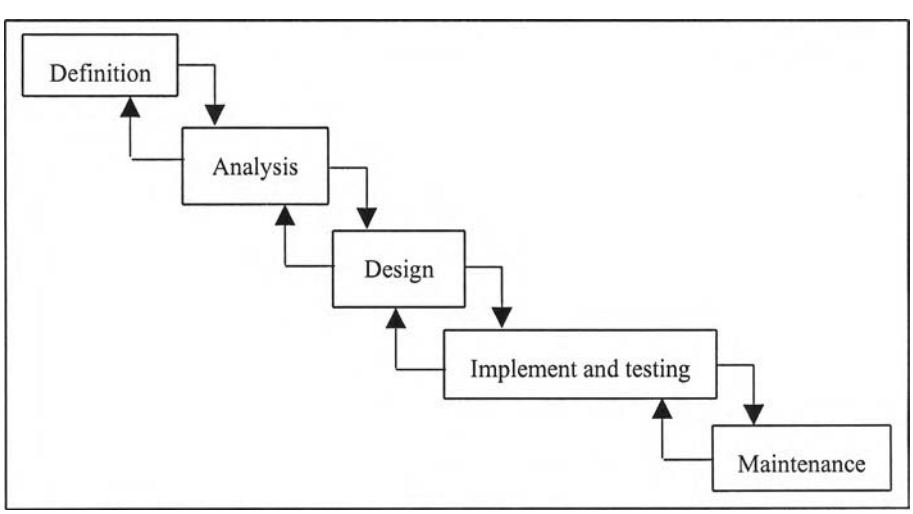
บทที่ 2

แนวความคิดและทฤษฎีที่เกี่ยวข้อง

2.1 การพัฒนาแบบเทคโนโลยีเชิงวัตถุ[2]

ในการพัฒนาซอฟต์แวร์เป็นเรื่องที่ค่อนข้างยาก ซับซ้อน และมีรายละเอียดค่อนข้างมาก จึงได้มีการแบ่งการพัฒนาซอฟท์แวร์ออกเป็น ขั้นตอน (Phase) ต่างๆ เรียกว่า Software Development Life Cycle Model ซึ่งการแบ่งการพัฒนาออกเป็นขั้นตอนนี้ ทำให้การปฏิบัติและติดตามผลของการพัฒนาได้ง่ายขึ้น จึงมีการกำหนดมาตรฐาน วงจรชีวิตของการพัฒนาซอฟต์แวร์ไว้หลายโมเดล เช่นวอเตอร์ฟอลโมเดล(Waterfall Model) แรพพิด โปรโตไทปิง โมเดล (Rapid Prototyping Model) สไปรัล โมเดล (Spiral Model) คอนเคอเรนท์เดวิลอปเมนท์ โมเดล (Concurrent development model) และ ฟอร์มอลเมทธอดโมเดล (Formal Method Model) เป็นต้น ในที่นี้ได้นำวอเตอร์ฟอลโมเดลมาเป็นหลักในการประยุกต์ใช้กับการวิเคราะห์และออกแบบเชิงวัตถุ เพราะเป็นที่นิยมและเข้าใจง่าย ซึ่งวอเตอร์ฟอลโมเดลจะมีการแบ่งการทำงานออกเป็น 5 ขั้นตอนดังรูป 2.1 โดยมีรายละเอียดคือ

1) ขั้นตอนของการศึกษาความต้องการของระบบ (Definition)
2) ขั้นตอนของการวิเคราะห์ (Analysis)
3) ขั้นตอนของการออกแบบ (Design)
4) ขั้นตอนของการปฏิบัติและทดสอบ (Implementation and Testing)
5) ขั้นตอนของการบำรุงรักษา(Maintenance)



รูปที่ 2.1 วอเตอร์ฟอลโมเดล

สัญลักษณ์ที่ใช้ช่วยในการวิเคราะห์และออกแบบระบบเชิงวัตถุที่ใช้ในการอธิบายนี้จะยึดตามหลักของยูเอ็มแอล(Unified modeling language) ซึ่งเป็นมาตรฐานในการพัฒนาระบบด้วยเทคโนโลยีเชิงวัตถุ (Object oriented technology) ซึ่งจัดทำโดยโอเอ็มจี (OMG ย่อมาจาก Object manage group)

2.1.1 ขั้นตอนของการศึกษาความต้องการของระบบ

เป็นการศึกษาและทำความเข้าใจกับระบบที่จะทำการพัฒนา โดยระบุขอบเขตของระบบเครื่องมือที่ใช้ในการแสดงความต้องการของระบบคือ ยูสเคสไดอะแกรม(Use case diagram) ซึ่งยูสเคสไดอะแกรมนั้นมีวัตถุประสงค์เพื่อใช้ในการอธิบายหน้าที่ของระบบให้ชัดเจนและให้เกิดความเข้าใจที่ตรงกันระหว่างลูกค้าหรือผู้ใช้ และผู้พัฒนาระบบ

2.1.2 ขั้นตอนของการวิเคราะห์

ในขั้นตอนของการวิเคราะห์ เป็นหลักสำคัญในการที่จะศึกษาและออกแบบถึงโครงสร้างของระบบงานตามขอบเขตของปัญหาที่สนใจตามยูสเคสไดอะแกรม โดยทำการศึกษายูสเคสไดอะแกรมเพื่อหาออบเจ็กต์ในระบบ จากนั้นทำการพิจารณารวบรวมออบเจ็กต์เพื่อกำหนดคลาสเบื้องต้นในการศึกษาและวิเคราะห์แบบจำลองปฏิสัมพันธ์ (Sequence diagram) เพื่อกำหนดรายละเอียดของคลาสเพิ่มเติม (Attribute[1] ,Operation[2]) และหาความสัมพันธ์ระหว่างคลาสจากขั้นตอนนี้และได้ผลลัพธ์เป็นคลาสไดอะแกรม(Class diagram)

2.1.3. ขั้นตอนการออกแบบระบบ

หลังจากได้วิเคราะห์ระบบแล้ว นำผลที่ได้จากการวิเคราะห์มาออกแบบ มาทำการออกแบบส่วนประกอบของโครงสร้างเชิงวัตถุและการติดต่อดังรูป 2.2 โดยมีขั้นตอน ดังนี้

2.1.3.1 การออกแบบในส่วนปัญหาหลักภายในขอบเขตความรู้ที่กำหนด (Problem-domain component) อาจมีการเพิ่มเติมในบางส่วนที่ระบบใหม่ต้องการใช้เท่านั้นและต้องไม่ทำให้ประสิทธิภาพของระบบลดลง พยายามลดความหนาแน่นของการปฏิบัติการในการโปรแกรม ส่วนที่อำนวยความสะดวกในการปฏิบัติการของโปรแกรมคอมพิวเตอร์อื่นๆ ควรเปลี่ยนแปลงอัลกอริธึมเพื่อเพิ่มประสิทธิภาพของการใช้งาน

2.1.3.2. การออกแบบในส่วนการติดต่อกับผู้ใช้งาน (Human interaction component) เป็นการออกแบบวัตถุที่จัดการเกี่ยวกับการติดต่อระหว่างวัตถุที่เกี่ยวข้องกับขอบเขตปัญหาที่กำหนดและผู้ใช้ ดังเช่น หน้าต่าง (window) และรายงาน (report)

---

[1] คุณลักษณะทั้งหมดที่มีภายในคลาส บอกถึงรายละเอียดของข้อมูล

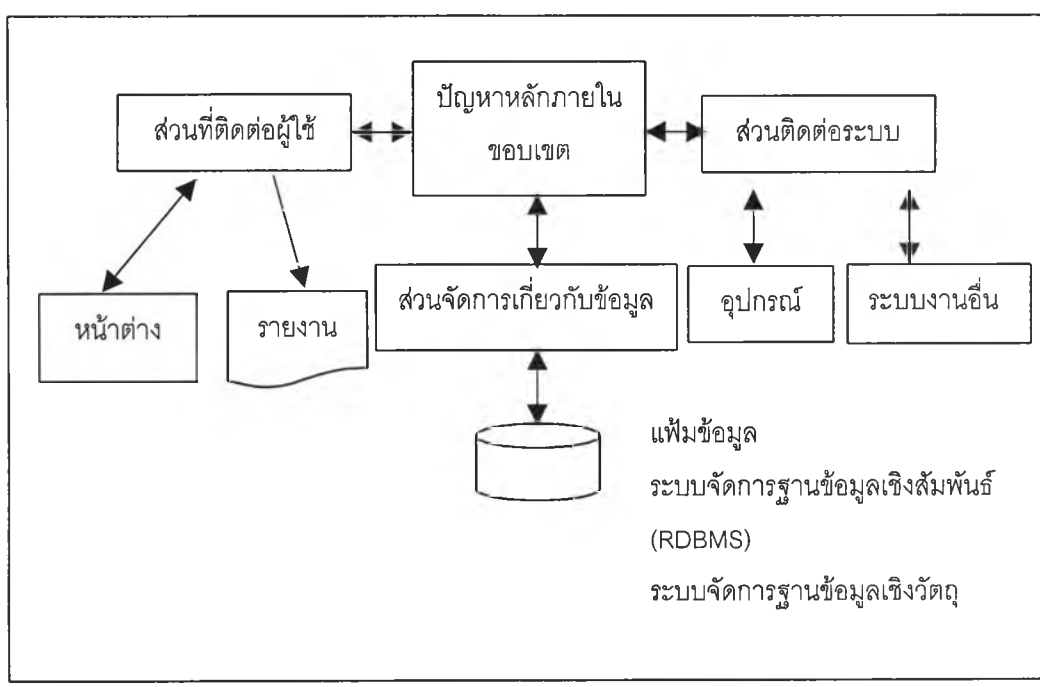
[2] เมทธอด(Method)ทั้งหมดที่มีภายในคลาส บอกถึงบริการของคลาส

2.1.3.3 การออกแบบในส่วนจัดการข้อมูล (Data-management component, DM) เป็นการออกแบบวัตถุที่จัดการการติดต่อระหว่างวัตถุที่เกี่ยวข้องกับขอบเขตของปัญหาและฐานข้อมูล หรือระบบจัดการแฟ้มข้อมูล (File-management system) ในส่วนจัดการข้อมูลของโมเดลเชิงวัตถุ มีไว้เพื่อจุดประสงค์ต่อไปนี้

1) เพื่อเก็บคลาสและวัตถุในส่วนขอบเขตปัญหาที่กำหนด ที่ซึ่งต้องการการคงอยู่ เป็นเสมือนการเตรียมตัวประสานไปยังหน่วยความจำที่จัดเก็บข้อมูลแบบฐานข้อมูลเชิงสัมพันธ์ ฐานข้อมูลเชิงวัตถุ หรืออื่นๆ การออกแบบในส่วนจัดการข้อมูลจะแยกส่วนของการจัดเก็บ การเรียก และการปรับปรุงข้อมูลออกจากส่วนของระบบที่มีอยู่ เพื่อเพิ่มความสามารถของการพกพาและการบำรุงรักษา

2) เพื่อเป็นการห่อหุ้มกลไกในการค้นหา และจัดเก็บวัตถุที่ต้องการการคงอยู่ทั้งหมดภายในขอบเขตปัญหาที่กำหนด

2.1.3.4 การออกแบบในส่วนติดต่อระบบ (System-Interaction component, SI) ประกอบด้วยวัตถุที่จัดการการติดต่อระหว่างวัตถุที่เกี่ยวข้องกับขอบเขตของปัญหาและระบบอื่นๆ หรืออุปกรณ์



รูปที่ 2.2 แผนภาพส่วนประกอบของโครงสร้างเชิงวัตถุและการติดต่อ

2.1.4 การพัฒนาและทดสอบ (Implementation and Testing)

2.1.4.1 การเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming) ใช้แนวคิดของชนิดข้อมูลนามธรรม (Abstract data type) ในการเขียนโปรแกรมเรียกชนิดข้อมูลนามธรรมนี้ว่าคลาส ประกอบด้วยสมาชิกที่เป็นข้อมูล (Data element) หรือแอททริบิวต์ (Attributes) และสมาชิกที่เป็นฟังก์ชัน (Member functions) หรือเมทธอด (Methods) วัตถุเป็นตัวตน (Instance) ของคลาสซึ่งมีลักษณะเช่นเดียวกับคลาส สมาชิกของคลาสชนิดข้อมูลจะเก็บสถานะต่างๆของวัตถุ และสมาชิกชนิดฟังก์ชันจะใช้ตอบสนองต่อวัตถุอื่นในระบบ วัตถุจึงประกอบด้วยสถานะของมันและบทบาทที่ตอบสนองต่อวัตถุอื่นๆ

การประกาศของคลาสมีการกำหนดส่วนที่เป็น Public Protected และ Private มีการปิดบัง (Encapsulate) ข้อมูลส่วนตัวของวัตถุ สามารถกระทำต่อวัตถุหรือเข้าถึงข้อมูลของวัตถุด้วยฟังก์ชันหรือเมทธอด โดยทั่วไปภาษาการเขียนโปรแกรมเชิงวัตถุมีการใช้เชิงวัตถุมีการใช้ลำดับชั้น (Hierarchy) ของชนิดวัตถุด้วยการถ่ายทอด (Inheritance) แอททริบิวต์และเมทธอด

ในปัจจุบันนิยมใช้โปรแกรมเชิงวัตถุเพิ่มมากสังเกตได้จากมีคอมไพเลอร์ (Compiler) ที่สนับสนุนการโปรแกรมเชิงวัตถุเช่น ไอเฟล (Eiffel) ซี พลัสพลัส (C++) ซิมูลาห์ (Simula) จาวา (Java) เดลไฟล์ (Delphi) เป็นต้น

การใช้หลักการของการโปรแกรมเชิงวัตถุ มีวัตถุประสงค์คือ

1) เพื่อให้เกิดการทำงานเป็นโมดูล (Modularization) คือสามารถแยกการทำงานเป็นหน่วยจำเพาะย่อยๆ ซึ่งแต่ละหน่วยสามารถทำงานได้โดยอิสระ แล้วจึงทำการรวมแต่ละหน่วยเข้าด้วยกันเป็นโปรแกรมใหญ่

2) ความเข้ากันได้ (Compatibility) เป็นประเด็นสำคัญ เพราะโดยทั่วไปแล้วเป็นการยากที่จะเข้าถึงข้อมูลที่มีโครงสร้างต่าง ๆ กันได้ การใช้โปรแกรมเชิงวัตถุจึงเป็นแนวทางในการแก้ปัญหา

3) การนำข้อมูลกลับมาใช้ใหม่ (Reuseability) สำหรับโปรแกรมขนาดใหญ่ ซึ่งมีโครงสร้างข้อมูลขนาดใหญ่ ทำให้เกิดการกระจัดกระจายของตัวแปร และข้อมูลต่างๆ การนำตัวแปรหรือข้อมูลกลับมาใช้ใหม่จึงเกิดความสับสน การโปรแกรมเชิงวัตถุสามารถที่จะแก้ปัญหานี้ได้โดยที่ผู้ใช้ไม่จำเป็นต้องรู้โครงสร้างทั้งหมดของโปรแกรม

4) ความต่อเนื่อง (Continunity) ซึ่งสามารถใช้คุณสมบัติการโปรแกรมเชิงวัตถุในแง่ของการสรุป (Abstraction) ได้

2.1.4.2 การทดสอบระบบสารสนเทศ (Information system testing) (Ronald J. Norman, 1996) รูปแบบของการทดสอบระบบสารสนเทศอาจมีข้อแตกต่างกัน ในสภาวะแวดล้อมที่แตกต่างกัน ได้แก่

1) การทดสอบโมดูล (Module testing) โมดูลเป็นส่วนโปรแกรมย่อย ๆ ที่มีหน้าที่การทำงานเฉพาะอย่าง ที่นักพัฒนาระบบต้องการทดสอบ

2) การทดสอบฟังก์ชัน (Function testing) ฟังก์ชันเป็นจำนวนโมดูลที่รวมกันเพื่อทำฟังก์ชันที่ผู้ใช้ระบุ (User-identified function) ฟังก์ชันที่ผู้ใช้ระบุได้แก่ฟังก์ชันที่อยู่ในเอกสารระบุความต้องการ (Requirements specification document) เช่น การพิมพ์รายงาน การแสดงใบกำกับสินค้า การเพิ่มรายการสินค้า เป็นต้น

3) การทดสอบระบบย่อย (Subsystem testing) ระบบย่อยเป็นการรวมฟังก์ชันหลายๆ ฟังก์ชันเข้าเป็นกลุ่ม การจำกัดความของระบบย่อยถูกระบุเมื่อระบบสารสนเทศกำลังถูกพัฒนา ในอีกความหมายหนึ่งคือเราไม่สามารถทำนายระบบย่อยโดยปราศจากการเริ่มต้นจากระบบและแบ่งเป็นระบบย่อยๆ

4) การทดสอบระบบรวม (System and integration testing) การทดสอบระบบรวมเป็นการทดสอบสำหรับระบบสารสนเทศทั้งหมดที่ถูกพัฒนา การพัฒนาโครงการ (Project) ใช้คำศัพท์เฉพาะ การทดสอบแอลฟา (Alpha testing) และการทดสอบเบต้า (Beta testing) ที่สถานะการทดสอบนี้ การทดสอบแอลฟาหมายถึงการทดสอบการยอมรับระบบทั้งระบบโดยหน่วยงานที่พัฒนาระบบซึ่งอาจมีผู้ใช้ร่วมทดสอบด้วยก็ได้ การทดสอบเบต้าหมายถึงการทดสอบโดยบุคคลหรือหน่วยงานภายนอกเช่น (1) ผู้ใช้ที่ไม่มีส่วนในการพัฒนาโครงการแต่เป็นผู้ใช้ที่จะใช้ระบบที่พัฒนาเสร็จสมบูรณ์แล้ว (2) หน่วยงานหรือลูกค้าภายนอกผู้ซึ่งจะใช้ประโยชน์ของระบบในที่สุด ยอมรับการทดสอบระบบ และให้ความคิดเห็นกลับมาตลอดการทดสอบเบต้า

5) การทดสอบการยอมรับของผู้ใช้ (User acception testing) การทดสอบระดับสุดท้ายก่อนการติดตั้งการใช้ระบบจริงคือ การทดสอบการยอมรับของผู้ใช้ ซึ่งเป็นโอกาสสุดท้ายสำหรับผู้ใช้ที่จะให้คำแนะนำสำหรับการเปลี่ยนแปลงก่อนการติดตั้งระบบสารสนเทศ ซึ่งจะสัมพันธ์กับเอกสารระบุความต้องการของผู้ใช้

2.1.5 **ขั้นตอนการบำรุงรักษา**

ภายหลังจากติดตั้งระบบแล้วจะต้องมีการบำรุงรักษาระบบ ซึ่งหมายถึงมีการประเมินผลการทำงานของระบบใหม่ รวมทั้งปรับปรุงเพิ่มเติมระบบงาน เพื่อให้สามารถตอบสนองความต้องการของผู้ใช้ที่มีการเปลี่ยนแปลงหรือเพิ่มเติมขึ้นจากเดิม

2.2 ภาษายูเอ็มแอล (UML ย่อมาจาก Unified modeling language)[3]

ยูเอ็มแอลเป็นภาษาหนึ่ง เป็นรูปแบบจำลอง และการผลิตสร้าง (Construct) ซอฟต์แวร์ขึ้นมาใช้เช่นเดียวกับแบบจำลองทางธุรกิจ (Business Modeling) โดยยูเอ็มแอลจะเป็นการเสนอรูปแบบการปฏิบัติในทางวิศวกรรมที่ดี ทั้งยังเหมาะกับระบบที่มีขนาดใหญ่และยุ่งยากซับซ้อนอีกด้วย

ยูเอ็มแอลประกอบด้วยไดอะแกรม(Diagram) 9 ไดอะแกรม

1) ยูสเคสไดอะแกรม (Use case diagram)
2) คลาสไดอะแกรม (Class diagram)
3) ออบเจ็กต์ไดอะแกรม (Object diagram)
4) ซีเควนซ์ไดอะแกรม (Sequence diagram)
5) สเตทไดอะแกรม (State diagram)
6) คอมโพเนนต์ไดอะแกรม (Component diagram)
7) ดีพอยเมนต์ไดอะแกรม (Deployment diagram)
8) คอลลาโบเรชั่นไดอะแกรม (Collaboration diagram)
9) แอคติวิตี้ไดอะแกรม (Activity diagram)

2.2.1 ยูสเคสไดอะแกรม

ยูสเคสไดอะแกรมเป็นแนวความคิดของไอวาร์ จาคอบสัน (Ivar Jacobson) ซึ่งโอเอ็มจีได้รวมไว้ในมาตราฐานของยูเอ็มแอลด้วย และได้มีวิธีการอื่นๆนำแนวคิดนี้ไปใช้อย่างกว้างขวาง ยูสเคสไดอะแกรมเป็นสิ่งที่ใช้ในการแสดงความต้องการของระบบทั้งหมดในลักษณะที่ผู้ใช้งานสามารถเข้าใจได้ง่าย โดยแสดงความสัมพันธ์ระหว่างผู้ใช้กับระบบต่างๆ สัญลักษณ์ที่ใช้แสดงดังรูป 2.3 มีรายละเอียดดังนี้

แอคเตอร์ คือสิ่งที่อยู่ภายนอกระบบที่จะพัฒนาขึ้นทั้งหมดที่ต้องการแลกเปลี่ยนหรือส่งข้อมูลให้กับระบบโดยที่ แอคเตอร์ อาจจะเป็นคน ระบบ หรือโปรแกรมอื่นๆ ก็ได้ โดยมากจะเป็นผู้เริ่มทำงานกับยูสเคส เช่นผู้ใช้ระบบ เป็นต้น

ยูสเคส จะเป็นตัวแทนงานที่เกิดขึ้นในขั้นตอนต่างๆ โดยจะใช้สัญลักษณ์เป็นรูปวงรี หรือวงกลม ถ้ายูสเคสใดมีกรอบสี่เหลี่ยมสีเทาล้อมรอบแสดงว่ามีไดอะแกรมย่อยที่ใช้อธิบายรายละเอียดของยูสเคสต่อไป

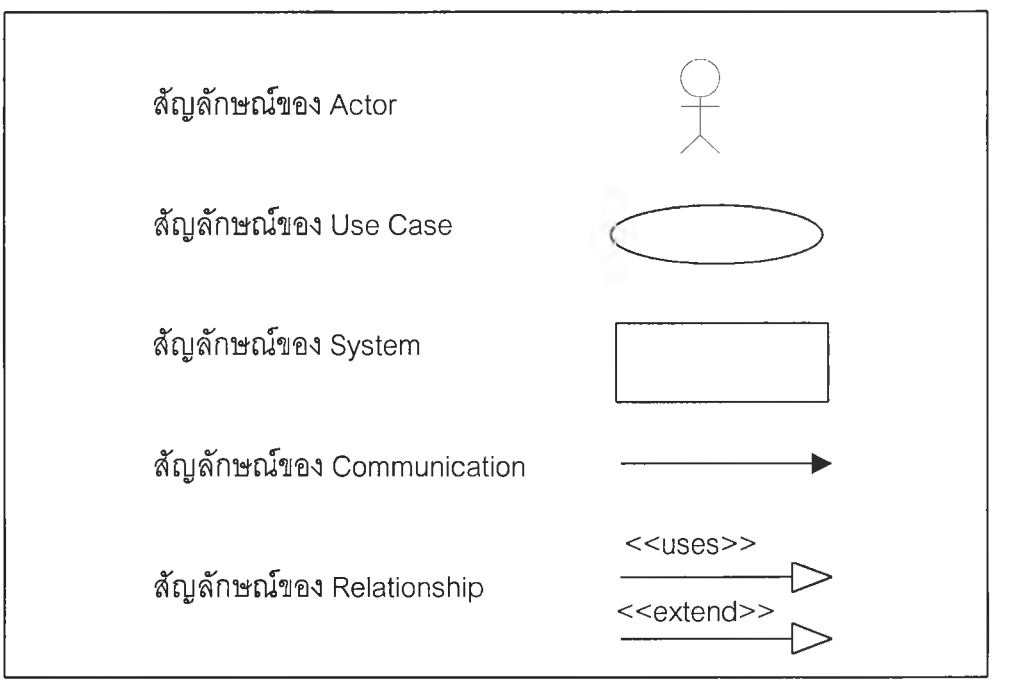
ระบบ เป็นตัวแทนของระบบ ซึ่งใช้สัญลักษณ์เป็นรูปสี่เหลี่ยมที่ถูกกระทำโดยแอคเตอร์ ภายในระบบประกอบไปด้วยยูสเคสต่าง ๆ ในที่นี้คือ ระบบสรรหาทรัพยากรบุคคล (Recruitment system)

การติดต่อกัน เป็นการแสดงความสัมพันธ์หรือการติดต่อสื่อสารกัน (การรับและให้ข้อมูลข่าวสารแก่กันและกัน) ระหว่างแอคเตอร์ และ ยูสเคส ซึ่งอาจจะเป็นการสื่อสารแบบทางเดียวหรือสองทางก็ได้ แสดงด้วยสัญลักษณ์เส้นที่มีหัวลูกศรสีดำ

ความสัมพันธ์ เป็นการแสดงความสัมพันธ์ระหว่าง ยูสเคสกับยูสเคสโดยใช้เส้นที่มีหัวลูกศรสีขาว ความสัมพันธ์ระหว่างยูสเคสมีได้ 2 แบบ คือ เอ็กเท็นด์ (Extends) และ ยูส(Uses)

เอ็กเท็นด์ เป็นการเพิ่มการทำงานให้กับยูสเคสโดยการเรียกใช้จากอีกยูสเคสหนึ่ง ลูกศรจะออกจากยูสเคสที่ถูกเรียกใช้งานไปยังยูสเคสที่ต้องการเพิ่มเติมการทำงาน และจะมีข้อความ <<extends>> ระบุอยู่ข้าง ๆ เส้น

ยูส เป็นการแสดงให้เห็นถึงการถ่ายทอด(Inherit) หน้าที่การทำงานจากยูสเคสหนึ่งไปอีกยูสเคสหนึ่งปลายลูกศรจะอยู่ที่ยูสเคสหลักที่จะถูกถ่ายทอดความสามารถออกไป โดยจะมีข้อความ <<uses>> ปรากฏอยู่ข้าง ๆ เส้น

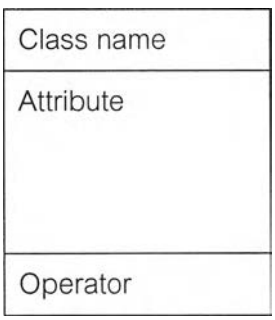


รูปที่ 2.3 แผนภาพสัญลักษณ์ของยูสเคสไดอะแกรม

2.2.2 คลาสไดอะแกรม

คลาสไดอะแกรมเป็นโมเดลชนิดหนึ่งที่เป็นโมเดลแบบสถิตย์ เป็นการอธิบายถึงคลาสและความสัมพันธ์ระหว่างคลาสที่มีโครงสร้างของข้อมูลรวมถึงพฤติกรรมของข้อมูลที่แตกต่างกัน ซึ่งคลาสหนึ่งคลาสนั้นสามารถกำหนดทิศทางของการนำไปสร้างโปรแกรม และการสร้างคลาสในออบเจ็กต์โอเรียนเท็ดได้

คลาสใช้สัญลักษณ์เป็นรูปสี่เหลี่ยม จะอธิบายถึงออบเจ็กต์ (Objects) ต่างๆว่ามีคุณลักษณะ (Attribute) เป็นเช่นไร หน้าที่การทำงาน (Operation) และความสัมพันธ์ (Relationship) ระหว่างคลาสแต่ละคลาส ซึ่งในแต่ละคลาสประกอบไปด้วยรายละเอียดพื้นฐาน 3 ส่วนดังรูป 2.4 มีรายละเอียดดังนี้



รูปที่ 2.4 แผนภาพส่วนประกอบพื้นฐานของคลาส

1) ชื่อคลาส แสดงชื่อของคลาสที่กำหนดในระบบ

2) คุณลักษณะ เป็นการกำหนดคุณลักษณะทั้งหมดที่มีภายในคลาส บอกถึงรายละเอียดของข้อมูล (Attribute) ภายในคลาสเดียวกันจะไม่ซ้ำกัน แต่ชื่อข้อมูลในคลาสอาจจะไปซ้ำกับชื่อข้อมูลในคลาสอื่นได้

3) หน้าที่การทำงาน เป็นส่วนที่ใช้อธิบายในคลาสนั้นมีเมทธอดอะไรบ้าง มีการรับค่าชุดของตัวแปร (Argument) อะไรบ้าง หรือมีการส่งค่าออกไปหรือไม่

2.2.2.1 สิทธิในการเข้าถึงข้อมูล

ในการเข้าถึงข้อมูลในคลาส จะมีการกำหนดสิทธิในการเข้าถึงข้อมูลโดยมีรายละเอียดดังนี้

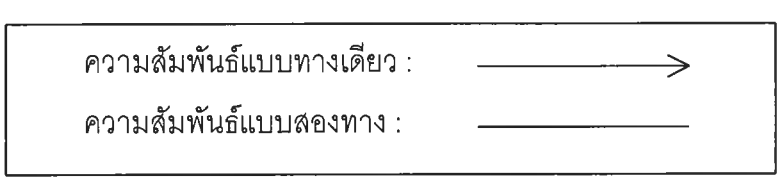
ตาราง 2.1 การแสดงสิทธิการเข้าถึงข้อมูล

สัญลักษณ์	ความหมาย	คำอธิบาย
-	Private	สามารถเข้าถึงได้เฉพาะตัวเอง, subclasses และ friends
+	Public	สามารถเข้าถึงได้ทุกคลาส
#	Protected	สามารถเข้าถึงได้เฉพาะตัวเองและ subclasses
?	Implementation	Scope ถูกตัดสินใจตอน Implement Program แต่ในตัวอย่างที่แสดงในเอกสารนี้หมายถึง package ในภาษา Java คือ สามารถเข้าถึงได้เฉพาะตัวเองและ classes ที่อยู่ใน package เดียวกัน

2.2.2.2 ความสัมพันธ์ระหว่างคลาส (Relationship)

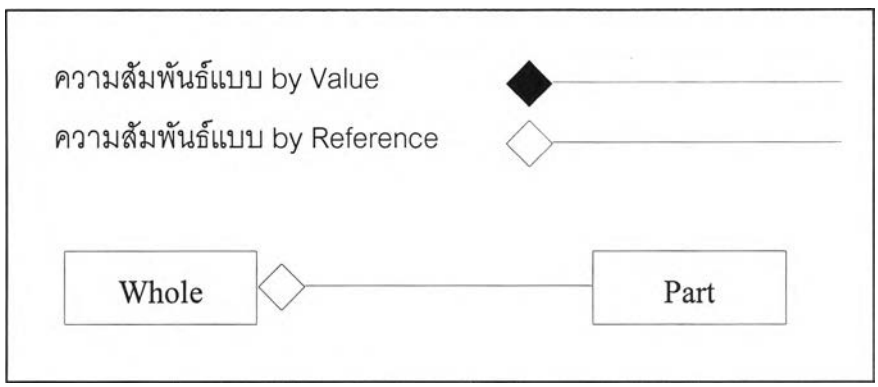
ยูเอ็มแอล ได้เตรียมความสัมพันธ์ระหว่างคลาสไว้ จะมีความสัมพันธ์อยู่ 4 รูปแบบซึ่งมีสัญลักษณ์ดังนี้

1) แอสโซซิเอชั่น (Association) สัญลักษณ์ดังรูป 2.5 จะแสดงความสัมพันธ์ระหว่างคลาส มีซึ่งมีได้ทั้งทางเดียว และ สองทาง



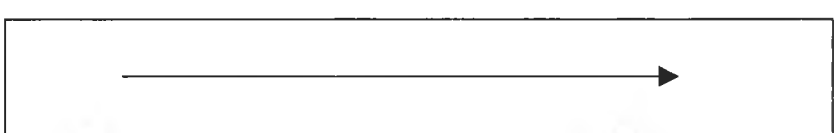
รูปที่ 2.5 แผนภาพความสัมพันธ์ระหว่างคลาสแบบแอสโซซิเอชั่น

2) แอกริเกชั่น (Aggregation) มีสัญลักษณ์ดังรูป 2.6 เป็นรูปแบบพิเศษของแอสโซซิเอชั่น คือเป็นความสัมพันธ์ระหว่างโฮล (Whole) และพาร์ท (Parts) ของมัน ซึ่งโฮลประกอบไปด้วย พาร์ทต่างๆของมัน ดังนั้นการคงอยู่ของพาร์ทจะต้องขึ้นกับโฮล หรือจะเรียกความสัมพันธ์แบบนี้ว่า “Whole-part”



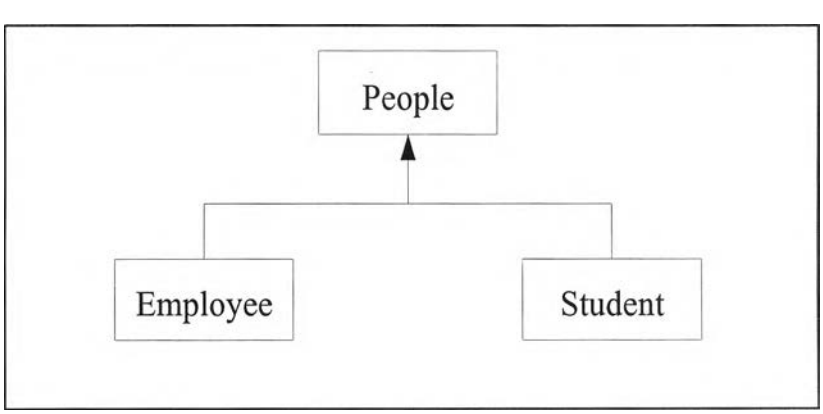
รูปที่ 2.6 แผนภาพความสัมพันธ์ระหว่างคลาสแบบแอกกริเกรชั่น

ความขึ้นต่อกัน (Depends on) มีสัญลักษณ์ดังรูป 2.7 เป็นรูปแบบความสัมพันธ์แบบหนึ่งที่ใช้แสดงความสัมพันธ์กันระหว่างคลาส 2 คลาส ในแง่ที่คลาสหนึ่งเรียกใช้บริการของอีกคลาสหนึ่งกล่าวคือคลาสของผู้ขอบริการขึ้นอยู่กับบริการของคลาสของผู้ให้บริการ แต่ไม่มีการขึ้นต่อกันภายในโครงสร้างของคลาส



รูปที่ 2.7 แผนภาพความสัมพันธ์ระหว่างคลาสแบบขึ้นต่อกัน

เจ็นเนอรัลไรเซชั่น (Generalization) มีสัญลักษณ์ดังรูป 2.8 ความสัมพันธ์รูปแบบนี้ใช้ แสดงความ สัมพันธ์ระหว่างคลาสกับคลาส ในแง่ที่คลาสหนึ่งถ่ายทอดคุณสมบัติและโครงสร้างจากอีกคลาสหนึ่ง โดยเรียกคลาสที่ถูกถ่ายทอดว่าคลาสบรรพบุรุษ(Super class) และเรียกคลาสที่ทำการถ่ายทอดว่าคลาสลูก(Sub class)



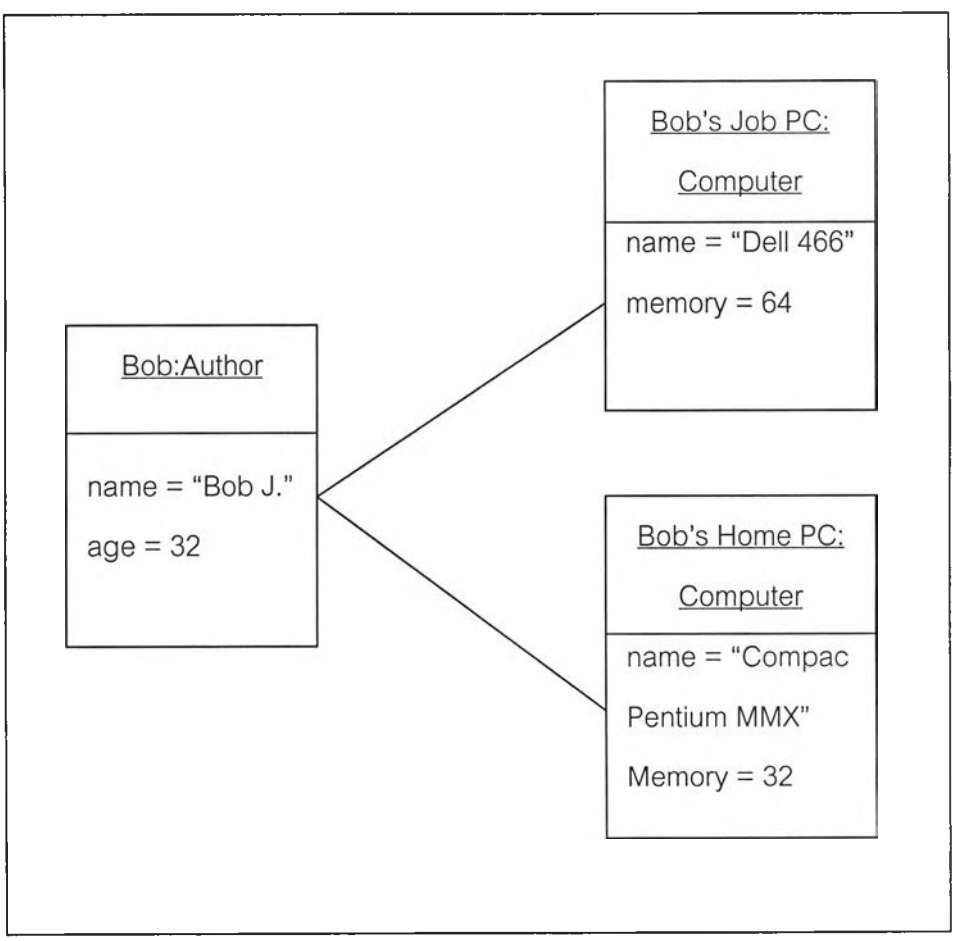
รูปที่ 2.8 แผนภาพความสัมพันธ์แบบเจ็นเนอรัลไรเซชั่น

2.2.3 ออบเจ็กต์ไดอะแกรม

ออบเจ็กต์ไดอะแกรมคือคลาสไดอะแกรมชนิดหนึ่งที่ใช้สัญลักษณ์เช่นเดียวกันกับคลาสไดอะแกรม ซึ่งทั้งสองไดอะแกรมมีความแตกต่างกันคือ ออบเจ็กต์ไดอะแกรมแสดงจำนวนของออบเจ็กต์ของคลาส ส่วนคลาสไดอะแกรมแสดงคลาสจริง ๆ หรืออาจกล่าวได้ว่าออบเจ็กต์ไดอะแกรมคือตัวอย่างของคลาสไดอะแกรมที่แสดงเหตุการณ์ต่าง ๆ ขณะระบบทำงาน ณ จุดเวลาหนึ่ง

สัญลักษณ์ที่ใช้จะแตกต่างจากคลาสไดอะแกรมเพียง 2 ส่วนดังรูป 2.9 คือชื่อของออบเจ็กต์จะต้องขีดเส้นใต้ และ ทุก ๆ ออบเจ็กต์ที่มีความสัมพันธ์กันจะต้องปรากฏในไดอะแกรม

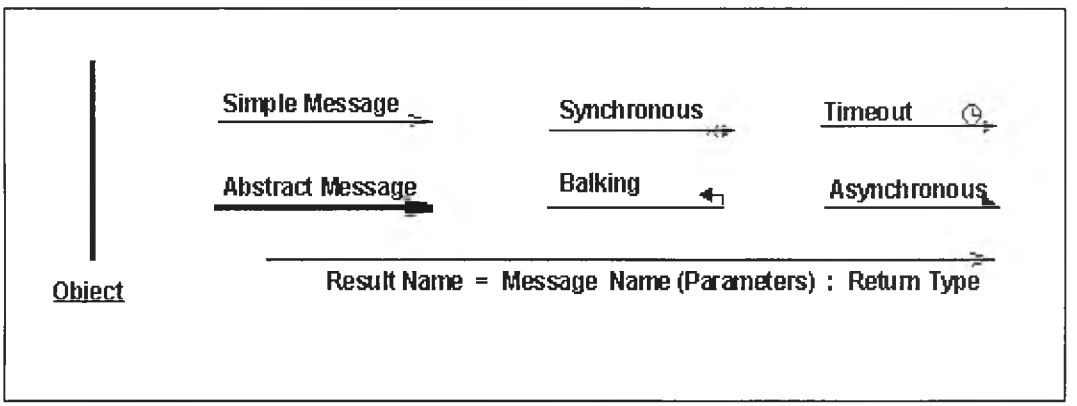
ออบเจ็กต์ไดอะแกรมสามารถใช้เป็นตัวอย่างคลาสไดอะแกรมที่ซับซ้อน โดยแสดงออบเจ็กต์ และความสัมพันธ์ระหว่างออบเจ็กต์ได้ นอกจากนี้ยังสามารถใช้แทนคอลลาโบเรชั่นไดอะแกรม กรณีที่ต้องการแสดงไดนามิคคอลลาโบเรชั่นไดอะแกรมระหว่างกลุ่มของออบเจ็กต์



รูปที่ 2.9 แผนภาพออบเจ็กต์ไดอะแกรม

2.2.4 ซีเควนซ์ไดอะแกรม

เป็นไดอะแกรมที่แสดงให้เห็นถึงการทำงานระหว่างออบเจ็กต์ต่างๆ ตามการส่งข้อความและเมื่อเหตุการณ์ต่างๆ ได้เกิดขึ้น มีสัญลักษณ์ดังรูป 2.10 ผู้เขียนโปรแกรมจะใช้ไดอะแกรมตัวนี้เป็นตัวช่วยเพื่อที่จะได้เขียนโปรแกรมให้ได้ตามที่ได้ออกแบบไว้



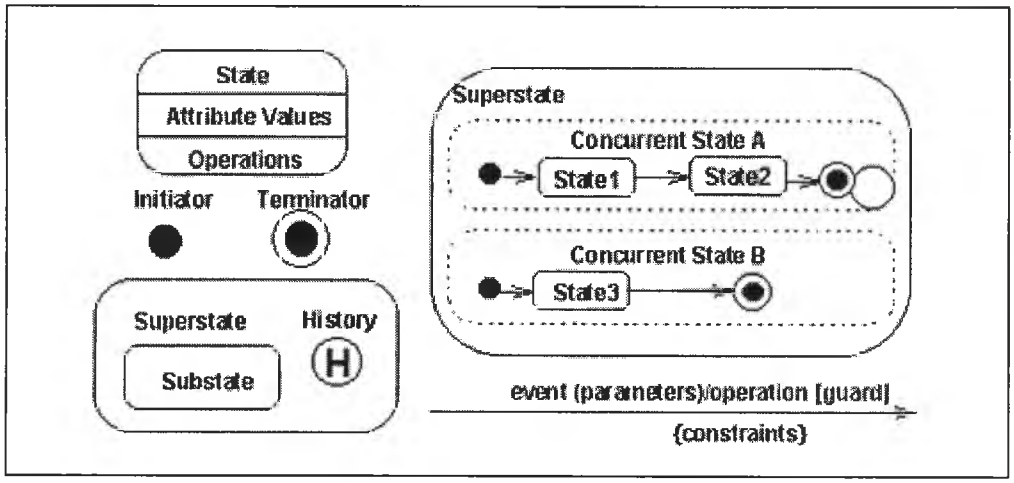
รูปที่ 2.10 แผนภาพสัญลักษณ์ที่ใช้ในซีเควนซ์ไดอะแกรม

ออบเจ็กต์ (Object) คือ อินสแตนท์ (Instance) ของคลาส ที่ทำหน้าที่รับ-ส่งข้อความ (Message) จากออบเจ็กต์อื่นๆ และทำการตอบสนองตามข้อความนั้นๆ เพื่อให้เกิดการทำงานในขั้นตอนต่างๆ ของระบบ โดยจะใช้สัญลักษณ์เป็นเส้นตรงแนวตั้งและมีชื่อของออบเจ็กต์และคลาสกำหนดอยู่ด้านล่าง

เมซเซจ (Message) เป็นข้อความที่ส่งไปมาระหว่างออบเจ็กต์

2.2.5 สเตทไดอะแกรม

เป็นวงจรชีวิตของออบเจ็กต์ ซับซีสเต็ม และระบบต่างๆ มีสัญลักษณ์ดังรูป 2.11 ซึ่ง สเตทจะเป็นตัวบ่งบอกถึงเหตุการณ์ต่างๆ ว่ามีผลกระทบอะไรเกิดขึ้นบ้าง สเตทไดอะแกรมจะทำการเชื่อมต่อกับทุกคลาส ที่มีพฤติกรรมที่ซับซ้อนให้ชัดเจนขึ้น และเป็นการอธิบายถึงพฤติกรรมของระบบ ซึ่งในแต่ละสเตทจะมีความแตกต่างกันขึ้นอยู่กับสถานะในปัจจุบัน รวมทั้งมีเหตุการณ์ใดบ้างที่มีผลกับการเปลี่ยนแปลงทั้งภายในและภายนอก โดยอาจมีจุดเริ่มต้นและจุดจบได้ในหลายๆ จุด

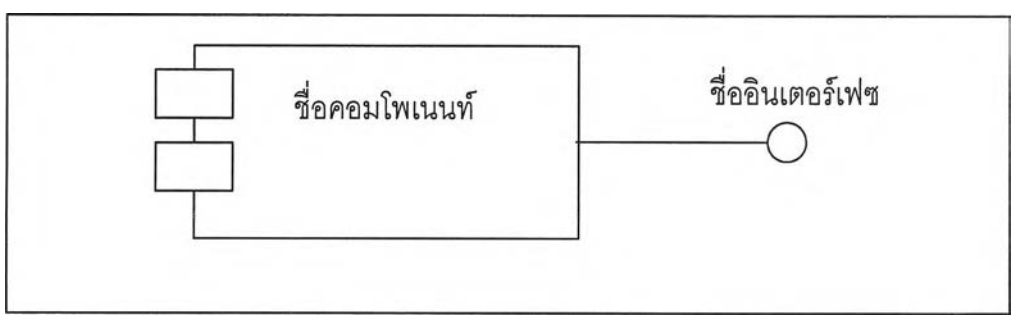


รูปที่ 2.11 แผนภาพสัญลักษณ์ที่ใช้ในสแตทไดอะแกรม

2.2.6 คอมโพเนนต์ไดอะแกรม

คอมโพเนนต์ไดอะแกรม แสดงถึง โครงสร้างและความสัมพันธ์กัน ระหว่างองค์ประกอบต่างๆ ของซอฟต์แวร์ (Software) มีสัญลักษณ์ดังรูป 2.12 ซึ่งองค์ประกอบดังกล่าวอาจเป็นโปรแกรมต้นฉบับ (Source Program), ส่วนของโปรแกรมที่ใช้เมื่อโปรแกรมทำงานเช่น ไลบรารี (Library) ต่างๆ หรืออาจเป็นโปรแกรมสำหรับทำงาน (Executable program) ก็ได้ ในการประยุกต์ใช้กับธุรกิจนั้นอาจผนวกเอาเอกสารและกระบวนการทำงานของธุรกิจเข้าเป็นส่วนหนึ่งของซอฟแวร์ได้

คอมโพเนนต์ไดอะแกรม คือ กราฟที่แสดงองค์ประกอบต่างๆ ของระบบที่เชื่อมโยงกันโดยใช้ความสัมพันธ์ในลักษณะการขึ้นต่อกัน โดยจะแสดงในรูปของเส้นประที่มีหัวลูกศรชี้ จากคอมโพเนนท์ลูกไปยังคอมโพเนนท์หลัก

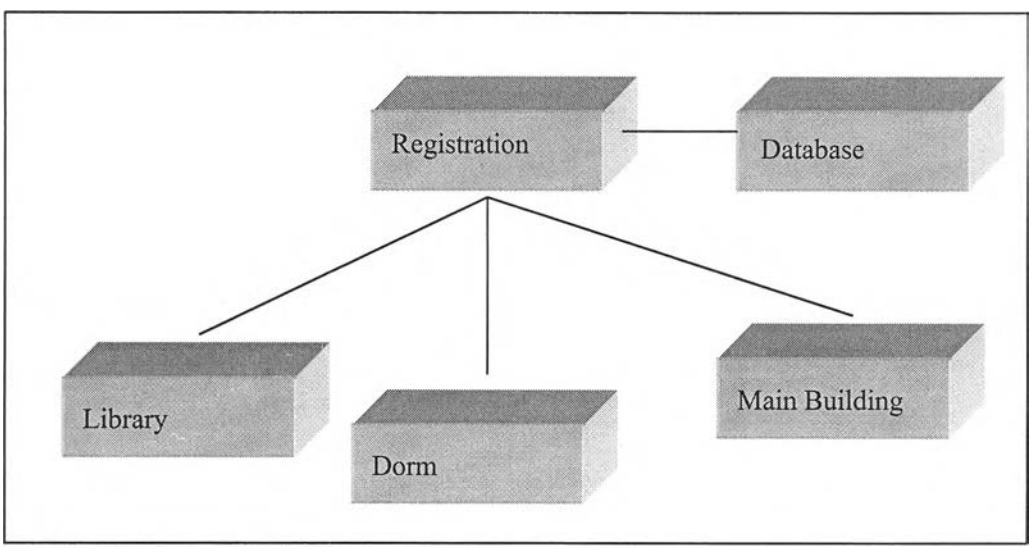


รูปที่ 2.12 แผนภาพสัญลักษณ์แทนคอมโพเนนท์และอินเตอร์เฟซ

2.2.7 ดีพอยเมนต์ไดอะแกรม

ดีพอยเมนท์ไดอะแกรมแสดงถึงส่วนประกอบต่างๆ และลักษณะของระบบในขณะทำงาน และองค์ประกอบซอฟต์แวร์ต่าง ๆ โพรเซส (Process) และวัตถุต่างๆ ที่ใช้ในการทำงานของระบบ ตัวอย่างดีพอยเมนท์ไดอะแกรมจะแสดงดังรูป 2.13 ซึ่งในไดอะแกรมนี้แสดงให้เห็นถึงการจัดวางคอมโพเนนท์ต่างๆ ของระบบสำหรับการทำงานได้อย่างชัดเจน สำหรับองค์ประกอบซอฟต์แวร์ที่แสดงในไดอะแกรมจะแสดงเฉพาะส่วนที่ปรากฏในขณะที่ระบบทำงานเท่านั้น สำหรับองค์ประกอบซอฟต์แวร์ที่ไม่ได้มีอยู่ในขณะที่ระบบมีการใช้งานจะแสดงในคอมโพเนนท์ไดอะแกรม

ดีพอยเมนท์ไดอะแกรมคือกราฟที่ประกอบด้วยโหนด(Node) ต่างๆ ที่เชื่อมโยงกันโดยการติดต่อสื่อสารระหว่างโหนดโดยที่โหนดแสดงถึงหน่วยประมวลผลซึ่งโดยทั่วไปจะประกอบด้วยหน่วยความจำและความสามารถในการประมวลผล อันได้แก่อุปกรณ์คำนวณต่างๆ หรือมนุษย์ รวมถึงอุปกรณ์ต่างๆ ที่ใช้ในการประมวลผล



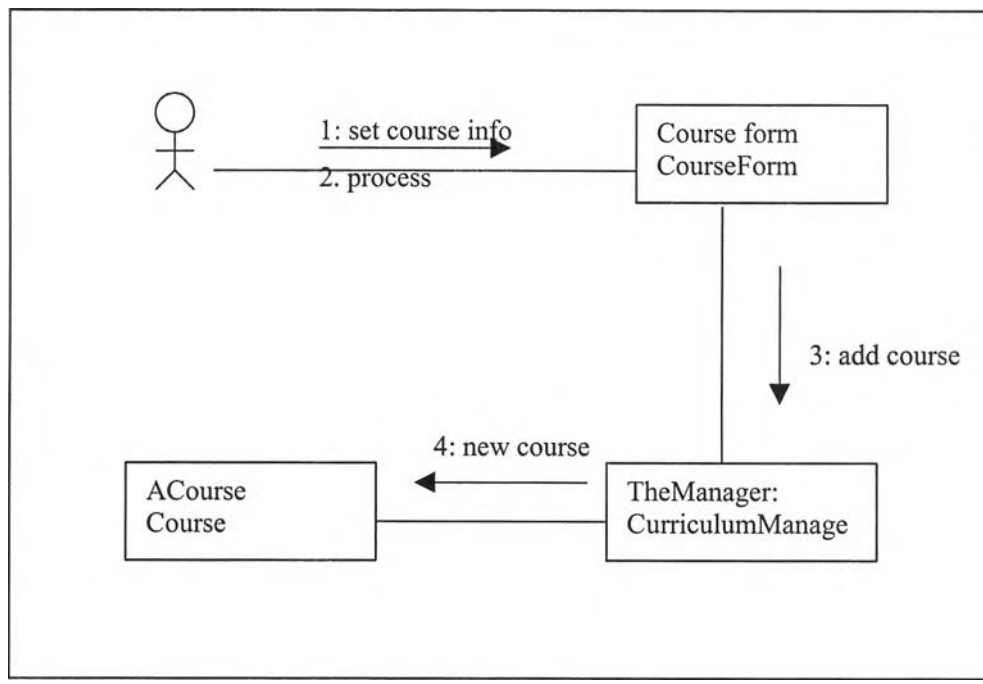
รูปที่ 2.13 แผนภาพตัวอย่างการใช้ดีพอยเมนท์ไดอะแกรม

2.2.8 คอลลาโบเรชั่นไดอะแกรม

คอลลาโบเรชั่นไดอะแกรม (Collaboration Diagram) นั้นใช้แสดงการติดต่อสื่อสารระหว่างออบเจ็กต์ต่าง ๆ ที่เกี่ยวข้องกับแต่ละออบเจ็กต์ที่สนใจรวมทั้งความสัมพันธ์ระหว่างแต่ละออบเจ็กต์ ในการติดต่อสื่อสารด้วย ตัวอย่างคอลลาโบเรชั่นไดอะแกรมแสดงดังรูป 2.14 ซึ่งคอลลาโบเรชั่นไดอะแกรมจะมีลักษณะที่ใกล้เคียงกับ ซีเควนซ์ไดอะแกรมมาก ซึ่งในบางครั้งในการทำงานอาจจะเลือกทำไดอะแกรมใดไดอะแกรมหนึ่งก็ได้ ซึ่งมีหลักเกณฑ์ในการพิจารณาดังนี้

1) ถ้าเป็นการกำหนดช่วงของเวลาที่แน่นอนและใช้เวลาเป็นสิ่งสำคัญที่สุด แล้วควรเลือกใช้ ซีเควนซ์ไดอะแกรม

2) หากให้ความสำคัญกับเนื้อหา (Context) แล้วควรเลือกใช้ คอลลาโบเรชั่นไดอะแกรม

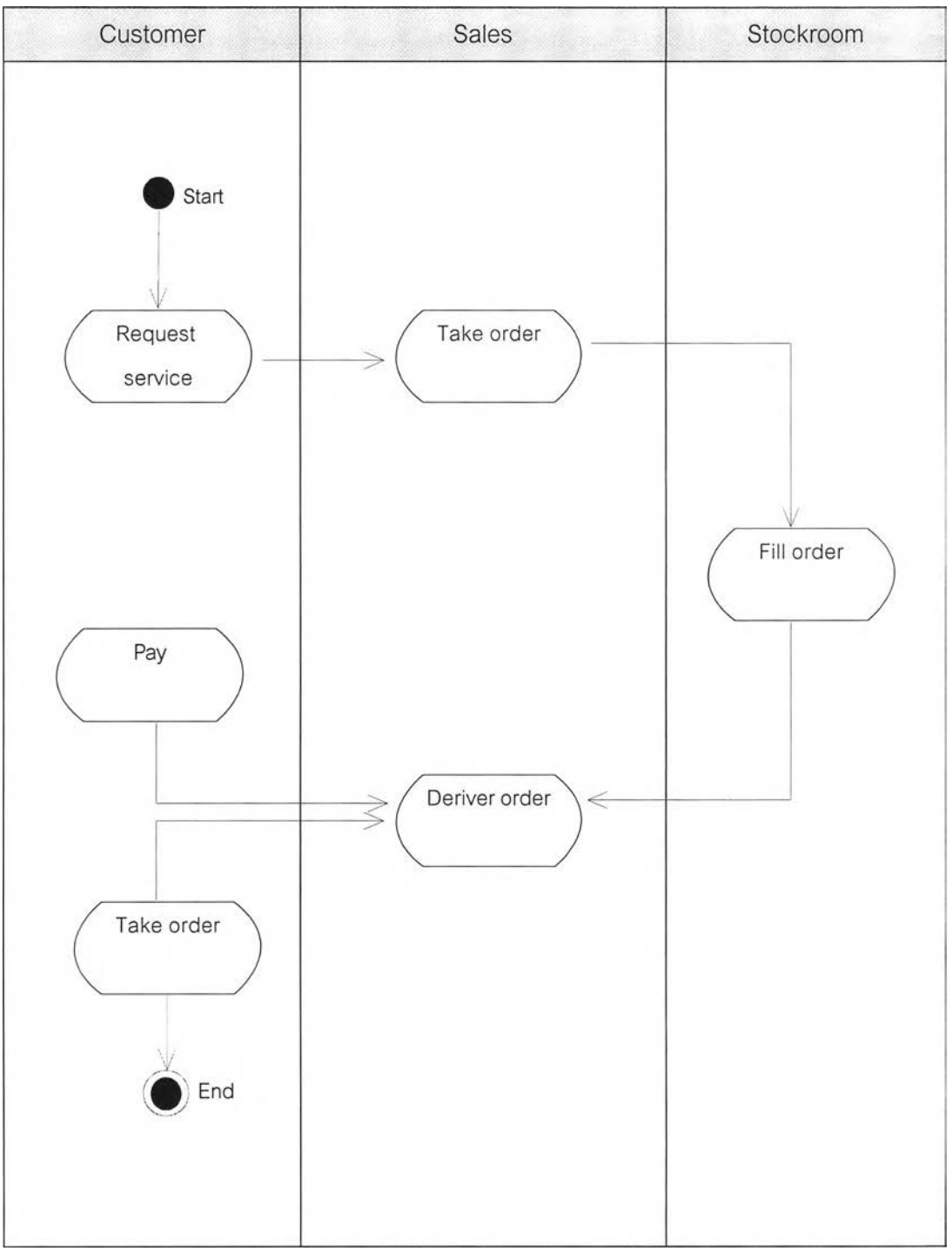


รูปที่ 2.14 แผนภาพตัวอย่างการใช้ คอลลาโบเรชั่นไดอะแกรม

2.2.9 แอคติวิตี้ไดอะแกรม

แอคติวิตี้ไดอะแกรมจะแสดงถึงลำดับขั้นตอนการทำงานของกิจกรรมต่างๆ ในระบบ ซึ่งสามารถใช้เป็นแบบในการปฏิบัติงานในหน้าที่ต่างๆ รวมทั้งยังอธิบายถึงขั้นตอนการทำงานในขั้นต่อๆ ไปของกิจกรรมอื่นด้วย แอคติวิตี้ไดอะแกรมเป็นการพิจารณาถึงสถานะของการทำงานในแต่ละกิจกรรม (Action) ซึ่งสถานะการทำงานนั้นเมื่อเริ่มต้นทำก็ต้องทำให้สำเร็จ ก่อนที่จะเริ่มทำงานในสถานะต่อไปได้ ตัวอย่างแอคติวิตี้ไดอะแกรมดังรูป 2.15

กิจกรรมที่แสดงในไดอะแกรมอาจมีการจัดให้อยู่ในลักษณะของลู่ (Swimlanes) เพื่อแสดงขอบเขตความรับผิดชอบของกิจกรรมต่าง ๆ โดยแต่ละคลาสที่เกี่ยวข้องจะมีการติดต่อส่งผ่าน (Transition) เพื่อเชื่อมโยงแต่ละกิจกรรมเหล่านั้นเข้าด้วยกัน



รูปที่ 2.15 แผนภาพตัวอย่างแอคติวิตี้ไดอะแกรม

2.3 ฐานข้อมูลวัตถุเชิงสัมพันธ์(Object Relational Database)[4]

ฐานข้อมูลวัตถุเชิงสัมพันธ์ เป็นการนำความสามารถของฐานข้อมูลออบเจ็กต์ และฐานข้อมูลเชิงสัมพันธ์ไว้ด้วยกัน และได้มีการกำหนดเข้าไว้ในมาตราฐานเอสคิวแอล 3 (SQL3 Standard) ซึ่งยังคงความสามารถของฐานข้อมูลเชิงสัมพันธ์อยู่ เนื่องจากข้อมูลยังคงถูกเก็บอยู่ในตารางของแถว (row) และสดมภ์ (column) การใช้ภาษาในการกำหนดข้อมูล ประมวลข้อมูล และสอบถามข้อมูลยังคงใช้ภาษาเอสคิวแอล (SQL) และผลลัพธ์ที่ได้ก็ยังคงเป็นตารางของข้อมูล

ในส่วนที่เพิ่มเติมจากฐานข้อมูลเชิงสัมพันธ์คือ มีการเพิ่มความสามารถในการจัดเก็บข้อมูล โดยมีโครงสร้างข้อมูลที่ซับซ้อนมากขึ้นเรียกว่า แอบสแทรค ดาต้า ไทป์(Abstract data type : ADTs) เป็นโครงสร้างข้อมูลที่รวบรวมข้อมูลพื้นฐานเข้าด้วยกันเพื่อใช้แทนข้อมูลหนึ่งออบเจ็กต์และรองรับหลักการของเอ็นแคปซูเลชั่น(Encapsulation) ซึ่งฐานข้อมูลเชิงสัมพันธ์ไม่สามารถทำได้ นอกจากนี้ ยังสนับสนุนคุณสมบัติสำคัญ 2 ประการคือ

1) การสร้างข้อมูลแอกริเกชั่น (Aggregration) คือ การกำหนดชนิดข้อมูลให้สามารถบรรจุข้อมูลชนิดอื่น ๆ ไว้ในชนิดข้อมูลหนึ่งได้ เช่น ข้อมูลรถยนต์ประกอบด้วย ประตู เครื่องยนต์และล้อ

2) การสร้างชนิดข้อมูล ซึ่งสามารถบรรจุข้อมูล หรือกลุ่มข้อมูลที่เป็นแอบสแทรค ดาต้า ไทป์ได้

แอบสแทรค ดาต้า ไทป์แบ่งออกเป็น 2 ชนิดคือ

1) แอบสแทรค ดาต้า ไทป์ที่มีการจัดเก็บแบบถาวร คือมีการจัดเก็บลงสื่อข้อมูลซึ่งจะมี OIDs(Object Identifiers) เป็นตัวชี้ (pointer) ในการเข้าถึงข้อมูล

2) แอบสแทรค ดาต้า ไทป์ที่ไม่มีการจัดเก็บแบบถาวรจะไม่มี OIDs (Object Identifiers) เป็นตัวชี้ ในการเข้าถึงข้อมูล เนื่องจากข้อมูลชนิดนี้จะมีการจัดเก็บไว้ในหน่วยความจำเพราะเป็นข้อมูลชั่วคราว

2.3.1 แอบสแทรค ดาต้า ไทป์ในระบบการจัดการฐานข้อมูลออราเคิล (Oracle8) มีดังต่อไปนี้

1) ออบเจ็กต์ เทเบิล (Object table) เป็นตารางพิเศษที่บรรจุออบเจ็กต์ไว้ และสามารถแสดงคุณสมบัติของออบเจ็กต์นั้นได้ ดังเช่นตัวอย่างการสร้างออบเจ็กต์เทเบิลชื่อ address ที่เก็บข้อมูลชนิดออบเจ็กต์ที่ชื่อ Address_T ดังรูป 2.16 ซึ่งในออบเจ็กต์แต่ละตัวจะมี OID เพื่อเป็นตัวชี้ไปยังข้อมูลออบเจ็กต์ที่เก็บอยู่

```
CREATE TYPE Address_T as OBJECT (
        Address1    varchar2(30),
        Add5ress2    varchar2(30),
        Address3    varchar2(30),
        City        varchar2(20),
        State       varchar2(2),
        Zip         number(5));
CREATE TABLE address OF Address_t;
```

รูปที่ 2.16 แผนภาพตัวอย่างการสร้างออบเจ็กต์เทเบิล

2) แวริเอเบิลอะเรย์ (Variable array) คือ ชนิดข้อมูลที่เป็นอะเรย์หนึ่งมิติที่มีขนาดคงที่ในการจัดเก็บข้อมูล และสามารถจัดอยู่เป็นสดมภ์หนึ่งของข้อมูลชนิดอื่นได้ดังตัวอย่างเช่นการสร้างแวริเอเบิลอะเรย์ชื่อ empSalary ดังรูป 2.17 ที่ใช้จัดเก็บข้อมูลชื่อพนักงานและเงินเดือนพนักงานตลอด หนึ่ง ปี

```
CREATE TYPE month as VARRAY (12) of number(5)
CREATE TABLE empSalary(
        Name varchar2(20),
        MonthSalary month);
```

รูปที่ 2.17 แผนภาพตัวอย่างการสร้างแวริเอเบิลอะเรย์

3) เนสติ้งเทเบิล (Nesting tables) คือ ชนิดข้อมูลที่เป็นตาราง ซึ่งมีอย่างน้อยหนึ่งสดมภ์ หรือมากกว่ามีชนิดของข้อมูลเป็นตาราง หรือแอบสแทรค ดาต้า ไทป์ ตัวอย่างเช่น การสร้างเนสติ้งเทเบิลชื่อ Emp_List ดังรูป 2.18 ซึ่งเก็บข้อมูลการทำงานของพนักงานแต่ละคน โดยมี Task_list ซึ่งเป็นสดมภ์ที่มีชนิดข้อมูลเป็นตารางใช้เก็บรายละเอียดงานที่ทำ

```
CREATE TYPE Task_T(
                Task_name       varchar2(20),
                Duration        number);
CREATE TYPE Emp_List_T(
                Emp_id          number,
                Last_name       varchar2(20),
                First_name      varchar2(20),
                Task_list       Task_T);
CREATE TABLE Emp_List of Emp_List_T
nested table Task_T store as Task_List_tb;
```

รูปที่ 2.18 แผนภาพตัวอย่างการสร้างเนสติ้งเทเบิล

4) เรฟ (REF) คือชนิดข้อมูลที่เก็บตัวชี้ ไปยังออบเจ็กต์เทเบิลอื่น ๆ โดยตัวชี้นี้จะชี้ไปยัง OID ซึ่งจะเป็นตำแหน่งที่อยู่ของออบเจ็กต์นั้น โดย OID ของออราเคิลจะใช้เนื้อที่ 128 ไบต์ ซึ่งออบเจ็กต์แต่ละตัวจะมี OID ที่ไม่ซ้ำกัน ดังตัวอย่างเช่น สร้างตารางชื่อ Employeeดังรูป 2.19 ซึ่งมีข้อมูล Dept มีชนิดข้อมูลเป็นเรฟ ซึ่งเก็บตัวชี้ที่ชี้ไปยัง OID ของข้อมูลในตาราง Department

```
CREATE TABLE Employee(
                Emp_id          number,
                Last_name       varchar2(20),
                First_name      varchar2(20),
                Dept            ref Depart_t);
INSERT INTO Employee
SELECT "05270", "RENON","JOHN",ref(Dx ) from Department Dx
WHERE Dx.DeptId = "01"
```

รูปที่ 2.19 แผนภาพตัวอย่างการสร้างเรฟ