การออกแบบและพัฒนาระบบสารสนเทศเชิงวัตถุสัมพันธ์โดยใช้ OONIAM/UML

กรณีศึกษา : ระบบงานควบคุมและตรวจสอบวัสดุ

The Design and Implementation of an Object-Relational System

Using OONIAM/UML

The Case Study of Stock Control System

โดย

นายศรุต สืบศิริพงศ์

นายสุชา สมานชาติ

อาจารย์ที่ปรึกษา

รศ.ดร. ศุภมิตร จิตตะยโศธร

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2544

ปริญญานิพนธ์ปีการศึกษา 2544

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การออกแบบและพัฒนาระบบสารสนเทศเชิงวัตถุสัมพันธ์โดยใช้ OONIAM/UML
กรณีศึกษา : ระบบงานควบคุมและตรวจสอบวัสดุ

ผู้จัดทำ

1. นาย ศรุต สืบศิริพงศ์ รหัสประจำตัว 41014413
2. นาย สุชา สมานชาติ รหัสประจำตัว 41014468

อาจารย์ที่ปรึกษา

(รศ.ดร. ศุภมิตร จิตตะยโศธร)

การออกแบบและพัฒนาระบบสารสนเทศเชิงวัตถุสัมพันธ์โดยใช้ OONIAM/UML
กรณีศึกษา : ระบบงานควบคุมและตรวจสอบวัสดุ

นายศรุต สืบศิริพงศ์
นายสุชา สมานชาติ
รศ.ดร.ศุภมิตร จิตตะยโศธร อาจารย์ที่ปรึกษา
ปีการศึกษา 2544

บทคัดย่อ

ในปัจจุบันเทคโนโลยีทางด้านฐานข้อมูลได้พัฒนาไปมาก ทั้งในแง่ของประสิทธิภาพการทำงานที่เร็วขึ้น และในแง่ของความสามารถที่มากขึ้น ทำให้ได้รับการนำไปใช้อย่างกว้างขวาง ในส่วนของแนวทางที่ใช้ในการจัดเก็บข้อมูลในฐานข้อมูลก็มีการพัฒนาและคิดค้นขึ้นมาอยู่เรื่อย ๆ เช่นกัน ตัวอย่างหนึ่งของแนวทางเหล่านี้ได้แก่ การนำแนวคิดเชิงวัตถุ (Object-Oriented Concept) ซึ่งมีจุดกำเนิดมาจากส่วนของวิธีการพัฒนาและเขียนโปรแกรมก่อน มาประยุกต์เข้ากับฐานข้อมูลสัมพันธ์ เรียกว่า ฐานข้อมูลเชิงวัตถุสัมพันธ์ (Object-Oriented Database) ซึ่งทำให้ระบบจัดการฐานข้อมูลที่ใช้ฐานข้อมูลเชิงวัตถุสัมพันธ์สามารถจัดเก็บข้อมูลที่มีลักษณะที่ซ้ำซ้อนได้ดีกว่าระบบจัดการฐานข้อมูลแบบเดิมซึ่งเป็นแบบฐานข้อมูลสัมพันธ์ (Relational Database)

วิทยานิพนธ์ฉบับนี้จะเน้นในด้านการศึกษาและวิจัยการออกแบบแอปพลิเคชั่นและฐานข้อมูลเชิงวัตถุสัมพันธ์ มีจุดประสงค์เพื่อหาสิ่งที่อาจปรับปรุงให้ดีขึ้นได้ในเครื่องมือที่ใช้ในการออกแบบแอปพลิเคชั่น โดยทดลองสร้างโปรแกรมและฐานข้อมูลที่สามารถปรับปรุงให้ใช้งานในองค์กรที่มีกฎเกณฑ์แตกต่างกันได้ที่เรียกว่า คอมมอนซอฟท์แวร์ (Common Software) โดยอาศัยโมเดลที่ชื่อว่า Unified Modeling Language (UML) ในการออกแบบระบบเชิงวัตถุ และใช้โมเดลไนแอมเชิงวัตถุ (OONIAM) ในการช่วยออกแบบฐานข้อมูลเชิงวัตถุสัมพันธ์ โดยจะไม่เน้นในส่วนของการพัฒนาระบบงานเพื่อนำไปใช้งานจริง แต่จะสร้างเพียงระบบต้นแบบ เพื่อวิเคราะห์ UML และ OONIAM ว่าสามารถนำมาใช้ในการออกแบบระบบเชิงวัตถุได้ดีเพียงใด

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

Sarut Suebsiripong

Sucha Smanchat

Assoc. Prof. Dr. Supamit Chittayasothorn

Advisor

ABSTRACT

At present, Database Technology is developed rapidly both in performance and ability. Consequently, it is widely used in many applications. The new methods for storing data in the database are also developed. One of these methods is applying the object-oriented concept, which is previously used in software development, to relational database which results in object-relational database. Object-relational database is better for storing data which contains redundancy than relational database.

This thesis focuses on study and research in application design and object-relation database design. The objective of the research is to improve the design tool (UML) in what is necessary when it is used to design object-relational database application. The database application built as an example in this thesis will be common software - can be customized to satisfy varies organization rules. The Unified Modeling Language (UML) will be used to design application and the object-oriented NIAM (OONIAM) will be used to design the object-relational database. The purpose of the application is to show how well can the UML and OONIAM be used to design application. The completeness of the application is not the main objective.

สารบัญ

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญตาราง

สารบัญรูปภาพ

สารบัญรูปภาพ(ต่อ)

สารบัญรูปภาพ(ต่อ)

สารบัญรูปภาพ(ต่อ)

บทที่ 1
บทนำ

1.1 ความเป็นมาของโครงการ

ในปัจจุบันมีการใช้ UML (Unified Modeling Language) อย่างแพร่หลายในการออกแบบแอปพลิเคชั่นของระบบ ซึ่ง UML เป็นเครื่องมือที่ใช้ในการออกแบบระบบเชิงวัตถุ

ทางด้านการพัฒนาของฐานข้อมูลก็มีฐานข้อมูลเชิงวัตถุสัมพันธ์ (Object-relational database) เกิดขึ้น ซึ่งเป็นการนำแนวคิดเชิงวัตถุไปประยุกต์กับฐานข้อมูลสัมพันธ์ (Relational database) ทำให้ประสิทธิภาพในการเก็บข้อมูลที่มีความซ้ำซ้อนดีขึ้น

โครงการนี้จะศึกษาการนำ UML มาใช้ในการออกแบบแอปพลิเคชั่นที่ใช้ฐานข้อมูลเชิงวัตถุสัมพันธ์ นอกจากนี้ในการสร้างแอปพลิเคชั่นสำหรับระบบหนึ่งต้องใช้เวลาและค่าใช้จ่ายมาก ดังนั้นหากสามารถทำให้แอปพลิเคชั่นที่สร้างขึ้นมาสามารถปรับให้เข้ากับระบบที่มีความคล้ายคลึงกันแต่มีกฎเกณฑ์ต่างกันได้ ก็จะสามารถประหยัดค่าใช้จ่ายได้มาก

1.2 จุดประสงค์ของโครงการ

1. เพื่อศึกษาค้นหาสิ่งที่สามารถนำมาปรับปรุงได้ของ UML เพื่อนำไปใช้ในการออกแบบแอปพลิเคชั่นที่มีการใช้ฐานข้อมูลเชิงวัตถุสัมพันธ์
2. ทดลองสร้างแอปพลิเคชั่นที่สามารถปรับตัวเข้ากับระบบอื่นๆ ได้โดยการนำคุณสมบัติการถ่ายทอดของแนวคิดเชิงวัตถุเข้ามาช่วย

1.3 ขั้นตอนการดำเนินงาน

1) ศึกษาการออกแบบระบบโดยใช้เครื่องมือเชิงวัตถุ (UML)
2) ศึกษาการออกแบบฐานข้อมูลเชิงวัตถุสัมพันธ์โดยใช้เครื่องมือเชิงวัตถุ (UML)
3) สร้างแอปพลิเคชั่นตัวอย่างเพื่อค้นหาสิ่งที่สามารถปรับปรุงได้ในการนำไปใช้ออกแบบแอปพลิเคชั่น และเพื่อแสดงการใช้คุณสมบัติการสืบทอดของแนวคิดเชิงวัตถุเพื่อช่วยในการทำให้แอปพลิเคชั่นสามารถปรับให้เข้ากับกฎขององค์กรต่างๆ ได้

1.4 ขอบเขตของโครงการ

โครงการนี้จะได้ศึกษาการออกแบบฐานข้อมูลเชิงวัตถุสัมพันธ์โดยใช้กรณีศึกษาคือระบบงานควบคุมและตรวจสอบวัสดุ โดยใช้เครื่องมือออกแบบเชิงวัตถุคือ UML และ NIAM จากนั้นจะทดลองสร้างเป็นแอปพลิเคชั่น โดยใช้ระบบจัดการฐานข้อมูล Informix โดยจะเน้นในด้านการพิจารณาผลดีผลเสียของการใช้ UML เป็นหลัก

1.5 Literature Review

ก่อนที่จะดำเนินงานตามที่ได้ตั้งเป้าหมายไว้ ได้ทำการค้นคว้าเอกสารทางวิชาการจากที่ต่างๆ เพื่อหาว่ามีเอกสารใดที่มีเนื้อหาตรงกับจุดประสงค์ของโครงการนี้หรือไม่

ตัวอย่างของเอกสารทางวิชาการที่ได้ศึกษามีดังนี้คือ

- "Evolution of Object-Relational Database Technology in DB2", Donald D. Chamberlin, IBM Research Division, Almaden Research Center
- "Object-Relational Database Management System (ORDBMS) Using Frame Model Approach", H K Wong and Anthony S. Fong, Department of Electronic Engineering City University of Hong Kong
- "On Transformations from UML Models to Object-Relational Databases", Wai Yin Mok, David P. Paper, Department of Business Information Systems and Education, Utah State University

1.6 เนื้อหาภายในปริญญานิพนธ์

บทที่ 1 บทนำ แสดงความเป็นมา, จุดประสงค์, ขั้นตอนการทำงาน และขอบเขตของโครงการ

บทที่ 2 UML เป็นเนื้อหารายละเอียดของโมเดล UML ประกอบด้วยความเป็นมาของ UML ความหมาย และการเขียนไดอะแกรมต่างๆของ UML

บทที่ 3 เป็นเนื้อหาความเป็นมา, ความหมาย, ส่วนประกอบ และการออกแบบฐานข้อมูลโดยการใช้ NIAM และ OONIAM

บทที่ 4 เป็นเนื้อหาเกี่ยวกับฐานข้อมูลเชิงวัตถุสัมพันธ์ ลักษณะและคุณสมบัติของฐานข้อมูลเชิงวัตถุสัมพันธ์ ชนิดข้อมูลแบบต่างๆในฐานข้อมูลเชิงวัตถุสัมพันธ์ และคำสั่ง SQL ตัวอย่างเพื่อใช้กับฐานข้อมูลเชิงวัตถุสัมพันธ์

บทที่ 5 ขั้นตอนการออกแบบแอปพลิเคชั่นโดยใช้ UML พร้อมทั้งปัญหาที่ได้พบจากการออกแบบและสร้างแอปพลิเคชั่น

บทที่ 6 ไดอะแกรมต่างๆของ UML และ OONIAM ที่ได้จากการออกแบบระบบงานควบคุมและตรวจสอบวัสดุ

บทที่ 7 การใช้คุณสมบัติการถ่ายทอดและโพลีมอร์ฟิซึมของแนวคิดเชิงวัตถุมาช่วยทำให้การแก้ไขแอปพลิเคชั่นเพื่อสามารถเข้ากับกฎขององค์กรต่างๆที่ไม่เหมือนกันได้ง่ายขึ้น และ ปัญหาและอุปสรรคต่างๆที่เกิดขึ้นรวมทั้งการแก้ไข

บทที่ 8 บทสรุปโครงงาน

ภาคผนวก หน้าจอของแอปพลิเคชั่นที่ได้สร้างขึ้น และเนื้อหาเอกสารวิชาการที่มีความเกี่ยวข้องกับปริญญานิพนธ์

บทที่ 2

Unified Modeling Language(UML)

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

การวิเคราะห์และออกแบบระบบด้วยวิธีการเชิงวัตถุ เป็นวิวัฒนาการ ไม่ใช่เป็นการปฏิวัติการวิเคราะห์และออกแบบระบบที่มีอยู่ เนื่องจากการพัฒนาระบบด้วยวิธีเชิงวัตถุนั้นยังคงข้อดีในกรพัฒนาระบบแบบเก่าแต่ก็ได้เพิ่มข้อดีใหม่ๆ เข้าไปด้วย ในการพัฒนาระบบแบบเก่า จะพบกับความซ้ำซ้อนเนื่องจากการแบ่งระบบออกเป็นส่วนย่อย 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

2.1 มุมมองของ UML

ในการออกแบบระบบที่มีขนาดใหญ่และมีความซับซ้อนมากๆ นั้นจะทำให้ผู้ออกแบบระบบไม่สามารถที่จะออกแบบระบบได้ครบถ้วน ดังนั้นจึงต้องมีการมองระบบเป็นมุมมองต่างๆ เพื่อทำให้ง่ายใน

การออกแบบ ดังนั้นระบบจึงมี View ที่ต่างๆกัน ซึ่งแต่ละ View จะแสดงมุมมองเฉพาะของระบบซึ่งอธิบายรวมกันเป็นระบบที่สมบูรณ์ ซึ่งจะประกอบด้วย View ต่างๆดังนี้

1. มุมมองยูสเคส (Use Case View)

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

2. มุมมองทางโลจิก (Logical View)

อธิบายการทำงานต่างๆที่ถูกออกแบบไว้ภายในระบบ ว่าระบบจะมีบริการอะไรให้กับผู้ใช้บ้าง โดยจะแสดงโครงสร้างแบบสถิต (dynamic collaboration) ซึ่งจะเกิดขึ้นเมื่ออ็อบเจ็คส่งแมสเซส ระหว่างกันในการทำงาน

โครงสร้างแบบสถิตจะอธิบายโดยใช้ คลาสไดอะแกรม (Class diagram) และ อ็อบเจ็คไดอะแกรม (Object diagram) ส่วนการทำงานร่วมกันแบบไดนามิกจะอธิบายโดยใช้ สเตทไดอะแกรม (State diagram), ซีเควนซ์ไดอะแกรม (Sequence diagram), คอลแลโบเรชั่นไดอะแกรม (Collaboration diagram) และ แอ็คทิวิตี้ไดอะแกรม (Activity diagram)

3. มุมมองคอมโพเนนท์ (Component View)

อธิบายการสร้างและความขึ้นต่อกันของโมดูล (Module) ที่ใช้ในการพัฒนาระบบ โดยใช้คอมโพเนนท์ไดอะแกรม (Component diagram) ในการอธิบาย

4. มุมมองดีพลอยเมนท์ (Deployment View)

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

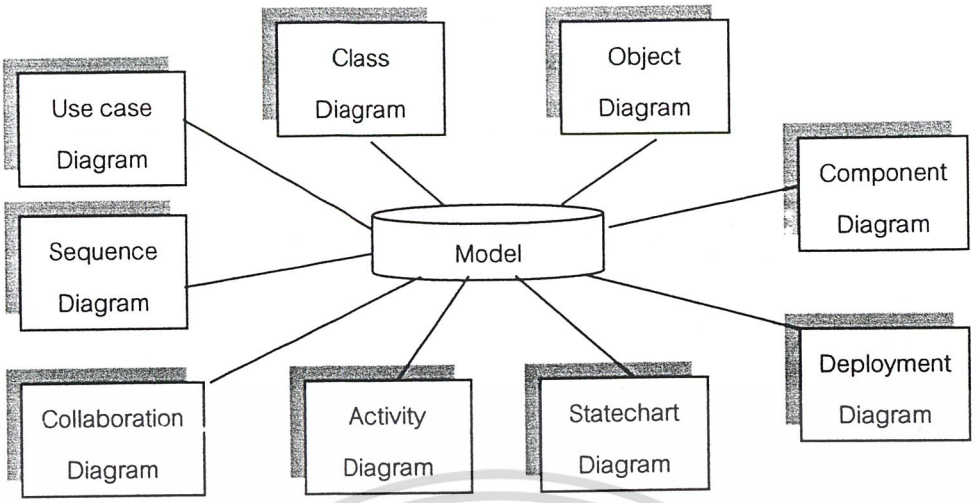
5. มุมมองโพรเซส (Process View)

แสดงการทำงานร่วมกันและการติดต่อกันของส่วนต่างๆในระบบ

2.2 แผนภาพของ UML

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

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



รูปที่ 2.1 แผนภาพแสดงไดอะแกรมของ UML ทั้งหมด

2.2.1 ยูสเคสไดอะแกรม (Use Case diagram)

ยูซเคสไดอะแกรมแสดงถึงตัวกระทำ (Actors), ยูซเคส และความสัมพันธ์ระหว่างยูซเคสกับตัวกระทำยูซเคสใช้แทนการทำงานที่ระบบทำได้ เช่นระบบย่อยหรือคลาส โดยมองจากตัวกระทำภายนอกที่มีการติดต่อกับระบบ

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

2.2.1.1 ยูสเคส (Use Case)

ยูซเคสใช้แทนหน่วยการทำงานที่ระบบ หรือระบบย่อย หรือคลาสมีให้ ซึ่งแสดงให้เห็นโดยการแลกเปลี่ยนข้อมูลระหว่างระบบกับตัวกระทำภายนอก

จุดเอ็กเทนชั่น (extension point) เป็นจุดอ้างอิงภายในยูซเคสซึ่งอาจมีการทำงานของยูซเคสอื่นเข้ามาแทรกได้ แต่ละจุดเอ็กเทนชั่นจะมีชื่อที่แตกต่างกันในยูซเคสหนึ่งๆและมีคำบรรยายตำแหน่งการทำงานภายในพฤติกรรมของยูซเคส

2.2.1.2 ตัวกระทำ (Actor)

คือเซตของบทบาทที่เหมือนกันของผู้ใช้ เช่น บทบาทเป็นพนักงานขาย ตัวกระทำจะถูกเขียนแทนด้วยรูปสติ๊กแมน(ดูตัวอย่าง “Salesperson” ในรูปที่ 2.2)

2.2.1.3 ความสัมพันธ์ของยูสเคส (Use Case Relationship)

ความสัมพันธ์ (Relationship) มาตรฐานระหว่างตัวกระทำกับยูซเคสและระหว่างยูซเคส กับยูสเคสมีอยู่หลายแบบ

แอสโซสิเอชั่น– Association: การกระทำร่วมกันระหว่างตัวกระทำกับยูซเคสแสดงในไดอะแกรมโดยเส้นตรง อาจมีมัลติพลิซิตี้และลูกศรกำกับทิศทางได้

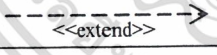
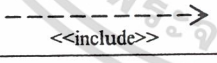
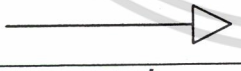
เอ็กเท็นด์ – Extend: ความสัมพันธ์แบบขยาย (Extend) ระหว่างยูซเคส A กับยูซเคส B

หมายความว่าตัวอย่าง (Instance) ของ B อาจถูกเอ็กเท็นด์ โดยพฤติกรรมที่กำหนดใน A แสดงในไดอะแกรมโดยเส้นประที่มีลูกศรหัวเปิดจากยูซเคสที่เป็นตัวให้การขยายไปยัง use case ที่เป็นฐาน โดยจะมีคีย์เวิร์ด <<extend>> กำกับไว้ ส่วนเงื่อนไขของความสัมพันธ์อาจใส่ไว้ใกล้กับคีย์เวิร์ด <<extend>>

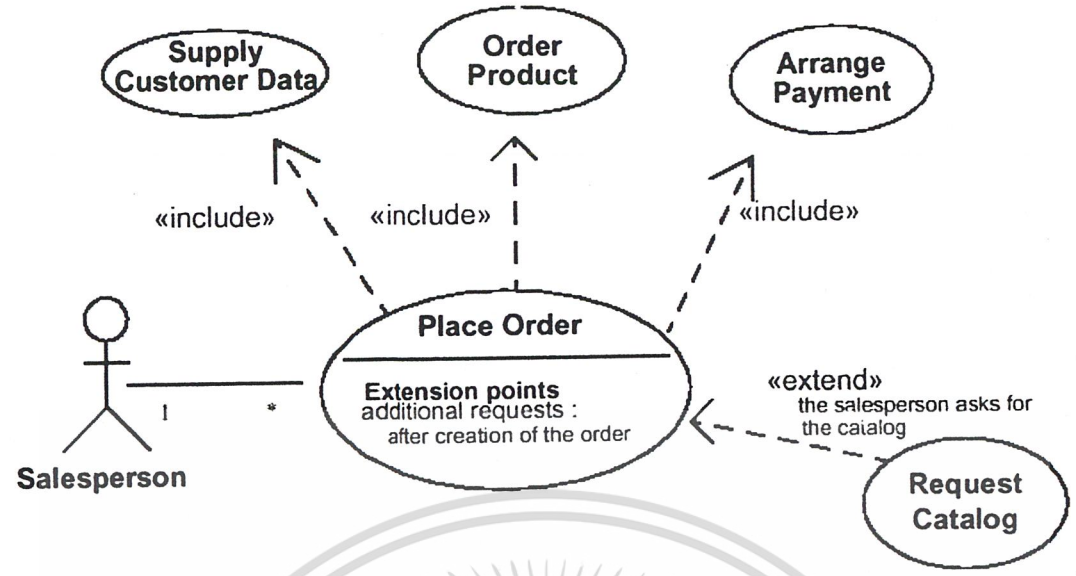
เจนเนอรัลไลเซชั่น – Generalization: ความสัมพันธ์แบบเจนเนอรัลไลเซชั่นจากยูซเคส A ไปยัง ยูซเคส B หมายถึง A เป็นพ่อแม่ (parent) หรืออยู่เหนือ B แสดงในไดอะแกรมโดยเส้นตรงที่มีหัวลูกศรปิดและกลวง หัวลูกศรชี้ที่ยูซเคสที่เป็นพ่อ

ความสัมพันธ์แบบเจนเนอรัลไลเซชั่นจากตัวกระทำ A ไปยังตัวกระทำ B หมายความว่าตัวกระทำ A สามารถติดต่อกับยูสเคสเดียวกับที่ตัวกระทำ B สามารถติดต่อได้

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

	Use case
	ตัวกระทำหรือ Actor
	Association
	Extend
	Include
	Generalization

ตารางที่ 2.1 สัญลักษณ์ที่ใช้ในยูซเคสไดอะแกรม



รูปที่ 2.2 ตัวอย่างยูซเคสไดอะแกรม

2.2.2 ซีเควนซ์ไดอะแกรม (Sequence Diagram)

ซีเควนซ์ไดอะแกรมใช้แสดงแทนการปฏิสัมพันธ์ (Interaction) ระหว่างอ็อบเจ็คในการทำงานให้ได้ตามจุดประสงค์หนึ่ง ซีเควนซ์ไดอะแกรมอาจใช้เพื่ออธิบายการทำงาน (Operation) หรือยูสเคสก็ได้

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

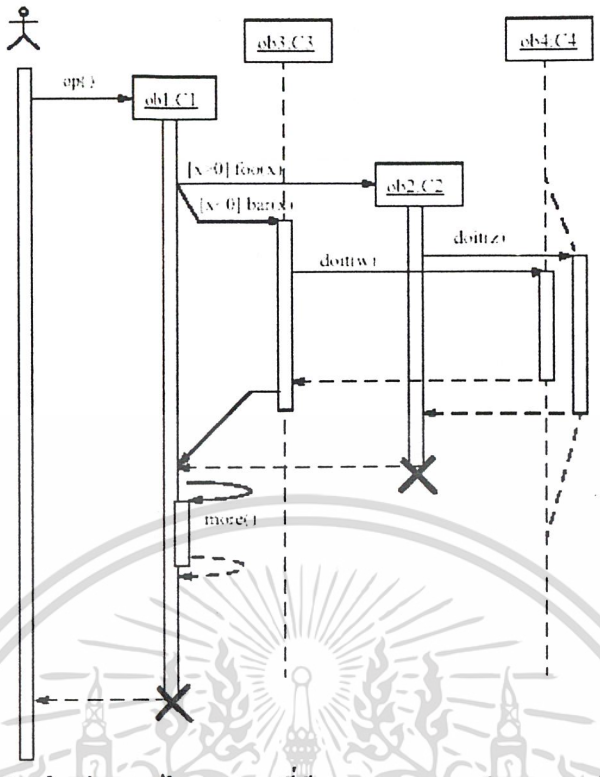
2.2.2.1 เส้นชีวิตของอ็อบเจ็ค (Object Lifeline)

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

2.2.2.2 แอ็คติเวชั่น(Activation)

แสดงช่วงเวลาที่อ็อบเจ็คมีการทำงาน ใช้แทนการทำงานของอ็อบเจ็คในช่วงเวลาหนึ่งรวมทั้งความสัมพันธ์ของการควบคุมระหว่างช่วงการทำงานและผู้เรียกให้ทำงาน เขียนแสดงสี่เหลี่ยมบางๆซึ่งด้านบนคือจุดเวลาเริ่มต้นและด้านล่างคือจุดสิ้นสุด ในการไหลของการควบคุมแบบโพรซีเยอร์ด้านบนของสัญลักษณ์แอ็คติเวชั่นจะอยู่ที่หัวลูกศรที่สั่งให้เกิดการทำงานและด้านล่างคือหางลูกศรของการออกจากโพรซีเยอร์

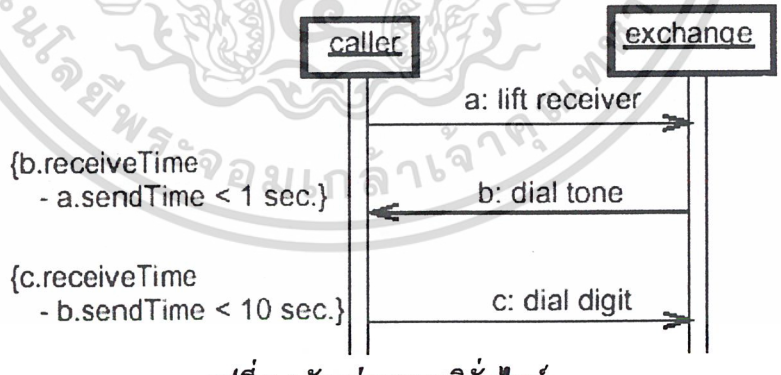
ในกรณีที่มีการเรียกอ็อบเจ็คที่มีแอ็คติเวชั่นอยู่แล้วซ้ำ จะมีอีกแอ็คติเวชั่นหนึ่งเพิ่มขึ้นมาทางด้านขวาของอันแรก (ดูตัวอย่างได้จากรูปที่ 2.3 ที่อ็อบเจ็ค ob4)



รูปที่ 2.3 ตัวอย่างซีเควนซ์ไดอะแกรมที่มีการควบคุมการทำงาน, การใช้เงื่อนไข, การเรียกหาตัวเอง(Recursion), การสร้าง และการทำลาย

2.2.2.3 ทรานสิชั่นไทม์ (Transition Times)

อาจระบุเวลาแบบต่างๆให้กับแมสเสจ เช่น "sending time" หรือ "receiving time"ซึ่งสามารถนำไปใช้เพื่อการบอกอายุของแมสเสจได้



รูปที่ 2.4 ตัวอย่างทรานสิชั่นไทม์

2.2.2.4 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

2.2.3 คอลแลโบเรชั่นไดอะแกรม (Collaboration Diagram)

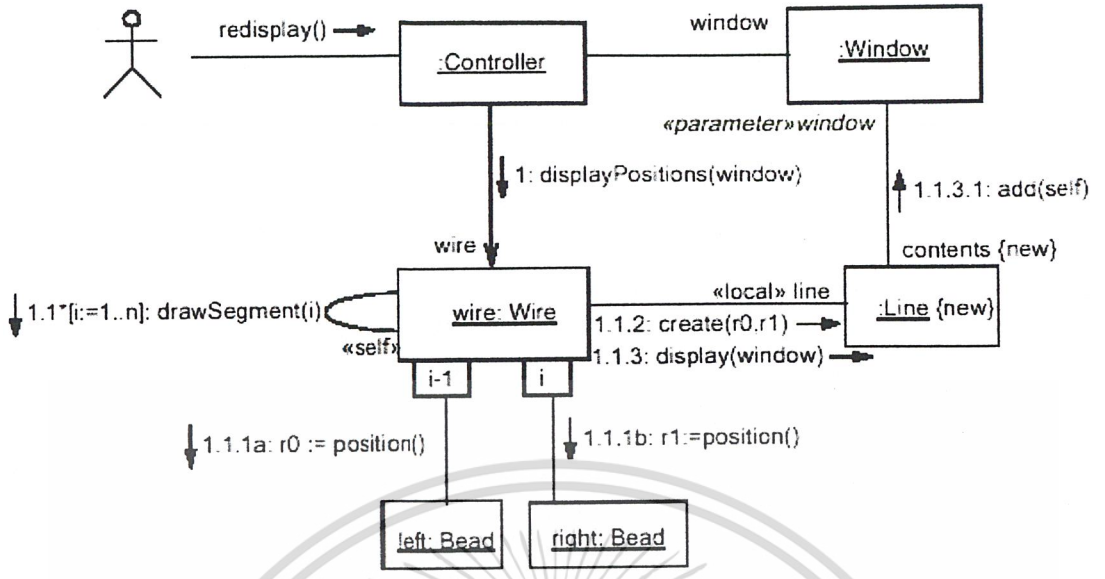
คอลแลโบเรชั่นไดอะแกรมแสดงแทนคอลแลโบเรชั่นซึ่งประกอบด้วยเซ็ตของบทบาบของอ็อบเจ็คและความสัมพันธ์ระหว่างอ็อบเจ็ค ไดอะแกรมนี้อาจใช้แสดงการปฏิสัมพันธ์ระหว่างอ็อบเจ็คโดยการระบุเซตของแมสเสจที่บอกการปฏิสัมพันธ์ระหว่างอ็อบเจ็คที่มีบทบาทอยู่ภายในคอลแลโบเรชั่นเพื่อบรรลุจุดประสงค์หนึ่ง

คอลแลโบเรชั่นไดอะแกรมจะแสดงออกมาเป็นกราฟที่เชื่อมต่อระหว่างอ็อบเจ็ค

คอลแลโบเรชั่นไดอะแกรมอาจใช้เพื่ออธิบายการทำงานหรือคลาสสิฟายเออร์หรือยูสเคส และยังช่วยในการออกแบบโพรซีเยอร์ มีการใช้ลูกศรเพื่อบอกการทำงานโดยตัวเส้นลูกศรแทนตัวกระตุ้นซึ่งถูกส่งไปในทิศทางที่หัวลูกศรชี้ มีหมายเลขกำกับตามลำดับการปฏิสัมพันธ์โดยเริ่มต้นด้วย 1

ส่วนประกอบในไดอะแกรมส่วนใหญ่จะเหมือนกับซีเควนซ์ไดอะแกรมต่างกันที่รูปแบบการเขียน โดยคอลแลโบเรชั่นไดอะแกรมจะไม่มีแกนเส้นชีวิตและแอ็กติเวชั่นเนื่องจากสนใจเฉพาะลำดับการส่งตัวกระตุ้น

ทั้งซีเควนซ์ไดอะแกรมและคอลแลโบเรชั่นไดอะแกรมต่างก็ใช้บอกการปฏิสัมพันธ์ โดยซีเควนซ์ไดอะแกรมแสดงลำดับของตัวกระตุ้นและสามารถเข้าใจได้ง่ายในการพิจารณาด้านเวลา ส่วนคอลแลโบเรชั่นไดอะแกรมจะแสดงความสัมพันธ์ตัวอย่าง(Instance) ช่วยให้ง่ายต่อการเข้าใจผลกระทบต่างๆที่เกิดขึ้นกับตัวอย่างนั้นๆและช่วยในการออกแบบโพรซีเยอร์



รูปที่ 2.5 ตัวอย่างคอลแลโบเรชั่นไดอะแกรม

2.2.4 คลาสไดอะแกรม (Class Diagram)

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

2.2.4.1 คลาสสิฟายเออร์(Classifier)

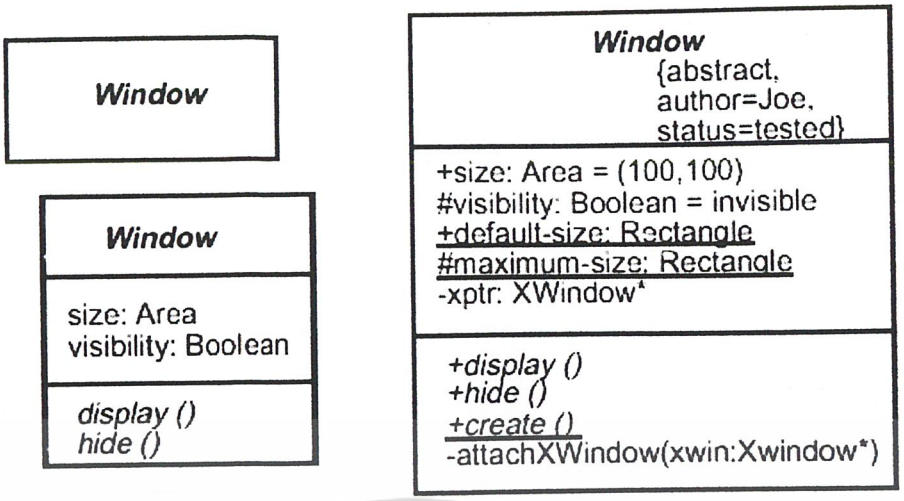
คือ ซูเปอร์คลาสของคลาส (*Class*), อินเตอร์เฟส และ ดาต้าไทป์ (*Datatypes*) ในการเขียนไดอะแกรมส่วนมากจะใช้สี่เหลี่ยมแทน คลาส (Class) ส่วนอินเตอร์เฟสและดาต้าไทป์จะใช้สี่เหลี่ยมโดยมีคีย์เวิร์ดกำกับไว้

2.2.4.2 คลาส (Class)

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

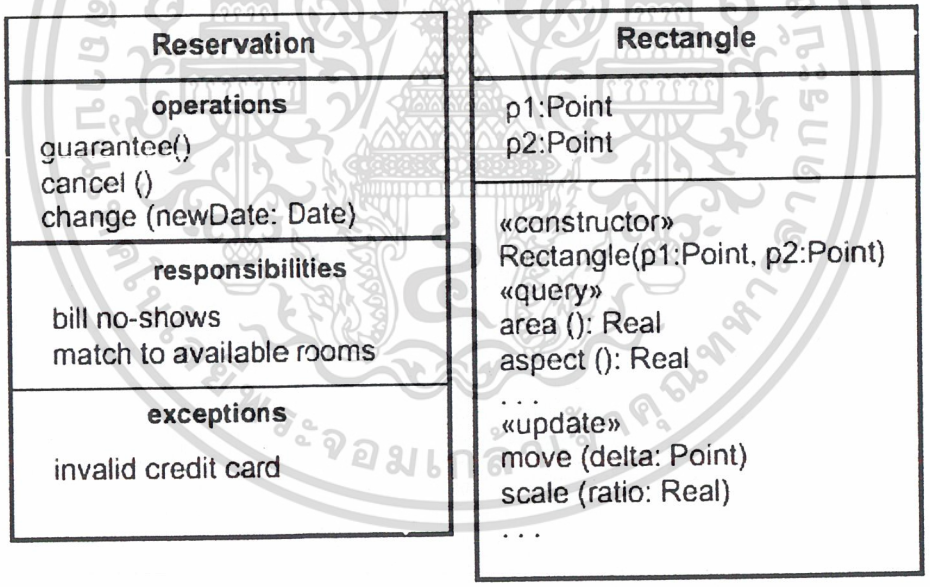
ถ้ามีการอ้างคลาสข้ามแพคเกจจะเขียนชื่อคลาสในรูป

Package-name::Class-name



รูปที่ 2.6 ตัวอย่างคลาส: แบบซ่อนรายละเอียด, แบบแสดงรายละเอียดระดับวิเคราะห์ และแสดงรายละเอียดระดับการ Implementation

อาจมีการกำหนดส่วน (Compartment) ในคลาสเพิ่มขึ้นมาตามที่ผู้ใช้ต้องการ เช่นกำหนด requirement compartment" รวมทั้งอาจมีการใช้สเตอริโอไทป์



รูปที่ 2.7 ตัวอย่างคลาสที่กำหนดส่วนเพิ่ม และการใช้สเตอริโอไทป์

2.2.4.3 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

2.2.4.4 การทำงาน (Operation)

เขียนในรูป

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

โดย visibilityสามารถเป็นได้ 3 แบบ

+ public visibility

protected visibility

- private visibility

เช่น

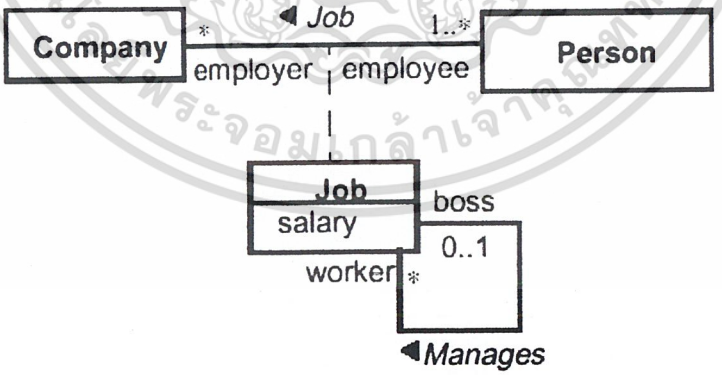
+create ()

+display (): Location

+hide ()

2.2.4.5 แอสโซสิเอชั่น (Association)

ไบนารีแอสโซสิเอชั่น (Binary Association) แสดงความเกี่ยวข้องกันของสองคลาสสิฟายเออร์ เขียนแทนด้วยเส้นตรงเชื่อมระหว่างสัญลักษณ์ของคลาสสิฟายเออร์สองอัน หรือเชื่อมคลาสสิฟายเออร์หนึ่งเข้ากับตัวเอง โดยมีการระบุบทบาบของแต่ละคลาสที่ปลายเส้นแต่ละด้าน และอาจมีชื่อของแอสโซสิเอชั่นกำกับไว้ด้วย (แอสโซสิเอชั่นนี้คือชนิดเดียวกับที่ใช้ในยูสเคส)



รูปที่ 2.8 ตัวอย่างแอสโซสิเอชั่น

ที่ปลายเส้นของแอสโซสิเอชั่นสามารถมีลูกศรเพื่อบอกทิศทางของแอสโซสิเอชั่นคือบอกว่าเป็นแอสโซสิเอชั่นจากคลาสใดไปสู่คลาสใด

2.2.4.6 มัลติพลิซิตี้ (Multiplicity)

มัลติพลิซิตี้แสดงพิกัดจำนวนที่เซตนั้นๆจะมีได้ สามารถนำไปใช้ประกอบกับแอสโซสิเอชั่นได้ ตัวอย่างของมัลติพลิซิตี้เช่น

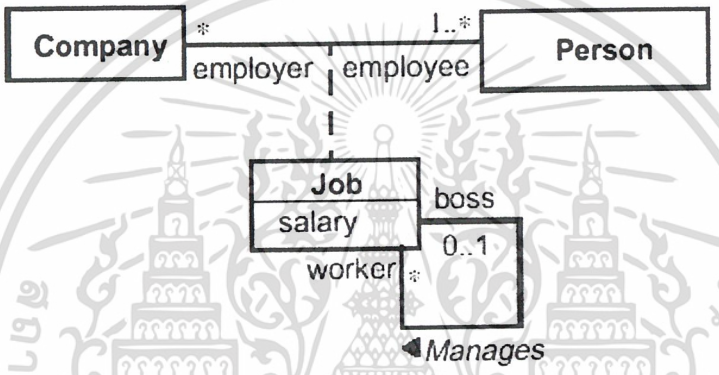
0..1 มีจำนวนได้ 0 ถึง 1

0..* มีจำนวนได้ตั้งแต่ 0 ขึ้นไป

* ไม่จำกัด

2.2.4.7 แอสโซสิเอชั่นคลาส (Association Class)

แอสโซสิเอชั่นคลาสคือแอสโซสิเอชั่นที่มีคุณสมบัติของคลาส เขียนแทนโดยรูปแบบสี่เหลี่ยมแบบเดียวกับคลาสพร้อมกับมีเส้นประเชื่อมเข้ากับแอสโซสิเอชั่น



รูปที่ 2.9 ตัวอย่างแอสโซสิเอชั่นคลาส

2.2.4.8 แอสโซสิเอชั่นแบบ n-ary (n-ary Association)

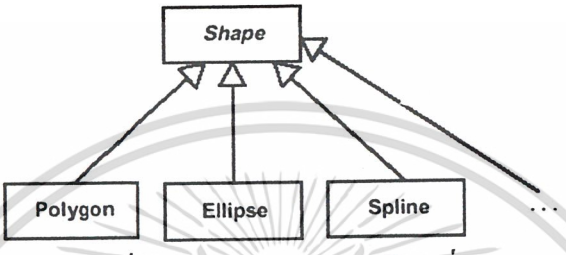
แอสโซสิเอชั่นแบบ n-ary แสดงความเกี่ยวข้องของคลาสสิฟายเออร์ตั้งแต่สามคลาสสิฟายเออร์ขึ้นไป เขียนแทนโดยรูปสี่เหลี่ยมข้าวหลามตัดโดยมีเส้นแอสโซสิเอชั่นเชื่อมไปถึงคลาสที่เกี่ยวข้อง ถ้าต้องการใส่ชื่อให้ก็สามารถเขียนไว้ข้างสี่เหลี่ยมข้าวหลามตัดนั้น

รูปที่ 2.10 ตัวอย่างแอสโซสิเอชั่นแบบ n-ary

2.2.4.9 เจนเนอรัลไลเซชั่น (Generalization)

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

เจนเนอรัลไรเซชั่นเขียนในไดอะแกรมโดยเส้นตรงที่มีหัวลูกศรปิดกลวงโดยที่หัวลูกศรชี้ไปที่สิ่งที่เป็นพ่อ

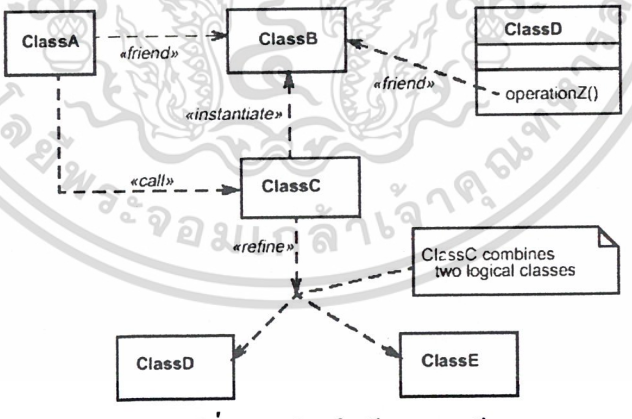


รูปที่ 2.11 ตัวอย่างเจนเนอรัลไลเซชั่น

2.2.4.10 ดีเพนเดนซี (Dependency)

ดีเพนเดนซีระบุความสัมพันธ์ระหว่างสองสิ่งในโมเดล บอกให้รู้ว่าการเปลี่ยนแปลงสิ่งที่เป็นเป้าหมายอาจต้องการการเปลี่ยนแปลงของสิ่งที่เป็นต้นทางของดีเพนเดนซี

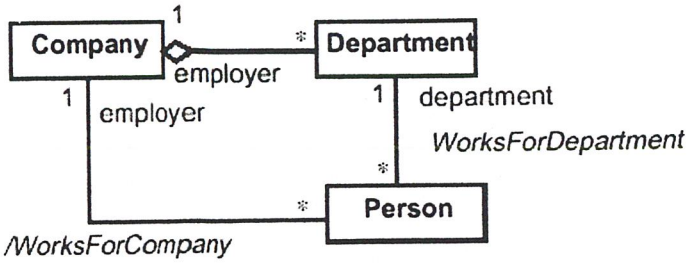
ดีเพนเดนซีเขียนแทนด้วยเส้นประโดยสิ่งที่อยู่ที่หางลูกศรจะขึ้นอยู่กับสิ่งที่อยู่ที่หัวลูกศร อาจมีการใส่ชื่อและสเตอริโอไทป์ให้กับลูกศรได้



รูปที่ 2.12 ตัวอย่างดีเพนเดนซี

2.2.4.11 อักกริเกชั่น (Aggregation)

เป็นแอสโซสิเอชั่นที่ระบุว่ามีการรวมกันหรือเป็นส่วนหนึ่งเกิดขึ้น เขียนแทนแบบเดียวกับแอสโซสิเอชั่นแต่ที่ปลายเส้นด้านที่จะเข้าไปรวมจะเปลี่ยนเป็นรูปสี่เหลี่ยมข้าวหลามตัด



รูปที่ 2.13 ตัวอย่างอักกริเกชั่น

2.2.5 อ็อบเจ็คไดอะแกรม(Object Diagram)

อ็อบเจ็คไดอะแกรมคือกราฟของตัวอย่าง(Instance) ซึ่งรวมทั้งอ็อบเจ็คค่าของข้อมูล(Data value) เป็นการแสดงตัวอย่างของคลาสไดอะแกรม

2.2.6 สเตทไดอะแกรม(State Diagram)

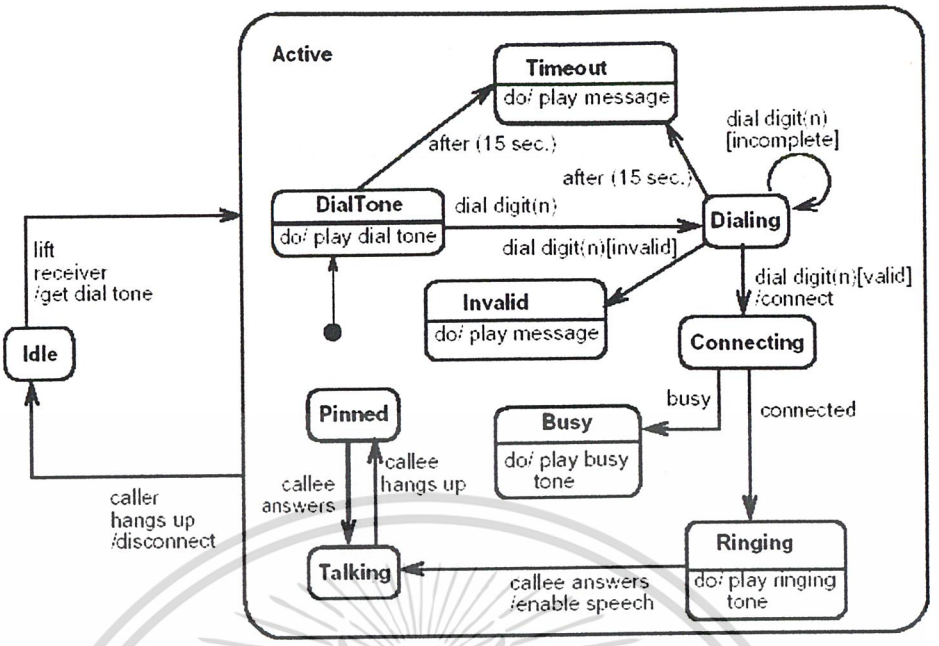
สเตทไดอะแกรม (State Diagram) หรือ สเตทชาร์ทไดอะแกรม (Statechart Diagram) หรือบางครั้งเรียกว่า สเตทแมชชีน (State Machine) เป็นสิ่งที่ใช้แสดงการเปลี่ยนสเตท (state) ของออบเจ็กต์ตั้งแต่เริ่มต้นจนสิ้นสุดการเปลี่ยนแปลงในรอบหนึ่งๆ (1 ซีเควนท์) โดยทั่วไปแล้ว สเตทไดอะแกรม (State diagram) จะใช้ในการอธิบายพฤติกรรมของคลาสแต่ละคลาส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

ลักษณะของการเปลี่ยนสเตท เช่น

- เมื่อเรากดสวิทช์ไฟ หลอดไฟฟ้าจะเปลี่ยนสถานะจากมืดเป็นสว่าง
- เมื่อเรากดปุ่มรีโมทคอนโทรลเพื่อเปลี่ยนช่อง โทรทัศน์จะเปลี่ยนสถานะจากรายการช่องหนึ่งเป็นรายการอีกช่องหนึ่ง
- เมื่อเวลาผ่านไประยะหนึ่ง เครื่องซักผ้าจะเปลี่ยนสถานะจาก “ซัก” เป็น “ปั่น”

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

ตัวอย่างของ สเตทไดอะแกรมแสดงได้ดังรูป



รูปที่ 2.14 ตัวอย่างสเตทไดอะแกรม (State Diagram)

2.2.6.1 สเตท (State)

สเตทเป็นเงื่อนไขระหว่างการทำงานของออบเจ็กต์ หรือเป็นสิ่งที่แสดงการตอบสนองเงื่อนไขการกระทำบางอย่าง หรือ รอคอยเพื่อรับเหตุการณ์เข้ามา สัญลักษณ์ที่ใช้แสดงสเตทจะเป็นรูปสี่เหลี่ยมหัวมน โดยภายในจะประกอบด้วยชื่อสเตทซึ่งจะถูกแบ่งด้วยเส้นภายในสเตทนั้นๆ และภายในสเตทยังสามารถถูกแบ่งออกเป็นส่วนย่อยๆ อื่นได้อีก โดยใช้เส้นตรงในการแบ่ง ซึ่งมีดังต่อไปนี้

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

action-label '/' action-expression

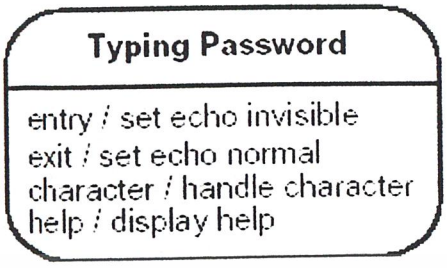
โดยที่จะมีชื่อแอ็กชั่นหลักๆ ที่ถูกสงวนไว้เพื่อจุดประสงค์หลักๆ ซึ่งไม่สามารถนำไปใช้เป็นชื่อของเหตุการณ์ได้ มีดังต่อไปนี้

- entry เป็นส่วนแสดงแอ็กชั่น จะอธิบายการทำงานเมื่อมีการเข้าสู่สเตทนี้
- exit เป็นส่วนแสดงแอ็กชั่น จะอธิบายการทำงานเมื่อมีการออกจากสเตทนี้
- do เป็นส่วนที่แสดงถึงกิจกรรม ที่ถูกกระทำภายในสเตทนี้ ซึ่งกิจกรรมนี้จะถูกกระทำไปจนกว่าที่ action-expression จะเสร็จสมบูรณ์
- include ส่วนนี้เป็นส่วนที่ใช้เรียกสเตทย่อย ซึ่ง action-expression จะประกอบด้วยชื่อของสเตทย่อยที่จะถูกเรียกนั้น

รูปแบบทั่วไปของรายการในทรานสิชันภายใน มีดังต่อไปนี้

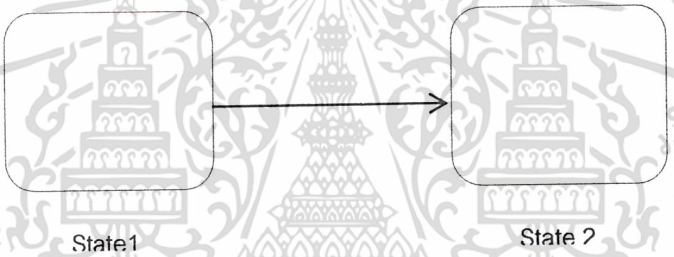
event-name '(' comma-separated-parameter-list ')' '[' guard-condition']' '/' action-expression

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



รูปที่ 2.15 ตัวอย่างสเตท (State)

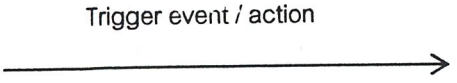
เราสามารถเอาสเตทการทำงานมาเชื่อมโยงถึงกันได้ด้วยกลไกที่เรียกว่า "ทรานสิชันไลน์" (Transition line) โดยใช้สัญลักษณ์เป็นลูกศร ลักษณะของการเชื่อมโยงระหว่างสเตทด้วยทรานสิชันไลน์จะมีลักษณะดังรูป



รูปที่ 2.16 การเชื่อมสเตทด้วยทรานสิชันไลน์ (Transition line)

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

นอกจากนี้เรายังสามารถใส่การคำนวณบางอย่างหรือการทำงานบางอย่าง (Action) ที่ก่อให้เกิดการเปลี่ยนแปลงสถานะการทำงาน และไม่ว่าเราจะใส่เหตุการณ์หรือการทำงานบางอย่างเพิ่มเติมลงในทรานสิชันไลน์ (Transition line) เราจะใส่เอาไว้ข้างๆทรานสิชันไลน์ ตัวอย่างการกำหนด ทริกเกอร์อีเวนต์หรือ แอ็กชัน (Action) มีลักษณะดังรูป



รูปที่ 2.17 การใส่รายละเอียดบนทรานสิชันไลน์

แต่ในบางครั้งการเปลี่ยนสถานะการทำงานก็เป็นไปได้อัตโนมัติ โดยอาจจะอาศัยผลลัพธ์จากสถานะการทำงานที่ผ่านมาทำให้เกิดสถานะใหม่ก็ได้ ลักษณะของการเปลี่ยนแปลงสถานะการทำงานโดยไม่มีเหตุการณ์หรือการทำงานเข้าไปกระตุ้น เราจะเรียกว่า "ทริกเกอร์เลส ทรานสิชัน" (triggerless transition)

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

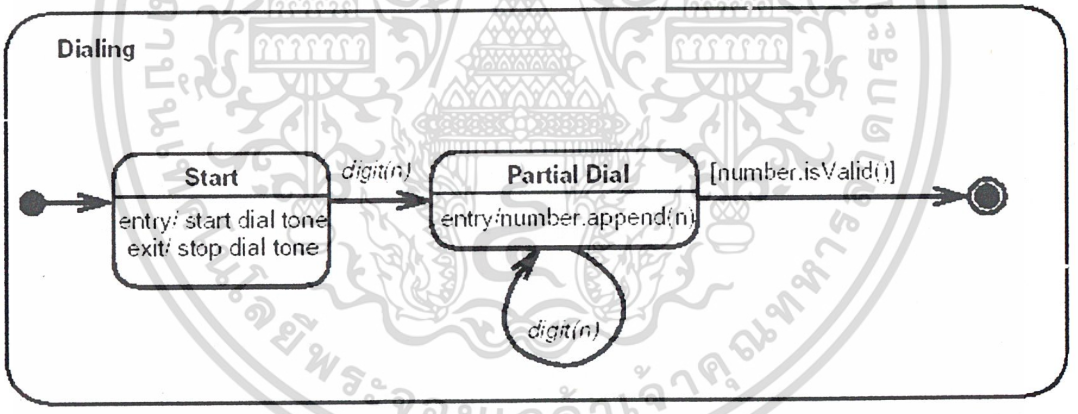
2.2.6.2 คอมโพสิทสเตท (Composite States)

เนื่องจากในสเตทหนึ่งๆ นั้นอาจจะสามารถแสดงถึงการทำงานที่ละเอียดได้อีก ดังนั้นเราจึงอาจมีการเพิ่มเติมรายละเอียดในแต่ละสเตทว่ามีการทำงานย่อยอะไรซ่อนอยู่บ้าง โดยเราจะเรียกสถานะการทำงานย่อยๆ ที่ซ่อนอยู่นั้นว่า "ซับสเตท" (Substates)

สถานะการทำงานหรือซับสเตทนั้น แบ่งเป็น 2 ประเภทคือ

1. ซีเควนท์เชียล (Sequential)

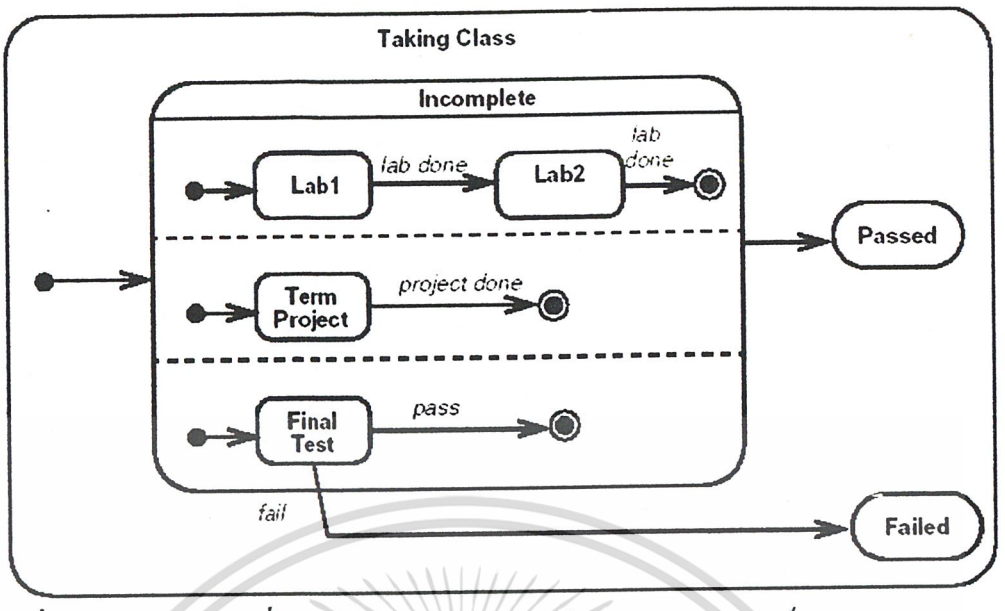
ลักษณะของซีเควนท์เชียลจะเป็นการเปลี่ยนสถานะการทำงานอย่างเป็นลำดับ จากสถานะหนึ่งไปยังสถานะหนึ่ง ลองพิจารณารูปตัวอย่างดังต่อไปนี้



รูปที่ 2.18 สเตท Dialing ที่มีการแบ่งสถานะการทำงานย่อยแบบซีเควนท์เชียล (Sequential)

2. คอนเคอเรนท์ (Concurrent)

ลักษณะของคอนเคอเรนท์นั้น สถานะการทำงานจะเปลี่ยนแปลงหลายอย่างพร้อมๆ กัน หนึ่ง ลองพิจารณารูปตัวอย่างดังต่อไปนี้

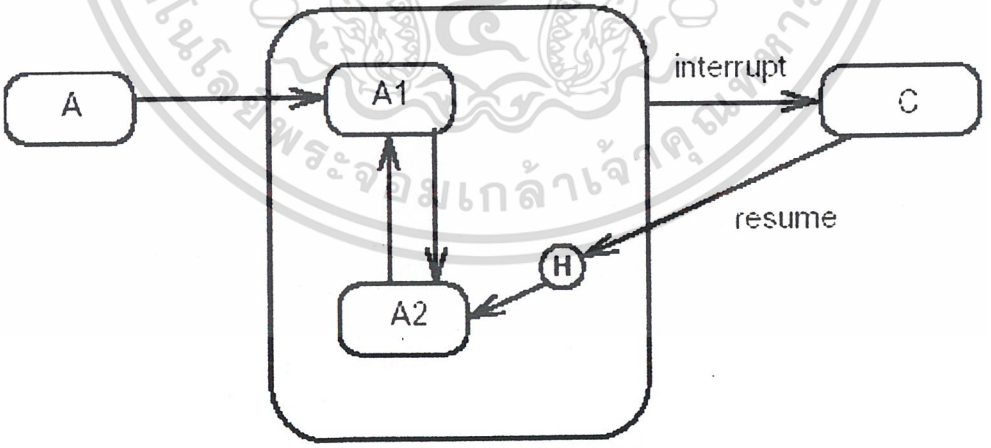


รูปที่ 2.19 สเตท Dialing ที่มีการแบ่งสถานะการทำงานย่อยแบบคอนเคอเรนท์ (Concurrent)

2.2.6.3 ฮิสทอรี่สเตท (History State)

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

สัญลักษณ์ของ ฮิสทอรี่สเตทจะเป็นรูปวงกลมที่มีอักษรตัว H อยู่ภายในวงกลม แสดงได้ดังรูปตัวอย่างต่อไปนี้



รูปที่ 2.20 ลักษณะการทำงานและสัญลักษณ์ที่ใช้ในฮิสทอรี่สเตท

สเตทไดอะแกรมแสดงให้เห็นถึงวงจรชีวิตของออบเจ็กต์ ระบบงานย่อย และระบบงานทั้งหมด โดยบ่งบอกว่าเหตุการณ์หรือภาวะต่างๆ มีผลต่อการเปลี่ยนแปลงของระบบงานอย่างไร ซึ่งสเตทไดอะแกรมหนึ่งอาจมีจุดเริ่มต้นและจุดสิ้นสุดได้หลายจุด

การใช้สเตทไดอะแกรมจะช่วยให้เราเข้าใจถึงพฤติกรรมของระบบมากขึ้น ในขณะที่ไดอะแกรมอื่นๆ อาจจะแสดงพฤติกรรมของระบบคืออะไร แต่จะไม่ลงไปในรายละเอียดมากนัก และเมื่อเราสามารถรู้ถึงพฤติกรรมของระบบได้ครบถ้วน เราก็จะสามารถตอบสนองต่อความต้องการของผู้ใช้งานได้อย่างตรงจุดประสงค์

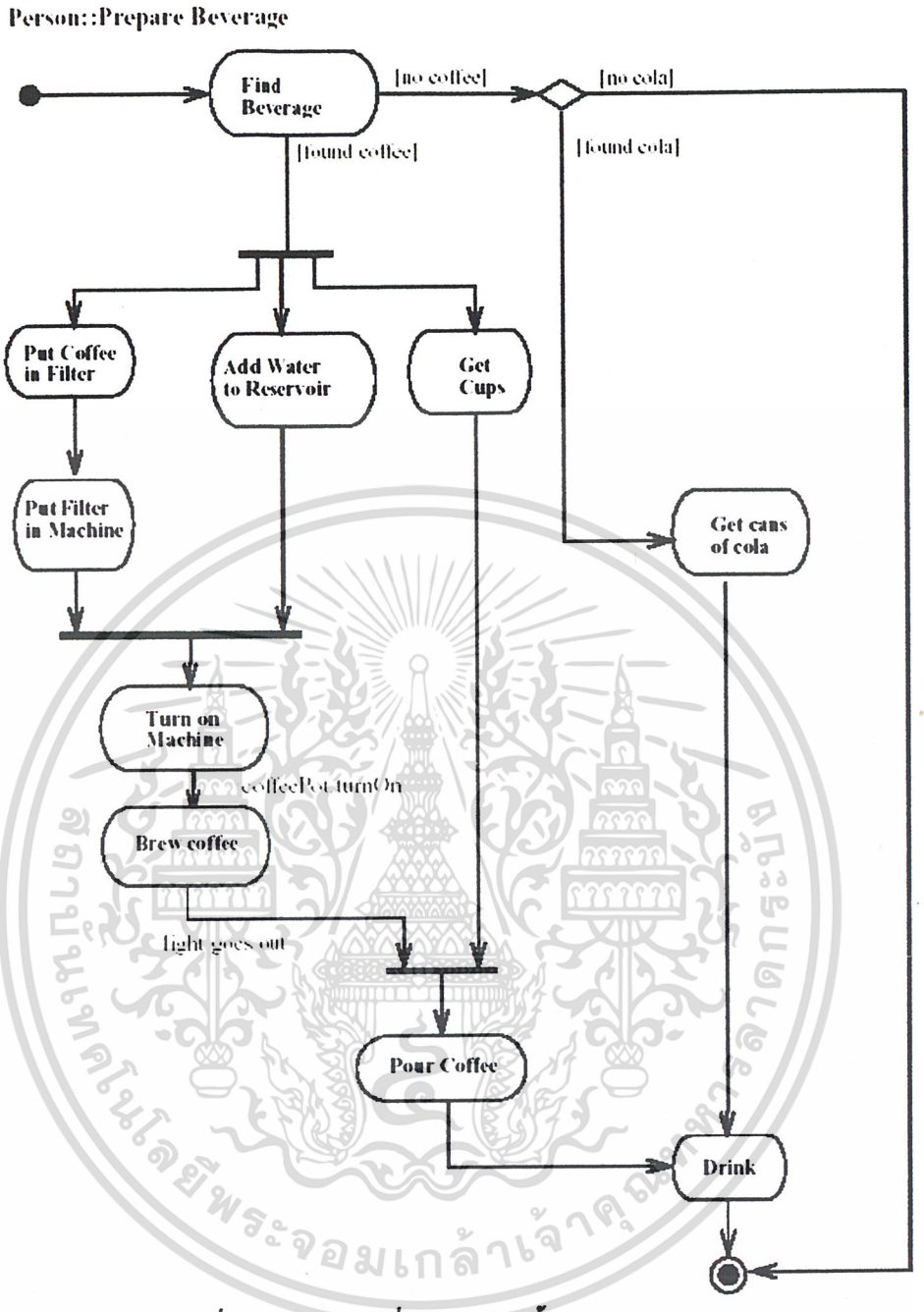
2.2.7 แอ็กทิวิตี้ ไดอะแกรม (Activity Diagram)

แอ็กทิวิตี้ไดอะแกรม จะมีลักษณะเดียวกันกับโฟลว์ชาร์ต(Flowchart) คือใช้สำหรับแสดงขั้นตอนการทำงาน แต่ต่างกันตรงที่แอ็กทิวิตี้ไดอะแกรม สามารถแสดงกิจกรรมที่ทำงานเป็นแบบขนานได้ เช่น งานที่ต้องแตกตัวเป็นหลายเธรด(Thread) และทำงานไปพร้อมๆ กัน โดยที่ขั้นตอนในการทำงานแต่ละขั้นตอนเราจะเรียกว่า "แอ็กทิวิตี้" (Activity) แอ็กทิวิตี้ไดอะแกรมสามารถนำมาใช้อธิบายยูสเคส, คลาส หรือโอเปอเรชั่น และการเปลี่ยนจากแอ็กทิวิตี้หนึ่งไปสู่อีกแอ็กทิวิตี้หนึ่งจะเกิดขึ้นโดยการเสร็จสิ้นการทำงานของแอ็กทิวิตี้แรก

แอ็กทิวิตี้ อาจเป็นลักษณะของการทำงานต่างๆ ได้แก่

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

แอ็กทิวิตี้ ไดอะแกรมจะต้องมีจุดเริ่มต้นกับจุดสิ้นสุด และในระหว่างจุดเริ่มต้นกับจุดสิ้นสุด ก็จะมีขั้นตอนหรือแอ็กทิวิตี้ต่างๆ ของระบบ ดังรูปตัวอย่างต่อไปนี้



รูปที่ 2.21 สัญลักษณ์ที่ใช้ในแอ็กทิวิตี้ไดอะแกรม

จากไดอะแกรมข้างต้น วงกลมทึบที่อยู่บนสุดแทนจุดเริ่มต้นของไดอะแกรมรูปสี่เหลี่ยมขอบมนจะแทนกิจกรรม หรือแอ็กทิวิตี้ของระบบที่เรากำลังสนใจ และวงกลมทึบที่อยู่ภายในวงกลมอีกทีจะแทนจุดสิ้นสุดของไดอะแกรมโดยปกติแล้วเราจะเขียนแอ็กทิวิตี้ไดอะแกรมโดยอ่านจากบนลงล่าง

รูปแบบการใช้แอ็กทิวิตี้ไดอะแกรม มีได้หลายแบบ ได้แก่

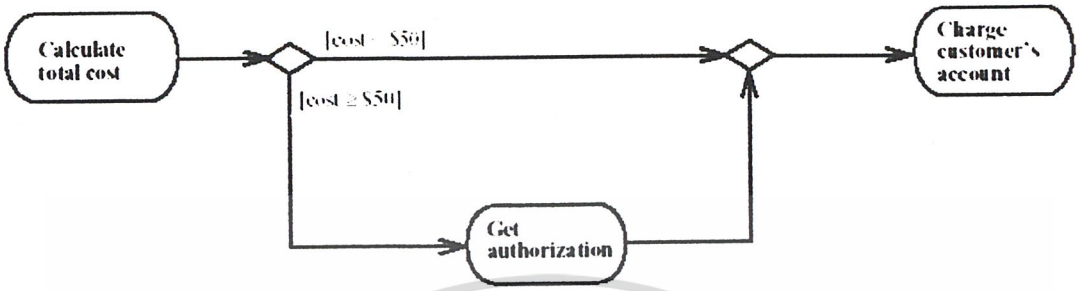
1. แบบมีทางเลือกให้ตัดสินใจ

เราสามารถกำหนดทางเลือกให้แก่แอ็กทิวิตี้ไดอะแกรมได้ 2 วิธีคือ

- ลากลูกศรของแต่ละทางเลือกไปยังแอ็กทิวิตี้ ผลลัพธ์ของทางเลือกโดยตรง

- ลากลูกศรให้ลูกศรของแต่ละทางเลือกผ่านรูปสี่เหลี่ยมขนมเปียกปูนเสียก่อน (ลักษณะเดียวกันที่เขียนโฟลว์ชาร์ต)

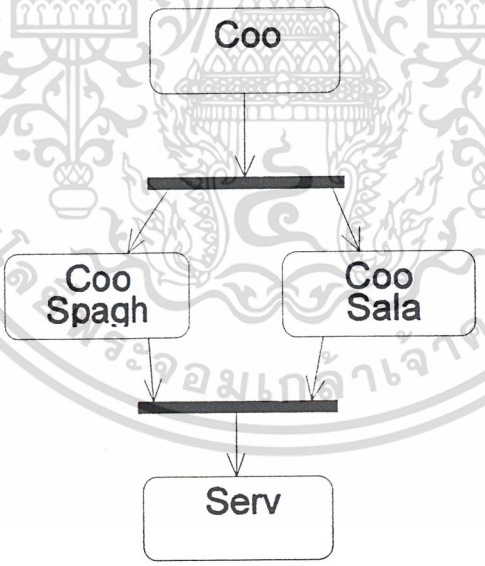
แสดงได้ดังรูปตัวอย่างต่อไปนี้



รูปที่ 2.22 การสร้างทางเลือกให้แก่แอ็กทิวิตี้ไดอะแกรม

2. แบบที่มีการทำงานพร้อมๆ กันหลายงาน

ในกรณีที่เรามีงานหลายงานที่มีการทำงานไปพร้อมๆ กัน ในการเขียนแอ็กทิวิตี้ไดอะแกรมเราจะใช้เส้นตรงแนวนอนเส้นหนามาเป็นสัญลักษณ์ที่ใช้จัดกลุ่มงานที่มีการทำพร้อมๆ กัน โดยมีลักษณะดังตัวอย่างต่อไปนี้



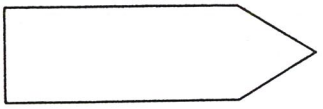
รูปที่ 2.23 สัญลักษณ์ที่ใช้ในการจัดกลุ่มงานที่ทำพร้อมกันในแอ็กทิวิตี้ไดอะแกรม

3. แอ็กทิวิตี้ไดอะแกรมสำหรับการส่งสัญญาณ

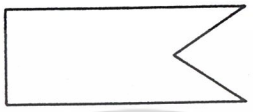
ในกระบวนการทำงานอาจเป็นไปได้ว่าจะมีการส่งสัญญาณบางอย่างในระหว่างการทำงาน เมื่อเกิดการส่ง-รับสัญญาณ เราก็จะเรียกว่าเกิด "แอ็กทิวิตี้" ขึ้นเช่นเดียวกัน

ในการเขียนแอ็กทิวิตี้ไดอะแกรมสำหรับการส่งสัญญาณ เราจะใช้รูปหลายเหลี่ยมแทน

แอ็กทิวิตี้ที่มีการส่งสัญญาณ แสดงได้ดังรูป

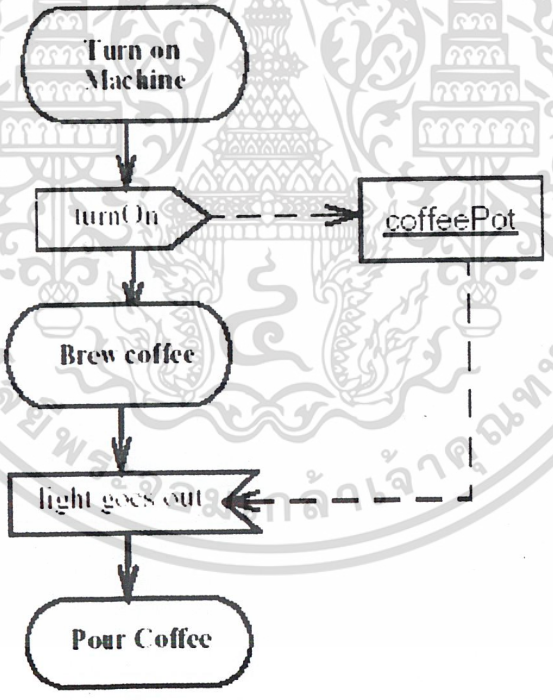


รูปที่ 2.24 สัญลักษณ์แทนเหตุการณ์ (Event) ที่เป็นอินพุท



รูปที่ 2.25 สัญลักษณ์แทนเหตุการณ์ (Event) ที่เป็นเอาท์พุท

รูปต่อไปเป็นตัวอย่างของการใช้แอ็กทิวิตี้ไดอะแกรมแสดงการส่งสัญญาณ โดยระบบที่สนใจคือการชงกาแฟโดยใช้กระติกน้ำร้อน



รูปที่ 2.26 ตัวอย่างการใช้สัญลักษณ์การส่งสัญญาณในแอ็กทิวิตี้ไดอะแกรม

2.2.7.1 สวิมเลน (Swimlanes)

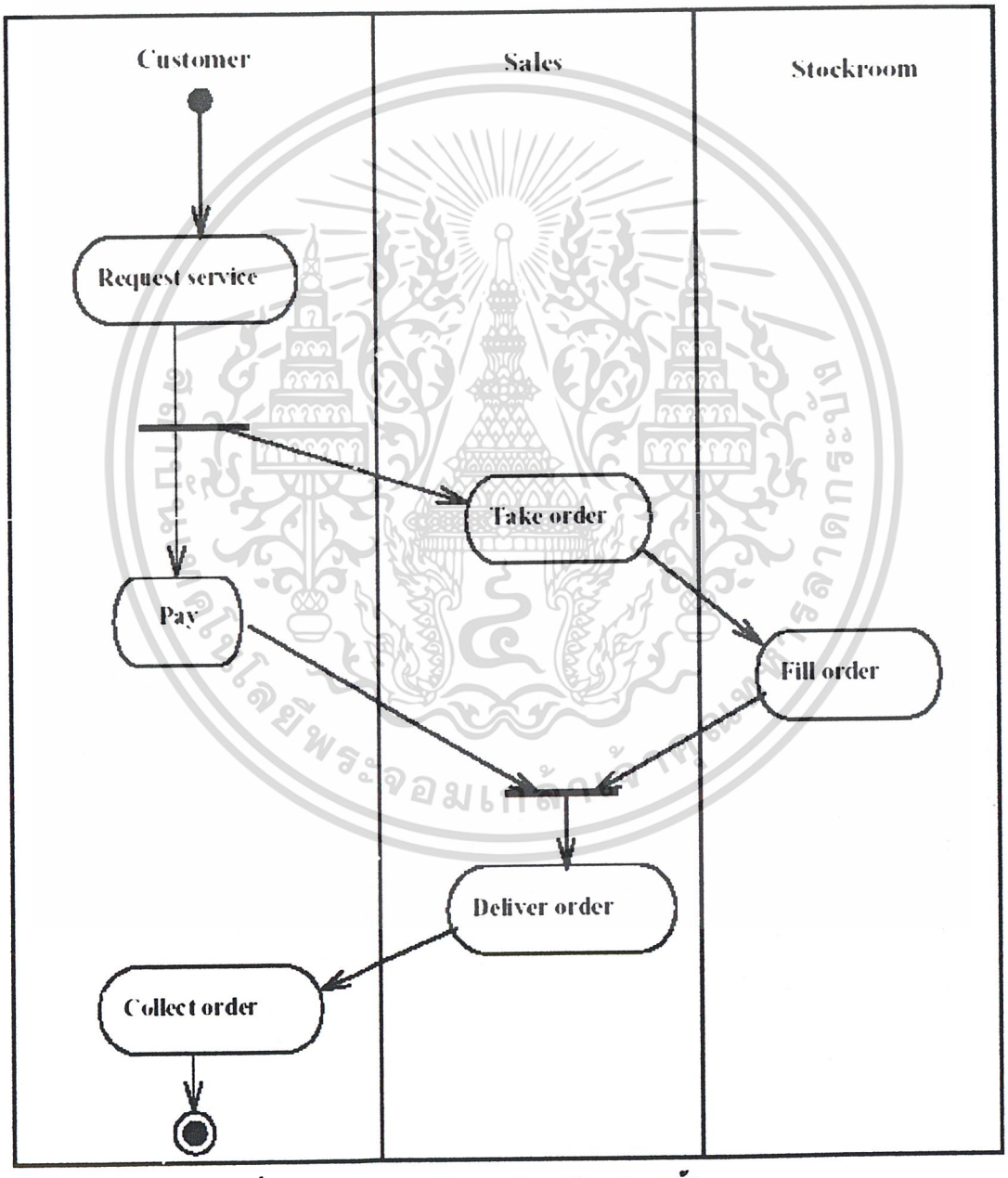
คุณลักษณะอย่างหนึ่งของการใช้แอ็กทิวิตี้ไดอะแกรม คือเราสามารถแสดงให้เห็นได้ว่าใครเป็นผู้มีหน้าที่รับผิดชอบในแต่ละแอ็กทิวิตี้ในกระบวนการทำงานหนึ่งๆ

หลักการของการแสดงหน้าที่ความรับผิดชอบของแต่ละแอ็กทิวิตี้ เราจะทำโดยการแบ่งกลุ่มของ

การรับผิดชอบเป็นกลุ่มๆ ซึ่งแอ็กทิวิตี้ไดอะแกรม มีลักษณะการแบ่งกลุ่มการรับผิดชอบด้วย เรียกกลไกดังกล่าวว่า “สวิมเลน”

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

แอ็กทิวิตี้หนึ่งๆ จะอยู่ภายในหนึ่งสวิมเลนเท่านั้น แต่การติดต่อหรือส่งผ่านระหว่างแอ็กทิวิตี้สามารถเกิดขึ้นข้ามจากสวิมเลนหนึ่งไปยังอีกสวิมเลนหนึ่งได้ ดังรูป



รูปที่ 2.27 ตัวอย่างการใช้งานสวิมเลนในแอ็กทิวิตี้ไดอะแกรม

จะเห็นว่าแอ็กทิวิตี้ไดอะแกรมมีลักษณะคล้ายคลึงกับโฟลว์ชาร์ตมาก โดยจะแสดงถึงลำดับขั้นตอนการทำงาน จุดที่มีการตัดสินใจ และการแตกการทำงานเป็นส่วนย่อยๆ ด้วยลักษณะดังกล่าวแอ็กทิวิตี้ไดอะแกรมจึงมีประโยชน์มากสำหรับใช้อธิบายถึงการทำงานของออบเจ็กต์ต่างๆ รวมไปถึงกระบวนการทำงานต่างๆ ด้วย

เราอาจสงสัยว่าทั้งแอ็กทิวิตี้ไดอะแกรมและสเตทไดอะแกรมต่างก็ใช้แสดงขั้นตอนการทำงานซึ่งระบบงานเกิดการเปลี่ยนแปลง ดังนั้นเราจะมีหลักอย่างไรว่าเมื่อไรควรจะใช้แอ็กทิวิตี้ไดอะแกรมหรือเมื่อไรเราควรใช้สเตทไดอะแกรม

ข้อแตกต่างของทั้งสองไดอะแกรม อยู่ที่ลักษณะของการเปลี่ยนแปลงของระบบ ถ้าเป็นการเปลี่ยนแปลงหรือเหตุการณ์ต่างๆ อันเป็นผลมาจากการกระทำภายในระบบเองซึ่งมีลักษณะต่อเนื่องกันและมักมีจุดที่ระบบงานต้องมีการตัดสินใจ ให้ใช้แอ็กทิวิตี้ไดอะแกรมแต่ถ้าเป็นการเปลี่ยนแปลงหรือการกระทำที่เกิดขึ้นในลักษณะไม่ต่อเนื่องกัน ให้ใช้สเตทไดอะแกรม

เมื่อเราสร้างแอ็กทิวิตี้ไดอะแกรมขึ้นมา เราต้องไม่ลืมว่าเป็นเพียงโมเดลหนึ่งของภาษา UML ที่พยายามสร้างมุมมองแบบไดนามิกของระบบ ดังนั้น จึงไม่มีแอ็กทิวิตี้ไดอะแกรมใดจะสามารถอธิบายทุกๆ อย่างของระบบได้ภายในไดอะแกรมเดียว เราจำเป็นต้องใช้แอ็กทิวิตี้ไดอะแกรมมาช่วยมากกว่าหนึ่งไดอะแกรม จึงจะเห็นกระบวนการทำงานได้อย่างครบถ้วน

แอ็กทิวิตี้ไดอะแกรมเป็นไดอะแกรมที่ค่อนข้างสร้างแล้วเข้าใจได้ง่าย อย่างไรก็ตาม แอ็กทิวิตี้ไดอะแกรมที่ดีควรมีคุณสมบัติต่อไปนี้

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

บทที่ 3
NIAM

3.1 บทนำ

การออกแบบฐานข้อมูล โดยใช้วิธีไนแอม (NIAM:Nijssen's Information Analysis Methodology) เป็นวิธีการในการออกแบบฐานข้อมูล โดยการแสดงความหมาย, ความสัมพันธ์ และข้อจำกัดต่างๆ ของข้อมูลด้วยแบบจำลองข้อมูลที่ประกอบไปด้วยสัญลักษณ์ต่างๆ เนื่องจากแนวคิดที่ให้ Conceptual Schema มีพื้นฐานมาจากโครงสร้างภาษาธรรมชาติ ใช้รูปประโยคที่มีประธาน กริยา กรรม วิธีแสดงรูปแบบความสัมพันธ์ของข้อมูลและข้อจำกัดของข้อมูลได้อย่างชัดเจน นอกจากนั้นยังสามารถแปลง Conceptual Schema เป็น Relational Database Schema ซึ่งจะอยู่ในรูปของ Fifth Normal Form ได้โดยตรง และเนื่องจากวิธีการนี้ใช้รูปสัญลักษณ์ที่แสดงความสัมพันธ์ของข้อมูลและง่ายต่อการเข้าใจ ดังนั้นจึงสะดวกในการออกแบบฐานข้อมูลของระบบงานใหญ่ๆ

3.2 ความหมายของ NIAM และการใช้

ไนแอมมีขั้นตอนในการออกแบบอยู่ 9 ขั้นตอน (CSDP 9 steps : Conceptual Schema Design Procedure)

1. กำหนดขอบเขตของงาน (Universe of Discourse : UOD) และความจริงที่เกิดขึ้นภายในขอบเขตของงานที่กำหนดไว้
2. วาด Conceptual Schema Diagram โดยคร่าวๆ จากความจริงในขอบเขตของงาน
3. จัดรูปของ Schema ให้เป็นระเบียบและหาชนิดความจริงที่ได้รับข้อมูลมาจากชนิดความจริง
4. เติมสัญลักษณ์แสดง Uniqueness constraints
5. ตรวจสอบความถูกต้องของชนิดความจริง
6. เติมสัญลักษณ์แสดง Lexical, Mandatory Role, Subtype constraints
7. ตรวจสอบ Unique Identifier ของแต่ละชนิดเอนติตี
8. เติมสัญลักษณ์แสดง Equality, Exclusion, Subset constraints
9. ตรวจสอบความสมบรูณ์ของ Conceptual Schema ว่าต้องสอดคล้องกับตัวอย่างข้อมูลและไม่มีความซ้ำซ้อนของข้อมูล

3.3 ส่วนประกอบพื้นฐานของไนแอม

ส่วนประกอบพื้นฐาน ประกอบไปด้วย

ชนิดเอนติตี้ (Entity Type) หมายถึง เซตของสิ่งที่สนใจทั้งที่อยู่ในรูปของนามธรรม หรือรูปธรรม ซึ่งอาจเป็นสิ่งที่จับต้องได้หรือไม่ได้ เช่น คน, ภาควิชา, บริษัท, รถยนต์ เป็นต้น

ชนิดเลเบิล (Lable Type, Value Type) หมายถึง เซตของสิ่งที่ใช้บ่งบอกความแตกต่าง หรือชื่อของแต่ละเอนติตี้ที่กำหนด เช่น ชื่อ, นามสกุล, รหัสประจำตัว, ทะเบียนรถยนต์ เป็นต้น

บทบาท (Role) หมายถึง ความสัมพันธ์ที่เกี่ยวข้องกับชนิดเอนติตี้ที่สัมผัสอยู่ เช่น เอนติตี้นักศึกษา แสดงบทบาท เป็นผู้ลงทะเบียนเรียนในวิชานั้นๆ เป็นต้น

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

ชนิดอ้างอิง (Reference Type) หมายถึง เซตของความสัมพันธ์ระหว่างสมาชิกของชนิดเอนติตี้กับสมาชิกของชนิดเลเบิลที่มีอยู่

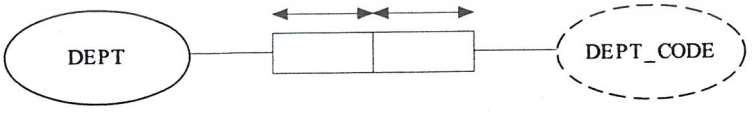
ชนิดความจริงแบบเนสท์ (Nested Fact Type) หมายถึง ชนิดเอนติตี้ที่แสดงความสัมพันธ์ในการกำหนดกลุ่มของชนิดความจริงที่มีตั้งแต่ 2 บทบาทขึ้นไป

กฎข้อบังคับความถูกต้องของข้อมูล (Integrity Constraints) หมายถึง สิ่งที่ใช้แสดงกฎที่ใช้ในการบังคับควบคุมความถูกต้องของข้อมูล

สัญลักษณ์และตัวอย่างการใช้ส่วนประกอบพื้นฐานของแบบจำลองไนแอมแสดงไว้ดังรูป

รูปที่ 3.1 สัญลักษณ์ของชนิดเอนติตีภาควิชา

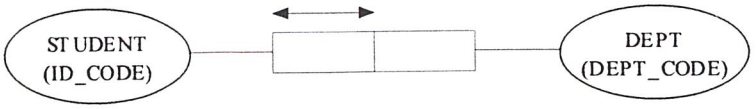
รูปที่ 3.2 สัญลักษณ์ของชนิดเฉเบิลรหัสภาควิชา



รูปที่ 3.3 ความสัมพันธ์อ้างอิงแบบ one to one

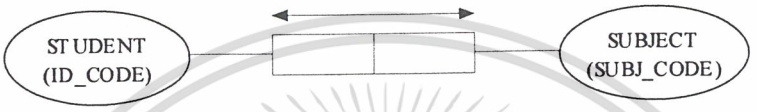
หมายความว่า ภาควิชาใดๆ จะมีรหัสภาควิชาเพียงรหัสเดียวเท่านั้นและไม่ซ้ำกับภาควิชาอื่น

รูปที่ 3.4 การเขียนความสัมพันธ์อ้างอิงแบบ one to one อย่างย่อ



รูปที่ 3.5 ความจริงแบบ many to one

หมายความว่านักศึกษาหนึ่งคนจะสังกัดภาควิชาได้เพียงภาคเดียว แต่ภาควิชาใดๆ สามารถมีนักศึกษาในสังกัดได้มากกว่าหนึ่งคน



รูปที่ 3.6 ความจริงแบบ many to many

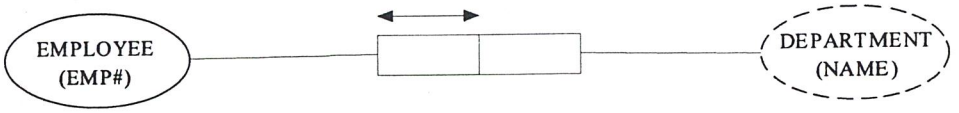
หมายความว่านักศึกษาหนึ่งคนสามารถลงทะเบียนเรียนได้หลายวิชาและแต่ละวิชาที่เปิดสอนสามารถรับจำนวนได้มากกว่าหนึ่งคนแต่นักศึกษาที่ลงทะเบียนเรียนวิชาใดๆ แล้ว จะลงทะเบียนซ้ำวิชาเดิมไม่ได้ (ตัวอย่างนี้ไม่เป็นจริงในทางปฏิบัติ)

ชนิดเอนติตีเป็นเซต (set) ซึ่งมีสมาชิกเป็นตัวอย่างเอนติตี (Entity Instance) เช่น ภาควิชา A (ภาควิชาโทรคมนาคม) ภาควิชา B (ภาควิชาวิศวกรรมคอมพิวเตอร์) เป็นตัวอย่างเอนติตีของชนิดเอนติตีภาควิชา

เครื่องหมายความสัมพันธ์ที่เป็นส่วนเชื่อมโยงระหว่างชนิดเอนติตี และชนิดเอนติตีหรือชนิดเลเบิลนั้น เรียกว่า บทบาท (role) จะเขียนความหมายของบทบาทนั้นไว้ภายในหรือข้างๆ สัญลักษณ์ของมัน

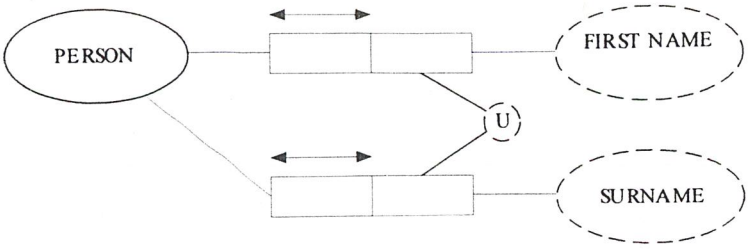
การแปลงข้อมูลที่วิเคราะห์มาให้อยู่ในรูปแบบจำลอง ก่อนอื่นต้องนำข้อมูลมากำหนดเป็นชนิดเอนติตี้และเลเบิลให้เรียบร้อยเสียก่อน จึงนำชนิดเอนติตีที่ได้มาเขียนเป็นประโยคความจริงมูลฐาน (Elementary Fact) แล้วเอาความจริงทั้งหมดที่ได้มาเขียนเป็นแบบจำลอง และเติมข้อจำกัดต่างๆ ลง ไปตามความเป็นจริงในขอบเขตของงาน

ตัวอย่างการใช้ข้อจำกัดต่างๆ แสดงไว้ดังรูปที่ 3.7 และรูปที่ 3.8



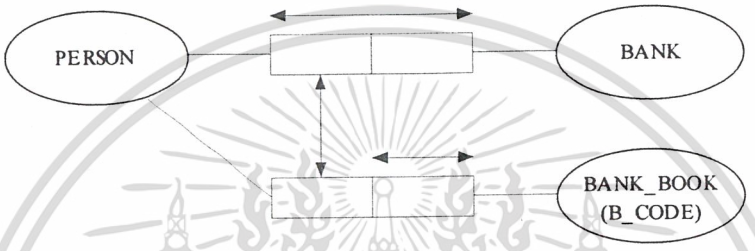
รูปที่ 3.7 การใช้ intra fact type uniqueness constraint

หมายความว่า Employee หนึ่งคนจะมีที่ทำงานได้ที่เดียวเท่านั้น



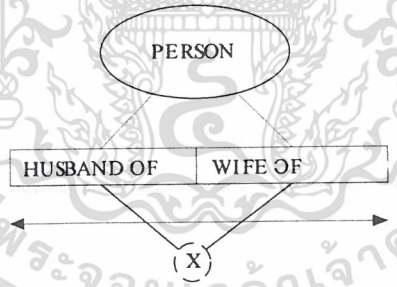
รูปที่ 3.8 การใช้ inter fact type uniqueness constraint

หมายความว่า บุคคลหนึ่งจะมีชื่อ 1 ชื่อ นามสกุล 1 นามสกุล ชื่อของบางคนอาจจะซ้ำกัน และนามสกุลจะต้องไม่ซ้ำกัน



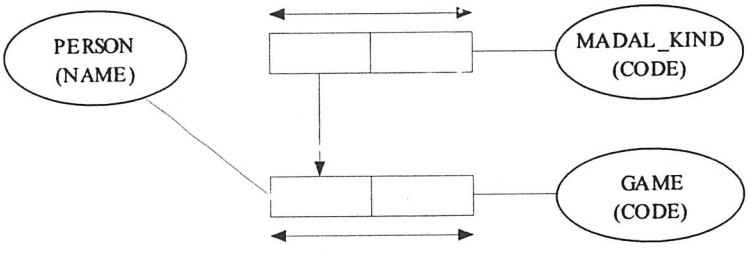
รูปที่ 3.9 การใช้ equality constraint

หมายความว่า ถ้าบุคคลหนึ่งเป็นลูกค้าของธนาคารใดแล้ว บุคคลนั้นต้องมีสมุดบัญชีของธนาคารนั้นด้วย หรือในทางกลับกัน ถ้าบุคคลใดมีสมุดบัญชีของธนาคารใดแล้ว ก็ต้องเป็นลูกค้าของธนาคารนั้นด้วย



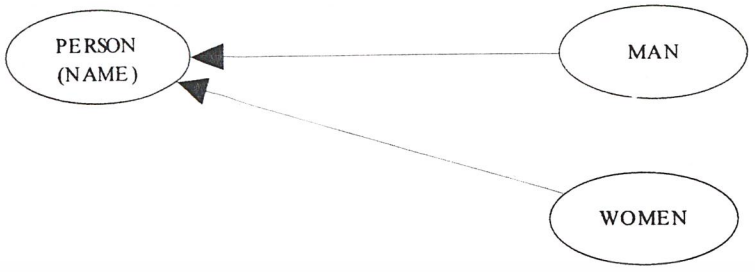
รูปที่ 3.10 การใช้ exclusion constraint

หมายความว่า บุคคลใดเป็นภรรยาของอีกบุคคลหนึ่งแล้ว บุคคลนั้นต้องไม่เป็นสามีของบุคคลใดๆ ในทางกลับกัน บุคคลที่เป็นสามีของบุคคลหนึ่งแล้วจะต้องไม่เป็นภรรยาของบุคคลใดด้วย



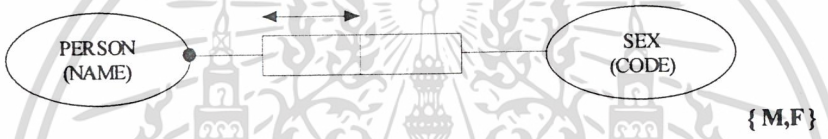
รูปที่ 3.11 การใช้ subset constraint

หมายความว่า บุคคลที่ชนะกีฬาการแข่งขันกีฬาทุกคนจะต้องเป็นบุคคลที่เล่นกีฬา แต่บุคคลที่เล่นกีฬาไม่จำเป็นต้องชนะการแข่งขันกีฬาทุกคน



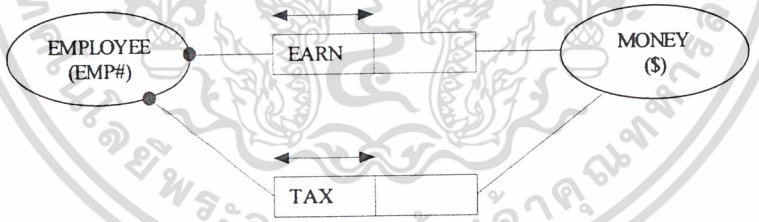
รูปที่ 3.12 การใช้ subtype constraint

หมายความว่า ตัวอย่างเอนติตีทุกตัวของชนิดเอนติตีผู้ชาย และชนิดเอนติตีผู้หญิง ต่างก็เป็นสมาชิกของชนิดเอนติตีบุคคล



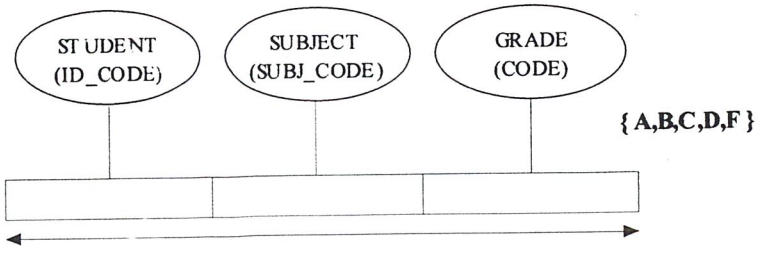
รูปที่ 3.13 การใช้ mandatory constraint, lexical constraint

หมายความว่าบุคคลทุกคนต้องมีเพศและสมาชิกของชนิดเอนติตีเพศมีเพียง M (Male) และ F (Female) เท่านั้น

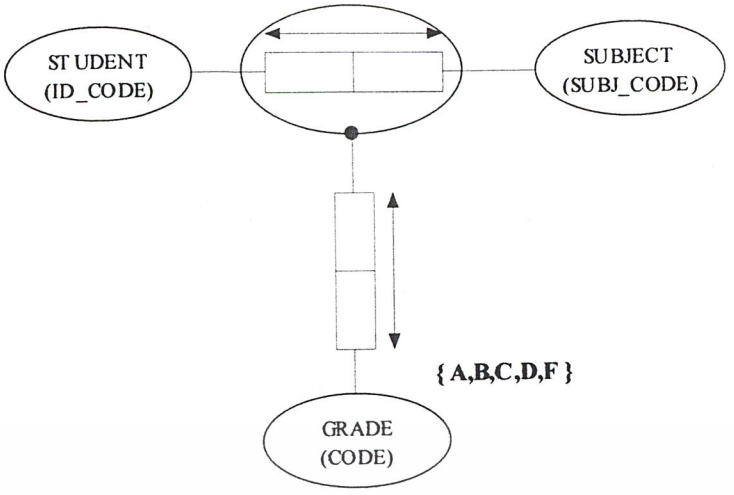


รูปที่ 3.14 Employee ทุกคนต้องมีรายได้ และทุกคนต้องเสียภาษี

นอกจากตัวอย่างความจริงแบบ Binary Fact Type ที่แสดงไว้ข้างต้นแล้ว ยังมีตัวอย่างชนิดอื่นอีก เช่น



รูปที่ 3.15 Ternary Fact Type หมายความว่า ข้อมูลการเรียนของนักศึกษาทุกคนจะต้องมีทั้งรหัสวิชาและเกรด



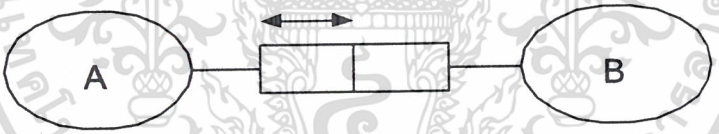
รูปที่ 3.16 Nested Fact Type มีความหมายเหมือนรูปที่ 3.15

3.4 กฎข้อบังคับความถูกต้องของข้อมูลที่ใช้ในแบบจำลองระดับแนวคิดไนแอม

3.4.1 Intra fact type constraints (internal uniqueness constraints)

เป็นกฎข้อบังคับความถูกต้อง เพื่อทำการกำหนดบทบาทที่ใช้แสดงความสัมพันธ์ระหว่างสมาชิกของชนิดเอนติตี้หนึ่งกับสมาชิกของชนิดเอนติตี้อื่น หรือกับสมาชิกของชนิดเลเบิล โดยสามารถแบ่งเป็นรูปแบบต่าง ๆ ได้ดังต่อไปนี้

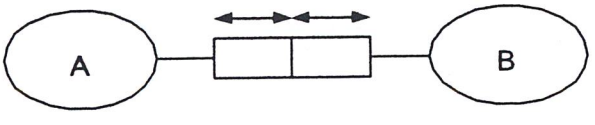
การกำหนดความสัมพันธ์แบบหนึ่งต่อหลายหน่วย (one to many relationship) ซึ่งสามารถแสดงบนแผนภาพได้ดังรูป



รูปที่ 3.17 ความสัมพันธ์แบบหนึ่งต่อหลายหน่วย

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

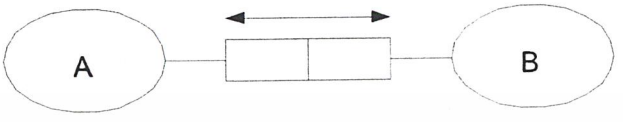
การกำหนดความสัมพันธ์แบบหนึ่งต่อหนึ่งหน่วย (one to one relationship) ซึ่งสามารถแสดงบนแผนภาพได้ดังรูป



รูปที่ 3.18 ความสัมพันธ์แบบหนึ่งต่อหนึ่งหน่วย

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

การกำหนดความสัมพันธ์แบบหลายต่อหลายหน่วย (many to many relationship) ซึ่งสามารถแสดงบนแผนภาพได้ดังรูป

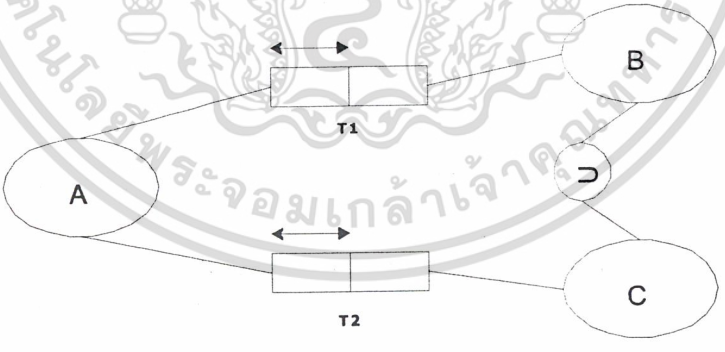


รูปที่ 3.19 ความสัมพันธ์แบบหลายหน่วยต่อหลายหน่วย

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

3.4.2 Inter fact type uniqueness constraints (external uniqueness constraints)

เป็นกฎข้อบังคับความถูกต้องที่แสดงให้เห็นว่าชนิดเอนติตี้ใดๆ มีความสัมพันธ์กับชนิดเลเบิล หรือ ชนิดเอนติตี้ ได้มากกว่าหนึ่ง โดยในทางกลับกัน ชนิดเลเบิล หรือ ชนิดเอนติตี้ เหล่านั้น สามารถบ่งบอกถึงลักษณะเฉพาะของชนิดเอนติตี้นั้นได้ดังแสดงในแผนภาพดังนี้

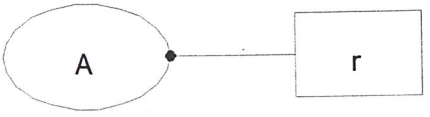


รูปที่ 3.20 Inter fact type uniqueness constrains

ลักษณะเช่นนี้สามารถแสดงได้ว่ากฎข้อบังคับความถูกต้องจะทำการควบคุม หากนำ T1 join T2 แล้วผลที่ได้ BC จะไม่เกิดความซ้ำซ้อนกันขึ้น เช่น คนหนึ่งคนอาจมีชื่อหรือนามสกุลซ้ำกันได้ แต่ถ้ารวมทั้งชื่อและนามสกุลแล้วจะไม่เกิดความซ้ำซ้อนดังนั้นจะสามารถบ่งบอกได้ว่าเป็นการระบุถึงคนใด

3.4.3 Mandatory role constraints

เป็นกฎข้อบังคับความถูกต้องที่ใช้ในการควบคุมเพื่อแสดงให้เห็นถึงการมีอยู่ของข้อมูลว่าต้องมีการบันทึกข้อมูลทุกครั้งที่เกิดมีความสัมพันธ์เกิดขึ้น สามารถแสดงได้ในแผนภาพดังนี้

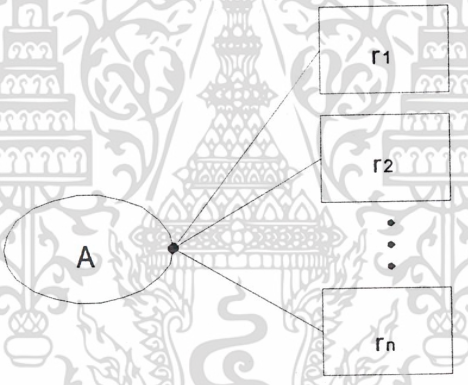


รูปที่ 3.21 Mandatory role constraints

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

3.4.4 Inclusion mandatory role constraints

เป็นกฎข้อบังคับความถูกต้องที่แสดงให้เห็นถึงทางเลือกของบทบาทในกลุ่มของความสัมพันธ์ ที่มีอยู่ว่าต้องมีการบันทึกข้อมูลอย่างน้อยบทบาทใดบทบาทหนึ่งของชนิดเอนติตี้นั้น ดังแสดงในแผนภาพได้ดังนี้

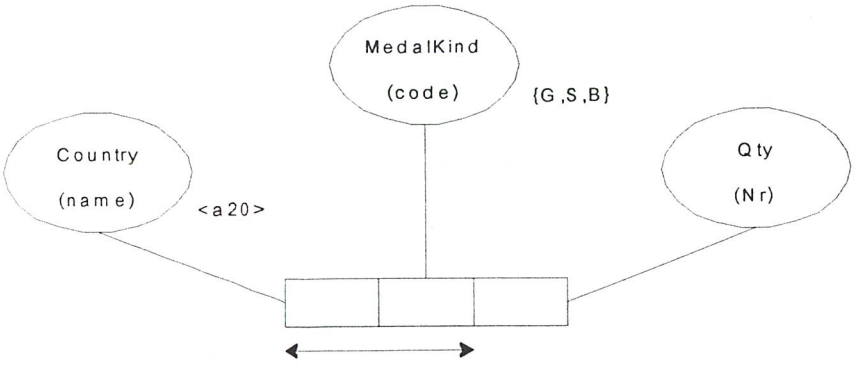


รูปที่ 3.22 Inclusion mandatory role constraints

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

3.4.5 Entity type constraints (Value constraints)

เป็นกฎข้อบังคับความถูกต้อง ที่ใช้ในการกำหนดค่าของสมาชิกภายในเซ็ตของข้อมูลที่เป็นไปได้ของชนิดเลเบิล หรือ ชนิดเอนติตี้หนึ่งๆ รวมไปถึงการกำหนดชนิดของข้อมูลในเซ็ตด้วย ดังแสดงได้ในแผนภาพดังนี้

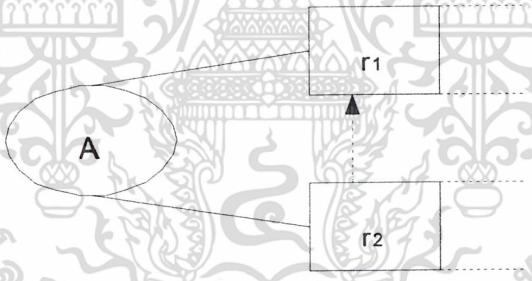


รูปที่ 3.23 Entity type constraints

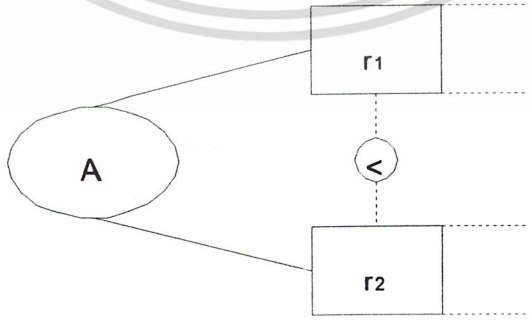
จากภาพนั้นจะมีกฎข้อคับความถูกต้องของข้อมูลเพื่อทำการระบุชนิดของเหรียญรางวัลในการแข่งขันกีฬาสามารถแยกออกได้เป็น เหรียญทองแดง, เหรียญเงิน, เหรียญทอง และระบุถึงจำนวนของเหรียญรางวัลที่ได้ว่าต้องอยู่ในช่วง 1 ถึง 200 เหรียญ รวมทั้งยังสามารถระบุชนิดของข้อมูลได้ด้วย ดังที่แสดงให้เห็นว่าชื่อประเทศนั้นกำหนดให้จัดเก็บได้มากที่สุด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

3.4.6 Subset constraint

เป็นกฎข้อบังคับความถูกต้องของข้อมูล ที่แสดงความสัมพันธ์ที่เป็นส่วนหนึ่งของความสัมพันธ์ที่มีอยู่ แต่จะมีลักษณะความสัมพันธ์ไปในทางเดียวดังแสดงความสัมพันธ์ได้โดยใช้สัญลักษณ์คือ A⟶B ซึ่งสามารถแสดงในแผนภาพได้ดังนี้



รูปที่ 3.24 Subset constraints



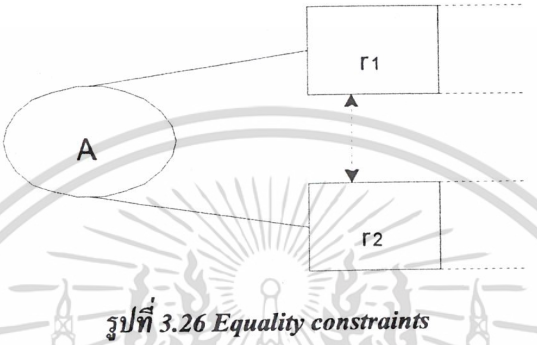
รูปที่ 3.25 Subset constraints (ต่อ)

ลักษณะดังกล่าวนี้แสดงให้เห็นกฎข้อบังคับความถูกต้องของข้อมูลว่า สมาชิกแต่ละตัวของชนิดเอนติตี้ A หากมีการบันทึกความสัมพันธ์ r2 แล้ว ต้องมีการบันทึกความสัมพันธ์ r1 ด้วย แต่ในทางกลับกัน

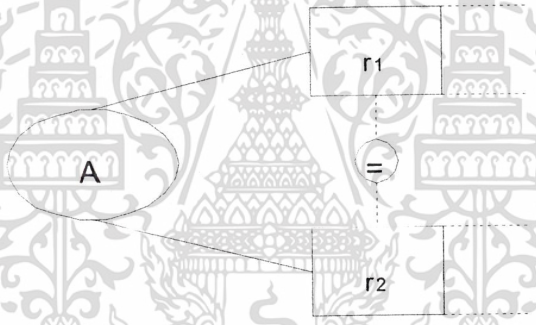
สมาชิกแต่ละตัวของชนิดเอนติตี้ A หากมีการบันทึกความสัมพันธ์ r1 แล้วไม่จำเป็นต้องมีการบันทึกความสัมพันธ์ r2 ก็ได้ เช่น บุคคลที่ชนะการแข่งขันกีฬา แสดงว่าต้องเป็นนักกีฬา แต่ผู้ที่เป็นนักกีฬาไม่จำเป็นต้องเป็นผู้ชนะการแข่งขันทุกคน

3.4.7 Equality constraints

เป็นกฎข้อบังคับความถูกต้องที่แสดงให้เห็นว่า ชนิดเอนติตี้เหล่านั้นจะต้องมีการถูกบันทึกข้อมูลควบคู่กันเสมอไป ใช้สัญลักษณ์แสดงความสัมพันธ์ได้คือ A↔B ซึ่งสามารถแสดงในแผนภาพได้ดังนี้



รูปที่ 3.26 Equality constraints

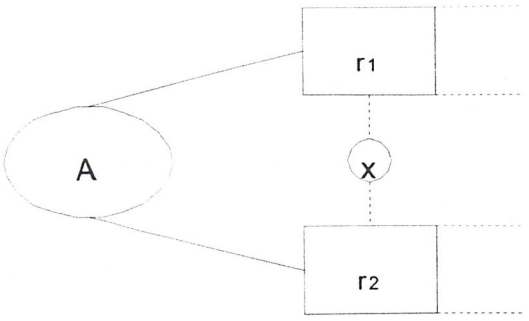


รูปที่ 3.27 Equality constraints (ต่อ)

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

3.4.8 Exclusion constraints

เป็นกฎข้อบังคับความถูกต้องที่มีลักษณะตรงข้ามกับ Equality constraints คือ แสดงความสัมพันธ์ที่ระบุว่าหากมีความสัมพันธ์แบบหนึ่งเกิดขึ้นจะต้องไม่มีความสัมพันธ์อีกแบบหนึ่งเกิดขึ้นโดยเด็ดขาด ซึ่งสามารถแสดงในแผนภาพได้ดังนี้

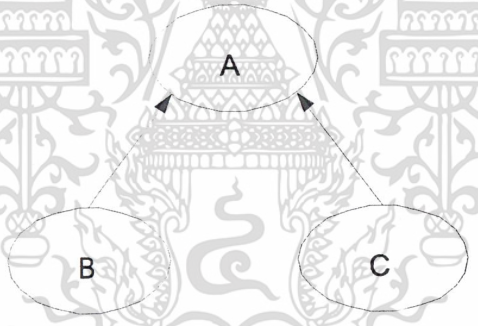


รูปที่ 3.28 Exclusion constraints

ลักษณะดังกล่าวนี้ แสดงให้เห็นกฎข้อบังคับความถูกต้องว่า หากมีการบันทึกข้อมูลความสัมพันธ์ r1 ของสมาชิกของชนิดเอนติตี้ A ใด จะต้องไม่มีการบันทึกข้อมูลความสัมพันธ์ r2 ของสมาชิกของชนิดเอนติตี้ A นั้นโดยเด็ดขาด เช่น ถ้าบุคคลใดถูกเลือกให้เป็นกรรมการในการตัดสินเกมส์นั้น บุคคลนั้นจะไม่มีสิทธ์เป็นผู้แข่งขันในเกมส์อย่างเด็ดขาด

3.4.9 Subtype constraints

เป็นกฎข้อบังคับความถูกต้องที่ระบุถึงการแบ่งกลุ่มของสมาชิกของชนิดเอนติตี้ที่มีอยู่อย่างชัดเจน ซึ่งสมาชิกของชนิดเอนติตี้ที่แบ่งแยกออกจากชนิดเอนติตี้ที่เป็น Super Type นั้น จะต้องมีลักษณะและคุณสมบัติที่แตกต่างกันอย่างชัดเจน ดังสามารถแสดงในแผนภาพได้ดังนี้

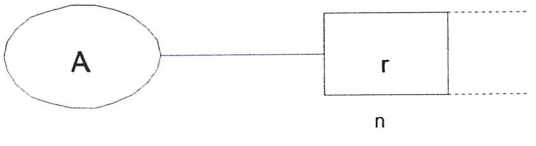


รูปที่ 3.29 Subtype constraints

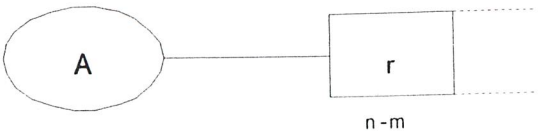
ลักษณะดังกล่าวนี้แสดงให้เห็นว่า สมาชิกของชนิดเอนติตี้ A โดยจะเรียกว่า Super Type นั้น สามารถแบ่งออกเป็น 2 กลุ่มได้คือ กลุ่มของชนิดเอนติตี้ B และกลุ่มของชนิดเอนติตี้ C ซึ่งเรียกว่า Subtype เช่น ชนิดเอนติตี้ของบุคคล สามารถแบ่งออกเป็น Subtype ผู้ชาย และ ผู้หญิง ได้

3.4.10 Occurrence frequency constraints

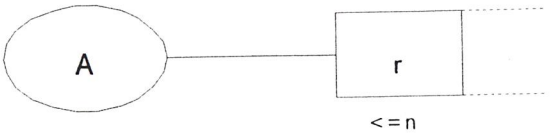
เป็นกฎข้อบังคับความถูกต้องของข้อมูลที่ใช้ในการระบุจำนวนครั้งที่สมาชิกของชนิดเอนติตี้ใดๆ จะสามารถแสดงบทบาทใดบทบาทหนึ่งได้ ซึ่งสามารถแสดงในแผนภาพได้ดังนี้



(a)



(b)



(c)

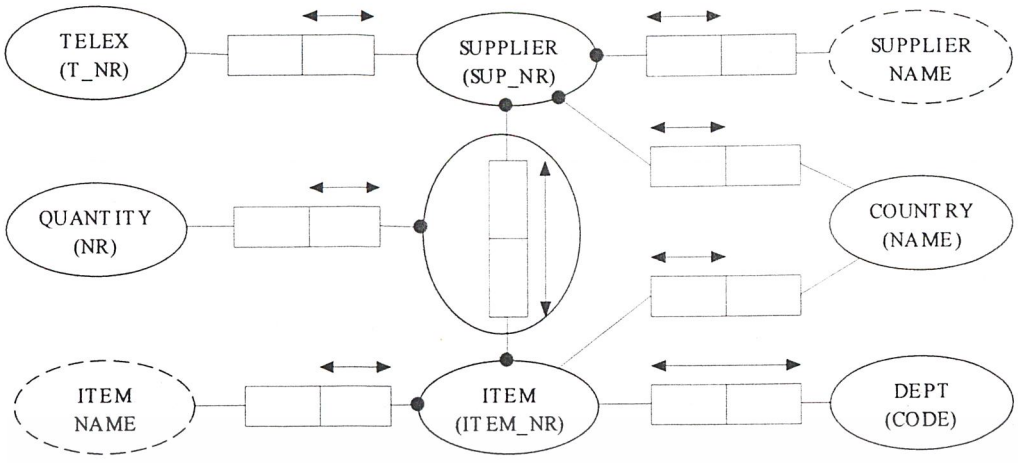
รูปที่ 3.30 Occurrence frequency constraints

จากรูปที่ 3.30 (a) เป็นการแสดงกฎข้อบังคับความถูกต้องของข้อมูลโดยที่แต่ละชนิดเอนติตี้ A จะมีการแสดงบทบาทในคอลัมน์ r เป็นจำนวน n ครั้ง จากรูปที่ 3.30 (b) เป็นการแสดงกฎข้อบังคับความถูกต้องของข้อมูลโดยที่แต่ละ ชนิดเอนติตี้ A ในการแสดงบทบาทในคอลัมน์ r ได้อย่างน้อยที่สุด n ครั้ง และมากที่สุด m ครั้ง และจากรูปที่ 3.30 (c) เป็นการแสดงกฎข้อบังคับความถูกต้องของข้อมูลโดยที่แต่ละชนิดเอนติตี้ A ในการแสดงบทบาทในคอลัมน์ r ได้อย่างน้อยที่สุด n ครั้ง เช่น ชมรมใดชมรมหนึ่งจะต้องมีสมาชิกอย่างน้อย 20 คน แต่จำนวนสูงสุดที่รับได้ต้องไม่เกิน 200 คน เป็นต้น

3.5 The Optimal Normal Form algorithm (ONF อัลกอริธึม)

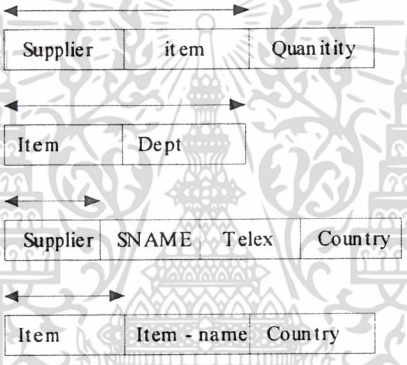
เป็นวิธีการจัดกลุ่มความจริงใน Conceptual Schema ให้เป็น Relational Database Schema โดยมีหลักการโดยสรุปดังต่อไปนี้

1. สร้าง 1 รีเลชัน สำหรับชนิดความจริงแบบไบนารีซึ่งมีความสัมพันธ์แบบ many to many โดยที่ Unique Identifier ของชนิดเอนติตีที่เกี่ยวข้องทั้งหมดเป็น Primary Key
2. สร้าง 1 รีเลชัน สำหรับแต่ละชนิดความจริงแบบ n-ary โดยที่ Unique Identifier ของชนิดเอนติตีซึ่งมี role ถูกบังคับด้วย Uniqueness Constraint เดียวกันเป็น Candidate Key
3. พิจารณาชนิดเอนติตีที่เกี่ยวข้องกับความจริงแบบไบนารีที่มีความสัมพันธ์เป็นแบบ one to one หรือ many to one โดยที่ role ของชนิดเอนติตีเหล่านั้นถูกบังคับด้วย Uniqueness Constraint ให้สร้างรีเลชันโดยมี Unique Identifier ของชนิดเอนติตีเหล่านั้นเป็น Candidate Key



รูปที่ 3.31 ตัวอย่างแบบจำลองข้อมูล (Conceptual Schema)

จาก Conceptual Schema ในรูปที่ เมื่อใช้ ONF อัลกอริธึม จะได้รีเลชันดังต่อไปนี้



รูปที่ 3.32 รีเลชันของแบบจำลองรูปที่ 3.31

โดยมีเครื่องหมาย ◄—► แอทริบิวท์หรือกลุ่มของแอทริบิวท์ที่เป็น Primary key และเครื่องหมาย ◄—► บน candidate key

3.6 Object Role Modeling (ORM)

Versions แรกๆ ของ ORM ถูกพัฒนาในยุโรปในกลางปี1970 เช่น binary relationship modeling และ NIAM มาตรฐาน ORM ได้รวมส่วนประกอบบางอย่างของแนวคิดเชิงวัตถุ (object – oriented) เช่น inheritance แต่แตกต่างจากวิธีการเชิงวัตถุโดยทั่วไป ซึ่งจริงๆ แล้วถือว่าเป็นข้อดีเพราะวิธีเชิงวัตถุไม่ได้ให้แนวทางความคิดที่ชัดเจนแก่ระบบสารสนเทศ ตัวอย่างเช่น วิธีเชิงวัตถุต่างๆ ไม่ค่อยมีการให้โครงสร้างที่สะดวกแก่การสื่อสารระหว่างนักออกแบบและผู้ใช้ (เช่น การพิจารณา fact instances และ เงื่อนไขต่างๆ) และวิธีเชิงวัตถุเหล่านี้ทำให้ส่วนของการวิเคราะห์ซับซ้อนด้วยรายละเอียดของการนำไป implement (เช่น การผูกมัดทันทีในวิธีจัดกลุ่ม facts เป็น objects และรายละเอียดที่ซ้ำซ้อน)

ORM เป็นวิธีการหนึ่งสำหรับการออกแบบและการเรียกค้น (query) ระบบสารสนเทศในระดับความคิด และการ map ระหว่าง conceptual และ logical (เช่น relational) levels ORM มีปรากฏในหลายรูปแบบ ซึ่งรวมถึง NIAM

ORM เป็นวิธีการดั้งเดิมอันหนึ่งที่ใช้สำหรับการออกแบบและการ query ระบบสารสนเทศในระดับความคิด ในยุโรป วิธีการนี้ถูกเรียกว่า NIAM เพราะว่าระบบสารสนเทศตามระเบียบแล้วมักถูก implemented บน DBMS ที่อยู่บนพื้นฐานของ logical data model (เช่น relational, object - relational, heirarchic) ORM รวมเอากระบวนการสำหรับการ map ระหว่าง conceptual และ logical levels ไว้ด้วย ORM มุ่งเน้นไปที่การออกแบบข้อมูล เพราะว่ามุมมองข้อมูลถือว่ามั่นคงที่สุด และให้พื้นฐานอย่างเป็นทางการสำหรับการนิยาม operations

ORM ทำให้กระบวนการออกแบบง่ายแก่ความเข้าใจและชัดเจน โดยการใช้ภาษาธรรมชาติ และอธิบายข้อมูลในเทอมของความสัมพันธ์เบื้องต้น

เพื่อความถูกต้อง ความชัดเจนและความสามารถในการประยุกต์ ระบบสารสนเทศต้องถูกระบุไว้อย่างดีที่สุดในตอนแรกที่ระดับแนวคิด โดยการใช้แนวคิดและภาษาที่ผู้คนสามารถเข้าใจได้โดยง่ายดาย การวิเคราะห์และออกแบบเกี่ยวข้องกับการสร้าง model ของแอปพลิเคชันอย่างเป็นทางการของส่วนแอปพลิเคชันหรือ universe of discourse (UoD) เพื่อให้ได้ความเข้าใจที่ดีของ UoD ORM ทำให้การทำงานนี้ง่ายขึ้นโดยการใช้ภาษาธรรมชาติ พร้อมกับไดอะแกรมที่สามารถใส่ข้อมูลตัวอย่าง และแสดงความหมายข้อมูลในรูปแบบของ elementary relationships

ที่เรียกว่า ORM เนื่องจากว่ามันแสดงภาพโลกในรูปแบบของ objects (entities หรือ values) ที่แสดงบทบาท (ส่วนของ relationships) เช่น ขณะนี้คุณกำลังแสดงบทบาทการอ่าน และ paper นี้กำลังแสดงบทบาทของการถูกอ่าน ซึ่งตรงข้ามกับเทคนิคการออกแบบอื่นเช่น Entity - Relationship (ER) และวิธีการเชิงวัตถุ ORM ไม่มีการใช้แอททริบิวท์โดยชัดเจน เช่นแทนที่จะใช้ countryBorn เป็นแอททริบิวท์ของ Person เราจะใช้ชนิดความสัมพันธ์ Person was born in Country

3.6.1 ข้อได้เปรียบที่สำคัญของ ORM

ข้อแรก ORM models และ queries มั่นคงกว่า (แอททริบิวท์ๆ ค่อยปรากฏในเอนติตีและความสัมพันธ์) เช่น ถ้าเราตัดสินใจที่จะบันทึกจำนวนประชากรของประเทศแล้ว แอททริบิวท์ countryBorn ของเราจำเป็นต้องถูกคิดให้เป็นความสัมพันธ์ใหม่

ข้อที่ 2 ORM models สะดวกแก่การใส่ตัวอย่างที่มากมาย (แอททริบิวท์ทำให้มันทำได้ลำบาก)

ข้อที่ 3 ORM เป็นแบบเดียวกันมากกว่า (เช่น เราไม่ต้องการเครื่องหมายแยกต่างหากเพื่อใส่เงื่อนไขเดียวกันแก่แอททริบิวท์ โดยใช้เป็นความสัมพันธ์)

ORM สื่อความหมายได้ดีกว่า ER หรือวิธีเชิงวัตถุ เครื่องหมายที่ใช้ role กำกับทำให้ง่ายต่อการระบุรายละเอียดความหลากหลายอย่างกว้างของเงื่อนไข และชนิดออบเจ็กต์ของ ORM ให้โดเมนทางความหมายที่ผูกกับโครงร่างด้วย ผลประโยชน์อย่างหนึ่งของ ORM คือ conceptual queries ที่ถูกกำหนดในรูปแบบของเส้นทางโครงร่าง ที่ย้ายจาก role หนึ่ง แม้ว่าชนิดออบเจ็กต์หนึ่งผ่านไปอีก role เท่ากับเป็น conceptual join

โมเดลเชิงวัตถุที่นิยมใช้กันมักสำเนาข้อมูลโดยการรวมความจริงเข้าเป็นคู่ของแอททริบิวท์ที่ inverse กัน ในอ็อบเจ็คต่างกัน ซึ่งใน ORM หรือ ER ไม่น่าเป็นไปได้ ยิ่งไปกว่านี้ เครื่องหมายเชิงวัตถุไม่ค่อยสนับสนุนการมีเงื่อนไขนัก (เช่น เงื่อนไขหนึ่งอาจต้องถูกทำสำเนาใน object แต่ละตัว หรือไม่ก็ละไปเลย)

เป็นการไม่ดีที่โมเดลเชิงวัตถุมีความมั่นคงน้อยกว่า ER models ด้วย ดังนั้นโมเดลเชิงวัตถุจึงควรถูกใช้ในการ implement เท่านั้น ไม่ใช่การวิเคราะห์ แม้ว่ารูปภาพที่ถูกใส่รายละเอียดโดยใช้ ORM เป็นที่น่าพอใจในการพัฒนาและการเปลี่ยนรูปโมเดลสำหรับจุดประสงค์การย่อ มันเป็นประโยชน์ที่จะซ่อนหรือย่อการแสดงรายละเอียดที่มีมาก โดยใช้วิธีการแยกแยะความหลากหลายที่ยังมีอยู่ ถ้ายังต้องการใช้ ER และไดอะแกรมเชิงวัตถุในการย่ออย่างกระชับด้วยก็ได้และพัฒนา ER และไดอะแกรมเชิงวัตถุให้ดีที่สุดด้วยมุมมองของไดอะแกรม ORM

3.6.2 The conceptual schema design procedure (CSDP)

กระบวนการออกแบบโครงร่างระดับคิด (CSDP) ของ ORM มุ่งเน้นไปที่การวิเคราะห์และออกแบบข้อมูล โครงร่างระดับแนวคิดแสดงรายละเอียดโครงสร้างข้อมูลของแอพลิเคชัน เกี่ยวกับชนิดความจริง เงื่อนไขบนชนิดความจริง และกฎที่ได้หาได้สำหรับการหาความจริงได้จากความจริงอื่น สำหรับแอพลิเคชันขนาดใหญ่ UoD ถูกแบ่งเป็นกลุ่มที่เหมาะสม CSDP ถูกนำไปใช้ และผลของโครงร่างย่อย ที่ถูกรวมไว้ในโครงร่างแนวคิดหลัก

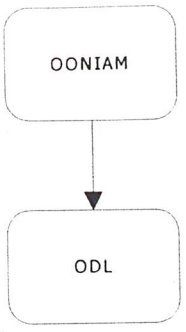
3.6.3 ขั้นตอนของ The conceptual schema design procedure (CSDP)

1. Transform familiar information examples into elementary facts, and apply quality checks.
2. Draw the fact types, and apply a population check.
3. Check for entity types that should be combined, and note any arithmetic derivations.
4. Add uniqueness constraints, and check arity of fact types.
5. Add mandatory role constraints, and check for logical derivations.
6. Add value, set comparison and subtyping constraints.
7. Add other constraints and perform final checks.

แสดง CSDP ที่ถูกใช้ใน FORM แม้ว่าจะมี CSDP ที่แตกต่างกันหลายแบบ แต่ทุก CSDP เห็นตรงกันในความสำคัญของการแสดงเป็นคำพูดทำให้เป็นกริยาในรูปของ elementary facts, การตรวจสอบโดยการใส่ข้อมูล และการวิเคราะห์กฎทางธุรกิจ

3.7 แบบจำลองแนวคิด OONIAM และการแปลงรูป

การสร้างแบบจำลองแนวคิด OONIAM ให้สามารถทำงานร่วมกับฐานข้อมูลเชิงวัตถุได้นั้นมีขั้นตอนดังรูปที่ 3.33



รูปที่ 3.33 ขั้นตอนการทำงานเพื่อนำ OONIAM ไปทำงานร่วมกับฐานข้อมูลเชิงวัตถุ

จากรูปที่ 3.33 OONIAM เป็นโมเดลสำหรับการออกแบบฐานข้อมูลเชิงวัตถุ เป็นโมเดลที่นำ NIAM เดิมมาเพิ่มขยายความสามารถให้รองรับแนวคิดเชิงวัตถุได้ สำหรับ ODL เป็นภาษานิยามฐานข้อมูลเชิงวัตถุ

ในบทนี้กล่าวถึง OONIAM และการแปลงรูปเป็นภาษานิยามเชิงวัตถุ (ODL) ซึ่งจะแบ่งส่วนในการอธิบายดังนี้ หัวข้อถัดไป เป็นการนำเสนอโมเดล OONIAM เพื่อทำให้ NIAM ที่มีอยู่เดิมสามารถรองรับแนวคิดเชิงวัตถุได้ ซึ่งในหัวข้อนี้จะได้กล่าวถึงโครงร่างหลัก (Main Schema) และโครงร่างย่อย (Sub Schema) ที่เป็นส่วนสำคัญในการสร้าง OONIAM โดยโครงร่างทั้งสองจะนำเสนอความสัมพันธ์ต่างๆ ของแอททริบิวต์ และความสัมพันธ์ของออบเจ็กต์ ในหัวข้อถัดไป จะกล่าวถึงการแปลง OONIAM ให้เป็น ODL โดยจะนำโครงร่างหลักและโครงร่างย่อยมาผ่านการประมวลผลเพื่อแปลงโครงร่างทั้งสองให้เป็นภาษา ODL

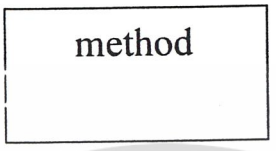
3.8 แบบจำลองแนวคิด OONIAM

เมื่อผู้ใช้งานต้องการที่จะออกแบบฐานข้อมูลบนระบบฐานข้อมูลเชิงวัตถุ ผู้ใช้งานต้องอาศัยเครื่องมือในการออกแบบฐานข้อมูล ซึ่งในปัจจุบันเครื่องมือที่เป็นที่นิยมนำมาช่วยในการออกแบบฐานข้อมูลมีด้วยกัน 2 เครื่องมือคือ ER และ NIAM ซึ่งเครื่องมือทั้งสองก็มีข้อดีข้อเสียที่แตกต่างกันออกไป การที่ผู้ใช้งานจะใช้เครื่องมือใดนั้นก็แล้วแต่ความถนัด คุ้นเคยและประสบการณ์ของแต่ละคน แต่มีข้อสังเกตอยู่ประการหนึ่งก็คือ NIAM นั้นสามารถออกแบบฐานข้อมูลได้ถึง 5NF เลย แต่ ER ไม่สามารถทำได้เพราะ ER ไม่มีการระบุถึงความสัมพันธ์ระหว่างแอททริบิวท์ จึงทำให้ ER สามารถออกแบบฐานข้อมูลได้แค่ 3NF เท่านั้น ซึ่งในการออกแบบฐานข้อมูลที่ใช้งานทั่วไปนั้น ส่วนมากแล้วการออกแบบแค่ 3NF ก็สามารถทำงานได้แล้ว

การใช้งาน ER และ NIAM นั้นจะใช้กับระบบฐานข้อมูลเชิงสัมพันธ์เท่านั้น แต่ในปัจจุบันระบบฐานข้อมูลมิได้มีเพียงระบบฐานข้อมูลเชิงสัมพันธ์เท่านั้น แต่มีระบบฐานข้อมูลเชิงวัตถุสัมพันธ์และระบบฐานข้อมูลเชิงวัตถุอีกด้วย ซึ่งระบบฐานข้อมูลทั้งสองได้รับการพัฒนาให้มีขีดความสามารถที่สูงขึ้น โดยมีแนวคิดให้สามารถรองรับข้อมูลและกระบวนการของข้อมูลที่สลับซับซ้อนมากขึ้นได้ เมื่อมีระบบฐานข้อมูลแบบใหม่ขึ้นมาแล้ว ก็ต้องมีการเครื่องมือในการออกแบบใหม่ด้วย ซึ่ง ER และ NIAM ที่มีแต่เดิมไม่สามารถใช้งานกับระบบฐานข้อมูลใหม่ทั้งสองนี้

ในหัวข้อนี้จะกล่าวถึงการนำเอา NIAM ที่มีแต่เดิมมาเพิ่มขยายความสามารถให้รองรับหลักการเชิงวัตถุได้ ซึ่งเครื่องมือใหม่นี้เรียกว่า โมเดลไนแอมเชิงวัตถุ (Object Oriented NIAM Schema Model) หรือ เรียกย่อๆ ว่า OONIAM เมื่อผู้ใช้งานสร้างแบบจำลองของฐานข้อมูลเชิงวัตถุโดยใช้ OONIAM แล้ว ผู้ใช้งานสามารถแปลงแบบจำลองที่สร้างขึ้นให้เป็นภาษานิยามเชิงวัตถุ (ODL) ได้เลย

สำหรับหลักการสร้าง OONIAM นั้น จะนำสัญลักษณ์เดิมที่มีอยู่ใน NIAM มาใช้ร่วมกับสัญลักษณ์ใหม่ที่เพิ่มเติมเข้าไปดังรูปที่ 3.34 เพื่อให้ NIAM ใหม่นี้รองรับหลักการเชิงวัตถุได้



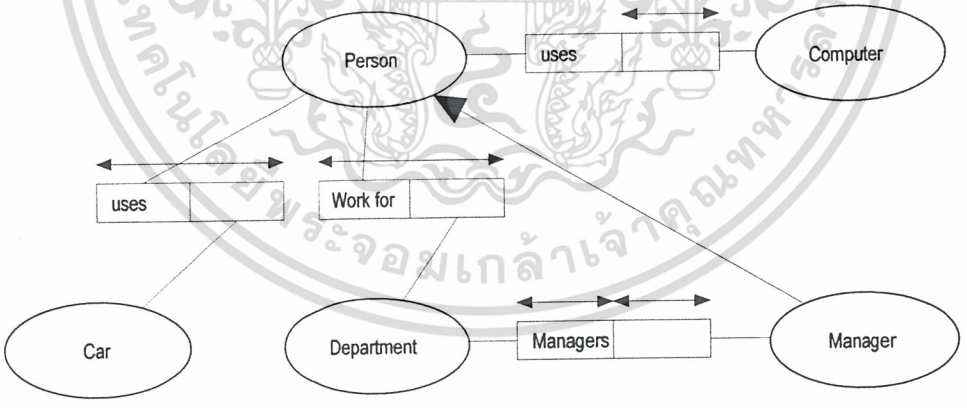
รูปที่ 3.34 สัญลักษณ์ของการประมวลผลข้อมูล (เมธอด)

สำหรับสัญลักษณ์ที่เพิ่มเข้าไปในรูปที่ 3.34 นั้นคือ กระบวนการประมวลผลนี้จะมีไว้ให้ผู้ใช้งานที่เพิ่มเข้าไปที่เอนติตี้ที่ต้องการ เพื่อให้เมธอดที่เพิ่มเข้าไปนี้จัดการกับข้อมูลที่ต้องการได้โดยรายละเอียดจะกล่าวถึงในหัวข้อ โครงร่างย่อย (Sub Schema)

OONIAM แบ่งส่วนสำคัญออกเป็นสองส่วนดังนี้

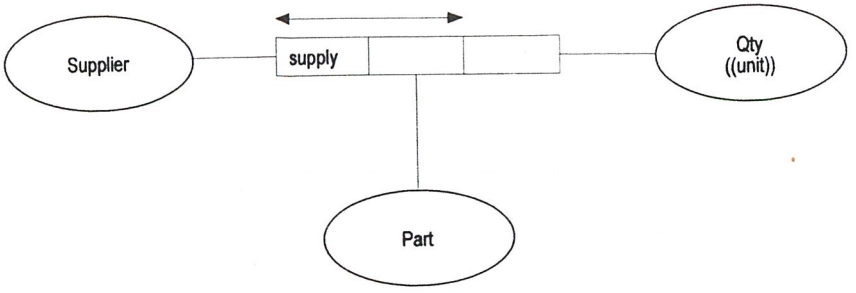
3.8.1.โครงร่างหลัก (Main Schema)

โครงร่างหลักนี้เป็นภาพรวมของความสัมพันธ์ระหว่างคลาส ดังนั้นในแต่ละคลาสจะไม่มีการแสดงแอททริบิวต์ และไม่มี Uniqueness Identifier ดังตัวอย่างในรูปที่ 3.35



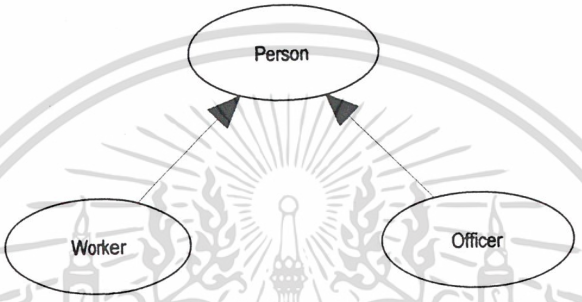
รูปที่ 3.35 โครงร่างหลักของคลาสหลายคลาส

หากมีความสัมพันธ์กันแบบ N-ary ก็จะสามารถมี Uniquness Identifier ได้ที่ entity ดังรูปที่ 3.36



รูปที่ 3.36 โครงร่างหลักแบบที่มี Uniqueness Identifier ที่ entity

นอกจากนี้ การอธิบายถึงความสัมพันธ์ระหว่างคลาสแล้ว ในโครงร่างหลักยังได้กล่าวถึง การถ่ายทอดคุณสมบัติของคลาส (Inheritance) อีกด้วย ดังรูปที่ 3.37

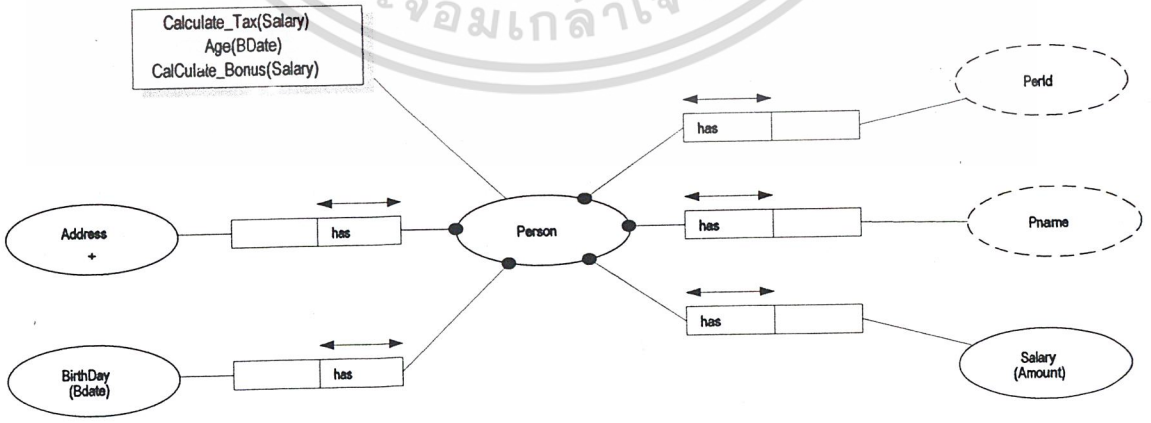


รูปที่ 3.37 การถ่ายทอดคุณสมบัติ

สำหรับโครงร่างหลักนี้จะแสดงความสัมพันธ์ระหว่างคลาสแบบหนึ่งต่อหนึ่ง หนึ่งต่อหลาย หลายต่อหนึ่ง หรือ หลายต่อหลายก็ได้ แล้วแต่การออกแบบคลาสนั้นๆ ซึ่งแอททริบิวต์ของแต่ละคลาสและ Uniqueness Identifier นั้นจะแสดงในโครงร่างย่อย

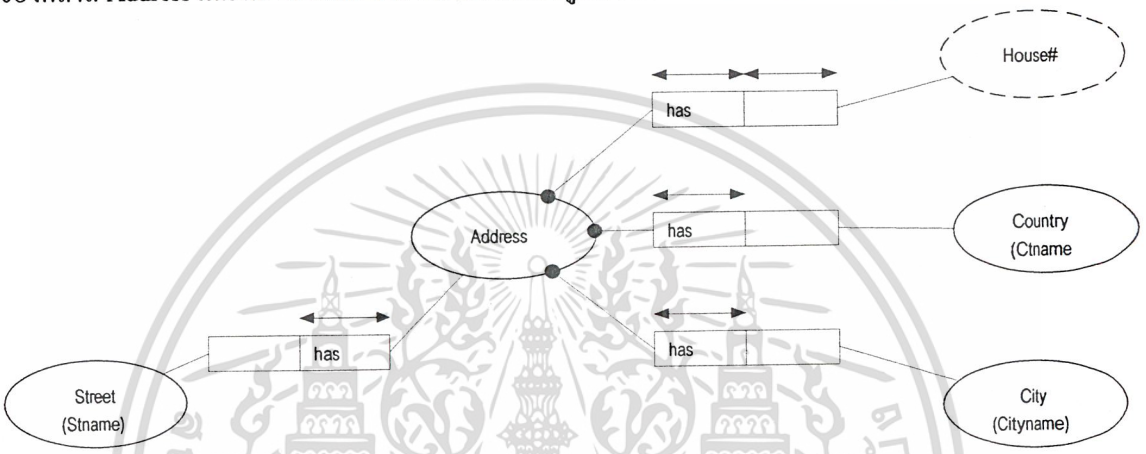
3.8.2 โครงร่างย่อย (Sub Schema)

โครงร่างย่อยเป็นการแสดงรายละเอียดของคลาส ซึ่งแต่ละคลาสนั้นอาจเป็นคลาสที่ซับซ้อน (Complex) ก็ได้ กล่าวคือ คลาสดังกล่าวอาจมีคลาสอื่นเป็นส่วนประกอบก็ได้ โดยโครงร่างย่อยนี้ประกอบด้วย แอททริบิวต์, เมธอด และคลาสที่อยู่ในคลาส ดังรูปที่ 3.38



รูปที่ 3.38 โครงร่างย่อยระดับที่ 1 ของคลาส Person

จากรูปที่ 3.38 เป็นการเขียนเพิ่มขยายโครงร่างหลักในรูปที่ กล่าวคือในรูปที่ 3.37 ได้บอกถึงความสัมพันธ์ของคลาส Person ที่มีต่อคลาสอื่นๆ แต่ยังไม่ได้อธิบายถึงรายละเอียดของคลาส Person เมื่อต้องการอธิบายรายละเอียดต่างๆ ของคลาสดังกล่าวจำเป็นต้องอธิบายในโครงร่างย่อยเท่านั้น จากโครงร่างย่อยในรูปที่ 3.38 ซึ่งเป็นโครงร่างย่อยระดับที่ 1 จะเห็นว่าคลาส Person จะมีแอททริบิวต์ต่างๆ ดังแสดงในรูป และมีเมธอดสำหรับจัดการกับคลาสนี้ด้วยกัน 3 เมธอด นอกจากนี้แล้วสัญลักษณ์ที่แสดงชื่อคลาสหลักนี้จะไม่มี Uniqueness Identifier ที่สำคัญคลาสนี้ยังมีคลาส Address เป็นคลาสย่อย (Embedded Class) อีกโดยคลาสย่อยนี้แสดงโดยการใช้เครื่องหมายบวก (+) ไว้ใต้ชื่อคลาส ซึ่งหากต้องการเขียนโครงร่างย่อยของคลาส Address และคลาส Land สามารถเขียนได้ดังรูปที่ 3.39 ซึ่งเป็นโครงร่างย่อยระดับที่ 2



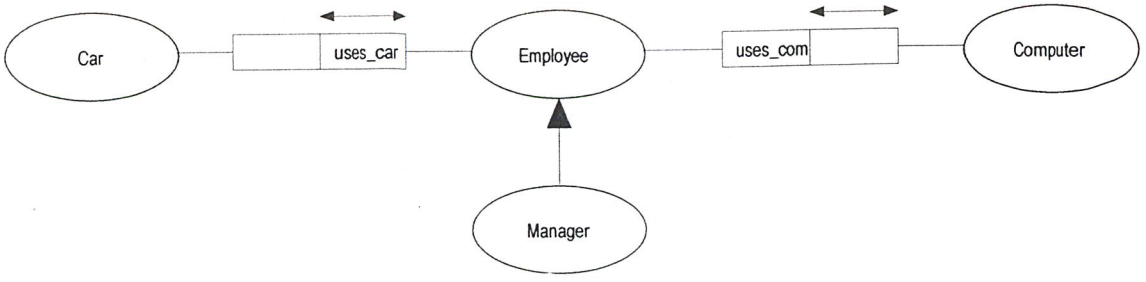
รูปที่ 3.39 โครงร่างย่อยของคลาส Address

จากรูปที่ 3.38 และรูปที่ 3.39 จะเห็นได้ว่าโครงร่างย่อยของคลาส Person มีด้วยกัน 2 ระดับคือ ระดับแรก เป็นการอธิบายคลาส Person (รูปที่ 3.38) ส่วนระดับที่ 2 เป็นการอธิบายคลาสที่เกี่ยวเนื่องกับคลาส Person (รูปที่ 3.39) ซึ่งจะสังเกตได้ว่าในโครงร่างย่อยแต่ละระดับนั้นหากต้องการอธิบายรายละเอียดของคลาสใด คลาสนั้นจะไม่มีการแสดง Uniqueness Identifier ในโมเดล ส่วนคลาสที่เกี่ยวเนื่องนั้นจะแสดงในโครงร่างย่อยในระดับถัดๆ ไป

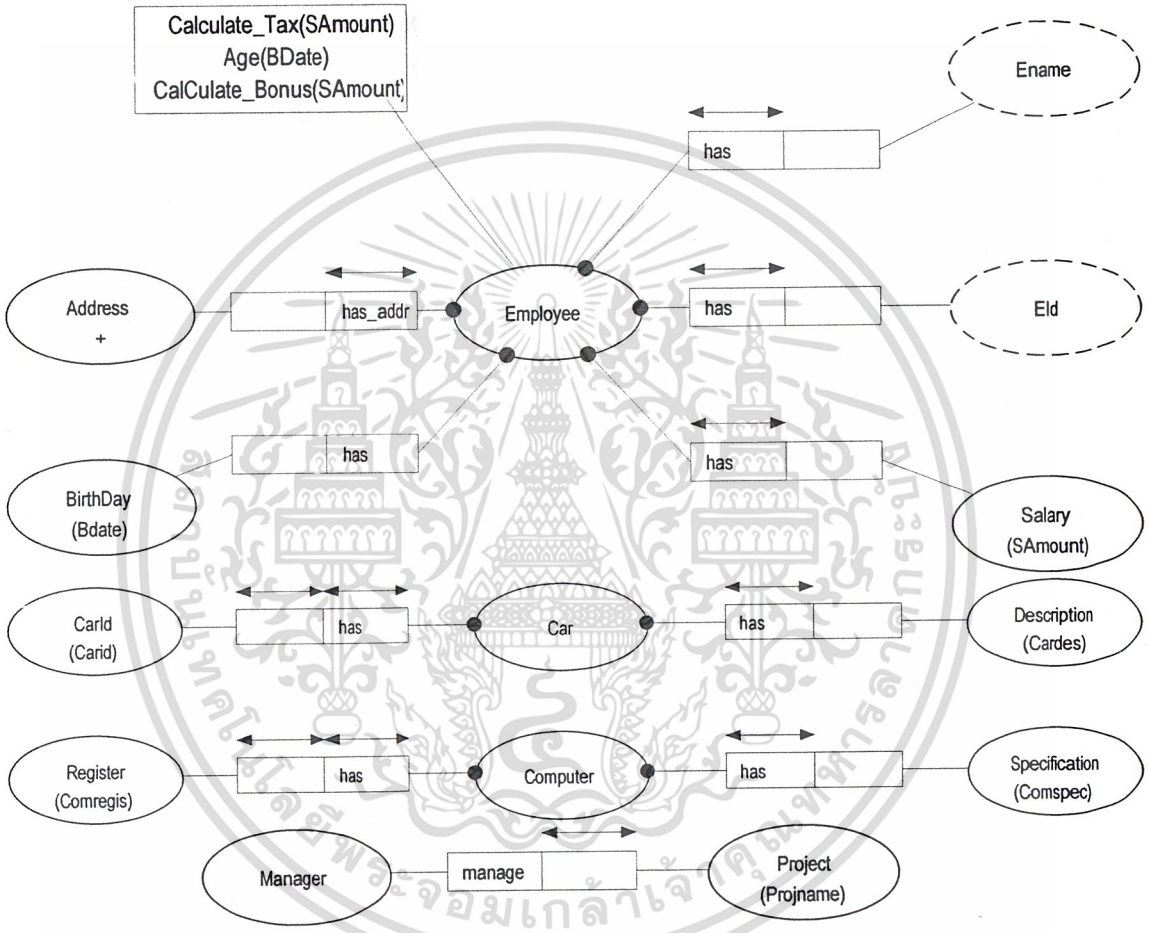
สำหรับโครงร่างย่อยทั่วๆ ไปนั้นอาจมีหลายระดับ ขึ้นอยู่กับความซับซ้อนของข้อมูลและการออกแบบของผู้ออกแบบ

3.9 การแปลงโมเดล OONIAM ให้เป็นภาษานิยามเชิงวัตถุ ODL

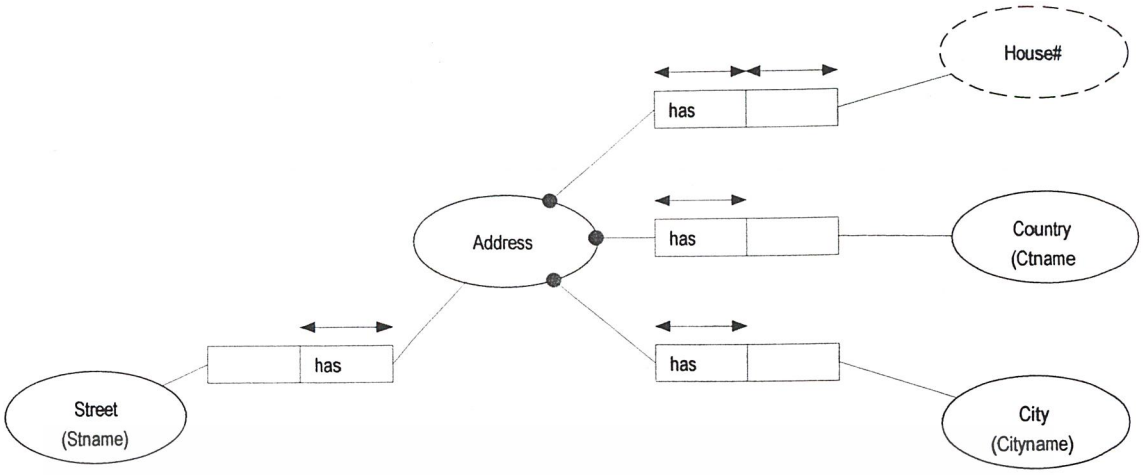
ก่อนที่จะกล่าวถึงการใช้เวลาในโมเดลนี้ จะขอกล่าวถึงการแปลงโมเดล OONIAM ซึ่งเป็นโมเดล NIAM เชิงวัตถุให้เป็นภาษานิยามเชิงวัตถุ ODL ก่อน สำหรับขั้นตอนการแปลงนั้นเริ่มต้นต้องแปลงจากโครงร่างย่อยก่อน จากนั้นจึงไปแปลงในโครงร่างหลัก เนื่องจากในโครงร่างย่อยได้อธิบายโครงสร้างของคลาสไว้อย่างละเอียด ดังนั้นเราจำเป็นต้องสร้างคลาสต่างๆ ก่อน จึงจะนำโครงร่างหลักมาใช้งานร่วมเพื่ออธิบายถึงความสัมพันธ์ระหว่างคลาส ซึ่งการแปลง OONIAM ให้เป็น ODL ดังที่กล่าวมาแล้วนั้นแสดงตัวอย่างในรูปที่ 3.40, รูปที่ 3.41 และ รูปที่ 3.42



รูปที่ 3.40 ตัวอย่างโครงร่างหลักของระบบ



รูปที่ 3.41 ตัวอย่างโครงร่างย่อยระดับที่ 1



รูปที่ 3.42 ตัวอย่างโครงร่างย่อยระดับที่ 2

จากรูปที่ 3.40, รูปที่ 3.41 และ รูปที่ 3.42 สามารถแปลงเป็นภาษานิยามเชิงวัตถุได้ดังนี้

```
interface Employee
(       extent Employees)
{
        typedef Struct AddressStruct {
                sring Houseid,
                sring Ctname,
                string Cityname,
                string Stnae;}
        attribute string Eid;
        atribute string Ename;
        attribute float Salary;
        attribute date Bdate;
        attribute set<AddressStruct> Address;
        relationship Car uses_car inverse Car::is_used_by;
        relationship set<Computer> uses_com inverse Computer::is_used_by;
        float Calculate_Tax(in float Samount);
        float Age(in float Bdate);
        float Calculate_Bonus(in float Samount)
};

interface Manager:Employee
(       extent Managers)
{
```

```
        attribute string ProName;
};

interface Computer
(       extent Computers)
{
        attribute string Comregis;
        attribute string Comspec;
        relationship Employee is_used_by inverse Employee::uses_com;
};
interface Car
(       extent Cars)
{
        attribute string Carid;
        attribute string Cardes;
        relationship set<Employee> is_used_by inverse Employee::uses_car;
};
```

3.10 หลักการแปลง OONIAM ให้เป็นภาษา ODL

สำหรับหลักการในการแปลง OONIAM ให้เป็นภาษา ODL มีหลักการดังนี้

3.10.1 การแปลงโครงร่างย่อย (Sub Schema)

โครงร่างย่อยเป็นการอธิบายรายละเอียดของแต่ละคลาส ซึ่งจะอธิบายแอททริบิวต์ คลาสย่อยและคลาสอ้างอิง โดยจะต้องแยกเป็นกรณีๆ ไป ซึ่งคลาสทั้งหมดได้ถูกสร้างขึ้นแล้วในขั้นตอนการแปลงโครงร่างหลัก ดังนั้นในโครงร่างย่อยนี้เป็นส่วนของการเพิ่มรายละเอียดของแอททริบิวต์ สร้างคลาสย่อยและสร้าลคลาสอ้างอิงเท่านั้น สำหรับอัลกอริธึมในการแปลงโครงร่างย่อยมีดังนี้

นำโครงร่างย่อยมาสร้างเป็นคลาสใหม่จัดเก็บลงใน Meta Class โดย

สร้างแอททริบิวต์ของคลาส โดยใช้ Uniqueness Identifier เป็นชื่อแอททริบิวต์ โดยสามารถแบ่งลักษณะของชนิดความจริง (Fact Type) ที่คลาสหลักมีกับแอททริบิวต์ ได้ดังนี้

ชนิดความจริงแบบหนึ่งต่อหนึ่ง (One to one) แอททริบิวต์นั้นเป็น Simple

ชนิดความจริงแบบหนึ่งต่อหลาย (One to many) แอททริบิวต์นั้นเป็น Collection

ชนิดความจริงแบบหลายต่อหนึ่ง (Many to one) แอททริบิวต์นั้นเป็น Simple

ชนิดความจริงแบบหลายต่อหลาย (Many to Many) แอททริบิวต์นั้นเป็น Collection

หากคลาสใดมีคลาสย่อย นำคลาสที่สร้างไว้ใน Meta Class มาสร้างเป็นคลาสย่อยของคลาสนั้น โดยสามารถแบ่งลักษณะของชนิดความจริง (Fact Type) ที่คลาสหลักมีกับคลาสย่อยนั้นได้ดังนี้

ชนิดความจริงแบบหนึ่งต่อหนึ่ง (One to one) คลาสย่อยนั้นเป็น Simple

ชนิดความจริงแบบหนึ่งต่อหลาย (One to many) คลาสย่อยนั้นเป็น Collection

ชนิดความจริงแบบหลายต่อหนึ่ง (Many to one) คลาสย่อยนั้นเป็น Simple

ชนิดความจริงแบบหลายต่อหลาย (Many to Many) คลาสย่อยนั้นเป็น Collection

3.10.2 การแปลงโครงร่างหลัก (Main Schema)

โครงร่างหลักนี้ได้อธิบายถึงความสัมพันธ์ในแต่ละคลาส รวมถึงการถ่ายทอดคุณสมบัติของคลาสอีกด้วย นอกจากนี้ยังอธิบายถึงความสัมพันธ์แบบ N-ary ด้วย สำหรับอัลกอริธึมในการแปลงโครงร่างหลักมีดังนี้

กำหนดคุณสมบัติให้กับคลาสที่มีการถ่ายทอดคุณสมบัติ

กำหนดความสัมพันธ์ระหว่างคลาส ดังนี้

ชนิดความจริงแบบหนึ่งต่อหนึ่ง (One to one) กำหนดให้ทั้งสองคลาสมีความสัมพันธ์กันแบบ Reference ทั้งสองคลาส

ชนิดความจริงแบบหนึ่งต่อหลาย (One to many) กำหนดให้ทั้งสองคลาสมีความสัมพันธ์กันแบบ Collection ของ Reference และ Reference ตามลำดับ

ชนิดความจริงแบบหลายต่อหนึ่ง (Many to one) กำหนดให้ทั้งสองคลาสมีความสัมพันธ์กันแบบ Reference และ Collection ของ Reference ตามลำดับ

ชนิดความจริงแบบหลายต่อหลาย (Many to Many) กำหนดให้ทั้งสองคลาสมีความสัมพันธ์กันแบบ Collection ของ Reference และ Collection ของ Reference ตามลำดับ

หากมีความสัมพันธ์แบบหลายบทบาท ให้สร้างคลาสใหม่เพื่อจัดเก็บ Simple Object ที่เกี่ยวข้องกับ N-ary โดยสร้างความสัมพันธ์หนึ่งต่อหนึ่งไปยังคลาส Complex และกำหนดให้คลาส Complex มีความสัมพันธ์แบบหนึ่งต่อหลายไปยังคลาสที่สร้างใหม่

บทที่ 4
ฐานข้อมูลเชิงวัตถุสัมพันธ์

ฐานข้อมูลเชิงวัตถุสัมพันธ์ (Object-Relational Database) คือการผสมกันระหว่างฐานข้อมูลแบบรีเลชั่นนัลและฐานข้อมูลเชิงวัตถุ ซึ่งฐานข้อมูลเชิงวัตถุสัมพันธ์นี้จะมีจุดเด่นของฐานข้อมูลทั้งสองแบบคือ สนับสนุนการใช้โครงสร้างข้อมูลแบบซับซ้อนเช่นเดียวกับฐานข้อมูลเชิงวัตถุ และมี Data Dependency เช่นเดียวกับฐานข้อมูลแบบรีเลชั่นนัล

4.1 ความสัมพันธ์ซับซ้อน(Nested Relations)

ความสัมพันธ์ซ้อนมีลักษณะคล้ายกับความสัมพันธ์ (Relation) แตกต่างกันที่ความสัมพันธ์ซ้อนสามารถมีโดเมนที่เป็นได้ทั้งค่าที่ไม่สามารถแบ่งย่อยได้ (Atomic Value) หรือความสัมพันธ์ก็ได้ 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}

ตารางที่ 4.1 ความสัมพันธ์ของเอกสารให้ชื่อ doc ซึ่งยังไม่เป็น 1NF

ตารางที่ 4.1 แสดงตัวอย่างความสัมพันธ์ของเอกสารให้ชื่อว่า *doc* ซึ่งสามารถแสดงเป็น 1 NF ดังตารางที่ 4.2 เนื่องจากโดเมนต้องไม่สามารถแบ่งย่อยได้ (atomic) เพื่อให้เป็น 1 NF ดังนั้นจึงจะต้องมีทับเพิ้ลหนึ่งสำหรับแต่ละคู่ (keyword, author) และ *date* จะถูกแทนด้วยสามแอททริบิวท์ ซึ่งแต่ละแอททริบิวท์แทนแต่ละฟิลด์ย่อยของ *date* และกำหนด multivaiued dependencies ดังนี้

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

จากนั้นสามารถทำให้เป็น 4 NF โดยให้ schema ดังนี้

(title, author)

(title, keyword)

(title, day, month, year)

Title	*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 4.2 flat-doc หรือ 1NF ของความสัมพันธ์ doc

Title	*Author*
Salesplan	Smith
Salesplan	Jones
Status report	Smith
Status report	Frick

Title	*Keyword*
Salesplan	Profit
Salesplan	Strategy
Status report	Profit
Status report	Personnel

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

ตารางที่ 4.3 4NF ของ flat-doc ในตารางที่ 4.2

4.2 ชนิดข้อมูลแบบซับซ้อนและแนวคิดเชิงวัตถุ(Complex Types and Object Orientation)

ชนิดข้อมูลแบบซับซ้อน (Complex Types) และแนวคิดเชิงวัตถุทำให้แนวคิดโมเดลแบบ ER เช่น Multivalued Attributes เป็นต้น สามารถนำมาแสดงได้โดยตรงโดยไม่ต้องแปลความหมาย (complex translation) ให้อยู่ในแบบรีเลชั่นนัล

4.2.1 ชนิดข้อมูลแบบโครงสร้างและแบบสะสม(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)**

4.2.2 การสืบทอดคุณสมบัติ(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 ก็สามารถทำได้ โดยใช้ 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* เลยหรือตรงกับทับเพิ้ลของ *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

4.2.3 การสืบทอดคุณสมบัติของฟังก์ชั่น(Inheritance of Functions)

การสืบทอดคุณสมบัติของฟังก์ชั่นที่กำหนดโดยผู้ใช้(User-Defined Function) เกิดขึ้นได้โดยอาร์กิวเมนท์ของฟังก์ชั่น

กำหนด Type และตารางดังนี้

```
create type person_t (name varchar(30));
create type employee_t (salary int,
                        startdate date,
                        address varchar(30),
                        city varchar(30),
                        state varchar(30),
                        zipcode int)
        under person_t
create type student_t ( gpa float)
        under person_t
create type student_emp_t ( percent float)
        under employee_t, student_t
create table person of type person_t
```

create table emp of type employee_t **under** person

create table student of type student_t **under** person

create table student_emp of type student_emp_t **under** student, emp

สร้างฟังก์ชั่น

create function overpaid (employee_t, Arg1)

returns Boolean

as select Arg1.salary > (select salary from emp where name='Joe')

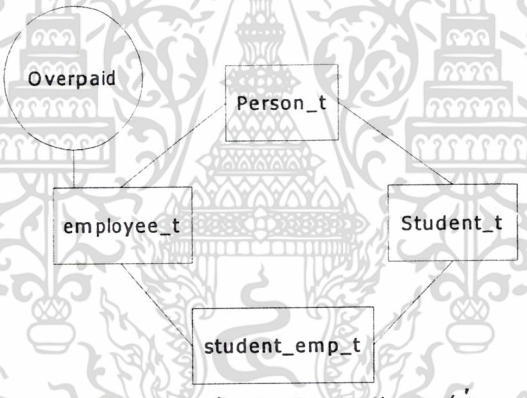
ฟังก์ชั่นนี้นำตัวอย่างข้อมูลของชนิด employee_t มาเป็นอาร์กิวเมนท์และคืนค่าบูลีนที่บอกว่า employee นั้นได้รับเงินเกินหรือไม่ (Overpaid)

พิจารณา

select e.name

from emp e

where overpaid(e)



รูปที่ 4.1 โครงสร้างของไทป์หลังจากการสร้างฟังก์ชั่น Overpaid

ขอบเขตของคำสั่งคือตาราง emp บวกกับตาราง student_emp ฟังก์ชั่น overpaid สามารถคำนวณโดยใช้ตาราง emp ได้ แต่ถ้าไม่มีการกำหนดฟังก์ชั่น overpaid ไว้กับชนิดข้อมูล student_emp_t แล้ว ORDBMS จะสืบทอดฟังก์ชั่นมาจากชนิดข้อมูลที่สูงกว่า(super type) มาโดยอัตโนมัติ

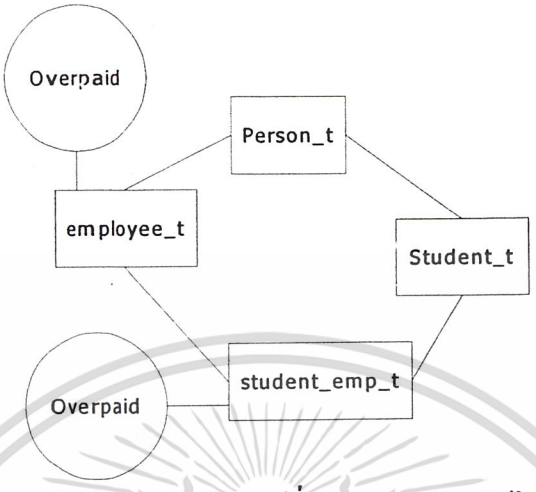
เมื่อระบบจัดการฐานข้อมูลเชิงวัตถุสัมพันธ์ถูกขอให้คำนวณค่าของฟังก์ชั่นและไม่มีฟังก์ชั่นที่มีชื่อและอาร์กิวเมนท์ที่ถูกต้อง ระบบจะค้นหาฟังก์ชั่นที่มีชื่อและอาร์กิวเมนท์ที่ถูกต้องในชนิดข้อมูลที่สูงกว่า ถ้ามีก็จะใช้ฟังก์ชั่นนั้น ดังนั้นจากตัวอย่าง student_emp_t จะสืบทอดฟังก์ชั่นจาก employee_t

นอกจากนี้ยังสามารถกำหนดฟังก์ชั่น overpaid ได้มากกว่าหนึ่งครั้งถ้ามีความต้องการให้ overpaid มีหลายความหมาย เช่น

create function overpaid (student_emp_t, Arg1)

returns Boolean

as select Arg1.salary > (select salary from student_emp where name = 'Bill')



รูปที่ 4.2 ระดับชั้นของไทป์หลังจากสร้างฟังก์ชั่น Overpaid สำหรับไทป์ student_emp_t

เมื่อสั่ง

select e.name

from emp e

where overpaid(e)

ระบบจัดการฐานข้อมูลเชิงวัตถุสัมพันธ์จะใช้ฟังก์ชั่น overpaid ของแต่ละตาราง ซึ่งจะได้ผลการทำงานเหมือนกับการ query สองครั้งดังนี้

select e.name

from only (emp)e

where overpaid(e)

select e.name

from only(student_emp)s

where overpaid(s)

แนวคิดนี้เรียกได้ว่าเป็น Polymorphism เพราะฟังก์ชั่นสามารถใช้ได้หลายความหมายต่อหลายชนิดข้อมูล

4.2.4 ชนิดข้อมูลแบบอ้างอิง(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

4.3 การค้นข้อมูลกับชนิดข้อมูลซับซ้อน(Querying with Complex Types)

คือการใช้ SQL ในการถามหาข้อมูล (query) เช่น หาชื่อและปีที่พิมพ์ของเอกสารแต่ละฉบับ คำสั่งในการถามคือ

select *name, date.year*

from *doc*

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

4.3.1 แอททริบิวท์ที่เป็นความสัมพันธ์(Relation-Valued Attributes)

สมมติความสัมพันธ์ *pdoc* ดังนี้

create table *pdoc*

(*name MyStrin*g,

author-list **setof**(**ref**(*people*)),

date MyDate,

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

ถ้าต้องการหาเอกสางที่มีคำว่า "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 ดังนั้น จำนวนสมาชิกที่นับได้ในเซ็ตก็คือจำนวนผู้เขียนนั่นเอง

4.3.2 การแสดงเส้นทาง(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*

การอ้างอิงสามารถใช้ซ่อนการ 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"

4.3.3 การทำให้ซับซ้อนและการคลายความซับซ้อน(Nesting and Unnesting)

การแปลงความสัมพันธ์ที่ซับซ้อน (Nested Relation) ให้เป็น 1 NF เรียกว่า "Unnesting" ดังเช่น ตารางที่ 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 4.4

title	*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 4.4 แสดงตารางที่ 4.2 หลังจากทำ Nesting

4.3.4 ฟังก์ชั่น(Functions)

ระบบวัตถุสัมพันธ์ยอมให้ผู้ใช้สามารถกำหนดฟังก์ชั่นขึ้นมาได้เอง ซึ่งสามารถกำหนดโดยใช้ภาษาสำหรับเขียนโปรแกรม 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

ในการกำหนดฟังก์ชั่นโดยภาษาสำหรับเขียนโปรแกรม(Programming Language)อื่นๆนั้นมีข้อดีที่ฟังก์ชั่นจะมีความสามารถสูงกว่ากำหนด้วย SQL เพราะ SQL ไม่สามารถทำงานบางอย่างได้ เช่นการคำนวณจำนวนเชิงซ้อน ฟังก์ชั่นที่กำหนดโดยภาษาสำหรับเขียนโปรแกรมนี้จะต้องคอมไพล์ภายนอกแล้วโหลดเข้ามา Execute พร้อมกับโค้ดของฐานข้อมูล ขั้นตอนเหล่านี้มีความเสี่ยงที่ข้อผิดพลาด (bug) ของโปรแกรมจะทำให้เกิดความเสียหายกับโครงสร้างของข้อมูล และยังสามารถลัดผ่านการควบคุมของระบบฐานข้อมูลด้วย

เมื่อฟังก์ชั่นที่ผู้ใช้กำหนดขึ้นเองนี้ถูกเรียกใช้ คำสั่งอาจจะถูกกระทำโดยระบบฐานข้อมูลเอง หรือถ่ายข้อมูลที่เกี่ยวข้องไปไว้แยกต่างหากก่อน กรณีแรกมีความเสี่ยงจากความผิดพลาดของฟังก์ชั่น ในขณะที่กรณีหลังจะใช้ overhead สูงมาก

4.4 การสร้างค่าข้อมูลและอ็อบเจ็คแบบซับซ้อน(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

ในการอัพเดทสามารถทำได้โดย **update** ของ SQL

บทที่ 5
การออกแบบและสร้างแอปพลิเคชั่นของระบบโดยใช้โมเดล UML และปัญหาในการใช้ UML

5.1 การออกแบบและสร้างแอปพลิเคชั่นโดยใช้ UML

ขั้นตอนที่ 1 สร้างยูสเคสไดอะแกรม

1. ค้นหาว่าระบบต้องทำงานอะไรได้บ้าง และมีอะไรบ้างที่เข้ามาใช้หรือเกี่ยวข้องกับระบบ
2. สิ่งที่ระบบต้องทำได้แต่ละอย่างจะเป็นหนึ่งยูสเคสในยูสเคสไดอะแกรมโดยไม่ต้องสนใจว่าการทำงานนั้นได้มาอย่างไรจากระบบ
3. สิ่งภายนอกที่เข้ามาเกี่ยวข้องหรือใช้ระบบแต่ละอย่างจะกำหนดเป็นตัวกระทำ (Actor)
4. ลากเส้นแอสโซสิเอชั่นระหว่างตัวกระทำกับการทำงานของระบบที่ตัวกระทำนั้นเกี่ยวข้อง
5. ถ้ายูสเคสหนึ่งมีการใช้การทำงานของอีกยูสเคสหนึ่ง ให้สร้างเส้นความสัมพันธ์แบบเอ็กเทนด์ระหว่างทั้งสองยูสเคสโดยให้หางลูกศรอยู่ที่ยูสเคสที่ถูกเรียกใช้การทำงาน และหัวลูกศรอยู่ที่ยูสเคสที่เป็นตัวใช้การทำงาน

ขั้นตอนที่ 2 สร้างซีเควนซ์ไดอะแกรม

นำแต่ละยูสเคสในยูสเคสไดอะแกรมมาเขียนซีเควนซ์ไดอะแกรมเพื่ออธิบายการปฏิสัมพันธ์ (Interaction) ระหว่างอ็อบเจ็คที่เกี่ยวข้องในการทำงานของยูสเคสนั้น

1. เขียนสัญลักษณ์ของอ็อบเจ็คสำหรับแต่ละอ็อบเจ็คและสัญลักษณ์ของตัวกระทำสำหรับแต่ละตัวกระทำไว้ด้านบนของไดอะแกรมเรียงกันในแนวนอนตามตัวอย่างในรูปที่ 2.3
2. เขียนเส้นชีวิตของแต่ละอ็อบเจ็ค
3. เขียนเส้นแสดงแมสเสจหรือตัวกระตุ้นที่มีการส่งติดต่อกันระหว่างอ็อบเจ็คและตัวกระทำตามลำดับเวลารวมทั้งแอ็กติเวชั่น
4. กำหนดทรานสิชั่นไทม์ (ถ้าจำเป็น)

รายละเอียดของแมสเสจและตัวกระตุ้นชนิดต่างมีอยู่ในบทที่ 2 หัวข้อที่ 2.2.2.4

ขั้นตอนที่ 3 สร้างคอลแลโบเรชั่นไดอะแกรม

เนื่องจากไดอะแกรมนี้มีลักษณะคล้ายกับซีเควนซ์ไดอะแกรม จึงสามารถใช้ซีเควนซ์ไดอะแกรมที่ได้จากขั้นตอนที่สองมาเปลี่ยนให้อยู่ในรูปของคอลแลโบเรชั่นไดอะแกรมได้ อีกทั้งถ้าใช้เครื่องมือช่วยในการออกแบบโมเดล UML เช่น Rational Rose ก็อาจมีตัวช่วยในการสร้างคอลแลโบเรชั่นไดอะแกรมขึ้นจากซีเควนซ์ไดอะแกรมได้โดยอัตโนมัติ

ขั้นตอนที่ 4 สร้างคลาสไดอะแกรม

รวบรวมคลาสที่มีอยู่ในระบบมาวาดเป็นโครงสร้าง

1. เขียนสัญลักษณ์ของคลาสตามรูปแบบในบทที่ 2 หัวข้อ 2.2.4.2 สำหรับแต่ละคลาสในระบบ
2. สำหรับคลาสที่จะอยู่ในฝั่นฐานข้อมูลให้ใส่สเตอริโอไทป์ <<entity>> ไว้ในส่วนชื่อ

(Name Compartment) ของคลาส

3. สำหรับคลาสที่จะอยู่ฝั่งแอปพลิเคชั่นให้ใส่สเตอริโอไทป์ <<boundary>> ไว้ในส่วนชื่อ (Name Compartment) ของคลาส
4. ใส่แอททริบิวท์และเมธอดหรือโอเปอเรชั่นของแต่ละคลาสลงในส่วนแอททริบิวท์และส่วนโอเปอเรชั่นของสัญลักษณ์คลาส
5. สำหรับแต่ละสองคลาสที่เกี่ยวข้องกันให้ลากเส้นความสัมพันธ์แบบแอสโซสิเอชั่นเชื่อมระหว่างสองคลาสนั้นพร้อมทั้งระบุมัลติพลิซิตี้ ถ้าความสัมพันธ์ระหว่างสองคลาสนั้นมีทิศทางให้ใส่หัวลูกศรตามทิศทางนั้นหรืออาจทั้งสองด้านก็ได้
6. สำหรับแต่ละสองคลาสที่คลาสหนึ่งเป็นสับคลาสหรือลูกของอีกคลาสหนึ่ง ให้เขียนเส้นความสัมพันธ์แบบเจนเนอรัลไลเซชั่นโดยให้หางลูกศรอยู่ที่คลาสลูกที่เป็นสับคลาสและให้หัวลูกศรอยู่ที่คลาสที่เป็นซูปเปอร์คลาส
7. สำหรับแต่ละสองคลาสที่คลาสหนึ่งเป็นส่วนหนึ่งของอีกคลาสหนึ่ง ให้เขียนเส้นความสัมพันธ์แบบแอสโซสิเอชั่นระหว่างสองคลาสโดยมีเครื่องหมายอักกริเกชั่นอยู่ที่คลาสที่เป็นฐาน
8. สำหรับคลาสใดที่มีความสัมพันธ์ขึ้นอยู่กับอีกคลาสหนึ่ง ให้เขียนเส้นดีเพนเดนซีโดยคลาสที่อยู่ที่หางลูกศรจะขึ้นอยู่กับคลาสที่หัวลูกศรชี้ไป

ขั้นตอนที่ 5 สเตทชาร์ทไดอะแกรม

ใช้สเตทชาร์ทไดอะแกรมในการบอกรายละเอียดการทำงานของคลาส

1. วาดสัญลักษณ์จุดเริ่มต้นและจุดสิ้นสุดการทำงาน
2. สร้างสเตทที่มีอยู่ในคลาสพร้อมทั้งระบุชื่อสเตทและทรานสิชั่นภายในที่มีในสเตทนั้น วิธีการสร้างสเตทอยู่ในบทที่ 2 หัวข้อ 2.2.6.1
3. สร้างทรานสิชั่นไลน์ระหว่างสเตทที่มีการเปลี่ยนแปลงจากสเตทหนึ่งไปยังอีกสเตทหนึ่งพร้อมทั้งระบุอีเวนท์และการกระทำเมื่อเกิดอีเวนท์ (ถ้ามี)
4. ถ้าต้องการแสดงรายละเอียดของสเตท สามารถทำเป็นคอมโพสิทสเตทได้ซึ่งภายในจะเป็นสเตทย่อยๆของสเตทนั้น

ขั้นตอนที่ 6 แอ็กทิวิตี้ไดอะแกรม

ใช้แอ็กทิวิตี้ไดอะแกรมในการบอกรายละเอียดการทำงานของคลาส

1. วาดสัญลักษณ์จุดเริ่มต้นและจุดสิ้นสุดการทำงาน
2. วาดสัญลักษณ์สำหรับแต่ละแอ็กทิวิตี้ที่มีในคลาส
3. ลากเส้นทรานสิชั่นเชื่อมระหว่างแอ็กทิวิตี้ที่ทำงานต่อเนื่องกัน
4. ถ้าในแอ็กทิวิตี้ไดอะแกรมมีการใช้แอ็กทิวิตี้ของหลายอ็อบเจ็คให้ใช้สวิมเลนเข้ามาช่วย
5. ถ้าต้องการแสดงรายระเอียดของแอ็กทิวิตี้ สามารถใช้แอ็กทิวิตี้ไดอะแกรมมาอธิบายแอ็กทิวิตี้นั้นได้

ทั้งสเตทชาร์ทไดอะแกรมและแอ็กทิวิตี้ไดอะแกรมต่างก็ใช้บอกรายละเอียดการทำงาน

ของคลาส มีวิธีการเลือกใช้ไดอะแกรมทั้งสองนี้คือ

ถ้าเป็นการเปลี่ยนแปลงหรือเหตุการณ์ต่างๆ อันเป็นผลมาจากการกระทำภายในคลาสเองซึ่งมีลักษณะต่อเนื่องกัน และมักมีจุดที่ระบบงานต้องมีการตัดสินใจ ให้ใช้แอ็กทิวิตี้ไดอะแกรม

ถ้าเป็นการเปลี่ยนแปลงหรือการกระทำที่เกิดขึ้นในลักษณะไม่ต่อเนื่องกัน ให้ใช้สเตทไดอะแกรม

ขั้นตอนที่ 7 ขยายรายละเอียด

หลังจากทั้งหกขั้นตอนแล้ว หากต้องการระบุรายละเอียดเพิ่มเติมของ

1. ยูสเคส - สามารถใช้สเตทไดอะแกรมหรือแอ็กทิวิตี้ไดอะแกรมมาแสดงรายละเอียดได้
2. โอเปอเรชั่นหรือเมธอด - สามารถใช้แอ็กทิวิตี้ไดอะแกรมมาแสดงรายละเอียดการทำงานได้

ขั้นตอนที่ 8 สร้างฐานข้อมูล (เชิงวัตถุสัมพันธ์)

1. จากคลาสไดอะแกรม คลาสที่อยู่ฝั่งฐานข้อมูลหรือที่มีสเตอริโอไทป์ <<entity>> กำกับอยู่จะถูกสร้างเป็นตารางโดยมีชนิดแถวข้อมูล (Row type) ที่สร้างด้วยคำสั่ง

   create row type type_name(……..)

   ดังที่อธิบายในบทที่ 4 โดยมีคอลัมน์เป็นแอททริบิวท์ตามที่ระบุในคลาสไดอะแกรม
2. โอเปอเรชั่นหรือเมธอดของคลาสจะถูกสร้างขึ้นด้วยคำสั่ง

   create function f_name(type_name arg1,….)

   โดย type_name คือชื่อของคลาสที่มีโอเปอเรชั่นหรือเมธอดนี้อยู่
3. คลาสที่เป็นส่วนหนึ่งของอีกคลาสหนึ่ง (ความสัมพันธ์แบบอักกรีเกชั่น) จะถูกสร้างเป็นแอททริบิวท์หนึ่งของคลาสหลักโดยจะมีลักษณะเป็นชนิดข้อมูลแบบสะสมหรือไม่ก็ขึ้นอยู่กับมัลติพลิซิตี้ที่ระบุในคลาสไดอะแกรม
4. คลาสที่เป็นสับคลาสของอีกคลาสหนึ่ง (ความสัมพันธ์แบบเจนเนอรัลไลเซชั่น) จะสร้างชนิดแถวข้อมูลสำหรับคลาสนั้นขึ้นโดยใช้การสืบทอดคุณสมบัติ (Inheritance) ดังที่อธิบายในบทที่ 4
5. คลาสที่เป็นสับคลาสจะมีโอเปอเรชั่นหรือเมธอดของซูเปอร์คลาส(ดูวิธีการสร้างในบทที่ 4 หัวข้อที่ 4.2.3)

ขั้นตอนที่ 9 แอสโซสิเอชั่น

แอสโซสิเอชั่นระหว่างคลาสฝั่งฐานข้อมูลจะเป็น Foreign key หรือข้อมูลชนิดอ้างอิง (Reference type) ระหว่างสองคลาสนั้นในฐานข้อมูล

ถ้ามีความสัมพันธ์แบบ 1 to many ให้นำคีย์หลัก (Primary Key) ของคลาสฝั่งที่มีมัลติพลิซิตี้เป็น 1 ไปเพิ่มเป็น Foreign Key ลงในคลาสฝั่งที่มีมัลติพลิซิตี้เป็น many

ถ้ามีความสัมพันธ์แบบ many to many ให้สร้างตารางใหม่โดยในตารางจะมีแอททริบิวท์เป็นคีย์หลักของทั้งสองคลาส

ขั้นตอนที่ 10 สร้างคลาสที่อยู่ฝั่งแอปพลิเคชั่น

1. จากคลาสไดอะแกรม คลาสที่อยู่ฝั่งแอปพลิเคชั่นหรือที่มีสเตอริโอไทป์ <<boundary>> กำกับอยู่จะถูกสร้างเป็นคลาสอยู่ในแอปพลิเคชั่น โดยมีแอททริบิวท์และโอเปอเรชั่นหรือเมธอดตามที่ระบุในคลาสไดอะแกรม
2. คลาสที่เป็นส่วนหนึ่งของอีกคลาสหนึ่ง (ความสัมพันธ์แบบอักกรีเกชั่น) จะถูกสร้างเป็นแอททริบิวท์หนึ่งของคลาสหลักโดยจะจำนวนเท่าไรก็ขึ้นอยู่กับมัลติพลิซิตี้ที่ระบุในคลาสไดอะแกรม
3. คลาสที่เป็นสับคลาสของอีกคลาสหนึ่ง (ความสัมพันธ์แบบเจนเนอรัลไลเซชั่น) จะถูกสร้างขึ้นโดยใช้การสืบทอดคุณสมบัติซึ่งมีวิธีการต่างกันขึ้นอยู่กับภาษาที่ใช้เขียนแอปพลิเคชั่นนั้น

ขั้นตอนที่ 11 รายละเอียดการทำงานของโอเปอเรชั่น

สามารถดูได้จากแอ็กทิวิตี้ไดอะแกรมที่เขียนเพื่ออธิบายโอเปอเรชั่นนั้น(ถ้ามี) หรือจากแอ็กทิวิตี้ไดอะแกรมหรือสเตทไดอะแกรมที่บอกการทำงานของคลาส

ขั้นตอนที่ 12 ตรวจสอบความถูกต้อง

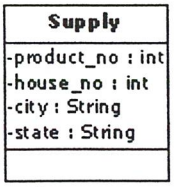
เพื่อความตรงกันระหว่างโมเดล UML ที่ออกแบบกับแอปพลิเคชั่นที่สร้างขึ้น

1. คลาสที่มีในโมเดล UML จะต้องมีอยู่ในแอปพลิเคชั่น และคลาสที่ไม่มีในโมเดล UML จะต้องไม่มีในแอปพลิเคชั่น
2. แอททริบิวท์ของคลาสในแอปพลิเคชั่นจะต้องตรงกับที่ระบุไว้ในคลาสในโมเดล UML
3. โอเปอเรชั่นหรือเมธอดต่างๆในแอปพลิเคชั่นจะต้องมีการทำงานตามที่ระบุในแอ็กทิวิตี้ไดอะแกรมในโมเดล UML
4. อ็อบเจ็คของคลาสต่างๆจะต้องมีการติดต่อกันโดยมีแมสเสจหรือตัวกระตุ้นตรงกับที่เขียนไว้ในซีเควนซ์ไดอะแกรม
5. การทำงานรวมของระบบจะต้องส่งผลให้สามารถทำงานได้เหมือนกับที่ระบุเป็นยูสเคสในยูสเคสไดอะแกรม โดยระบบจะต้องไม่มีการทำงานที่ไม่มีการออกแบบไว้

5.2 ปัญหาในการใช้ UML ออกแบบแอปพลิเคชั่น

หลังจากที่ได้ศึกษาการออกแบบแอปพลิเคชั่นโดยใช้โมเดล UML และทดลองสร้างแอปพลิเคชั่นแล้ว พบว่าการใช้โมเดล UML มีปัญหาดังนี้

1. คลาสไดอะแกรมในโมเดล UML ไม่สามารถแสดงความสัมพันธ์ระหว่างแอททริบิวท์ภายในคลาสได้ (Functional Dependency หรือ FD) ซึ่งอาจทำให้เกิดค่าของแอททริบิวท์ที่ซ้ำซ้อนได้ ดังตัวอย่างต่อไปนี้

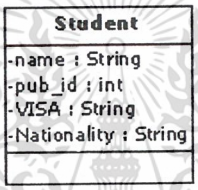


รูปที่ 5.1 คลาสที่มี FD ระหว่างแอททริบิวท์

จากตัวอย่างจะเห็นว่าค่าของ city จะ determine state เสมอ เนื่องจากเราไม่ทราบความสัมพันธ์ของแอททริบิวท์ทั้งสองในคลาส

วิธีแก้ไขสามารถทำได้โดยการแยกแอททริบิวท์นั้นออกมาสร้างเป็นคลาสใหม่

2. ปัญหาความสัมพันธ์ระหว่างแอททริบิวท์ในคลาสและระหว่างแอททริบิวท์กับคลาส ปัญหานี้เกิดจากผู้อิมพลิเมนต์ไม่ทราบความสัมพันธ์ระหว่างแอททริบิวท์สองตัวหรือมากกว่านั้นว่ามีความสัมพันธ์ หรือมีกฎในการป้อนอินพุตอย่างไร ดังตัวอย่างต่อไปนี้



รูปที่ 5.2 คลาสที่เกิดปัญหาปัญหาความสัมพันธ์ระหว่างแอททริบิวท์ในคลาส

จากตัวอย่าง ถ้ากำหนดว่านักศึกษาที่เป็นคนไทย ให้กรอกหมายเลขบัตรประชาชน ไม่เช่นนั้นให้กรอกหมายเลข VISA ซึ่งในโมเดลนี้ ไม่สามารถบอกได้

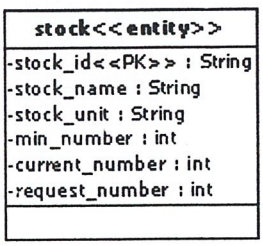
วิธีแก้ไขสามารถทำได้โดยใช้ OONIAM โดยใช้ Exclusion Constraint

3. ในคลาสไดอะแกรมของ UML มีความสัมพันธ์แบบอักกริเกชั่น แต่ไม่อาจบอกได้ว่าคลาสที่เป็นส่วนหนึ่งของอีกคลาสนั้นถูกฝัง (Embedded) อยู่ในอีกคลาสหนึ่งเลยหรือไม่

วิธีแก้ไขสามารถทำได้โดยใช้ OONIAM ซึ่งมีสับสกีมมา (Subschema) ซึ่งใช้บอกว่าเป็นการฝัง แต่ถ้าอยู่ในเมนสกีมมา (Main Schema) แสดงว่าไม่ใช่การฝัง

4. ในคลาสไดอะแกรมไม่สามารถบอกได้ว่าแอททริบิวท์ใดในคลาสเป็นคีย์หลัก (Primary Key) ในการสร้างตาราง

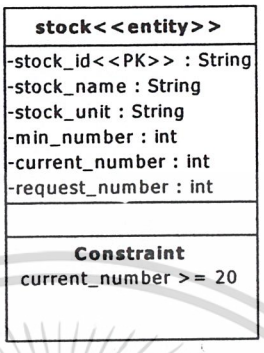
วิธีแก้ไขทำได้โดยใช้สเตอริโอไทป์กำกับไว้ที่แอททริบิวท์ที่เป็นคีย์หลักเช่น



รูปที่ 5.3 การใช้สเตอริโอไทป์บอกว่าแอททริบิวท์ใดเป็นคีย์หลัก

จากรูปใช้ <<PK>> เป็นสเตอริโอไทป์บอกว่า stock_id เป็นคีย์หลัก

5. ใน UML ไม่สามารถกำหนดข้อบังคับ (Constraint) ได้อย่างชัดเจน การกำหนดข้อบังคับของ UML อาจทำได้ทางหนึ่งคือใช้ user-defined compartment ของคลาสเช่น



รูปที่ 5.4 การกำหนดข้อบังคับให้แอททริบิวท์ในคลาส

แต่วิธีนี้อาจใช้ไม่ได้กับข้อบังคับที่มีความซับซ้อน

วิธีแก้ไขสามารถทำได้โดยใช้ OONIAM ซึ่งมีการกำหนดข้อบังคับได้อย่างชัดเจน

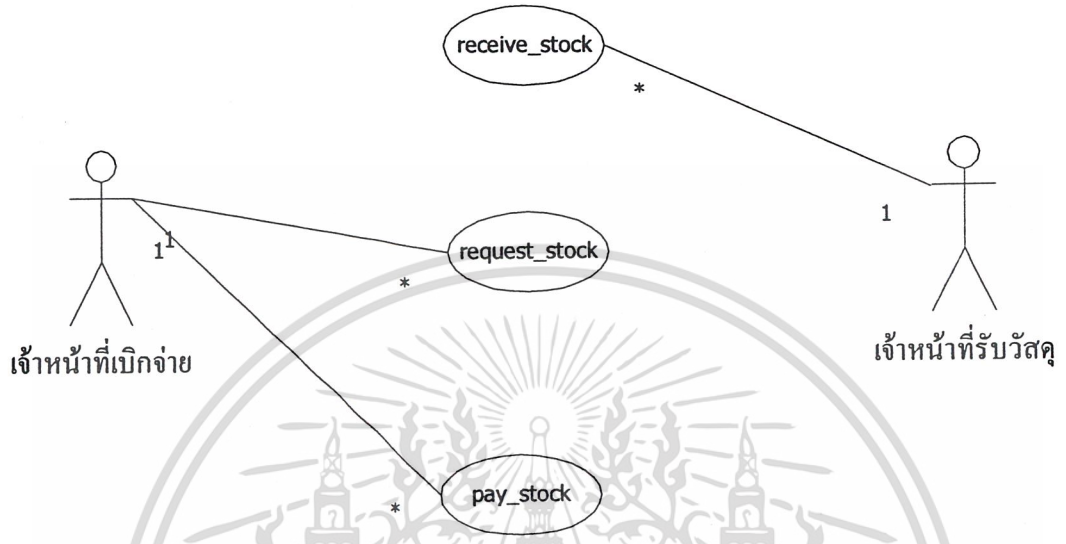
6. เนื่องจาก UML ไม่ได้บอกแน่นอนว่าจะต้องเขียนไดอะแกรมลงไปละเอียดเพียงใด ซึ่งจะเป็นปัญหาเมื่อผู้ออกแบบไม่ทราบถึงความชำนาญของผู้เขียนโปรแกรมและเขียนไดอะแกรมไม่ละเอียดพอ ทำให้ผู้เขียนโปรแกรมไม่สามารถสร้างแอปพลิเคชั่นจากการออกแบบนั้นได้

7. หลังจากการออกแบบ เมื่อนำไปสร้างเป็นแอปพลิเคชั่น อาจเกิดปัญหาทั้งทางด้านไวยากรณ์ของภาษาที่ใช้เขียนและเทคนิคการเขียนโปรแกรม ทำให้ต้องกลับมาแก้ไขไดอะแกรมของ UML ใหม่ เช่น ในการออกแบบกำหนดให้เมื่อรับวัสดุชิ้นหนึ่งต้องเก็บลงฐานข้อมูลทันที แต่เมื่อนำไปเขียนเป็นแอปพลิเคชั่นแล้วพบว่าทำให้เกิดความยุ่งยากในการจัดการกับฐานข้อมูล ทำให้ต้องกลับมาแก้ไขโดยใช้อาร์เรย์เก็บไว้ชั่วคราว แล้วจึงเก็บลงฐานข้อมูลเมื่อการรับวัสดุครั้งนั้นครบทุกชิ้นแล้ว

บทที่ 6

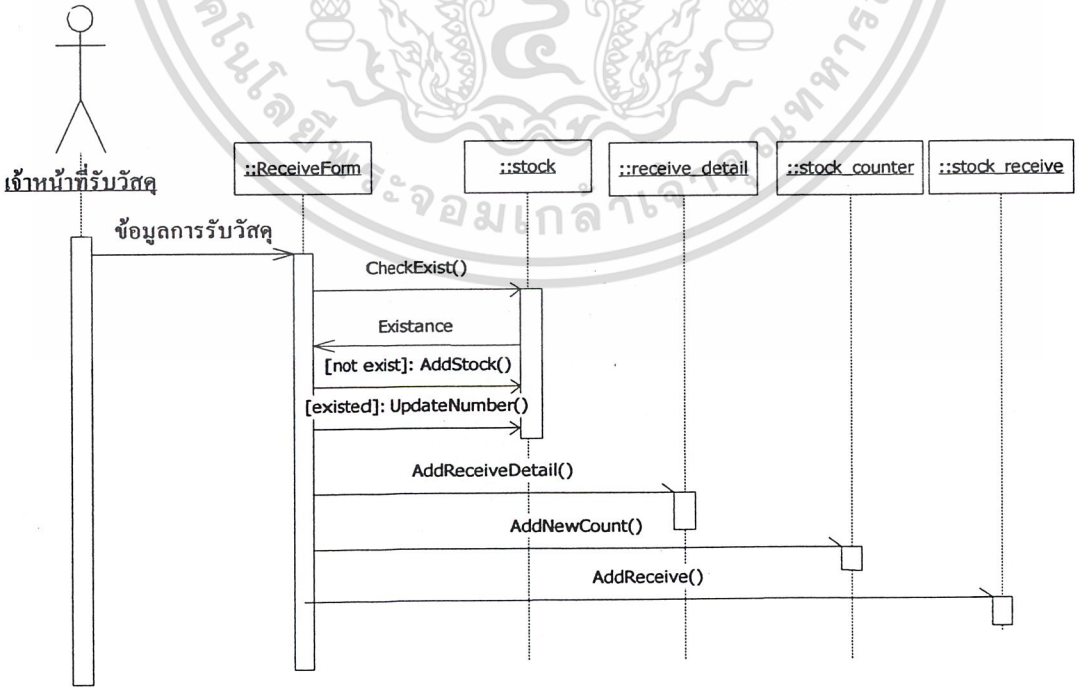
ไดอะแกรม UML และ OONIAM ของกรณีศึกษาระบบงานควบคุมและตรวจสอบพัสดุ

ระบบงานควบคุมและตรวจสอบวัสดุมีการทำงานหลัก 3 อย่างดังแสดงในยูสเคสไดอะแกรมดังนี้

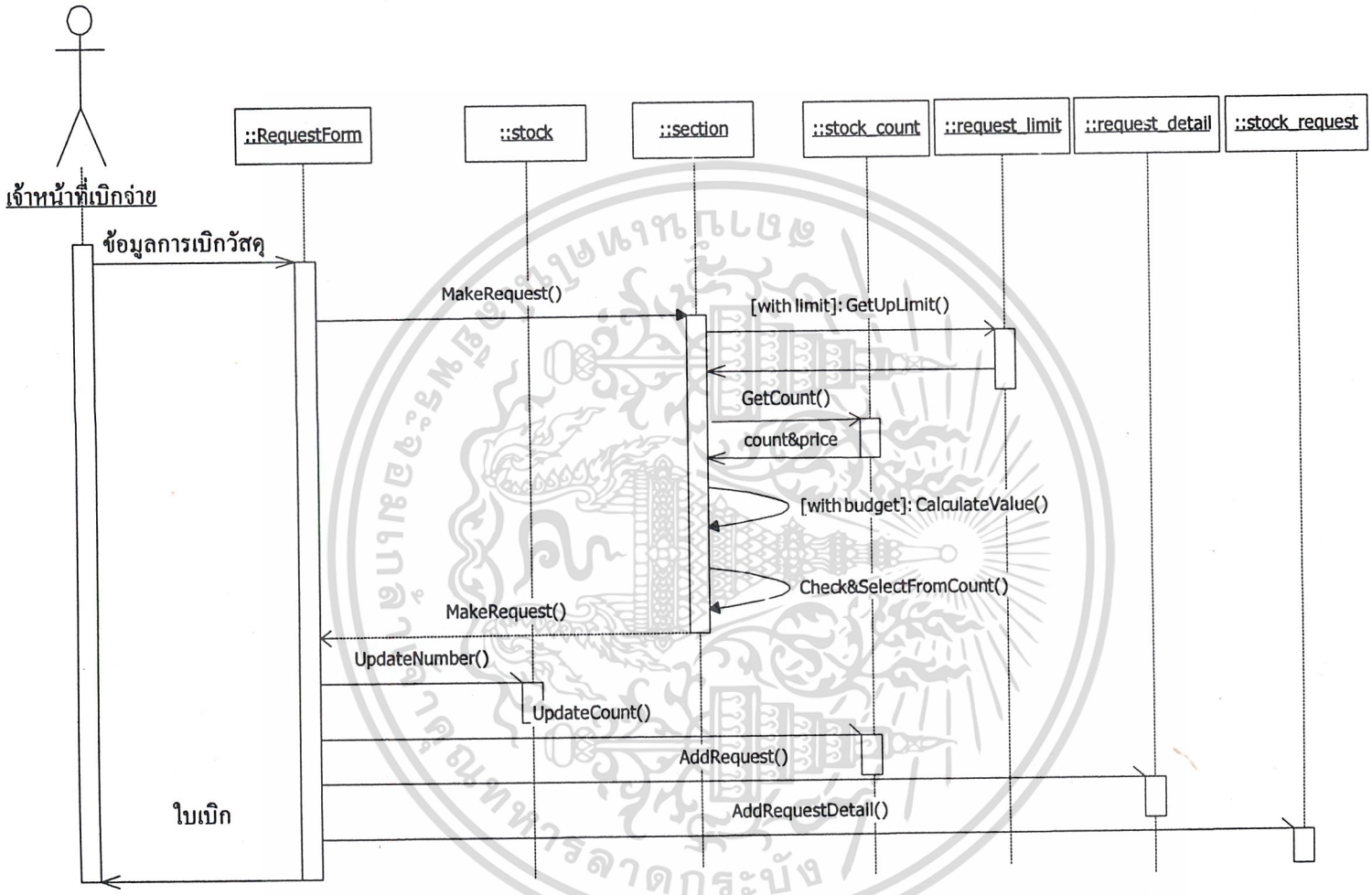


รูปที่ 6.1 ยูสเคสไดอะแกรมของระบบงานควบคุมและตรวจสอบวัสดุ

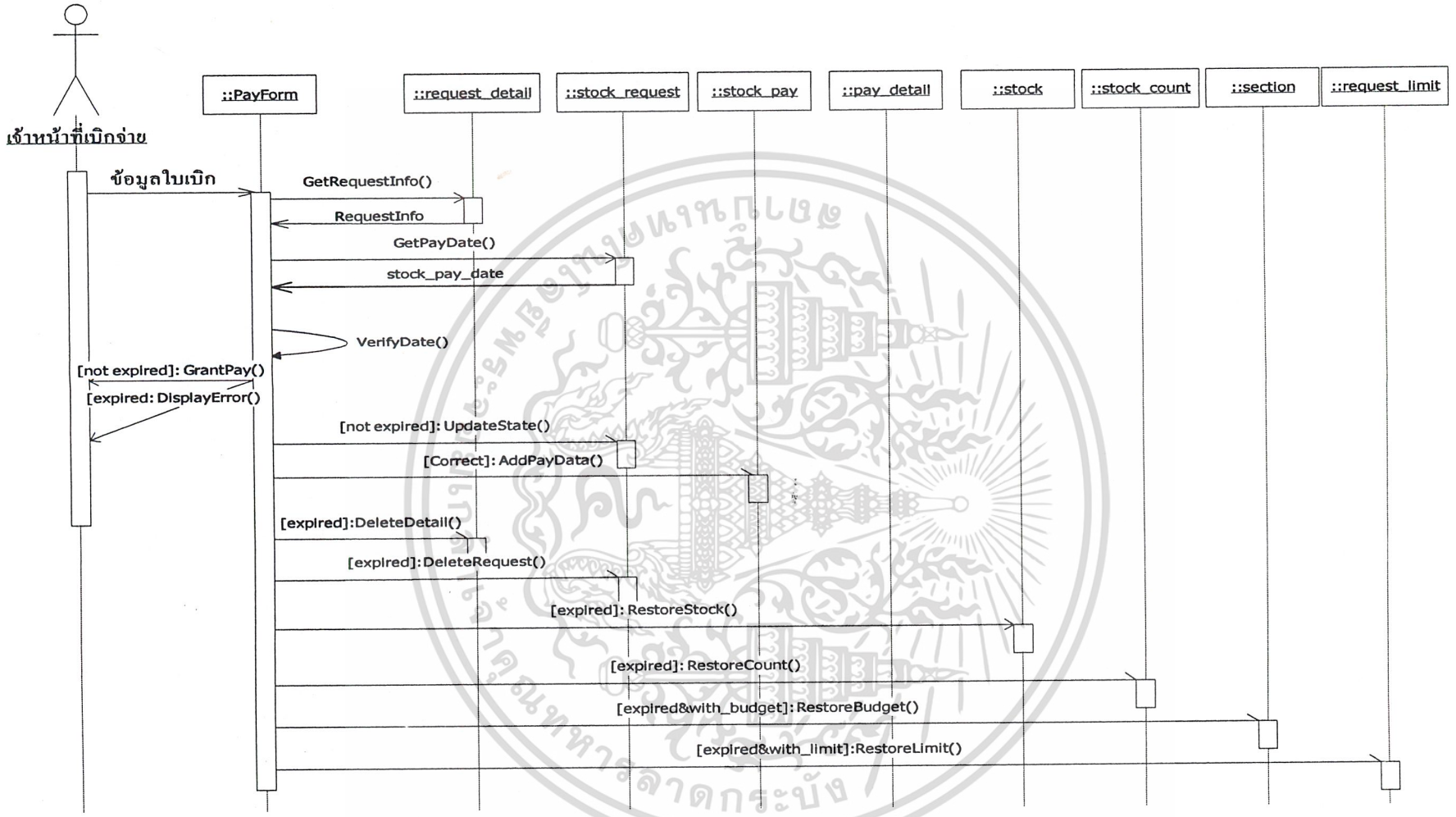
ยูสเคส receive_stock คือการรับวัสดุเข้า ยูสเคส request_stock คือการเบิกวัสดุ ยูสเคส pay_stock คือการจ่ายวัสดุ จากยูสเคสทั้งสามสามารถเขียนซีเควนซ์ไดอะแกรมแสดงการทำงานได้ดังนี้



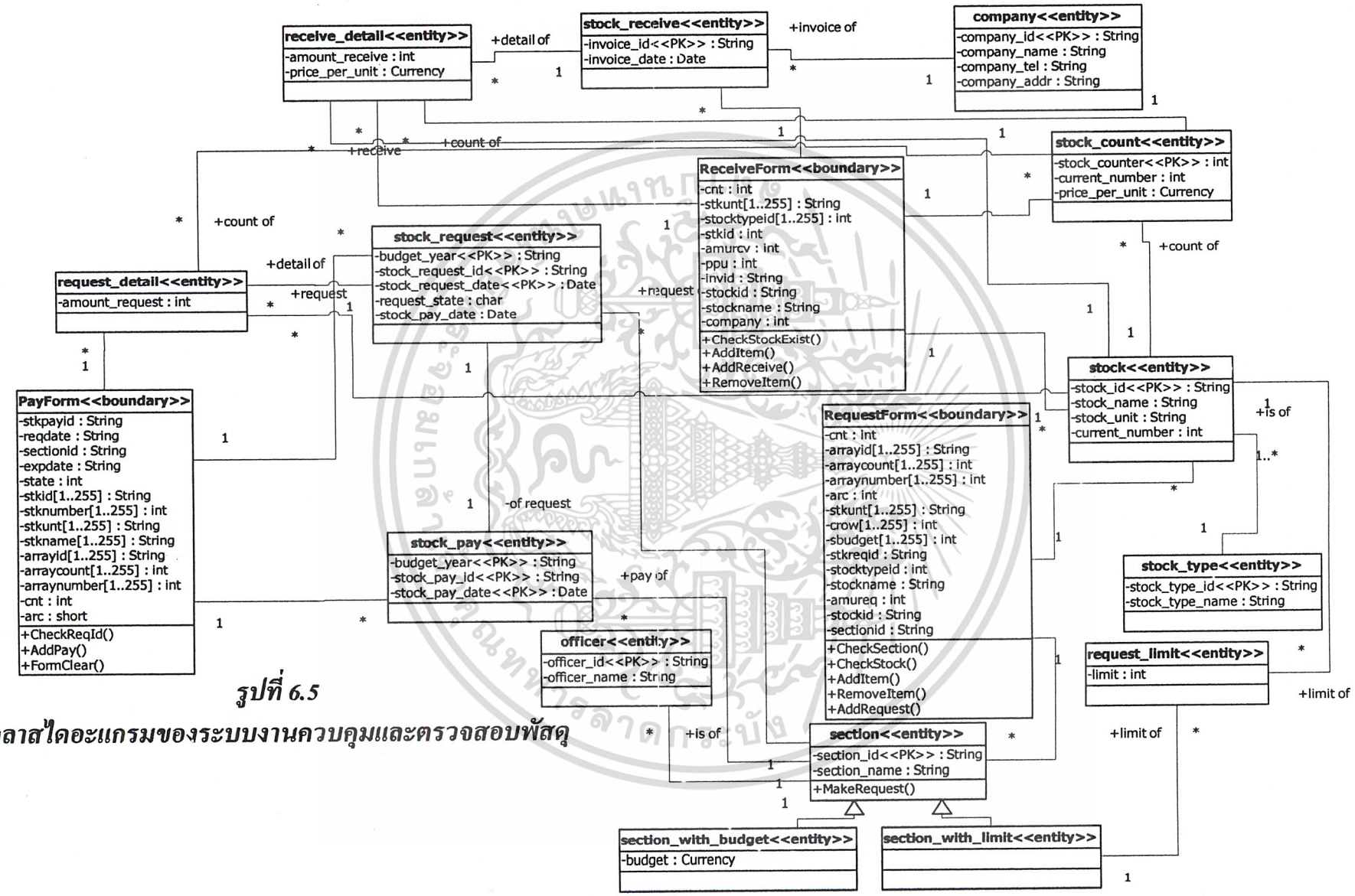
รูปที่ 6.2 ซีเควนซ์ไดอะแกรมของยูสเคส stock_receive



รูปที่ 6.3 ซีเควนซ์ไดอะแกรมของยูสเคส stock_request

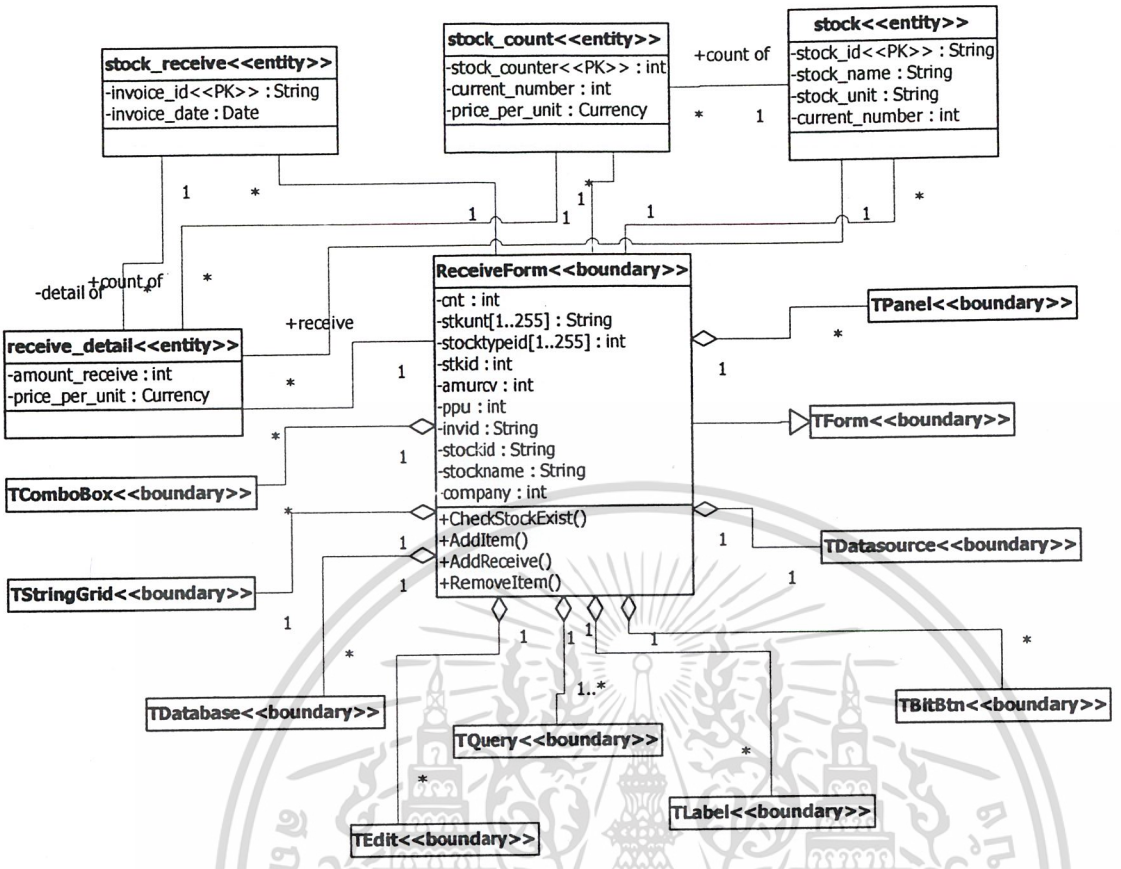


รูปที่ 6.4 ซีเควนซ์ไดอะแกรมของยูสเคส stock_pay

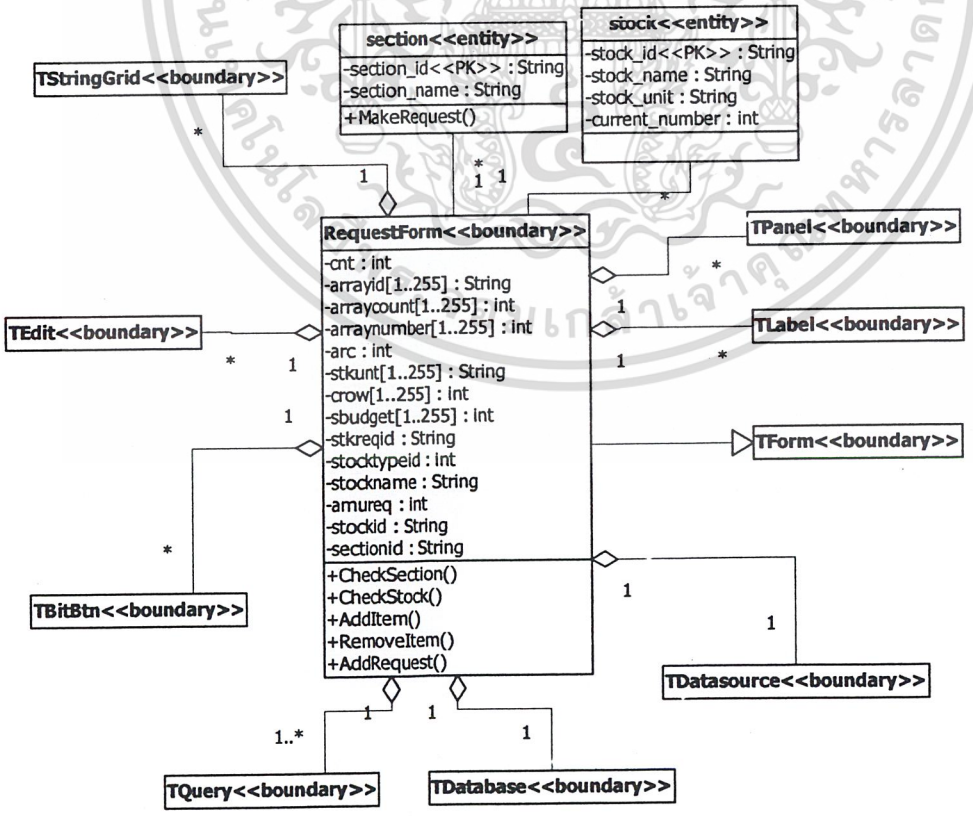


รูปที่ 6.5

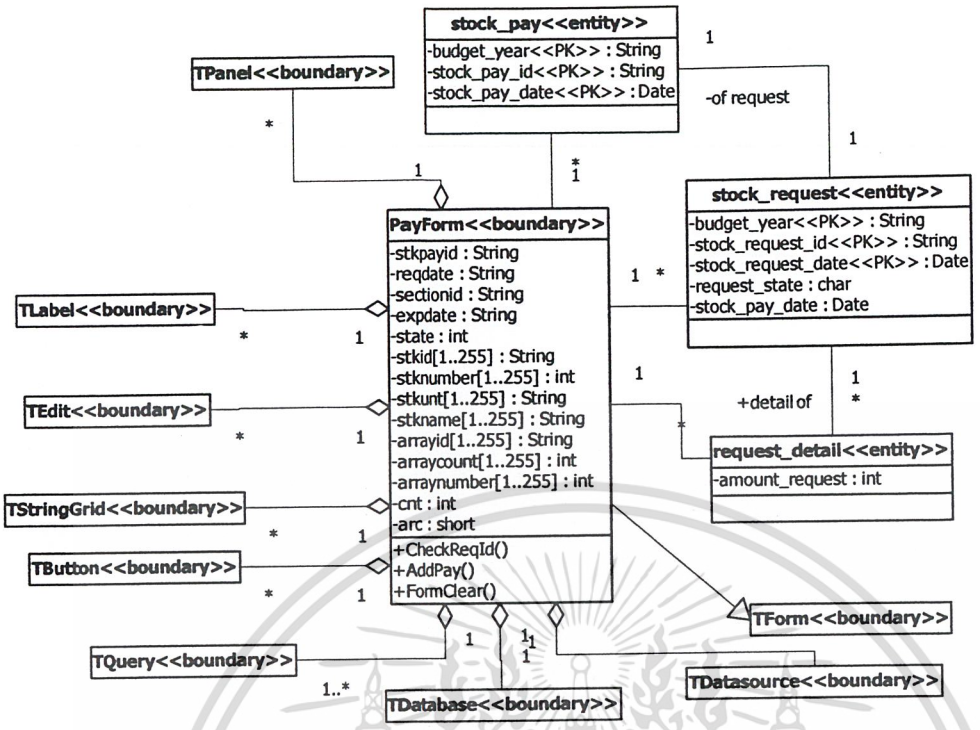
คลาสไดอะแกรมของระบบงานควบคุมและตรวจสอบพัสดุ



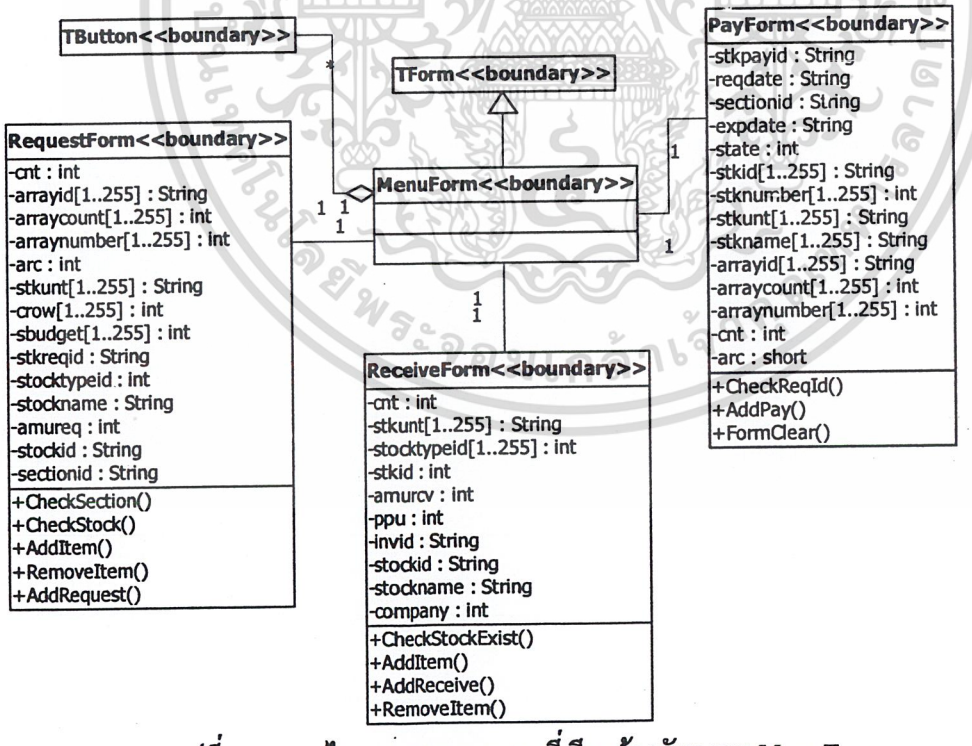
รูปที่ 6.6 คลาสไดอะแกรมของคลาสที่เกี่ยวข้องกับคลาส ReceiveForm



รูปที่ 6.7 คลาสไดอะแกรมของคลาสที่เกี่ยวข้องกับคลาส RequestForm

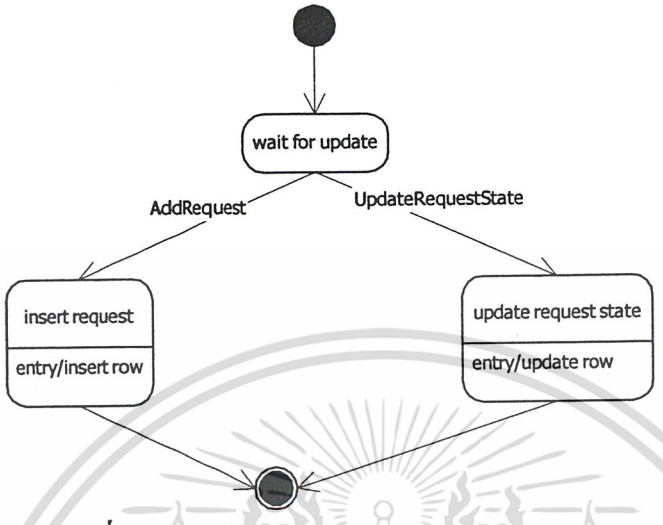


รูปที่ 6.8 คลาสไดอะแกรมของคลาสที่เกี่ยวข้องกับคลาส PayForm



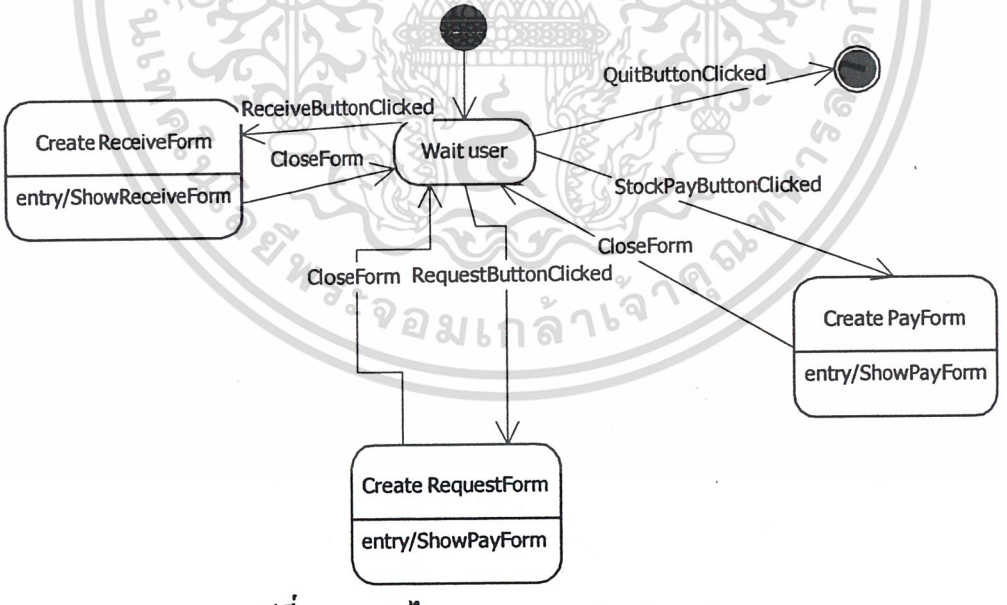
รูปที่ 6.9 คลาสไดอะแกรมของคลาสที่เกี่ยวข้องกับคลาส MenuForm

สเตทไดอะแกรมและแอ็กทิวิตี้ไดอะแกรมมีดังนี้ และเนื่องจากคลาสที่อยู่ฝั่งฐานข้อมูลส่วนใหญ่เป็นคลาสที่ไม่มีการทำงาน ดังนั้นทางผู้จัดทำจึงไม่สร้างสเตทไดอะแกรมและแอ็กทิวิตี้ไดอะแกรมสำหรับคลาสเหล่านี้แต่จะมีตัวอย่างแสดงในรูปที่ 6.10

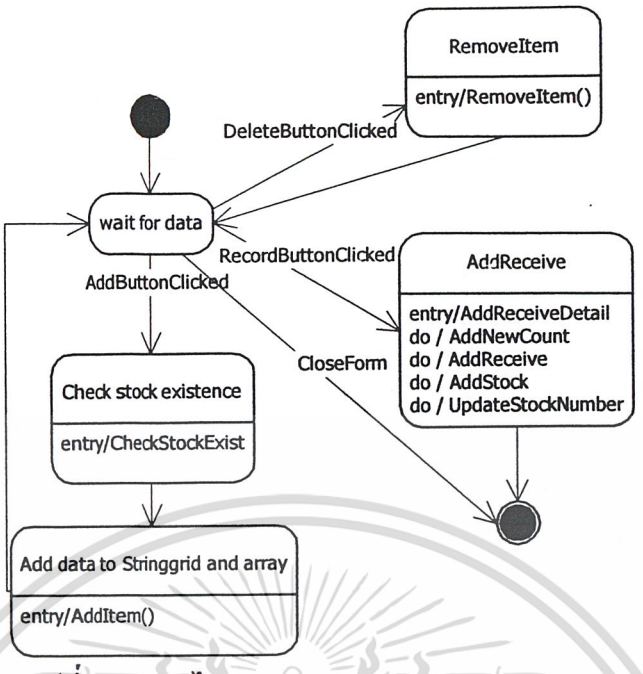


รูปที่ 6.10 สเตทไดอะแกรมของคลาส stock_request

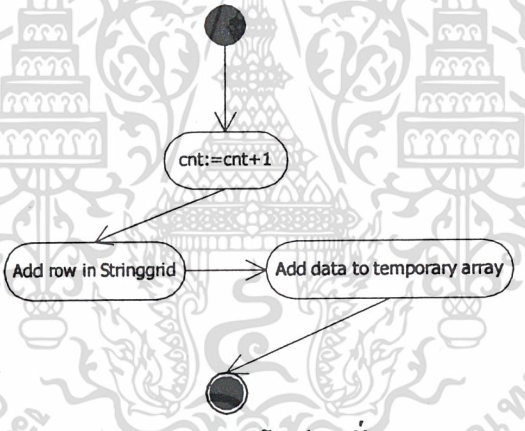
จะเห็นได้ว่าคลาส stock_request นี้ไม่มีการทำงานของตัวเอง สเตท insert request และ update request state เป็นเพียงการใช้คำสั่ง SQL เพื่อการ insert และ update



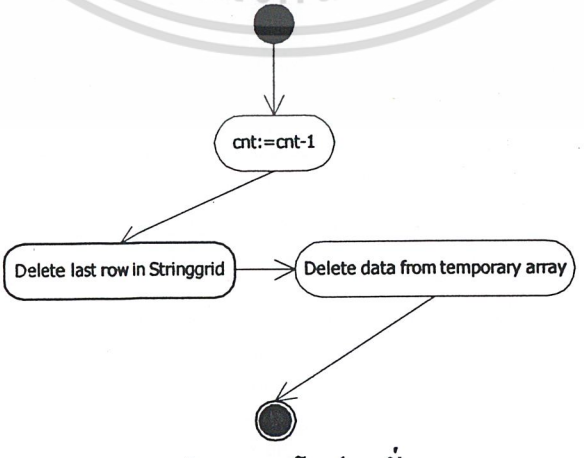
รูปที่ 6.11 สเตทไดอะแกรมของคลาส MenuForm



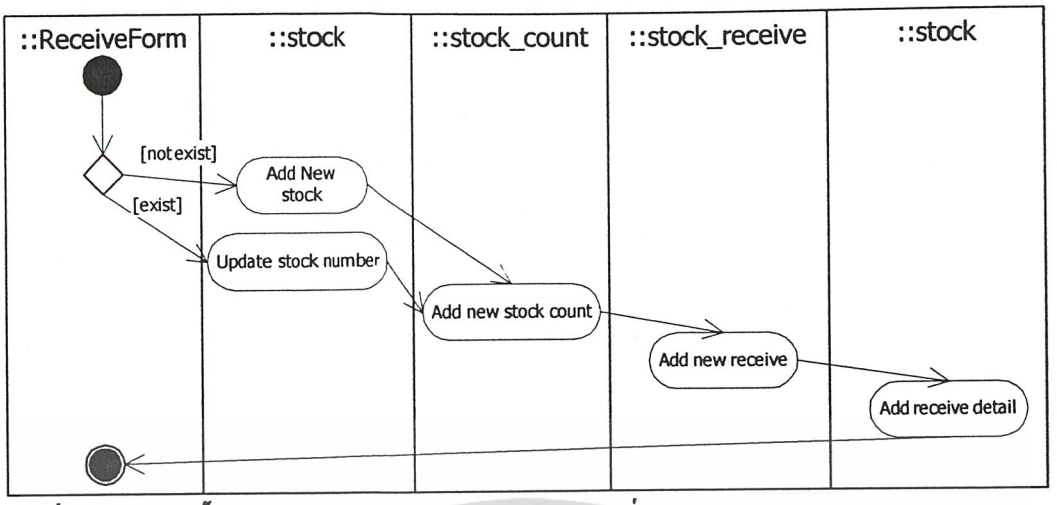
รูปที่ 6.12 สเตทไดอะแกรมของคลาส ReceiveForm



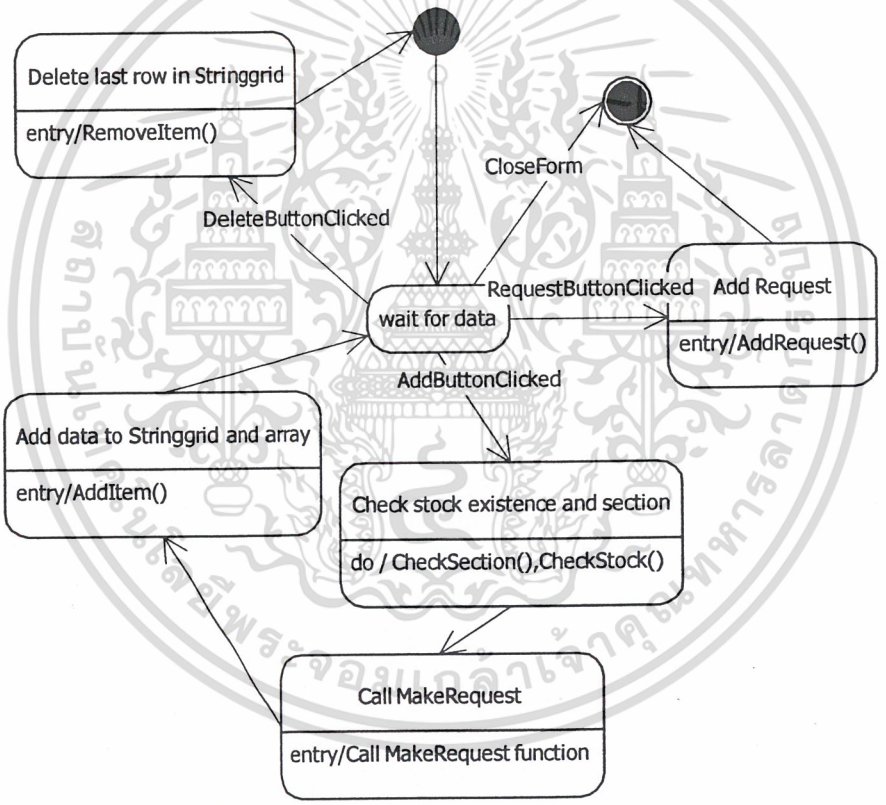
รูปที่ 6.13 แอ็กทิวิตี้ไดอะแกรมแสดงการทำงานของโอเปอเรชั่น AddItem ของคลาส ReceiveForm และ RequestForm



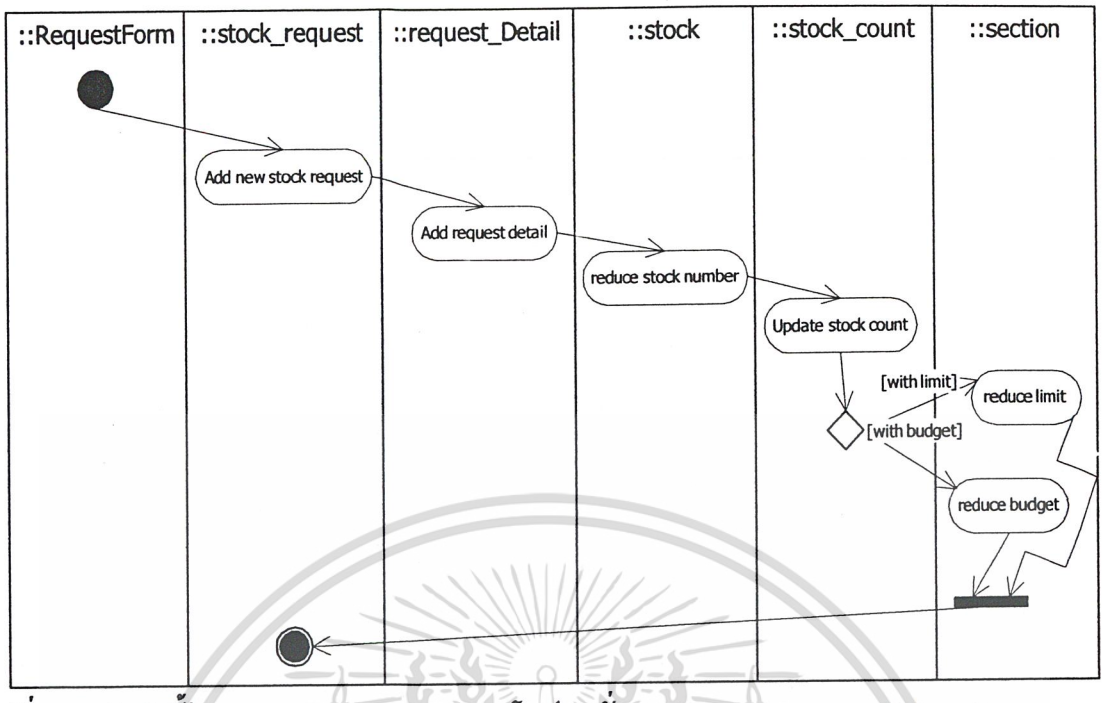
รูปที่ 6.14 แอ็กทิวิตี้ไดอะแกรมแสดงการทำงานของโอเปอเรชั่น RemoveItem ของคลาส ReceiveForm และ RequestForm



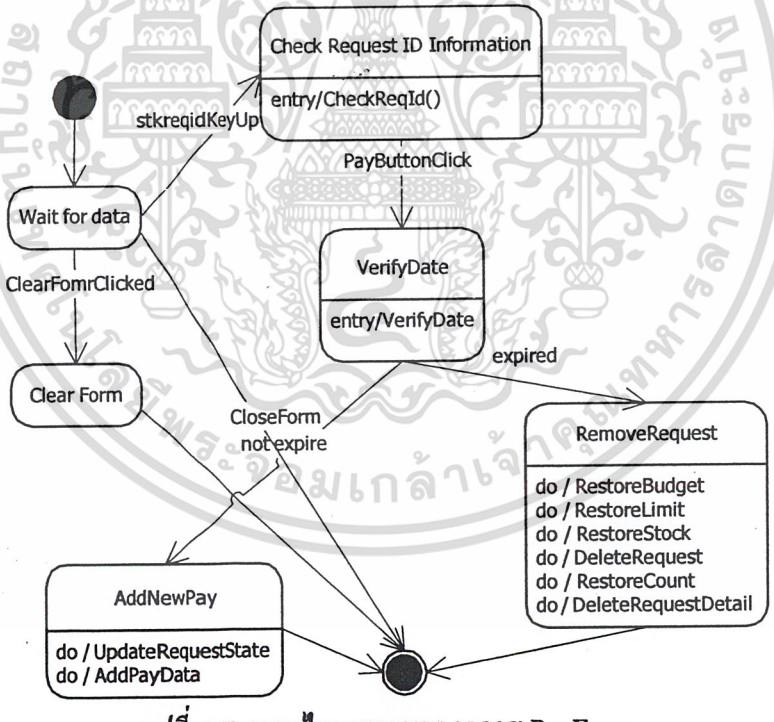
รูปที่ 6.15 แอ็กทิวิตี้ไดอะแกรมแสดงการทำงานของโอเปอเรชั่น AddReceive ของคลาส ReceiveForm



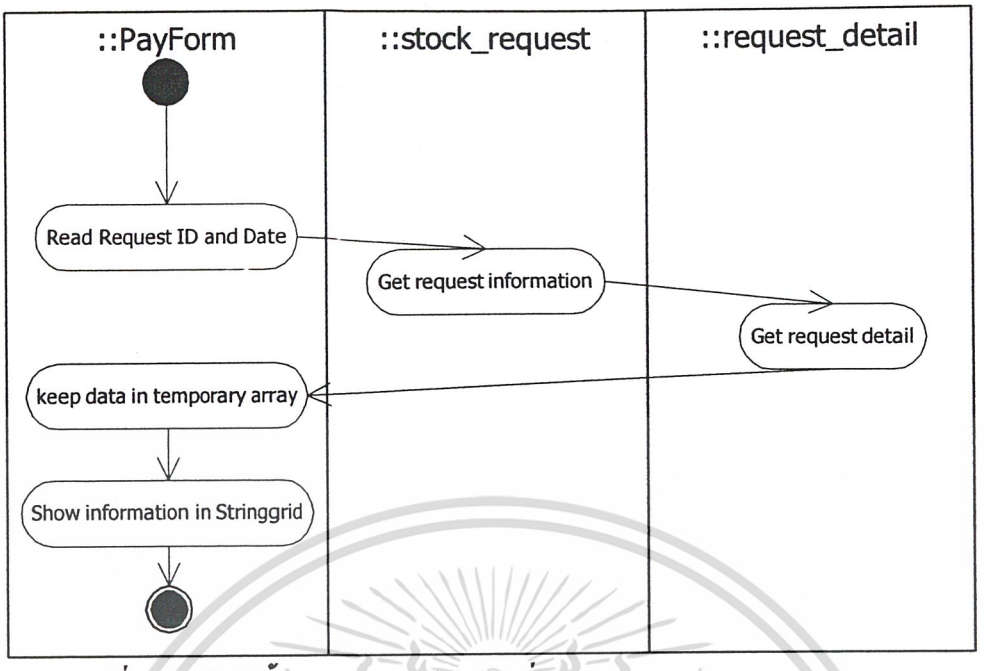
รูปที่ 6.16 สเตทไดอะแกรมของคลาส RequestForm



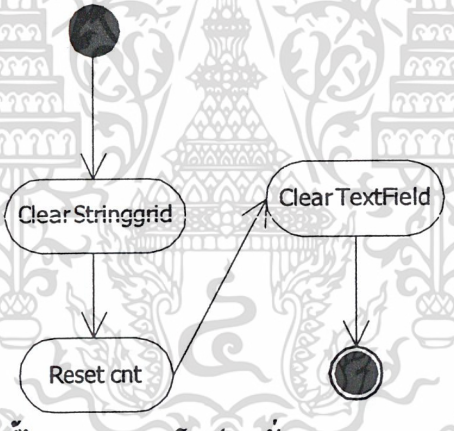
รูปที่ 6.17 แอ็กทิวิตี้ไดอะแกรมแสดงการทำงานของโอเปอเรชั่น AddRequest ของคลาส RequestForm



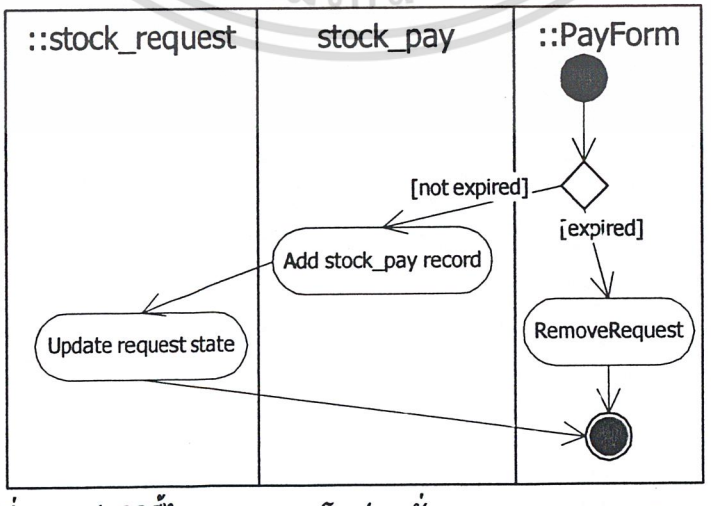
รูปที่ 6.18 สเตทไดอะแกรมของคลาส PayForm



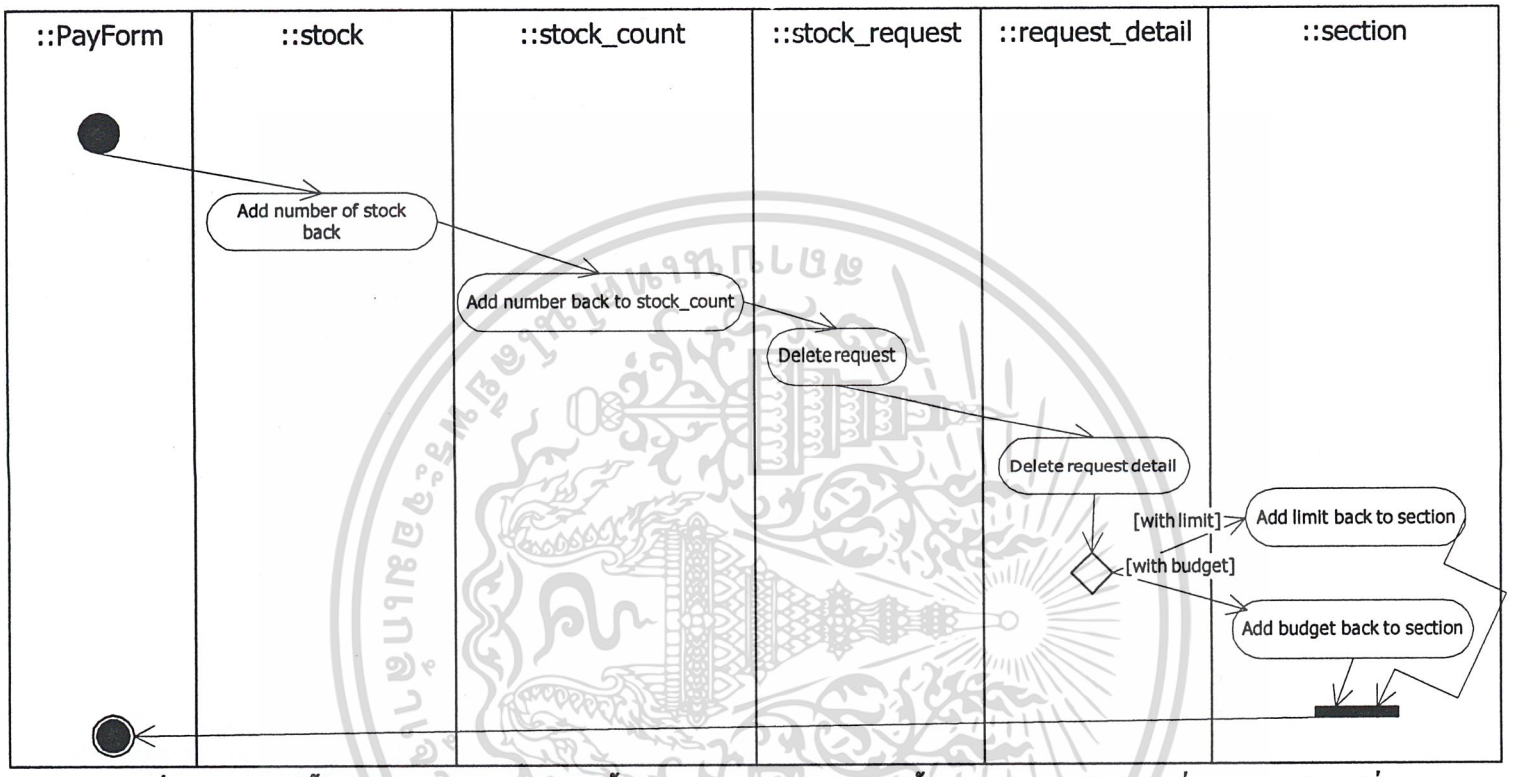
รูปที่ 6.19 แอ็กทิวิตี้ไดอะแกรมของโอเปอเรชั่น CheckReqId ของคลาส PayForm



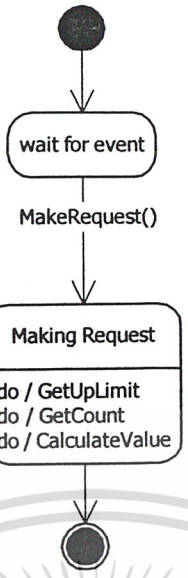
รูปที่ 6.20 แอ็กทิวิตี้ไดอะแกรมของโอเปอเรชั่น FormClear ของคลาส PayForm



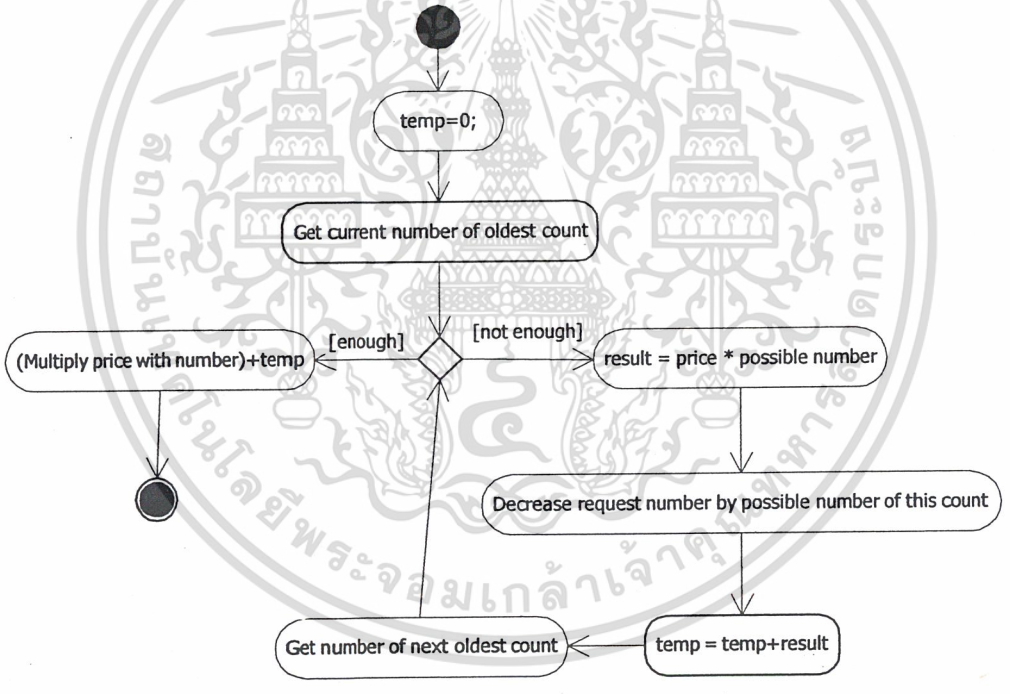
รูปที่ 6.21 แอ็กทิวิตี้ไดอะแกรมของโอเปอเรชั่น AddPay ของคลาส PayForm



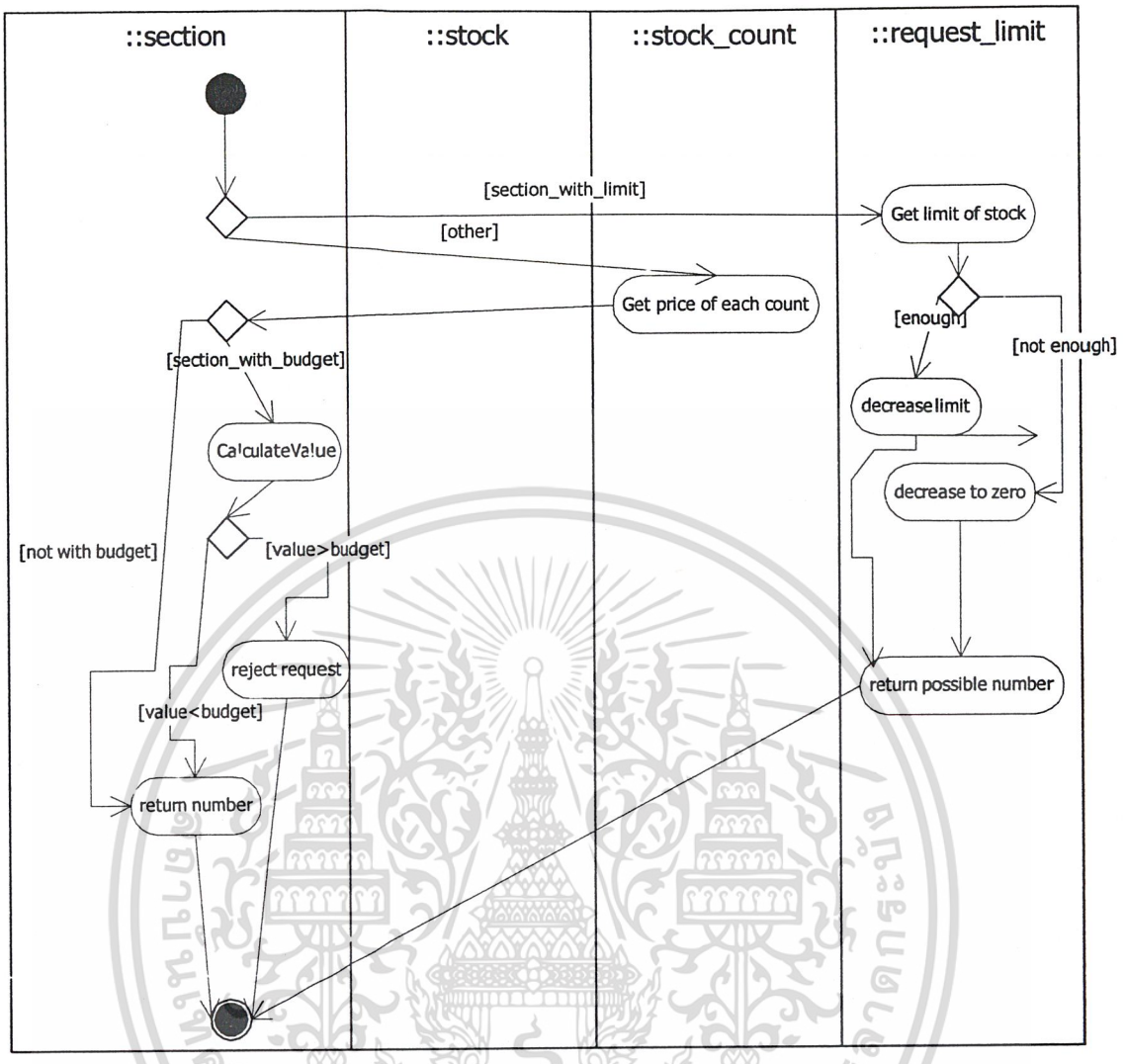
*รูปที่ 6.22 แอ็กทิวิตี้ไดอะแกรมย่อยของแอ็กทิวิตี้ **RemoveRequest** ในแอ็กทิวิตี้ไดอะแกรมของโอเปอเรชั่น **AddPay** ในรูปที่ 6.21*



รูปที่ 6.23 สเตทไดอะแกรมของคลาส section



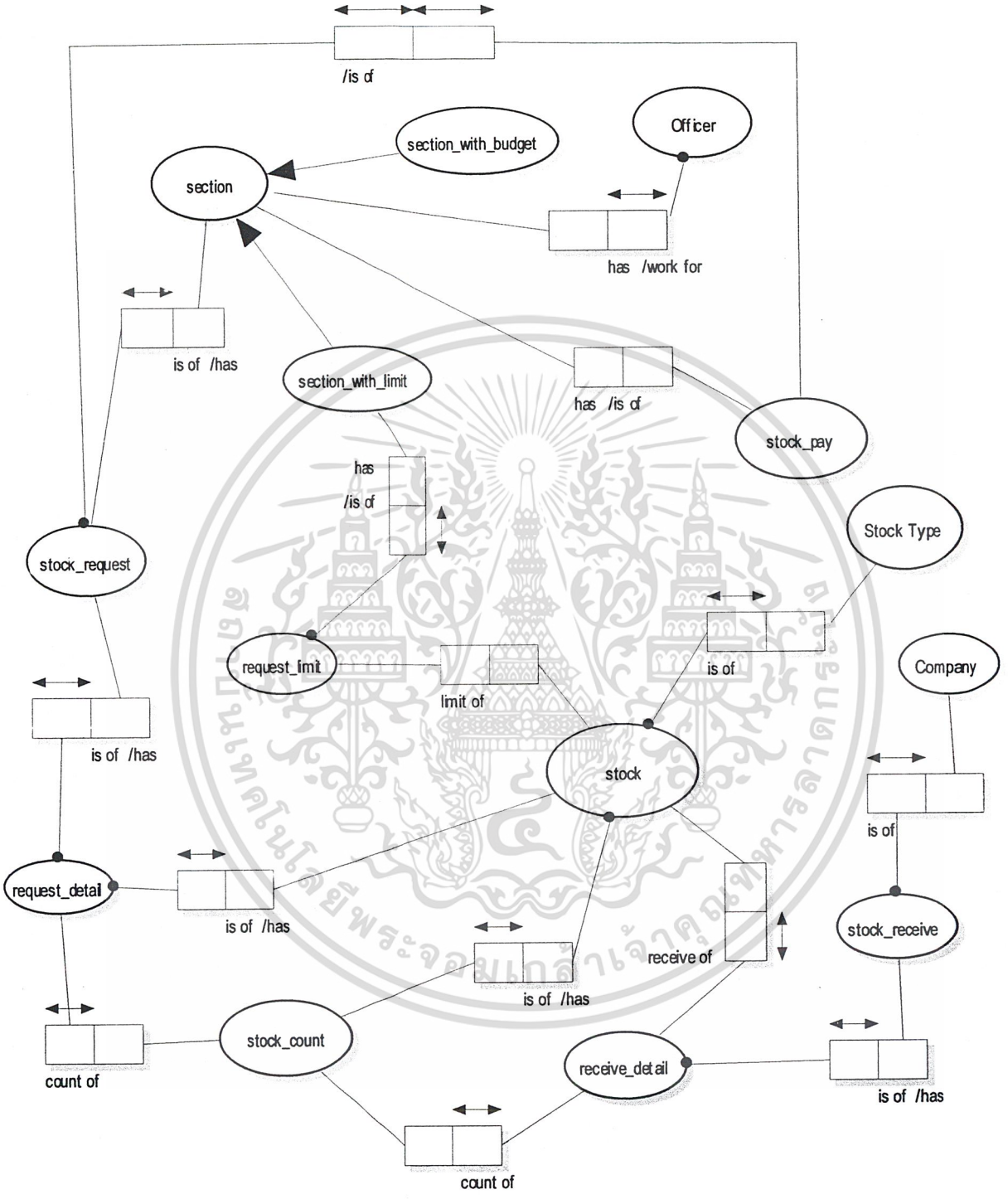
รูปที่ 6.24 แอ็กทิวิตี้ไดอะแกรมย่อยของแอ็กทิวิตี้ CalculateValue ในแอ็กทิวิตี้ไดอะแกรมแสดงการทำงานของโอเปอเรชั่น MakeRequest ในรูปที่ 6.25



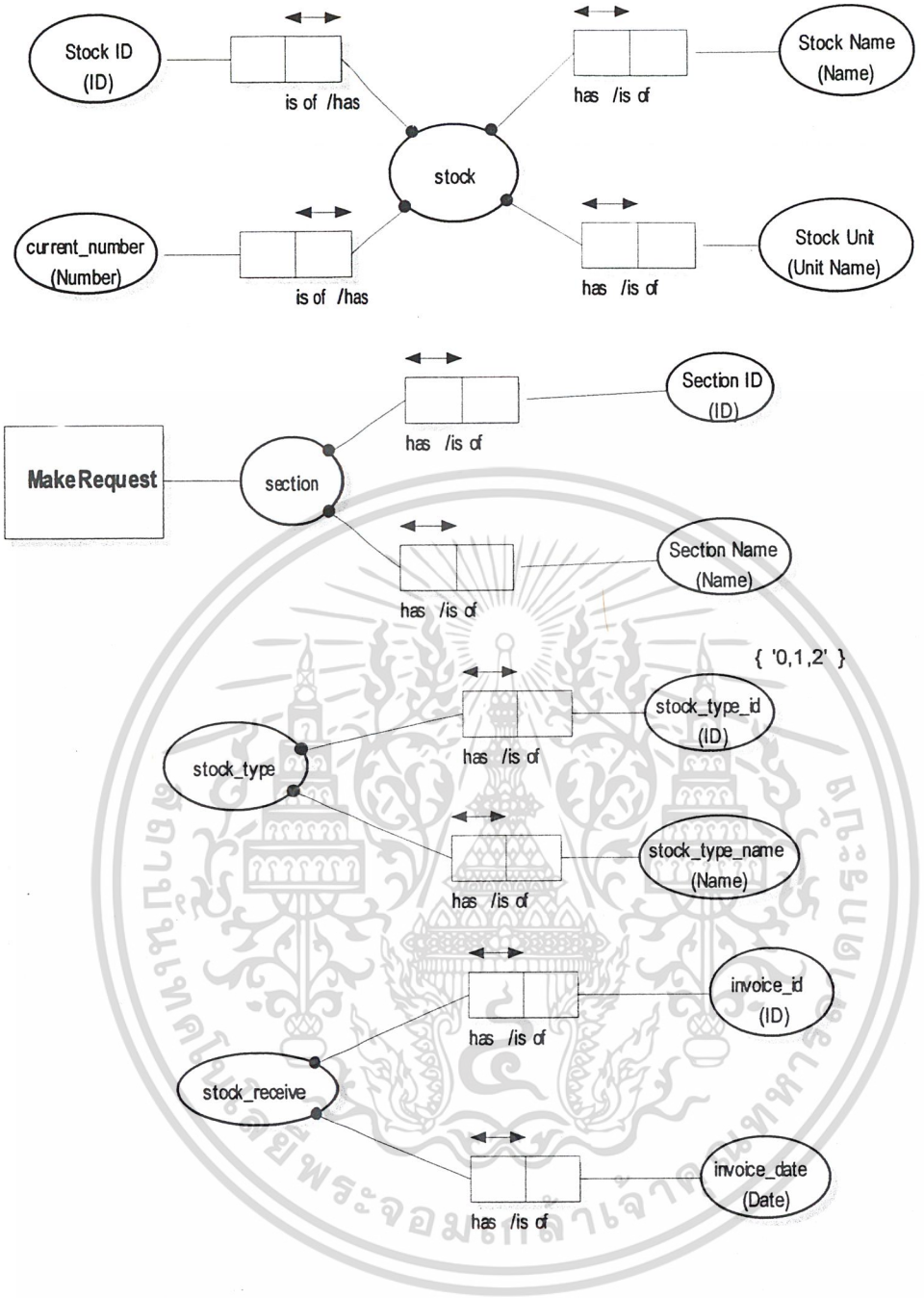
รูปที่ 6.25 แอ็กทิวิตี้ไดอะแกรมแสดงการทำงานของโอเปอเรชั่น MakeRequest ในคลาส section

- การรับวัสดุเข้า เมื่อกดปุ่มเพิ่มวัสดุ จะนำรายละเอียดของวัสดุนั้นเก็บในอาร์เรย์ชั่วคราว จนเมื่อเพิ่มวัสดุรับเข้ามาครั้งนั้นเสร็จแล้วจึงกดปุ่มบันทึกเพื่อนำข้อมูลที่เก็บไว้ในอาร์เรย์ไปเก็บลงในฐานข้อมูล
- การเบิกวัสดุ เมื่อกดปุ่มเพิ่มวัสดุที่จะเบิก จะนำรายละเอียดของการเบิกวัสดุนั้น เช่น จำนวน วัสดุนั้นเป็นการรับเข้ามาครั้งที่เท่าไร (เก็บอยู่ใน stock_count) มาเก็บไว้ในอาร์เรย์ชั่วคราว จนเมื่อเพิ่มวัสดุที่ต้องการเบิกครบแล้ว จึงกดปุ่มบันทึกเพื่อนำข้อมูลที่เก็บไว้ในอาร์เรย์ไปเก็บลงในฐานข้อมูล
- การจ่ายวัสดุ เมื่อใส่หมายเลขการเบิกและวันที่เบิก จะค้นหารายละเอียดของการเบิกนั้นมาเก็บไว้ในอาร์เรย์ชั่วคราวพร้อมทั้งแสดงรายละเอียดทางหน้าจอ เมื่อกดปุ่มเพื่อบันทึกการจ่ายจะเทียบว่าการเบิกนั้นหมดอายุแล้วหรือยัง ถ้ายังจะเปลี่ยนสถานการณ์เบิกและบันทึกการจ่ายลงในฐานข้อมูล ถ้าเลยระยะเวลาที่กำหนดแล้วจะนำข้อมูลที่เก็บไว้ในอาร์เรย์มาบวกค่าจำนวนต่างๆกลับ และลบการเบิกนั้นออกจากฐานข้อมูล

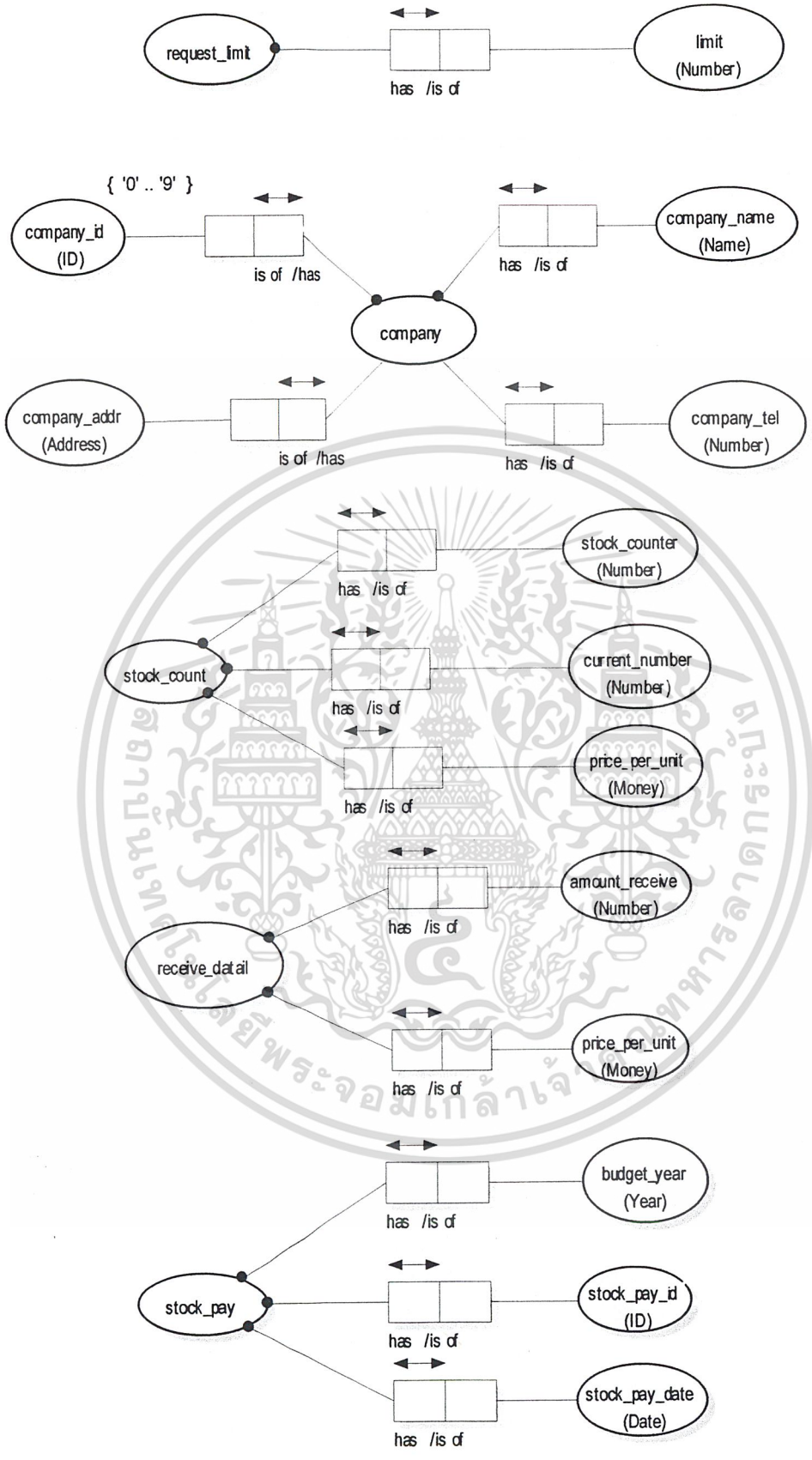
เพื่อแก้ปัญหาดังที่กล่าวไว้ในบทที่ 5 จึงได้ใช้ OONIAM แทนคลาสไดอะแกรมได้ดังนี้



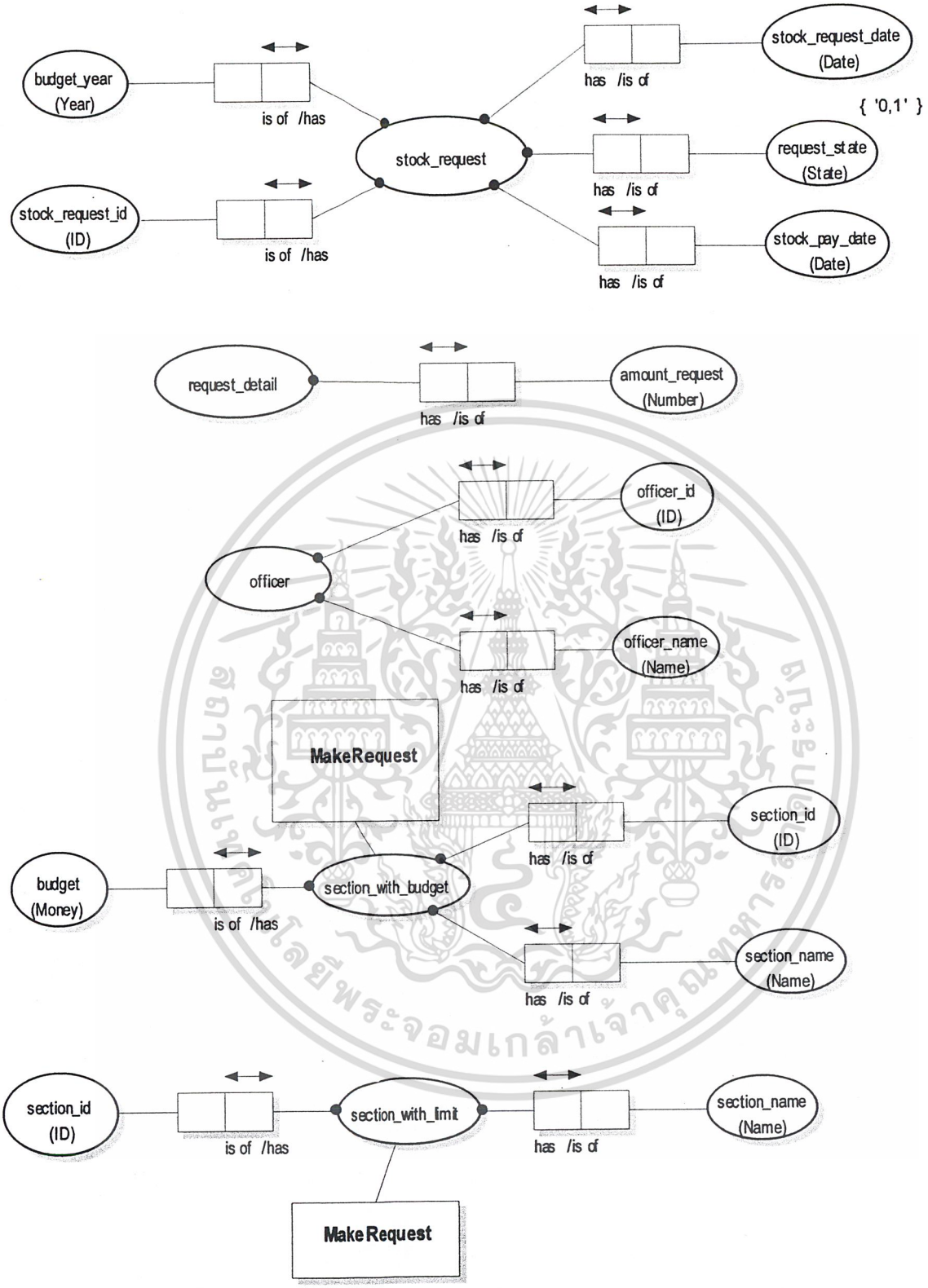
รูปที่ 6.26 Main Schema แสดงความสัมพันธ์ระหว่างคลาสในระบบงานควบคุมและตรวจสอบวัสดุ



รูปที่ 6.27 Sub Schema แสดงรายละเอียดของคลาสใน Main Schema



รูปที่ 6.28 Sub Schema แสดงรายละเอียดของคลาสใน Main Schema (ต่อ)



รูปที่ 6.29 Sub Schema แสดงรายละเอียดของคลาสใน Main Schema (ต่อ)

บทที่ 7
การทำให้แอปพลิเคชั่นสามารถแก้ไขเข้ากับกฎขององค์กรโดยใช้การสืบทอดคุณสมบัติ

7.1 การทำให้แอปพลิเคชั่นของกรณีศึกษาสามารถแก้ไขเข้ากับกฎขององค์กร

เป้าหมายของการนำการสืบทอดคุณสมบัติมาใช้คือ การทำให้แอปพลิเคชั่นสามารถแก้ไขเข้ากับกฎขององค์กรโดยแก้ไขเฉพาะส่วนฐานข้อมูลและแก้ไขโปรแกรมให้น้อยที่สุด การแก้ไขฐานข้อมูลนั้นทำที่เดียวไม่ต้องแก้ไขที่เวิร์คสเตชั่น (workstation) ทุกเครื่อง ซึ่งเป็นการประหยัดกว่าการแก้ไขตัวโปรแกรมซึ่งต้องทำการแก้ไขทุกเวิร์คสเตชั่นที่ใช้

จากเนื้อหาในบทที่ 4 หัวข้อ 4.2.2 และ 4.2.3 เรื่องการสืบทอดคุณสมบัติของชนิดข้อมูลและฟังก์ชั่น จึงได้วิธีการทำให้แอปพลิเคชั่นสามารถแก้ไขเข้ากับกฎขององค์กร คือใช้การสืบทอดคุณสมบัติคลาสหรือไทป์บนฐานข้อมูลที่ต้องแก้ไขกฎ (ใช้ฟังก์ชั่นในการ implement กฎ) มาเป็นไทป์ใหม่ โดยปกติสับไทป์ที่สืบทอดคุณสมบัติมาเมื่อถูกเรียกฟังก์ชั่น ถ้าไม่มีฟังก์ชั่นนั้นอยู่ก็จะไปสืบทอดฟังก์ชั่นมาจากซูเปอร์ไทป์ (การสร้างฟังก์ชั่นของไทป์มีอยู่ในบทที่ 4 หัวข้อที่ 4.2.3) ฟังก์ชั่นหรือกฎของทั้งซูเปอร์ไทป์และสับไทป์จึงเหมือนกัน ดังนั้นการแก้ไขกฎจึงทำได้โดยการโอเวอร์โหลดฟังก์ชั่นที่ implement กฎที่ต้องการแก้ไข (Polymorphism) โดยใช้ฟังก์ชั่นที่มีชื่อเดียวกันแต่ส่งพารามิเตอร์เป็นสับไทป์แทน (เหมือนกับเป็นการสร้างฟังก์ชั่นให้กับสับไทป์)

ในกรณีศึกษาระบบงานควบคุมและตรวจสอบวัสดุ ในการเบิกโดยทั่วไปจะมี 2 ลักษณะคือ เบิกได้ไม่จำกัดโดยขึ้นอยู่กับการอนุมัติของเจ้าหน้าที่ที่มีอำนาจ การเบิกโดยแต่ละหน่วยงานมีงบประมาณกำหนดให้และเบิกได้โดยมูลค่าการเบิกต้องไม่เกินงบประมาณที่มีอยู่ นอกจากนี้ได้เพิ่มการเบิกโดยมีการจำกัดจำนวนวัสดุเพื่อเป็นตัวอย่างเพิ่มขึ้น ซึ่งได้นำทั้งสามแบบนี้มาสร้างเป็นแอปพลิเคชั่นตัวอย่าง แต่เนื่องจากไม่ได้สร้างส่วนการ customize ให้กับผู้ใช้ จึงได้รวมทั้งสามแบบไว้ในแอปพลิเคชั่นเดียว

การใช้การสืบทอดคุณสมบัติมีอยู่ที่คลาส section (หน่วยงานที่เบิกได้ไม่จำกัด) ซึ่งเป็นคลาสฝั่งฐานข้อมูล สืบทอดเป็นคลาส section_with_budget (หน่วยงานที่เบิกตามงบประมาณ) และคลาส section_with_limit (หน่วยงานที่เบิกได้จำกัดจำนวน) โดยทั้งสามคลาสจะมีฟังก์ชั่น MakeRequest โดยจะเป็นการ implement กฎในการเบิกของทั้งสามคลาส

ฟังก์ชั่น MakeRequest ของคลาส section จะดูเพียงว่ามีวัสดุให้เบิกพอหรือไม่ ถ้าไม่พอก็จะให้เบิกเท่าที่มีอยู่

ฟังก์ชั่น MakeRequest ของคลาส section_with_limit จะตรวจสอบว่าหน่วยงานนั้นสามารถเบิกวัสดุนั้นๆได้จำนวนเท่าใด ถ้าไม่พอก็จะให้เบิกเท่ากับจำนวนที่สามารถเบิกได้ โดยจะมีการเก็บจำนวนที่เบิกได้ไว้ในคลาส request_limit

ฟังก์ชั่น MakeRequest ของคลาส section_with_budget จะนำวัสดุที่เบิกมาคิดมูลค่าซึ่งจะเก็บอยู่ใน stock_count วัสดุชนิดหนึ่งในการรับเข้ามาแต่ละครั้งอาจมีราคาไม่เท่ากัน เช่น โต๊ะ เมื่อรับเข้ามาครั้ง

แรกราคา 1000 บาท แต่เมื่อรับเข้าครั้งที่สองราคา 900 บาท ดังนั้นจึงต้องมีการเก็บข้อมูลเหล่านี้เอาไว้เพื่อนำมาคิดมูลค่าการเบิก

ฟังก์ชั่น MakeRequest ของทั้งสามคลาสจะอยู่บนฐานข้อมูลเป็น user-defined function แต่เนื่องจากการสร้างฟังก์ชั่นให้กับไทป์จำเป็นต้องมีพารามิเตอร์ตัวหนึ่งที่ใช้บอกว่าเป็นของไทป์ใด ดังนั้นจึงจะต้องมีการแก้ไขตัวโปรแกรมในส่วนที่เรียกฟังก์ชั่น MakeRequest เพื่อให้เรียกฟังก์ชั่นของแต่ละคลาสอย่างถูกต้อง

ถ้าจะมีการเพิ่มกฎการเบิกขึ้นมาอีก หรือต้องการสร้างแอปพลิเคชั่นใหม่ที่มีกฎการเบิกแตกต่างออกไป ก็สามารถทำได้โดยสืบทอดคุณสมบัติจากไทป์ section มาเป็นไทป์ใหม่ จากนั้นโอเวอร์โหลดฟังก์ชั่น MakeRequest โดยใช้พารามิเตอร์ตัวหนึ่งเป็นชนิดของไทป์ใหม่นั้น แล้วแก้ไขโปรแกรมเพื่อเพิ่มให้มีการเรียกใช้ฟังก์ชั่น MakeRequest ของไทป์ใหม่ที่เกิดขึ้น

ตัวอย่างส่วนการสร้างฟังก์ชั่น MakeRequest ทั้งสามมีดังนี้ (ใช้เป็นโพรซีเยอร์ซึ่งไม่มีการคืนค่า)

```
CREATE PROCEDURE MakeRequest(Arg1 section);
CREATE PROCEDURE MakeRequest(Arg1 section_with_limit);
CREATE PROCEDURE MakeRequest(Arg1 section_with_budget);
```

7.2 ปัญหาและอุปสรรคที่เกิดขึ้นและแนวทางการแก้ไข

จากการนำการออกแบบไปสร้างเป็นแอปพลิเคชั่นจริง เกิดปัญหาเกี่ยวกับระบบจัดการฐานข้อมูล Informix ไม่ยอมให้มีการผ่านพารามิเตอร์เป็นชนิดข้อมูลหรือไทป์ ดังนั้นแม้ว่าระบบจัดการฐานข้อมูล Informix จะสามารถทำการสืบทอดคุณสมบัติของไทป์ได้ ก็ไม่สามารถสร้างฟังก์ชั่นของแต่ละไทป์ได้ ทำให้ไม่สามารถทำได้ตามที่ออกแบบไว้ซึ่งควรจะเป็นไปได้ในทางทฤษฎี

การแก้ไขที่ได้ทำในแอปพลิเคชั่นคือ ใช้การสืบทอดคุณสมบัติของไทป์ section เหมือนเดิมบนฐานข้อมูล และสร้างฟังก์ชั่น MakeRequest 3 ฟังก์ชั่นสำหรับแต่ละไทป์ไว้ในโปรแกรม เมื่อได้รับรหัสหน่วยงานก็จะนำไปดูว่าหน่วยงานนั้นอยู่ในไทป์ใด แล้วเรียก MakeRequest สำหรับไทป์นั้น การแก้ไขแบบนี้จำเป็นต้องมีการแก้ไขโปรแกรมค่อนข้างมากซึ่งไม่ตรงตามเป้าหมายที่ตั้งไว้นัก

บทที่ 8
บทสรุป

ผลจากการดำเนินการวิจัย ในช่วงที่เริ่มดำเนินการระยะแรกได้สังเกตเห็นปัญหาหลายประการของการออกแบบโดย UML แต่เมื่อได้ศึกษา UML อย่างละเอียดมากขึ้นกลับได้รู้ว่า UML สามารถแก้ปัญหาเหล่านั้นได้อยู่แล้ว สุดท้ายจึงได้ข้อที่สามารถปรับปรุง UML ได้ตามที่ได้กล่าวไว้ในบทที่ 5 ซึ่งปัญหาเหล่านี้ส่วนใหญ่ได้มาในระหว่างการทดลองสร้างแอปพลิเคชั่นและการค้นคว้ารายละเอียดของ UML

ส่วนการใช้คุณสมบัติการถ่ายทอดของแนวคิดเชิงวัตถุมาช่วยในการสร้างแอปพลิเคชั่นที่สามารถปรับเข้ากับกฎต่างๆได้นั้น พบว่ามีเอกสารอ้างอิงถึงวิธีทำอยู่ซึ่งได้นำมาบรรจุไว้ในบทที่ 4 ซึ่งสามารถนำมาใช้ในการทำแอปพลิเคชั่นที่สามารถแก้ไขให้เข้ากับกฎขององค์ได้ในทางทฤษฎี แต่มีปัญหาทางด้านระบบจัดการฐานข้อมูล จึงไม่สามารถนำมาทดลองสร้างได้จริง และได้เสนอแนวการแก้ไขไว้ในบทที่ 7

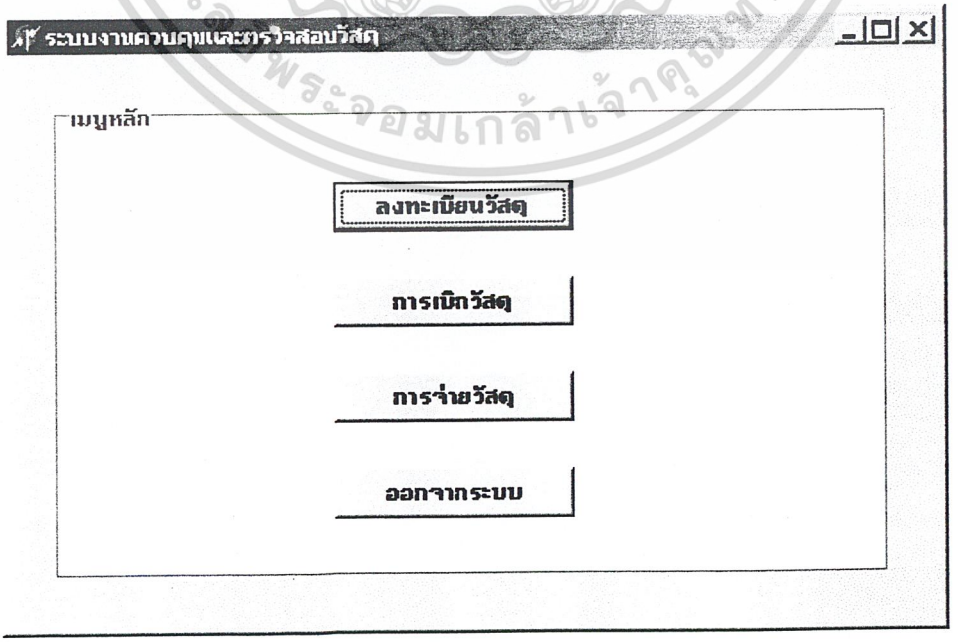
แนวทางในการพัฒนาต่อไปคือ ใน UML มีการใช้ OCL (Object Constraint Language) ซึ่งโครงงานนี้ได้ศึกษาไปไม่ถึง OCL สามารถใช้กำหนดกฎข้อบังคับต่างๆสำหรับ UML ได้ จึงเสนอให้มีการศึกษาในเรื่อง OCL ต่อไป เพื่อเปรียบเทียบกับการใช้ OONIAM ว่าควรใช้วิธีใดในการออกแบบเพื่อให้ผู้เขียนโปรแกรมสามารถเข้าใจการออกแบบนั้นและสามารถสร้างแอปพลิเคชั่นได้อย่างถูกต้อง

ในด้านการใช้การสืบทอดคุณสมบัติ ซึ่งในโครงการนี้ประสบปัญหาด้านระบบจัดการฐานข้อมูล จึงเสนอให้ทำการทดลองสร้างต่อไปโดยใช้ระบบจัดการฐานข้อมูลเชิงวัตถุสัมพันธ์อื่นเพื่อพิสูจน์ทฤษฎี

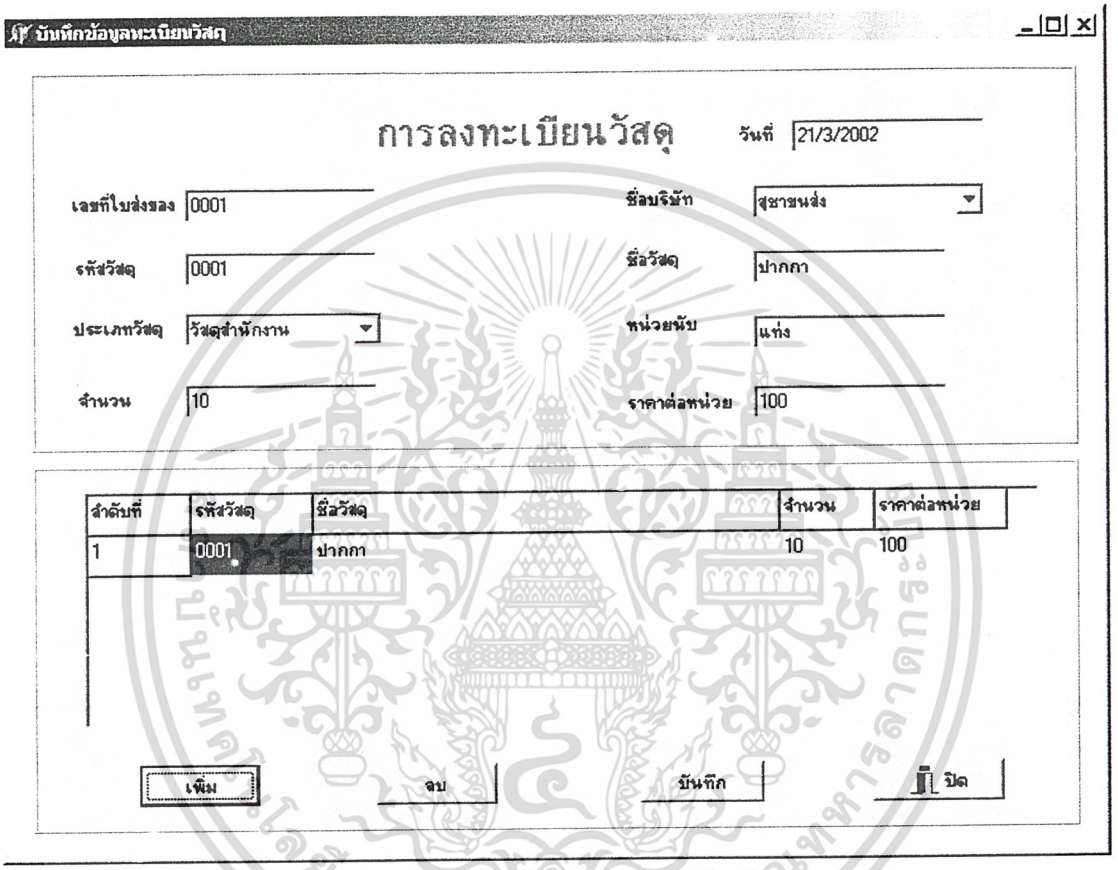
ภาคผนวก ก.
หน้าจอของแอปพลิเคชั่น

ได้ออกแบบส่วนการนำเข้าข้อมูล ทางจอภาพของแอปพลิเคชั่น และแสดงรายละเอียดและคำอธิบายรายละเอียดของหน้าจอต่างๆ ดังนี้

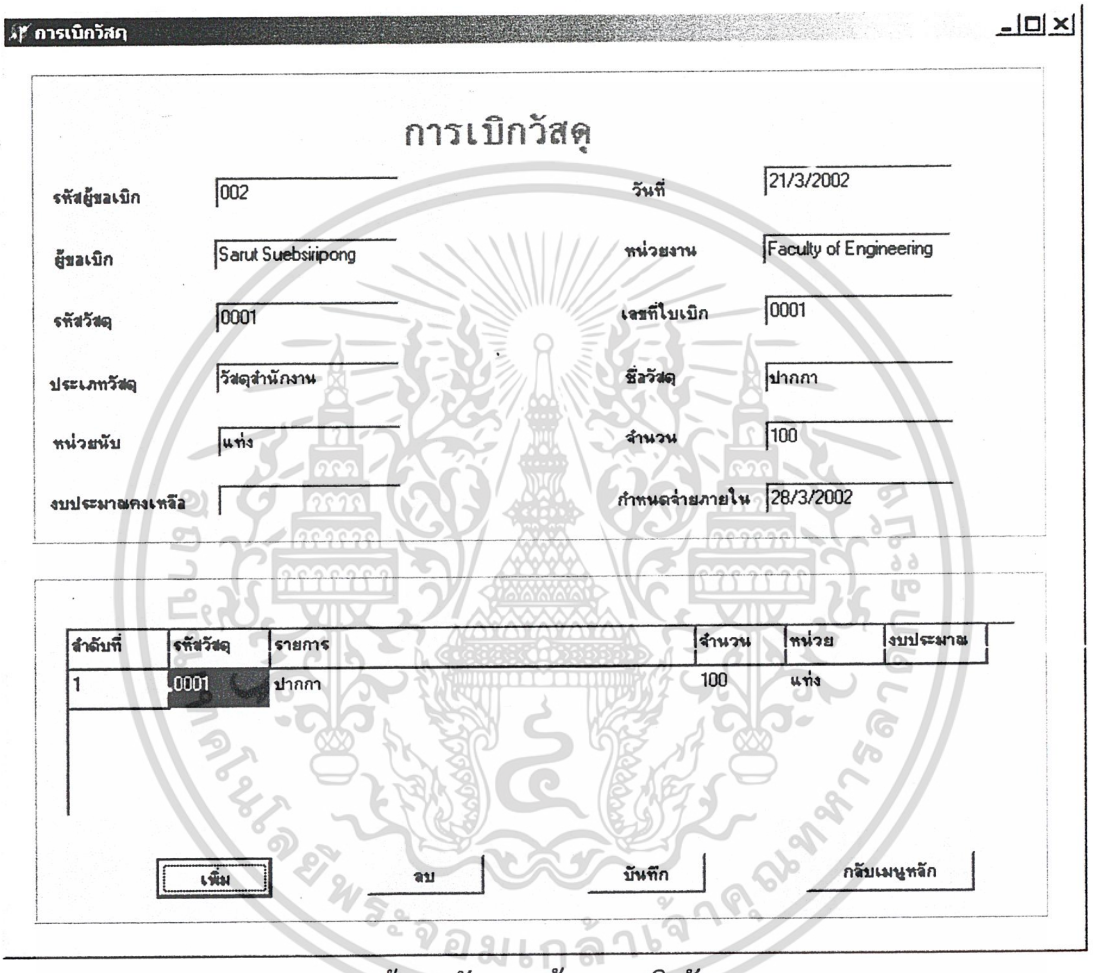
1. หน้าจอหลักของระบบงานวัสดุซึ่งจะแสดงรายละเอียดเกี่ยวกับงานด้านต่างๆของงานวัสดุ (รหัสหน้าจอ 01)ซึ่งมีงานดังนี้คือ
    - เมื่อเลือกเข้าสู่การลงทะเบียนวัสดุจะเข้าสู่หน้าจอรหัสที่ 02 เมื่อต้องการลงทะเบียนวัสดุที่เข้ามา
    - เมื่อเลือกเข้าสู่การเบิก-จ่าย วัสดุ จะเข้าสู่หน้าจอรหัสที่ 03 เมื่อต้องการทำการบันทึกข้อมูลการเบิก หรือ การจ่าย
    - เมื่อเลือกปิดจะออกจากระบบนี้
2. หน้าจอนี้เป็นการบันทึกรายละเอียดเกี่ยวกับการนำวัสดุเข้าคลัง(รหัสหน้าจอ 02) เมื่อมีการกรอกข้อมูลทั้งหมดจะนำข้อมูลไปปรับปรุงในฐานข้อมูลของวัสดุ และสามารถที่จะ เพิ่มหรือลบข้อมูลในนี้ได้ เมื่อเลือกปิดจะกลับเข้าสู่หน้าจอรหัสที่ 01
3. หน้าจอนี้เป็นการแสดงบันทึกการเบิก(รหัสหน้าจอ 03) เมื่อมีผู้มาขอเบิกรายการวัสดุที่มีอยู่ในคลัง ซึ่งจะเป็นการจองวัสดุที่มีอยู่ก่อน โดยที่ยังไม่มีการจ่าย ซึ่งจะสามารถแก้ไขปรับปรุงได้ ถ้ามีการบันทึกผิดพลาด
4. หน้าจอนี้จะเป็นการบันทึกการจ่ายวัสดุ และตัดวัสดุออกจากคลังวัสดุ เมื่อมีการจ่ายวัสดุ (รหัสหน้าจอ 04) เมื่อเลือกปิดก็จะกลับไปสู่หน้าจอรหัสที่ 01 เพื่อทำงานอย่างอื่นต่อไป



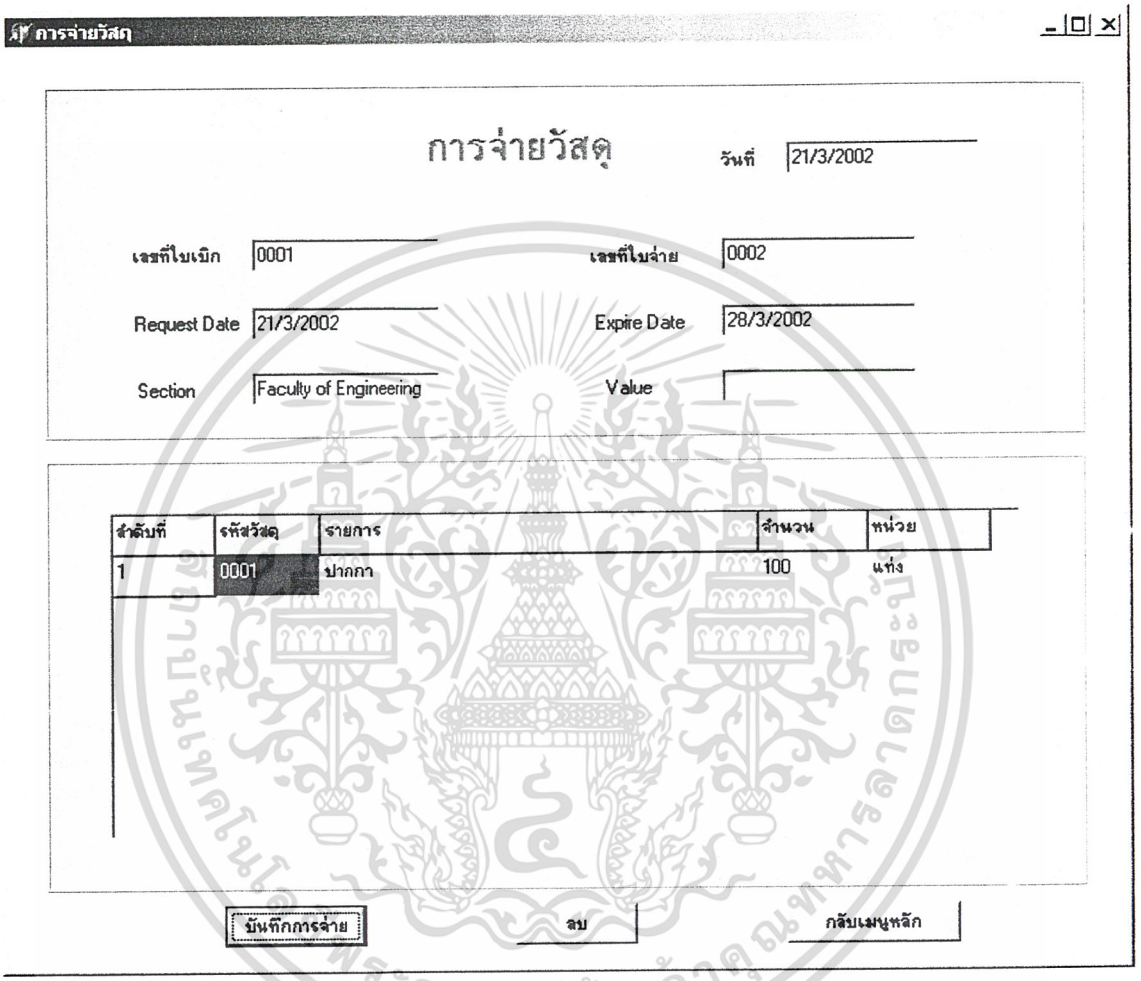
หน้าจอรหัส 01 หน้าจอหลักของระบบงานวัสดุ



หน้าจอรหัส 02 หน้าจอการนำวัสดุเข้าคลัง



หน้าจอรหัส 03 หน้าจอการเบิกวัสดุ



หน้าจอรหัส 04 หน้าจอการจ่ายวัสดุ

ภาคผนวก ข.
เอกสารวิชาการที่เกี่ยวข้องกับปริญญานิพนธ์
Literature Review

Proceedings of the 34th Hawaii International Conference on System Sciences - 2001

On Transformations from UML Models to Object-Relational Databases

Wai Yin Mok, David P. Paper

Department of Business Information Systems and Education

Utah State University

Logan, Utah, USA

ในการแปลงโมเดล UML ให้เป็นฐานข้อมูลเชิงวัตถุสัมพันธ์

มีเป้าหมายในการออกแบบฐานข้อมูลเชิงวัตถุสัมพันธ์โดยมีโมเดล UML เป็นอินพุท โดยกำจัดความซ้ำซ้อน และทำให้เกิดคุณสมบัติ termination, confluence, และ observable determinism ของทริกเกอร์ (trigger)

ขั้นตอนที่ 1 Class Diagrams to Nested Tables

คลาสไดอะแกรม (Class Diagram) ประกอบด้วยเซตของคลาส, อินเตอร์เฟส(Interface), Collaboration และความสัมพันธ์ระหว่างคลาสไดอะแกรม ความสัมพันธ์แบ่งเป็น dependencies, generalizations, realizations ซึ่งเป็นความสัมพันธ์แบบไบนารี่ และ association ซึ่งเป็นโครงสร้างความสัมพันธ์ระหว่างคลาสต่างๆ association นั้นถ้าไม่ระบุ จะไม่มีทิศทาง ในขณะที่ dependencies, generalizations, realizations จะมีทิศทางเสมอ ดังนั้นในการมองความสัมพันธ์ แบบ association จึงสามารถมองได้ทั้งสองทิศทาง ส่วนอีกสามแบบจะต้องมองตามทิศทางที่กำหนดเท่านั้น

ตั้งสมมติฐานดังนี้

1. แต่ละคลาสในคลาสไดอะแกรมมีบทบาทที่แตกต่างกัน นอกจากจะระบุไว้เป็นอย่างอื่นอย่างชัดเจน
2. ไม่มี Null เนื่องจาก UML ไม่อนุญาตให้มี Null
3. แต่ละคลาสและแต่ละความสัมพันธ์ในคลาสไดอะแกรมเป็น BCNF

อัลกอริทึม 0 ดังต่อไปนี้ใช้ในการกำจัดส่วนที่เป็น Semantical Overload ออกจากคลาสไดอะแกรม

Algorithm 0

Input: คลาสไดอะแกรมของ UML D ซึ่งมีคลาสและความสัมพันธ์เป็น BCNF

Output: คลาสไดอะแกรมที่สมมูลกับคลาสไดอะแกรมที่เป็นอินพุท

1. เปลี่ยนบทบาท (Roles) และชื่อต่างๆใน D ให้เป็นสับคลาส (Subclass)
2. สำหรับแต่ละไซเคิล(ไซเคิลคือเซตของความสัมพันธ์ที่เล็กที่สุดที่ทำให้ Graham Reduction ไม่สามารถกระทำได้) ถ้าลูกศรบอกทิศในไซเคิลกันไม่ให้สามารถไปต่อในไซเคิลได้ ไซเคิลนั้นจะไม่ก่อให้เกิดปัญหา ถ้าไม่เช่นนั้นจะแบ่งพิจารณาเป็นสองกรณี คือ ไซเคิลนั้นเกิดจากความสัมพันธ์มากกว่าหนึ่ง และไซเคิลนั้นเกิดจากความสัมพันธ์เพียงหนึ่งความสัมพันธ์

   ในกรณีแรกสมมติว่าไซเคิลประกอบด้วยความสัมพันธ์จำนวน m คือ $R_1,\ldots R_m$, $m \geq 2$ จากนั้นให้ r_j เป็นตัวอย่าง (Instance) ของ R_j สำหรับ $1 \leq j \leq m$ ถ้า $r_i = {}^{\pi} R_i (r_1 \bowtie \ldots \bowtie r_{i-1} \bowtie r_{i+1} \bowtie \ldots \bowtie r_m)$ เราสามารถนำ R_i ออกได้เนื่องจาก R_i สามารถหาได้จากความสัมพันธ์อื่น ถ้าไม่มีความสัมพันธ์ที่มีลักษณะดังกล่าว ให้เพิ่มหน้าที่ (Roles) ให้กับความสัมพันธ์ในไซเคิลจนทำ Graham Reduction ได้ จากนั้นกลับไปข้อที่ 1

   ในกรณีที่สอง ไซเคิลเป็นลูป (Loop) ภายในตัวเอง กล่าวคือความสัมพันธ์นั้นเชื่อมคลาสเข้ากับตัวเอง ถ้าอ็อบเจคในคลาสเกี่ยวข้องกับตัวมันเองเท่านั้นในความสัมพันธ์ ความสัมพันธ์นั้นจะเป็นความซ้ำซ้อนซึ่งสามารถเอาออกได้ ไม่เช่นนั้นจะหมายความว่าคลาสมีสองหน้าที่ในความสัมพันธ์ ให้เพิ่มหน้าที่ให้กับคลาสแล้วกลับไปข้อที่ 1

จากนั้นคลาสไดอะแกรมที่ได้จะนำไปเป็นอินพุทให้กับอัลกอริทึม 1 ซึ่งสามารถหาได้ใน D. Maier. *The Theory of Relational Databases*.Computer Science Press, Rockville, Maryland, 1983.

ขั้นตอนที่ 2 Statechart Diagrams to triggers

UML ได้ใช้ Statechart ในการทำโมเดลพฤติกรรม และในการจะเปลี่ยน Statechart ให้เป็นฐานข้อมูลจะใช้ triggers

ในการแม็ป Statechart ให้เป็น trigger นั้น trigger จะต้องมีคุณสมบัติบางข้อ คือ termination, confluence และ observation determinism

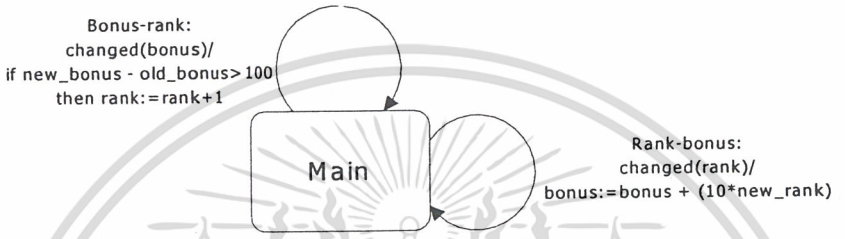
Problematic Triggers and Basic Concepts of Statecharts

เพื่ออธิบายปัญหาของ trigger จะยกตัวอย่าง 3 ข้อ

Example 1 (nonterminating) สมมติให้มีสอง trigger คือ Bonus-rank และ Rank-bonus Bonus-rank กำหนดว่าเมื่อฟิลด์ bonus เพิ่มขึ้น 100 ฟิลด์ rank จะเพิ่มขึ้น 1 Rank-bonus กำหนดว่าเมื่อ rank เพิ่มขึ้น bonus จะเพิ่มขึ้น 10 เท่าของค่า rank ใหม่

Example 2 (terminating, non-confluent) สมมติให้มีสาม trigger คือ Good-sales, Great-sales และ Rank-raise. Good sales กำหนดว่าเมื่อฟิลด์ sales มากกว่า 50 ฟิลด์ salary จะถูกเพิ่มขึ้น 10 Great-sales กำหนดว่าเมื่อ sales มากกว่า 100 rank จะเพิ่มขึ้น 1 Rank-raise กำหนดว่าเมื่อ rank เท่ากับ 15 salary จะเพิ่มขึ้น 10% ให้ Rank-raise มี priority สูงกว่า Good-sales และ Great-sales

Example 3 (terminating, confluent, not observably deterministic) สมมติให้มีสาม trigger คือ Good-sales, Rank-raise และ New-rank สอง trigger แรกเหมือนกับตัวอย่างข้างบน New-rank ใช้สำหรับแสดงฟิลด์ rank และ salary เมื่อ rank ถูกเปลี่ยนแปลง

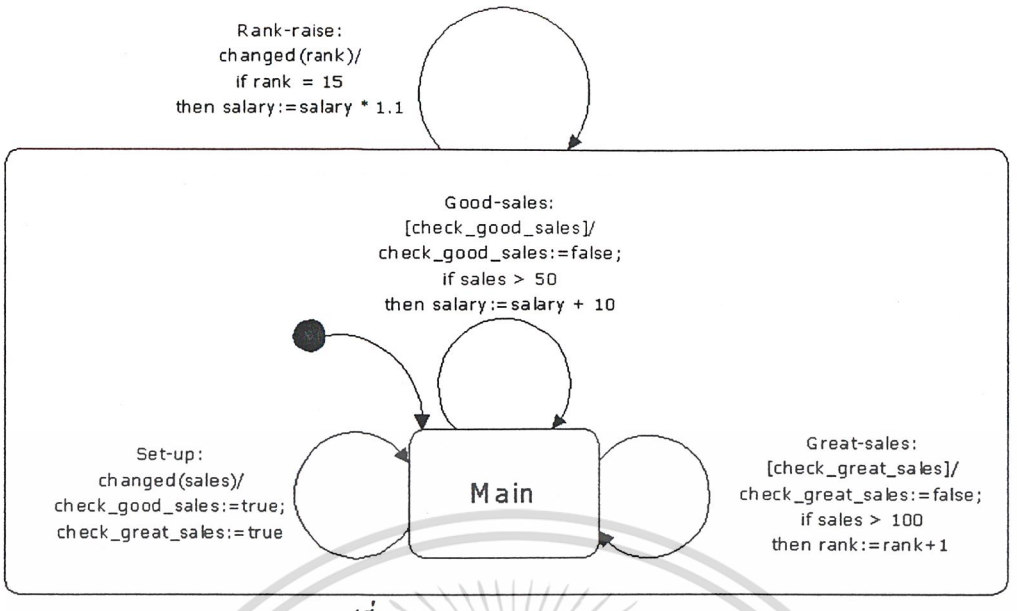


รูปที่ A1 Statechart ของ Example 1

จากรูปที่ 1 state ที่เป็นต้นทางกับปลายทางคือ state เดียวกัน event ที่ Bonus-rank สนใจคือ changed(bonus) ซึ่งจะถูกสร้างขึ้นเมื่อ bonus มีการเปลี่ยนแปลง Rank-bonus ก็ทำงานคล้ายกัน ถ้า trigger ทั้งสองทำให้ trigger อีกตัวหนึ่งทำงานสลับกันไปเรื่อยๆ และถ้า bonus เพิ่มขึ้นครั้งละมาก กว่า 100 และ rank มีค่า 10 หรือ rank ถูกเปลี่ยนแปลงให้มีค่ามากกว่า 10 แล้ว การทำงานนี้จะไม่มีที่ สิ้นสุด

รูปที่ 2 แสดง statechart ของตัวอย่างที่ 2 ซึ่งนอกจากจะมี 3 transition ตาม trigger 3 ตัวจากตัวอย่างแล้ว ยังมี transition ชื่อ Set-up ซึ่งมีเพื่อตั้ง flag ชื่อ check good sales และ check great sales ให้มีค่าความจริงเป็นจริงเมื่อเกิด event changed(sales) จากนั้นจะเลือกว่าจะทำ good-sales หรือ great-sales ซึ่งทั้งสองทางในขั้นตอนแรกจะมีการตั้งให้ flag ทั้งสองมีค่าเป็นเท็จเพื่อป้องกันไม่ให้ทำงานมากกว่าหนึ่งครั้ง ส่วน Rank-raise จะปรับ salary เมื่อเกิด event changed(rank)

เนื่องจาก Good-sales และ Great-sales ออกและเข้าสู่ Main และ Rank-raise ออกและเข้าสู่ parent state ของ Main เมื่อมีการออกจาก parent, state ที่อยู่ภายในหรือเป็นลูกก็จะถูกออกจากการทำงานไปด้วย เพื่อที่จะสร้างโมเดลให้ Rank-raise มี priority สูงกว่า Good-sales และ Great-sales จะต้องให้ Rank-raise ออกจาก parent state ของ Main เพราะการออกจาก parent จะมี priority สูงกว่าการออกจาก state ที่เป็นลูก

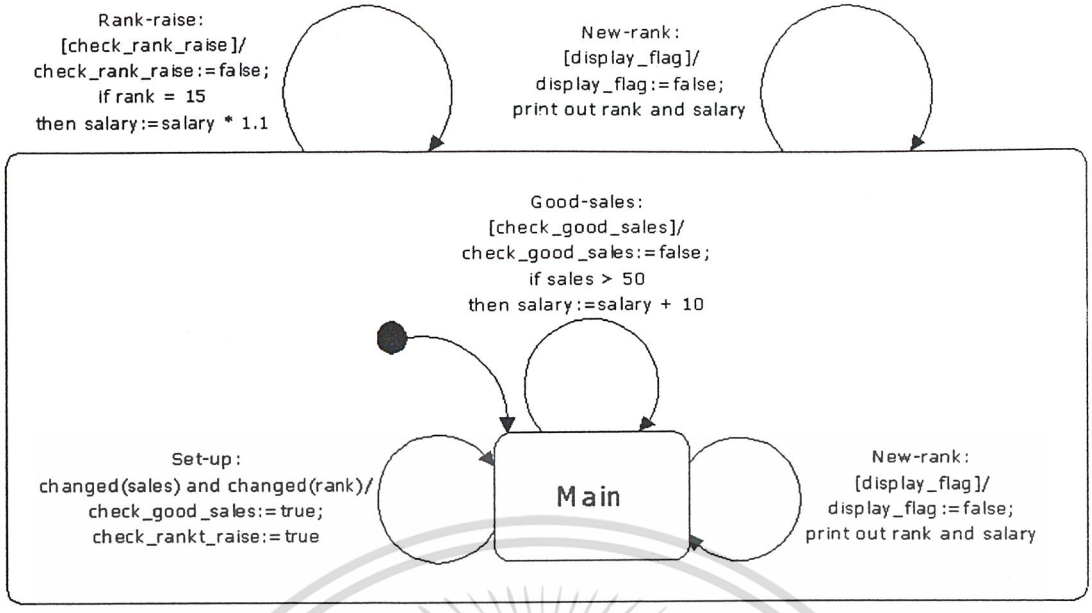


รูปที่ A2 Statechart ของ Example 2

ในการทำงาน event จะมีอยู่เพียงชั่วเวลาหนึ่ง ถ้าไม่มีการนำเอาไปใช้ในช่วงเวลาที่มันเกิดขึ้น event นั้นก็จะสูญไป ส่วนเงื่อนไขต่างๆนั้นจะมีอยู่ตลอดการทำงาน เนื่องจาก Good-sales และ Great-sales ขัดแย้งกันจึงจะไม่ทำงานพร้อมกัน และแทนที่จะให้ทั้งสอง transition ตรวจจับ event changed(sales) ได้มีการใช้เงื่อนไขมาป้องกัน ซึ่ง flag จะถูกตั้งให้เป็นจริงใน Set-up เท่านั้น และเมื่อ flag มีค่าเป็นจริง Good-sales และ Great-sales รับประกันว่าจะทำงานเป็นลำดับ ส่วน Rank-raise จะต่างกันเนื่องจากมี priority สูงกว่า รับประกันว่าจะทำงานเสมอ ดังนั้น event changed(rank) จะไม่สูญหาย

statechart ดังรูปที่ 2 ไม่ confluent เพราะถ้าทำงานทั้งสาม transition ในลำดับที่ต่างกัน อาจจะทำให้สถานะฐานข้อมูลไม่เหมือนกัน

รูปที่ 3 แสดง statechart ของ trigger ในตัวอย่างที่ 3 ซึ่ง Rank-raise ยังคงมี priority สูงกว่า Good-sales อย่างไรก็ตามในการทำงาน New-rank อาจจะมาก่อน Rank-raise หรือระหว่าง Rank-raise กับ Good-sales หรือหลังจาก Good-sales ดังนั้นผลลัพธ์อาจจะไม่เหมือนกันเนื่องจากลำดับการทำงานที่ต่างกัน ในการทำโมเดลของสถานะการณ์นี้จะใช้ flag ดังในรูปที่ 3 เนื่องจาก New-rank อาจทำงานก่อน Rank-raise จึงต้องสร้าง New-rank ขึ้นมาอีกอันหนึ่งบน parent state ของ Main หนึ่งในสอง New-rank แต่ไม่ทั้งคู่จะไม่ถูกทำงานเป็นระยะๆ และสมมติให้ flag display_flag ถูกตั้งให้เป็นจริงจากภายนอก และถูกตั้งให้เป็นเท็จภายใน transition นั้น



รูปที่ A3 Statechart ของ Example 3

Termination

ในรูปที่ 1 นั้นการทำงานอาจไม่มีที่สิ้นสุดและมีลักษณะเป็นไซเคิลซึ่งเป็น loop วนเข้าหาตัวเอง ในความเป็นจริง statechart ที่นำไปสู่การทำงานที่ไม่สิ้นสุดจะมีไซเคิลเสมอแม้ว่า statechart ที่มีไซเคิลไม่จำเป็นที่จะต้องมีการทำงานที่ไม่สิ้นสุด นอกจากนี้ในรูปที่ 1 transition ทำงานด้วย event ที่สร้างขึ้นภายใน ซึ่งหมายถึง event จะมีอยู่ตลอดเวลาหรือเงื่อนไขจะไม่มีทางเป็นเท็จ transition นี้จึงเป็น “self-feeding” แต่ในบางครั้ง event ที่สร้างขึ้นเองภายใน transition ก็อาจหมดลงได้ดังรูปที่ 4

รูปที่ A4 Statechart ที่ Transition สร้าง event ขึ้นเองและสามารถหมดไปได้

Algorithm 2

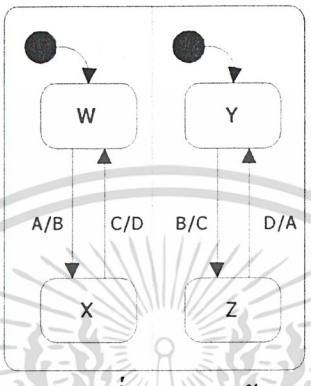
Input: Statechart

Output: Yes หรือ No

1. ถ้าไม่มีไซเคิลที่สร้างขึ้นจาก state และ transition ใน statechart, statechart จะ terminate คำตอบจะเป็น No ถ้าไม่เช่นนั้น ให้ Γ เป็นเซ็ตของไซเคิลนั้น สำหรับแต่ละสมาชิก γ ใน Γ ถ้า γ มี transition ที่ใช้ event ซึ่งสร้างขึ้นจากภายนอก γ จะไม่ทำให้เกิด nontermination สามารถนำ γ ออกจาก Γ ได้
2. ถ้าไม่มีไซเคิลที่เกิดขึ้นจาก event ที่สร้างขึ้นภายใน statechart, statechart จะ terminate และคำตอบเป็น No ถ้าไม่เช่นนั้น ให้ Σ เป็นเซ็ตของไซเคิลนั้น

3. ถ้ามีสมาชิก γ ใน Γ สมาชิก σ ใน Σ ซึ่ง event ใน σ ทำให้ระบบเปลี่ยนจาก state หนึ่งเข้าสู่ state ใน γ แล้ว statechart จะไม่มีทาง terminate และคำตอบเป็น Yes ถ้าไม่ เช่นนั้น statechart จะ terminate และคำตอบเป็น No.

ในขั้นตอนที่ 2 ของอัลกอริทึ่ม 2 ไซเคิลถูกสร้างขึ้นจาก event ที่อ่านหรือเขียนค่าจาก data item หรือ event ถูกสร้างขึ้นอย่างชัดเจนโดย transition ใน statechart เช่นรูปที่ 5



รูปที่ A5 Statechart ที่ event ถูกสร้างจาก transition

Γ คือ {[W, X], [Y, Z]} และ Σ คือ {[A, B, C, D]} ให้ γ คือ [W, X] และ σ คือ [A, B, C, D] event ที่ทำให้ statechart เปลี่ยนจาก state หนึ่งไปเป็นอีก state หนึ่งใน γ คือ [A, C] ซึ่งเป็นส่วนหนึ่งของ σ ดังนั้น statechart ดังรูปที่ 5 จะไม่ terminate

Confluence

Statechart ที่ nonconfluence เกิดขึ้นจากการทำงานแบบ nondeterminism และ transition ที่ขัดแย้งกันใน statechart หรือกล่าวได้อีกอย่างหนึ่งว่าเมื่อ statechart พบกับ nondeterminism (เมื่อมี transition ที่ขัดแย้งกันเป็นเส้นทางในการทำงานมากกว่าหนึ่ง) ฐานข้อมูลหลังจากทำงานเสร็จสิ้นอาจไม่เหมือนกันเนื่องจากลำดับการทำงาน transition ที่ขัดแย้งกันต่างกัน อย่างไรก็ตาม อาจมี transition ที่ขัดแย้งกันซึ่งนำไปสู่ฐานข้อมูลที่เหมือนกันแม้ว่าลำดับการทำงานต่างกัน เช่นรูปที่ 2 Good-sales และ Great-sales ขัดแย้งกันแต่ไม่ว่าลำดับการทำงานจะเป็นอย่างไรก็จะได้ผลลัพธ์สุดท้ายเหมือนกันเพราะไม่มีปัญหา read-write racing หรือ write-write racing ซึ่งมีความหมายดังนี้

Transition t_1 และ t_2 จะมีปัญหา read-write racing ถ้า t_1 อ่านค่าจาก data item x และ t_2 เขียนค่าลง x

Transition t_1 และ t_2 จะมีปัญหา write-write racing ถ้าทั้ง t_1 และ t_2 ต่างเขียนค่าลงใน data item x

Good-sales อ่านค่าจาก sales และเขียนค่าลง salary และ Great-sales อ่านค่าจาก sales และเขียนค่าลง rank จึงไม่เกิดทั้งปัญหา read-write racing และ write-write racing ดังนั้น statechart ที่มี เพียง Good-sales และ Great-sales จะ confluent แต่ถ้าเพิ่ม Rank-raise เข้าไปใน statechart, Rank-raise กับ Good-sales จะเกิดปัญหา write-write racing และ Rank-raise กับ Great-sales จะเกิดปัญหา read-write racing ในการ

ทำงานที่มีลำดับการทำงานต่างกันอาจจะให้ผลต่างกันได้ ดังนั้นเมื่อเพิ่ม Rank-raise เข้าไปแล้ว statechart นี้จะไม่ confluent (nonconfluent)

วิธีจะพิจารณาว่า statechart นั้นๆ confluent หรือไม่ทำได้โดยเมื่อ statechart พบ nondeterminism ซึ่งมี C เป็นเซ็ตของ transition ที่ขัดแย้งกัน ถ้าแต่ละ transition ใน C ไม่เกิดปัญหา read-write racing หรือ write-write racing กับ transition อื่นใน C แล้ว สามารถทำ transtion เหล่านั้นได้โดยไม่ต้องสนใจลำดับการทำงาน statechart จะยังคงทำให้ฐานข้อมูลออกมาเหมือนเดิมซึ่งก็คือ statechart นั้น confluent

เซ็ตของ transition ที่มากที่สุดในการจะทำให้ statechart ไม่ confluent คือ transition ภายในเซ็ตแต่ละ transition เกิดปัญหา read-write racing หรือ write-write racing กับอีก transition หนึ่งในเซ็ตนั้น

Observation Determinism

ปัญหาของ statechart ในรูปที่ 3 เกิดขึ้นจากการพิมพ์ค่าออกมาก่อนที่ statechart จะคงที่ (stable) เพื่อที่จะหลีกเลี่ยงปัญหานี้และเพื่อให้มีคุณสมบัติ Observation Determinism ควรกำหนดว่าการพิมพ์หรือแสดงค่าต้องทำหลังจาก statechart คงที่แล้วเท่านั้น

บรรณานุกรม

[1] สุนทริน วงศ์ศิริกุล : "พัฒนาโมเดลยุคใหม่ UML Unified Modeling Language มาตรฐานการสร้างโมเดลระบบงาน"

[2] อภิเนตร อูนากูล : "Object-Oriented Analysis And Design", 1

[3] "OMG Unified Modeling Language Specification", Version 1.3, June 1999

[4] Michael Stonebraker and Paul Brown with Dorothy Moore : "OBJECT-RELATIONAL DBMSs TRACKING THE NEXT GREAT WAVE", Morgan Kaufmann Publishers Inc., 2nd Edition

[5] Abraham Silberschatz, Henry F. Korth, S. Sudarshan : "Database System Concept", international edition, 3rd edition, McGraw-Hill, 1997