การออกแบบและพัฒนาระบบสารสนเทศเชิงวัตถุสัมพันธ์โดยใช้ OONIAM/UML

กรณีศึกษา : ระบบห้องสมุด

The Design and Implementation of an Object-Relational System

Using OONIAM/UML

The Case Study of Library System

นายวัชกร เรืองโรจน์วิชัย

นายวิจักษ์ วันโสภา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2544

การออกแบบและพัฒนาระบบสารสนเทศเชิงวัตถุสัมพันธ์โดยใช้ OONIAM/UML
กรณีศึกษา : ระบบห้องสมุด

The Design and Implementation of an Object-Relational System
Using OONIAM/UML
The Case Study of Library System

โดย

นายวัชกร เรืองโรจน์วิชัย 41014378

นายวิจักษ์ วันโสภา 41014382

อาจารย์ที่ปรึกษา

รศ.ดร. ศุภมิตร จิตตะยโศธร

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2544

ปริญญานิพนธ์ปีการศึกษา 2544

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การออกแบบและพัฒนาระบบสารสนเทศเชิงวัตถุสัมพันธ์โดยใช้ OONIAM/UML
กรณีศึกษา : ระบบห้องสมุด

ผู้จัดทำ

1. นายวัชกร เรืองโรจน์วิชัย รหัสประจำตัว 41014378
2. นายวิจักษ์ วันโสภา รหัสประจำตัว 41014382

อาจารย์ที่ปรึกษา

(รศ.ดร. ศุภมิตร จิตตะยโศธร)

การออกแบบและพัฒนาระบบสารสนเทศเชิงวัตถุสัมพันธ์โดยใช้ OONIAM/UML
กรณีศึกษา : ระบบห้องสมุด

นายวัชกร เรืองโรจน์วิชัย
นายวิจักษ์ วันโสภา
รศ.ดร.ศุภมิตร จิตตะยโศธร อาจารย์ที่ปรึกษา
ปีการศึกษา 2544

บทคัดย่อ

การออกแบบและพัฒนาระบบในปัจจุบัน ได้มีการหันมาใช้แนวคิดเชิงวัตถุกันอย่างแพร่หลาย ทั้งในด้านการเขียนโปรแกรม และในด้านการจัดการฐานข้อมูล แต่สิ่งที่สำคัญคือเครื่องมือที่ใช้ในการออกแบบระบบจะต้องสามารถนำเสนอรายละเอียดและการทำงานของระบบได้อย่างครบถ้วน เพื่อที่จะสร้างความเข้าใจที่ถูกต้องชัดเจน และง่ายสำหรับสร้างและบำรุงรักษาระบบ

เครื่องมือที่มีการใช้งานกันอย่างแพร่หลายคือ UML โครงงานนี้จึงได้ทำการทดลองใช้ UML ออกแบบระบบห้องสมุด แล้วสร้างโปรแกรมแอปพลิเคชันขึ้นจริงจากข้อมูลได้จาก UML เพื่อหาสิ่งที่ UML ไม่ได้นำเสนอได้ใน UML พร้อมทั้งนำเสนอวิธีการปรับปรุงเพื่อให้ได้รายละเอียดของระบบมากยิ่งขึ้น

โดยโปรแกรมที่ทำการทดลองนี้ ใช้ระบบจัดการฐานข้อมูลเชิงวัตถุสัมพันธ์คือ อินฟอร์มิกส์ ซึ่งรองรับดาต้าไทป์ที่ซับซ้อน และการถ่ายทอดคุณสมบัติ เราจึงได้สร้างโปรแกรมให้มีความสามารถในการสร้างตารางและรูทีนให้ระบบใหม่ ด้วยการถ่ายทอดตารางและรูทีนในดาต้าเบสของระบบกลาง โดยที่สามารถปรับเปลี่ยน คุณสมบัติของตารางและรูทีนได้

The Design and Implementation of an Object-Relational System Using OONIAM/UML

The Case Study of Library System

Mr. Vatchakorn Raungrojvichai 41014378

Mr. Wichak Wansopha 41014382

Assoc. Prof. Dr. Suphamit Chittayasothon Adivisor

ABSTRACT

Nowaday, There are lot of designing and development that using object oriented concept both of programming or DBMS. But the designed tool is the most important thing that give necessary information for the developer to make application program and maintain it.

One designed tool that widely use now is UML. The experiment of this project is using UML diagram that designed library system to make the application program of library system. During making the application program we try to find the defect of UML representation and purpose solution of these defect.

We use object oriented database management system : Informix as DBMS connected with the application program. Because of using OODBMS that support collection datatype and inheritance so we make the program can inherit table and routine for new system.

กิตติกรรมประกาศ

โครงงานและวิทยานิพนธ์ฉบับนี้สามารถสำเร็จลุล่วงได้ ก็เนื่องจากผู้จัดทำได้รับความช่วยเหลือและคำแนะนำ จากบุคคลหลายฝ่าย

รศ.ดร. ศุภมิตร จิตตะยโศธร อาจารย์ที่ปรึกษาที่คอยให้คำแนะนำและดูแลมาโดยตลอด

คณาจารย์ภาควิชาคอมพิวเตอร์ทุกท่านที่ให้ความรู้มาตลอดทั้ง 4 ปี

บิดามารดา ที่ให้กำเนิดและเป็นกำลังใจเสมอมา

ผู้จัดทำขอขอบพระคุณอย่างสูง

นายวัชกร เรืองโรจน์วิชัย

นายวิจักษ์ วันโสภา

สารบัญ

สารบัญ (ต่อ)

ตารางสารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญรูปภาพ

หน้าที่

สารบัญรูปภาพ (ต่อ)

หน้า

สารบัญรูปภาพ (ต่อ)

หน้าที่

สารบัญรูปภาพ (ต่อ)

หน้าที่

สารบัญตาราง

หน้าที่

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันการพัฒนาระบบด้วยแนวคิดเชิงวัตถุกำลังได้รับความนิยมอย่างมาก เนื่องจากมีข้อดีหลายประการดังต่อไปนี้

- ช่วยให้นักพัฒนาสามารถออกแบบระบบได้อย่างครบถ้วนมากยิ่งขึ้น
- ช่วยเพิ่มความเข้าใจในโดเมนของปัญหา
- สนับสนุนการนำกลับมาใช้
- สนับสนุนการปรับเปลี่ยนขนาดของระบบ
- สนับสนุนการทำงานพร้อมกัน

ในช่วงแรกของโครงงานนี้ได้ทำการศึกษาวิธีการและเครื่องมือออกแบบระบบตามแนวคิดเชิงวัตถุ ที่มีการใช้งานอยู่ในปัจจุบัน เพื่อหาวิธีการออกแบบที่สามารถให้ข้อมูลครบถ้วนสำหรับนักพัฒนาโปรแกรม ซึ่งได้พบวิธีการออกแบบที่ใช้สัญลักษณ์และรูปภาพที่แตกต่างกันไป

UML เป็นเครื่องมือในการออกแบบระบบตามแนวคิดเชิงวัตถุที่ได้รับการยอมรับและมีการใช้งานอย่างแพร่หลายในปัจจุบันเพื่อที่จะทำความเข้าใจที่ถูกต้องในการใช้งาน UML และเพื่อหาจุดบกพร่องของ UML จึงได้ทำการทดลองสร้างระบบห้องสมุดเชิงวัตถุจากโดยใช้ไดอะแกรมของ UML ในการออกแบบ และทำการสร้างโปรแกรมเชิงวัตถุระบบห้องสมุดด้วยภาษาเดลไฟ จัดเก็บฐานข้อมูลในอินฟอร์มิกส์ ซึ่งเป็นระบบจัดการฐานข้อมูลเชิงวัตถุสัมพันธ์ ที่มีความสามารถรองรับดาต้าไทป์ที่ซับซ้อน สามารถสร้างรูทีนเก็บไว้ในดาต้าเบสเซิร์ฟเวอร์ และยังสามารถสร้างตารางในฐานข้อมูลด้วยการถ่ายทอดคุณสมบัติของตารางที่มีอยู่ได้

ในระหว่างการสร้างโปรแกรม โดยดูจากไดอะแกรมของ UML ก็ทำการหาจุดที่ UML ยังขาด พร้อมทั้งเสนอวิธีการปรับปรุงเพิ่มเติม UML เพื่อให้ได้ไดอะแกรมที่สามารถให้ข้อมูลที่ครบถ้วนสำหรับที่นักพัฒนาจะต้องใช้ในการสร้างโปรแกรม

และจากที่เราต้องการสร้างโปรแกรมเชิงวัตถุ ที่ติดต่อกับระบบจัดการฐานข้อมูลเชิงวัตถุสัมพันธ์ ซึ่งมีความสามารถรองรับการถ่ายทอดคุณสมบัติ จึงได้ทำการสร้างโปรแกรมที่มีความสามารถในการสร้างตารางและรูทีนของระบบห้องสมุดอีกระบบที่อาจจะแตกต่างไปจากตารางและรูทีนของระบบเดิม โดยการถ่ายทอดคุณสมบัติจากตารางและรูทีนของระบบเดิม ทำให้สามารถสร้างระบบห้องสมุดอีกระบบขึ้นใหม่ได้อย่างง่ายดายและรวดเร็วยิ่งขึ้น

1.2 วัตถุประสงค์

1.2.1 เพื่อหาวิธีการออกแบบระบบตามแนวคิดเชิงวัตถุที่ดีที่สุดโดยพยายามนำเสนอสิ่งที่น่าจะปรับปรุงใน UML

1.2.2 เพื่อสร้างโปรแกรมที่มีความสามารถในการ ถ่ายทอดคุณสมบัติของตารางและรูทีนที่เก็บไว้ในดาต้าเบสเซิร์ฟเวอร์โดยใช้ความสามารถของระบบจัดการฐานข้อมูลเชิงวัตถุสัมพันธ์ โดยที่ สามารถปรับเปลี่ยนให้มีความแตกต่างไปได้ หรือ Customize ได้

1.3 วิธีการดำเนินงาน

1.3.1 ทำการศึกษาค้นคว้าเครื่องมือและวิธีการออกแบบตามแนวคิดเชิงวัตถุที่มีการใช้งานอยู่ในปัจจุบัน

1.3.2 ศึกษาการใช้งาน ระบบจัดการฐานข้อมูลเชิงวัตถุสัมพันธ์ ได้แก่ อินฟอร์มิกส์

1.3.3 สร้างไดอะแกรมของระบบห้องสมุดด้วยวิธีการของ UML

1.3.4 สร้างโปรแกรมจากไดอะแกรมของ UML หากพบข้อบกพร่องที่ไดอะแกรมของ UML ไม่สามารถนำเสนอได้ ก็นำเสนอวิธีการแก้ไข จนกว่าจะได้ไดอะแกรมที่มีความสมบูรณ์ มีข้อมูลครบถ้วนสำหรับการเขียนโปรแกรม

1.3.5 สร้างโปรแกรมที่มีความสามารถในการ Customize ได้

1.4 อุปกรณ์ที่ใช้ในการดำเนินงาน

1.4.1 Informix Dynamic Server 2000 (IDS 2000)

1.4.2 Rational Rose 2000 Enterprise Edition

1.4.3 Visio Modeler

1.4.4 Server Studio JE

1.4.5 Delphi

1.4.6 Application Program อื่นๆ

บทที่ 2
ทฤษฎีและความรู้พื้นฐานของแนวคิดเชิงวัตถุ

เทคโนโลยีในการพัฒนาระบบในปัจจุบันส่วนใหญ่ใช้หลักการทางวัตถุ ซึ่งบรรจุไว้ด้วยองค์ประกอบต่างๆ ที่เรียกว่า โมเดลของวัตถุ (Object Model) ในโมเดลของวัตถุนั้นก็แบ่งออกเป็นการแยกแยะเอกลักษณ์(Abstraction) การซ่อนรายละเอียด(Encapsulation) การรวมกลุ่มความสัมพันธ์(Modularity) ลำดับชั้น(Hierarchy) ชนิด(Typing) การทำงานพร้อมกัน(Concurrency) และการรักษาสถานะ(Persistence) สำหรับการวิเคราะห์และออกแบบระบบด้วยวิธีเชิงวัตถุนั้น จะแตกต่างจากการออกแบบระบบด้วยวิธีโครงสร้างเนื่องจากรูปแบบในการคิดและการมองระบบจะต่างกัน นอกจากนั้นการออกแบบและพัฒนาระบบแบบโครงสร้างจะขึ้นอยู่กับภาษาในการเขียนโปรแกรมโครงสร้างด้วย ส่วนการวิเคราะห์และออกแบบระบบก็จะขึ้นอยู่กับภาษาในการเขียนโปรแกรมเชิงวัตถุเช่นกัน และในภาษาสำหรับการเขียนโปรแกรมเชิงวัตถุแต่ละภาษาก็มีรายละเอียดปลีกย่อยที่ต่างกันอีกด้วย

2.1 ออบเจ็กต์(Object)

ออบเจ็กต์ หรือวัตถุ คือสิ่งที่เรากำหนดขึ้นมาจากนามธรรมขึ้นมาเป็นรูปธรรมที่มีอยู่จริง โดยสิ่งที่มีอยู่จริงไม่จำเป็นต้องจับต้องได้เท่านั้น แต่รวมไปถึงอะไรก็ตามที่เราสามารถแสดงลักษณะใดๆ ก็ได้ ซึ่งคุณสมบัติเหล่านี้จะเป็นคุณสมบัติเฉพาะของแต่ละวัตถุซึ่งสามารถเปลี่ยนแปลงไปมาได้ เช่น เรากำหนดว่าสิ่งที่ขับเคลื่อนด้วยเครื่องยนต์และมีสี่ล้อคือรถ นี่เป็นการนิยามหรือนามธรรม(Abstraction) ของรถที่เรากำหนดขึ้น ดังนั้นออบเจ็กต์ของสิ่งนี้ก็คือ รถกระบะ รถตู้ และอื่นๆ นั่นเอง โดยส่วนประกอบภายในของออบเจ็กต์นั้นจะมีการกำหนดสถานะ(State) และพฤติกรรม(Behavior) หรือเมธทอด(Method) เอาไว้ด้วย

สถานะของออบเจ็กต์คือสภาวะการทำงานที่ออบเจ็กต์นั้นมีอยู่ ซึ่งออบเจ็กต์อื่นจะไม่สามารถเห็นสภาวะการทำงานนี้ได้ และพฤติกรรมหรือเมธทอดคือการทำงานของออบเจ็กต์นั้นๆ ว่ามีการทำงานอะไรบ้าง ซึ่งเป็นส่วนที่ออบเจ็กต์อื่นสามารถที่จะทราบได้ว่าออบเจ็กต์นั้นสามารถทำงานอะไรได้

ลักษณะของออบเจ็กต์อีกอย่างที่เราให้ความสำคัญเป็นอย่างมากก็คือ การสร้างส่วนที่ใช้ติดต่อหรือเชื่อมโยง (Interface) กันระหว่างออบเจ็กต์ โดยที่แต่ละออบเจ็กต์จะไม่สามารถเข้าไปเกี่ยวข้องกับการทำงานของออบเจ็กต์อื่นได้ และไม่จำเป็นที่จะต้องรู้ด้วยว่าภายในออบเจ็กต์นั้นมีการทำงานอย่างไร แต่เราให้ความสนใจที่การเชื่อมโยงหรือติดต่อกับออบเจ็กต์เท่านั้น ข้อมูลที่ใช้ในการส่งผ่านการเชื่อมโยงหรือติดต่อไปยังแต่ละออบเจ็กต์เราเรียกว่า แมสเซจ(Message) การที่มีการเชื่อมโยงแบบนี้ เป็นส่วนที่สำคัญในการแสดงคุณลักษณะของวัตถุ และสิ่งที่อยู่ภายนอกก็จะต้องเห็นส่วนอินเทอร์เฟซของวัตถุด้วย วัตถุสามารถซ่อนรายละเอียดที่ไม่สำคัญสำหรับผู้ใช้ได้หรือที่เรียกว่า Encapsulation ตัวอย่างเช่น ในภาษา C++ มีคำสั่งเพื่อใช้ซ่อนแอททริบิวต์และเมธอดได้

2.2 แอททริบิวต์ (Attribute)

วัตถุจำเป็นต้องมีการเก็บข้อมูลของตัวเอง โดยข้อมูลที่วัตถุเก็บไว้นั้นอาจจะเก็บไว้เฉยๆ หรือเก็บไว้เพื่อที่จะให้ผู้อื่นเรียกใช้งาน หรือเก็บไว้ใช้ในการประมวลผล เพื่อตอบสนองกับเหตุการณ์ภายนอก หรืออาจจะเก็บไว้เพื่อเป็นสถานะของตัววัตถุเอง สรุปว่าวัตถุโดยส่วนใหญ่จะต้องเก็บข้อมูล เพื่อให้วัตถุสามารถทำหน้าที่รับผิดชอบได้อย่างถูกต้อง

โดยทั่วไปแอททริบิวต์ของวัตถุจะถูกซ่อนไว้จากผู้ที่จะใช้งานวัตถุ ทั้งนี้เนื่องจากว่าการเข้าใช้งานแอททริบิวต์โดยตรงเปรียบเสมือนเป็นการเข้าถึงโครงสร้างของวัตถุ ดังนั้นการออกแบบวัตถุที่ดีจึงควรที่จะซ่อนแอททริบิวต์ของวัตถุไว้เพื่อไม่ให้ผู้ใช้เห็น

บางครั้งการออกแบบวัตถุที่ไม่ดีก็อาจจะทำให้วัตถุนั้นมีแต่แอททริบิวต์โดยไม่มีเมธอดได้ ซึ่งในกรณีนี้นักออกแบบจะต้องย้อนกลับไปดูว่าวัตถุนั้นควรจะมีเมธอดอะไร เพราะวัตถุที่มีแต่แอททริบิวต์นั้นก็ไม่แตกต่างจากโครงสร้างข้อมูลเลย

2.3 เมธอด(Method) หรือพฤติกรรม(Behavior)

วัตถุที่ทำงานแบบพาสสิฟ (Passive Object) จะมีบริการไว้ให้วัตถุอื่นๆ เรียกใช้ ตัวอย่างเช่น ADT ก็เป็นวัตถุที่ทำงานแบบพาสสิฟ ในตัวอย่างของภาษา C++ วัตถุสแต็กสามารถเก็บข้อมูลพื้นฐานได้ สามารถเพิ่มและลบข้อมูลออกได้ และมีการตรวจสอบความถูกต้องของข้อมูลง่ายๆ ส่วนวัตถุที่ทำงานแบบแอกทีฟ(Active Object) นั้นจะเป็นวัตถุหลักของเธรด ซึ่งจะทำหน้าที่เรียกใช้บริการจากวัตถุที่ทำงานแบบพาสสิฟ

เมธอดสามารถแบ่งออกเป็น 3 แบบคือ Simple, Automation และ Continuous เมธอดทั้ง 3 แบบนี้มีความสำคัญที่แตกต่างกันไป คือ

2.3.1 เมธอดแบบ Simple

สำหรับเมธอดแบบนี้ วัตถุจะให้บริการกับการร้องขอเข้ามาและไม่ค้างค่าในหน่วยความจำไว้สำหรับให้บริการครั้งต่อไป การกระทำเป็นหนึ่งเดียวและสมบูรณ์ในตัว วัตถุพื้นฐานจะมีข้อมูลชนิดพื้นฐานและโอเปอเรชันที่เกี่ยวข้องกับข้อมูลเหล่านี้ ตัวอย่างเช่น วัตถุไบนารีทรี(Binary Tree) ก็มีเมธอดแบบ Simple เช่น Insert Item หรือ Remove Item เป็นต้น ส่วนตัวอย่างอื่นๆ คือ ฟังก์ชัน cos(x) นี้ไม่ได้เก็บสถานะและข้อมูลเก่าไว้เลย เรียกวัตถุที่มีเฉพาะเมธอดแบบนี้ว่า Primitive

2.3.2 เมธอดแบบ Automation

เมธอดของวัตถุแบบนี้ ผู้พัฒนาจะมองวัตถุเป็นเหมือนเครื่องจักรชนิดหนึ่งที่เรียกว่า Automation หรือ Finite State Machine (FSM) วัตถุชนิดนี้จะมีกลุ่มของสถานะที่มีขอบเขต วัตถุหนึ่งอาจจะมีสถานะ 1 หรือมากกว่าในเวลาใดๆ การที่วัตถุจะอยู่ในสถานะใดๆ นั้น ก็จะขึ้นอยู่กับอีเวนต์หรือเหตุการณ์ที่เข้ามาจากสภาวะแวดล้อมภายนอก และเนื่องจากว่าวัตถุแบบนี้ทำงานแบบ State Machine เพื่อตอบสนองกับอีเวนต์ภายนอกจึงเรียกวัตถุแบบนี้ว่าวัตถุแบบรีแอกทีฟ(Reactive Object)

อีเวนต์ภายนอกที่มีเข้ามาอาจจะทำให้วัตถุเปลี่ยนสถานะ วิธีการในการพัฒนาระบบเชิงวัตถุบางวิธีอ้างว่า วัตถุทั้งหมดมีสถานะ ตัวอย่างเช่น ตัวแปลง A/D แบบ Sample-And-Hold อาจเป็นวัตถุที่มีสถานะต่างๆ ดังนี้

- Enabled
- Sampling
- Holding
- Disabled

2.3.3 เมธอดแบบ Continuous

วัตถุที่มีเมธอดแบบ Continuous จะมีกลุ่มของสถานะที่ไม่จำกัดและไม่มีขอบเขต ตัวอย่างหนึ่งคือ วัตถุอัลกอริทึม ซึ่งวัตถุนี้เป็นวัตถุที่จะปฏิบัติอัลกอริทึมบนสายของข้อมูลที่ต่อเนื่อง วัตถุที่มีเมธอดแบบ Continuous นั้นการทำงานของวัตถุปัจจุบันจะขึ้นอยู่กับการทำงานของเมธอดและอินพุทก่อนหน้า ซึ่งความเกี่ยวข้องระหว่างสถานะนี้อาจจะเป็นความสัมพันธ์แบบต่อเนื่อง ตัวอย่างของระบบ Continuous คือระบบ Fuzzy และตัวควบคุม PID ซึ่งเป็นเครื่องสร้างจำนวนแบบ pseudo-random และตัวกรองดิจิตอล การทำงานของเมธอดของระบบเหล่านี้จะขึ้นอยู่กับอินพุทก่อนหน้า

2.4 แมสเซจ(Message)

การสื่อสารระหว่างวัตถุสามารถทำได้โดยการส่งแมสเซจ แมสเซจเป็นการแยกแยะเอกลักษณ์ของข้อมูล(Data) หรือข้อมูลสารสนเทศ(Information) การควบคุมที่ส่งมาจากวัตถุหนึ่งไปยังวัตถุอื่นๆ ในเชิงการเขียนโปรแกรมแมสเซจสามารถทำได้หลายแบบ ตัวอย่างเช่น การเรียกฟังก์ชัน อินเทอร์รัพ เป็นต้น

ในขั้นตอนการวิเคราะห์ระบบจะมีการกำหนดแมสเซจ ส่วนในขั้นตอนของการออกแบบนั้นจึงจะมากำหนดรูปแบบการซิงโครไนเซชัน และความต้องการทางด้านเวลาของแต่ละแมสเซจ และเมื่อวัตถุได้รับแมสเซจมา วัตถุก็จะเปลี่ยนแมสเซจที่เข้ามาให้เป็นโอเปอเรชัน การเปลี่ยนสถานะ คำสั่งหรือข้อมูลตามที่เหมาะสม

การใช้แมสเซจทำให้แต่ละวัตถุมีความเกี่ยวข้องกันน้อยลง ในขั้นตอนของการวิเคราะห์ผู้พัฒนาระบบจะไม่ได้กำหนดรายละเอียดในการติดต่อของแมสเซจว่า จะมีรายละเอียดของการเรียกฟังก์ชัน Rendezvous ไทม์เอาท์(Time out) และเมื่อถึงขั้นตอนของการออกแบบ จึงจะจัดการกับรายละเอียดเหล่านี้

ส่วนอินเทอร์เฟซของวัตถุเป็นส่วนที่วัตถุใช้ติดต่อกับโลกภายนอก ซึ่งอินเทอร์เฟซทจะทำหน้าที่กำหนดเซ็ตของโพรโตคอลเพื่อสื่อสารกับวัตถุอื่นๆ โพรโตคอลของส่วนอินเทอร์เฟซนั้นประกอบด้วย 3 สิ่งดังนี้คือ

1. เงื่อนไขก่อนหน้า(Precondition)
2. ซิกเนเจอร์(Signature)
3. เงื่อนไขท้าย(Postcondition)

เงื่อนไขก่อนหน้าเป็นเงื่อนไขที่จะต้องมีค่าเป็นจริงก่อนที่จะส่งแมสเซจ โดยปกติเงื่อนไขก่อนหน้านั้นจะเป็นหน้าที่ของวัตถุที่เป็นผู้ส่งแมสเซจจะต้องตรวจสอบ ส่วนเงื่อนไขท้ายนั้นเป็นเงื่อนไขที่จะต้องมีค่าเป็นจริงหลังจากที่วัตถุได้ประมวลผลแมสเซจเสร็จเรียบร้อยแล้ว และเป็นหน้าที่ของวัตถุที่รับแมสเซจ สำหรับแมสเซจซิกเนเจอร์นั้นเป็นรูปแบบในการส่งแมสเซจ ซึ่งอาจจะเป็นการเรียกฟังก์ชันพร้อมพารามิเตอร์(Parameter) และค่าที่คืนกลับมา(Return Type) หรือเป็นแมสเซจ post/pend ของ RTOS หรือข้อตกลงของแมสเซจบนบัสก็ได้

2.5 ความรับผิดชอบ(Responsibility)

ความรับผิดชอบของวัตถุคือ หน้าที่และบทบาทของวัตถุต่อระบบ โดยส่วนอินเทอร์เฟซและเมธอดของวัตถุนั้นสามารถใช้เพื่อให้วัตถุทำหน้าที่รับผิดชอบได้ ความรับผิดชอบของวัตถุควรจะเป็นสิ่งแรกที่ผู้พัฒนาระบบกำหนดให้กับวัตถุ เพราะถ้าไม่รู้ว่าวัตถุนั้นมีหน้าทีอย่างไร ก็จะไม่สามารถกำหนดคุณลักษณะอื่นๆ ให้กับวัตถุได้อย่างถูกต้องและครบถ้วน

2.6 การนามธรรม(Abstraction)

การนามธรรมคือการกำหนดขอบเขตหรือนิยามขึ้นมาให้เห็นเด่นชัดว่าสิ่งที่เรากำหนดนั้นคืออะไร แตกต่างกับสิ่งอื่นๆ อย่างไรบ้าง โดยข้อกำหนดต่างๆ ที่กำหนดขึ้นจะพิจารณาจากความเป็นจริงของสิ่งหลายๆ สิ่งที่จัดอยู่ในประเภทเดียวกัน ว่ามีความเหมือนกันที่ใด สิ่งนั้นมีการทำงานเหมือนกันหรือเปล่า สิ่งนั้นใช้ในสถานการณ์เดียวกันหรือไม่ และมองข้ามสิ่งที่หลายๆ สิ่งมีเหมือนกัน แต่อาจแตกต่างในรายละเอียด ที่ไม่มีผลกระทบให้เกิดความแตกต่าง ซึ่งการกำหนดขอบเขตดังกล่าวจะมีเกี่ยวข้องกับมุมมองของผู้ที่พิจารณาด้วยว่าต้องการแสดงให้เห็นถึงด้านใด

เราสามารถที่จะกำหนดรายละเอียดเจาะจงลงไปได้ว่า ออบเจ็กต์แต่ละตัวมีจุดเด่นอย่างไร จะเห็นได้ว่าการทำนามธรรมสิ่งต่างๆ ที่ดีได้นั้นจะช่วยให้เราไม่ต้องมีการทำงานที่ซ้ำซ้อน ไม่ต้องมีการสร้างใหม่หลายครั้ง และขุบเอาหลายๆ สิ่งมาเป็นสิ่งเดียวกัน แต่จะยากในเรื่องของการมองสิ่งต่างๆ เพื่อที่จะนำมากำหนดเป็นการนามธรรม การทำนามธรรมจะเป็นส่วนที่สำคัญมากในเรื่องของการออกแบบโปรแกรม ซึ่งแบ่งออกเป็น 2 ชนิดคือ

1. การนามธรรมสถานะ(Abstraction of State)

เป็นการทำการนามธรรมสิ่งที่เกี่ยวกับการทำงานของออบเจ็กต์ว่าแต่ละออบเจ็กต์มีการทำงานอะไรบ้าง ใช้ข้อมูลอะไรบ้างในการทำงาน แต่ไม่กล่าวถึงว่าทำงานอย่างไร เมื่อออบเจ็กต์ได้รับแมสเซจเข้ามาทางส่วนอินเทอร์เฟซแล้ว ออบเจ็กต์นั้นจะเอาข้อมูลที่ต้องการมาไว้ที่ภายในออบเจ็กต์ เพื่อให้ออบเจ็กต์อื่นไม่สามารถที่จะทราบได้ว่าออบเจ็กต์นั้นเอาข้อมูลไปทำอะไร และ

มีการทำงานกับข้อมูลนั้นอย่างไรบ้าง โดยเราเรียกการซ่อนรายละเอียดของการทำงานเช่นนี้ว่า การ Encapsulation ซึ่งในส่วนนี้เสมือนเป็นการเตรียมข้อมูลให้กับการทำงานภายในออบเจ็กต์

การกำหนดการเข้าถึงข้อมูลภายในออบเจ็กต์จะทำการแยกเป็นส่วนของการติดต่อกับข้อมูลภายนอกและวิธีการทำงานออกจากกัน พร้อมทั้งทำการซ่อนการทำงานและรายละเอียดของข้อมูลไว้ภายใน นั่นก็คือการซ่อนข้อมูลนั่นเอง(Information Hiding) ซึ่งการเข้าถึงข้อมูลนั้นก็ต้องเรียกผ่านฟังก์ชันการทำงานหรือเมธอดเท่านั้น โดยการซ่อนเร้นข้อมูลนี้จะมองว่าวัตถุ(ออบเจ็กต์) แต่ละตัวเกิดจากการนำเอาข้อมูลและฟังก์ชันที่เกี่ยวข้องเข้ามารวมกันโดยมีข้อมูลเป็นแกนกลาง ซึ่งถูกห่อหุ้มอยู่ โดยเมธอดจะเป็นตัวดำเนินการเพียงอย่างเดียว ที่ยอมให้มีการเข้าถึงข้อมูลที่ถูกซ่อนอยู่ได้ ดังนั้นจะเห็นว่าเราสามารถที่จะเปลี่ยนแปลงรายละเอียดของโปรแกรมได้โดยไม่ต้องเปลี่ยนแปลงส่วนอินเทอร์เฟซ ทำให้โปรแกรมใดก็ตามที่เรียกใช้ยังคงทำงานได้เหมือนเดิม ซึ่งทำให้การบำรุงรักษาโปรแกรมเป็นไปได้โดยง่าย

2. การนามธรรมเมธอด(Abstraction of Method)

เป็นการนามธรรมของส่วนที่จะใช้เป็นส่วนอินเทอร์เฟซกับออบเจ็กต์อื่น เพื่อให้ออบเจ็กต์อื่นสามารถทราบว่า ถ้าต้องการให้ออบเจ็กต์นี้ทำงานหรือติดต่อกับออบเจ็กต์นี้ จะต้องติดต่อผ่านทางส่วนของอินเทอร์เฟซที่เรากำหนดไว้เท่านั้น

ภายในการทำงานของเมธอดของแต่ละออบเจ็กต์ที่สามารถเรียกใช้งานออบเจ็กต์อื่น เพื่อที่จะเอาผลลัพท์ที่ได้จากออบเจ็กต์อื่นๆ นั้นมาใช้งานต่อไป หรือยอมให้มีการใช้งานร่วมกันระหว่างออบเจ็กต์ (Collaboration Among Object) ได้นั่นเอง

2.7 คลาส(Class)

ในโลกของการพัฒนาระบบด้วยวิธีเชิงวัตถุ คำว่าคลาสคือ การแยกแยะเอกลักษณ์ของคุณสมบัติพื้นฐานจากกลุ่มของวัตถุที่เหมือนกัน อาจจะกล่าวได้ว่าคลาสเป็นชนิดของวัตถุ ค่า 0 หรือ -3 หรือ 9999 เป็นอินสแทนซ์หรือวัตถุของคลาสอินทิเจอร์ (Integer) สัตว์เลี้ยงลูกด้วยนมจะมีขน เช่น สุนัข แมว คน สังเกตได้ว่าวัตถุเหล่านี้แบ่งคุณสมบัติพื้นฐานใช้ร่วมกันอยู่ โดยค่าของคุณสมบัติก็จะแตกต่างกันไปแล้วแต่อินสแทนซ์ แต่ว่าทุกอินสแทนซ์จะต้องมีคุณสมบัตินี้ ตัวอย่างเช่น คลาสบัญชีอาจจะมีแอททริบิวต์ของยอดเงินคงเหลือ แต่ว่าบางบัญชีอาจจะมีเงินคงเหลือเป็นบวกหรือเป็นลบก็ได้

นักออกแบบด้วยวิธีเชิงวัตถุไม่สามารถสร้างคลาสขึ้นมาได้ลอยๆ แต่พวกเขาจะต้องสังเกตจากวัตถุและแยกแยะเอกลักษณ์ของวัตถุออกมาให้ได้ ตัวอย่างอินสแทนซ์ของคลาสในการเขียนโปรแกรมทั่วไป เช่น ชนิดข้อมูล Struct ในภาษา C จะกำหนดโครงสร้างข้อมูล ส่วนวัตถุจะทำหน้าที่กำหนดชนิดวัตถุ

คลาสจะกำหนดแอททริบิวต์และเมธอดของวัตถุ ซึ่งเป็นอินสแทนซ์ แต่ว่าคลาสไม่มีความรับผิดชอบ (Responsibility) คุณสมบัติทั้งหมดของคลาสอยู่ในรูปของชนิด(Type) ไม่ใช่ค่า(Value) ถ้าคลาสมีแอททริบิวต์ดังเช่น สีหรือรูปทรงแล้ว อินสแทนซ์ของคลาสก็จะมีคุณสมบัติของสีหรือรูปทรงด้วย แต่จะแตกต่างกันไปตามค่าของแต่ละวัตถุ อย่างไรก็ตามความรับผิดชอบขึ้นอยู่กับการใช้วัตถุในสิ่งแวดล้อมใด วัตถุคอนเทนเนอร์(Container Object) พื้นฐานอาจจะเก็บบัญชีธนาคารและมีหน้าที่สำหรับเข้าใช้บัญชีเช่น ฝาก

ถอน หรือดูยอดบัญชีของลูกค้าทุกๆ คน ส่วนวัตถุคอนเทนเนอร์ของคลาสเดียวกันอาจจะเก็บเรคอร์ดข้อมูลในคลังสินค้า และยังสามารถเข้าใช้ระบบควบคุมของคลังสินค้าได้ด้วย จะเห็นได้ว่าความรับผิดชอบในชนิดนั้นเหมือนกันแต่ว่าจะแตกต่างกัน เมื่อนำวัตถุไปใช้งานจริง คลาสของวัตถุจะกำหนดความรับผิดชอบของวัตถุเฉพาะเมื่อวัตถุที่ถูกสร้างขึ้นจากคลาสทั้งหมดถูกใช้ในสภาวะแวดล้อมและมีหน้าที่ที่เหมือนกัน

2.8 การซ่อนเร้นข้อมูล(Encapsulation)

จากที่ได้อธิบายมาแล้วข้างต้นว่าออบเจ็กต์มีความสามารถที่จะซ่อนเร้นข้อมูลต่างๆ ภายในออบเจ็กต์จากออบเจ็กต์อื่นๆ ได้ ซึ่งจากคุณสมบัตินี้เองมีประโยชน์เมื่อเราทำการเปลี่ยนแปลงสิ่งต่างๆ ภายในออบเจ็กต์แล้วจะไม่มีผลกระทบต่อออบเจ็กต์ สืบเนื่องมาจากออบเจ็กต์อื่นไม่สามารถที่จะอ้างถึงหรือเข้าถึงข้อมูลภายในออบเจ็กต์นั้นตรงๆ นั่งเอง สำหรับการกำหนดลักษณะการเข้าถึงนั้นมีอยู่ด้วยกัน 3 แบบ คือ Public Private และ Protect โดยแต่ละแบบมีคุณสมบัติต่างๆ ดังนี้

1. **Public** เป็นการกำหนดให้ออบเจ็กต์อื่นๆ ทั้งที่อยู่ภายในคลาสเดียวกันหรือต่างคลาสกันสามารถที่เข้าถึงข้อมูลได้
2. **Private** เป็นการกำหนดให้ออบเจ็กต์อื่นๆ ที่ไม่อยู่ภายในคลาสเดียวกัน ไม่สามารถที่จะเข้าถึงข้อมูลนี้ได้ แต่จะต้องอาศัยการเข้าถึงโดยผ่านเมธอด
3. **Protect** เป็นการกำหนดให้ออบเจ็กต์ที่อยู่ภายในคลาสเดียวกันหรือออบเจ็กต์ที่อยู่ในคลาสที่ได้รับการสืบทอดคุณสมบัติ(Inheritance) สามารถเข้าถึงข้อมูลได้

2.9 โพลิมอร์ฟิซึม (Polymorphism)

คำว่าโพลิมอร์ฟิซึมนั้น แปลว่ารูปแบบหลายๆ รูปแบบ ในทางการออกแบบโปรแกรมเชิงวัตถุนี้จะให้ความหมายของคำๆ นี้คือ แต่ละออบเจ็กต์อาจที่จะใช้ชื่อในการติดต่อหรืออินเทอร์เฟซได้เหมือนกันได้ แต่การทำงานภายในของชื่อที่ใช้ในการติดต่อของแต่ละออบเจ็กต์นั้นไม่เหมือนกันก็ได้ ซึ่งในการทำงานของโปรแกรมจะเป็นตัวเลือกหรือจัดการเองว่า จะเอาออบเจ็กต์ใดที่ชื่อเหมือนกันในขณะนั้นมาทำงาน ยกตัวอย่างของโพลิมอร์ฟิซึมเช่น เปลี่ยนเกียร์ คือชื่อของการทำงานที่เรากำหนดไว้ในส่วนของการทำการติดต่อหรืออินเทอร์เฟซ แต่มีออบเจ็กต์ 2 ตัวที่ใช้ชื่อนี้ในส่วนของการติดต่อคือ อัตโนมัติ(Auto) และธรรมดา(Manual) ซึ่งการเปลี่ยนเกียร์แบบอัตโนมัติจะมีการทำงานโดยใช้เครื่องยนต์ควบคุม แต่การเปลี่ยนเกียร์แบบธรรมดาจะทำงานโดยใช้คนควบคุมเอง เป็นต้น

2.10 ความสัมพันธ์ระหว่างคลาส

เมื่อพิจารณาความเหมือนและความแตกต่างของคลาสดอกไม้ที่ประกอบไปด้วยดอกไม้ ดอกมะลิ ดอกกุหลาบสีแดง ดอกกุหลาบสีเหลือง ผึ้งและเกสรดอกไม้ จะสามารถกำหนดความสัมพันธ์ระหว่างคลาสดอกไม้เหล่านี้ได้ดังต่อไปนี้

- ดอกมะลิเป็นชนิดหนึ่งของดอกไม้
- ดอกกุหลาบเป็นชนิดหนึ่งของดอกไม้
- ดอกกุหลาบสีแดงและดอกกุหลาบสีเหลืองเป็นชนิดหนึ่งของดอกกุหลาบ
- เกสรดอกไม้เป็นส่วนหนึ่งของดอกไม้
- ผึ้งจะกินน้ำหวานจากดอกไม้บางชนิด

จากประโยคตัวอย่างสามารถกำหนดความสัมพันธ์ระหว่างคลาสได้ด้วยเหตุผลใดเหตุผลหนึ่ง อย่างแรกคือ ความสัมพันธ์ระหว่างคลาสเป็นสิ่งบ่งบอกถึงชนิดที่เหมือนกัน ตัวอย่างเช่น ดอกกุหลาบ และดอกมะลิต่างก็เป็นดอกไม้เหมือนกัน หมายความว่า ทั้งดอกมะลิและดอกกุหลาบจะต้องมีเกสรดอกไม้ มีกลีบดอก และส่วนประกอบพื้นฐานอื่นๆ ที่ดอกไม้ต้องมี ส่วนอย่างที่สองคือ ความสัมพันธ์ระหว่างคลาสเป็นสิ่งที่บ่งบอกถึงความเกี่ยวข้องระหว่างคลาส เช่น ดอกกุหลาบสีแดงจะมีความใกล้ชิดกับดอกกุหลาบมากกว่า และดอกกุหลาบก็มีความใกล้ชิดกับดอกไม้มากกว่า และความสัมพันธ์สุดท้ายคือ ผึ้งช่วยในการผสมเกสรดอกไม้ และก็กินน้ำหวานจากดอกไม้ด้วย

จากตัวอย่างสามารถสรุปความสัมพันธ์ระหว่างคลาสได้ 3 แบบใหญ่ๆ คือ ความสัมพันธ์ในการสืบทอดคุณสมบัติ(Generalization/Specialization) ซึ่งเป็นความสัมพันธ์แบบเป็น หรือ “is a” ตัวอย่างเช่น ดอกกุหลาบเป็นดอกไม้ หมายความว่า ดอกกุหลาบเป็นชนิดหนึ่งที่เฉพาะของดอกไม้ ส่วนความสัมพันธ์ระหว่างคลาสแบบที่สองคือ ความสัมพันธ์แบบเป็นส่วนหนึ่ง (Aggregation) หรือ “part of” เช่น เกสรดอกไม้ไม่ได้เป็นชนิดหนึ่งของดอกไม้ แต่เป็นส่วนหนึ่งของดอกไม้ ส่วนความสัมพันธ์ที่สามคือ ความสัมพันธ์แบบแอซโซซิเอต (Associate) ซึ่งเป็นความสัมพันธ์ของคลาสที่บ่งบอกถึงความเกี่ยวเนื่องของคลาสที่ไม่มีความหมายเหมือนกันเลย ตัวอย่างเช่น ดอกกุหลาบกับผึ้ง ซึ่งไม่ได้มีความหมายเหมือนกันเลย แต่ดอกกุหลาบกับผึ้งมีความเกี่ยวข้องกันคือ ผึ้งช่วยผสมเกสรดอกไม้ให้กับดอกกุหลาบ และดอกกุหลาบก็เป็นแหล่งอาหารให้กับผึ้ง เป็นต้น

2.10.1 ความสัมพันธ์แบบแอซโซซิเอชัน (Association)

เมื่อวัตถุหนึ่งใช้บริการของอีกวัตถุหนึ่งแต่ว่าไม่ได้เป็นเจ้าของวัตถุนั้น จะเรียกความสัมพันธ์นี้ว่าแอซโซซิเอชัน ซึ่งความสัมพันธ์แบบนี้จะใช้ได้อย่างเหมาะสมก็ต่อเมื่อ

- วัตถุใช้บริการของอีกวัตถุหนึ่ง แต่ไม่ได้มีความสัมพันธ์แบบเป็นส่วนหนึ่งของ
- วงจรชีวิตของคลาสที่ใช้ไม่ได้เป็นหน้าที่ความรับผิดชอบของคลาสที่เรียกใช้ทั้งการสร้างและทำลายวัตถุ
- ความสัมพันธ์ระหว่างวัตถุนั้นน้อยกว่าความสัมพันธ์แบบเป็นส่วนหนึ่งของ
- ความสัมพันธ์เป็นคุณลักษณะของไคลเอ็นต์-เซิร์ฟเวอร์
- วัตถุมีการถูกเรียกใช้บริการจากหลายๆ วัตถุ และถูกใช้มากเท่าๆ กับที่วัตถุอื่นๆ เรียกใช้

ตัวอย่างแผนภาพคลาสของ Windows นั้นใช้อุปกรณ์อินพุทหลายๆ ชนิด โดยมี 2 ชนิดที่แสดงไว้คือ คีย์บอร์ดและเมาส์ แต่ก็อาจจะมีอุปกรณ์อื่นๆ ที่เป็นอินพุทด้วย เช่น ไมโครโฟนหรือโมเด็ม เป็นต้น

2.10.2 ความสัมพันธ์แบบการสืบทอดคุณสมบัติ(Inheritance)

เมื่อคลาสหนึ่งเป็นความเฉพาะของอีกคลาสหนึ่ง จะใช้ความสัมพันธ์ที่เรียกว่า การสืบทอดคุณสมบัติ ซึ่งหมายความว่า คลาสลูกจะมีคุณสมบัติทุกอย่างที่พ่อแม่มี ถึงแม้ว่าคลาสลูกจะมีความเฉพาะมากกว่าก็ตาม คลาสลูกอาจจะเพิ่มคุณสมบัติของคลาสพ่อแม่ โดยการเพิ่มแอททริบิวต์และเมธอดเข้าไป เรียกความสัมพันธ์นี้ว่า “เป็นชนิดหนึ่งของ” เช่น สัตว์เลี้ยงลูกด้วยนมเป็นชนิดหนึ่งของสัตว์ และอินฟราเรดเซนเซอร์เป็นชนิดหนึ่งของเซนเซอร์

และเพื่อความง่ายในการนำกลับมาใช้ใหม่ ผู้พัฒนาสามารถสร้าวลำดับขั้นตอนของชนิด(Type Hierarchies) ขึ้นมาจากคลาส และความสัมพันธ์ในการสืบทอดคุณสมบัติ

2.10.3 ความสัมพันธ์แบบเป็นส่วนหนึ่งของ (Aggregation)

ความสัมพันธ์แบบเป็นส่วนหนึ่งของนั้น จะใช้ก็ต่อเมื่อมีวัตถุหนึ่งบรรจุอีกวัตถุหนึ่งไว้ทั้งในทางตรรกและทางกายภาพ คลาสที่ใหญ่กว่าเรียกว่าเจ้าของหรือ Owner หรือ Whole ซึ่งเป็นคลาสที่มีหัวลูกศรรูปสี่เหลี่ยมอยู่ ส่วนคลาสที่เล็กว่านั้นเรียกว่า คลาสส่วนประกอบหรือคลาสคอมโพเนนท์(Component) หรือ Part หรือ Owned โดยทั่วไปคลาสเจ้าของมีหน้าที่สร้างและทำลายคลาสคอมโพเนนท์ ใน UML ยอมให้คลาสส่วนประกอบสามารถมีคลาสเจ้าของได้หลายคลาส และเมื่อวัตถุมีหลายเจ้าของ ผู้พัฒนาก็จะพิจารณาว่าคลาสใดจะทำหน้าที่สร้างและทำลายคลาสคอมโพเนนท์นั้น

บทที่ 3

UML (Unified Model Language)

การพัฒนาระบบแบบโครงสร้างนั้น จะพยายามให้นักพัฒนาระบบแก้ปัญหาด้วยการแบ่งปัญหาออกเป็นส่วนๆ แล้วแก้ปัญหาในแต่ละส่วนด้วยอัลกอริทึม (Algorithm) และโครงสร้างข้อมูล (Data Structure) เฉพาะสำหรับการพัฒนาระบบด้วยวิธีเชิงวัตถุก็เช่นเดียวกัน แต่จะมีมุมมองต่อปัญหาเป็นวัตถุประกอบกับการเขียนโปรแกรมด้วยภาษาในการเขียนโปรแกรมเชิงวัตถุ การพัฒนาระบบด้วยวิธีเชิงวัตถุไม่เพียงจะเป็นแนวความคิดใหม่ในการเขียนโปรแกรมเท่านั้น ข้อดีของแบบจำลองเชิงวัตถุยังสามารถนำไปประยุกต์กับการออกแบบฐานข้อมูลเชิงวัตถุได้ เนื่องจากหลักการของการพัฒนาโปรแกรมเชิงวัตถุ เป็นแนวคิดในการจัดการกับความซ้ำซ้อนของสิ่งต่างๆ อย่างมีระบบ ซึ่งสามารถนำไปวิเคราะห์และออกแบบได้อย่างกว้างขวาง

การวิเคราะห์และออกแบบระบบด้วยวิธีการเชิงวัตถุ เป็นวิวัฒนาการ ไม่ใช่เป็นการปฏิวัติการวิเคราะห์และออกแบบระบบที่มีอยู่ เนื่องจากการพัฒนาระบบด้วยวิธีเชิงวัตถุนั้นยังคงข้อดีในกรพัฒนาระบบแบบเก่าแต่ก็ได้เพิ่มข้อดีใหม่ๆ เข้าไปด้วย ในการพัฒนาระบบแบบเก่า จะพบกับความซ้ำซ้อน เนื่องจากการแบ่งระบบออกเป็นส่วนย่อย เมื่อส่วนย่อยเหล่านี้มีมากขึ้น ส่งผลให้การดูแลและการแก้ไขระบบเป็นไปได้ยาก ในขณะที่หลักการของเชิงวัตถุนั้น มีการแบ่งปัญหาออกเป็นวัตถุ ซึ่งมีความผู้พันธ์กันน้อย ทำให้การดูแลและแก้ไขส่วนต่างๆ ของระบบหรือวัตถุสามารถทำได้ง่าย และมีผลกระทบต่อส่วนอื่นๆ ของระบบน้อยที่สุด

แนวความคิดของการพัฒนาระบบด้วยวิธีเชิงวัตถุเกิดขึ้นครั้งแรกเมื่อประมาณปี 1970 โดยเริ่มจากการออกแบบสถาปัตยกรรมฮาร์ดแวร์ของคอมพิวเตอร์ นักพัฒนาฮาร์ดแวร์ได้ใช้แนวคิดขอการพัฒนาระบบด้วยวิธีเชิงวัตถุช่วยในการลดช่องว่างระหว่างการแยกแยะเอกลักษณ์ในระดับสูง ซึ่งเป็นระดับการเขียนโปรแกรม และแยกเอกลักษณ์ในระดับล่างเป็นระดับการทำงานของฮาร์ดแวร์ ด้วยการนำหลักการของการพัฒนาระบบด้วยวิธีเชิงวัตถุมาใช้ในการออกแบบและพัฒนา สามารถตรวจสอบความผิดพลาดได้ง่าย-ช่วยเพิ่มประสิทธิภาพในการปฏิบัติคำสั่ง ลดชนิดของคำสั่งของฮาร์ดแวร์ มีการคอมไพล์ภาษาระดับสูงเป็นภาษาระดับล่างที่ไม่ซับซ้อน และยังช่วยลดพื้นที่ในการเก็บข้อมูลด้วย

UML เป็นภาษามาตรฐานทางอุตสาหกรรมผลิตโปรแกรมประยุกต์ ใช้สำหรับแสดงรายละเอียด จำลอง สร้าง และจัดการเอกสารต่างๆ ในการผลิตโปรแกรมประยุกต์ โดยทำให้การออกแบบโปรแกรมประยุกต์หรือการสร้างพิมพ์เขียวทำได้ง่าย นิยามโดย Grady Booch, Ivar Jacobson และ Jim Rumbaugh

3.1 มุมมองของ UML

ในการออกแบบระบบที่มีขนาดใหญ่และมีความซับซ้อนมากๆ นั้นจะทำให้ผู้ออกแบบระบบไม่สามารถที่จะออกแบบระบบได้ครบถ้วน ดังนั้นจึงต้องมีการมองระบบเป็นมุมมองต่างๆ เพื่อทำให้ง่ายในการออกแบบ ดังนั้นระบบจึงมี View ที่ต่างๆกัน ซึ่งแต่ละ View จะแสดงมุมมองเฉพาะของระบบซึ่งอธิบายรวมกันเป็นระบบที่สมบูรณ์ ซึ่งจะประกอบด้วย View ต่างๆดังนี้

3.2 มุมมองยูซเคส(Use Case View)

อธิบายการทำงานต่างๆ ของระบบที่ถูกมองจากภายนอกหรือผู้ใช้ระบบ Use-case view ซึ่งอธิบายโดย ยูซเคสไดอะแกรม (Use-case diagram) เป็นมุมมองสำหรับลูกค้า , ผู้ออกแบบ , ผู้พัฒนาระบบ และ ผู้ทดสอบระบบ

3.2.1 มุมมองทางโลจิก(Logical View)

อธิบายการทำงานต่างๆที่ถูกออกแบบไว้ภายในระบบ ว่าระบบจะมีบริการอะไรให้กับผู้ใช้บ้าง โดยจะแสดงโครงสร้างแบบสถิต (dynamic collaboration) ซึ่งจะเกิดขึ้นเมื่อออบเจ็กต์ส่งแมสเซส ระหว่างกันในการทำงาน

โครงสร้างแบบสถิตจะอธิบายโดยใช้ คลาสไดอะแกรม (Class diagram) และออบเจ็กต์ไดอะแกรม (Object diagram) ส่วนการทำงานร่วมกันแบบไดนามิกจะอธิบายโดยใช้สเตทไดอะแกรม(State diagram), ซีเควนซ์ไดอะแกรม (Sequence diagram), คอลแลโบเรชั่นไดอะแกรม (Collaboration diagram) และ แอ็คทิวิตี้ไดอะแกรม (Activity diagram)

3.2.2 มุมมองคอมโพเนนท์(Component View)

อธิบายการสร้างและความขึ้นต่อกันของโมดูล (Module) ที่ใช้ในการพัฒนาระบบ โดยใช้คอมโพเนนท์ไดอะแกรม (Component diagram) ในการอธิบาย

3.2.3 มุมมองดีพลอยเมนท์(Deployment View)

อธิบายการจัดวางระบบให้เหมาะสมในด้านกายภาพ (Physical) แสดงด้วยคอมพิวเตอร์และโหนด (nodes) ต่างๆ เพื่อให้ระบบมีเสถียรภาพมากขึ้นโดยใช้ ดีพลอยเมนท์ไดอะแกรม (Deployment Diagram) ในการอธิบาย

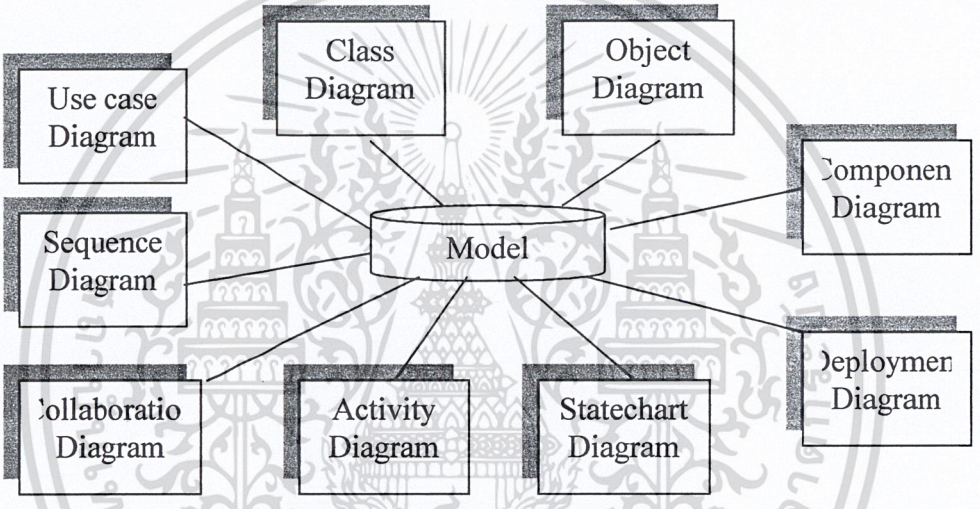
3.2.4 มุมมองโพรเซส(Process View)

แสดงการทำงานร่วมกันและการติดต่อกันของส่วนต่างๆในระบบ

3.3 แผนภาพของ UML

UML ประกอบด้วยไดอะแกรมทั้งหมด 8 ไดอะแกรม ดังแสดงใน รูปที่ 3.19 เพื่อใช้ในการจำลองระบบงาน เปรียบได้กับการมองในแง่มุมต่างๆ เพื่อให้สามารถเข้าใจระบบงานให้มากที่สุด โดยที่ผู้จำลองแบบไม่จำเป็นต้องใช้ทุกไดอะแกรม โดยสามารถเลือกใช้ไดอะแกรมที่เหมาะสม

ในปริญญานิพนธ์นี้สนใจเฉพาะไดอะแกรมที่ใช้ในการออกแบบแอปพลิเคชั่นเท่านั้นคือ ยูซเคสไดอะแกรม, ซีเควนซ์ไดอะแกรม, คอลแลโบเรชั่นไดอะแกรม, คลาสไดอะแกรม, อ็อบเจ็คไดอะแกรม, สเตทไดอะแกรม และแอ็คติวิตี้ไดอะแกรม



รูปที่ 3-1 แสดง แผนภาพแสดงไดอะแกรมของ UML ทั้งหมด

3.3.1 ยูสเคสไดอะแกรม(Use Case diagram)

ยูซเคสไดอะแกรมแสดงถึงตัวกระทำ(Actors), ยูซเคส และความสัมพันธ์ระหว่างยูซเคสกับตัวกระทำยูซเคสใช้แทนการทำงานที่ระบบทำได้ เช่นระบบย่อยหรือคลาส โดยมองจากตัวกระทำภายนอกที่มีการติดต่อกับระบบ

แต่ละยูซเคสไดอะแกรมคือกราฟของตัวกระทำ, ยูซเคส, ส่วนติดต่อหรืออินเตอร์เฟส(Interface)(ถ้ามี) และความสัมพันธ์ระหว่างสิ่งเหล่านี้

ยูสเคส(Use Case)

ยูซเคสใช้แทนหน่วยการทำงานที่ระบบ หรือระบบย่อย หรือคลาสมีให้ ซึ่งแสดงให้เห็นโดยการแลกเปลี่ยนข้อมูลระหว่างระบบกับตัวกระทำภายนอก

จุดเอ็กเทนชั่น(extension point) เป็นจุดอ้างอิงภายในยูซเคสซึ่งอาจมีการทำงานของยูซเคสอื่นเข้ามาแทรกได้ แต่ละจุดเอ็กเทนชั่นจะมีชื่อที่แตกต่างกันในยูซเคสหนึ่งๆและมีคำบรรยายตำแหน่งการทำงานภายในพฤติกรรมของยูซเคส

ตัวกระทำ(Actor)

คือเซตของบทบาทที่เหมือนกันของผู้ใช้ เช่น บทบาทเป็นพนักงานขาย

ตัวกระทำจะถูกเขียนแทนด้วยรูปสติ๊กแมน(ดูตัวอย่าง “Salesperson” ในรูปที่???)

ความสัมพันธ์ของยูสเคส(Use Case Relationship)

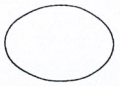
ความสัมพันธ์(Relationship)มาตรฐานระหว่างตัวกระทำกับยูซเคสและระหว่างยูซเคสกับยูซเคสมีอยู่หลายแบบ

แอสโซสิเอชั่น– Association: การกระทำร่วมกันระหว่างตัวกระทำกับยูซเคสแสดงในไดอะแกรมโดยเส้นตรง

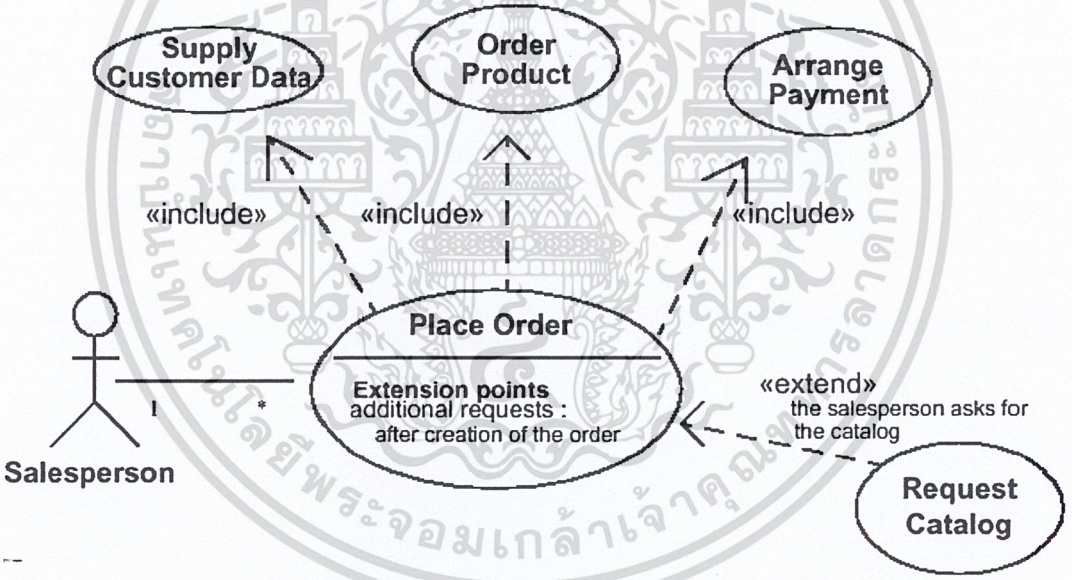
เอ็กเท็นด์ – Extend:ความสัมพันธ์แบบขยาย(Extend) ระหว่างยูซเคส A กับยูซเคส B หมายความว่าตัวอย่าง(Instance) ของ B อาจถูกเอ็กเท็นด์ ดยพฤติกรรมที่กำหนดใน A แสดงในไดอะแกรมโดยเส้นประที่มีลูกศรหัวเปิดจากยูซเคสที่เป็นตัวให้การขยายไปยัง use case ที่เป็นฐานโดยจะมีคีย์เวิร์ด <<extend>> กำกับไว้ ส่วนเงื่อนไขของความสัมพันธ์อาจใส่ไว้ใกล้กับคีย์เวิร์ด <<extend>>

เจนเนอรัลไลเซชั่น – Generalization: ความสัมพันธ์แบบเจนเนอรัลไลเซชั่นจากยูซเคส A ไปยัง ยูซเคส B หมายถึง A เป็นพ่อแม่(parent) หรืออยู่เหนือ B แสดงในไดอะแกรมโดยเส้นตรงที่มีหัวลูกศรปิดและกลวง หัวลูกศรชี้ที่ยูซเคสที่เป็นพ่อ ความสัมพันธ์แบบเจนเนอรัลไลเซชั่นจากตัวกระทำ A ไปยังตัวกระทำ B หมายความว่าตัวกระทำ A สามารถติดต่อกับยูสเคสเดียวกับที่ตัวกระทำ B สามารถติดต่อได้

อินคลูด – Include: ความสัมพันธ์แบบรวมเข้าด้วยกันจากยูซเคส A ไปยังยูซเคส B หมายถึงภายใน A อาจรวมเอาพฤติกรรมที่ระบุโดย B เข้าไปด้วย แสดงในไดอะแกรมโดยประที่มีลูกศรหัวเปิดจากยูซเคสที่เป็นฐานไปยังยูซเคสที่ถูกรวมเข้าด้วยกัน โดยจะมีคีย์เวิร์ด <<include >> กำกับ

	Use case
	ตัวกระทำหรือ Actor
	Association
	Extend
	Include
	Generalization

ตารางที่ 3-1 แสดงสัญลักษณ์ที่ใช้ในยูซเคสไดอะแกรม



รูปที่ 3-2 แสดงตัวอย่างยูซเคสไดอะแกรม

3.3.2 ซีเควนซ์ไดอะแกรม(Sequence Diagram)

ซีเควนซ์ไดอะแกรมใช้แสดงแทนการปฏิสัมพันธ์(Interaction) ระหว่างอ็อบเจ็คในการทำงานให้ได้ตามจุดประสงค์หนึ่ง ซีเควนซ์ไดอะแกรมอาจใช้เพื่ออธิบายการทำงาน(Operation) หรือยูสเคส

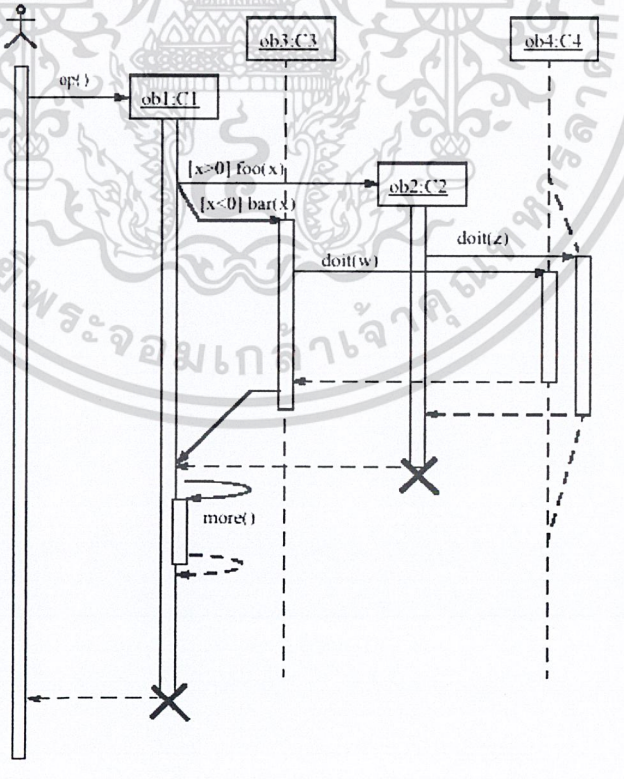
ซีเควนซ์ไดอะแกรมมี 2 มิติ คือ มิติในแนวตั้งแทนเวลา และมิติในแนวนอนแทนอ็อบเจ็คต่างๆ (ลำดับการเรียงของอ็อบเจ็คไม่มีผลต่อไดอะแกรม) มีการใช้ลูกศรชนิดต่างๆแทน “แมสเสจและตัวกระตุ้น” (Message and Stimulus)

เส้นชีวิตของอ็อบเจ็ค(Object Lifeline)

แสดงการมีอยู่ของอ็อบเจ็คในช่วงเวลาหนึ่งเขียนแทนด้วยเส้นประในแนวตั้ง ถ้าอ็อบเจ็คถูกสร้างขึ้นและถูกทำลายในระหว่างช่วงเวลาในไดอะแกรมแล้ว เส้นชีวิต(Lifeline) ของมันจะต้องเริ่มและสิ้นสุดลงอย่างเหมาะสม ถ้าอ็อบเจ็คถูกสร้างขึ้นในไดอะแกรมลูกศรที่เป็นตัวกระตุ้นให้สร้างอ็อบเจ็คจะชี้ไปที่สัญลักษณ์อ็อบเจ็คของอ็อบเจ็คที่ถูกสร้าง จากรูปด้านบนคือเส้น op() ที่สร้างอ็อบเจ็ค ob1 และถ้าอ็อบเจ็คถูกทำลายระหว่างช่วงเวลาในไดอะแกรม การถูกทำลายจะแทนด้วยเครื่องหมาย "X" ที่ลูกศรที่สั่งให้ทำลายอ็อบเจ็คนั้นหรือที่ลูกศรออกจากอ็อบเจ็บสุดท้ายในกรณีที่อ็อบเจ็คทำลายตัวเอง

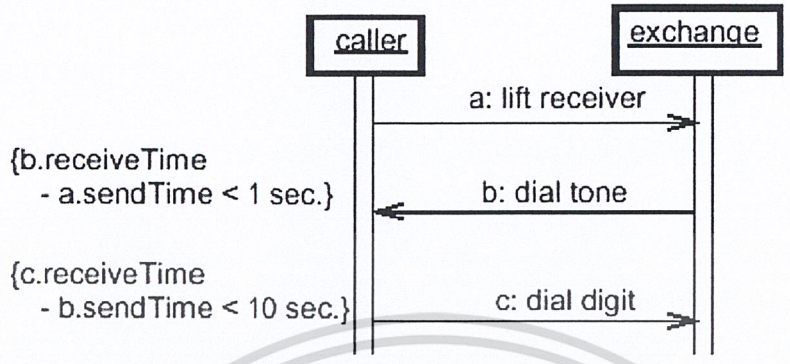
แอ็คติเวชั่น(Activation)

แสดงช่วงเวลาที่อ็อบเจ็คมีการทำงาน ใช้แทนการทำงานของอ็อบเจ็คในช่วงเวลาหนึ่ง รวมทั้งความสัมพันธ์ของการควบคุมระหว่างช่วงการทำงานและผู้เรียกให้ทำงาน เขียนแสดงสี่เหลี่ยมบางๆซึ่งด้านบนคือจุดเวลาเริ่มต้นและด้านล่างคือจุดสิ้นสุด ในการไหลของการควบคุมแบบโพรซีเยอร์ด้านบนของสัญลักษณ์แอ็คติเวชั่นจะอยู่ที่หัวลูกศรที่สั่งให้เกิดการทำงานและด้านล่างคือหางลูกศรของการออกจากโพรซีเยอร์ ในกรณีที่มีการเรียกอ็อบเจ็คที่มีแอ็คติเวชั่นอยู่แล้วซ้ำ จะมีอีกแอ็คติเวชั่นหนึ่งเพิ่มขึ้นมาทางด้านขวาของอันแรก(ดูตัวอย่างได้จากรูปที่ ??? ที่อ็อบเจ็ค ob4) การเรียกหาตัวเอง(Recursion)



รูปที่ 3-3 แสดงตัวอย่างซีเควนซ์ไดอะแกรมที่มีการควบคุมการทำงาน, การใช้เงื่อนไข, การสร้างและการทำลายทรานสิชั่นไทม์(Transition Times)

อาจระบุเวลาแบบต่างๆให้กับแมสเสจ เช่น “sending time” หรือ “receiving time”ซึ่งสามารถนำไปใช้เพื่อการบอกอายุของแมสเสจได้



รูปที่ 3-4 แสดงตัวอย่างทรานสิชั่นไทม์

แมสเสจและตัวกระตุ้น(Message and Stimulus)

ตัวกระตุ้น(Stimulus) คือการสื่อสารระหว่างสองอ็อบเจ็คซึ่งส่งข้อมูลโดยคาดว่าจะเกิดการกระทำขึ้นตัวกระตุ้นจะเรียกให้เกิดการทำงาน การสร้างสัญญาณ(Signal) หรือทำให้อ็อบเจ็คถูกสร้างหรือถูกทำลาย

แมสเสจ(Message) คือการกำหนดของตัวกระตุ้น เช่น ระบุบทบาทซึ่งอ็อบเจ็คตัวส่งและรับแมสเสจจะต้องทำงานให้สอดคล้องตาม หรือการกระทำ(Action)ซึ่งเมื่อถูกสั่งให้ทำแล้วจะส่ง Stimulus ซึ่งสอดคล้องกับแมสเสจออกมา

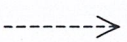
ตัวกระตุ้นจะถูกเขียนแทนด้วยลูกศรจากเส้นชีวิต(Lifeline) ของอ็อบเจ็คหนึ่งไปยังเส้นชีวิตของอีกอ็อบเจ็คหนึ่งหรือกลับเข้าหาตัวเองก็ได้ ลูกศรจะถูกกำกับด้วยชื่อของตัวกระตุ้น (การทำงาน(Operation)หรือสัญญาน) พร้อมด้วยอาร์กิวเมนท์

ชนิดของลูกศรแบบต่างๆ

 การเรียกโพรซีเยอร์หรือกลุ่มการควบคุมหนึ่ง การทำงานภายในทั้งหมดต้องเสร็จสิ้นก่อนที่กลับไปทำงานในระดับด้านนอก

 แสดงการเข้าสู่ลำดับต่อไปในลำดับการทำงาน

 แสดงตัวกระตุ้นแบบ Asynchronous ใช้แสดงการสื่อสารระหว่างสองอ็อบเจ็คแบบ Asynchronous ในลำดับการทำงานของโพรซีเยอร์

 การออกจากโพรซีเยอร์

การแยกการทำงาน – Branching: แสดงโดยลูกศรหลายเส้นออกจากจุดเดียวกัน แต่ละเส้นกำกับโดยเงื่อนไขการเลือกทำโดยทุกเส้นจะต้องแตกต่างกัน(เลือกได้ทางเดียว)

การทำงานซ้ำ – Iteration: กลุ่มของลูกศรที่เชื่อมกันนำมารวมกันและกำกับไว้ว่าเป็น Iteration แสดงถึงการส่งกลุ่มของ Stimuli สามารถเกิดได้หลายครั้ง

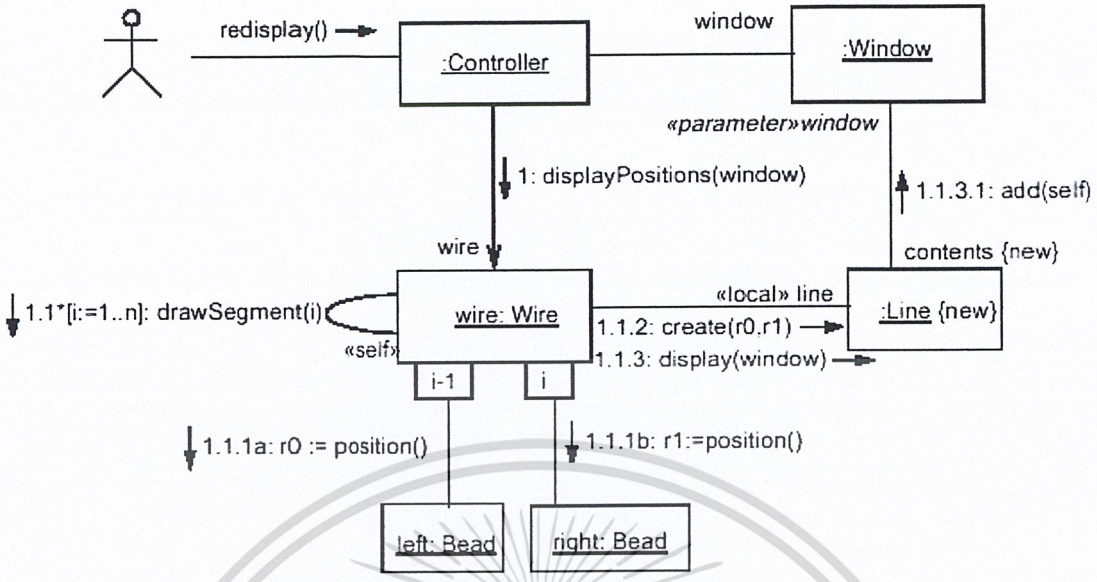
3.3.3 คอลแลโบเรชั่นไดอะแกรม(Collaboration Diagram)

คอลแลโบเรชั่นไดอะแกรมแสดงแทนคอลแลโบเรชั่นซึ่งประกอบด้วยเซ็ตของบทบาบของอ็อบเจ็คและความสัมพันธ์ระหว่างอ็อบเจ็ค ไดอะแกรมนี้อาจใช้แสดงการปฏิสัมพันธ์ระหว่างอ็อบเจ็คโดยการระบุเซตของแมสเสจที่บอกการปฏิสัมพันธ์ระหว่างอ็อบเจ็คที่มีบทบาทอยู่ภายในคอลแลโบเรชั่นเพื่อบรรลุจุดประสงค์หนึ่ง

คอลแลโบเรชั่นไดอะแกรมจะแสดงออกมาเป็นกราฟที่เชื่อมต่อระหว่างอ็อบเจ็ค คอลแลโบเรชั่นไดอะแกรมอาจใช้เพื่ออธิบายการทำงานหรือคลาสสิฟายเออร์หรือยูสเคส และยังช่วยในการออกแบบโพรซีเยอร์ มีการใช้ลูกศรเพื่อบอกการทำงานโดยตัวเส้นลูกศรแทนตัวกระตุ้นซึ่งถูกส่งไปในทิศทางที่หัวลูกศรชี้ มีหมายเลขกำกับตามลำดับการปฏิสัมพันธ์โดยเริ่มต้นด้วย 1

ส่วนประกอบในไดอะแกรมส่วนใหญ่จะเหมือนกับซีเควนซ์ไดอะแกรมต่างกันที่รูปแบบการเขียน โดยคอลแลโบเรชั่นไดอะแกรมจะไม่มีแกนเส้นชีวิตและแอ็กติเวชั่นเนื่องจากสนใจเฉพาะลำดับการส่งตัวกระตุ้น

ทั้งซีเควนซ์ไดอะแกรมและคอลแลโบเรชั่นไดอะแกรมต่างก็ใช้บอกการปฏิสัมพันธ์ โดยซีเควนซ์ไดอะแกรมแสดงลำดับของตัวกระตุ้นและสามารถเข้าใจได้ง่ายในการพิจารณาด้านเวลา ส่วนคอลแลโบเรชั่นไดอะแกรมจะแสดงความสัมพันธ์ตัวอย่าง(Instance) ช่วยให้ง่ายต่อการเข้าใจผลกระทบต่างๆที่เกิดขึ้นกับตัวอย่างนั้นๆและช่วยในการออกแบบโพรซีเยอร์



รูปที่ 3-5 แสดงตัวอย่างไดอะแกรม Collaboration

3.3.4 คลาสไดอะแกรม(Class Diagram)

คลาสไดอะแกรม(Class Diagram) คือกราฟของคลาสสิฟายเออร์(Classifier) ซึ่งเชื่อมต่อกันด้วยความสัมพันธ์คงที่(Static Relationship) คลาสไดอะแกรมยังอาจมี อินเตอร์เฟส(Interfaces), แพคเกจ(Packages), ความสัมพันธ์(Relationship) หรือแม้แต่อ็อบเจ็คและลิงก์

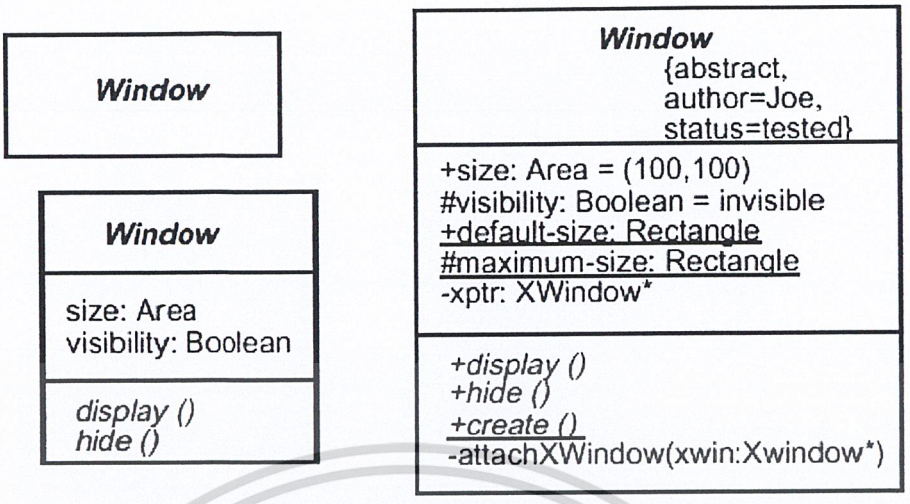
คลาสสิฟายเออร์(Classifier)

คือ ซูเปอร์คลาสของคลาส(*Class*), อินเตอร์เฟส และ ดาต้าไทป์(*Datatypes*) ในการเขียนไดอะแกรมส่วนมากจะใช้สี่เหลี่ยมแทน คลาส(Class) ส่วนอินเตอร์เฟสและดาต้าไทป์จะใช้สี่เหลี่ยมโดยมีคีย์เวิร์ดกำกับไว้

คลาส(Class)

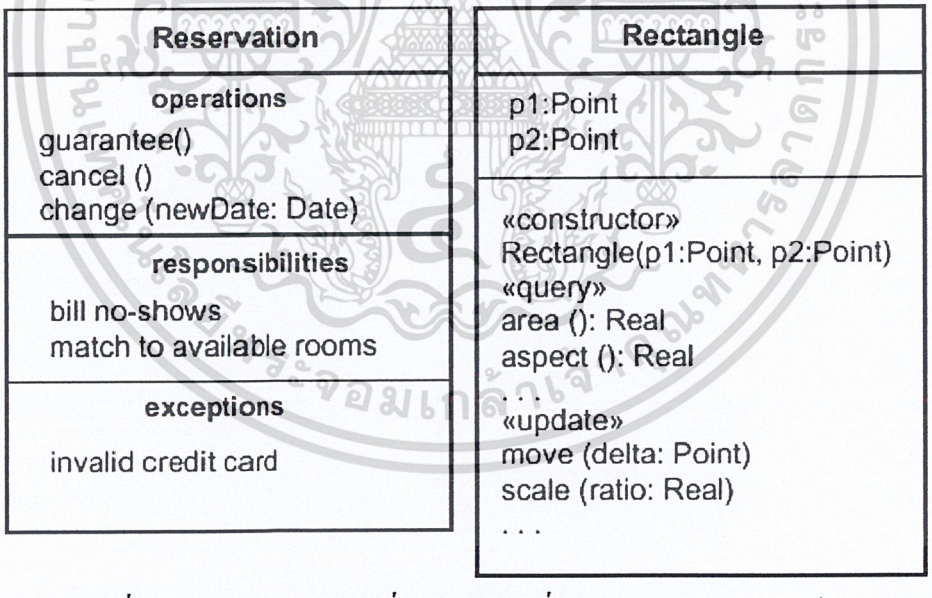
เขียนแทนในไดอะแกรมด้วยสี่เหลี่ยม ภายในแบ่งตามแนวนอนเป็น 3 ส่วน ส่วนบนคือชื่อคลาส คุณสมบัติทั่วไปของคลาสและสเตอริโอไทป์(Stereotype)เรียกว่า "Name Compartment" ส่วนกลางคือรายการแอททริบิวท์ที่มีในคลาส ส่วนล่างคือรายการการทำงาน(Operation) ที่มีในคลาส

ถ้ามีการอ้างคลาสข้ามแพคเกจจะเขียนชื่อคลาสในรูป Package-name::Class-name



รูปที่ 3-6 แสดงตัวอย่างคลาส: แบบซ่อนรายละเอียด, แบบแสดงรายละเอียดระดับวิเคราะห์ และแสดงรายละเอียดระดับการ implementation

อาจมีการกำหนดส่วน(Compartment) ในคลาสเพิ่มขึ้นมาตามที่ผู้ใช้ต้องการ เช่นกำหนด "requirement compartment" รวมทั้งอาจมีการใช้สเตอริโอไทป์



รูปที่ 3-7 แสดงตัวอย่างคลาสที่กำหนดส่วนเพิ่ม และการใช้สเตอริโอไทป์

แอททริบิวท์(Attributes)

เขียนในรูป

visibility name [*multiplicity*] : *type-expression* = *initial-value* { *property-string* }

โดย visibility สามารถเป็นได้ 3 แบบ

+ public visibility

protected visibility

- private visibility

เช่น

+size: Area = (100,100)

#visibility: Boolean = invisible

การทำงาน(Operation)

เขียนในรูป

visibility name (*parameter-list*) : *return-type-expression* { *property-string* }

โดย visibilityสามารถเป็นได้ 3 แบบ

+ public visibility

protected visibility

- private visibility

เช่น

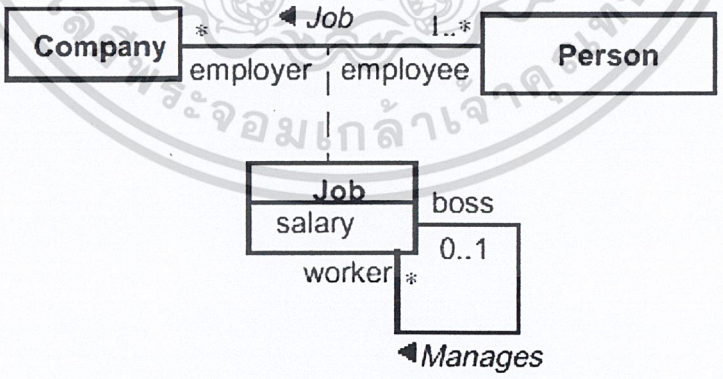
+create ()

+display (): Location

+hide ()

แอสโซสิเอชั่น(Association)

ไบนารีแอสโซสิเอชั่น(Binary Association) แสดงความเกี่ยวข้องกันของสองคลาสสิฟายเออร์ เขียนแทนด้วยเส้นตรงเชื่อมระหว่างสัญลักษณ์ของคลาสสิฟายเออร์สองอัน หรือเชื่อมคลาสสิฟายเออร์หนึ่งเข้ากับตัวเอง อาจมีชื่อของแอสโซสิเอชั่นกำกับไว้ด้วย



รูปที่ 3-8 แสดงตัวอย่างแอสโซสิเอชั่น

ที่ปลายเส้นของแอสโซสิเอชั่นสามารถมีลูกศรเพื่อบอกทิศทางของแอสโซสิเอชั่นคือบอกว่าเป็นแอสโซสิเอชั่นจากคลาสใดไปสู่คลาสใด

มัลติพลิซิตี้(Multiplicity)

มัลติพลิซิตี้แสดงพิกัดจำนวนที่เซ็ตนั้นๆจะมีได้ สามารถนำไปใช้ประกอบกับแอสโซสิเอชั่นได้ ตัวอย่างของมัลติพลิซิตี้เช่น

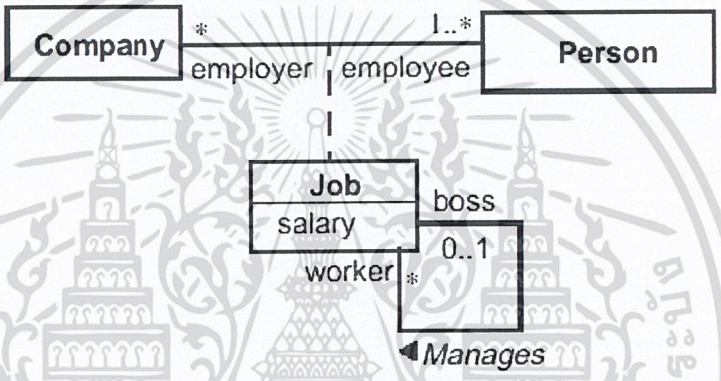
0..1 มีจำนวนได้ 0 ถึง 1

0..* มีจำนวนได้ตั้งแต่ 0 ขึ้นไป

* ไม่จำกัด

แอสโซสิเอชั่นคลาส(Association Class)

แอสโซสิเอชั่นคลาสคือแอสโซสิเอชั่นที่มีคุณสมบัติของคลาส เขียนแทนโดยรูปแบบสี่เหลี่ยมแบบเดียวกับคลาสพร้อมกับมีเส้นประเชื่อมเข้ากับแอสโซสิเอชั่น



รูปที่ 3-9 แสดงตัวอย่างแอสโซสิเอชั่นคลาส

แอสโซสิเอชั่นแบบ n-ary (n-ary Association)

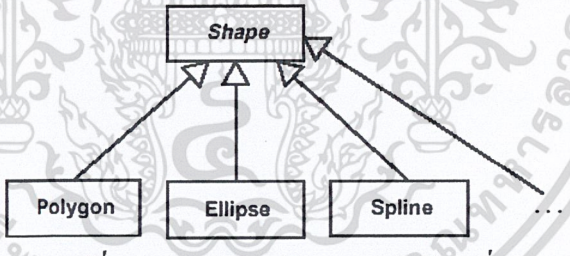
แอสโซสิเอชั่นแบบ n-ary แสดงความเกี่ยวข้องของคลาสสิฟายเออร์ตั้งแต่สามคลาสสิฟายเออร์ขึ้นไป เขียนแทนโดยรูปสี่เหลี่ยมข้าวหลามตัดโดยมีเส้นแอสโซสิเอชั่นเชื่อมไปถึงคลาสที่เกี่ยวข้อง ถ้าต้องการใส่ชื่อให้ก็สามารถเขียนไว้ข้างสี่เหลี่ยมข้าวหลามตัดนั้น

รูปที่ 3-10 แสดงตัวอย่างแอสโซสิเอชั่นแบบ n-ary

เจนเนอรัลไลเซชั่น(Generalization)

เจนเนอรัลไรเซชั่นคือความสัมพันธ์ระหว่างสิ่งที่เป็นกลางมากกว่าหรือเป็นพ่อ(parent) กับสิ่งที่เฉพาะเจาะจงหรือลูก(child)ซึ่งจะมีทุกอย่างของพ่อรวมทั้งข้อมูลที่เพิ่มเติมเข้าไป เจนเนอรัลไรเซชั่นถูกนำไปใช้กับคลาส, แพคเกจ และ ยูสเคส เป็นต้น

เจนเนอรัลไรเซชั่นเขียนในไดอะแกรมโดยเส้นตรงที่มีหัวลูกศรปิดกลวงโดยที่หัวลูกศรชี้ไปที่สิ่งที่เป็นพ่อ

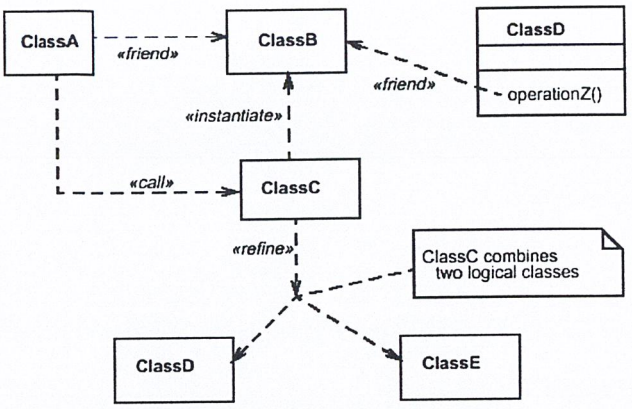


รูปที่ 3-11 แสดงตัวอย่างเจนเนอรัลไลเซชั่น

ดีเพนเดนซี(Dependency)

ดีเพนเดนซีระบุความสัมพันธ์ระหว่างสองสิ่งในโมเดล บอกให้รู้ว่าการเปลี่ยนแปลงสิ่งที่เป็นเป้าหมายอาจต้องการการเปลี่ยนแปลงของสิ่งที่เป็นต้นทางของดีเพนเดนซี

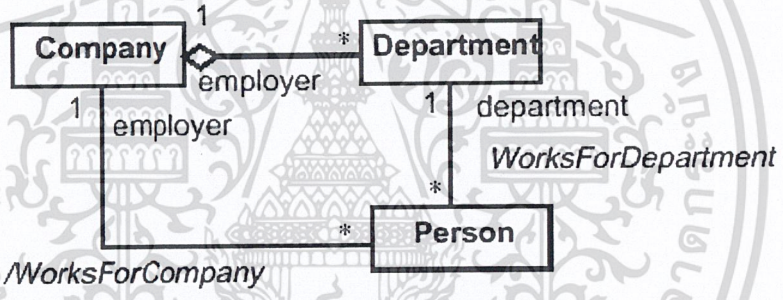
ดีเพนเดนซีเขียนแทนด้วยเส้นประโดยสิ่งที่อยู่ที่หางลูกศรจะขึ้นอยู่กับสิ่งที่อยู่ที่หัวลูกศร อาจมีการใส่ชื่อและสเตอริโอไทป์ให้กับลูกศรได้



รูปที่ 3-12 แสดงตัวอย่างดีเพนเดนซี

อะกรีเกชั่น(Aggregation)

เป็นแอสโซสิเอชั่นที่ระบุว่ามีการรวมกันหรือเป็นส่วนหนึ่งเกิดขึ้น เขียนแทนแบบเดียวกับแอสโซสิเอชั่นแต่ที่ปลายเส้นด้านที่จะเข้าไปรวมจะเปลี่ยนเป็นรูปสี่เหลี่ยมข้าวหลามตัด



รูปที่ 3-13 ตัวอย่างอะกรีเกชั่น

3.3.5 สเตทไดอะแกรม(State Diagram)

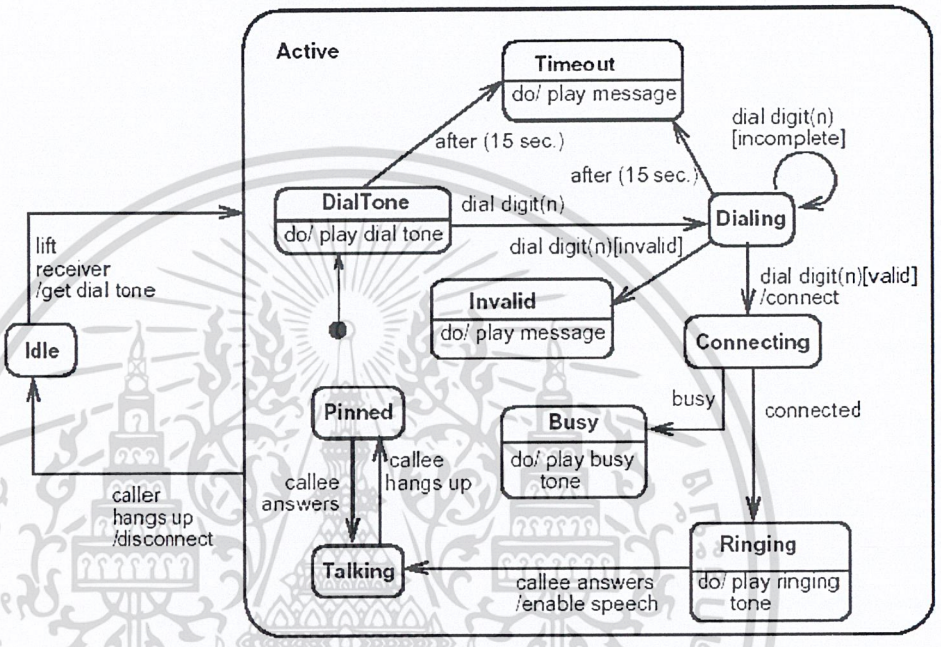
สเตทไดอะแกรม(State Diagram) หรือ สเตทชาร์ทไดอะแกรม(Statechart Diagram) หรือบางครั้งเรียกว่า สเตทแมชชีน(State Machine) เป็นสิ่งที่ใช้แสดงการเปลี่ยนสเตท (state) ของออบเจ็กต์ตั้งแต่เริ่มต้นจนสิ้นสุดการเปลี่ยนแปลงในรอบหนึ่งๆ (1 ซีเควนท์) โดยทั่วไปแล้ว สเตทไดอะแกรม (State diagram) จะใช้ในการอธิบายพฤติกรรมของคลาสแต่ละคลาส แต่ก็สามารถนำมาอธิบายพฤติกรรมของไดอะแกรมหรือโมเดลอื่นๆ ได้ด้วยเช่นกัน เช่น ยูซ-เคส(Use-cases), แอ็กเตอร์(Actors), ระบบย่อย (Subsystems), โอเปอเรชัน(Operations) และเมธอด(Methods) เป็นต้น

ลักษณะของการเปลี่ยนสเตท เช่น

- เมื่อเรากดสวิทช์ไฟ หลอดไฟฟ้าจะเปลี่ยนสถานะจากมืดเป็นสว่าง
- เมื่อเรากดปุ่มรีโมทคอนโทรลเพื่อเปลี่ยนช่อง โทรทัศน์จะเปลี่ยนสถานะจากรายการช่องหนึ่งเป็นรายการอีกช่องหนึ่ง

- เมื่อเวลาผ่านไประยะหนึ่ง เครื่องซักผ้าจะเปลี่ยนสถานะจาก “ซัก” เป็น “ปั่น”

สเตทของออบเจ็กต์ ณ เวลาใดเวลาหนึ่ง จะมีการเปลี่ยนแปลงสถานะเมื่อมีการกระทำจากภาพนอก เช่น หลอดไฟจะมืดหรือสว่างได้ก็ต่อเมื่อเราปิดหรือเปิดสวิทช์ไฟ เป็นต้น แต่มีออบเจ็กต์บางอย่างสามารถเปลี่ยนสถานะได้ด้วยตัวเอง เราจะเรียกออบเจ็กต์ดังกล่าวว่า “Object with life” หรือ “Actor” เช่น นาฬิกา ลูกค้า เป็นต้น ตัวอย่างของ สเตทไดอะแกรมแสดงได้ดังรูป



รูปที่ 3-14 แสดงตัวอย่างสเตทไดอะแกรม (State Diagram)

สเตท(State)

สเตทเป็นเงื่อนไขระหว่างการทำงานของออบเจ็กต์ หรือเป็นสิ่งที่แสดงการตอบสนองเงื่อนไข การกระทำบางอย่าง หรือ รอคอยเพื่อรับเหตุการณ์เข้ามา สัญลักษณ์ที่ใช้แสดงสเตทจะเป็นรูปสี่เหลี่ยมหัวมน โดยภายในจะประกอบด้วยชื่อสเตทซึ่งจะถูกแบ่งด้วยเส้นภายในสเตทนั้นๆ และภายในสเตทยังสามารถถูกแบ่งออกเป็นส่วนย่อยๆ อื่นได้อีก โดยใช้เส้นตรงในการแบ่ง ซึ่งมีดังต่อไปนี้

- ส่วนของชื่อสเตท (Name compartment)
- ส่วนแสดงทรานสิชันภายใน (Internal transition compartment) ในส่วนนี้จะแสดงรายการทำงานทั้งหมด (action) ที่สามารถถูกกระทำได้ในสเตทนั้นๆ โดยแสดงได้ดังนี้

action-label ‘/’ action-expression

โดยที่จะมีชื่อแอ็กชั่นหลักๆ ที่ถูกสงวนไว้เพื่อจุดประสงค์หลักๆ ซึ่งไม่สามารถนำไปใช้เป็นชื่อของเหตุการณ์ได้ มีดังต่อไปนี้

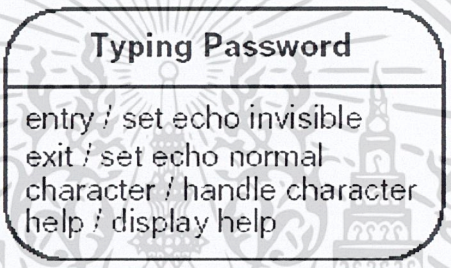
- entry เป็นส่วนแสดงแอ็กชั่น จะอธิบายการทำงานเมื่อมีการเข้าสู่สเตทนี้
- exit เป็นส่วนแสดงแอ็กชั่น จะอธิบายการทำงานเมื่อมีการออกจากสเตทนี้

- do เป็นส่วนที่แสดงถึงกิจกรรม ที่ถูกกระทำภายในสเตทนี้ ซึ่งกิจกรรมนี้จะถูกกระทำไปจนกว่าที่ action-expression จะเสร็จสมบูรณ์
- include ส่วนนี้เป็นส่วนที่ใช้เรียกสเตทย่อย ซึ่ง action-expression จะประกอบด้วยชื่อของสเตทย่อยที่จะถูกเรียกนั้น

รูปแบบทั่วไปของรายการในทรานสิชันภายใน มีดังต่อไปนี้

event-name '(' comma-separated-parameter-list ')' '[' guard-condition']' '/' action-expression

โดยที่ event-name สามารถที่จะปรากฏได้หลายครั้งต่อหนึ่งสเตท ถ้าการ์ด คอนดิชัน (guard conditions) แตกต่างกัน ตัวอย่างของสเตทแสดงได้ดังรูป



รูปที่ 3-15 แสดงตัวอย่างสเตท (State)

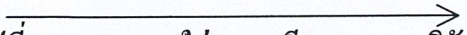
เราสามารถเอาสเตทการทำงานมาเชื่อมโยงถึงกันได้ด้วยกลไกที่เรียกว่า "ทรานสิชันไลน์" (Transition line) โดยใช้สัญลักษณ์เป็นลูกศร ลักษณะของการเชื่อมโยงระหว่างสเตทด้วยทรานสิชันไลน์จะมีลักษณะดังรูป

รูปที่ 3-16 แสดงการเชื่อมสเตทด้วยทรานสิชันไลน์ (Transition line)

ดังที่กล่าวมาแล้วว่า ทรานสิชันไลน์คือองค์ประกอบหนึ่งของสเตทไดอะแกรม โดยจะเชื่อมโยงสเตท สถานะการทำงานต่างๆ รวมทั้งแสดงให้เห็นถึงลำดับการทำงานด้วย ซึ่งในส่วนของทรานสิชันไลน์นั้น เราสามารถใส่รายละเอียดบางอย่างเพิ่มเติมเข้าไปได้ กล่าวคือ เราอาจจะมีการใส่เหตุการณ์ที่ทำให้เกิดทรานสิชัน (Transition) หนึ่งๆ ขึ้น ซึ่งเราเรียกเหตุการณ์นั้นว่า "ทริกเกอร์อีเวนต์" (Trigger event)

นอกจากนี้เรายังสามารถใส่การคำนวณบางอย่างหรือการทำงานบางอย่าง(Action) ที่ก่อให้เกิดการเปลี่ยนแปลงสถานะการทำงาน และไม่ว่าเราจะใส่เหตุการณ์หรือการทำงานบางอย่างเพิ่มเติมลงใน ทรานสิชันไลน์ (Transition line) เราจะใส่เอาไว้ข้างๆทรานสิชันไลน์ ตัวอย่างการกำหนด ทริกเกอร์อีเวนต์ หรือ แอ็กชัน(Action) มีลักษณะดังรูป

Trigger event / action



รูปที่ 3-17 แสดงการใส่รายละเอียดบนทรานสิชันไลน์

แต่ในบางครั้งการเปลี่ยนสถานะการทำงานก็เป็นไปได้อัตโนมัติ โดยอาจจะอาศัยผลลัพธ์จากสถานะการทำงานที่ผ่านมาทำให้เกิดสถานะใหม่ก็ได้ ลักษณะของการเปลี่ยนแปลงสถานะการทำงานโดยไม่มีเหตุการณ์หรือการทำงานเข้าไปกระตุ้น เราจะเรียกว่า "ทริกเกอร์เลสทรานสิชัน" (triggerless transition)

นอกจากนี้แล้วยังมีอีกวิธีการหนึ่งที่จะทำให้เกิดการเปลี่ยนแปลงสถานะการทำงาน ได้แก่การกำหนด "การ์ดคอนดิชัน" (Guard Condition) ซึ่งเป็นลักษณะของการกำหนดเงื่อนไขทางตรรกะบางอย่าง เช่น อาจจะใช้ระยะเวลาเป็นตัวกำหนดการเปลี่ยนแปลงสถานะ เป็นต้น

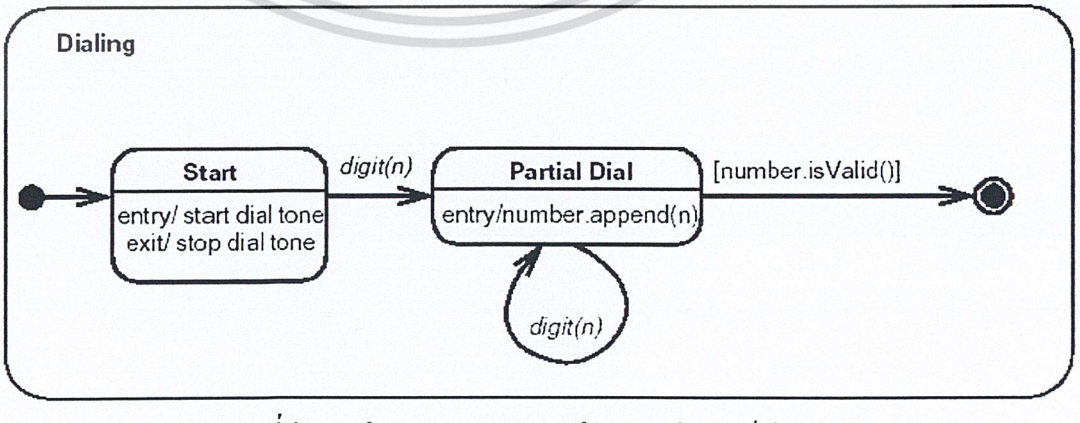
คอมโพสิทสเตท (Composite States)

เนื่องจากในสเตทหนึ่งๆ นั้นอาจจะสามารถแสดงถึงการทำงานที่ละเอียดได้อีก ดังนั้นเราจึงอาจมีการเพิ่มเติมรายละเอียดในแต่ละสเตทว่ามีการทำงานย่อยอะไรซ่อนอยู่บ้าง โดยเราจะเรียกสถานะการทำงานย่อยๆ ที่ซ่อนอยู่นั้นว่า "ซับสเตท" (Substates)

สถานะการทำงานหรือซับสเตทนั้น แบ่งเป็น 2 ประเภทคือ

1. ซีเควนท์เชียล (Sequential)

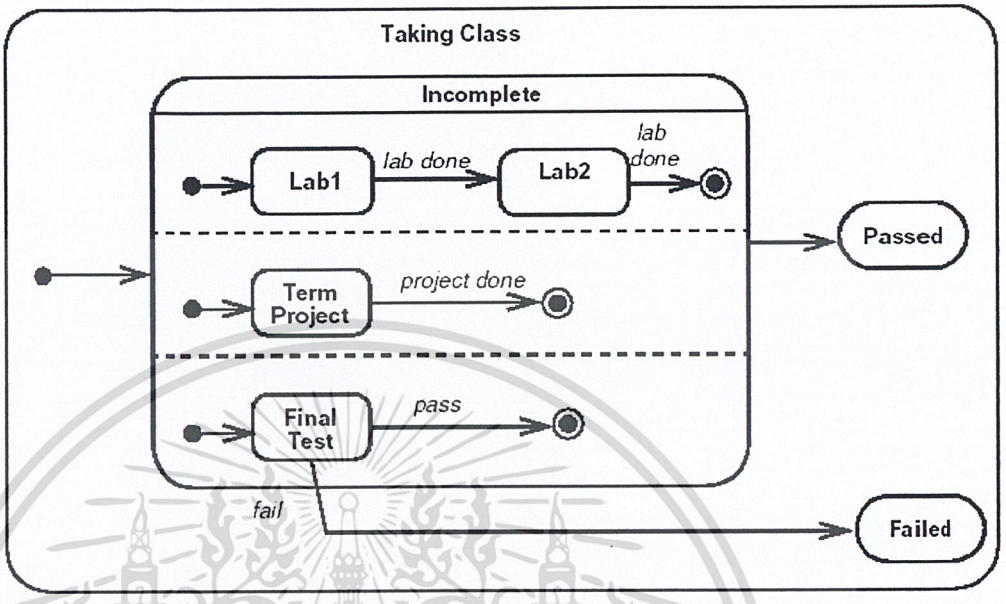
ลักษณะของซีเควนท์เชียลจะเป็นการเปลี่ยนสถานะการทำงานอย่างเป็นลำดับ จากสถานะหนึ่งไปยังสถานะหนึ่ง ลองพิจารณารูปตัวอย่างดังต่อไปนี้



รูปที่ 3-18 แสดงสเตท Dialing ที่มีการแบ่งสถานะการทำงานย่อยแบบซีเควนท์เชียล (Sequential)

2. คอนเคอเรนท์ (Concurrent)

ลักษณะของคอนเคอเรนท์นั้น สถานะการทำงานจะเปลี่ยนแปลงหลายอย่างพร้อมๆ กัน หนึ่ง ลองพิจารณารูปตัวอย่างดังต่อไปนี้

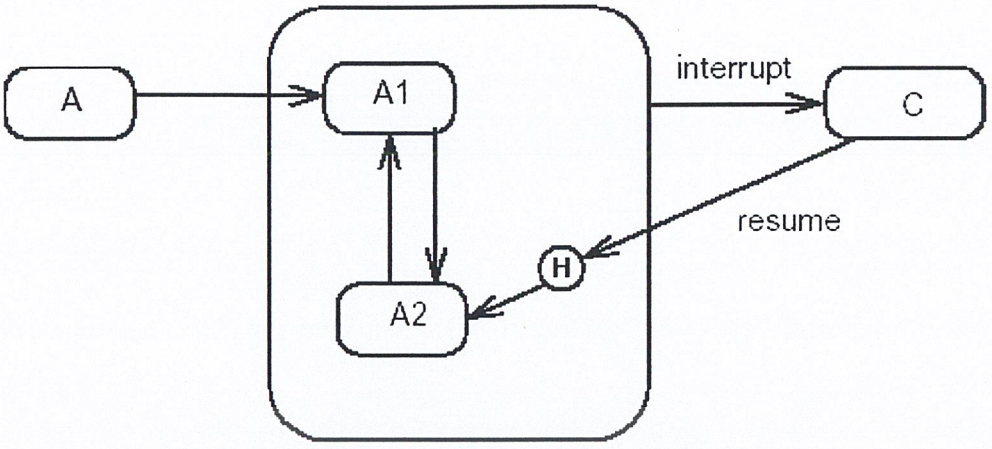


รูปที่ 3-19 แสดงสเตท Dialing ที่มีการแบ่งสถานะการทำงานย่อยแบบคอนเคอเรนท์ (Concurrent)

ฮิสทอรี่สเตท (History State)

การที่เกิดการเปลี่ยนสถานะจากสถานะหนึ่งไปยังอีกสถานะหนึ่ง และมีการเปลี่ยนแปลงจากสถานะดังกล่าวกลับมายังสถานะเดิม เราจะมีส่วนที่เรียกว่า "ฮิสทอรี่สเตท" (History State) เพื่อเป็นตัวบอกว่าเมื่อมีการเปลี่ยนแปลงกลับมาสถานะเดิม จะให้กลับมาที่จุดไหน

สัญลักษณ์ของ ฮิสทอรี่สเตทจะเป็นรูปวงกลมที่มีอักษรตัว H อยู่ภายในวงกลม แสดงได้ดังรูปตัวอย่างต่อไปนี้



รูปที่ 3-20 แสดงลักษณะการทำงานและสัญลักษณ์ที่ใช้ในฮิสทอรี่สเตท

สเตทไดอะแกรมแสดงให้เห็นถึงวงจรชีวิตของออบเจ็กต์ ระบบงานย่อย และระบบงานทั้งหมด โดยบ่งบอกว่าเหตุการณ์หรือภาวะต่างๆ มีผลต่อการเปลี่ยนแปลงของระบบงานอย่างไร ซึ่งสเตทไดอะแกรมหนึ่งอาจมีจุดเริ่มต้นและจุดสิ้นสุดได้หลายจุด

การใช้สเตทไดอะแกรมจะช่วยให้เราเข้าใจถึงพฤติกรรมของระบบมากขึ้น ในขณะที่ไดอะแกรมอื่นๆ อาจจะแสดงพฤติกรรมของระบบคืออะไร แต่จะไม่ลงไปในรายละเอียดมากนัก และเมื่อเราสามารถรู้ถึงพฤติกรรมของระบบได้ครบถ้วน เราก็จะสามารถตอบสนองต่อความต้องการของผู้ใช้งานได้อย่างตรงจุด

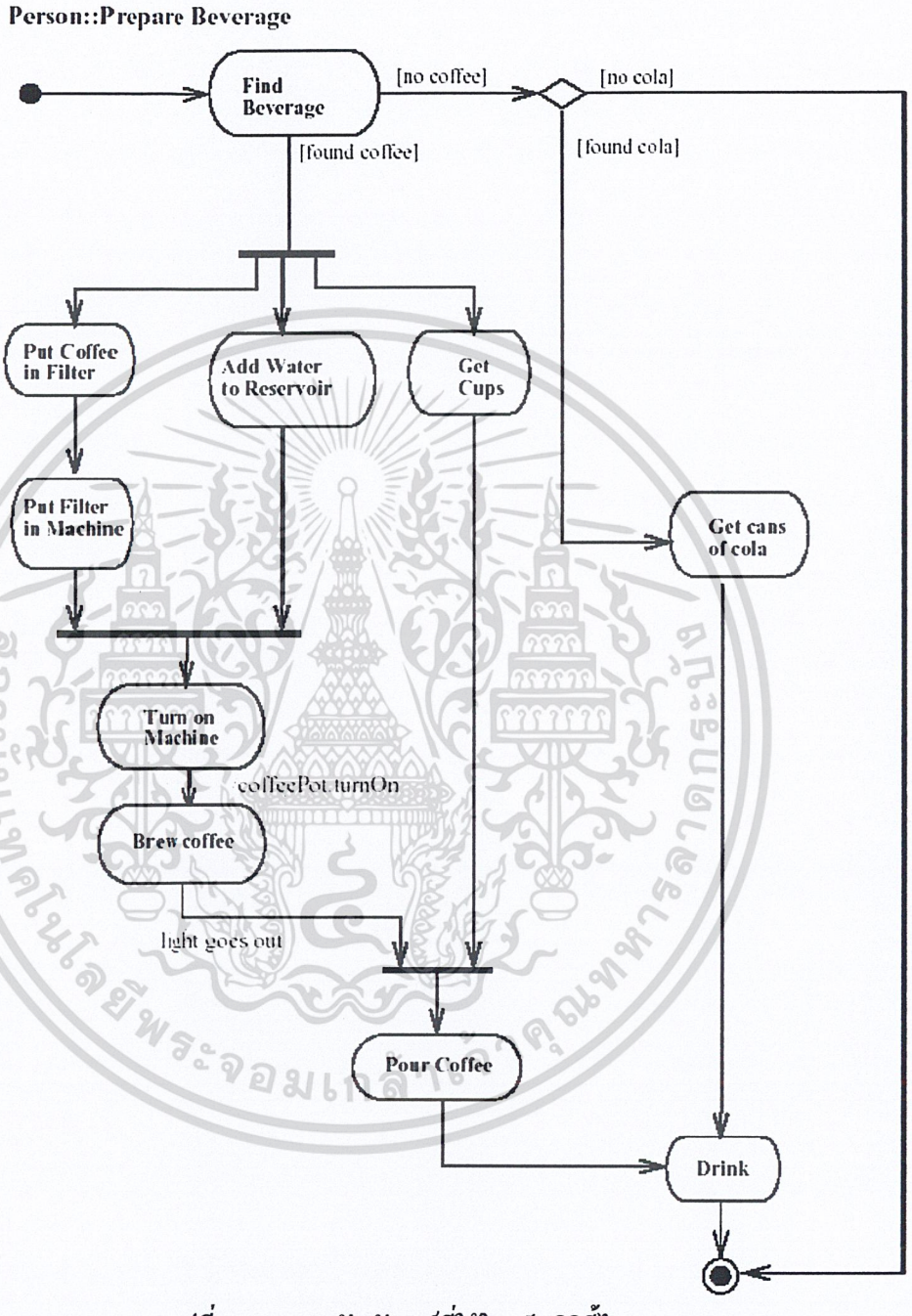
3.3.6 แอ็กทิวิตี้ ไดอะแกรม (Activity Diagram)

แอ็กทิวิตี้ไดอะแกรม จะมีลักษณะเดียวกันกับโฟลว์ชาร์ต(Flowchart) คือใช้สำหรับแสดงขั้นตอนการทำงาน แต่ต่างกันตรงที่แอ็กทิวิตี้ไดอะแกรม สามารถแสดงกิจกรรมที่ทำงานเป็นแบบขนานได้ เช่น งานที่ต้องแตกตัวเป็นหลายเธรด(Thread) และทำงานไปพร้อมๆ กัน โดยที่ขั้นตอนในการทำงานแต่ละขั้นตอนเราจะเรียกว่า “แอ็กทิวิตี้” (Activity) แอ็กทิวิตี้ไดอะแกรมสามารถนำมาใช้อธิบายยูสเคส, คลาสหรือโอเปอเรชั่น และการเปลี่ยนจากแอ็กทิวิตี้หนึ่งไปสู่อีกแอ็กทิวิตี้หนึ่งจะเกิดขึ้นโดยการเสร็จสิ้นการทำงานของแอ็กทิวิตี้แรก

แอ็กทิวิตี้ อาจเป็นลักษณะของการทำงานต่างๆ ได้แก่

- การคำนวณผลลัพธ์บางอย่าง
- การเปลี่ยนแปลงสถานะ (state) ของระบบ
- การส่งค่าบางอย่างกลับคืนมา
- การเรียกให้โอเปอเรชันอื่นๆ ทำงาน
- การส่งสัญญาณ
- การสร้างหรือการทำลายออบเจ็กต์

แอ็กทิวิตี้ ไดอะแกรมจะต้องมีจุดเริ่มต้นกับจุดสิ้นสุด และในระหว่างจุดเริ่มต้นกับจุดสิ้นสุด ก็จะมีขั้นตอนหรือแอ็กทิวิตี้ต่างๆ ของระบบ ดังรูปตัวอย่างต่อไปนี้



รูปที่ 3-21 แสดงสัญลักษณ์ที่ใช้ในแอ็กทิวิตี้ไดอะแกรม

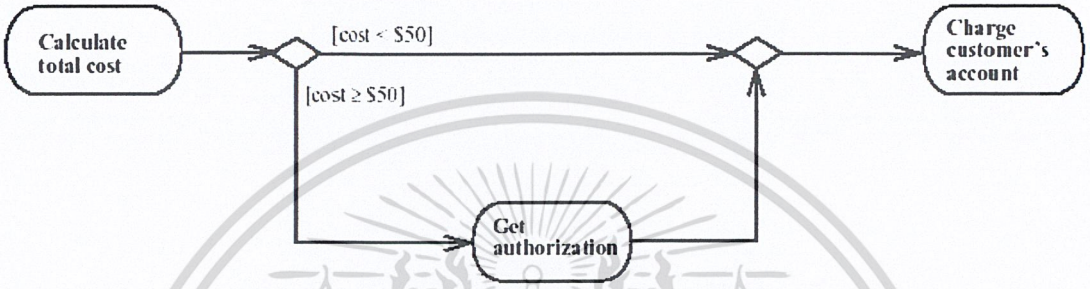
จากไดอะแกรมข้างต้น วงกลมทึบที่อยู่บนสุดแทนจุดเริ่มต้นของไดอะแกรมรูปสี่เหลี่ยมขอบมนจะแทนกิจกรรม หรือแอ็กทิวิตี้ของระบบที่เรากำลังสนใจ และวงกลมทึบที่อยู่ภายในวงกลมอีกทีจะแทนจุดสิ้นสุดของไดอะแกรม โดยปกติแล้วเราจะเขียนแอ็กทิวิตี้ไดอะแกรมโดยอ่านจากบนลงล่าง

รูปแบบการใช้แอ็กทิวิตี้ไดอะแกรม มีได้หลายแบบ ได้แก่

1. แบบมีทางเลือกให้ตัดสินใจ

เราสามารถกำหนดทางเลือกให้แก่แอ็กทิวิตี้ไดอะแกรมได้ 2 วิธีคือ

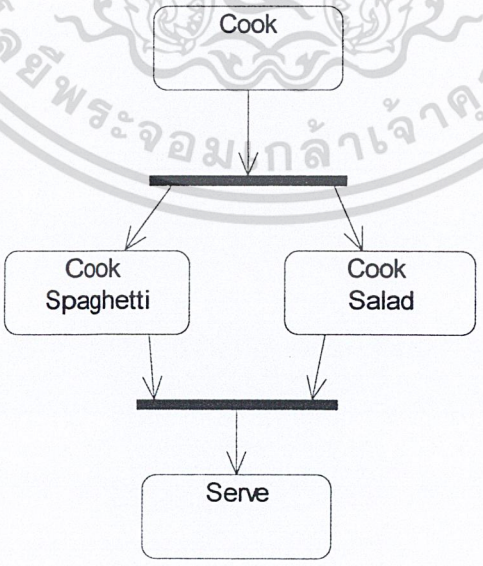
- ลากลูกศรของแต่ละทางเลือกไปยังแอ็กทิวิตี้ ผลลัพธ์ของทางเลือกโดยตรง
- ลากลูกศรให้ลูกศรของแต่ละทางเลือกผ่านรูปสี่เหลี่ยมขนมเปียกปูนเสียก่อน (ลักษณะเดียวกันที่เขียนโฟลว์ชาร์ต) แสดงได้ดังรูปตัวอย่างต่อไปนี้



รูปที่ 3-22 แสดงการสร้างทางเลือกให้แก่แอ็กทิวิตี้ไดอะแกรม

2. แบบที่มีการทำงานพร้อมๆ กันหลายงาน

ในกรณีที่เรามีงานหลายงานที่มีการทำงานไปพร้อมๆ กัน ในการเขียนแอ็กทิวิตี้ไดอะแกรมเราจะใช้เส้นตรงแนวนอนเส้นหนามาเป็นสัญลักษณ์ที่ใช้จัดกลุ่มงานที่มีการทำพร้อมๆ กัน โดยมีลักษณะดังตัวอย่างต่อไปนี้



รูปที่ 3-23 แสดงสัญลักษณ์ที่ใช้ในการจัดกลุ่มงานที่ทำพร้อมกันในแอ็กทิวิตี้ ไดอะแกรม (Activity Diagram)

3. แอ็กทิวิตี้ไดอะแกรมสำหรับการส่งสัญญาณ

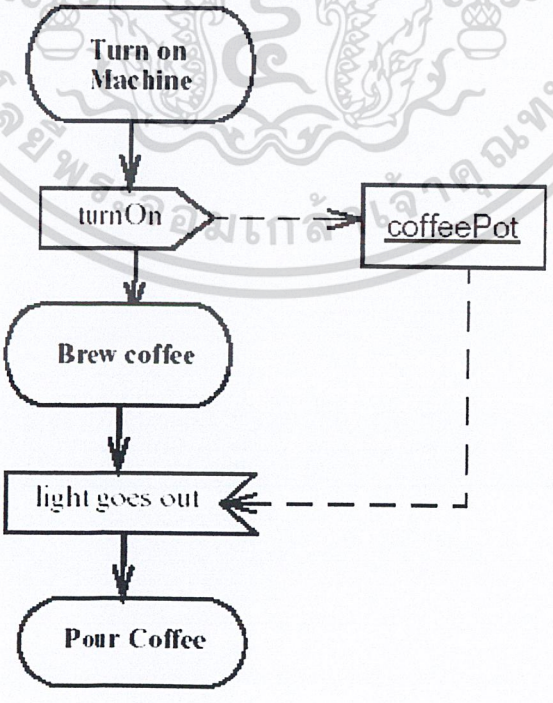
ในกระบวนการทำงานอาจเป็นไปได้ว่าจะมีการส่งสัญญาณบางอย่างในระหว่างการทำงาน เมื่อเกิดการส่ง-รับสัญญาณ เราก็จะเรียกว่าเกิด “แอ็กทิวิตี้” ขึ้นเช่นเดียวกัน

ในการเขียนแอ็กทิวิตี้ไดอะแกรมสำหรับการส่งสัญญาณ เราจะใช้รูปหลายเหลี่ยมแทนแอ็กทิวิตี้ที่มีการส่งสัญญาณ แสดงได้ดังรูป

รูปที่ 3-24 แทนเหตุการณ์ (Event) ที่เป็นอินพุท

รูปที่ 3-25 แทนเหตุการณ์ (Event) ที่เป็นเอาท์พุท

รูปต่อไปเป็นตัวอย่างของการใช้แอ็กทิวิตี้ไดอะแกรมแสดงการส่งสัญญาณ โดยระบบที่สนใจคือการชงกาแฟโดยใช้กระติกน้ำร้อน



รูปที่ 3-26 แสดงตัวอย่างการใช้สัญลักษณ์การส่งสัญญาณในแอ็กทิวิตี้ไดอะแกรม

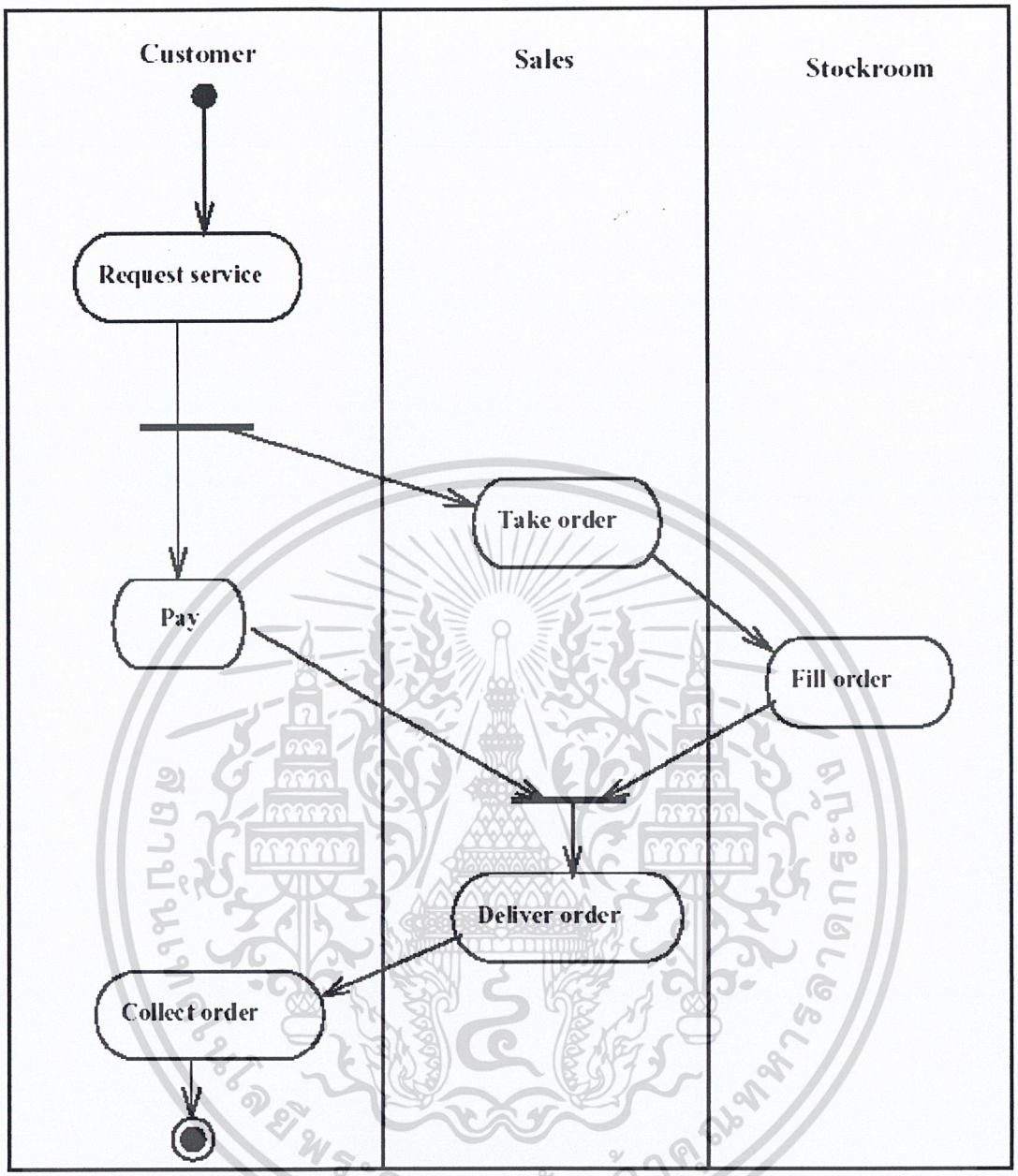
สวิมเลน (Swimlanes)

คุณลักษณะอย่างหนึ่งของการใช้แอ็กทิวิตี้ไดอะแกรม คือเราสามารถแสดงให้เห็นได้ว่าใครเป็นผู้มีหน้าที่รับผิดชอบในแต่ละแอ็กทิวิตี้ในกระบวนการทำงานหนึ่งๆ

หลักการของการแสดงหน้าที่ความรับผิดชอบของแต่ละแอ็กทิวิตี้ เราจะทำโดยการแบ่งกลุ่มของการรับผิดชอบเป็นกลุ่มๆ ซึ่งแอ็กทิวิตี้ไดอะแกรม มีลักษณะการแบ่งกลุ่มการรับผิดชอบด้วย เรียกกลไกดังกล่าวว่า “สวิมเลน”

ในแต่ละสวิมเลนจะมีการกำหนดชื่อกำกับเอาไว้ เช่น ในกระบวนการของการสั่งซื้อสินค้า เราอาจแบ่งกลุ่มของคนที่มีส่วนเกี่ยวข้องเป็น 3 ส่วน หรือ 3 สวิมเลนได้แก่ ลูกค้า, ฝ่ายขาย และคลังสินค้า เป็นต้น

แอ็กทิวิตี้หนึ่งๆ จะอยู่ภายในหนึ่งสวิมเลนเท่านั้น แต่การติดต่อหรือส่งผ่านระหว่างแอ็กทิวิตี้สามารถเกิดขึ้นข้ามจากสวิมเลนหนึ่งไปยังอีกสวิมเลนหนึ่งได้ ดังรูป



รูปที่ 3-27 แสดงตัวอย่างการใช้งานสวิมเลนในแอ็กทิวิตี้ไดอะแกรม

จะเห็นว่าแอ็กทิวิตี้ไดอะแกรมมีลักษณะคล้ายคลึงกับโฟลว์ชาร์ตมาก โดยจะแสดงถึงลำดับขั้นตอนการทำงาน จุดที่มีการตัดสินใจ และการแตกการทำงานเป็นส่วนย่อยๆ ด้วยลักษณะดังกล่าวแอ็กทิวิตี้ไดอะแกรมจึงมีประโยชน์มากสำหรับใช้อธิบายถึงการทำงานของออบเจ็กต์ต่างๆ รวมไปถึงกระบวนการทำงานต่างๆ ด้วย

เราอาจสงสัยว่าทั้งแอ็กทิวิตี้ไดอะแกรมและสเตทไดอะแกรมต่างก็ใช้แสดงขั้นตอนการทำงานซึ่งระบบงานเกิดการเปลี่ยนแปลง ดังนั้นเราจะมีหลักอย่างไรว่าเมื่อไรควรจะใช้แอ็กทิวิตี้ไดอะแกรมหรือเมื่อไรเราควรใช้สเตทไดอะแกรม

ข้อแตกต่างของทั้งสองไดอะแกรม อยู่ที่ลักษณะของการเปลี่ยนแปลงของระบบ ถ้าเป็นการเปลี่ยนแปลงหรือเหตุการณ์ต่างๆ อันเป็นผลมาจากการกระทำภายในระบบเองซึ่งมีลักษณะต่อเนื่องกัน

และมักมีจุดที่ระบบงานต้องมีการตัดสินใจ ให้ใช้แอ็กทิวิตี้ไดอะแกรมแต่ถ้าเป็นการเปลี่ยนแปลงหรือการกระทำที่เกิดขึ้นในลักษณะไม่ต่อเนื่องกัน ให้ใช้สเตทไดอะแกรม

เมื่อเราสร้างแอ็กทิวิตี้ไดอะแกรมขึ้นมา เราต้องไม่ลืมว่าเป็นเพียงโมเดลหนึ่งของภาษา UML ที่พยายามสร้างมุมมองแบบไดนามิกของระบบ ดังนั้น จึงไม่มีแอ็กทิวิตี้ไดอะแกรมใดจะสามารถอธิบายทุกๆ อย่างของระบบได้ภายในไดอะแกรมเดียว เราจำเป็นต้องใช้แอ็กทิวิตี้ไดอะแกรมมาช่วยมากกว่าหนึ่งไดอะแกรม จึงจะเห็นกระบวนการทำงานได้อย่างครบถ้วน

แอ็กทิวิตี้ไดอะแกรมเป็นไดอะแกรมที่ค่อนข้างสร้างแล้วเข้าใจได้ง่าย อย่างไรก็ตาม แอ็กทิวิตี้ไดอะแกรมที่ดีควรมีคุณสมบัติต่อไปนี้

- มุ่งเน้นไปที่การติดต่อสื่อสารของระบบเชิงไดนามิก กล่าวคือ เป็นการติดต่อที่สภาพของระบบมีการเปลี่ยนแปลงไม่หยุดนิ่ง
- เฉพาะส่วนที่มีความสำคัญต่อกระบวนการทำงานเท่านั้น
- แสดงรายละเอียดในแต่ละระดับของการทำงาน ซึ่งรายละเอียดดังกล่าวจะเลือกแสดงเฉพาะที่มีความสำคัญต่อการเข้าใจการทำงานของระบบเท่านั้น
- ถ้าการทำงานส่วนใดมีความสำคัญ แอ็กทิวิตี้ไดอะแกรมก็ควรแสดงให้เห็น ไม่ควรละเอาไว้หรือแสดงอย่างย่นย่อ

บทที่ 4
ทฤษฎีและความรู้พื้นฐานของภาษา SQL3

SQL3 เป็นมาตรฐานของภาษา SQL ที่เพิ่มความสามารถต่างๆ ขึ้น โดยเฉพาะความสามารถที่เกี่ยวกับแนวคิดเชิงวัตถุ หรือออบเจ็กต์ เช่นมีการสนับสนุนการทำออบเจ็กต์คิวรี่(Object Queries) และการเชื่อมโยงติดต่อกันระหว่างออบเจ็กต์ภายใน (Object Traversal) ได้โดยไม่ต้องมีการเชื่อมโยง (join) นอกจากนี้ SQL3 ได้มีการกำหนดชนิดของข้อมูลต่างๆ เพื่อรองรับกับชนิดข้อมูลที่มีความซับซ้อนมากขึ้นในปัจจุบันและเพื่อสามารถนำมาประยุกต์ใช้ฐานข้อมูลเชิงวัตถุสัมพันธ์ด้วย

4.1 ชนิดข้อมูล

4.1.1 ชนิดข้อมูลพื้นฐาน (Predefined Data Type หรือ Built-in Data Type)

เป็นชนิดข้อมูลพื้นฐานที่มีอยู่ใน SQL92 เดิมด้วย ซึ่งประกอบไปด้วย ชนิดข้อมูล Binary Large Object, Bit, Bit Varying, Boolean, Character, Character Varying, Character Large Object, Date, Decimal, Double, Enumerated, Float, Integer, Interval Boolean, Numeric, Precision, Real, Smallint, Time และ Timestamp

4.1.2 Row Type

ใน SQL3 เราสามารถกำหนดชนิดของ Tuple ได้ซึ่งเรียกว่า Row Type โดยการประกาศ Row Type มีรูปแบบดังนี้

```
CREATE ROW TYPE <Name_of_RowType>
(<Component Declaration>);
```

ตัวอย่างเช่น เราต้องการจะสร้าง Row Type ที่ใช้สำหรับเก็บรายละเอียดของพนักงาน โดยกำหนดให้มีข้อมูลต่างๆ ดังนี้ รหัส, ชื่อ, แผนก, เงินเดือน เราต้องทำการประกาศสร้าง Row Type ดังนี้

```
CREATE ROW TYPE EmployeeType
(
        ID              INT,
        Name            CHAR (30),
        Department      CHAR (30),
        Salary          INT
);
```

นอกจากนี้แล้วเรายังสามารถที่จะใช้ชนิดข้อมูลนี้ได้เหมือนกับชนิดข้อมูลภายในที่มากับระบบฐานข้อมูล นอกจากนี้เราสามารถที่จะประกาศให้ภายใน Row Type ประกอบด้วย Row Type อื่นได้อีกด้วย ตัวอย่างเช่น เราเพิ่มข้อมูล Address ใน Row Type EmployeeType

```
CREATE ROW TYPE AddressType
(
        Street    CHAR(30),
        City      CHAR(30)
);
CREATE ROW TYPE EmployeeType
(
        ID              INT,
        Name            CHAR(30),
        Department      CHAR(30),
        Salary          INT,
        Address         AddressType
);
```

หรือสามารถที่จะประกาศแบบนี้

```
CREATE ROW TYPE EmployeeType
(
        ID              INT,
        Name            CHAR(30),
        Department      CHAR(30),
        Salary          INT,
        Address         ROW(Street CHAR(30),City CHAR(30))
);
```

จากตัวอย่างข้างต้น การประกาศแบบที่สองนั้นจะเห็นได้ว่า เราไม่สามารถที่จะอ้างอิงชื่อที่ใช้เรียก Row Type AddressType ได้ เพราะว่าไม่ได้ประกาศไว้ ดังนั้นเราจึงไม่สามารถที่จะนำ Row Type ไปใช้ที่อื่นได้อีก เมื่อได้ Row Type แล้วเราก็สามารถนำไปสร้างตาราง Relation ที่มีลักษณะของ Tuple เหมือนกับ Row Type ได้โดยมีรูปแบบดังนี้

CREATE TABLE <Relation_Name> **OF TYPE** <Row_Type>;

ตัวอย่างเช่น เราต้องการสร้างตารางที่มีชื่อว่า Employee โดยอ้างอิงจาก Row Type EmployeeType และจากการสร้างตารางที่ประกอบด้วย 5 คอลัมน์ มีลักษณะแอททริบิวต์ดังนี้

CREATE TABLE Employee **OF TYPE** EmployeeType;

ID	Name	Department	Salary	Address
101	John	Service	12000	Maple St.,Beverly Hills
102	Mary	Account	10000	Oak St.,Brentwood

ตารางที่ 4-1 ตัวอย่างการจัดเก็บข้อมูลที่เป็น Row Type

สำหรับการเข้าถึงข้อมูลที่เป็น Row Type สามารถใช้งานเหมือนเป็นหน่วยเดียวหรือเข้าถึงส่วนย่อยๆ โดยใช้ Dot Notation 2 ครั้ง ".." ดังนี้

SELECT Name,Department,Address..City
FROM Employee

4.1.3 Object Identifier (OID)

เป็นตัวระบุความแตกต่าง (Identifier) ของออบเจ็กต์แต่ละตัวซึ่งจะไม่เหมือนกันและจะไม่มีการเปลี่ยนแปลงค่า โดยเราสามารถทำการสร้างโอไอดี (OID) ขึ้นมาเป็นคอลัมน์หนึ่ง ซึ่งสามารถเรียกดูผ่าน SQL Command ได้เหมือนกับแอททริบิวต์ชนิดอื่นๆ โดยมีรูปแบบกำหนดดังนี้

VALUES FOR <Attribute> **ARE SYSTEM GENERATED**

ตัวอย่างการใช้งานมีดังนี้

CREATE TABLE Employee **OF TYPE** EmployeeType
VALUES FOR Emp_ID **ARE SYSTEM GENERATED**

4.1.4 Reference

SQL3 มีความสามารถในการอ้างถึงออบเจ็กต์อื่นๆ โดยไม่ได้อ้างถึงข้อมูลจริง แต่จะใช้โอไอดี ในการอ้างถึง ดังนี้

```
CREATE ROW TYPE EmployeeType
(
        ID              INT,
        Name            CHAR(30),
        Department      CHAR(30),
        Salary          INT,
        Address         REF(AddressType)
);
```

โดยในคอลัมน์ที่เรากำหนดให้อ้างถึงนี้จะไม่ได้ทำการเก็บข้อมูลจริง แต่จะทำการเก็บ Object Identifier (OID) ซึ่งใช้อ้างถึงไปยังออบเจ็กต์อื่นแทน

4.1.5 Abstract Data Type (ADT)

เป็นชนิดข้อมูลแบบออบเจ็กต์ ซึ่งชนิดข้อมูลชนิดนี้เราสามารถที่จะระบุให้มีแอททริบิวต์และการดำเนินการต่างๆ ซึ่งจะใช้สำหรับการทำเปรียบเทียบหรือเรียงข้อมูล และใช้สำหรับการติดต่อกับภายนอก ซึ่งความแตกต่างระหว่าง ADT กับ Row Type คือ Row Type ไม่มีคุณสมบัติในเรื่องของการซ่อนเร้นข้อมูล (Encapsulation) โดยรูปแบบการทำ ADT ใน SQL3 สามารถประกาศได้ดังนี้

```
CREATE ROW TYPE <Type_Name>
( <List of Attribute and their Types>,
Option declaration of = and < Function for the Type,
Declaration of Functions(Method) for the Type);
```

โดย <Type_Name> คือชื่อของ ADT ที่ต้องการประกาศ จากนั้นก็เป็นการประกาศแอททริบิวต์ต่างๆ และชนิดข้อมูลของแต่ละแอททริบิวต์ และตามด้วยการประกาศฟังก์ชันที่ใช้สำหรับนำมาใช้ในการเปรียบเทียบซึ่งอนุญาตให้ประกาศได้สองเครื่องหมายคือ = และ < โดยในการประกาศจะใช้คำว่า **EQUALS, LESS THAN** และตามด้วยชื่อของฟังก์ชันที่เราสร้างขึ้นมาเพื่อสนับสนุนการเปรียบเทียบ ซึ่งการประกาศฟังก์ชันนี้เป็นส่วนที่สามารถจะเพิ่มเติม (Option) ซึ่งถ้าเราไม่ได้สร้างฟังก์ชันนี้ขึ้นมาก็สามารถที่จะใช้คำว่า **DEFAULT** เพื่อให้ ADT ใช้ฟังก์ชันที่มีอยู่แล้วกับ ADT นี้ได้เลย แต่ถ้าไม่ต้องการให้ใช้เครื่องหมายเหล่านี้กับ ADT ที่เรากำหนดก็สามารถใช้คำว่า **NONE** แทนได้ และส่วนสุดท้ายของการประกาศ ADT ก็คือส่วนของการ

ประกาศฟังก์ชันต่างๆ ที่ต้องการสร้างขึ้นเพื่อใช้กับข้อมูลของ ADT ของที่ประกาศ ตัวอย่างการใช้งานมีดังนี้

CREATE TYPE Address_ADT
(Street CHAR(30),
City CHAR(30),
EQUALS ADDREQ,
LESS THAN ADDRLT
other function should be declare here);

สำหรับการประกาศฟังก์ชันหรือเมธอด (Method) ใน SQL3 มี Built-in Function ให้ดังนี้

1. **Constructor Function** เป็นการสร้างอินสแทนซ์ของ ADT จะทำการส่งค่า (Return) ออบเจ็กต์ใหม่ของ ADT ทุกๆ แอททริบิวต์ของออบเจ็กต์จะกำหนดค่าตั้งต้นเป็น NULL โดยชื่อของคอนสตรัคเตอร์ฟังก์ชันจะมีชื่อเดียวกับ ADT ที่เราสร้างขึ้น
2. **Observer Function** เป็นฟังก์ชันที่ทำการส่งคืนค่าของทุกๆ แอททริบิวต์โดยที่ออบเซิร์ฟเวอร์ฟังก์ชัน จะถูกกำหนดให้มีชื่อเดียวกับแอททริบิวต์และมีพารามิเตอร์ 1 ตัว เป็นชนิดเดียวกับ ADT ที่กำหนด
3. **Mutator Function** เป็นการกำหนดค่าของแอททริบิวต์นั้นให้เป็นค่ามิวเตเตอร์ฟังก์ชันจะถูกกำหนดให้มีชื่อเดียวกับแอททริบิวต์และมีพารามิเตอร์ 2 ตัว ตัวแรกเป็นชนิดเดียวกับ ADT ที่กำหนด ตัวที่สองเป็นชนิดของข้อมูลแอททริบิวต์นั้น

การปกป้องการเข้าถึงข้อมูลทำได้ 3 ระดับ คือ Public, Private และ Protect

4.1.6 ชนิดข้อมูลแบบ Collection

เป็นชนิดข้อมูลซึ่งประกอบด้วยสมาชิก (element) ของข้อมูลอยู่ภายใน โดยสมาชิกของ Collection จะอยู่ในคอลัมน์เดียวกันหมด โดยข้อมูลชนิด Collection แบ่งออกเป็น 3 ชนิดคือ Set, Multiset และ List โดยชนิดของข้อมูลใน Collection อาจจะเป็นชนิดข้อมูลพื้นฐาน, ADT หรือ Collection เองก็ได้โดยที่ชนิดของข้อมูลแบบต่างๆ แสดงไว้ในตารางต่อไปนี้

Collection data type	คุณสมบัติ
Set	เป็นชนิดของ Collection type ที่ไม่มีลำดับของข้อมูล และข้อมูลภายในจะซ้ำกันไม่ได้
MultiSet	มีลักษณะคล้ายกับชนิดข้อมูลแบบ Set แต่สามารถที่จะมีข้อมูลที่ซ้ำกันได้
List	เป็น Collection type ที่มีลำดับของข้อมูลและสามารถที่จะมีค่าข้อมูลซ้ำกันได้

ตาราง 4-2 แสดงชนิดของข้อมูล Collection แบบต่างๆ

4.2 Integrity Constraint

ความสามารถในการทำการรักษาความถูกต้อง (Integrity Constraint) ของ SQL3 มีดังนี้

4.2.1 Key Constraint

เป็นการประกาศคีย์หลักซึ่งจะประกาศในขั้นตอนการ **CREATE TABLE** ทำได้สองแบบคือการใช้คีย์เวิร์ด **PRIMARY KEY** และ **UNIQUE**

<ColumnName> <DataType> <PRIMARY KEY หรือ UNIQUE>

หรือ

<PRIMARY KEY หรือ UNIQUE> (<ColumnName>)

สำหรับการประกาศคีย์หลักที่มีมากกว่า 1 แอททริบิวต์ทำได้ดังนี้

<PRIMARY KEY หรือ UNIQUE> (<ColumnName>, <ColumnName>)

สำหรับการประกาศโดยใช้คีย์เวิร์ด **UNIQUE** เป็นการกำหนดให้คอลัมน์ดังกล่าวต้องไม่มีค่าซ้ำ รูปแบบการประกาศเหมือนกับการประกาศคีย์หลักทุกประการ

4.2.2 Referential Integrity Constraint

เป็นการกำหนดความสัมพันธ์ระหว่างคีย์หลักและฟอร์เรนคีย์ (Foreign Key) การประกาศทำได้ 2 แบบคือ

REFERENCES <Table> (<Attribute>)

หรือ

FOREIGN KEY <Attribute> **REFERENCES** < Table> (<Attribute>)

4.2.3 Attribute Constraint

การกำหนดการบังคับความถูกต้องระดับแอททริบิวต์ จะกำหนดได้ 2 แบบคือ

1. ในการกำหนดรีเลชันสกีมา (Definition ของ Relation Schema)
2. ในโดเมน (Domain) ซึ่งจะประกาศเป็น Domain Constraint

โดยมีรายละเอียดของแต่ละประเภทดังนี้

4.2.3.1 NOT NULL Constraint

เป็นการกำหนดไม่ให้แอททริบิวต์นั้นมีค่า NULL โดยใช้คีย์เวิร์ด "**NOT NULL**" ในตอนที่ประกาศแอททริบิวต์ดังนี้

PRESC# INT **REFERENCES** MovieExec(CERT#) NOT NULL

4.2.3.2 Attribute-based CHECK Constraint

จะทำการเช็คความถูกต้องของแอททริบิวต์ทุกครั้งที่มีการแก้ไขข้อมูล เช่น การอินเสิร์ท (Insert) หรืออัพเดท (Update) จะต้องมีการเช็คความถูกต้องของแอททริบิวต์ที่กำหนดไว้ รูปแบบดังนี้

PRESC# INT **REFERENCES** MovieExec(CERT#) **CHECK** (PRESC# >= 100000)

หรือ

GENDER CHAR(1) **CHECK** (GENDER IN ('F', 'M'))

4.2.3.3 Domain Constraint

ทำการสร้างโดเมนใหม่ขึ้นมา แล้วทำการกำหนดค่าในโดเมน ดังนี้

CREATE DOMAIN GenderDomain CHAR(1)

CHECK (VALUE IN ('F', 'M'));

เมื่อมีการสร้างรีเลชันสามารถนำโดเมนที่สร้างขึ้นไปอ้างถึงเหมือนกับเป็นข้อมูลชนิดอื่นๆ ได้ การสร้างโดเมนคอนสเตรนท์ดีกว่าสองแบบแรกเพราะสามารถใช้งานได้ในทุกรีเลชัน

4.2.4 Global Constraint

เป็นการประกาศคอนสเทรนท์ที่มีความซับซ้อนมากขึ้น มีการประกาศได้ 2 ลักษณะ

4.2.4.1 Tupled-Based CHECK Constraint

จะทำการตรวจสอบทุกครั้งที่มีการอินเสิร์ทหรืออัพเดททัปเปิล โดยจะควบคุมความถูกต้องแต่บนรีเลชันที่สร้างขึ้นมา ดังตัวอย่างการประกาศ Tuple-Based CHECK บนคอลัมน์ Gender

```
CREATE TABLE Employee
( Name      CHAR(30) PRIMARY KEY,
  Address   VARCHAR(225),
  Gender    CHAR(1),
  CHECK     (Gender = 'M' OR Name NOT LIKE 'Mr.%')
```

4.2.4.2 Assertion

เป็นการกำหนดเงื่อนไขความถูกต้อง โดยเงื่อนไข (Condition) ในแอสเสิร์ทชัน (Assertion) จะต้องเป็นจริงเสมอ ถ้าการแก้ไข (Modified) ใดทำให้เงื่อนไขเป็นเท็จจะถูกปฏิเสธทันที (Reject) แอสเสิร์ทชันแต่เดิมในมาตรฐาน SQL92 จะควบคุมความถูกต้องทุกๆ รีเลชัน แต่ใน SQL3 เหลือควบคุมแค่บนรีเลชันเดียว แสดงได้ดังนี้

```
CREATE ASSERTION RichPRES CHECK
(NOT EXISTS
    (SELECT   *
     FROM     Studio,MovieExec
     WHERE    PRESC# = CERT# AND NETWORTH < 10000000));
```

4.2.5 Triggers

เป็นแอคทีฟคอมโพเนนท์ที่จะเข้ามาทำแอคชั่น (Action) เมื่อเกิดเหตุการณ์ (Event) ตามที่กำหนดไว้ โดยเหตุการณ์ที่เกิดได้ทั้ง Before, After, Instead of โดยถ้าเป็น Before จะทำการเช็คเหตุการณ์ก่อนที่จะทำแอคชัน ถ้าเป็น After จะทำแอคชันหลังจากเกิดเหตุการณ์ แต่ถ้าเป็น Instead of ถ้าเกิดเหตุการณ์จะไม่เกิดแอคชัน ตัวอย่างการสร้างทริกเกอร์ มีดังนี้

```
CREATE TRIGGER NETWORTHTrigger
AFTER UPDATE OF NETWORTH ON MovieExec
REFERENCING
```

```
        OLD AS OldTuple,
        NEW AS NewTuple
    WHEN (OldTuple.NETWORTH > NewTuple.NETWORTH)
        UPDATE MovieExec
        SET NETWORTH = OldTuple.NETWORTH
        WHERE CERT# = NewTuple.CERT#
    FOR EACH ROW
```

4.3 การสร้างรูทีน

สามารถสร้างรูทีนเก็บไว้ในดาต้าเบสเซิร์ฟเวอร์ได้ รูทีนที่สร้างขึ้นอาจจะเป็นได้ทั้ง Function หรือ Procedure โดยที่ขั้นตอนสร้างและใช้งานรูทีนมีดังนี้

4.3.1 เขียนคำสั่งในการสร้างรูทีน เช่น

```
CREATE FUNCTION discount_price(per_cent REAL,id1 char(5) )
RETURNING MONEY;
    DEFINE money REAL;
    SELECT price INTO money FROM price_table WHERE id = id1;
RETURN money*(100-per_cent)/100;
END FUNCTION;
                                หรือ
CREATE PROCEDURE add_price(arg INT);
    .
    .
END PROCEDURE;
```

4.3.2 ใช้คำสั่งเพื่อเรียกใช้งานรูทีน เช่น

```
EXECUTE FUNCTION discount_price(15);
```

4.4 Routine Inheritance

การที่มีตารางที่สร้างขึ้นโดยการถ่ายทอดคุณสมบัติมาจากตารางที่มีอยู่แล้ว เช่น

```
CREATE TYPE person
    (name CHAR(30),
     age integer)
CREATE TABLE people OF TYPE person;
```

และ

CREATE TYPE children UNDER person;

CREATE TABLE student **OF TYPE** children **UNDER** people;

เรียก person ว่าเป็น SUPER TYPE และเรียก children ว่าเป็น SUB TYPE

เราสามารถสร้างรูทีนที่สามารถเรียกใช้ได้กับทั้งตารางที่เป็น SUPER TYPE และตารางที่เป็น SUB TYPE เพื่อให้สอดคล้องตามหลักของ Object Oriented คือ Child class จะต้องเรียกใช้ method ของ Parent class ได้ โดยรูทีนที่สร้างขึ้นให้กับตาราง SUPER TYPE นั้นจะต้องมีการรับพารามิเตอร์เป็น Type ของตาราง SUPER TYPE ด้วยเช่น

CREATE PROCEDURE increase_age(arg person, name1 CHAR(30));

UPDATE arg SET age = age + 1 WHERE name = name1;

END PROCEDURE;

แล้วตาราง SUB TYPE ก็จะสามารถเรียกใช้งานรู้ทีนของ SUPER TYPE ได้ เช่น

EXECUTE PROCEDURE increase_age(student,"Dang");

4.5 เปรียบเทียบ SQL3 กับ SQL92

นอกจากข้อดีต่างๆ ของภาษา SQL3 ที่ได้กล่าวไปแล้ว SQL3 ยังมีข้อดีในการที่จะเขียนโค้ดคำสั่งได้ง่ายกว่าภาษา SQL92 เนื่องจาก SQL3 มี "Object Navigation Query" คือการทำคิวรี่โดยอาศัย OID เพื่ออ้างถึงออบเจ็กต์อื่นๆ ที่สามารถที่จะนำมาใช้ในการทำคิวรี่แทนการทำ "Join Query" ของภาษา SQL92 ดังตัวอย่างคิวรี่ต่อไปนี้

SELECT ID,Name,State,Dno(major),Name(major),Building(major)

FROM Student

จากตัวอย่างคิวรี่ข้างต้นจะเห็นได้ว่า ID, Name และ State เป็นแอททริบิวต์ของออบเจ็กต์ Student ส่วน Department Number, Department Name และ Building เป็นแอททริบิวต์ที่มาจากออบเจ็กต์ Department โดยอาศัยความสัมพันธ์จาก major ที่อยู่ในออบเจ็กต์ Student ในการที่จะเอาข้อมูลจากออบเจ็กต์ Department อีกทีหนึ่ง ซึ่งจะเห็นได้ว่าเป็นการแสดงข้อมูลของ Department โดยไม่ต้องมีการอ้างออบเจ็กต์ Department ตรงๆ ซึ่งถ้าเป็นใช้ภาษา SQL92 ทำคิวรี่นี้เราจะได้ดังนี้

```
SELECT S.ID, S.Name, S.State, D.Dno, D.Name, D.Building
FROM Department D, Student S
WHERE S.majorDept = D.deptNo
UNION ALL
SELECT S.ID, S.Name, S.State, D.Dno, D.Name, D.Building
FROM Departament D, TA S
WHERE S.majorDept = D.deptNo
```

ซึ่งนอกจากนี้แล้ว SQL3 ยังมีความสามารถที่จะเรียกฟังก์ชันที่ผู้ใช้เป็นคนสร้าง (User-Defined Method) เช่นสมมติเราต้องการที่จะทำการคิวรี่หาระยะทางระหว่าง Staff member (โดยที่ Staff เป็นออบเจ็กต์ที่อยู่ใน Staff คลาส) กับ Staff member ที่มี ID 6966 เราสามารถทำได้ดังนี้

```
SELECTdistance(s1.place,s2.place)
FROM Staff s1,Staff s2
WHERE s1.id = 6966
```

ในขณะที่ภาษา SQL92 จะเขียนได้ยากกว่า ดังนี้

```
SELECT SQRT((s1.latitude-s2. latitude)*
        (s1. latitude-s2. latitude)+(s1.longtitude-s2.longtitude)*
        (s1.longtitude-s2.longtitude))
FROM Staff s1,Staff s2
WHERE s1.id = 6966
```

บทที่ 5
ทฤษฎีพื้นฐานเกี่ยวกับระบบฐานข้อมูลเชิงวัตถุสัมพันธ์ (ORDBMS)

ระบบฐานข้อมูลเชิงวัตถุสัมพันธ์ คือ ฐานข้อมูลชนิดที่นำแนวความคิดแบบออบเจ็กต์โอเรียนเต็ด (Object-Oriented Database) มาผสมผสานกับระบบฐานข้อมูลเชิงสัมพันธ์ (Relation Database) เพื่อทำให้ระบบฐานข้อมูลเชิงสัมพันธ์มีความสามารถทางด้านออบเจ็กต์ได้ เช่น การปกป้องข้อมูล การสืบทอดคุณสมบัติ และการทำโพลมอร์ฟีซึม เป็นต้น โดยระบบฐานข้อมูลเชิงวัตถุสัมพันธ์จะมองข้อมูลและทำการจัดเก็บข้อมูลเป็นแบบตาราง โดยมีการจัดการฐานข้อมูลแบบเชิงสัมพันธ์ แต่การติดต่อกับผู้ใช้งานได้นำระบบของออบเจ็กต์โอเรียนเต็ดมาใช้งาน ซึ่งระบบออบเจ็กต์โอเรียนเต็ดจะมีการสนับสนุนและมีการดูแลระบบฐานข้อมูลขนาดใหญ่และซับซ้อนได้ดีกว่า

ฐานข้อมูลเชิงวัตถุสัมพันธ์เสมือนการเพิ่มชั้นของออบเจ็กต์โอเรียนเต็ดครอบบนชั้นของระบบฐานข้อมูลสัมพันธ์ที่มีอยู่ และฐานข้อมูลเชิงวัตถุสัมพันธ์เป็นฐานข้อมูลเชิงสัมพันธ์ที่สามารถมีแอททริบิวต์เป็นแอททริบิวต์ที่สามารถแบ่งแยกย่อยได้อีก โดยภาพรวมแล้วเรายังมองเป็นฐานข้อมูลเชิงสัมพันธ์ ในคอลัมน์สามารถเป็นได้ทั้งออบเจ็กต์หรือเป็นแอททริบิวต์ที่มีหลายค่า เช่น ลิสต์ หรือเซต โดยทั่วไปโมเดลที่เป็นฐานข้อมูลเชิงวัตถุสัมพันธ์สามารถใช้งานเสมือนเป็นฐานข้อมูลเชิงสัมพันธ์ได้

5.1 คุณสมบัติของฐานข้อมูลเชิงวัตถุสัมพันธ์

- มีความสามารถเดิมของฐานข้อมูลเชิงสัมพันธ์
- สนับสนุนการสร้างออบเจ็กต์ที่มีความซับซ้อน
- มีความยืดหยุ่นสูงยอมให้สร้างชนิดข้อมูล ฟังก์ชันและตัวดำเนินการ (Operator) ใหม่ได้
- มีความสามารถสร้างแอททริบิวต์เป็นออบเจ็กต์หรือเป็นแอททริบิวต์ที่สามารถแบ่งแยกย่อยได้
- มีคุณสมบัติแนวคิดเชิงวัตถุ เช่น การปกป้องข้อมูล การสืบทอดคุณสมบัติ และโพลิมอร์ฟีซึม

5.2 Nested Relations

มีลักษณะคล้ายกับความสัมพันธ์ (Relation) แตกต่างกันที่ Nested Relation สามารถมีโดเมนที่เป็นได้ทั้งค่าที่ไม่สามารถแบ่งย่อยได้ (Atomic Value) หรือความสัมพันธ์ (Relation) ก็ได้ ดังนั้นค่าของแอททริบิวท์ (Attribute) จึงสามารถเป็นความสัมพันธ์ และความสัมพันธ์ยังสามารถอยู่ ในความสัมพันธ์อื่นได้ วัตถุแบบซับซ้อนจึงสามารถแทนได้ด้วยทับเพิ้ล (Tuple) เดียวซึ่งเป็น Nested Relation ถ้าเรามองว่าทับเพิ้ลนี้เป็น Data Item เราจะได้ความสัมพันธ์แบบหนึ่งต่อหนึ่ง (one-to-one) ระหว่าง Data Item และออบเจ็คในมุมมองของผู้ใช้ของฐานข้อมูล

Title	Author-list	date	keyword-list
Salesplan	{Smith, Jones}	(1, April, 89)	{profit, strategy}
Status report	{Jones, Frick}	(17, Junes, 94)	{profit, personnel}

ตาราง 5- 1 แสดง ความสัมพันธ์ของเอกสารให้ชื่อ doc ซึ่งยังไม่เป็น 1 NF

ตาราง 5-1 แสดงตัวอย่างความสัมพันธ์ของเอกสารให้ชื่อว่า *doc* ซึ่งสามารถแสดงเป็น 1 NF ดังตารางที่ 5-2 เนื่องจากโดเมนต้องไม่สามารถแบ่งย่อยได้ (atomic) เพื่อให้เป็น 1 NF ดังนั้นจึงจะต้องมีทับเพิ้ลหนึ่งสำหรับแต่ละคู่ (keyword, author) และ *date* จะถูกแทนด้วยสามแอททริบิวท์ ซึ่งแต่ละแอททริบิวท์แทนแต่ละฟิลด์ย่อยของ *date* และกำหนด multivalued dependencies ดังนี้

- title ->-> author
- title ->-> keyword
- title -> day month year

จากนั้นสามารถทำให้เป็น 4 NF โดยให้ schema ดังนี้

(title, author)

(title, keyword)

(title, day, month, year)

t i t l e	Author	Day	month	year	keyword
salesplan	Smith	1	April	89	profit
salesplan	Jones	1	April	89	profit
salesplan	Smith	1	April	89	strategy
salesplan	Jones	1	April	89	strategy
status report	Jones	17	June	94	profit
status report	Frick	17	June	94	profit
status report	Jones	17	June	94	personnel
status report	Frick	17	June	94	personnel

ตารางที่ 5-2 flat-doc หรือ 1 NF ของความสัมพันธ์ doc

T i t l e	**author**
Salesplan	Smith
Salesplan	Jones
Status report	Smith
Status report	Frick

Title	***K e y w o r d***
Salesplan	Profit
Salesplan	Strategy
Status report	Profit
Status report	Personnel

Title	**day**	**month**	**year**
Salesplan	1	April	89
status report	17	June	94

ตารางที่ 5-3 4 NF ของ flat-doc ในตารางที่ 5-2

5.3 Structured and Collection Types

พิจารณาคำสั่งต่อไปนี้ซึ่งนิยามความสัมพันธ์ *doc* โดยการใช้แอททริบิวท์แบบซับซ้อน

create type *MyString* **char varying**

create type *MyDate*

(*day* **integer**,

month **char(10)**,

year **integer)**

create type *Document*

(*name MyString*,

author-list **setof**(*MyString*),

date MyDate,

keyword-list **setof**(MyString))

create table *doc* **of type** *Document*

ตารางที่เกิดจากคำสั่งด้านบนต่างจากตารางตามนิยามของฐานข้อมูลแบบรีเลชั่นนัล เพราะอนุญาตให้มีเซ็ตของแอททริบิวท์ และแอททริบิวท์ที่เป็นเรคคอร์ด (record) ได้ ซึ่งสิ่งเหล่านี้ทำให้ สามารถแสดง composite attributes และ multivalued attributes ของไดอะแกรมแบบ ER ได้โดยตรง

ชนิดข้อมูลที่ถูกสร้างขึ้นจากคำสั่งเช่นด้านบนนี้ จะถูกเก็บลงในฐานข้อมูล ดังนั้นคำสั่งอื่นๆจึงสามารถนำชนิดข้อมูลเหล่านี้ไปใช้ได้ด้วย

ชนิดข้อมูลซับซ้อนยังสนับสนุนชนิดข้อมูลสะสม (collection type) อื่นเช่นอาร์เรย์และ มัลติเซ็ต (ข้อมูลสะสมที่สามารถมีข้อมูลสองตัวที่มีค่าซ้ำกันได้) ดังตัวอย่างต่อไปนี้

author-array MyString[10]

print-runs **multiset(integer)**

5.4 Inheritance

การสืบทอดคุณสมบัติ (Inheritance) สามารถทำได้ในระดับชนิดข้อมูลและระดับตาราง สมมติให้มีนิยามชนิดข้อมูลดังนี้

create type *Person*
(*name MyString,*
social-security **integer)**

เราอาจต้องการเก็บว่าใครเป็นนักเรียนและใครเป็นอาจารย์ เนื่องจากนักเรียนและอาจารย์ต่างก็เป็น *Person* อยู่แล้ว เราจึงสามารถใช้การสืบทอดคุณสมบัติเพื่อนิยามชนิดข้อมูลนักเรียนและอาจารย์ดังนี้

create type *Student*
(*degree MyString,*
department MyString)
under *Person*
create type *Teacher*
(*salary* **integer,**
department MyString)
under *Person*

ทั้ง *Teacher* และ *Student* ถ่ายทอดแอททริบิวท์ของ *Person* คือ *name* และ *social-security* กล่าวได้ ว่า *Student* และ *Teacher* เป็นสับไทป์ (Subtype) ของ *Person* และ *Person* เป็นซูเปอร์ไทป์ (Supertype) ของ *Student* และ *Teacher*

ถ้าเราต้องการเก็บข้อมูลของ Teacher Assistant ซึ่งเป็นทั้งอาจารย์และนักเรียน ก็สามารถทำได้โดยใช้ multiple inheritance

create type *TeachingAssistant*
under *Student, Teacher*

จากคำสั่งด้านบนอาจทำให้เกิดปัญหาว่า *department* จะถ่ายทอดจาก *Student* หรือ *Teacher* เพื่อหลีกเลี่ยงปัญหานี้ เราสามารถตั้งชื่อใหม่โดยใช้ **as** ดังตัวอย่างต่อไปนี้

create type *TeachingAssistant*
under *Student* **with** (*department* **as** *student-dept*),
Teacher **with** (*department* **as** teacher-dept)

ถ้าใช้การถ่ายทอดคุณสมบัติในระดับตาราง

create table *people*
(*name MyString*,
social-security **integer**)

จากนั้นให้นิยามของ *students* และ *teachers*

create table *students*
(*degree MyString*,
department MyString)
under *people*

create table *teachers*
(*salary* **integer**,
department MyString)
under *people*

ตารางย่อย *students* และ *teachers* จะถ่ายทอดคุณสมบัติของตาราง *people* มีเงื่อนไขสำหรับ Subtable และ Supertable อยู่ดังนี้

- แต่ละทับเพิ้ลของ Supertable สามารถเหมือนกับทับเพิ้ลของ Subtable ได้มากที่สุดหนึ่งทับเพิ้ล

- แต่ละทับเพิ้ลของ Subtable ต้องเหมือนกับทับเพิ้ลของ Supertable หนึ่งทับเพิ้ลเท่านั้น

ถ้าไม่มีเงื่อนไขข้อแรก จากตัวอย่างอาจจะมีสองทับเพิ้ลใน *students* หรือ *teachers* ที่อ้างไปถึงคนคนเดียวกัน ถ้าไม่มีเงื่อนไขข้อที่สองอาจจะมีทับเพิ้ลใน *students* หรือ *teachers* ที่ไม่ตรงกับ *people* เลย หรือตรงกับทับเพิ้ลของ *people* มากกว่าหนึ่งทับเพิ้ล

การถ่ายทอดคุณสมบัติระดับตารางก็สามารถทำ multiple inheritance ได้เหมือนกับระดับชนิดข้อมูล

create table *teaching-assistants*

under *students* **with** (*department* **as** *student-dept*),

teachers **with** (*department* **as** *teacher-dept*)

เงื่อนไขทั้งสองข้อจะรับประกันว่า ถ้ามีเอนติตี้หนึ่งอยู่ในตาราง *teaching-assistants* เอนติตี้นั้นก็จะมีอยู่ในตาราง *teachers* และ *students* ด้วย

5.5 Reference Types

แอททริบิวท์ของชนิดข้อมูลหนึ่งสามารถเป็นสิ่งอ้างอิงไปสู่อ็อบเจ็คของชนิดข้อมูลที่ระบุ ไว้ เช่น การอ้างอิงไปหา people เป็นชนิดข้อมูล **ref**(*Person*) ฟิลด์ author-list ในชนิด *Document* สามารถนิยามใหม่โดย

author-list **setof**(**ref**(*Person*))

ซึ่งเป็นเซ็ตของการอ้างอิงไปหาอ็อบเจ็ค *Person*

ทับเพิ้ลต่างๆของตารางก็สามารถมีการอ้างอิงไปหาได้ การอ้างไปหาทับเพิ้ลของตาราง *people* มีชนิดเป็น **ref**(*people*) เราสามารถใช้คีย์หลัก (Primary Key) ในการอ้างถึงทับเพิ้ลในตารางหรืออีกทางหนึ่งคือแต่ละแอททริบิวท์ในตารางมีตัวระบุทับเพิ้ล (Tuple Identifier) เป็นแอททริบิวท์แฝง และตัวที่ใช้อ้างถึงทับเพิ้ลก็คือตัวระบุทับเพิ้ล ตารางย่อย(Subtable) จะถ่ายทอดตัวระบุทับเพิ้ลมาพร้อมกับแอททริบิวท์อื่นๆ

5.6 Querying with Complex Types

คือการใช้ SQL ในการถามหาข้อมูล (query) เช่น หาชื่อและปีที่พิมพ์ของเอกสารแต่ละฉบับ คำสั่งในการถามคือ

select *name, date.year*

from *doc*

ใช้จุด (.) ในการอ้างถึงฟิลด์ *year* ในแอททริบิวท์ *date*

5.7 Relation-Valued Attributes

สมมติความสัมพันธ์ *pdoc* ดังนี้

create table *pdoc*
(*name MyString*,
author-list **setof**(**ref**(*people*)),
date MyDate,
keyword-list **setof**(*MyString*))

ถ้าต้องการหาเอกสารที่มีคำว่า "database" อยู่ใน keyword สามารถหาได้โดย

select *name*
from *pdoc*
where "database" **in** *keyword-list*

ถ้าต้องการความสัมพันธ์ในแบบ "document-name, author's name" สามารถหาได้โดย

select *B.name, Y.name*
from *pdoc* **as** *B*, *B.author-list* **as** *Y*

เนื่องจาก *author-list* และ *pdoc* เป็นฟิลด์ที่เป็นเซ็ต (set-valued field) จึงสามารถใช้ในส่วนของ **from** ได้

ฟังก์ชั่นภายในเช่น **min max count** สามารถใช้ได้กับค่าที่เป็นความสัมพันธ์ได้ทั้งหมด เช่น หาชื่อและจำนวนของผู้เขียนของเอกสารแต่ละฉบับ หาได้โดย

select *name*, **count**(au*thor-list*)
from *pdoc*

เนื่องจาก *author-list* เป็นค่าที่เป็นเซ็ตซึ่งประกอบด้วยหนึ่งทับเพิ้ลสำหรับผู้เขียนแต่ละคน ดังนั้นจำนวนสมาชิกที่นับได้ในเซ็ตก็คือจำนวนผู้เขียนนั่นเอง

5.8 Path Expressions

เครื่องหมายจุดที่ใช้ในการอ้างถึงฟิลด์ย่อยในแอททริบิวท์สามารถใช้กับการอ้างอิง (reference) ได้ สมมติให้มีตาราง *people* ดังที่นิยามไว้ก่อนแล้ว และนิยามตาราง *phd-student* ดังนี้

create table *phd-students*
(*advisor* **ref**(*people*))
under *people*

สามารถหาชื่อที่ปรึกษาของนักศึกษาปริญญาเอกทั้งหมดได้โดย

select *phd-students.advisor.name*
from *phd-students*

การอ้างอิงสามารถใช้ช่อนการ join ได้ จากตัวอย่างถ้าไม่มีการใช้การอ้างอิง ฟิลด์ *advisor* ของ *phd-students* จะเป็น foreign key ของตาราง *people* ในการหาชื่อที่ปรึกษาของนักศึกษาปริญญาเอกจำเป็นต้อง join ตาราง *phd-students* กับ *people* การใช้การอ้างอิงนี้ช่วยให้การถามหา (query) ง่ายขึ้นมาก

Expression ที่มีรูปแบบเช่น "*students.advisor.name*" เรียกว่า "Path Expression"

5.9 Nesting and Unnesting

การแปลงความสัมพันธ์ที่ซับซ้อน (Nested Relation) ให้เป็น 1 NF เรียกว่า "Unnesting" ดังเช่นตารางที่ 1 เมื่อ unnest แล้วจะได้ตารางที่ 2

กระบวนการที่ย้อนกลับการ Unnesting คือNesting จากตารางที่ 2 สามารถทำ Nesting โดย

select *title, author,* (*day, month, year*) **as** *date,*
set (*keyword*) **as** *keyword-list*
from *flat-doc*
group by *title, author, date*

ผลลัพธ์จากคำสั่งนี้แสดงในตารางที่ 5-4

T i t l e	author	date (date, month, year)	keyword-list
salesplan	Smith	(1, April, 89)	{profit, strategy}
salesplan	Jones	(1, April, 89)	{profit, strategy}
status report	Jones	(17, Junes, 94)	{profit, personnel}
status report	Frick	(17, Junes, 94)	{profit, personnel}

ตารางที่ 5-4 แสดงตารางที่ 5-2 หลังจากทำ Nesting

5.10 Functions

ระบบวัตถุสัมพันธ์ยอมให้ผู้ใช้สามารถกำหนดฟังก์ชั่นขึ้นมาได้เอง ซึ่งสามารถกำหนดโดยใช้ภาษาโปรแกรมมิ่ง เช่น C หรือ C++ หรือภาษาจัดการข้อมูล เช่น SQL

ในการใช้ SQL สร้างฟังก์ชั่น สมมติว่ากำหนดชื่อเอกสาร ให้บอกจำนวนผู้เขียนเอกสารนั้น สามารถกำหนดเป็นฟังชั่นได้ดังนี้

create function *author-count*(*one-doc Document*)
return integer as
select count (*author-list*)
from *one-doc*

Document เป็นชื่อชนิดข้อมูล **select** จะกระทำกับความสัมพันธ์ *one-doc* ซึ่งก็คือ argument ของฟังก์ชั่น ในการ return ของฟังก์ชั่น ถ้าผลลัพธ์มีมากกว่าหนึ่งทับเพิ้ล ระบบสามารถจะดำเนินการต่อได้สองทางคือ คิดว่าเป็นความผิดพลาด (error) หรือเลือกทับเพิ้ลใดทับเพิ้ลหนึ่งมาเป็นผลลัพธ์

ในการกำหนดฟังก์ชั่นโดยภาษาโปรแกรมมิ่งอื่นๆนั้นมีข้อดีที่ฟังก์ชั่นจะมีความสามารถสูงกว่ากำหนด้วย SQL เพราะ SQL ไม่สามารถทำงานบางอย่างได้ เช่นการคำนวณจำนวนเชิงซ้อน ฟังก์ชั่นที่กำหนดโดยภาษาโปรแกรมมิ่งนี้จะต้องคอมไพล์ภายนอกแล้วโหลดเข้ามา Execute พร้อมกับโค้ดของฐานข้อมูล ขั้นตอนเหล่านี้มีความเสี่ยงที่ข้อผิดพลาด (bug) ของโปรแกรมจะทำให้เกิดความเสียหายกับโครงสร้างของข้อมูล และยังสามารถลัดผ่านการควบคุมของระบบฐานข้อมูลด้วย

เมื่อฟังก์ชั่นที่ผู้ใช้กำหนดขึ้นเองนี้ถูกเรียกใช้ คำสั่งอาจจะถูกกระทำโดยระบบฐานข้อมูลเอง หรือถ่ายข้อมูลที่เกี่ยวข้องไปไว้แยกต่างหากก่อน กรณีแรกมีความเสี่ยงจากความผิดพลาดของฟังก์ชั่น ในขณะที่กรณีหลังจะใช้ overhead สูงมาก

5.11 Creation of Complex Values and Objects

สามารถสร้างทับเพิ้ลของชนิดข้อมูลที่กำหนดโดยความสัมพันธ์ *doc* ดังนี้

("salesplan", **set**("Smith", "Jones"), (1, "April", 90), **set**("profit", "strategy"))

ทับเพิ้ลที่สร้างขึ้นนี้สามารถนำมาใช้ได้หลายอย่าง เช่นถ้าต้องการ insert เข้าไปในความสัมพันธ์ *doc* ทำได้โดย

insert into doc
values ("salesplan", **set**("Smith", "Jones"), (1, "April", 90), **set**("profit", "strategy"))

นอกจากนี้ยังสามารถนำไปใช้ในการ query ได้ เช่น

select *name, date*
from *doc*
where *name* **in set**("salesplan", "opportunities", "risks")

คำสั่งนี้ใช้หาชื่อและวันเวลาของเอกสารทุกฉบับที่มีชื่อคือ "salesplan" หรือ "opprtunities" หรือ "risks"

ในการสร้างอ็อบเจ็คใหม่ สามารถใช้ Construction function, Constructor function ของชนิดข้อมูล T คือ T() เมื่อถูกเรียกใช้จะสร้างอ็อบเจ็คใหม่ (uninitialized)ที่มีชนิดเป็น T ขึ้นมา ใส่ค่าให้กับฟิลด์ **oid** แล้วส่งอ็อบเจ็คกลับออกมา หลังจากนั้นอ็อบเจ็คจะต้องถูก initialize

ในการ update สามารถทำได้โดย **update** ของ SQL

บทที่ 6

NIAM Conceptual Schema

6.1 แบบจำลองระดับแนวคิดไนแอม (The NIAM Conceptual Schema)

แบบจำลองระดับแนวคิดไนแอมนั้น คำว่า NIAM นั้นย่อมาจาก Nijssen's Informiation Analysis Methodology เป็นรูปแบบหนึ่งของแบบจำลองระดับแนวคิดที่ได้มีการคิดค้นมาตั้งแต่ปลายปี ค.ศ. 1977 โดย Prof. G.M. Nijssen และ E.D. Falkenberg ซึ่งไนแอมเป็นแบบจำลองที่มีพื้นฐานมาจากโครงสร้างภาษาธรรมชาติ ที่ใช้รูปประโยคมีประธาน กริยา และกรรม ที่มีความหมาย ใช้เครื่องหมายแสดงความสัมพันธ์ของข้อมูล และข้อจำกัดของข้อมูลได้อย่างชัดเจน

การสร้างแบบจำลองระดับแนวคิดไนแอมจะมีพื้นฐานอยู่บนการกำหนดตัวอย่างข้อมูล และหลังจากที่ได้ผ่านกระบวนการ (Procedure) ที่ได้กำหนดไว้ เราจะได้แผนภาพที่มีความหมายในการแทนแบบจำลองระดับแนวคิดดังกล่าว นอกจากนี้แล้วยังสามารถแปลงแบบจำลองนี้เป็นโครงสร้างของฐานข้อมูลเชิงสัมพันธ์ (Relation Database Schema) ซึ่งจะได้ผลลัพธ์อยู่ในรูปของ Fifth Normal Form จึงมีการนำแบบจำลองระดับแนวคิดไนแอม มาใช้ช่วยในการออกแบบฐานข้อมูลเชิงสัมพันธ์กันอย่างแพร่หลาย

6.2 องค์ประกอบสำคัญของแบบจำลองระดับแนวคิดไนแอม

6.2.1 ส่วนประกอบพื้นฐาน ประกอบไปด้วย

1. **ชนิดเอนติตี้ (Entity Type)** หมายถึง เซตของสิ่งที่สนใจทั้งที่อยู่ในรูปของนามธรรม หรือรูปธรรม ซึ่งอาจเป็นสิ่งที่จับต้องได้หรือไม่ได้ เช่น คน, ภาควิชา, บริษัท, รถยนต์ เป็นต้น
2. **ชนิดเลเบิล (Lable Type, Value Type)** หมายถึง เซตของสิ่งที่ใช้บ่งบอกความแตกต่าง หรือชื่อของแต่ละเอนติตี้ที่กำหนด เช่น ชื่อ, นามสกุล, รหัสประจำตัว, ทะเบียนรถยนต์ เป็นต้น
3. **บทบาท (Role)** หมายถึง ความสัมพันธ์ที่เกี่ยวข้องกับชนิดเอนติตี้ที่สัมผัสอยู่ เช่น เอนติตี้นักศึกษา แสดงบทบาท เป็นผู้ลงทะเบียนเรียนในวิชานั้นๆ เป็นต้น
4. **ประโยคความจริงมูลฐาน (Element Fact Type)** หรืออาจเรียกว่าชนิดความจริง (Fact Type) หมายถึง เซตของความสัมพันธ์ระหว่างสมาชิกของชนิดเอนติตี้ตั้งแต่ 2 เอนติตี้ขึ้นไป โดยขนาดของชนิดความจริงจะขึ้นอยู่กับจำนวนบทบาทที่เกี่ยวข้อง โดยที่ชนิดความจริงที่มีจำนวน 2 บทบาท จะเรียกว่า Binary fact type ส่วนชนิดความจริงที่มี 3 บทบาท จะเรียกว่า Ternary fact type สำหรับชนิดความจริงที่มีมากกว่า 3 บทบาทขึ้นไป จะรวมเรียกว่า n-ary fact type
5. **ชนิดอ้างอิง (Reference Type)** หมายถึง เซตของความสัมพันธ์ระหว่างสมาชิกของชนิดเอนติตี้กับสมาชิกของชนิดเลเบิลที่มีอยู่

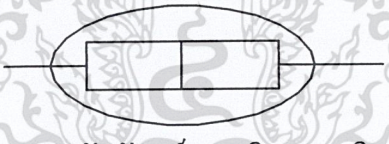
6. **ชนิดความจริงแบบเนส (Nested Fact Type)** หมายถึง ชนิดเอนติตี้ที่แสดงความสัมพันธ์ในการกำหนดกลุ่มของชนิดความจริงที่มีตั้งแต่ 2 บทบาทขึ้นไป
7. **กฎข้อบังคับความถูกต้องของข้อมูล (Integrity Constraints)** หมายถึง สิ่งที่ใช้แสดงกฎที่ใช้ในการบังคับควบคุมความถูกต้องของข้อมูล

6.2.2 สัญลักษณ์พื้นฐานที่ใช้ในแบบจำลองระดับแนวคิดไนแอม

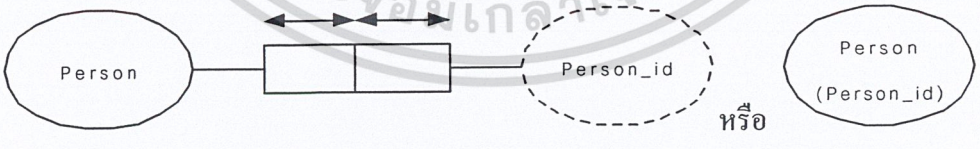
รูปที่ 6-1 แสดงสัญลักษณ์ของชนิดเอนติตี้ A และตัวอย่างของชนิดเอนติตี้คน

รูปที่ 6-2 แสดงสัญลักษณ์ของชนิดเลเบิล B และตัวอย่างของชนิดเลเบิลชื่อคน

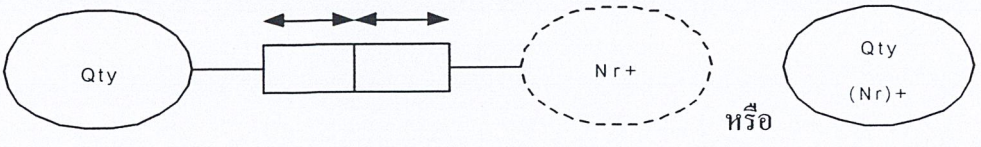
รูปที่ 6-3 แสดงสัญลักษณ์ชนิดความจริงแบบ 2 บทบาท และ 3 บทบาท



รูปที่ 6-4 แสดงสัญลักษณ์ของชนิดความจริงแบบเนส



รูปที่ 6-5 แสดงสัญลักษณ์ของชนิดอ้างอิงที่มีความสัมพันธ์แบบหนึ่งต่อหนึ่งหน่วย

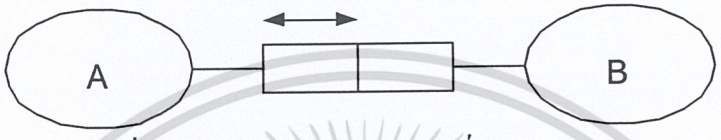


รูปที่ 6-6 แสดงสัญลักษณ์ของชนิดอ้างอิงที่มีความสัมพันธ์หนึ่งต่อหนึ่งกับชนิดเลเบิลแบบตัวเลข

6.2.2 กฎข้อบังคับความถูกต้องของข้อมูลที่ใช้ในแบบจำลองระดับแนวคิดไนแอม

1. **Intra fact type constraints (internal uniqueness constraints)** เป็นกฎข้อบังคับความถูกต้อง เพื่อทำการกำหนดบทบาทที่ใช้แสดงความสัมพันธ์ระหว่างสมาชิกของชนิดเอนติตี้หนึ่งกับสมาชิกของชนิดเอนติตี้อื่น หรือกับสมาชิกของชนิดเลเบิล โดยสามารถแบ่งเป็นรูปแบบต่าง ๆ ได้ดังต่อไปนี้

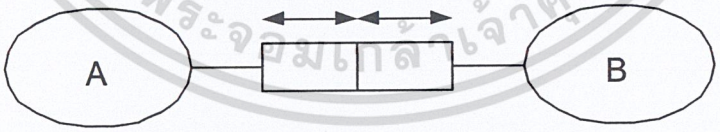
   1.1. **การกำหนดความสัมพันธ์แบบหนึ่งต่อหลายหน่วย (one to many relationship)** ซึ่งสามารถแสดงบนแผนภาพได้ดังรูป



รูปที่ 6-7 แสดงความสัมพันธ์แบบหนึ่งต่อหลายหน่วย

ลักษณะเช่นนี้สามารถแสดงได้ว่า ชนิดเอนติตี้ A จะแสดงความสัมพันธ์กับ ชนิดเอนติตี้ หรือ ชนิดเลเบิล B ได้อย่างมากที่สุดเพียงหนึ่งความสัมพันธ์เท่านั้น แต่ในทางกลับกัน ชนิดเอนติตี้ หรือ ชนิดเลเบิล B จะแสดงความสัมพันธ์กันชนิดเอนติตี้ A ได้หลายความสัมพันธ์ โดยกฎข้อบังคับความถูกต้องจะต้องทำการควบคุมไม่ให้เกิดการซ้ำซ้อนของข้อมูลในคอลัมน์ A ขึ้นได้ เช่น คนหนึ่งคนจะมีมารดาได้เพียงคนเดียวเท่านั้น แต่ในทางกลับกัน คนเพียงคนเดียวอาจเป็นมารดาของคนหลายคนได้

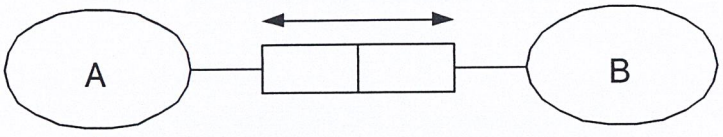
1.2. **การกำหนดความสัมพันธ์แบบหนึ่งต่อหนึ่งหน่วย (one to one relasionship)** ซึ่งสามารถแสดงบนแผนภาพได้ดังรูป



รูปที่ 6-8 แสดงความสัมพันธ์แบบหนึ่งต่อหนึ่งหน่วย

ลักษณะเช่นนี้สามารถแสดงได้ว่า ชนิดเอนติตี้ A จะแสดงความสัมพันธ์กับ ชนิดเอนติตี้ หรือ ชนิดเลเบิล B ได้เพียงหนึ่งความสัมพันธ์เท่านั้น โดยกฎข้อบังคับจะทำการควบคุมไม่ให้เกิดความสัมพันธ์ของข้อมูลมากกว่าหนึ่งความสัมพันธ์ เช่น คนหนึ่งคนจะมีเลขประจำตัวได้เพียงหมายเลขเดียวเท่านั้น และในทางกลับกัน หมายเลขประจำตัวหนึ่งหมายเลขจะต้องหมายถึงคนเพียงคนเดียว

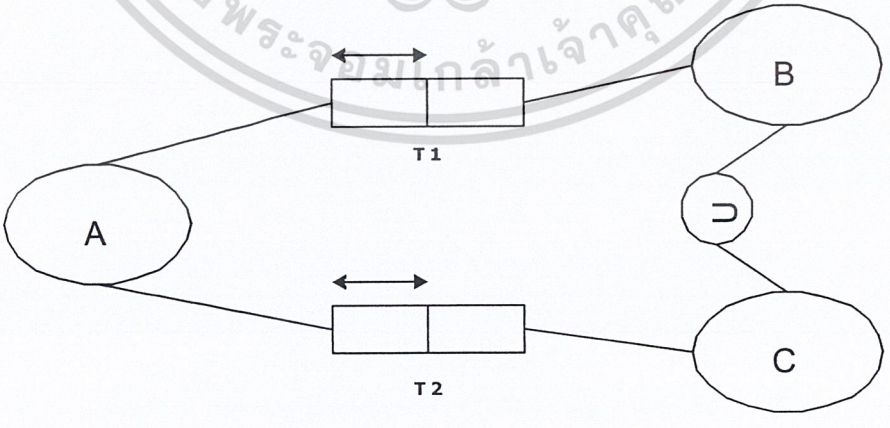
1.3. **การกำหนดความสัมพันธ์แบบหลายต่อหลายหน่วย (many to many relationship)** ซึ่งสามารถแสดงบนแผนภาพได้ดังรูป



รูปที่ 6-9 แสดงความสัมพันธ์แบบหลายหน่วยต่อหลายหน่วย

ลักษณะเช่นนี้สามารถแสดงได้ว่า ชนิดเอนติตี้ A จะแสดงความสัมพันธ์กับ ชนิดเอนติตี้ B ได้หลายความสัมพันธ์ และในทางกลับกัน ชนิดเอนติตี้ B ก็จะแสดงความสัมพันธ์กับ ชนิดเอนติตี้ A ได้หลายความสัมพันธ์เช่นกัน โดยกฎข้อบังคับความถูกต้องจะต้องทำการควบคุมความสัมพันธ์ A และ B ไม่ให้เกิดความซ้ำซ้อนเกิดขึ้นได้ เช่น นักศึกษาหนึ่งคนอาจลงทะเบียนเรียนได้หลายวิชา และวิชาใดๆ ก็สามารถรองรับนักศึกษาได้หลายคน แต่นักศึกษาหนึ่งคนจะไม่สามารถลงทะเบียนวิชาใดๆ ได้มากกว่า 1 ครั้งของการลงทะเบียน

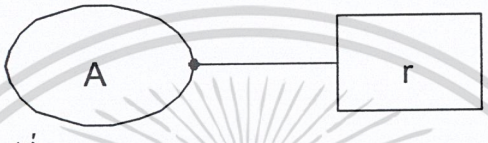
2. **Inter fact type uniqueness constraints (external uniqueness constraints)** เป็นกฎข้อบังคับความถูกต้องที่แสดงให้เห็นว่าชนิดเอนติตี้ใดๆ มีความสัมพันธ์กับชนิดเลเบิล หรือ ชนิดเอนติตี้ ได้มากกว่าหนึ่ง โดยในทางกลับกัน ชนิดเลเบิล หรือ ชนิดเอนติตี้ เหล่านั้น สามารถบ่งบอกถึงลักษณะเฉพาะของชนิดเอนติตี้นั้นได้ดังแสดงในแผนภาพดังนี้



รูปที่ 6-10 แสดง Inter fact type uniqueness constrains

ลักษณะเช่นนี้สามารถแสดงได้ว่ากฎข้อบังคับความถูกต้องจะทำการควบคุมหากนำ T1 join T2 แล้วผลที่ได้ BC จะไม่เกิดความซ้ำซ้อนกันขึ้น เช่น คนหนึ่งคนอาจมีชื่อหรือนามสกุลซ้ำกันได้ แต่ถ้า รวมทั้งชื่อและนามสกุลแล้วจะไม่เกิดความซ้ำซ้อนดังนั้นจะสามารถบ่งบอกได้ว่าเป็นการระบุถึงคนใด

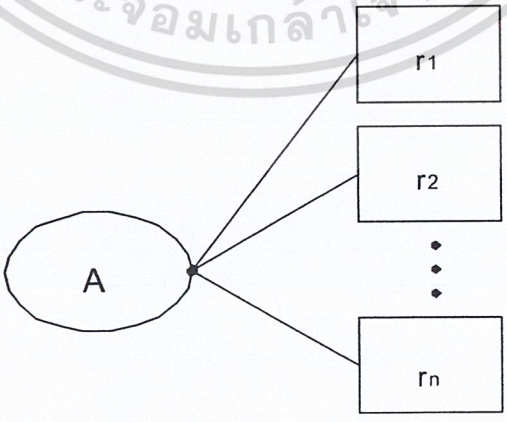
3. **Mandatory role constraints** เป็นกฎข้อบังคับความถูกต้องที่ใช้ในการควบคุมเพื่อแสดงให้เห็นถึงการมีอยู่ของข้อมูลว่าต้องมีการบันทึกข้อมูลทุกครั้งที่เกิดมีความสัมพันธ์เกิดขึ้น สามารถแสดงได้ในแผนภาพดังนี้



รูปที่ 6-11 ภาพแสดง Mandatory role constraints

จากภาพจุดทึบที่เชื่อมต่อระหว่าง ชนิดเอนติตี้ กับ Role นั้น แสดงให้เห็นว่าสมาชิกทุกตัวใน ชนิดเอนติตี้ A จะต้องถูกบันทึกข้อมูลเมื่อมีบทบาท r เกิดขึ้น โดยแสดงให้เห็นว่า pop(A) = pop(r) เช่น นักศึกษาทุกคนต้องมีการบันทึกชื่อและนามสกุล เป็นต้น

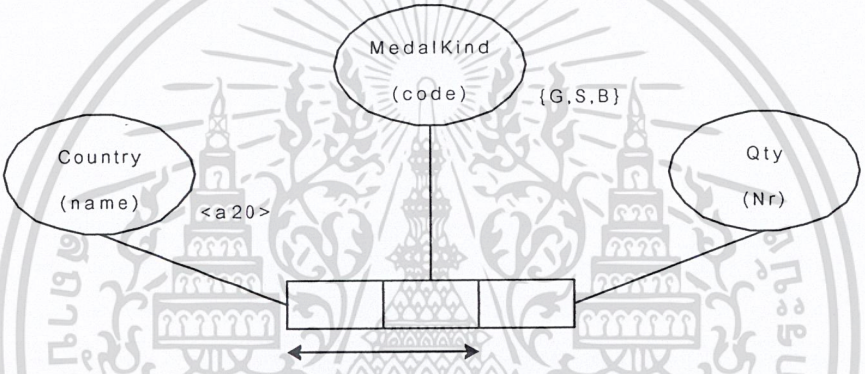
4. **Inclusion mandatory role constraints** เป็นกฎข้อบังคับความถูกต้องที่แสดงให้เห็นถึงทางเลือกของบทบาทในกลุ่มของความสัมพันธ์ ที่มีอยู่ว่าต้องมีการบันทึกข้อมูลอย่างน้อยบทบาทใดบทบาทหนึ่งของชนิดเอนติตี้นั้น ดังแสดงในแผนภาพได้ดังนี้



รูปที่ 6-12 แสดง Inclusion mandatory role constraints

จากภาพสามารถแสดงกฎข้อบังคับความถูกต้องของข้อมูล คือสมาชิกของชนิดเอนติตี้ A ใดๆ ต้องมีการบันทึกความสัมพันธ์เกิดขึ้นความสัมพันธ์ในความสัมพันธ์หนึ่ง ซึ่งแสดงได้ว่า $pop(A) = pop(r_1) \cup pop(r_2) \cup \ldots \cup pop(r_n)$ เช่น บุคคลใดๆ จะต้องมีการระบุข้อมูลของบุตร หรือข้อมูลของบิดามารดาของบุคคลนั้นๆ อย่างน้อยที่สุดหนึ่งข้อมูล

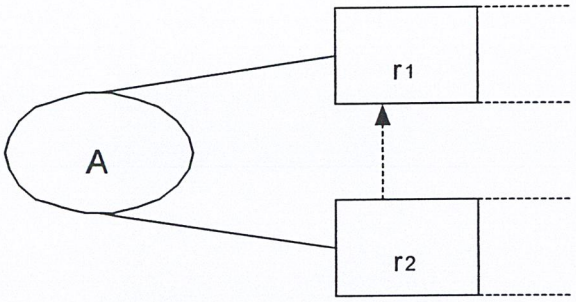
5. **Entity type constraints (Value constraints)** เป็นกฎข้อบังคับความถูกต้อง ที่ใช้ในการกำหนดค่าของสมาชิกภายในเซ็ตของข้อมูลที่เป็นไปได้ของชนิดเลเบิล หรือชนิดเอนติตี้หนึ่งๆ รวมไปถึงการกำหนดชนิดของข้อมูลในเซ็ตด้วย ดังแสดงได้ในแผนภาพดังนี้



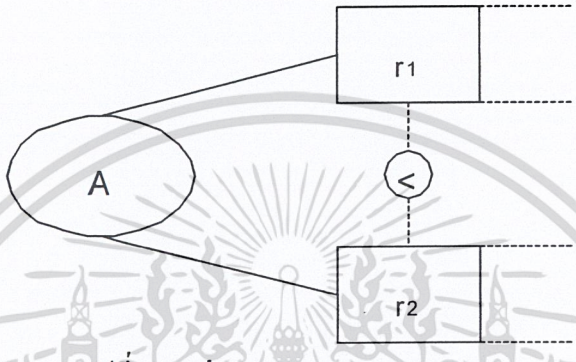
รูปที่ 6-13 แสดง Entity type constraints

จากภาพนั้นจะมีกฎข้อคับความถูกต้องของข้อมูลเพื่อทำการระบุชนิดของเหรียญรางวัลในการแข่งขันกีฬาสามารถแยกออกได้เป็น เหรียญทองแดง, เหรียญเงิน, เหรียญทอง และระบุถึงจำนวนของเหรียญรางวัลที่ได้ว่าต้องอยู่ในช่วง 1 ถึง 200 เหรียญ รวมทั้งยังสามารถระบุชนิดของข้อมูลได้ด้วย ดังที่แสดงให้เห็นว่าชื่อประเทศนั้นกำหนดให้จัดเก็บได้มากที่สุด 20 ตัวอักษร

6. **Subset constraint** เป็นกฎข้อบังคับความถูกต้องของข้อมูล ที่แสดงความสัมพันธ์ที่เป็นส่วนหนึ่งของความสัมพันธ์ที่มีอยู่ แต่จะมีลักษณะความสัมพันธ์ไปในทางเดียว ดังแสดงความสัมพันธ์ได้โดยใช้สัญลักษณ์คือ $A \rightarrow B$ ซึ่งสามารถแสดงในแผนภาพได้ดังนี้



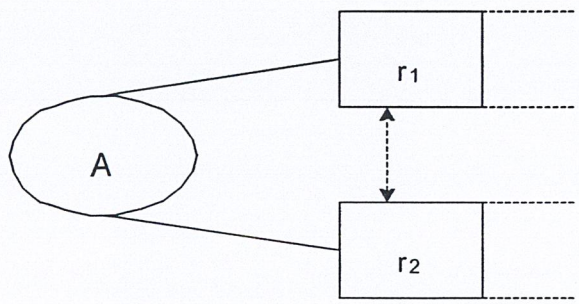
รูปที่ 6-14 แสดง Subset constraints



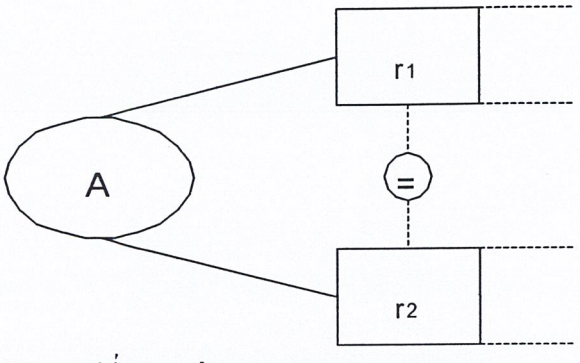
รูปที่ 6-14(ต่อ) แสดง Subset constraints

ลักษณะดังกล่าวนี้แสดงให้เห็นกฎข้อบังคับความถูกต้องของข้อมูลว่า สมาชิกแต่ละตัวของชนิดเอนติตี้ A หากมีการบันทึกความสัมพันธ์ r2 แล้ว ต้องมีการบันทึกความสัมพันธ์ r1 ด้วย แต่ในทางกลับกัน สมาชิกแต่ละตัวของชนิดเอนติตี้ A หากมีการบันทึกความสัมพันธ์ r1 แล้วไม่จำเป็นต้องมีการบันทึกความสัมพันธ์ r2 ก็ได้ เช่น บุคคลที่ชนะการแข่งขันกีฬา แสดงว่าต้องเป็นนักกีฬา แต่ผู้ที่เป็นนักกีฬาไม่จำเป็นต้องเป็นผู้ชนะการแข่งขันทุกคน

7. **Equality constraints** เป็นกฎข้อบังคับความถูกต้องที่แสดงให้เห็นว่า ชนิดเอนติตี้เหล่านั้นจะต้องมีการถูกบันทึกข้อมูลควบคู่กันเสมอไป ใช้สัญลักษณ์แสดงความสัมพันธ์ได้คือ A↔B ซึ่งสามารถแสดงในแผนภาพได้ดังนี้



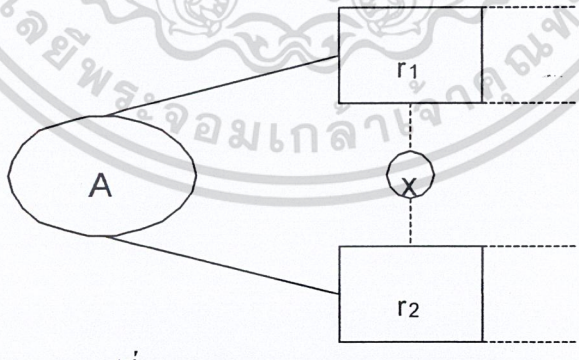
รูปที่ 6-15 แสดง Equality constraints



รูปที่ 6-15 (ต่อ) แสดง Equality constraints

ลักษณะดังกล่าวนี้ สามารถแสดงถึงกฎข้อบังคับความถูกต้องของข้อมูลว่า หากมีการบันทึกข้อมูลความสัมพันธ์ r1 ก็ต้องมีการบันทึกข้อมูลความสัมพันธ์ r2 ของสมาชิกของชนิดเอนติตี้ A ด้วย เช่น หากบุคคลใดจะทำการบันทึกระยะเวลาของการออกกำลังกาย ก็จะต้องทำการบันทึกข้อมูลของอัตราการเต้นของหัวใจด้วย และในทางกลับกัน หากมีการบันทึกข้อมูลอัตราการเต้นของหัวใจ ก็จะต้องทำการบันทึกข้อมูลระยะเวลาการออกกำลังกายด้วยเช่นกัน

8. **Exclusion constraints** เป็นกฎข้อบังคับความถูกต้องที่มีลักษณะตรงข้ามกับ Equality constraints คือ แสดงความสัมพันธ์ที่ระบุว่าหากมีความสัมพันธ์แบบหนึ่งเกิดขึ้นจะต้องไม่มีความสัมพันธ์อีกแบบหนึ่งเกิดขึ้นโดยเด็ดขาด ซึ่งสามารถแสดงในแผนภาพได้ดังนี้

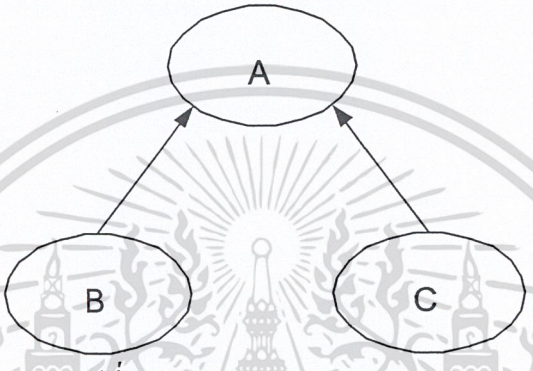


รูปที่ 6-16 แสดง Exclusion constraints

ลักษณะดังกล่าวนี้ แสดงให้เห็นกฎข้อบังคับความถูกต้องว่า หากมีการบันทึกข้อมูลความสัมพันธ์ r1 ของสมาชิกของชนิดเอนติตี้ A ใด จะต้องไม่มีการบันทึกข้อมูลความสัมพันธ์ r2 ของสมาชิกของชนิดเอนติตี้ A นั้นโดยเด็ดขาด เช่น ถ้าบุคคล

ใดถูกเลือกให้เป็นกรรมการในการตัดสินเกมส์นั้น บุคคลนั้นจะไม่มีสิทธ์เป็นผู้แข่งขันในเกมส์อย่างเด็ดขาด

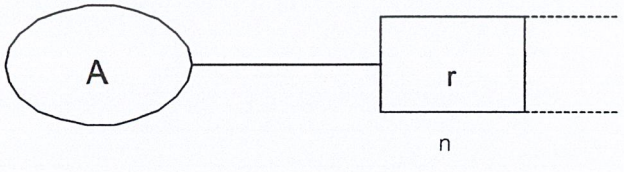
9. **Subtype constraints** เป็นกฎข้อบังคับความถูกต้องที่ระบุถึงการแบ่งกลุ่มของสมาชิกของชนิดเอนติตี้ที่มีอยู่อย่างชัดเจน ซึ่งสมาชิกของชนิดเอนติตี้ที่แบ่งแยกออกจากชนิดเอนติตี้ที่เป็น Super Type นั้น จะต้องมีลักษณะและคุณสมบัติที่แตกต่างกันอย่างชัดเจน ดังสามารถแสดงในแผนภาพได้ดังนี้



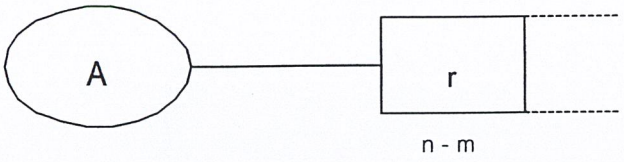
รูปที่ 6-17 แสดง Subtype constraints

ลักษณะดังกล่าวนี้แสดงให้เห็นว่า สมาชิกของชนิดเอนติตี้ A โดยจะเรียกว่า Super Type นั้นสามารถแบ่งออกเป็น 2 กลุ่มได้คือ กลุ่มของชนิดเอนติตี้ B และกลุ่มของชนิดเอนติตี้ C ซึ่งเรียกว่า Subtype เช่น ชนิดเอนติตี้ของบุคคล สามารถแบ่งออกเป็น Subtype ผู้ชาย และ ผู้หญิง ได้

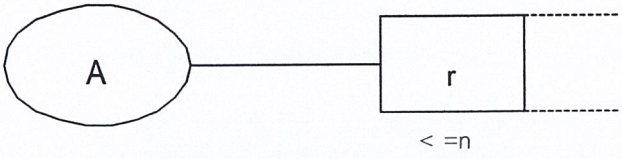
10. **Occurrence frequency constraints** เป็นกฎข้อบังคับความถูกต้องของข้อมูลที่ใช้ในการระบุจำนวนครั้งที่สมาชิกของชนิดเอนติตี้ใดๆ จะสามารถแสดงบทบาทใดบทบาทหนึ่งได้ ซึ่งสามารถแสดงในแผนภาพได้ดังนี้



(6-18a)



(6-18b)



(6-18c)

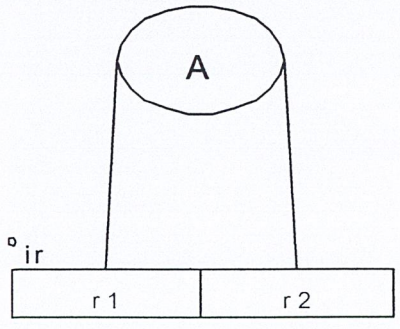
รูปที่ 6-18 แสดง Occurrence frequency constraints

จากรูปที่ 6-18(a) เป็นการแสดงกฎข้อบังคับความถูกต้องของข้อมูลโดยที่แต่ละชนิดเอนติตี้ A จะมีการแสดงบทบาทในคอลัมน์ r เป็นจำนวน n ครั้ง จากรูปที่ 6-18(b) เป็นการแสดงกฎข้อบังคับความถูกต้องของข้อมูลโดยที่แต่ละ ชนิดเอนติตี้ A ในการแสดงบทบาทในคอลัมน์ r ได้อย่างน้อยที่สุด n ครั้งและมากที่สุด m ครั้ง และจากรูปที่ 6-18(c) เป็นการแสดงกฎข้อบังคับความถูกต้องของข้อมูลโดยที่แต่ละชนิดเอนติตี้ A ในการแสดงบทบาทในคอลัมน์ r ได้อย่างน้อยที่สุด n ครั้ง เช่น ชมรมใดชมรมหนึ่งจะต้องมีสมาชิกอย่างน้อย 20 คน แต่จำนวนสูงสุดที่รับได้ต้องไม่เกิน 200 คน เป็นต้น

11. **Irreflexive constraints** เป็นกฎข้อบังคับความถูกต้องที่แสดงให้เห็นว่าสมาชิกใดๆ ของชนิดเอนติตี้ จะแสดงบทบาทเป็นทั้งประธานและกรรมของประโยคไม่ได้ กล่าวคือ ไม่มีคุณสมบัติการสะท้อนกลับของข้อมูล ดังเขียนในรูปประโยคสัญลักษณ์ได้คือ

R is irreflexive iff for all x $\sim xRx$

ซึ่งสามารถแสดงในแผนภาพได้ดังนี้



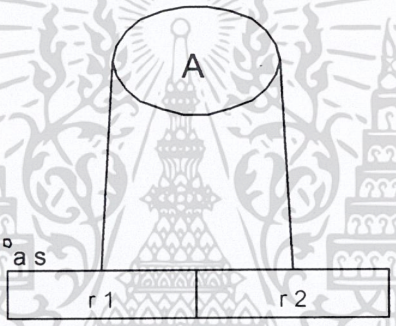
รูปที่ 6-19 แสดง Irrflexive constraints

ลักษณะดังกล่าวแสดงให้เห็นว่า กฎข้อบังคับความถูกต้องของข้อมูล แต่ละสมาชิกของชนิดเอนติตี้ A หากมีความสัมพันธ์ r1 เกิดขึ้นแล้ว จะมีความสัมพันธ์ r2 เกิดขึ้นด้วยไม่ได้บนสมาชิกตัวเดียวกัน สรุปได้ว่า สมาชิกของชนิดเอนติตี้ A ไม่มีคุณสมบัติการสะท้อนกลับ เช่น เด็กชายแดงจะเป็นพ่อของเด็กชายแดงไม่ได้

12. **Asymmetric constraints** เป็นกฎข้อบังคับความถูกต้องของข้อมูลที่เป็นสมาชิกใดๆ ของชนิดเอนติตี้ ที่ถูกจับคู่กันในการแสดงบทบาทแล้ว สมาชิกคู่นั้นจะไม่สามารถแสดงบทบาทในทิศทางที่กลับกันได้ กล่าวคือไม่มีคุณสมบัติการสมมาตรของข้อมูล ดังสามารถเขียนอยู่ในรูปประโยคสัญลักษณ์ได้คือ

$$\text{R is asymmetric iff for all x,y } xRy \rightarrow \sim yRx$$

ซึ่งสามารถแสดงในแผนภาพได้ดังนี้



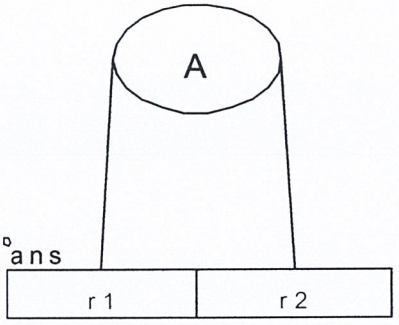
รูปที่ 6-20 แสดง Asymmetric constraints

ลักษณะดังกล่าวแสดงให้เห็นว่า กฎข้อบังคับความถูกต้องของข้อมูล แต่ละสมาชิกของเอนติตี้ A หากมีความสัมพันธ์ r1 เกิดกับสมาชิกตัวใดแล้ว จะต้องไม่มีความสัมพันธ์ r2 เกิดขึ้นย้อนกลับไปที่สมาชิกตัวเดิม สรุปได้ว่า สมาชิกของเอนติตี้ A ไม่มีคุณสมบัติการสลับที่ เช่น หากเด็กชายแดงเป็นลูกของนายดำแล้ว นายดำจะเป็นลูกของเด็กชายแดงอีกไม่ได้

13. **Antisymmetric constraints** เป็นกฎข้อบังคับความถูกต้องของข้อมูลที่เป็นสมาชิกใดๆ ของชนิดเอนติตี้ ที่ถูกจับคู่กันในการแสดงบทบาทแล้ว สมาชิกคู่นั้นจะไม่สามารถแสดงบทบาทในทิศทางที่กลับกันได้ และจะไม่สามารถแสดงบทบาทกับชนิดเอนติตี้เดิมได้ด้วย กล่าวคือไม่มีคุณสมบัติการสมมาตรของข้อมูลและคุณสมบัติการสะท้อนกลับ ดังสามารถเขียนอยู่ในรูปประโยคสัญลักษณ์ได้คือ

$$\text{R is antisymmetric iff for all x,y } x \neq y \;\; xRy \rightarrow \sim yRx$$

ซึ่งสามารถแสดงในแผนภาพได้ดังนี้



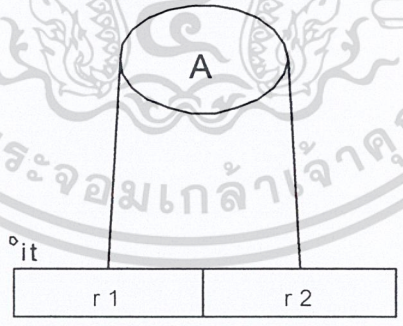
รูปที่ 6-21 แสดง Antisymmetric constraints

ลักษณะดังกล่าวแสดงให้เห็นว่ากฎข้อบังคับความถูกต้องของข้อมูล แต่ละสมาชิกของชนิดเอนติตี้ A หากมีความสัมพันธ์ r1 เกิดกับสมาชิกตัวใดแล้ว จะต้องไม่มีความสัมพันธ์ r2 เกิดขึ้นย้อนกลับไปที่สมาชิกตัวเดิม สรุปได้ว่า สมาชิกของชนิดเอนติตี้ A ไม่มีคุณสมบัติการสลับที่ เช่น หากเด็กชายแดงเป็นลูกของนายดำแล้ว นายดำจะเป็นลูกของเด็กชายแดงอีกไม่ได้

14. **Intransitive constraints** เป็นกฎข้อบังคับความถูกต้องที่แสดงให้เห็นว่าความสัมพันธ์ที่ปรากฏอยู่บน fact type จะไม่มีคุณสมบัติการถ่ายทอด ดังสามารถเขียนอยู่ในรูปประโยคสัญลักษณ์ได้คือ

$$\text{R is intransitive iff for all x,y,z } xRy \ \& \ yRz \rightarrow \sim xRz$$

ซึ่งสามารถแสดงในแผนภาพได้ดังนี้



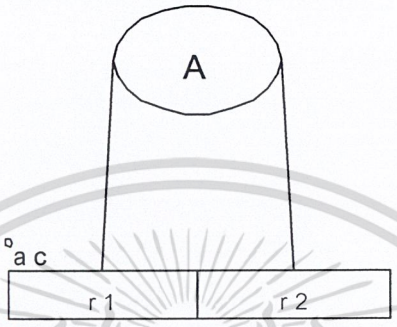
รูปที่ 6-22 แสดง Intransitive constraints

ลักษณะดังกล่าวแสดงให้เห็นว่า กฎข้อบังคับความถูกต้องของข้อมูลนั้นทำการควบคุมสมาชิกของชนิดเอนติตี้ A ไม่ให้เกิดคุณสมบัติการถ่ายทอด เช่น ถ้าเด็กชายแดงเป็นลูกของนายดำ และนายดำเป็นลูกของนายฟ้า จะไม่สามารถบันทึกข้อมูลที่ว่าเด็กชายแดงเป็นลูกของนายฟ้าได้

15. **Acyclic Ring Constraints** เป็นกฎข้อบังคับความถูกต้องที่ควบคุมความสัมพันธ์ไม่ให้มีลักษณะของความสัมพันธ์การย้อนกลับได้ ดังสามารถเขียนอยู่ในรูปประโยคสัญลักษณ์ได้คือ

$$\text{R is Acyclic Ring iff for all x,y,z } xRy \ \&\ yRz \rightarrow \sim zRx$$

ซึ่งสามารถแสดงในแผนภาพได้ดังนี้



รูปที่ 6-23 แสดง Acyclic Ring constraints

ลักษณะดังกล่าวแสดงให้เห็นว่า กฎข้อบังคับความถูกต้องของข้อมูลทำการควบคุมสมาชิกของ ชนิดเอนติตี้ A เกิดความสัมพันธ์ย้อนกลับได้ เช่น ถ้าเด็กชายแดงเป็นลูกของนายดำ และนายดำเป็นลูกของนายฟ้า จะไม่สามารถบันทึกข้อมูลที่ว่า นายฟ้าเป็นลูกของนายแดงได้

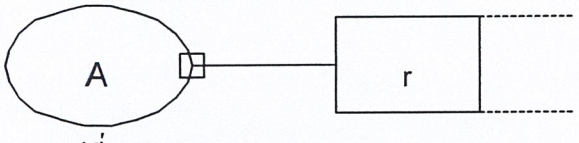
16. **Cardinality constraints** เป็นกฎข้อบังคับความถูกต้องที่ใช้ควบคุมจำนวนสมาชิกของ ชนิดเอนติตี้ ที่เป็นไปได้ดังแสดงได้ในแผนภาพดังนี้

รูปที่ 6-24 แสดง Cardinality constraints

จากภาพแสดงให้เห็นกฎข้อบังคับความถูกต้องของข้อมูลที่ทำการควบคุมจำนวนสมาชิกของชนิดเอนติตี้ A ให้มีการเกิดขึ้นได้อย่างมากที่สุด ไม่เกิน n สมาชิก ในเวลาในเวลาหนึ่ง เช่น คนที่เป็นคณบดีมีได้เพียงคนเดียวเท่านั้นในเวลาใดเวลาหนึ่ง

17. **Relative closure constraints** เป็นกฎข้อบังคับความถูกต้องที่แสดงให้เห็นถึงการมีความสัมพันธ์ของชนิดเอนติตี้หนึ่ง ว่ามีการเกิดบทบาทของชนิดเอนติตี้ดังกล่าว

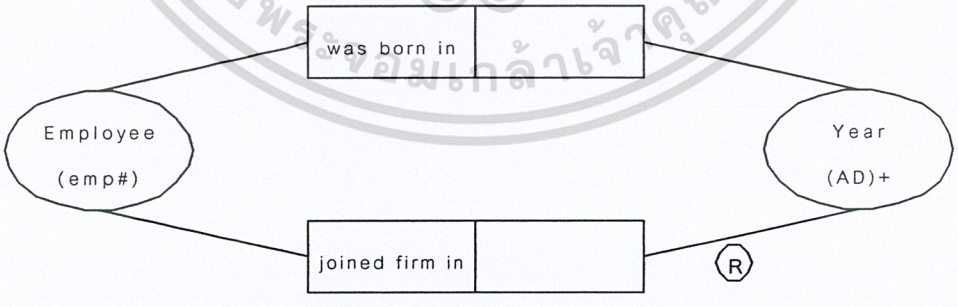
จริงหรือไม่ ดังนั้นถ้าไม่มีการเกิดบทบาทดังกล่าวถือได้ว่า ชนิดเอนติตี้นั้น ไม่มีความสัมพันธ์นั้นอยู่ในข้อมูลความจริง ดังแสดงในแผนภาพได้ดังนี้



รูปที่ 6-25 แสดง Relative closure constraints

จากภาพกล่องสี่เหลี่ยมที่เชื่อมต่อระหว่างชนิดเอนติตี้กับบทบาทนั้น แสดงให้เห็นว่าสมาชิกในชนิดติตี้ A ที่มีการบันทึกข้อมูลเมื่อมีบทบาท r เกิดขึ้นนั้นแสดงว่ามีความสัมพันธ์ดังกล่าวเกิดขึ้นจริง หากสมาชิกในชนิดเอนติตี้ A ใดๆ ที่ไม่มีการบันทึกข้อมูลบทบาท r กล่าวได้ว่าสมาชิกนั้นไม่มีความสัมพันธ์ดังกล่าวเลย ถ้าไม่มีการใส่สัญลักษณ์ดังกล่าวนั้นไม่อาจสรุปได้ว่ามีความสัมพันธ์นั้นหรือไม่ อาจมีแต่ข้อมูลที่ทราบมาไม่ครบถ้วนก็เป็นได้ เช่น พนักงานคนใดที่มีบทบาทในการจัดการแสดงว่ามีสถานะเป็นผู้จัดการหากไม่มีการบันทึกบทบาทดังกล่าว ถือได้ว่าไม่มีสถานะเป็นผู้จัดการ เป็นต้น

18. **Textual constraints** เป็นกฎข้อบังคับความถูกต้องในกลุ่มของ Static constraints ที่ไม่สามารถอธิบายได้โดยใช้สัญลักษณ์แสดงความหมายของกฎข้อบังคับดังกล่าวบนแบบจำลอง ดังนั้นจึงมีการกำหนดสัญลักษณ์เพื่อระบุว่ามีกฎข้อบังคับความถูกต้องอยู่ และทำการแสดงกฎข้อบังคับดังกล่าวโดยรูปประโยค ดังแสดงในแผนภาพดังนี้



Tc1 if Employee was born in Year y1 and Employee joined firm in Year y2 then y1<y2

รูปที่ 6-26 แสดง Textual constraints

จากภาพเป็นการแสดงให้เห็นว่า ความสัมพันธ์ระหว่างชนิดเอนติตี้ Employee กับชนิดเอนติตี้ Year มีกฎข้อบังคับเกิดขึ้นโดยใช้สัญลักษณ์ดังที่ปรากฏในภาพ และมีการอธิบายความหมายของกฎข้อบังคับดังกล่าวโดยใช้ประโยคที่ว่า ถ้าพนักงานมีการระบุปีเกิด และปีที่เข้าทำงานแล้ว ปีเกิดจะต้องมีค่าน้อยกว่าปีที่เข้าทำงานเสมอ

บทที่ 7

OONIAM Conceptual Schema

OONIAM Conceptual Schema เป็นแบบจำลองระดับแนวคิดที่ได้ทำการพัฒนามาจากแบบจำลองระดับแนวคิดไนแอมเดิมที่เป็นที่นิยมใช้ เพราะง่ายและสะดวกในการนำมาใช้งาน อีกทั้งยังสามารถแสดงสัญลักษณ์ของกฎข้อบังคับความถูกต้องของข้อมูลบนแบบจำลองได้จำนวนมาก ซึ่งเป็นส่วนที่สำคัญที่จะทำให้การออกแบบฐานข้อมูลเป็นไปอย่างสมบูรณ์ สำหรับคำว่า OONIAM นั้นย่อมาจาก Object Oriented Nijssen's Information Analysis Methodology แนวความคิดที่ได้ทำการสร้างแบบจำลองระดับแนวคิด OONIAM ขึ้นมานั้น เพื่อให้แบบจำลองระดับแนวคิดไนแอมมีความสามารถที่จะรองรับคุณสมบัติเชิงวัตถุได้ ซึ่งระบบในปัจจุบันส่วนใหญ่ได้ทำการออกแบบโดยใช้แนวคิดเชิงวัตถุ ทำให้ผู้ออกแบบที่มีความเข้าใจในการออกแบบฐานข้อมูลเชิงสัมพันธ์สามารถทำการออกแบบฐานข้อมูลเชิงวัตถุได้สะดวกมากยิ่งขึ้น โดยไม่ต้องทำการศึกษาเพิ่มเติมมากนัก รวมไปถึงการปรับปรุงมาตรฐานข้อมูลเชิงสัมพันธ์ให้อยู่ในรูปฐานข้อมูลเชิงวัตถุจะทำให้สะดวกมากขึ้นไปด้วย

7.1 ส่วนประกอบและสัญลักษณ์ที่ใช้ใน OONIAM Conceptual Schema

ส่วนประกอบและสัญลักษณ์ที่ใช้ในการออกแบบจำลองระดับแนวคิด OONIAM นี้ แบ่งออกเป็น 2 กลุ่ม คือ

7.1.1 ส่วนประกอบและสัญลักษณ์ที่นำมาจากแบบจำลองระดับแนวคิดไนแอม

ประกอบไปด้วยส่วนประกอบที่ใช้ในแบบจำลองระดับแนวคิดไนแอมทั้งหมด โดยมีส่วนประกอบดังนี้

1. ชนิดเอนติตี้ (Entity Type)
2. ชนิดเลเบิล (Lable Type, Value Type)
3. บทบาท (Role)
4. ประโยคความจริงมูลฐาน (Element Fact Type) หรืออาจเรียกว่าชนิดความจริง (Fact Type)
5. ชนิดอ้างอิง (Reference Type)
6. ชนิดความจริงแบบเนส (Nested Fact Type)
7. กฎข้อบังคับความถูกต้องของข้อมูล (Integrity Constraints)

ซึ่งรายละเอียดของส่วนประกอบต่างๆ เหล่านี้ได้กล่าวไว้ในบทที่ 2 และสัญลักษณ์ที่ใช้ที่นำมาจากแบบจำลองระดับแนวคิดไนแอมรวมไปถึงสัญลักษณ์ที่แสดงถึงกฎข้อบังคับความถูกต้องของข้อมูล ที่สามารถปรากฏอยู่บนแบบจำลองระดับแนวคิดไนแอมด้วย

7.1.2 ส่วนประกอบและสัญลักษณ์เพิ่มเติมที่ใช้ในแบบจำลองระดับแนวคิด OONIAM

สำหรับส่วนประกอบและสัญลักษณ์ที่ทำการเพิ่มเติม เพื่อรองรับในการแสดงคุณสมบัติเชิงวัตถุในที่นี้ได้มีการกำหนดรูปแบบเพิ่มเติมขึ้น 3 รูปแบบดังนี้

1. **ชนิดคอมเพลกซ์ออบเจ็กต์(Complex Object Type)**

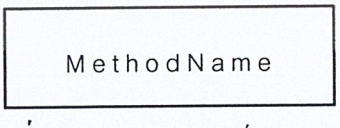
หมายถึงการรวมกันของชนิดเอนติตี้ย่อยๆ จำนวนมากกว่า 1 เอนติตี้ขึ้นไปที่มีความสัมพันธ์ในลักษณะเดียวกัน โดยประกอบขึ้นเป็นออบเจ็กต์คลาสหนึ่งคลาส และมีคุณสมบัติของคลาสในแบบจำลองเชิงวัตถุ ดังนั้นอาจจะมีการกำหนดเมธอด(Method) ของออบเจ็กต์คลาสนั้นๆ หรือไม่ก็ได้ เช่น ที่อยู่ เป็นต้น สัญลักษณ์ที่ใช้ในการแสดงชนิดคอมเพลกซ์ออบเจ็กต์สามารถแสดงได้ดังนี้

รูปที่ 7-1 แสดงสัญลักษณ์ชนิดคอมเพลกซ์ออบเจ็กต์

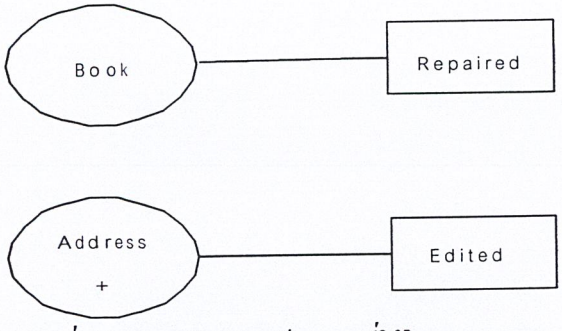
จากภาพข้างต้นเป็นสัญลักษณ์ที่ได้กำหนดเพิ่มเติม เพื่อรองรับกับชนิดคอมเพลกซ์ออบเจ็กต์ Address ซึ่ง ประกอบด้วยเอนติตี้ย่อยๆ ได้แก่ บ้านเลขที่, ถนน, เมือง

2. **เมธอด(Method)**

หมายถึงการกระทำใดๆ ที่เกิดขึ้นกับชนิดเอนติตี้หรือชนิดคอมเพลกซ์ออบเจ็กต์โดยรวมไปถึง Derived rules ที่ปรากฏอยู่บนแบบจำลองระดับแนวคิดไนแอมด้วย ในที่นี้สามารถรองรับกับกฎหรือขั้นตอนต่างๆ ที่เกิดขึ้นได้ด้วย โดยสัญลักษณ์ที่ทำการกำหนดไว้ในรูปที่ 7-2 และรูปที่ 7-3 จะแสดงให้เห็นถึงการเชื่อมโยงเมธอดกับชนิดเอนติตี้ที่เกี่ยวข้องหรือชนิดคอมเพลกซ์ออบเจ็กต์



รูปที่ 7-2 แสดงสัญลักษณ์ของเมธอด



รูปที่ 7-3 แสดงสัญลักษณ์เมธอดที่ใช้ในแบบจำลอง

3. คุณสมบัติของการเอนแคปซูเลชั่น(Encapsulation)

ซึ่งในที่นี้จะทำการกำหนดเพิ่มเติมอยู่หน้าชนิดเอนติตี้, ชนิดเลเบิล, ชนิดคอมเพลกซ์ออบเจ็กต์ รวมไปถึงชนิดความจริงแบบเนส ด้วยสัญลักษณ์ดังกล่าวยังปรากฏอยู่หน้าเมธอด เพื่อแสดงคุณสมบัติดังนี้

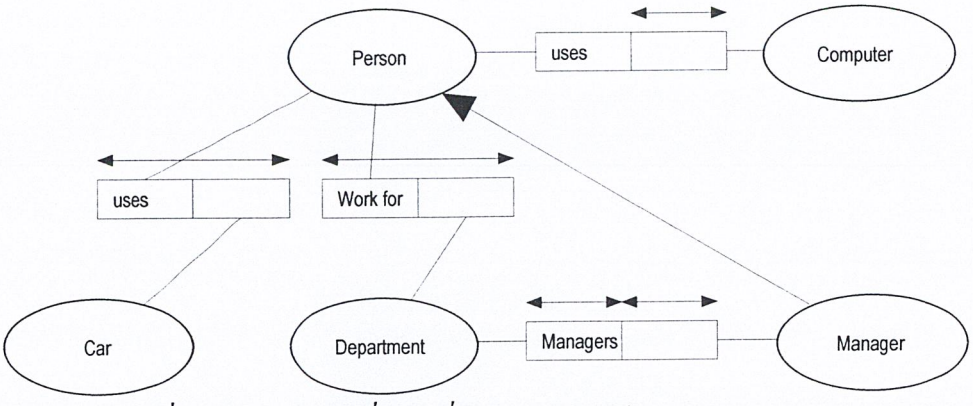
- **สัญลักษณ์(+) (Public)** หมายถึงองค์ประกอบที่มีสัญลักษณ์ดังกล่าวจะสามารถให้คลาสอื่นๆ เข้าถึงและเรียกใช้งาน
- **สัญลักษณ์(-) (Private)** หมายถึงองค์ประกอบที่มีสัญลักษณ์ดังกล่าวจะสามารถเรียกใช้หรืออ้างถึงได้เพียงคลาสเดียวเท่านั้น และไม่สามารถอ้างถึงหรือเรียกใช้งานจากคลาสลูกได้
- **สัญลักษณ์(#) (Protected)** หมายถึงองค์ประกอบที่มีลักษณะดังกล่าวจะสามารถเรียกใช้หรืออ้างถึงได้เพียงคลาสเดียว แต่อนุญาตให้คลาสลูกสามารถใช้งานได้ด้วย

7.2 Main Schema และ Sub Schema

ในแบบจำลองที่สร้างขึ้น จะมีการแบ่งเป็น 2 ระดับ ได้แก่

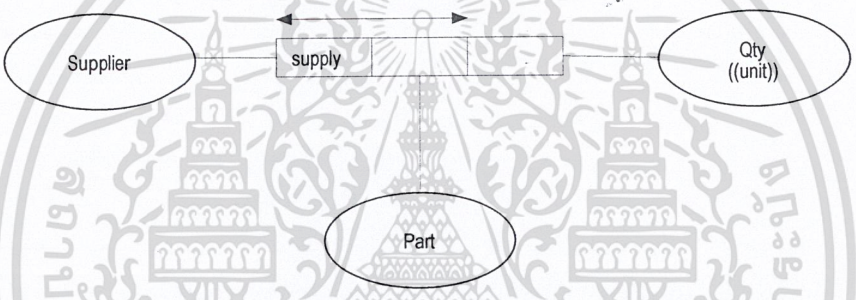
7.2.1 Main Schema

Main Schema หรือโครงร่างหลักนี้เป็นภาพรวมของความสัมพันธ์ระหว่างคลาส ดังนั้นในแต่ละคลาสจะไม่มีการแสดงแอตทริบิวต์ และไม่มี Uniqueness Identifier ดังตัวอย่างในแสดงในรูปที่ 7-4



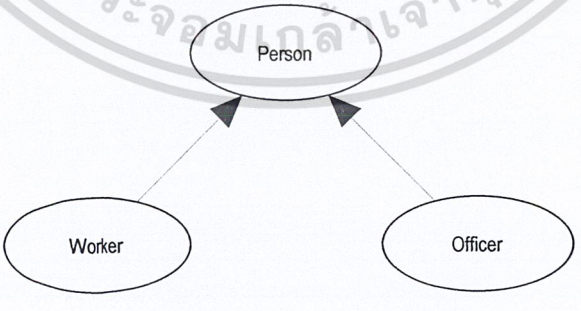
รูปที่ 7-4 โครงร่างหลักที่แสดงที่ถึงความสัมพันธ์ของคลาสหลายคลาส

หากมีความสัมพันธ์กันแบบ N-ary ก็จะสามารถมี Uniquness Identifier ได้ที่ entity ดังรูปที่ 7-5



รูปที่ 7-5 โครงร่างหลักแบบที่มี Uniqueness Identifier ที่ entity

นอกจากการอธิบายถึงความสัมพันธ์ระหว่างคลาสแล้ว ในโครงร่างหลักยังได้กล่าวถึงการถ่ายทอดคุณสมบัติของคลาส (Inheritance) อีกด้วย ดังรูปที่ 6-6

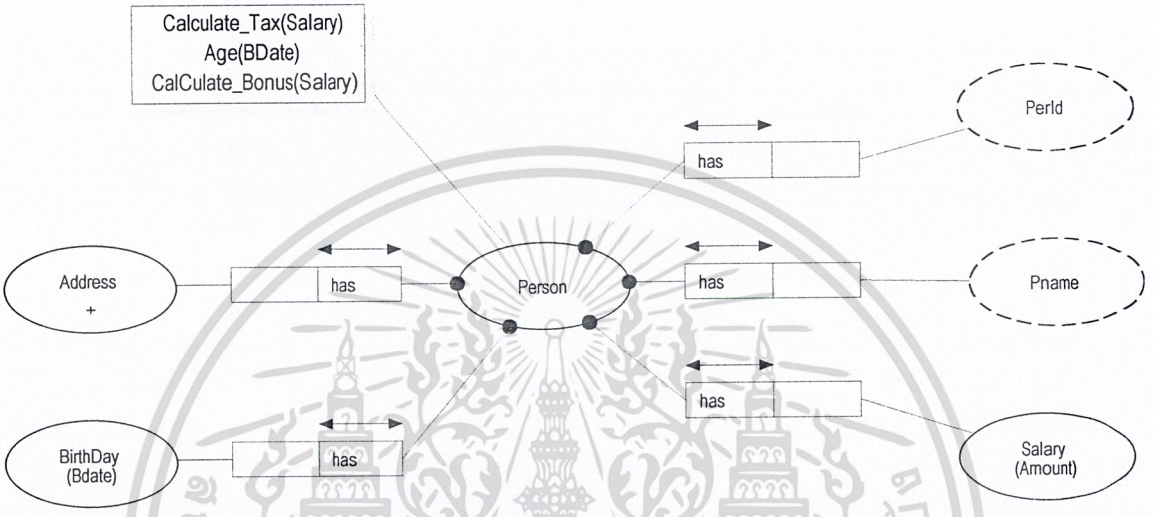


รูปที่ 7-6 การถ่ายทอดคุณสมบัติ

สำหรับโครงร่างหลักนี้จะแสดงความสัมพันธ์ระหว่างคลาสแบบหนึ่งต่อหนึ่ง หนึ่งต่อหลาย หลายต่อหนึ่ง หรือ หลายต่อหลายก็ได้ แล้วแต่การออกแบบคลาสนั้นๆ ซึ่งแอตทริบิวต์ของแต่ละคลาสและ Uniqueness Identifier นั้นจะแสดงในโครงร่างย่อย

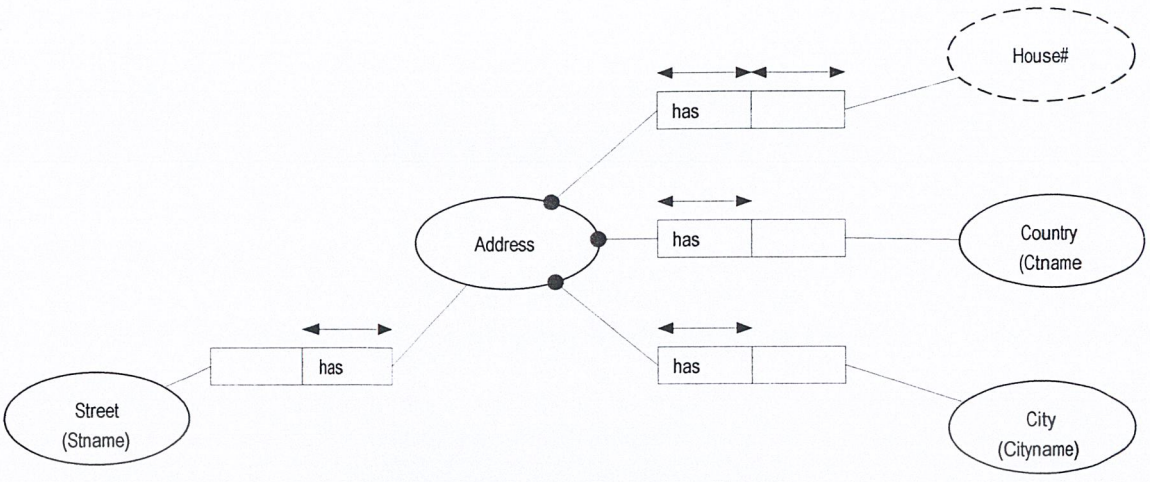
7.2.2 Sub Schema

Sub Schema หรือ โครงร่างย่อยเป็นการแสดงรายละเอียดของคลาส ซึ่งแต่ละคลาสนั้นอาจเป็นคลาสที่ซับซ้อน (Complex) ก็ได้ กล่าวคือ คลาสดังกล่าวอาจมีคลาสอื่นเป็นส่วนประกอบก็ได้ โดยโครงร่างย่อยนี้ประกอบด้วย แอตทริบิวต์, เมธอด และคลาสที่อยู่ในคลาส ดังรูปที่ 7-7



รูปที่ 7-7 โครงร่างย่อยระดับที่ 1 ของคลาส Person

จากรูปที่ 7-7 เป็นการเขียนเพิ่มขยายโครงร่างหลักในรูปที่ 7-6 กล่าวคือในรูปที่ 7-6 ได้บอกถึงความสัมพันธ์ของคลาส Person ที่มีต่อคลาสอื่นๆ แต่ยังไม่ได้อธิบายถึงรายละเอียดของคลาส Person เมื่อต้องการอธิบายรายละเอียดต่างๆ ของคลาสดังกล่าวจำเป็นต้องอธิบายในโครงร่างย่อยเท่านั้น จากโครงร่างย่อยในรูปที่ 7-7 ซึ่งเป็นโครงร่างย่อยระดับที่ 1 จะเห็นว่าคลาส Person จะมีแอตทริบิวต์ต่างๆ ดังแสดงในรูป และมีเมธอดสำหรับจัดการกับคลาสนี้ด้วยกัน 3 เมธอด นอกจากนี้แล้วสัญลักษณ์ที่แสดงชื่อคลาสหลักนี้จะไม่มี Uniqueness Identifier ที่สำคัญ คลาสนี้ยังมีคลาส Address เป็นคลาสย่อย (Embedded Class) อีก โดยคลาสย่อยนี้แสดงโดยการใช้เครื่องหมายบวก (+) ไว้ใต้ชื่อคลาส ซึ่งหากต้องการเขียนโครงร่างย่อยของคลาส Address สามารถเขียนได้ดังรูปที่ 7-8 ซึ่งเป็นโครงร่างย่อยระดับที่ 2



รูปที่ 7-8 โครงร่างย่อยของคลาส Address

จากรูปที่ 7-7 และรูปที่ 7-8 จะเห็นได้ว่าโครงร่างย่อยของคลาส Person มีด้วยกัน 2 ระดับคือ ระดับแรก เป็นการอธิบายคลาส Person (รูปที่ 7-7) ส่วนระดับที่ 2 เป็นการอธิบายคลาสที่เกี่ยวเนื่องกับคลาส Person (รูปที่ 7-8) ซึ่งจะสังเกตได้ว่าในโครงร่างย่อยแต่ละระดับนั้นหากต้องการอธิบายรายละเอียดของคลาสใด คลาสนั้นจะไม่มีการแสดง Uniqueness Identifier ในโมเดล ส่วนคลาสที่เกี่ยวเนื่องนั้นจะแสดงในโครงร่างย่อยในระดับถัดๆ ไป

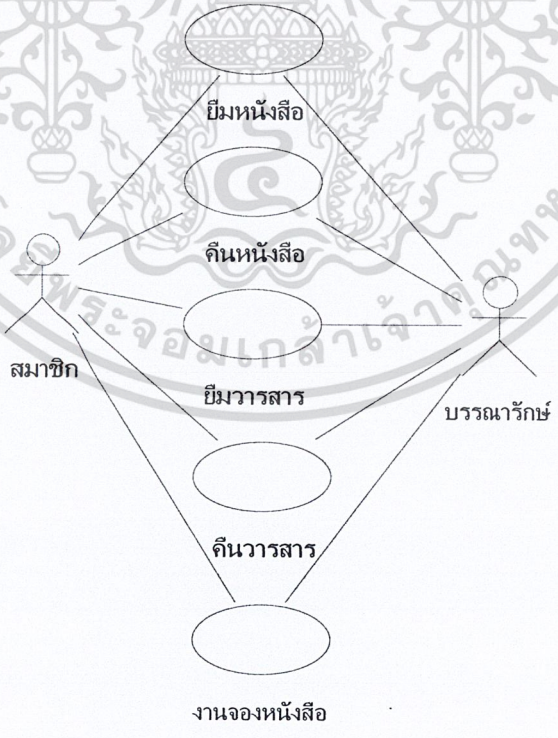
สำหรับโครงร่างย่อยทั่วๆ ไปนั้นอาจมีหลายระดับ ขึ้นอยู่กับความซับซ้อนของข้อมูลและการออกแบบของผู้ออกแบบ

บทที่ 8
การออกแบบและการสร้างแอพพลิเคชันโดยใช้ UML

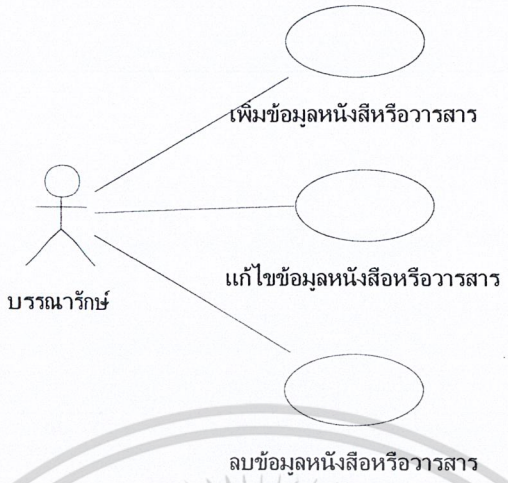
ในการสร้างและออกแบบแอพพลิเคชันโดยใช้โมเดล UML นั้นมีหลายขั้นตอน ในที่นี้จะนำเสอนวิธีการออกแบบแอพพลิเคชันโดยใช้โมเดล UML แล้วสามารถนำไปสร้างเป็นแอพพลิเคชันได้จริง โดยไดอะแกรมหลักๆ ที่จะนำมาพิจารณามีดังต่อไปนี้คือ ยูซเคสไดอะแกรม (Use-case Diagram) คลาสไดอะแกรม (Class Diagram) ซีเควนท์ไดอะแกรม (Sequence Diagram) และ แอ็กทิวิตี้ไดอะแกรม (Activity Diagram) โดยระบบที่เรานำมาศึกษาคือ ระบบห้องสมุด ซึ่งขั้นตอนต่างๆในการออกแบบโดยใช้โมเดล UML มีดังต่อไปนี้

8.1 การสร้างยูซเคสไดอะแกรม

ยูซเคสไดอะแกรมมีหน้าที่แสดงภาพรวมการทำงานของระบบที่มีทั้งการติดต่อภายในตัวมันเองและติดต่อกับภายนอก (Actor) ดังนั้นเราต้องทำการศึกษาค้นหาว่าระบบงานทำงานอะไรบ้างและ มีอะไรบ้างที่มาติดต่อเกี่ยวข้องกับระบบงาน ซึ่งในระบบห้องสมุดจะมีงานหลักๆ ที่เรานำมาพิจารณาอยู่ 3 งานด้วยกัน ก็คือ งานสมาชิก งานทรัพยากรห้องสมุด และงานบริการยืม-คืนของห้องสมุด ซึ่งได้เป็นไดอะแกรมดังต่อไปนี้



รูปที่ 8-1 แสดงยูซเคสไดอะแกรม งานบริการยืม-คืน

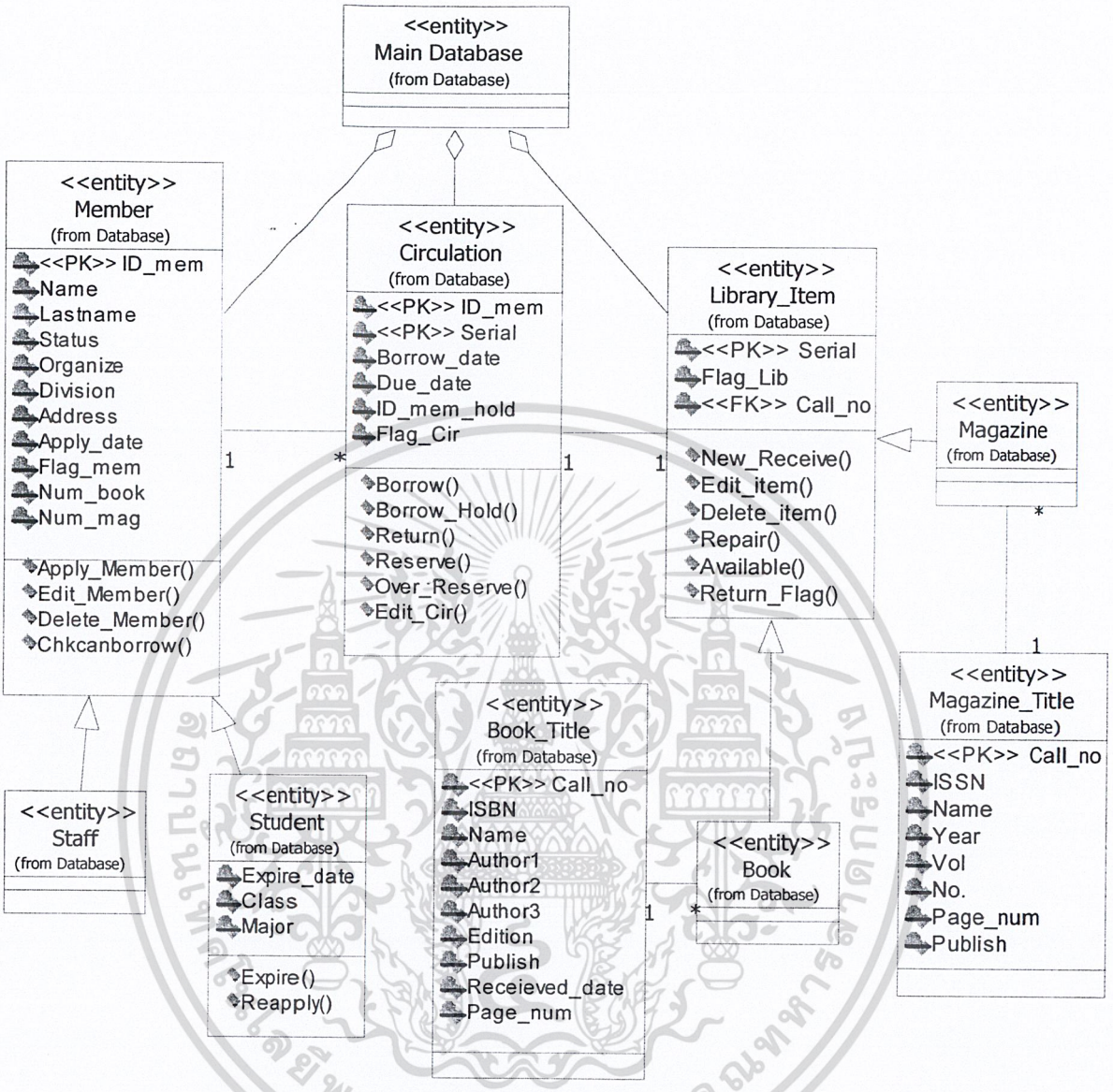


รูปที่ 8-2 แสดงยูซเคสไดอะแกรม งานทรัพยากรห้องสมุด

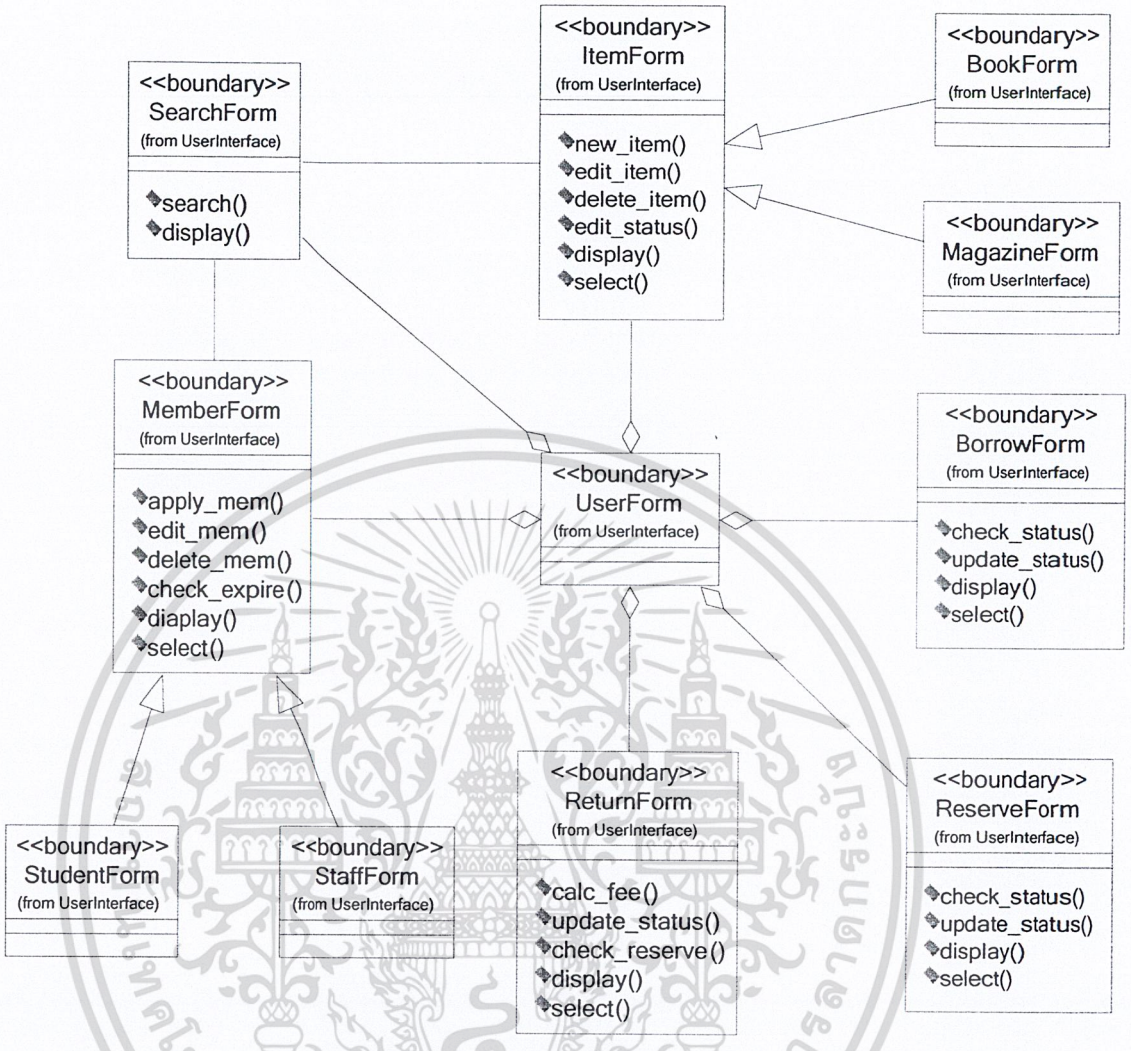
รูปที่ 8-3 แสดงยูซเคสไดอะแกรม งานสมาชิก

8.2 การสร้างคลาสไดอะแกรม

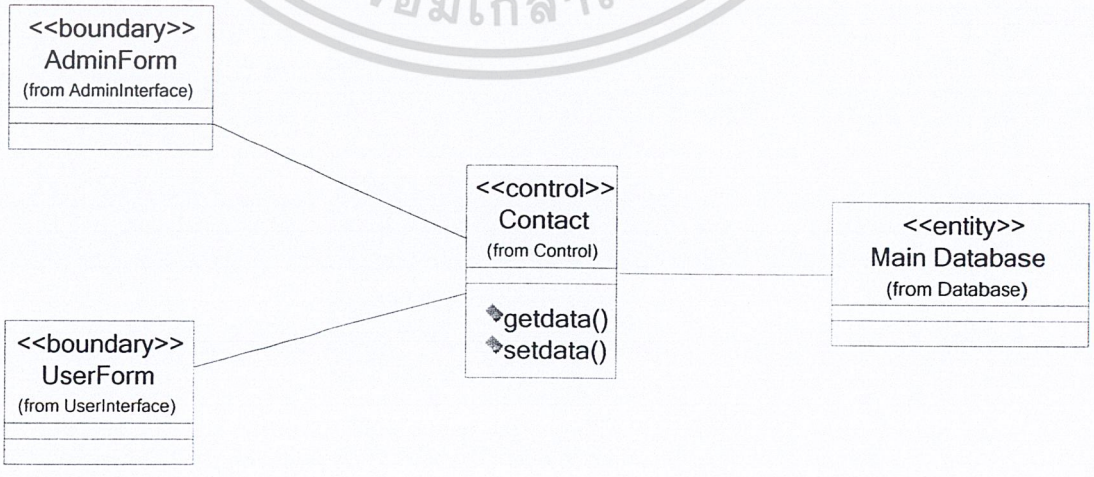
คลาสไดอะแกรมจะแสดงความสัมพันธ์และโครงสร้างของคลาสทั้งหมดที่มีอยู่ในระบบ ซึ่งจะทำให้เราสามารถรู้ได้ว่ามีคลาสไหนบ้างและแต่ละคลาสมีความสัมพันธ์กับคลาสอื่นๆ อย่างไร ซึ่งเราจะต้องนำไปใช้ใน ซีเควนท์ไดอะแกรมและไดอะแกรมอื่นๆ เพื่อความเข้าใจในระบบงาน ตัวอย่างของคลาสไดอะแกรมของระบบห้องสมุด แบ่งออกเป็น 3 ส่วนหลักๆ คือ ส่วนฐานข้อมูล ส่วนติดต่อผู้ใช้ และส่วนตัวกลางในการติดต่อระหว่างอินเทอร์เฟซกับฐานข้อมูล ซึ่งแสดงได้ดังไดอะแกรมต่อไปนี้



รูปที่ 8-4 แสดงคลาสไดอะแกรมในส่วนฐานข้อมูล



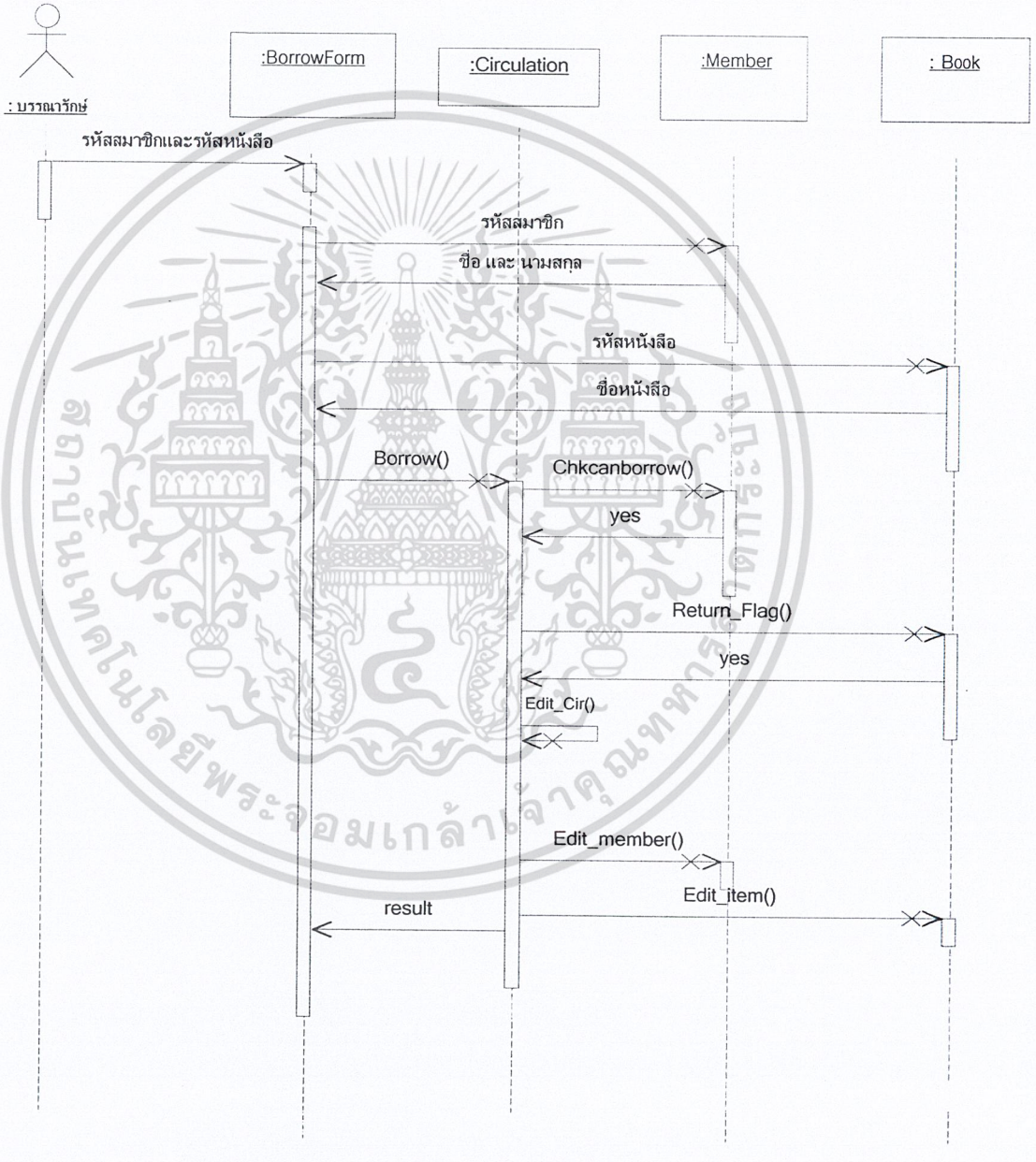
รูปที่ 8-5 แสดงคลาสไดอะแกรมในส่วนติดต่อกับผู้ใช้



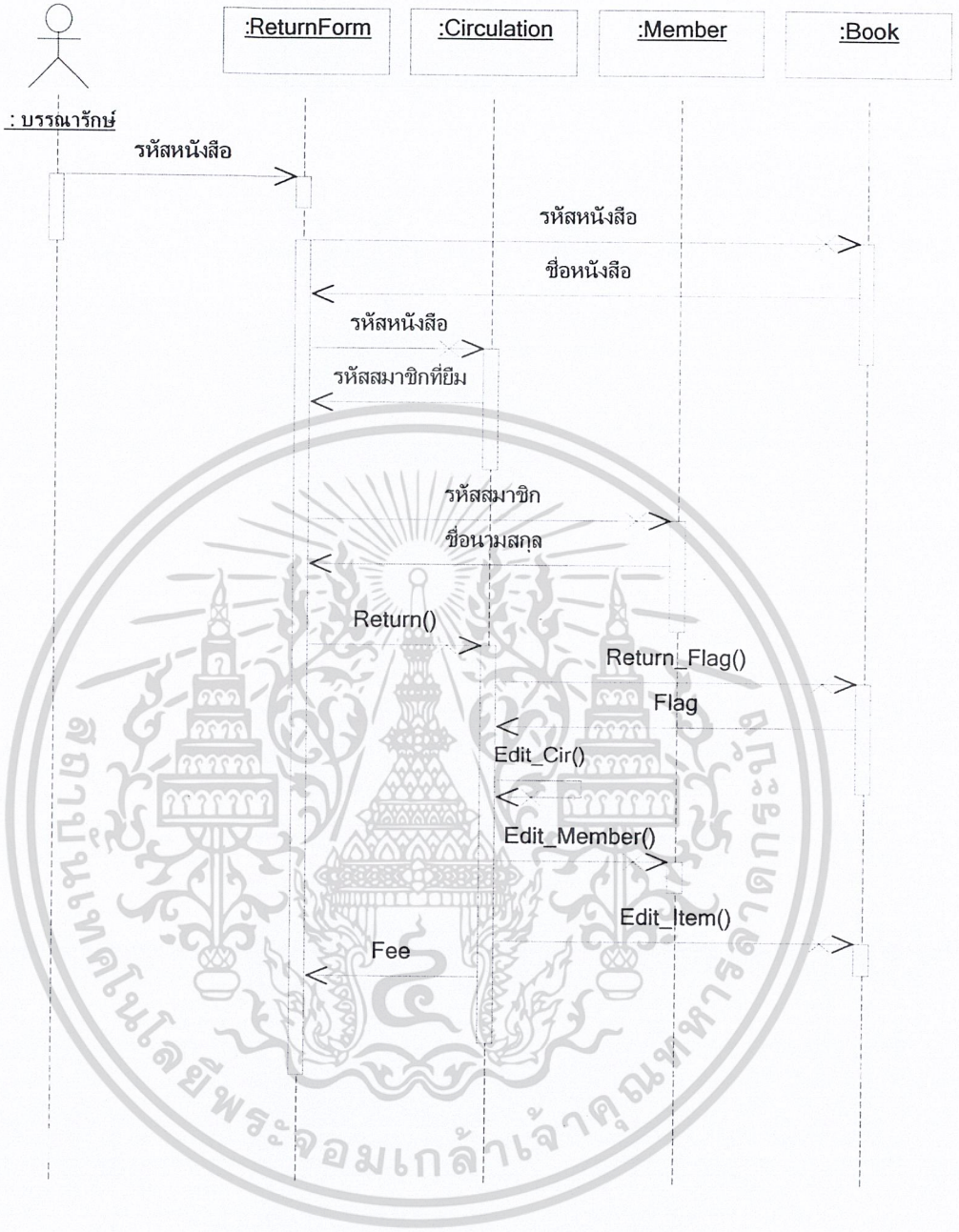
รูปที่ 8-6 แสดงคลาสไดอะแกรมในส่วนตัวกลางในการติดต่อระหว่างอินเทอร์เฟซกับฐานข้อมูล

8.3 การสร้างซีเควนท์ไดอะแกรม

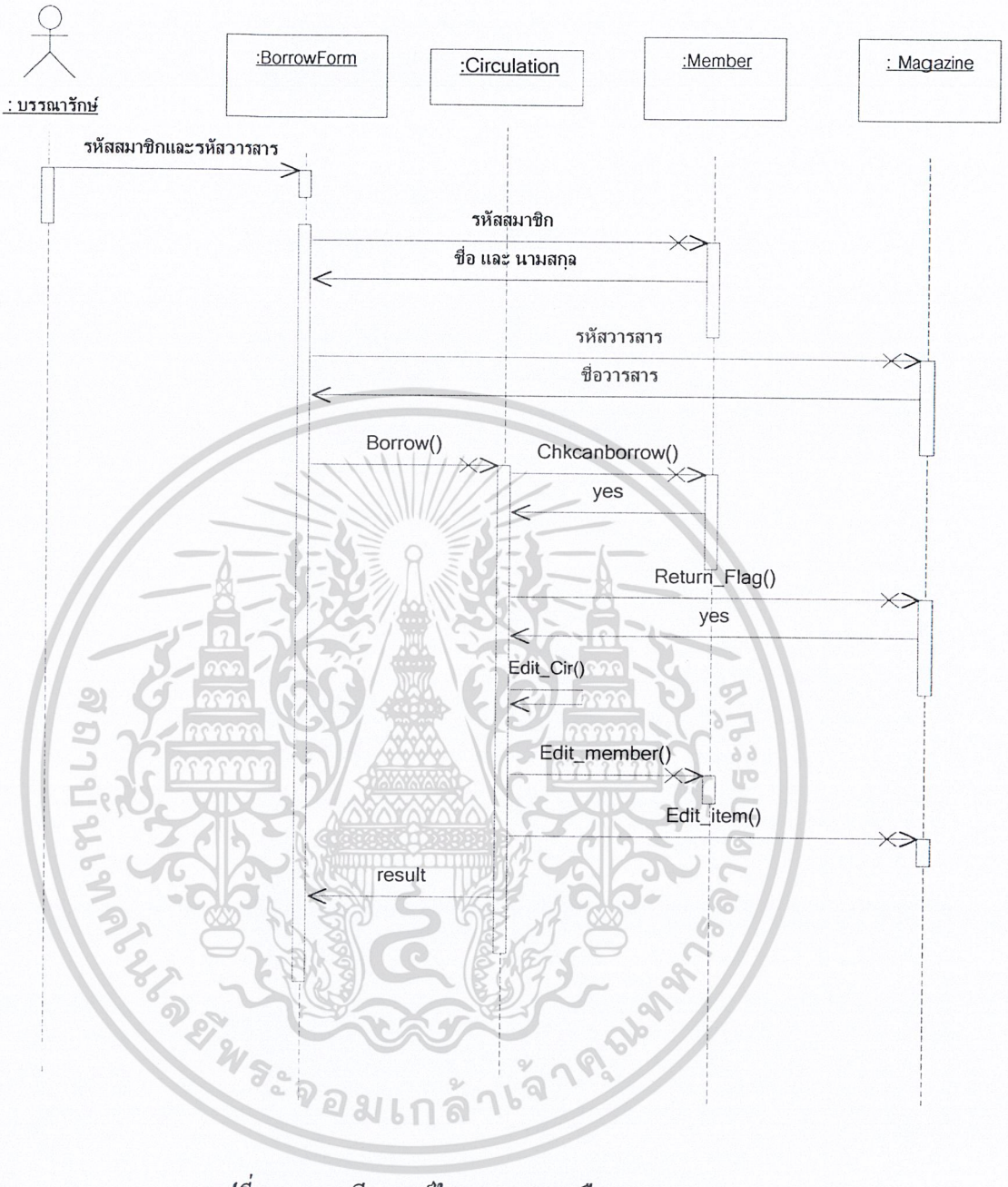
ซีเควนท์ไดอะแกรมจะแสดงความสัมพันธ์ระหว่างออบเจ็กต์ในคลาสต่างๆ ว่ามีปฏิสัมพันธ์กันอย่างไร และมีลำดับการทำงานอย่างไร ซึ่งเราจะนำซีเควนท์ไดอะแกรมมาอธิบายยูซเคสแต่ละยูซเคสในยูซเคสไดอะแกรม ซึ่งจะทำให้เราสามารถเข้าใจกระบวนการทำงานข้างในของยูซเคสแต่ยูซเคสได้ ซึ่งซีเควนท์ไดอะแกรมของระบบห้องสมุดสามารถแสดงได้ดังนี้



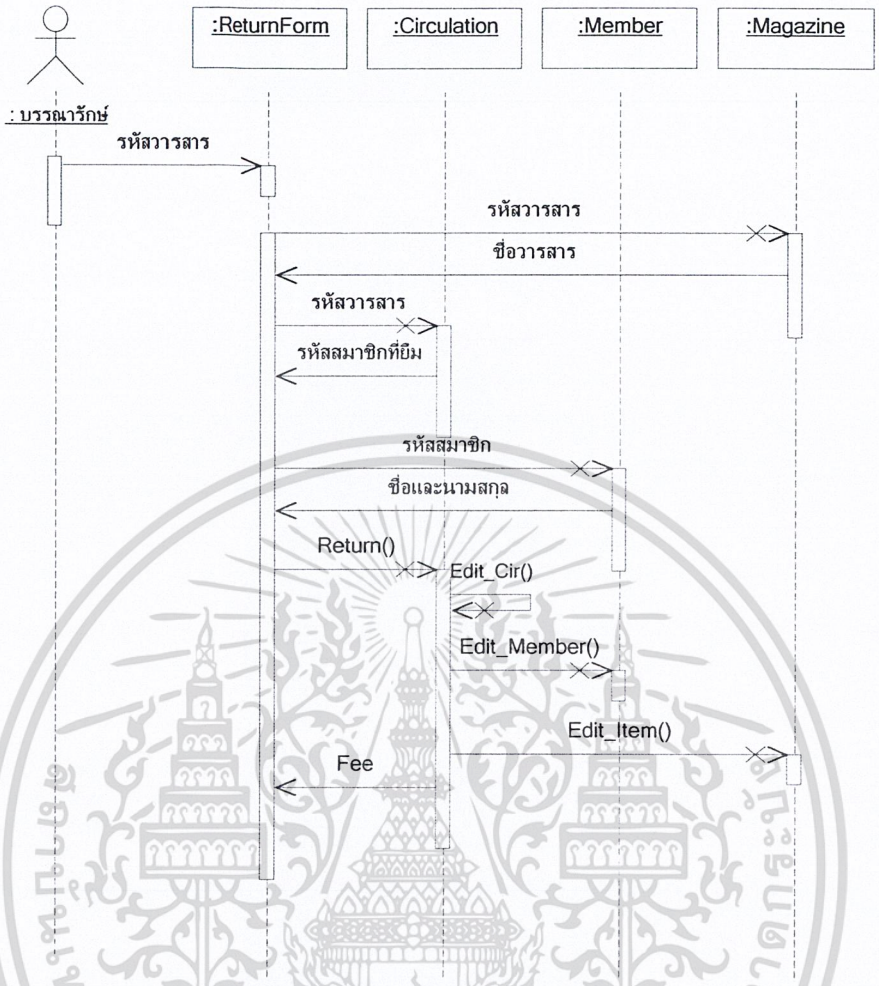
รูปที่ 8-7 แสดงซีเควนท์ไดอะแกรมงานยืมหนังสือ



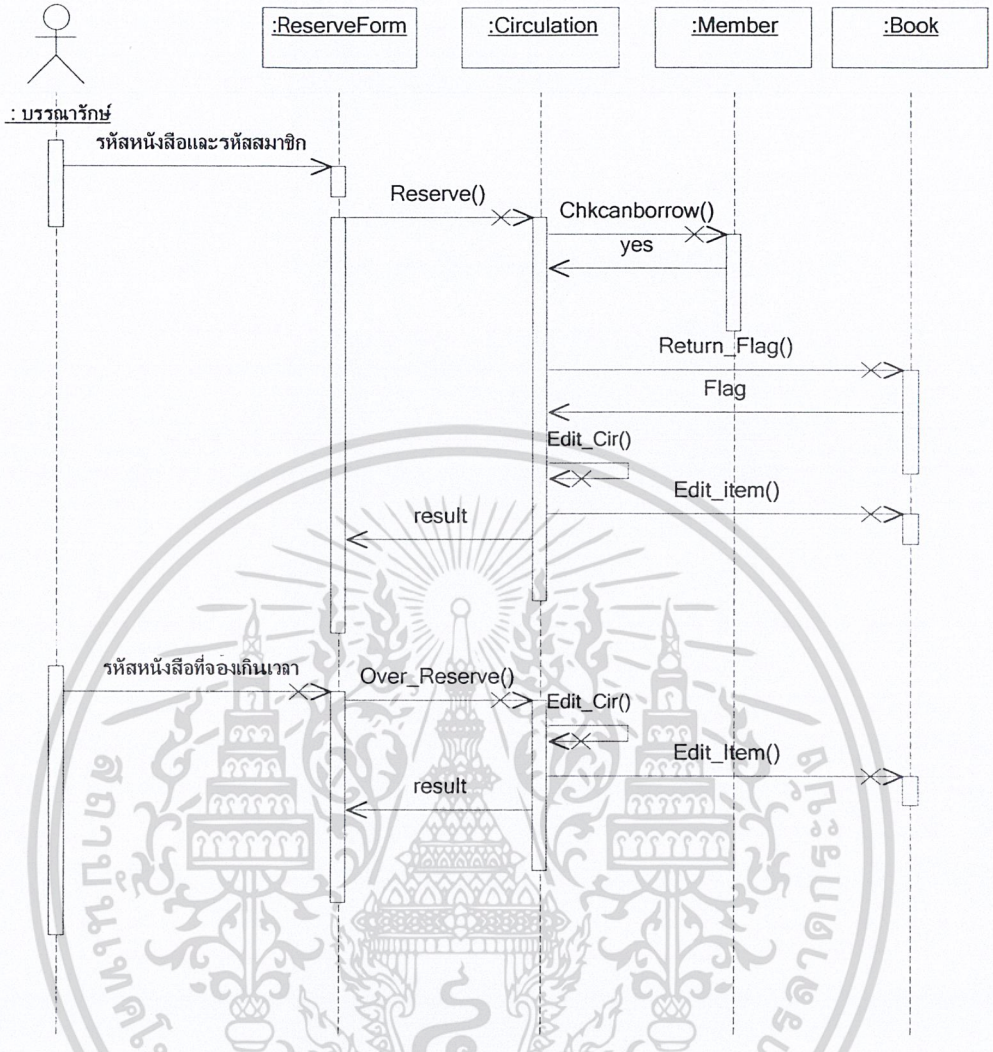
รูปที่ 8-8 แสดงซีเควนท์ไดอะแกรมงานคืนหนังสือ



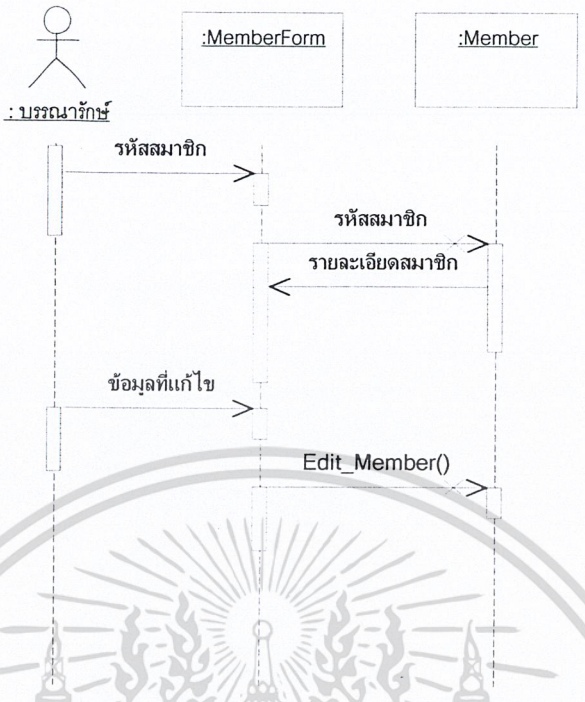
รูปที่ 8-9 แสดงซีเควนท์ไดอะแกรมงานยืมวารสาร



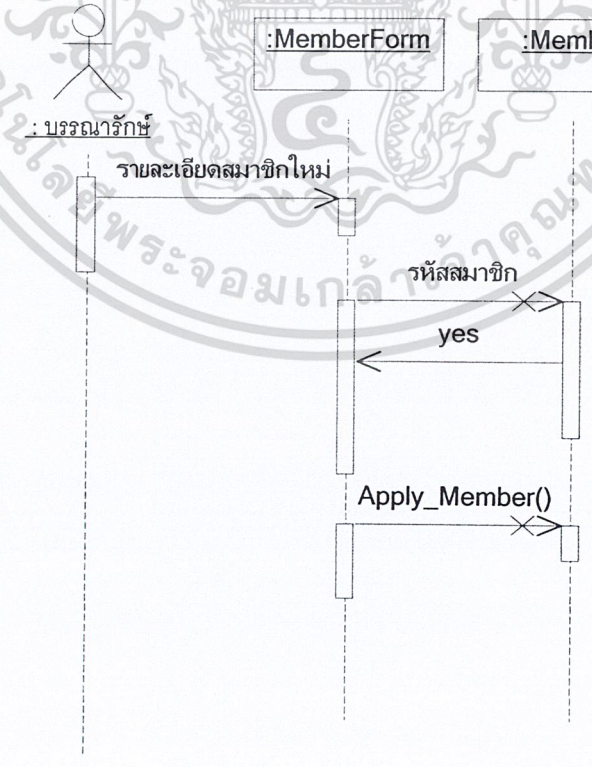
รูปที่ 8-10 แสดงซีเควนท์ไดอะแกรมงานคืนวารสาร



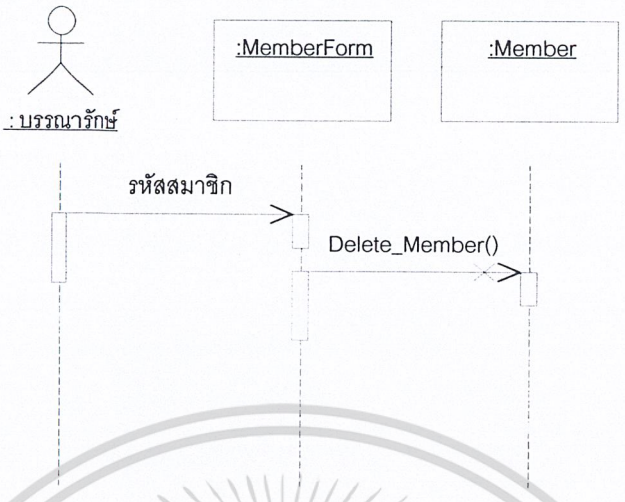
รูปที่ 8-11 แสดงซีเควนท์ไดอะแกรมงานจองหนังสือ



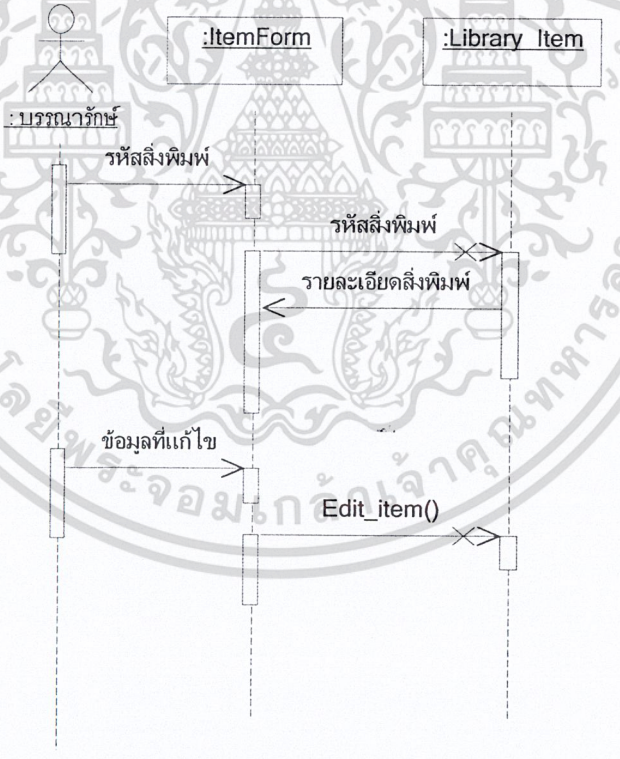
รูปที่ 8-12 แสดงซีเควนท์ไดอะแกรมงานแก้ไขข้อมูลสมาชิก



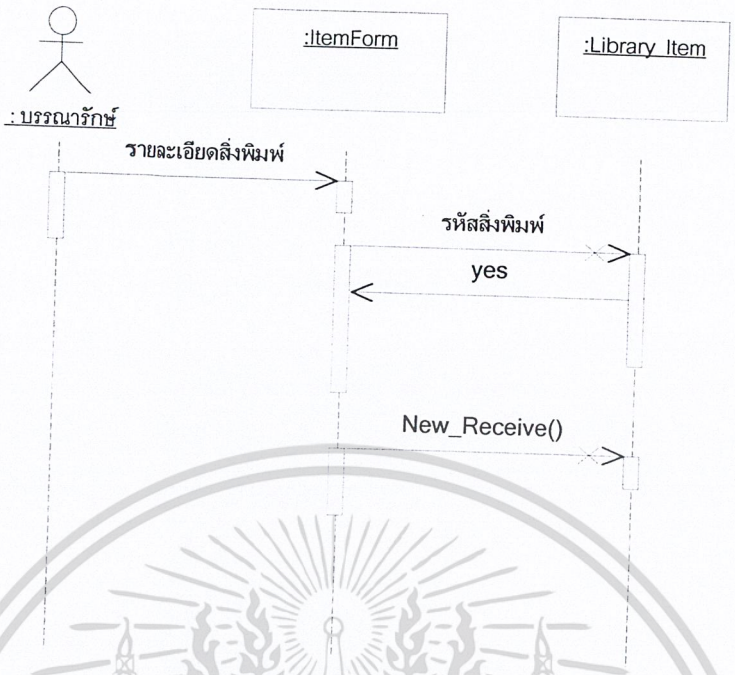
รูปที่ 8-13 แสดงซีเควนไดอะแกรมงานเพิ่มข้อมูลสมาชิก



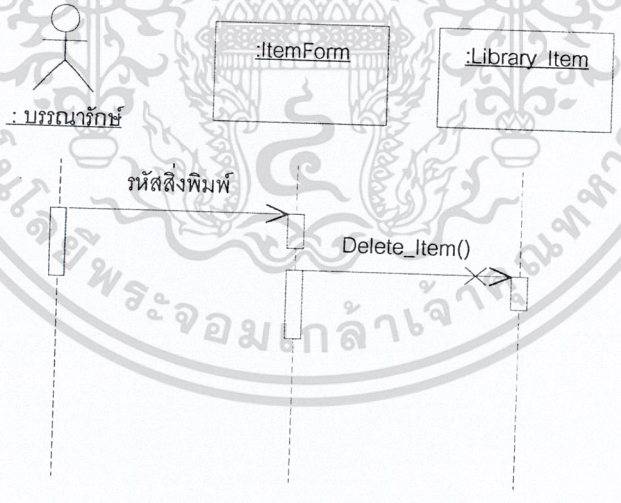
รูปที่ 8-14 แสดงซีเควนท์ไดอะแกรมงานลบข้อมูลสมาชิก



รูปที่ 8-15 แสดงซีเควนท์ไดอะแกรมงานแก้ไขข้อมูลหนังสือหรือวารสาร



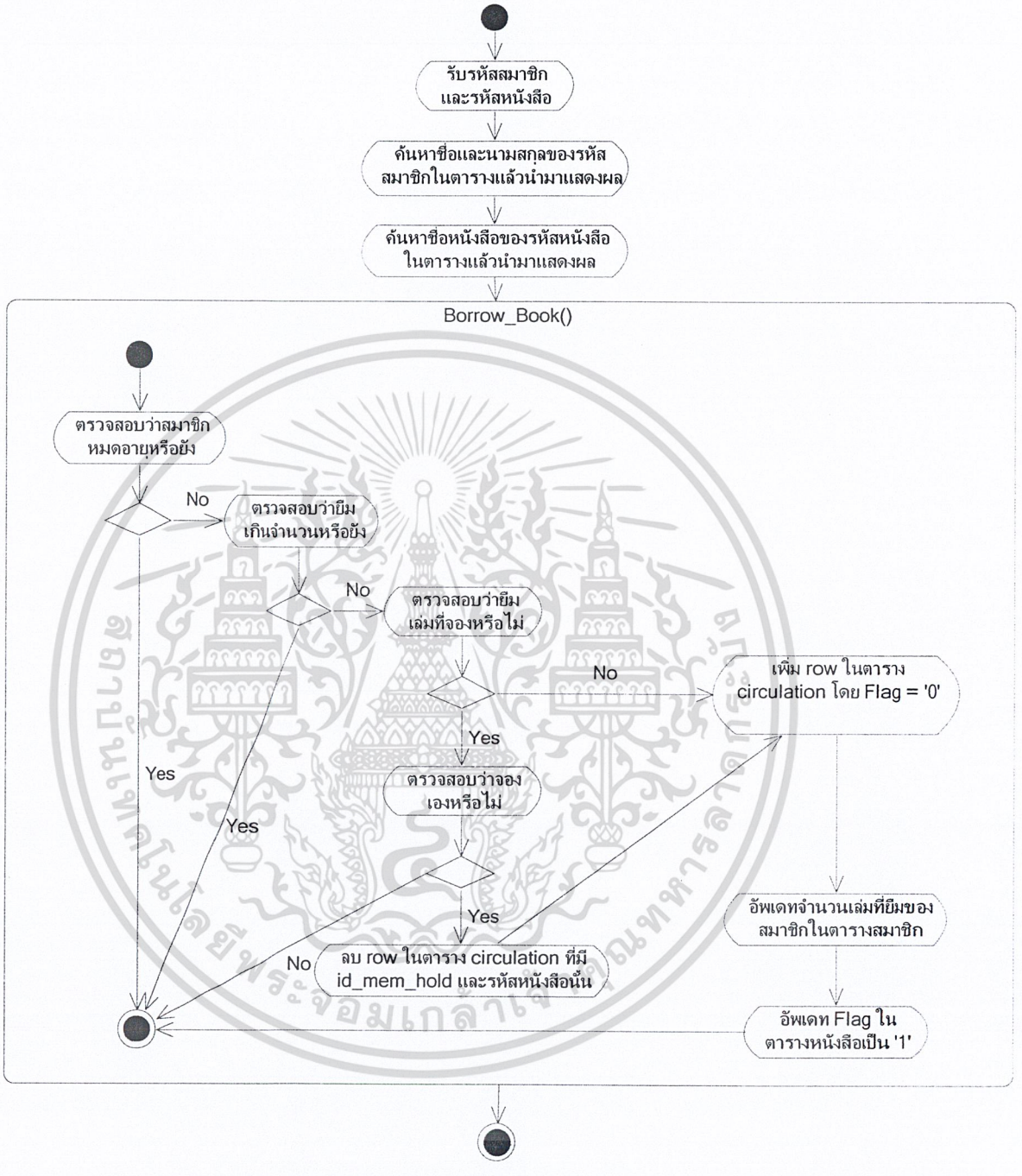
รูปที่ 8-16 แสดงซีเควนท์ไดอะแกรมงานเพิ่มข้อมูลหนังสือหรือวารสาร



รูปที่ 8-17 แสดงซีเควนท์ไดอะแกรมงานลบข้อมูลหนังสือหรือวารสาร

8.4 การสร้างแอ็กทิวิตี้ไดอะแกรม

หลังจากที่เราได้ซีเควนท์ไดอะแกรมแล้ว เราก็จะทราบถึงความสัมพันธ์ของออบเจ็กต์ในคลาสต่างๆ รวมถึงลำดับการทำงานด้วย แต่ว่าในซีเควนท์ไดอะแกรมก็ยังไม่สามารถอธิบายถึงรายละเอียดการทำงานได้อย่างละเอียดพอ ดังนั้นแอ็กทิวิตี้ไดอะแกรมจึงเข้ามาช่วยตรงส่วนนี้ โดยที่แอ็กทิวิตี้ไดอะแกรมจะมาอธิบายการทำงานของยูซเคส ซึ่งคล้ายกับซีเควนท์แต่จะลงในรายละเอียดลึกกว่า และไม่ได้แสดงถึงความสัมพันธ์ระหว่างออบเจ็กต์ด้วย ซึ่งแอ็กทิวิตี้ไดอะแกรมของระบบห้องสมุดสามารถแสดงได้ดังนี้

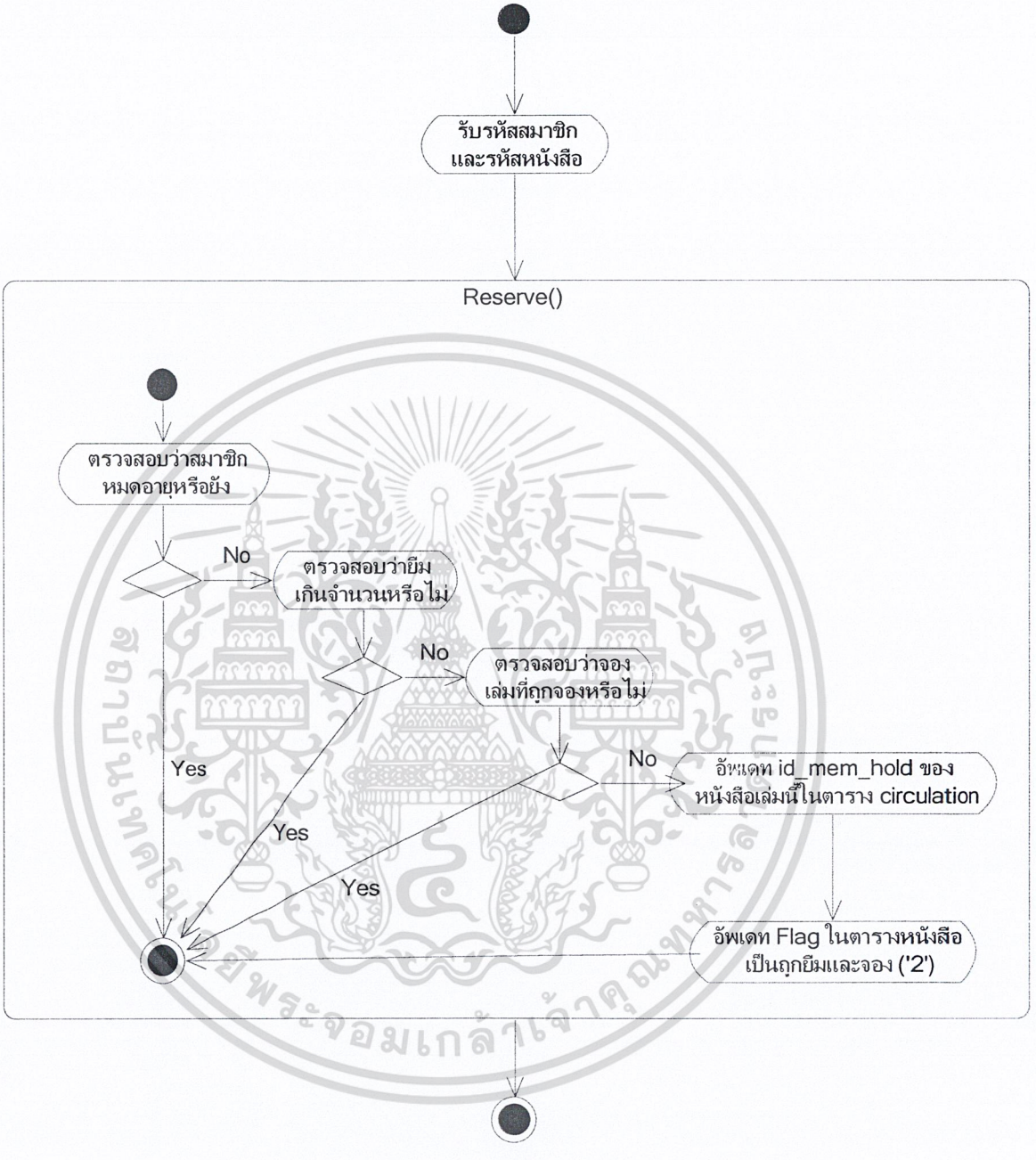


รูปที่ 8-18 แสดงแอ็กทิวิตี้ไดอะแกรมงานยืมหนังสือ

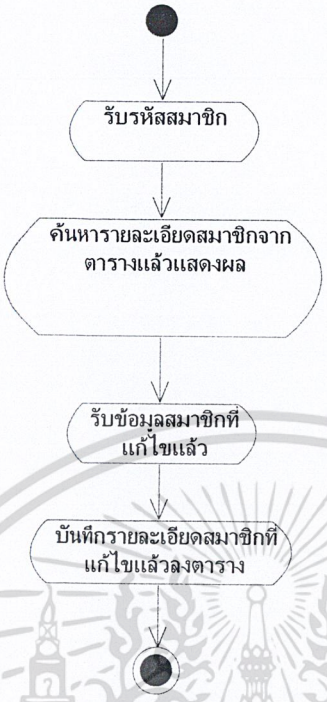
รูปที่ 8-19 แสดงแอ็กทิวิตี้ไดอะแกรมงานคืนหนังสือ

รูปที่ 8-20 แสดงแอ็กทิวิตี้ไดอะแกรมงานยืมวารสาร

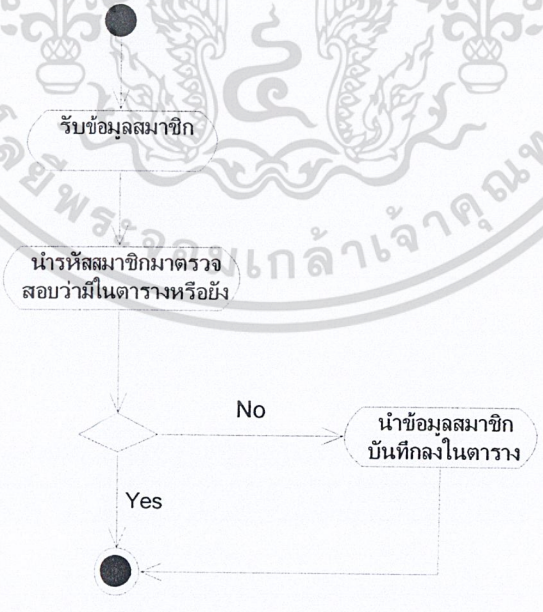
รูปที่ 8-21 แสดงแอ็กทิวิตี้ไดอะแกรมงานคืนวารสาร



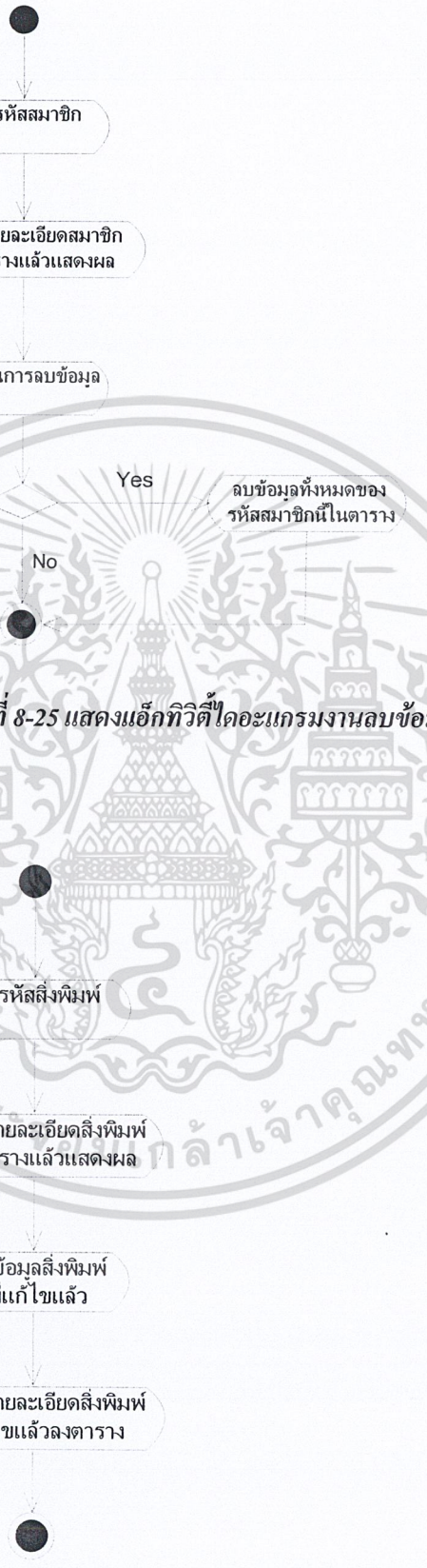
รูปที่ 8-22 แสดงแอ็กทิวิตี้ไดอะแกรมงานจองหนังสือ



รูปที่ 8-23 แสดงแอ็กทิวิตี้ไดอะแกรมงานแก้ไขข้อมูลสมาชิก

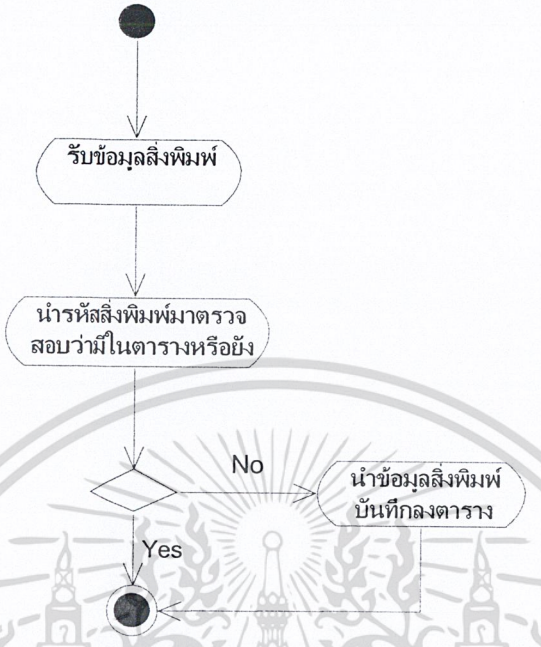


รูปที่ 8-24 แสดงแอ็กทิวิตี้ไดอะแกรมงานเพิ่มข้อมูลสมาชิก

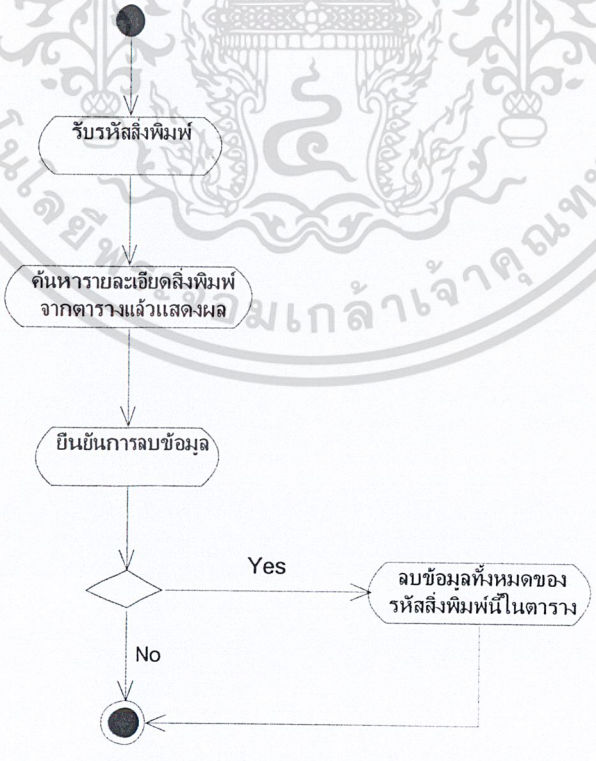


รูปที่ 8-25 แสดงแอ็กทิวิตี้ไดอะแกรมงานลบข้อมูลสมาชิก

รูปที่ 8-26 แสดงแอ็กทิวิตี้ไดอะแกรมงานแก้ไขข้อมูลหนังสือหรือวารสาร



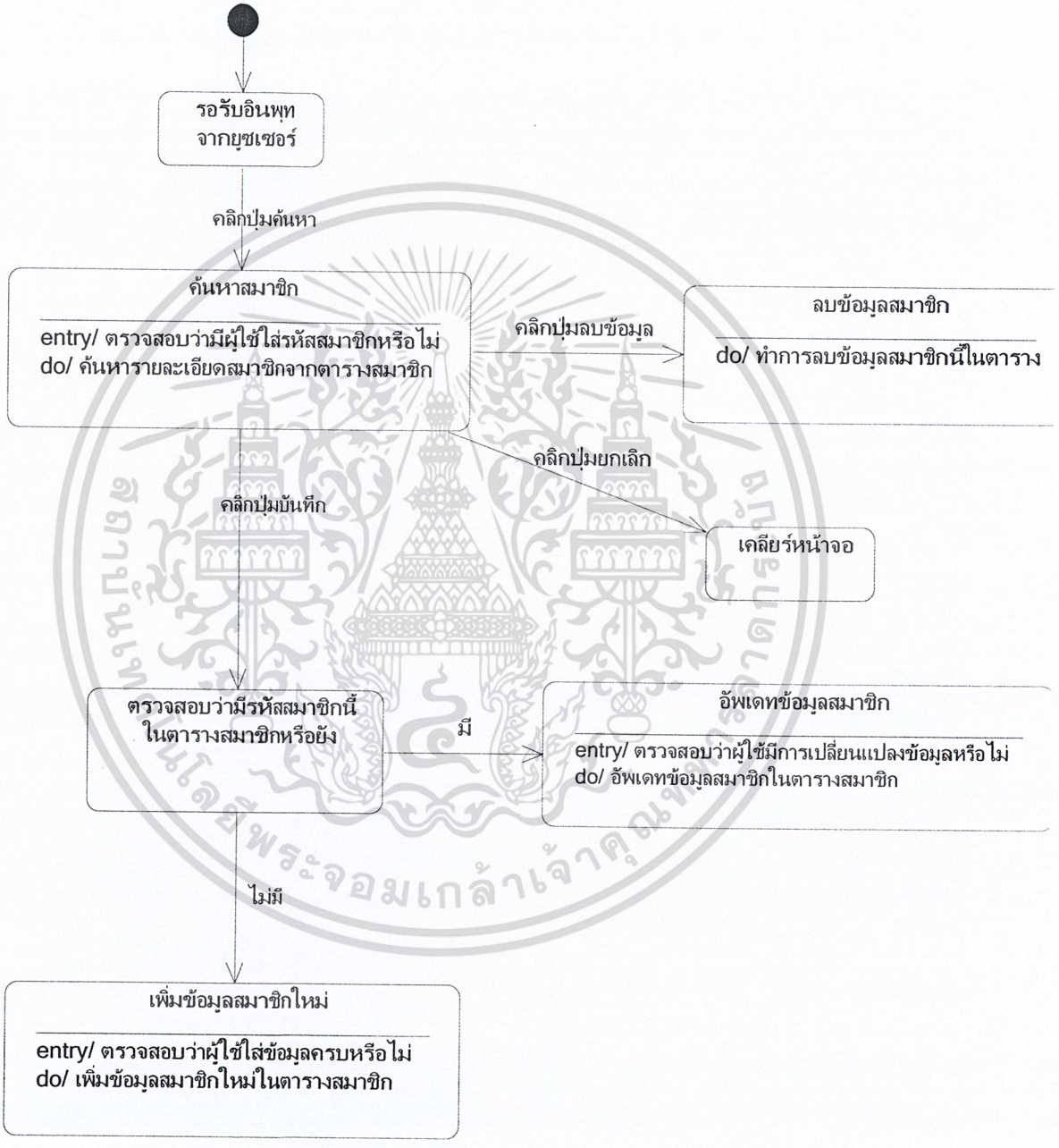
รูปที่ 8-27 แสดงแอ็กทิวิตี้ไดอะแกรมงานเพิ่มข้อมูลหนังสือหรือวารสาร



รูปที่ 8-28 แสดงแอ็กทิวิตี้ไดอะแกรมงานลบข้อมูลหนังสือหรือวารสาร

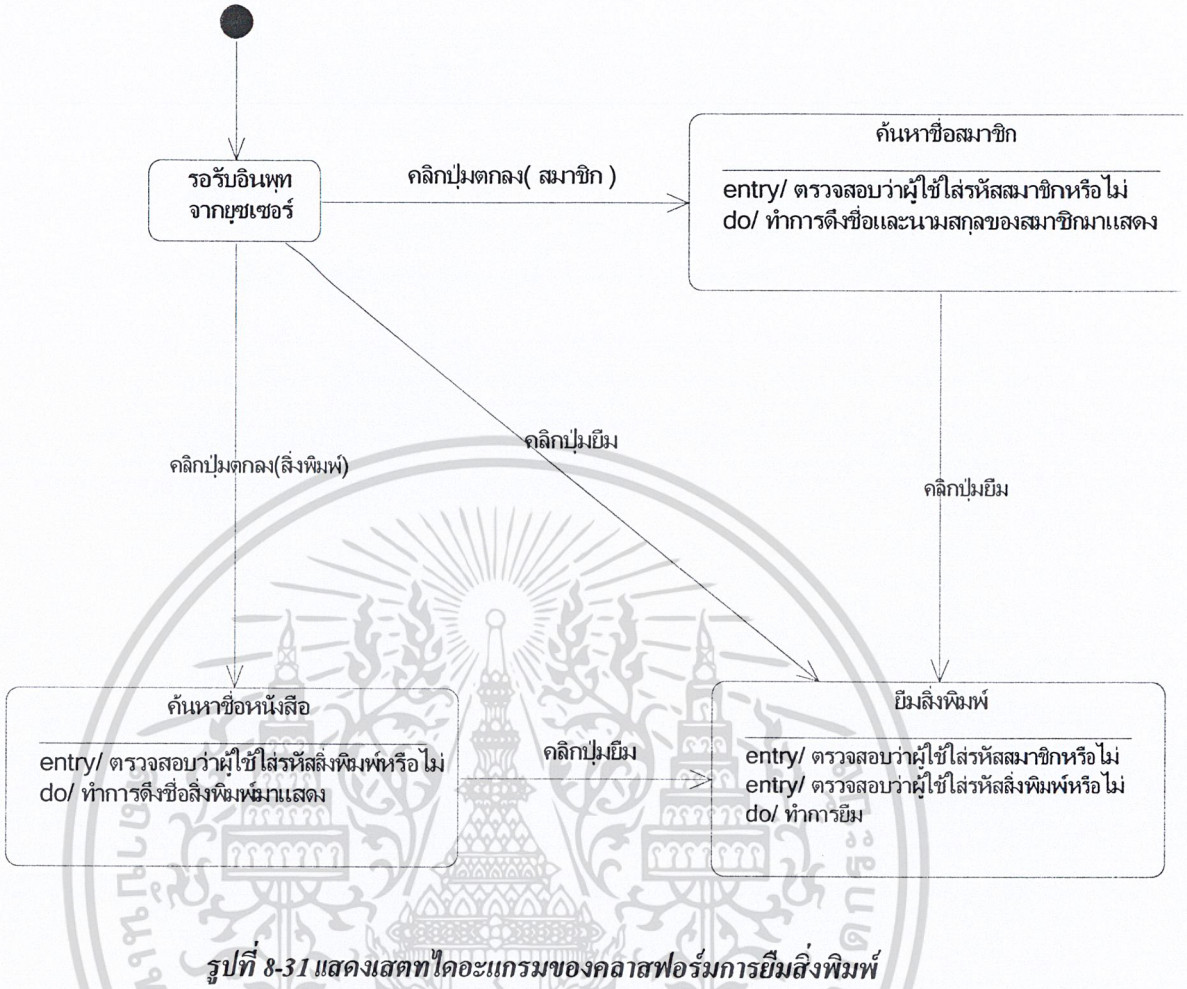
8.5 การสร้างสเตทไดอะแกรม

สร้างสเตทไดอะแกรมเพื่อทำการอธิบายพฤติกรรมของออบเจ็กต์ของคลาสต่างๆ ว่ามีการเปลี่ยนแปลงสเตท อย่างไรบ้างเมื่อเกิดเหตุการณ์ขึ้น รวมทั้งแสดงการทำงานที่อยู่ภายในแต่ละสเตท ซึ่ง สเตทไดอะแกรมของระบบห้องสมุด มีดังต่อไปนี้

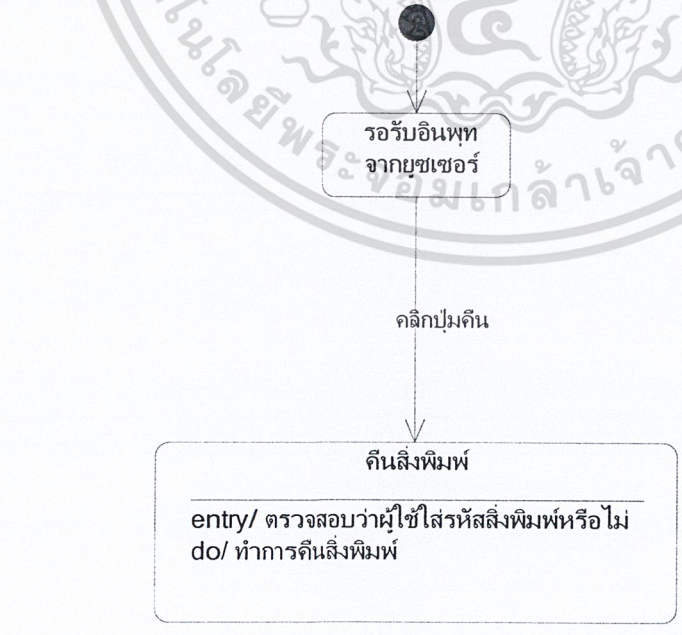


รูปที่ 8-29 แสดงเสตทไดอะแกรมของคลาสฟอร์มสมาชิก

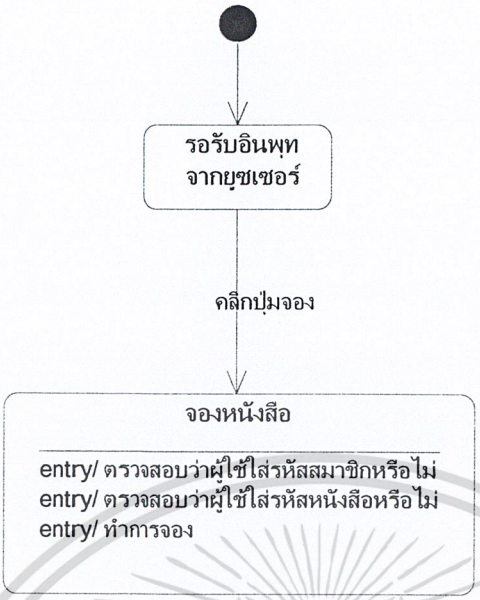
รูปที่ 8-30 แสดงเสตทไดอะแกรมของคลาสฟอร์มสิ่งพิมพ์



รูปที่ 8-31 แสดงเสตทไดอะแกรมของคลาสฟอร์มการยืมสิ่งพิมพ์



รูปที่ 8-32 แสดงสเตทไดอะแกรมของคลาสฟอร์มการคืนสิ่งพิมพ์



รูปที่ 8-33 แสดงสเตทไดอะแกรมของคลาสฟอร์มการจองหนังสือ

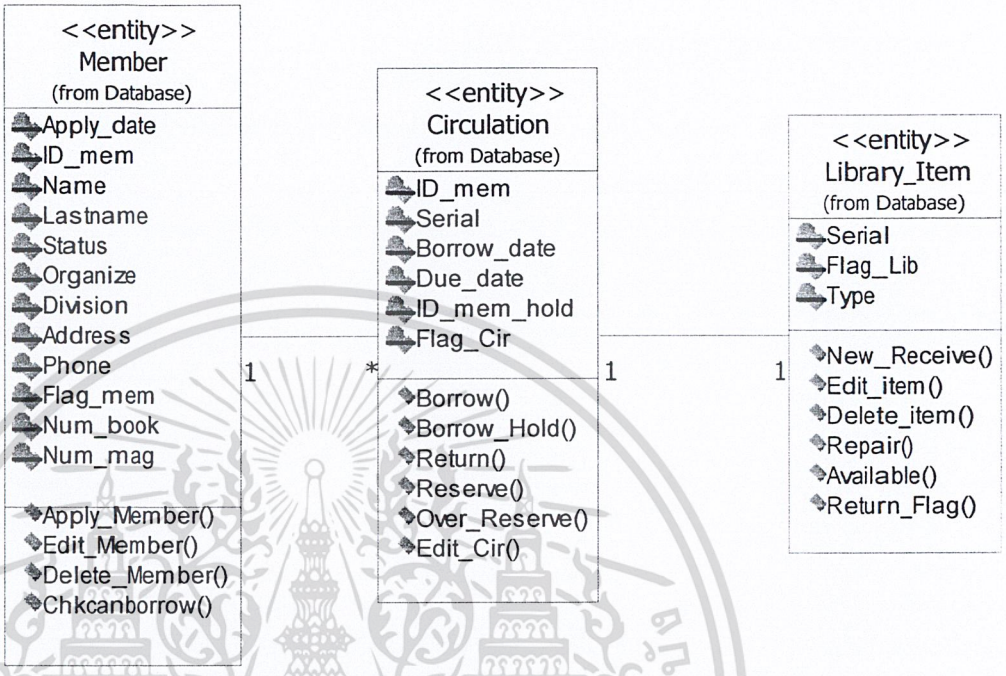
8.6 การนำไดอะแกรมไปพัฒนาเป็นโปรแกรมแอพพลิเคชัน

ในการสร้างโปรแกรมแอพพลิเคชันนั้นอาจเป็นไปได้ว่าผู้ที่ทำการออกแบบ กับผู้ที่ทำการเขียนโปรแกรมเป็นคนละคนกัน ดังนั้นเมื่อผู้เขียนโปรแกรมจะนำไดอะแกรมที่ออกแบบโดยโมเดล UML ไปใช้จะต้องมีหลักในการมองไดอะแกรม จึงจะสามารถนำไดอะแกรมที่ออกแบบมาอย่างดีแล้วไปเขียนโปรแกรมได้อย่างมีประสิทธิภาพได้ อนึ่งตัวโมเดล UML นั้นเป็นภาษาซึ่งไม่อาจสามารถที่จะทำให้ความเข้าใจเป็นมาตรฐานได้ ซึ่งการมองไดอะแกรมซึ่งถูกออกแบบโดยใช้โมเดล UML นั้นจะถูกต้องหรือดีแค่ไหน ขึ้นอยู่ผู้ที่ทำการออกแบบและผู้ที่นำไดอะแกรมไปใช้ด้วย

หลังจากที่เราได้ทำการศึกษาและค้นคว้าโมเดล UML มาได้ช่วงระยะเวลาหนึ่ง เราจึงนำเสนอเป็นวิธีที่จะสามารถมองไดอะแกรมต่างๆ ให้สามารถนำไปเขียนโปรแกรมได้ แต่อาจจะไม่ใช่วิธีที่ถูกต้องหรือดีที่สุด ซึ่งมีหลักการดังต่อไปนี้

- เราสามารถมองภาพรวมของระบบทั้งหมด โดยจะรู้ได้ว่าในระบบของเราทำงานอะไรบ้าง โดยดูได้จากยูซเคสไดอะแกรม ซึ่งงานต่างๆ เหล่านี้เราจะต้องนำไปสร้างแอพพลิเคชันเพื่อรองรับการทำงานนั้นๆ ในที่นี้จะขอยกตัวอย่างหนึ่งในงานยืมหนังสือของระบบห้องสมุด ซึ่งแสดงไว้ในรูปที่ 8-1
- จากนั้นเราจะไปดูที่คลาสไดอะแกรม เพื่อดูว่าโครงสร้างของคลาสเป็นอย่างไรบ้าง มีคลาสไหนที่เป็นคลาสของฐานข้อมูล หรือคลาสไหนเป็นคลาสแอพพลิเคชันบ้าง โดยเราจะต้องนำคลาสของฐานข้อมูลไปสร้างฐานข้อมูลทั้งหมด และคลาสของแอพพลิเคชันไปเขียนโปรแกรม ซึ่งเราจะสามารถรู้ได้ว่าในแต่ละคลาสจะมีแอทริบิวต์และเมธอดการทำงานอะไร

บ้าง ซึ่งสำหรับการเขียนการทำงานทำงานของเมธอดแต่ละเมธอดนั้นจะใช้ไดอะแกรมอื่นๆ ที่จะกล่าวต่อไป จะยกตัวอย่างคลาสที่เกี่ยวข้องกับงานยืมหนังสือดังต่อไปนี้



รูปที่ 8-34 แสดงคลาสฝั่งฐานข้อมูลที่เกี่ยวข้องกับงานยืมสิ่งพิมพ์

จะเห็นได้ว่างานยืมหนังสือจะเกี่ยวข้องกับคลาสฝั่งฐานข้อมูลอยู่ 3 คลาสด้วยกัน ได้แก่ คลาสสมาชิก (Member) คลาสสิ่งพิมพ์ (Library_Item) และคลาส Circulation โดยที่เราจะเห็นแอทริบิวต์เมธอดของคลาสแต่ละคลาส แต่เรายังไม่สามารถรู้ได้ว่างานยืมหนังสือนี้ เรียกใช้เมธอดไหนและใช้แอทริบิวต์ไหนบ้าง ซึ่งเราจะได้รู้ได้จากซีเควนท์ไดอะแกรมและแอ็กทิวิตี้ไดอะแกรม จากนั้นเราจะมาพิจารณาที่คลาสของแอพพลิเคชัน ซึ่งก็คือ คลาสของฟอร์มการยืม โดยเราจะรู้ได้ว่าฟอร์มการยืมนี้ประกอบด้วยอะไรบ้างและมีเมธอดการทำงานอะไรบ้างแต่กระบวนการทำงานของคลาสนี้ยังไม่สามารถบอกได้ ซึ่งการทำงานของคลาสนี้ดูได้จากสเตทไดอะแกรม

- จากนั้นเราจะมาดูซีเควนท์ไดอะแกรมเพื่อที่จะได้รู้ว่างานแต่ละงานในยูซเคสไดอะแกรมนั้นมีการติดต่อปฏิสัมพันธ์กันระหว่างคลาสอย่างไร ซึ่งในซีเควนท์ไดอะแกรมนี้จะบอกตรงจุดนี้และบอกลำดับการทำงานด้วย ยกตัวอย่างซีเควนท์ไดอะแกรมของงานยืมหนังสือที่แสดงในรูปที่ 8-7 จะเห็นได้ว่างานยืมหนังสือมีการติดต่อระหว่างคลาสด้วยกัน 4 คลาสคือ คลาสฟอร์มการยืม คลาสCirculation คลาสสมาชิก และคลาสหนังสือ ซึ่งลำดับในการทำงานของงานยืมหนังสือ ก็แสดงไว้ในซีเควนท์ไดอะแกรมแล้ว แต่บางฟังก์ชันการทำงานก็ยังไม่ได้อธิบายละเอียดในซีเควนท์ไดอะแกรมนี้ เนื่องจากว่าจุดประสงค์หลักของซีเควนท์

ไดอะแกรมเพื่อแสดงให้ดูว่างานนี้มีการติดต่อกับคลาสอะไรบ้าง แต่สำหรับรายละเอียดฟังก์ชันการทำงานจะไปแสดงในแอ็กทิวิตี้ไดอะแกรม

- เมื่อเราต้องการที่จะสร้างเมธอดเพื่อรองรับยูซเคสหนึ่งๆ เราจะไปดูในแอ็กทิวิตี้ไดอะแกรม ซึ่งจะบอกรายละเอียดการทำงานอย่างละเอียด ซึ่งสามารถนำไปเขียนเป็นเมธอดการทำงานได้ ตัวอย่างแอ็กทิวิตี้ไดอะแกรมงานยืมหนังสือ ดังรูปที่ 8-18 จะเห็นได้ว่าแอ็กทิวิตี้ไดอะแกรมอธิบายการทำงานของงานยืมหนังสือไว้อย่างละเอียด เริ่มตั้งรับอินพุทจากผู้ใช้ จนกระทั่งเมธอดการยืมหนังสือ ซึ่งพอที่จะสามารถนำไปเขียนโปรแกรมได้

- ในการสร้างเมธอดหรือฟังก์ชันการทำงานของยูซเคสนั้น เราอาจต้องใช้ไดอะแกรมมากกว่า 1 ไดอะแกรมประกอบกันใช้งาน เพราะว่าไม่มีไดอะแกรมใดที่อธิบายรายละเอียดได้ครบทุกอย่าง อย่างเช่นเราจะสร้างเมธอดการยืมของงานยืมหนังสือ เราต้องใช้ คลาสไดอะแกรมประกอบกับซีเควนท์ไดอะแกรมประกอบกับแอ็กทิวิตี้ไดอะแกรม จึงจะสามารถมองไดอะแกรมงานยืมหนังสือได้อย่างเข้าใจ

- เมื่อเราสร้างเมธอดการยืมของงานยืมหนังสือได้เรียบร้อยแล้ว เราจะต้องมาสร้างฟอร์มที่จะต้องเรียกใช้งานเมธอดนี้ ซึ่งก็คือฟอร์มการยืมนั่นเอง ซึ่งเราจะใช้คลาสไดอะแกรมประกอบกับสเตทไดอะแกรม เพื่อนำมาสร้างฟอร์มนี้ โดยที่คลาสไดอะแกรมจะบอกถึงส่วนประกอบของฟอร์มว่ามีอะไรบ้าง ส่วนสเตทไดอะแกรมจะบอกถึงกระบวนการทำงานของฟอร์มนี้เมื่อมีอีเวนท์ต่างๆ เกิดขึ้น ตัวอย่างสเตทไดอะแกรมของฟอร์มการยืมดังรูปที่ 8-31 จะเห็นได้ว่าไดอะแกรมนี้จะแสดงให้เห็นว่าเมื่อมีอีเวนท์เกิดขึ้นในฟอร์ม ควรจะทำอะไร ซึ่งจะสามารถนำไปเขียนเป็นโปรแกรมได้

บทที่ 9
สรุปผลการวิจัยและข้อเสนอแนะ

ในโครงงานวิจัยนี้ได้ศึกษาค้นคว้าการออกแบบระบบงานด้วยโมเดล UML ซึ่งเป็นโมเดลสำหรับออกแบบระบบเชิงวัตถุและฐานข้อมูลเชิงสัมพันธ์ ซึ่งในโมเดล UML นี้ถือได้ว่าเป็นโมเดลที่มีประสิทธิภาพสูงที่ถูกนำมาใช้ในการออกแบบระบบเชิงวัตถุกันอย่างแพร่หลาย ซึ่งโมเดล UML เป็นภาษาที่นำรูปภาพและสัญลักษณ์มาใช้อธิบายการทำงานต่างๆ ซึ่งข้อดีของ UML มีดังต่อไปนี้

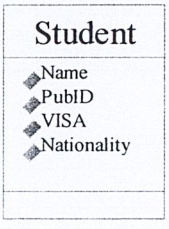
- UML เป็นโมเดลที่มีรูปแบบการใช้งานเป็นมาตรฐาน ดังนั้นจึงเป็นการง่ายต่อการนำไปอธิบายให้คนอื่นเข้าใจโดยที่มีรูปแบบการใช้โมเดลในลักษณะเดียวกัน
- UML ครอบคลุมทุกส่วนในกระบวนการออกแบบระบบ ตั้งแต่ขั้นตอนของการหาความต้องการของระบบ (requirement) การออกแบบระบบ (design) การนำไปใช้งานจริง (implementation) การติดตั้งระบบ (installation) ไปจนถึงขั้นตอนของการจัดทำเอกสาร (documentation)
- มีบริษัทชั้นนำและอุตสาหกรรมต่างๆ ให้การยอมรับและให้การสนับสนุน

แต่อย่างไรก็ตามถึงแม้ว่า UML จะเป็นภาษาที่มีมาตรฐาน แต่ก็มีรายละเอียดที่ซับซ้อนและยืดหยุ่นมากพอควร จนทำให้การออกแบบระบบหนึ่งๆ อาจเป็นไปได้หลายแนวทาง ซึ่งขึ้นอยู่กับการผู้ออกแบบ นอกจากนี้ยังอาจทำให้ผู้ที่ออกแบบ กับผู้ที่นำโมเดลไปใช้ มีความเข้าใจไม่ตรงกันสักทีเดียว จึงอาจกลายเป็นว่า UML ได้เพิ่มความยุ่งยากในการออกแบบให้กับระบบ ซึ่งปัญหาเหล่านี้อาจไม่เกิดถ้าเรามีความเข้าใจในโมเดล UML เป็นอย่างดี ซึ่งจะสามารถนำไปใช้ได้อย่างมีประสิทธิภาพสูงสุด

นอกจากนี้แล้วเมื่อได้ศึกษาและค้นคว้า รวมทั้งทำการทดลองโดยนำ UML มาออกแบบระบบห้องสมุดจึงได้พบปัญหาที่ UML ยังไม่สนับสนุน ดังต่อไปนี้

- ปัญหาความสัมพันธ์ระหว่างแอตทริบิวท์ในคลาสและระหว่างแอตทริบิวท์กับคลาส

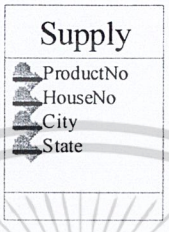
  ปัญหานี้เกิดจากผู้อิมพลิเมนต์ไม่ทราบความสัมพันธ์ระหว่างแอตทริบิวท์สองตัวหรือมากกว่านั้นว่ามีความสัมพันธ์ หรือมีกฎในการป้อนอินพุตอย่างไร ดังตัวอย่างต่อไปนี้



รูปที่ 9-1 แสดงตัวอย่างปัญหาความสัมพันธ์ระหว่างแอตทริบิวต์ในคลาส

จากตัวอย่าง ถ้ากำหนดว่านักศึกษาที่เป็นคนไทย ให้กรอกหมายเลขบัตรประชาชน ไม่เช่นนั้นให้กรอกหมายเลข VISA ซึ่งในโมเดลนี้ ไม่สามารถบอกได้ แนวทางแก้ไขสามารถทำได้โดยใช้ OONIAM

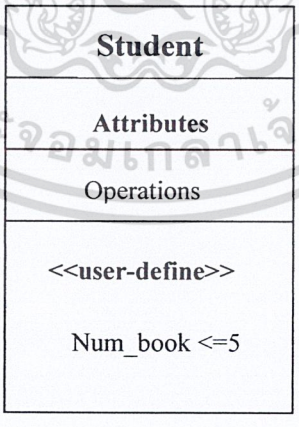
- คลาสไดอะแกรม (Class Diagram) ไม่ได้นำเสนอ Function Dependency (FD)
  ซึ่งอาจทำให้เกิดค่าของแอตทริบิวท์ที่ซ้ำซ้อนได้ ดังตัวอย่างต่อไปนี้



รูปที่ 9-2 แสดงตัวอย่างคลาสไดอะแกรมที่ไม่ได้เสนอ FD

จากตัวอย่างจะเห็นว่าค่าของ city จะ determine state เสมอ เนื่องจากเราไม่ทราบความสัมพันธ์ของแอตทริบิวท์ทั้งสองในคลาส แนวทางแก้ไขสามารถทำได้โดยการใช้ OONIAM

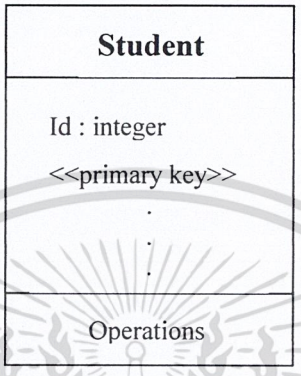
- การจัดการกฎข้อบังคับความถูกต้อง (Integrity Rules)
  ยังไม่มีจุดที่แน่นอนว่า จะใส่กฎข้อบังคับความถูกต้องไว้ตรงส่วนใดของ UML ยกตัวอย่าง เช่น นักศึกษาจะยืมหนังสือได้ไม่เกิน 5 เล่ม เราจะเก็บกฎนี้ไว้ตรงส่วนใดของ UML แนวทางการแก้ไขสามารถทำได้โดยอาจเก็บไว้ในแอ็กทิวิตี้ไดอะแกรมเมื่อมีการมายืมหนังสือก็จะทำการเช็คเงื่อไขนี้ก่อน หรือ อาจเก็บไว้ในคลาสไดอะแกรมในส่วนของ user-define list



รูปที่ 9-3 แสดงตัวอย่างการเก็บกฎข้อบังคับไว้ในคลาสไดอะแกรม

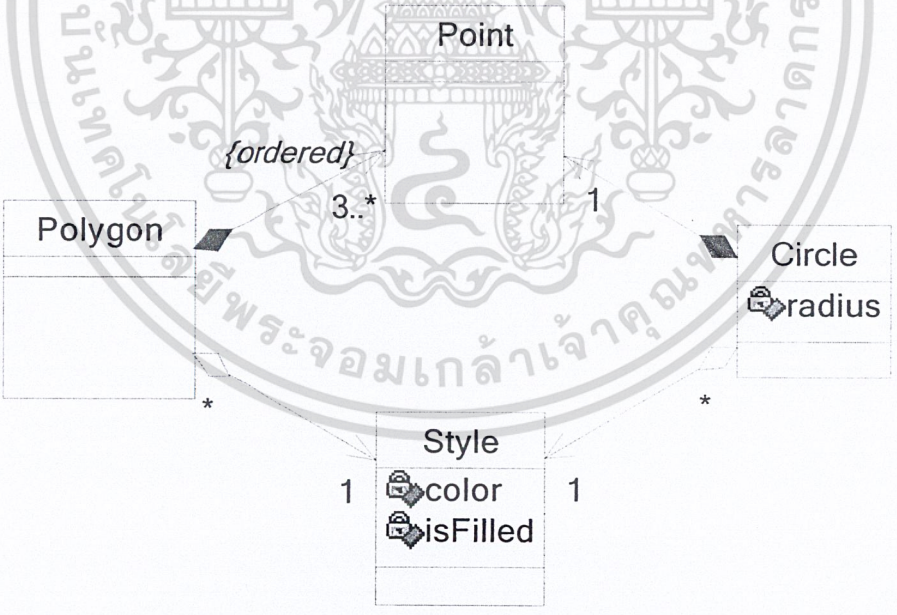
- ไพรมารี่คีย์และฟอร์เรนคีย์ (Primary key and Forieng key)

เนื่องจากการใช้โมเดล UML ซึ่งเป็นโมเดลเชิงวัตถุ ดังนั้นจึงไม่สนับสนุนในเรื่องของไพรมารี่คีย์และฟอร์เรนคีย์ เนื่องจากการออกแบบเชิงวัตถุได้ใช้ oid (object identifier) มาแทน แต่ในการออกแบบระบบเชิงวัตถุสัมพันธ์ ซึ่งยังสามารถที่จะใช้ไพรมารี่คีย์และฟอร์เรนคีย์ได้อยู่ ซึ่งใน UML ไม่มีที่ให้เขียนไว้ แนวทางการแก้ไข อาจใช้ สเตอริโอไทป์กำกับแอททริบิวต์ที่เป็นไพรมารี่คีย์หรือฟอร์เรนคีย์นั้น



รูปที่ 9-4 แสดงตัวอย่างการกำหนดไพรมารี่คีย์หรือฟอร์เรนคีย์

- ปัญหาการแสดง Embedded Object



รูปที่ 9-5 แสดงตัวอย่างการใช้ Aggregation ของ UML

จากรูปที่ 9-5 เป็นการใช้ Aggregation ในคลาสไดอะแกรมของ UML นั้นมีจุดประสงค์เพื่อแสดงว่าคลาส 2 คลาส มีความสำคัญและจำเป็นต่อกันมากแค่ไหน แต่ไม่สามารถบอกได้ชัดเจนว่าควรจะอิมพลีเมนต์คลาสไหนให้เป็น Embedded Class หรือไม่ แนวทางการแก้ไขคือใช้ไดอะแกรมของ OONIAM

นอกจากปัญหาต่างๆ ที่ได้จากการศึกษาและค้นคว้า UML แล้ว เมื่อเราได้ทำการนำระบบห้องสมุดที่ออกแบบด้วย UML ไปพัฒนาเป็นแอพพลิเคชันโดยใช้ระบบฐานข้อมูลเชิงวัตถุสัมพันธ์ Informix เราได้พบปัญหาที่เกิดขึ้นจากการใช้ระบบฐานข้อมูลดังกล่าว ซึ่งไม่เป็นไปตามทฤษฎีที่ศึกษาไว้ดังต่อไปนี้

- ปัญหาการสืบทอดคุณสมบัติฟังก์ชันการทำงาน (Function Inheritance)

  เนื่องจาก Informix เป็นระบบฐานข้อมูลเชิงวัตถุสัมพันธ์ ซึ่งได้รวมข้อดีของระบบฐานข้อมูลเชิงวัตถุ กับระบบฐานข้อมูลเชิงสัมพันธ์เข้าไว้ด้วยกัน โดยที่ข้อดีของระบบฐานข้อมูลเชิงวัตถุที่ได้นำมารวมไว้ก็คือ คุณสมบัติเชิงวัตถุ เช่น คุณสมบัติการสืบทอดคุณสมบัติ และคุณสมบัติพอร์ลิมอฟีซึม ดังนั้นตามทฤษฎีการทำคุณสมบัติเชิงวัตถุมาใช้จึงไม่น่าเกิดปัญหาใดๆ แต่ในความเป็นจริง ระบบฐานข้อมูล Informix นั้นรองรับการสืบทอดคุณสมบัติเฉพาะสกีมา (Schema) เท่านั้น เช่น การสืบทอดโรว์ไทป์ (Row Type) และการสืบทอดตาราง (Table) ไม่ได้รองรับการสืบทอดฟังก์ชันการทำงาน ซึ่งได้เรียกว่า “รูทีน” โดยที่รูทีนในระบบฐานข้อมูล Informix ไม่ได้ทำการซ่อนเร้นข้อมูลไว้ (Encapsulation) ดังนั้นไม่ว่าคลาสใดๆ ก็สามารถที่จะเข้าถึงรูทีนนั้นได้ ทำให้เราต้องสร้างรูทีนเพื่อรองรับคลาสทุกคลาส เช่น จากตัวอย่างระบบห้องสมุดที่เราได้ทำการศึกษามา มีคลาสนักศึกษาและคลาสข้าราชการที่สืบทอดมาจากคลาสสมาชิก และมีรูทีนที่ทำหน้าที่ลบข้อมูลสมาชิกที่ทั้งคลาสนักศึกษาและคลาสข้าราชการต้องใช้ ซึ่งในกรณีนี้ไม่มีสิ่งที่จะแยกแยะได้ว่าจะให้รูทีนนี้เป็นของคลาสใด ดังนั้นจึงต้องทำการสร้างรูทีนที่มีหน้าที่เหมือนกันถึง 2 รูทีน ซึ่งไม่เป็นไปตามจุดประสงค์ของคุณสมบัติเชิงวัตถุ จากที่ได้กล่าวมาข้างต้นทำให้ระบบฐานข้อมูล Informix ไม่รองรับการสืบทอดคุณสมบัติฟังก์ชันการทำงาน

- ปัญหาการส่งค่าอาร์กิวเมนต์เป็นโรว์ไทป์

  จากปัญหาข้างต้นที่ได้กล่าวมา Informix ได้นำเสนอทางออกโดยที่สามารถให้ส่งค่าอาร์กิวเมนต์หนึ่งเป็นโรว์ไทป์ได้ เรียกวิธีการนี้ว่าการแมนนิพูเลทโรว์ไทป์ (Manipulate Rowtype) โดยถ้าใช้วิธีนี้แล้วจากตัวอย่างรูทีนลบข้อมูลสมาชิกดังกล่าวข้างต้น จะไม่เกิดปัญหา เนื่องจากเราใช้โรว์ไทป์เป็นตัวแยกแยะว่ารูทีนนี้เป็นของคลาสใด อนึ่ง ในระบบฐานข้อมูล Informix ได้รองรับการโอเวอร์ไรด์รูทีน (Override Routine) อยู่แล้ว

  แต่เมื่อได้ทำการทดลองแล้ว ปรากฏว่าระบบฐานข้อมูล Informix รองรับการส่งอาร์กิวเมนต์เป็นโรว์ไทป์ได้จริง แต่เกิดปัญหาที่การนำไปใช้ เนื่องจาก Informix รองรับแต่การเรียกใช้ รูทีนที่แมนนิพูเลทโรว์ไทป์ โดยใช้คีย์เวิร์ด “Select” เท่านั้น ไม่รองรับการเรียกใช้รูทีนที่แมนนิพูเลทโรว์ไทป์ โดยใช้คียเวิร์ด “Execute” ดังนั้นภายในรูทีนที่แมนนิพูเลทโรว์ไทป์จะไม่สามารถทำการ เพิ่มข้อมูล (Insert) แก้ไขข้อมูล (Update) หรือ ลบข้อมูล (Delete) ได้เลย เพราะจะทำให้เกิดผลกระทบกับข้อมูลที่จะทำการคิวรี่ ดูได้จากตัวอย่างดังต่อไปนี้

  สร้างรูทีนลบข้อมูลสมาชิกได้ดังนี้

```
Create Procedure Delete_Member(student STUDENT_T, id integer)
```

```
Delete From student Where id_mem = id;
End Procedure;
```

โดยที่ STUDENT_T เป็นโรว์ไทป์ของคลาส Student

เรียกใช้รูทีนได้ดังนี้

```
Execute Procedure Delete_Member(student,1);
```

จากตัวอย่างข้างต้นเมื่อทำการคิวรี่ จะทำให้เกิด Error ขึ้นเนื่องจาก Informix ไม่รองรับการเรียกใช้รูทีนแบบนี้

เรียกใช้รูทีนได้ใหม่ดังนี้

```
Select Delete_Member(student,1) From Student;
```

จากตัวอย่างข้างต้นจะทำการคิวรี่ผ่าน แต่จะเกิด Error ขึ้นเนื่องจากเกิดกิจกรรมที่มีผลกระทบกับข้อมูลในขณะที่ทำการคิวรี่

ดังนั้นวิธีแมนนิพูเลทโรว์ไทป์จึงยังไม่สามารถแก้ไขปัญหานี้ได้

- ปัญหาการพอลิมอร์ฟีซึม (Polymorphism)

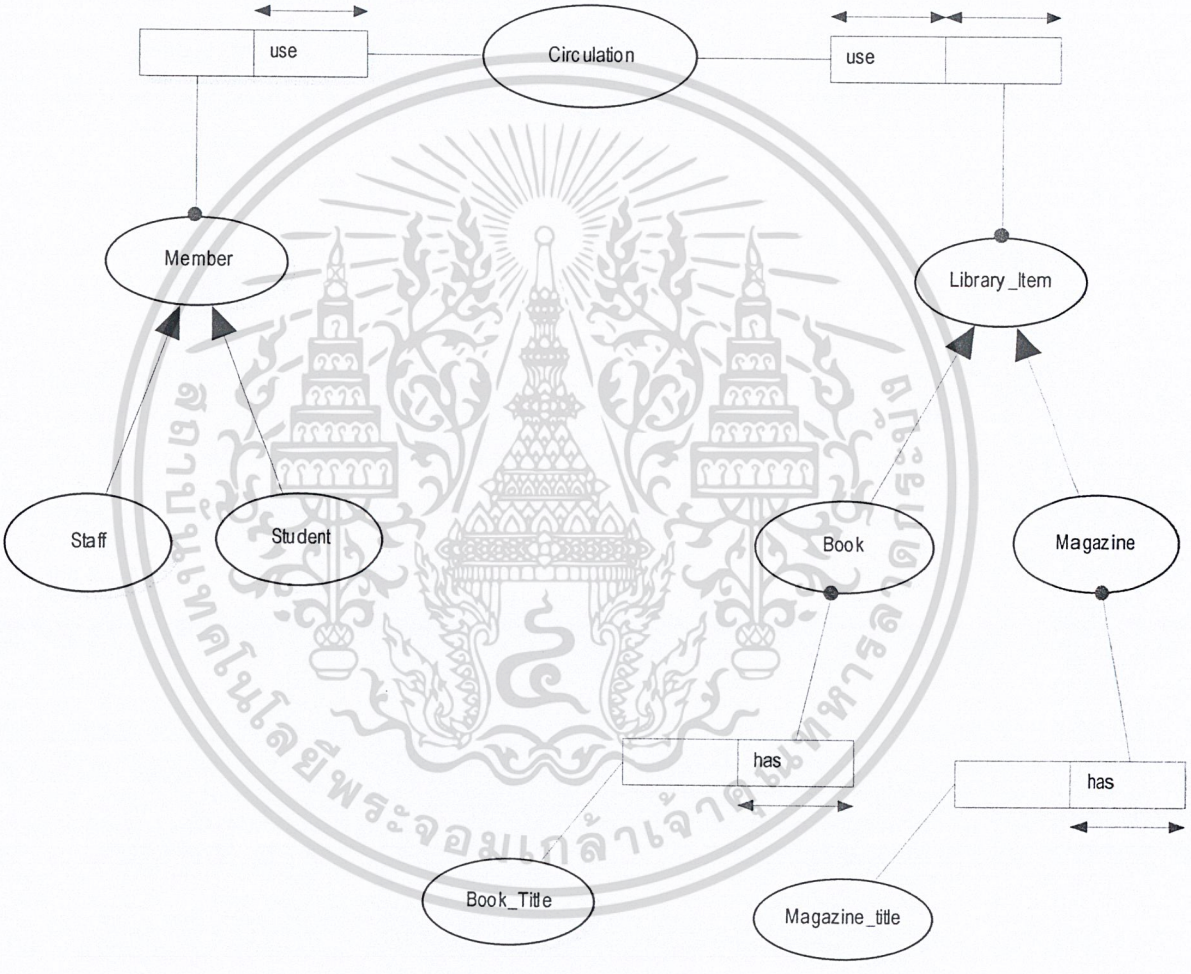
เนื่องจากไม่สามารถที่จะทำการสืบทอดคุณสมบัติของฟังก์ชันการทำงานได้ ดังนั้นจึงไม่สามารถที่จะทำการพอลิมอร์ฟีซึมได้ด้วย เพราะคุณสมบัติพอลิมอร์ฟีซึมเป็นผลพลอยได้ที่มาจากคุณสมบัติสืบทอดคุณสมบัติอีกที

ภาคผนวก ก

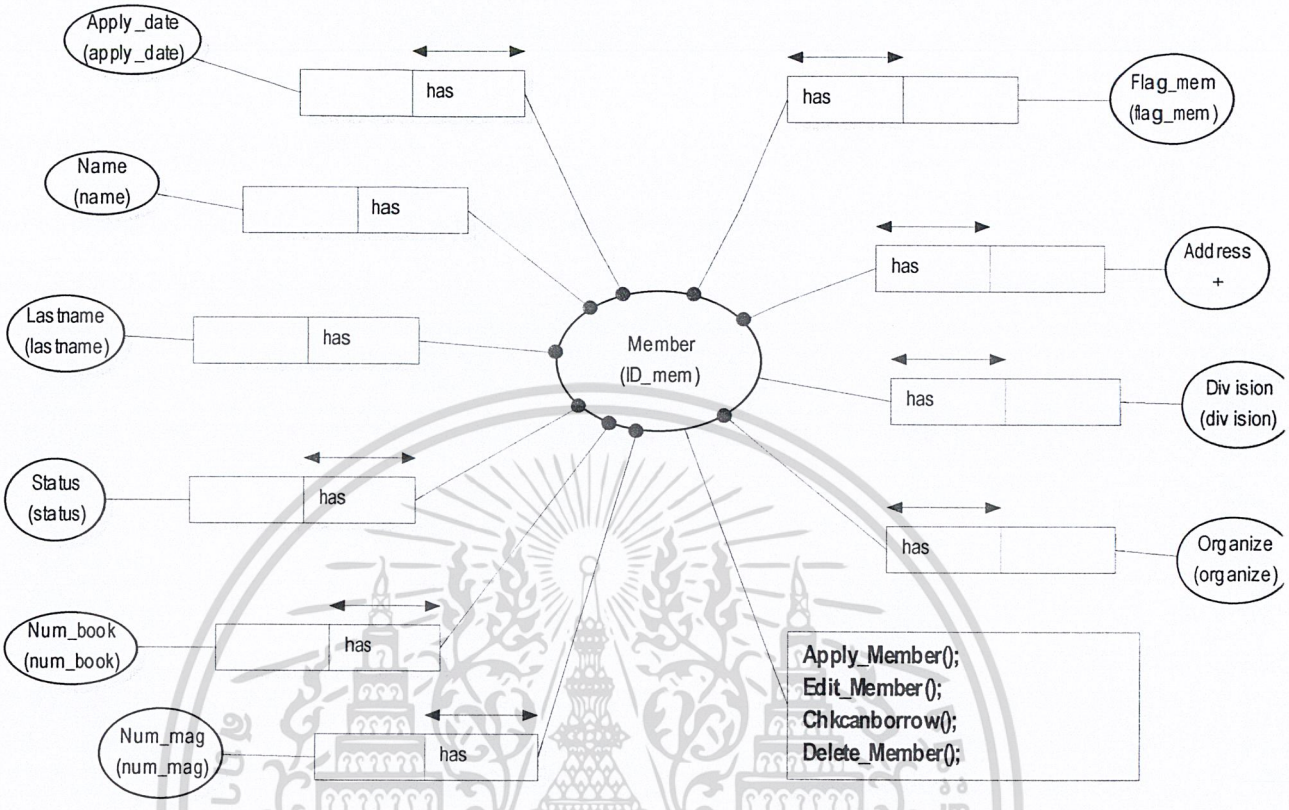
แบบจำลองฐานข้อมูล (OONIAM)

กรณีศึกษา : ระบบห้องสมุด(Case Study : Library System)

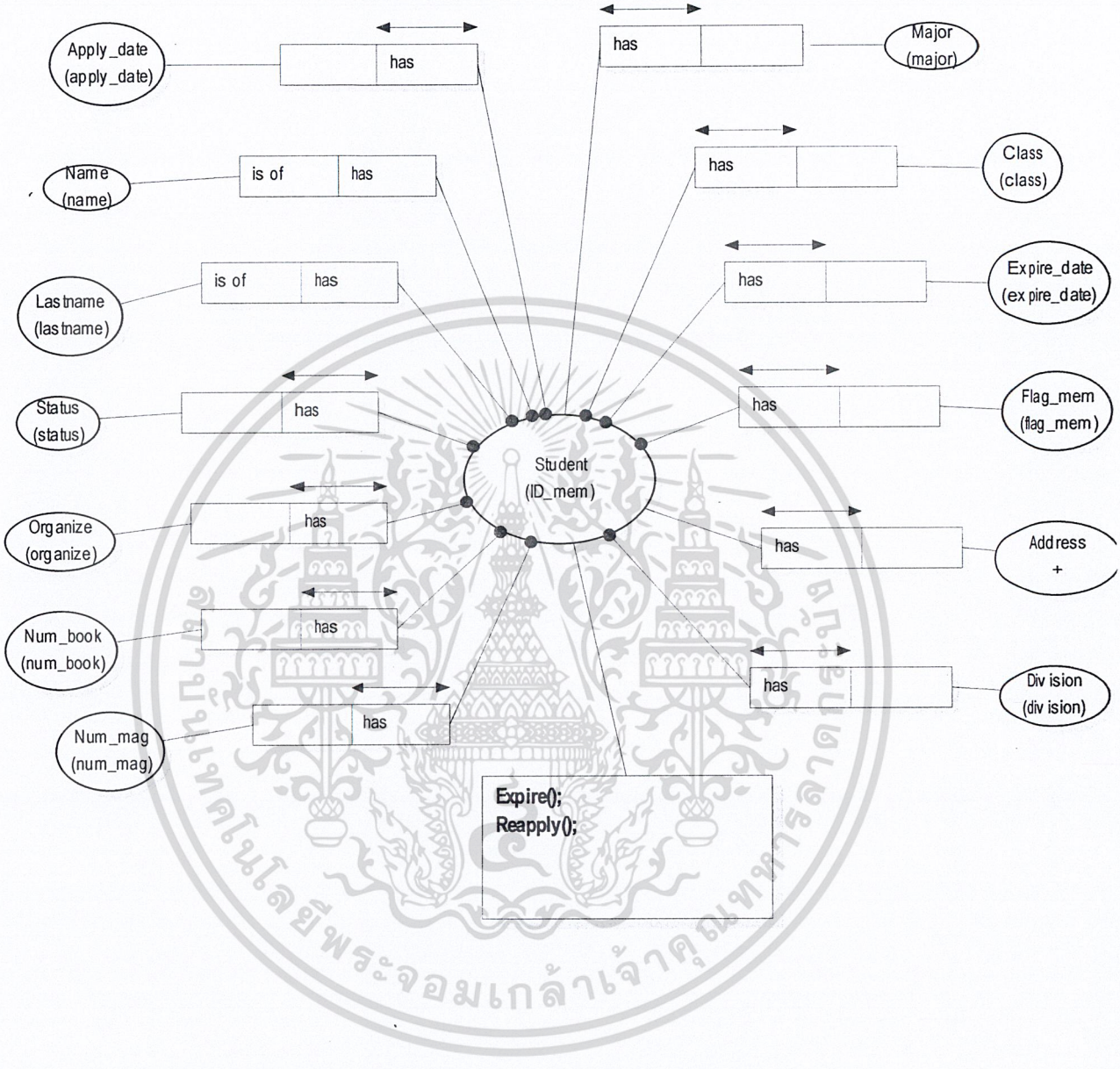
Main Schema : Library System



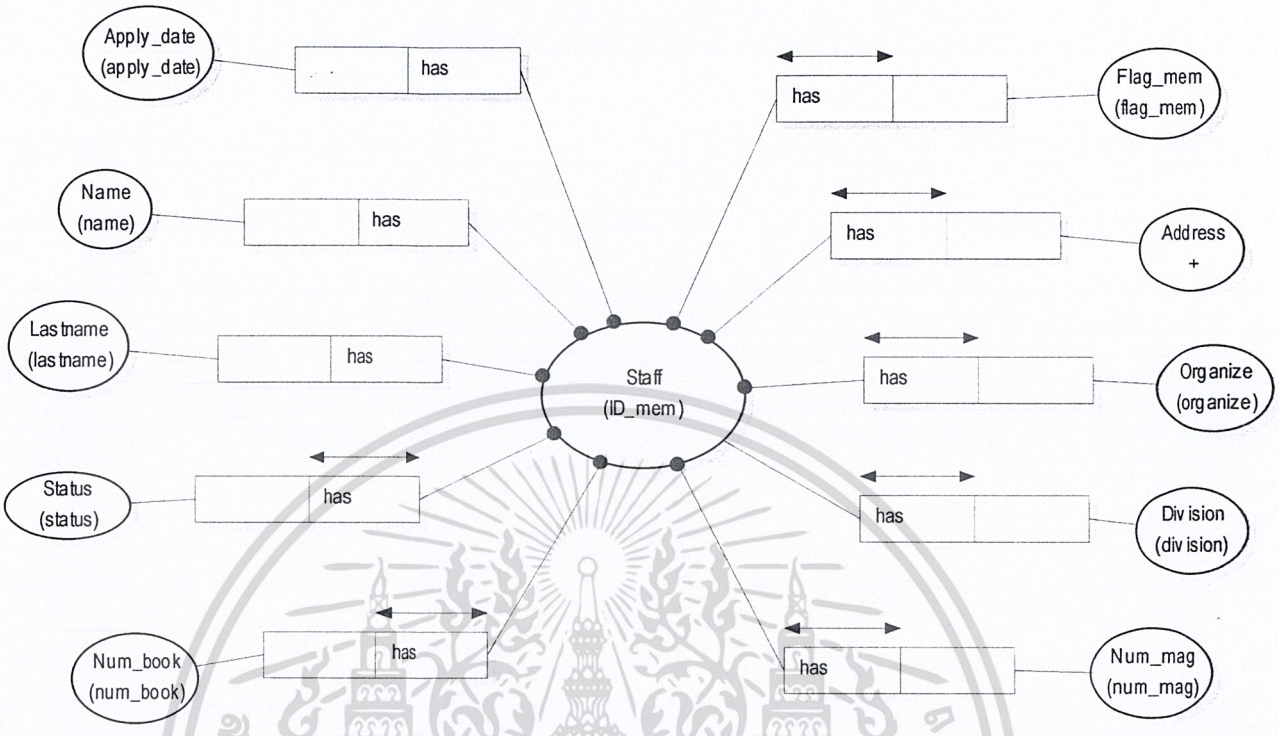
Sub Schema level 1 : Class Member



Sub Schema level 1 : Class Student

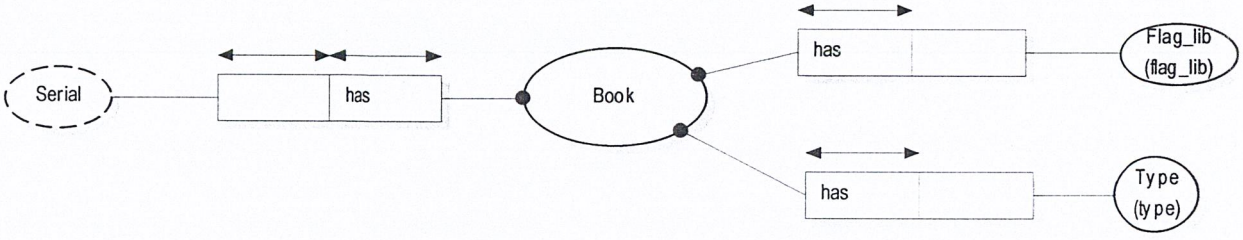


Sub Schema level 1 : Class Staff

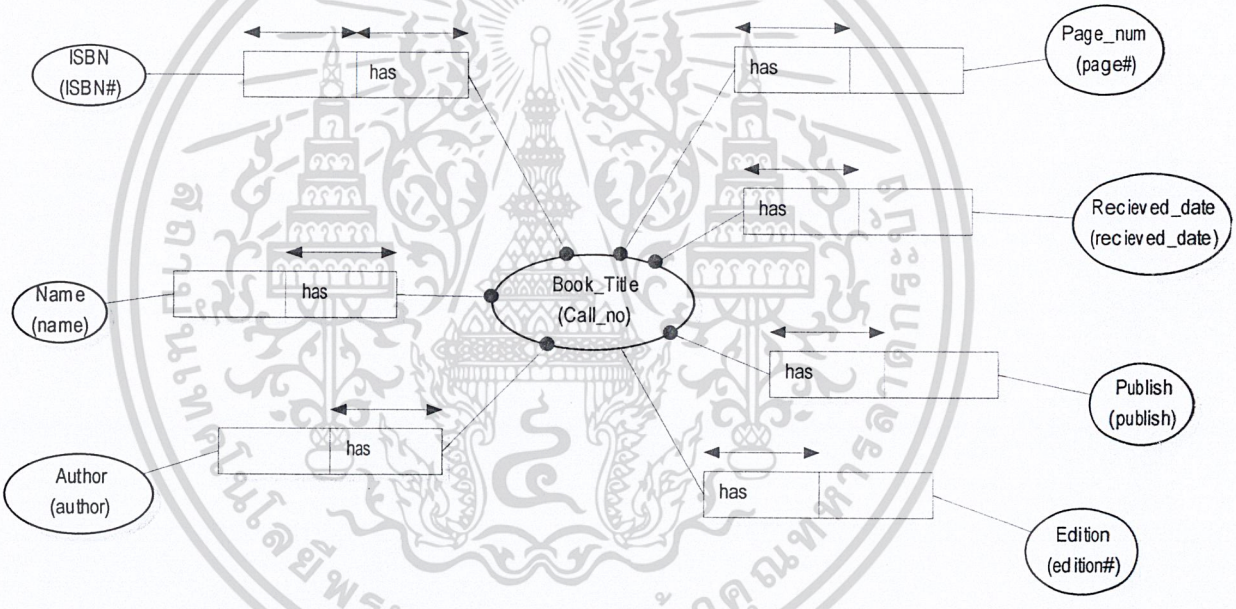


Sub Schema level 1 : Class Library

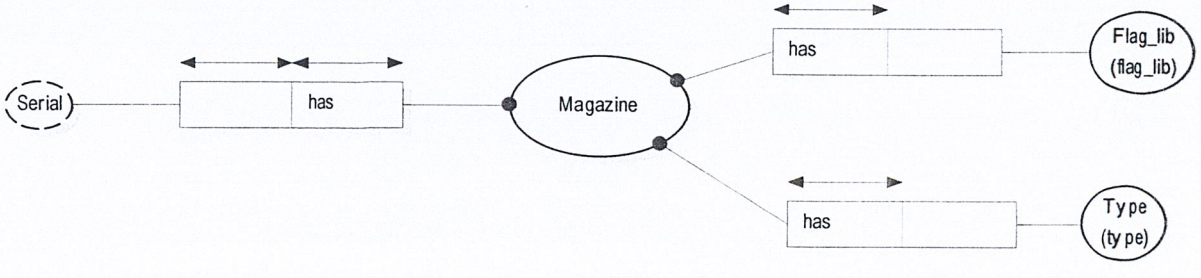
Sub Schema level 1 : Class Book



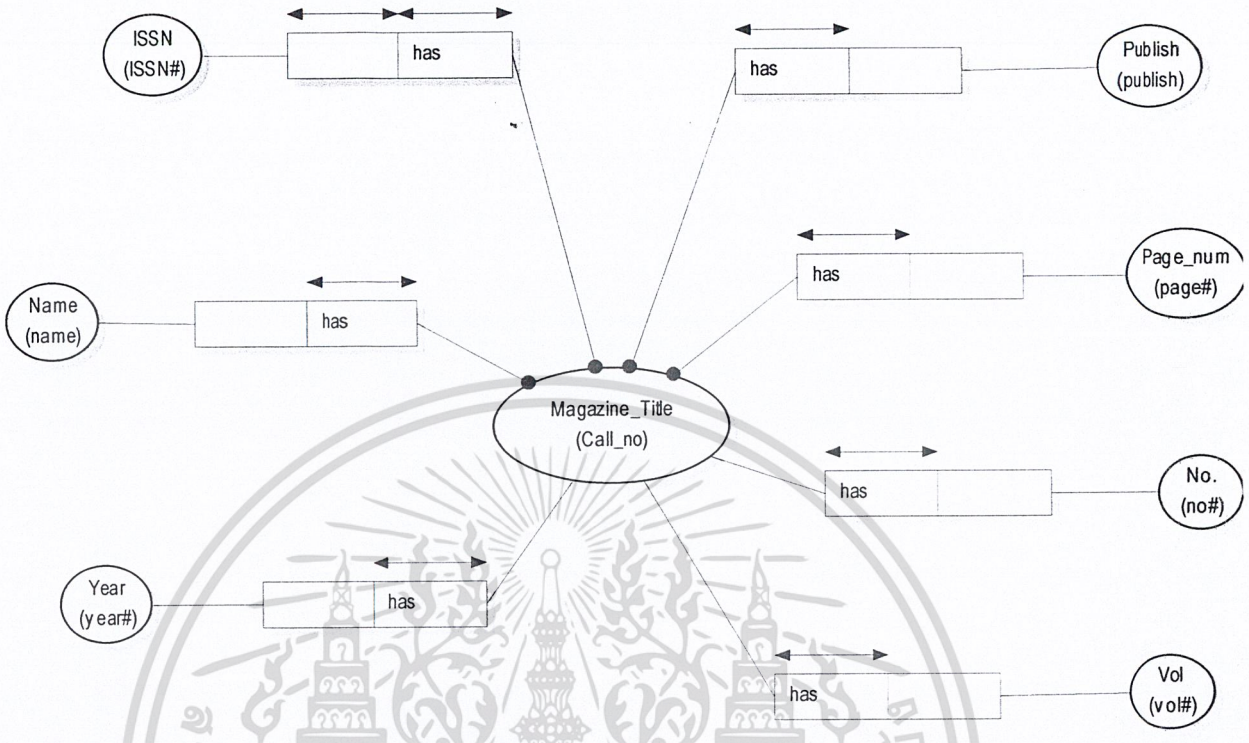
Sub Schema level 1 : Class Book_Title



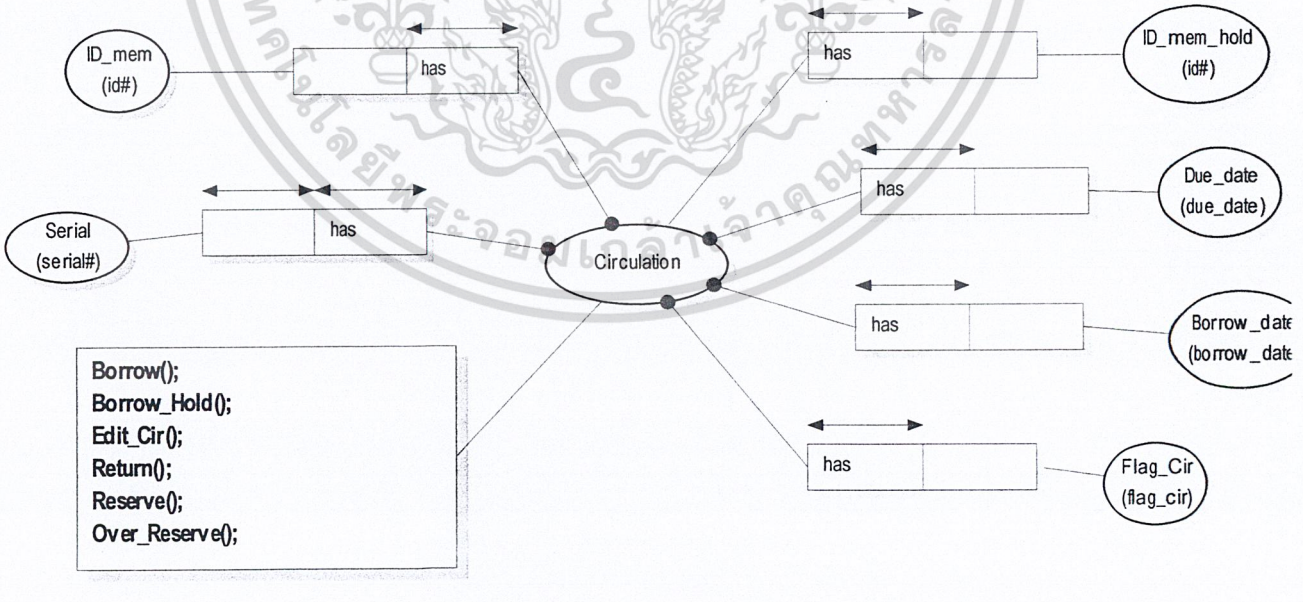
Sub Schema level 1 : Class Magazine



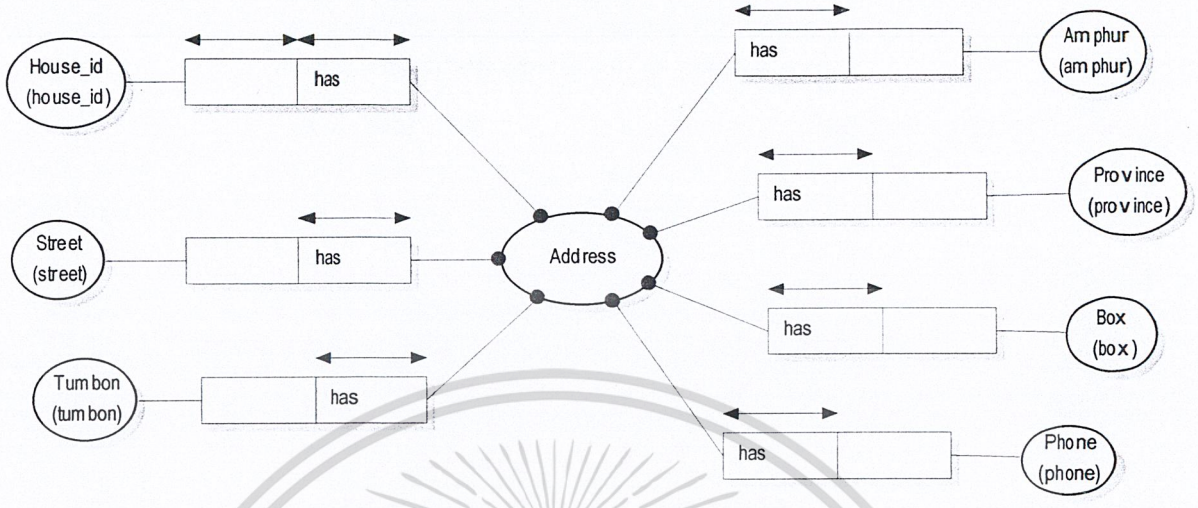
Sub Schema level 1 : Class Magazine_Title



Sub Schema level 1 : Class Circulation



Sub Schema level 2 : Class Address



บรรณานุกรม

[1] Informix Inc., " INFORMIX-Universal Server Informix Guide to SQL : Totorial, Version 9.1", March 1997

[2] Informix Inc., " INFORMIX-Universal Server Informix Guide to SQL : Syntax, Version 9.1", March 1997

[3] Martin Fowler, Kendall Scott, "UML Distilled Second Edition A Brief Guide to the Standard Object Modeling Language", Addison-Wesley Inc.,1998

[4] Morgan Kaufmann Publishers, Inc., "Object-Relational DBMSs Tracking The Next Great Wave",Second Edition, 1997

[5] สุนทริน วงศ์ศิริกุล, "พัฒนาโมเดลยุคใหม่ UML Unified Modeling Language", Success Media CO., LTD