การพัฒนาบรรณาธิกรณ์แผนภาพสถานะเพื่อเขียนข้อกำหนดรูปนัยคาเฟโอบีเจ

นายมาณพ รัติโชติ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2544

ISBN 974-03-0913-5

A DEVELOPMENT OF A STATE DIAGRAM EDITOR FOR WRITING A FORMAL SPECIFICATION IN CAFEOBJ

Mr. Manop Rattichote

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 2001

ISBN 974-03-0913-5

หัวข้อวิทยานิพนธ์	การพัฒนาบรรณาธิกรณ์แผนภาพสถานะเพื่อเขียนข้อกำหนดรูปนัยคาเฟโอบีเจ
โดย	นายมาณพ รัติโชติ
สาขาวิชา	วิทยาศาสตร์คอมพิวเตอร์
อาจารย์ที่ปรึกษา	ผู้ช่วยศาสตราจารย์ วิวัฒน์ วัฒนาวุฒิ

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

……..……..คณบดีคณะวิศวกรรมศาสตร์
(ศาสตราจารย์ ดร.สมศักดิ์ ปัญญาแก้ว)

คณะกรรมการสอบวิทยานิพนธ์

……..……..ประธานกรรมการ
(รองศาสตราจารย์ ดร.วันชัย ริ้วไพบูลย์)

……..……..อาจารย์ที่ปรึกษา
(ผู้ช่วยศาสตราจารย์ วิวัฒน์ วัฒนาวุฒิ)

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

……..……..กรรมการ
(อาจารย์ ดร.อาทิตย์ ทองทักษ์)

มาณพ รัติโชติ : การพัฒนาบรรณาธิกรณ์แผนภาพสถานะเพื่อเขียนข้อกำหนดรูปนัยคาเฟโอบีเจ (A DEVELOPMENT OF A STATE DIAGRAM EDITOR FOR WRITING A FORMAL SPECIFICATION IN CAFEOBJ) อ.ที่ปรึกษา : ผู้ช่วยศาสตราจารย์ วิวัฒน์ วัฒนาวุฒิ, 87 หน้า. ISBN 974-03-0913-5

วิทยานิพนธ์นี้ได้ออกแบบกฎการแปลงจากแผนภาพสถานะไปเป็นข้อกำหนดรูปนัยคาเฟโอบีเจ ที่พัฒนามาจากภาษาโอบีเจ โดยกฎการแปลงจะถูกใช้เป็นแนวทางในการพิจารณาแต่ละส่วนประกอบของแผนภาพสถานะของยูเอ็มแอล เพื่อหาความสัมพันธ์กับวากยะสัมพันธ์ของภาษาคาเฟโอบีเจ โดยแต่ละแผนภาพสถานะจะอธิบายถึงพฤติกรรมของวัตถุ

จากกฎการแปลงที่ได้ ทำการพัฒนาเครื่องมือสำหรับการวาดแผนภาพสถานะเพื่อใช้สร้างข้อกำหนดรูปนัยคาเฟโอบีเจ สำหรับอำนวยความสะดวกในการกำหนดสมการความเท่ากันของสัจพจน์ของตัวดำเนิน โดยการดึงส่วนประกอบที่เหมาะสมสำหรับการเขียนสมการ

โปรแกรมที่พัฒนาได้รับการทดสอบ และผลลัพธ์ได้รับการยืนยันว่ามีวากยะสัมพันธ์ถูกต้องจากโปรแกรมแปลงภาษาคาเฟโอบีเจ

ภาควิชา วิศวกรรมคอมพิวเตอร์	ลายมือชื่อนิสิต
สาขาวิชา วิทยาศาสตร์คอมพิวเตอร์	ลายมือชื่ออาจารย์ที่ปรึกษา
ปีการศึกษา 2544	ลายมือชื่ออาจารย์ที่ปรึกษาร่วม

##4170463621 : MAJOR COMPUTER SCIENCE
KEYWORD : FORMAL SPECIFICATION / CafeOBJ / STATE DIAGRAM / ALGEBRAIC SPECIFICATION / FORMAL METHOD
MANOP RATTICHOTE : A DEVELOPMENT OF A STATE DIAGRAM EDITOR FOR WRITING A FORMAL SPECIFICATION IN CAFEOBJ. THESIS ADVISOR : ASSIST. PROF. WIWAT VATANAWOOD, 87 pp. ISBN 974-03-0913-5

This research proposed a formal specification method by using the transformation rules to map state diagram notations into CafeOBJ specification, a novel successor of OBJ algebraic language. The transformation rules are used as a guideline to consider each component of UML state diagram and provide a corresponding formal definition in CafeOBJ syntax. In this research, it assumes that each state diagram describes the behavior property of an object.

From the defined transformation rules, a translation tool was developed for drawing the state diagram and generating the CafeOBJ formal specification. The tool helps to define the equation of axiom of each operation by retrieving the appropriate parts – operation, variable for writing equations.

The translation tool is tested and the results are syntactically verified using CafeOBJ Interpreter.

Department Computer Engineering Student's signature
Field of study Computer Science Advisor's signature
Academic year 2001 Co-advisor's signature

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้ได้สำเร็จลุล่วงไปได้ด้วยความช่วยเหลืออย่างดียิ่งของ ผศ. วิวัฒน์ วัฒนาวุฒิ อาจารย์ที่ปรึกษา และขอขอบคุณ รองศาสตราจารย์ ดร. วันชัย ริ้วไพบูลย์ อ.ดร. ธาราทิพย์ สุวรรณศาสตร์ และ อ.ดร. อาทิตย์ ทองทักษ์ กรรมการวิทยานิพนธ์ที่กรุณาเสียสละเวลาให้คำแนะนำ ตรวจและแก้ไขต้นฉบับวิทยานิพนธ์

ขอขอบคุณ เพื่อน ๆ กลุ่ม Formal Method Group ในห้องปฏิบัติการวิศวกรรมซอฟต์แวร์ที่เสียสละเวลาในการให้คำปรึกษา และช่วยตรวจสอบผลการวิจัยที่ได้ ขอขอบคุณเพื่อน ๆ ที่ให้กำลังใจและข้อเสนอแนะต่าง ๆ และ ขอขอบคุณท่านอื่น ๆ ที่มีส่วนช่วยในการทำวิทยานิพนธ์ที่ไม่ได้กล่าวนามมา ณ โอกาสนี้ด้วย

สุดท้ายนี้ ผู้วิจัยใคร่ขอกราบขอบพระคุณ บิดา มารดาที่สนับสนุนในด้านต่าง ๆ และให้กำลังใจแก่ผู้วิจัยเสมอมา

มาณพ รัติโชติ

สารบัญ

หน้า

สารบัญ (ต่อ)

สารบัญภาพ

รูปที่ หน้า

สารบัญภาพ (ต่อ)

รูปที่ หน้า

สารบัญภาพ (ต่อ)

รูปที่ หน้า

บทที่ 1
บทนำ

1 ความเป็นมาและความสำคัญของปัญหา

วิธีรูปนัย (Formal Method) เป็นวิธีที่ใช้ในการเขียนข้อกำหนดของซอฟต์แวร์ (Software Specification) เพื่อช่วยลดความกำกวมในการตีความของผู้ที่จะนำข้อกำหนดที่เขียนขึ้นไปใช้ในขั้นตอนต่อไป เช่น เพื่อนำไปเขียนโปรแกรมหรือเพื่อแก้ไขข้อกำหนดของโปรแกรมที่มีอยู่แล้วให้ถูกต้อง ตามความต้องการของลูกค้าที่เปลี่ยนแปลงไป เป็นต้น ถ้าหากว่าข้อกำหนดของซอฟต์แวร์อยู่ในรูปอื่นเช่น *วิธีอรูปนัย* (Informal Method) ซึ่งใช้ภาษาธรรมชาติ (Natural Language) หรือ *วิธีกึ่งรูปนัย* (Semi-formal Method) ซึ่งใช้แผนภาพ (Diagram) มาอธิบายพฤติกรรมของระบบ อาจทำให้การตีความข้อกำหนดของผู้ที่นำไปใช้ ไม่ตรงกับที่ผู้เขียนข้อกำหนดต้องการอธิบาย อาจส่งผลให้การเขียนโปรแกรม หรือการแก้ไขข้อกำหนด มีข้อผิดพลาด (Error) เกิดขึ้น ดังนั้นจึงได้มีผู้คิดค้นวิธีการเขียนข้อกำหนดที่มีประสิทธิภาพขึ้นมาใช้ นั่นคือ *วิธีรูปนัย* โดยพยายามลดความกำกวมของภาษาธรรมชาติลง โดยการนำเอา *สัญกรณ์* (Notation) และทฤษฎีทางคณิตศาสตร์มาช่วยในการเขียนข้อกำหนดให้ชัดเจนและเป็นที่เข้าใจแบบเดียวกัน โดยเรียกข้อกำหนดที่ได้ว่า *ข้อกำหนดรูปนัย* (Formal Specification) เนื่องจากทฤษฎีและสัญกรณ์ต่าง ๆ ทางคณิตศาสตร์นั้น ได้มีการกำหนดความหมายไว้อย่างชัดเจนแล้ว นอกจากนี้การที่ทำการเขียนข้อกำหนดของระบบซอฟต์แวร์ด้วยวิธีรูปนัย ยังสามารถช่วยหลีกเลี่ยงการออกแบบที่ผิดพลาด ความไม่เข้ากันของระบบ และยังสามารถทำการพิสูจน์ (Prove) ได้ว่าข้อกำหนดที่ได้เขียนขึ้นนั้นมีความถูกต้องมากน้อยเพียงใดด้วยการพิสูจน์ทางคณิตศาสตร์ ดังนั้นการเขียนข้อกำหนดโดยวิธีรูปนัยสามารถช่วยในการหาข้อผิดพลาดทำได้อย่างรวดเร็ว ทำให้ซอฟต์แวร์มีคุณภาพ (Quality) และความน่าเชื่อถือ (Reliability) อีกทั้งยังทำให้เห็นความต้องการของลูกค้าได้ชัดเจนยิ่งขึ้น

คาเฟโอบีเจ (CafeOBJ) เป็น *ภาษาข้อกำหนดรูปนัย* (Formal Specification Language) ภาษาหนึ่งที่นำเอา *รูปแบบข้อกำหนดทางพีชคณิต* (Algebraic Specification Paradigm) มาเป็นพื้นฐานในการเขียนข้อกำหนด และสามารถนำมาอธิบายถึง *ข้อกำหนดเชิงพฤติกรรมของวัตถุ* (Object Behavioral Specification) ที่ทำให้ข้อกำหนดเชิงวัตถุรูปนัยมีประสิทธิภาพ ปัจจุบันยังไม่มีการนำข้อกำหนดรูปนัยมาใช้ในกระบวนการพัฒนาซอฟต์แวร์มากนัก โดยมีสาเหตุดังต่อไปนี้

- นักพัฒนาซอฟต์แวร์ที่มีความรู้ ความเข้าใจ หลักการทางคณิตศาสตร์ ตรรกะ (Logic) และสัญกรณ์ที่ใช้ในภาษาข้อกำหนดรูปนัยมีจำนวนไม่มากนัก

- สัญกรณ์รูปนัย (Formal Notation) ไม่เหมาะในการสื่อสารกับลูกค้าที่เป็นผู้ใช้ระบบ เนื่องจากลูกค้าโดยทั่วไปไม่มีความรู้เกี่ยวและคุ้นเคยกับสัญกรณ์ที่ใช้ในการเขียนข้อกำหนด
- ขาดเครื่องมือที่ช่วยสนับสนุนในการเขียนข้อกำหนดที่ดีพอ

ดังนั้นงานวิจัยนี้จึงได้มีแนวความคิดที่จะพัฒนาเครื่องมือที่จะช่วยให้การเขียนข้อกำหนดรูปนัยของวัตถุ (Object) ให้สามารถใช้งานได้ง่ายและผู้ใช้รู้วากยะสัมพันธ์ (Syntax) ของคาเฟโอบีเจน้อย โดยการใช้แผนภาพสถานะ (State Diagram) ซึ่งเป็นวิธีกึ่งรูปนัยที่ง่ายแก่การเข้าใจและเป็นที่นิยมมาเป็นเครื่องมือ โดยสร้างข้อกำหนดรูปนัยจากแผนภาพสถานะ ให้อยู่ในรูปของภาษาคาเฟโอบีเจ และสามารถนำแผนภาพสถานะที่ได้เป็นตัวช่วยให้การสื่อสารกับลูกค้าทำได้ง่ายขึ้น นอกจากนี้การใช้แผนภาพสถานะยังช่วยให้ผู้เขียนข้อกำหนดของซอฟต์แวร์ สามารถตรวจสอบความสมบูรณ์ของข้อกำหนดที่เขียนขึ้นได้ดีกว่า

2 วัตถุประสงค์

2.1 เพื่อกำหนดความสัมพันธ์ระหว่าง วากยะสัมพันธ์ของ แผนภาพสถานะ กับ คาเฟโอบีเจ

2.2 เพื่อพัฒนาเครื่องมือซอฟต์แวร์ เพื่อสนับสนุนการสร้างข้อกำหนดรูปนัยคาเฟโอบีเจโดยใช้แผนภาพสถานะ

3 ขอบเขตงานวิจัย

3.1 พัฒนาเครื่องมือซอฟต์แวร์เพื่อทำหน้าที่เป็นโปรแกรมบรรณาธิกรณ์ของแผนภาพสถานะ

3.2 ผู้ใช้เครื่องมือสามารถที่จะกำหนดส่วนของข้อกำหนดรูปนัยในแต่ละองค์ประกอบของแผนภาพสถานะได้ ซึ่งประกอบไปด้วย กิจกรรม (Activity) และสมการสัจพจน์ที่อยู่ภายในสถานะ และ เหตุการณ์ (Event) เงื่อนไข (Condition) และการกระทำ (Action) ของการเปลี่ยนสถานะ

3.3 เครื่องมือจะทำการตรวจสอบความถูกต้องของ ชนิดข้อมูล และ การดำเนินการ ที่สามารถนำมาใช้ในการเขียนข้อกำหนดรูปนัยคาเฟโอบีเจ รวมทั้งตรวจสอบวากยะสัมพันธ์ของข้อกำหนดคาเฟโอบีเจ

3.4 ช่วยแจกแจงการดำเนินการ หรือ ชนิดข้อมูลที่กำหนดไว้แล้ว ให้แก่ผู้ใช้ เพื่อนำกลับมาใช้ใหม่ โดยเครื่องมือนี้จะไม่ครอบคลุมถึงข้อกำหนดที่มีลักษณะดังต่อไปนี้ คือ

1) ชนิดข้อมูลที่เป็นระเบียน (Record) เช่น

```
record date {
        day : Int
        month : Int
```

year : Int

}

2) การดำเนินการที่ไม่ได้มีรูปแบบเป็นมาตรฐาน ซึ่งมีอยู่ด้วยกัน 3 ลักษณะ ได้แก่

- การดำเนินการ ที่อยู่หน้าตัวถูกดำเนินการ (Prefix operator) ทั้งหมด เช่น การดำเนินการลบ (-) ที่เปลี่ยนค่าตัวเลขให้เป็นตรงกันข้าม เป็นต้น
- การดำเนินการ ที่อยู่ระหว่างตัวถูกดำเนินการ (Infix operator) เช่น การดำเนินการบวก (+) ซึ่งใช้สำหรับการหาผลรวมของตัวเลข 2 ตัว เป็นต้น
- การดำเนินการ ที่อยู่หลังตัวถูกดำเนินการ (Postfix operator) เช่น การดำเนินการแฟคทอเรียล (Factorial) ที่ใช้สำหรับการหาผลคูณของตัวเลขจาก 1 ถึง n เมื่อ n คือ ตัวถูกดำเนินการ เป็นต้น

เนื่องจาก ในขั้นตอนการกำหนดสมการของสัจพจน์นั้น ไม่สามารถที่จะทราบได้ว่าการดำเนินการที่ต้องการคือการดำเนินการไหน จึงเป็นไปไม่ได้ที่ดึงการดำเนินการหรือตัวแปรที่เหมาะสมมาแสดง

3) เพรดิเคต (Predicate) เนื่องจาก เพรดิเคตสามารถแปลงเป็นการดำเนินการที่มีชนิดของผลลัพธ์เป็น บูลีน (Boolean) คือให้ค่าเป็นจริง (True) หรือ เท็จ (False) เท่านั้น

3.5 ช่วยเรียบเรียงข้อกำหนดรูปนัยและจัดพิมพ์

3.6 ทดสอบการใช้งานโดยทำการตรวจสอบชนิดข้อมูล การดำเนินการ และวากยะสัมพันธ์ ของข้อกำหนดรูปนัยคาเฟโอบีเจ โดยใช้ตัวอย่างจากกรณีศึกษาอย่างน้อย 3 กรณีศึกษา

3.7 เครื่องมือที่พัฒนาขึ้นจะทำงานบนระบบปฏิบัติการวินโดว์ 95 ขึ้นไป

4 ขั้นตอนการดำเนินงาน

4.1 ศึกษางานวิจัยที่เกี่ยวข้อง

4.2 ศึกษาวิธีรูปนัยและวากยะสัมพันธ์ของภาษาคาเฟโอบีเจ

4.3 หาความสัมพันธ์ระหว่างแผนภาพสถานะกับภาษาคาเฟโอบีเจ

4.4 ออกแบบขั้นตอนวิธีในการแปลงจากแผนภาพสถานะเป็นภาษาคาเฟโอบีเจ

4.5 พัฒนาเครื่องมือซอฟต์แวร์

4.6 ตรวจสอบผลการวิจัย

4.7 สรุปผลการวิจัยและจัดทำเอกสาร

5 ประโยชน์ที่จะได้รับ

5.1 เป็นเครื่องมือที่ช่วยสนับสนุนในการเขียนข้อกำหนดรูปนัยคาเฟโอบีเจ

5.2 ช่วยให้สามารถเข้าใจข้อกำหนดรูปนัยได้ดีขึ้นจากแผนภาพที่สร้างขึ้น

5.3 ช่วยให้ผู้เขียนข้อกำหนดซอฟต์แวร์ตรวจสอบความสมบูรณ์ของข้อกำหนดที่เขียนขึ้น

บทที่ 2
งานวิจัยและทฤษฎีที่เกี่ยวข้อง

1 งานวิจัยที่เกี่ยวข้อง

1.1 งานวิจัยเรื่อง Component-based Algebraic Specification : Behavioral Specification for Component-based Software Engineering [1][2] โดย Shusaku Iida Kokichi Futatsugi และ Razvan Diaconescu

เป็นงานวิจัยที่นำเสนอแนวทางในการใช้แผนภาพคลาส (Class Diagram) ของ ยูเอ็มแอล (UML : Unified Modeling Language) เพื่อเป็นส่วนประกอบในการเขียนข้อกำหนดรูปนัยคาเฟโอบีเจอธิบายพฤติกรรมของคอมโพเนนต์ (Component) โดยแต่ละคอมโพเนนต์สามารถนำเสนอโดยการใช้แผนภาพคลาสที่มีหลักในการแปลงดังนี้

- พิจารณาให้ชนิดข้อมูลแฝง (Hidden Sort) เป็น ชื่อคลาส (Class)
- พิจารณาให้การดำเนินการ (Operation) คือ เมทธอด (Method)
- พิจารณาให้การสังเกต (Observation) คือ เมทธอดที่เข้าถึงแอตทริบิวต์ (Attribute)
- พิจารณาให้สถานะเริ่มต้น คือ เมทธอดที่มีชื่อเดียวกับชื่อคลาส

รูปที่ 2.1 แสดงตัวอย่างการอธิบายมอดูลตัวนับ (COUNTER) ซึ่งเป็นข้อกำหนดสำหรับการบวกหรือลบเลขจำนวนเต็ม โดยการใช้แผนภาพคลาสที่แสดงความสัมพันธ์ระหว่างวัตถุที่ประกอบกันเป็นระบบ และแผนภาพเอดีเจ (ADJ Diagram) ที่แสดงชนิดข้อมูล การดำเนินการ ชนิดข้อมูลของอาร์กิวเมนต์และผลลัพธ์ รวมทั้งสมการของสัจพจน์ มาอธิบาย เนื่องจากแผนภาพคลาสอย่างเดียวไม่สามารถแสดงสมการของสัจพจน์ได้ โดยแผนภาพเอดีเจใช้สัญลักษณ์ต่างๆดังรูปที่ 2.2 และประกาศตัวแปรที่ใช้รวมทั้งสมการของสัจพจน์ไว้ใต้แผนภาพ

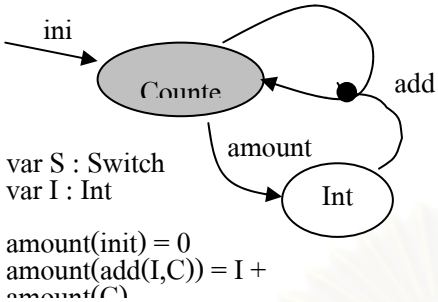
แผนภาพเอดีเจของมอดูลตัวนับในรูปที่ 2.1 สามารถอธิบายแจกแจงชนิดข้อมูล และการดำเนินการดังนี้

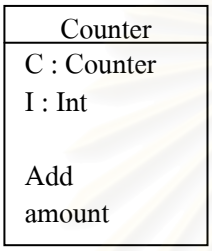
- ชนิดข้อมูลแฝง คือ Counter
- ชนิดข้อมูลสังเกตค่า คือ Int
- การดำเนินการ มีทั้งหมด 3 ตัว คือ

1) init เป็นการดำเนินการที่ตั้งค่าเริ่มต้นของตัวนับ โดยไม่มีข้อมูลเข้า แต่ผลลัพธ์มีชนิดข้อมูลเป็น Counter

2) add เป็นการดำเนินการที่เพิ่ม หรือ ลด ค่าของตัวนับ โดยมีชนิดข้อมูลเข้า 2 ชนิด คือ Counter กับ Int และมีผลลัพธ์ เป็นชนิดข้อมูล Counter

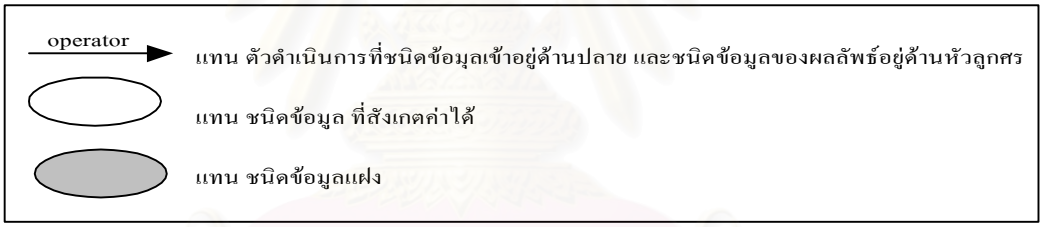
3) read เป็นการดำเนินการที่ใช้ในการดูค่าของตัวนับ โดยมีชนิดข้อมูล Counter เป็นข้อมูลเข้า และ มีผลลัพธ์ เป็นชนิดข้อมูล Int






```
Mod* COUNTER{
    Protecting(INT)

    *[Counter]*

    op init : → Counter
    bop add : Int Counter → Counter
    bop amount : Counter → Int

    var I : Int
    var C : Counter

    eq amount(init(0)) = 0 .
    eq amount(add(I,C)) = I + amount(C)
}
```

รูปที่ 2.1 แผนภาพเอดีเจ แผนภาพคลาส และข้อกำหนดของคอมโพเนนต์ตัวนับ [2]



รูปที่ 2.2 สัญลักษณ์ของแผนภาพเอดีเจ

การใช้แผนภาพคลาสร่วมกับแผนภาพเอดีเจยังคงมีความยากที่ผู้ใช้จะเข้าใจและนำไปใช้ โดยเฉพาะถ้ามอดูลมีขนาดใหญ่และมีความซับซ้อน

1.2 งานวิจัยเรื่อง FOOD: a Graphical Interface for Object-Oriented Algebraic Specifications [3] โดย T.H. Tse

งานวิจัยนี้ได้นำเสนอชุดของส่วนต่อประสานกราฟิก (Graphical Interface) สำหรับระบบการโปรแกรมเชิงวัตถุแบบฟังก์ชัน (Functional Object-Oriented Programming System: FOODS) ซึ่งเป็นภาษาข้อกำหนดรูปนัยเชิงวัตถุที่นำเสนอ และมีการพิจารณาและประมวลความหมายเชิงพีชคณิต (Algebraic Semantics) ไว้ด้วย การประกาศเชิงพีชคณิตของคลาส เมทธอด ลักษณะประจำ และ การสืบทอด (Inheritance) ได้รับการอธิบายไว้ด้วยสัญกรณ์ของแผนภาพการไหลของข้อมูล (Data Flow Diagram) แผนภาพสถานะ และแผนภาพวัตถุ (Object Diagram) ส่วนการกำหนด

สัจพจน์เชิงพีชคณิต (Algebraic Axioms) ซึ่งเป็นคุณสมบัติเชิงพฤติกรรม (Behavioral Properties) ของวัตถุ ได้รับการอธิบายโดยแผนภาพสถานะ แผนภาพการไหลของข้อมูล และแผนภูมิโครงสร้างของวัตถุ (Object Structure Chart) งานวิจัยนี้แสดงตัวอย่างการใช้แผนภาพสถานะเพื่ออธิบายคุณสมบัติเชิงพฤติกรรม

1.3 งานวิจัยเรื่อง Integrating UML and Algebraic Specification Techniques [4] โดย Liliana Favre และ Silvia Clérici

เป็นงานวิจัยที่นำเสนอการใช้แบบจำลองเสถียร (Static Model) ของ ยูเอ็มแอล สำหรับการเขียนข้อกำหนดรูปนัยของภาษา GSBL^{oo} ผู้ใช้สามารถเขียนแผนถาพคลาสเพื่อแสดงถึงความสัมพันธ์ระหว่างวัตถุที่สนใจ โดยความสัมพันธ์ที่งานวิจัยนี้นำเสนอได้แก่ การขึ้นแก่กัน (Dependencies) การร่วมกัน (Associations) การมีลักษณะทั่วไป (Generalizations) และ การทำให้เป็นจริง (Realizations) จากนั้นแผนภาพคลาสดังกล่าว จะได้รับการแปลงเป็นข้อกำหนดรูปนัยของภาษา GSBL^{oo} และนำข้อกำหนดที่ได้ไปสร้างเป็นโปรแกรมเชิงวัตถุของภาษาเอเฟล (Eiffel) ต่อไป

แต่อย่างไรก็ตาม คุณสมบัติเชิงพฤติกรรมของแต่ละวัตถุนั้นไม่ได้รับการกล่าวถึง ซึ่งการใช้แผนภาพคลาสอย่างเดียวนั้น ผู้ใช้เครื่องมือยังไม่สามารถเห็นพฤติกรรมของวัตถุได้อย่างชัดเจน

2 ทฤษฎีที่เกี่ยวข้อง

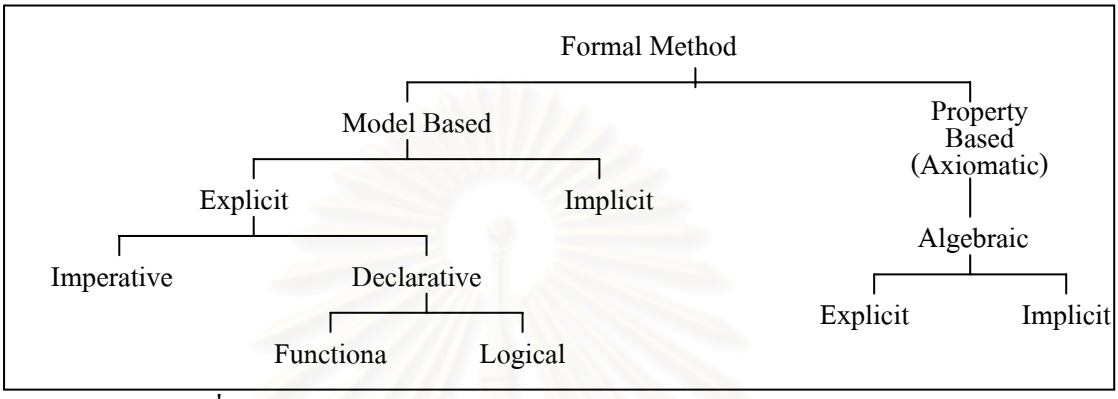
2.1 วิธีรูปนัย (Formal Method)

วิธีรูปนัยเป็นวิธีการที่นำเอาสัญกรณ์และทฤษฎีทางคณิตศาสตร์ที่ได้มีการกำหนดความหมายไว้อย่างดีแล้วมาเป็นเครื่องมือในการอธิบายถึงพฤติกรรมของระบบเพื่อไม่ให้เกิดความกำกวมเหมือนกับที่เกิดขึ้นจากการใช้ภาษาธรรมชาติ และเรายังสามารถตรวจสอบกระบวนการพัฒนาซอฟต์แวร์ได้ตั้งแต่ขั้นตอนของการออกแบบโดยอาศัยการพิสูจน์ทางคณิตศาสตร์

ข้อกำหนดรูปนัยสามารถแบ่งออกเป็น 2 กลุ่มหลัก [5] ด้วยกันคือ *ข้อกำหนดเชิงโมเดล* (Model-Oriented Specification) และ *ข้อกำหนดเชิงพร็อบเพอร์ตี้เบส* (Property Based Specification) ซึ่งถึงแม้ว่าทั้งสองแบบจะมีพื้นฐานมาจาก *ภินทนคณิตศาสตร์* (Discrete Mathematics) เหมือนกัน แต่จุดที่แตกต่างของทั้งสองกลุ่มคือรูปแบบโครงสร้างและวิธีการในการเขียนข้อกำหนด แต่ละกลุ่มจะเหมาะสมในการที่จะนำไปใช้ในการอธิบายข้อกำหนดระบบคอมพิวเตอร์ที่แตกต่างกัน รูปที่ 2.3 แสดงการจัดกลุ่มของรูปแบบการเขียนข้อกำหนด

2.1.1 *ข้อกำหนดเชิงโมเดล*

ข้อกำหนดของระบบถูกนำเสนอเป็นต้นแบบเชิงคณิตศาสตร์นามธรรม (Abstract Mathematical Model) การดำเนินการต่างๆ ถูกแสดงในรูปของการรวมแบบมาตรฐานของชนิดข้อมูลพื้นฐาน และชนิดข้อมูลอื่นๆ ที่สร้างขึ้นใหม่ ซึ่งเหมาะสำหรับการอธิบายระบบที่ชัดเจน



รูปที่ 2.3 การจัดกลุ่มของวิธีรูปนัยตามรูปแบบการเขียนข้อกำหนด [5]

2.1.1.1 *ข้อกำหนดโดยปริยาย* (Implicit Specification)

ข้อกำหนดโดยปริยายนั้น จะบอกรายละเอียดของการทำงานเพียงแค่ว่าต้องการอะไรเพื่อให้การทำงานสำเร็จ แต่จะไม่บอกว่าจะทำได้อย่างไร โดยแสดงในรูปของ *เงื่อนไขก่อนการทำงาน* (Pre-condition) และ *เงื่อนไขหลังการทำงาน* (Post-condition)

2.1.1.2 *ข้อกำหนดชัดแจ้ง* (Explicit Specification)

ข้อกำหนดชัดแจ้ง เป็นข้อกำหนดที่บอกว่าจะทำอย่างไรให้บรรลุถึงเป้าหมาย ซึ่งในส่วนของข้อกำหนดโดยปริยายต้องมี *การทำให้ละเอียดขึ้น* (Refinement) โดยการแปลงให้เป็นข้อกำหนดที่ชัดแจ้งมากขึ้น ซึ่งข้อกำหนดแบบนี้ยังถูกแบ่งออกเป็นกลุ่มย่อยอีกคือ

1) *ข้อกำหนดเชิงคำสั่ง* (Imperative Specification)

ข้อกำหนดเชิงคำสั่ง หรือเรียกว่า ข้อกำหนดเชิงสถานะ (State-Based Specification) เป็นข้อกำหนดที่แสดงในรูปของการเปลี่ยนแปลงค่าของตัวแปรเป็นหลักซึ่งลำดับก่อนหลังของการเรียงคำสั่งภายในข้อกำหนดเชิงคำสั่งนั้นมีความสำคัญ เนื่องจากการเปลี่ยนลำดับก่อนหลังของข้อกำหนดจะก่อให้เกิดผลลัพธ์ที่แตกต่างกัน

2) *ข้อกำหนดเชิงประกาศ* (Declarative Specification)

ข้อกำหนดเชิงประกาศ เป็นข้อกำหนดที่จะแสดงในรูปของกลุ่มของสมการหรือความสัมพันธ์ของข้อมูลเป็นหลักโดยแบ่งออกเป็น

- *ข้อกำหนดเชิงฟังก์ชัน* (Functional Specification) ข้อกำหนดที่ได้จะแสดงในรูปของฟังก์ชันไม่มีการใช้ตัวแปรในลักษณะของ *ตัวแปรส่วนกลาง* (Global Variable) หรือ

ตัวแปรภายนอก (External Variable) การทำงานจะใช้เพียงการส่งผ่านค่าผลลัพธ์ของฟังก์ชันเท่านั้น

- *ข้อกำหนดเชิงตรรกะ* (Logical Specification) ข้อกำหนดที่ได้จะแสดงอยู่ในรูปของนิพจน์ที่สามารถหาค่าความเป็นจริงได้ โดยทั่วไปจะใช้ทฤษฎีตรรกศาสตร์ภาคแสดงในการเขียน

2.1.2 *ข้อกำหนดเชิงพร็อบเพอร์ตี้*

ข้อกำหนดแบบพร็อบเพอร์ตี้เบสจะอธิบายพฤติกรรมของระบบ โดยมีชุดของสัจพจน์ที่เป็นตัวกำหนด *คุณสมบัติ* (Property) ของระบบ ซึ่งคุณสมบัติของระบบ ณ ช่วงเวลาใดๆ ต้องสอดคล้องกับสัจพจน์ที่มีอยู่เท่านั้น โดยใช้ทฤษฎีตรรกศาสตร์ภาคแสดงอันดับแรกบนพื้นฐานของ *ตรรกศาสตร์ของฮอร์* (Hoare Logic) ในการบอกถึงเงื่อนไขก่อนและหลังการทำงานในรูปของสัจพจน์ นิยมใช้อธิบายพฤติกรรมของระบบกระจายและโพรโตคอล

2.1.2.1 *ข้อกำหนดเชิงพีชคณิต* (Algebraic Specification)

เป็นข้อกำหนดที่ใช้อธิบายชนิดข้อมูลเชิงนามธรรม [6][7] (ADTs : Abstract Data Type) ที่อธิบายในรูปของการดำเนินการ (Operations) และ ความสัมพันธ์ระหว่างการดำเนินการ ของชนิดข้อมูลเชิงนามธรรม โดยมีส่วนประกอบ 4 ส่วน ดังนี้

- ส่วนการแนะนำ เป็นส่วนที่ประกาศชนิดของข้อมูลหรือวัตถุ
- ส่วนการบรรยาย เป็นส่วนที่อธิบาย การดำเนินการของชนิดข้อมูลเชิงนามธรรม
- ส่วนการกำหนดลายเซ็นการดำเนินการ (Operation Signature) เป็นส่วนที่กำหนดวากยะสัมพันธ์ของส่วนต่อประสาน (Interface) ของชนิดข้อมูลเชิงนามธรรมหรือวัตถุ
- ส่วนสัจพจน์ เป็นส่วนที่อธิบายถึงคุณสมบัติของการดำเนินการ โดยเขียนในรูปของสมการเชิงพีชคณิต

นั่นคือ การจำลองพฤติกรรมของระบบทำได้โดยการใช้พีชคณิตและสัจพจน์ในการอธิบายลักษณะพฤติกรรมของชนิดข้อมูล และสามารถเขียนข้อกำหนดของระบบที่มีขนาดใหญ่โดยการประกอบข้อกำหนดของระบบย่อยๆที่มีความสมบูรณ์แล้วด้วยทฤษฎีการแบ่งแยก (Category Theory) โดยข้อกำหนดแบบนี้จะใช้ศัพท์ที่หมายถึงชนิดข้อมูลคำว่า *Sort* แทน *Type*

2.2 คาเฟโอบีเจ [8][9]

คาเฟโอบีเจ เป็นภาษาที่ใช้อธิบายข้อกำหนดของระบบ ที่พัฒนามาจากภาษาโอบีเจ (OBJ) ซึ่งเป็นภาษาเชิงพีชคณิต (Algebraic Language) ที่ใช้ตรรกะ (Logic) ในการอธิบาย เพื่อให้สามารถอธิบายระบบได้อย่างชัดเจน ตรรกะที่เพิ่มเข้าไป คือ ตรรกะการเขียนใหม่ (Rewriting Logic) และ

พีชคณิตเชิงพฤติกรรม (Behavioral Algebra) หรือเรียกอีกอย่างว่า พีชคณิตของชนิดข้อมูลแฝง (Hidden-Sorted Algebra)

ตรรกะที่ใช้อธิบายพฤติกรรมของระบบงานดังกล่าว นิยมใช้ตรรกะการเขียนใหม่เพื่ออธิบายข้อกำหนดการทำงานที่พร้อมกัน (Concurrent Specification) และ ใช้ตรรกะพีชคณิตเชิงพฤติกรรม เพื่ออธิบายข้อกำหนดเชิงพฤติกรรม (Behavioral Specification) ซึ่งตรรกะที่เพิ่มเข้าไปนี้เป็นส่วนที่ใช้ในการเขียนข้อกำหนดเชิงวัตถุรูปนัย (Formal Object-Oriented Specification)

โครงสร้างของภาษาคาเฟโอบีเจ ประกอบด้วย 3 ส่วน ดังรูปที่ 2.4 คือ

1) ส่วนการประกาศมอดูล (Module Declaration)

ในส่วนนี้ใช้สำหรับการประกาศชื่อมอดูล การประกาศพารามิเตอร์ (ถ้ามี) และการประกาศการนำเข้ามอดูลอื่นมาใช้เพื่อการอธิบายคุณสมบัติของมอดูลปัจจุบัน โดยมอดูลในภาษาคาเฟโอบีเจมีได้ 2 ลักษณะ คือ

- โมเดลแบบยึดติด (Tight Modules หรือ Tight Denotation) คือ มอดูลที่เป็นเอกเทศ (Unique) เพราะเมื่อนำข้อกำหนดรูปนัยไปสร้างเป็นระบบที่สามารถทำงานได้นั้น ผู้พัฒนาจะสามารถทำได้แค่ 1 รูปแบบ หรือ 1 โมเดล (Model) เท่านั้น
- โมเดลแบบอิสระ (Loose Modules หรือ Loose Denotation) คือ มอดูลที่มีลักษณะเป็นคลาสของโมเดล หรือมอดูลที่มีมากกว่า 1 โมเดล

การประกาศการนำเข้า มีอยู่ 3 แบบ คือ

- การนำเข้าแบบป้องกัน (Protecting Mode) การนำเข้าแบบนี้จะไม่ยอมให้มีการเพิ่มหรือยุบส่วนย่อย (Element) ของมอดูลที่นำเข้ามาโดยเด็ดขาด
- การนำเข้าแบบขยาย (Extending Mode) การนำเข้าแบบนี้สามารถทำการเพิ่มส่วนย่อยของมอดูลที่นำเข้ามาได้ แต่จะไม่ยอมให้มีการยุบส่วนย่อย
- การนำเข้าแบบการใช้ (Using Mode) การนำเข้าแบบนี้สามารถทำการเพิ่มหรือยุบส่วนย่อยของมอดูลที่นำเข้ามาได้ตามต้องการ

2) ส่วนการกำหนดลายเซ็น (Signatures Declaration)

ในส่วนนี้ใช้ในการกำหนดชนิดของข้อมูล (Sort) และสัญลักษณ์ทางฟังก์ชันที่กระทำกับชนิดข้อมูลที่ได้กำหนดไว้แล้ว โดยจะมีรูปแบบของแต่ละส่วนดังนี้

ก) การกำหนดชนิดของข้อมูล (Sort Declaration)

โดยจะแบ่งได้เป็น 2 ประเภท คือ

- ชนิดข้อมูลที่สังเกตค่าได้ (Visible Sort) เช่น จำนวนเต็ม (Integer) บูลีน (Boolean) เป็นต้น มีรูปแบบในการกำหนด ดังนี้

[sort_name1 sort_name2 …]

โดยที่

[ ] ใช้สำหรับชนิดข้อมูลที่สังเกตค่าได้

sort_name1 sort_name2 … คือ รายการชื่อชนิดข้อมูลที่ประกาศ

- ชนิดข้อมูลแฝง (Hidden Sort) เช่น แถวลำดับของจำนวนเต็ม (Array of Integer) ลิฟต์ (Lift) เป็นต้น ซึ่งค่าของชนิดข้อมูลสามารถพิจารณาได้โดยผ่านชนิดข้อมูลที่สังเกตค่าได้ เช่น จำนวนเต็ม โดยมีรูปแบบในการกำหนด ดังนี้

[hsort_name1 hsort_name2 …]

โดยที่

[ ] ใช้สำหรับชนิดข้อมูลแฝง

hsort_name1 hsort_name2 … คือ รายการชนิดข้อมูลแฝงที่ประกาศ

การกำหนดว่าเป็น ชนิดข้อมูลที่สังเกตค่าได้ หรือ ชนิดข้อมูลแฝงนั้น ขึ้นอยู่กับการพิจารณาของผู้เขียนข้อกำหนดว่าต้องการให้ชนิดข้อมูลใดเก็บซ่อนหรือเปิดเผยค่าให้รู้ได้

ในกรณีที่มีการกำหนดในลักษณะของชนิดข้อมูลย่อย (Subsort) กล่าวคือ ชนิดข้อมูลหนึ่งเป็นเซตย่อยของอีกชนิดข้อมูลหนึ่ง จะมีรูปแบบ ดังนี้

[sort_name1 < sort_name2]

[hsort_name1 < hsort_name2]

โดยที่

sort_name1 < sort_name2 คือ sort_name1 เป็น ชนิดข้อมูลย่อยของ sort_name2 สำหรับชนิดข้อมูลสังเกตค่า

hsort_name1 < hsort_name2 คือ hsort_name1 เป็นชนิดข้อมูลย่อยของ hsort_name2 สำหรับชนิดข้อมูลแฝง

ข) การกำหนดการดำเนินการ (Operation Declaration)

สามารถแบ่งได้เป็น 3 ชนิด คือ

- การดำเนินการเชิงพฤติกรรม (Behavioral Operation) เป็นการดำเนินการที่มีผลทำให้สถานะของชนิดข้อมูล หรือวัตถุเปลี่ยนแปลง โดยการดำเนินการประเภทนี้จะมีอาร์กิวเมนต์ (Argument or Arity) อย่างน้อย 1 ตัวเป็น ชนิดข้อมูลแฝงของมอดูลปัจจุบัน มีรูปแบบ ดังนี้

bop op_name : sort_name1 sort_name2 … → sort

โดยที่

bop คือ คำสำคัญที่ใช้กำหนดการดำเนินการเชิงพฤติกรรม

op_name คือ ชื่อการดำเนินการ

sort_name1 sort_name2 … คือ รายการของอาร์กิวเมนต์ ที่มีอย่างน้อย 1 อาร์กิวเมนต์เป็นชนิดข้อมูลแฝงของมอดูลปัจจุบัน

sort คือ ชนิดของผลลัพธ์

- การดำเนินการสังเกตค่า (Observation Operation) เป็นการดำเนินการที่ไม่มีผลทำให้สถานะของชนิดข้อมูล หรือวัตถุเปลี่ยนแปลง แต่ใช้สำหรับการดูค่าของชนิดข้อมูล หรือวัตถุ ณ สถานะใดสถานะหนึ่ง โดยชนิดของผลลัพธ์ของการดำเนินการประเภทนี้ จะเป็นแบบสังเกตค่าได้ มีรูปแบบ ดังนี้

op op_name : sort_name1 sort_name2 … → vsort

หรือ

bop op_name : sort_name1 sort_name2 … → vsort

โดยที่

op bop คือ คำสำคัญที่ใช้กำหนดการดำเนินการ

op_name คือ ชื่อการดำเนินการ

sort_name1 sort_name2 … คือ รายการของอาร์กิวเมนต์

vsort คือ ชนิดข้อมูลที่สังเกตค่าได้ (Visible Sort)

- การดำเนินการค่าคงที่ (Constant Operation) เป็นการดำเนินการที่ใช้สำหรับการกำหนดค่าคงที่ของชนิดข้อมูลหรือวัตถุ โดยที่การดำเนินการประเภทนี้ไม่มีรายการของอาร์กิวเมนต์ มีรูปแบบ ดังนี้

op op_name : → sort

โดยที่

op คือ คำสำคัญที่ใช้กำหนดการดำเนินการ

op_name คือ ชื่อการดำเนินการ

sort คือ ชนิดข้อมูลใดๆ

3) ส่วนการกำหนดสัจพจน์ (Axioms Declaration)

เป็นส่วนที่ใช้ในการกำหนดพฤติกรรมของชนิดข้อมูลหรือวัตถุ ด้วยการใช้สมการทางคณิตศาสตร์มาเป็นตัวกำหนด ประกอบด้วยการกำหนดตัวแปรและสมการ ดังนี้

ก) การกำหนดตัวแปร (Variable Declaration)

การกำหนดตัวแปร ใช้สำหรับการกำหนดตัวแปรทั้งหมดที่อ้างอิงในการเขียนสมการ เพื่ออธิบายพฤติกรรมของชนิดข้อมูลหรือวัตถุ มีรูปแบบ ดังนี้

```
var var_name : sort_name
```

หรือ

```
vars var_name1, var_name2, … : sort_name
```

โดยที่

var vars คือ คำสำคัญที่ใช้กำหนดตัวแปร

var_name คือ ชื่อของตัวแปร

var_name1, var_name2, … คือ รายการชื่อของตัวแปร ที่มีชนิดข้อมูลเดียวกัน

sort_name คือ ชนิดข้อมูล

ข) การกำหนดสมการ (Equation Declaration)

สามารถแบ่งได้เป็น 2 ประเภท คือ

- สมการแบบไม่มีเงื่อนไข (Unconditional Equation Declaration) มีรูปแบบ ดังนี้

```
eq term = term .
```

หรือ

```
beq term = term .
```

โดยที่

eq beq คือ คำสำคัญที่ใช้ในการกำหนดสมการ และสมการเชิงพฤติกรรมแบบไม่มีเงื่อนไข

term คือ พจน์ทางคณิตศาสตร์

- สมการแบบมีเงื่อนไข (Conditional Equation Declaration) มีรูปแบบ ดังนี้

```
ceq term = term if boolean_term .
```

หรือ

```
bceq term = term if boolean_term .
```

โดยที่

ceq cbeq คือ คำสำคัญที่ใช้ในการกำหนดสมการ และสมการเชิงพฤติกรรมแบบมีเงื่อนไข

term คือ พจน์ทางคณิตศาสตร์

boolean_term คือ term ที่ให้ผลลัพธ์เป็น จริง หรือ เท็จ

ทั้งนี้ เพื่อความเข้าใจรายละเอียดได้ง่ายขึ้น จึงได้นำเสนอวากยะสัมพันธ์ในรูปแบบของแผนภูมิต้นไม้ของคาเฟโอบีเจในภาคผนวก ก

คาเฟโอบีเจเป็นเครื่องมือสำหรับที่ใช้ในการตรวจสอบวากยะสัมพันธ์ และเป็นตัวพิสูจน์ความถูกต้องของข้อกำหนดที่ได้เขียนขึ้น โดยตัวอย่างของข้อกำหนดรูปนัยคาเฟโอบีเจเป็นดังรูปที่ 2.4 ซึ่งเป็นมอดูลสำหรับการนับ ที่มีสวิตซ์เป็นตัวควบคุม โดยจะมีการนำเข้ามอดูลย่อย 2 มอดูล คือ

- มอดูลตัวนับ

  เป็นมอดูลที่ทำหน้าที่ บวก หรือ ลบเลขจำนวนเต็ม

- มอดูลสวิตซ์

  เป็นมอดูลที่ทำหน้าที่ควบคุมว่าจะทำการบวก หรือ การลบ โดยถ้าสวิตซ์เป็น on จะเป็นการบวกเลข ถ้าสวิตซ์เป็น off จะเป็นการลบเลข

2.3 แผนภาพสถานะของยูเอ็มแอล [10][11][12]

แผนภาพสถานะได้มีการใช้มาตั้งแต่ยุคเริ่มต้นของการทำแบบจำลองเชิงวัตถุ (Object-Oriented Modeling) โดยนำหลักการของเครื่องจักรสถานะจำกัด (Finite State Machine) ซึ่งใช้ในการออกแบบฮาร์ดแวร์ มาประยุกต์เพื่อให้เหมาะสมกับการออกแบบซอฟต์แวร์ โดยจะมีการรับและตอบสนองต่อสิ่งเร้าที่เกิดจากเหตุการณ์จากภายนอก หรือเป็นพฤติกรรมจากตัวมันเอง ซึ่งจะมีผลทำให้มีการเปลี่ยนสถานะตามความเหมาะสม

แผนภาพสถานะใช้สำหรับการอธิบายพฤติกรรมของวัตถุหรือการตอบโต้กันระหว่างวัตถุ โดยจะอธิบายลำดับที่เป็นไปได้ของสถานะและการกระทำตลอดอายุ (Lifetime) ของวัตถุหรือระบบ และเพื่อให้ระบบที่ออกแบบไม่ซับซ้อนมากจนเกินไป เมื่อระบบมีขนาดใหญ่ผู้ออกแบบจึงมักจะแบ่งระบบออกเป็นส่วนย่อยๆ ที่มีขนาดเหมาะสม และส่วนย่อยๆ นี้เองที่ในการวิธีการเชิงวัตถุ มักจะเรียกว่าเป็น คลาส หรือโครงร่างของวัตถุในระบบ

สัญลักษณ์ (Notations) และความหมาย (Semantics) ของแผนภาพสถานะเชิงวัตถุ ได้รับการพัฒนามาจากแผนภาพสถานะของ David Harel และยังได้มีการรวมลักษณะการทำให้เกิดผลใน 2 ลักษณะคือ การดำเนินการสัมพันธ์กับการเปลี่ยนสถานะ หรือเรียกว่า เครื่องกลมีลี (Mealy Machine) และ การดำเนินการสัมพันธ์กับสถานะ หรือเรียกว่า เครื่องกลมอร์ (Moore Machine)

```
mod* COUNTER-WITH-SWITCH {
        protecting(COUNTER + SWITCH)

        *[ Cws ]*

        op init : -> Cws
        bop put : Int Cws -> Cws
        bop add : Cws -> Cws
        bop sub : Cws -> Cws
        bop read : Cws -> Cws
        bop counter : Cws -> Counter
        bop switch : Cws -> Switch

        var N : Int
        var C : Cws

        eq read(C) = read(counter(C)) .

        eq switch(init) = init-switch .
        eq switch(put(N,C)) = switch(C) .
        eq switch(add(C)) = on(switch(C)) .
        eq switch(sub(C)) = off(switch(C)) .

        eq counter(init) = init-counter .
        eq counter(put(N,C)) = add(N,counter(C)) if state(switch(C)) == on .
        eq counter(put(N,C)) = add(-N,counter(C)) if state(switch(C)) == off .
        eq counter(add(C)) = counter(C) .
        eq counter(sub(C)) = counter(C) .
}
```

ส่วนการกำหนดมอดูล

ส่วนการกำหนดลายเซ็นการดำเนินการ

ส่วนการกำหนดสัจพจน์

รูปที่ 2.4 ข้อกำหนดรูปนัยคาเฟโอบีเจของมอดูลตัวนับที่มีสวิตซ์

แผนภาพสถานะประกอบด้วย 2 กลุ่ม คือ เซตของสถานะ (States) และเซตของการเปลี่ยนแปลง (Transitions) โดยมีรายละเอียดดังนี้

2.3.1 เซตของสถานะ

สถานะของแผนภาพสถานะ ซึ่งแทนด้วยสัญลักษณ์ดังรูปที่ 2.5 เป็นการบอกให้ทราบว่าเมื่อวัตถุเข้าสู่สถานะนั้นแล้ว ไม่ว่าก่อนหน้านี้จะอยู่ในสถานะไหน วัตถุจะมีพฤติกรรมเหมือนกัน โดยในสถานะประกอบด้วย 2 ส่วน คือ

- ชื่อสถานะ
- การดำเนินการ ซึ่งไม่มีผลทำให้สถานะเปลี่ยนแปลง

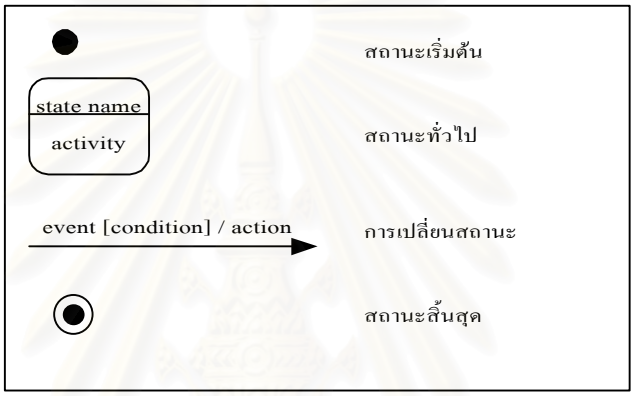
แผนภาพสถานะยังมีสถานะพิเศษอีก 2 สถานะ คือ

1) สถานะเริ่มต้น (Start State) เป็นจุดเริ่มต้นของวัตถุ ซึ่งทุกแผนภาพสถานะต้องมี แทนด้วยสัญลักษณ์ดังรูปที่ 2.5
2) สถานะสิ้นสุด (End State) เป็นจุดสิ้นสุดของวัตถุ แทนด้วยสัญลักษณ์ดังรูปที่ 2.5

2.3.2 เซตของการเปลี่ยนแปลง

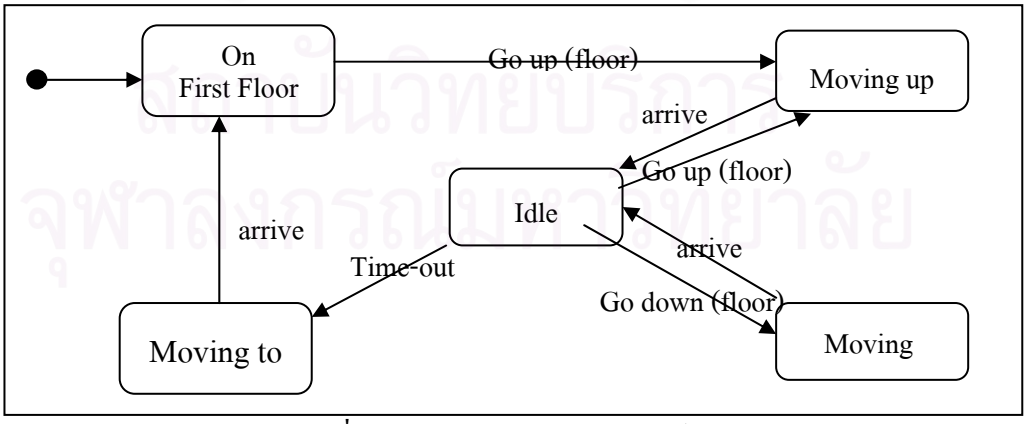
การเปลี่ยนแปลงของแผนภาพสถานะ แทนด้วยสัญลักษณ์ดังรูปที่ 2.5 คือการที่วัตถุเปลี่ยนพฤติกรรม จากพฤติกรรม ณ สถานะต้นทาง (Source State) เป็นพฤติกรรม ณ สถานะเป้าหมาย (Target State) โดยการเปลี่ยนแปลงประกอบด้วย 3 ส่วน คือ

- เหตุการณ์ (Event) ซึ่งจะมีผลทำให้มีการเปลี่ยนสถานะ
- เงื่อนไข (Guard Condition) ซึ่งจะถูกตรวจสอบว่าเป็น จริง หรือ เท็จ เมื่อมีเหตุการณ์เกิดขึ้น
- การดำเนินการ (Action) ซึ่งจะถูกประมวลผลเมื่อเงื่อนไขเป็นจริง



รูปที่ 2.5 สัญลักษณ์ของแผนภาพสถานะ

ตัวอย่างดังรูปที่ 2.6 แสดงแผนภาพสถานะของลิฟต์สำหรับการ ขึ้น-ลงอาคาร โดยสถานะเริ่มต้นนั้น ลิฟต์จะอยู่ที่ชั้นที่ 1 ถ้าหากลิฟต์ไม่ได้อยู่ที่ชั้นที่ 1 มากกว่าระยะเวลาที่กำหนดแล้ว ลิฟต์จะเคลื่อนมายังชั้นที่ 1 โดยอัตโนมัติ



รูปที่ 2.6 แผนภาพสถานะของลิฟต์ [12]

บทที่ 3
กฎการแปลงระหว่างแผนภาพสถานะกับคาเฟโอบีเจ

ในการพัฒนาบรรณาธิกรณ์แผนภาพสถานะเพื่อเขียนข้อกำหนดรูปนัยคาเฟโอบีเจ ลำดับแรกนั้นต้องหาความสัมพันธ์ระหว่างวากยะสัมพันธ์ของแผนภาพสถานะและวากยะสัมพันธ์ของภาษารูปนัยคาเฟโอบีเจก่อน เพื่อกำหนดกฎในการแปลงโดยจำเป็นที่ต้องพิจารณาดูว่าแต่ละส่วนของแผนภาพสถานะนั้น จะมีความเกี่ยวข้องอย่างไรกับแต่ละส่วนของภาษาคาเฟโอบีเจ โดยมีรายละเอียด ดังต่อไปนี้

1 กฎการแปลงและความสัมพันธ์ระหว่างแผนภาพสถานะและข้อกำหนดรูปนัยคาเฟโอบีเจ

จากการศึกษาแผนภาพสถานะของยูเอ็มแอล พบว่า แผนภาพสถานะ 1 แผนภาพนั้นมักใช้สำหรับอธิบายพฤติกรรมของวัตถุ 1 วัตถุ และมีส่วนประกอบที่สำคัญ คือ

1) สถานะ

มีอยู่ด้วยกัน 3 รูปแบบ คือ

- สถานะเริ่มต้น

  วัตถุใดๆ ต้องมีสถานะเริ่มต้นเสมอ

- สถานะทั่วไป

  มีกิจกรรมภายในได้ โดยเมื่อกิจกรรมถูกประมวลผลจะไม่มีผลทำให้เกิดการเปลี่ยนสถานะ

- สถานะสิ้นสุด

2) การเปลี่ยนสถานะ

การเปลี่ยนสถานะ มีส่วนประกอบย่อยอีกคือ เหตุการณ์ เงื่อนไข และการกระทำ โดยที่ทั้ง 3 ส่วนมีผลก่อให้เกิดการเปลี่ยนสถานะ โดยการเปลี่ยนสถานะนั้นอาจเป็นการเปลี่ยนสถานะจากสถานะเริ่มต้นไปยังสถานะทั่วไป หรือ จากสถานะทั่วไปไปยังสถานะทั่วไปใดๆ หรือ จากสถานะทั่วไปไปยังสถานะสิ้นสุด เป็นต้น และหากว่าการเปลี่ยนสถานะมีเงื่อนไขและการกระทำเป็นส่วนประกอบแล้ว การกระทำจะถูกประมวลผลก็ต่อเมื่อเงื่อนไขเป็นจริงเท่านั้น ซึ่งการเปลี่ยนสถานะจากสถานะเริ่มต้นไปยังสถานะทั่วไปที่พร้อมทำงานนั้น เป็นการสร้างวัตถุขึ้นมาเฉยๆ หรืออาจจะมีการกำหนดค่าเริ่มต้นให้กับวัตถุด้วย

ส่วนสัญกรณ์ทางภาษาของข้อกำหนดรูปนัยคาเฟโอบีเจนั้น มีส่วนประกอบที่สำคัญดังนี้

- ส่วนการกำหนดมอดูล

- ส่วนการกำหนดลายเซ็นการดำเนินการ (Operation Signature) เป็นส่วนที่ใช้กำหนดการดำเนินการที่เป็นไปได้ทั้งหมด รวมทั้งชนิดข้อมูลของอาร์กิวเมนต์ และผลลัพธ์ของแต่ละการดำเนินการ โดยจะมีทั้งการดำเนินที่กำหนดค่าเริ่มต้นของมอดูล เปลี่ยนสถานะของมอดูล หรือสังเกตค่าของมอดูล
- ส่วนการกำหนดสัจพจน์ เป็นการอธิบายคุณสมบัติของการดำเนินการ ที่เขียนอยู่ในรูปของสมการ ซึ่งอาจจะเป็นสมการที่มีหรือไม่มีเงื่อนไขก็ได้

จากส่วนประกอบต่างๆ ของแผนภาพสถานะนั้น เมื่อนำมาเปรียบเทียบกับสัญกรณ์ทางภาษาของข้อกำหนดรูปนัยคาเฟโอบีเจ จะได้ว่า

- แผนภาพ 1 แผนภาพ สามารถนำมาอธิบายพฤติกรรมของข้อกำหนดรูปนัยคาเฟโอบีเจ 1 มอดูลได้ จากรูปที่ 3.1 คือหมายเลข 1
- เหตุการณ์ การกระทำ และกิจกรรมของแผนภาพ มีผลให้มีการเปลี่ยนสถานะหรือค่าของวัตถุ ดังนั้นจึงสามารถที่จะแปลงเป็นการดำเนินการของข้อกำหนดรูปนัยคาเฟโอบีเจ โดยที่การกระทำที่ทำให้สถานะเปลี่ยนจากสถานะเริ่มต้นเป็นสถานะพร้อมทำงาน สามารถแปลงเป็นการดำเนินการกำหนดค่าเริ่มต้นของมอดูล ซึ่งจากรูปที่ 3.1 คือหมายเลข 2 ส่วนการกระทำ ณ ตำแหน่งอื่นๆ นั้นก็แปลงเป็นการดำเนินการเชิงพฤติกรรม เช่นเดียวกับเหตุการณ์ และกิจกรรม ซึ่งจากรูปที่ 3.1 คือหมายเลข 3 4 และ 6 ตามลำดับ
- เงื่อนไขของการเปลี่ยนสถานะ ที่มีผลกับการกระทำของการเปลี่ยนสถานะชุดเดียวกัน สามารถแปลงเป็นเงื่อนไขของสมการของสัจพจน์ของการกระทำนั้นๆ ซึ่งจากรูปที่ 3.1 คือหมายเลข 5

โดยที่ แต่ละหมายเลขคือกฎแต่ละข้อ ส่วนกฎข้อที่ 7 ใช้สำหรับการที่แผนภาพสถานะที่ใช้แผนภาพย่อยที่ได้ทำการอธิบายไว้แล้วเข้ามาอธิบายร่วมด้วย ซึ่งในข้อกำหนดรูปนัยคาเฟโอบีเจก็คือการนำเข้ามอดูล โดยมีรายละเอียดของกฎแต่ละข้อดังต่อไปนี้

กฎข้อที่ 1 แผนภาพสถานะ **S** ใดๆ จะถูกกำหนดเป็นมอดูลข้อกำหนดรูปนัยคาเฟโอบีเจ **S** ที่มีชนิดข้อมูลแฝง **s** ดังนี้

```
Module* S {
        *[ s ]*
}
```

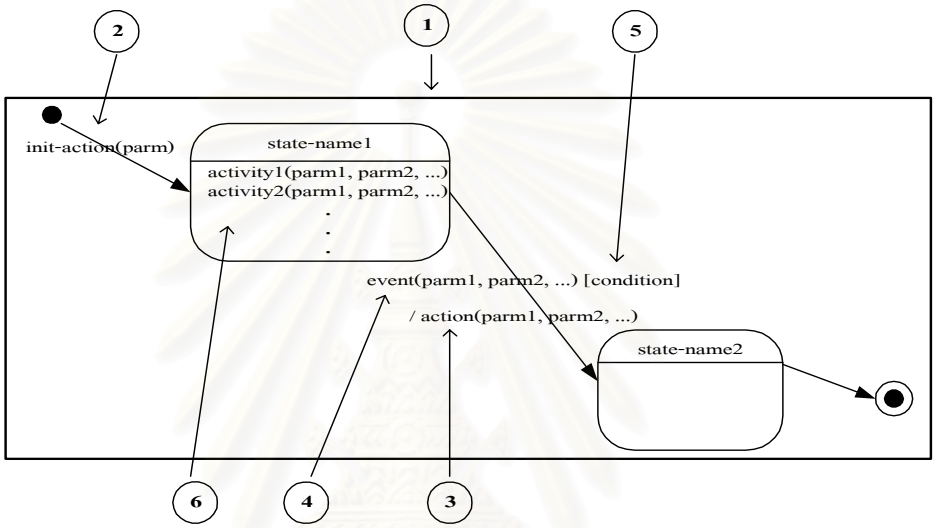
โดยที่

Module คือ คำสำคัญที่ใช้กำหนดจุดเริ่มต้นของมอดูล

S คือ ชื่อของมอดูล

s คือ ชื่อของชนิดข้อมูลแฝง

เนื่องจากแผนภาพสถานะ 1 แผนภาพนั้น มักได้รับการออกแบบให้จำลองพฤติกรรมของวัตถุเพียง 1 วัตถุเท่านั้น และข้อกำหนดรูปนัยคาเฟโอบีเจก็ใช้ 1 มอดูลสำหรับการเขียนข้อกำหนดของวัตถุใดๆ โดยมีชนิดข้อมูลที่เป็นชนิดข้อมูลแฝง 1 ชนิดข้อมูลเสมอ และใช้แผนภาพสถานะเพื่อการอธิบายพฤติกรรมของโมเดลแบบอิสระเท่านั้น



รูปที่ 3.1 แผนภาพสถานะ S ใดๆ

กฎข้อที่ 2 การแปลงการกระทำเริ่มต้นของแผนภาพสถานะเป็นการดำเนินการเริ่มต้นของภาษาคาเฟโอบีเจ ดังนี้

ในการกำหนดคุณสมบัติแบบเสถียรของวัตถุ สามารถแปลงได้ ดังนี้

op **p** : → **s**

ในการกำหนดคุณสมบัติแบบพลวัตของวัตถุ สามารถแปลงได้ดังนี้

op **p** : **id** → **s**

โดยที่

op คือ คำสำคัญที่ใช้ในการระบุการดำเนินการ

p คือ ชื่อการดำเนินการเริ่มต้น

id คือ ตัวแปรดัชนีที่ระบุว่าวัตถุตัวไหนที่กำลังสนใจ

s คือ ชื่อชนิดข้อมูลแฝง

เนื่องจากแต่ละแผนภาพสถานะจะมีการเปลี่ยนจากสถานะเริ่มต้น ไปเป็นสถานะกำเนิดวัตถุ (Initial State) โดยการเปลี่ยนสถานะนี้ จะมี 1 การกระทำ (Action) ซึ่งเป็นการดำเนินการที่ก่อให้เกิด Instant ของวัตถุ เรียกการดำเนินการนี้ว่า การดำเนินการเริ่มต้น (Initial Operation) ที่อาจจะมีหรือไม่มีอาร์กิวเมนต์ก็ได้ ขึ้นอยู่กับคุณลักษณะของระบบ โดยถ้าเป็นการกำหนดคุณสมบัติแบบเสถียร (Static Property : ระบบที่จำนวนของวัตถุที่ประกอบกันเป็นระบบมีจำนวนคงที่) ของวัตถุแล้ว การดำเนินการเริ่มต้นจะไม่มีอาร์กิวเมนต์ แต่หากเป็นการกำหนดคุณสมบัติแบบพลวัต (Dynamic Property : ระบบที่จำนวนของวัตถุที่ประกอบกันเป็นระบบมีจำนวนไม่คงที่) ของวัตถุแล้ว การดำเนินการเริ่มต้นจะมีตัวระบุ (Identifier) เป็นอาร์กิวเมนต์ เพื่อบอกว่าจะจัดเตรียมค่าเริ่มต้นให้กับวัตถุตัวไหน ส่วนชนิดข้อมูลของผลลัพธ์ของการดำเนินการเริ่มต้น ต้องเป็นชนิดข้อมูลของมอดูลปัจจุปัน

กฎข้อที่ 3 การแปลงการกระทำใดๆ ของแผนภาพสถานะ เป็นการดำเนินการเชิงพฤติกรรมของภาษาคาเฟโอบีเจได้ดังนี้

ในการกำหนดคุณสมบัติแบบเสถียรของวัตถุ สามารถแปลงได้ ดังนี้

bop **a** : **ls s** → **s**

ในการกำหนดคุณสมบัติแบบพลวัตของวัตถุ การดำเนินการอาจจะมีตัวระบุ เป็นอาร์กิวเมนต์ของการดำเนินการ ดังนั้นจะได้ว่า

bop **a** : **id ls s** → **s**

โดยที่

bop คือ คำสำคัญที่ใช้ในการระบุการดำเนินการเชิงพฤติกรรม

a คือ ชื่อการดำเนินการ

id คือ ตัวแปรดัชนีที่ระบุว่าวัตถุตัวไหนที่กำลังสนใจ

ls คือ รายการของอาร์กิวเมนต์ ที่เป็นชนิดข้อมูลใดๆ

s คือ ชนิดข้อมูลแฝงของมอดูลปัจจุบัน

การกระทำของแผนภาพสถานะที่เป็นส่วนประกอบของการเปลี่ยนสถานะ เป็นส่วนที่จะถูกประมวลผลก่อนเข้าสู่สถานะเป้าหมาย โดยเป็นการเปลี่ยนคุณสมบัติของวัตถุที่กำลังสนใจ ดังนั้นจึงสามารถแปลงเป็นการดำเนินการเชิงพฤติกรรมของภาษาคาเฟโอบีเจ โดยที่ การดำเนินการประเภทนี้ จะมีอาร์กิวเมนต์ 1 ตัว ที่เป็นชนิดข้อมูลแฝงของมอดูลปัจจุบัน และผลลัพธ์จะมีชนิดข้อมูลเป็นชนิดข้อมูลแฝงของมอดูลปัจจุบันเช่นเดียวกัน

กฎข้อที่ 4 การแปลงเหตุการณ์ใดๆ ของแผนภาพสถานะ เป็นการดำเนินการเชิงพฤติกรรมของภาษาคาเฟโอบีเจได้ ดังนี้

ในการกำหนดคุณสมบัติแบบเสถียรของวัตถุ สามารถแปลงได้ ดังนี้

bop **e** : **ls s** → **s**

ในการกำหนดคุณสมบัติแบบพลวัตของวัตถุ เหตุการณ์อาจจะมีตัวระบุเป็นอาร์กิวเมนต์ของการดำเนินการ ดังนั้นสามารถแปลงได้ ดังนี้

bop **e** : **id ls s** → **s**

โดยที่

bop คือ คำสำคัญที่ใช้ในการระบุการดำเนินการเชิงพฤติกรรม

e คือ ชื่อเหตุการณ์ หรือ การดำเนินการ

id คือ ตัวแปรดัชนีที่ระบุว่าวัตถุตัวไหนที่กำลังสนใจ

ls คือ รายการของอาร์กิวเมนต์ ที่เป็นชนิดข้อมูลใดๆ

s คือ ชนิดข้อมูลแฝงของมอดูลปัจจุบัน

เหตุการณ์ของแผนภาพสถานะที่เป็นส่วนประกอบของการเปลี่ยนสถานะ มีผลทำให้มีการเปลี่ยนสถานะจาก สถานะต้นทาง (Source State) ไปยังสถานะเป้าหมาย (Target State) โดยมีส่วนทำให้คุณสมบัติที่กำลังสนใจมีการเปลี่ยนแปลง ดังนั้นจึงสามารถแปลงเป็นการดำเนินการเชิงพฤติกรรมของภาษาคาเฟโอบีเจ โดยที่ การดำเนินการประเภทนี้ ต้องมีอาร์กิวเมนต์ 1 ตัวที่เป็นชนิดข้อมูลของมอดูลปัจจุบัน และชนิดข้อมูลของผลลัพธ์เป็นชนิดข้อมูลของมอดูลปัจจุบันเช่นเดียวกัน นั่นคือ เหตุการณ์เป็นการดำเนินการเชิงพฤติกรรมที่สามารถเกิดขึ้นได้ตลอดอายุของวัตถุ

กฎข้อที่ 5 การแปลงเงื่อนไขของแผนภาพสถานะ เป็นส่วนประกอบของสมการของสัจพจน์ของภาษาคาเฟโอบีเจได้ ดังนี้

สำหรับการเปลี่ยนสถานะที่ไม่มีเงื่อนไข สามารถแปลงได้ดังนี้

eq **t** = **t'**

สำหรับการเปลี่ยนสถานะที่มีเงื่อนไข ของแผนภาพสถานะ สามารถแปลงได้ดังนี้

ceq **t** = **t'** if **c**

โดยที่

eq คือ คำสำคัญที่ระบุสมการที่ไม่มีเงื่อนไข

ceq คือ คำสำคัญที่ระบุสมการที่มีเงื่อนไข

t, t' คือ พจน์ทางคณิตศาสตร์ ที่เป็นการรวมกันของ การดำเนินการ ตัวแปร และค่าคงที่

c คือ องค์ประกอบของเงื่อนไข

เงื่อนไขของแผนภาพสถานะที่เป็นส่วนหนึ่งของการเปลี่ยนสถานะ จะเป็นตัวที่คอยตรวจสอบว่าการดำเนินการของการเปลี่ยนสถานะจะถูกประมวลผลหรือไม่ โดยเงื่อนไขนั้น จะให้ผลลัพธ์ที่มีชนิดข้อมูลเป็นบูลีน คือให้ค่าจริง หรือเท็จ ดังนั้นการดำเนินการที่อยู่นอกสุดของพจน์ทางคณิตศาสตร์ (Term) ของการดำเนินการจะต้องเป็นการดำเนินการสังเกตค่า หรือค่าคงที่เท่านั้น เพราะฉะนั้น เงื่อนไขจะเป็นส่วนหนึ่งของสัจพจน์หรือสมการของการดำเนินการที่เป็นส่วนประกอบของการเปลี่ยนสถานะชุดเดียวกัน แต่อย่างไรก็ตาม การเปลี่ยนสถานะไม่จำเป็นต้องมีเงื่อนไขก็ได้เช่นกัน

กฎข้อที่ 6 การแปลงกิจกรรมใดๆ ของแผนภาพสถานะ เป็นการดำเนินการเชิงพฤติกรรมของภาษาคาเฟโอบีเจได้ดังนี้

ในการกำหนดคุณสมบัติแบบเสถียรของวัตถุ สามารถแปลงได้ ดังนี้

bop **A** : **ls s** → **s**

ในการกำหนดคุณสมบัติแบบพลวัตของวัตถุ การดำเนินการอาจจะมีตัวระบุเป็นอาร์กิวเมนต์ ดังนั้นสามารถแปลงได้ดังนี้

bop **A** : **id ls s** → **s**

โดยที่

bop คือ คำสำคัญที่ใช้ในการระบุการดำเนินการเชิงพฤติกรรม

A คือ ชื่อการดำเนินการ

id คือ ตัวแปรดัชนีที่ระบุว่าวัตถุตัวไหนที่กำลังสนใจ

ls คือ รายการของอาร์กิวเมนต์ ที่เป็นชนิดข้อมูลใดๆ

s คือ ชนิดข้อมูลแฝงของมอดูลปัจจุบัน

กิจกรรมของแผนภาพสถานะที่เป็นส่วนประกอบของสถานะ เป็นการดำเนินการที่ใช้ช่วงระยะเวลาเพื่อให้การประมวลผลสำเร็จ ซึ่งเป็นการประมวลผลที่ไม่มีผลต่อการเปลี่ยนสถานะ แต่มีผลทำให้ค่าของวัตถุ ณ สถานะนั้นเปลี่ยนแปลง ดังนั้นจึงสามารถแปลงเป็นการดำเนินการเชิงพฤติกรรมของภาษาคาเฟโอบีเจ โดยที่การดำเนินการประเภทนี้จะมีอาร์กิวเมนต์ 1 ตัว เป็นชนิดข้อมูลแฝงของมอดูลปัจจุบัน และผลลัพธ์มีชนิดข้อมูลเป็นชนิดข้อมูลแฝงของมอดูลปัจจุบันเช่นเดียวกัน

กฎข้อที่ 7 การแปลงส่วนต่อประสานของวัตถุเป็นการดำเนินการส่วนไม่ซ้ำของภาษาคาเฟโอบีเจได้ดังนี้

ในการกำหนดคุณสมบัติแบบเสถียรของวัตถุ สามารถแปลงได้ ดังนี้

bop **p** : **s** → **s'**

ในการกำหนดคุณสมบัติแบบพลวัตของวัตถุ การดำเนินการส่วนไม่ซ้ำ จะต้องมีตัวระบุเป็นอาร์กิวเมนต์ สามารถแปลงได้ดังนี้

bop **p** : **id s** → **s'**

โดยที่

bop คือ คำสำคัญที่ใช้ในการระบุการดำเนินการเชิงพฤติกรรม

p คือ ชื่อการดำเนินการส่วนไม่ซ้ำ

id คือ ตัวแปรดัชนีที่ระบุว่าวัตถุตัวไหนที่กำลังสนใจ

s คือ ชนิดข้อมูลแฝงของมอดูลปัจจุบัน

s' คือ ชนิดข้อมูลแฝงของมอดูลนำเข้า

การเขียนข้อกำหนดรูปนัยคาเฟโอบีเจนั้น ถ้าระบบมีขนาดใหญ่หรือมีความซับซ้อน ก็จะทำการแตกให้เป็นวัตถุเล็กๆ ที่ง่ายต่อการเขียน การตรวจสอบ การหาข้อผิดพลาด และความเข้าใจ ซึ่งเมื่อแต่ละวัตถุมีความถูกต้องสมบูรณ์แล้ว จึงนำมาประกอบเข้าด้วยกัน โดยในภาษาคาเฟโอบีเจนั้น การนำมอดูลมาประกอบเข้าด้วยกันสามารถทำได้โดย การนำเข้า (Importation) ซึ่งสามารถนำเข้าได้ทั้งโมเดลแบบยึดติด และโมเดลแบบอิสระ แต่ในการนำเข้าโมเดลแบบยึดติดนั้น ทุกๆ ส่วนย่อยของมอดูลที่นำเข้ามา สามารถที่จะเรียกใช้ได้โดยตรงเลย ซึ่งมีลักษณะเหมือนกับการกำหนดเป็นข้อมูลหรือการดำเนินการสาธารณะ (Public) ของแนวคิดเชิงวัตถุ แต่ในการนำเข้าของโมเดลแบบอิสระนั้น ทุกๆ ส่วยย่อยของมอดูลที่นำเข้ามา ไม่สามารถเรียกใช้ได้โดยตรง ซึ่งมีลักษณะคล้ายกับการกำหนดเป็นข้อมูลหรือการดำเนินการส่วนตัว (Private) ดังนั้นการที่จะเรียกใช้ส่วนย่อยของโมเดลแบบอิสระ จึงต้องมีการกำหนดส่วนต่อประสาน (Interface) ขึ้นมา 1 ตัวต่อ 1 โมเดลแบบอิสระ เพื่อใช้สำหรับการเข้าถึงข้อมูลหรือการดำเนินการของโมเดลแบบอิสระที่ถูกนำเข้ามา ดังนั้นจึงสามารถแปลงเป็นการดำเนินการส่วนไม่ซ้ำ (Projection Operation) ของภาษาคาเฟโอบีเจ โดยที่การดำเนินการนี้จะมีชนิดข้อมูลแฝงของมอดูลปัจจุบันเป็นอาร์กิวเมนต์ และผลลัพธ์ของการดำเนินการนี้จะมีชนิดข้อมูลเป็นชนิดข้อมูลแฝงของมอดูลที่นำเข้ามา

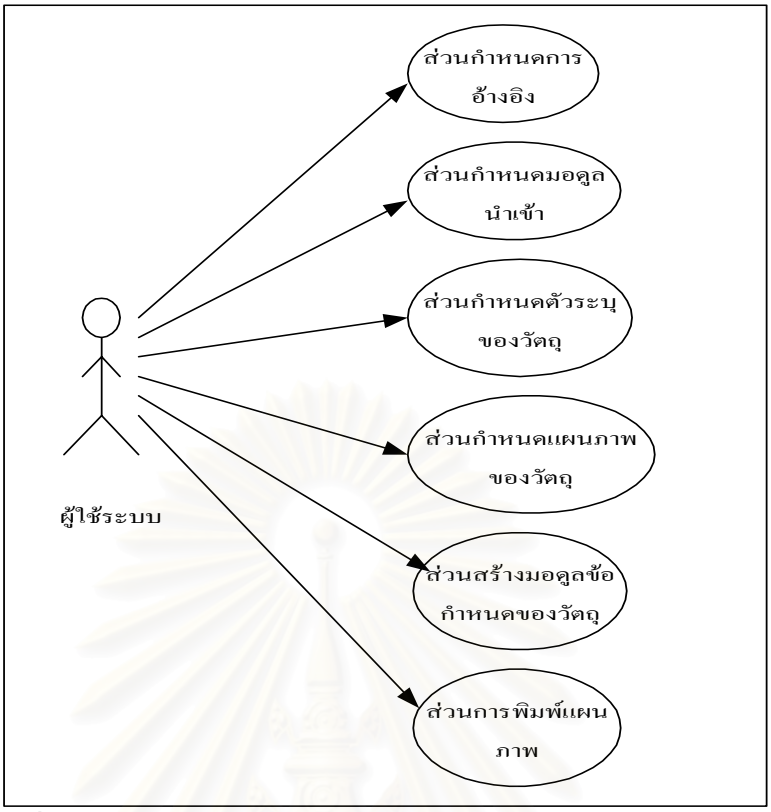
บทที่ 4
การออกแบบบรรณาธิกรณ์แผนภาพสถานะเพื่อสร้างข้อกำหนดรูปนัยคาเฟโอบีเจ

เมื่อมีการกำหนดกฎการแปลงจากแผนภาพสถานะเป็นข้อกำหนดรูปนัยคาเฟโอบีเจแล้ว ก็ทำการออกแบบบรรณาธิกรณ์แผนภาพสถานะที่ใช้ในการกำหนดส่วนประกอบต่างของแผนภาพ รวมทั้งสร้างข้อกำหนดรูปนัยจากแผนภาพ ซึ่งจะกล่าวถึงการออกแบบโดยละเอียดดังนี้

บรรณาธิกรณ์แผนภาพสถานะ สามารถแบ่งได้เป็น 6 ส่วน โดยแสดงด้วยแผนภาพกรณีการใช้ (Use case diagram) ดังรูปที่ 4.1 ที่ประกอบด้วยส่วนการทำงานหลักๆ ดังต่อไปนี้ คือ

- **ส่วนกำหนดการอ้างอิง** ใช้สำหรับการเตรียมมอดูลของข้อกำหนดรูปนัยคาเฟโอบีเจที่ต้องการนำมาอ้างอิง เพื่ออธิบายการทำงานของมอดูลที่กำลังจะเขียนข้อกำหนด
- **ส่วนกำหนดมอดูลนำเข้า** ใช้สำหรับกำหนดว่าจะมีมอดูลใดบ้างที่ได้จากขั้นตอนแรก ที่จะนำเข้ามาเพื่ออธิบายมอดูลที่กำลังจะเขียนข้อกำหนดจริงๆ โดยในขั้นนี้จะมีการกำหนดรายละเอียดต่างๆ ของข้อกำหนดที่นำเข้าด้วย เช่น การเปลี่ยนชื่อ ชนิดข้อมูล การดำเนินการ เป็นต้น
- **ส่วนกำหนดตัวระบุของวัตถุ** ใช้สำหรับการกำหนดพารามิเตอร์ ตัวแปร การดำเนินการสังเกตค่า การดำเนินการโปรเจคชัน และคุณสมบัติของการดำเนินการสังเกตค่าทั้งหมดที่จำเป็นต้องมี
- **ส่วนกำหนดแผนภาพของวัตถุ** ใช้สำหรับการวาดแผนภาพสถานะ เพื่อแสดงพฤติกรรมของมอดูล รวมทั้งกำหนดคุณสมบัติของแต่ละองค์ประกอบของแผนภาพสถานะ
- **ส่วนสร้างมอดูลข้อกำหนดของวัตถุ** ใช้สำหรับสร้างข้อกำหนดรูปนัยคาเฟโอบีเจจากแผนภาพสถานะ ซึ่งมีการสร้างเป็นแฟ้มของข้อกำหนดเพียง 1 แฟ้ม ที่รวมทุกๆ มอดูลที่ต้องใช้ เพื่อความสะดวกในการนำข้อกำหนดไปใช้ในการพิสูจน์โดยโปรแกรมแปลงภาษาคาเฟโอบีเจ (CafeOBJ Interpreter)
- **ส่วนการพิมพ์แผนภาพ** ใช้สำหรับการพิมพ์แผนภาพสถานะ

โดยแต่ละส่วนการทำงานมีรายละเอียดการออกแบบโปรแกรม ดังต่อไปนี้



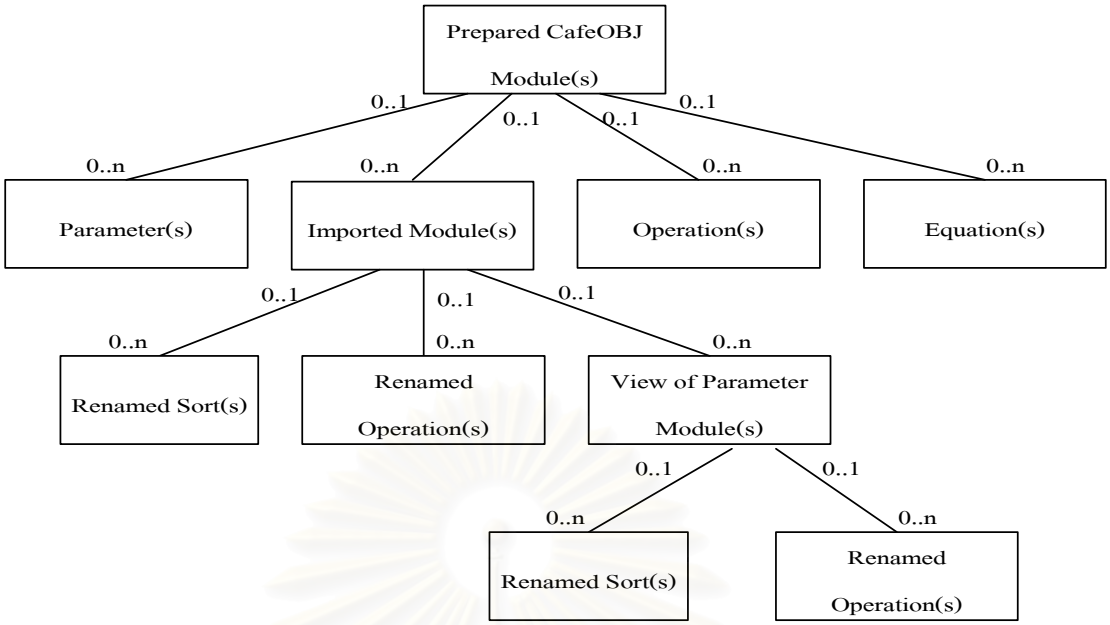
รูปที่ 4.1 แผนภาพกรณีการใช้ของระบบบรรณาธิกรณ์แผนภาพสถานะ

1 ส่วนกำหนดการอ้างอิง

ส่วนรับข้อมูลเป็นส่วนที่ใช้สำหรับการอ่านข้อกำหนดรูปนัยที่มีอยู่แล้วในรูปของแฟ้มข้อความ ซึ่งรูปแบบของแฟ้มข้อมูลจะอยู่ในรูปแบบวากยะสัมพันธ์ (Syntax) ของข้อกำหนดรูปนัยคาเฟโอบีเจ ดังแสดงในภาคผนวก ก ซึ่งเป็นแฟ้มข้อมูลที่สร้างมาจากแผนภาพสถานะก่อนหน้านี้ หรือเป็นข้อกำหนดที่มีอยู่ก่อนแล้ว โดยในส่วนนี้นั้นเป็นการอ่านข้อมูลมาเก็บไว้ แต่ไม่ใช่มอดูลที่นำเข้าจริงๆ ที่สามารถนำมาใช้ใหม่ได้ เป็นแต่เพียงการเตรียมมอดูลไว้ เพื่อนำเข้าจริงๆ ในขั้นของการกำหนดมอดูลนำเข้า ซึ่งแฟ้มข้อมูลนำเข้า มีตัวอย่างดังรูปที่ 4.2 และโครงสร้างข้อมูลสำหรับเก็บข้อกำหนดรูปนัยคาเฟโอบีเจแต่ละมอดูลเป็นดังรูปที่ 4.3

```
mod! ON-OFF
  {
     [ Value ]

     op on : -> Value
     op off : -> Value
  }

mod* SWITCH
  {
     protecting(ON-OFF)

     *[ Switch ]*

     op init-sw : -> Switch

     bop on : Switch ->
Switch
     bop off : Switch ->
Switch
     bop state : Switch ->
Value

     var S : Switch

     eq state(init-sw) = off .
     eq state(on( S)) = on .
```

รูปที่ 4.2 ตัวอย่างแฟ้มข้อมูลของข้อกำหนดรูปนัยคาเฟโอบีเจ



รูปที่ 4.3 โครงสร้างข้อมูลสำหรับเก็บข้อมูลของมอดูลอ้างอิง 1 มอดูล

รายละเอียดของข้อมูลที่ต้องจัดเก็บสำหรับมอดูลที่เตรียมนำเข้า 1 มอดูล ประกอบด้วย

1) ชื่อของมอดูล
2) ประเภทของมอดูล ซึ่งมี 2 ประเภทคือ โมเดลแบบยึดติด หรือ โมเดลแบบอิสระ
3) รายการของพารามิเตอร์ ซึ่งอาจจะมีหรือไม่มีก็ได้ โดย 1 รายการ ประกอบด้วย
   - ชื่อตัวแปรที่เป็นพารามิเตอร์
   - มอดูลของตัวแปรที่กำหนด
4) รายการของมอดูลนำเข้า ซึ่งเป็นข้อกำหนดการนำเข้ามอดูลที่นำเข้าโดยมอดูลที่เตรียมนำเข้า ซึ่งอาจจะมีหรือไม่มีก็ได้ โดย 1 มอดูลที่นำเข้า ประกอบด้วย

   4.1) ชื่อของโมดูลที่นำเข้า

   4.2) โหมดในการนำเข้า

   4.3) รายการของพารามิเตอร์ที่มีการเปลี่ยนแปลงข้อกำหนด ซึ่งอาจจะมีหรือไม่มีก็ได้ ถ้ามีการเก็บข้อมูลในส่วนนี้แสดงว่ามอดูลนำเข้าเป็นพารามิเตอร์มอดูล โดย 1 รายการประกอบด้วย
      - ชื่อของพารามิเตอร์
      - ชื่อของมอดูลที่ต้องการให้พารามิเตอร์มอดูลเป็น
      - รายการของชนิดข้อมูลของพารามิเตอร์มอดูลที่มีการเปลี่ยนชื่อ
      - รายการของการดำเนินการของพารามิเตอร์มอดูลที่มีการเปลี่ยนชื่อ

   4.4) รายการของชนิดข้อมูลของมอดูลนำเข้าที่มีการเปลี่ยนชื่อ

   โดยที่ รายการของชนิดข้อมูลและการดำเนินการที่มีการเปลี่ยนชื่อ 1 รายการ ประกอบด้วย

- ประเภทของข้อมูลที่มีการเปลี่ยนชื่อ ซึ่งอาจจะเป็น sort หรือ hsort สำหรับชนิดข้อมูลที่มีการเปลี่ยนชื่อ และ op หรือ bop สำหรับการดำเนินการที่มีการเปลี่ยนชื่อ
- ชื่อเดิมของข้อมูลที่มีการเปลี่ยนชื่อ
- ชื่อใหม่ของข้อมูลที่มีการเปลี่ยนชื่อ

5) รายการของชนิดข้อมูล

6) ประเภทของชนิดข้อมูล ซึ่งมี 2 ประเภทคือ ชนิดข้อมูลสังเกตค่า หรือ ชนิดข้อมูลแฝง

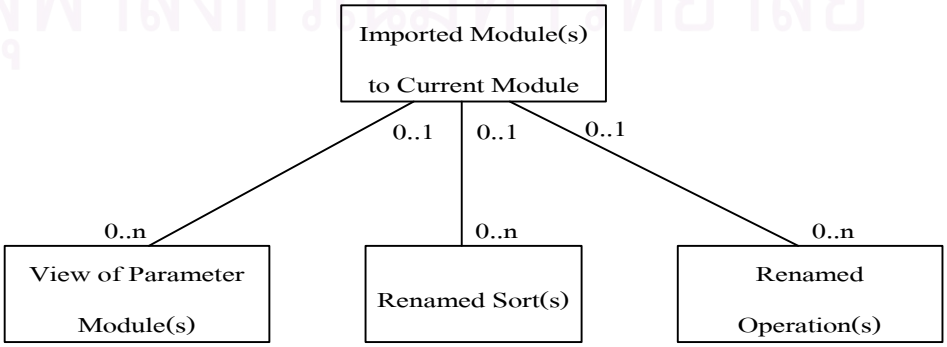
7) รายการของการดำเนินการ โดย 1 การดำเนินการ ประกอบด้วย
   - ชนิดของการดำเนินการ ซึ่งมี 2 ประเภทคือ op หรือ bop
   - ชื่อของการดำเนินการ
   - รายการของชนิดข้อมูลของอาร์กิวเมนต์ (หรือเรียกว่า Arity)
   - ชนิดข้อมูลของผลลัพธ์ (หรือเรียกว่า Coarity)
   - ลักษณะประจำของการดำเนินการ (Operation Attribute)

8) รายการของตัวแปร

9) รายการของสัจพจน์

2 ส่วนกำหนดมอดูลนำเข้า

ในส่วนนี้ เป็นการนำเข้ามอดูลที่ได้เตรียมไว้แล้วจากขั้นตอนส่วนรับข้อมูล เพื่อนำมาใช้ในการอธิบายคุณสมบัติของมอดูลปัจจุบัน โดยเป็นการกำหนดคุณสมบัติของมอดูลนำเข้าด้วยว่าจะมีลักษณะเป็นแบบใด ซึ่งเมื่อมีการนำเข้ามอดูลใดแล้ว ก็ทำการสร้างรายการของการดำเนินการที่สามารถนำมาเพื่อกำหนดสัจพจน์ของแต่ละการดำเนินการของมอดูลที่กำลังเขียนข้อกำหนดอยู่ เนื่องจากมอดูลที่นำเข้าอาจจะมีการนำเข้ามอดูลอื่นๆ อีก และ ทุกๆ การดำเนินการของการนำเข้าที่เป็นลำดับขั้นนั้น ก็สามารถนำมาใช้ได้เช่นเดียวกัน ดังนั้นในส่วนนี้จึงต้องเก็บรายการของการดำเนินการในทุกลำดับขั้นของการนำเข้าด้วย โดยโครงสร้างข้อมูลของแต่ละมอดูลที่นำเข้า เป็นได้ดังรูปที่ 4.4



รูปที่ 4.4 โครงสร้างข้อมูลสำหรับเก็บข้อมูลมอดูลนำเข้า 1 มอดูล

รายละเอียดของข้อมูลที่ต้องจัดเก็บสำหรับมอดูลที่นำเข้า 1 มอดูล ประกอบด้วย

1) ชื่อของมอดูลที่นำเข้า
2) รายการของพารามิเตอร์
3) รายการของชนิดข้อมูลที่มีการเปลี่ยนชื่อ
4) รายการของการดำเนินการที่มีการเปลี่ยนชื่อ

รายการของชนิดข้อมูล และการดำเนินการที่มีการเปลี่ยนชื่อ 1 รายการประกอบด้วย

- ชื่อของพารามิเตอร์มอดูลของชนิดข้อมูล หรือการดำเนินการที่มีการเปลี่ยนชื่อ
- ชื่อของมอดูลที่มีชนิดข้อมูล หรือการดำเนินการ ที่พารามิเตอร์มอดูลสามารถเปลี่ยนชื่อชื่อชนิดข้อมูล หรือการดำเนินการไปเป็นของมอดูลนี้ได้
- ประเภทของข้อมูลที่มีการเปลี่ยนชื่อ ซึ่งอาจจะเป็น sort หรือ hsort สำหรับชนิดข้อมูลที่มีการเปลี่ยนชื่อ และ op หรือ bop สำหรับการดำเนินการที่มีการเปลี่ยนชื่อ
- ชื่อเดิมของข้อมูลที่มีการเปลี่ยนชื่อ
- ชื่อใหม่ของข้อมูลที่มีการเปลี่ยนชื่อ

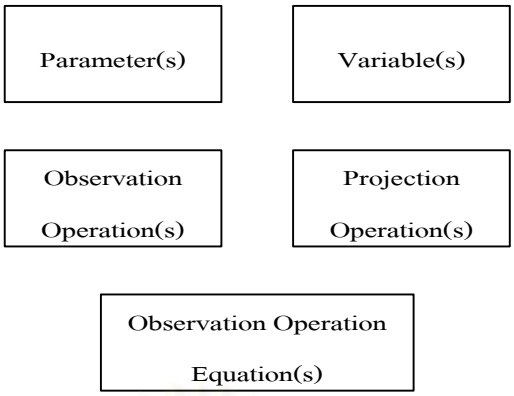
3 ส่วนกำหนดตัวระบุของวัตถุ

เมื่อทำการนำเข้ามอดูลเรียบร้อยแล้ว ถ้าหากว่ามอดูลที่นำเข้าเป็นโมเดลแบบอิสระแล้วนั้น อาจจะเป็นไปได้ที่ต้องมีการกำหนดการดำเนินการโปรเจคชัน ซึ่งเป็นการดำเนินการที่เป็นตัวเชื่อมต่อการเข้าถึงลักษณะบางประการของมอดูลที่นำเข้า เช่น การดำเนินการ ค่าคงที่ เป็นต้น นอกจากนี้ในส่วนนี้ยังเป็นส่วนที่ใช้สำหรับกำหนด พารามิเตอร์ ตัวแปร การดำเนินการสังเกตค่า รวมทั้งคุณสมบัติทางสัจพจน์ (Axiom) ของการดำเนินการสังเกตค่าด้วย เนื่องจาก ลักษณะดังกล่าวข้างต้นไม่มีผลต่อการเปลี่ยนแปลงพฤติกรรมของระบบ จึงไม่สามารถกำหนดให้เป็นส่วนหนึ่งของแผนภาพสถานะได้

โดยมอดูลที่เป็นโมเดลแบบอิสระ สามารถสังเกตได้จากสัญลักษณ์ที่เป็นเครื่องหมายดอกจันท์ (*) หลังคำสงวน “mod” หรือ “module” จากรูปที่ 4.2 มอดูลสวิตซ์ (SWITCH) คือมอดูลแบบอิสระ ดังนั้น หากมีการนำเข้ามอดูลสวิตซ์แล้ว จะต้องมีการกำหนดการดำเนินการโปรเจคชันเพื่อเข้าถึงการดำเนินการ on off และ state และเข้าถึงค่าคงที่ init-sw โดยโครงสร้างข้อมูลของตัวระบุของวัตถุ แสดงได้ดังรูปที่ 4.5 ซึ่งมีรายละเอียดดังนี้

ตัวระบุของวัตถุที่ต้องจัดเก็บสำหรับรายละเอียดทั่วไป ประกอบด้วย

1) รายการของตัวแปรที่เป็นพารามิเตอร์
2) รายการของตัวแปร โดย 1 ตัวแปร ประกอบด้วย



รูปที่ 4.5 โครงสร้างข้อมูลสำหรับเก็บตัวระบุของวัตถุของมอดูลปัจจุบัน

- ชื่อตัวแปร
- ชนิดข้อมูลของตัวแปร

3) รายการของการดำเนินการสังเกตค่า ซึ่งมีลักษณะข้อมูลที่เก็บเช่นเดียวกับการดำเนินของส่วนรับข้อมูลเข้า

4) รายการของสมการของสัจพจน์ของการดำเนินการสังเกตค่า

5) รายการของการดำเนินการโปรเจคชัน

4 ส่วนกำหนดแผนภาพของวัตถุ

หลังจากทำการกำหนดมอดูลนำเข้า และรายละเอียดของมอดูลเรียบร้อยแล้ว ก็จะเป็นการสร้างแผนภาพสถานะเพื่อใช้สำหรับอธิบายพฤติกรรมของมอดูล โดยแผนภาพสถานะประกอบด้วยคอมโพเนนต์ต่างๆ ดังนี้

- สถานะเริ่มต้น
- การเปลี่ยนสถานะ
- การเปลี่ยนสถานะเข้าสู่สถานะเดิม
- สถานะ
- สถานะสิ้นสุด

สัญลักษณ์ของแต่ละคอมโพเนนต์นั่น จะใช้แบบเดียวกับสัญลักษณ์ของแผนภาพสถานะของยูเอ็มแอล โดยมีรายละเอียดของแต่ละดอมโพเนนต์ดังนี้

1) สถานะเริ่มต้น (Start State) แสดงถึงจุดเริ่มต้นของพฤติกรรมของมอดูล และไม่มีส่วนประกอบย่อยอื่นอีกในสถานะเริ่มต้น

2) การเปลี่ยนสถานะ (Transition) โดยเป็นส่วนที่บอกถึงการเปลี่ยนแปลงของสถานะ ว่ามีการเปลี่ยนแปลงจากสถานะใด ไปยังสถานะใด และข้อกำหนดใดที่ทำให้เกิดการเปลี่ยนสถานะ โดยมีส่วนประกอบย่อย 3 ส่วนคือ

2.1) เหตุการณ์ (Event) ที่ทำให้เกิดการเปลี่ยนสถานะ

2.2) เงื่อนไข (Condition) ที่ทำให้เกิดการเปลี่ยนสถานะ

2.3) การกระทำ (Action) ที่กระทำเพื่อเข้าสู่สถานะเป้าหมาย

โดยการเปลี่ยนสถานะจะเกิดขึ้นก็ต่อเมื่อ มีเหตุการณ์ที่ระบุเกิดขึ้น และเมื่อเหตุการณ์เกิดขึ้นแล้ว ก็ทำการตรวจสอบเงื่อนไขดูว่า เงื่อนไขที่ระบุเป็นจริงหรือไม่ ถ้าเงื่อนไขเป็นจริง การกระทำก็จะกระทำการ เพื่อเข้าสู่สถานะเป้าหมาย แต่ถ้าหากว่า เหตุการณ์ ที่ระบุไม่เกิด หรือ เงื่อนไขที่ระบุไม่เป็นจริงแล้ว การกระทำก็จะไม่เกิดขึ้น และไม่มีการเปลี่ยนสถานะเกิดขึ้นเช่นเดียวกัน

แต่อย่างไรก็ตาม การเปลี่ยนสถานะไม่ต้องมีส่วนประกอบย่อยก็สามารถเกิดขึ้นได้เช่นเดียวกัน

3) การเปลี่ยนสถานะเข้าสู่สถานะเดิม (Self Transition) มีส่วนประกอบเช่นเดียวกับ การเปลี่ยนสถานะ แต่เป็นการวนกลับเข้าสู่สถานะเดิม ซึ่งอาจจะเป็นเพราะเงื่อนไขไม่ถูกต้องหรือพฤติกรรมของมอดูลไม่มีการเปลี่ยนแปลง ถึงแม้การกระทำจะกระทำการก็ตาม เป็นต้น

4) สถานะ (State) โดยเป็นตัวบอกให้ทราบว่า ณ สถานะใดๆ พฤติกรรมของมอดูลจะมีลักษณะเหมือนกัน โดยมีส่วนประกอบย่อย คือ

4.1) กิจกรรม (Activity) ซึ่งเป็นการดำเนินการที่ไม่มีผลต่อการเปลี่ยนแปลงพฤติกรรมของมอดูล

เพื่อให้สามารถกำหนดพฤติกรรมของมอดูลในรูปของข้อกำหนดรูปนัยคาเฟโอบีเจ ได้อย่างครบถ้วน จึงได้ทำการเพิ่มลักษณะบางส่วนของ แผนภาพสถานะของยูเอ็มแอล เข้ามายังส่วนนี้ คือ

4.2) สัจพจน์ (Axiom) เป็นส่วนที่ใช้ในการกำหนดความสัมพันธ์ ที่เป็นตัวกำหนดคุณสมบัติของ เหตุการณ์ การกระทำ และ กิจกรรม โดยเขียนอยู่ในรูปของสมการทางคณิตศาสตร์

5) สถานะสิ้นสุด (End State) เป็นจุดที่บอกถึงจุดสิ้นสุดของมอดูล และไม่มีส่วนประกอบย่อยอื่นอีกในสถานะสิ้นสุด เช่นเดียวกับสถานะเริ่มต้น

ดังนั้นข้อมูลของส่วนนี้ ที่ต้องเก็บมีดังนี้

- รายการของกิจกรรม
- รายการของสมการของสัจพจน์
- การกระทำ
- เงื่อนไข
- เหตุการณ์

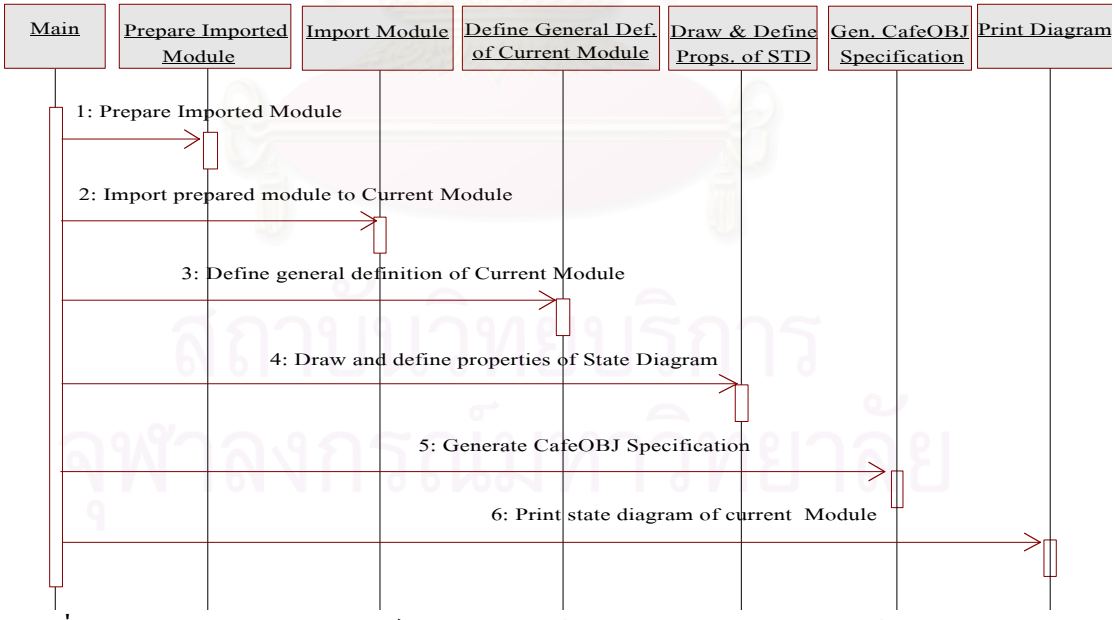
5 ส่วนสร้างมอดูลข้อกำหนดของวัตถุ

ในส่วนนี้ เป็นการสร้างข้อกำหนดรูปนัยคาเฟโอบีเจจากแผนภาพสถานะ ที่ได้ทำการวาดไว้แล้ว โดยระบบจะบันทึกมอดูลที่สร้างขึ้นลงแฟ้มข้อความ โดยชื่อแฟ้มข้อมูลจะมีชื่อเดียวกับชื่อแฟ้มแผนภาพสถานะ แต่จะมีนามสกุล “mod”

6 ส่วนการพิมพ์แผนภาพ

ในส่วนนี้ ใช้สำหรับการพิมพ์แผนภาพสถานะที่ได้กำหนดไว้แล้วออกทางเครื่องพิมพ์ เพื่อใช้เป็นเอกสารประกอบการออกแบบระบบ

การสร้างข้อกำหนดรูปนัยคาเฟโอบีเจโดยใช้แผนภาพสถานะนั้น เพื่อให้การกำหนดส่วนต่างๆ เป็นไปอย่างถูกต้อง สามารถแสดงลำดับขั้นตอนของการกำหนดส่วนต่างๆด้วยแผนภาพลำดับเหตุการณ์ดังรูปที่ 4.6 นั่นคือ ทำการเตรียมมอดูลที่จะนำเข้าก่อนเป็นอันดับแรก ตามด้วยการกำหนดการนำเข้ามอดูล และการกำหนดรายละเอียดทั่วไปของมอดูลที่กำลังเขียนข้อกำหนด หลังจากนั้นก็ทำการวาดแผนภาพสถานะเพื่อแสดงพฤติกรรมของมอดูลที่สนใจพร้อมกับการกำหนดคุณสมบัติต่างๆ ของแต่ละคอมโพเนนต์ เมื่อวาดแผนภาพสถานะสมบูรณ์แล้วจึงทำการสร้างข้อกำหนดรูปนัยคาเฟโอบีเจของมอดูล หรือทำการบันทึกลงแฟ้ม



รูปที่ 4.6 แผนภาพลำดับเหตุการณ์ของการเขียนข้อกำหนดรูปนัยคาเฟโอบีเจด้วยแผนภาพสถานะ

จากการออกแบบข้างต้น สามารถนำไปพัฒนาเครื่องมือซอฟต์แวร์โดยใช้โปรแกรมภาษา Microsoft Visual C++ ซึ่งแสดงรายละเอียดการพัฒนาในภาคผนวก ข

ตัวอย่างการสร้างข้อกำหนดรูปนัยคาเฟโอบีเจจากแผนภาพสถานะ จากตัวอย่างนี้เป็นการเขียนข้อกำหนดของมอดูลสวิตซ์ ซึ่งแผนภาพสถานะจะเป็นดังรูปที่ 4.7 และสามารถสร้างเป็นข้อกำหนดรูปนัยคาเฟโอบีเจซึ่งเป็นแฟ้มข้อความได้ดังรูปที่ 4.2

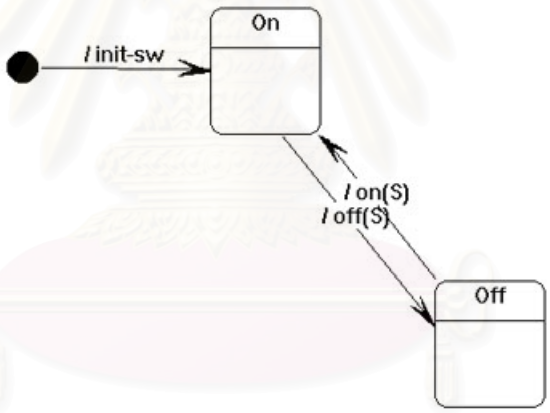
สภาพแวดล้อมที่ใช้ในการพัฒนาเครื่องมือซอฟต์แวร์

เครื่องคอมพิวเตอร์ที่ใช้ในการพัฒนา มีรายละเอียดดังนี้

- คอมพิวเตอร์พีซี Pentium II 350 MHz.
- หน่วยความจำหลัก 256 MB
- ฮาร์ดดิสก์ความจุ 10 GB

ซอฟต์แวร์ที่ใช้สำหรับการพัฒนา มีรายละเอียดดังนี้

- ระบบปฏิบัติการวินโดว์ 98/2000
- Microsoft Visual C++ version 6 Service Pack 5
- CJLibrary version 6.09



รูปที่ 4.7 แผนภาพสถานะของมอดูลสวิตซ์

บทที่ 5
การทดสอบโปรแกรม

ในบทนี้ กล่าวถึงขั้นตอนการติดตั้งโปรแกรม สภาวะที่ใช้ในการทดสอบโปรแกรม และการทดสอบเครื่องมือบรรณาธิกรณ์แผนภาพสถานะสำหรับสร้างข้อกำหนดรูปนัยคาเฟโอบีเจภายใต้ระบบปฏิบัติการวินโดว์ 98/2000

1 ขั้นตอนการติดตั้ง STD2CafeOBJ

ทำการติดตั้งเครื่องมือ โดยการเรียก SETUP.EXE แล้วปฏิบัติตามขั้นตอนที่โปรแกรมแนะนำ จนกว่าการติดตั้งเครื่องมือจะเสร็จเรียบร้อย

2 สภาวะที่ใช้ทดสอบโปรแกรม

เครื่องคอมพิวเตอร์ที่ใช้ในการทดสอบ มีรายละเอียดดังนี้

- คอมพิวเตอร์พีซี Pentium II 350 MHz.
- หน่วยความจำหลัก 256 MB
- ฮาร์ดดิสก์ความจุ 10 GB

3 ระบบที่ใช้ทดสอบโปรแกรม

ขั้นตอนการทดสอบเครื่องมือที่สร้างขึ้นนั้น มีลำดับดังนี้

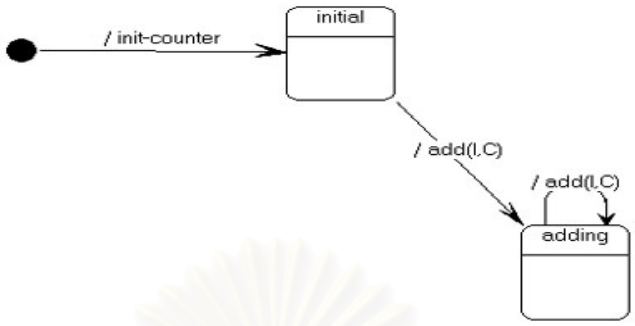
- กำหนดระบบที่ใช้ทดสอบ ซึ่งมี 3 ระบบ คือ ระบบการนับเลข ระบบการนับเลขที่มีสวิตซ์ และระบบลิฟต์
- สร้างแผนภาพสถานะของแต่ละระบบที่ใช้ทดสอบ
- สร้างข้อกำหนดรูปนัยคาเฟโอบีเจของแต่ละระบบที่ใช้ทดสอบจากแผนภาพสถานะ
- ใช้ตัวแปลงภาษาคาเฟโอบีเจสำหรับตรวจสอบวากยะสัมพันธ์ของข้อกำหนดที่ได้
- ใช้ตัวแปลงภาษาคาเฟโอบีเจสำหรับตรวจสอบความถูกต้องของข้อกำหนดที่ได้

โดยมีรายละเอียดในการทดสอบทั้ง 3 ระบบ ดังนี้

3.1 ระบบการนับเลข (Counter)

ระบบการนับเลข เป็นระบบสำหรับการเพิ่ม หรือ การลดเลขจำนวนเต็ม โดยการบวก หรือ การลบ ทั้งนี้ขึ้นอยู่กับค่าที่ผู้ใช้เป็นผู้ใส่ให้กับระบบ โดยมีการนำเข้ามอดูลมาตรฐาน ‘INT’ หรือเลข

จำนวนเต็มซึ่งเป็นชนิดข้อมูลสังเกตค่า เพื่ออธิบายพฤติกรรมของระบบการนับเลขด้วย ซึ่งเมื่อสร้างเป็นแผนภาพสถานะจะได้ดังรูปที่ 5.1



รูปที่ 5.1 แผนภาพสถานะของมอดูลตัวนับ

จากแผนภาพสถานะ ดังรูปที่ 5.1 สามารถสร้างเป็นข้อกำหนดรูปนัยคาเฟโอบีเจ ดังรูปที่ 5.2

```
mod* COUNTER{
      protecting(INT)

      *[ Counter ]*

      op init-counter : -> Counter
      bop add : Int Counter -> Counter

      -- Observation operation.
      bop read : Counter -> Int

      var I : Int
      var C : Counter

      eq read(init-counter) = 0 .
      eq read(add(I,C)) = read(C) + I .
}
```

รูปที่ 5.2 ข้อกำหนดของมอดูลตัวนับ

การตรวจสอบผลลัพธ์ของมอดูลตัวนับ

เมื่อนำข้อกำหนดของมอดูลตัวนับ ไปทำการตรวจสอบกับตัวแปลงภาษาคาเฟโอบีเจ ซึ่งจะทำการตรวจสอบ 2 ขั้นตอนคือ

ก) ขั้นตอนการตรวจสอบความถูกต้องตามวากยะสัมพันธ์ของภาษาคาเฟโอบีเจ จะได้ผลดังรูปที่ 5.3 ซึ่งแสดงว่าข้อกำหนดของมอดูลตัวนับที่สร้างจากแผนภาพสถานะ ถูกต้องตามวากยะสัมพันธ์ของภาษาคาเฟโอบีเจ

ข) ขั้นตอนการตรวจสอบความถูกต้องของข้อกำหนด โดยแยกทดสอบตามกรณีการทดสอบที่จะเป็นไปได้ทั้งหมด ดังนี้

- กรณีที่ 1 ทำการใส่ค่าบวกให้กับตัวนับ ซึ่งผลลัพธ์ที่ได้ต้องเป็นเลขจำนวนเต็มบวก ซึ่งทดสอบโดยใส่ค่าบวก 5 ให้กับตัวนับ และผลที่ได้คือ 5

```
CafeOBJ> in counter.mod
processing input : d:\user\g41mrt\cafeobj\test\.\counter.mod
-- defining module* COUNTER....._..*
** system already proved =*= is a congruence of COUNTER done.
COUNTER>
```

รูปที่ 5.3 ผลการตรวจสอบวากยะสัมพันธ์ของมอดูลตัวนับ

- กรณีที่ 2 ทำการใส่ค่าลบให้กับตัวนับ ซึ่งผลลัพธ์ที่ได้ต้องเป็นเลขจำนวนเต็มลบ ซึ่งทดสอบโดยใส่ค่าลบ 5 ให้กับตัวนับ และผลที่ได้คือ -5
- กรณีที่ 3 ทำการใส่ค่า 2 ค่าให้กับตัวนับ ซึ่งผลลัพธ์ที่ได้ต้องเป็นเลขจำนวนเต็มที่ได้จากการบวกของเลข 2 จำนวนที่ใส่เข้าไป

  ซึ่งทดสอบโดยใส่ค่าลบ 7 ตามด้วยค่าบวก 10 ให้กับตัวนับ และผลที่ได้คือ 3

ผลการตรวจสอบเป็นดังรูปที่ 5.4

```
COUNTER> red read(add(5,init-counter)) .                          <-- 1
-- reduce in COUNTER : read(add(5,init-counter))
5 : NzNat
(0.000 sec for parse, 3 rewrites(0.000 sec), 6 matches)
COUNTER> red read(add(-5,init-counter)) .                         <-- 2
-- reduce in COUNTER : read(add(-5,init-counter))
-5 : NzInt
(0.000 sec for parse, 3 rewrites(0.000 sec), 7 matches)
COUNTER> red read(add(10,add(-7,init-counter))) .                 <-- 3
-- reduce in COUNTER : read(add(10,add(-7,init-counter)))
3 : NzNat
(0.000 sec for parse, 5 rewrites(0.000 sec), 13 matches)
COUNTER>
```

รูปที่ 5.4 ผลการตรวจสอบความถูกต้องของข้อกำหนดของมอดูลตัวนับ

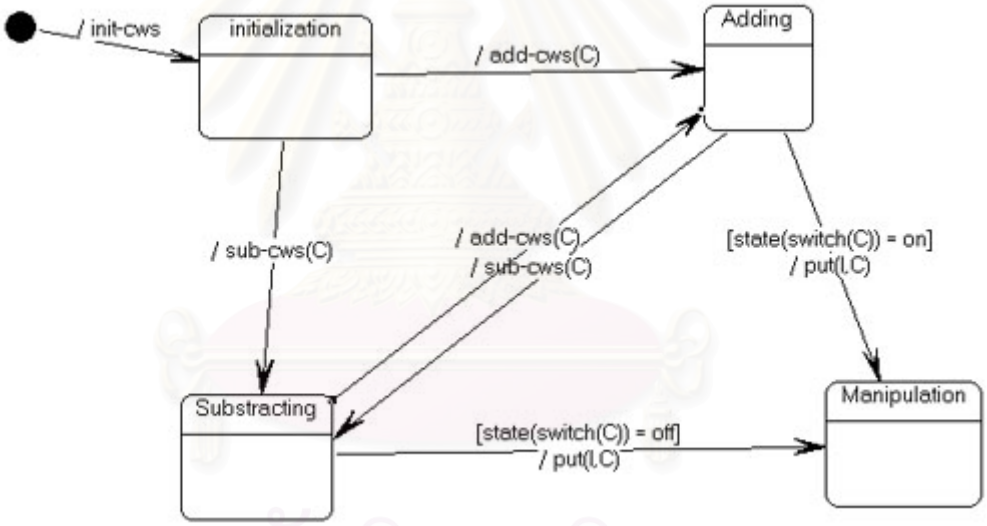
3.2 ระบบการนับเลขที่มีสวิตซ์ (Counter with Switch)

ระบบการนับเลขที่มีสวิตซ์ เป็นระบบที่ใช้สำหรับการบวก และ การลบ เลขจำนวนเต็ม เช่นเดียวกับมอดูลตัวนับ แต่จะเป็นการบวก หรือ การลบ นั้น ตัวสวิตซ์จะเป็นตัวควบคุม โดยเมื่อมีการกดปุ่ม on ก็จะเป็นการบวก แต่ถ้ากดปุ่ม off ก็จะเป็นการลบ โดยมีมอดูลนำเข้า 2 มอดูล คือ มอดูลตัวนับ ดังรูปที่ 5.2 และมอดูลสวิตซ์ ดังรูปที่ 5.5

```
mod! ON-OFF
  {
     [ Value ]

     op on : -> Value
     op off : -> Value
  }

  mod* SWITCH
  {
     protecting(ON-OFF)

     *[ Switch ]*

     op init-sw : -> Switch
     bop on : Switch -> Switch
     bop off : Switch -> Switch
     bop state : Switch -> Value

     var S : Switch

     eq state(init-sw) = off .
     eq state(on (S)) = on .
     eq state(off (S)) = off .

  }
```

รูปที่ 5.5 ข้อกำหนดของมอดูลสวิตซ์

เมื่อทำการนำเข้ามอดูลทั้ง 2 เรียบร้อยแล้ว ทุกชนิดข้อมูล และการดำเนินการที่มอดูลตัวนับและมอดูลสวิตซ์รู้จัก มอดูลตัวนับที่มีสวิตซ์ก็จะรู้จักด้วย และเนื่องจากทั้งมอดูลตัวนับและมอดูลสวิตซ์ เป็นโมเดลแบบอิสระ จึงสามารถกำหนดการดำเนินการโปรเจคชันได้ 2 การดำเนินการ และสร้างเป็นแผนภาพสถานะของมอดูลตัวนับที่มีสวิตซ์ได้ดังรูปที่ 5.6



รูปที่ 5.6 แผนภาพสถานะของมอดูลตัวนับที่มีสวิตซ์

จากแผนภาพสถานะ ดังรูปที่ 5.6 สามารถสร้างเป็นข้อกำหนดรูปนัยคาเฟโอบีเจ ดังรูปที่ 5.7

การตรวจสอบผลลัพธ์ของมอดูลตัวนับที่มีสวิตซ์

เมื่อนำข้อกำหนดของมอดูลตัวนับที่มีสวิตซ์ไปทำการตรวจสอบกับตัวแปลภาษาคาเฟโอบีเจ ซึ่งจะทำการตรวจสอบ 2 ขั้นตอนคือ

```
mod! ON-OFF
  {
     [ Value ]

     op on : -> Value
     op off : -> Value
  }

  mod* SWITCH
  {
     protecting(ON-OFF)

     *[ Switch ]*

     op init-sw : -> Switch
     bop on : Switch -> Switch
     bop off : Switch -> Switch
     bop state : Switch -> Value

     var S : Switch

     eq state(init-sw) = off .
     eq state(on (S)) = on .
     eq state(off (S)) = off .
  }

mod* COUNTER{
     protecting(INT)

     *[ Counter ]*

     op init-counter : -> Counter
     bop add : Int Counter -> Counter
     bop read : Counter -> Int

     var I : Int
     var C : Counter

     eq read(init-counter) = 0 .
     eq read(add(I,C)) = read(C) + I .
}
```

```
mod* CWS{
     protecting(SWITCH)
     protecting(COUNTER)

     *[ Cws ]*

     op init-cws : -> Cws
     bop sub-cws : Cws -> Cws
     bop put : Int Cws -> Cws
     bop add-cws : Cws -> Cws

     -- Observation operation.
     bop get-counter : Cws -> Int
     bop get-switch : Cws -> Value

     -- Projection operation.
     bop switch : Cws -> Switch
     bop counter : Cws -> Counter

     var I : Int
     var C : Cws

     eq get-counter(C) = read(counter(C)) .
     eq get-switch(C) = state(switch(C)) .

     eq switch(add-cws(C)) = on(switch(C)) .
     eq counter(add-cws(C)) = counter(C) .
     eq switch(init-cws) = init-sw .
     eq counter(init-cws) = init-counter .
     eq switch(sub-cws(C)) = off(switch(C)) .
     eq counter(sub-cws(C)) = counter(C) .
     eq switch(put(I,C)) = switch(C) .
     ceq counter(put(I,C)) = add(I,counter(C))
          if state(switch(C)) == on .
     ceq counter(put(I,C)) = add(-(I),counter(C))
          if state(switch(C)) == off .
}
```

รูปที่ 5.7 ข้อกำหนดรูปนัยคาเฟโอบีเจของมอดูลตัวนับที่มีสวิตซ์

ก) ขั้นตอนการตรวจสอบความถูกต้องตามวากยะสัมพันธ์ของภาษาคาเฟโอบีเจ ได้ผลดังรูปที่ 5.8 ซึ่งแสดงว่าข้อกำหนดของมอดูลตัวนับที่มีสวิตซ์ที่สร้างจากแผนภาพสถานะ ถูกต้องตามวากยะสัมพันธ์ของภาษาคาเฟโอบีเจ

```
CafeOBJ> in cws.mod
processing input : d:\user\g41mrt\cafeobj\test\.\cws.mod
-- defining module! ON-OFF..._* done.
-- defining module* SWITCH....._...*
** system already proved =*= is a congruence of SWITCH done.
-- defining module* COUNTER....._..*
** system already proved =*= is a congruence of COUNTER done.
-- defining module* CWS....._.....*
** system failed to prove =*= is a congruence of CWS done.
CWS>
```

รูปที่ 5.8 ผลการตรวจสอบวากยะสัมพันธ์ของมอดูลตัวนับที่มีสวิตซ์

ข) ขั้นตอนการตรวจสอบความถูกต้องของข้อกำหนด โดยหากมีการกดปุ่ม ‘add-cws’ จะเป็นการเพิ่มค่า แต่ถ้าหากกดปุ่ม ‘sub-cws’ จะเป็นการลดค่าของตัวนับที่มีสวิตซ์ตามจำนวนที่ใส่เข้าไป โดยแยกทดสอบตามกรณีการทดสอบที่จะเป็นไปได้ทั้งหมด ดังนี้

- กรณีที่ 1 ตรวจสอบสถานะของตัวนับที่มีสวิตซ์ เมื่อต้องการบวกเลขจำนวนเต็ม ซึ่งสถานะต้องเป็น ‘on’
  ซึ่งทดสอบโดยการกดปุ่ม add-cws และผลที่ได้คือ สถานะของสวิตซ์เป็น ‘on’
- กรณีที่ 2 ตรวจสอบสถานะของตัวนับที่มีสวิตซ์ เมื่อต้องการลบเลขจำนวนเต็ม ซึ่งสถานะต้องเป็น ‘off’
  ซึ่งทดสอบโดยการกดปุ่ม add-cws และผลที่ได้คือ สถานะของสวิตซ์เป็น ‘off’
- กรณีที่ 3 ตรวจสอบค่าของตัวนับ เมื่อมีการกดปุ่ม ‘add-cws’ และใส่ค่าบวก ซึ่งผลลัพธ์ที่ได้ต้องเป็นเลขจำนวนเต็มบวก
  ซึ่งทดสอบโดยการกดปุ่ม ‘add-cws’ ตามด้วยใส่ค่าบวก 5 ผลที่คือ 5
- กรณีที่ 4 ตรวจสอบค่าของตัวนับ เมื่อมีการกดปุ่ม ‘sub-cws’ และใส่ค่าบวก ซึ่งผลลัพธ์ที่ได้ต้องเป็นเลขจำนวนเต็มลบ
  ซึ่งทดสอบโดยการกดปุ่ม ‘sub-cws’ ตามด้วยใส่ค่าบวก 5 ผลที่คือ -5

ผลการทดสอบเป็นดังรูปที่ 5.9

```
CWS> red get-switch(add-cws(init-cws)) .                    <-- 1
-- reduce in CWS : get-switch(add-cws(init-cws))
on : Value
(0.000 sec for parse, 4 rewrites(0.000 sec), 6 matches)
CWS> red get-switch(sub-cws(init-cws)) .                    <-- 2
-- reduce in CWS : get-switch(sub-cws(init-cws))
off : Value
(0.000 sec for parse, 4 rewrites(0.000 sec), 9 matches)
CWS> red get-counter(put(5,add-cws(init-cws))) .            <-- 3
-- reduce in CWS : get-counter(put(5,add-cws(init-cws)))
5 : NzNat
(0.000 sec for parse, 11 rewrites(0.000 sec), 20 matches)
CWS> red get-counter(put(5,sub-cws(init-cws))) .            <-- 4
-- reduce in CWS : get-counter(put(5,sub-cws(init-cws)))
-5 : NzInt
(0.000 sec for parse, 16 rewrites(0.050 sec), 38 matches)
```

รูปที่ 5.9 ผลการตรวจสอบความถูกต้องของมอดูลตัวนับที่มีสวิตซ์

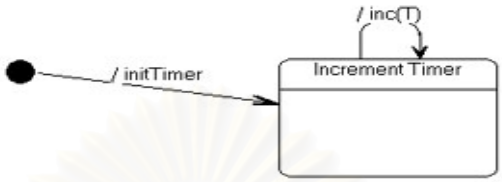
3.3 ระบบลิฟต์ (Lift)

ระบบลิฟต์ เป็นระบบในการลำเลียงบุคคลหรือสิ่งของในแนวดิ่ง ที่มีจุดพักตามชั้นต่างๆ การที่จะเขียนข้อกำหนดของระบบลิฟต์ได้นั้น ยังมีระบบย่อยๆ ที่เข้ามามีส่วนเกี่ยวข้องด้วย 3 ระบบ คือ

- ระบบจับเวลา เป็นระบบสำหรับใช้ในการจับเวลา โดยจะเป็นการเพิ่มค่าครั้งละ 1 จนกว่าจะมีเหตุการณ์ใดเหตุการณ์หนึ่งเกิดขึ้น จึงจะมีการตั้งค่าใหม่

- ระบบกดปุ่ม ใช้เพื่อสั่งการลิฟต์ว่า จะต้องเคลื่อนที่ขึ้น หรือลง
- ระบบระบุชั้น ใช้เพื่อสั่งการลิฟต์ว่า จะต้องเคลื่อนที่ไปยังชั้นไหน

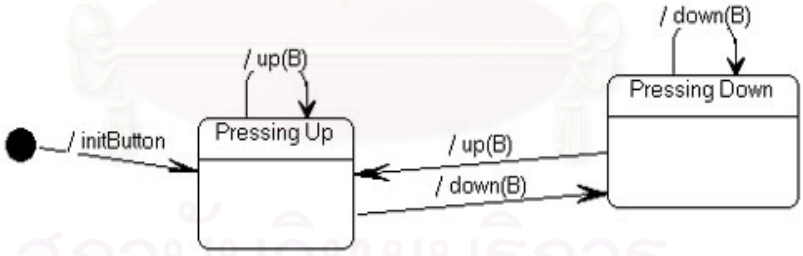
ดังนั้น ก่อนที่จะสามารถเขียนข้อกำหนดของระบบลิฟต์ได้นั้น ต้องมีการเขียนข้อกำหนดของ ระบบย่อยทั้ง 3 ระบบข้างต้นก่อน โดยแผนภาพสถานะและข้อกำหนดของทั้ง 3 ระบบ แสดงดังรูปที่ 5.10 ถึงรูปที่ 5.15



รูปที่ 5.10 แผนภาพสถานะของระบบจับเวลา

```
mod* TIMER{
      protecting(INT)

      *[ Timer ]*

      op initTimer : -> Timer
      bop inc : Timer -> Timer

      -- Observation operation.
      bop val : Timer -> Nat

      var T : Timer

      eq val(initTimer) = 0 .
      eq val(inc(T)) = val(T) + 1 .
}
```

รูปที่ 5.11 ข้อกำหนดรูปนัยของระบบจับเวลา

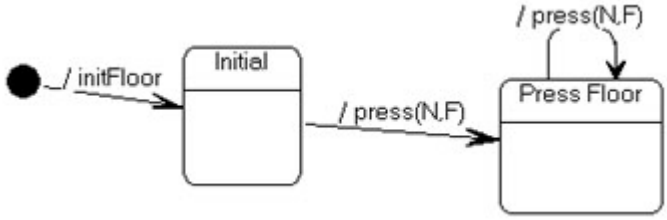


รูปที่ 5.12 แผนภาพสถานะของระบบกดปุ่ม

```
mod! UP-DOWN
 {
     [ Value ]

     op up : -> Value
     op down : -> Value
 }

mod* BUTTON{
     protecting(UP-DOWN)

     *[ Button ]*
```

```
     op initButton : -> Button
     bop down : Button -> Button
     bop up : Button -> Button

     -- Observation operation.
     bop status : Button -> Value

     var B : Button

     eq status(initButton) = up .
     eq status(up(B)) = up .
     eq status(down(B)) = down .
}
```

รูปที่ 5.13 ข้อกำหนดรูปนัยของระบบกดปุ่ม

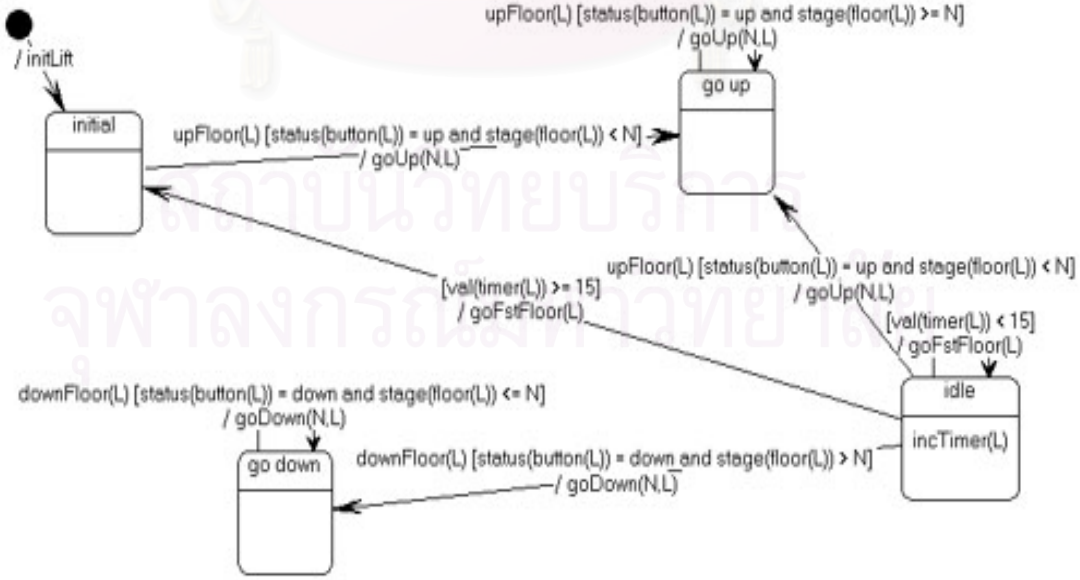


รูปที่ 5.14 แผนภาพสถานะของระบบระบุชั้น

```
mod* FLOOR{
      protecting(NAT)

      *[ Floor ]*

      op initFloor : -> Floor
      bop press : Nat Floor -> Floor

      -- Observation operation.
      bop stage : Floor -> Nat

      var F : Floor
      var N : Nat

      eq stage(initFloor) = 1 .
      eq stage(press(N,F)) = N .
}
```

รูปที่ 5.15 ข้อกำหนดรูปนัยของระบบระบุชั้น

เมื่อได้ข้อกำหนดของระบบย่อยทั้ง 3 เรียบร้อยแล้ว ก็ทำการนำเข้ามอดูลทั้ง 3 เพื่ออธิบายถึงพฤติกรรมการทำงานของระบบลิฟต์ได้ ซึ่งจะได้แผนภาพสถานะของมอดูลลิฟต์ ดังรูปที่ 5.16



รูปที่ 5.16 แผนภาพสถานะของระบบลิฟต์

และจากแผนภาพสถานะ ดังรูปที่ 5.16 สามารถสร้างเป็นข้อกำหนดรูปนัยคาเฟโอบีเจของระบบลิฟต์ ได้ดังรูปที่ 5.17

```
mod* LIFT{
   protecting(TIMER)
   protecting(BUTTON)
   protecting(FLOOR)

   *[ Lift ]*

   bop incTimer : Lift -> Lift
   op initLift : -> Lift
   bop upFloor : Lift -> Lift
   bop goUp : Nat Lift -> Lift
   bop downFloor : Lift -> Lift
   bop goDown : Nat Lift -> Lift
   bop goFstFloor : Lift -> Lift

   -- Observation operation.
   bop getTimer : Lift -> Nat
   bop getFloor : Lift -> Nat
   bop getButton : Lift -> Value

   -- Projection operation.
   bop timer : Lift -> Timer
   bop button : Lift -> Button
   bop floor : Lift -> Floor

   var N : Nat
   var L : Lift

   eq getFloor(L) = stage(floor(L)) .
   eq getTimer(L) = val(timer(L)) .
   eq getButton(L) = status(button(L)) .

   eq timer(initLift) = initTimer .
   eq button(initLift) = initButton .
   eq floor(initLift) = initFloor .

   eq timer(goFstFloor(L)) = timer(L) .
   eq button(goFstFloor(L)) = button(L) .
   ceq floor(goFstFloor(L)) = initFloor
      if val(timer(L)) >= 15 .
   eq timer(downFloor(L)) = timer(L) .
   eq button(downFloor(L)) = down(button(L)) .
   eq floor(downFloor(L)) = floor(L) .
   ceq timer(goDown(N,L)) = timer(L)
      if status(button(L)) == down and
      stage(floor(L)) <= N .
   ceq timer(goDown(N,L)) = initTimer
      if status(button(L)) == down and
      stage(floor(L)) > N .
   eq button(goDown(N,L)) = button(L)
   ceq floor(goDown(N,L)) = floor(L)
      if status(button(L)) == down and
      stage(floor(L)) <= N .
   ceq floor(goDown(N,L)) = press(N,floor(L))
      if status(button(L)) == down and
      stage(floor(L)) > N .
   eq timer(upFloor(L)) = timer(L) .
   eq button(upFloor(L)) = up(button(L)) .
   eq floor(upFloor(L)) = floor(L) .
   ceq timer(goUp(N,L)) = timer(L)
      if status(button(L)) == up and
      stage(floor(L)) >= N .
   ceq timer(goUp(N,L)) = initTimer
      if status(button
      (L)) == up and stage(floor(L)) < N .
   eq button(goUp(N,L)) = button(L) .
   ceq floor(goUp(N,L)) = floor(L)
      if status(button(L)) == up and
      stage(floor(L)) >= N .
   ceq floor(goUp(N,L)) = press(N,floor(L))
      if status(button(L)) == up and
      stage(floor(L)) < N .
   eq timer(incTimer(L)) = inc(timer(L)) .
   eq button(incTimer(L)) = button(L) .
   eq floor(incTimer(L)) = floor(L) .
   ceq floor(goFstFloor(L)) = floor(L)
      if val(timer(L)) < 15 .
}
```

รูปที่ 5.17 ข้อกำหนดรูปนัยคาเฟโอบีเจของระบบลิฟต์

การตรวจสอบผลลัพธ์ของมอดูลลิฟต์

เมื่อนำข้อกำหนดของมอดูลลิฟต์ ไปทำการตรวจสอบกับตัวแปลภาษาคาเฟโอบีเจ ซึ่งจะทำการตรวจสอบ 2 ขั้นตอนคือ

ก) ขั้นตอนการตรวจสอบความถูกต้องตามวากยะสัมพันธ์ของภาษาคาเฟโอบีเจ ได้ผลดังรูปที่ 5.18 ซึ่งแสดงว่าข้อกำหนดของมอดูลลิฟต์ที่สร้างจากแผนภาพสถานะ ถูกต้องตามวากยะสัมพันธ์ของภาษาคาเฟโอบีเจ

```
CafeOBJ> in lift.mod
processing input : d:\user\g41mrt\cafeobj\test\.\lift.mod
-- defining module* TIMER....._..*
** system already proved =*= is a congruence of TIMER done.
-- defining module! UP-DOWN..._* done.
-- defining module* BUTTON....._...*
** system already proved =*= is a congruence of BUTTON done.
-- defining module* FLOOR....._..*
** system already proved =*= is a congruence of FLOOR done.
-- defining module* LIFT....._.....*
** system failed to prove =*= is a congruence of LIFT done.
LIFT>
```

รูปที่ 5.18 ผลการตรวจสอบวากยะสัมพันธ์ของมอดูลลิฟต์

ข) ขั้นตอนการตรวจสอบความถูกต้องของข้อกำหนด โดยแยกทดสอบตามกรณีการทดสอบที่จะเป็นไปได้ทั้งหมด ดังนี้

- กรณีที่ 1 ตรวจสอบสถานะของการกดปุ่ม เมื่อมีการกดปุ่มขึ้น จะได้สถานะของลิฟต์เป็น ‘up’
  ซึ่งตรวจสอบโดยเปรียบเทียบการกดปุ่มว่าเท่ากับ ‘up’ รึไม่ ผลที่ได้คือ ‘true’ แสดงว่าสถานะของลิฟต์เป็น ‘up’
- กรณีที่ 2 ตรวจสอบสถานะของการกดปุ่ม เมื่อมีการกดปุ่มลง จะได้สถานะของลิฟต์เป็น ‘down’
  ซึ่งตรวจสอบโดยเปรียบเทียบการกดปุ่มว่าเท่ากับ ‘down’ รึไม่ ผลที่ได้คือ ‘true’ แสดงว่าสถานะของลิฟต์เป็น ‘down’
- กรณีที่ 3 ตรวจสอบการเคลื่อนที่ของลิฟต์เมื่อมีการกดปุ่มขึ้น และชั้นที่จะไปมีค่ามากกว่าชั้นปัจจุบัน ลิฟต์จะมีการเคลื่อนที่ขึ้น ดังนั้นจะได้ชั้นที่กดเป็นคำตอบ
  ซึ่งตรวจสอบโดยกดปุ่มขึ้น ตามด้วยกดชั้นที่ต้องการไปคือ 6 และชั้นปัจจุบันคือ 1 ผลที่ได้คือ 6
- กรณีที่ 4 ตรวจสอบการเคลื่อนที่ของลิฟต์เมื่อมีการกดปุ่มลง และชั้นที่จะไปมีค่ามากกว่าชั้นปัจจุบัน ลิฟต์จะไม่มีการเคลื่อนที่ ดังนั้นจะได้ชั้นเดิมเป็นคำตอบ
  ซึ่งตรวจสอบโดยกดปุ่มลง ตามด้วยกดชั้นที่ต้องการไปคือ 5 และชั้นปัจจุบันคือ 1 ผลที่ได้คือ 1
- กรณีที่ 5 ตรวจสอบการเคลื่อนที่ของลิฟต์เมื่อมีการกดปุ่มลง และชั้นที่จะไปมีค่าน้อยกว่าชั้นปัจจุบัน ลิฟต์มีการเคลื่อนที่ลง ดังนั้นจะได้ชั้นที่กดเป็นคำตอบ
  ซึ่งตรวจสอบโดยกดปุ่มลง ตามด้วยกดชั้นที่ต้องการไปคือ 7 และชั้นปัจจุบันคือ 5 ผลที่ได้คือ 5
- กรณีที่ 6 ตรวจสอบการเคลื่อนที่ของลิฟต์เมื่อมีการกดปุ่มขึ้น และชั้นที่จะไปมีค่าน้อยกว่าชั้นปัจจุบัน ลิฟต์ไม่มีการเคลื่อนที่ ดังนั้นจะได้ชั้นเดิมเป็นคำตอบ

ซึ่งตรวจสอบโดยกดปุ่มขึ้น ตามด้วยกดชั้นที่ต้องการไปคือ 4 และชั้นปัจจุบันคือ 5 ผลที่ได้คือ 5

- กรณีที่ 7 ตรวจสอบเวลาเมื่อลิฟต์ ไปถึงชั้นที่ต้องการแล้ว เมื่อลิฟต์เคลื่อนที่ไปถึงชั้นที่ต้องการแล้ว เวลาจะถูกตั้งค่าเป็น 0 ดังนั้นจะได้คำตอบเป็น 0
- กรณีที่ 8 ตรวจการเคลื่อนที่ของลิฟต์ เมื่อลิฟต์ไม่มีการเคลื่อนที่ 15 วินาที ขึ้นไป ลิฟต์จะเคลื่อนที่ไปยังชั้น 1 ดังนั้นคำตอบที่ได้คือ 1
- กรณีที่ 9 ตรวจการเคลื่อนที่ของลิฟต์ เมื่อลิฟต์ไม่มีการเคลื่อนที่ น้อยกว่า 15 วินาที ลิฟต์จะยังไม่มีการเคลื่อนที่ ดังนั้นคำตอบที่ได้ คือชั้นล่าสุดที่ลิฟต์หยุดอยู่

ผลการทดสอบเป็นดังรูปที่ 5.19

```
LIFT> in liftprove.txt
processing input : d:\user\g41mrt\thesis\.\liftprove.txt
-- opening module LIFT.. done.
-- reduce in %LIFT : getButton(upFloor(initLift)) == up                    <-- 1
true : Bool
(0.000 sec for parse, 5 rewrites(0.050 sec), 10 matches)
-- reduce in %LIFT : getButton(downFloor(initLift)) == down                <-- 2
true : Bool
(0.000 sec for parse, 5 rewrites(0.000 sec), 9 matches)
-- reduce in %LIFT : getFloor(goUp(6,upFloor(initLift)))                   <-- 3
6 : NzNat
(0.000 sec for parse, 23 rewrites(0.060 sec), 59 matches)
-- reduce in %LIFT : getFloor(goDown(5,downFloor(initLift)))               <-- 4
1 : NzNat
(0.000 sec for parse, 14 rewrites(0.000 sec), 27 matches)
-- reduce in %LIFT : getFloor(goDown(7,downFloor(goUp(5,upFloor(initLift))))) <-- 5
5 : NzNat
(0.000 sec for parse, 56 rewrites(0.000 sec), 150 matches)
-- reduce in %LIFT : getFloor(goUp(4,upFloor(goDown(5,downFloor(goUp(8,
upFloor(initLift)))))))                                                   <-- 6
5 : NzNat
(0.000 sec for parse, 186 rewrites(0.060 sec), 517 matches)
-- reduce in %LIFT : getTimer(goUp(4,upFloor(initLift)))                   <-- 7
0 : Zero
(0.000 sec for parse, 21 rewrites(0.050 sec), 51 matches)
-- reduce in %LIFT :
getFloor(goFstFloor(incTimer(incTimer(incTimer(incTimer(incTimer(incTimer(incTim
er(incTimer(incTimer(incTimer(incTimer(incTimer(incTimer(incTimer(incTimer(incTi
mer(goDown(4, downFloor(goUp(6, upFloor(initLift))))))))))))))))))))))   <-- 8
1 : NzNat
(0.000 sec for parse, 116 rewrites(0.050 sec), 372 matches)
-- reduce in %LIFT :
getFloor(goFstFloor(incTimer(incTimer(incTimer(incTimer(incTimer(incTimer(incTim
er(incTimer(incTimer(incTimer(incTimer(incTimer(incTimer(goDown(4,          <-- 9
downFloor(goUp(6,upFloor(initLift))))))))))))))))))))
4 : NzNat
```

รูปที่ 5.19 ผลการตรวจสอบความถูกต้องของข้อกำหนดของมอดูลลิฟต์

บทที่ 6
สรุปผลการวิจัย

จากการศึกษา และวิจัย สามารถสรุปผลการวิจัย ปัญหา ข้อจำกัดของการวิจัยที่พบ ข้อเสนอแนะ รวมทั้งการนำเสนอผลงานในการประชุมวิชาการ ดังนี้

1 สรุปผลการวิจัย

จากการศึกษาหาความสัมพันธ์ระหว่างแผนภาพสถานะของยูเอ็มแอล และภาษาคาเฟโอบีเจ เพื่อดูว่าแต่ละส่วนของแผนภาพสถานะสอดคล้องกับส่วนใดของสัญกรณ์ทางภาษาของข้อกำหนดรูปนัยคาเฟโอบีเจ สรุปได้ว่า มักจะใช้ 1 แผนภาพสถานะต่อ 1 มอดูลของข้อกำหนดรูปนัยคาเฟโอบีเจ ที่การดำเนินการสามารถแสดงในรูปของ เหตุการณ์ การกระทำของการเปลี่ยนสถานะ และกิจกรรมของสถานะ และทำการขยายคุณสมบัติของสถานะเพื่อแสดงถึงสัจพจน์ของการดำเนินการที่มีผลต่อสถานะนั้นๆ ดังนั้นแต่ละสถานะจะประกอบด้วยสมการของสัจพจน์ด้วย และจากผลสรุปที่ได้ก็สร้างกฎการแปลงจากแผนภาพสถานะเป็นภาษาคาเฟโอบีเจ โดยมีกฎการแปลง 7 ข้อ ซึ่งมีใจความสำคัญดังต่อไปนี้

- แผนภาพสถานะใดๆ เทียบเท่ากับมอดูลหนึ่งของคาเฟโอบีเจได้ โดยควรจะมีชื่อมอดูลตามชื่อวัตถุ และมีชื่อชนิดข้อมูลแฝงเดียวกันกับชื่อมอดูล
- การกระทำสู่สถานะเริ่มต้น เป็น การดำเนินการที่ไม่มีอาร์กิวเมนต์สำหรับกำหนดคุณสมบัติของวัตถุแบบเสถียร หรือ มีอาร์กิวเมนต์เป็นตัวระบุวัตถุสำหรับกำหนดคุณสมบัติของวัตถุแบบพลวัต ส่วนผลลัพธ์มีชนิดข้อมูลเป็นชนิดข้อมูลของมอดูลปัจจุบัน
- เหตุการณ์ของการเปลี่ยนสถานะ เป็น การดำเนินการที่มีอาร์กิวเมนต์อย่างน้อย 1 ตัวที่มีชนิดข้อมูลเป็นชนิดข้อมูลของมอดูลปัจจุบัน ส่วนผลลัพธ์มีชนิดข้อมูลเป็นชนิดข้อมูลของมอดูลปัจจุบัน
- การกระทำสู่สถานะใดๆ ที่ไม่ใช่สถานะเริ่มต้น เป็น การดำเนินการที่มีอาร์กิวเมนต์อย่างน้อย 1 ตัวที่มีชนิดข้อมูลเป็นชนิดข้อมูลของมอดูลปัจจุบัน ส่วนผลลัพธ์มีชนิดข้อมูลเป็นชนิดข้อมูลของมอดูลปัจจุบัน
- กิจกรรมภายในสถานะ เป็น การดำเนินการที่มีอาร์กิวเมนต์อย่างน้อย 1 ตัวที่มีชนิดข้อมูลเป็นชนิดข้อมูลของมอดูลปัจจุบัน ส่วนผลลัพธ์มีชนิดข้อมูลเป็นชนิดข้อมูลของมอดูลปัจจุบัน
- เงื่อนไขของการเปลี่ยนสถานะ เป็น เงื่อนไขของสัจพจน์ ณ สถานะที่เงื่อนไขมีผลกระทบ

- หากมอดูลนำเข้ามีลักษณะเป็นโมเดลแบบอิสระ จะมีการดำเนินการ 1 ตัวที่เป็นตัวเชื่อมระหว่างมอดูลปัจจุบันกับการดำเนินการของมอดูลนำเข้า โดยเรียกการดำเนินการประเภทนี้ว่า การดำเนินการโปรเจคชัน ที่ชนิดข้อมูลของอาร์กิวเมนต์เป็นชนิดข้อมูลของมอดูลปัจจุบัน และมีตัวระบุวัตถุเป็นอาร์กิวเมนต์ด้วยสำหรับระบบพลวัต ส่วนผลลัพธ์มีชนิดข้อมูลเป็นชนิดข้อมูลของมอดูลนำเข้า

การวิจัยได้ครอบคลุมถึงการพัฒนาเครื่องมือซอฟต์แวร์สำหรับสร้างข้อกำหนดโดยใช้กฎการแปลงดังกล่าวมาเป็นหลัก โดยซอฟต์แวร์ที่สร้างขึ้น เป็นบรรณาธิกรณ์แผนภาพสถานะ ที่สามารถกำหนดคุณสมบัติต่างๆ คือ

- เหตุการณ์ เงื่อนไข และการกระทำ ของการเปลี่ยนสถานะ
- กิจกรรม และสมการสัจพจน์ของสถานะ
- การนำเข้ามอดูลเพื่อนำมาอธิบายพฤติกรรมของการดำเนินการของมอดูลปัจจุบัน
- การดำเนินการสังเกตค่า และสมการสัจพจน์
- การดำเนินการโปรเจคชันที่เหมาะสม

นอกจากนี้ ซอฟต์แวร์ที่สร้างขึ้นยังอำนวยความสะดวกในการเขียนข้อกำหนดรูปนัยคาเฟโอบีเจ โดยการแสดงรายการของตัวแปร หรือการดำเนินการทั้งหมดที่สามารถเรียกใช้ภายในมอดูลปัจจุบันได้ในขั้นตอนของการเขียนสมการของสัจพจน์ โดยเป็นการแสดงรายการของตัวแปรหรือการดำเนินการที่เหมาะสม ณ ตำแหน่งที่ต้องการ และสามารถสร้างข้อกำหนดรูปนัยคาเฟโอบีเจจากแผนภาพที่สร้างขึ้น โดยการเก็บเป็นแฟ้มข้อความ 1 แฟ้มที่ลำดับตำแหน่งของมอดูลให้เรียบร้อยแล้ว เพื่อความสะดวกในการนำไปพิสูจน์ด้วยตัวแปลงภาษาคาเฟโอบีเจ

ผู้วิจัยได้ใช้ระบบทดสอบ 3 ระบบ คือ ระบบตัวนับ ระบบตัวนับที่มีสวิตซ์ และระบบลิฟต์ โดยเครื่องมือสามารถสร้างข้อกำหนดของภาษาคาเฟโอบีเจได้ ทั้ง 3 ระบบ และจากการนำผลที่ได้ไปทดสอบกับตัวแปลภาษาคาเฟโอบีเจ ทั้งการตรวจสอบวากยะสัมพันธ์และความถูกต้องของข้อกำหนด ก็ปรากฏว่า ไม่มีข้อผิดพลาดใดๆ

อย่างไรก็ตาม การใช้งานเครื่องมือที่ได้พัฒนาขึ้นนั้น ผู้ใช้จำเป็นต้องรู้วากยะสัมพันธ์ของภาษาคาเฟโอบีเจบ้าง และต้องมีความเข้าใจคุณลักษณะของแผนภาพสถานะของยูเอ็มแอลด้วย

2 ประโยชน์ของเครื่องมือสร้างข้อกำหนดรูปนัยคาเฟโอบีเจจากแผนภาพสถานะ

1) เป็นเครื่องมือที่ช่วยในการสนับสนุนการเขียนข้อกำหนดรูปนัยภาษาคาเฟโอบีเจ
2) เป็นเครื่องมือที่ช่วยในการตรวจสอบความสมบูรณ์ของข้อกำหนดที่เขียนขึ้น
3) เป็นเครื่องมือที่ช่วยในการสื่อสารระหว่าง ผู้ออกแบบระบบ กับลูกค้า

3 ปัญหาและข้อจำกัดที่พบจากการวิจัย

1) เครื่องมือนี้ไม่สามารถแสดงตัวแปรหรือการดำเนินการที่เหมาะสม สำหรับการดำเนินการระบุตำแหน่งของอาร์กิวเมนต์ โดยใช้ขีดล่าง (_ : underscore) เนื่องจากไม่สามารถหาได้ว่าชื่อของการดำเนินการคืออะไร ตัวอย่างเช่น การดำเนินการ “if _ then _”
2) การระบุโหมดของการนำเข้าทั้ง 3 โหมด ในมอดูลที่จะนำเข้า ไม่สามารถใช้ตัวย่อได้ เนื่องจากส่วนของมอดูลที่อยู่ก่อนการกำหนดการนำเข้า อาจจะมี ตัวย่อของโหมดการนำเข้าปรากฏอยู่ ซึ่งจะทำให้การวิเคราะห์กระจาย (Parser) ไม่ถูกต้อง
3) รูปแบบของมอดูลนำเข้าต้องถูกต้องตามวากยะสัมพันธ์ของภาษาคาเฟโอบีเจ
4) การแสดงตัวแปรหรือการดำเนินการที่เหมาะสม จะแสดงได้เฉพาะของการดำเนินการที่ผ่านการเตรียมการนำเข้า ในมอดูลการนำเข้าข้อมูลเท่านั้น
5) การกำหนดการดำเนินการในมอดูลปัจจุบันจะต้องมี วงเล็บ และเครื่องหมายคอมมา (,) ขั้นระหว่างอาร์กิวเมนต์ ในกรณีที่การดำเนินการมีอาร์กิวเมนต์
6) หลังจากกำหนดสมการของสัจพจน์เรียบร้อยแล้ว แต่เกิดมีการเปลี่ยน รายละเอียดของการดำเนินการ หรือเงื่อนไข ทางผู้ใช้จะต้องทำการตรวจสอบรายการของสมการสัจพจน์เพื่อแก้ไขให้ตรงกับรายละเอียดที่เปลี่ยนไป
7) ไม่มีการพิจารณาการนำเข้า ที่มีโหมดนำเข้าเป็น การขยาย และ การใช้

4 ข้อเสนอแนะ

จากการศึกษาวิจัย ถึงแม้ว่าเครื่องมือที่สร้างขึ้น จะสามารถสร้างข้อกำหนดรูปนัยของภาษาคาเฟโอบีเจได้ แต่ก็ยังมีบางส่วนของข้อกำหนดที่ต้องกำหนดแยกออกไปจากแผนภาพสถานะ เช่น การดำเนินการสังเกตค่า การดำเนินการโปรเจคชัน และสัจพจน์ของการดำเนินการสังเกตค่า เป็นต้น และยังไม่สนับสนุนการออกแบบในเชิงพฤติกรรมของระบบงานที่ซับซ้อนได้ดีนักเพราะไม่เห็นภาพรวมของทั้งระบบ ดังนั้นเพื่อให้สามารถกำหนดส่วนประกอบต่างๆ ของทั้งระบบได้โดยครบถ้วนสมบูรณ์ จึงน่าจะมีการนำแผนภาพคลาสมาเป็นส่วนเสริมด้วย เพื่อการกำหนดส่วนประกอบต่างๆ จะสามารถกระทำได้ในแผนภาพเลย เช่น การดำเนินการโปรเจคชัน ก็กำหนดที่เส้นความสัมพันธ์ระหว่างคลาส เป็นต้น

5 ผลงานตีพิมพ์

ผลงานวิจัยนี้ได้รับคัดเลือกให้นำเสนอในงานประชุมวิชาการและตีพิมพ์ในเอกสาร Proceedings of the 2nd International Conference on Intelligent Technologies (InTech 2001) ซึ่งจัดขึ้นที่มหาวิทยาลัยอัสสัมชัญ ระหว่างวันที่ 27-29 พฤศจิกายน 2544 โดยรายละเอียดแสดงอยู่ในภาคผนวก ง

รายการอ้างอิง

[1] Shusaku Iida. An Algebraic Formal Method for Component based Software Developments, Ph.D. Thesis, Japan Advanced Institute of Science and Technology, March 1999

[2] Shusaku Iida, Kokichi Futatsuki and Razvan Diaconnescu. Component-Based Algebraic Specification : behavioral specification for component-based software engineering, 7th OOPSLA workshop on Behavioral Semantics of Object-Oriened Business and System Specification, 1998

[3] T.H. Tse. FOOD : a Graphical Interface for Object-Oriented Algebraic Specifications, Dept. of Computer Science, The University of of Hong Kong, 1996

[4] Liliana Favre and Silvia Clerici. Integrating UML and algebraic specification techniques, In Proceeding of Technology of Object-Oriented Languages and Systems, IEEE, 1999

[5] Andrew Harry. Formal Methods Fact File VDM and Z, Chichester: John Wiley & Sons, 1996

[6] Kenneth Slonneger and Barry L. Kurtz. Formal Syntax and Semantics of Programming Language, Addison-Wesley, 1995

[7] Sangwook Kim. Software Engineering, http://woorisol.kyungpook.ac.kr/lab/prof/SoftEng/ch10.htm, Dec 1999

[8] Razvan Diaconnescu and Kokichi Futatsuki. CafeOBJ Report: The Languange, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification, World Scientific, 1998

[9] Kokichi Futatsuki and Ataru Nakagawa. An Overview of CAFÉ Specification Environment: an Algebraic approach for creating, verifying, and maintaining formal specifications over networks, Proceeding of 1st IEEE International Conference on Formal Engineering Methods, IEEE, 1997

[10] Object Management Group. OMG Unified Modeling Language Specification version 1.3, OMG. 1999

[11] Martin Fowler and Kendall Scott. UML Distilled: Applying the Standard Object Modeling Language, Addison-Wesley, 1997

[12] Hans-Erik Eriksson and Magnus Penker. UML Toolkit, United States of America: John Wiley & Sons, 1998

ภาคผนวก

ภาคผนวก ก
วากยะสัมพันธ์ของภาษาคาเฟโอบีเจ

เนื่องจากข้อมูลเข้าซึ่งเป็นข้อกำหนดของมอดูลต่างๆ นั้น เป็นข้อกำหนดของภาษาคาเฟโอบีเจ ดังนั้นมอดูลที่จะนำเข้า จะต้องมีความถูกต้องตามวากยะสัมพันธ์ของภาษาคาเฟโอบีเจก่อน ซึ่งมีรูปแบบตาม BNF ดังนี้

CafeOBJ programs

Program ::= { *module* | *view* | *eval* }

Module declaration

module ::= *module_type module_name* [*parameters*]
[*principal_sort*] "{" *module_elt* … "}"

module_type ::= module | module! | module*

module_name ::= *ident*

parameters ::= "(" *parameter*, … ")"

parameter ::= [protecting | extending]
parameter_name :: *module_expr*

parameter_name ::= *ident*

principal_sort ::= principal_sort *sort_name*

module_elt ::= *sharing* | *import* | *sort* | *record* | *operation* | *variable* | *axiom*
| *comment*

sharing ::= share "(" *module_name* ")"

import ::= { protecting | extending | using }
"(" *module_expr*, … ")"

sort ::= *visible_sort* | *hidden_sort*

visible_sort ::= "[" *sort_secl*, … "]"

hidden_sort ::= "*[" *sort_decl*, … "]*"

sort_desl ::= *sort_name* … [*supersorts* …]

supersorts ::= < *sort_name* …

sort_name ::= *sort_symbol*[*qualifier*]

sort_symbol ::= *ident*

qualifier ::= “.”*module_expr*[*qualifier*]

record ::= record *sort_name* [*super* …]
“{“ { *slot* | *comment* } … “}”

super ::= “[“ *sort_name* [“(“ *slot_rename*, … “)”] “]”

slot ::= *slot_name* : *sort_name* | *slot_name* = “(“*term*”)” : *sort_name*

slot_name ::= *ident*

slot_rename ::= *slot_name* -> *slot_name*

operation ::= { op | bop } *operation_symbol* : [*arity*] -> *coarity* [*op_attrs*]

arity ::= *sort_name* …

coarity ::= *sort_name*

op_attrs ::= “{“ *op_attr* … “}”

op_attr ::= constr | associative | commutative | idempotent
| { id: | idr: } “(“ *term* “)” | coherent
| strat: “(“ *natural* … “)”
| prec: *natural* | l-assoc | r-assoc

variable ::= var *var_name* : *sort_name* | vars *var_name* … : *sort_name*

var_name ::= *ident*

axiom ::= *equation* | *cequation* | *transition* | *ctransition*

equation ::= { eq | beq } [*label*] *term* = *term* “.”

cequation ::= { ceq | bceq } [*label*] *term* = *term* if *term* “.”

transition ::= { trans | btrans } [*label*] *term* => *term* “.”

ctransition ::= { ctrans | bctrans } [*label*] *term* => *term* if *term* “.”

label ::= “[“ *ident* “]:”

comment ::= [-- | --> | ** | **>] ASCII character …

Module Expressions

module_expr ::= *module_name* | *sum* | *rename* | *instantiation* | “(“ *module_expr* “)”

sum ::= *module_expr* { + *module_expr* } …

rename ::= *module_expr* * "{" *rename_map* "}"

instantiation ::= *module_expr* "(" { *ident[qualifier]* < = *aview*}, … ")"

rename_map ::= *sort-map* | *op_map*

sort_map ::= { sort | hsort } *sort_name -> ident*

op_map ::= { op | bop } *op_name -> operator_symbol*

op_name ::= *operation_symbol* | "(" *operation_symbol* ")" *qualifier*

aview ::= *view_name* | *module_expr*

| view to *module_expr* "{" *view_elt*, …"}"

view_name ::= *ident*

view_elt ::= *sort_map* | *op_view* | *variable*

op_view ::= op_map | *term -> term*

View Declarations

view ::= view *view_name* from *module_expr* to *module_expr*

"{" *view_elt*, … "}"

Evaluations

Eval ::= { reduce | behavioral-reduce | execute } *context term* "."

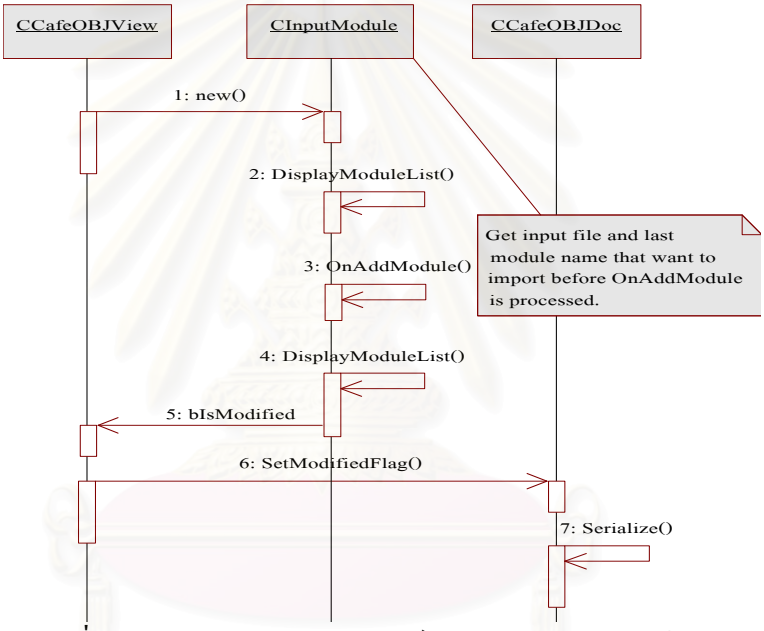
context ::= in module_expr :

ภาคผนวก ข
การพัฒนาบรรณาธิกรณ์แผนภาพสถานะเพื่อสร้างข้อกำหนดรูปนัยคาเฟโอบีเจ

การออกแบบบรรณาธิกรณ์แผนภาพสถานะในบทที่ 4 สามารถพัฒนาโดยใช้โปรแกรมภาษาใดๆ ก็ได้ แต่ในการทำวิทยานิพนธ์นี้ได้เลือกใช้โปรแกรมภาษา Microsoft Visual C++ สำหรับการพัฒนา โดยมีรายละเอียดดังนี้

1 ส่วนกำหนดการอ้างอิง

ลำดับการทำงานของส่วนนี้สามารถแสดงด้วยแผนภาพลำดับเหตุการณ์ดังรูปที่ ข-1



รูปที่ ข-1 แผนภาพลำดับเหตุการณ์ของส่วนกำหนดการอ้างอิง

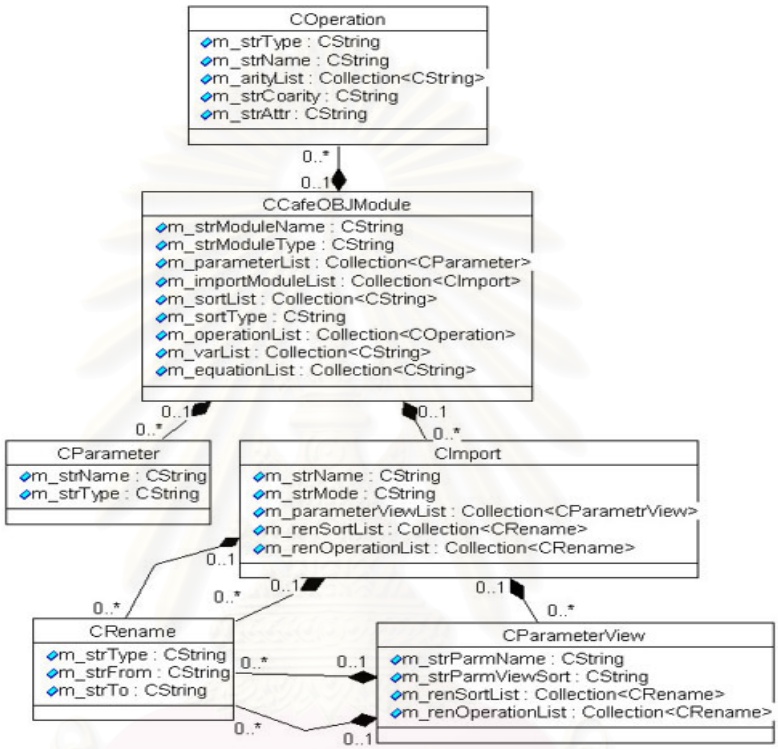
โครงสร้างข้อมูลของแต่ละมอดูล สามารถแสดงเป็นแผนภาพคลาสได้ดังรูปที่ ข-2 ซึ่งมีรายละเอียดดังนี้

```
class CParameter : public CObject
{
    CString     m_strName;
    CString     m_strType;
};
typedef CTypedPtrList<CObList,CParameter*> CParameterList;
```

คลาส **CParameter** คือ โครงสร้างข้อมูลสำหรับการเก็บตัวแปรเสริม (Parameter) 1 ตัวแปร ที่ประกอบด้วย

- ชื่อตัวแปร
- มอดูลของตัวแปรที่กำหนด

ส่วน **CParameterList** ใช้สำหรับการเก็บรายการของตัวแปรเสริม ในกรณีที่มีตัวแปรเสริมมากกว่า 1 ตัวแปร



รูปที่ ข-2 แผนภาพคลาสสำหรับเก็บข้อมูลของมอดูลอ้างอิง 1 มอดูล

```
class CRename : public CObject
{
    CString     m_strType;
    CString     m_strFrom;
    CString     m_strTo;
};
typedef CTypedPtrList<CObList, CRename*> CRenSortList;
typedef CTypedPtrList<CObList, CRename*> CRenOperationList;
```

คลาส **CRename** คือ โครงสร้างข้อมูลสำหรับ การเก็บชนิดข้อมูล หรือการดำเนินการ ของมอดูลที่นำเข้า ที่มีการเปลี่ยนชื่อ 1 รายการ ซึ่งประกอบด้วย

- ชนิดของข้อมูลที่จะมีการเปลี่ยนชื่อ คือ sort หรือ hsort สำหรับชนิดข้อมูล และ op หรือ bop สำหรับการดำเนินการ
- ชื่อเดิม
- ชื่อใหม่

โดย CRename เป็นคลาสพื้นฐานสำหรับ การเก็บข้อมูล 2 ประเภท คือ

- รายการของ ชนิดข้อมูลที่มีการเปลี่ยนชื่อ ซึ่งเก็บโดย **CRenSortList**
- รายการของ การดำเนินการที่มีการเปลี่ยนชื่อ ซึ่งเก็บโดย **CRenOperationList**

```
class CParameterView : public CObject
{
    CString             m_strParmName;
    CString             m_strParmViewSort;
    CRenSortList        m_renSortList;
    CRenOperationList   m_renOperationList;
};
typedef CTypedPtrList<CObList, CParameterView*> CParameterViewList;
```

คลาส **CParameterView** คือ โครงสร้างข้อมูลสำหรับการเก็บส่วนของพารามิเตอร์ ที่มีการเปลี่ยนแปลงชนิดข้อมูลหรือการดำเนินการเป็นของอีกมอดูลหนึ่ง สำหรับ 1 พารามิเตอร์ ที่ประกอบด้วย

- ชื่อพารามิเตอร์
- ชื่อมอดูลที่ต้องการให้พารามิเตอร์มอดูลเป็น
- รายการของชนิดข้อมูลของพารามิเตอร์มอดูล ที่มีการเปลี่ยนชื่อ
- รายการของการดำเนินการพารามิเตอร์มอดูล ที่มีการเปลี่ยนชื่อ

CParameterViewList ใช้สำหรับการเก็บรายการของพารามิเตอร์ที่มีการเปลี่ยนแปลงข้อกำหนด

```
class CImport : public CObject
{
    CString             m_strName;
    CString             m_strMode;
    CParameterViewList  m_parameterViewList;
```

```
    CRenSortList        m_renSortList;
    CRenOperationList   m_renOperationList;
};
typedef CTypedPtrList<CObList, CImport*> CImportList;
```

คลาส **CImport** คือโครงสร้างข้อมูลที่มีการนำเข้า 1 มอดูล ที่ประกอบด้วย

- ชื่อของมอดูลที่นำเข้า
- วิธีในการนำเข้า
- รายการของพารามิเตอร์ที่มีการเปลี่ยนแปลงข้อกำหนด
- รายการของชนิดข้อมูลของมอดูลที่นำเข้า ที่มีการเปลี่ยนชื่อ
- รายการของการดำเนินการของมอดูลที่นำเข้า ที่มีการเปลี่ยนชื่อ

CImportList ใช้สำหรับการเก็บรายการของมอดูลที่นำเข้า

```
class COperation : public CObject
{
    CString             m_strType;
    CString             m_strName;
    CString             m_strCoarity;
    CStringList         m_arityList;
    CString             m_strAttr;
};
typedef CTypedPtrList<CObList,COperation*> COperationList;
```

คลาส **COperation** คือ โครงสร้างข้อมูลสำหรับการเก็บการดำเนินการ 1 การดำเนินการ ที่ประกอบด้วย

- ชนิดของการดำเนินการ
- ชื่อการดำเนินการ
- รายการชนิดข้อมูลของอาร์กิวเมนต์ (หรือเรียกว่า arity)
- ชนิดข้อมูลข้อมูลของผลลัพธ์ (หรือเรียกว่า coarity)
- ลักษณะประจำของการดำเนินการ (operation attribute)

COperationList ใช้สำหรับการเก็บรายการของการดำเนินการ

```
class CCafeOBJModule : public CObject
{
    CString             m_strModuleName;
    CString             m_strModuleType;
    CParameterList      m_parameterList;
    CImportList         m_importModuleList;
    CStringList         m_sortList;
    CString             m_sortType;
    COperationList      m_operationList;
    CStringList         m_varList;
    CStringList         m_equationList;
};
typedef CTypedPtrList<CObList, CCafeOBJModule*> CCafeOBJModuleList;
```

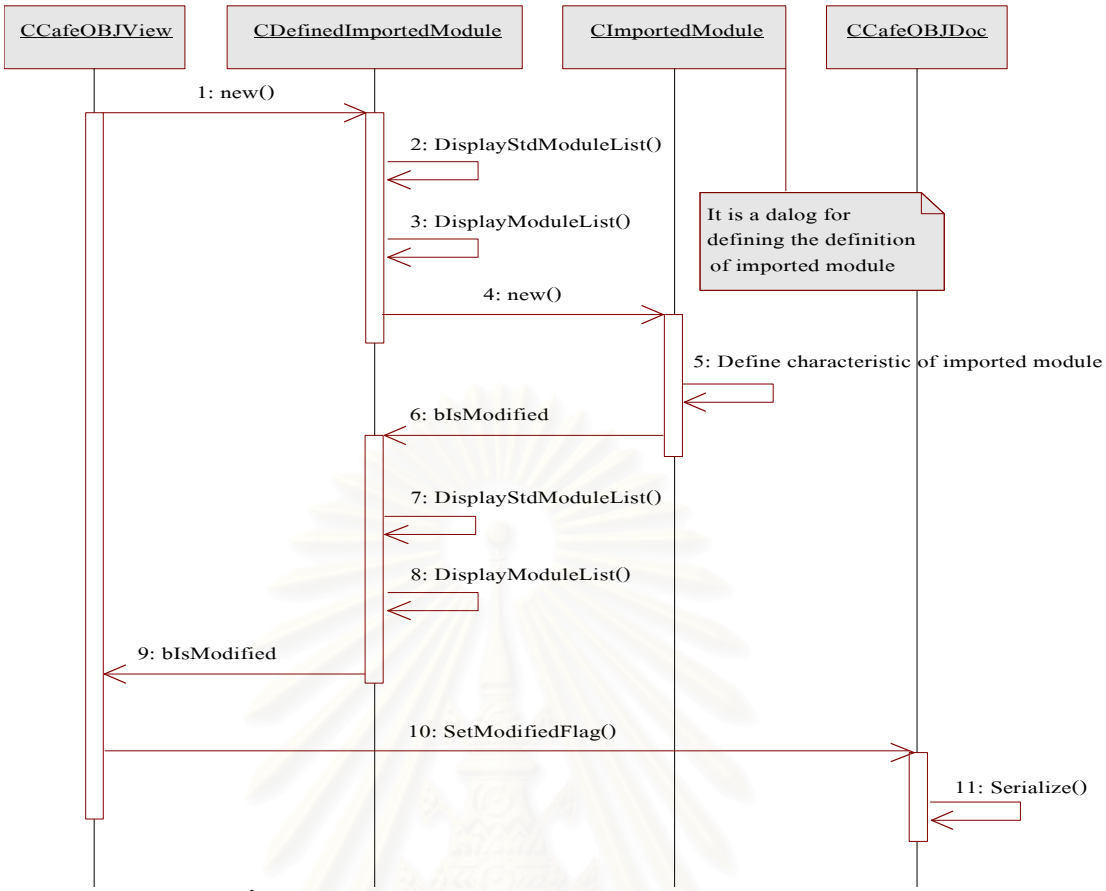
คลาส **CCafeOBJModule** คือ โครงสร้างข้อมูลสำหรับการเก็บข้อกำหนดของมอดูลนำเข้า 1 มอดูล ซึ่งประกอบไปด้วย

- ชื่อของมอดูล
- ประเภทของมอดูล เป็นโมเดลแบบยึดติด หรือโมเดลแบบอิสระ
- รายการของพารามิเตอร์
- รายการของมอดูลนำเข้า
- รายการของชนิดข้อมูล
- ประเภทของชนิดข้อมูล เป็นชนิดข้อมูลสังเกตค่า หรือชนิดข้อมูลแฝง
- รายการของการดำเนินการ
- รายการของตัวแปร
- รายการของสัจพจน์

CCafeOBJModuleList ใช้สำหรับการเก็บรายการของมอดูลทั้งหมดที่มีการดึงเข้ามาเพื่อใช้ในมอดูลปัจจุบัน

2 ส่วนกำหนดมอดูลนำเข้า

ลำดับการทำงานของส่วนนี้สามารถแสดงด้วยแผนภาพลำดับเหตุการณ์ดังรูปที่ ข-3



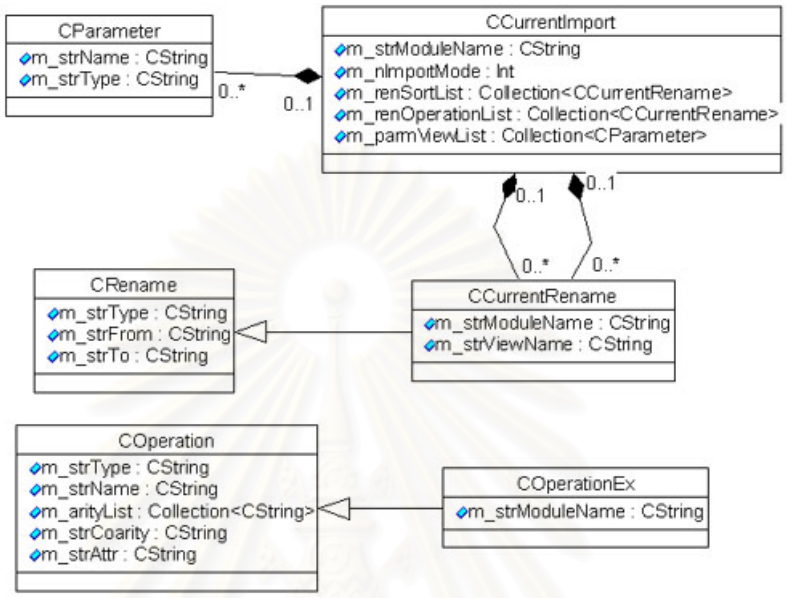
รูปที่ ข-3 แผนภาพลำดับเหตุการณ์ของส่วนกำหนดมอดูลนำเข้า

โครงสร้างข้อมูลของแต่ละมอดูลที่นำเข้า แสดงเป็นแผนภาพคลาสได้ดังรูปที่ ข-4 ซึ่งมีรายละเอียดดังนี้

```
class CCurrentRename : public CRename
{
    CString     m_strModuleName;
    CString     m_strViewName;
};
typedef CTypedPtrList<CObList, CCurrentRename*> CCurrentRenSortList;
typedef CTypedPtrList<CObList, CCurrentRename*> CCurrentRenOperList;
```

คลาส **CCurrentRename** คือ โครงสร้างข้อมูลสำหรับการเก็บชนิดข้อมูลหรือการดำเนินการของมอดูลที่นำเข้า ที่มีการเปลี่ยนชื่อ 1 รายการ โดยเป็นการขยายโครงสร้างของ คลาส CRename ที่มีการเพิ่มสมาชิกเข้าไป 2 ตัวคือ

- ชื่อของพารามิเตอร์มอดูลของ ชนิดข้อมูล หรือ การดำเนินการที่มีการเปลี่ยนชื่อ
- ชื่อของมอดูลที่พารามิเตอร์มอดูลสามารถเปลี่ยนชื่อของ การดำเนินการ หรือ ชนิดข้อมูล ไปเป็นของมอดูลนี้แทนได้



รูปที่ ข-4 แผนภาพคลาสสำหรับเก็บข้อมูลมอดูลนำเข้า 1 มอดูล

โดย CCurrentRename เป็นคลาสพื้นฐานสำหรับ การเก็บข้อมูล 2 ประเภท คือ

- รายการของ ชนิดข้อมูลที่มีการเปลี่ยนชื่อ ซึ่งเก็บโดย **CCurrentRenSortList**
- รายการของ การดำเนินการที่มีการเปลี่ยนชื่อ ซึ่งเก็บโดย **CCurrentRenOperList**

```
class CCurrentImport : public CObject
{
    CString             m_strModuleName;   // module that imported to current module
    int                 m_nImportMode;     // import mode
    CParameterList      m_parmViewList;    // if parameter view to another module
    CCurrentRenSortList     m_renSortList;   // renamed sort
    CCurrentRenOperList     m_renOperationList;    // renamed operation
};
typedef CTypedPtrList<CObList, CCurrentImport*> CCurrentImportList;
```

คลาส **CCurrentImport** คือ โครงสร้างข้อมูลสำหรับการเก็บข้อกำหนดของมอดูลนำเข้า 1 มอดูล ซึ่งประกอบไปด้วย

- ชื่อมอดูลนำเข้า
- รายการของพารามิเตอร์
- รายการของชนิดข้อมูลที่มีการเปลี่ยนชื่อ
- รายการของการดำเนินการที่มีการเปลี่ยนชื่อ

CCurrentImportList ใช้สำหรับการเก็บรายการของมอดูลนำเข้าทั้งหมด

โดยในส่วนนี้นั้น ทุกๆ มอดูลที่อยู่ใน CCurrentImportList รวมทั้งมอดูลที่นำเข้ามาในทุกลำดับขั้น ที่มีมอดูลเริ่มต้นคือมอดูลใน CCurrentImportList จะถูกดึงมาเพื่อสร้างรายการของ การดำเนินการทั้งหมดที่สามารถนำกลับมาใช้เพื่ออธิบายพฤติกรรมของมอดูลที่กำลังเขียนข้อกำหนดอยู่ โดยการดำเนินการมีลักษณะโครงสร้างข้อมูลดังนี้

```
class COperationEx : public COperation
{
        CString m_strModuleName;
};
typedef CTypedPtrList<CObList,COperationEx*> COperationExList;
```

คลาส **COperationEx** คือ โครงสร้างข้อมูลสำหรับการเก็บการดำเนินการ 1 รายการ โดยเป็นการขยายโครงสร้างของ คลาส COperation ที่มีการเพิ่มสมาชิกเข้าไป 1 ตัวคือ

- ชื่อมอดูล ที่การดำเนินการอยู่

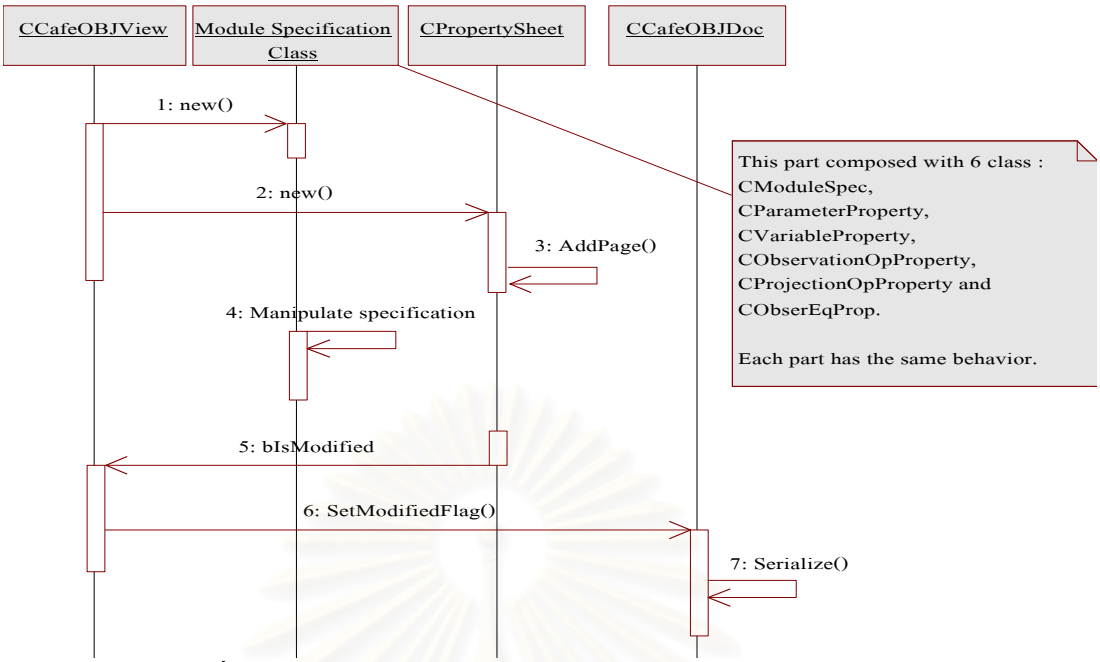
COperationExList ใช้สำหรับเก็บรายการของการดำเนินการทั้งหมดที่สามารถนำกลับมาใช้ในขั้นตอนการกำหนดสัจพจน์ได้

3 ส่วนกำหนดตัวระบุของวัตถุ

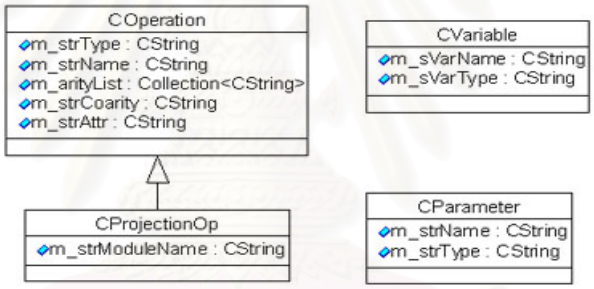
ลำดับการทำงานของส่วนนี้สามารถแสดงด้วยแผนภาพลำดับเหตุการณ์ดังรูปที่ ข-5

โครงสร้างข้อมูลของรายละเอียดทั่วไป แสดงเป็นแผนภาพคลาสดังรูปที่ ข-6 ซึ่งมีรายละเอียดดังนี้

โครงสร้างข้อมูลของพารามิเตอร์ คือ คลาส CParameter และ CParameterList ซึ่งมีลักษณะเช่นเดียวกับโครงสร้างข้อมูลที่กำหนดไว้ในส่วนกำหนดการอ้างอิง



รูปที่ ข-5 แผนภาพลำดับเหตุการณ์ของส่วนกำหนดตัวระบุของวัตถุ



รูปที่ ข-6 แผนภาพคลาสสำหรับเก็บรายละเอียดของตัวระบุของวัตถุ

```
class CVariable : public CObject
{
        CString m_sVarName;
        CString m_sVarType;
};
typedef CTypedPtrList<CObList, CVariable*> CVariableList;
```

คลาส **CVariable** คือ โครงสร้างข้อมูลสำหรับการเก็บตัวแปรของมอดูลที่กำลังเขียนข้อกำหนด ซึ่งประกอบไปด้วย

- ชื่อตัวแปร
- ชนิดตัวแปร

CVariableList ใช้สำหรับเก็บรายการของตัวแปรทั้งหมด

โครงสร้างข้อมูลของการดำเนินการสังเกตค่า คือ คลาส **COperation** และ **COperationList** ซึ่งมีลักษณะเช่นเดียวกับโครงสร้างข้อมูลที่กำหนดไว้ในส่วนกำหนดการอ้างอิง

โครงสร้างข้อมูลของการดำเนินการโปรเจคชัน ซึ่งเป็นการดำเนินการสำหรับการเข้าถึงการดำเนินการของมอดูลนำเข้า ซึ่งมีได้ 1 การดำเนินการต่อ 1 มอดูลนำเข้าที่เป็นโมเดลแบบอิสระ มีลักษณะดังนี้

```
class CProjectionOp : public COperation
{
    CString      m_strModuleName;
};
typedef CTypedPtrList<CObList, CProjectionOp*> CProjectionOpList;
```

คลาส **CProjectionOp** คือ โครงสร้างข้อมูลสำหรับการเก็บการดำเนินการ 1 รายการ โดยเป็นการขยายโครงสร้างของ คลาส COperation ที่มีการเพิ่มสมาชิกเข้าไป 1 ตัวคือ

- ชื่อของมอดูล ที่การดำเนินการโปรเจคชัน สามารถเข้าถึงการดำเนินการภายในมอดูลได้

CProjectionOpList ใช้สำหรับเก็บรายการของการดำเนินการโปรเจคชันทั้งหมด

CStringList* m_equations ใช้สำหรับเก็บรายการของคุณสมบัติการดำเนินการสังเกตค่าทั้งหมด

4 ส่วนกำหนดแผนภาพของวัตถุ

เป็นส่วนที่ใช้สำหรับการกำหนดคุณสมบัติของวัตถุว่ามีลักษณะอย่างไร ซึ่งมีรายละเอียดดังนี้


```
class CDrawObj : public CObject
{
    CString m_sCondition;
    CObArray m_oaEqnInfo;
     CString m_sAction;
    CStringList m_activityList;
```

```
    CString m_sTransEvent;
    .
    .
    .
};
typedef CTypedPtrList<CObList, CDrawObj*> CDrawObjList;
```

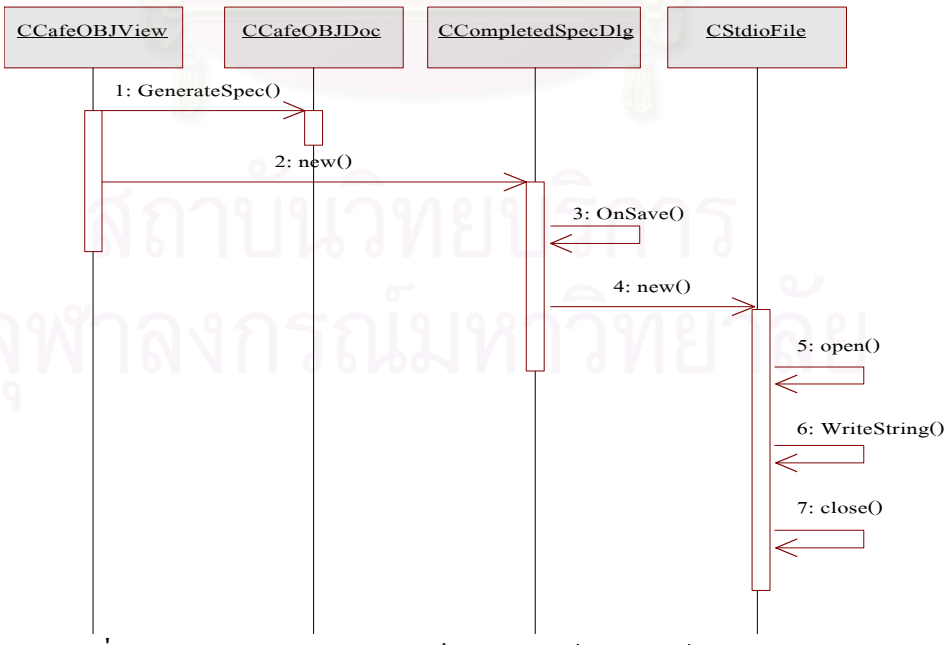
คลาส **CDrawObj** เป็นคลาสสำหรับเก็บข้อมูลของแต่ละคอมโพเนนต์ของแผนภาพสถานะ โดยมีข้อมูลหลักๆ สำหรับการกำหนดข้อกำหนดรูปนัยคาเฟโอบีเจดังนี้

- รายการของกิจกรรม
- รายการของสมการของสัจพจน์
- เหตุการณ์
- เงื่อนไข
- การกระทำ

CDrawObjList ใช้สำหรับเก็บรายการของคอมโพเนนต์ของแผนภาพสถานะทั้งหมด

5 ส่วนสร้างมอดูลข้อกำหนดของวัตถุ

ลำดับการทำงานของส่วนนี้สามารถแสดงด้วยแผนภาพลำดับเหตุการณ์ดังรูปที่ ข-7 โดยเป็นสร้างข้อกำหนดรูปนัยคาเฟโอบีเจจากแผนภาพสถานะที่ได้สร้างขึ้น จากนั้นก็ทำการบันทึกลงแฟ้มข้อมูลถ้าต้องการ



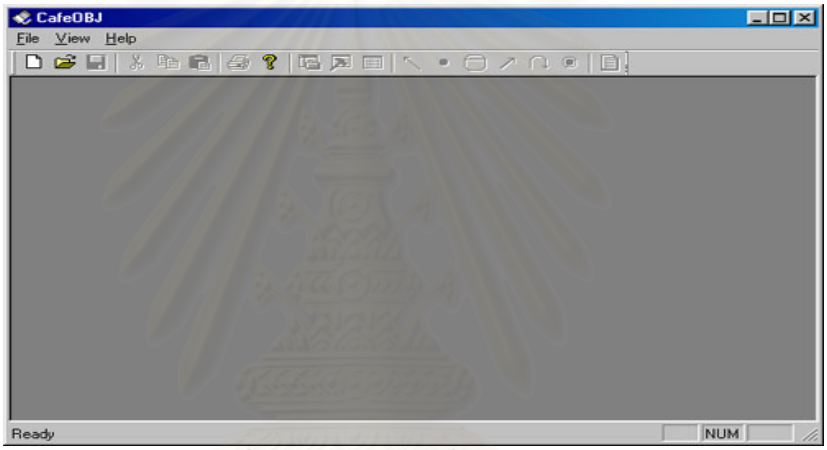
รูปที่ ข-7 แผนภาพลำดับเหตุการณ์ของส่วนสร้างมอดูลข้อกำหนดของวัตถุ

ภาคผนวก ค
คู่มือการใช้งาน

โปรแกรมบรรณาธิกรณ์แผนภาพสถานะ เป็นโปรแกรมสำหรับการเขียนข้อกำหนดรูปนัยคาเฟโอบีเจ ที่กำหนดโดยผ่านการวาดเป็นแผนภาพสถานะ เพื่อให้ผู้ใช้งานสามารถเข้าใจระบบที่กำลังจะพัฒนาได้ดียิ่งขึ้น โดยมีวิธีการใช้โปรแกรมดังนี้

การเริ่มต้นใช้โปรแกรม

เรียกใช้โปรแกรม CafeOBJ.exe เมื่อโปรแกรมพร้อมใช้งาน จะปรากฏหน้าจอดังรูปที่ ค-1



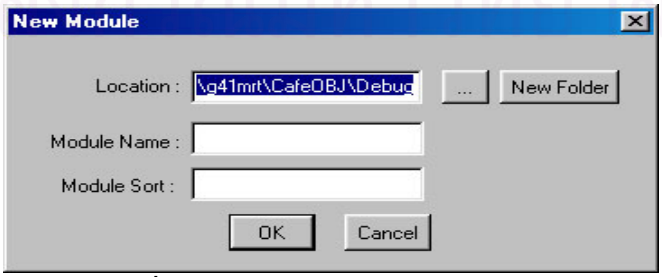
รูปที่ ค-1 หน้าจอการเริ่มต้นโปรแกรม

การสร้างแผนภาพของข้อกำหนดใหม่

ในการสร้างแผนภาพใหม่ ให้เลือกเมนูดังนี้

File → New

จะปรากฏหน้าจอดังรูปที่ ค-2

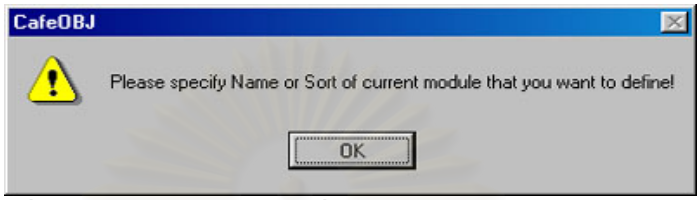


รูปที่ ค-2 หน้าจอการสร้างแผนภาพใหม่

โดยมีส่วนประกอบดังนี้

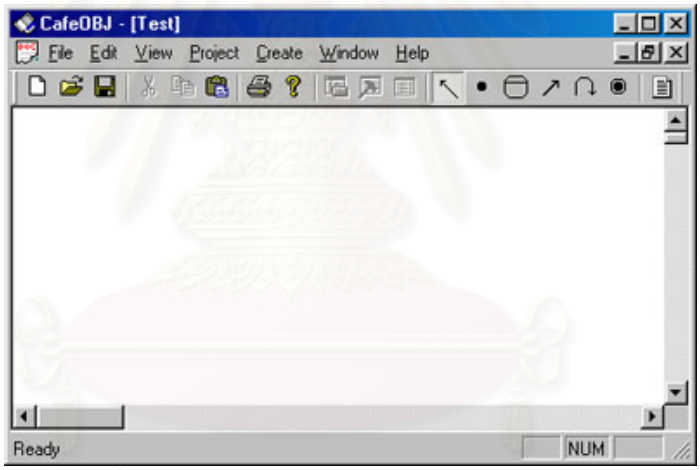
- Location คือ ไดเรคทอรี สำหรับเก็บแฟ้มข้อมูลของแผนภาพ และ ข้อกำหนด
- Module name คือ ชื่อของมอดูล
- Module sort คือ ชื่อของชนิดข้อมูล

ถ้าหากผู้ใช้ไม่ใส่ ชื่อมอดูลหรือชนิดข้อมูล จะมีข้อความเตือนดังรูปที่ ค-3



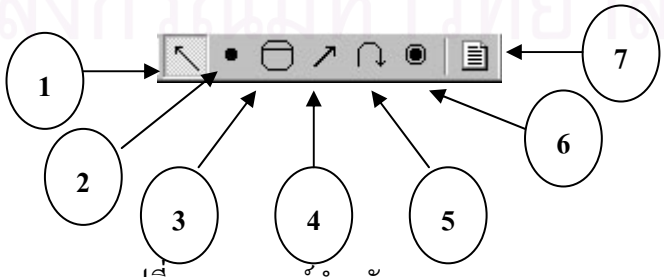
รูปที่ ค-3 ข้อความเตือนให้ใส่ชื่อมอดูล หรือ ชนิดข้อมูลของมอดูล

เมื่อมีการกำหนดทุกอย่างถูกต้องก็กดปุ่ม OK ก็จะปรากฏหน้าจอดังรูปที่ ค-4



รูปที่ ค-4 หน้าจอแรกของการวาดแผนภาพสถานะ

ซึ่งจะมีเมนูให้เลือกใช้ เพื่อวาดส่วนประกอบของแผนภาพดังรูปที่ ค-5



รูปที่ ค-5 เมนูบาร์สำหรับวาดแผนภาพ

โดยที่ แต่ละเมนู มีหน้าที่ต่างๆ ดังนี้

1 สำหรับเลือกส่วนประกอบของแผนภาพ

2 สำหรับวาดสถานะเริ่มต้น

3 สำหรับวาดสถานะ

4 สำหรับวาดการเปลี่ยนสถานะ

5 สำหรับวาดการเปลี่ยนสถานะภายในสถานะเดิม

6 สำหรับวาดสถานะสิ้นสุด

7 สำหรับสร้างข้อกำหนด

เมื่อได้หน้าจอดังรูปที่ ค-4 เรียบร้อยแล้วก็ทำการกำหนดข้อกำหนดของมอดูล โดยมีลำดับดังนี้

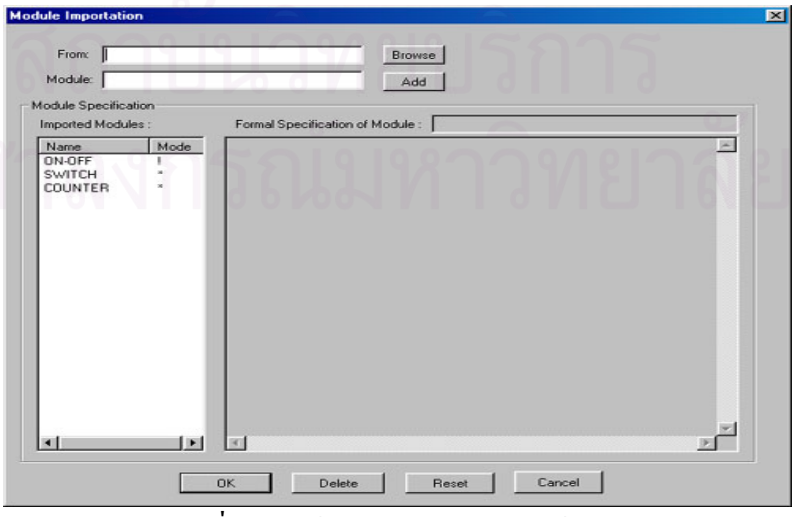
- การเตรียมมอดูลอ้างอิง
- การนำเข้ามอดูล
- การกำหนดรายละเอียดทั่วไปของมอดูล
- การวาดแผนภาพและกำหนดคุณสมบัติของแต่ละส่วนประกอบที่จำเป็น
- การสร้างข้อกำหนดจากแผนภาพ

การเตรียมมอดูลอ้างอิง

ในการเตรียมมอดูลอ้างอิง ให้เลือกเมนู ดังนี้

Project → Insert Module

จะปรากฏหน้าจอดังรูปที่ ค-6



รูปที่ ค-6 หน้าจอการเตรียมมอดูลอ้างอิง

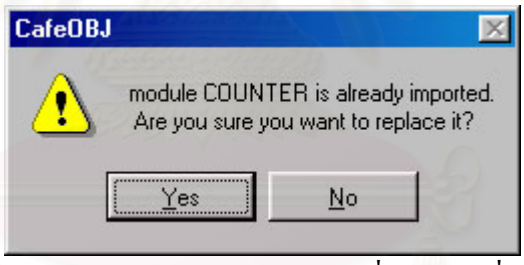
โดยหน้าจอประกอบด้วย

- From เพื่อกำหนดแฟ้มข้อมูลที่อยู่ของมอดูล
- Module Name เพื่อกำหนดชื่อมอดูลสุดท้ายที่อยู่ในแฟ้มข้อมูลที่จะเตรียมนำเข้าเพื่ออ้างอิง
- Imported Module คือ มอดูลที่ได้ทำการนำเข้าแล้วเพื่อการอ้างอิงต่อไป
- Module Specification คือ ข้อกำหนดของมอดูลที่เลือกจาก Imported Module

และมีปุ่มต่างๆ ดังนี้

- OK ทำการบันทึกการเปลี่ยนแปลงที่เกิดขึ้น
- Delete ทำการลบมอดูลที่เลือกจากมอดูลเตรียมนำเข้าเพื่ออ้างอิง
- Reset ทำการล้างข้อมูลในช่องข้อความ From ช่อง Module Name และ ช่อง Module Specification
- Cancelทำการปิดหน้าจอโดยไม่สนใจการเปลี่ยนแปลงที่เกิดขึ้น

โดยที่ แต่ละมอดูลจะอยู่ในแฟ้มข้อมูลเดียวกัน หรือแยกเป็นหลายๆ แฟ้มข้อมูลก็ได้ ซึ่งถ้าหากชื่อของมอดูลนำเข้าซ้ำกัน ระบบจะทำการเตือนให้ทราบว่ามีมอดูลนั้นๆ อยู่แล้ว เพื่อให้ทำการเลือกว่าจะเก็บมอดูลที่ได้เตรียมไว้แล้ว หรือจะแทนที่มอดูลเก่าด้วยมอดูลใหม่ ดังรูปที่ ค-7



รูปที่ ค-7 หน้าจอแสดงข้อความเตือน เมื่อมอดูลมีชื่อซ้ำกัน

การนำเข้ามอดูล

เมื่อมีการเตรียมมอดูลเพื่ออ้างอิงเรียบร้อยแล้ว ก็ทำการนำเข้ามอดูล โดยเลือกเมนูดังนี้

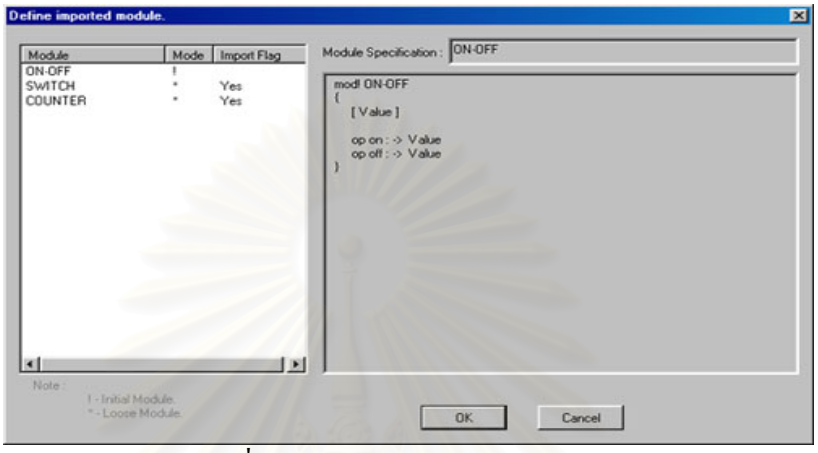
Project → Import Module

จะปรากฏหน้าจอดังรูปที่ ค-8

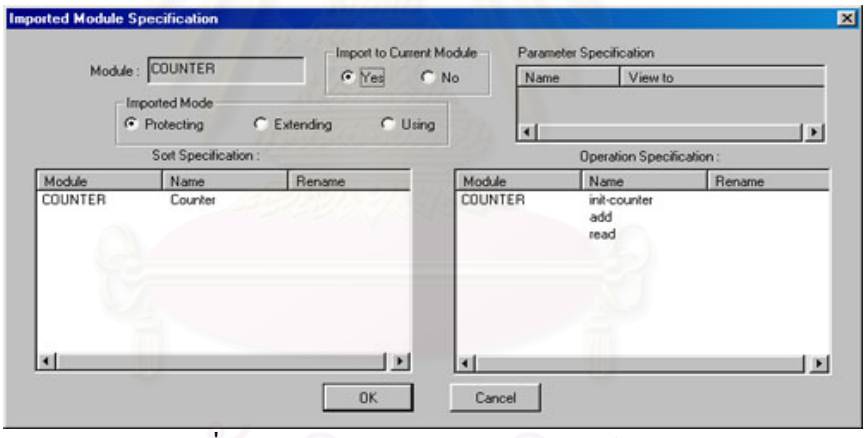
วิธีการกำหนดการนำเข้า สามารถทำได้โดยการกดเมาส์ซ้าย 2 ครั้ง ก็จะได้หน้าจอดังรูปที่ ค-9 ซึ่งใช้สำหรับการกำหนดรายละเอียดการนำเข้ามอดูลต่างๆ ซึ่งคุณสมบัติต่างๆ ที่สามารถกำหนดได้คือ

- การกำหนดโหมดในการนำเข้าว่าเป็น โหมดอะไร ซึ่งมีให้เลือก 3 โหมด คือ โหมดป้องกัน (Protecting) โหมดขยาย (Extending) และ โหมดการใช้ (Using)

- การกำหนดการแทนที่ (Instantiation) ของ มอดูลพารามิเตอร์ ในกรณีที่มอดูลที่นำเข้ามีพารามิเตอร์ด้วย
- การกำหนดการเปลี่ยนชื่อชนิดข้อมูล
- การกำหนดการเปลี่ยนชื่อการดำเนินการ

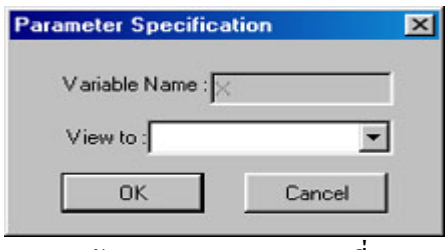


รูปที่ ค-8 การกำหนดการนำเข้ามอดูล



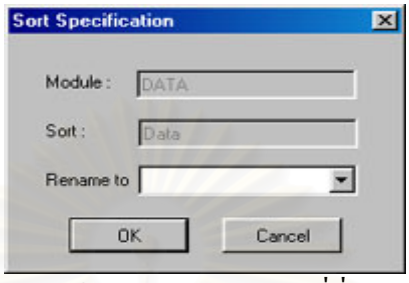
รูปที่ ค-9 การกำหนดคุณสมบัติการนำเข้ามอดูล

การกำหนดการแทนที่สามารถกำหนดโดยการกดเมาส์ซ้ายที่ตัวแปรที่ต้องการแทนที่ 2 ครั้ง จะปรากฏหน้าจอดังรูปที่ ค-10 เพื่อทำการเลือกว่าจะแทนที่มอดูลพารามิเตอร์ด้วยมอดูลใด



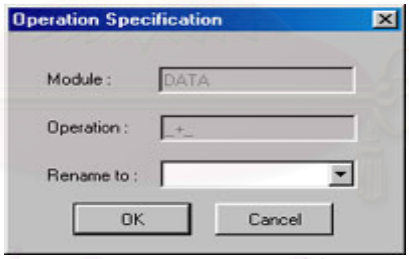
รูปที่ ค-10 หน้าจอสำหรับการกำหนดการแทนที่ของมอดูลพารามิเตอร์

การกำหนดการเปลี่ยนชื่อชนิดข้อมูลสามารถกำหนดได้โดยการกดเมาส์ขวาที่ชนิดข้อมูลที่ต้องการเปลี่ยนชื่อ 2 ครั้ง จะปรากฏหน้าจอดังรูปที่ ค-11 เพื่อทำการเลือกว่าจะเปลี่ยนชื่อชนิดข้อมูลเป็นชื่อใด ซึ่งหากชนิดข้อมูลที่เลือกเป็นของ มอดูลพารามิเตอร์ ชนิดข้อมูลที่สามารถเป็นได้นั้นต้องเป็นชนิดข้อมูลของ “view to” เท่านั้น ซึ่งจะแสดงให้เลือกชนิดข้อมูลที่ต้องการได้เลย



รูปที่ ค-11 หน้าจอสำหรับการแทนที่ชื่อชนิดข้อมูล

การกำหนดการเปลี่ยนชื่อการดำเนินการ สามารถกำหนดได้โดยการกดเมาส์ซ้ายที่การดำเนินการที่ต้องการเปลี่ยนชื่อ 2 ครั้ง จะปรากฏหน้าจอดังรูปที่ ค-12 เพื่อทำการเลือกว่าจะเปลี่ยนชื่อการดำเนินการเป็นชื่อใด ซึ่งหากการดำเนินการที่เลือกเป็นของ มอดูลพารามิเตอร์ การดำเนินการที่สามารถเป็นได้นั้น ต้องเป็นการดำเนินการของ “view to” เท่านั้น ซึ่งจะแสดงให้เลือกชนิดข้อมูลที่ต้องการได้เลย



รูปที่ ค-12 หน้าจอสำหรับการแทนที่ชื่อการดำเนินการ

การกำหนดตัวระบุของวัตถุ

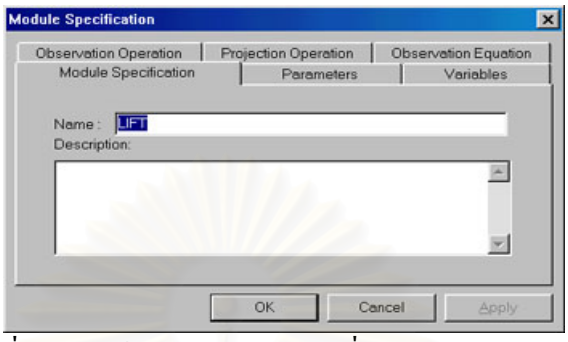
หลังจากกำหนดมอดูลนำเข้าแล้วก็ทำการกำหนดรายละเอียดทั่วไปของมอดูล โดยเลือกเมนูดังนี้

Project → Module Specification

จะปรากฏหน้าจอดังรูปที่ ค-13 ซึ่งเป็นหน้าจอสำหรับกำหนดชื่อมอดูลและคำอธิบาย และมีลักษณะเป็นแท็บสำหรับเลือกเพื่อกำหนดรายละเอียดอื่นๆ ดังต่อไปนี้

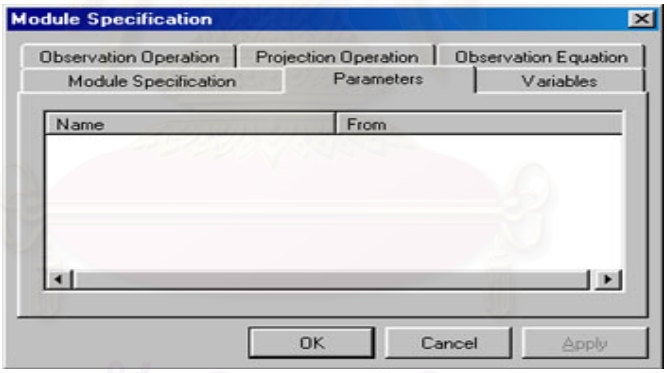
- กำหนดพารามิเตอร์ ถ้ามี
- กำหนดตัวแปรทั้งหมดของมอดูล ถ้ามี

- กำหนดการดำเนินการสังเกตค่า ถ้ามี
- กำหนดการดำเนินการ โปรเจคชัน ถ้ามี
- กำหนดสัจพจน์ของการดำเนินการสังเกตค่า ถ้ามีการดำเนินการสังเกตค่า



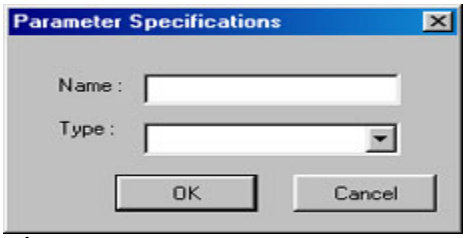
รูปที่ ค-13 หน้าจอสำหรับกำหนดชื่อมอดูลและคำอธิบาย

1) การกำหนดพารามิเตอร์ของมอดูล สามารถทำได้โดยคลิกเมาส์ซ้ายที่แท็บ Parameter จะปรากฏหน้าจอดังรูปที่ ค-14 ซึ่งแสดงรายการของพารามิเตอร์ทั้งหมดที่ได้กำหนดไว้แล้ว



รูปที่ ค-14 หน้าจอแสดงรายการพารามิเตอร์

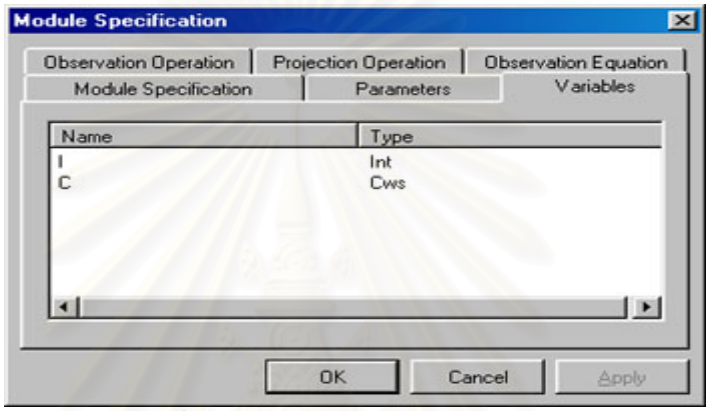
- การเพิ่มพารามิเตอร์ ทำได้โดยกดเมาส์ขวาจะปรากฏเมนูให้เลือก ก็เลือก Insert จะปรากฏหน้าจอดังรูปที่ ค-15 เพื่อกำหนดชื่อ และชนิดของพารามิเตอร์ที่ต้องการเพิ่ม



รูปที่ ค-15 หน้าจอสำหรับกำหนดพารามิเตอร์

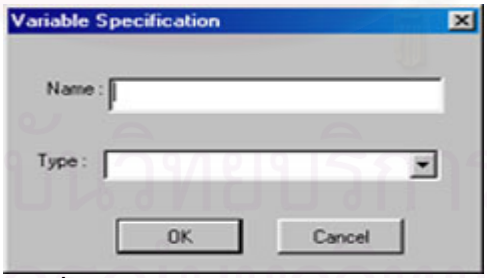
- การแก้ไขพารามิเตอร์ ทำได้โดยกดเมาส์ขวาจะปรากฏเมนูให้เลือก ก็เลือก Edit จะปรากฏหน้าจอดังรูปที่ ค-15 เพื่อทำการเปลี่ยนชื่อ หรือชนิดของพารามิเตอร์
- การลบพารามิเตอร์ ทำได้โดยกดเมาส์ขวาจะปรากฏเมนูให้เลือก ก็เลือก Delete รายการของพารามิเตอร์ที่เลือกจะถูกลบออกจากหน้าจอที่แสดง

2) การกำหนดตัวแปรของมอดูล สามารถทำได้โดยคลิกเมาส์ซ้ายที่แท็บ Variable จะปรากฏหน้าจอดังรูปที่ ค-16 ซึ่งแสดงรายการของตัวแปรทั้งหมดที่ได้กำหนดไว้แล้ว



รูปที่ ค-16 หน้าจอแสดงรายการตัวแปร

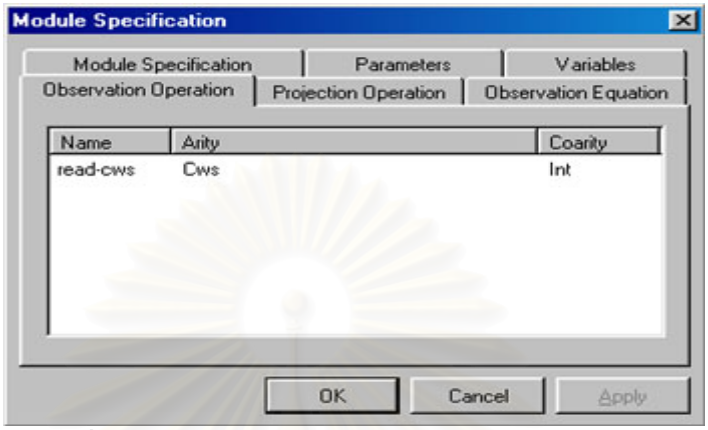
- การเพิ่มตัวแปร ทำได้โดยกดเมาส์ขวาจะปรากฏเมนูให้เลือก ก็เลือก Insert จะปรากฏหน้าจอดังรูปที่ ค-17 เพื่อกำหนดชื่อ และชนิดข้อมูลของตัวแปรที่ต้องการเพิ่ม



รูปที่ ค-17 หน้าจอสำหรับกำหนดตัวแปร

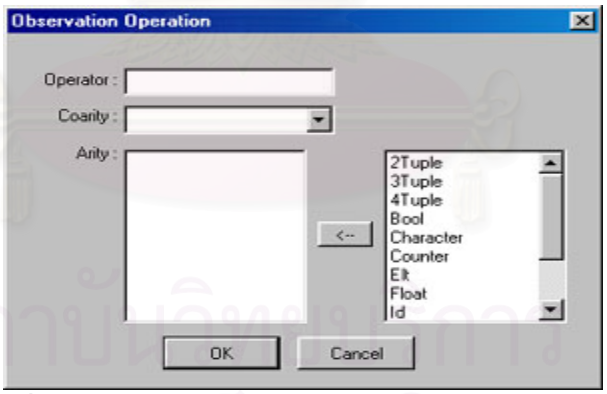
- การแก้ไขตัวแปร ทำได้โดยกดเมาส์ขวาจะปรากฏเมนูให้เลือก ก็เลือก Edit จะปรากฏหน้าจอดังรูปที่ ค-17 เพื่อทำการเปลี่ยนชื่อ หรือชนิดข้อมูลของตัวแปร
- การลบตัวแปร ทำได้โดยกดเมาส์ขวาจะปรากฏเมนูให้เลือก ก็เลือก Delete รายการของตัวแปรที่เลือกจะถูกลบออกจากหน้าจอที่แสดง

3) การกำหนดการดำเนินการสังเกตค่า สามารถทำได้โดยคลิกเมาส์ซ้ายที่แท็บ Observation Operation จะปรากฏหน้าจอดังรูปที่ ค-18 ซึ่งแสดงรายการของการดำเนินการสังเกตค่าทั้งหมดที่ได้กำหนดไว้แล้ว



รูปที่ ค-18 หน้าจอแสดงรายการการดำเนินการสังเกตค่า

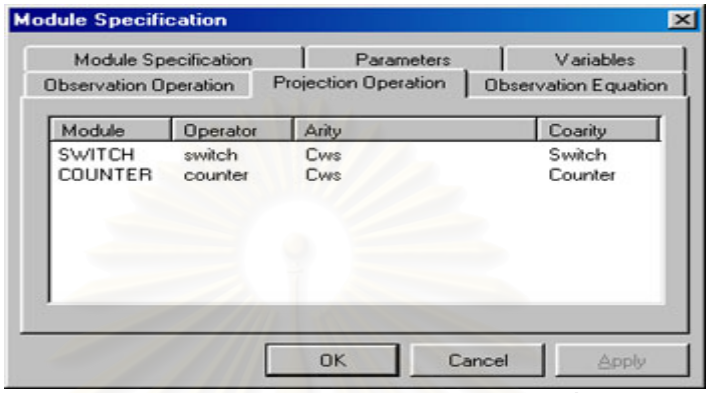
- การเพิ่มการดำเนินการสังเกตค่า ทำได้โดยกดเมาส์ขวาจะปรากฏเมนูให้เลือก ก็เลือก Insert จะปรากฏหน้าจอดังรูปที่ ค-19 เพื่อกำหนดชื่อ ชนิดข้อมูลของอาร์กิวเมนต์ และชนิดข้อมูลที่เป็นผลลัพธ์ของการดำเนินการสังเกตค่าที่ต้องการเพิ่ม



รูปที่ ค-19 หน้าจอสำหรับกำหนดการดำเนินการสังเกตค่า

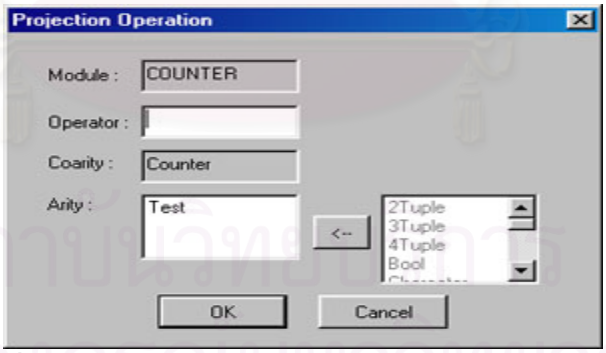
- การแก้ไขการดำเนินการสังเกตค่า ทำได้โดยกดเมาส์ขวาจะปรากฏเมนูให้เลือก ก็เลือก Edit จะปรากฏหน้าจอดังรูปที่ ค-19 เพื่อทำการเปลี่ยนชื่อ ชนิดข้อมูลของอาร์กิวเมนต์ หรือ ชนิดข้อมูลของผลลัพธ์
- การลบการดำเนินการสังเกตค่า ทำได้โดยกดเมาส์ขวาจะปรากฏเมนูให้เลือก ก็เลือก Delete รายการของการดำเนินการที่เลือกจะถูกลบออกจากหน้าจอที่แสดง

4) การกำหนดการดำเนินการโปรเจคชัน สามารถทำได้โดยคลิกเมาส์ซ้ายที่แท็บ Projection Operation จะปรากฏหน้าจอดังรูปที่ ค-20 ซึ่งแสดงรายการของการดำเนินการโปรเจคชันที่สามารถมีได้ทั้งหมด ซึ่งการดำเนินการประเภทนี้ไม่สามารถเพิ่มจากรายการที่แสดงได้ โดยในสดมภ์แรกคือโมดูลที่สามารถกำหนดการดำเนินการโปรเจคชันได้



รูปที่ ค-20 หน้าจอแสดงรายการการดำเนินการโปรเจคชันที่สามารถกำหนดได้

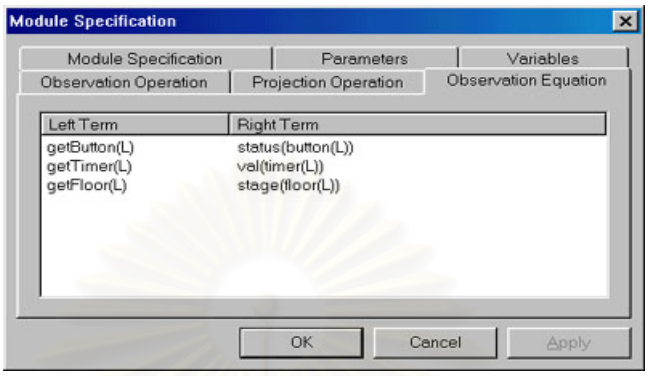
- การแก้ไขการดำเนินการโปรเจคชัน ทำได้โดยกดเมาส์ขวาจะปรากฏเมนูให้เลือก ก็เลือก Edit จะปรากฏหน้าจอดังรูปที่ ค-21 เพื่อทำการเปลี่ยนชื่อ หรือชนิดข้อมูลของอาร์กิวเมนต์ของการดำเนินการโปรเจคชัน โดยสามารถเพิ่มชนิดข้อมูลของอาร์กิวเมนต์ได้ก็ต่อเมื่อ มอดูลที่นำเข้าเป็นระบบพลวัต



รูปที่ ค-21 หน้าจอสำหรับกำหนดการดำเนินการโปรเจคชัน

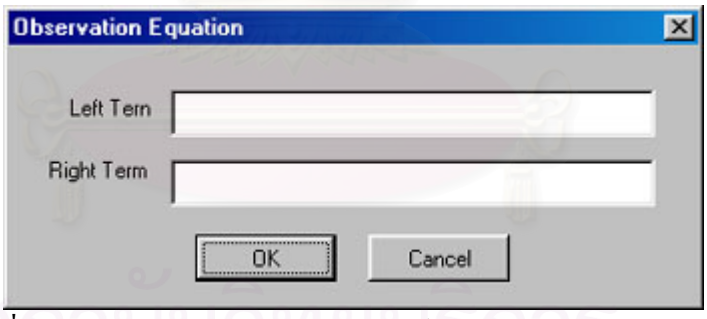
- การลบการดำเนินการโปรเจคชัน ทำได้โดยกดเมาส์ขวาจะปรากฏเมนูให้เลือก ก็เลือก Delete รายการของการดำเนินการที่เลือกจะถูกลบออกจากหน้าจอที่แสดง ยกเว้นสดมภ์แรกที่จะไม่ถูกลบออกจากหน้าจอ

5) การกำหนดสัจพจน์ของการดำเนินการสังเกตค่า สามารถทำได้โดยคลิกเมาส์ซ้ายที่แท็บ Observation Equation จะปรากฏหน้าจอดังรูปที่ ค-22 ซึ่งแสดงรายการสัจพจน์ของการดำเนินการสังเกตค่าทั้งหมดที่ได้กำหนดไว้แล้ว



รูปที่ ค-22 หน้าจอแสดงรายการสัจพจน์ของการดำเนินการสังเกตค่า

- การเพิ่มสัจพจน์ของการดำเนินการสังเกตค่า ทำได้โดยกดเมาส์ขวาจะปรากฏเมนูให้เลือก ก็เลือก Insert จะปรากฏหน้าจอดังรูปที่ ค-23 เพื่อกำหนดสมการของสัจพจน์ในรูปของเทอมทางด้านซ้ายมือ และเทอมทางด้านขวามือ โดยเทอมคือ ค่าคงที่ การดำเนินการ หรือการรวมกันของค่าคงที่และการดำเนินการ



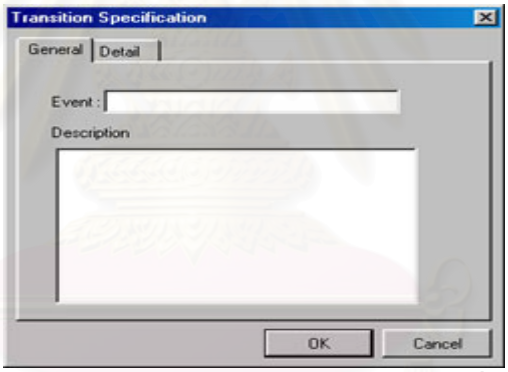
รูปที่ ค-23 หน้าจอสำหรับกำหนดสัจพจน์ของการดำเนินการสังเกตค่า

- การแก้ไขสัจพจน์ของดำเนินการสังเกตค่า ทำได้โดยกดเมาส์ขวาจะปรากฏเมนูให้เลือก ก็เลือก Edit จะปรากฏหน้าจอดังรูปที่ ค-23 เพื่อทำการเปลี่ยนสมการของสัจพจน์ที่เลือก
- การลบสัจพจน์ของการดำเนินการสังเกตค่า ทำได้โดยกดเมาส์ขวาจะปรากฏเมนูให้เลือก ก็เลือก Delete รายการของสัจพจน์ที่เลือกจะถูกลบออกจากหน้าจอที่แสดง

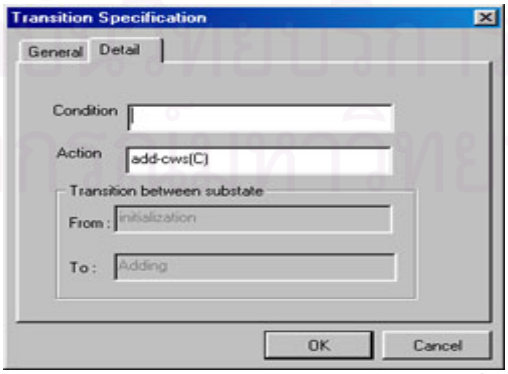
การวาดแผนภาพและกำหนดคุณสมบัติของแต่ละส่วนประกอบ

เมื่อมีการกำหนดรายละเอียดต่างๆ เรียบร้อยแล้วก็ทำการวาดแผนภาพสถานะ เพื่อกำหนดพฤติกรรมของมอดูล โดยเลือกชนิดของส่วนประกอบที่จะวาดจากเมนูบาร์ ในรูปที่ ค-5 หรือ เลือกจากเมนูย่อยของ Create

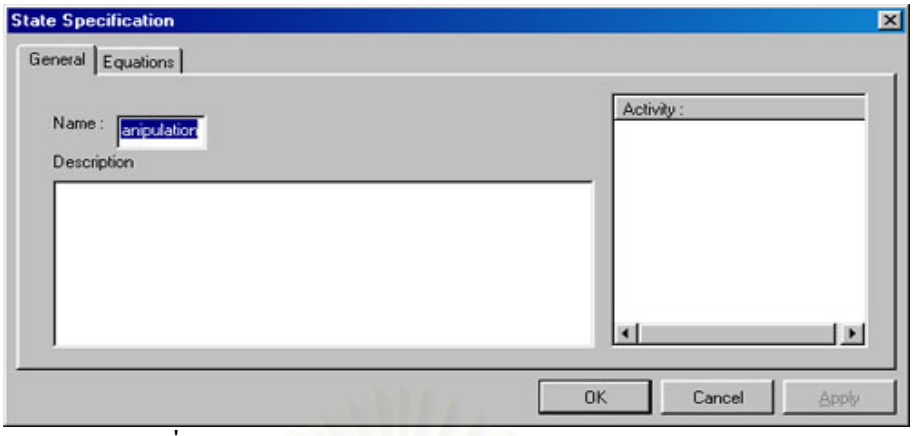
เมื่อเลือกแล้วก็กดตัวชี้ลงยังตำแหน่งที่ต้องการ และทำการกำหนดชื่อของแต่ละสถานะ จากนั้นก็ทำการวาดการเปลี่ยนสถานะ เมื่อต้องการที่จะกำหนดคุณสมบัติของแต่ละส่วนประกอบ ก็สามารถทำได้โดยการกดเมาส์ซ้าย 2 ครั้ง หรือ กดเมาส์ขวา ณ ส่วนประกอบที่ต้องการ จะปรากฏเมนูให้เลือก ซึ่งมีให้เลือกได้ 4 อย่างคือ copy paste delete และ properties ก็ให้เลือก properties โดย properties ของ การเปลี่ยนสถานะจะเป็นดังรูปที่ ค-24 สำหรับกำหนดเหตุการณ์ที่จะทำให้เกิดการเปลี่ยนสถานะพร้อมคำอธิบายทั่วๆ ไป และ ค-25 สำหรับกำหนดเงื่อนไขของการเปลี่ยนสถานะและการกระทำที่จะถูกประมวลผล ส่วนของ สถานะ จะเป็นดังรูปที่ ค-26 สำหรับกำหนดชื่อของสถานะ กิจกรรมภายในสถานะ และคำอธิบายทั่วๆ ไป และ ค-27 สำหรับกำหนดสมการสัจพจน์ของการดำเนินการที่เหมาะสม



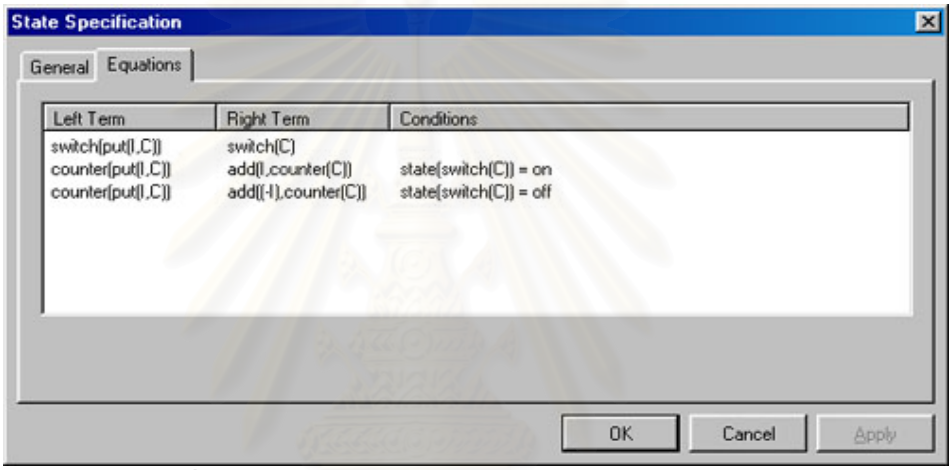
รูปที่ ค-24 หน้าจอสำหรับกำหนดคุณสมบัติของการเปลี่ยนสถานะ (1)



รูปที่ ค-25 หน้าจอสำหรับกำหนดคุณสมบัติของการเปลี่ยนสถานะ (2)



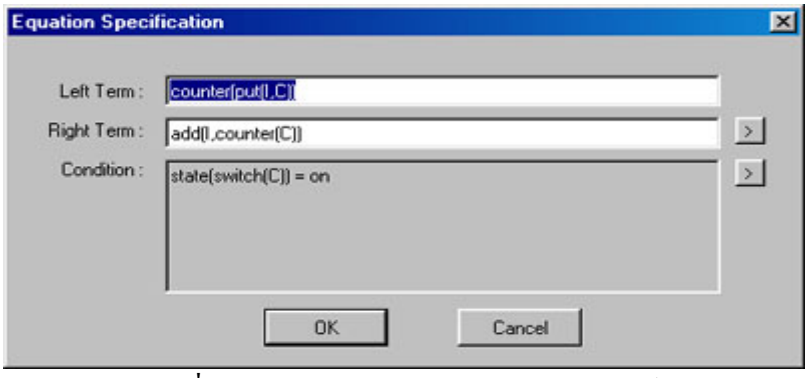
รูปที่ ค-26 หน้าจอสำหรับกำหนดคุณสมบัติของสถานะ (1)



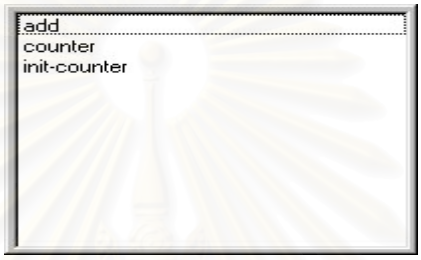
รูปที่ ค-27 หน้าจอสำหรับกำหนดคุณสมบัติของสถานะ (2)

การเพิ่ม การแก้ไข และการลบสัจพจน์ สามารถกระทำได้ดังนี้

- การเพิ่มสัจพจน์ ทำได้โดยกดเมาส์ขวาจะปรากฏเมนูให้เลือก ก็เลือก Insert จะปรากฏหน้าจอดังรูปที่ ค-28 เพื่อกำหนดสมการของสัจพจน์ในรูปของเทอมทางซ้าย และเทอมทางขวาของเครื่องหมายเท่ากับ โดยเมื่อมีการกดปุ่มหลังกล่องข้อความของเทอมด้านขวา (Right Term) จะเป็นหน้าจอสำหรับเลือก การดำเนินการนอกสุด (Outer most operator) ที่สามารถนำมาเขียนเป็นสมการได้ โดยไม่ผิดวากยะสัมพันธ์ของภาษาคาเฟโอบีเจ ดังรูปที่ ค-29 และเมื่อกดปุ่มหลังกล่องข้อความของเงื่อนไข (Condition) จะเป็นหน้าจอสำหรับการเลือกเงื่อนไขที่สามารถเป็นส่วนหนึ่งของสมการสัจพจน์ได้ ดังรูปที่ ค-30



รูปที่ ค-28 หน้าจอสำหรับการกำหนดสัจพจน์



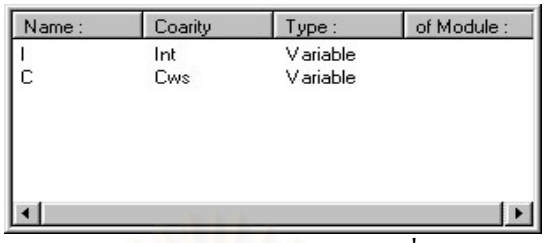
รูปที่ ค-29 หน้าจอสำหรับการเลือกการดำเนินการที่เหมาะสม

รูปที่ ค-30 หน้าจอสำหรับการเลือกเงื่อนไขที่สามารถใช้ได้

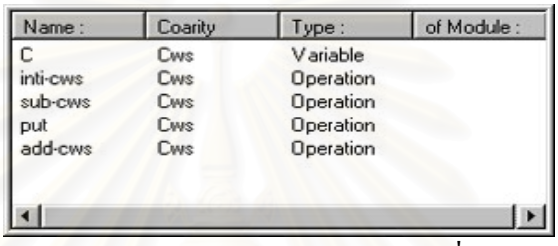
- การแก้ไขสัจพจน์ ทำได้โดยกดเมาส์ขวาจะปรากฏเมนูให้เลือก ก็เลือก Edit จะปรากฏหน้าจอดังรูปที่ ค-28 เพื่อทำการเปลี่ยนสัจพจน์ที่เลือก
- การลบสัจพจน์ ทำได้โดยกดเมาส์ขวาจะปรากฏเมนูให้เลือก ก็เลือก Delete รายการของการดำเนินการที่เลือกจะถูกลบออกจากหน้าจอที่แสดง

นอกจากนี้ในการกำหนดเหตุการณ์ การกระทำ และกิจกรรม จะเป็นการกำหนดรูปแบบของการดำเนินการ ที่ระบุชื่อ และพารามิเตอร์ของการดำเนินการ โดยพารามิเตอร์จะเป็นการนำตัวแปรมาเป็นตัวกำหนด ซึ่งตัวแปรที่สามารถใช้ได้จะต้องผ่านการกำหนดไว้แล้วในขั้นตอนของการกำหนดรายละเอียดของมอดูล ซึ่งระบบจะแสดงรายการของตัวแปรที่สามารถนำมากำหนดเป็นส่วนหนึ่งของการดำเนินการได้ โดยจะมีลักษณะดังรูปที่ ค-31 ส่วนการกำหนดสัจพจน์นั้น จะแสดง

ตัวแปร หรือการดำเนินการที่เหมาะสม ณ ตำแหน่งที่จะเขียน ดังรูปที่ ค-32 ซึ่งจะปรากฏเมื่อมีการกดปุ่ม “(“ หรือ “,”



รูปที่ ค-31 หน้าจอแสดงรายการของตัวแปรที่สามารถเลือกใช้ได้



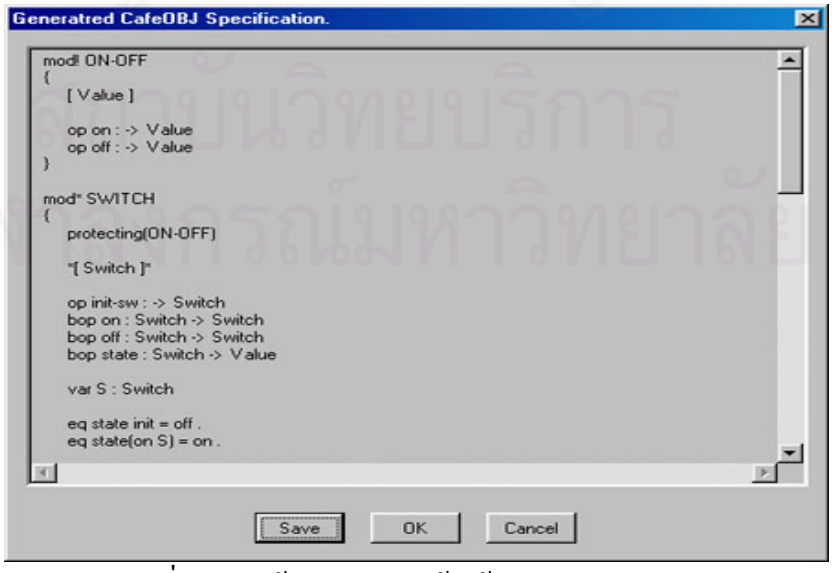
รูปที่ ค-32 รายการของตัวแปรหรือ การดำเนินการที่สามารถเรียกใช้ได้

การสร้างข้อกำหนดจากแผนภาพ

เมื่อทำการกำหนดส่วนประกอบทุกอย่างเรียบร้อยแล้วก็ทำการสร้างข้อกำหนดจากแผนภาพสถานะ โดยเลือกเมนูดังนี้

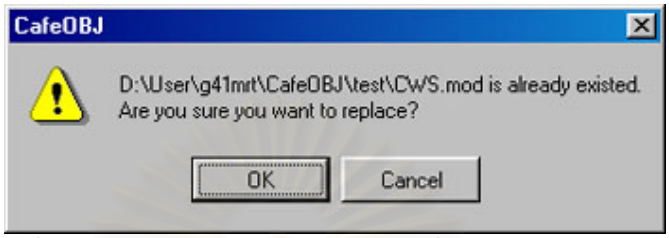
Project → Generate Specification

จะได้หน้าจอดังรูปที่ ค-33



รูปที่ ค-33 หน้าจอสำหรับสร้างข้อกำหนดของมอดูล

หากต้องการบันทึกข้อกำหนดที่ได้ลงแฟ้มข้อมูลก็ทำได้โดยการกดปุ่ม Save โดยการบันทึกข้อมูลนั้น จะบันทึกลงแฟ้มชื่อ เดียวกับชื่อแฟ้มของแผนภาพสถานะ แต่จะมีนามสกุล mod ซึ่งถ้าหากแฟ้มข้อมูลมีอยู่แล้ว จะปรากฏข้อความเตือนว่าต้องการจะบันทึกทับแฟ้มข้อมูลเดิมหรือไม่ ดังรูปที่ ค-34 และถ้าการบันทึกข้อมูลเป็นผลสำเร็จจะปรากฏหน้าจอดังรูปที่ ค-35



รูปที่ ค-34 หน้าจอแสดงข้อความเตือนเมื่อมีแฟ้มข้อมูลอยู่แล้ว

รูปที่ ค-35 หน้าจอแสดงการบันทึกข้อมูลเป็นผลสำเร็จ

ภาคผนวก ง
ผลงานตีพิมพ์

ผลงานวิจัยนี้ได้รับคัดเลือกให้นำเสนอในงานประชุมวิชาการและตีพิมพ์ในเอกสาร Proceedings of the 2nd International Conference on Intelligent Technologies (InTech 2001) ซึ่งได้จัดขึ้นที่ มหาวิทยาลัยอัสสัมชัญ ระหว่างวันที่ 27-29 พฤศจิกายน 2544

A Formal Specification Method from State Diagram

Manop Rattichote[1] and Wiwat Vatanawood[2]
Department of Computer Engineering
Faculty of Engineering, Chulalongkorn University
Bangkok, 10330, Thailand
E-mail: b0463621@student.chula.ac.th[1], wiwat@chula.ac.th[2]

Abstract: This paper proposes a formal specification method by using the transformation rules to map state diagram notations into CafeOBJ specification, a novel successor of OBJ algebraic language. The transformation rules are used as a guideline to consider each component of UML state diagram and provide a corresponding formal definition in CafeOBJ syntax. In our approach, we imply that each state diagram describes the behavior property of an object. Both static system and dynamic system of CafeOBJ are covered. We demonstrate our formal specification method with a case study and the correctness of final specification from our method is proved with CafeOBJ interpreter.
Key words: Formal specification, Transformation rule, State diagram, CafeOBJ language

1. Introduction

Formal specification becomes a popular technique for writing software specification. It is used to describe a software system and its desired properties by using languages with the mathematically defined syntax and semantic. By using a formal specification, developer can gain a deeper understanding on the system. It helps avoid design flaws, inconsistencies, ambiguities, and incompleteness. Nevertheless, the correctness of formal specification can be proved by using mathematics [1,2]. However, the notations used in formal specification are difficult for communicating with end user [3].

Although formal specifications is very useful, no tools are available for practitioner that unfamiliar with mathematical notations to understand and to write a software specification. In this paper, we propose a specification method to map UML state diagram into algebraic specification. CafeOBJ is selected as the algebraic specification language in our approach.

This paper is organized as follows. In Section 2 we present the brief overviews of UML state diagram and CafeOBJ syntax. The related works are reviewed in section 3. In Section 4, we propose a set of transformation rules in order to map state diagram into CafeOBJ. In Section 5, we demonstrate a case study. Section 6 is a conclusion.

2. Overview

2.1. State diagram

State diagram is included in UML [4] to describe the dynamic behavior of an object during its lifetime. A state diagram may be attached to the definitions of classes, use cases, or entire systems in order to visualize, specify, construct, and document the dynamic property of an individual object. In our approach, a state diagram is considered as the set of events, conditions, and actions on each transition lines and activities in each state. Figure 1 shown the state diagram of a switch, an example case study in [5]. The switch has 2 states – on and off, and 5 transition lines.

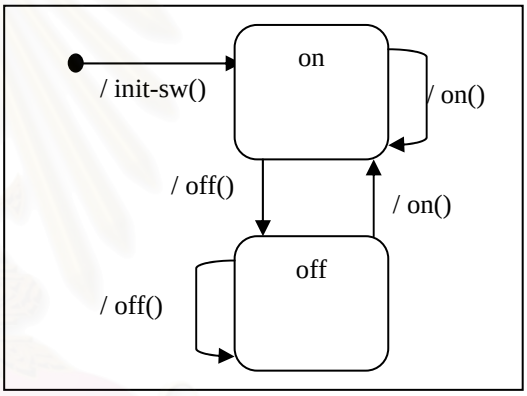


Figure 1: state diagram of SWITCH object

2.2. CafeOBJ [5, 6]

CafeOBJ is an algebraic specification language that is a direct successor of OBJ [5]. When it comes to write a CafeOBJ specification, three main parts: sort declaration, signature declaration, and axiom declaration, are very important. In sort declaration part, the definitions of all sorts (types of objects) are declared. Since CafeOBJ supports modular specification technique, any related object specification module can be imported. All of imported modules are defined in sort declaration part as well. Signature declaration part defines all of the possible operations in the modules in terms of sorts of inputs (arity) and sort of output (coarity). The axiom declaration part is used to define

behavior properties of each operation that appears in signature declaration part. Figure 2 shows the sample of CafeOBJ specification of *switch* object, an example case study in [5]. A module named *"ON-OFF"*, represents the two possible values of switch *'on'* and *'off'* and will be imported by module named *"SWITCH"*. A module named "*SWITCH*" represents a switch object. Its sort declaration part declares the imported *'ON-OFF'* module and a hidden sort named *Switch* is declared. The signature declaration part defines a set of operations: *init-sw, on, off, status*, and their sorts of input and output parameters. In the axiom declaration part, all of the equations are defined to represent the possible reactions of the *Switch*.

```
module! ON_OFF{
              [Value]

              op on  : → Value
              op off : → Value
}

module* SWITCH{
sort          pr(ON-OFF)
declaration
              *[Switch]*

              op init-sw : → Switch
signature     bop on : Switch → Switch
declaration   bop off : Switch → Switch
              bop status : Switch → Value

              var S : Switch

axoim         eq status(init-sw) = on .
declaration   eq status(on(S)) = on .
              eq status(off(S)) = off .
}
```

Figure 2: CafeOBJ specification of *SWITCH* module

3. Related Work

In this section, we review several related researches on formal specification method using diagrams such as UML class diagram, state diagram, and object diagram etc. as follows.

In [7], Liliana Favre, and Sivia Clerici proposed an approach to integrate UML static models to algebraic specification style. They map class diagram of UML into a set of $GSBL^{00}$ language and demonstrate how to describe the semantics of entire system by using class diagrams to express the relations among classes and semi-automatically generate object-oriented codes from UML model. However, the behavior property of each object is excluded in their approach.

In [2], T.H. Tse proposed a set of graphical interface for Functional Object-Oriented Programming System (FOOPS), a formal object-oriented specification language with algebraic semantics. The algebraic declaration of classes, methods, attributes, and inheritances are specified in the notions of data flow diagram, state diagram, and object diagram.

In [3], Scott A. DeLoach proposed the approach to translate from Rumbaugh's Object Modeling Technique (OMT) to algebraic specification. This model defines how object-oriented concepts are represented algebraically using an object-oriented algebraic specification language O-SLANG (O-SLANG combines basic algebraic specification constructs with category theory operations to capture internal object class structure as well as relationships between classes) and formal transformations from OMT to O-SLANG algebraic specification.

In [8], Shusaku Iida, Kokichi Futatsugi, and Razvan Diaconescu proposed a method to specify component-based algebraic specification by using class diagram of UML. This approach represents each class in UML with a corresponding hidden sort. Relationships among classes are the projection operation of the compound object. However, UML class diagram provides only the static behavior of the system.

In this research, we proposed a method to specify dynamic behavior of the system.

4. State Diagram-to-CafeOBJ Transformation rules

It is conventional mean to begin the software specification method with some common used diagrams. Then, a set of formal specifications can be derived. In our approach, we propose a formal specification method to transform each given state diagram into a corresponding CafeOBJ specification. By using our transformation rules as a guideline, a practitioner can write CafeOBJ specification to describe both static and dynamic properties of the target system. We especially consider on the specification of the behavior of hidden sort (object).

Rule 1 Each state diagram **S** is formally defined as a CafeOBJ module with a hidden sort **s**.

```
module* S{
        *[s]*
}
```

Where

S is the module name

s is the hidden sort name

We names the module **S** after the state diagram name **S** in order to assume that each state diagram can be considered as an object or module in CafeOBJ.

Rule 2 Initial operation of state diagram can be formally defined as an operation **p** in CafeOBJ.

For static systems, initial operation can be defined as follows:

op **p** : → **s**

For dynamic systems, initial operation may have an identifier as parameter and can be defined as follows:

op **p** : **ID** → **s**

Where

p is the operation name
ID is the identifier
s is the hidden sort name

Rule 3 Each event **e** of state diagram can be formally defined as a behavioral operation in CafeOBJ.

For static systems, the declaration of behavioral operation is

bop **e** : **ls s** → **s**

For dynamic systems, each event may have an identifier as parameter.

bop **e**: **ID ls s** → **s**

Where

e is the event name
ls is the list of parameters which are either visible or hidden sorts
ID is the identifier
s is the hidden sort

Rule 4 Each transition **T** with a condition can be formally defined as a conditional equation in CafeOBJ as follows:

ceq **T** = **T'** if **(c)**

Where

T, **T'** are left-term and right-term in an equation respectively.
c is the guard condition

Rule 5 Each action **a** in state diagram can be formally defined as a behavioral operation in CafeOBJ as follows:

For static systems, the declaration of behavioral operation is

bop **a** : **ls s** → **s**

For dynamic systems, each action may have an identifier as parameter.

bop **a** : **ID ls s** → **s**

Where

a is the action
ID is an identifier
ls is the list of parameters which are either visible or hidden sorts
s is the hidden sort

Rule 6 Each activity **A** in state can be formally defined as a behavioral operation in CafeOBJ as follows:

For static systems, each activity can be formally defined as

bop **A : ls s** → **s**

For dynamic systems, activities may have an identifier as parameter and can be defined as follows:

bop **A** : **ID ls s** → **s**

Where

A is the activity name
ID is the identifier
s is the hidden sort
ls is the list of parameters which are either visible or hidden sorts

Rule 7 For each hidden sort imported to current module. The states of imported hidden sort can be observed by using a projection operation **p,** as follows:

For static system, the declaration of projection operation is

bop **p** : **s** → **s'**

For dynamic system, it must have an identifier as a parameter.

bop **p** : **ID s** → **s'**

Where:

p is the projection operation
s is the hidden sort of current module
s' is the hidden sort of imported module
ID is the identifier

5. Case Study

We demonstrate our proposed formal specification method in this section. By using a case study of lift specification, we firstly draw four state diagrams, which relevant to the LIFT object. Then, the CafeOBJ specification will be written using the proposed transformation rules as a guideline.

5.1 State diagram and CafeOBJ specification of *LIFT* object.

Our *lift* consists of 3 sub-objects - TIMER, BUTTON, and FLOOR. We need to specify these 3 sub-objects firstly as follows:

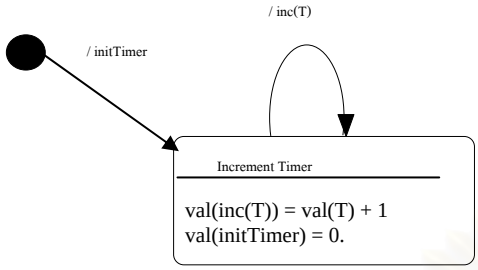
TIMER Object

It is used for time measurement. It counts up continuously. Its state diagram and CafeOBJ syntax are shown in Figure 3.

BUTTON Object

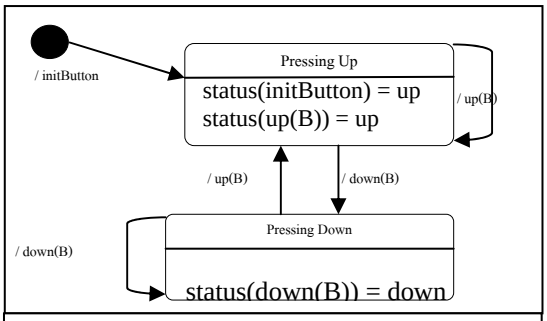
It is a button in front of lift for controlling lift to go up or go down. It has 2 states – up and down.
It will be up when up button is pressed and down when down button is pressed. Its initial value is up. Both state diagram and CafeOBJ of BUTTON are shown in Figure 4.

FLOOR Object
It is the panel in the lift to enable the user to choose the floor he/she wants to go. Its behavior is, to take the user to the chosen floor. Its initial value is 1. Its state diagram and CafeOBJ are shown in Figure 5.




```
module* TIMER{
        pr(NAT)
        *[Timer]*

        op initTimer : → Timer
        bop inc : Timer → Timer
        bop val : Timer → Nat

        var T : Timer

        eq val(initTimer) = 0 .
        eq val(inc(T)) = val(T) + 1 .
}
```

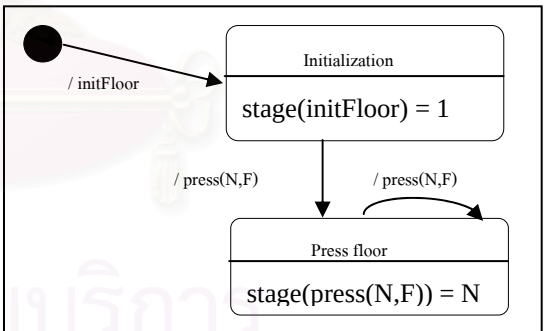
Figure 3: State diagram and CafeOBJ specification of *TIMER* object




```
module! UP-DOWN{
        [Value]

        ops up down : -> Value
}

module* BUTTON{
        pr(UP-DOWN)

        *[Button]*

        op initButton : -> Button
        bop up : Button -> Button
        bop down : Button -> Button
        bop status : Button -> Value

        var B : Button

        eq status(initButton) = up .
        eq status(up(B)) = up .
        eq status(down(B)) = down .
}
```

Figure 4: State diagram and CafeOBJ specification of *BUTTON* object

LIFT Object
When we want to define a lift, we need to know its behavior. A lift in our case study is as follows:
It consists of 3 sub-objects, each object has affect to the behavior of lift. When state of button is changed, lift is ready to go up or go down. When floor panel is pressed, lift is ready to go to specific floor. Suppose lift is on the first floor, a user press up button and press target after that lift will go to pressed floor. If lift user press down while lift is on the first floor, then no action is processed. If lift is on the 5th floor, user presses down button and presses at 6th floor then lift has no action, etc. After going to desired floor, lift will initialize timer to 0. If no action more than defined time (in this case, defined time is 15 seconds), then lift will go to the 1st floor automatically. Its state diagram is shown in Figure 6 and CafeOBJ specification is shown Figure 7.




```
module* FLOOR{
        pr(NAT)

        *[Floor]*
        op initFloor : -> Floor
        bop press : Nat Floor -> Floor
        bop stage : Floor -> Nat

        var F : Floor
        var N : Nat

        eq stage(initFloor) = 1 .
        eq stage(press(N,F)) = N .
}
```

Figure 5: State diagram and CafeOBJ specification of *FLOOR* objectt

5.2 Prove score of LIFT specification
From the CafeOBJ specification of LIFT derived in case, we can write the prove score for proving the correctness of specification as follows:
LIFT proving, we must observe the value of 3 objects - BUTTON, TIMER, and FLOOR. Each

object has its observation operation as such getButton for BUTTON, getTimer for TIMER, and getFloor for FLOOR.

1.) Observe the value of BUTTON through object lift.

red getButton(upFloor(initLift)) == up

We prove that if user press up button, then the value of Button object must be up. So, CafeOBJ interpreter must return true.

red getButton(downFloor(initLift)) == down

We prove that if user press down button, then the value of Button object must be down. As the same way, CafeOBJ interpreter must return true.

2.) Observe the value of FLOOR through object lift.

red getFloor(goUp(6,upFloor(initLift))

We prove that if Lift is on the first floor and user press up button and go to 6th floor, then after process is completed, floor must be 6. So, CafeOBJ interpreter must return 6.

red getFloor(goDown(5,(downFloor(initLift)))

We want to prove that if Lift is on the 1st floor but user press down button and want to go to 5th floor, then Lift doesn't do anything. Then CafeOBJ interpreter must return 1, these mean that Lift is on the same floor if condition is incorrect.

Same as previous proved, if condition is true Lift can go to desired floor, if condition is false Lift has no action.

3.) Observe the value of TIMER through object lift.

red getTimer(goUp(4,upFloor(initLift)))

For prove in case of process is executed, and observe the value of Timer that if desired floor is arrived, then Timer must be set to 0. If Lift stop on other floor more than or equal 15 second, then Lift must go to 1st floor automatically. It can be prove by the following prove score :

```
red
getFloor(goFstFloor(incTimer(incTimer(incTimer(incTimer(incTimer(incTimer(incTimer(incTimer(incTimer(incTimer(incTimer(incTimer(incTimer(incTimer(goUp(8,upFloor(goUp(4,upFloor(initLift))))))))))))))))))))))))))))))))
```

If Timer is less than 15 then Lift must on the same floor. We can prove by following prove score:

```
red
getFloor(goFstFloor(incTimer(incTimer(incTimer(incTimer(incTimer(incTimer(incTimer(incTimer(incTimer(incTimer(incTimer(incTimer(incTimer(incTimer(goUp(8,upFloor(goUp(4,upFloor(initLift))))))))))))))))))))))))))))))))
```

By these prove score, CafeOBJ interpreter must return 8.

6. Conclusion and future work

In this approach, we attempt to define relationship between state diagram of UML and CafeOBJ specification which we hope that user who use this approach can understand the CafeOBJ specifications well without mathematical training. Finally, this approach can help designer to check the completeness, consistencies, and design flaws of specification.

The transformation rules are used as a guideline to consider each component of UML state diagram and provide a corresponding formal definition in CafeOBJ syntax. In our approach, we imply that each state diagram describes the behavior property of an object. Both static system and dynamic system of CafeOBJ are covered. We demonstrate our formal specification method with a case study and the correctness of final specification from our method is proved with CafeOBJ interpreter.

7. Acknowledgement

This research was partially supported by a grant from the Thai Government-Private Sector Joint Research Project, Chulalongkorn University and Freewill Solution Company Limited.

8. References

[1] Edmund M. Clarke, Jeannette M. Wing, and et al, *Formal Methods: State of the Art and Future Directions*, ACM Computing Surveys, 1996.

[2] T.H. Tse, *FOOD: a Graphical Interface for Object-Oriented Algebraic Specifications*, Dept. of Computer Science, The University of Hong Kong, 1996.

[3] Scott Allan DeLoach, *Formal Transformation from Graphically-Based Object-Oriented Representations to Theory-Based Specifications*, DoD, 1996.

[4] OMG, *OMG Unified Modeling Language specification version 1.3*, OMG, 1999.

[5] Razvan Diaconescu, and Kokichi Futasugi, *CafeOBJ Report*, World Scientific, 1998.

[6] Ataru T. Nakagawa, Toshimi Sawada, and Kokichi Futasugi, *CafeOBJ User's Manual version 1.4*, World Scientific, 1998.

[7] Liliana Favre, and Silvia Clèrici, *Integrating UML and algebraic specification techniques*, In proceeding of Technology of Object-Oriented Languages and Systems, IEEE, 1999.

[8] Shusaku Iida, Kokichi Futatsugi, and Razvan Diaconescu, *Component-Based Algebraic Specification: behavioral specification for component-based software engineering*, 7th OOPSLA workshop on Behavioral Semantics of Object-Oriented Business and System Specification, 1998.

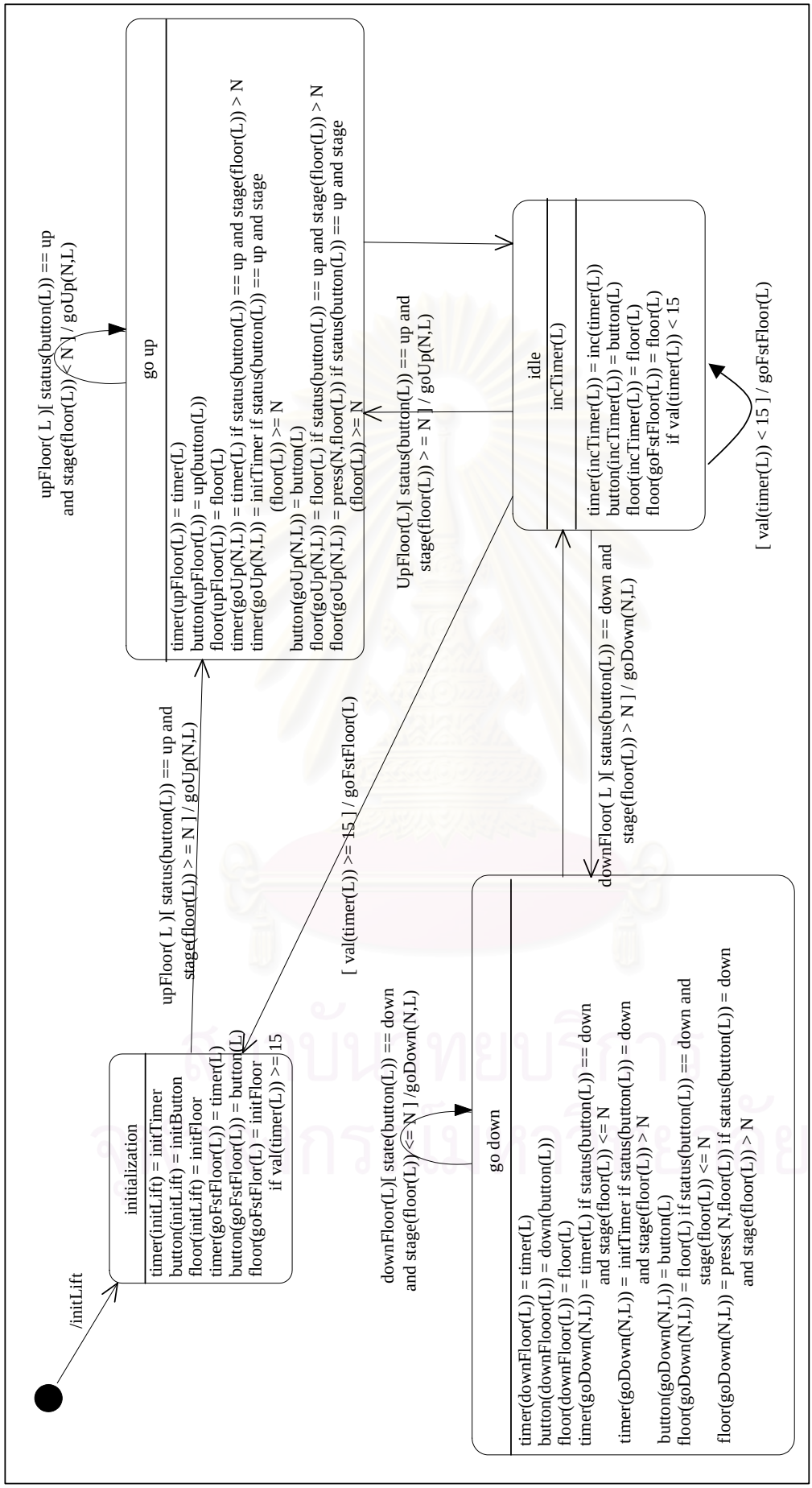


Figure 6: State diagram of *LIFT* object

```
module* LIFT{
    pr(TIMER + BUTTON + FLOOR)

    *[Lift]*

    op initLift : -> Lift
    -- go up floor
    bop goUp : Nat Lift -> Lift
    -- go down floor
    bop goDown : Nat Lift -> Lift
    -- pressing Up
    bop upFloor : Lift -> Lift
    -- pressing Down
    bop downFloor : Lift -> Lift
    bop incTimer : Lift -> Lift
    bop goFstFloor : Lift -> Lift

    -- Observation operation for FLOOR
    bop getFloor : Lift -> Nat
    -- Observation operation for TIMER
    bop getTimer : Lift -> Nat
    -- Observation operation for BUTTON
    bop getButton : Lift -> Value

    -- Projection operation of TIMER
    bop timer : Lift -> Timer
    -- Projection operation of FLOOR
    bop floor : Lift -> Floor
    -- Projection operation of BUTTON
    bop button : Lift -> Button

    var N : Nat
    var L : Lift

    eq getTimer(L) = val(timer(L)) .
    eq getButton(L) = status(button(L)) .
    eq getFloor(L) = stage(floor(L)) .

    eq timer(initLift) = initTimer .
    eq timer(upFloor(L)) = timer(L) .
    eq timer(downFloor(L)) = timer(L) .
    ceq timer(goUp(N,L)) = initTimer
        if status(button(L)) == up and
        stage(floor(L)) < N .
    ceq timer(goUp(N,L)) = timer(L)
        if status(button(L)) == up and
        stage(floor(L)) >= N .
    ceq timer(goDown(N,L)) = initTimer
        if status(button(L)) == down and
        stage(floor(L)) > N .
    ceq timer(goDown(N,L)) = timer(L)
        if status(button(L)) == down and
        stage(floor(L)) <= N .
    eq timer(incTimer(L)) = inc(timer(L)) .
    ceq timer(goFstFloor(L)) = initTimer
        if val(timer(L)) == 15 .

    eq button(initLift) = initButton .
    eq button(upFloor(L)) = up(button(L)) .
    eq button(downFloor(L)) = down(button(L)) .
    eq button(goUp(N,L)) = button(L) .
    eq button(goDown(N,L)) = button(L) .
    eq button(incTimer(L)) = button(L) .
    eq button(goFstFloor(L)) = button(L) .

    eq floor(initLift) = initFloor .
    eq floor(upFloor(L)) = floor(L) .
    eq floor(downFloor(L)) = floor(L) .
    ceq floor(goUp(N,L)) = floor(L)
        if status(button(L)) == up and
        stage(floor(L)) >= N .
    ceq floor(goUp(N,L)) = press(N,floor(L))
        if status(button(L)) == up and
        stage(floor(L)) < N .
    ceq floor(goDown(N,L)) = floor(L)
        if status(button(L)) == down and
        stage(floor(L)) <= N .
    ceq floor(goDown(N,L)) = press(N,floor(L))
        if status(button(L)) == down and
        stage(floor(L)) > N .
    eq floor(incTimer(L)) = floor(L) .
    ceq floor(goFstFloor(L)) = initFloor
        if val(timer(L)) >= 15 .
    ceq floor(goFstFloor(L)) = floor(L)
        if val(timer(L)) < 15 .
}
```

Figure 7: CafeOBJ specification of *LIFT* object

ประวัติผู้เขียนวิทยานิพนธ์

นายมาณพ รัติโชติ เกิดเมื่อวันที่ 9 ตุลาคม พ.ศ. 2513 ที่จังหวัดสงขลา สำเร็จการศึกษาหลักสูตรวิทยาศาสตรบัณฑิต (วท.บ.) สาขาสถิติ คณะวิทยาศาสตร์ มหาวิทยาลัยศรีนครินทรวิโรฒ สงขลา เมื่อปีการศึกษา 2536 จากนั้นได้เข้าทำงานด้านคอมพิวเตอร์ที่ บริษัทแคนนอนไฮเทค (ประเทศไทย) จำกัด และเข้าศึกษาต่อหลักสูตรวิทยาศาสตรมหาบัณฑิต (วท.ม.) ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2541