แนะนำ UML เบื้องต้น

ดร.บรรจง หะรังษี และ นางญาณวรรณ สินธุภิญโญ
นักวิจัย งานโครงการเครือข่าย NECTECNet
ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ

ABSTRACT -- UML is a new wave in the software industry that has continued to grow up day by day and year by year. Currently it is widely said among system analysts, programmers, software engineers and software-related people, of applying the UML methodology to office and organization applications. For those who are interested in the software engineering process, this tutorial article is aimed at introducing the basics of UML: What is UML? and What are the fundamental working mechanisms of UML? In addition it is highly hoped in the end that this article will do a certain degree, assist those people in making a decision whether to bring UML to use with your office or not.

บทคัดย่อ-- UML เป็นคลื่นลูกใหม่มาแรงในแวดวงอุตสาหกรรมซอฟต์แวร์ ซึ่งมีการเติบโตขึ้นเรื่อยๆ อย่างดีวันดีคืน ปัจจุบันในแวดวงนักวิเคราะห์ระบบ โปรแกรมเมอร์ หรือผู้ที่ทำงานเกี่ยวกับซอฟต์แวร์เป็นที่กล่าวกันอย่างแพร่หลายถึงการนำเอา UML มาประยุกต์ใช้กับระบบงานต่างๆ ในสำนักงานหรือองค์กรของตน บทความ Tutorial ฉบับนี้มีจุดมุ่งหมายที่จะแนะนำโดยเบื้องต้นว่า UML คืออะไรและมีกลไกการทำงานโดยพื้นฐานอย่างไร เพื่อให้บุคคลที่เกี่ยวข้องโดยเฉพาะทางด้านซอฟต์แวร์ได้ทราบและเรียนรู้ถึงภาพโดยรวมของ UML นอกจากนั้นแล้วผู้เขียนยังหวังว่าบทความนี้อาจจะมีส่วนช่วยในการตัดสินใจถึงการที่จะนำเอา UML มาใช้กับหน่วยงานของท่านหรือไม่

คำสำคัญ -- Software Methodology, Modeling Language, Object-Oriented Analysis and Design, UML, Object-Oriented Programming, Software Engineering, Software Development, System Analysis and Design

ผู้อ่านหลายท่านคงได้คุ้นหูถึงคำว่า UML หรือ Unified Modeling Language และคงไม่แน่ใจว่า UML คืออะไร จะนำไปใช้งานได้อย่างไร บทความนี้ผู้เขียนจึงอยากจะแนะนำให้ผู้อ่านรู้จักและเข้าใจถึง UML ในแง่มุมต่างๆ ดังนี้

- นิยามของ UML และทำไมเราต้องใช้ UML
- กลไกการทำงานเบื้องต้นของ UML ว่ามีองค์ประกอบอะไรบ้าง
- ภาพโดยรวมของกระบวนการทำงานทั้งหมดของ UML

1. UML คืออะไร

การดำเนินงานโครงการพัฒนาซอฟต์แวร์ประกอบด้วย การเก็บรวมรวมข้อมูลเกี่ยวกับความต้องการของผู้ใช้ (Requirement Collection) ในการใช้ระบบ การวิเคราะห์ข้อมูลเหล่านั้น (Analysis) การออกแบบ (Design) และการเขียน

โปรแกรมหรือการสร้างซอฟต์แวร์ (Implementation) เป็นงานศิลปะประเภทหนึ่ง ที่มีความละเอียดอ่อน เพราะต้องการความพิถีพิถันในทุกขั้นตอนนับตั้งแต่การวิเคราะห์ข้อมูลที่ได้รับจนกระทั่งการพัฒนาขึ้นมาเป็นซอฟต์แวร์ (ระบบ คือ ซอฟต์แวร์ที่เราจะทำการพัฒนาขึ้นมาใช้)

จุดนี้จึงเกิดเป็นคำถามว่าเราจะมีวิธีการหรือเครื่องมือที่ดีอันหนึ่งหรือไม่ ที่จะช่วยให้การดำเนินงานโครงการทำซอฟต์แวร์โครงการหนึ่งเป็นไปอย่างมีประสิทธิภาพ ประสิทธิผล และมีการประสานกลมกลืนนับตั้งแต่ต้นจนจบ คำตอบก็คือ "มี" และ UML ก็คือเครื่องมือซึ่งสามารถ *"ช่วย"* เราได้นับตั้งแต่การวิเคราะห์ การออกแบบ และการดำเนินการพัฒนา

ตัว UML จริงๆ แล้วไม่สามารถดำเนินการสร้างซอฟต์แวร์ได้ กล่าวคือ ไม่สามารถทำการสร้างโปรแกรมได้ (Code Generation) แต่ทว่าผลพวงภายหลังจากการออกแบบมีรูปแบบหรือหน้าตาที่โปรแกรมเมอร์สามารถที่จะดำเนินการพัฒนาโปรแกรม (Coding) ได้อย่างเร็วและง่ายดายมาก

UML มององค์ประกอบต่างๆ ของซอฟต์แวร์ที่จะทำการพัฒนาขึ้นมาในรูปของออบเจ็คต์ (Object) [1] และออบเจ็คต์แต่ละตัวนั้นมีความเกี่ยวข้องกันโดยอาศัยความสัมพันธ์(Relationships) เป็นตัวเชื่อมโยง อีกทั้งออบเจ็คต์เหล่านั้นสามารถติดต่อสื่อสารกันได้ การติดต่อสื่อสารกันระหว่างออบเจ็คต์นี้เองเป็นกลไกภายในซอฟต์แวร์ที่ทำให้ซอฟต์แวร์ทำงานตามที่ผู้ใช้ต้องการได้

จากการมองซอฟต์แวร์เป็นออบเจ็คต์ นี่เอง UML จึงช่วยให้การเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming) เป็นไปได้ง่าย เพราะการเขียนโปรแกรมเชิงวัตถุก็มององค์ประกอบของซอฟต์แวร์เป็นออบเจ็คต์เช่นเดียวกัน

1.1 ประวัติโดยย่อของ UML

อาจจะมีคำถามหนึ่งที่ว่า UML เป็นวิธีการเดียวหรือไม่ที่สามารถนำมาประยุกต์ใช้ในการดำเนินงานโครงการทำซอฟต์แวร์ คำตอบก็คือ ยังมีวิธีการอื่นๆ อีกหลายวิธีที่เกิดขึ้นมาในช่วงประมาณ 20 ปีที่ผ่านมา ซึ่งแต่ละวิธีมีทั้งจุดอ่อนและจุดแข็งที่แตกต่างกันไป แต่ทว่ามีอยู่ 4 วิธีการต่อไปนี้ซึ่งเป็นตัวที่เด่นและเป็นที่รู้จักกันดีในแวดวงอุตสาหกรรมซอฟต์แวร์

- วิธีการของ Booch
- วิธีการ Object Modeling Technique (OMT)
- วิธีการ Object-Oriented Software Engineering (OOSE)
- วิธีการของ Coad และ Yourdon

ในปี 1994 ผู้นำของ 3 ค่ายคือ Grady Booch (วิธีการของ Booch), James Rumbaught (วิธีการ OMT) และ Ivar Jacobson (วิธีการ OOSE) ได้ร่วมมือกันและเริ่มต้นกระบวนการผสานวิธีการทั้งสามเข้าเป็นหนึ่งเดียว (Unified) และจุดนี้จึงเกิดเป็นจุดกำเนิดของ ***Unified Modeling Language*** หรือ ***UML*** นั่นเอง

นับแต่นั้นเป็นต้นมาจึงมีการร่วมมือกันของกลุ่มบริษัทต่างๆ ทั้งในอุตสาหกรรมซอฟต์แวร์และคอมพิวเตอร์ โดยมีแกนนำคือ Rational Software Cooperation และในปี ค.ศ. 1997 UML เวอร์ชัน 1.1 ก็ถือกำเนิดขึ้นมาและได้รับการยอมรับให้เป็นมาตรฐานอุตสาหกรรมซอฟต์แวร์นับจากนั้นเป็นต้นมา

กลุ่มของบริษัทยักษ์ใหญ่ที่เข้ามารวมตัวกันเป็นสมาชิกในการร่วมพัฒนา UML ประกอบไปด้วย Microsoft, IBM, Oracle, Sun และ Compaq ซึ่งจะเห็นได้ว่าบริษัทเหล่านี้ล้วนมีชื่อเสียงในแวดวงอุตสาหกรรมซอฟต์แวร์และคอมพิวเตอร์ทั้งสิ้น

1.2 ความหมายของคำว่า "โมเดล"

ก่อนอื่นมารู้จักกับคำว่า "ปัญหา" ก่อน ในโครงการทำซอฟต์แวร์หนึ่งๆ นั้น แต่ละขั้นตอนของการดำเนินงานไม่ว่าจะเป็นการวิเคราะห์ ออกแบบ หรือพัฒนา ก็คือปัญหาที่เราต้องดำเนินการแก้ไข ดังนั้นเมื่อกล่าวถึงคำว่า "ปัญหา" ความ

หมายก็คือ สิ่งที่เราต้องดำเนินการแก้ไขจนกระทั่งโครงการทำซอฟต์แวร์เสร็จสิ้นการดำเนินงาน

คำว่า "โมเดล" ในวลี Modeling Language มีความหมายดังนี้

โมเดล คือ ความพยายามในการที่จะอธิบายปัญหาของซอฟต์แวร์ที่จะดำเนินการพัฒนาขึ้นมา ตัวโมเดลจะแสดงให้เห็นถึงออบเจ็คต์ต่างๆ ที่เกี่ยวข้อง และความสัมพันธ์ระหว่างออบเจ็คต์เหล่านั้น นอกจากนั้นโมเดลยังแสดงให้เห็นถึงวิธีการที่จะแก้ไขปัญหา เราอาจจะใช้ไดอะแกรม เนื้อความ (Text) หรือรูปแบบอื่นๆ ซึ่งเป็นที่ยอมรับกันระหว่างผู้พัฒนาและผู้ใช้ระบบในการนำเสนอโมเดลๆ หนึ่ง UML จะใช้ทั้งไดอะแกรมและเนื้อความเพื่อทำการนำเสนอโมเดลของมัน

ดังนั้นเมื่อกล่าวถึง Modeling Language ความหมายของมันก็คือ ภาษาที่เราเอาไว้อธิบายโมเดลนั่นเอง Modeling Language ทั้งหลายมักจะใช้ไดอะแกรมหรือเนื้อความในการอธิบายถึง ออบเจ็คต์และความสัมพันธ์ระหว่างออบเจ็คต์เหล่านั้น

UML เป็น Modeling Language ภาษาหนึ่งซึ่งสามารถใช้ในการแก้ไขปัญหาในการดำเนินงานโครงการซอฟต์แวร์ ในการแก้ไขปัญหาหนึ่งๆ UML จะใช้โมเดลที่มีรูปแบบแตกต่างกันจำนวนหนึ่งโดยแต่ละโมเดลจะมีมุมมอง (View) ของปัญหาในแง่ที่แตกต่างกันออกไป แต่เมื่อเอาโมเดลเหล่านั้นมาประกอบกันเข้า เราก็จะสามารถดำเนินการวิเคราะห์ ออกแบบ และพัฒนาซอฟต์แวร์ได้อย่างมีประสิทธิภาพ โมเดลที่ UML ใช้จะมีลักษณะต่อเนื่องกันไป กล่าวคือ โมเดลหนึ่งจะอาศัยโมเดลที่สร้างขึ้นมาก่อนหน้านี้เพื่อทำการสร้างโมเดลตัวต่อไป

2. ทำไมต้องใช้ UML

เนื่องด้วยซอฟต์แวร์ในปัจจุบันมีทั้งขนาดและความซับซ้อนที่มากยิ่งขึ้นและยิ่งขึ้นตามลำดับ การดำเนินงานโครงการซอฟต์แวร์ซึ่งเป็นงานที่ละเอียดอ่อนและสลับซับซ้อนจึงจำต้องมีเครื่องมือที่ดีอันหนึ่งเพื่อที่จะสามารถจัดการกับขนาดและความซับซ้อนของซอฟต์แวร์เหล่านั้นได้

วิธีการที่โปรแกรมเมอร์มักจะใช้กันอยู่เสมอคือการกระโดดลงไปทำการพัฒนาโปรแกรม (Coding) เลย กล่าวคือ "คิดไปด้วยและทำการพัฒนาโปรแกรมไปด้วย" โดยปราศจากกระบวนการที่เป็นระบบระเบียบและผสานกลมกลืน นับตั้งแต่การวิเคราะห์ ออกแบบแล้วจึงมาทำการพัฒนาโปรแกรมนั่นเอง สิ่งนี้สามารถเปรียบเทียบได้กับการขึ้นสังเวียนชกมวยเลยโดยปราศจากการเรียนรู้เรื่องมวยอย่างถูกต้องและเหมาะสมก่อน

นอกจากเวลาที่ต้องเสียไปในการพัฒนาซอฟต์แวร์ที่มากกว่าปกติแล้ว ซอฟต์แวร์ที่ได้รับยังอาจเต็มไปด้วยบัก (Software Bugs) ซึ่งยากต่อการแก้ไข อีกทั้งเมื่อคำนึงถึงการนำซอฟต์แวร์เหล่านั้นมาใช้งานซ้ำ (Code Reuse) ในวันข้างหน้า หรือการจัดการกับซอฟต์แวร์เหล่านั้นเมื่อจำเป็นต้องมีเวอร์ชันถัดๆ ไป (Software Configuration Management) การดำเนินการจะกระทำได้ยากมาก

UML เป็นเครื่องมือที่ดีอันหนึ่งซึ่งจะทำให้ผู้ประกอบการซอฟต์แวร์สามารถดำเนินการทุกขั้นทุกตอนอย่างสมเหตุสมผลอย่างเป็นระบบระเบียบและมีความต่อเนื่องกันไปตั้งแต่ต้นจนจบการดำเนินงาน

3. กลไกการทำงานเบื้องต้นของ UML

จากที่ได้กล่าวเอาไว้ในตอนต้นว่า UML ประกอบด้วยโมเดลจำนวนหนึ่งที่นำเอามาใช้ร่วมกันเพื่อการดำเนินงานโครงการซอฟต์แวร์ โมเดลดังกล่าวคือ [2]

- Use Case Diagram
- Sequence Diagram
- Class Diagram
- Activity Diagram
- Collaboration Diagram
- Component Diagram
- Deployment Diagram

- Object Diagram
- Statechart Diagram

จากโมเดลต่างๆ ของ UML ที่แสดงให้ดูข้างบน ผู้เขียนเห็นว่าโมเดล 3 โมเดลแรก คือ Use Case Diagram , Sequence Diagram และ Class Diagram เป็นโมเดลพื้นฐานที่ใช้ในระบบงานทั่วไป พอเพียงสำหรับผู้อ่านที่จะทำความเข้าใจเบื้องต้นเกี่ยวกับกลไกการทำงานของ UML ส่วนโมเดลอื่นเป็นส่วนเพิ่มเติมที่ผู้ใช้สามารถเลือกมาใช้เพื่ออธิบายโมเดลที่ใช้ถ่ายถอดแนวความคิดของผู้พัฒนาหรือผู้วิเคราะห์ระบบ

3.1 Use Case Diagram

Use Case แปลตรงตัวมีความหมายว่า กรณีของการใช้งานที่เกิดจากมุมมองของผู้ใช้ระบบ ตัวอย่างง่ายๆ ของ Use Case ก็คือ

Create Order (ทำการลงใบสั่งซื้อสินค้า)
Modify Order (ทำการเปลี่ยนแปลงหรือแก้ไขใบสั่งซื้อสินค้า)
Delete Order (ทำการลบใบสั่งซื้อสินค้า)

หรืออาจจะมองได้ว่ากรณีการใช้งานดังกล่าว ก็คือการอธิบายฟังก์ชันการทำงานต่างๆของระบบนั่นเอง แต่ละกรณีข้างบนถือเป็นหนึ่งกรณีของการใช้งาน หรือเรียกทับศัพท์ว่าหนึ่ง Use Case ต่อจากนี้ไปผู้เขียนจะใช้คำว่า Use Case ทับศัพท์ไปเลย

โดยปกติแล้วเราสามารถสร้างแต่ละ Use Case ขึ้นมาโดยอาศัยการสัมภาษณ์จากกลุ่มผู้ใช้ระบบ กรณีที่เราไม่รู้ว่าใครคือกลุ่มคนเป้าหมายที่เราควรจะไปทำการสัมภาษณ์สามารถหาข้อมูลเหล่านี้ได้จากฝ่ายบริหาร ซึ่งจะชี้ให้เรารู้ว่าแผนกไหน กองไหน และใครที่เราควรจะไปติดต่อเพื่อทำการสัมภาษณ์ มองกันอย่างง่ายๆ กลุ่มคนเหล่านั้นที่เราจะไปทำการสัมภาษณ์แต่ละคน ก็คือ Actor หรือผู้ใช้ระบบนั่นเอง

Actor ในความหมายของ UML ก็คือ ผู้ที่ติดต่อกับระบบโดยอยู่ภายนอกระบบ Actor อาจจะขอให้ระบบทำอะไรให้สักอย่างหนึ่ง หรือในทางกลับกันระบบอาจจะขอให้ Actor นั้นทำอะไรให้สักอย่างหนึ่ง [3] ดังนั้นจริงๆ แล้ว Actor ยังหมายความรวมถึงสิ่งอื่นๆ ที่อยู่นอกระบบซึ่งสามารถทำการติดต่อกับระบบได้ อาทิเช่น ระบบสินค้าคงคลัง ระบบบัญชี ระบบพัสดุ เป็นต้น

มาดูกันที่ตัวอย่างของ Use Case เพื่อทำความเข้าใจหรือมองเห็นภาพได้ง่ายขึ้น ตัวอย่างในรูปที่ 1 คือ ตัวอย่างของ Use Case เพื่อทำการลงข้อมูลการสั่งซื้อสินค้าเข้าไปในระบบ โดยแสดงเป็นรูปแบบ (Template) อย่างง่ายๆ รูปแบบหนึ่ง รูปแบบอย่างง่ายนี้สามารถใช้ในการนิยาม Use Case ทั่วๆ ไป รูปแบบดังกล่าวมีองค์ประกอบดังนี้

- ชื่อของ Use Case
- ภาพรวมของการทำงาน (Overview)
- Actor หลัก (Primary Actor)
- Actor รอง (Secondary Actor)
- จุดเริ่มต้น (Starting Point)
- จุดสิ้นสุด (End point)
- การทำงานของ Use Case (Flow of Events)
- การทำงานของ Use Case เมื่อมีปัญหาเกิดขึ้น (Alternative Flow of Events)
- ผลของการทำงานของ Use Case (Measurable Result)

แต่ละองค์ประกอบมีความหมายดังนี้

- ชื่อของ Use Case : เป็นการกำหนดชื่อของ Use Case ตัวนี้ ชื่อที่ใช้ควรสั้นและกระชับแต่สื่อความหมายที่ดีของ Use Case ตัวนี้
- ภาพรวมของการทำงาน (Overview) : อธิบายภาพการทำงานของ Use Case ตัวนี้โดยรวมว่าทำอะไร ส่วนนี้ควรมีความยาวไม่เกิน 5 บรรทัด
- Actor หลัก (Primary Actor) : ใครคือผู้ใช้ของ Use Case ตัวนี้ จากตัวอย่าง Use Case : Create Order ผู้ใช้คือ ตัวแทนฝ่ายขายสินค้า

รูปที่ 1 : ตัวอย่างของ Use Case เพื่อทำการลงข้อมูลการสั่งซื้อสินค้าเข้าไปในระบบ

Use Case : Create Order

ภาพรวมของการทำงาน (Overview)

จุดประสงค์หลักของ Use Case นี้คือ เพื่อทำการลงข้อมูลในใบสั่งซื้อสินค้าจากลูกค้า

Actor หลัก (Primary Actor)

ตัวแทนฝ่ายขายสินค้า

Actor รอง (Secondary Actor)

ไม่มี

จุดเริ่มต้น (Starting Point)

Use Case ตัวนี้เริ่มต้นเมื่อ Actor ตัวแทนฝ่ายขายสินค้าขอให้ระบบทำการลงข้อมูลการสั่งซื้อสินค้า

จุดสิ้นสุด (End point)

คำขอเพื่อทำการลงข้อมูลการสั่งซื้อสินค้าเสร็จสิ้นสมบูรณ์หรือไม่ก็ถูกยกเลิก

การทำงานของ Use Case (Flow of Events)

จาก User Interface บนจอเพื่อทำการลงข้อมูลการสั่งซื้อ Actor จะต้องทำการใส่ข้อมูลเกี่ยวกับการสั่งซื้อ เป็นต้นว่า ***วันที่ลูกค้าต้องการให้สินค้าส่งมอบถึงมือ (Required Date) ปริมาณที่ต้องการสั่งซื้อ (Quantity) ต้องการให้ส่งมอบสินค้าโดยบริษัทรับส่งสินค้าไหน (ShipVia) ต้องการให้ส่งมอบสินค้าที่ไหน (ShipAddress)*** หลังจากนั้นแล้ว Actor สามารถเลือกที่จะทำการบันทึกข้อมูลลงไปไว้ในฐานข้อมูล หรือยกเลิกการทำงานทั้งหมด ถ้า Actor เลือกทำการบันทึก ใบสั่งซื้อสินค้าใบใหม่ก็จะถูกสร้างขึ้นมาในฐานข้อมูล

การทำงานของ Use Case เมื่อมีปัญหาเกิดขึ้น (Alternative Flow of Events)

ถ้าไม่มีสินค้าที่ต้องการอยู่ในคลังสินค้า ระบบจะต้องแจ้งให้ Actor ทราบพร้อมกันนั้นก็ต้องยกเลิกการทำงานที่เหลือของ Use Case นี้

ผลของการทำงานของ Use Case (Measurable Result)

จะมีใบสั่งซื้อสินค้าใหม่ 1 ใบ ถูกสร้างขึ้นมาในระบบ

หมายเหตุ ในแต่ละส่วนของ Use Case ข้างบน เราได้ใช้ภาษาอังกฤษประกอบการอธิบาย เพื่อให้รู้ที่มาว่าในภาษาอังกฤษเขาใช้ Term ต่างๆ กันอย่างไร แต่เมื่อมีการนำ Use Case ไปใช้งานจริง ไม่จำเป็นต้องมีภาษาอังกฤษอีกต่อไป

- Actor รอง (Secondary Actor) : ใครคือผู้ใช้อื่นๆ นอกเหนือจาก Actor หลัก ที่เป็นผู้ใช้ Use Case ตัวนี้ ตัวอย่าง Use Case : Create Order ข้างบนไม่มีผู้ใช้อื่นๆ ที่จะใช้ Use Case ตัวนี้ร่วมกับ Actor หลัก ในบางระบบงานอาจจะเป็นไปได้ว่ามีผู้ใช้อื่น เช่น ผู้จัดการฝ่ายขายซึ่งสามารถใช้ Use Case ตัวนี้ร่วมกับตัวแทนฝ่ายขายสินค้า
- จุดเริ่มต้น (Starting Point) : เป็นการอธิบายเงื่อนไขหรือข้อกำหนดต่างๆ ณ จุดเริ่มต้นของ Use Case โดยเงื่อนไขเหล่านี้ โดยปกติแล้วจะต้อง "มี" หรือ "เป็น" อย่างนั้นก่อนการทำงานของ Use Case นี้ ในตัวอย่างของ Use Case : Create Order ที่แสดงให้ดู เงื่อนไขคือ Use Case จะเริ่มต้นเมื่อ Actor ขอให้ระบบทำการลงข้อมูลการสั่งซื้อสินค้าในใบสั่งซื้อสินค้า
- จุดสิ้นสุด (End point) : คล้ายกับจุดเริ่มต้น เป็นการอธิบายเงื่อนไขหรือข้อกำหนดต่างๆ ณ จุดสิ้นสุดการทำงานของ Use Case ตัวนี้ ในตัวอย่างของ Use Case : Create Order ที่แสดงให้ดู เงื่อนไขที่จุดสิ้นสุดคือ คำขอเพื่อการลงบันทึกข้อมูลการสั่งซื้อสินค้าจะต้องเสร็จสิ้นสมบูรณ์หรือไม่ก็ถูกยกเลิก กล่าวคือสิ่งใดสิ่งหนึ่งเท่านั้นจะเกิดขึ้น แต่ไม่ใช่ทั้งสองเกิดขึ้นพร้อมๆ กัน
- การทำงานของ Use Case (Flow of Events) : อธิบายขั้นตอนของการทำงานของ Use Case นี้ ตัวอย่างของ Use Case ข้างบนแสดงให้เห็นขั้นตอนการทำงานของ Use Case นี้เพื่อทำการลงบันทึกข้อมูลการสั่งซื้อสินค้านับตั้งแต่เริ่มต้นจนกระทั่งสิ้นสุด ขั้นตอนการทำงานเหล่านี้เรียกว่า ขั้นตอนการทำงานปกติของ Use Case
- การทำงานของ Use Case เมื่อมีปัญหาเกิดขึ้น (Alternative Flow of Events) : การทำงานของ Use Case ในกรณีเมื่อมีปัญหาใดๆ ก็ตามเกิดขึ้น และทำให้การทำงานตามปกติของ Use Case ไม่สามารถดำเนินต่อไปได้ จะทำให้มีการส่งต่อการทำงานของ Use Case จากขั้นตอนการทำงานปกติ (ในส่วนของการทำงานของ Use Case) มาที่ส่วนนี้ของ Use Case ในตัวอย่างของ Use Case : Create Order ที่แสดงให้ดู เมื่อระบบพบว่าสินค้าตัวที่ลูกค้าต้องการไม่มีอยู่ในคลังสินค้า นั่นหมายถึงระบบมีปัญหาเกิดขึ้น ระบบก็จะทำการแจ้งให้ Actor ทราบและยกเลิกการทำงานที่เหลือของ Use Case นี้ทันที
- ผลของการทำงานของ Use Case (Measurable Result) : ผลการทำงานของ Use Case แสดงให้เห็นว่าเมื่อเสร็จสิ้นการทำงานของ Use Case นี้แล้ว จะมีอะไรเกิดขึ้นเป็นผลพวง ในตัวอย่างของ Use Case : Create Order ที่แสดงให้ดู จะมีใบสั่งซื้อสินค้าใบใหม่เพิ่มขึ้น 1 ใบในฐานข้อมูลของเรา

รูปแบบของ Use Case ที่แสดงให้ดูในรูปที่ 1 เป็นเพียงรูปแบบง่ายๆ รูปแบบหนึ่งเท่านั้น ส่วนการใช้งานจริงผู้ใช้อาจเพิ่มเติมส่วนต่างๆ ลงไปเองได้ตามความเหมาะสม ทั้งนี้ขึ้นอยู่กับการตกลงกันระหว่างกลุ่มผู้พัฒนาระบบและผู้ใช้ระบบนั้นๆ

3.2 Sequence Diagram

Sequence Diagram คือ ไดอะแกรมที่แสดงลำดับขั้นตอน (Sequence) การทำงานภายในของ Use Case ตัวหนึ่ง โดยตัว Use Case เองแล้วเราจะไม่สามารถมองเห็นลำดับขั้นตอนการทำงานภายในของ Use Case ตัวนั้นได้ ตัว Sequence Diagram ต่างหากที่ทำให้เราสามารถมองเห็นลำดับขั้นตอนการทำงานภายในของ Use Case ตัวนั้นได้

นอกจากนี้ Sequence Diagram ยังแสดงให้เห็นถึงการติดต่อกันระหว่าง

- ออบเจ็คต์ต่างๆ ของ Use Case นั้น และ
- ออบเจ็คต์ และ Actor ของ Use Case นั้น

การติดต่อกันดังกล่าวจะทำให้มีข้อความ (Message) วิ่งไปมาในไดอะแกรมนั้น

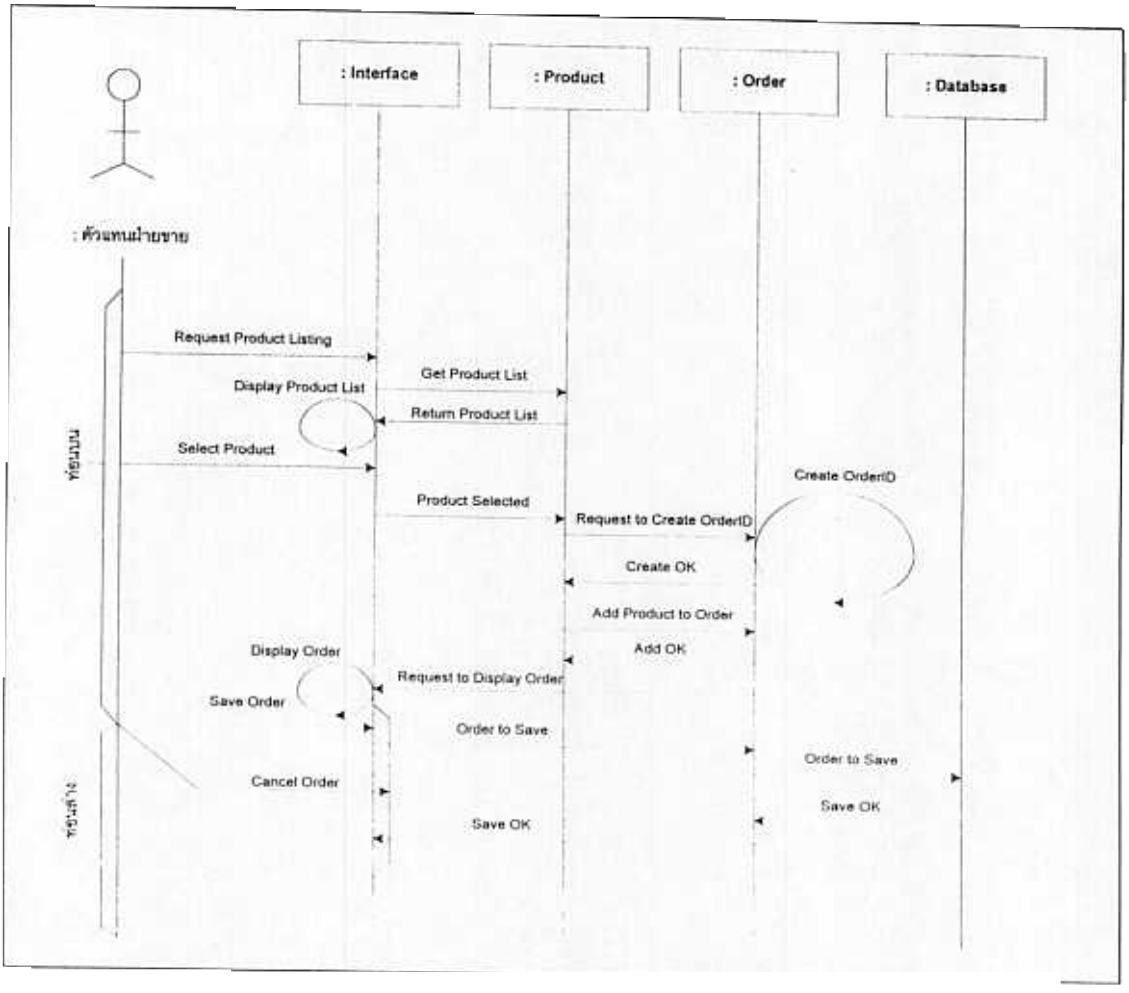
ที่กล่าวว่า "ออบเจ็คต์ต่างๆ ของ Use Case" สิ่งนี้หมายความว่าใน Use Case ตัวหนึ่งๆ จะมีออบเจ็คต์ที่เกี่ยวข้องกับ Use Case นั้นมากกว่า 1 ประเภท อาทิเช่น Use Case : Create Order อย่างน้อยจะประกอบไปด้วยออบเจ็คต์ 2 ประเภท กล่าวคือ Product และ Order

Product คือ สินค้าที่ลูกค้าต้องการซื้อ ส่วน Order คือ ใบสั่งซื้อสินค้าใบหนึ่งของลูกค้า ดังนั้นจึงสามารถกล่าวได้ว่าออบเจ็คต์ Product และ Order ก็คือ "ออบเจ็คต์ของ Use Case นี้"

รูปที่ 2 แสดงให้เห็นถึง Sequence Diagram ของ Create Order ซึ่งจะเห็นได้ว่ามี ออบเจ็คต์ที่เกี่ยวข้อง 4 ประเภท กล่าวคือ Interface, Product, Order และ Database (สังเกตุกล่องสี่เหลี่ยมข้างบนไดอะแกรม) ส่วนรูปตัวคนทางซ้ายมือก็คือ Actor ตัวแทนฝ่ายขาย (ซึ่งก็คือ Actor ของ Use Case: Create Order นั่นเอง)

แกนในแนวตั้ง 5 แกนคือ แกนเวลาของ ไดอะแกรม ลูกศรในไดอะแกรมคือการส่งข้อความจากออบเจ็คต์หนึ่งไปหาออบเจ็คต์อีกตัวหนึ่ง เป็นต้นว่าข้อความ Get Product List ซึ่งส่งจากออบเจ็คต์ Interface ไปยังออบเจ็คต์ Product ข้อความที่อยู่ด้านบนของไดอะแกรมจะเป็นข้อความที่เกิดก่อนข้อความที่อยู่ด้านล่างเรียงตามลำดับเวลา เป็นต้นว่า ข้อความ Get Product List จะเกิดขึ้นก่อนข้อความ Return Product List ดังนั้นจึงสามารถกล่าวได้ว่า Sequence Diagram เป็นไดอะแกรมที่เน้นเรื่องการเกิดขึ้นเรียงตามลำดับเวลา (Temporal)

รูปที่ 2 Sequence Diagram ของ Create Order



ของข้อความต่างๆ ภายใน ไดอะแกรม

ก่อนอธิบายตัว Sequence Diagram สำหรับ Create Order ขออธิบายข้อจำกัดของ Create Order ที่เราใช้เป็นตัวเดินเรื่องทั้งหมดก่อน เพื่อจุดประสงค์ในการแสดงให้ดูเป็นตัวอย่าง การลงบันทึกในใบสั่งซื้อนี้จึงเป็นการลงบันทึกแบบง่ายๆ กล่าวคือ สามารถมีสินค้าในใบสั่งซื้อหนึ่งใบได้เพียงประเภทเดียว (ในทางปฏิบัติอาจจะมีหลายประเภทสินค้าในหนึ่งใบสั่งซื้อได้) ดังนั้นตัวอย่าง Sequence Diagram สำหรับ Create Order จะแสดงให้เห็นถึงลำดับของเหตุการณ์การลงบันทึกแบบง่ายๆ ดังกล่าว

Sequence Diagram สำหรับ Create Order มีการทำงานเรียงตามลำดับเวลาดังนี้

1. Request Product Listing : จาก User Interface ที่แสดงบนหน้าจอ Actor จะทำการขอให้ระบบแสดงรายการของสินค้าทั้งหมดออกมาทางหน้าจอ
2. Get Product List : ออบเจ็คต์ User Interface หรือเรียกง่ายๆ ว่า Interface ก็จะทำการส่งต่อคำขอนั้นไปให้ "เจ้าของเรื่อง" ซึ่งก็คือ Product
3. Return Product List : "เจ้าของเรื่อง" ออบเจ็คต์ Product ก็จะทำการส่งรายการสินค้าทั้งหมดที่มีอยู่กลับคืนมาให้ Interface
4. Display Product List : ออบเจ็คต์ Interface เมื่อได้รับรายการสินค้าแล้วก็จะทำการแสดงรายการของสินค้าเหล่านั้นออกมาทางหน้าจอให้ Actor ดู ซึ่งจะเห็นได้ว่าเป็นการส่งข้อความเข้าหาตัวเอง (Reflexive Message)
5. Select Product : Actor จะทำการคลิกที่สินค้าตัวหนึ่ง (จากรายการสินค้าทั้งหมดที่แสดงบนหน้าจอ) ซึ่งลูกค้าต้องการซื้อ
6. Product Selected : Interface ก็จะทำการส่งต่อข้อมูลสินค้าที่ Actor ทำการเลือกแล้วนั้นไปยัง Product
7. Request to Create OrderID : เมื่อ Product ได้รับข้อมูลสินค้านั้นแล้ว ก็จะส่งคำขอไปยัง Order เพื่อขอให้สร้าง ID ใหม่ให้ตัวหนึ่งสำหรับใบสั่งซื้อสินค้าใบนี้
8. Create OrderID : เจ้าของเรื่องคือ Order ก็จะทำการสร้าง ID ใหม่ขึ้นมา ID หนึ่งสำหรับใบสั่งซื้อนี้ ซึ่งจะเห็นได้ว่าเป็นการส่งข้อความเข้าหาตัวเอง (Reflexive Message) เช่นเดียวกับ Display Product List
9. Create OK : หลังจากนั้นออบเจ็คต์ Order ก็ส่งข้อความไปบอก Product ว่าคำขอเพื่อสร้าง ID นั้นได้ทำสำเร็จแล้ว
10. Add Product to Order : หลังจากนั้น Product จึงทำการส่งข้อมูลของสินค้า (ที่ลูกค้าต้องการซื้อ) ไปให้ Order เพื่อทำการลงบันทึกในใบสั่งซื้อสินค้านั้น
11. Add OK : เมื่อเสร็จสิ้นการบันทึกแล้ว Order จึงบอกให้ Product รู้ว่าได้ดำเนินการเสร็จแล้ว
12. Request to Display Order : พร้อมกันนั้น Order ก็ส่งคำขอไปยัง Interface เพื่อขอให้ Interface ทำการแสดงข้อมูลรายละเอียดทั้งหมดของใบสั่งซื้อนี้ออกมาทางหน้าจอเพื่อให้ Actor ดู
13. Display Order : Interface ทำการแสดงข้อมูลรายละเอียดของใบสั่งซื้อนี้ออกทางหน้าจอ (ตามคำขอจาก Order)
14. Save Order : ต่อจากนั้น โดย Interface Actor ก็จะทำการคลิกปุ่ม Save เพื่อทำการ Save ข้อมูลของใบสั่งซื้อสินค้านั้น
15. Cancel Order : ให้สังเกตในไดอะแกรมที่ข้อความ Save Order ที่ส่งจากตัวแทนฝ่ายขายจะมีข้อความอีกอันหนึ่งคือ ข้อความ Cancel Order แยกออกมาจากจุดร่วมเดียวกัน จุดที่มีการแยกออกดังกล่าวเป็นสัญญลักษณ์ของการตัดสินใจที่จะเลือกทำอันใดอันหนึ่งเท่านั้น ความหมายของจุดแยกนี้ก็คือ ถ้า Actor ตัดสินใจทำการ Save ขั้นตอนการทำงานที่เหลือเรียงตามลำดับเวลาจะประกอบไปด้วยข้อความ

Order to Save (จาก Interface): Interface ส่งข้อมูลของใบสั่งซื้อนี้ไปให้เจ้าของเรื่อง Order

Order to Save (จาก Order): เจ้าของเรื่อง Order ส่งต่อข้อมูลนั้นให้ Database เพื่อจะได้ทำการบันทึกข้อมูลนั้นต่อไป

Save OK (จาก Database): Database แจ้งให้ Order รู้ว่าได้ทำการบันทึกเรียบร้อยแล้ว

Save OK (จาก Order): Order แจ้ง Interface ว่าการบันทึกข้อมูลได้ดำเนินการเสร็จสิ้นแล้ว

ถ้าหากตัดสินใจยกเลิก (อาจเป็นเพราะราคาสินค้าที่ลูกค้าคิดว่าแพงเกินไป) คือ ไม่ Save การทำงานจะสิ้นสุดทันที โปรดสังเกตุว่าการตัดสินใจเลือกทำการ Save หรือ Cancel เป็น Statement หนึ่งที่ปรากฏในส่วนของการทำงานของ Use Case ด้วย (ให้ดูในส่วนของการทำงานของ Use Case ของ Use Case: Create Order ด้วย)

3.2.1 ความสอดคล้องกันระหว่าง Use Case และ Sequence Diagram

Sequence Diagram ในรูปที่ 2 จริงๆ แล้วถูกสร้างขึ้นมาจาก Use Case: Create Order (เนื่องจากบทความนี้เป็นบทความที่แสดงให้เห็นภาพโดยรวมของ UML จึงยังไม่สอนวิธีการสร้าง Sequence Diagram จาก Use Case) ในเบื้องต้นนี้เราจะชี้ให้เห็นถึงความสอดคล้องกันระหว่าง Use Case และ Sequence Diagram สำหรับ Create Order

ท่อนบนของ Sequence Diagram (หาคำว่า " ท่อนบน" ในไดอะแกรม) สำหรับ Create Order เทียบเท่ากับ Statement ดังนี้ (ให้ดูในส่วนการ**ทำงานของ Use Case** ของ Use Case: Create Order ในรูปที่ 1 ประกอบด้วย)

"จาก User Interface บนจอเพื่อทำการบันทึกข้อมูลการสั่งซื้อ Actor จะต้องทำการใส่ข้อมูลเกี่ยวกับการสั่งซื้อ เป็นต้นว่า วันที่ลูกค้าต้องการให้สินค้าส่งมอบถึงมือ (RequiredDate) ปริมาณที่ต้องการสั่งซื้อ (Quantity) ต้องการให้ส่งมอบสินค้าโดยบริษัทรับส่งสินค้าไหน (ShipVia) ต้องการให้ส่งมอบสินค้าที่ไหน (ShipAddress)"

ท่อนล่างของ Sequence Diagram (หาคำว่า " ท่อนล่าง" ในไดอะแกรม) เทียบเท่ากับ Statement ส่วนที่เหลือของ Use Case นี้ กล่าวคือ "หลังจากนั้น Actor สามารถเลือกที่จะทำการบันทึกข้อมูลลงไปไว้ในฐานข้อมูล หรือยกเลิกการทำงานทั้งหมด ถ้า Actor เลือกทำการบันทึกข้อมูล ใบสั่งซื้อสินค้าใบใหม่ก็จะถูกสร้างขึ้นมาในฐานข้อมูล"

ดังนั้น ณ จุดนี้จึงขอสรุปเกี่ยวกับ Sequence Diagram ไว้ดังนี้

Sequence Diagram ถูกสร้างขึ้นมาจาก Use Case ตัว Use Case มิได้บอกรายละเอียดของการทำงานภายในว่าเป็นอย่างไร Sequence Diagram ต่างหากเป็นตัวบอกว่ามีรายละเอียดการทำงานภายในอย่างไร และรายละเอียดการทำงานภายในเหล่านั้นก็คือ ข้อความที่วิ่งไปมาในไดอะแกรมนั้น

ส่วนคำถามถัดมาที่ผู้อ่านอาจอยากจะถามคือ ข้อความที่วิ่งไปมาเหล่านั้นมีประโยชน์อะไร เราจะไขคำตอบสำหรับคำถามนี้ในหัวข้อ Class Diagram ข้างล่าง

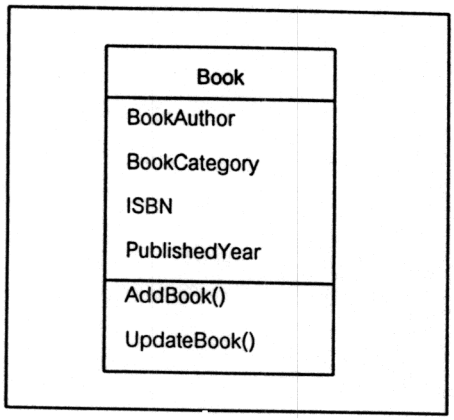
3.3 Class Diagram

ก่อนอื่นขอนิยามคำว่า "Class" ก่อน "Class" คือ เซ็ตของออบเจ็คต์ที่มีคุณสมบัติ (Properties) และพฤติกรรม (Methods) เช่นเดียวกัน ตัวอย่างเช่น Class ของหนังสือมีลักษณะดังแสดงในรูปที่ 3 และเราอาจมีออบเจ็คต์ต่างๆ ดังนี้อยู่ใน Class Book ดังตารางที่ 1 ข้างล่าง

ในตารางจะเห็นได้ว่าทั้ง 3 ออบเจ็คต์ที่แสดงในตารางก็คือ เซ็ทของออบเจ็คต์ที่มีคุณสมบัติ BookAuthor, BookCategory, ISBN และ PublishedYear เหมือนกัน แต่เนื้อหาสาระ (Content) ของคุณสมบัติเหล่านั้นจะไม่เหมือนกัน

เช่น หนังสือที่มี ISBN 1111 ผู้แต่ง คือ รัชนี ส่วนหนังสือที่มี ISBN 1234 ผู้แต่ง คือ ชาลี

รูปที่ 3 ตัวอย่าง Class Book



ตารางที่ 1 ตัวอย่างของ Class Book

Book-Author	Book-Category	ISBN	Published-Year
รัชนี	Algorithm	1111	1998
ชาลี	OS	1234	1995
นพพร	DBMS	1345	1996

นอกจากการมีคุณสมบัติที่เหมือนกันแล้ว ออบเจ็คต์ในคลาสเดียวกันยังมีพฤติกรรมที่เหมือนกันอีกด้วย ตัวอย่างเช่น ในการเปลี่ยนแปลงรายละเอียดของหนังสือเล่มหนึ่งๆ ออบเจ็คต์ทั้งหลายที่อยู่ใน Class นี้สามารถใช้พฤติกรรม UpdateBook() เพื่อทำการเปลี่ยนแปลงรายละเอียดของหนังสือเล่มนั้นๆ

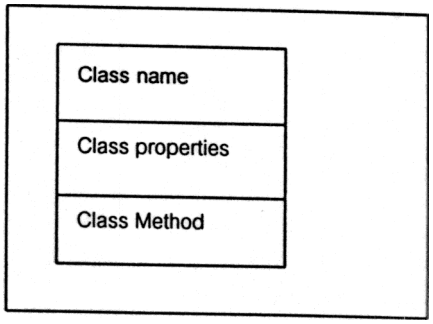
จากความหมายของ Class ที่ได้นิยามไว้ข้างบน ความหมายของ Class Diagram ก็คือภาพที่ประกอบไปด้วย Class ต่างๆ ที่มาประกอบหรือรวมตัวกันกลายเป็นระบบหรือซอฟต์แวร์ตัวหนึ่ง เป็นต้น ยกตัวอย่างเช่น Class : Interface, Product, Order, Database เป็นต้น

โดย Class ทั้งสี่ส่วนนี้มีที่มาจาก Sequence Diagram สำหรับ Create Order (ให้สังเกตุว่ากล่องสี่เหลี่ยมข้างบน Diagram คือ ออบเจ็คต์ของคลาสทั้งสี่นั่นเอง)

ในการดำเนินการพัฒนาซอฟต์แวร์ตัวหนึ่งๆ โปรแกรมเมอร์จะทำการพัฒนาซอฟต์แวร์ตาม Class ต่างๆ เหล่านั้นทั้งหมดที่ได้กำหนดขึ้นมา

Class Diagram ใน UML มีรูปแบบดังรูปที่ 4 คือ

รูปที่ 4 รูปแบบของ Class Diagram



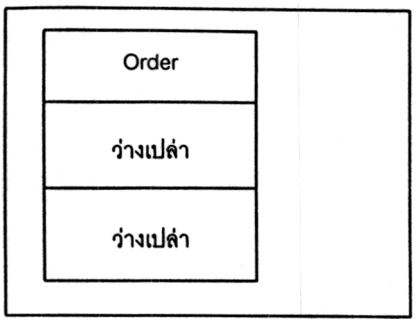
ซึ่งจะเห็นได้ว่าแต่ละส่วนของไดอะแกรมนี้ (ซึ่งมีอยู่ 3 ส่วน) จะสอดคล้องกับ Class Book ที่แสดงในรูปที่ 3

ในทางปฏิบัติระบบที่พัฒนาจะมี Class ต่างๆที่เกี่ยวข้องมากกว่า 1 Class เช่น ระบบการสั่งซื้อสินค้า อาจจะมี Class customer , Class Order ซึ่ง Class ทั้งสองอาจมีความสัมพันธ์กันในลักษณะใดลักษณะหนึ่ง แต่ในบทความฉบับนี้ต้องการเน้นให้เห็นถึงสัญญลักษณ์ของ Class Diagram และองค์ประกอบของ Class จึงยังไม่กล่าวถึงความสัมพันธ์ของ Class เหล่านั้น

เราจะใช้ตัวอย่างการทำการบันทึกข้อมูลเกี่ยวกับการสั่งซื้อสินค้า Create Order ในสองหัวข้อที่แล้วกล่าวคือ Use Case และ Sequence Diagram เพื่อแสดงให้ผู้อ่านเห็นถึงการได้มาซึ่ง Class Diagram สำหรับ Class Order (ส่วน Class อื่นๆ ก็จะอาศัยวิธีการเดียวกัน เพื่อจะให้ได้มาซึ่ง Class Diagram ของมัน)

เมื่อตอนเริ่มต้นสิ่งแรกที่เรามีสำหรับ Class Order คือ Class Order ซึ่งว่างเปล่าดังแสดงในรูปที่ 5

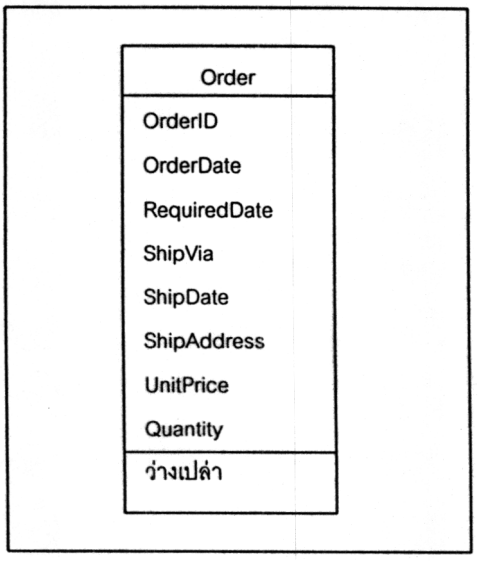
รูปที่ 5 ตัวอย่าง Class Order ที่ว่างเปล่า



3.3.1 Class Properties

ในส่วนของ Class Properties คุณสมบัติที่เกี่ยวข้องกับ Class Order มีดังนี้

รูปที่ 6 ตัวอย่าง Class Order



คุณสมบัติดังกล่าวได้มาจาก Use Case : Create Order โดยสังเกตุจากตัวอักษรเข้มใน **"ส่วนของการทำงานของ Use Case"** ความจริงอย่างหนึ่งที่เรารู้ก็คือ ในขณะที่เราทำการสัมภาษณ์ Actor ของ Use Case นี้ Actor จะเล่าให้เราฟังว่า

"เมื่อมีลูกค้าโทรเข้าสั่งของ ลูกค้าจะต้องการบอกเราในรายละเอียด เช่น วันที่ลูกค้าต้องการให้สินค้าส่งมอบถึงมือ (RequiredDate) ต้องการสินค้านั้นกี่ชิ้น (Quantity) ต้องการให้ส่งมอบสินค้าอย่างไร (ShipVia) ต้องการให้ส่งมอบสินค้าที่ไหน (ShipAddress) เป็นต้น"

Actor โดยปกติจะต้องมีรายละเอียด เช่น ราคาของสินค้าต่อชิ้น (UnitPrice) วันที่ที่สินค้าถูกส่งออกไปโดยบริษัทที่รับส่งของ (ShipDate) ส่วนคุณสมบัติของ OrderID ซอฟต์แวร์ที่จะดำเนินการสร้างจะทำการคำนวณให้เราโดยอัตโนมัติ ดังนั้นโดยสรุปแหล่งที่มาของคุณสมบัติของ Class หนึ่งๆ จะมาจาก Use Case (อาจจะมากกว่า 1 Use Case เช่น นอกจากมาจาก Create Order แล้วยังสามารถมาจาก Modify Order หรือ Delete Order ได้ด้วย) และข้อมูลจากการสัมภาษณ์กับ Actor ของ Use Case นั้น

3.3.2 Class Methods

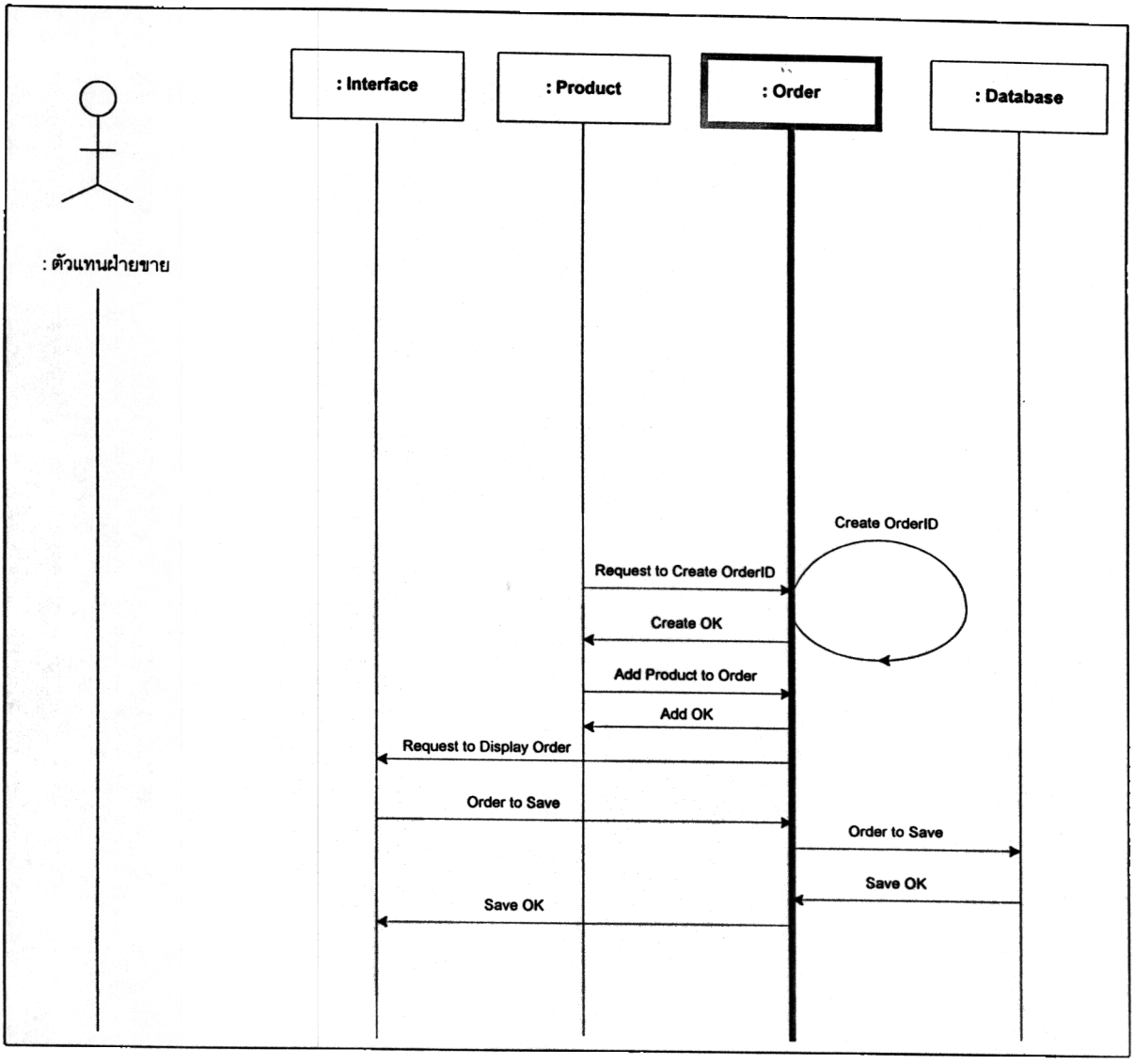
สิ่งสุดท้ายที่เราต้องการคือ Class Methods ซึ่งจะได้มาจาก Sequence Diagram สิ่งที่เราควรทราบเบื้องต้นก็คือ ข้อความแต่ละข้อความใน Sequence Diagram สามารถเทียบเท่ากับ Method 1 Method มาดูกันที่ตัวอย่างการกำหนด Class Methods ของ Class Order เพื่อจะได้เห็นภาพที่แท้จริงกันเลย

หลักการหา Method มีดังนี้

1. จาก Sequence Diagram สำหรับ Create Order พิจารณาเฉพาะข้อความที่วิ่งเข้าหรือออกจากแกนเวลาของ Order เท่านั้น รูปที่ 7 เป็นไดอะแกรมที่ตัดออกมาจากไดอะแกรมตัวสมบูรณ์ (ในรูปที่ 2) ไดอะแกรมที่ตัดออกมานี้ประกอบด้วยข้อความเฉพาะที่เกี่ยวข้องกับ Class Order เท่านั้น ซึ่งจะเห็นได้ว่ามีลูกศรวิ่งเข้าออกจากแกนเวลาของ Order

สิ่งหนึ่งที่ผู้อ่านอาจจะรู้สึกว่าน่าจะเป็นคือ ข้อความต่างๆที่วิ่งไปมาแต่ละข้อความน่าจะมี Method ตัวหนึ่งมาทำหน้าที่ในการสื่อสารนั้น ยกตัวเอย่างเช่น ข้อความ Request to Create OrderID จาก Product ไปหา Order จะมี Method : RequestToCreateOrderID() มาทำหน้าที่ในการสื่อสาร ซึ่งในที่นี้ก็คือ คำขอให้ Order ช่วยสร้าง ID ให้ 1 ID สำหรับใบสั่งซื้อสินค้านี้

รูปที่ 7 Sequence Diagram สำหรับ Create Order เฉพาะข้อความที่เกี่ยวข้องกับ Class Order



ณ จุดนี้คำถามหนึ่งที่ผู้อ่านอยากจะถามก็คือ ข้อความ Request to Create Order ID ควรจะเป็นของ Class ไหน ระหว่าง Class Product และ Order คำตอบของคำถามนี้จึงเกิดเป็นหลักการข้อที่ 2

2. หลักการในข้อนี้เป็นหลักการอันหนึ่งที่สามารถใช้ได้และเป็นที่ยอมรับโดยทั่วไป

"ถ้าข้อความที่พิจารณามีเนื้อหาสาระหนัก (weight) ไปทาง Class ไหนมากกว่า (ระหว่าง 2 Class) ข้อความนั้นก็จะกลายเป็น Method หนึ่ง Method ของ Class นั้น"

โดยใช้หลักการนี้ เราสามารถทำการกำหนด Method ให้กับ Class Order ได้ดังตารางที่ 2

เราสรุปได้ว่า Methods ของ Class Order ประกอบไปด้วย

- Create OrderID
- Create OK
- Add OK
- Request to Display Order
- Order to Sace (จาก interface)
- Save Order (จาก Order)

ตารางที่ 2 ตารางแสดงข้อความที่เกี่ยวข้องกับคลาสหนึ่งๆ

Message	Class			
	Interface	Product	Order	Database
Request to Create Order ID		∏		
Create Order ID			∏	
Create OK			∏	
Add Product to Order		∏		
Add OK			∏	
Request to Display Order			∏	
Order to Save (จาก Interface)	∏			
Order to Save (จาก Order)			∏	
Save OK (จาก Database)				∏
Save OK (จาก Order)			∏	

(ให้สังเกตุเครื่องหมาย ∏ ในคอลัมน์ Order ในตารางที่ 2)

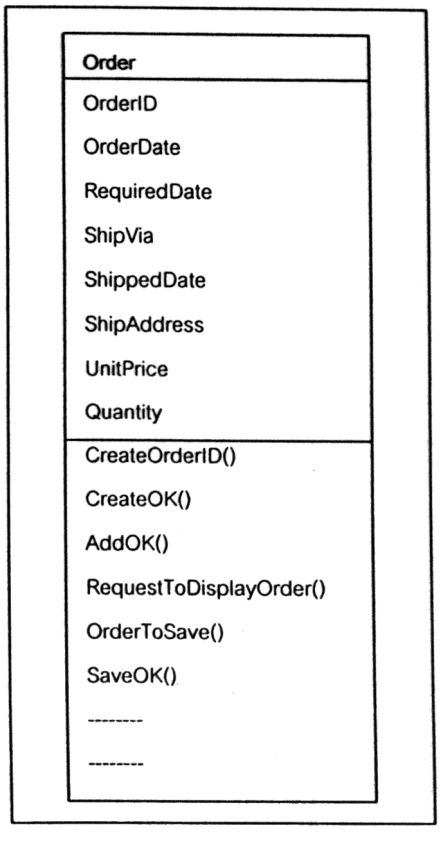
มาดูกันที่ Method อื่นๆ ซึ่งไม่อยู่ใน Class Order กันก่อน สาเหตุของการที่ Methodอื่นๆ (ให้ดูในตารางที่ 2 ประกอบตรงเครื่องหมาย ∏ ที่ไม่อยู่ในคอลัมน์ Order) กล่าวคือ

- RequestToCreateOrderID()
- AddProductToOrder()
- OrderToSave()
- SaveOK() (จาก Database)

Methods ของ Class Order สามารถอธิบายได้ดังนี้

- RequestToCreateOrderID() : ควรจะเป็นของ Product เพราะ Product เป็นคนขอให้ Order ทำการสร้าง OrderID ใหม่ขึ้นมาตัวหนึ่ง
- AddProductToOrder() : ควรจะเป็นของ Product เพราะProduct เป็นคนขอให้ Order ทำการใส่ข้อมูลของสินค้าที่ลูกค้าต้องการซื้อลงไปในใบสั่งซื้อสินค้านั้น
- OrderToSave() : ควรจะเป็นของ Interface เพราะ Interface เป็นคนส่งข้อมูลของใบสั่งซื้อสินค้าไปให้ Order

รูปที่ 8 Class Order และ Method



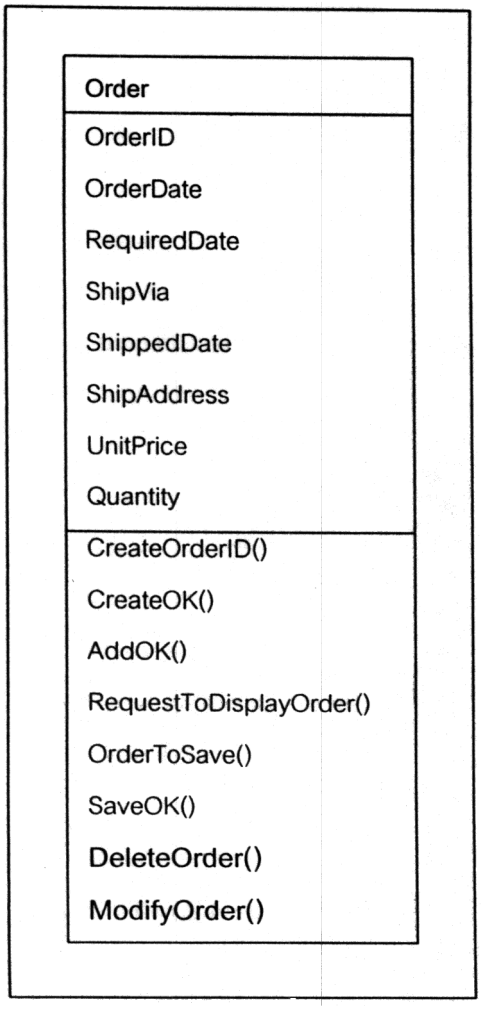
- SaveOK() (จาก Database) : ควรจะเป็นของ Database เพราะ Database เป็นคนแจ้งให้ Order ทราบว่าใบสั่งซื้อสินค้านั้นได้ทำการ Save เรียบร้อยแล้ว

สรุปโดยหลักการก็คือ ถ้า Class 1 เป็นผู้ขอให้ Class 2 ทำอะไรให้ก็ตาม (โดยการส่งข้อความไปให้ Class 2) Class 1 จะมี *น้ำหนักมากกว่า* ทั้งนี้เพราะ Class 1 เท่านั้นถึงจะที่รู้ดีว่ากำลังให้ Class 2 ทำอะไรให้อยู่ ดังนั้น Method ทั้งสี่ข้างบนจึงควรไปอยู่กับ Class อื่นๆ เหล่านั้นที่มิใช่ Order

ส่วน Method ที่เป็นของ Class Order ก็สามารถตอบได้โดยอาศัยหลักการเดียวกับหลักการในย่อหน้าที่แล้ว

3.3.3 Method ของ Class Order จาก Sequence Diagram อื่นๆ

รูปที่ 9 Class Order และ Method



Class Order อาจจะมี Sequence Diagram อื่นๆ (นอกจาก Create Order Sequence Diagram) ที่เกี่ยวข้องกับมัน อาทิเช่น Delete Order Sequence Diagram หรือ Modify Order Sequence Diagram เป็นต้น Sequence Diagram อื่นๆ เหล่านี้ก็จะเป็นที่มาของ Method อื่นๆ (นอกเหนือจาก Mehtod ในรูปที่ 8) เป็นต้นว่า DeleteOrder(), ModifyOrder() , และอื่นๆ ดังนั้น Class Diagram สำหรับ Order จึงสามารถรวมเอา Methods อื่นๆ เหล่านั้นเข้าไปด้วย กล่าวคือ (ให้ดูที่ Method ที่เป็นตัวอักษรเข้ม)

3.3.4 Class อื่นๆ นอกจาก Class Order

สำหรับ Class อื่นๆ เช่น Class : Interface, Product, Database เป็นต้น ผู้อ่านสามารถใช้วิธีการเดียวกันที่นำเสนอมาตั้งแต่ต้นในหัวข้อ Class Diagram เพื่อทำการหาคุณสมบัติและ Method ของมัน ในที่สุดแล้วเราก็จะได้ไดอะแกรมของทุก Class ในระบบของเราออกมาและสามารถนำเอาไปทำการพัฒนาเป็นโปรแกรมได้ต่อไป

4. ภาพโดยรวมของกระบวนการทำงานทั้งหมดของ UML

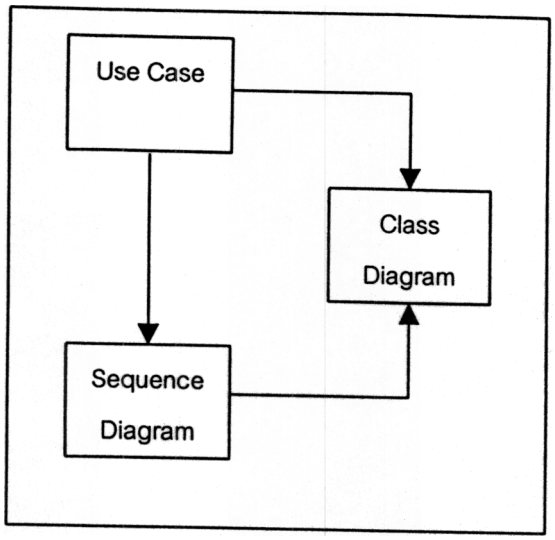
จากในหัวข้อ 3.1, 3.2 และ 3.3 ที่ได้นำเสนอไปแล้วในเรื่อง Use Case, Sequence Diagram และ Class Diagram เรียงตามลำดับหัวข้อดังกล่าว เราจึงสามารถสรุปภาพโดยรวมของกระบวนการทำงานทั้งหมดของ UML ดังแสดงไว้ในรูปที่ 10

กล่าวคือ

- สิ่งแรกที่เราต้องทำคือ การหา Use Case ทั้งหมดในระบบ
- ถัดมาคือ การสร้าง Sequence Diagram ที่สอดคล้องกับ Use Case ตัวหนึ่งๆ
- สิ่งสุดท้ายคือการสร้าง Class Diagram Class Diagram จะประกอบไปด้วย Class ทั้งหมดในระบบ ส่วน Class หนึ่ง Class ในไดอะแกรมนั้นโดยปกติจะถูกสร้างขึ้นมาจากหลาย Use Case และหลาย Sequence Diagram โดยที่ทั้ง Use Case และ Sequence

Diagram ดังกล่าวนั้นจะต้องมีความเกี่ยวข้องกับ Class ตัวนั้น

รูปที่ 10 กระบวนการทำงานของ UML



หนังสืออ้างอิง

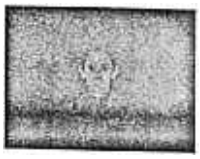
[1] *VB6 UML Design and Development*, Jake Sturm ,Wrox Press ltd. , 1999

[2] *Instant UML*, Pierre-Alain Muller , Wrox press Ltd , 1997

[3] *Applying Use Cases* , Geri Scheider , Jason P.Winters , Asddison-Wesley ,March 1999

ประวัติผู้เขียน

ดร.บรรจง หะรังษี เกิดที่จังหวัดลพบุรีเมื่อวันที่ 19 กรกฎาคม พ.ศ. 2508 จบการศึกษาระดับปริญญาตรี สาขาวิทยาการคอมพิวเตอร์ เกียรตินิยมอันดับสอง จากมหาวิทยาลัยหอการค้าไทยเมื่อ พ.ศ 2532 จบการศึกษาระดับปริญญาโท และปริญญาเอก สาขา Computer Science and Engineering จาก University of New South Wales, Australia เมื่อปี พ.ศ. 2537 และพ.ศ 2541 ตามลำดับ ได้เริ่มทำงานที่ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติในตำแหน่งนักวิจัย เมื่อเดือนมิถุนายน พ.ศ 2541 ปัจจุบันรับผิดชอบในงานด้านการวางแผน วิเคราะห์ และออกแบบโครงสร้างพื้นฐานของ NECTEC's MIS



นางญาณวรรณ สินธุภิญโญ จบการศึกษาระดับปริญญาโทจากจุฬาลงกรณ์มหาวิทยาลัย สาขาเทคโนโลยีสารสนเทศ เมื่อปี พศ.2538 เริ่มทำงานกับศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติในตำแหน่งนักวิจัย โดยรับผิดชอบงานในด้านการวิเคราะห์ ออกแบบและพัฒนาระบบสารสนเทศด้านต่างๆขององค์กร ในระหว่างเดือนตุลาคม พศ.2540 – พฤศจิกายน 2540 ได้รับทุนจาก the Japan International Co-operation Agency (JICA)เพื่อไปอบรมในด้าน Computer Software Technology ที่ประเทศสิงค์โปร์ ปัจจุบันปฏิบัติงานในงานโครงการเครือข่าย NECTECNet