอนทิตี
   - สามารถจัดเก็บและเรียกใช้ข้อมูลเกี่ยวกับพฤติกรรมที่กำหนดไว้แล้วได้
2) แก้ไขบรรณาธิกรณ์สำหรับสร้างเค้าร่างในสิ่งแวดล้อมสำหรับพัฒนาโปรแกรมด้วยแผนภาพเอนทิตีและความสัมพันธ์ ให้มีความสามารถเพิ่มเติมดังนี้
   - สามารถเรียกบรรณาธิกรณ์สำหรับกำหนดพฤติกรรมเพื่อกำหนดพฤติกรรมของชนิดของเอนทิตีแต่ละตัวได้
   - แก้ไขส่วนของการสร้างชุดคำสั่งอัตโนมัติให้สามารถสร้างชุดคำสั่งเกี่ยวกับพฤติกรรมที่กำหนดไว้ในบรรณาธิกรณ์เพิ่มเติมจากความสามารถเดิมในการสร้างโครงร่างชุดคำสั่ง
3) ชุดคำสั่งที่สร้างจะจำกัดเฉพาะพฤติกรรมที่แสดงได้ด้วยประโยคการเปลี่ยนแปลงทั้งสามแบบดังกล่าวเท่านั้น ถ้าต้องการกำหนดพฤติกรรมอื่นๆผู้พัฒนาต้องเขียนโปรแกรมเพิ่มเติมด้วยตัวเองหลังจากที่สร้างชุดคำสั่งอัตโนมัติแล้ว
4) การทดสอบโปรแกรมจะทำโดยทดลองสร้างโปรแกรมประยุกต์ 3 ระบบได้แก่ ระบบแถวคอย ระบบแท็งค์ และระบบเครือข่าย
5) ซอฟต์แวร์ที่ใช้ในการพัฒนาได้แก่ จาวา ดีเวลลอปเมนต์ ทูลคิด (Java Development Toolkit)
6) ระบบที่พัฒนาขึ้นจะถูกทดสอบและทำงานภายใต้ระบบปฏิบัติการวินโดวส์

1.4 ขั้นตอนและวิธีการดำเนินงาน

1) ศึกษาแนวคิดและการใช้งานระบบวัตถุพร้อมทำงานและการเชื่อมโยงองค์ประกอบด้วยแผนภาพเอนทิตีและความสัมพันธ์
2) ศึกษาและรวบรวมรูปแบบพฤติกรรมของวัตถุพร้อมทำงาน
3) ศึกษาการพัฒนาซอฟต์แวร์เชิงวัตถุ ภาษาจาวา
4) ออกแบบบรรณาธิกรณ์สำหรับพฤติกรรมของวัตถุพร้อมทำงานและการสร้างชุดคำสั่งอัตโนมัติ
5) พัฒนาโปรแกรม
6) ทดสอบโปรแกรม
7) สรุปผล และจัดทำวิทยานิพนธ์

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1) ช่วยลดเวลาในการเขียนชุดคำสั่งด้วยตัวเอง
เมื่อเพิ่มความสามารถในการสร้างชุดคำสั่งที่เกี่ยวกับพฤติกรรมให้กับหน่วยสร้างชุดคำสั่งในสิ่งแวดล้อมสำหรับพัฒนาโปรแกรมด้วยแผนภาพเอนทิตีและความสัมพันธ์ ทำให้สามารถสร้างชุดคำสั่งได้มากขึ้นขึ้น ผู้พัฒนาไม่ต้องเสียเวลากับชุดคำสั่งดังกล่าวซึ่งสร้างได้โดยอัตโนมัติ สามารถให้เวลากับสั่งในรายละเอียดอื่นๆได้มากขึ้น

2) ลดข้อผิดพลาดในการเขียนโปรแกรม
หน่วยสร้างชุดคำสั่งจะเป็นผู้ควบคุมการสร้างคำสั่งต่างๆที่จำเป็นเพื่อให้วัตถุมีคุณสมบัติเป็นวัตถุพร้อมทำงาน ซึ่งถ้าผู้พัฒนาต้องเขียนคำสั่งเหล่านี้ด้วยตัวเองจะต้องอาศัยความรู้หรือประสบการณ์เพื่อเขียนคำสั่งเหล่านี้ให้ครบถ้วนและยังอาจเกิดการหลงลืมในบางครั้ง เช่นผู้พัฒนาอาจลืมเขียนคำสั่งเพื่อเพิ่มกลไกการเริ่มทำงานให้กับตัวแปรพร้อมทำงาน ทำให้วัตถุไม่สามารถแสดงพฤติกรรมดังที่ต้องการ การกำหนดดัชนีฟังก์ชันอาจเกิดการผิดพลาดทำให้เรียกฟังก์ชันผิดเป็นต้น เมื่อเกิดข้อผิดพลาดเหล่านี้ผู้พัฒนาจะต้องเสียเวลาเพื่อทำการหาสาเหตุ ดังนั้น การที่ชุดคำสั่งส่วนที่มีความซับซ้อนถูกสร้างขึ้นได้โดยอัตโนมัติ คำสั่งต่างๆที่ใช้เพื่อรองรับพฤติกรรมที่กำหนดจะถูกสร้างขึ้นอย่างครบถ้วน ทำให้มีข้อผิดพลาดในโปรแกรมน้อยลง

3) ผู้พัฒนาสามารถพัฒนาโปรแกรมประยุกต์ในขอบเขตเรื่องราวใหม่ๆได้ง่ายขึ้น
ผู้พัฒนากำหนดพฤติกรรมของวัตถุผ่านบรรณาธิกรณ์แบบวิชวล สามารถแสดงข้อมูลต่างๆที่จำเป็นสำหรับการกำหนดพฤติกรรมเช่น เบราว์เซอร์ที่แสดงรายการฟังก์ชันและข้อมูลสมาชิกที่ได้กำหนดไว้แล้ว เมื่อต้องการเลือกฟังก์ชันใดก็สามารถทำได้โดยการใช้เมาส์คลิก ไม่ต้องคีย์ชื่อเหล่านั้นดังเช่นการทำงานแบบตัวอักษร นอกจากนี้ผู้พัฒนาไม่ต้องจำกฎเกณฑ์หรือรูปแบบของฟังก์ชันที่มีความซับซ้อน สามารถกำหนดพฤติกรรมผ่านหน้าจอที่ออกแบบไว้ให้ใช้งานได้ง่าย ทำให้สามารถเขียนโปรแกรมได้อย่างสะดวกรวดเร็ว

บทที่ 2

แนวคิดและทฤษฎีที่เกี่ยวข้อง

ในบทนี้จะกล่าวถึงรายละเอียดของงานวิจัยต่างๆที่เกี่ยวข้องกับระบบวัตถุพร้อมทำงาน เริ่มจากระบบวัตถุพร้อมทำงานแบบโครงสร้างโดยอธิบายแนวความคิดและการกำหนดพฤติกรรมของวัตถุพร้อมทำงาน จากนั้นจะอธิบายงานวิจัยเกี่ยวกับตัวแปลภาษาบรรยายระบบวัตถุพร้อมทำงานแบบโครงสร้าง และระบบวัตถุพร้อมทำงานในภาษาจาวา นอกจากนี้ยังได้แสดงแนวความคิดและองค์ประกอบของสิ่งแวดล้อมสำหรับพัฒนาโปรแกรมด้วยแผนภาพเอนทิตีและความสัมพันธ์ และท้ายที่สุดได้อธิบายถึงแบบจำลองเอ็มวีซีซึ่งเป็นแบบจำลองที่ใช้กันอย่างแพร่หลายในการเขียนโปรแกรมเชิงวัตถุ

2.1 ระบบวัตถุพร้อมทำงานแบบโครงสร้าง (Structural Active Object System :SAOS)

แนวความคิดระบบวัตถุพร้อมทำงานแบบโครงสร้าง [1] ถูกเสนอขึ้นครั้งแรกโดย Toshimi Minoura และ Sungwoon Choi SAOS หมายถึง ระบบที่ประกอบด้วยวัตถุพร้อมทำงานต่างๆที่ถูกนำมารวมกันโดยวิธีการประกอบวัตถุแบบโครงสร้างและลำดับชั้น (Structural & Hierarchical Object Composition (SHOC)) วัตถุพร้อมทำงานคือวัตถุประเภทหนึ่งที่แตกต่างจากวัตถุทั่วๆไปในการเขียนโปรแกรมเชิงวัตถุ โดยทั่วไปแล้ววัตถุต่างๆจะถือว่าเป็นวัตถุที่คอยตอบสนองต่อคำร้องขอ (passive object) นั่นคือจะทำงานได้ก็ต่อเมื่อมีวัตถุอื่นๆสั่งให้ทำงานโดยการส่งคำร้องขอ (message) มาให้ แต่วัตถุพร้อมทำงานจะสามารถทำงานได้ด้วยตัวเองโดยไม่ต้องมีการรับคำขอจากวัตถุตัวอื่นๆ การเริ่มทำงานของวัตถุพร้อมทำงานนี้จะขึ้นอยู่กับการเปลี่ยนแปลงค่าของตัวแปรในวัตถุนั้นๆ ในโลกของความเป็นจริงมีวัตถุมากมายที่ทำงานแบบวัตถุพร้อมทำงาน แต่เมื่อนำมาเขียนโปรแกรมเชิงวัตถุที่คอยตอบสนองต่อคำขอจะทำให้มีข้อบกพร่อง กล่าวคือวัตถุนั้นๆจะไม่มีส่วนควบคุมในตัวเอง(encapsulation of control) ต้องมีการสร้างส่วนควบคุมแยกออกไปต่างหาก จึงไม่สะดวกสำหรับการนำวัตถุนั้นๆใช้ใหม่ในงานอื่นๆ

ตัวอย่างเช่น ในปัจจุบันนี้สัญญาณไฟจราจรบางชนิดที่ใช้ตัวจับเวลา จะต้องมีตัวจับเวลาในตัวเองเพื่อคอยจับเวลาที่จะต้องเปลี่ยนเป็นสี แดง เหลือง เขียวตามเวลาที่กำหนดไว้ ถ้าต้องการเขียนโปรแกรมเพื่อจำลองการทำงานของสัญญาณไฟนี้ในรูปแบบวัตถุที่คอยตอบสนองต่อคำร้องขอ นอกจากจะต้องมีสัญญาณไฟแล้ว จะต้องสร้างวัตถุอีกตัวหนึ่งขึ้นมาเพื่อคอยส่งคำขอให้สัญญาณไฟเปลี่ยนสีตามเวลา แต่ถ้านำมาเขียนโปรแกรมแบบวัตถุพร้อมทำงานจะสามารถสร้างสัญญาณไฟที่เหมือนสัญญาณไฟในโลกความเป็นจริง นั่นคือการควบคุมการเปลี่ยนสัญญาณไฟจะอยู่ภายในตัววัตถุที่เป็นสัญญาณไฟ และสามารถทำงานได้เองโดยไม่ต้องรอคำร้องขอจากวัตถุอื่น

2.1.1 พฤติกรรมของวัตถุพร้อมทำงาน

การกำหนดพฤติกรรมของวัตถุพร้อมทำงานสามารถกำหนดได้โดยใช้ประโยคแสดงการเปลี่ยนแปลง (transition statement) ซึ่งแบ่งออกได้เป็น 3 แบบได้แก่

2.1.1.1 กฎการเปลี่ยนแปลง (transition rules)

การกำหนดพฤติกรรมแบบนี้จะต้องมีการกำหนดเงื่อนไข และส่วนของคำสั่งที่ต้องถูกประมวลผลถ้าเงื่อนไขเป็นจริงโดยใช้ตัวแปรพร้อมทำงาน (active variable) ซึ่งเมื่อตัวแปรเหล่านี้มีการเปลี่ยนแปลงค่า กฎการเปลี่ยนแปลงจะถูกกระตุ้น เงื่อนไขที่กำหนดไว้จะถูกตรวจสอบ และทำการประมวลผลคำสั่งถ้าเงื่อนไขเป็นจริง

ตัวอย่างของกฎการเปลี่ยนแปลงแสดงได้จากพฤติกรรมของตัวสร้างงานซึ่งเป็นองค์ประกอบหนึ่งในระบบแถวคอย ตัวสร้างงานมีหน้าที่ป้อนงานให้เข้าสู่แถวคอยตามเวลาที่กำหนด ดังนั้นจึงสามารถกำหนดเงื่อนไขและการกระทำของตัวสร้างงานได้ดังนี้

เงื่อนไข : เมื่อสถานะของตนเองเป็น "พร้อม"

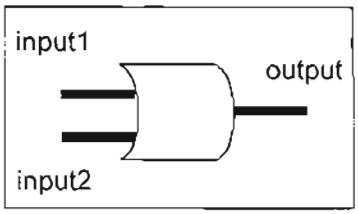
การกระทำ : ป้อนงานสู่แถวคอย

จากตัวอย่างนี้เมื่อสถานะของตัวสร้างงานหรือจำนวนงานในแถวคอยมีการเปลี่ยนแปลงระบบจะตรวจสอบเงื่อนไขที่กำหนดไว้ ถ้าเงื่อนไขเป็นจริงการกระทำ "ป้อนงานสู่แถวคอย" จะถูกประมวลผล

2.1.1.2 สมการกำหนดค่า (equational assignment)

การกำหนดพฤติกรรมแบบนี้คล้ายกับกฎการเปลี่ยนแปลงแต่จะอยู่ในรูปสมการกำหนดค่า ประกอบด้วยส่วนด้านซ้ายของเครื่องหมาย "=" และส่วนทางด้านขวาซึ่งจะต้องประกอบด้วยตัวแปรพร้อมทำงานอย่างน้อยหนึ่งตัว เมื่อไรก็ตามที่ตัวแปรพร้อมทำงานในสมการมีการเปลี่ยนแปลงค่า ระบบจะทำการคำนวณค่าในส่วนด้านขวา และกำหนดให้ค่าทางด้านซ้ายมีค่าเท่ากับผลจากการคำนวณนั้น

ตัวอย่างของสมการกำหนดค่าแสดงได้จากพฤติกรรมของแอนเกต (AND gate) ดังแสดงในรูปที่ 2.1



รูปที่ 2.1 แสดงข้อมูลเข้าและผลลัพธ์ของแอนเกต

จากรูปจะเห็นว่าแอนเกตประกอบด้วยตัวแปรที่เป็นข้อมูลเข้าสู่เกต 2 ตัวได้แก่ "input1" และ "input2" และตัวแปรที่แสดงผลลัพธ์ของเกตได้แก่ "output" ตัวแปรทุกตัวอาจมีค่าได้เป็นจริงหรือเท็จ ค่าของ "output" คำนวนได้จาก "input1 && input2" นั่นคือถ้าทั้ง "input1" และ "input2" เป็นจริงจะได้ค่า "output" เป็นจริงด้วย มิฉะนั้นค่าของ "output" จะเป็นเท็จ สิ่งที่ต้องการคือ ค่าของ "output" จะต้องเท่ากับผลของการคำนวณเสมอ ดังนั้นจึงกำหนดสมการกำหนดค่าเป็น

"output = input1 && input2"

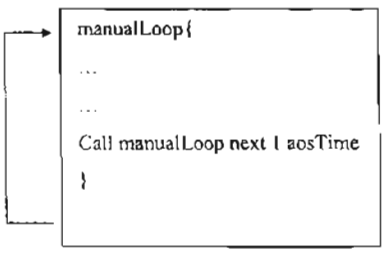
เมื่อไรก็ตามที่ "input1" หรือ "input2" เปลี่ยนแปลงค่าจะต้องมีการคำนวณผลลัพธ์ใหม่ทุกครั้งเพื่อกำหนดค่าที่ถูกต้องให้กับ "output"

2.1.1.3 การกระทำตามเหตุการณ์ (event routines)

พฤติกรรมแบบนี้จะอยู่ในรูปของการเรียกฟังก์ชันล่วงหน้า (future call) และ การกำหนดค่าล่วงหน้า (future assignment) การเรียกฟังก์ชันล่วงหน้าจะเป็นการเรียกฟังก์ชัน โดยมีการกำหนดเวลาหน่วงไว้ด้วย ฟังก์ชันดังกล่าวจะถูกประมวลผลก็ต่อเมื่อเวลาผ่านไปเท่ากับที่กำหนดไว้ในเวลาหน่วง ส่วนการกำหนดค่าล่วงหน้าก็มีลักษณะการทำงานคล้ายกัน แต่มีความแตกต่างคือเป็นการกำหนดค่าให้กับตัวแปรใดๆตามเวลาหน่วงที่กำหนดแทนที่จะเป็นการเรียกฟังก์ชัน

ตัวอย่างของการกำหนดค่าล่วงหน้าเห็นได้จากพฤติกรรมของสัญญาณไฟจราจรที่เปลี่ยนสีไปตามเวลาที่กำหนด เช่นในขณะที่สีของไฟจราจรเป็นสีแดง อาจกำหนดให้ค่าของสีเปลี่ยนเป็นสีเหลืองภายในอีก 2 วินาทีข้างหน้า และเมื่อไฟเป็นสีเหลืองให้กำหนดค่าของสีเป็นสีเขียวในอีก 1 วินาทีข้างหน้า และเมื่อไฟเป็นสีเขียว ให้เปลี่ยนเป็นสีแดงในอีก 3 วินาทีข้างหน้าเป็นต้น

ส่วนตัวอย่างของการเรียกฟังก์ชันล่วงหน้าเห็นได้จากพฤติกรรมของวาล์วซึ่งเป็นองค์ประกอบหนึ่งในระบบแท็งก์ รูปที่ 2.2 แสดงให้เห็นการเรียกฟังก์ชัน "manualLoop" ของวาล์ว ในทุกๆ 1 หน่วยเวลาวาล์วจะต้องประมวลผลฟังก์ชัน "manualLoop" ซึ่งจะแสดงระดับน้ำ ณ ขณะนั้นของวาล์วอย่างถูกต้อง ดังนั้นจึงสามารถใช้การเรียกฟังก์ชันล่วงหน้าในฟังก์ชันดังกล่าวให้ทำการเรียกตัวเองอีกครั้งในอีก 1 หน่วยเวลาข้างหน้า ด้วยวิธีนี้จะทำให้เกิดการประมวลผลฟังก์ชัน "manualLoop" แบบวนซ้ำได้ตามเวลาที่ต้องการ



รูปที่ 2.2 แสดงการเรียกฟังก์ชัน "manualLoop" ของวาล์ว

2.1.2 ส่วนควบคุมของระบบวัตถุพร้อมทำงาน

ส่วนควบคุมของระบบวัตถุพร้อมทำงาน (SAOS runtime kernel) พัฒนาขึ้นด้วยภาษาซีพลัสพลัส (C+ +) โปรแกรมระบบวัตถุพร้อมทำงานทุกระบบจะต้องทำงานอยู่ภายใต้ส่วนควบคุมนี้ซึ่งมีหน้าที่ในการกำกับดูแลวัตถุทุกตัวที่อยู่ในระบบให้ทำงานได้ตามพฤติกรรมที่ได้กำหนดไว้ เมื่อผู้พัฒนาต้องการให้วัตถุพร้อมทำงานมีพฤติกรรมตามประโยคการเปลี่ยนแปลงดังที่ได้กล่าวมาแล้ว จะมีวิธีในการติดต่อกับส่วนควบคุมนี้เพื่อให้รับทราบและจดจำพฤติกรรมเหล่านั้น เมื่อระบบเริ่มทำงาน ส่วนควบคุมจะทำให้วัตถุทำงานในเวลาและเงื่อนไขที่เหมาะสมตามที่กำหนดไว้ ส่วนควบคุมนี้ได้ออกแบบให้สามารถนำมาใช้ใหม่ได้ ดังนั้นเมื่อต้องการสร้างโปรแกรมในเรื่องราวใหม่ๆ ผู้พัฒนาจึงไม่ต้องพัฒนาส่วนควบคุมด้วยตัวเอง เพียงแต่กำหนดพฤติกรรมเพื่อให้ส่วนควบคุมนี้จะคอยจัดการให้เกิดการทำงานดังที่ต้องการได้

กลไกที่สำคัญในส่วนควบคุมของระบบวัตถุพร้อมทำงานที่จะทำให้วัตถุต่างๆมีพฤติกรรมตามประโยคการเปลี่ยนแปลงแบบต่างๆได้คือการใช้ตัวแปรพร้อมทำงาน (active variable) และกลไกการเริ่มทำงาน (trigger element) ตัวแปรพร้อมทำงานเป็นตัวแปรที่มีความสัมพันธ์กับฟังก์ชัน ทุกครั้งที่ตัวแปรเหล่านี้มีการเปลี่ยนแปลงค่าจะมีการเรียกให้ฟังก์ชันที่เกี่ยวข้องให้ทำงาน การสร้างตัวแปรพร้อมทำงานทำได้โดยประกาศให้ตัวแปรมีชนิดเป็นคลาส “AInteger” “ABoolean” หรือ “ADouble” ตามชนิดของข้อมูลที่ต้องการเก็บในตัวแปรนั้น โดยคลาส “AInteger”ใช้เก็บค่าอินทีเจอร์ คลาส “ABoolean” ใช้เก็บค่าบูลลีน และคลาส “ADouble” ใช้เก็บค่าตัวเลขทศนิยม ส่วนการกำหนดให้ตัวแปรพร้อมทำงานมีความสัมพันธ์กับฟังก์ชันที่ต้องการทำได้โดยการสร้างกลไกการเริ่มทำงานซึ่งจะเก็บข้อมูลเกี่ยวกับฟังก์ชันที่สัมพันธ์กับตัวแปรเริ่มทำงาน ด้วยวิธีนี้ระบบจะทราบว่าเมื่อตัวแปรพร้อมทำงานมีการเปลี่ยนแปลงค่าจะต้องทำการประมวลผลฟังก์ชันใดบ้าง

ส่วนควบคุมของระบบวัตถุพร้อมทำงานนี้ต่อมาได้มีผู้พัฒนาเป็นภาษาจาวาเพื่อนำไปใช้กับโปรแกรมต่างๆที่จะพัฒนาด้วยภาษาจาวา รายละเอียดของส่วนควบคุมในภาษาจาวา และขั้นตอนการสร้างประโยคการเปลี่ยนแปลงแบบต่างๆสำหรับส่วนควบคุมที่เป็นภาษาจาวาได้แสดงไว้ในหัวข้อ 2.3

2.2 ตัวแปลภาษาบรรยายระบบวัตถุพร้อมทำงานแบบโครงสร้าง (Structural Active Object System Description Language Translator)

ระบบวัตถุพร้อมทำงานแบบโครงสร้างถูกพัฒนาครั้งแรกด้วยภาษาซีพลัสพลัสวัตถุพร้อมทำงานแต่ละตัวจะถูกโปรแกรมด้วยภาษาซีพลัสพลัส การเขียนโปแกรมเกี่ยวกับพฤติกรรมของวัตถุเหล่านั้นให้เป็นแบบพร้อมทำงานทำได้ไม่ง่ายนัก เนื่องจากจะต้องทำตามกฎเกณฑ์ต่างๆที่จะทำให้วัตถุสามารถทำงานสอดคล้องกับโครงร่างของระบบ ทำให้เสียเวลาในการเขียนโปรแกรมมาก Raghava Pareddy จึงออกแบบภาษาบรรยายระบบวัตถุพร้อมทำงาน (SAOS description language) [3] ซึ่งเป็นส่วนขยายเพิ่มเติมจากภาษาซีพลัสพลัส เพื่อนำมาใช้กำหนดพฤติกรรมของวัตถุพร้อมทำงาน โดยใช้คำสำคัญ (keyword) ที่ได้ออกแบบขึ้นเพื่อสร้างประโยคการเปลี่ยนแปลงต่างๆ คำสำคัญเหล่านี้เป็นส่วนประกอบหลักในภาษาบรรยาย หลังจากเขียนโปรแกรมด้วยภาษาบรรยายแล้วจึงใช้ตัวแปลภาษาบรรยาย (SAOS Translator) ให้เป็นโปรแกรมในภาษาซีพลัสพลัส

ตัวอย่างเช่น การสร้างกฎการเปลี่ยนแปลงสำหรับตัวประมวลผลในแถวคอยในรูปภาษาบรรยายจะใช้คำว่า "when" เพื่อบอกเงื่อนไข และใช้ "transition" เพื่อกำหนดการกระทำ ตัวอย่างภาษาบรรยายสำหรับกำหนดกฎการเปลี่ยนแปลงของตัวประมวลผลเขียนได้ดังรูปที่ 2.3 นั่นคือเมื่อไรก็ตามที่เงื่อนไขที่กำหนดหลังคำว่า "when" เป็นจริง จะต้องมีการประมวลผลการกระทำที่กำหนดไว้หลังคำว่า "when" ซึ่งจะเป็นคำสั่งที่อยู่ในฟังก์ชัน "start" เป็นต้น

```
...
tranition start();
...
transition Processor :: start()
        when((input.njobs > 0) and (avail == true)){
                counter ++;
                inQ->njobs = inQ->njobs-1;
                avail = false;
                tm.startTimer(random() % MAXPROCESSRANGE);
}
```

รูปที่ 2.3 แสดงตัวอย่างภาษาบรรยายในการกำหนดกฎการเปลี่ยนแปลงของตัวประมวลผล

นอกจากคำว่า "when" แล้ว ยังมีการกำหนดคำสำคัญที่ใช้แทนประโยคการเปลี่ยนแปลงต่างๆรวมทั้งพฤติกรรมอื่นๆของวัตถุพร้อมทำงานอีก โดยใช้คำ "always" สำหรับสมการกำหนดค่าตัวอย่างเช่น สมการ "always output = *input1 && *input2" เป็นการกำหนดให้ "output" มีค่าเท่ากับผลลัพธ์ของการคำนวณตามสมการเสมอ โดยต้องมีการคำนวณใหม่ทุกครั้งที่ "input1" หรือ "input2" เปลี่ยนแปลง

นอกจากนี้ยังมีการใช้คำสำคัญ "say" เพื่อส่งข้อความไปให้วัตถุอื่นๆในระบบ ส่วน "on" ใช้เพื่อรับข้อความจากวัตถุอื่น เพื่อให้มีการกระทำใดๆที่ตอบสนองต่อข้อความที่ได้รับ

ภาษาบรรยายที่ออกแบบขึ้นนี้ทำให้ผู้พัฒนาโปรแกรมประยุกต์สามารถเขียนโปรแกรมระบบวัตถุพร้อมทำงานได้เร็วขึ้น เข้าใจได้ง่ายขึ้นด้วยขนาดของโปรแกรมที่เล็กลง อย่างไรก็ตามการใช้งานภาษาบรรยายและตัวแปลภาษาดังกล่าวสามารถใช้งานได้กับโปรแกรมในภาษาซีพลัสพลัสเท่านั้น ไม่สามารถนำมาใช้กับสิ่งแวดล้อมสำหรับพัฒนาโปรแกรมด้วยแผนภาพเอนทิตีและความสัมพันธ์ซึ่งพัฒนาด้วยภาษาจาวาได้ นอกจากนี้การใช้งานภาษาบรรยายดังกล่าวยังไม่สะดวกต่อการใช้งานนัก ผู้พัฒนายังต้องศึกษาและจดจำรูปแบบของภาษาบรรยายเพื่อให้สามารถใช้งานได้ถูกต้อง สิ่งแวดล้อมสำหรับการเขียนโปรแกรมยังอยู่ในรูปแบบตัวอักษร(text mode) ไม่มีสิ่งอำนวยความสะดวกและเครื่องมือเพื่อให้ผู้พัฒนาเขียนโปรแกรมได้ง่ายขึ้นเช่นเดียวกับสิ่งแวดล้อมในการเขียนโปรแกรมแบบวิชวลทั่วๆไป

2.3 ระบบวัตถุพร้อมทำงานในภาษาจาวา

Yanbing Lu ได้ทำการแปลงส่วนควบคุมของระบบวัตถุพร้อมทำงานและโปรแกรมประยุกต์ของระบบวัตถุพร้อมทำงานจากภาษาซีพลัสพลัส เป็นภาษาจาวา [4] เพื่อให้ระบบต่างๆสามารถทำงานได้บนเบาว์เซอร์ที่รองรับโปรแกรมภาษาจาวา

2.3.1 ขั้นตอนการสร้างพฤติกรรมของวัตถุพร้อมทำงาน

ส่วนควบคุมระบบวัตถุพร้อมทำงานในภาษาจาวามีหลักการ วิธีการทำงาน และองค์ประกอบเหมือนกับส่วนควบคุมที่พัฒนาด้วยภาษาซีพลัสพลัส ดังนั้นจึงรองรับการกลไกการทำงานของตัวแปรพร้อมทำงานและ กลไกการเริ่มทำงานเช่นเดียวกับส่วนควบคุมในภาษาซีพลัสพลัส และสามารถนำมาใช้เพื่อให้เกิดพฤติกรรมที่ต้องการได้ดังต่อไปนี้

2.3.1.1 ขั้นตอนการสร้างกฎการเปลี่ยนแปลง

1) กำหนดตัวแปรที่ใช้ในประโยคเงื่อนไขให้เป็นตัวแปรพร้อมทำงาน
2) สร้างฟังก์ชันที่จะถูกเรียกให้ทำงาน โดยเขียนประโยค "if" ตามเงื่อนไขที่ระบุในกฎการเปลี่ยนแปลง คำสั่งการทำงานต่างๆต้องอยู่ภายใต้ประโยค "if" เพื่อที่คำสั่งเหล่านี้ถูกเรียกก็ต่อเมื่อเงื่อนไขเป็นจริงเท่านั้น
3) สร้างกลไกการเริ่มทำงานสำหรับตัวแปรพร้อมทำงานทุกตัวในประโยคเงื่อนไขให้มีความสัมพันธ์กับฟังก์ชันที่สร้างขึ้น

จากวิธีการดังกล่าว เมื่อตัวแปรพร้อมทำงานตัวใดตัวหนึ่งในประโยคเงื่อนไขมีการเปลี่ยนแปลงค่า ฟังก์ชันที่มีความสัมพันธ์จะถูกเรียก เงื่อนไขภายในจะถูกตรวจสอบและถ้าเงื่อนไขเป็นจริง คำสั่งภายใต้ประโยคเงื่อนไขจะทำงาน

2.3.1.2 ขั้นตอนการสร้างสมการกำหนดค่า

1) กำหนดตัวแปรที่ใช้ในการคำนวณทางด้านขวาของสมการให้เป็นตัวแปรพร้อมทำงานอย่างน้อยหนึ่งตัว
2) สร้างฟังก์ชันที่จะถูกเรียก พร้อมทั้งทำการเขียนสมการที่ต้องการในฟังก์ชันดังกล่าว
3) สร้างกลไกการเริ่มทำงานสำหรับตัวแปรพร้อมทำงานทุกตัวที่ใช้ในการคำนวณให้มีความสัมพันธ์กับฟังก์ชันที่สร้างขึ้น

เมื่อตัวแปรพร้อมทำงานมีการเปลี่ยนแปลง ฟังก์ชันที่แสดงสมการไว้จะถูกเรียก ทำให้เกิดการคำนวณและกำหนดค่าได้ตามที่ต้องการ

2.3.1.3 ขั้นตอนการสร้างการกระทำตามเหตุการณ์

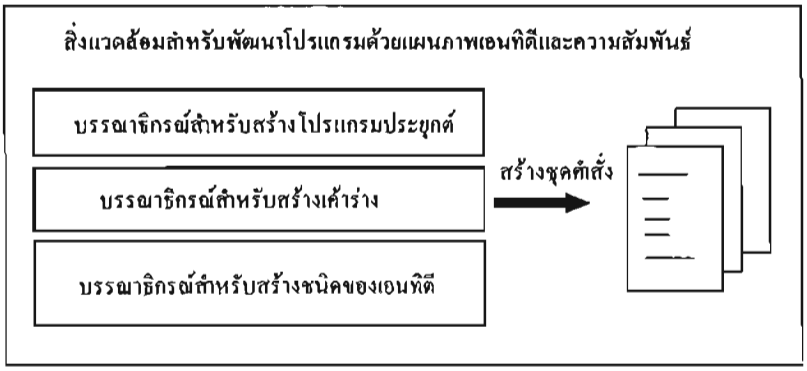
การกำหนดการเรียกฟังก์ชันล่วงหน้า และการกำหนดค่าล่วงหน้าไม่ต้องสร้างตัวแปรพร้อมทำงานและกลไกการเริ่มทำงาน เพียงแต่ใช้ฟังก์ชัน "fCall" และ "fAssign" พร้อมทั้งกำหนดพารามิเตอร์ต่างๆที่จำเป็น เช่น ฟังก์ชันที่ต้องการเรียก ตัวแปรและค่าที่ต้องการกำหนด และเวลาหน่วงเป็นต้น

2.3.2 วิธีการเรียกฟังก์ชันของส่วนควบคุมระบบวัตถุพร้อมทำงานในภาษาจาวา

ส่วนควบคุมระบบวัตถุพร้อมทำงานในภาษาจาวามีข้อจำกัดเกี่ยวกับการเรียกฟังก์ชันต่างๆในวัตถุพร้อมทำงาน ในภาษาซีพลัสพลัส เมื่อส่วนควบคุมพบว่าจะต้องมีการเรียกฟังก์ชันในวัตถุพร้อมทำงานจะสามารถเรียกฟังก์ชันนั้นได้โดยตรงจากส่วนควบคุมผ่านตัวชี้ (pointer) เมื่อต้องการเปลี่ยนไปเรียกฟังก์ชันอื่นจะทำได้โดยการเปลี่ยนให้ตัวชี้ชี้ไปยังฟังก์ชันที่ต้องการ แต่ในภาษาจาวาไม่รองรับการใช้ตัวชี้ที่จะชี้ไปยังฟังก์ชันต่างๆได้ ดังนั้นเมื่อส่วนควบคุมต้องการเรียกฟังก์ชันในวัตถุพร้อมทำงาน จะเรียกไปยังฟังก์ชันที่มีชื่อว่า "dispath" พร้อมทั้งส่งพารามิเตอร์เป็นตัวเลขจำนวนเต็มที่เรียกว่าดัชนีฟังก์ชัน (function index) ซึ่งเป็นหมายเลขประจำของฟังก์ชันที่ต้องการเรียก จึงมีข้อกำหนดว่าวัตถุพร้อมทำงานทุกตัวที่ทำงานภายใต้ส่วนควบคุมระบบวัตถุพร้อมทำงานในภาษาจาวานี้จะต้องกำหนดดัชนีฟังก์ชันให้กับฟังก์ชันทุกตัวที่กำหนดในประโยคการเปลี่ยนแปลง และมีฟังก์ชัน "dispatch" ซึ่งมีหน้าที่ตรวจสอบดัชนีฟังก์ชันที่ส่งมาจากส่วนควบคุม และทำการเรียกฟังก์ชันที่มีดัชนีตรงกับพารามิเตอร์ที่ได้รับ

2.4 สิ่งแวดล้อมสำหรับพัฒนาโปรแกรมด้วยแผนภาพเอนทิตีและความสัมพันธ์ (Entity-Relationship Software Development Environment : ERSDE)

ในงานวิจัยของ พรศิริ หมื่นไชยศรี ได้มีการพัฒนาสิ่งแวดล้อมสำหรับสร้างโปรแกรมโดยมุ่งเน้นให้ผู้ใช้มีแนวทางในการเชื่อมต่อองค์ประกอบซอฟต์แวร์โดยใช้แผนภาพเอนทิตีและความสัมพันธ์เป็นตัวแบบ [2,5,6] ผู้พัฒนาสามารถสร้างเอนทิตีที่ต้องการจากการเลือกชนิดของเอนทิตีในแผนภาพแล้วทำการเชื่อมต่อเอนทิตีเหล่านั้นเข้าด้วยกัน โปรแกรมจะสามารถทำงานได้ทันที เนื่องจากเอนทิตีที่สร้างขึ้นมีคุณสมบัติเป็นวัตถุพร้อมทำงาน รูปที่ 2.4 แสดงองค์ประกอบของสิ่งแวดล้อมสำหรับพัฒนาโปรแกรมด้วยแผนภาพเอนทิตีและความสัมพันธ์ ซึ่งประกอบด้วยบรรณาธิกรณ์สำหรับสร้างชนิดของเอนทิตี บรรณาธิกรณ์สำหรับสร้างเค้าร่าง และบรรณาธิกรณ์สำหรับสร้างโปรแกรมประยุกต์ เมื่อผู้พัฒนาต้องการสร้างโปรแกรมในขอบเขตเรื่องราวใหม่ จะต้องเริ่มจากการสร้างชนิดของเอนทิตีทั้งหมดที่ต้องใช้ในโปรแกรมนั้นโดยใช้บรรณาธิกรณ์สำหรับสร้างชนิดของเอนทิตี จากนั้นจึงกำหนดความสัมพันธ์ของชนิดของเอนทิตีเหล่านั้นในบรรณาธิกรณ์สำหรับสร้างเค้าร่าง ในขั้นตอนนี้ผู้พัฒนาสามารถสั่งให้ระบบสร้างชุดคำสั่งโดยระบบจะสร้างแฟ้มข้อมูลชุดคำสั่งสำหรับเอนทิตีต่างๆที่มีอยู่ในแผนภาพเอนทิตีและความสัมพันธ์ แต่ชุดคำสั่งที่สร้างได้ยังไม่สามารถนำไปประมวลผลได้และยังไม่มีคำสั่งที่รองรับพฤติกรรมของวัตถุพร้อมทำงาน ผู้พัฒนาจะต้องเขียนชุดคำสั่งเพิ่มเติมเอง ทำการคอมไพล์ ผลลัพธ์ที่ได้จากการคอมไพล์จะถูกนำไปใช้ในบรรณาธิกรณ์สำหรับสร้างโปรแกรมประยุกต์เพื่อให้ผู้ใช้งานได้สร้างโปรแกรมประยุกต์โดยการเชื่อมต่อเอนทิตีต่างๆเข้าด้วยกันตามแผนภาพเอนทิตีและความสัมพันธ์



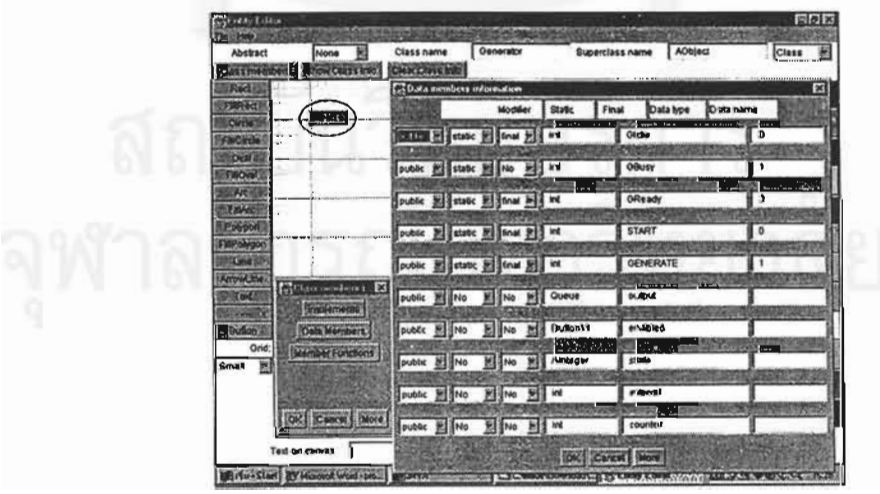
รูปที่ 2.4 แสดงองค์ประกอบของสิ่งแวดล้อมสำหรับพัฒนาโปรแกรมด้วยแผนภาพเอนทิตีและความสัมพันธ์

ตัวอย่างแผนภาพเอนทิตีและความสัมพันธ์ที่ได้มีการกำหนดไว้ได้แก่ การจำลองการทำงานของระบบแถวคอยซึ่งประกอบด้วยชนิดของเอนทิตี 3 ชนิดคือ ตัวสร้างงาน (generator) แถวคอย (queue) และตัวประมวลผล (processor) ผู้ใช้งานสามารถสร้างเอนทิตีดังกล่าวโดยอาจสร้างเพียงหนึ่งตัวหรือหลายตัวก็ได้โดยการเลือกชนิดเอนทิตีที่ต้องการจากแผนภาพเอนทิตีและความสัมพันธ์ จากนั้นจึงเชื่อมโยงเอนทิตีเข้าด้วยกัน โปรแกรมจะเริ่มแสดงการจำลองระบบแถวคอยได้ทันที

สิ่งแวดล้อมสำหรับสร้างโปรแกรมดังกล่าวแบ่งออกได้เป็น 3 ส่วนซึ่งมีรายละเอียดดังนี้

2.4.1 บรรณาธิกรณ์สำหรับสร้างชนิดของเอนทิตี (The Entity-Type Editor)

บรรณาธิกรณ์นี้ใช้สำหรับสร้างชนิดของเอนทิตี ผู้พัฒนาสามารถใช้รูปภาพสัญลักษณ์แทนชนิดของเอนทิตีแต่ละตัว พร้อมทั้งกำหนดข้อมูลสมาชิก(data member) และฟังก์ชันสมาชิก (member function) สำหรับชนิดของเอนทิตีนั้น หน้าจอหลักของบรรณาธิกรณ์สำหรับสร้างชนิดของเอนทิตีแสดงไว้ดังรูปที่ 2.5 ซึ่งเป็นหน้าจอสำหรับกำหนดข้อมูลสมาชิกของเอนทิตี



รูปที่ 2.5 แสดงบรรณาธิกรณ์สำหรับสร้างชนิดของเอนทิตี

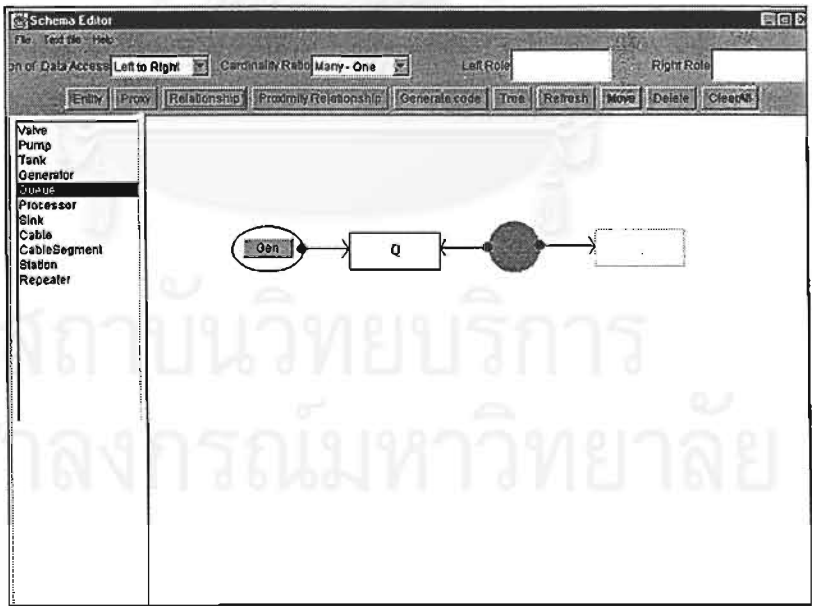
บรรณาธิกรณ์สำหรับสร้างชนิดของเอนทิตีประกอบด้วยหน้าจอสำหรับสร้างรูปสัญลักษณ์ที่ใช้แสดงแทนชนิดของเอนทิตีแต่ละตัว จากรูปที่ 2.5 สัญลักษณ์วงรีที่มีปุ่ม "Gen" ตรงกลางใช้แทนตัวสร้างงาน ซึ่งเป็นชนิดของเอนทิตีชนิดหนึ่งในระบบแถวคอย ผู้พัฒนาสามารถกำหนดชื่อชนิดของเอนทิตี ชื่อคลาสแม่ (super class) พร้อมทั้งฟังก์ชันสมาชิกและข้อมูลสมาชิก

2.4.2 บรรณาธิกรณ์สำหรับสร้างเค้าร่าง (The Schema Editor)

บรรณาธิกรณ์สำหรับสร้างเค้าร่างใช้สำหรับสร้างแผนภาพเอนทิตีและความสัมพันธ์ของแต่ละขอบเขตปัญหา โดยใช้สัญลักษณ์ที่กำหนดขึ้นจากบรรณาธิกรณ์สำหรับสร้างชนิดของเอนทิตี จากนั้นจึงเชื่อมโยงความสัมพันธ์ระหว่างชนิดของเอนทิตีเหล่านั้น พร้อมทั้งกำหนดคาร์ดินัลลิตีเรโช (cardinality ratios) ทิศทางการเข้าถึงข้อมูล (direction of data access) และ บทบาทของความสัมพันธ์ (role)

รูปที่ 2.6 แสดงหน้าจอของบรรณาธิกรณ์สำหรับสร้างเค้าร่าง ซึ่งมีส่วนประกอบที่สำคัญได้แก่รายการแสดงชนิดของเอนทิตี พื้นที่สำหรับสร้างแผนภาพ และปุ่มอื่นๆที่ใช้จัดการกับแผนภาพ แผนภาพที่แสดงบนหน้าจอในรูปคือแผนภาพเอนทิตีและความสัมพันธ์ของระบบแถวคอย ประกอบด้วยชนิดของเอนทิตีต่างๆดังนี้

1) ตัวสร้างงาน ใช้สัญลักษณ์รูปวงรีที่มีปุ่ม "Gen" ตรงกลาง
2) แถวคอย ใช้สัญลักษณ์รูปสี่เหลี่ยม
3) ตัวประมวลผล ใช้สัญลักษณ์รูปวงกลมทึบ



รูปที่ 2.6 แสดงบรรณาธิกรณ์สำหรับสร้างเค้าร่าง

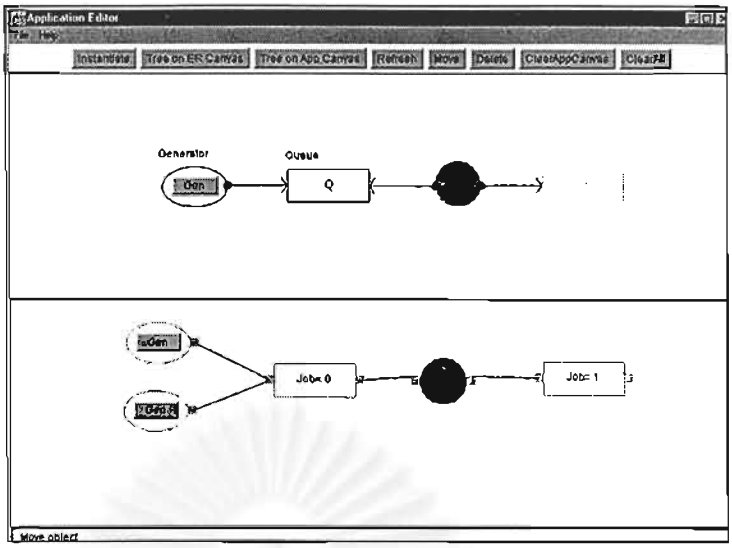
จากแผนภาพความสัมพันธ์ใน รูปที่ 2.6 จะเห็นได้ว่า ตัวสร้างงานมีความสัมพันธ์กับแถวคอย แถวคอยมีความสัมพันธ์กับตัวประมวลผล และส่วนที่เป็นด้านออกของตัวประมวลผลจะเชื่อมต่อเข้ากับแถวคอยแถวต่อไป ปลายของลูกศรแสดงคาร์ดินัลลิตีเรโช ปลายลูกศรด้านใดมีวงกลมเล็กๆสีดำแสดงว่าเอนทิตีนั้นสามารถเชื่อมต่อกับเอนทิตีอีกชนิดหนึ่งได้มากกว่าหนึ่งตัว แผนภาพดังรูปแสดงให้เห็นว่าตัวสร้างงานหลายๆตัวสามารถเชื่อมต่อกับแถวคอยเดียวกันได้ และแถวคอยแต่ละตัวสามารถเชื่อมต่อกับตัวประมวลผลหลายๆตัวได้

ส่วนบทบาทของความสัมพันธ์สามารถกำหนดได้ในแต่ละด้านของความสัมพันธ์ซึ่งจะแสดงถึงบทบาทของชนิดของเอนทิตีในความสัมพันธ์นั้น เช่นในความสัมพันธ์ระหว่างตัวสร้างงานกับแถวคอย บทบาทของตัวสร้างงานได้แก่ "สร้างงาน" และบทบาทของแถวคอยได้แก่ "รับงาน" เป็นต้น

เมื่อผู้พัฒนากำหนดแผนภาพความสัมพันธ์เรียบร้อยแล้ว สามารถสั่งให้ระบบสร้างโครงร่างของโปรแกรมเป็นชุดคำสั่งภาษาจาวาได้ โครงร่างของโปรแกรมที่สร้างจะเป็นการกำหนดคลาส พร้อมทั้งประกาศข้อมูลสมาชิก และฟังก์ชันสมาชิกดังที่กำหนดไว้ในบรรณาธิกรณ์สำหรับสร้างชนิดของเอนทิตี ระบบจะสร้างคลาส 2 ชนิดสำหรับแต่ละชนิดของเอนทิตี ได้แก่คลาสโมเดลซึ่งจัดการเกี่ยวกับพฤติกรรมหลักๆ และคลาสวิวที่จัดการเกี่ยวกับการแสดงผลแบบกราฟิกบนหน้าจอ ทั้งนี้เพื่อให้เป็นไปตามแนวความคิดการออกแบบของแบบจำลองเอ็มวีซีซึ่งได้แสดงรายละเอียดไว้ในหัวข้อ 2.5 คลาสโมเดลที่สร้างขึ้นจะมีชื่อคลาสเป็นชื่อเดียวกันชนิดของเอนทิตี ส่วนคลาสวิวจะมีชื่อคลาสขึ้นต้นด้วย "Edit" ตามด้วยชื่อชนิดของเอนทิตี ตัวอย่างเช่น เมื่อทำการสร้างชุดคำสั่งสำหรับตัวสร้างงานซึ่งมีชื่อชนิดของเอนทิตีว่า "Generator" ระบบจะสร้างคลาสโมเดลของตัวสร้างงานโดยมีชื่อคลาสว่า "Generator" และคลาสวิวของตัวสร้างงานโดยมีชื่อคลาสว่า "EditGenerator" เป็นต้น

2.4.3 บรรณาธิกรณ์สำหรับสร้างโปรแกรมประยุกต์ (application editor)

บรรณาธิกรณ์สำหรับสร้างโปรแกรมประยุกต์ใช้ในการสร้างโปรแกรมประยุกต์ของแต่ละขอบเขตปัญหา หน้าจอหลักของบรรณาธิกรณ์สำหรับสร้างโปรแกรมประยุกต์แสดงดัง รูปที่ 2.7 ประกอบด้วยพื้นที่ส่วนบนที่แสดงแผนภาพเอนทิตีและความสัมพันธ์ของระบบแถวคอยที่ได้กำหนดไว้ในบรรณาธิกรณ์สำหรับสร้างเค้าร่าง พื้นที่ด้านล่างสำหรับให้ผู้ใช้งานสร้างโปรแกรมประยุกต์ของระบบแถวคอย เมื่อผู้ใช้งานเลือกชนิดของเอนทิตีในแผนภาพด้านบนและคลิกที่บริเวณพื้นที่ด้านล่าง ระบบจะสร้างเอนทิตีชนิดนั้นขึ้น หลังจากเชื่อมต่อเอนทิตีเหล่านี้เข้าด้วยกันตามกฎเกณฑ์ที่กำหนดไว้ในแผนภาพ ระบบจะสามารถจำลองการทำงานของระบบแถวคอยได้ทันที ผู้พัฒนาสามารถแก้ไขการเชื่อมต่อได้ในหลายลักษณะบนจอภาพโดยไม่ต้องแก้ไขและคอมไพล์โปรแกรมใหม่



รูปที่ 2.7 แสดงบรรณาธิกรณ์สำหรับสร้างโปรแกรมประยุกต์

ประโยชน์จากงานวิจัยนี้สามารถสรุปได้ดังนี้

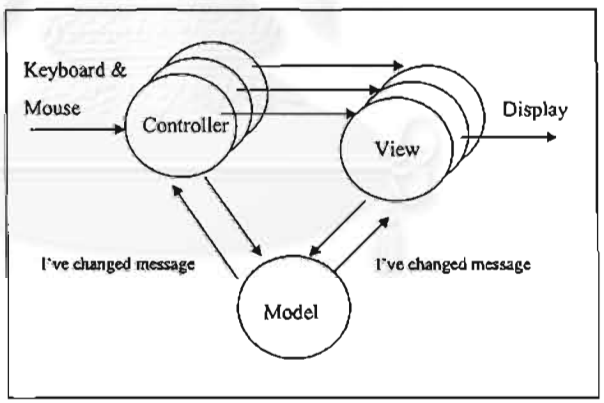
1) ผู้ใช้งานสามารถเชื่อมต่อองค์ประกอบโดยมีแผนภาพเอนทิตีและความสัมพันธ์เป็นแนวทางในการเชื่อมต่อ
2) สามารถสร้างโปรแกรมประยุกต์ที่ประมวลผลได้ทันทีที่เอนทิตีเชื่อมต่อกัน
3) สามารถสร้างวัตถุจากชนิดของเอนทิตีได้มากกว่าหนึ่งตัว และทดลองเชื่อมต่อวัตถุเหล่านี้ได้ในหลายลักษณะตราบเท่าที่ยังอยู่ภายใต้กฎเกณฑ์ความสัมพันธ์ในแผนภาพเอนทิตีและความสัมพันธ์ที่ได้กำหนดไว้ และเมื่อผู้ใช้ได้ทำการเชื่อมต่อระบบจะสามารถแสดงผลการทำงานได้ทันที
4) ผู้ใช้งานสามารถสร้างโปรแกรมในขอบเขตเรื่องราวที่แตกต่างออกไป โดยสร้างชนิดของเอนทิตีใหม่ๆ พร้อมทั้งความสัมพันธ์ของเอนทิตีเหล่านั้นได้โดยมีเครื่องมือที่ช่วยสร้างชุดคำสั่งในภาษาจาวาตามข้อมูลที่ผู้ใช้งานกำหนดไว้ได้โดยอัตโนมัติ คือเลือกเอนทิตีและความสัมพันธ์สำหรับขอบเขตเรื่องราวใหม่ ซึ่งจะสร้างชุดคำสั่งได้ดังต่อไปนี้
    - การประกาศตัวแปรข้อมูล (data) และ ฟังก์ชันสมาชิก (member function) ในแต่ละชนิดของเอนทิตี
    - การสร้างคลาสรูปภาพเพื่อใช้แสดงแทนเอนทิตี
    - การสร้างเมธอดรองรับการเชื่อมต่อระหว่างเอนทิตีโดยมิต้องแก้ไขโปรแกรมเมื่อต้องเปลี่ยนแปลงลักษณะการเชื่อมต่อ

2.5 แบบจำลองเอ็มวีซี

แบบจำลองเอ็มวีซี (MVC : Model-View-Controller) เสนอโดย Reenskaug [7] เป็นแบบจำลองสำหรับการเขียนโปรแกรมเชิงวัตถุที่ใช้กันอย่างกว้างขวางโดยเฉพาะในโครงร่างของระบบติดต่อกับผู้ใช้งานแบบกราฟิก (Graphical User Inteface Framework) แบบจำลองนี้เสนอให้แบ่งคลาสของโครงร่างออกเป็น 3 ชนิดดังต่อไปนี้

1) คลาสโมเดล (model class) ใช้สำหรับเก็บข้อมูลซึ่งจะแสดงและถูกจัดการโดยโปรแกรมที่มีระบบติดต่อกับผู้ใช้งานแบบกราฟิก
2) คลาสวิว (view class) ใช้ควบคุมการแสดงข้อมูลในคลาสโมเดลบนหน้าจอ
3) คลาสคอนโทรลเลอร์ (controller class) มีหน้าที่ในการรับข้อมูลนำเข้าจากแป้นพิมพ์และเมาส์ พร้อมทั้งส่งข้อความที่เหมาะสมให้กับคลาสโมเดล และคลาสวิวเพื่อให้สามารถเปลี่ยนแปลงแก้ไขข้อมูลในคลาสวิวได้

ตัวอย่างเช่น สำหรับโปรแกรมประมวลผลคำ (word processing) คลาสโมเดลจะถูกใช้เพื่อเก็บข้อมูลคำต่างๆ คลาสวิวจัดการการแสดงผลคำเหล่านี้บนหน้าจอ และคลาสคอนโทรลเลอร์รับข้อมูลที่ผู้ใช้ป้อนผ่านแป้นพิมพ์เป็นต้น รูปที่ 2.8 แสดงการทำงานร่วมกันของคลาสต่างๆในแบบจำลองเอ็มวีซี จากรูปแสดงให้เห็นว่าคลาสวิว และคลาสคอนโทรลเลอร์มีคลาสโมเดลได้เพียงหนึ่งคลาส ในขณะที่คลาสโมเดลอาจมีคลาสวิวหรือคลาสคอนโทรลเลอร์ได้มากกว่าหนึ่งคลาส



รูปที่ 2.8 แสดงการทำงานร่วมกันของคลาสต่างๆในแบบจำลองเอ็มวีซี

เหตุผลในการแยกคลาสต่างๆเหล่านี้ออกจากกันเนื่องจากในบางครั้งการแสดงผลข้อมูลชุดหนึ่งๆอาจแสดงได้ในหลายรูปแบบ เช่นอาจแสดงในรูปตาราง หรือรูปกราฟในแบบต่างๆ เมื่อมีการแก้ไขข้อมูลจากการแสดงผลในแบบใดแบบหนึ่ง จะทำให้เกิดความยุ่งยากในการทำให้การแสดงผลในรูปแบบอื่นๆมีความถูกต้องตรงกันทั้งหมด ปัญหานี้สามารถแก้ไขได้โดยการแยกส่วนของข้อมูลออกจากส่วนแสดงผลพร้อมทั้งเก็บรายการรูปแบบการแสดงผลทั้งหมดไว้ การแสดงผลทุกรูปแบบจะใช้ข้อมูลจากแหล่งข้อมูลเดียวกันเพียงชุดเดียวใน

คลาสโมเดล ถ้ามีการแก้ไขเปลี่ยนแปลงข้อมูลดังกล่าวจะสามารถจัดการให้การแสดงผลทุกรูปแบบเปลี่ยนแปลงตามได้อย่างถูกต้อง ตัวอย่างเช่น จากแผนภาพที่แสดงในรูปที่ 2.8 เมื่อคลาสโมเดลได้รับข้อความ "I've change" ข้อความดังกล่าวจะถูกส่งต่อให้กับคลาสวิวทุกคลาสเพื่อปรับปรุงการแสดงผลตามการเปลี่ยนแปลงนั้นๆได้อย่างถูกต้อง

บทที่ 3

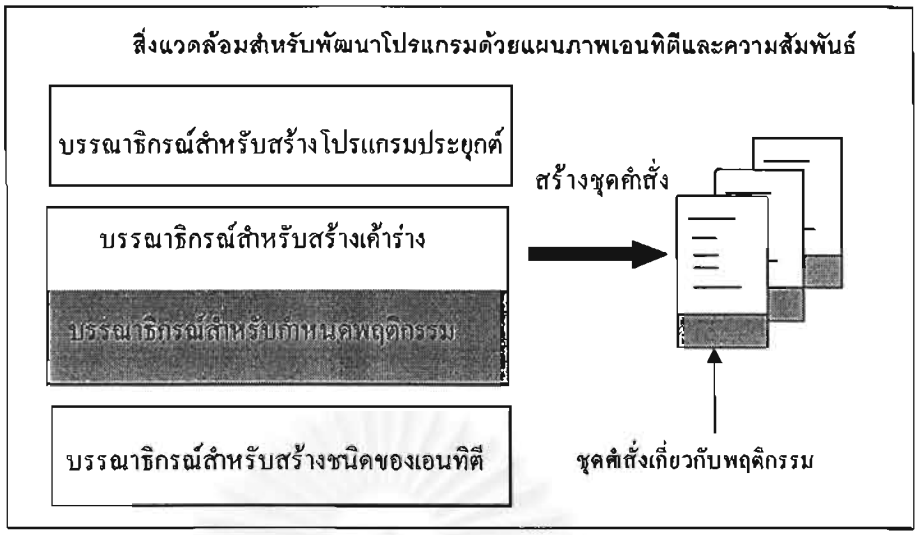
การออกแบบระบบ

สิ่งแวดล้อมสำหรับพัฒนาโปรแกรมด้วยแผนภาพเอนทิตีและความสัมพันธ์มีความสามารถในการสร้างชุดคำสั่งได้ในบางส่วน เช่นการสร้างคลาสสำหรับเอนทิตีต่างๆ การประกาศค่าข้อมูลสมาชิก ฟังก์ชันสมาชิก และคำสั่งที่รองรับการเชื่อมต่อกับเอนทิตีอื่นๆขณะทำงาน แต่ยังไม่สามารถสร้างชุดคำสั่งที่เกี่ยวข้องกับพฤติกรรมของวัตถุได้ ดังนั้นวิทยานิพนธ์นี้จึงมีเป้าหมายเพื่อเพิ่มความสามารถในการสร้างชุดคำสั่งให้สามารถสร้างชุดคำสั่งเกี่ยวกับพฤติกรรมตามประโยคการเปลี่ยนแปลงของระบบวัตถุพร้อมทำงาน จึงได้พัฒนาบรรณาธิกรณ์ใหม่เพื่อใช้สำหรับกำหนดพฤติกรรมของวัตถุพร้อมทำงาน และให้หน่วยสร้างชุดคำสั่งนำข้อมูลพฤติกรรมที่ได้กำหนดไว้แล้วสร้างเป็นชุดคำสั่งเพื่อรองรับพฤติกรรมที่ต้องการ

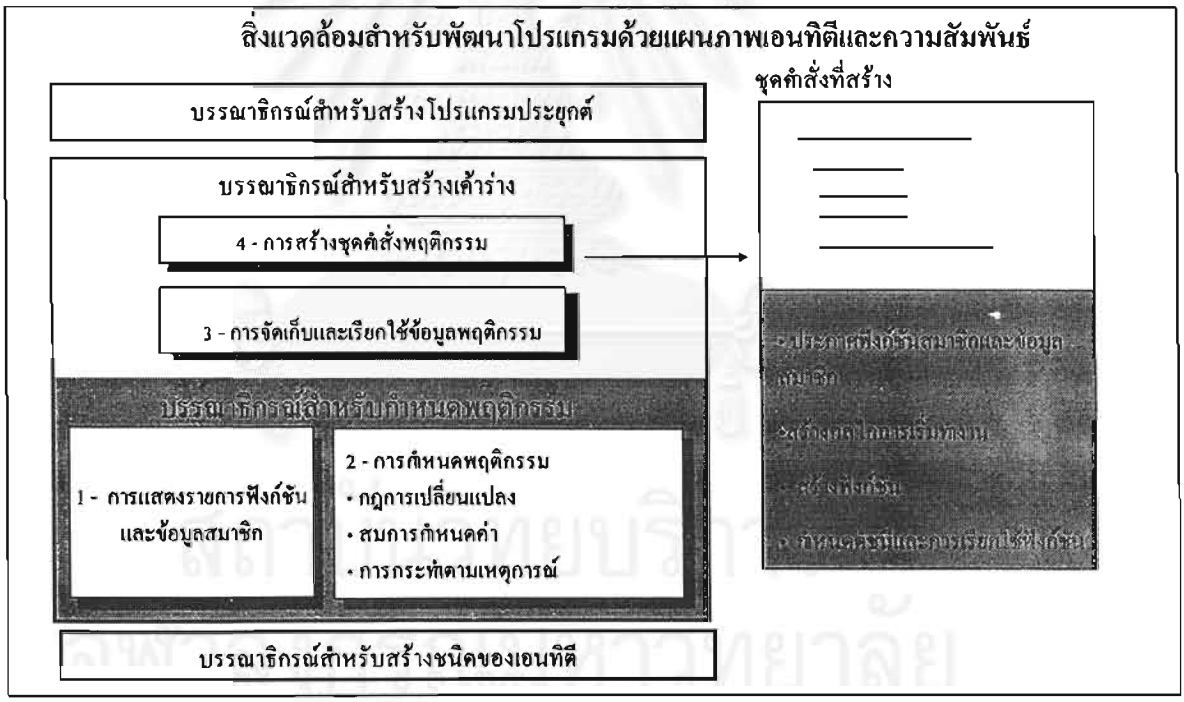
ในบทนี้จะอธิบายถึงองค์ประกอบของสิ่งแวดล้อมในการพัฒนาโปรแกรมด้วยแผนภาพเอนทิตีและความสัมพันธ์ที่ได้แก้ไขโดยเพิ่มเติมบรรณาธิกรณ์สำหรับกำหนดพฤติกรรมของวัตถุพร้อมทำงาน พร้อมทั้งแสดงรายละเอียดของฟังก์ชันต่างๆในบรรณาธิกรณ์สำหรับกำหนดพฤติกรรม รวมถึงการออกแบบส่วนติดต่อกับผู้ใช้งาน การออกแบบคลาสต่างๆที่ใช้ในระบบ และขั้นตอนในการสร้างชุดคำสั่ง

3.1 ส่วนประกอบของบรรณาธิกรณ์สำหรับกำหนดพฤติกรรมของวัตถุพร้อมทำงาน

บรรณาธิกรณ์สำหรับกำหนดพฤติกรรมของวัตถุพร้อมทำงานจะถูกนำไปใช้งานเป็นส่วนหนึ่งของบรรณาธิกรณ์สำหรับสร้างเค้าร่าง รูปที่ 3.1 แสดงส่วนประกอบในสิ่งแวดล้อมเพื่อการพัฒนาโปรแกรมด้วยแผนภาพเอนทิตีและความสัมพันธ์ที่ได้มีการเพิ่มเติมบรรณาธิกรณ์สำหรับกำหนดพฤติกรรมโดยถูกผนวกรวมกับบรรณาธิกรณ์สำหรับสร้างเค้าร่าง หลังจากที่ผู้พัฒนาสร้างชนิดของเอนทิตี และสร้างเค้าร่างแล้ว สามารถกำหนดพฤติกรรมของเอนทิตีเหล่านั้นด้วยบรรณาธิกรณ์สำหรับกำหนดพฤติกรรม และเมื่อทำการสร้างชุดคำสั่งอัตโนมัติจะมีชุดคำสั่งเกี่ยวกับพฤติกรรมของวัตถุพร้อมทำงานเพิ่มเติมขึ้นจากโครงร่างของชุดคำสั่งที่สร้างได้โดยอัตโนมัติอยู่แล้ว อย่างไรก็ตามผู้พัฒนายังคงต้องเขียนรายละเอียดการทำงานเพิ่มเติมและทำการคอมไพล์เพื่อนำผลลัพธ์ไปใช้ในบรรณาธิกรณ์สำหรับสร้างโปรแกรมประยุกต์เช่นเดิม รูปที่ 3.2 แสดงรายละเอียดภายในบรรณาธิกรณ์สำหรับกำหนดพฤติกรรมซึ่งประกอบด้วยส่วนประกอบหลักๆได้แก่ การแสดงรายการฟังก์ชันสมาชิกและข้อมูลสมาชิก การกำหนดพฤติกรรม การจัดเก็บและเรียกใช้ข้อมูลพฤติกรรม และการสร้างชุดคำสั่งพฤติกรรม



รูปที่ 3.1 แสดงองค์ประกอบของสิ่งแวดล้อมสำหรับพัฒนาโปรแกรมด้วยแผนภาพเอนทิตีและความสัมพันธ์ที่เพิ่มเติมบรรณาธิกรณ์สำหรับกำหนดพฤติกรรม



รูปที่ 3.2 แสดงส่วนประกอบในบรรณาธิกรณ์สำหรับกำหนดพฤติกรรม

ฟังก์ชันต่างๆในบรรณาธิกรณ์สำหรับกำหนดพฤติกรรมมีรายละเอียดดังต่อไปนี้

1) การแสดงรายการฟังก์ชันสมาชิกและข้อมูลสมาชิก

เป็นส่วนควบคุมการแสดงฟังก์ชันสมาชิกและข้อมูลสมาชิกที่ได้กำหนดไว้แล้วในบรรณาธิกรณ์สำหรับสร้างชนิดของเอนทิตี เพื่อให้ผู้ใช้สำรวจและเรียกดูรายการฟังก์ชันและข้อมูลของวัตถุพร้อมทำงานต่างๆได้อย่างสะดวก ทั้งยังสามารถเลือกใช้ข้อมูลเหล่านี้เมื่อต้องการกำหนดพฤติกรรมต่างๆให้กับวัตถุพร้อมทำงาน

2) การกำหนดพฤติกรรม

เป็นส่วนติดต่อกับผู้ใช้งานแบบกราฟิกสำหรับกำหนดพฤติกรรมของวัตถุพร้อมทำงานตามประโยคการเปลี่ยนแปลงประเภทต่างๆได้แก่ กฎการเปลี่ยนแปลง สมการกำหนดค่า และการกระทำตามเหตุการณ์

3) การจัดเก็บและเรียกใช้ข้อมูลพฤติกรรม

เป็นส่วนควบคุมการจัดเก็บข้อมูลเกี่ยวกับพฤติกรรมที่ได้กำหนดไว้แล้วลงในแฟ้มข้อมูล และเรียกใช้ข้อมูลพฤติกรรมจากแฟ้มเพื่อแสดงบนหน้าจอ

4) การสร้างชุดคำสั่งพฤติกรรม

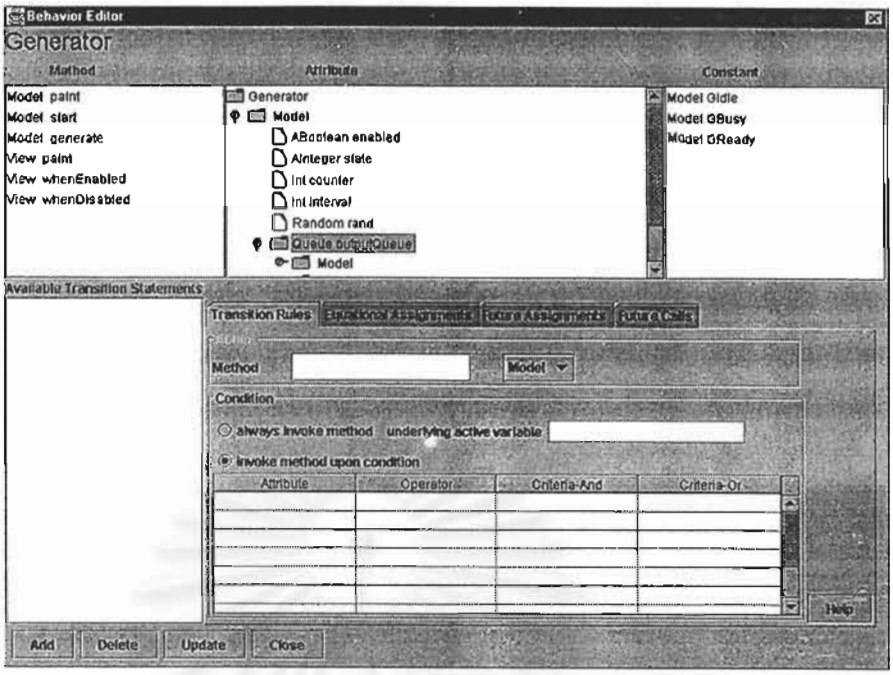
เป็นส่วนควบคุมการสร้างชุดคำสั่งเพื่อสร้างคลาสและชุดคำสั่งสำหรับชนิดของเอนทิตีแต่ละตัวที่แสดงในแผนภาพเอนทิตีและความสัมพันธ์

3.2 การออกแบบส่วนติดต่อกับผู้ใช้งาน

รูปที่ 3.3 แสดงการออกแบบหน้าจอหลักของบรรณาธิกรณ์สำหรับกำหนดพฤติกรรมของวัตถุพร้อมทำงาน หน้าจอแบ่งออกเป็นสองส่วนหลักๆ ส่วนด้านบนเป็นการแสดงรายการฟังก์ชันและข้อมูลสมาชิก ส่วนด้านล่างใช้สำหรับกำหนดพฤติกรรมของวัตถุพร้อมทำงานแยกตามประเภทต่างๆได้แก่กฎการเปลี่ยนแปลง สมการกำหนดค่า ส่วนการกระทำตามเหตุการณ์แบ่งออกเป็นการกำหนดค่าล่วงหน้าและการเรียกฟังก์ชันล่วงหน้า ผู้ใช้งานสามารถคลิกเลือกข้อมูลต่างๆที่แสดงด้านบนเพื่อกำหนดเป็นประโยคการเปลี่ยนแปลงในส่วนด้านล่างของหน้าจอ

รายละเอียดของฟังก์ชันสมาชิกและข้อมูลสมาชิกจะถูกเก็บแยกตามวัตถุพร้อมทำงานแต่ละตัว ระบบจะแสดงข้อมูลเหล่านี้แยกเป็น 3 ประเภทได้แก่ วิธีการ (method) คุณลักษณะเฉพาะ (attribute) และค่าคงที่ (constant) รายการข้อมูลเหล่านี้แบ่งย่อยออกได้เป็นข้อมูลในคลาสโมเดลและข้อมูลในคลาสวิว การแสดงคุณลักษณะเฉพาะของเอนทิตีจะแสดงเป็นลำดับชั้นแบบต้นไม้ซึ่งเป็นไปตามข้อกำหนดดังต่อไปนี้

1) ในลำดับชั้นแรกแบ่งเป็นกลุ่มข้อมูลโมเดลและวิว ซึ่งในแต่ละกลุ่มแสดงคุณลักษณะเฉพาะในคลาสโมเดลและคลาสวิวตามลำดับ



รูปที่ 3.3 แสดงหน้าจอของบรรณาธิกรณ์สำหรับกำหนดพฤติกรรมของวัตถุพร้อมทำงาน

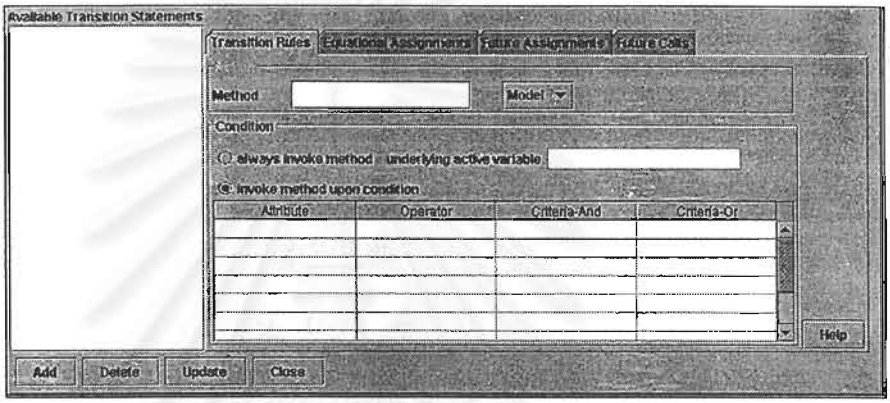
2) คุณลักษณะเฉพาะที่มีชนิด (Type) เป็นคลาสอื่นๆในแพ็กเก็ต "gui" ของระบบวัตถุพร้อมทำงาน จะต้องแสดงคุณลักษณะย่อยๆของคลาส อยู่ภายใต้กลุ่มข้อมูลของคุณลักษณะเฉพาะนั้นๆ กฎเกณฑ์นี้จะส่งผลถึงคุณลักษณะย่อยๆของคลาสด้วย ดังนั้น การแสดงคุณลักษณะดังกล่าวจะแสดงวนซ้ำไปเรื่อยๆจนกว่าคุณลักษณะเฉพาะทุกตัวจะมีชนิดข้อมูลเป็นชนิดพื้นฐาน (primitive type) หรือเป็นตัวแปรพร้อมทำงาน ซึ่งได้แก่ไบท์ (byte), อินทีเจอร์ (int), ลอง (long), ช็อต (short), ดับเบิล (double) , โฟลท (float), คาร์แรกเตอร์ (char), สตริง (String) , บูลลีน(boolean), แอดทีฟอินทีเจอร์(AInteger) , แอดทีฟดับเบิล(ADouble) และ แอดทีฟบูลลีน(ABoolean)

3) เนื่องจากการทำงานของวัตถุพร้อมทำงานขึ้นอยู่กับการเปลี่ยนแปลงสถานะภายในวัตถุและสถานะของวัตถุรอบข้าง ดังนั้นในการกำหนดพฤติกรรมของวัตถุพร้อมทำงาน นอกจากใช้คุณลักษณะเฉพาะภายในเอนทิตีแล้ว บางครั้งยังต้องกำหนดจากคุณลักษณะเฉพาะของ เอนทิตีข้างเคียงที่มีความสัมพันธ์กันด้วย ภายในคลาสโมเดลจึงมีกลุ่มข้อมูลเพิ่ม 2 กลุ่มได้แก่กลุ่มข้อมูลเข้า (input) และกลุ่มผลลัพธ์ (output) ที่แสดงคุณลักษณะเฉพาะของเอนทิตีที่เป็นข้อมูลเข้า และผลลัพธ์ตามลำดับ

4) ตามปกติคลาสโมเดลของเอนทิตีจะสืบทอดคุณสมบัติมาจากคลาส แอ็กทีฟอ็อปเจ็ค (AObject) ถ้าเอนทิตีใดสืบทอดคุณสมบัติมาจากคลาสอื่นๆ ภายในคลาสโมเดลจะมีกลุ่มข้อมูลพาร์เรนท์ (parent) ซึ่งจะแสดงคุณสมบัติเฉพาะของคลาสแม่ที่ได้รับสืบทอดมา

5) ข้อมูลในกลุ่มสุดท้ายได้แก่กลุ่ม "SaosMain" ซึ่งตัวแปรพร้อมทำงานในคลาสภายในคลาส "SaosMain" ที่ผู้ใช้งานสามารถนำมาใช้กำหนดพฤติกรรมได้

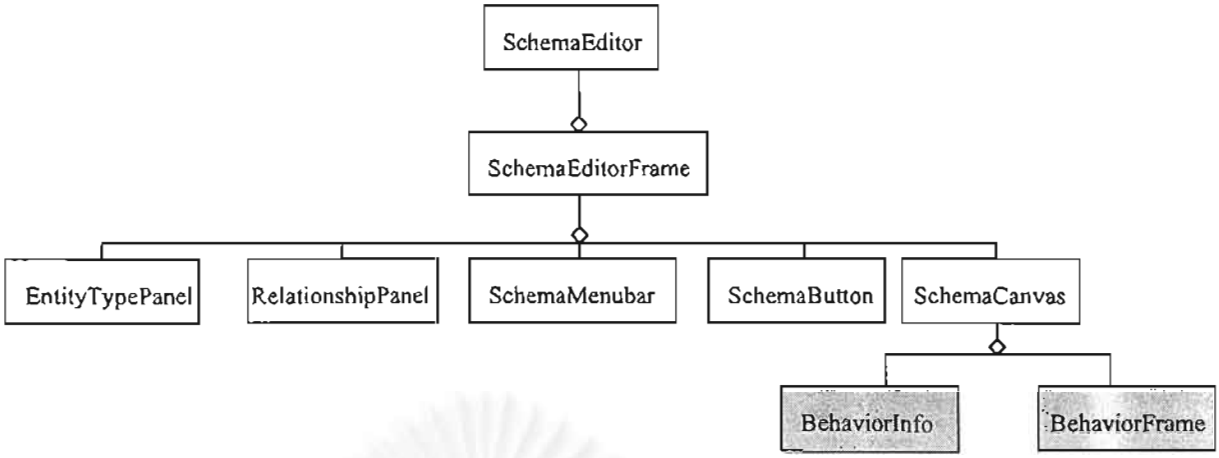
ส่วนด้านล่างของหน้าจอใช้สำหรับกำหนดพฤติกรรมของวัตถุ ประกอบด้วยหน้าจอเพื่อรับข้อมูลแบ่งตามประโยคการเปลี่ยนแปลงโดยการกระทำตามเหตุการณ์ได้ถูกแบ่งย่อยเป็นการกำหนดค่าล่วงหน้า (future assignment) และการเรียกฟังก์ชันล่วงหน้า (future call) เมื่อผู้ใช้คลิกเลือกรายการของประโยคการเปลี่ยนแปลงที่ต้องการ จะปรากฏหน้าจอสำหรับรับค่าของประโยคการเปลี่ยนแปลงนั้นๆ ตัวอย่างในรูปที่ 3.4เป็นหน้าจอสำหรับกำหนดกฎการเปลี่ยนแปลง ด้านซ้ายมือเป็นรายการประโยคการเปลี่ยนแปลงที่ได้กำหนดไว้แล้ว ประเภทของประโยคการเปลี่ยนแปลงที่แสดงในรายการจะเปลี่ยนไปตามรายการที่เลือก ส่วนด้านล่างเป็นปุ่มที่ใช้เพิ่ม ลบ และเปลี่ยนแปลงประโยคการเปลี่ยนแปลง ผู้ใช้สามารถเลือกข้อมูลต่างๆจากรายการที่แสดงได้โดยการคลิกเลือกช่องว่างที่ต้องการเติม จากนั้นจึงดับเบิลคลิกที่วิธีการ ลักษณะเฉพาะและค่าคงที่ที่ต้องการ



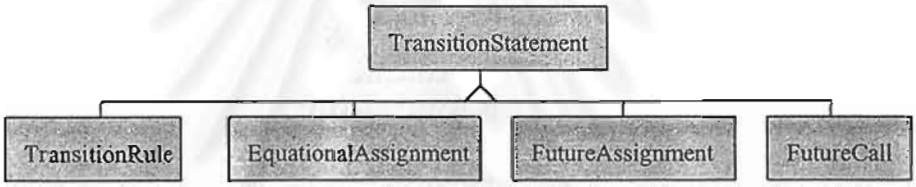
รูปที่ 3.4 แสดงหน้าจอสำหรับกำหนดกฎการเปลี่ยนแปลง

3.3 การออกแบบคลาส

รูปที่ 3.5 แสดงแผนภาพคลาสในภาพรวมของบรรณาธิกรณ์สำหรับสร้างเค้าร่าง รูปสี่เหลี่ยมสีขาวในแผนภาพแทนคลาสในบรรณาธิกรณ์เดิม ส่วนสี่เหลี่ยมสีเทา "BehaviorInfo" และ "BehaviorFrame" เป็นคลาสในบรรณาธิกรณ์สำหรับกำหนดพฤติกรรมโดยถือเป็นองค์ประกอบหนึ่งในคลาส "SchemaCanvas" คลาส "BehaviorInfo" ใช้เก็บและจัดการกลุ่มของประโยคการเปลี่ยนแปลงซึ่งจัดเก็บอยู่ในคลาสแยกต่างประเภทต่างๆ ได้แก่ คลาส "TransitionRule" "EquationalAssignment" "FutureAssignment" และ "FutureCall" รูปที่ 3.6 แสดงให้เห็นว่า คลาสประโยคการเปลี่ยนแปลงทุกคลาสที่ได้กล่าวมานั้นล้วนเป็นคลาสลูก(subclass) ที่สืบทอดคุณสมบัติจากคลาสแม่เดียวกันคือคลาส "TransitionStatement" แต่มีคุณลักษณะเฉพาะที่ใช้เก็บข้อมูลที่แตกต่างกันตามชนิดของพฤติกรรม



รูปที่ 3.5 แสดงแผนภาพคลาสในบรรณาธิกรณ์สำหรับสร้างเค้าร่าง



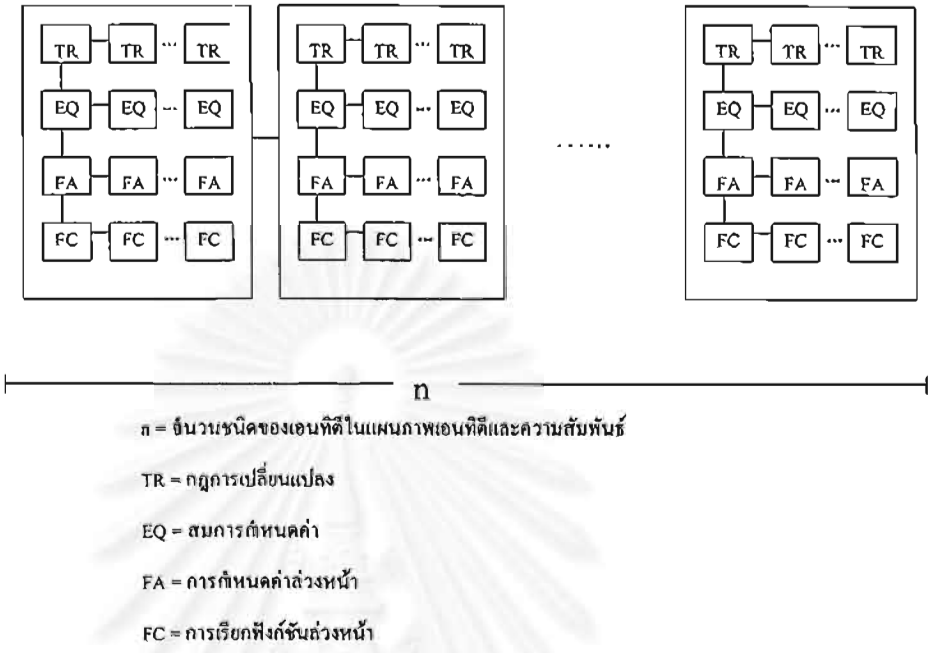
รูปที่ 3.6 แสดงแผนภาพคลาสในภาพรวมของคลาสประโยคการเปลี่ยนแปลงชนิดต่างๆ

ในชนิดของเอนทิตีหนึ่งๆอาจมีการกำหนดประโยคการเปลี่ยนแปลงได้หลายประโยค โดยจะเก็บอยู่ในรูปแบบโครงสร้างเวคเตอร์ดังที่แสดงในรูปที่ 3.7 ภายในเวคเตอร์ประกอบด้วยหน่วยเก็บข้อมูล (element) ที่มีจำนวน n โดยที่ n คือจำนวนชนิดของเอนทิตีในแผนภาพเอนทิตีและความสัมพันธ์ ภายในแต่ละหน่วยเก็บข้อมูลประกอบด้วยเวคเตอร์ย่อยที่มี 4 แถว แต่ละแถวใช้เก็บข้อมูลที่แตกต่างกันดังนี้

1) แถวที่ 1 ใช้สัญลักษณ์ "TR" สำหรับเก็บกฎการเปลี่ยนแปลง
2) แถวที่ 2 ใช้สัญลักษณ์ "EQ" สำหรับเก็บสมการกำหนดค่า
3) แถวที่ 3 ใช้ด้วยสัญลักษณ์ "FA" สำหรับเก็บการกำหนดค่าล่วงหน้า
4) แถวที่ 4 ใช้สัญลักษณ์ "FC" สำหรับเก็บการเรียกฟังก์ชันล่วงหน้า

จำนวนประโยคการเปลี่ยนแปลงเหล่านี้มีได้ไม่จำกัด ถ้ามีการเพิ่มเติมประโยคการเปลี่ยนแปลง ก็สามารถจัดเก็บข้อมูลที่เพิ่มต่อท้ายในแถวที่ตรงกับชนิดของประโยคการเปลี่ยนแปลงที่เพิ่มขึ้นมา เมื่อสั่งให้จัด

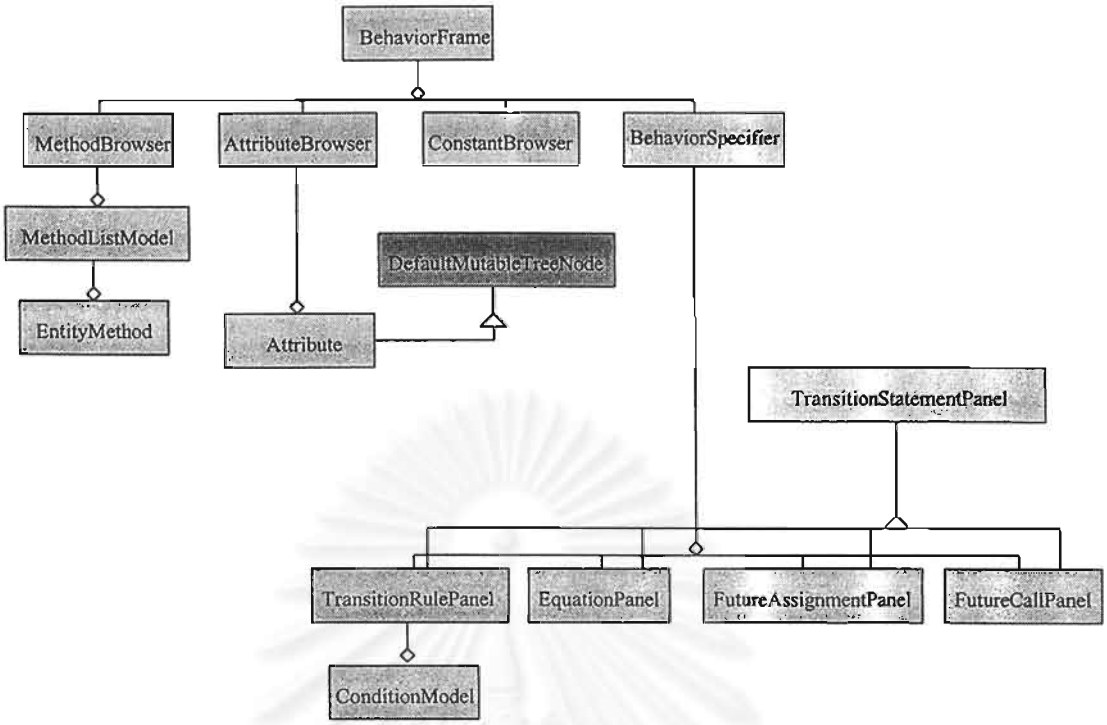
เก็บข้อมูลลงในแฟ้ม ข้อมูลในเวคเตอร์เหล่านี้จะถูกเขียนเรียงลำดับกันไปแบบอนุกรม และเมื่อเปิดแฟ้มข้อมูล ระบบจะอ่านรหัสเหล่านี้จะจัดเก็บในเวคเตอร์ที่มีโครงสร้างดังรูปพร้อมสำหรับแสดงผลต่อผู้ใช้งาน



รูปที่ 3.7 แสดงโครงสร้างของเวคเตอร์เก็บข้อมูลพฤติกรรม

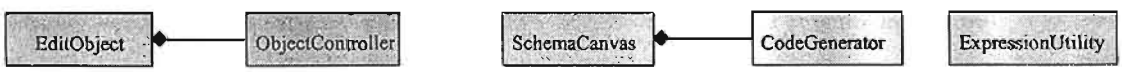
คลาส "BehaviorFrame" เป็นคลาสหลักที่ใช้จัดการเกี่ยวกับการติดต่อกับผู้ใช้งานในบรรณาธิกรณ์สำหรับกำหนดพฤติกรรม องค์ประกอบภายในคลาส "BehaviorFrame" แสดงได้ดังรูปที่ 3.8 เนื่องจากเป็นคลาสที่สืบทอดคุณสมบัติจากคลาส "JFrame" จึงมีความสามารถที่จะแสดงผลเป็นหน้าจอได้โดยสามารถบรรจุองค์ประกอบอื่นๆที่ใช้ในการติดต่อกับผู้ใช้งานเช่น ปุ่ม เท็กซ์ฟิลด์ ลิสต์เป็นต้น ภายในคลาส "BehaviorFrame" มีคลาสย่อยๆเป็นองค์ประกอบได้แก่ "MethodBrowser" "ConstantBrowser" "AttributeBrowser" และ "BehaviorSpecifier" สามคลาสแรกเป็นคลาสที่จัดการเกี่ยวกับการอ่านฟังก์ชันและข้อมูลสมาชิกจากเอนทิตีเพื่อแสดงบนหน้าจอ ส่วนคลาสสุดท้ายเป็นส่วนที่ผู้ใช้งานป้อนข้อมูลพฤติกรรม

คลาส "BehaviorSpecifier" ประกอบด้วยคลาสย่อยๆที่ใช้ควบคุมการกำหนดพฤติกรรมแบบต่างๆได้แก่ "TransitionRulePanel" "EquationaPanel" "FutureAssignmentPanel" และ "FutureCallPanel" คลาสต่างๆเหล่านี้มีชื่อคล้ายกับคลาสที่ใช้เก็บข้อมูลพฤติกรรมแต่มีส่วนท้ายเป็น "Panel" มีหน้าที่ในการแสดงหน้าจอเพื่อให้ผู้ใช้งานกำหนดพฤติกรรมแบบต่างๆรวมทั้งการจัดการข้อมูลที่ผู้ใช้งานกำหนดไว้แล้ว



รูปที่ 3.8 แสดงแผนภาพคลาสในภาพรวมขององค์ประกอบภายในส่วนติดต่อกับผู้ใช้งานของบรรณาธิกรณ์สำหรับกำหนดพฤติกรรม

รูปที่ 3.9 แสดงแผนภาพคลาสของ "ObjectController" คลาส "CodeGenerator" และคลาส "ExpressionUtility" คลาส "ObjectController" มีหน้าที่ในการค้นหาข้อมูลในเอนทิตีต่างๆ จะมีการอ้างอิงไปถึงวัตถุในคลาส "EditObject" ซึ่งเก็บข้อมูลของเอนทิตีเช่น ชื่อคลาส ชื่อคลาสแม่ ข้อมูลสมาชิกและฟังก์ชันสมาชิกเป็นต้น ส่วน คลาส "CodeGenerator" ทำหน้าที่สร้างชุดคำสั่ง โดยระหว่างการทำงานจะต้องค้นหาข้อมูลต่างๆที่เกี่ยวข้องสำหรับการสร้างชุดคำสั่งที่เก็บอยู่ในคลาส "SchemaCanvas" ดังนั้นจึงต้องมีการอ้างอิงถึงคลาสดังกล่าวเพื่อให้สามารถเรียกดูข้อมูลเหล่านั้นได้ คลาส "ExpressionUtility" ใช้สำหรับค้นหาตัวถูกดำเนินการ (operand) ที่ใช้ในสมการกำหนดค่ารายละเอียดเพิ่มเติมเกี่ยวกับการทำงานของคลาสต่างๆที่ได้กล่าวมานี้แสดงไว้ในบทที่ 4 การพัฒนาระบบ



รูปที่ 3.9 แสดงแผนภาพคลาสในภาพรวมของคลาส "ObjectController"
คลาส "CodeGenerator" และ คลาส "ExpressionUtility"

3.4 ขั้นตอนการสร้างชุดคำสั่ง

การสร้างชุดคำสั่งเพื่อรองรับการทำงานของประโยคการเปลี่ยนแปลงที่กำหนดให้สำหรับชนิดของเอนทิตีแต่ละตัวเป็นไปตามวิธีการกำหนดพฤติกรรมของส่วนควบคุมการทำงานของระบบวัตถุพร้อมทำงานในภาษาจาวาที่พัฒนาโดย Yanbing Lu [4] ส่วนข้อกำหนดในการสร้างคลาสโมเดลและคลาสวิวเพื่อให้ใช้งานได้ในบรรณาธิกรณ์สำหรับสร้างโปรแกรมประยุกต์ยังคงยึดการสร้างชุดคำสั่งของบรรณาธิกรณ์สำหรับสร้างเค้าร่างเดิม ขั้นตอนการสร้างชุดคำสั่งทั้งหมดสามารถสรุปได้ดังต่อไปนี้

1) สร้างคลาสโปรแกรมประยุกต์ระดับบน (top-level application-specific class) สำหรับควบคุมการสร้างวัตถุในขอบเขตเรื่องราวหนึ่งๆ คลาสที่สร้างได้มีชื่อเป็น "ApplicationSpecific" ตัวอย่างคลาสโปรแกรมประยุกต์สำหรับระบบแถวคอยแสดงดังรูปที่ 3.10
2) สร้างคลาสโมเดลและคลาสวิวสำหรับแต่ละชนิดของเอนทิตีในระบบ คลาสโมเดลจะมีชื่อเดียวกับชนิดของเอนทิตี ส่วนคลาสวิวจะมีชื่อขึ้นต้นด้วย "Edit" ตามด้วยชื่อของเอนทิตีดังรายละเอียดที่แสดงไว้ในหัวข้อ 2.4
3) สร้างแฟ้มข้อมูลชื่อ "ErrorLog.txt" สำหรับบันทึกรายละเอียดการสร้างชุดคำสั่งโดยบันทึกวันเวลา ชนิดของเอนทิตีที่สร้างชุดคำสั่งรวมทั้งความผิดพลาดที่อาจเกิดขึ้นในแฟ้มข้อมูลนี้เพื่อใช้ในการตรวจสอบ
4) เขียนคำสั่งเพื่อประกาศข้อมูลสมาชิกและฟังก์ชันสมาชิกในคลาสโมเดลหรือคลาสวิวของเอนทิตีตามชนิดของข้อมูลที่ได้กำหนดไว้ในบรรณาธิกรณ์สำหรับสร้างชนิดของเอนทิตี
5) เขียนชุดคำสั่งเพื่อสร้างกลไกการเริ่มทำงานโดยมีหลักเกณฑ์ดังนี้
   5.1) สำหรับกฎการเปลี่ยนแปลงที่มีการระบุเงื่อนไข จะต้องสร้างกลไกการเริ่มทำงานสำหรับตัวแปรพร้อมทำงานทุกตัวที่แสดงอยู่ในส่วนเงื่อนไข โดยกำหนดให้มีความสัมพันธ์กับฟังก์ชันที่ระบุไว้ในส่วนการกระทำ
   5.2) สำหรับกฎการเปลี่ยนแปลงที่มิได้ระบุเงื่อนไข แต่ระบุตัวแปรพร้อมทำงานที่ต้องตรวจสอบการเปลี่ยนแปลง จะต้องสร้างกลไกการเริ่มทำงานสำหรับตัวแปรดังกล่าวและกำหนดให้มีความสัมพันธ์กับฟังก์ชันที่ระบุไว้ในส่วนการกระทำ
   5.3) สำหรับสมการกำหนดค่า สร้างกลไกการเริ่มทำงานสำหรับตัวแปรพร้อมทำงานทุกตัวในส่วนด้านขวาของสมการ โดยกำหนดให้มีความสัมพันธ์กับวิธีการใหม่ที่มีชื่อขึ้นต้นด้วย "always" ตามด้วยชื่อตัวแปรด้านซ้าย
   5.4) สำหรับการกำหนดค่าล่วงหน้า และการเรียกฟังก์ชันล่วงหน้า ไม่ต้องสร้างกลไกการเริ่มทำงาน
   5.5) ถ้าตัวแปรพร้อมทำงานเป็นตัวแปรภายในของชนิดของเอนทิตี ไม่ใช่ตัวแปรของเอนทิตีที่เป็นข้อมูลเข้าหรือผลลัพธ์ ระบบจะเขียนคำสั่งการสร้างกลไกการเริ่มทำงานไว้ในฟังก์ชัน "initialize( )"

5.6) ถ้าตัวแปรพร้อมทำงานเป็นตัวแปรของเอนทิตีที่เป็นอินพุต ระบบจะเขียนคำสั่งการสร้างกลไกการเริ่มทำงานไว้ในฟังก์ชัน "connectModelViewInput( )"

5.7) ถ้าตัวแปรพร้อมทำงานเป็นตัวแปรของเอนทิตีที่เป็นเอาท์พุต ระบบจะเขียนคำสั่งการสร้างกลไกการเริ่มทำงานไว้ในฟังก์ชัน "connectModelViewOutput( )"

6) เขียนชุดคำสั่งเพื่อสร้างกลไกการเริ่มทำงานโดยมีหลักเกณฑ์ดังนี้

6.1) สำหรับกฎการเปลี่ยนแปลง สร้างฟังก์ชันที่ระบุไว้ในส่วนกระทำ โดยสร้างไว้ในคลาสโมเดลหรือคลาสวิวขึ้นอยู่กับชนิดของฟังก์ชัน จากนั้นจึงเขียนเงื่อนไขที่ต้องตรวจสอบไว้ในฟังก์ชัน ถ้ามีหลายเงื่อนไขที่มีความสัมพันธ์กับฟังก์ชันเดียวกันให้เชื่อมเงื่อนไขเหล่านั้นด้วย "หรือ" (or)

6.2) สำหรับสมการกำหนดค่า สร้างฟังก์ชันใหม่โดยมีชื่อฟังก์ชันขึ้นต้นด้วย "always" ตามด้วยชื่อตัวแปรที่ต้องการกำหนดค่า จากนั้นจึงเขียนสมการดังกล่าวในฟังก์ชันที่สร้างขึ้น

6.3) สำหรับการกำหนดค่าล่วงหน้า สร้างฟังก์ชันที่ระบุไว้ใน "วิธีการที่เกิดการกำหนดค่าล่วงหน้า" (calling method) เขียนคำสั่ง "fAssign" ในฟังก์ชันดังกล่าว

6.4) สำหรับการเรียกฟังก์ชันล่วงหน้า สร้างฟังก์ชันที่ระบุไว้ใน "วิธีการที่เรียกฟังก์ชันค่าล่วงหน้า" (calling method) เขียนคำสั่ง "fCall" ในฟังก์ชันดังกล่าว

6.5) ถ้ามีฟังก์ชันซ้ำกันระหว่างประโยคการเปลี่ยนแปลงต่างชนิดกัน ไม่ต้องสร้างฟังก์ชันใหม่แต่ให้เขียนคำสั่งของประโยคการเปลี่ยนแปลงนั้นๆลงในฟังก์ชันที่มีอยู่แล้ว โดยเรียงลำดับคำสั่งจาก กฎการเปลี่ยนแปลง การกำหนดค่าล่วงหน้า และการเรียกฟังก์ชันล่วงหน้า

7) เขียนชุดคำสั่งกำหนดดัชนีฟังก์ชันและการเรียกใช้ฟังก์ชันโดยมีกฎเกณฑ์ดังนี้

7.1) กำหนดหมายเลขจำนวนเต็มเรียงลำดับให้กับฟังก์ชันต่างๆต่อไปนี้

7.1.1) ฟังก์ชันที่ระบุไว้ในส่วนการกระทำในกฎการเปลี่ยนแปลง

7.1.2) ฟังก์ชันที่สร้างขึ้นใหม่สำหรับเขียนสมการในสมการกำหนดค่า

7.1.3) ฟังก์ชันที่ถูกเรียก (called method) ในการเรียกฟังก์ชันล่วงหน้า

7.2) การกำหนดหมายเลขดังกล่าวสำหรับคลาสโมเดลและคลาสวิวให้ใช้หมายเลขต่างชุดกัน

7.3) ประกาศตัวแปรแบบสแตติก ไฟนอล (static final) ที่มีชนิดเป็นอินทีเจอร์ (int) ให้มีค่าเท่ากับดัชนีของฟังก์ชันที่ได้กำหนดไว้ โดยให้มีชื่อของตัวแปรเป็นอักษรภาษาอังกฤษตัวใหญ่ตามชื่อของฟังก์ชัน และประกาศตัวแปรดังกล่าวในคลาสโมเดลหรือคลาสวิวตามชนิดของฟังก์ชัน

8) สร้างฟังก์ชัน "dispatch( )" ที่รับค่าพารามิเตอร์เป็นค่าดัชนีของฟังก์ชัน จากนั้นเขียนคำสั่งที่ใช้ในการตรวจสอบ และเรียกใช้ฟังก์ชันที่มีค่าดัชนีตรงกับพารามิเตอร์ที่ได้รับมา

```
import saos.gui.*;
import saos.kernel.*;
import java.awt.*;
import java.io.*;
import java.lang.Math;
import java.util.*;
class ApplicationSpecific extends AObject
{
 public ApplicationSpecific()
 {
  super("ApplicationSpecific");
 }
 public EditApp createObject(int index, EditObject object, AppCanvas canvas, int x, int y)
 {
  try
  {
   switch (index)
   {
   /* instantiate application-specific */
    case 0:
     EditGenerator temp0 =  new EditGenerator( (EditCompound) (object), canvas, x, y);
     insert(temp0, true);
     return temp0;
    case 1:
     EditProcessor temp1 = new EditProcessor( (EditCompound) (object), canvas, x, y);
     insert(temp1, true);
     return temp1;
    case 2:
     EditQueue temp2 = new EditQueue( (EditCompound) (object), canvas, x, y);
     insert(temp2, true);
     return temp2;
    case 3:
     EditQueue temp3 =  new EditQueue( (EditCompound) (object), canvas, x, y);
     insert(temp3, true);
     return temp3;
    default: return null;
   }
  } catch (SaosException ie)
    {
     System.out.println("Error: " + ie);
    }
  return null;
 }
}
```

รูปที่ 3.10 แสดงชุดคำสั่งของคลาสโปรแกรมประยุกต์ระดับบนของระบบแถวคอย

บทที่ 4

การพัฒนาระบบ

ในบทนี้เป็นการอธิบายรายละเอียดการทำงานของคลาสต่างๆในบรรณาธิกรณ์สำหรับกำหนดพฤติกรรมของวัตถุพร้อมทำงาน ซึ่งพัฒนาขึ้นด้วยภาษาจาวา บนระบบปฏิบัติการวินโดว์ โดยใช้องค์ประกอบจากไลบรารีของจาวาดีเวลลอปเมนต์ทูลคิด (Java Development Toolkit : JDK) เวอร์ชัน 1.2 ส่วนประกอบที่ใช้ติดต่อกับผู้ใช้งานทั้งหมดเช่น เท็กซ์ฟิลด์ ลิสต์ แผนภาพแบบต้นไม้ เป็นคลาสต่างๆในแพ็กเก็ตสวิง (swing)

4.1 คลาส "BehaviorInfo"

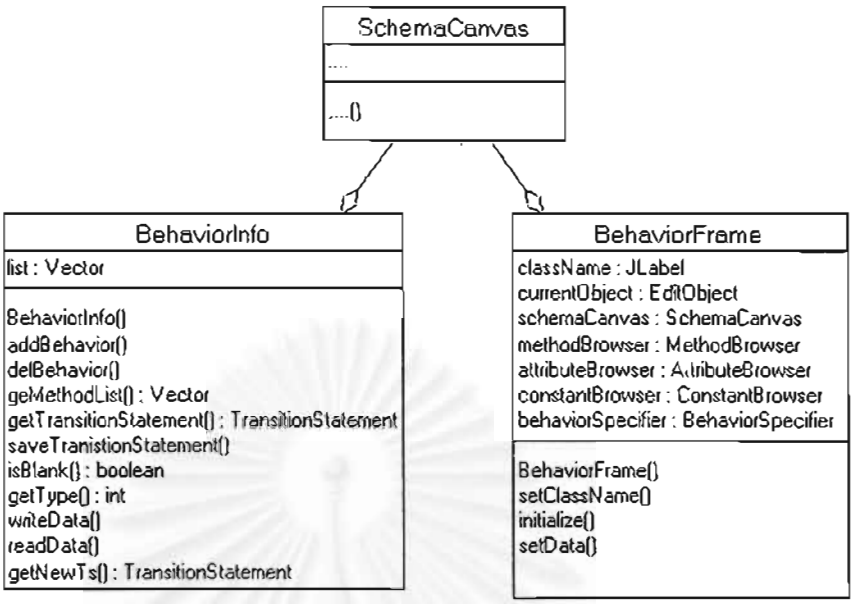
รายละเอียดของคลาส "BehaviorInfo" แสดงในรูปที่ 4.1 เป็นคลาสที่ใช้เก็บและจัดการข้อมูลพฤติกรรมภายในประกอบด้วยเวคเตอร์ "list" พร้อมทั้งมีวิธีการต่างๆที่ใช้จัดการข้อมูลเหล่านั้นเช่น การจัดเก็บประโยคการเปลี่ยนแปลงลงเวคเตอร์ (saveTransitionStatement) การเพิ่มประโยคการเปลี่ยนแปลงที่ต้องการ (addTransitionStatement) การลบประโยคการเปลี่ยนแปลง (delBehavior) เป็นต้น ชุดคำสั่งของวิธีการเหล่านี้แสดงในรูปที่ 4.2

4.2 คลาส "BehaviorFrame"

รายละเอียดของคลาส "BehaviorFrame" แสดงในรูปที่ 4.1 ถือเป็นคลาสที่เป็นที่รวมขององค์ประกอบต่างๆในระบบ วิธีการหลักในคลาสนี้ได้แก่ "initailize" ซึ่งจะทำการสร้างวัตถุต่างๆที่เป็นองค์ประกอบภายในและจัดรูปแบบการแสดงผล ชุดคำสั่งในวิธีการ "initialize" แสดงดังรูปที่ 4.3

4.3 คลาสสำหรับจัดการประโยคการเปลี่ยนแปลงชนิดต่างๆ

รายละเอียดของคลาสสำหรับจัดการข้อมูลพฤติกรรมแสดงดังรูปที่ 4.4 ทุกคลาสจะมีคลาสแม่เดียวกันคือคลาส "TransitionStatement" และมีวิธีการที่มีชื่อเหมือนกันเช่น "assign" "readData" และ "writeData" วิธีการเหล่านี้จะถูกเขียนทับ (override) ในคลาสลูกซึ่งจะมีรายละเอียดของคำสั่งในการกระทำแตกต่างกันไปตามชนิดของประโยคการเปลี่ยนแปลง รูปที่ 4.5 แสดงชุดคำสั่งบางส่วนในคลาส "TransitionStatement" ซึ่งจะเห็นว่าไม่มีคำสั่งใดๆในวิธีการ "assign" "readData" และ "writeData" ส่วนรูปที่ 4.6 แสดงชุดคำสั่งบางส่วนในคลาส "TransitionRule" โดยแสดงการเขียนทับวิธีการในคลาสแม่ สำหรับคลาสลูกคลาสอื่นๆชุดคำสั่งในวิธีการเหล่านี้จะแตกต่างจากที่แสดงในรูปที่ 4.6



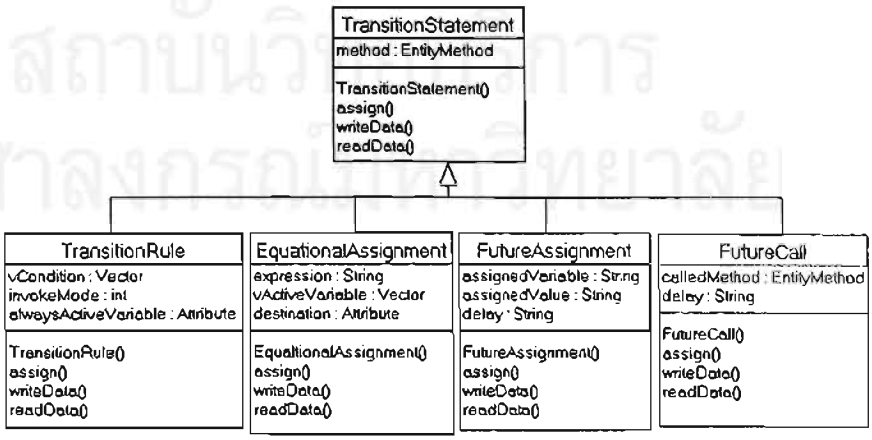
รูปที่ 4.1 แสดงรายละเอียดภายในคลาส "BehaviorInfo" และ "BehaviorFrame"

```
public class BehaviorInfo{
  Vector list;
  ...
  public void addBehavior(TransitionPanel tp,TransitionStatement ts){
    ((Vector) list.elementAt(getType(tp))).addElement((Object) ts);
  }

  public void delBehavior(TransitionPanel tp,TransitionStatement ts){
    ((Vector) list.elementAt(getType(tp))).removeElement(ts);
  }

  public void saveTransitionStatement(TransitionPanel tp, int index, TransitionStatement ts){
    TransitionStatement tmp;
    tmp = (TransitionStatement) ((Vector) list.elementAt(getType(tp))).elementAt(index);
    tmp.assign(ts);
  }
public int getType(TransitionPanel tp){
    if(tp instanceof TransitionRulePanel)
       return 0;
    if(tp instanceof EquationPanel)
       return 1;
    if(tp instanceof FutureAssignmentPanel)
       return 2;
    if(tp instanceof FutureCallPanel)
       return 3;
    return 0;
  }
}
```

รูปที่ 4.2 แสดงรายละเอียดของวิธีการที่สำคัญในคลาส "BehaviorInfo"

```
public class BehaviorFrame extends JFrame{
   Container container = null;
   JLabel className;
   EditObject currentObject = null;
   SchemaCanvas schemaCanvas;
   MethodBrowser methodBrowser;
   AttributeBrowser attributeBrowser;
   ConstantBrowser constantBrowser;
   BehaviorSpecifier behaviorSpecifier;
  ....
  private void initialize(){
      schemaCanvas.portInformation();
      this.setSize(1000,700);
      this.setResizable(false);
      container = this.getContentPane();
      //Set class name label according to the current object
      className = new JLabel("",JLabel.LEFT);
      className.setForeground(Color.red);
      className.setFont(new Font("Sans", Font.PLAIN,22));
      setClassName();
      container.add(className,BorderLayout.NORTH);

      //set selected behavior info
      setData();

      //instanciate browsers object and behavior Specifier
      behaviorSpecifier = new BehaviorSpecifier(this);
      methodBrowser = new MethodBrowser(this);
      attributeBrowser = new AttributeBrowser(this);
      constantBrowser = new ConstantBrowser(this);

      //set layout of the screen
      JPanel browserPanel = new JPanel();
      browserPanel.setLayout(new BorderLayout());
      browserPanel.setSize(new Dimension(100,50));
      browserPanel.add(methodBrowser,BorderLayout.WEST );
      browserPanel.add(attributeBrowser,BorderLayout.CENTER );
      browserPanel.add(constantBrowser,BorderLayout.EAST );
      container.add(browserPanel);
      container.add(behaviorSpecifier,BorderLayout.SOUTH);
   }
  ...
}
```

รูปที่ 4.3 แสดงวิธีการ "initialize" ในคลาส "BehaviorFrame"



รูปที่ 4.4 แสดงรายละเอียดของคลาสสำหรับจัดการประโยคการเปลี่ยนแปลงชนิดต่างๆ

```
public class TransitionStatement{
 public EntityMethod method;

 TransitionStatement(String methodName, String category){
  this.method = new EntityMethod(category,methodName);
 }

 TransitionStatement(){}

 public void assign(TransitionStatement ts){    //To be overriden in subclass
 }

 public void writeData(DataOutputStream out)throws IOException{   //To be overriden in subclass
 }

 public void readData(DataInputStream in)throws IOException{     //To be overriden in subclass
 }
}
```

รูปที่ 4.5 แสดงวิธีการในคลาส "TransitionStatement"

```
public class TransitionRule extends TransitionStatement{
 static final int ALWAYS_INVOKE = 0;
 static final int UPON_CONDITION =1;
 Vector vCondition;
 int invokeMode;
 Attribute alwaysActiveVariable;
 ...
 public void assign(TransitionStatement ts){
  TransitionRule source;
  source = (TransitionRule) ts;
  this.method = source.method;
  this.vCondition = source.vCondition;
  this.invokeMode = source.invokeMode;
  this.alwaysActiveVariable = source.alwaysActiveVariable;
 }

 public void writeData(DataOutputStream out)throws IOException{
  Vector tmpVector;
  out.writeUTF(method.name);
  out.writeUTF(method.category);
  out.writeInt(invokeMode);
  alwaysActiveVariable.writeData(out);
  out.writeInt(vCondition.size());
  for(int i= 0 ; i< vCondition.size();i++){
   tmpVector = (Vector) vCondition.elementAt(i);
   out.writeInt(tmpVector.size());
   for(int j = 0 ; j < tmpVector.size() ; j++){
    if(j == 0)
     ((Attribute) tmpVector.elementAt(j)).writeData(out);
    else
     out.writeUTF((String) tmpVector.elementAt(j));
   }
  }
 }
```

รูปที่ 4.6 แสดงชุดคำสั่งในคลาส "TransionRule"

```
public void readData(DataInputStream in)throws IOException{
   Attribute tmpAttribute;
   Vector colData;
   Vector rowData = new Vector(10,10);
   String sMethodName =in.readUTF();
   String sCategory = in.readUTF();
   this.method = new EntityMethod(sCategory,sMethodName);
   this.invokeMode = in.readInt();
   this.alwaysActiveVariable = new Attribute();
   alwaysActiveVariable.readData(in);
   int iRow = in.readInt();
   for (int i = 0; i<iRow ; i++){
    int iCol = in.readInt();
    colData = new Vector(10,10);
    for(int j = 0; j<iCol ; j++){
     if(j == 0){
      tmpAttribute = new Attribute();
      tmpAttribute.readData(in);
      colData.addElement((Object) tmpAttribute);
     }else
      colData.addElement((Object) in.readUTF());
    }
    rowData.addElement((Object) colData);
   this.vCondition = rowData;
   }
  }
}
```

รูปที่ 4.6 แสดงชุดคำสั่งในคลาส "TransionRule" (ต่อ)

4.4 คลาส "MethodBrowser"

รายละเอียดของคลาส "MethodBrowser" แสดงในรูปที่ 4.7 คลาสนี้จะอ่านข้อมูลฟังก์ชันสมาชิกจากแฟ้มข้อมูลเอนทิตี และแสดงรายการฟังก์ชันเหล่านั้นในรูปลิสต์ แยกตามชนิดว่าเป็นชนิดโมเดลหรือวิว ผู้ใช้งานสามารถเลือกดับเบิลคลิกฟังก์ชันเหล่านี้ในลิสต์เพื่อใช้กำหนดพฤติกรรม ภายในคลาสนี้ประกอบด้วยคลาสย่อย "MethodListModel" ซึ่งทำหน้าที่เป็นตัวเชื่อมระหว่างข้อมูลซึ่งอยู่ในรูปเวคเตอร์ และการแสดงผลข้อมูลนั้นในลิสต์ ข้อมูลที่อยู่ในเวคเตอร์เป็นวัตถุในคลาส "EntityMethod" ซึ่งวัตถุแต่ละตัวใช้เก็บชื่อ และ ชนิดของฟังก์ชันสมาชิก

4.5 คลาส "ConstantBrowser"

รายละเอียดของคลาส "ConstantBrowser" แสดงในรูปที่ 4.7 คลาสนี้มีหลักการทำงานคล้ายกับ "MethodBrowser" แต่ข้อมูลที่จะนำมาแสดงจะเป็นข้อมูลสมาชิกที่เป็นค่าคงที่ โดยพิจารณาจากข้อมูลที่มีส่วนขยายเป็น สแตติก ไฟนอล (static final) ซึ่งแสดงว่าเป็นข้อมูลสมาชิกระดับคลาส และจะไม่ถูกแก้ไขยีก ข้อมูลสมาชิกดังกล่าวจะแสดงในลิสต์ แยกตามชนิดโมเดลและวิวเช่นเดียวกัน

4.6 คลาส "AttributeBrowser"

รายละเอียดของคลาส "AttributeBrowser" แสดงในรูปที่ 4.7 ภายในคลาสนี้มีวัตถุชนิด "JTree" ที่สามารถแสดงข้อมูลต่างๆในรูปแบบต้นไม้ได้ การทำงานจะเริ่มจากการอ่านข้อมูลสมาชิกภายในเอนทิตีทั้งหมดที่ไม่ใช่ค่าคงที่ และแบ่งกลุ่มออกเป็นชนิดโมเดลและชนิดวิว สำหรับข้อมูลสมาชิกแต่ละตัว ถ้าตัวใดมีชนิดเป็นเอนทิตีอื่นในระบบเดียวกัน หรือเป็นคลาสในแพ็คเกจ "gui" ในระบบวัตถุพร้อมทำงานก็จะสร้างต้นไม้ย่อยๆของข้อมูลนั้นวนซ้ำไป (recursive) จนกว่าข้อมูลที่เป็นหน่วยใบ (leave node) ของต้นไม้จะเป็นข้อมูลพื้นฐานตัวแปรพร้อมทำงาน หรือเป็นคลาสอื่นๆ นอกจากนี้ยังมีการพิจารณาความสัมพันธ์กับเอนทิตีรอบข้างด้วยเพื่อสร้างกลุ่มข้อมูล "Input" และ "Output" ที่สอดคล้องกับความสัมพันธ์ที่แสดงในแผนภาพเอนทิตีและความสัมพันธ์ตามข้อกำหนดที่แสดงไว้ในบทที่ 3 ด้วย

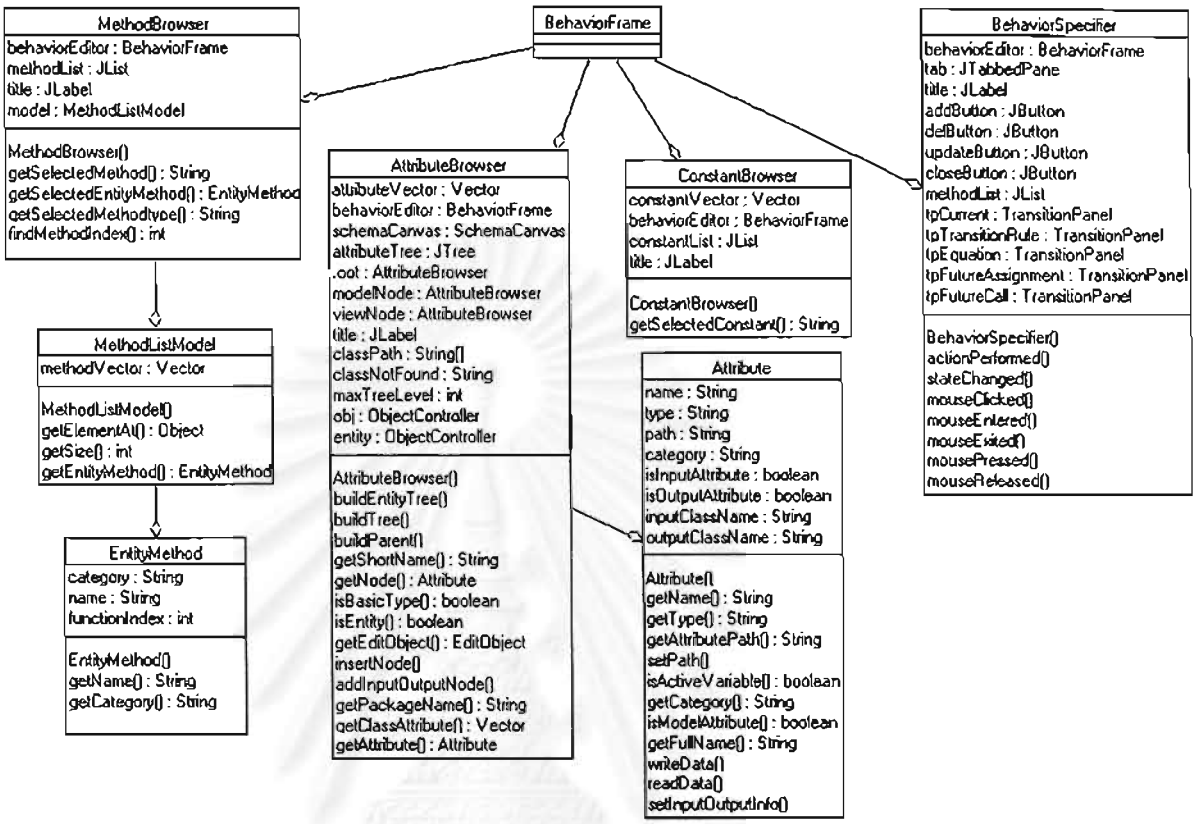
ชุดคำสั่งของวิธีการที่ใช้ในการแสดงข้อมูลคุณลักษณะของเอนทิตีแบบลำดับชั้นแสดงในรูปที่ 4. 8 ได้แก่ วิธีการ "buildEntityTree" โดยจะเริ่มจากการสร้างสร้างหน่วยข้อมูล "Model" ที่แสดงคุณลักษณะเฉพาะในคลาสโมเดล หน่วยข้อมูล"View"ที่แสดงคุณลักษณะเฉพาะของคลาสวิว และ "Parent" ที่แสดงต้นไม้ย่อยของคลาสแม่ ภายในวิธีการนี้จะเรียกไปยังวิธีการ "buildTree" เพื่อสร้างต้นไม้ย่อยๆของคุณลักษณะแต่ละตัวภายในหน่วยข้อมูลต่างดังที่ได้กล่าวแล้ว

แต่ละหน่วย (node) ของต้นไม้เป็นวัตถุในคลาส "Attribute" สืบทอดคุณสมบัติจากคลาส "DefaultMutableTreeNode" ซึ่งเป็นคลาสที่มีอยู่แล้วในแพ็คเกจสวิงใช้สำหรับเป็นหน่วยของต้นไม้โดยเฉพาะ คลาส "Attribute" มีการเพิ่มเติมข้อมูลรายละเอียดของคุณลักษณะเช่น ชื่อ ทางเดินจากรากของต้นไม้ถึงหน่วยชนิดโมเดลหรือวิว เป็นต้น

4.7 คลาส "BehaviorSpecifier"

รายละเอียดของคลาส "BehaviorSpecifier" แสดงในรูปที่ 4.7 ส่วนรายละเอียดของคลาสที่เป็นองค์ประกอบภายในแสดงในรูปที่ 4.9 คลาส "BehaviorSpecifier" นี้สืบทอดคุณสมบัติจากคลาส "JPanel" ใช้เพื่อเป็นที่รวมของของวัตถุต่างๆที่ประกอบกันเป็นหน้าจอสำหรับกำหนดพฤติกรรม องค์ประกอบที่สำคัญได้แก่ ลิสต์ ที่ใช้แสดงประโยคการเปลี่ยนแปลงที่ได้สร้างไว้แล้ว และแท็ป ซึ่งเป็นองค์ประกอบที่แสดงหน้าจอหลายหน้าจอซ้อนทับกัน ผู้ใช้งานสามารถเลือกหน้าจอที่ต้องการได้โดยคลิกเลือกที่ชื่อหน้าจอที่ต้องการ ภายในแท็ปประกอบด้วยวัตถุของคลาส 4 คลาสได้แก่ "TransitionRulePanel" "EquationPanel" "FutureAssignmentPanel" และ "FutureCallPanel" โดยที่ทุกๆคลาสสืบทอดคุณสมบัติจากคลาสแม่เดียวกันคือ "TransitionPanel" คลาสแต่ละชนิดมีหน้าที่จัดการเกี่ยวกับการรับข้อมูลและการแสดงผลประโยคการเปลี่ยนแปลงชนิดต่างๆโดยการเขียนทับวิธีการต่างๆที่ประกาศไว้แทนคลาสแม่เช่น "showText()" "saveText()" "isDataValid" เป็นต้น

รูปที่ 4.10 แสดงชุดคำสั่งบางส่วนของวิธีการต่างๆในคลาส "TransitionPanel" จะเห็นว่าวิธีการบางวิธีการจะเว้นว่างไว้ โดยจะปล่อยให้คลาสลูกเป็นคลาสที่จะจัดการการกระทำเหล่านั้นรูปที่ 4.11 แสดงชุดคำสั่งคลาส "TransitionRulePanel" ซึ่งแสดงให้เห็นวิธีการต่างๆที่เขียนทับวิธีการของคลาสแม่



รูปที่ 4.7 แสดงรายละเอียดของคลาสต่างๆที่เป็นองค์ประกอบในคลาส "BehaviorFrame"

```
public class AttributeBrowser extends JPanel{
  static final String classNotFound = "NotFound";
  static final int maxTreeLevel = 6;
  Vector attributeVector;
  BehaviorFrame behaviorEditor;
  SchemaCanvas schemaCanvas;
  JTree attributeTree;
  Attribute root;
  Attribute viewNode = null;
  Attribute modelNode = null;
  JLabel title;
  JScrollPane scrollPane;
  static String classPath[] = {"saos.gui","saos.kernel","java.lang","java.util","java.awt"};
  ObjectController obj;
  ObjectController entity = new ObjectController();
```

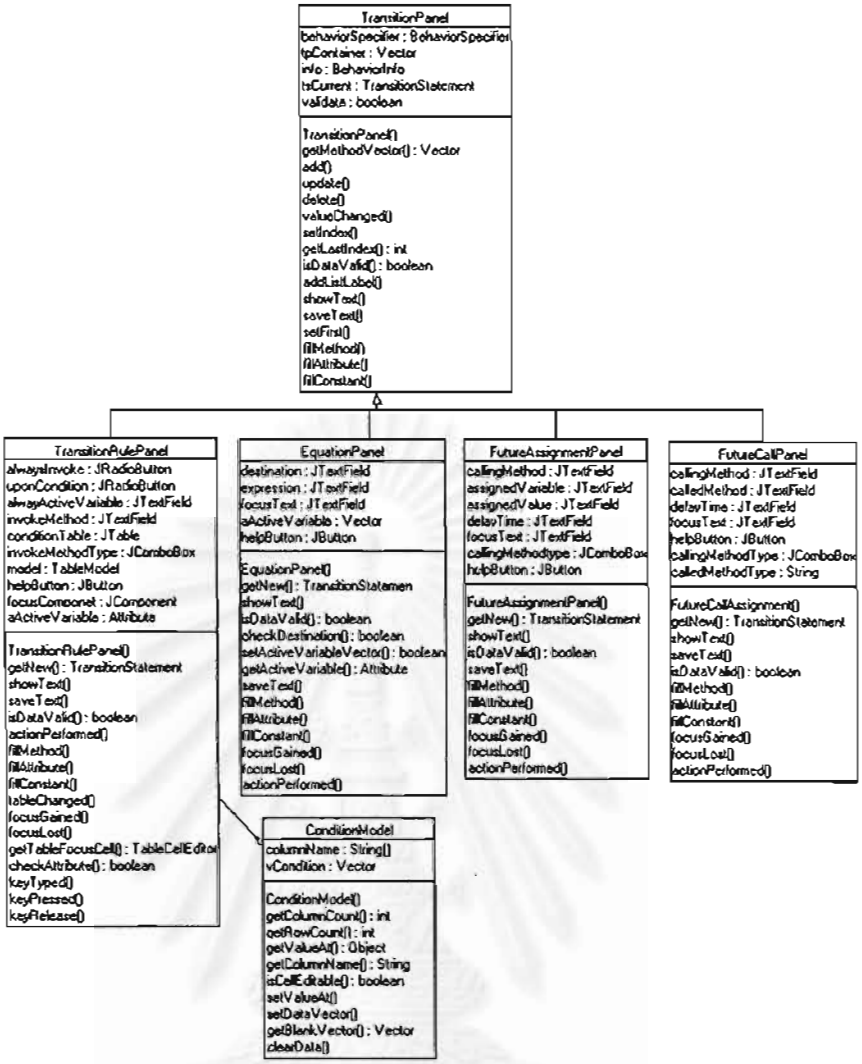
รูปที่ 4. 8 แสดงชุดคำสั่งในคลาส "AttributeBrowser"

```
private void buildEntityTree(Attribute node){
   Attribute tmpMainNode;
   String parent;
   boolean containModelAttribute;
   Attribute modelNode,viewNode;
  if(node.equals(root))
    entity.setObject(behaviorEditor.currentObject);
   else
    entity.setObject(getEditObject(node.getType()));

   //check whether the current entity contains any attribute in model class
   containModelAttribute = entity.containModelAttribute();

   //get the parent name of the current entity
   parent = entity.getParent();
   if(containModelAttribute == true || parent.equals("AObject") == false){
    modelNode = new Attribute("Model","","","Model");
    modelNode.setInputOutputInfo(node);

    //add model node in the tree
    node.add(modelNode);

    //call build tree to recursively build subtree of the current node
    buildTree(modelNode,null);
    tmpMainNode = modelNode;
    if(node.equals(root))
      this.modelNode = modelNode;
   }else
    tmpMainNode = node;

   //if the entity contains any attribute in view class
   //insert view node and build subtree
   if(entity.containViewAttribute() == true){
    viewNode = new Attribute("View","","","View");
    viewNode.setInputOutputInfo(node);
    node.add(viewNode);
    if(node.equals(root))
      this.viewNode = viewNode;
    buildTree(viewNode,null);
   }
   //if the parent of the current node is not "AObject"
   //build subtree for the parent
   if(parent.equals("AObject") == false){
    Attribute parentNode = new Attribute("Parent",parent,"","Model");
    parentNode.setInputOutputInfo(node);
    buildTree(parentNode,tmpMainNode);
   }
  }

  private void buildTree(Attribute node,Attribute mainNode){
   String type = node.getType();
   String parent = null;
   Vector AttributeVector = new Vector(10,10);
   Attribute childNode;
   String parentPackage = null;
   boolean isEntity;
   boolean isGui = false;

   //add the node in the tree
   if(mainNode != null){
    mainNode.add(node);
   }
```

รูปที่ 4. 8 แสดงชุดคำสั่งในคลาส "AttributeBrowser" (ต่อ)

```
isEntity = isEntity(type);
   //if the current node is an entity,  build sub tree for the
   //endtity
   if(isEntity){
    if(node.getLevel () < maxTreeLevel)
      buildEntityTree(node);
   //if the current node is the model node
   //get the list of attribute in model class
   }else if(node.getName().equals("Model")){
    AttributeVector = entity.getModelAttribute();//get Model Attribute
//if the current node is the view node
   //get the list of attribute in view class
  }else if(node.getName().equals("View")){
    AttributeVector = entity.getViewAttribute();//get View Attribute

   //the current node is a class in gui package
   //get the attribute of the class
  }else{// It is a class
    Class nodeClass = null;
    String sPackageName = getPackageName(node);
    if(sPackageName != null && sPackageName.equals(classNotFound) == false &&
     sPackageName.equals("saos.gui")){
     isGui = true;
     try{
      nodeClass = Class.forName(sPackageName + "." + node.getType());
      if(nodeClass.getSuperclass() != null)
       parentPackage = getPackageName(new Attribute("",
       getShortName(nodeClass.getSuperclass().getName()),"",""));
     }catch (ClassNotFoundException e){
      System.out.println("Error : " + e);
     }
     parent = getShortName(nodeClass.getSuperclass().getName());
    }
    AttributeVector = getClassAttribute(node);
   }

   // for each attribute in the attribute vector, if it is a basic type
   // add a new node for the attribute into tree, other wise build subtree  for the attribute
   for (int i = 0; i< AttributeVector.size(); i++){
    childNode = (Attribute) AttributeVector.elementAt(i);
    childNode.isInputAttribute = node.isInputAttribute;
    childNode.inputClassName = node.inputClassName;
    childNode.isOutputAttribute = node.isOutputAttribute;
    childNode.outputClassName = node.outputClassName;
    if (isBasicType(childNode) == false && node.getLevel() < maxTreeLevel)
     buildTree(childNode,node);
    else{
     node.add(childNode);
     childNode.setPath();
    }
   }

   if(isGui && parent.equals("AObject") == false &&
    parentPackage != null && parentPackage.equals("saos.gui"))
    buildParent(node,parent);

 }

 private void buildParent(Attribute node, String parent){
  Attribute parentNode = new Attribute("Parent",parent,"",node.category);
  parentNode.setInputOutputInfo(node);
  buildTree(parentNode,node);
 }
...
}
```

รูปที่ 4. 8 แสดงชุดคำสั่งในคลาส "AttributeBrowser" (ต่อ)



รูปที่ 4.9 แสดงรายละเอียดของคลาสต่างๆที่ใช้สำหรับกำหนดพฤติกรรม

```
public abstract class TransitionPanel extends JPanel
   implements ListSelectionListener{
  BehaviorSpecifier behaviorSpecifier;
  Vector tpContainer;
  BehaviorInfo info;
  TransitionStatement tsCurrent;
  boolean validate = true;
  ...
  public boolean isDataValid(){
     return true;
  }
  public TransitionStatement getNew(){
      return new TransitionStatement("<method>","");
  }
  public void showText(){ // to be overidden in subclass  }
  public void saveText(int index){ // to be overidden in subclass }
 public void fillMethod(MouseEvent e){ // to be overidden in subclass }
 public void fillConstant(MouseEvent e){ // to be overidden in subclass }
  public void fillAttribute(MouseEvent e){ // to be overidden in subclass }
 }
```

รูปที่ 4.10 แสดงชุดคำสั่งในคลาส "TransitionPanel"

```
public class TransitionRulePanel extends TransitionPanel implements ActionListener
        ,FocusListener,TableModelListener,KeyListener{

 JRadioButton alwaysInvoke , uponCondition;
 JTextField alwaysActiveVariable, invokedMethod;
 JTable conditionTable;
 JComboBox invokedMethodType;
 TableModel model;
 JButton helpButton;
 JComponent focusComponent;
 Attribute aActiveVariable;

 public TransitionStatement getNew(){
  Vector tmp = ((ConditionModel) model).getBlankVector();
  return new TransitionRule("<method>","Model",tmp);
 }
 public void showText(){
  TransitionRule trCurrent = (TransitionRule) tsCurrent;
  if (trCurrent != null){
   invokedMethod.setText(trCurrent.method.name);
   invokedMethodType.setSelectedItem(trCurrent.method.category);
   alwaysActiveVariable.setText(trCurrent.alwaysActiveVariable.path);
   ((ConditionModel) model).setDataVector(trCurrent.vCondition);
   if (trCurrent.invokeMode == TransitionRule.ALWAYS_INVOKE)
     alwaysInvoke.doClick();
   else
     uponCondition.doClick();
  }else{
   alwaysActiveVariable.setText(null);
   uponCondition.doClick();
   ((ConditionModel) model).clearData();
   invokedMethod.setText(null);
   invokedMethodType.setSelectedIndex(0);
  }
   conditionTable.repaint();
 }
 public void saveText(int index){
  TransitionRule tr = (TransitionRule) tsCurrent;
  tr.method.name = this.invokedMethod.getText();
  tr.method.category = (String) this.invokedMethodType.getSelectedItem();
  if (alwaysInvoke.isSelected()){
   tr.invokeMode = TransitionRule.ALWAYS_INVOKE;
   tr.alwaysActiveVariable = aActiveVariable;
   tr.vCondition = ((ConditionModel) model).getBlankVector();
  }else{
   tr.invokeMode = TransitionRule.UPON_CONDITION;
   tr.alwaysActiveVariable = new Attribute();
   tr.vCondition = ((ConditionModel) model).vCondition;
  }
  tpContainer.removeElementAt (index);
  tpContainer.insertElementAt ((Object) tsCurrent.method.name,index);
  info.saveTransitionStatement(this,index,tr);
 }
 public boolean isDataValid(){
  MethodBrowser mb = behaviorSpecifier.behaviorEditor.methodBrowser;
  String sMethod = invokedMethod.getText();
  String sMethodType = (String) invokedMethodType.getSelectedItem();
  String msg = "";
  if(mb.findMethodIndex(new EntityMethod(sMethodType,sMethod)) == -1){
   msg = "Invoked method " + sMethod + " : " + sMethodType +
     " does not exist in this entity";
   JOptionPane.showConfirmDialog(this,msg,"Behavior Editor",JOptionPane.CLOSED_OPTION );
   return false;
  }
  ...
}
```

รูปที่ 4.11 แสดงชุดคำสั่งในคลาส "TransitionRulePanel"

นอกจากนี้คลาส "BehaviorSpecifier" ยังมีปุ่มต่างๆที่ใช้สำหรับ สร้าง ลบ และ จัดเก็บประโยคการเปลี่ยนแปลงทั้งหลายได้แก่ปุ่ม "Add" "Delete" และ "Update" ตามลำดับ เมื่อกดปุ่มเหล่านี้ "BehaviorSpecifier" จะส่งข้อความ "message" ไปให้คลาส "TransitionPanel" กระทำการ สร้าง ลบ หรือ จัดเก็บ ตามปุ่มที่กดรายละเอียดวิธีการเพิ่ม ลบ และแก้ไขแสดงได้ดังรูปที่ 4.12

```
public class BehaviorSpecifier extends JPanel implements ActionListener
  ,ChangeListener,MouseListener{

  BehaviorFrame behaviorEditor;
  JTabbedPane tab;
  JLabel title;
  JPanel buttonPanel;
  JButton addButton;
  JButton delButton;
  JButton updateButton;
  JButton closeButton;
  JList methodList;
  TransitionPanel tpCurrent;
  TransitionPanel tpTransitionRule;
  TransitionPanel tpEquation;
  TransitionPanel tpFutureAssignment;
  TransitionPanel tpFutureCall;

  int activeTab = 0;
  boolean stateChange = true;
  ...
  public void actionPerformed(ActionEvent e){
    int index = 0;
    String msg= "";
    String command = e.getActionCommand();
    if (command.equals("Add"))
      tpCurrent.add(methodList);
    else if (command.equals("Update"))
      tpCurrent.update(methodList);
    else if (command.equals("Delete"))
      tpCurrent.delete(methodList);
    else if (command.equals("Close")){
      if(tpCurrent.tpContainer.size() > 0){
        if(tpCurrent.update(methodList) == true)
          behaviorEditor.dispose();
      }
        else
          behaviorEditor.dispose ();
  }
...
}
```

รูปที่ 4.12 แสดงชุดคำสั่งในคลาส "BehaviorSpecifier"

4.8 คลาส "ObjectController"

รายละเอียดของคลาส "ObjectController" แสดงในรูปที่ 4.13 ข้อมูลต่างๆของเอนทิตีเช่น ชื่อ คลาสแม่ ฟังก์ชันสมาชิก และข้อมูลสมาชิก เก็บอยู่ในคลาส "EditObject" ซึ่งมีโครงสร้างการเก็บข้อมูลในเวคเตอร์ซ้อนเวคเตอร์ ซึ่งเป็นการยากสำหรับวัตถุอื่นๆในการเข้าถึงข้อมูลเหล่านี้เพราะต้องรู้ตำแหน่งที่แน่นอน คลาส "ObjectController" จึงถูกสร้างขึ้นเพื่อให้การเรียกใช้ข้อมูลในเอนทิตีต่างๆง่ายขึ้น วัตถุที่ต้องการเข้าถึงข้อมูลดัง

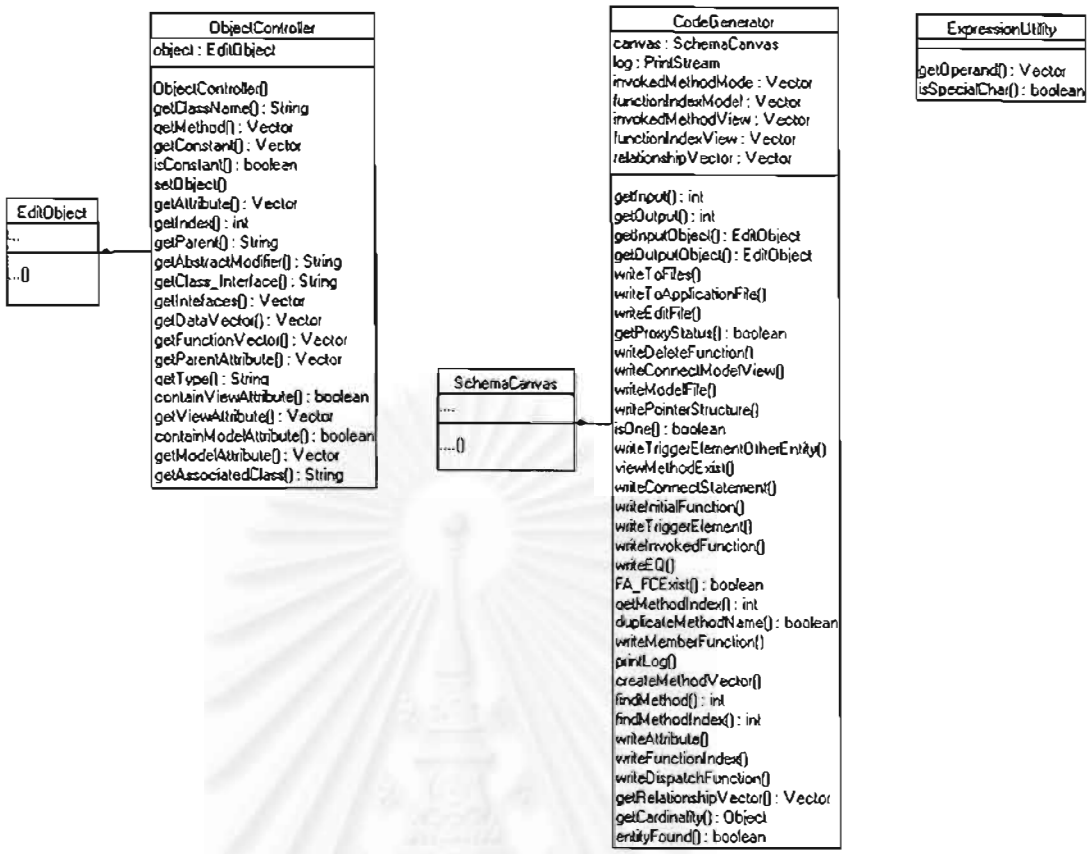
กล่าวสามารถสร้างวัตถุคลาส "ObjectController" ซึ่งจะมีการอ้างอิงไปถึงวัตถุในคลาส "EditObject" ที่ต้องการ จากนั้นจึงเรียกข้อมูลที่ต้องการผ่านวิธีการของคลาส "ObjectController" เช่นวิธีการ "getClassName()" "getMethod" "getConstant" ดังนั้นจึงมีเพียงคลาส "ObjectController" ที่จะต้องรู้ตำแหน่งที่แน่นอนของข้อมูลในเอนทิตี และวัตถุทั้งหลายเรียกใช้วิธีการต่างๆเพื่อเข้าถึงข้อมูลที่ต้องการได้ ตัวอย่างวิธีการต่างๆในคลาส "ObjectController" แสดงในรูปที่ 4.14

4.9 คลาส "CodeGenerator"

รายละเอียดของคลาส "CodeGenerator" แสดงในรูปที่ 4.13 มีหน้าที่หลักในการสร้างชุดคำสั่งของเอนทิตีต่างๆในระบบทั้งชุดคำสั่งสำหรับคลาสโมเดลและคลาสวิว เมื่อผู้ใช้งานกดปุ่มให้สร้างชุดคำสั่ง วัตถุในคลาส "SchemaCanvas" จะทำการสร้างวัตถุ "CodeGenerator" ขึ้นโดยมีการอ้างอิงถึง "SchemaCanvas" เนื่องจากข้อมูลต่างๆที่ใช้ในการสร้างชุดคำสั่งจะถูกเก็บในเวคเตอร์ต่างๆใน "SchemaCanvas" รวมถึงเวคเตอร์ใน "BehaviorInfo" ซึ่งเป็นข้อมูลเกี่ยวกับพฤติกรรมที่ได้กำหนดไว้แล้วด้วย

4.10 คลาส "ExpressionUtiltiy"

รายละเอียดของคลาส "ExpressionUtility" แสดงในรูปที่ 4.13 คลาสนี้เป็นคลาสนามธรรม (abtract class) มีวิธีการที่สำคัญได้แก่ "getOperand" วิธีการนี้จะรับพารามิเตอร์ที่เป็นสมการในรูปสตริง และทำการแยกตัวดำเนินการทั้งหมดที่ใช้ในสมการดังกล่าว จากนั้นจึงส่งค่ากลับในรูปของเวคเตอร์ของตัวดำเนินการ



รูปที่ 4.13 แสดงรายละเอียดของคลาส "ObjectController" คลาส "CodeGenerator" และคลาส "ExpressionUtility"

```
public class ObjectController{

  EditObject object;

  public ObjectController(EditObject object){
      setObject(object);
  }

  public String getClassName(){
    return (String) object.information.elementAt(1);
  }

  public Vector getMethod(){
    Vector info = new Vector(10,10);
    Vector method = new Vector(10,10);
    EntityMethod tmpMethod = null;
    String category = "";
    String name = "";
```

รูปที่ 4.14 แสดงวิธีการในคลาส "ObjectController"

```
if(object.information.size() >=7){
    info = (Vector) object.information.elementAt(6);
    for (int i=0; i<info.size(); i++){
     name = (String) ((Vector) info.elementAt(i)).elementAt(4);
     if (((Vector) info.elementAt(i)).size() >= 7)
       category =  (String) ((Vector) info.elementAt(i)).elementAt(6);
     tmpMethod = new EntityMethod(category,name);
     method.addElement((Object) tmpMethod);
    }
  }
  return method;
 }

 public Vector getConstant(){
  Vector info = new Vector(10,10);
  Vector constant = new Vector(10,10);
  String sConstant = "";
  if(object.information.size() >=7){
    info = (Vector) object.information.elementAt(5);
    for (int i=0; i<info.size(); i++)
     if (IsConstant((Vector) info.elementAt(i))){
      if(((Vector) info.elementAt(i)).size() >= 7)
        sConstant =  (String) ((Vector) info.elementAt(i)).elementAt(6) + " " +
              (String) ((Vector) info.elementAt(i)).elementAt(4);
      else
        sConstant =  "Model" + " " +
              (String) ((Vector) info.elementAt(i)).elementAt(4);
      constant.addElement((String) sConstant);
     }
  }
  return constant;
 }
 ...
}
```

รูปที่ 4.14 แสดงวิธีการในคลาส "ObjectController" (ต่อ)

บทที่ 5

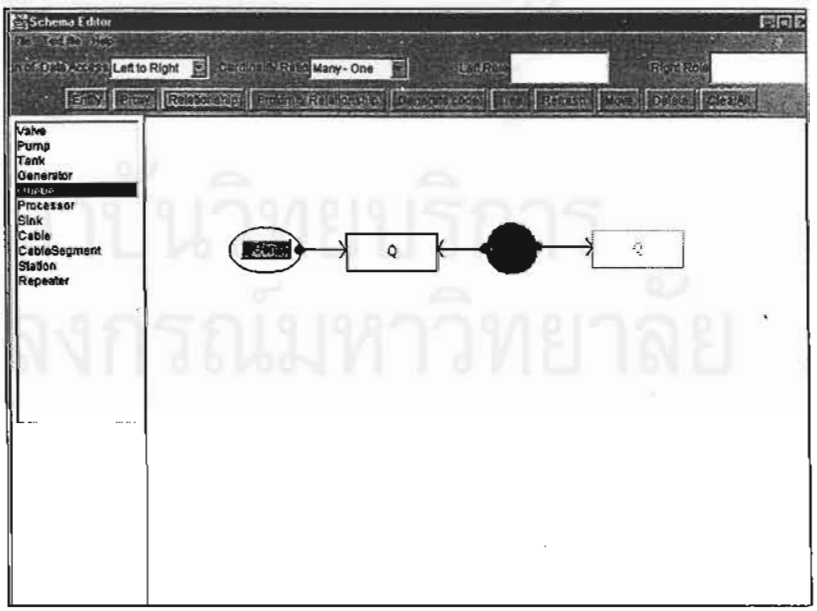
การใช้งานบรรณาธิกรณ์สำหรับกำหนดพฤติกรรมของวัตถุพร้อมทำงาน

ในบทนี้เป็นการอธิบายการใช้งานบรรณาธิกรณ์สำหรับกำหนดพฤติกรรมของวัตถุพร้อมทำงานพร้อมทั้งวิธีการในการกำหนดประโยคการเปลี่ยนแปลงชนิดต่างๆ

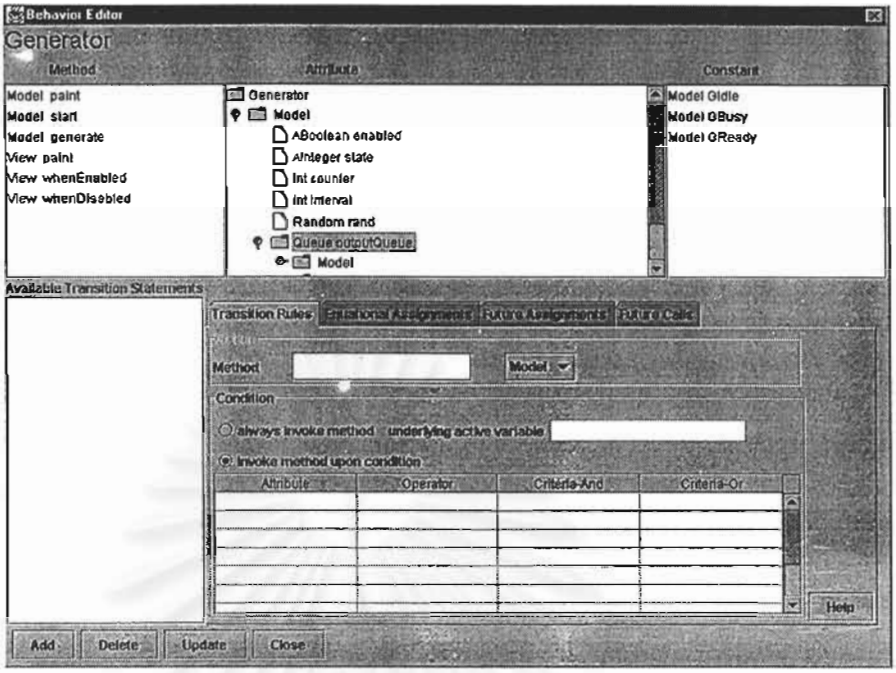
5.1 การเข้าสู่บรรณาธิกรณ์สำหรับกำหนดพฤติกรรม

บรรณาธิกรณ์สำหรับกำหนดพฤติกรรมของวัตถุพร้อมทำงานเป็นองค์ประกอบหนึ่งในบรรณาธิกรณ์สำหรับสร้างเค้าร่าง เมื่อผู้ใช้งานกำหนดแผนภาพเอนทิตีและความสัมพันธ์เรียบร้อยแล้ว จะสามารถเข้าสู่บรรณาธิกรณ์สำหรับกำหนดพฤติกรรมเพื่อสร้างประโยคการเปลี่ยนแปลงให้กับชนิดของเอนทิตีต่างๆในแผนภาพได้โดยการดับเบิลคลิกที่ชนิดของเอนทิตีที่ต้องการ หรือกดปุ่ม "Behavior Editor" จากนั้นจึงคลิกเลือกชนิดของเอนทิตีที่ต้องการ ระบบจะแสดงหน้าจอดังรูปเพื่อให้ผู้ใช้งานกำหนดรูปแบบพฤติกรรมของชนิดของเอนทิตีนั้นๆ

ตัวอย่างเช่น รูปที่ 5.1 แสดงแผนภาพเอนทิตีและความสัมพันธ์ของระบบแถวคอยที่กำหนดไว้ในบรรณาธิกรณ์สำหรับสร้างเค้าร่างเมื่อต้องการกำหนดพฤติกรรมของตัวสร้างงาน สามารถทำได้โดยดับเบิลคลิกที่สัญลักษณ์ตัวสร้างงานบนหน้าจอ ระบบจะแสดงหน้าจอดังรูปที่ 5.2 ด้านซ้ายบนสุดของหน้าจอแสดงข้อความ "Generator" ตามเอนทิตีที่ได้เลือก



รูปที่ 5.1 แสดงแผนภาพเอนทิตีและความสัมพันธ์ของระบบแถวคอยในบรรณาธิกรณ์สำหรับสร้างเค้าร่าง

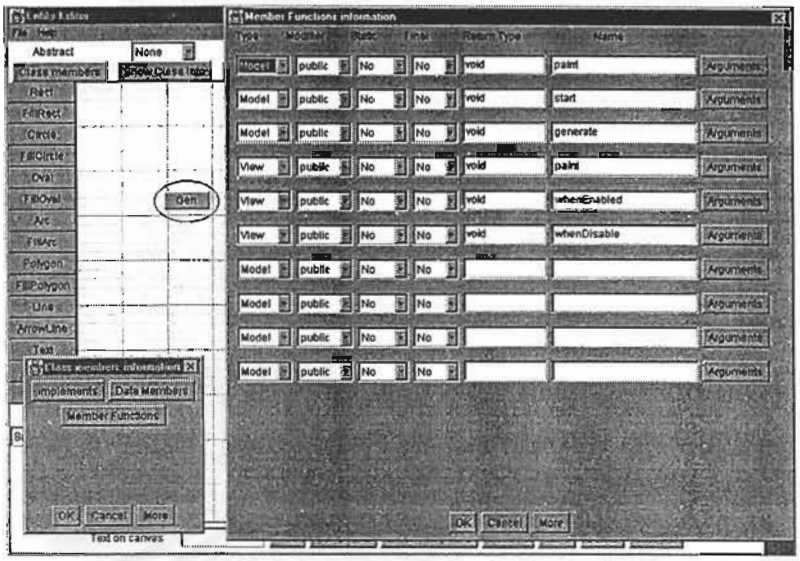


รูปที่ 5.2 แสดงหน้าจอของบรรณาธิกรณ์สำหรับกำหนดพฤติกรรมของวัตถุพร้อมทำงาน

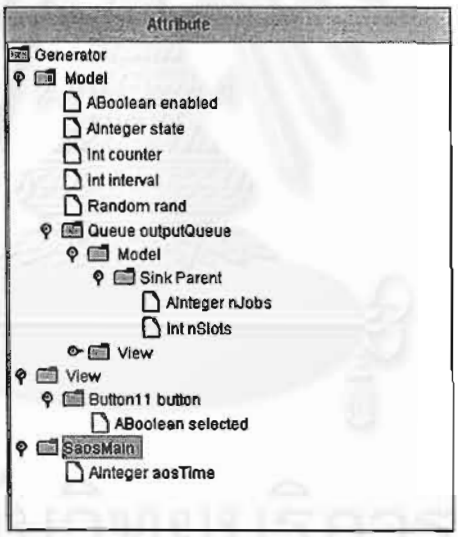
5.2 การแสดงฟังก์ชันสมาชิกและข้อมูลสมาชิก

จากรูปที่ 5.2 ส่วนบนของหน้าจอเป็นการแสดงข้อมูลต่างๆภายในตัวสร้างงาน แบ่งเป็นวิธีการ คุณลักษณะเฉพาะ และ ค่าคงที่ ข้อมูลเหล่านี้จะแบ่งย่อยออกเป็นข้อมูลในคลาสโมเดลและข้อมูลในคลาสวิวการเพิ่มเติมแก้ไขข้อมูลเหล่านี้รวมถึงการกำหนดให้มีชนิดเป็นโมเดลหรือวิว สามารถทำได้ในบรรณาธิกรณ์สำหรับสร้างชนิดของเอนทิตี รูปที่ 5.3 แสดงหน้าจอสำหรับกำหนดวิธีการในคลาสโมเดลและคลาสวิวในบรรณาธิกรณ์สำหรับสร้างชนิดของเอนทิตี เนื่องจากบรรณาธิกรณ์เดิมไม่สามารถเลือกชนิดของวิธีการและคุณลักษณะได้ ทำให้ข้อมูลทั้งหมดที่กำหนดเป็นข้อมูลในคลาสโมเดลเท่านั้น บรรณาธิกรณ์ใหม่นี้ได้มีการแก้ไขเพิ่มเติมรายการเลือกให้สามารถเลือกชนิดได้ ทำให้ผู้ใช้งานสามารถใช้ข้อมูลเหล่านี้กำหนดรูปแบบพฤติกรรมได้ทั้งในคลาสโมเดลและคลาสวิว

การแสดงรายการข้อมูลเกี่ยวกับวิธีการและค่าคงที่จะแสดงในลักษณะเป็นรายชื่อเรียงลำดับ ส่วนข้อมูลเกี่ยวกับคุณลักษณะเฉพาะแสดงในรูปแบบลำดับชั้นแบบต้นไม้ รูปที่ 5.4 แสดงลำดับชั้นคุณลักษณะเฉพาะของตัวสร้างงาน กลุ่มโมเดล (Model) แสดงคุณลักษณะเฉพาะของตัวสร้างงานที่กำหนดให้เป็นข้อมูลในคลาสโมเดล ได้แก่ "enabled" "state" "counter" "interval" และ "rand" ส่วนกลุ่มข้อมูลวิว (View) แสดงข้อมูลในคลาสวิวได้แก่ "button" มีชนิดข้อมูลเป็น "Button11" ซึ่งมิได้เป็นข้อมูลพื้นฐานหรือเป็นตัวแปรพร้อมทำงาน แต่เป็นคลาสหนึ่งในแพ็กเก็ต "gui" ดังนั้นภายในกลุ่มข้อมูล "button" จึงแสดงคุณลักษณะย่อย "selected" ซึ่งมีชนิดเป็น แอ็คทีฟบูลลีน ซึ่งเป็นระดับที่นำไปใช้กำหนดพฤติกรรมได้

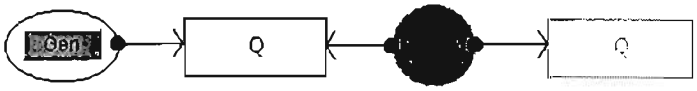


รูปที่ 5.3 แสดงการกำหนดวิธีการในคลาสโมเดลและคลาสวิว



รูปที่ 5.4 แสดงลำดับชั้นคุณลักษณะเฉพาะของชนิดของเอนทิตีตัวสร้างงาน

จากแผนภาพเอนทิตีและความสัมพันธ์ของระบบแถวคอยในรูปที่ 5.5 จะเห็นว่าผลลัพธ์ของตัวสร้างงานได้แก่แถวคอย ดังนั้นกลุ่มข้อมูลย่อย "outputQueue" จึงถูกแสดงอยู่ภายใต้กลุ่มข้อมูลโมเดล โดยที่ภายใน "outputQueue" แสดงคุณลักษณะเฉพาะต่างๆของแถวคอย และถึงแม้แถวคอยจะมีความสัมพันธ์กับตัวสร้างงานและตัวประมวลผล แต่ลูกศรที่แสดงทิศทางการเข้าถึงข้อมูลโยงจากตัวสร้างงานสู่แถวคอย และตัวประมวลผลถึงแถวคอยแต่ไม่มีในทิศทางตรงกันข้าม ดังนั้นจึงไม่มีการแสดงกลุ่มข้อมูลเข้าและผลลัพธ์ในแถวคอย



รูปที่ 5.5 แสดงแผนภาพเอนทิตีและความสัมพันธ์ของระบบแถวคอย

5.3 การเพิ่ม แก้ไข และลบประโยคการเปลี่ยนแปลง

การเพิ่มประโยคการเปลี่ยนแปลง ทำได้โดยการคลิกเลือกประเภทประโยคการเปลี่ยนแปลงที่ต้องการ จากนั้นจึงคลิกที่ปุ่ม "Add" ระบบจะแสดงค่าเริ่มต้นของข้อมูลต่างๆในประโยคการเปลี่ยนแปลงนั้น ผู้ใช้สามารถป้อนข้อมูลในฟิลด์ต่างๆด้วยตัวเอง หรือคลิกที่ฟิลด์ที่ต้องการแล้วจึงดับเบิลคลิกที่วิธีการ คุณลักษณะเฉพาะ หรือค่าคงที่จากรายการที่แสดงที่ด้านบนของหน้าจอ ข้อมูลที่เลือกจะปรากฏในช่องว่างดังที่ต้องการ

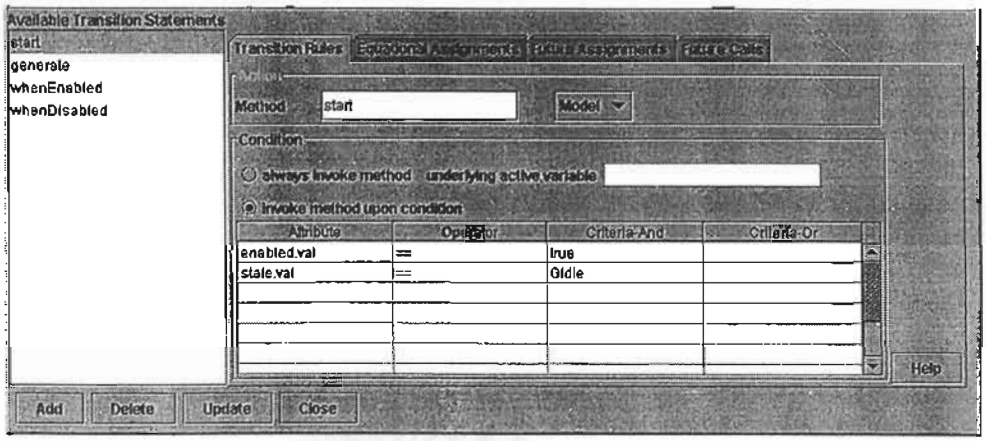
การแก้ไขประโยคการเปลี่ยนแปลงที่ได้กำหนดไว้แล้ว ทำได้โดยคลิกเลือกรายการประโยคการเปลี่ยนแปลงในรายการ "Available Transition Statements" จะปรากฏข้อมูลประโยคการเปลี่ยนแปลงที่เลือก ผู้ใช้สามารถแก้ไขได้โดยการเลือกข้อมูลต่างๆด้วยวิธีเดียวกับการเพิ่มประโยคการเปลี่ยนแปลง เมื่อแก้ไขเรียบร้อยแล้วจึงกดปุ่ม "Update" เพื่อยืนยันการแก้ไข

ผู้ใช้สามารถลบประโยคการเปลี่ยนแปลงได้โดยคลิกเลือกประโยคการเปลี่ยนแปลงที่ต้องการลบในรายการ "Available Transition Statements" จากนั้นจึงกดปุ่ม "Delete"

5.4 การกำหนดประโยคการเปลี่ยนแปลงแบบต่างๆ

5.4.1 การกำหนดกฎการเปลี่ยนแปลง

กฎการเปลี่ยนแปลงเป็นการกำหนดคู่ของเงื่อนไขและการกระทำ เมื่อเงื่อนไขที่กำหนดไว้เป็นจริง ระบบจะทำงานตามที่กำหนดไว้ในการกระทำ รูปที่ 5.6 แสดงหน้าจอสำหรับกำหนดกฎการเปลี่ยนแปลงซึ่งแบ่งออกเป็นส่วนเงื่อนไข (Condition) และส่วนกระทำ (Action) ในส่วนกระทำ ผู้ใช้ต้องเลือกชื่อของวิธีการจากรายการที่ได้กำหนดไว้สำหรับแต่ละเอนทิตี โดยอาจเป็นวิธีการชนิดโมเดลหรือวิวก็ได้ ระบบจะสร้างกลไกของกฎการเปลี่ยนแปลงในคลาสโมเดลหรือคลาสวิวตามแต่ชนิดของวิธีการที่เลือกนี้ ส่วนเงื่อนไขกำหนดใส่ในคอลัมภ์ทั้ง 4 ในตารางที่แสดงด้านล่าง คอลัมภ์แรกแสดงคุณลักษณะเฉพาะ คอลัมภ์ต่อมาแสดงตัวกระทำ (operator) ที่ใช้ในการกำหนดเงื่อนไข คอลัมภ์ที่ 3 และ 4 ใช้แสดงค่าที่ใช้ในการทดสอบเงื่อนไข ถ้ามีเงื่อนไขที่ต้องทดสอบมากกว่าหนึ่งเงื่อนไข และค่าที่ใช้ในการทดสอบเงื่อนไขอยู่ในคอลัมภ์ที่ 3 เงื่อนไขจะถูกเชื่อมแบบ "และ" แต่ถ้าเงื่อนไขที่ใช้ในการทดสอบอยู่ในคอลัมภ์ที่ 4 เงื่อนไขจะถูกเชื่อมแบบ "หรือ"



รูปที่ 5.6 แสดงหน้าจอกฎการเปลี่ยนแปลง

ตัวอย่างเช่น เงื่อนไขที่แสดงในรูปที่ 5.6 มีความหมายดังนี้คือ

if (enabled.val = = true && state.val == GIdle)

ในบางครั้งผู้พัฒนาอาจต้องการให้ระบบทำงานทันทีเมื่อตัวแปรพร้อมทำงานตัวใดตัวหนึ่งมีการเปลี่ยนแปลงค่าโดยไม่ต้องตรวจสอบเงื่อนไข กรณีนี้สามารถกำหนดได้โดยใช้ตัวเลือก "ทำงานทุกครั้ง" (always invoke method) โดยกำหนดเพียงแค่ ตัวแปรพร้อมทำงานที่ต้องตรวจสอบ (underlying active variable) เมื่อไรก็ตามที่ตัวแปรพร้อมทำงานเหล่านี้มีการเปลี่ยนแปลงค่า ระบบจะเรียกวิธีการที่กำหนดไว้ให้ทำงานทันที

การกำหนดกฎการเปลี่ยนแปลงสำหรับชนิดของเอนทิตีหนึ่งๆสามารถกำหนดไว้มากกว่าหนึ่งกฎ รายชื่อของกฎการเปลี่ยนแปลงที่กำหนดไว้แล้วแสดงไว้ในรายการประโยคการเปลี่ยนแปลงที่มีอยู่ (Available Transition Statements) กฎการเปลี่ยนแปลงเหล่านี้อาจมีวิธีการซ้ำกันได้ แต่ต้องมีตัวเลือกในการเรียกวิธีการแบบเดียวกัน เช่นต้องเป็น "เรียกวิธีการตามเงื่อนไข" (invoke upon condition) หรือเป็นตัวเลือก "ทำงานทุกครั้ง" เหมือนกันทั้งหมด ถ้าเป็นการเรียกวิธีการตามเงื่อนไข เงื่อนไขของกฎทั้งสองจะถูกเชื่อมแบบ "หรือ" ซึ่งหมายความว่าไม่ว่าเงื่อนไขใดเงื่อนไขหนึ่งเป็นจริงก็จะเรียกให้วิธีการนั้นทำงาน

5.4.2 สมการกำหนดค่า

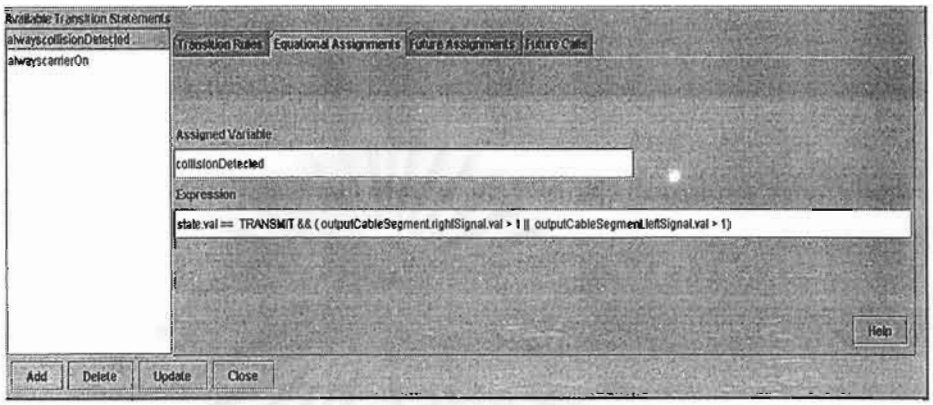
สมการกำหนดค่าเป็นการกำหนดสมการโดยที่ด้านขวาของสมการจะประกอบด้วยตัวแปรพร้อมทำงานอย่างน้อยหนึ่งตัว เมื่อตัวแปรดังกล่าวเปลี่ยนแปลงค่าจะต้องคำนวณสมการใหม่ และกำหนดผลลัพธ์ที่ได้ให้กับตัวแปรทางด้านซ้าย รูปที่ 5.7 แสดงสมการกำหนดค่าซึ่งมีรูปแบบดังนี้

```
collisionDetected = state.val == TRANSMIT && ( outputCableSegment.rightSignal.val > 1 ||
        outputCableSegment.leftSignal.val > 1)
```

"state" "outputCableSegment.rightSignal" และ "outputCableSegment.leftSignal" เป็นตัวแปรพร้อมทำงาน เมื่อตัวใดตัวหนึ่งมีการเปลี่ยนแปลงค่า ระบบจะหาผลลัพธ์ของสมการ

state.val == TRANSMIT && (outputCableSegment.rightSignal.val > 1 ||
outputCableSegment.leftSignal.val > 1)

ซึ่งอาจมีค่าเป็นจริงหรือเท็จ จากนั้นจะกำหนดให้ "collisionDetected" มีค่าเท่ากับเป็นจริงหรือเท็จตามผลลัพธ์จากการคำนวณดังกล่าว



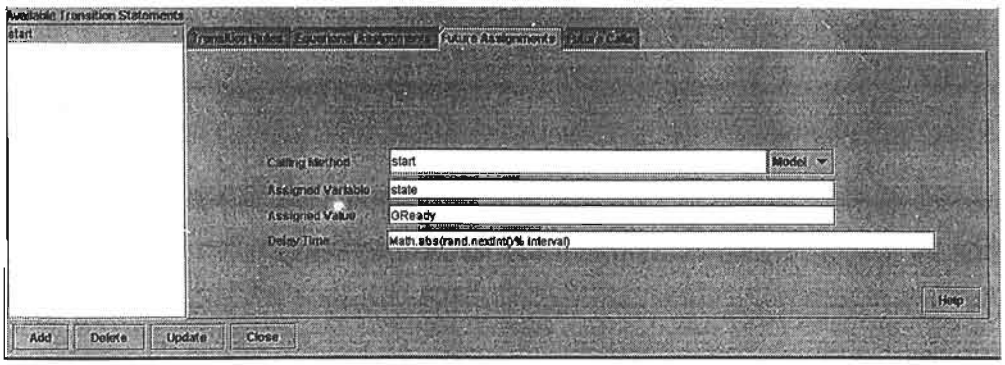
รูปที่ 5.7 แสดงหน้าจอสมการกำหนดค่า

5.4.3 การกำหนดค่าล่วงหน้า

การกำหนดค่าล่วงหน้าเป็นรูปแบบหนึ่งของการกระทำตามเหตุการณ์ เป็นการกำหนดค่าให้กับตัวแปรใดๆ โดยระบุเวลาหน่วงไว้ การกำหนดค่าจะเกิดขึ้นก็ต่อเมื่อเวลาได้ผ่านไปเท่ากับเวลาหน่วงที่ระบุ รูปที่ 5.8 แสดงการกำหนดค่าล่วงหน้า ข้อมูลที่ต้องกำหนดมีดังต่อไปนี้คือ

1) วิธีการที่เรียกการกำหนดค่าล่วงหน้า (Calling Method)
2) ตัวแปรที่ต้องการกำหนดค่า (Assigned Variable)
3) ค่าที่ต้องการกำหนด (Assigned Value)
4) เวลาหน่วง (Delay Time)

จากตัวอย่างดังรูป เป็นการกำหนดให้ "state" มีค่าเท่ากับ "GReady" เมื่อเวลาผ่านไปเท่ากับผลลัพธ์ที่ได้จากการคำนวณ "Math.abs(rand.nextInt( ) % intervale)" ซึ่งก็คือการคำนวณเวลาสุ่มนั่นเอง โดยที่คำสั่งให้เริ่มนับเวลาจะอยู่ในวิธีการ "start"



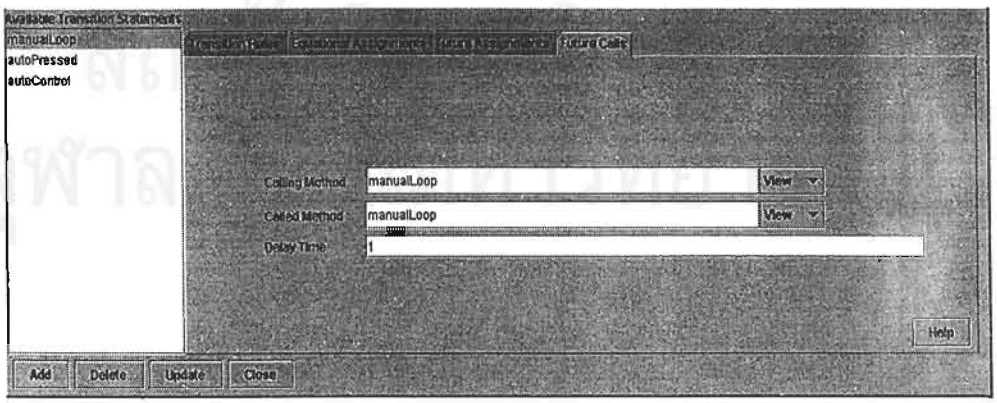
รูปที่ 5.8 แสดงหน้าจอการกำหนดค่าล่วงหน้า

5.4.4 การเรียกฟังก์ชันล่วงหน้า

การเรียกฟังก์ชันล่วงหน้าก็เป็นรูปแบบหนึ่งของการกระทำตามเหตุการณ์เช่นเดียวกับการกำหนดค่าล่วงหน้า โดยมีการกำหนดฟังก์ชันที่ต้องการเรียกพร้อมกับระบุเวลาหน่วงไว้ ฟังก์ชันดังกล่าวจะทำงานเมื่อเวลาได้ผ่านไปเท่ากับเวลาหน่วง รูปที่ 5.9 แสดงหน้าจอการเรียกฟังก์ชันล่วงหน้า ข้อมูลที่ต้องกำหนดมีดังต่อไปนี้คือ

1) วิธีการที่เรียกฟังก์ชันล่วงหน้า (Calling Method)
2) ฟังก์ชันที่ต้องการเรียก (Called Method)
3) เวลาหน่วง (Delay Time)

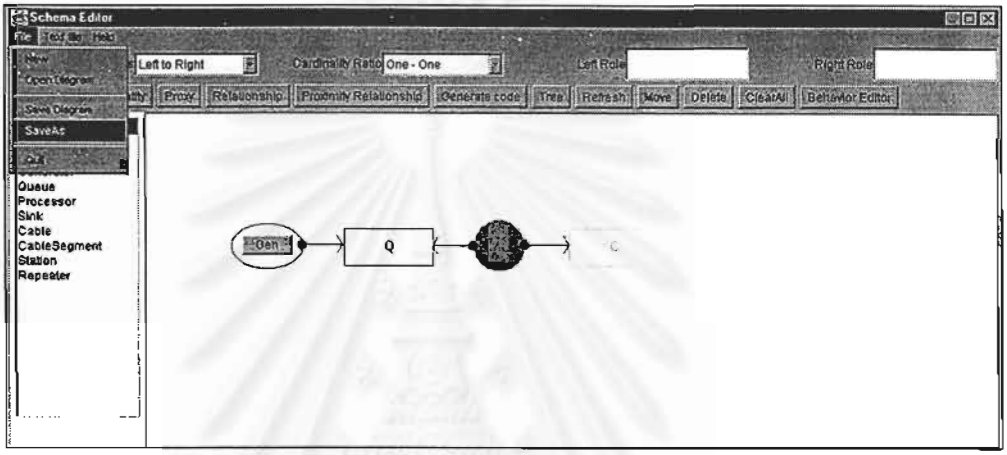
จากตัวอย่างดังรูปเป็นการกำหนดให้ เรียก "manualLoop" เมื่อเวลาผ่านไป 1 หน่วยเวลาของส่วนควบคุมการทำงาน คำสั่งในการเรียกฟังก์ชันดังกล่าวจะอยู่ภายในตัวฟังก์ชันนั้นเอง นั่นคือเมื่อระบบประมวลผลในฟังก์ชัน "manualLoop" ระบบจะเรียกฟังก์ชันนั้นอีกครั้งใน 1 หน่วยเวลาข้างหน้า หรืออาจกล่าวได้ว่าระบบจะเรียกฟังก์ชันดังกล่าววนซ้ำทุกๆ 1 หน่วยเวลา



รูปที่ 5.9 แสดงหน้าจอการเรียกฟังก์ชันล่วงหน้า

5.5 การจัดเก็บและการเรียกใช้ข้อมูลพฤติกรรม

เมื่อผู้ใช้งานต้องการจัดเก็บข้อมูลพฤติกรรมที่ได้กำหนดไว้แล้วสามารถทำได้โดยกดปุ่ม "Close" เพื่อกลับไปที่หน้าจอบรรณาธิกรณ์สำหรับสร้างเค้าร่าง จากนั้นจึงคลิกที่เมนู "File/Save As" ดังรูปที่ 5.10 จากนั้นจึงป้อนชื่อแฟ้มข้อมูลที่ต้องการ ระบบจะจัดเก็บข้อมูลแผนภาพเอนทิตีและความสัมพันธ์พร้อมทั้งข้อมูลพฤติกรรมลงในแฟ้มข้อมูลดังกล่าว



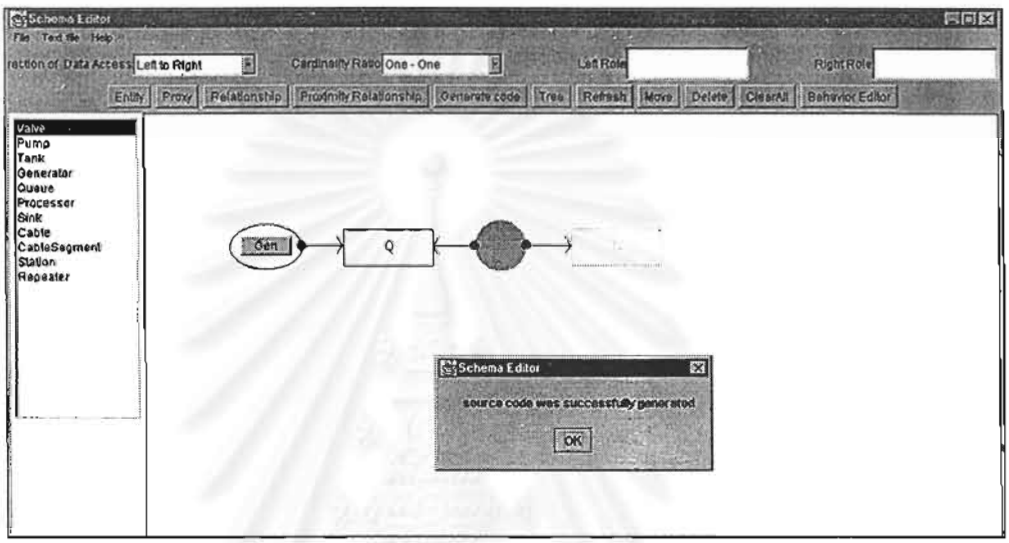
รูปที่ 5.10 แสดงการจัดเก็บข้อมูลพฤติกรรม

เมื่อต้องการเรียกใช้ข้อมูลแผนภาพเอนทิตีและความสัมพันธ์และข้อมูลพฤติกรรมจากแฟ้มข้อมูลที่เก็บไว้สามารถทำได้โดยคลิกที่เมนู "File/Open Diagram" ดังรูปที่ 5.11 จากนั้นจึงเลือกชื่อแฟ้มข้อมูลที่ต้องการ ระบบจะอ่านข้อมูลจากแฟ้มเพื่อแสดงบนหน้าจอ

รูปที่ 5.11 แสดงการเรียกใช้ข้อมูลจากแฟ้มข้อมูล

5.6 การสร้างชุดคำสั่ง

เมื่อผู้ใช้งานต้องการสร้างชุดคำสั่งสามารถทำได้โดยกดปุ่ม "Generate Code" จากหน้าจอบรรณาธิกรณ์สำหรับสร้างเค้าร่าง ระบบจะสร้างชุดคำสั่งสำหรับเอนทิตีต่างๆในแผนภาพเอนทิตีและความสัมพันธ์ที่แสดงอยู่บนหน้าจอ เมื่อสร้างชุดคำสั่งเรียบร้อยแล้ว จะแสดงข้อความดังรูปที่ 5.12 ชุดคำสั่งที่สร้างได้จะเก็บอยู่ในไดเรกทอรีเดียวกับไดเรกทอรีที่ติดตั้งบรรณาธิกรณ์สำหรับสร้างเค้าร่าง



รูปที่ 5.12 แสดงหน้าจอเมื่อสร้างชุดคำสั่งเสร็จ

บทที่ 6

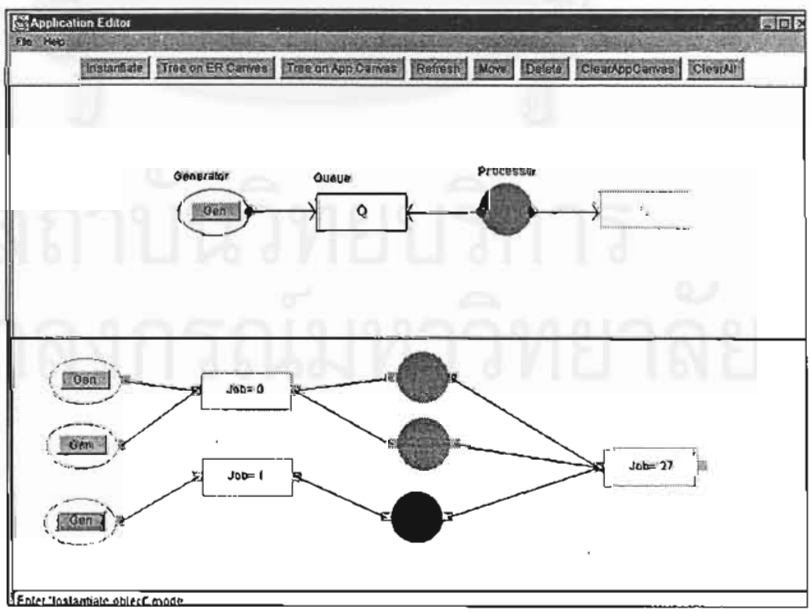
การทดสอบการสร้างชุดคำสั่ง

หลังจากพัฒนาบรรณาธิกรณ์สำหรับกำหนดพฤติกรรมเสร็จสมบูรณ์แล้ว ผู้วิจัยได้ทำการทดลองกำหนดพฤติกรรมของเอนทิตีทุกตัวในโปรแกรมระบบพร้อมทำงาน 3 โปรแกรมได้แก่ระบบแถวคอย ระบบแท็งค์ และระบบเครือข่าย ในบทนี้ได้อธิบายถึงการทำงานของโปรแกรมที่ใช้ในการทดสอบ และผลจากการทดลองสร้างชุดคำสั่งโดยยกตัวอย่างเอนทิตีต่างๆที่มีรูปแบบพฤติกรรมที่แตกต่าง ชุดคำสั่งที่แสดงในบทนี้เป็นเพียงบางส่วนของชุดคำสั่งทดลองสร้างโดยอัตโนมัติเท่านั้น ส่วนชุดคำสั่งของทุกเอนทิตีที่ใช้ในการทดสอบแสดงไว้ในภาคผนวก ก.

6.1 ภาพรวมการทำงานของโปรแกรมที่ใช้ในการทดสอบ

6.1.1 ระบบแถวคอย

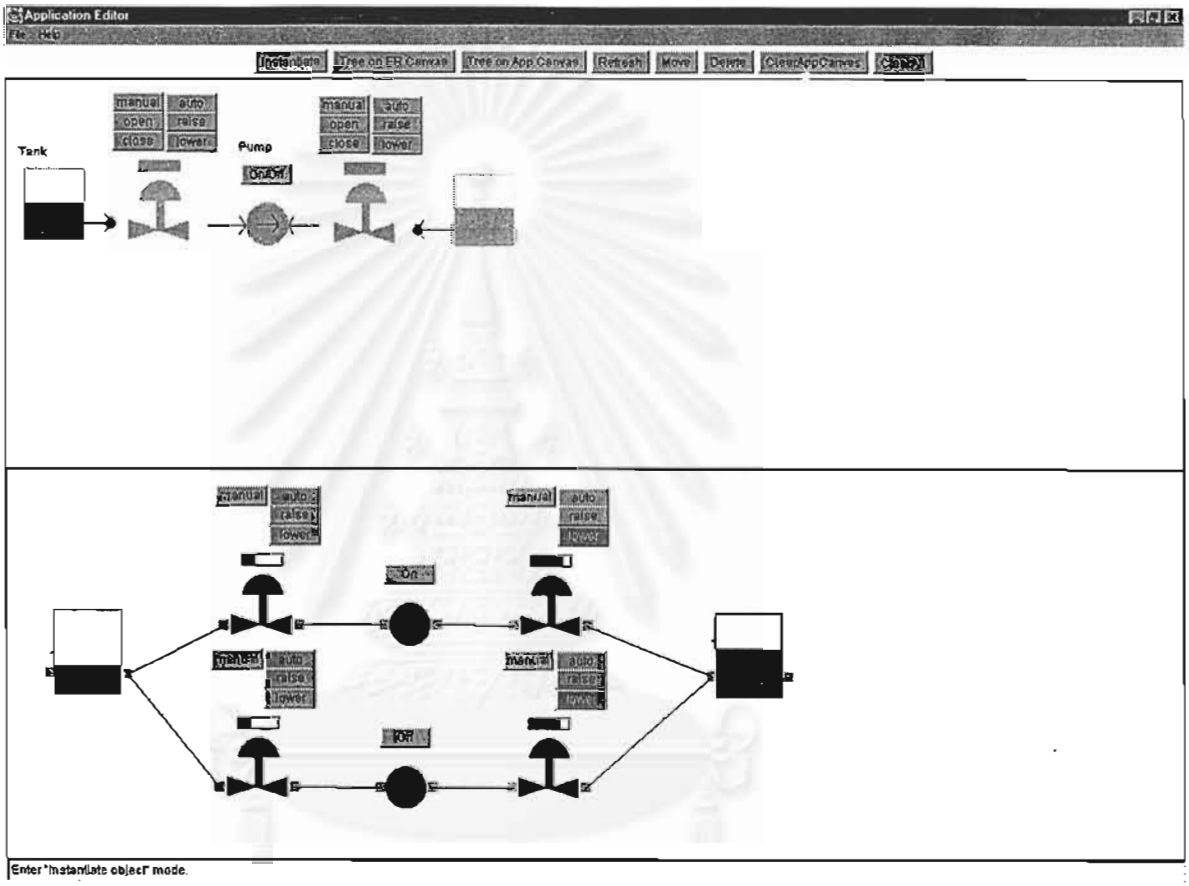
ระบบแถวคอยประกอบด้วยเอนทิตี 3 ชนิดได้แก่ ตัวสร้างงาน (Generator) แถวคอย (Queue) และตัวประมวลผล (Processor) รูปที่ 6.1 แสดงการทำงานของระบบแถวคอยที่ทำงานอยู่บนบรรณาธิกรณ์สำหรับสร้างโปรแกรมประยุกต์ การทำงานของระบบเริ่มจากตัวสร้างงานจะสร้างงานตามเวลาสุ่ม จากนั้นจึงส่งงานไปสู่แถวคอย ตัวประมวลผลแต่ละตัวจะดึงงานจากแถวคอยเพื่อประมวลผล และส่งต่อให้กับแถวคอยถัดไป



รูปที่ 6.1 แสดงระบบแถวคอยในบรรณาธิกรณ์สำหรับสร้างโปรแกรมประยุกต์

6.1.2 ระบบแท็งค์

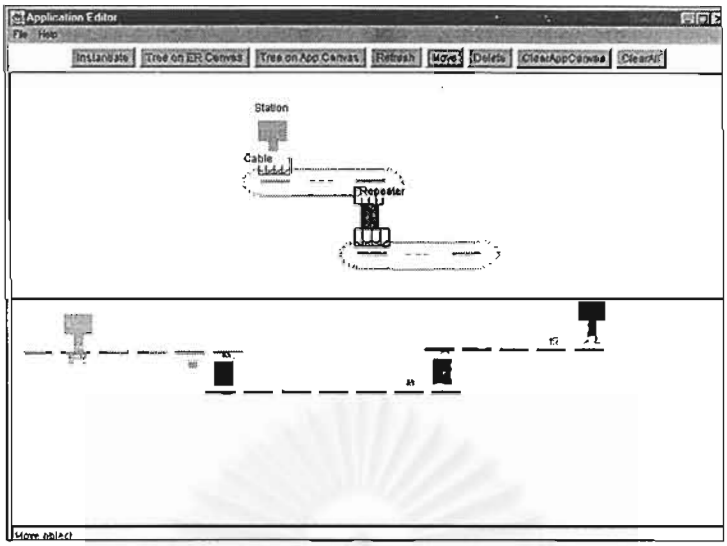
ระบบแท็งค์ประกอบด้วย แท็งค์ (Tank) ซึ่งบรรจุของเหลวโดยแสดงระดับของเหลวด้วยสี่เหลี่ยมทึบ วาล์ว(Valve) ซึ่งควบคุมปริมาณการไหลของของเหลวตามลักษณะการควบคุมที่เลือกได้จากปุ่มต่างๆของวาล์ว และเครื่องสูบน้ำ (Pump) ที่ทำให้ของเหลวจะไหลจากซ้ายไปขวา แผนภาพเอนทิตีและความสัมพันธ์และโปรแกรมประยุกต์ของระบบแท็งค์บนบรรณาธิกรณ์สำหรับสร้างโปรแกรมประยุกต์แสดงดังรูปที่ 6.2



รูปที่ 6.2 แสดงระบบแท็งค์ในบรรณาธิกรณ์สำหรับสร้างโปรแกรมประยุกต์

6.1.3 ระบบเครือข่าย

ระบบเครือข่ายประกอบด้วยหน่วยย่อยของสายส่งสัญญาณ (cable secment) สายส่งสัญญาณ (cable) ซึ่งเป็นอาร์เรย์ของหน่วยย่อยของสายส่งสัญญาณ ตัวสร้างสัญญาณ (station) และ เครื่องทวนสัญญาณ (repeater) การทำงานจะเริ่มจากตัวสร้างสัญญาณส่งสัญญาณให้กับหน่วยย่อยของสายส่งซึ่งจะส่งสัญญาณไปทั้งด้านซ้ายและขวา ส่วนเครื่องทวนสัญญาณทำหน้าที่ส่งต่อสัญญาณจากสายส่งหนึ่งไปยังอีกสายส่งหนึ่งแผนภาพเอนทิตีและความสัมพันธ์ และโปรแกรมประยุกต์ของระบบเครือข่ายที่สร้างบนบรรณาธิกรณ์สำหรับสร้างโปรแกรมประยุกต์แสดงดังรูปที่ 6.3



รูปที่ 6.3 แสดงระบบเครือข่ายในบรรณาธิกรณ์สำหรับสร้างโปรแกรมประยุกต์

6.2 ผลการสร้างชุดคำสั่งโดยอัตโนมัติ

6.2.1 การสร้างชุดคำสั่งกฎการเปลี่ยนแปลง

ตัวอย่างการกำหนดกฎการเปลี่ยนแปลงเห็นได้ในตัวสร้างงานซึ่งเป็นเอนทิตีหนึ่งในระบบแถวคอย ข้อมูลสมาชิกของตัวสร้างงานที่สำคัญได้แก่ "state" ซึ่งแสดงสถานะในเวลาใดเวลาหนึ่งของตัวสร้างงาน และอาจมีค่าได้เป็น "GIdle" หมายถึงว่างงาน "GBusy" หมายถึงไม่ว่าง และ "GReady" หมายถึงพร้อมที่จะสร้างงาน ส่วน "enabled" เป็นข้อมูลที่เก็บว่าตัวสร้างงานกำลังทำงานอยู่หรือไม่ มีค่าได้เป็น จริงหรือเท็จ ทั้ง "state" และ "enabled" มีชนิดข้อมูลเป็น "AInteger" ซึ่งถือว่าเป็นตัวแปรพร้อมทำงาน ส่วนข้อมูลสมาชิกที่สำคัญในคลาสวิวได้แก่ "button" มีชนิดเป็น "Button11" ซึ่งเป็นคลาสในแพ็กเก็ต "gui" คลาสดังกล่าวมีตัวแปรพร้อมทำงาน "selected" ที่ใช้เก็บค่าว่า "button" ได้ถูกกดหรือไม่ ส่วนหนึ่งของพฤติกรรมของตัวสร้างงานสามารถกำหนดเป็นกฎการเปลี่ยนแปลงได้ดังแสดงใน ตารางที่ 6.1

ตารางที่ 6.1 แสดงกฎการเปลี่ยนแปลงสำหรับตัวสร้างงานในระบบแถวคอย

เลขที่	เงื่อนไข	การกระทำ
1.	enabled.val = = true && state.val = = GIdle	start (Model)
2.	state.val = = GReady && outputQueue.nJobs.val < outputQueue.nSlots	generate (Model)
3.	button.selected.val = = true	whenEnabled (View)
4.	enabled.val = = false	whenDisabled (View)

จากกฎการเปลี่ยนแปลงข้อที่ 1 เมื่อ "state" มีค่าเป็น "GIdle" และ ค่า "enabled" มีค่าเป็นจริง ให้เรียกคำสั่งในฟังก์ชัน "start" สำหรับกฎการเปลี่ยนแปลงข้อที่ 2 เมื่อ "state" มีค่าเป็น "GReady" และถ้าจำนวนงานในเอาท์พุตน้อยกว่าจำนวนที่เป็นไปได้ (outputQueue.nJobs.val < outputQueue.nSlots) คำสั่งใน "generate" จะถูกเรียก

กฎการเปลี่ยนแปลงในคลาสวิวข้อแรกหมายถึง เมื่อมีการกดปุ่มให้ทำงาน(button.selected.val = = true) ให้เรียกคำสั่งในฟังก์ชัน "whenEnabled" และเมื่อตัวสร้างงานไม่สามารถทำงานได้ (model.enabled.val = = false) ให้เรียกคำสั่งในฟังก์ชัน "whenDisabled"

จากพฤติกรรมดังกล่าวโปรแกรมจะสร้างชุดคำสั่งในคลาสโมเดลของตัวสร้างงานแสดงได้ในรูปที่ 6.4 จะเห็นว่ามีการประกาศค่าคงที่ "START" และ "GENERATE" ซึ่งเป็นค่าดัชนีฟังก์ชันให้มีค่าเท่ากับ 0 และ 1 ตามลำดับ ดัชนีฟังก์ชันนี้ถูกใช้ในฟังก์ชัน "dispatch( )" เพื่อใช้เรียกฟังก์ชัน "start( )" และ "generate( )" ได้อย่างถูกต้อง

นอกจากนี้ยังมีการสร้างกลไกการเริ่มทำงานสำหรับตัวแปรพร้อมทำงานและฟังก์ชัน 3 คู่ใน "initailize( )" ได้แก่ ความสัมพันธ์ระหว่าง "state – start" "state – generate" และ "enabled – start" ส่วนความสัมพันธ์ระหว่าง "output.nJobs - generate" ถูกสร้างใน "connectModelViewOutput" เนื่องจาก "nJobs" ไม่ใช่ตัวแปรพร้อมทำงานภายในตัวสร้างงานแต่เป็นตัวแปรพร้อมทำงานของแถวคอยซึ่งติดต่อกับตัวสร้างงานผ่านตัวแปร "outputQueue" สำหรับการสร้างคำสั่งในฟังก์ชันต่างๆนั้น ภายในฟังก์ชัน "start" ได้แสดงประโยค "if" ตามเงื่อนไขที่กำหนดไว้ในกฎการเปลี่ยนแปลงข้อที่หนึ่ง รายละเอียดการทำงานภายใน "start" สามารถเพิ่มเติมได้ภายใต้เงื่อนไขดังกล่าว ส่วนฟังก์ชัน "generate" แสดงประโยค "if" ดังที่กำหนดในกฎการเปลี่ยนแปลงข้อ 2 เตรียมพร้อมให้ผู้พัฒนาเพิ่มเติมรายละเอียดการทำงานเช่นเดียวกัน

ชุดคำสั่งในคลาสวิวของตัวสร้างงานแสดงไว้ในรูปที่ 6.5 มีการประกาศค่าคงที่สำหรับดัชนีฟังก์ชัน "WHENENABLED" และ "WHENDISABLED" พร้อมทั้งสร้างฟังก์ชัน "dispatch( )" เพื่อเรียกฟังก์ชันเหล่านั้นตามดัชนีของฟังก์ชัน ในฟังก์ชัน "initialize( )" มีการสร้างกลไกการเริ่มทำงานระหว่าง "button.selected – whenEnabled" และ "model.enabled – whenDisabled" และในฟังก์ชันทั้งสองได้แสดงเงื่อนไขตามที่กำหนดไว้ในกฎการเปลี่ยนข้อที่ 3 และ 4 ตามลำดับ ส่วนคำสั่งในคอนสตรักเตอร์และในฟังก์ชัน "connectModelViewInput( )" และ ฟังก์ชัน "connectModelViewOutput( )" เป็นคำสั่งที่สร้างได้โดยอัตโนมัติอยู่แล้วในบรรณาธิกรณ์สำหรับสร้างเค้าร่างเดิม

```
public class Generator  extends  AObject
{
 public static final int GIdle = 0;
 public static final int GBusy = 1;
 public static final int GReady = 2;
 public ABoolean enabled;
 public AInteger state;
 public int counter;
 public int interval = 8;
 public static Random rand;
 static final int START = 0;

 static final int GENERATE = 1;
 public Queue outputQueue ;
 ...
 public void start () throws SaosException
  {
   if(enabled.val == true && state.val == GIdle)
   {
     // **** implement function here****
    }
  }
  public void generate ()
  {
    if(state.val == GReady && outputQueue.nJobs.val < outputQueue.nSlots)
    {
     // **** implement function here****
    }
  }
 public void initialize()
 {
  try{
     enabled.addTE((AObject) this,START,"start()", SaosMain.AosUser);
     state.addTE((AObject) this,START,"start()", SaosMain.AosUser);
     state.addTE((AObject) this,GENERATE,"generate()", SaosMain.AosUser);
  }catch (SaosException e){
    System.out.println(" Error : " + e);
  }
 }

 public void connectModelViewInput(PortView port) throws SaosException
 {
 }

 public void connectModelViewOutput(PortView port) throws SaosException
 {
  outputQueue = (Queue) port.address.objectPtr;
  outputQueue.nJobs.addTE((AObject) this,GENERATE,"generate()", SaosMain.AosUser);
 }

 public void dispatch(int funcIndex) throws SaosException
 {
  switch(funcIndex)
  {
     case START : start(); break;
     case GENERATE : generate(); break;
    default: System.out.println("ERROR : dispatch(): " + this +
        ":funcIndex= " + funcIndex);
  }
 }
}
```

รูปที่ 6.4 แสดงชุดคำสั่งคลาสโมเดลของตัวสร้างงานที่สร้างขึ้นโดยอัตโนมัติ

```
class EditGenerator extends EditApp implements Constants
{
 Generator model;
 public Button11 button;
 static final int WHENENABLED = 0;
 static final int WHENDISABLED = 1;
 ...

 public void whenEnabled ()
 {
  if(button.selected.val == true)
  {
    // **** implement function here****
  }
 }

 public void whenDisabled ()
 {
  if(model.enabled.val == false)
  {
    // **** implement function here****
  }
 }

 public void initialize() throws SaosException
 {
  insert(model, true);
  view.updateCoordinates(x,y, x - view.x, y - view.y);
  view.setColor(Color.green);
  view.setCanvas(canvas);
  outPortStatus = false;
  portNumber = 2;
  view.initialize(canvas.getGraphics());
  outPortView.initialize();
  button.selected.addTE((AObject) this,WHENENABLED,"whenEnabled()", SaosMain.AosUser);
  model.enabled.addTE((AObject) this,WHENDISABLED,"whenDisabled()", SaosMain.AosUser);
 }

 public void connectModelViewInput(PortView port) throws SaosException
 {
  model.connectModelViewInput(port);
 }

 public void connectModelViewOutput(PortView port) throws SaosException
 {
  model.connectModelViewOutput(port);
 }
 public void dispatch(int funcIndex) throws SaosException
 {
  switch(funcIndex)
  {
    case WHENENABLED : whenEnabled(); break;
    case WHENDISABLED : whenDisabled(); break;
   default: System.out.println("ERROR : dispatch(): " + this +
       ":funcIndex= " + funcIndex);
  }
 }
 public void delete()
 {
   view.delete();
 }
}
```

รูปที่ 6.5 แสดงชุดคำสั่งคลาสวิวของตัวสร้างงานที่สร้างขึ้นโดยอัตโนมัติ

ในบางกรณีที่ต้องการให้ระบบทำงานอย่างใดอย่างหนึ่งทันทีเมื่อตัวแปรพร้อมทำงานเปลี่ยนแปลงโดยไม่ต้องกำหนดเงื่อนไข สามารถทำได้โดยกำหนดตัวเลือกของประโยคการเปลี่ยนแปลงเป็น "ทำงานทุกครั้ง" ตัวอย่างเอนทิตีที่ใช้พฤติกรรมดังที่กล่าวนี้ได้แก่แท็งค์ระบบแท็งค์ ทุกครั้งที่เวลาของระบบเปลี่ยนแปลงแท็งค์จะทำการคำนวณระดับน้ำ ดังนั้นกฎการเปลี่ยนแปลงของแท็งค์จึงกำหนดได้ดังแสดงในตารางที่ 6.2

ตารางที่ 6.2 แสดงกฎการเปลี่ยนแปลงสำหรับแท็งค์ในระบบแท็งค์

เลขที่	เงื่อนไข	การกระทำ
1.	ทุกครั้งที่ SaosMain.aosTime เปลี่ยนแปลง	AosTimeChanged (Model)
2.	ทุกครั้งที่ refValue เปลี่ยนแปลง	RefValueChanged(View)

"aosTime" เป็นตัวแปรพร้อมทำงานที่มีอยู่แล้วในคลาส "SaosMain" ที่เก็บเวลาของระบบพร้อมทำงาน และจะเปลี่ยนแปลงค่าทุกครั้งตามช่วงเวลาที่กำหนด ดังนั้นฟังก์ชัน "aosTimeChanged" จะถูกเรียกทุกๆ 1 หน่วยเวลาในคลาส "SaosMain" ตัวแปร"refValue" เป็นตัวแปรพร้อมทำงานในคลาสโมเดลที่เก็บข้อมูลระดับของเหลวในแท็งค์ เมื่อมีการเปลี่ยนจะเรียกให้ฟังก์ชัน "refValueChanged" ทำงาน ชุดคำสั่งในคลาสโมเดลและคลาสวิวของแท็งค์แสดงดัง รูปที่ 6.6 และ รูปที่ 6.7 ซึ่งแสดงให้เห็นว่ามีการสร้างดัชนีฟังก์ชัน "AOSTIMECHANGED" ในคลาสโมเดล และ "REFVALUECHANGED" ในคลาสวิว พร้อมทั้งฟังก์ชัน dispath ( ) เพื่อเรียกไปยังฟังก์ชันทั้งสองอย่างถูกต้อง พร้อมทั้งสร้างกลไกการเริ่มทำงานของตัวแปรพร้อมทำงาน "aosTime" ให้เชื่อมโยงกับ ฟังก์ชัน "aosTimeChanged" และกลไกการเริ่มทำงานของ "refValue" ให้เชื่อมโยงกับฟังก์ชัน "refValueChanged" ในฟังก์ชันทั้งสองจะไม่มีเงื่อนไขกำหนดอยู่ ดังนั้นคำสั่งในฟังก์ชันจะทำงานทันทีที่ค่าตัวแปรเริ่มทำงานที่เกี่ยวข้องเปลี่ยนแปลง

6.2.2 การสร้างชุดคำสั่งการกำหนดค่าล่วงหน้า

จากกฎการเปลี่ยนแปลงของตัวสร้างงานที่กำหนดในหัวข้อที่ผ่านมา เมื่อ "state" มีค่าเป็น "GIdle" และ "enabled" มีค่าเป็นจริง ฟังก์ชัน "start" จะทำงาน ภายในฟังก์ชันดังกล่าวมีการกำหนดค่าล่วงหน้า ดังแสดงในตารางที่ 6.3 นั่นคือต้องการให้ "state" มีค่าเป็น "GReady" เมื่อเวลาผ่านไปเท่ากับเวลาหน่วงซึ่งเป็นค่าที่ได้จากการสุ่มภายในช่วงตัวเลขที่กำหนดโดย "interval" ดังนั้นเมื่อตัวสร้างงานเริ่มทำงานไประยะเวลาหนึ่งสถานะของตัวสร้างงานจะเปลี่ยนเป็น "GReady" ซึ่งจะทำให้เงื่อนไขในกฎการเปลี่ยนแปลงข้อที่ 2 เป็นจริงทำให้ระบบทำงานต่อไปได้ เมื่อสร้างชุดคำสั่งโดยเพิ่มการกำหนดค่าล่วงหน้าดังที่แสดงข้างต้น ทำให้ฟังก์ชัน "start" มีคำสั่ง "fAssign" เพิ่มขึ้นดังแสดงใน รูปที่ 6.8

ตารางที่ 6.3 แสดงการกำหนดค่าล่วงหน้าของตัวสร้างงานในแถวคอย

เลขที่	ฟังก์ชันที่ต้องการให้เกิดการกำหนดค่า	ตัวแปรที่ต้องการกำหนดค่า	ค่าที่ต้องการ	เวลาหน่วง
1.	start (Model)	state	GReady	Math.abs(rand.nextInt( ) % interval)

```
public class Tank  extends  AObject
{
...
 static final int AOSTIMECHANGED = 0;
 public Vector inputValve ;
 public Vector outputValve ;
...
public void aosTimeChanged ()
 {
     // **** implement function here****
 }
 public void initialize()
 {
  try{
     SaosMain.aosTime.addTE((AObject) this,AOSTIMECHANGED,"aosTimeChanged()", SaosMain.AosUser);
  }catch (SaosException e){
    System.out.println(" Error : " + e);
  }
 }
public void dispatch(int funcIndex) throws SaosException
 {
  switch(funcIndex)
  {
    case AOSTIMECHANGED : aosTimeChanged(); break;
    default: System.out.println("ERROR : dispatch(): " + this +
       ":funcIndex= " + funcIndex);
  }
 }
}
```

รูปที่ 6.6 แสดงชุดคำสั่งคลาสโมเดลของแท็งค์ที่สร้างขึ้นโดยอัตโนมัติ

```
class  EditTank  extends  EditApp implements Constants
{
 Tank  model;
 public RatioIndicatorV tankLevel;
 public RightArrow refArrow;
 static final int REFVALUECHANGED = 0;
 .....

public void refValueChanged ()
 {
     // **** implement function here****
 }
 public void initialize() throws SaosException
 {
    ....
    model.refValue.addTE((AObject) this,REFVALUECHANGED,"refValueChanged()", SaosMain.AosUser);
 }

 public void dispatch(int funcIndex) throws SaosException
 {
  switch(funcIndex)
  {
    case REFVALUECHANGED : refValueChanged(); break;
    default: System.out.println("ERROR : dispatch(): " + this +
       ":funcIndex= " + funcIndex);
  }
 }
}
```

รูปที่ 6.7 แสดงชุดคำสั่งในคลาสวิวของแท็งค์ที่สร้างขึ้นโดยอัตโนมัติ

```
public void start () throws SaosException
 {
  if(enabled.val == true && state.val == GIdle)
  {
    // **** implement function here****
   FAssign.fAssign(this,state,GReady,
        Math.abs(rand.nextInt()% interval), "state",  SaosMain.AosUser);
  }
 }
```

รูปที่ 6.8 แสดงฟังก์ชัน "start" ในคลาสโมเดลของตัวสร้างงานเมื่อมีการกำหนดค่าล่วงหน้า

6.2.3 การสร้างชุดคำสั่งการเรียกฟังก์ชันล่วงหน้า

การทำงานของวาล์วระบบแท็งค์มีฟังก์ชันที่สำคัญ 2 ฟังก์ชันได้แก่ "manualLoop" และ "autoControl" ทั้งสองฟังก์ชันจะมีการทำงานแบบวนซ้ำนั่นคือจะต้องเรียกฟังก์ชันดังกล่าวทุกๆ 1 วินาที ดังนั้นจึงมีกำหนดการเรียกฟังก์ชันล่วงหน้าดังแสดงในตารางที่ 6.4

ตารางที่ 6.4 แสดงการเรียกฟังก์ชันล่วงหน้าของวาล์วในระบบแท็งค์

เลขที่	ฟังก์ชันที่ต้องการให้เกิดการเรียกฟังก์ชันล่วงหน้า	ฟังก์ชันที่ต้องการเรียก	เวลาหน่วง
1.	manualLoop (View)	manualLoop (View)	1 หน่วยเวลา
2.	autoControl(View)	autoControl(View)	1 หน่วยเวลา

ชุดคำสั่งที่สร้างได้สำหรับวาล์วแสดงดังรูปที่ 6.9 จะเห็นว่ามีการกำหนดดัชนีฟังก์ชันให้กับ "MANUALLOOP" และ "AUTOCONTROL" พร้อมกับฟังก์ชัน "dispatch()" ที่จะเรียกฟังก์ชันทั้งสองได้อย่างถูกต้อง ภายในฟังก์ชันทั้งสองมีคำสั่ง "fCall" ที่ทำให้เกิดการเรียกฟังก์ชันล่วงหน้าดังที่ต้องการ

6.2.4 การสร้างชุดคำสั่งสมการกำหนดค่า

สมการกำหนดค่าถูกใช้ในตัวสร้างสัญญาณในระบบเครือข่าย ตัวสร้างสัญญาณมีตัวแปรพร้อมทำงาน "collisionDetected" มีชนิดเป็น "ABoolean" ซึ่งมีค่าได้เป็นจริงหรือเท็จ สามารถคำนวณได้จากสมการกำหนดค่าดังแสดงในตารางที่ 6. 5

สิ่งที่ต้องการคือให้ "collisionDetected" มีค่าเท่ากับผลลัพธ์ของสมการเสมอ ดังนั้นเมื่อตัวแปรพร้อมทำงานในสมการได้แก่ "state" , "rightSignal", "leftSignal" มีเปลี่ยนแปลงค่าจะต้องคำนวณสมการและกำหนดค่าใหม่เสมอ

```
class EditValve extends EditApp implements Constants
{
  .....
  static final int MANUALLOOP = 1;
  .....
  static final int AUTOCONTROL = 8;

 public void manualLoop () throws SaosException
 {
   ...
   FCall.fCall(this,MANUALLOOP,1, "manualLoop", SaosMain.AosUser);
 }

public void dispatch(int funcIndex) throws SaosException
 {
  switch(funcIndex)
  {
    case MANUALLOOP : manualLoop(); break;
    case AUTOCONTROL : autoControl(); break;
   default: System.out.println("ERROR : dispatch(): " + this +
       ":funcIndex= " + funcIndex);
  }
 }

public void autoControl () throws SaosException
 {
   FCall.fCall(this,AUTOCONTROL,1, "autoControl", SaosMain.AosUser);
 }
}
```

รูปที่ 6.9 แสดงชุดคำสั่งในคลาสวิวของวาล์วที่สร้างขึ้นโดยอัตโนมัติ

ตารางที่ 6. 5 แสดงสมการกำหนดค่าสำหรับตัวสร้างสัญญาณในระบบเครือข่าย

เลขที่	ตัวแปรที่ต้องการกำหนดค่า	การคำนวณ
1.	CollisionDetected	state.val == TRANSMIT && (outputCableSegment.rightSignal.val > 1 || outputCableSegment.leftSignal.val > 1)

รูปที่ 6.10 แสดงชุดคำสั่งสำหรับสมการที่กำหนดค่าดังข้างต้นในตัวสร้างสัญญาณ จะเห็นว่ามีการสร้างฟังก์ชันใหม่ได้แก่ "alwayscollisionDetected" ซึ่งมีสมการกำหนดค่าแสดงอยู่ เนื่องจาก "collisionDetected" เป็นตัวแปรพร้อมทำงานจึงใช้วิธีการ "setVal" เพื่อกำหนดค่าให้ตัวแปรดังกล่าวแทนการใช้เครื่องหมาย "=" ดังสมการทั่วไป นอกจากนั้นยังมีการประกาศค่า "ALWAYSCOLLISIONDETECTED" เป็นดัชนีฟังก์ชันเพื่อใช้ในฟังก์ชัน "dispatch()" พร้อมทั้งสร้างกลไกการเริ่มทำงานให้กับตัวแปรพร้อมทำงาน "state","rightSignal" และ "leftSignal" ให้มีความสัมพันธ์กับฟังก์ชัน "alwayscollisionDetected" ด้วย

```
public class Station  extends  AObject
{
 ...
 static final int ALWAYSCOLLISIONDETECTED = 1;
 ...
 public void alwayscollisionDetected() throws SaosException
 {
    collisionDetected.setVal(
                state.val == TRANSMIT && ( outputCableSegment.rightSignal.val > 0 ||
                outputCableSegment.leftSignal.val > 0));
 {

 public void initialize()
 {
  try{
        ...
        state.addTE((AObject) this,ALWAYSCOLLISIONDETECTED,"alwayscollisionDetected()",
        SaosMain.AosUser);
  }catch (SaosException e){
    System.out.println(" Error : " + e);
  }
 }

 public void connectModelViewOutput(PortView port) throws SaosException
 {
          ...
  outputCableSegment = (CableSegment) port.address.objectPtr;
  outputCableSegment.rightSignal.addTE((AObject)
        this,ALWAYSCOLLISIONDETECTED,"alwayscollisionDetected()", SaosMain.AosUser);
  outputCableSegment.leftSignal.addTE((AObject)
        this,ALWAYSCOLLISIONDETECTED,"alwayscollisionDetected()", SaosMain.AosUser);
           ...
 }

 public void dispatch(int funcIndex) throws SaosException
 {
  switch(funcIndex)
  {
    ....
  case ALWAYSCOLLISIONDETECTED : alwayscollisionDetected(); break;
  default: System.out.println("ERROR : dispatch(): " + this +
       ":funcIndex= " + funcIndex);
  }
 }

}
```

รูปที่ 6.10 แสดงชุดคำสั่งในคลาสโมเดลของตัวสร้างสัญญาณที่สร้างขึ้นโดยอัตโนมัติ

6.3 การวิเคราะห์ผลการทดสอบ

หลังจากทำการทดลองสร้างชุดคำสั่งอัตโนมัติสำหรับทุกชนิดของเอนทิตีในระบบที่ใช้ทดสอบ ปรากฏว่ามีสัดส่วนของจำนวนชุดคำสั่งที่สร้างได้โดยอัตโนมัติวัดโดยบรรทัดของชุดคำสั่งของเปรียบเทียบกับชุดคำสั่งที่ใช้งานจริงของวัตถุต่างๆแสดงได้ดังตารางที่ 6. 6 ส่วนกราฟในรูปที่ 6.11 รูปที่ 6.12 และ รูปที่ 6.13 แสดงสัดส่วนบรรทัดคำสั่งที่ใช้งานจริงกับบรรทัดคำสั่งที่สร้างได้โดยอัตโนมัติจากสิ่งแวดล้อมสำหรับพัฒนาโปรแกรม

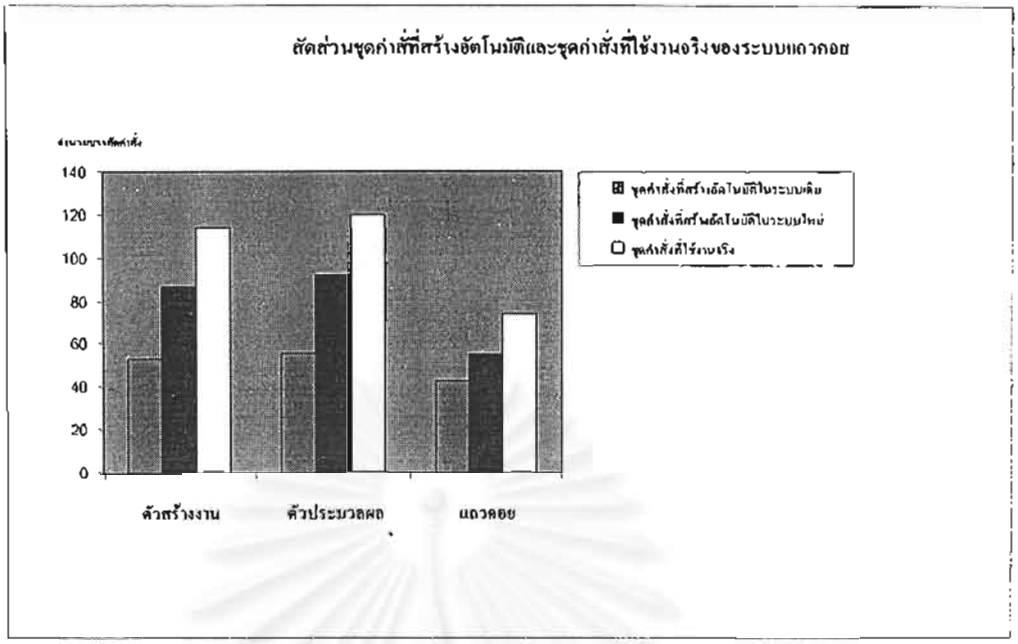
ด้วยแผนภาพเอนทิตีและความสัมพันธ์เดิมและจากสิ่งแวดล้อมที่แก้ไขเพิ่มเติมของระบบแถวคอย ระบบแท็งค์ และระบบเครือข่ายตามลำดับ

จากผลการทดลองแสดงให้เห็นว่า ระบบที่พัฒนาขึ้นนี้สามารถช่วยแบ่งเบาภาระในการสร้างชุดคำสั่งของระบบวัตถุพร้อมทำงาน สัดส่วนของชุดคำสั่งที่สร้างได้นั้นขึ้นอยู่กับความซับซ้อนของวัตถุแต่ละตัว ในระบบแถวคอยมีค่าสัดส่วนค่อนข้างสูงเนื่องจากเป็นระบบที่มีการทำงานที่ไม่ซับซ้อนซึ่งอาศัยกลไกของวัตถุพร้อมทำงานเป็นส่วนใหญ่ และกลไกเหล่านี้สามารถสร้างได้โดยอัตโนมัติ คำสั่งที่ต้องเพิ่มเติมเอง ได้แก่คำสั่งในส่วนการกระทำของกฎการเปลี่ยนแปลงเป็นคำสั่งง่ายๆเช่น ปรับเปลี่ยนสถานะ เปลี่ยนการแสดงผลบนหน้าจอของหน้าจอเป็นต้น ในขณะที่วัตถุต่างๆในระบบแท็งค์ และ ระบบเครือข่ายมีการทำงานที่ซับซ้อนนอกเหนือไปจากกลไกของวัตถุพร้อมทำงาน เช่น ในส่วนการกระทำของกฎการเปลี่ยนแปลงของแท็งจะต้องคำนวณปริมาณน้ำที่ไหลเข้าและไหลออก เพื่อแสดงระบบน้ำได้อย่างถูกต้องเป็นต้น การทำงานดังกล่าวไม่สามารถสร้างได้โดยอัตโนมัติ ทำให้สัดส่วนบรรทัดคำสั่งที่สร้างได้มีค่าน้อยกว่าระบบแถวคอย

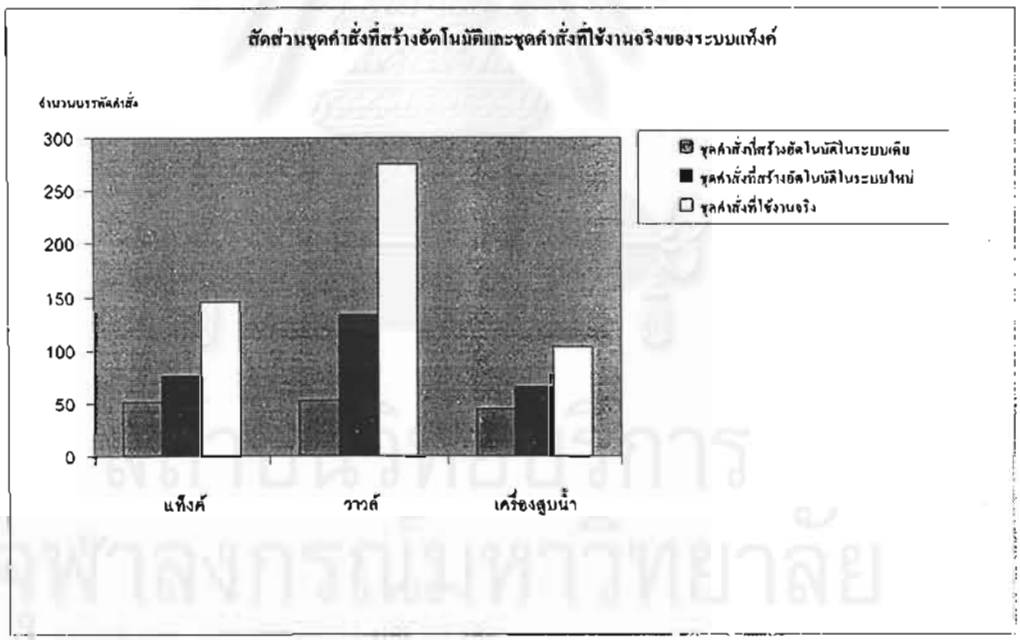
เมื่อเปรียบเทียบจำนวนบรรทัดที่สร้างได้ในสิ่งแวดล้อมสำหรับการพัฒนาโปรแกรมเดิม จะเห็นว่ามีจำนวนบรรทัดที่สร้างได้โดยอัตโนมัติเพิ่มขึ้น คำสั่งที่เพิ่มขึ้นเป็นคำสั่งเกี่ยวกับการสร้างกลไกการเริ่มทำงานและคำสั่งในฟังก์ชัน “dispatch()” เป็นส่วนใหญ่ ส่วนการประกาศค่าตัวแปรและการกำหนดคำสั่งเกี่ยวกับการเชื่อมต่อสามารถสร้างได้อยู่แล้วในระบบเดิม อย่างไรก็ตาม ผู้ใช้สามารถกำหนดค่าตัวแปรและฟังก์ชันต่างๆในคลาสวิวได้ ทำให้มีคำสั่งในการประกาศค่าตัวแปรและฟังก์ชันในคลาสวิวเพิ่มขึ้นจากเดิม และเนื่องจากตัวสร้างชุดคำสั่งสามารถจัดการกับการประกาศค่าคงที่ดัชนีฟังก์ชันได้โดยอัตโนมัติ ทำให้ผู้ใช้ไม่ต้องกำหนดดัชนีเหล่านี้ด้วยตัวเอง

ตารางที่ 6. 6 แสดงจำนวนบรรทัดคำสั่งของวัตถุพร้อมทำงานต่างๆในชุดคำสั่งที่สร้างอัตโนมัติเปรียบเทียบกับชุดคำสั่งที่ใช้งานจริง

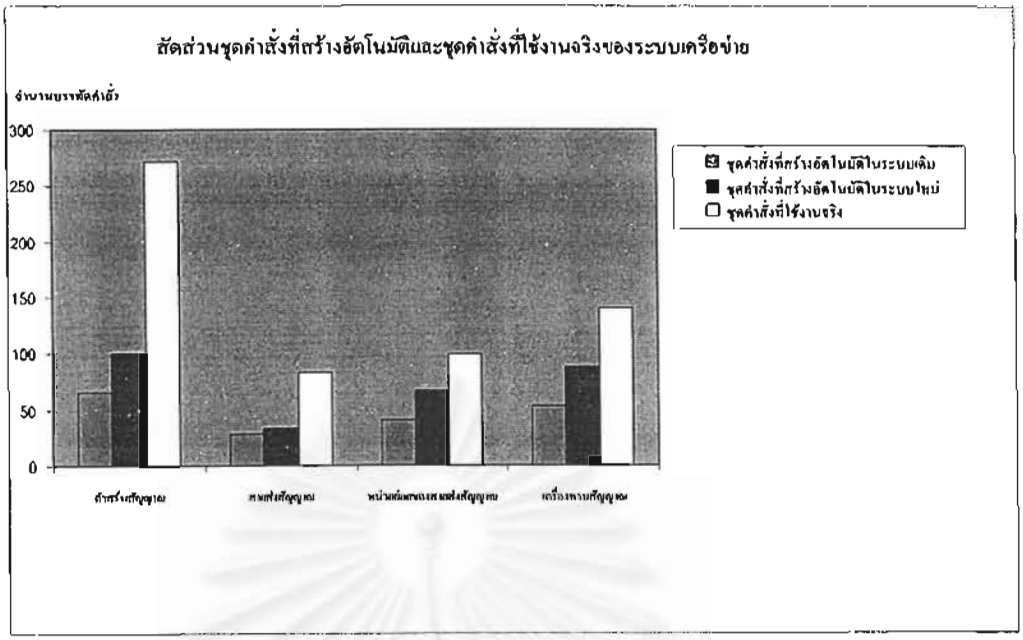
ระบบ	วัตถุพร้อมทำงาน	ชุดคำสั่งที่สร้างอัตโนมัติในระบบเดิม	ชุดคำสั่งที่สร้างอัตโนมัติในระบบใหม่	ชุดคำสั่งที่ใช้งานจริง	ร้อยละของบรรทัดคำสั่งที่สร้างได้ในระบบเดิมต่อชุดคำสั่งที่ใช้งานจริง	ร้อยละของบรรทัดคำสั่งที่สร้างได้ในระบบใหม่ต่อชุดคำสั่งที่ใช้งานจริง
แถวคอย	ตัวสร้างงาน	53	87	114	46.5	76.3
	แถวคอย	43	55	74	58.1	74.3
	ตัวประมวลผล	56	92	120	46.7	76.7
	รวม	**152**	**234**	**308**	**49.34**	**76.0**
แท็งค์	แท็งค์	51	76	146	34.9	52.1
	วาล์ว	52	134	275	18.9	48.7
	เครื่องสูบน้ำ	45	66	103	43.7	64.1
	รวม	**148**	**276**	**524**	**28.2**	**52.7**
เครือข่าย	เครื่องส่งสัญญาณ	66	100	271	24.4	36.9
	สายส่งสัญญาณ	29	34	83	34.9	41.0
	หน่วยย่อยของสายส่งสัญญาณ	41	67	99	41.4	67.7
	เครื่องทวนสัญญาณ	53	88	140	37.9	62.9
	รวม	**189**	**289**	**593**	**48.7**	**31.9**



รูปที่ 6.11 กราฟแสดงสัดส่วนชุดคำสั่งที่ใช้งานจริงต่อชุดคำสั่งที่สร้างอัตโนมัติในระบบแถวคอย



รูปที่ 6.12 กราฟแสดงสัดส่วนชุดคำสั่งที่ใช้งานจริงต่อชุดคำสั่งที่สร้างอัตโนมัติในระบบแท็งค์



รูปที่ 6.13 กราฟแสดงสัดส่วนชุดคำสั่งที่ใช้งานจริงต่อชุดคำสั่งที่สร้างอัตโนมัติในระบบเครือข่าย

บทที่ 7

บทสรุปและข้อเสนอแนะ

7.1 บทสรุป

วิทยานิพนธ์นี้ได้พัฒนาบรรณาธิกรณ์สำหรับกำหนดพฤติกรรมของวัตถุพร้อมทำงาน ที่ผู้ใช้งานสามารถกำหนดของวัตถุพร้อมทำงานผ่านการติดต่อกับผู้ใช้งานแบบกราฟิก พฤติกรรมของวัตถุพร้อมทำงานที่กำหนดได้จากบรรณาธิกรณ์แบ่งออกเป็น กฎการเปลี่ยนแปลง สมการกำหนดค่า และการกระทำตามเหตุการณ์ การพัฒนานี้รวมถึงความสามารถในการสร้างชุดคำสั่งที่รองรับพฤติกรรมที่กำหนดโดยผู้ใช้งานด้วย เมื่อพัฒนาบรรณาธิกรณ์เสร็จสมบูรณ์แล้วจึงได้ทำการทดสอบความสามารถของบรรณาธิกรณ์ โดยกำหนดพฤติกรรมและสร้างชุดคำสั่งของวัตถุพร้อมทำงานทั้งหมดในระบบต่างๆ ได้แก่ ระบบแถวคอย ระบบแท็งค์ และ ระบบเครือข่าย ซึ่งวัตถุเหล่านี้ได้เคยถูกพัฒนาขึ้นและใช้งานได้จริงในบรรณาธิกรณ์สำหรับสร้างโปรแกรมประยุกต์มาแล้ว โดยสามารถสรุปประโยชน์ของบรรณาธิกรณ์สำหรับกำหนดพฤติกรรมได้ดังต่อไปนี้

1) ช่วยลดเวลาในการเขียนชุดคำสั่งด้วยตัวเอง
จากผลการทดลองแสดงให้เห็นว่าโปรแกรมสามารถสร้างชุดคำสั่งโดยอัตโนมัติได้เพิ่มเติมจากโปรแกรมเดิมโดยเฉพาะคำสั่งที่รองรับกลไกของพฤติกรรมต่างๆทำให้ผู้พัฒนาไม่ต้องเสียเวลาในการเขียนคำสั่งเหล่านี้ด้วยตัวเอง และสามารถให้เวลากับคำสั่งในรายละเอียดอื่นๆได้มากขึ้น

2) ลดข้อผิดพลาดในการเขียนโปรแกรม
คำสั่งที่สร้างได้อัตโนมัติครอบคลุมถึงการสร้างกลไกการเริ่มทำงาน และการสร้างดัชนีฟังก์ชัน จากผลการทดลองแสดงให้เห็นว่าโปรแกรมสามารถสร้างคำสั่งเหล่านี้ได้อย่างถูกต้องและครบถ้วน ดังนั้นเมื่อผู้พัฒนาไม่ต้องเขียนคำสั่งเหล่านี้ด้วยตนเองเป็นการลดโอกาสในการเกิดข้อผิดพลาดในโปรแกรม และลดเวลาในการค้นหาและแก้ไขข้อผิดพลาดเหล่านี้ นอกจากนี้ การสร้างประโยคการเปลี่ยนแปลงต่างๆ สามารถทำได้โดยการเลือกจากรายการที่แสดงอยู่เป็นส่วนใหญ่ เช่นการเลือกตัวแปรพร้อมทำงานตัวหนึ่ง สามารถคลิกเลือกได้จากรายการแสดงแบบต้นไม้ ผู้ใช้ไม่ต้องป้อนชื่อเหล่านั้นผ่านแป้นคีย์ข้อมูลซึ่งบางครั้งประกอบด้วยเส้นทางข้อมูลที่ค่อนข้างยาว ทำให้เกิดข้อผิดพลาดจากการป้อนข้อมูลน้อยลง

3) ผู้พัฒนาสามารถพัฒนาโปรแกรมประยุกต์ในขอบเขตเรื่องราวใหม่ๆได้ง่ายขึ้น
ระบบการติดต่อกับผู้ใช้งานของบรรณาธิกรณ์สำหรับกำหนดพฤติกรรมของวัตถุพร้อมทำงานเป็นแบบวิชวล แสดงข้อมูลของแต่ละเอนทิตีต่างๆอย่างชัดเจน เช่นรายการฟังก์ชันสมาชิก และข้อมูลสมาชิกต่างๆเป็นต้น นอกจากนี้ยังสามารถข้อมูลสมาชิกของเอนทิตีอื่นๆที่มีความสัมพันธ์กัน

ได้อย่างถูกต้อง ทำให้ผู้ใช้งานเห็นภาพรวมของระบบและสามารถเลือกใช้ข้อมูลเหล่านี้ในการกำหนดพฤติกรรมได้อย่างง่ายดายโดยการใช้เมาส์คลิก นอกจากนี้ผู้ใช้สามารถสร้างการกำหนดค่าล่วงหน้า และการเรียกฟังก์ชันล่วงหน้าได้อย่างสะดวกโดยการเติมตัวแปรต่างๆที่จำเป็น โดยไม่ต้องเขียนคำสั่งด้วยตนเองซึ่งเป็นรูปแบบที่มีความซับซ้อน ทำให้สามารถเขียนโปรแกรมได้อย่างสะดวก

7.2 ข้อเสนอแนะ

1) บรรณาธิกรณ์แต่ละตัวยังมิได้ถูกรวมเป็นโปรแกรมเดียวกัน ทำให้ไม่สะดวกต่อการใช้งาน ทั้งยังมีส่วนประกอบบางส่วนที่ต้องถูกเรียกใช้จากบรรณาธิกรณ์แต่ละตัวซึ่งต้องเก็บแบบซ้ำซ้อนแยกไปตามแต่ละไดเรกทอรีของบรรณาธิกรณ์แต่ละตัว ดังนั้นจึงควรรวบรวมบรรณาธิกรณ์เหล่านี้เข้าด้วยกันภายใต้โปรแกรมหลักเพียงโปรแกรมเดียว ที่สามารถเรียกใช้บรรณาธิกรณ์เหล่านี้และใช้ส่วนประกอบต่างๆร่วมกันได้
2) บรรณาธิกรณ์สำหรับกำหนดพฤติกรรมช่วยแก้ไขปัญหาในการแปลงจากการออกแบบระบบไปสู่การเขียนโปรแกรม แต่การออกแบบระบบวัตถุพร้อมทำงานหรือการศึกษาทำความเข้าใจระบบเดิมยังมีความยุ่งยากซับซ้อน จึงควรมีการกำหนดแผนภาพที่ใช้อธิบายการทำงานของระบบพร้อมทำงานเพื่อใช้เป็นสื่อกลางสำหรับทำความเข้าใจพฤติกรรม และสามารถใช้เป็นเครื่องมือในการช่วยออกแบบ นอกจากนี้การปรับปรุงบรรณาธิกรณ์สำหรับกำหนดพฤติกรรมให้สามารถอ่านชุดคำสั่งของวัตถุพร้อมทำงานและแปลงเป็นแผนภาพดังกล่าว จะทำให้ผู้เริ่มศึกษาระบบวัตถุพร้อมทำงานสามารถเข้าใจโปรแกรมต่างๆที่สร้างขึ้นได้รวดเร็วขึ้น

รายการอ้างอิง

[1] Toshimi Minoura and Sungwoon Choi. "Structural active-object systems fundamentals", Technical Report 93-40-04, Dept. of Computer Science, Oregon State University, 1993.

[2] Pornsiri Muenchaisri. Software Composition with Extended Entity-Relationship Diagrams, (Doctoral dissertation) Dept. of Computer Science, Oregon State University, 1997.

[3] Raghava Pareddy. Structure Active Object System Translator, Master Project, Dept. of Computer Science, Oregon State University, 1994.

[4] Yanbing Lu. Structure Active-Object Systems in Java, Technical Report, Dept. of Computer Science, Oregon State University, 1996.

[5] Pornsiri Muenchaisri and Toshimi Minoura. "Entity-Relationship Software Development Environment", the proceeding of Technology of Object-Oriented Language and Systems (TOOLS USA'99), Santa Barbara, CA, August 1-5, 1999.

[6] Pornsiri Muenchaisri. "Visualization of A Structural Active-Object System with UML", 3rd National Computer Science and Engineering Conference, December 15-17, 1999. Bangkok, Thailand.

[7] Ted Lewis, Larry Rosenstein, Wolfgang Press, Andre Weinand, Erich Gamma, Paul Calder, Glenn Andert, John Vlissides and Kurt Schmucker. Object Oriented Application Frameworks . m.p.:Manning Publications Co.,1995

ภาคผนวก

ภาคผนวก ก.

ชุดคำสั่งที่สร้างโดยอัตโนมัติของโปรแกรมประยุกต์ระบบต่างๆ

ในภาคผนวกนี้แสดงชุดคำสั่งที่สร้างโดยอัตโนมัติของโปรแกรมประยุกต์ 3 โปรแกรมที่ใช้ในการทดสอบความสามารถของบรรณาธิกรณ์สำหรับกำหนดพฤติกรรมของวัตถุพร้อมทำงาน ได้แก่ ระบบแถวคอย ระบบแท็งค์ และระบบเครือข่าย โดยแต่ละโปรแกรมแสดงประโยคการเปลี่ยนแปลงที่กำหนดให้กับเอนทิตีแต่ละตัว จากนั้นจึงแสดงชุดคำสั่งที่สร้างได้ของเอนทิตีเหล่านั้น

ก-1 ระบบแถวคอย

ก-1-1 ตัวสร้างงาน

ประโยคการเปลี่ยนแปลงที่กำหนดให้กับตัวสร้างงานได้แก่กฎการเปลี่ยนแปลงดังแสดงไว้ใน ตารางที่ ก- 1 และ การกำหนดค่าล่วงหน้าดังแสดงใน ตารางที่ ก- 2

- กฎการเปลี่ยนแปลง

ตารางที่ ก- 1 กฎการเปลี่ยนแปลงสำหรับตัวสร้างงานในระบบแถวคอย

เลขที่	เงื่อนไข	การกระทำ
1.	enabled.val = = true && state.val = = GIdle	start (Model)
2.	state.val = = GReady && outputQueue.nJobs.val < outputQueue.nSlots	generate (Model)
3.	button.selected.val = = true	whenEnabled (View)
4.	enabled.val = = false	whenDiabled (View)

- การกำหนดค่าล่วงหน้า

ตารางที่ ก- 2 การกำหนดค่าล่วงหน้าสำหรับตัวสร้างงานในระบบแถวคอย

เลขที่	ฟังก์ชันที่ต้องการให้เกิดการกำหนดค่า	ตัวแปรที่ต้องการกำหนดค่า	ค่าที่ต้องการ	เวลาหน่วง
1.	start (Model)	state	GReady	Math.abs(rand.nextInt( ) % interval)

จากประโยคการเปลี่ยนแปลงที่กำหนด จะได้ชุดคำสั่งคลาสโมเดลของตัวสร้างงานแสดงได้ดังรูปที่ ก- 1 และชุดคำสั่งในคลาสวิวของตัวสร้างงานแสดงได้ดังรูปที่ ก- 2

```
import saos.gui.*;
import saos.kernel.*;
import java.awt.*;
import java.io.*;
import java.lang.Math;
import java.util.*;

public class Generator  extends  AObject
{
  public static final int GIdle = 0;
  public static final int GBusy = 1;
  public static final int GReady = 2;
  public ABoolean enabled;
  public AInteger state;
  public int counter;
  public int interval = 8;
  public static Random rand;
  static final int START = 0;
  static final int GENERATE = 1;
  public Queue outputQueue ;

  public Generator(String name)
  {
    super(name);
  }

  public void paint (Graphics g)
  {
  }

  public void start () throws SaosException
  {
   if(enabled.val == true && state.val == GIdle)
   {
      // **** implement function here****
    FAssign.fAssign(this,state,GReady,
          Math.abs(rand.nextInt() % interval), "state",  SaosMain.AosUser);
   }
  }

  public void generate ()
  {
   if(state.val == GReady && outputQueue.nJobs.val < outputQueue.nSlots)
   {
      // **** implement function here****
   }
  }

  public void initialize()
  {
   try{
      enabled.addTE((AObject) this,START,"start()", SaosMain.AosUser);
      state.addTE((AObject) this,START,"start()", SaosMain.AosUser);
      state.addTE((AObject) this,GENERATE,"generate()", SaosMain.AosUser);
   }catch (SaosException e){
      System.out.println(" Error : " + e);
   }
  }

  public void connectModelViewInput(PortView port) throws SaosException
  {
  }
```

รูปที่ ก- 1 แสดงชุดคำสั่งในคลาสโมเดลของตัวสร้างงานที่สร้างได้โดยอัตโนมัติ

```
public void connectModelViewOutput(PortView port) throws SaosException
{
 outputQueue = (Queue) port.address.objectPtr;
 outputQueue.nJobs.addTE((AObject) this,GENERATE,"generate()", SaosMain.AosUser);
}

public void dispatch(int funcIndex) throws SaosException
{
 switch(funcIndex)
 {
   case START : start(); break;
   case GENERATE : generate(); break;
  default: System.out.println("ERROR : dispatch(): " + this +
      ":funcIndex= " + funcIndex);
 }
}
}
```

รูปที่ ก- 1 แสดงชุดคำสั่งในคลาสโมเดลของตัวสร้างงานที่สร้างได้โดยอัตโนมัติ (ต่อ)

```
import saos.gui.*;
import saos.kernel.*;
import java.awt.*;
import java.io.*;
import java.lang.Math;
import java.util.*;

class EditGenerator extends EditApp implements Constants
{
 Generator model;
 public Button11 button;
 static final int WHENENABLED = 0;
 static final int WHENDISABLED = 1;

 public EditGenerator(EditCompound object, AppCanvas canvas, int x, int y)
 {
  super(x, y);
  model = new Generator("Generator");
  view  = (EditCompound) object.clone();
  this.canvas = canvas;
  name = new String("Generator");
  outPortView = new PortView(OUT, canvas, this, SINGLE/MULTIPLE);
 }


 public void paint (Graphics g)
 {
 }

 public void whenEnabled ()
 {
  if(button.selected.val == true)
  {
    // **** implement function here****
  }
 }
```

รูปที่ ก- 2 แสดงชุดคำสั่งคลาสวิวของตัวสร้างงานที่สร้างได้โดยอัตโนมัติ

```
public void whenDisabled ()
 {
  if(model.enabled.val == false)
  {
     // **** implement function here****
  }
 }


 public void initialize() throws SaosException
 {
  insert(model, true);
  view.updateCoordinates(x,y, x - view.x, y - view.y);
  view.setColor(Color.green);
  view.setCanvas(canvas);
  outPortStatus = false;
  portNumber = 2;
  view.initialize(canvas.getGraphics());
  outPortView.initialize();
   button.selected.addTE((AObject) this,WHENENABLED,"whenEnabled()", SaosMain.AosUser);
   model.enabled.addTE((AObject) this,WHENDISABLED,"whenDisabled()", SaosMain.AosUser);
 }

 public void connectModelViewInput(PortView port) throws SaosException
 {
  model.connectModelViewInput(port);
 }

 public void connectModelViewOutput(PortView port) throws SaosException
 {
  model.connectModelViewOutput(port);
 }

 public void dispatch(int funcIndex) throws SaosException
 {
  switch(funcIndex)
  {
    case WHENENABLED : whenEnabled(); break;
    case WHENDISABLED : whenDisabled(); break;
   default: System.out.println("ERROR : dispatch(): " + this +
       ":funcIndex= " + funcIndex);
  }
 }

 public void delete()
 {
   view.delete();
 }
}
```

รูปที่ ก- 2 แสดงชุดคำสั่งคลาสวิวของตัวสร้างงานที่สร้างได้โดยอัตโนมัติ (ต่อ)

ก-1-2 แถวคอย

ประโยคการเปลี่ยนแปลงที่กำหนดให้กับแถวคอยได้แก่กฎการเปลี่ยนแปลงดังแสดงไว้ใน ตารางที่ ก- 3

- กฎการเปลี่ยนแปลง

ตารางที่ ก- 3 กฎการเปลี่ยนแปลงสำหรับแถวคอยในระบบแถวคอย

เลขที่	เงื่อนไข	การกระทำ
1.	ทุกครั้งที่ nJobs เปลี่ยนแปลง	ifupdate (View)

จากประโยคการเปลี่ยนแปลงที่กำหนดจะได้ชุดคำสั่งในคลาสโมเดลของแถวคอยดังแสดงใน รูปที่ ก- 3 และชุดคำสั่งคลาสวิวของแถวคอยดังแสดงใน รูปที่ ก- 4

```
import saos.gui.*;
import saos.kernel.*;
import java.awt.*;
import java.io.*;
import java.lang.Math;
import java.util.*;

public class Queue  extends  Sink
{

  public Queue(String name)
  {
    super(name);
  }

  public void paint (Graphics g)
  {
  }

  public void initialize()
  {
  }

  public void connectModelViewInput(PortView port) throws SaosException
  {
  }

  public void connectModelViewOutput(PortView port) throws SaosException
  {
  }

  public void dispatch(int funcIndex)
  {
  }
}
```

รูปที่ ก- 3 แสดงชุดคำสั่งคลาสโมเดลของแถวคอยในระบบแถวคอยที่สร้างได้โดยอัตโนมัติ

```
import saos.gui.*;
import saos.kernel.*;
import java.awt.*;
import java.io.*;
import java.lang.Math;
import java.util.*;

class  EditQueue  extends  EditApp implements Constants
{
 Queue  model;
 public Label label;
 static final int IFUPDATE = 0;

 public EditQueue(EditCompound object, AppCanvas canvas, int x, int y)
 {
  super(x, y);
  model = new Queue("Queue");
  view  = (EditCompound) object.clone();
  this.canvas = canvas;
  name = new String("Queue");
  inPortView = new PortView(IN, canvas, this, SINGLE/MULTIPLE);
  outPortView = new PortView(OUT, canvas, this, SINGLE/MULTIPLE);
 }

 public void ifupdate ()
 {
     // **** implement function here****
 }

 public void initialize() throws SaosException
 {
  insert(model, true);
  view.updateCoordinates(x,y, x - view.x, y - view.y);
  view.setColor(Color.green);
  view.setCanvas(canvas);
  inPortStatus = true;
  outPortStatus = true;
  portNumber = 3;
  view.initialize(canvas.getGraphics());
  inPortView.initialize();
  outPortView.initialize();
    model.nJobs.addTE((AObject) this,IFUPDATE,"ifupdate()", SaosMain.AosUser);
 }

 public void connectModelViewInput(PortView port) throws SaosException
 {
  model.connectModelViewInput(port);
 }

 public void connectModelViewOutput(PortView port) throws SaosException
 {
  model.connectModelViewOutput(port);
 }

 public void dispatch(int funcIndex) throws SaosException
 {
  switch(funcIndex)
  {
    case IFUPDATE : ifupdate(); break;
   default: System.out.println("ERROR : dispatch(): " + this +
       ":funcIndex= " + funcIndex);
  }
 }
 public void delete()
 {
   view.delete();
 }
}
```

รูปที่ ก- 4 แสดงชุดคำสั่งคลาสวิวของแถวคอยในระบบแถวคอยที่สร้างได้โดยอัตโนมัติ

ก-1-3 ตัวประมวลผล

ประโยคการเปลี่ยนแปลงที่กำหนดให้กับตัวประมวลผลได้แก่กฎการเปลี่ยนแปลงดังแสดงไว้ใน ตารางที่ ก- 4 และ การกำหนดค่าล่วงหน้าดังแสดงใน ตารางที่ ก- 5

- กฎการเปลี่ยนแปลง

ตารางที่ ก- 4 กฎการเปลี่ยนแปลงสำหรับตัวประมวลผลในระบบแถวคอย

เลขที่	เงื่อนไข	การกระทำ
1.	state == PIdle && inputQueue.nJobs > 0	start (Model)
2.	state == PComplete && outputQueue.nJobs < outputQueue.nSlots	stop(Model)
3.	state == PIdle	ifAvail(View)

- การกำหนดค่าล่วงหน้า

ตารางที่ ก- 5 การกำหนดค่าล่วงหน้าของตัวประมวลผลในระบบแถวคอย

เลขที่	ฟังก์ชันที่ต้องการให้เกิดการกำหนดค่า	ตัวแปรที่ต้องการกำหนดค่า	ค่าที่ต้องการ	เวลาหน่วง
1.	start (Model)	state	PComplete	Math.abs(rand.nextInt( ) % MaxProcesingTime)

จากประโยคการเปลี่ยนแปลงที่กำหนดจะได้ชุดคำสั่งในคลาสโมเดลของแถวคอยดังแสดงใน รูปที่ ก- 5 และชุดคำสั่งคลาสวิวของแถวคอยดังแสดงในรูปที่ ก- 6

```
import saos.gui.*;
import saos.kernel.*;
import java.awt.*;
import java.io.*;
import java.lang.Math;
import java.util.*;

public class Processor  extends  AObject
{
  public static final int PIdle = 0;
  public static final int PBusy = 1;
  public static final int PComplete = 2;
  public int counter;
  public AInteger state;
  public static final int maxProcesingTime = 6;
  public static Random rand;
```

รูปที่ ก- 5 แสดงชุดคำสั่งคลาสโมเดลของตัวประมวลผลในระบบแถวคอยที่สร้างได้โดยอัตโนมัติ

```
 static final int START = 0;
 static final int STOP = 1;
 public Queue inputQueue ;
 public Queue outputQueue ;

 public Processor(String name)
 {
   super(name);
 }

 public void print (int level)
 {
 }

 public void start () throws SaosException
 {
  if(state.val == PIdle && inputQueue.nJobs.val > 0)
  {
     // **** implement function here****
    FAssign.fAssign(this,state,PComplete,
         Math.abs(rand.nextInt() %  maxProcesingTime), "state",  SaosMain.AosUser);
  }
 }

public void stop ()
 {
  if(state.val == PComplete && outputQueue.nJobs.val < outputQueue.nSlots)
  {
     // **** implement function here****
  }
 }

 public void initialize()
 {
  try{
     state.addTE((AObject) this,START,"start()", SaosMain.AosUser);
     state.addTE((AObject) this,STOP,"stop()", SaosMain.AosUser);
  }catch (SaosException e){
    System.out.println(" Error : " + e);
  }
 }

 public void connectModelViewInput(PortView port) throws SaosException
 {
  inputQueue = (Queue) port.address.objectPtr;
  inputQueue.nJobs.addTE((AObject) this,START,"start()", SaosMain.AosUser);
 }

 public void connectModelViewOutput(PortView port) throws SaosException
 {
  outputQueue = (Queue) port.address.objectPtr;
  outputQueue.nJobs.addTE((AObject) this,STOP,"stop()", SaosMain.AosUser);
 }

 public void dispatch(int funcIndex) throws SaosException
 {
  switch(funcIndex)
  {
    case START : start(); break;
    case STOP : stop(); break;
   default: System.out.println("ERROR : dispatch(): " + this +
       ":funcIndex= " + funcIndex);
  }
 }

}
```

รูปที่ ก- 5 แสดงชุดคำสั่งคลาสโมเดลของตัวประมวลผลในระบบแถวคอยที่สร้างได้โดยอัตโนมัติ (ต่อ)

```
import saos.gui.*;
import saos.kernel.*;
import java.awt.*;
import java.io.*;
import java.lang.Math;
import java.util.*;

class  EditProcessor  extends  EditApp implements Constants
{
 Processor  model;
 public static final int PIdle = 0;
 public static final int PBusy = 1;
 static final int IFAVAIL = 0;

 public EditProcessor(EditCompound object, AppCanvas canvas, int x, int y)
 {
  super(x, y);
  model = new Processor("Processor");
  view  = (EditCompound) object.clone();
  this.canvas = canvas;
  name = new String("Processor");
  inPortView = new PortView(IN, canvas, this, SINGLE/MULTIPLE);
  outPortView = new PortView(OUT, canvas, this, SINGLE/MULTIPLE);
 }

 public void paint (Graphics g)
 {
 }

 public void ifAvail ()
 {
  if(model.state.val == PIdle)
  {
     // **** implement function here****
  }
 }

public void initialize() throws SaosException
 {
  insert(model, true);
  view.updateCoordinates(x,y, x - view.x, y - view.y);
  view.setColor(Color.green);
  view.setCanvas(canvas);
  inPortStatus = false;
  outPortStatus = false;
  portNumber = 3;
  view.initialize(canvas.getGraphics());
  inPortView.initialize();
  outPortView.initialize();
   model.state.addTE((AObject) this,IFAVAIL,"ifAvail()", SaosMain.AosUser);
 }

 public void connectModelViewInput(PortView port) throws SaosException
 {
  model.connectModelViewInput(port);
 }

 public void connectModelViewOutput(PortView port) throws SaosException
 {
  model.connectModelViewOutput(port);
 }
```

รูปที่ ก- 6 แสดงชุดคำสั่งคลาสวิวของตัวประมวลผลในระบบแถวคอยที่สร้างได้โดยอัตโนมัติ

```
public void dispatch(int funcIndex) throws SaosException
 {
  switch(funcIndex)
  {
    case IFAVAIL : ifAvail(); break;
   default: System.out.println("ERROR : dispatch(): " + this +
      ":funcIndex= " + funcIndex);
  }
 }


public void delete()
 {
   view.delete();
 }
}
```

รูปที่ ก- 6 แสดงชุดคำสั่งคลาสวิวของตัวประมวลผลในระบบแถวคอยที่สร้างได้โดยอัตโนมัติ (ต่อ)

ก-2 ระบบแท็งค์

ก-2-1 แท็งค์

ประโยคการเปลี่ยนแปลงที่กำหนดให้กับแท็งค์ได้แก่กฎการเปลี่ยนแปลงดังแสดงไว้ในตารางที่ ก- 6

- กฎการเปลี่ยนแปลง

ตารางที่ ก- 6 กฎการเปลี่ยนแปลงของแท็งค์ในระบบแท็งค์

เลขที่	เงื่อนไข	การกระทำ
1.	ทุกครั้งที่ SaosMain.aosTime เปลี่ยนแปลง	aosTimeChanged (Model)
2.	ทุกครั้งที่ refValue เปลี่ยนแปลง	refValueChanged(View)

จากประโยคการเปลี่ยนแปลงที่กำหนดจะได้ชุดคำสั่งในคลาสโมเดลของแท็งค์ดังแสดงในรูปที่ ก- 7 และชุดคำสั่งคลาสวิวของแท็งค์ดังแสดงใน รูปที่ ก- 8

```
import saos.gui.*;
import saos.kernel.*;
import java.awt.*;
import java.io.*;
import java.lang.Math;
import java.util.*;

public class Tank  extends  AObject
{
 public double K1;
 public double K2;
 public ADouble refValue;
 public RatioIndicatorV tankLevel;
 static final int AOSTIMECHANGED = 0;
 public Vector inputValve ;
 public Vector outputValve ;

 public Tank(String name)
 {
   super(name);
 }

 public void sumValveOpeningInFlow ()
 {
 }

 public void sumValveOpeningOutFlow (Vector valve)
 {
 }


 public void aosTimeChanged ()
 {
     // **** implement function here****
 }

 public void initialize()
 {
  try{
     SaosMain.aosTime.addTE((AObject) this,AOSTIMECHANGED,"aosTimeChanged()", SaosMain.AosUser);
  }catch (SaosException e){
     System.out.println(" Error : " + e);
  }
 }

 public void connectModelViewInput(PortView port) throws SaosException
 {
  inputValve.addElement((Object) port.address.imPortObjVector.lastElement());
 }

 public void connectModelViewOutput(PortView port) throws SaosException
 {
  outputValve.addElement((Object) port.address.imPortObjVector.lastElement());
 }

 public void dispatch(int funcIndex) throws SaosException
 {
  switch(funcIndex)
  {
    case AOSTIMECHANGED : aosTimeChanged(); break;
   default: System.out.println("ERROR : dispatch(): " + this +
       ":funcIndex= " + funcIndex);
  }
 }

}
```

รูปที่ ก- 7 แสดงชุดคำสั่งคลาสโมเดลของแท็งค์ในระบบแท็งค์ที่สร้างได้โดยอัตโนมัติ

```
import saos.gui.*;
import saos.kernel.*;
import java.awt.*;
import java.io.*;
import java.lang.Math;
import java.util.*;

class EditTank extends EditApp implements Constants
{
 Tank model;
 public RatioIndicatorV tankLevel;
 public RightArrow refArrow;
 static final int REFVALUECHANGED = 0;

 public EditTank(EditCompound object, AppCanvas canvas, int x, int y)
 {
  super(x, y);
  model = new Tank("Tank");
  view = (EditCompound) object.clone();
  this.canvas = canvas;
  name = new String("Tank");
  inPortView = new PortView(IN, canvas, this, SINGLE/MULTIPLE);
  outPortView = new PortView(OUT, canvas, this, SINGLE/MULTIPLE);
 }

 public void paint (Graphics g)
 {
 }

 public void myConfigure ()
 {
 }

 public void refValueChanged ()
 {
     // **** implement function here****
 }

 public void initialize() throws SaosException
 {
  insert(model, true);
  view.updateCoordinates(x,y, x - view.x, y - view.y);
  view.setColor(Color.green);
  view.setCanvas(canvas);
  inPortStatus = true;
  outPortStatus = true;
  portNumber = 3;
  view.initialize(canvas.getGraphics());
  inPortView.initialize();
  outPortView.initialize();
    model.refValue.addTE((AObject) this,REFVALUECHANGED,"refValueChanged()", SaosMain.AosUser);
 }

 public void connectModelViewInput(PortView port) throws SaosException
 {
  model.connectModelViewInput(port);
 }

 public void connectModelViewOutput(PortView port) throws SaosException
 {
  model.connectModelViewOutput(port);
 }
```

รูปที่ ก- 8 แสดงชุดคำสั่งคลาสวิวของแท็งค์ในระบบแท็งค์ที่สร้างได้โดยอัตโนมัติ

```
public void dispatch(int funcIndex) throws SaosException
{
 switch(funcIndex)
 {
   case REFVALUECHANGED : refValueChanged(); break;
   default: System.out.println("ERROR : dispatch(): " + this +
      ":funcIndex= " + funcIndex);
 }
}

public void delete()
{
  view.delete();
}
}
```

รูปที่ ก- 8 แสดงชุดคำสั่งคลาสวิวของแท็งค์ในระบบแท็งค์ที่สร้างได้โดยอัตโนมัติ (ต่อ)

ก-2-2 วาล์ว

ประโยคการเปลี่ยนแปลงที่กำหนดที่กำหนดให้กับตัววาล์วได้แก่กฎการเปลี่ยนแปลงดังแสดงไว้ใน ตารางที่ ก- 7 และ การเรียกฟังก์ชันล่วงหน้าดังแสดงใน ตารางที่ ก- 8

- กฎการเปลี่ยนแปลง

ตารางที่ ก- 7 กฎการเปลี่ยนแปลงของวาล์วในระบบแท็งค์

เลขที่	เงื่อนไข	การกระทำ
1.	opening.value.val == 0.0	whenOpeningChanged (Model)
2.	manual.selected.val == true	manualPressed (View)
3.	manual.selected.val == true	manualLoop (View)
4.	auto.selected.val == true	autoPressed (View)
5.	open.selected.val == true	openPressed (View)
6.	closed.selected.val == true	closePressed (View)
7.	raise.selected.val == true	raisePressed (View)
8.	lower.selected.val == true	lowerPressed (View)
9.	state.val == CLOSED	whenStateChanged (View)

- การเรียกฟังก์ชันล่วงหน้า

ตารางที่ ก- 8 การเรียกฟังก์ชันล่วงหน้าของวาล์วในระบบแท็งค์

เลขที่	ฟังก์ชันที่ต้องการให้เกิดการเรียกฟังก์ชันล่วงหน้า	ฟังก์ชันที่ต้องการเรียก	เวลาหน่วง
1.	manualLoop (View)	manualLoop (View)	1
2.	autoControl(View)	autoControl(View)	1
3.	autoPressed(View)	autoControl(View)	1

จากประโยคการเปลี่ยนแปลงที่กำหนดจะได้ชุดคำสั่งในคลาสโมเดลของแถวคอยดังแสดงใน รูปที่ ก- 9 และชุดคำสั่งคลาสวิวของแถวคอยดังแสดงใน รูปที่ ก- 10

```
import saos.gui.*;
import saos.kernel.*;
import java.awt.*;
import java.io.*;
import java.lang.Math;
import java.util.*;

public class Valve  extends  AObject
{
  public static final int CLOSED = 0;
  public static final int HALF_OPEN = 1;
  public static final int OPEN = 2;
  public RatioIndicatorH opening;
  public AInteger state;
  static final int WHENOPENINGCHANGED = 0;
  public Pump inputPump ;
  public Pump outputPump ;

  public Valve(String name)
  {
    super(name);
  }
  public void setFlow (double flow)
  {
  }
  public void whenOpeningChanged ()
  {
   if(opening.value.val == 0.0)
   {
      // **** implement function here****
   }
  }

  public void initialize()
  {
    try{
       opening.value.addTE((AObject) this,WHENOPENINGCHANGED,"whenOpeningChanged()",
SaosMain.AosUser);
    }catch (SaosException e){
      System.out.println(" Error : " + e);
    }
  }
```

รูปที่ ก- 9 แสดงชุดคำสั่งคลาสโมเดลของวาล์วในระบบแท็งค์ที่สร้างได้โดยอัตโนมัติ

```
public void connectModelViewInput(PortView port) throws SaosException
{
 inputPump = (Pump) port.address.objectPtr;
}

public void connectModelViewOutput(PortView port) throws SaosException
{
 outputPump = (Pump) port.address.objectPtr;
}

public void dispatch(int funcIndex) throws SaosException
{
 switch(funcIndex)
 {
   case WHENOPENINGCHANGED : whenOpeningChanged(); break;
   default: System.out.println("ERROR : dispatch(): " + this +
      ":funcIndex= " + funcIndex);
 }
}

}
```

รูปที่ ก- 9 แสดงชุดคำสั่งคลาสโมเดลของวาล์วในระบบแท็งค์ที่สร้างได้โดยอัตโนมัติ (ต่อ)

```
import saos.gui.*;
import saos.kernel.*;
import java.awt.*;
import java.io.*;
import java.lang.Math;
import java.util.*;

class EditValve extends EditApp implements Constants
{
 Valve model;
 public double speed;
 public Button11 manual;
 public Button11 auto;
 public Button11 open;
 public Button11 close;
 public Button11 raise;
 public Button11 lower;
 public double refValueChangeRate;
 public double KP;
 public double KI;
 public double KD;
 public RatioIndicatorH opening;
 public static final int CLOSED = 0;
 public static final int HALF_OPEN = 1;
 static final int MANUALPRESSED = 0;
 static final int MANUALLOOP = 1;
 static final int AUTOPRESSED = 2;
 static final int OPENPRESSED = 3;
 static final int CLOSEDPRESSED = 4;
 static final int RAISEPRESSED = 5;
 static final int LOWERPRESSED = 6;
 static final int WHENSTATECHANGED = 7;
 static final int AUTOCONTROL = 8;
```

รูปที่ ก- 10 แสดงชุดคำสั่งคลาสวิวของวาล์วในระบบแท็งค์ที่สร้างได้โดยอัตโนมัติ

```
public EditValve(EditCompound object, AppCanvas canvas, int x, int y)
{
 super(x, y);
 model = new Valve("Valve");
 view  = (EditCompound) object.clone();
 this.canvas = canvas;
 name = new String("Valve");
 inPortView = new PortView(IN, canvas, this, SINGLE/MULTIPLE);
 outPortView = new PortView(OUT, canvas, this, SINGLE/MULTIPLE);
}

public void paint (Graphics g)
{
}

public void myConfigure ()
{
}

public void manualPressed ()
{
 if(manual.selected.val == true)
 {
   // **** implement function here****
 }
}
public void manualLoop () throws SaosException
{
 if(manual.selected.val == true)
 {
   // **** implement function here****
  FCall.fCall(this,MANUALLOOP,1, "manualLoop()",  SaosMain.AosUser);
 }
}

public void autoPressed () throws SaosException
{
 if(auto.selected.val == true)
 {
   // **** implement function here****
  FCall.fCall(this,AUTOCONTROL,1, "autoControl()",  SaosMain.AosUser);
 }
}

public void openPressed ()
{
 if(open.selected.val == true)
 {
   // **** implement function here****
 }
}

public void closedPressed ()
{
 if(close.selected.val == true)
 {
   // **** implement function here****
 }
}

public void raisePressed ()
{
 if(raise.selected.val == true)
 {
   // **** implement function here****
 }
}
```

รูปที่ ก- 10 แสดงชุดคำสั่งคลาสวิวของวาล์วในระบบแท็งค์ที่สร้างได้โดยอัตโนมัติ (ต่อ)

```
public void lowerPressed ()
{
 if(lower.selected.val == true)
 {
   // **** implement function here****
 }
}
public void whenStateChanged ()
{
 if(model.state.val == CLOSED)
 {
   // **** implement function here****
 }
}
public void autoControl () throws SaosException
{
  FCall.fCall(this,AUTOCONTROL,1, "autoControl()",  SaosMain.AosUser);
}

public void initialize() throws SaosException
{
 insert(model, true);
 view.updateCoordinates(x,y, x - view.x, y - view.y);
 view.setColor(Color.green);
 view.setCanvas(canvas);
 inPortStatus = false;
 outPortStatus = false;
 portNumber = 3;
 view.initialize(canvas.getGraphics());
 inPortView.initialize();
 outPortView.initialize();
  manual.selected.addTE((AObject) this,MANUALPRESSED,"manualPressed()", SaosMain.AosUser);
  manual.selected.addTE((AObject) this,MANUALLOOP,"manualLoop()", SaosMain.AosUser);
  auto.selected.addTE((AObject) this,AUTOPRESSED,"autoPressed()", SaosMain.AosUser);
  open.selected.addTE((AObject) this,OPENPRESSED,"openPressed()", SaosMain.AosUser);
  close.selected.addTE((AObject) this,CLOSEDPRESSED,"closedPressed()", SaosMain.AosUser);
  raise.selected.addTE((AObject) this,RAISEPRESSED,"raisePressed()", SaosMain.AosUser);
  lower.selected.addTE((AObject) this,LOWERPRESSED,"lowerPressed()", SaosMain.AosUser);
  model.state.addTE((AObject) this,WHENSTATECHANGED,"whenStateChanged()", SaosMain.AosUser);
}

public void connectModelViewInput(PortView port) throws SaosException
{
 model.connectModelViewInput(port);
}

public void connectModelViewOutput(PortView port) throws SaosException
{
 model.connectModelViewOutput(port);
}

public void dispatch(int funcIndex) throws SaosException
{
 switch(funcIndex)
 {
   case MANUALPRESSED : manualPressed(); break;
   case MANUALLOOP : manualLoop(); break;
   case AUTOPRESSED : autoPressed(); break;
   case OPENPRESSED : openPressed(); break;
   case CLOSEDPRESSED : closedPressed(); break;
   case RAISEPRESSED : raisePressed(); break;
   case LOWERPRESSED : lowerPressed(); break;
   case WHENSTATECHANGED : whenStateChanged(); break;
   case AUTOCONTROL : autoControl(); break;
  default: System.out.println("ERROR : dispatch(): " + this +
      ":funcIndex= " + funcIndex);
 }
}
```

รูปที่ ก- 10 แสดงชุดคำสั่งคลาสวิวของวาล์วในระบบแท็งค์ที่สร้างได้โดยอัตโนมัติ (ต่อ)

```
 public void delete()
 {
   view.delete();
 }
}
```

รูปที่ ก- 10 แสดงชุดคำสั่งคลาสวิวของวาล์วในระบบแท็งค์ที่สร้างได้โดยอัตโนมัติ (ต่อ)

ก-2-3 เครื่องสูบน้ำ

ประโยคการเปลี่ยนแปลงที่กำหนดที่กำหนดให้กับตัวเครื่องสูบน้ำได้แก่กฎการเปลี่ยนแปลงดังแสดงไว้ในตารางที่ ก- 9

- กฎการเปลี่ยนแปลง

ตารางที่ ก- 9 กฎการเปลี่ยนแปลงของเครื่องสูบน้ำในระบบแท็งค์

เลขที่	เงื่อนไข	การกระทำ
1.	ทุกครั้งที่ onOffButton.selected เปลี่ยนแปลง	buttonPressed (View)
2.	state.val == ON	whenStateChanged (View)

จากประโยคการเปลี่ยนแปลงที่กำหนดจะได้ชุดคำสั่งในคลาสโมเดลของเครื่องสูบน้ำดังแสดงในรูปที่ ก- 11 และชุดคำสั่งคลาสวิวของเครื่องสูบน้ำดังแสดงในรูปที่ ก- 12

```
import saos.gui.*;
import saos.kernel.*;
import java.awt.*;
import java.io.*;
import java.lang.Math;
import java.util.*;

public class Pump  extends  AObject
{
 public double flow = 1.0;

 public Pump(String name)
 {
   super(name);
 }
 public void setFlow (double flow)
 {
 }
 public void initialize()
 {
 }

 public void connectModelViewInput(PortView port) throws SaosException
 {
 }
```

รูปที่ ก- 11 แสดงชุดคำสั่งคลาสโมเดลของเครื่องสูบน้ำในระบบแท็งค์ที่สร้างได้โดยอัตโนมัติ

```
 public void connectModelViewOutput(PortView port) throws SaosException
 {
 }

 public void dispatch(int funcIndex)
 {
 }

}
```

รูปที่ ก- 11 แสดงชุดคำสั่งคลาสโมเดลของเครื่องสูบน้ำในระบบแท็งค์ที่สร้างได้โดยอัตโนมัติ (ต่อ)

```
import saos.gui.*;
import saos.kernel.*;
import java.awt.*;
import java.io.*;
import java.lang.Math;
import java.util.*;

class EditPump extends EditApp implements Constants
{
 Pump model;
 public AInteger state;
 public Button11 onOffButton;
 public static final int ON = 0;
 public static final int OFF = 1;
 static final int BUTTONPRESSED = 0;
 static final int WHENSTATECHANGED = 1;

 public EditPump(EditCompound object, AppCanvas canvas, int x, int y)
 {
  super(x, y);
  model = new Pump("Pump");
  view  = (EditCompound) object.clone();
  this.canvas = canvas;
  name = new String("Pump");
  inPortView = new PortView(IN, canvas, this, SINGLE/MULTIPLE);
  outPortView = new PortView(OUT, canvas, this, SINGLE/MULTIPLE);
 }

 public void paint (Graphics g)
 {
 }

 public void myConfigure ()
 {
 }

 public void buttonPressed ()
 {
    // **** implement function here****
 }

 public void whenStateChanged ()
 {
  if(state.val == ON)
  {
    // **** implement function here****
  }
 }
```

รูปที่ ก- 12 แสดงชุดคำสั่งคลาสวิวของเครื่องสูบน้ำในระบบแท็งค์ที่สร้างได้โดยอัตโนมัติ

```
public void initialize() throws SaosException
{
 insert(model, true);
 view.updateCoordinates(x,y, x - view.x, y - view.y);
 view.setColor(Color.green);
 view.setCanvas(canvas);
 inPortStatus = false;
 outPortStatus = false;
 portNumber = 3;
 view.initialize(canvas.getGraphics());
 inPortView.initialize();
 outPortView.initialize();
  onOffButton.selected.addTE((AObject) this,BUTTONPRESSED,"buttonPressed()", SaosMain.AosUser);
  state.addTE((AObject) this,WHENSTATECHANGED,"whenStateChanged()", SaosMain.AosUser);
}

public void connectModelViewInput(PortView port) throws SaosException
{
 model.connectModelViewInput(port);
}

public void connectModelViewOutput(PortView port) throws SaosException
{
 model.connectModelViewOutput(port);
}

public void dispatch(int funcIndex) throws SaosException
{
 switch(funcIndex)
 {
   case BUTTONPRESSED : buttonPressed(); break;
   case WHENSTATECHANGED : whenStateChanged(); break;
  default: System.out.println("ERROR : dispatch(): " + this +
      ":funcIndex= " + funcIndex);
 }
}

public void delete()
{
  view.delete();
}
}
```

รูปที่ ก- 12 แสดงชุดคำสั่งคลาสวิวของเครื่องสูบน้ำในระบบแท็งค์ที่สร้างได้โดยอัตโนมัติ (ต่อ)

ก-3 ระบบเครือข่าย

ก-3-1 ตัวสร้างสัญญาณ

ประโยคการเปลี่ยนแปลงที่กำหนดที่กำหนดให้กับตัวสร้างสัญญาณได้แก่กฎการเปลี่ยนแปลงดังแสดงไว้ในตารางที่ ก- 10 สมการกำหนดค่าดังแสดงไว้ในตารางที่ ก- 11 และ การเรียกฟังก์ชันล่วงหน้าดังแสดงในตารางที่ ก- 12

- กฎการเปลี่ยนแปลง

ตารางที่ ก- 10 กฎการเปลี่ยนแปลงของตัวสร้างสัญญาณในระบบเครือข่าย

เลขที่	เงื่อนไข	การกระทำ
1.	ทุกครั้งที่ carrierOn เปลี่ยนแปลง	stateChanged (Model)
2.	ทุกครั้งที่ nDataRequests เปลี่ยนแปลง	stateChanged (Model)
3.	ทุกครั้งที่ timer.status เปลี่ยนแปลง	stateChanged (Model)
4.	ViewColor.val == true	darken (view)

- สมการกำหนดค่า

ตารางที่ ก- 11 สมการกำหนดค่าของตัวสร้างสัญญาณในระบบเครือข่าย

เลขที่	ตัวแปรที่ต้องการกำหนดค่า	การคำนวณ
1.	collisionDetected	state.val == TRANSMIT && (outputCableSegment.rightSignal.val > 1 || outputCableSegment.leftSignal.val > 1)
2.	carrierOn	outputCableSegment.rightSignal.val > 0 && outputCableSegment.leftSignal.val > 0

- การเรียกฟังก์ชันล่วงหน้า

ตารางที่ ก- 12 การเรียกฟังก์ชันล่วงหน้าของตัวสร้างสัญญาณในระบบเครือข่าย

เลขที่	ฟังก์ชันที่ต้องการให้เกิดการเรียกฟังก์ชันล่วงหน้า	ฟังก์ชันที่ต้องการเรียก	เวลาหน่วง
1.	genRequest (View)	genRequest(View)	Math.abs(rand.nextInt()) % 3 + RQS_INTERVAL

จากประโยคการเปลี่ยนแปลงที่กำหนดจะได้ชุดคำสั่งในคลาสโมเดลของตัวสร้างสัญญาณดังแสดงใน รูปที่ ก- 13 และชุดคำสั่งคลาสวิวของตัวสร้างสัญญาณดังแสดงในรูปที่ ก- 14

```
import java.lang.Math;
import java.util.*;

public class Station  extends  AObject
{
 public static final int IDLE = 0;
 public static final int TRANSMIT = 1;
 public static final int JAM = 2;
 public static final int DEFER_NO_WAIT = 3;
 public static final int DELAY_NO_WAIT = 4;
 public static final int BACK_OFF_DELAY = 5;
 public static final int BACK_OFF = 6;
 public static final int DEFER_WAIT = 7;
 public static final int DELAY_WAIT = 8;
 public static final int BACK_OFF_DEFER = 9;
 public static final int RECEIVE = 10;
 public Timer timer;
 public Random rand;
 public AInteger nDataRequests;
 public ABoolean carrierOn;
 public ABoolean collisionDetected;
 public AInteger state;
 static final int STATECHANGED = 0;
 static final int ALWAYSCOLLISIONDETECTED = 1;
 static final int ALWAYSCARRIERON = 2;
 static final int GENREQUEST = 3;
 public CableSegment outputCableSegment ;

 public Station(String name)
 {
   super(name);
 }

 public void enterTransmit ()
 {
 }

 public void enterJam ()
 {
 }

 public void enterDelayNoWait ()
 {
 }

 public void enterBackOffDelay ()
 {
 }

 public void enterBackOffDefer ()
 {
 }

 public void enterDelayWait ()
 {
 }

 public void detectCarrier ()
 {
 }
```

รูปที่ ก- 13 แสดงชุดคำสั่งคลาสโมเดลของตัวสร้างสัญญาณที่สร้างได้โดยอัตโนมัติ

```
  public void  detectCollision ()
 {
 }

 public void stateChanged ()
 {
     // **** implement function here****
 }


 public void alwayscollisionDetected() throws SaosException
 {
    collisionDetected.setVal(
      state.val == TRANSMIT && ( outputCableSegment.rightSignal.val > 1 ||
          outputCableSegment.leftSignal.val > 1));
 }

 public void alwayscarrierOn() throws SaosException
 {
    carrierOn.setVal(
      outputCableSegment.rightSignal.val > 0 &&  outputCableSegment.leftSignal.val > 0);
 }

 public void genRequest () throws SaosException
 {
    FCall.fCall(this,GENREQUEST,
    Math.abs(rand.nextInt()) % 3 + RQS_INTERVAL, "genRequest()",  SaosMain.AosUser);
 }

 public void initialize()
 {
  try{
     timer.status.addTE((AObject) this,STATECHANGED,"stateChanged()", SaosMain.AosUser);
     nDataRequests.addTE((AObject) this,STATECHANGED,"stateChanged()", SaosMain.AosUser);
     carrierOn.addTE((AObject) this,STATECHANGED,"stateChanged()", SaosMain.AosUser);
     state.addTE((AObject) this,ALWAYSCOLLISIONDETECTED,"alwayscollisionDetected()",
SaosMain.AosUser);
   }catch (SaosException e){
     System.out.println(" Error : " + e);
   }
 }
 public void connectModelViewInput(PortView port) throws SaosException
 {
 }
 public void connectModelViewOutput(PortView port) throws SaosException
 {
   outputCableSegment = (CableSegment) port.address.objectPtr;
   outputCableSegment.rightSignal.addTE((AObject)
this,ALWAYSCOLLISIONDETECTED,"alwayscollisionDetected()", SaosMain.AosUser);
   outputCableSegment.leftSignal.addTE((AObject)
this,ALWAYSCOLLISIONDETECTED,"alwayscollisionDetected()", SaosMain.AosUser);
   outputCableSegment.rightSignal.addTE((AObject) this,ALWAYSCARRIERON,"alwayscarrierOn()",
SaosMain.AosUser);
   outputCableSegment.leftSignal.addTE((AObject) this,ALWAYSCARRIERON,"alwayscarrierOn()",
SaosMain.AosUser);
  }
 public void dispatch(int funcIndex) throws SaosException
 {
  switch(funcIndex)
  {
    case STATECHANGED : stateChanged(); break;
    case ALWAYSCOLLISIONDETECTED : alwayscollisionDetected(); break;
    case ALWAYSCARRIERON : alwayscarrierOn(); break;
    case GENREQUEST : genRequest(); break;
   default: System.out.println("ERROR : dispatch(): " + this +
      ":funcIndex= " + funcIndex);
  }
 }
}
```

รูปที่ ก- 13 แสดงชุดคำสั่งคลาสโมเดลของตัวสร้างสัญญาณที่สร้างได้โดยอัตโนมัติ (ต่อ)

```
import saos.gui.*;
import saos.kernel.*;
import java.awt.*;
import java.io.*;
import java.lang.Math;
import java.util.*;

class EditStation extends EditApp implements Constants
{
 Station model;
 public int portNumber = 0;
 public ABoolean viewColor;
 static final int DARKEN = 0;

 public EditStation(EditCompound object, AppCanvas canvas, int x, int y)
 {
  super(x, y);
  model = new Station("Station");
  view = (EditCompound) object.clone();
  this.canvas = canvas;
  name = new String("Station");
  outPortView = new PortView(OUT, canvas, this, SINGLE/MULTIPLE);
 }

 public void paint (Graphics g)
 {
 }
 public void darken ()
 {
  if(viewColor.val == true)
  {
     // **** implement function here****
  }
 }
 public void initialize() throws SaosException
 {
  insert(model, true);
  view.updateCoordinates(x,y, x - view.x, y - view.y);
  view.setColor(Color.green);
  view.setCanvas(canvas);
  outPortStatus = false;
  portNumber = 2;
  view.initialize(canvas.getGraphics());
  outPortView.initialize();
   viewColor.addTE((AObject) this,DARKEN,"darken()", SaosMain.AosUser);
 }
 public void connectModelViewInput(PortView port) throws SaosException
 {
  model.connectModelViewInput(port);
 }
 public void connectModelViewOutput(PortView port) throws SaosException
 {
  model.connectModelViewOutput(port);
 }
 public void dispatch(int funcIndex) throws SaosException
 {
  switch(funcIndex)
  {
    case DARKEN : darken(); break;
   default: System.out.println("ERROR : dispatch(): " + this +
       ":funcIndex= " + funcIndex);
  }
 }
 public void delete()
 {
   view.delete();
 }
}
```

รูปที่ ก- 14 แสดงชุดคำสั่งคลาสวิวของตัวสร้างสัญญาณในระบบเครือข่ายที่สร้างได้โดยอัตโนมัติ

ก-3-2 สายส่งสัญญาณ

สำหรับสายส่งสัญญาณไม่มีการกำหนดประโยคการเปลี่ยนแปลงใดๆ ชุดคำสั่งคลาสโมเดลที่สร้างได้ของสายส่งสัญญาณในระบบเครือข่ายแสดงได้ดังรูปที่ ก- 15 และชุดคำสั่งในคลาสวิวของสายส่งสัญญาณในระบบเครือข่ายแสดงได้ดัง รูปที่ ก- 16

```
import saos.gui.*;
import saos.kernel.*;
import java.awt.*;
import java.io.*;
import java.lang.Math;
import java.util.*;

public class Cable  extends  AObject
{
 public int nSegments;
 public CableSegment segs[];

 public Cable(String name)
 {
   super(name);
 }

 public void initialize()
 {
 }

 public void connectModelViewInput(PortView port) throws SaosException
 {
 }

 public void connectModelViewOutput(PortView port) throws SaosException
 {
 }

 public void dispatch(int funcIndex)
 {
 }

}
```

รูปที่ ก- 15 แสดงชุดคำสั่งคลาสโมเดลของสายส่งสัญญาณในระบบเครือข่ายที่สร้างได้โดยอัตโนมัติ

```
import saos.gui.*;
import saos.kernel.*;
import java.awt.*;
import java.io.*;
import java.lang.Math;
import java.util.*;

class  EditCable  extends  EditApp implements Constants
{
 Cable  model;
 public int portNumber = 0;
 public EditCableSegment editSegs[];

 public EditCable(EditCompound object, AppCanvas canvas, int x, int y)
 {
   super(x, y);
   model = new Cable("Cable");
   view  = (EditCompound) object.clone();
```

รูปที่ ก- 16 แสดงชุดคำสั่งคลาสวิวของสายส่งสัญญาณในระบบเครือข่ายที่สร้างได้โดยอัตโนมัติ

```
  this.canvas = canvas;
  name = new String("Cable");
  inPortView = new PortView(IN, canvas, this, SINGLE/MULTIPLE);
  outPortView = new PortView(OUT, canvas, this, SINGLE/MULTIPLE);
 }


 public void paint (Graphics g)
 {

 }

 public void draft (Graphics g, int x, int  y, int w, int h)
 {

 }

 public void initialize() throws SaosException
 {
  insert(model, true);
  view.updateCoordinates(x,y, x - view.x, y - view.y);
  view.setColor(Color.green);
  view.setCanvas(canvas);
  inPortStatus = false;
  outPortStatus = false;
  portNumber = 3;
  view.initialize(canvas.getGraphics());
  inPortView.initialize();
  outPortView.initialize();
 }

 public void connectModelViewInput(PortView port) throws SaosException
 {
  model.connectModelViewInput(port);
 }

 public void connectModelViewOutput(PortView port) throws SaosException
 {
  model.connectModelViewOutput(port);
 }

 public void dispatch(int funcIndex)
 {
 }

 public void delete()
 {
   view.delete();
 }
}
```

รูปที่ ก- 16 แสดงชุดคำสั่งคลาสวิวของสายส่งสัญญาณในระบบเครือข่ายที่สร้างได้โดยอัตโนมัติ (ต่อ)

ก-3-3 หน่วยย่อยของสายส่งสัญญาณ

ประโยคการเปลี่ยนแปลงที่กำหนดให้กับหน่วยย่อยของสายส่งสัญญาณได้แก่กฎการเปลี่ยนแปลงดังแสดงไว้ในตารางที่ ก- 13 และ การกำหนดค่าล่วงหน้าดังแสดงในตารางที่ ก- 14

- กฎการเปลี่ยนแปลง

ตารางที่ ก- 13 กฎการเปลี่ยนแปลงของหน่วยย่อยของสายส่งสัญญาณในระบบเครือข่าย

เลขที่	เงื่อนไข	การกระทำ
1.	ทุกครั้งที่ localSignal เปลี่ยนแปลง	propagateRight (Model)
2.	ทุกครั้งที่ rightSignal เปลี่ยนแปลง	propagateRight (Model)
3.	ทุกครั้งที่ localSignal เปลี่ยนแปลง	propagateLeft (Model)
4.	ทุกครั้งที่ leftSignal เปลี่ยนแปลง	propagateLeft (Model)
5.	rightSignal > 0 || leftSignal > 0 || localSignal > 0	darken(View)

- การกำหนดค่าล่วงหน้า

ตารางที่ ก- 14 การกำหนดค่าล่วงหน้าของหน่วยย่อยของสายส่งสัญญาณในระบบเครือข่าย

เลขที่	ฟังก์ชันที่ต้องการให้เกิดการกำหนดค่า	ตัวแปรที่ต้องการกำหนดค่า	ค่าที่ต้องการ	เวลาหน่วง
1.	propagateRight (Model)	rightSegment.rightSignal	rightSignal + localSignal	1
2.	propagateLeft (Model)	leftSegment.leftSignal	leftSignal + localSignal	1

ชุดจากประโยคการเปลี่ยนแปลงที่กำหนดจะได้ชุดคำสั่งในคลาสโมเดลของหน่วยย่อยของสายส่งสัญญาณดังแสดงในรูปที่ ก- 17 และชุดคำสั่งคลาสวิวของหน่วยย่อยของสายส่งสัญญาณดังแสดงใน รูปที่ ก- 18

```
import saos.gui.*;
import saos.kernel.*;
import java.awt.*;
import java.io.*;
import java.lang.Math;
import java.util.*;

public class CableSegment  extends  AObject
{
 public CableSegment rightSegment;
 public CableSegment leftSegment;
 public AInteger localSignal;
 public AInteger rightSignal;
 public AInteger leftSignal;
 public static final int PROPAGATE_LEFT = 3;
 static final int PROPAGATERIGHT = 0;
 static final int PROPAGATELEFT = 1;

 public CableSegment(String name)
 {
   super(name);
 }

 public void propagateRight () throws SaosException
 {
     // **** implement function here****
   FAssign.fAssign(this,rightSegment.rightSignal,rightSignal.val + localSignal.val,1, "rightSegment.rightSignal",
SaosMain.AosUser);
 }

 public void propagateLeft () throws SaosException
 {
     // **** implement function here****
   FAssign.fAssign(this,leftSegment.leftSignal,leftSignal.val + localSignal.val,1, "leftSegment.leftSignal",
SaosMain.AosUser);
 }

 public void initialize()
 {
  try{
     rightSignal.addTE((AObject) this,PROPAGATERIGHT,"propagateRight()", SaosMain.AosUser);
     localSignal.addTE((AObject) this,PROPAGATERIGHT,"propagateRight()", SaosMain.AosUser);
     leftSignal.addTE((AObject) this,PROPAGATELEFT,"propagateLeft()", SaosMain.AosUser);
     localSignal.addTE((AObject) this,PROPAGATELEFT,"propagateLeft()", SaosMain.AosUser);
  }catch (SaosException e){
    System.out.println(" Error : " + e);
  }
 }

 public void connectModelViewInput(PortView port) throws SaosException
 {
 }
 public void connectModelViewOutput(PortView port) throws SaosException
 {
 }
 public void dispatch(int funcIndex) throws SaosException
 {
  switch(funcIndex)
  {
    case PROPAGATERIGHT : propagateRight(); break;
    case PROPAGATELEFT : propagateLeft(); break;
   default: System.out.println("ERROR : dispatch(): " + this +
       ":funcIndex= " + funcIndex);
  }
 }
}
```

รูปที่ ก- 17 แสดงชุดคำสั่งคลาสโมเดลของหน่วยย่อยของสายส่งสัญญาณในระบบเครือข่าย
ที่สร้างได้โดยอัตโนมัติ

```
import saos.gui.*;
import saos.kernel.*;
import java.awt.*;
import java.io.*;
import java.lang.Math;
import java.util.*;

class EditCableSegment extends EditApp implements Constants
{
 CableSegment model;
 public int portNumber;
 static final int DARKEN = 0;

 public EditCableSegment(EditCompound object, AppCanvas canvas, int x, int y)
 {
  super(x, y);
  model = new CableSegment("CableSegment");
  view = (EditCompound) object.clone();
  this.canvas = canvas;
  name = new String("CableSegment");
  inPortView = new PortView(IN, canvas, this, SINGLE/MULTIPLE);
  outPortView = new PortView(OUT, canvas, this, SINGLE/MULTIPLE);
 }

 public void paint (Graphics g)
 {
 }

 public void darken ()
 {
  if(model.leftSignal.val > 0 || model.rightSignal.val > 0
    || model.localSignal.val > 0)
  {
    // **** implement function here****
  }
 }

 public void initialize() throws SaosException
 {
  insert(model, true);
  view.updateCoordinates(x,y, x - view.x, y - view.y);
  view.setColor(Color.green);
  view.setCanvas(canvas);
  inPortStatus = false;
  outPortStatus = false;
  portNumber = 3;
  view.initialize(canvas.getGraphics());
  inPortView.initialize();
  outPortView.initialize();
   model.leftSignal.addTE((AObject) this,DARKEN,"darken()", SaosMain.AosUser);
   model.rightSignal.addTE((AObject) this,DARKEN,"darken()", SaosMain.AosUser);
   model.localSignal.addTE((AObject) this,DARKEN,"darken()", SaosMain.AosUser);
 }

 public void connectModelViewInput(PortView port) throws SaosException
 {
  model.connectModelViewInput(port);
 }

 public void connectModelViewOutput(PortView port) throws SaosException
 {
  model.connectModelViewOutput(port);
 }
```

รูปที่ ก- 18 แสดงชุดคำสั่งคลาสวิวของหน่วยย่อยของสายส่งสัญญาณในระบบเครือข่ายที่สร้างได้โดยอัตโนมัติ

```
public void dispatch(int funcIndex) throws SaosException
{
 switch(funcIndex)
 {
   case DARKEN : darken(); break;
  default: System.out.println("ERROR : dispatch(): " + this +
      ":funcIndex= " + funcIndex);
 }
}

public void delete()
{
  view.delete();
}
}
```

รูปที่ ก- 18 แสดงชุดคำสั่งคลาสวิวของหน่วยย่อยของสายส่งสัญญาณในระบบเครือข่ายที่สร้างได้โดยอัตโนมัติ (ต่อ)

ก-3-4 เครื่องทวนสัญญาณ

ประโยคการเปลี่ยนแปลงที่กำหนดให้กับเครื่องทวนสัญญาณได้แก่กฎการเปลี่ยนแปลงดังแสดงไว้ใน ตารางที่ ก- 15 การกำหนดค่าล่วงหน้าดังแสดงในตารางที่ ก- 16 และการเรียกฟังก์ชันล่วงหน้าดังแสดงในตารางที่ ก- 17

- กฎการเปลี่ยนแปลง

ตารางที่ ก- 15 กฎการเปลี่ยนแปลงของเครื่องทวนสัญญาณในระบบเครือข่าย

เลขที่	เงื่อนไข	การกระทำ
1.	ทุกครั้งที่ inputCableSegment.rightSignal เปลี่ยนแปลง	readFrom1 (Model)
2.	ทุกครั้งที่ inputCableSegment.leftSignal เปลี่ยนแปลง	readFrom1 (Model)
3.	ทุกครั้งที่ outputCableSegment.rightSignal เปลี่ยนแปลง	readFrom2 (Model)
4.	ทุกครั้งที่ outputCableSegment.leftSignal เปลี่ยนแปลง	readFrom2 (Model)
5.	count.val > 0 || inputCableSegment.localSignal > 0 || inputCableSegment.rightSignal > 0 || inputCableSegment.leftSignal > 0 || outputCableSegment.localSignal > 0 || outputCableSegment.rightSignal > 0 || outputCableSegment.leftSignal > 0 ||	darken(View)

- การกำหนดค่าล่วงหน้า

ตารางที่ ก- 16 การกำหนดค่าล่วงหน้าของเครื่องทวนสัญญาณในระบบเครือข่าย

เลขที่	ฟังก์ชันที่ต้องการให้เกิดการกำหนดค่า	ตัวแปรที่ต้องการกำหนดค่า	ค่าที่ต้องการ	เวลาหน่วง
1.	readFrom1 (Model)	outputCableSegment.localSignal	inputCableSegment.rightSignal.val + inputCableSegment.leftSignal.val	repeaterDelay
2.	readFrom2 (Model)	inputCableSegment localSignal	outputCableSegment.rightSignal.val + outputCableSegment.leftSignal.val	repeaterDelay

- การเรียกฟังก์ชันล่วงหน้า

ตารางที่ ก- 17 การเรียกฟังก์ชันล่วงหน้าของเครื่องทวนสัญญาณในระบบเครือข่าย

เลขที่	ฟังก์ชันที่ต้องการให้เกิดการเรียกฟังก์ชันล่วงหน้า	ฟังก์ชันที่ต้องการเรียก	เวลาหน่วง
1.	readFrom1 (Model)	decrement(Model)	repeaterDelay
2.	readFrom2 (Model)	decrement(Model)	repeaterDelay

จากประโยคการเปลี่ยนแปลงที่กำหนดจะได้ชุดคำสั่งในคลาสโมเดลของเครื่องทวนสัญญาณแสดงในรูปที่ ก- 19 และชุดคำสั่งคลาสวิวของเครื่องทวนสัญญาณดังแสดงใน

```
import saos.gui.*;
import saos.kernel.*;
import java.awt.*;
import java.io.*;
import java.lang.Math;
import java.util.*;

public class Repeater  extends  AObject
{
 public AInteger count;
 public static final int repeaterDelay = 5;
 public int oldNumberSignals = 0;
 static final int READFROM1 = 0;
 static final int READFROM2 = 1;
 static final int DECREMENT = 2;
 public CableSegment inputCableSegment ;
 public CableSegment outputCableSegment ;

 public Repeater(String name)
 {
   super(name);
 }
```

รูปที่ ก- 19 แสดงชุดคำสั่งคลาสโมเดลของเครื่องทวนสัญญาณในระบบเครือข่ายที่สร้างได้โดยอัตโนมัติ

```
public void myReconfigure ()
{
}

public void increment ()
{
}

public void decrement ()
{
}

public void readFrom1 () throws SaosException
{
    // **** implement function here****
   FAssign.fAssign(this,outputCableSegment.localSignal,inputCableSegment.rightSignal.val +
inputCableSegment.leftSignal.val,repeaterDelay, "outputCableSegment.localSignal",  SaosMain.AosUser);
   FCall.fCall(this,DECREMENT,repeaterDelay, "decrement()",  SaosMain.AosUser);
}

public void readFrom2 () throws SaosException
{
    // **** implement function here****
   FAssign.fAssign(this,inputCableSegment.localSignal,outputCableSegment.rightSignal.val +
outputCableSegment.leftSignal.val,repeaterDelay, "inputCableSegment.localSignal",  SaosMain.AosUser);
   FCall.fCall(this,DECREMENT,repeaterDelay, "decrement()",  SaosMain.AosUser);
}

public void initialize()
{
}

public void connectModelViewInput(PortView port) throws SaosException
{
 inputCableSegment = (CableSegment) port.address.objectPtr;
 inputCableSegment.leftSignal.addTE((AObject) this,READFROM1,"readFrom1()", SaosMain.AosUser);
 inputCableSegment.rightSignal.addTE((AObject) this,READFROM1,"readFrom1()", SaosMain.AosUser);
}

public void connectModelViewOutput(PortView port) throws SaosException
{
 outputCableSegment = (CableSegment) port.address.objectPtr;
 outputCableSegment.leftSignal.addTE((AObject) this,READFROM2,"readFrom2()", SaosMain.AosUser);
 outputCableSegment.rightSignal.addTE((AObject) this,READFROM2,"readFrom2()", SaosMain.AosUser);
}

public void dispatch(int funcIndex) throws SaosException
{
 switch(funcIndex)
 {
   case READFROM1 : readFrom1(); break;
   case READFROM2 : readFrom2(); break;
   case DECREMENT : decrement(); break;
  default: System.out.println("ERROR : dispatch(): " + this +
     ":funcIndex= " + funcIndex);
 }
}

}
```

รูปที่ ก- 19 แสดงชุดคำสั่งคลาสโมเดลของเครื่องทวนสัญญาณในระบบเครือข่ายที่สร้างได้โดยอัตโนมัติ (ต่อ)

```
import saos.gui.*;
import saos.kernel.*;
import java.awt.*;
import java.io.*;
import java.lang.Math;
import java.util.*;

class EditRepeater extends EditApp implements Constants
{
 Repeater model;
 public int portNumber = 0;
 public int portNumber = 0;
 static final int DARKEN = 0;

 public EditRepeater(EditCompound object, AppCanvas canvas, int x, int y)
 {
  super(x, y);
  model = new Repeater("Repeater");
  view = (EditCompound) object.clone();
  this.canvas = canvas;
  name = new String("Repeater");
  inPortView = new PortView(IN, canvas, this, SINGLE/MULTIPLE);
  outPortView = new PortView(OUT, canvas, this, SINGLE/MULTIPLE);
 }


 public void paint (Graphics g)
 {

 }


 public void darken ()
 {
  if(model.count.val > 0 || model.inputCableSegment.localSignal.val > 0
     || model.inputCableSegment.rightSignal.val > 0 || model.inputCableSegment.leftSignal.val > 0
     || model.outputCableSegment.localSignal.val > 0 || model.outputCableSegment.rightSignal.val > 0
     || model.outputCableSegment.leftSignal.val > 0)
  {
     // **** implement function here****
  }
 }

 public void initialize() throws SaosException
 {
  insert(model, true);
  view.updateCoordinates(x,y, x - view.x, y - view.y);
  view.setColor(Color.green);
  view.setCanvas(canvas);
  inPortStatus = false;
  outPortStatus = false;
  portNumber = 3;
  view.initialize(canvas.getGraphics());
  inPortView.initialize();
  outPortView.initialize();
   model.count.addTE((AObject) this,DARKEN,"darken()", SaosMain.AosUser);
 }

 public void connectModelViewInput(PortView port) throws SaosException
 {
  model.inputCableSegment = (CableSegment) port.address.objectPtr;
  model.inputCableSegment.localSignal.addTE((AObject) this,DARKEN,"darken()", SaosMain.AosUser);
  model.inputCableSegment.rightSignal.addTE((AObject) this,DARKEN,"darken()", SaosMain.AosUser);
  model.inputCableSegment.leftSignal.addTE((AObject) this,DARKEN,"darken()", SaosMain.AosUser);
  model.connectModelViewInput(port);
 }
```

รูปที่ ก- 20 แสดงชุดคำสั่งคลาสวิวของเครื่องทวนสัญญาณในระบบเครือข่ายที่สร้างได้โดยอัตโนมัติ

```
public void connectModelViewOutput(PortView port) throws SaosException
{
 model.outputCableSegment = (CableSegment) port.address.objectPtr;
 model.outputCableSegment.localSignal.addTE((AObject) this,DARKEN,"darken()", SaosMain.AosUser);
 model.outputCableSegment.rightSignal.addTE((AObject) this,DARKEN,"darken()", SaosMain.AosUser);
 model.outputCableSegment.leftSignal.addTE((AObject) this,DARKEN,"darken()", SaosMain.AosUser);
 model.connectModelViewOutput(port);
}

public void dispatch(int funcIndex) throws SaosException
{
 switch(funcIndex)
 {
   case DARKEN : darken(); break;
  default: System.out.println("ERROR : dispatch(): " + this +
      ":funcIndex= " + funcIndex);
 }
}

public void delete()
{
  view.delete();
}
}
```

รูปที่ ก- 20 แสดงชุดคำสั่งคลาสวิวของเครื่องทวนสัญญาณในระบบเครือข่ายที่สร้างได้โดยอัตโนมัติ (ต่อ)

ประวัติผู้เขียนวิทยานิพนธ์

นางสาว จันทร์พร ลัภยพร เกิดเมื่อวันที่ 8 สิงหาคม พ.ศ. 2514 สำเร็จการศึกษาหลักสูตรวิศวกรรมศาสตร์บัณฑิต (วศ.บ.) สาขาวิศวกรรมอุตสาหการ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย เมื่อปีการศึกษา 2537 หลังจากนั้นได้ทำงานในบริษัท แอนเดอร์เซน คอนซัลติ้ง จำกัด จนถึงพ.ศ. 2541 จึงเข้าศึกษาต่อหลักสูตรวิทยาศาสตร์มหาบัณฑิต (วท.บ.) ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย