การเพิ่มเติมความสามารถ UML สำหรับการพัฒนาซอฟต์แวร์แบบ AOP

นายศิวดล เสถียรพัฒนากูล

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์
มหาวิทยาลัยเทคโนโลยีสุรนารี
ปีการศึกษา 2549

EXTENSIBLE UML FOR AOP

Siwadol Sateanpattanakul

A Thesis Submitted in Partial Fulfillment of the Requirements for the

Degree of Master of Engineering in Computer Engineering

Suranaree University of Technology

Academic Year 2006

การเพิ่มเติมความสามารถ UML สำหรับการพัฒนาซอฟต์แวร์แบบ AOP

มหาวิทยาลัยเทคโนโลยีสุรนารี อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

คณะกรรมการสอบวิทยานิพนธ์

_____

(รศ. ดร.กิตติศักดิ์ เกิดประสพ)
ประธานกรรมการ

_____

(ผศ. ดร.พิชโยทัย มหัทธนาภิวัฒน์)
กรรมการ (อาจารย์ที่ปรึกษาวิทยานิพนธ์)

_____

(รศ. ดร.นิตยา เกิดประสพ)
กรรมการ

_____

(ผศ. ดร.คะชา ชาญศิลป์)
กรรมการ

_____

(รศ. ดร.เสาวณีย์ รัตนพานี)
รองอธิการบดีฝ่ายวิชาการ

_____

(รศ. น.อ. ดร.วรพจน์ ขำพิศ)
คณบดีสำนักวิชาวิศวกรรมศาสตร์

ศิวดล เสถียรพัฒนากูล : การเพิ่มเติมความสามารถของ UML สำหรับการพัฒนาซอฟต์แวร์แบบ AOP (EXTENSIBLE UML FOR AOP) อาจารย์ที่ปรึกษา : ผู้ช่วยศาสตราจารย์ ดร. พิชโยทัย มหัทธนาภิวัฒน์, 115 หน้า

ภาษาออกแบบเชิงโมเดล (Unified Modeling Language : UML) เป็นภาษาที่ได้รับการยอมรับในการออกแบบโครงสร้างสำหรับการสร้าง และพัฒนาซอฟต์แวร์ โดยที่โครงสร้างของ UML นั้นสามารถแบ่งรูปแบบสำหรับการออกแบบได้เป็น 2 ส่วนใหญ่ ๆ ด้วยกันซึ่งประกอบด้วย Structure และ Behavior โดยส่วนมาก UML มักจะนำมาใช้ในการออกแบบสำหรับการพัฒนาซอฟต์แวร์เชิงวัตถุ (Object-Oriented Programming : OOP) ซึ่งเป็นวิธีการที่ได้รับความนิยมมากในการพัฒนาซอฟต์แวร์ยุคปัจจุบันโดยความสามารถของ OOP นั้นยังมีข้อจำกัดตรงที่ไม่สามารถจัดการโค้ดที่มีการเรียกใช้ซ้ำ ๆ กันได้ซึ่งอาจทำให้เกิดผลกระทบต่อการพัฒนาซอฟต์แวร์ได้ ดังนั้นจึงได้เกิดการคิดค้นวิธีการพัฒนาซอฟต์แวร์อีกอย่างหนึ่งเพิ่มขึ้นมาเรียกว่า Aspect-Oriented Programming (AOP) โดยที่วิธีการพัฒนาซอฟต์แวร์แบบ AOP นี้สามารถจัดการปัญหาที่เกิดจากการพัฒนาซอฟต์แวร์แบบ OOP ได้เป็นอย่างดี โดยที่การพัฒนาซอฟต์แวร์แบบ AOP จะช่วยจัดการลดความซ้ำซ้อนของโค้ดซึ่งจะทำให้ปริมาณโค้ดลดลงกว่าเดิม ทำให้สามารถจัดการโค้ดได้ง่ายขึ้น และมีประสิทธิภาพมากยิ่งขึ้น และถ้านำความสามารถของ AOP มาใช้ในการออกแบบซอฟต์แวร์ร่วมกับ UML น่าจะช่วยให้ UML นั้นมีความยืดหยุ่น และมีประสิทธิภาพมากยิ่งขึ้น โดยทุกวันนี้ได้มีการนำเสนอมาตรฐานที่ให้ทั้ง UML และ AOP ทำงานร่วมกันเป็นจำนวนมากแต่ยังขาดในส่วนการนำมาใช้งานได้จริง โดยงานวิจัยนี้จะนำเสนอมาตรฐานของภาษาออกแบบเชิงแบบจำลองที่สามารถสนับสนุนการทำงานร่วมกันของเทคนิคการเขียนโปรแกรมเชิงลักษณะ และการเขียนโปรแกรมเชิงวัตถุ และได้นำมาตรฐานที่ได้ออกแบบมาสร้างเครื่องมือซึ่งอยู่ในรูปแบบของซอฟต์แวร์ ซึ่งสามารถทำให้การนำ AOP และ UML มาประยุกต์ใช้งานร่วมกันได้ง่าย และมีประสิทธิภาพมากขึ้น

สาขาวิชาวิศวกรรมคอมพิวเตอร์ ลายมือชื่อนักศึกษา_____

ปีการศึกษา 2549 ลายมือชื่ออาจารย์ที่ปรึกษา_____

ลายมือชื่ออาจารย์ที่ปรึกษาร่วม_____

SIWADOL SATEANPATTANAKUL : EXTENSIBLE UML FOR AOP.

THESIS ADVISOR : ASST. PROF. PICHAYOTAI MAHATTHANAPIWAT, Ph. D., 115 PP.

UNIFIED MODELING LANGUAGE, ASPECT-ORIENTED PARADIGM, OBJECT-ORIENTED PARADIGM

Unified Modeling Language (UML) is a language that is used extensively to design system structure for software development. UML structure can be divided into 2 types: structure diagram and behavior diagram. Mostly, UML is used to design Object-Oriented paradigm that is one of the most widely used techniques for software development. However, there are still some limitations in object-oriented software development due to code tangling problem. Aspect-oriented paradigm is the way to solve this problem because it has the solution to manage cross-cutting problem that reduces the problem of code tangling and make software structure clean and configurable. Thus we used AOP to design software process with UML. UML is implemented for flexibility and performance. Although there are introduction of many standards designed for UML and AOP, it's realization is lacking. This thesis introduced UML standard to support AOP and OOP design software system in system analysis and design phase and developed tool to support this standard.

School of Computer Engineering

Academic Year 2006

Student's Signature_____

Advisor's Signature_____

Co-advisor's Signature_____

กิตติกรรมประกาศ

วิทยานิพนธ์นี้สำเร็จลุล่วงด้วยดี ผู้วิจัยขอกราบขอบพระคุณ บุคคล และกลุ่มบุคคลต่าง ๆ ที่ได้กรุณาให้คำปรึกษา แนะนำ ช่วยเหลือ อย่างดียิ่ง ทั้งในด้านวิชาการ และ ด้านการดำเนินงานวิจัยดังนี้

- ผู้ช่วยศาสตราจารย์ ดร. พิชโยทัย มหัทธนาภิวัฒน์, อาจารย์ที่ปรึกษาวิทยานิพนธ์
- ผู้ช่วยศาสตราจารย์ ดร. คะชา ชาญศิลป์, อาจารย์ที่ปรึกษาวิทยานิพนธ์ร่วม
- รองศาสตราจารย์ ดร. กิตติศักดิ์ เกิดประสพ หัวหน้าสาขาวิชา รองศาสตราจารย์ ดร. นิตยา เกิดประสพ อาจารย์ประจำสาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี
- คุณสุพิชชา โชคไพบูลย์ อาจารย์ ชาญวิทย์ แก้วกสิ คุณพิชญา แก้วกสิ คุณฉัตรชัย วิริยาภิรมย์ ที่ให้ความช่วยเหลือด้วยดีมาตลอด
- คณะนักพัฒนาโครงการ Amateras

ท้ายที่สุด ขอกราบขอบพระคุณบิดา มารดา และญาติ ๆ ทุกคนที่ให้การอุปการะเลี้ยงดู อบรม และส่งเสริมการศึกษาเป็นอย่างดีมาตลอดในอดีต ทำให้ผู้วิจัยมีความรู้ ความสามารถ มีจิตใจที่เข้มแข็งและช่วยเหลือตัวเองได้จนประสบความสำเร็จในชีวิตตลอดมา

ศิวดล เสถียรพัฒนากูล

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญรูป

รูปที่ **หน้า**

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มาของปัญหาการวิจัย

การพัฒนาด้านซอฟต์แวร์ในยุคปัจจุบัน การพัฒนาอุตสาหกรรมด้านซอฟต์แวร์ต้องอาศัยกระบวนการและเทคนิคมากมายมาจัดการปัญหาที่อาจเกิดขึ้นมา วิศวกรรมซอฟต์แวร์ (Software Engineering) มีบทบาทสำคัญในการรับมือกับปัญหาเหล่านี้ ยิ่งทุกวันนี้การพัฒนาด้านคอมพิวเตอร์มีการเจริญเติบโตอย่างรวดเร็ว และต่อเนื่องยิ่งทำให้ต้องมีการพัฒนากระบวนการขึ้นมารองรับและสามารถตอบคำถามที่เกิดขึ้นจากปัญหาต่าง ๆ ดังนั้นจึงทำให้เทคนิคและกระบวนการต่าง ๆ ทางด้านวิศวกรรมซอฟต์แวร์ได้มีการพัฒนาไปอย่างรวดเร็วตามไปด้วย ทฤษฎีต่าง ๆ ที่เกี่ยวข้องกับกระบวนการพัฒนาซอฟต์แวร์ (Software Development Process) และวัฏจักรของการพัฒนาซอฟต์แวร์ (Software Development Life Cycle, SDLC) ถูกสร้างขึ้นมาอย่างมากมาย การนำกระบวนการเหล่านี้มาใช้นั้นจะขึ้นอยู่กับปัญหาที่ได้รับมาจากความต้องการต่าง ๆ จากผู้ใช้ และขีดจำกัดต่าง ๆ ทางด้านทรัพยากรในขณะนั้น โดยกระบวนเหล่านี้เป็นแนวทางที่มุ่งเน้นให้ได้ซอฟต์แวร์ที่เหมาะสม มีคุณภาพ และสามารถตอบสนองความต้องการของผู้ใช้งานได้อย่างดี โดยอยู่ภายใต้ข้อจัดกัดต่าง ๆ

ในกระบวนการและวัฏจักรของการพัฒนาซอฟต์แวร์นั้นประกอบด้วยขั้นตอนต่าง ๆ หลายส่วนด้วยกันโดยจะแยกกันเป็นเฟส (Phase) ซึ่งแต่ละวัฏจักรของการพัฒนาซอฟต์แวร์นั้นอาจจะมีจำนวนเฟสมากน้อยไม่เท่ากัน โดยขึ้นอยู่กับแต่ละแนวการทำงานของกระบวนการนั้น แต่ว่าเฟสหลัก ๆ ส่วนมากของวัฏจักรเหล่านั้นมักจะเหมือนกันทำให้สามารถทำการปรับกระบวนการในการทำงานของเฟสหลักแต่ละเฟสได้ โดยที่จะไม่ไปกระทบกับการทำงานของเฟสอื่น หรือถ้าจะมีผลกระทบต่อเฟสอื่นเกิดขึ้นก็จะมีผลกระทบน้อย ซึ่งกระบวนการหรือวัฏจักรของการพัฒนาซอฟต์แวร์นั้นมีอยู่หลายรูปแบบ เช่น

1) วัฏจักรของการพัฒนาซอฟต์แวร์ในรูปแบบน้ำตก (Waterfall Model)
2) วัฏจักรของการพัฒนาซอฟต์แวร์ในรูปแบบก้นหอย (Spiral Model)
3) วัฏจักรของการพัฒนาซอฟต์แวร์ในรูปแบบวน (Iterative and Incremental Model)

โดยที่เฟสหลัก ๆ ของกระบวนการนั้นจะประกอบไปด้วยเฟสต่อไปนี้

1) การหาความต้องการของระบบ (Requirement)
2) การวิเคราะห์ และออกแบบระบบ (System Analysis and Design)
3) การสร้าง และพัฒนาระบบ (Implementation)
4) การทดสอบ และตรวจสอบระบบ (Verification and Testing)

จะเห็นได้ว่าเฟสหลักเหล่านี้มักจะอยู่ปะปนไปกับทุกกระบวนการ แม้ว่าเทคโนโลยีทางด้านการพัฒนาซอฟต์แวร์จะมีการเจริญขึ้นอย่างมากก็ไม่ได้ทำให้กระบวนการ และขั้นตอนที่กล่าวมาข้างต้นนั้นล้าสมัยหรือไม่สามารถจะนำมาใช้กับการแก้ปัญหาที่เกิดขึ้นกับการพัฒนาซอฟต์แวร์สมัยใหม่ แต่การที่จะนำกระบวนการนี้มาใช้ให้เกิดประโยชน์และสามารถทำงานได้อย่างมีประสิทธิภาพสูงสุดนั้นจะต้องรู้จักนำเทคโนโลยี หรือเทคนิคของการพัฒนาซอฟต์แวร์ใหม่ ๆ มาปรับใช้ให้เข้ากับกระบวนการเหล่านั้น

การพัฒนาซอฟต์แวร์เชิงวัตถุ (Object-oriented Paradigm) นั้นถือว่าเป็นหนึ่งในวิธีการที่ได้รับความนิยมเป็นอย่างมาก ทั้งยังเป็นวิธีการที่สามารถตอบสนองต่อความต้องการในการพัฒนาของผู้พัฒนา โปรแกรมเป็นอย่างดี ด้วยความสามารถที่หลากหลาย และสามารถจัดการและตอบปัญหาของการพัฒนาซอฟต์แวร์ในปัจจุบันเป็นอย่างดี ด้วยเหตุนี้เองทำให้ช่วงหลังได้เกิดการพัฒนากระบวนการพัฒนาซอฟต์แวร์แบบใหม่ ๆ มาตรฐาน และภาษาที่ทำงานร่วมกับการพัฒนาโปรแกรมเชิงวัตถุทั้งทางตรง และทางอ้อมเกิดขึ้นอย่างรวดเร็วและมีจำนวนมาก ภาษาออกแบบเชิงแบบจำลอง (Unified Modeling Language, UML) ถือว่าเป็นตัวอย่างอย่างหนึ่งที่มีการพัฒนารูปแบบออกมาเพื่อมารองรับการทำงานของการพัฒนาโปรแกรมเชิงวัตถุนี้ ภาษาออกแบบเชิงแบบจำลองนี้ถูกสร้างมาเพื่ออธิบายการทำงานของระบบที่จะพัฒนาด้วยการพัฒนาโปรแกรมเชิงวัตถุในรูปของสัญลักษณ์ และแผนภาพต่าง ๆ

แม้ว่าการพัฒนาซอฟต์แวร์เชิงวัตถุนี้จะได้รับการยอมรับว่าสามารถแก้ปัญหาของการพัฒนาซอฟต์แวร์ต่าง ๆ ได้อย่างเหมาะสม แต่ความสามารถของการพัฒนาซอฟต์แวร์เชิงวัตถุเองนั้นไม่สามารถอธิบายลักษณะของปัญหาจากการพัฒนาซอฟต์แวร์บางอย่างได้ โดยคุณสมบัติของการพัฒนาซอฟต์แวร์เชิงวัตถุ นี้มีอยู่ 3 อย่างประกอบด้วย

1) การสืบทอด (Inheritance)
2) การถูกห่อหุ้มอย่างมิดชิด (Encapsulate)
3) การพ้องรูป (Polymorphism)

ด้วยคุณสมบัติเหล่านี้ทำให้การเขียนโปรแกรมเชิงวัตถุมีความสามารถในการทำ Vertical Grouping หรือการดึงคุณสมบัติที่ซ้ำ ๆ กันไปไว้ในคลาสแม่ได้ แต่ปัญหาที่เกิดขึ้นต่อจากนี้เป็นปัญหาที่ไม่

สามารถจัดการด้วยการพัฒนาซอฟต์แวร์เชิงวัตถุเองได้ และเป็นปัญหาที่เกิดขึ้นจากการทำ Vertical Grouping เองด้วย ปัญหานั้นคือการที่วิธีการพัฒนาซอฟต์แวร์เชิงวัตถุไม่สามารถจัดการการเรียกใช้ข้อมูล หรือโค้ดของการเขียนโปรแกรมที่เกิดการเรียกใช้ซ้ำ ๆ กันได้ และด้วยเหตุนี้เองทำให้เกิดการพัฒนาโปรแกรมเชิงลักษณะ (Aspect-oriented Paradigm) ขึ้นซึ่งการเขียนโปรแกรมเชิงลักษณะเป็นการแยกประเภทของโค้ดและข้อมูลที่เหมือนกันออกมาอยู่ในกลุ่มเดียวกัน โดยอาจจะอยู่ในรูปแบบของคลาสแม่เดียวกัน หรือไม่ก็จะอยู่ในรูปของอินเตอร์เฟสเดียวกัน โดยที่การเขียนโปรแกรมเชิงลักษณะนี้จะช่วยแก้ไขปัญหาที่ไม่สามารถทำ Vertical Grouping ซึ่งการเขียนโปรแกรมเชิงลักษณะนี้จะมาช่วยในการทำงาน และวิธีการเขียนโปรแกรมเชิงวัตถุให้สมบูรณ์ขึ้น ปัญหาที่เกิดขึ้นกับการพัฒนาซอฟต์แวร์เชิงวัตถุนี้ทำให้ภาษาออกแบบเชิงแบบจำลอง นั้นเกิดข้อผิดพลาด และทำงานได้ไม่ครอบคลุมกับลักษณะของปัญหา ดังนั้นเพื่อเป็นการเพิ่มความสามารถของภาษาออกแบบเชิงแบบจำลองให้เหมาะสม สามารถทำงานร่วมกับการเขียนโปรแกรมเชิงลักษณะ จึงจะต้องเพิ่มความสามารถของการเขียนโปรแกรมเชิงลักษณะลงไปในภาษาออกแบบเชิงแบบจำลอง โดยการเพิ่มความสามารถลงไปในภาษาออกแบบเชิงแบบจำลองนั้นจะช่วยให้ตัวของภาษาออกแบบเชิงแบบจำลองมีความยืดหยุ่นและมีประสิทธิภาพมากยิ่งขึ้น

ดังนั้นงานวิจัยนี้มุ่งเน้นเพื่อศึกษา และพัฒนาภาษาพร้อมทั้งเครื่องมือที่สนับสนุนการทำงาน และการพัฒนาซอฟต์แวร์เชิงวัตถุ พร้อมทั้งการพัฒนาซอฟต์แวร์เชิงลักษณะด้วย โดยเครื่องมือนี้จะเป็นตัวสนับสนุนการทำงานของการพัฒนาซอฟต์แวร์ทั้ง 2 ชนิดโดยการให้ภาษาออกแบบเชิงแบบจำลองที่เพิ่มความสามารถของการพัฒนาซอฟต์แวร์เชิงลักษณะลงไปแล้วเป็นตัวกลาง คอยผสมผสานการทำงานในการวิเคราะห์ และออกแบบของวิธีการพัฒนาซอฟต์แวร์ทั้ง 2 ชนิด โดยที่เครื่องมือนี้จะเป็นตัวคอยสนับสนุนการทำงานอยู่ในเฟสของการวิเคราะห์ และออกแบบระบบ ซึ่งผลที่ได้จากการออกแบบของเครื่องมือนี้ในเฟสของการวิเคราะห์ และออกแบบระบบจะเป็นสัญลักษณ์ต่าง ๆ ซึ่งหลังจากนั้นสามารถนำผลที่ได้ไปใช้ในเฟสของการสร้าง และพัฒนาระบบต่อไปได้ โดยที่สามารถนำสัญลักษณ์นั้นเข้าไปการทำงานอีกขั้นตอนที่เครื่องมือเตรียมไว้ให้ หรือจะนำไปออกแบบเองก็ได้ โดยในส่วนที่เครื่องมือเตรียมเอาไว้ให้นั้นจะเป็นการนำเอาสัญลักษณ์ และโครงสร้างที่ได้ถูกออกแบบมาจากเฟสของการการวิเคราะห์และออกแบบระบบ มาสร้างให้อยู่ในรูปแบบของโค้ดซึ่งอยู่ในเฟสของการสร้างและพัฒนาระบบ โดยโค้ดที่ได้หลังจากการสร้างนั้นจะอยู่ในรูปแบบของโครงสร้างอย่างหยาบ ๆ เท่านั้นยังไม่สามารถทำงานได้อย่างสมบูรณ์ ดังนั้นหลังจากที่ได้โครงสร้างของโค้ดมาแล้วจึงต้องนำเอาโครงสร้างที่ได้มาทำการเพิ่มการทำงานตามความต้องการของซอฟต์แวร์นั้น ๆ เพื่อทำให้ซอฟต์แวร์นั้นสามารถทำงานได้อย่างถูกต้องต่อไป

1.2 วัตถุประสงค์การวิจัย

1.2.1 เพื่อปรับปรุง และพัฒนาภาษาออกแบบเชิงแบบจำลอง (UML) ให้สามารถทำงานได้อย่างสอดคล้องกับเทคนิค และวิธีการพัฒนาซอฟต์แวร์ในปัจจุบัน

1.2.2 เพื่อพัฒนากระบวนการการวิเคราะห์ และออกแบบระบบซอฟต์แวร์โดยอาศัยเทคนิค, แนวคิดและวิธีการออกแบบซอฟต์แวร์เชิงโมเดลที่ได้เพิ่มคุณสมบัติ และความสามารถของการพัฒนาซอฟต์แวร์เชิงลักษณะ (AOP)

1.2.3 เพื่อพัฒนาเครื่องมือที่สามารถช่วยในกระบวนการการวิเคราะห์ และออกแบบระบบซอฟต์แวร์ซึ่งเครื่องมือสามารถสร้างความสัมพันธ์ระดับคลาส โดยความสัมพันธ์นี้จะประกอบไปด้วยความสัมพันธ์ของคลาสจากการเขียนโปรแกรมเชิงวัตถุ (OOP) กับเขียนโปรแกรมเชิงวัตถุเอง และโปรแกรมเชิงวัตถุกับการเขียนโปรแกรมเชิงลักษณะด้วย

1.2.4 เพื่อให้สามารถนำผลลัพธ์จากกระบวนการวิเคราะห์ และออกแบบระบบซอฟต์แวร์ไปใช้ในกระบวนการการสร้าง และพัฒนาระบบได้อย่างสอดคล้อง และต่อเนื่อง

1.2.5 เพื่อให้ผู้ที่วิเคราะห์ และออกแบบระบบสามารถดำเนินการ ออกแบบด้วยสัญลักษณ์ทาง UML ได้อย่างสะดวก และรวดเร็ว

1.3 ข้อตกลงเบื้องต้น

1.3.1 เครื่องมือนี้จะถูกติดตั้งในรูปแบบของ Plug-in ของ Eclipse เท่านั้น

1.3.2 เครื่องมือนี้จะทำการสร้างไฟล์ที่เป็นเอกซ์เอ็มแอล (Extensible Markup Language, XML) จากสัญลักษณ์ที่ได้ทำการออกแบบ และ XML นั้นสามารถนำไปสร้างเป็นโค้ดจริงที่ใช้ในกระบวนการการสร้าง และพัฒนาระบบต่อไป ซึ่งผู้ที่วิเคราะห์ และออกแบบระบบสามารถสร้างไฟล์ XML เองและสามารถนำมาสร้างเป็นโค้ดเองได้โดยไม่ต้องใช้เครื่องมือที่สร้างสัญลักษณ์ แต่จะต้องสร้างไฟล์ XML ให้อยู่ในรูปแบบที่กำหนดเท่านั้น

1.3.3 เครื่องมือนี้จะทำงานได้อย่างถูกต้อง และแม่นยำในเฉพาะการออกแบบที่เป็นแผนภาพคลาส (Class Diagram) เท่านั้น

1.3.4 เครื่องมือนี้จะทำการสร้างโครงสร้างของโค้ดที่จะนำไปใช้ในกระบวนการการสร้างและพัฒนาระบบเท่านั้น โดยโครงสร้างที่ได้นั้นต้องนำไปพัฒนาต่อจึงจะสามารถทำงานได้อย่างสมบูรณ์

1.3.5 เครื่องมือนี้จะรองรับไฟล์ที่มีนามสกุล .acld ของแผนภาพสัญลักษณ์ และ .xml ของการนำไปสร้างเป็นโค้ดเท่านั้น

1.4 ขอบเขตของการวิจัย

1.4.1 เครื่องมือที่พัฒนานี้สามารถรองรับมาตรฐานของภาษาออกแบบเชิงแบบจำลองเฉพาะที่เป็นแผนภาพของคลาส (Class Diagram) เท่านั้น

1.4.2 เครื่องมือที่พัฒนานี้จะสามารถรองรับความสามารถของการเขียนโปรแกรมเชิงลักษณะ แบบโครงสร้างเท่านั้น

1.4.3 เครื่องมือนี้จะทำการสร้าง XML โดย XML ที่ได้สามารถนำไปใช้ในกระบวนการสร้างโค้ดต่อไป โดยถ้าจะเขียน XML เองโดยไม่ผ่านเครื่องมือ และจะนำมาใช้ในกระบวนการโค้ด XML ที่สร้างนั้นต้องอยู่ในรูปแบบที่ทางเครื่องมือนี้ได้วางไว้จึงจะสามารถทำงานได้อย่างถูกต้อง

1.4.4 เครื่องมือนี้จะรองรับความสัมพันธ์ของภาษาออกแบบเชิงแบบจำลองเฉพาะที่เป็นแบบถ่ายทอดคุณสมบัติหรือพฤติกรรม (Generalization) และความสัมพันธ์ที่เชื่อมโยง หรือเกี่ยวข้องกันของคลาส (Association) เท่านั้น

1.4.5 เครื่องมือนี้จะสร้างไฟล์โครงสร้างของโค้ดที่เป็นสกุลแบบ .java และเป็นแบบ .aj สำหรับไฟล์ที่เป็นการเขียนโปรแกรมเชิงลักษณะ

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1.5.1 ช่วยให้กระบวนการวิเคราะห์ และออกแบบระบบซอฟต์แวร์ สามารถทำได้สะดวกรวดเร็ว และมีประสิทธิภาพมากขึ้น

1.5.2 ช่วยให้สามารถนำเทคนิคการเขียนโปรแกรมที่มีประสิทธิภาพมาลดข้อเสีย และปรับปรุงเทคนิคการเขียนโปรแกรมแบบเดิม

1.5.3 ได้เครื่องมือที่ช่วยในการออกแบบระบบซอฟต์แวร์ ซึ่งจะช่วยให้เวลาในการออกแบบลดน้อยลง และเพิ่มความถูกต้องของการออกแบบให้เพิ่มขึ้น

1.5.4 ช่วยลดภาระของผู้ออกแบบ รวมไปถึงผู้ที่ทำการพัฒนาระบบ เนื่องจากเครื่องมือจะทำการสร้างโครงสร้างของโค้ดให้โดยอัตโนมัติ

1.5.5 ช่วยให้การนำไปใช้ในกระบวนการสร้างและพัฒนาซอฟต์แวร์เป็นไปอย่างสอดคล้อง และต่อเนื่องจากขั้นตอนการวิเคราะห์และออกแบบระบบ

บทที่ 2
ปริทัศน์วรรณกรรมและงานวิจัยที่เกี่ยวข้อง

ในบทนี้จะเป็นการนำเสนอวรรณกรรมและงานวิจัยที่เกี่ยวข้องกับงานวิจัยนี้ โดยในหัวข้อที่ 2.1 จะเป็นการกล่าวถึงกระบวนการในการพัฒนาซอฟต์แวร์ รวมทั้งเทคนิคและการวิเคราะห์ และออกแบบระบบ แบบต่าง ๆ และในหัวข้อที่ 2.2 จะเป็นการกล่าวถึงภาษา และมาตรฐานที่ใช้ในการวิเคราะห์ และออกแบบระบบซอฟต์แวร์ ในหัวข้อที่ 2.3 จะกล่าวถึงวิธีการเขียนและพัฒนาโปรแกรมเชิงวัตถุ (OOP) ซึ่งถือว่าเป็นวิธีการพัฒนามีประสิทธิภาพ และมีการสร้างภาษาที่ไว้สำหรับสนับสนุนการพัฒนาโปรแกรมเชิงวัตถุ และรวมไปถึงเครื่องมือต่าง ๆ ออกมาอย่างมากมาย และในหัวข้อที่ 2.4 จะกล่าวถึงวิธีการเขียนและพัฒนาโปรแกรมเชิงลักษณะ (AOP) เนื่องจากงานการวิเคราะห์และออกแบบระบบซอฟต์แวร์ มีจุดประสงค์เพื่อสามารถวิเคราะห์และ ออกแบบระบบให้สามารถอธิบายการทำงานของระบบ ซึ่งจะรวมไปถึงเทคนิคและวิธีการพัฒนาซอฟต์แวร์ที่สามารถทำงานได้อย่างเหมาะสมกับระบบนั้น ๆ ด้วย

2.1 กระบวนการและวัฏจักรในการพัฒนาซอฟต์แวร์ (Software Development Processes and Software Development Life Cycle, SDLC)

กระบวนการพัฒนาซอฟต์แวร์ถูกสร้างขึ้นจากองค์กรด้านพัฒนาซอฟต์แวร์ และหน่วยงานของกระทรวงกลาโหมของสหรัฐ เป็นหน่วยงานหนึ่งที่ทำการคิดค้นและพัฒนากระบวนการนี้ด้วย โดยใช้ ISO 12207 เป็นมาตรฐานสำหรับการพัฒนา และตรวจสอบการพัฒนาซอฟต์แวร์

ISO 15504 หรือที่เรียกว่า Software Process Improvement Capability Determination (SPICE) เป็นโครงสร้างสำหรับการประเมินผลของขั้นตอนการพัฒนาซอฟต์แวร์ โดยมาตรฐานของ SPICE นั้นถูกนำไปใช้อย่างแพร่หลายในกระบวนการพัฒนาซอฟต์แวร์หลาย ๆ ชนิด โดยมาตรฐานของ SPICE นี้มีเป้าหมายเพื่อสร้างรูปแบบที่ชัดเจนของแบบจำลองสำหรับการเปรียบเทียบกันของกระบวนการ

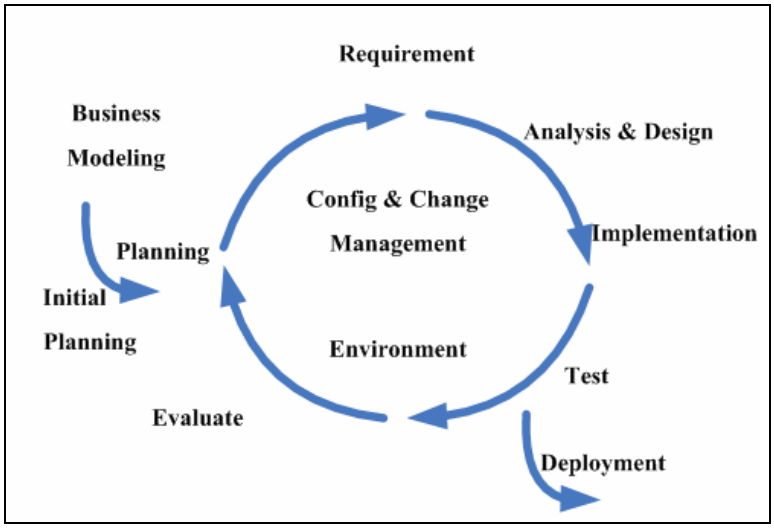
ทั้งนี้กระบวนการพัฒนาซอฟต์แวร์นั้นถูกสร้างและออกแบบมาเพื่อรับมือรวมทั้งสามารถจัดการขั้นตอนการทำงานของการพัฒนาซอฟต์แวร์ให้มีการทำงานอย่างเป็นระเบียบมาตรฐานเพื่อให้ได้มาซึ่งซอฟต์แวร์ที่มีคุณภาพ

2.1.1 ขั้นตอนและกิจกรรมต่าง ๆ ที่อยู่ในกระบวนการพัฒนาซอฟต์แวร์

ขั้นตอนและกิจกรรมต่าง ๆ ของกระบวนการพัฒนาซอฟต์แวร์ถูกสร้างขึ้นมาเพื่อจัดการกับปัญหาต่าง ๆ ที่เกิดขึ้นจากการทำงาน โดยจะสามารถแบ่งเป็นขั้นตอนได้ดังนี้

1) ขั้นตอนการหาความต้องการ (Requirement)
2) ขั้นตอนการออกแบบระบบ (Architecture or Analysis and Designs)
3) ขั้นตอนการทำการสร้างและพัฒนา (Implementation)
4) ขั้นตอนการทำการทดสอบ (Testing)
5) ขั้นตอนการทำการส่งมอบ (Deployment)

ซึ่งขั้นตอนที่กล่าวมาทั้งหมดนั้นในแต่ละกิจกรรมจะประกอบไปด้วยกระบวนการย่อย ๆ ที่สามารถรองรับและจัดการกับปัญหาที่เกิดขึ้น



รูปที่ 2.1 ขั้นตอนของกระบวนการพัฒนาซอฟต์แวร์

2.1.2 แบบจำลองของกระบวนการพัฒนาซอฟต์แวร์ (Process Models)

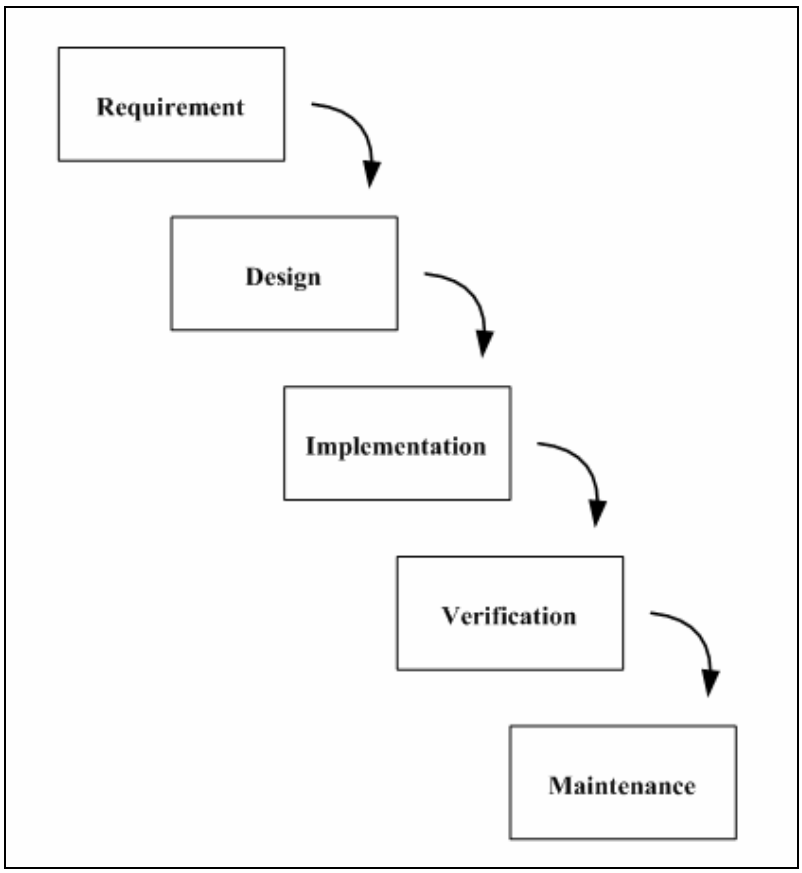
ในช่วงเวลาหลายปีที่ผ่านมาได้มีการพยายามที่จะพัฒนากระบวนการ โดยการทำซ้ำ ๆ เพื่อปรับปรุงกระบวนการ และวิธีการ เพื่อให้ได้ซอฟต์แวร์ที่มีทั้งปริมาณ และคุณภาพที่ดีโดยบางครั้งอาจใช้วิธีการที่ดูเหมือนจะไม่มีกฎเกณฑ์มาใช้ในการทำซอฟต์แวร์ หรือใช้เทคนิคในการบริหารจัดการซอฟต์แวร์มาใช้ในการทำซอฟต์แวร์ ซึ่งถ้าหากไม่มีการนำเอาการบริหารจัดการซอฟต์แวร์มาใช้โครงการพัฒนาซอฟต์แวร์มีความเป็นไปได้มากที่จะไม่สามารถส่งงานได้ทันตามเวลาและไม่สามารถคุมงานให้งบประมาณที่มีอยู่ได้ ซึ่งแบบจำลองของกระบวนการพัฒนาซอฟต์แวร์นั้น

ได้นำเอาข้อมูลที่ได้จากการพัฒนาซอฟต์แวร์มาวิเคราะห์ แล้วได้ทำการอธิบายขั้นตอนของการพัฒนาเพื่อที่จะได้สามารถบริหารจัดการซอฟต์แวร์ได้อย่างมีประสิทธิภาพ

แบบจำลองของกระบวนการพัฒนาซอฟต์แวร์นั้นได้ถูกคิดค้นขึ้นมาอย่างมากมายเพื่อที่จะตอบสนองต่อปัญหาที่เกิดขึ้นจากการพัฒนาซอฟต์แวร์ โดยที่แบบจำลองของกระบวนการพัฒนาซอฟต์แวร์แต่ละแบบนั้นมีข้อดี และข้อด้อยแตกต่างกันออกไปตามแต่ละยุคสมัยของเทคโนโลยี ซึ่งแบบจำลองของกระบวนการพัฒนาซอฟต์แวร์นั้นสามารถแบ่งได้ดังนี้

1) แบบจำลองของกระบวนการพัฒนาซอฟต์แวร์แบบน้ำตก (Waterfall Model)

แบบจำลองกระบวนการพัฒนาซอฟต์แวร์แบบน้ำตก เป็นการนำเอาความสามารถของการบริหารจัดการมาช่วยในการพัฒนาซอฟต์แวร์ โดยที่จะแบ่งช่วงของการพัฒนาซอฟต์แวร์ออกเป็นช่วง ๆ ซึ่งคล้ายกับกระบวนการพัฒนาซอฟต์แวร์ทั่ว ๆ ไปแต่จะแต่งต่างตรงวิธีคิดของกระบวนการซึ่งกระบวนการนี้คือจะต้องทำขั้นตอนบนให้เสร็จเสียก่อน ถึงจะไปทำงานในขั้นตอนต่อไปได้



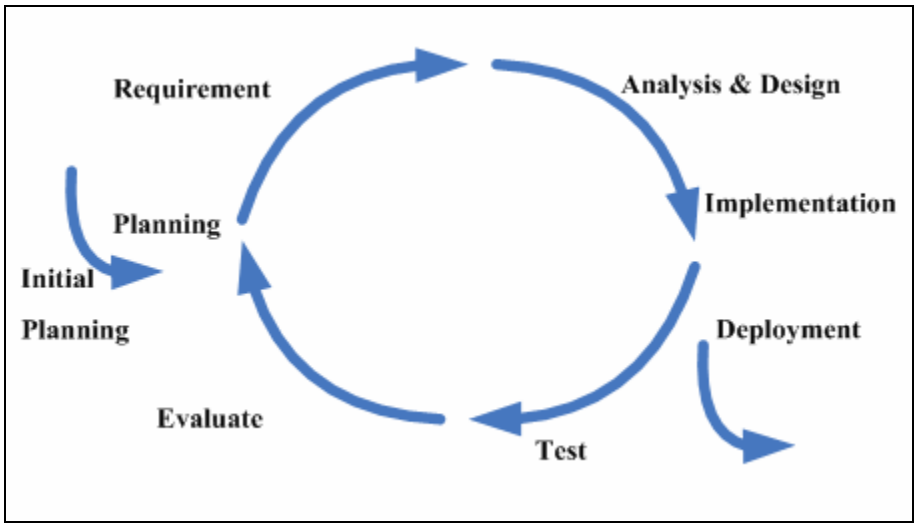
รูปที่ 2.2 ขั้นตอนของกระบวนการพัฒนาซอฟต์แวร์แบบน้ำตก

ถึงแม้การทำงานของกระบวนการพัฒนาซอฟต์แวร์แบบน้ำตกจะเป็นแบบเป็นขั้นตอนไหลจากด้านบนลงสู่ด้านล่างก็จริง แต่ก็สามารถย้อนกลับไปปรับปรุงขั้นตอนก่อนหน้าได้ตามลำดับ และท้ายที่สุดเมื่อดำเนินมาถึงกิจกรรมที่เรียกว่า Verification และ Validation ผู้ใช้ก็จะได้เห็น และใช้งานระบบใหม่ที่ได้พัฒนาขึ้น

2) แบบจำลองของกระบวนการพัฒนาซอฟต์แวร์แบบวนซ้ำ และ เพิ่มขึ้น

(Iterative and Incremental Development Model)

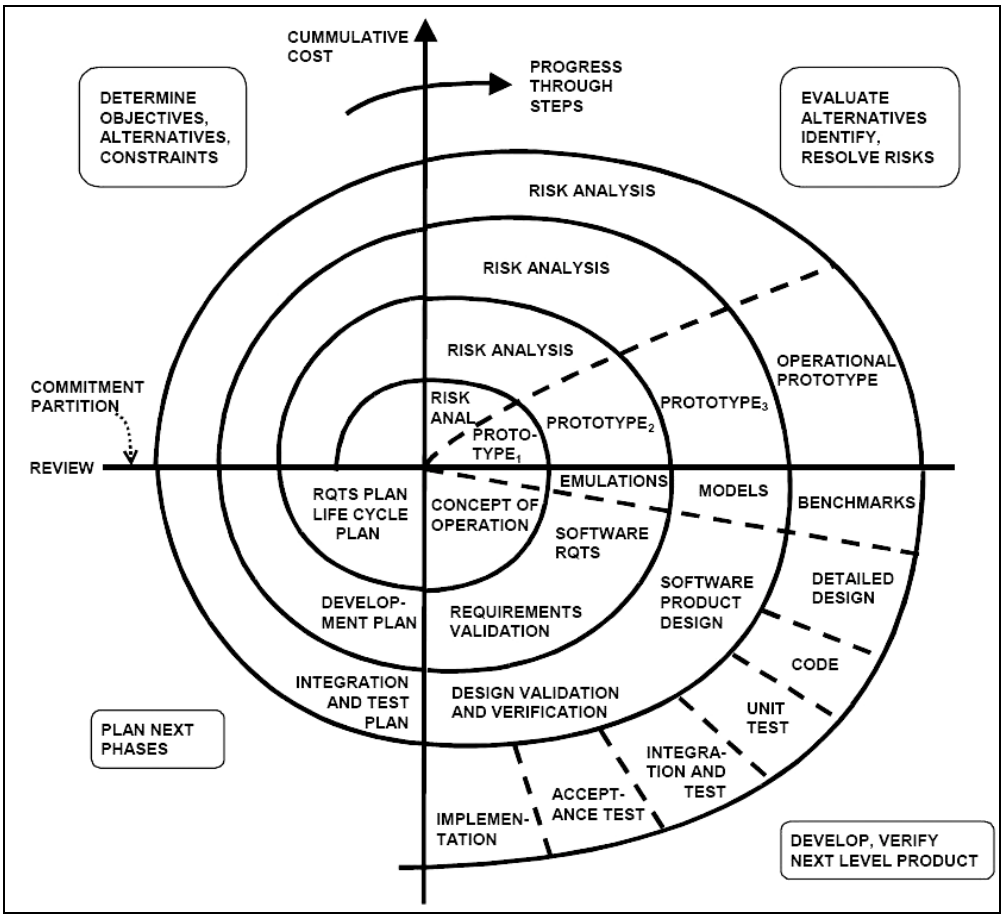
แบบจำลองของกระบวนการพัฒนาซอฟต์แวร์แบบวนซ้ำ และ เพิ่มขึ้น เป็นแบบจำลองที่สร้างมาเพื่อแก้ไขจุดบกพร่องของการทำงานของแบบจำลองน้ำตก โดยแบบจำลองน้ำตกนั้นเป็นวิธีการทำงานแบบดั้งเดิมจะต้องทำงานให้เสร็จในรอบเดียว นับว่าเป็นความเสี่ยงอย่างยิ่งต่อความล้มเหลวของโครงการ เนื่องจากการทำงานในยุคปัจจุบันมักมีการเปลี่ยนแปลง Environment ต่าง ๆ เกิดขึ้นเสมอ เช่น Requirement Change, Technology Change เป็นต้น



รูปที่ 2.3 ขั้นตอนของกระบวนการพัฒนาซอฟต์แวร์แบบวนซ้ำ

ดังนั้นจะเห็นว่าในสภาพการทำงานจริง ๆ มักจะมีการทำงานแบบแก้ไขใหม่โดยจำทำการวนซ้ำเรื่อย ๆ ซึ่งก็คือ Iteration นั่นเอง (การวนทำซ้ำใหม่เมื่อพบข้อผิดพลาดหรือบกพร่อง) วิธีการนี้จึงมีความสำคัญมาก นอกจากนี้ยังพบว่าการทำงานโดยการค่อย ๆ เพิ่มงานเข้าไปในงานเดิมที่ทำเสร็จแล้วเรื่อย ๆ จนหมดทั้งโครงการก็จะเป็นการลดความเสี่ยงงานได้ด้วย เนื่องจากทำงานเสร็จเป็นช่วง ๆ โดยทำการแบ่งซอยงานใหญ่ ๆ ออกเป็นงานย่อย ๆ และเมื่อเกิดข้อผิดพลาดขึ้นจะมีผลกระทบต่องานเพียงส่วนย่อยที่กำลังดำเนินการอยู่เท่านั้น ไม่ใช่กระทบหมดทั้งโครงการ

วิธีการทำงานเช่นนี้เรียกการทำงานแบบ Incremental จะเห็นว่าการทำงานแบบนี้เป็นการพยายามลดความเสี่ยงต่อการล้มเหลวของโครงการ และพยายามแก้ไขข้อผิดพลาดด้วย



รูปที่ 2.4 ขั้นตอนของกระบวนการพัฒนาซอฟต์แวร์แบบก้นหอย

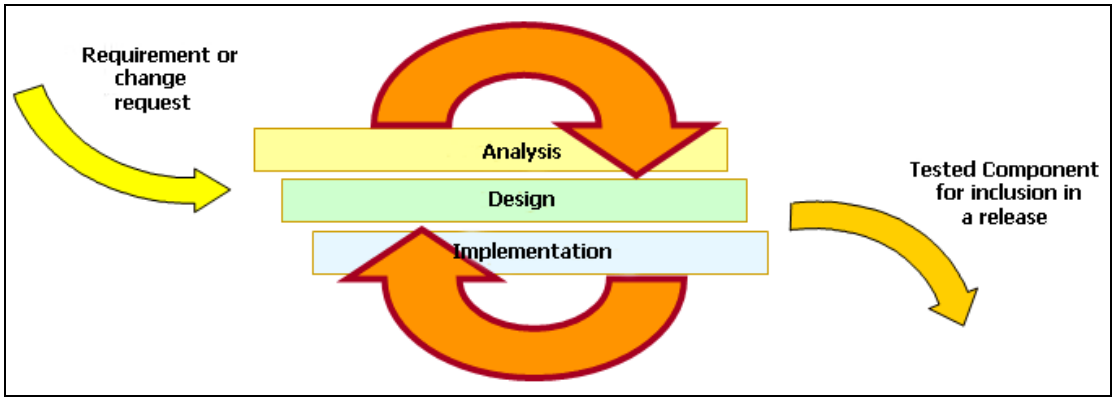
3) แบบจำลองของกระบวนการพัฒนาซอฟต์แวร์แบบก้นหอย (Spiral Model)

Boehm (1988) ได้นำเสนอแบบจำลองกระบวนการพัฒนาซอฟต์แวร์แบบก้นหอย ซึ่งการทำงานของแบบจำลองนี้จะทำการวนซ้ำเพื่อพัฒนาซอฟแวร์ในรูปแบบของช่วงเวลาสั้น ๆ เรียกว่า Iteration โดยใช้การผสมผสานแนวทางของการพัฒนาซอฟต์แวร์แบบน้ำตก และแนวทางการพัฒนาต้นแบบ (Prototype) ยกตัวอย่างเช่น โครงการพัฒนาซอฟต์แวร์ขนาดใหญ่นั้นจะถูกแบ่งออกเป็นส่วนย่อย ๆ และภายในแต่ละส่วนจะเป็นการพัฒนาแบบเพิ่มขึ้น (Increment) จากต้นแบบเป็นหลัก ทำให้สามารถตรวจสอบความต้องการ (Requirement) และทำการทดสอบ (Testing)

ระบบได้บ่อย ๆ ช่วยลดความเสี่ยงที่ทำให้โครงการล้มเหลวเพราะจะทำให้เห็นความก้าวหน้าของงานได้อย่างชัดเจนชัดเจน ซึ่งทำให้สามารถควบคุมงบประมาณและระยะเวลาได้อย่างดี

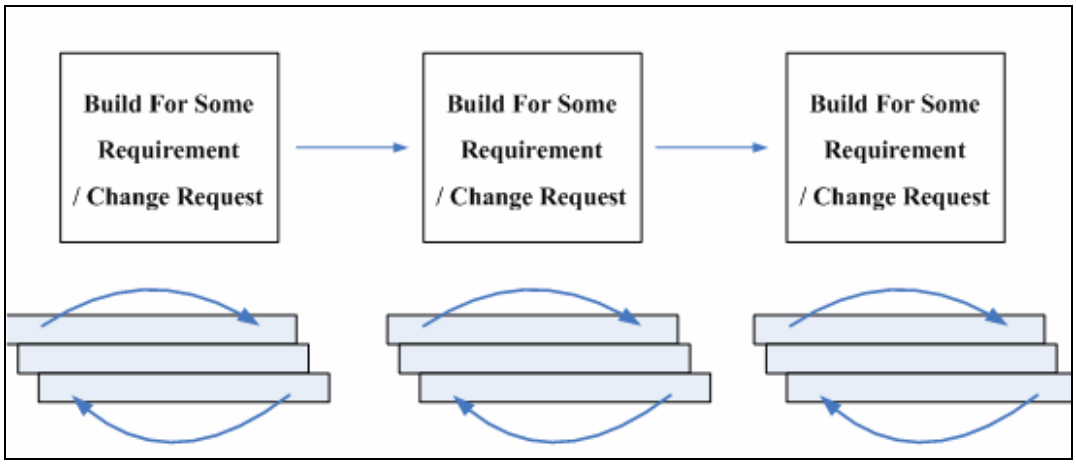
4) แบบจำลองของกระบวนการพัฒนาซอฟต์แวร์แบบกระฉับกระเฉง (Agile Model)

การพัฒนาซอฟต์แวร์ด้วยแบบจำลองนี่โดยมากจะเป็นการพยายามเพื่อลดความเสี่ยงของการพัฒนาซอฟต์แวร์ โดยพัฒนาซอฟแวร์ในรูปแบบของช่วงเวลาสั้น ๆ เรียกว่าการแบ่งช่วง โดยอาจใช้เวลาตั้งแต่ 2 ถึง 6 สัปดาห์ ซึ่งแต่ละช่วงของเวลาจะเปรียบเสมือนโครงการของมันเอง ซึ่งจะประกอบไปด้วยกิจกรรมต่าง ๆ ที่จำเป็นต่อการทำงาน ซึ่งอาจจะประกอบไปด้วยงานทางด้านการวางแผน การหาความต้องการ การวิเคราะห์และออกแบบระบบ ทำการสร้างและทดสอบระบบ จนสุดท้ายถึงการทำเอกสาร



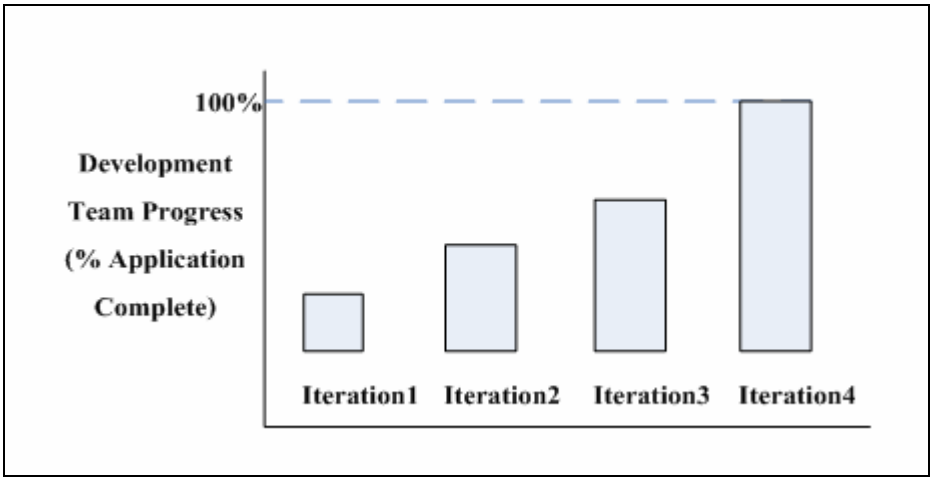
รูปที่ 2.5 ขั้นตอนของกระบวนการพัฒนาซอฟต์แวร์แบบกระฉับกระเฉง

การพัฒนาด้วยแบบจำลองนี้พยายามที่จะออกผลการผลิตออกมาทุก ๆ ช่วงของงานแต่ละช่วง และลักษณะการทำงานของบุคลากรในการพัฒนานั้นจะนิยมการติดต่อสื่อสารแบบสม่ำเสมอ ตลอดเวลา คือการคุยกันแทนที่จะเขียนเอกสารโดยทีมงานจะทำงานร่วมกันภายใต้ Bullpen ซึ่งประกอบไปด้วยคนที่เกี่ยวข้องกับการทำซอฟแวร์ให้เสร็จ อย่างน้อยได้แก่ โปรแกรมเมอร์ และ ลูกค้า หรืออาจจะรวมถึง ผู้ทำการทดสอบระบบ และผู้จัดการด้วย



รูปที่ 2.6 ขั้นตอนของกระบวนการพัฒนาซอฟต์แวร์แบบกระฉับกระเฉง

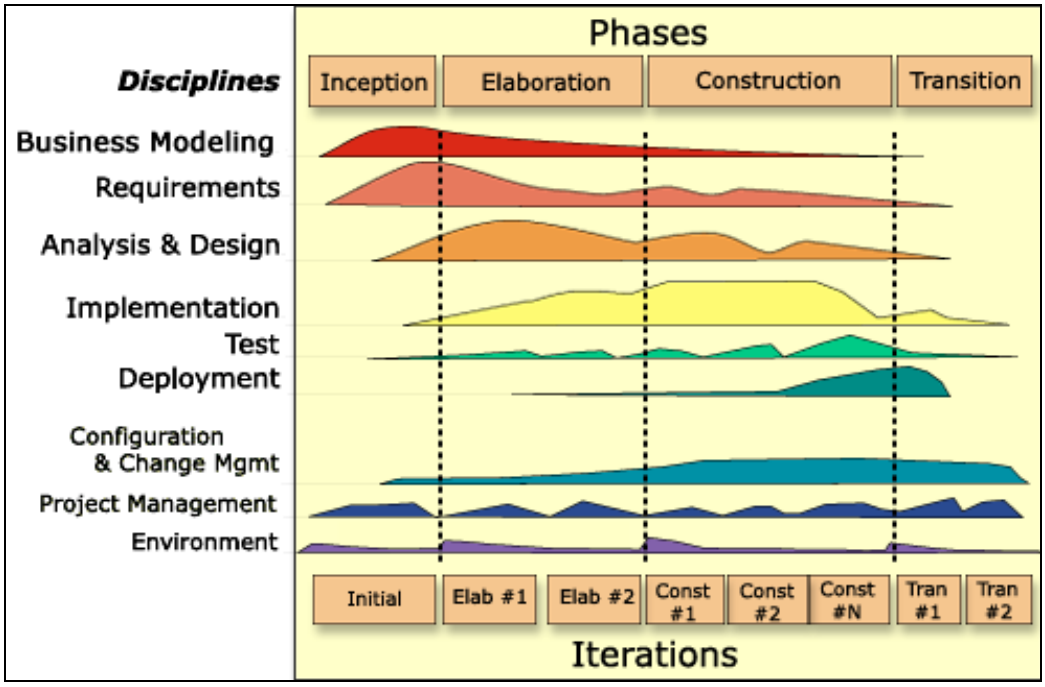
กระบวนการพัฒนาซอฟต์แวร์แบบกระฉับกระเฉง จะเป็นการแบ่งช่วงของการพัฒนาซอฟต์แวร์ โดยทำให้เสร็จทีละช่วง ๆ ไปเมื่อมีการพัฒนาจนหมดรอบของแต่ละรอบก็จะทำให้ซอฟต์แวร์ที่พัฒนานั้นค่อย ๆ สมบูรณ์ขึ้นเรื่อย ๆ โดยทำให้ได้เอกสารใหม่ในแต่ละช่วง และจะทำให้มีความสามารถของงานใหม่ ๆ ออกมา



รูปที่ 2.7 ความก้าวหน้าของกระบวนการพัฒนาซอฟต์แวร์แบบกระฉับกระเฉง

5) แบบจำลองของกระบวนการพัฒนาซอฟต์แวร์แบบอาร์ยูพี (Rational Unified Process, RUP)

อาร์ยูพี (Rational Software Corporation, 2002) เป็นโครงสร้างของการพัฒนาซอฟต์แวร์แบบวนซ้ำอีกอันหนึ่ง โดยจะทำการกำหนดถึงงาน หรือ สิ่งที่จะต้องดำเนินการเพื่อพัฒนาซอฟต์แวร์ โดยจะแบ่งระยะเวลา หรือช่วง (Phase) การทำงานออกเป็นช่วง ๆ โดยมีเป้าหมายเพื่อให้ได้ซอฟต์แวร์ที่มีคุณภาพสูง ตรงตามความต้องการของผู้ใช้ ตามเวลา และงบประมาณที่กำหนดไว้ตามแผนงานของโครงการ



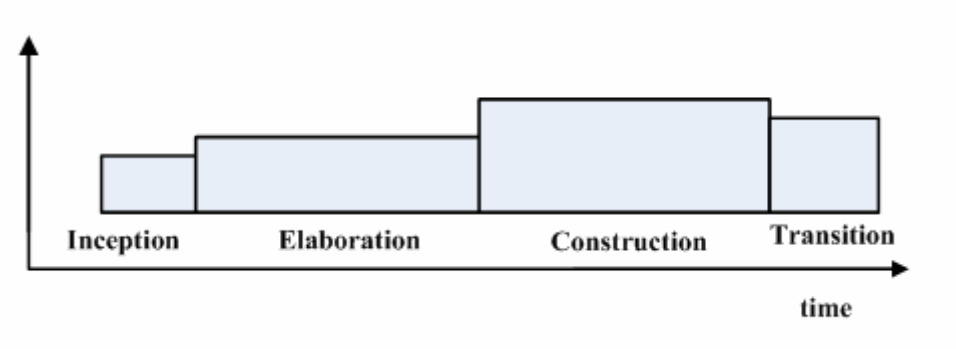
รูปที่ 2.8 ขั้นตอนของกระบวนการพัฒนาซอฟต์แวร์แบบอาร์ยูพี

RUP ประกอบด้วยมุมมอง 2 มิติ คือ

1) มุมมองแบบแนวนอน ซึ่งมุมมองทางด้านนี้จะแสดงให้เห็นถึงเวลาและวงจรการพัฒนาในแต่ละช่วงของโครงการ และการเปลี่ยนแปลงของกระบวนการทำงาน โดยจะนำเสนอออกมาในรูปแบบของช่วงการทำงานโดยจะทำการแบ่งการทำงานออกเป็นส่วนย่อย ๆ และระยะการทำงาน (Milestone)
2) มุมมองในแนวตั้ง จะแสดงให้เห็นถึงกลุ่มกิจกรรมที่เกิดขึ้น ซึ่งจะถูกแบ่งกลุ่มออกตามวิธีการหรือลักษณะของการทำงานของกิจกรรมนั้น ๆ จะ

อธิบายให้เห็นถึงส่วนประกอบของกระบวนการ กิจกรรมที่เกิดขึ้น ขั้นตอนของงาน บทบาท และ อื่น ๆ เป็นต้น

โดยถ้านำเอาช่วงเวลาของการพัฒนามา และทำการแบ่งเป็นส่วน ๆ เพื่อจะกำหนดว่าจะใช้ระยะเวลาต่อช่วงเป็น และงานที่ควรทำแต่ละช่วงนั้นใช้เวลามากน้อยแค่ไหน จะสามารถแบ่งออกมาได้เป็นดังภาพนี้



รูปที่ 2.9 ความก้าวหน้าของกระบวนการพัฒนาซอฟต์แวร์แบบอาร์ยูพี

ในวงจรการพัฒนาช่วงเริ่มต้น (Inception) และช่วงกำหนดรายละเอียด (Elaboration) จะถูกกำหนดให้มีสัดส่วนของการทำงานที่น้อยกว่าการทำงานในช่วงการสร้าง (Construction) อย่างไรก็ตาม เครื่องมือต่าง ๆ ในปัจจุบันที่มีประสิทธิภาพสูงจะช่วยให้การทำงานในช่วงการสร้างลดน้อยลง และสามารถน้อยกว่าการทำงานในช่วง เริ่มต้น และ รายละเอียด รวมกันก็ได้

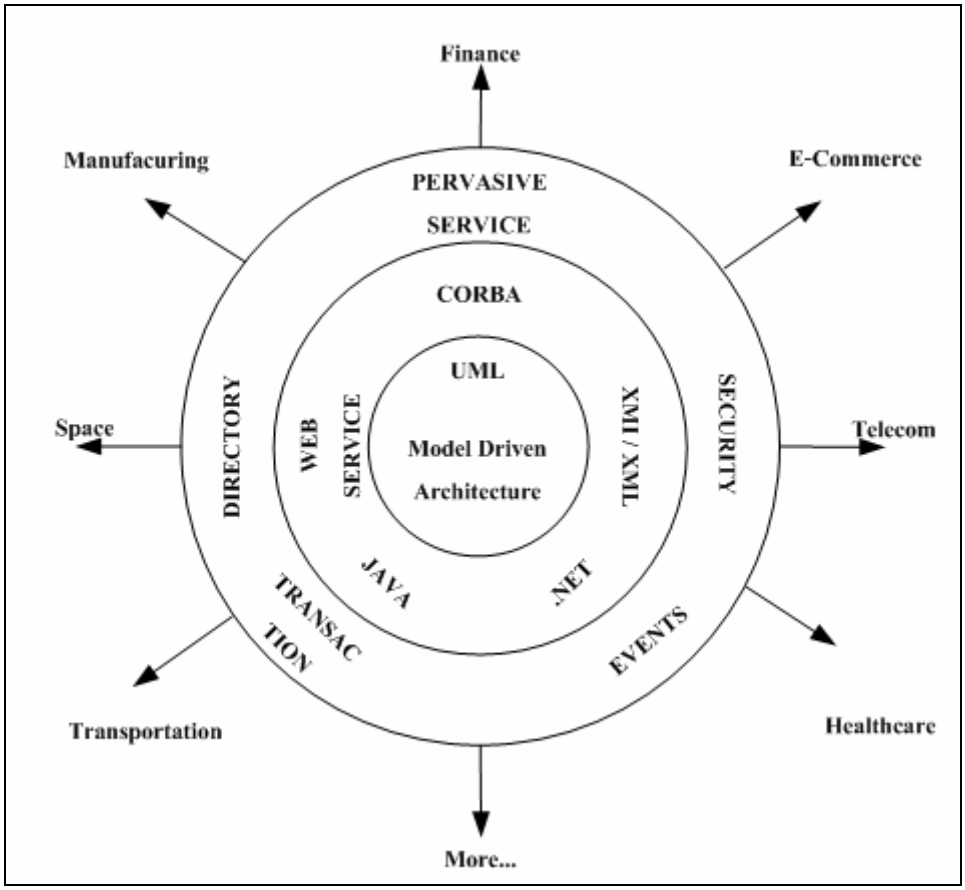
2.2 สถาปัตยกรรมของซอฟต์แวร์ (Software Architectures)

Best (1995) สถาปัตยกรรมของซอฟต์แวร์นั้นมุ่งเน้นไปที่ส่วนประกอบต่าง ๆ ของซอฟต์แวร์ที่ต้องใช้ร่วมกัน และปฏิกิริยาที่กระทำร่วมกันต่อส่วนประกอบต่าง ๆ เหล่านี้ ซึ่งหมายความว่าสถาปัตยกรรมของการพัฒนาซอฟต์แวร์นั้น เป็นมาตรฐานที่มีอยู่ตามส่วนประกอบต่าง ๆ ในการพัฒนาซอฟต์แวร์ ซึ่งมาตรฐานเหล่านี้ควรจะเป็นมาตรฐานที่ยืดหยุ่นมากพอที่จะสามารถสนับสนุนในการพัฒนาซอฟต์แวร์อย่างมีประสิทธิภาพ

2.2.1 สถาปัตยกรรมการขับเคลื่อนด้วยแบบจำลอง (Model-Driven Architectures)

Object Management Group (OMG, 2001) ได้ทำการสนับสนุน และพัฒนาข้อกำหนดในลักษณะเฉพาะ เพื่อปรับปรุงวิธีปฏิบัติของการพัฒนาซอฟต์แวร์ขนาดใหญ่ และการ

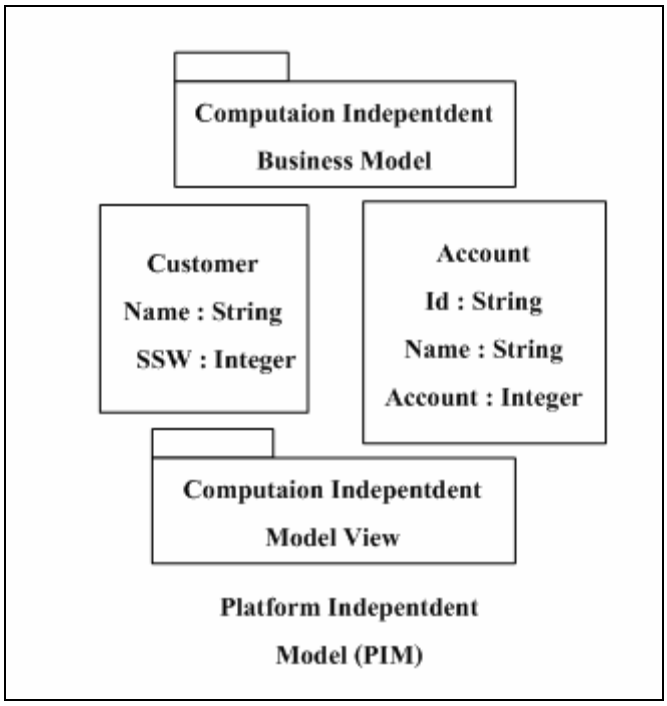
นำมาใช้ โดยทำการสร้างโครงสร้างของแนวคิด ซึ่งจะทำการแยกแนวคิดการทำงานออกมาจากเทคโนโลยี ซึ่งจะทำให้เกิดความยืดหยุ่นอย่างมากเมื่อเกิดการปรับตัวเทคโนโลยีและสถาปัตยกรรม โดยโครงสร้างของแนวคิดฐานที่ช่วยในการทำงานจริงนี้ ทาง OMG ได้เรียกว่าสถาปัตยกรรมการขับเคลื่อนด้วยแบบจำลอง (Model-Driven Architectures, MDA)



รูปที่ 2.10 แนวคิดสถาปัตยกรรมการขับเคลื่อนด้วยแบบจำลอง

การแยกขั้นตอนการออกแบบซอฟต์แวร์ออกมาจากสถาปัตยกรรม และเทคโนโลยีที่ใช้ในการพัฒนานั้นถือว่าเป็นหนึ่งในความตั้งใจของการพัฒนาด้วยแนวคิดสถาปัตยกรรมการขับเคลื่อนด้วยแบบจำลอง ซึ่งการออกแบบนี้มุ่งเน้นไปการทำงานของซอฟต์แวร์ต้องถูกต้องตามความต้องการที่ถูกระบุมา ในขณะที่สถาปัตยกรรมและเทคโนโลยีซึ่งจะถูกจัดเป็นโครงสร้างพื้นฐาน นั้นจะไม่ถูกระบุเป็นการทำงานของซอฟต์แวร์ที่ถูกกำหนดมาแต่ต้น แต่จะเป็นการทำงานของซอฟต์แวร์เชิงประสิทธิภาพแทน โดยสถาปัตยกรรมการขับเคลื่อนด้วยแบบจำลองจะเรียกการทำงานของส่วนนี้ว่าเป็น แบบจำลองซึ่งเป็นอิสระต่อสถาปัตยกรรมและเทคโนโลยี (Platform

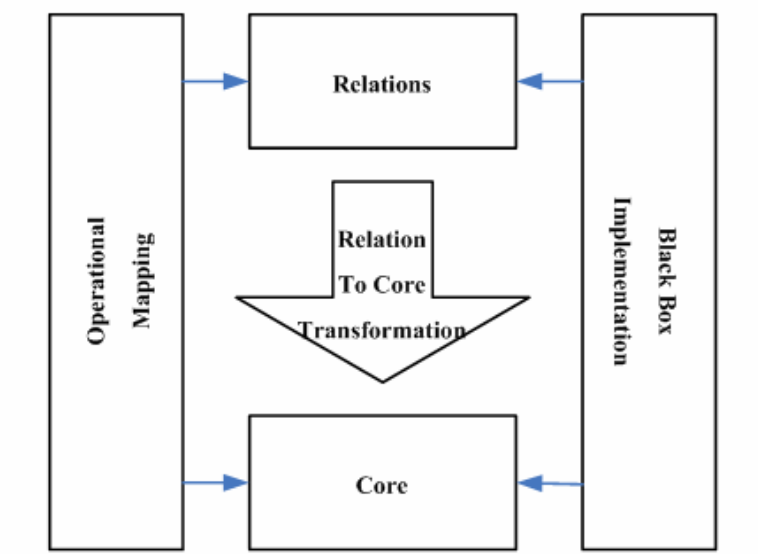
Independent Model, PIM) โดยแสดงให้เห็นถึงการออกแบบระบบการทำงานของซอฟต์แวร์ที่สามารถนำไปทำงานได้เหมือนเดิม ถึงแม้ว่าเกิดการเปลี่ยนแปลงทางด้านเทคโนโลยี และสถาปัตยกรรมของซอฟต์แวร์



รูปที่ 2.11 แบบจำลองซึ่งเป็นอิสระต่อสถาปัตยกรรมและเทคโนโลยี

สถาปัตยกรรมการขับเคลื่อนด้วยแบบจำลองมีแนวคิดในการเปลี่ยนแปลงรูปแบบของแบบจำลอง ซึ่งจะถูกกำหนดให้สร้างขึ้นตามมาตรฐานที่ OMG กำหนดเรียกว่า การสืบค้น การเรียกดู และการเปลี่ยนรูป (Queries/Views/Transformations, QVT) โดยจะคอยทำหน้าที่ปรับเปลี่ยนแบบจำลองให้ไปอยู่ในรูปแบบที่กำหนด หรือนำรูปแบบที่ถูกกำหนดมาทำให้อยู่ในรูปของแบบจำลอง

สถาปัตยกรรมการขับเคลื่อนด้วยแบบจำลองนั้นถูกนำไปใช้ในการสร้างมาตรฐานและภาษาต่าง ๆ มากมายรวมไปถึงภาษาออกแบบเชิงแบบจำลอง (Unified Modeling Language, UML), กลุ่มของวัตถุอำนวยความสะดวกพื้นฐาน (Meta-Object Facility, MOF) และ XML Metadata interchange (XMI) เป็นต้น



รูปที่ 2.12 สถาปัตยกรรม QVT

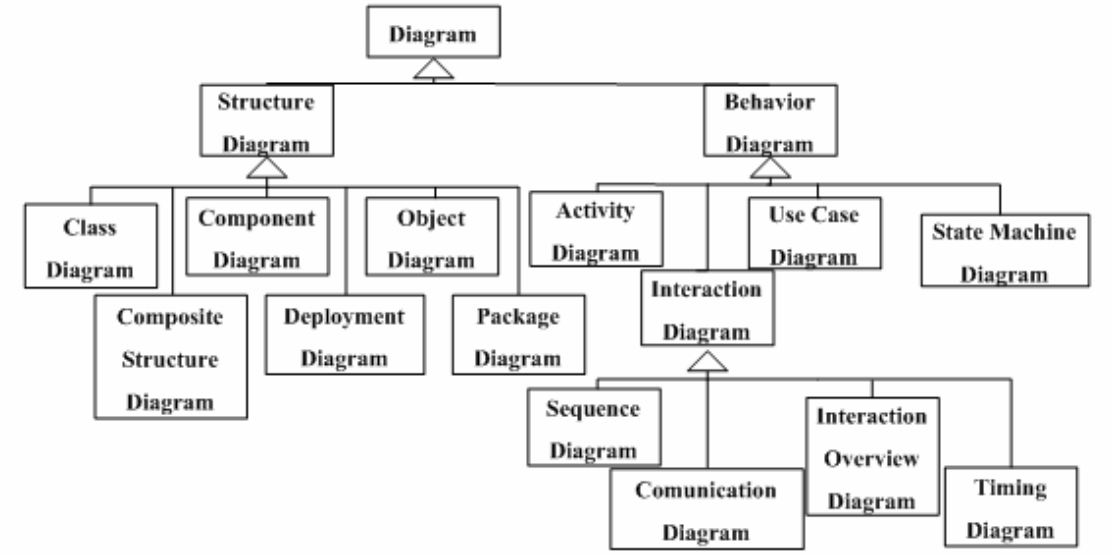
ภาษาออกแบบเชิงแบบจำลอง (Unified Modeling Language, UML)

เป็นภาษามาตรฐานที่ไม่ถูกจำกัดสิทธิทางลิขสิทธิ์ซึ่งออกแบบมาเพื่อใช้สำหรับการออกแบบเชิงวัตถุ ภาษาออกแบบเชิงแบบจำลองเป็นภาษาเพื่อใช้อธิบายพฤติกรรมและการทำงานด้วยแบบจำลองในลักษณะต่าง ๆ โดยมีทั้งมาตรฐานของกราฟิกที่เป็นสัญลักษณ์ ซึ่งนำไปใช้สำหรับสร้างแบบจำลองในลักษณะนามธรรม (Abstract Model) ของระบบ ซึ่งหมายถึงแบบจำลองของภาษาออกแบบเชิงแบบจำลอง และ ณ ตอนนี้ภาษาออกแบบเชิงแบบจำลองมีชนิดของแผนภาพทั้งหมด 13 ชนิด ซึ่งสามารถแยกประเภทออกเป็นกลุ่มของการทำงานได้เป็น 3 ประเภทได้แก่

1) แผนภาพประเภทโครงสร้าง (Structure Diagrams) : แผนภาพประเภทนี้จะเน้นลักษณะการทำงานส่วนแบบจำลองการทำงานของระบบ โดยจะประกอบไปด้วยแผนภาพดังนี้

- Class Diagram
- Component Diagram
- Composite Structure Diagram
- Deployment Diagram
- Object Diagram
- Package Diagram

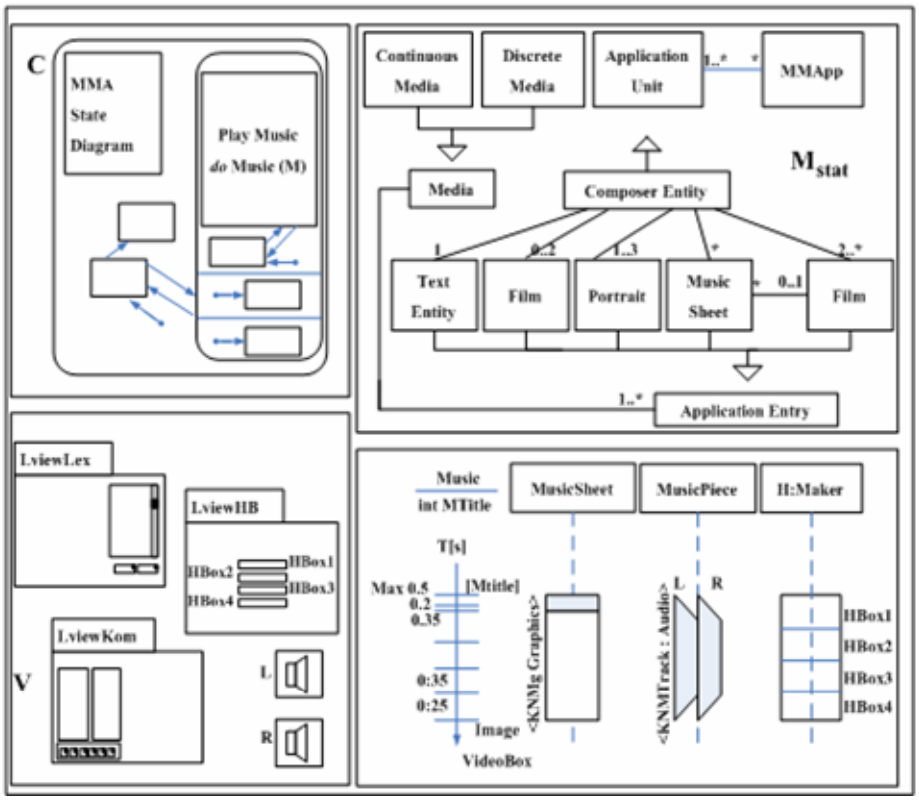
2) แผนภาพประเภทพฤติกรรม (Behavior Diagrams) : แผนภาพประเภทนี้จะเน้นลักษณะการทำงานแบบที่จะเกิดขึ้นเมื่อเกิดการทำงานของระบบโดยจะประกอบไปด้วยแผนภาพดังนี้
   - Activity Diagram
   - State Machine Diagram
   - Use Case Diagram
3) แผนภาพประเภทปฏิกิริยา (Interaction Diagrams) : แผนภาพประเภทนี้เป็นส่วนหนึ่ง (Subset) ของแผนภาพประเภทพฤติกรรม ซึ่งจะเน้นลักษณะการทำงานแบบการไหลของกลุ่มของข้อมูลเมื่อเกิดการทำงาน
   - Communication Diagram
   - Interaction Overview Diagram (UML 2.0)
   - Sequence Diagram
   - UML Timing Diagram (UML 2.0)



รูปที่ 2.13 โครงสร้างของภาษาออกแบบเชิงแบบจำลอง

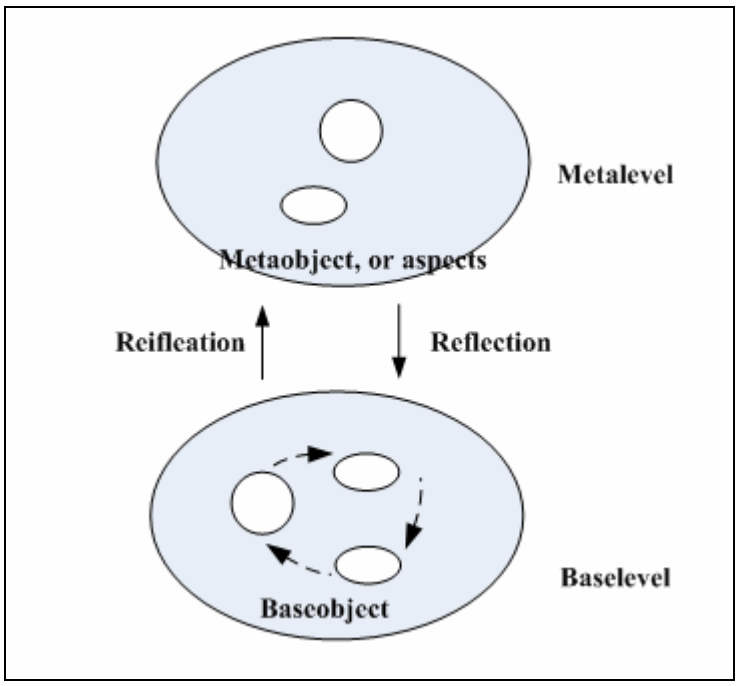
UML นั้นสามารถถูกเพิ่มเติมความสามารถเพิ่มเข้าไปได้ เพื่อทำให้เข้ากับการทำงานของภาษาของการเขียนโปรแกรมคอมไพเลอร์ (Compiler) และระบบของโปรแกรมที่ทำงาน (Runtime Environment)

Sauer and Engels (1999) ได้นำเสนอแนวคิดและงานวิจัยที่จะทำให้ภาษาออกแบบเชิงแบบจำลองนั้นสามารถทำการออกแบบให้สนับสนุนการทำงานกับงานทางด้านมัลติมีเดียโดยลักษณะของงาน คือ การทำงานของซอฟต์แวร์แบบมัลติมีเดียนั้น จะทำงานแตกต่างจากการทำงานของซอฟต์แวร์ทั่ว ๆ ไป เพราะการทำงานที่เกิดขึ้นของซอฟต์แวร์แบบสื่อประสมนั้นต้องทำงานกับการเกิดการเปลี่ยนแปลงพฤติกรรมของงานประเภทสื่อประสมอยู่ตลอดเวลา ซึ่งอาจจะเกิดการทำงานที่ผิดปกติไปจากเดิมได้อยู่เสมอ ๆ ดังนั้นได้จึงได้นำแนวคิดการเขียนโปรแกรมแบบแบบจำลอง-มุมมอง-ตัวควบคุม (Model-View-Controller, MVC) มาใช้ร่วมกับการพัฒนาและออกแบบงานประเภทสื่อประสม และได้ทำการเรียกเทคนิคนี้ว่า ภาษาเสมือนจริงสำหรับการออกแบบจำลองเชิงวัตถุของงานสื่อประสม (A Visual Language for the Object-Oriented Modeling of Multimedia Applications, OMMMA-L) โดยทำการพัฒนามาจากภาษาออกแบบเชิงแบบจำลองซึ่งได้ทำการออกแบบ และได้ผลดังรูปที่ 2.14



รูปที่ 2.14 คุณสมบัติรวมของ OMMMA-L

Suzuki and Yamamoto (1999) ได้นำเสนอแนวคิดในการแสดงการทำงานของสถาปัตยกรรมการสะท้อน (Reflective Architectures) ซึ่งทำงานในขอบข่ายของ UML โดยการเพิ่มความสามารถของ UML ขึ้นเพื่อจะทำให้ระบบสามารถทำการปรับตัวได้ง่ายหลังจากเกิดการเปลี่ยนแปลงทางด้านสิ่งแวดล้อมที่ระบบนั้นทำงานอยู่



รูปที่ 2.15 การทำงานของสถาปัตยกรรมการสะท้อน

Klein, Rausch, Sihling and Wen (2001) ได้นำเสนอแนวคิด รวมถึงงานวิจัยที่จะทำให้ภาษาออกแบบเชิงแบบจำลองนั้นสนับสนุนการทำงานของระบบซึ่งเคลื่อนที่ได้ (Mobile Agent) ซึ่งได้ทำการเพิ่มความสามารถของภาษาออกแบบเชิงแบบจำลองให้สนับสนุนทั้งทางด้านการวิเคราะห์ และออกแบบ โดยทำให้สนับสนุนกับการทำงานทางด้านของระบบการติดต่อสื่อสาร ดังนั้นจึงได้ทำการเพิ่มความสามารถของ UML ดังรูปที่ 2.16

ซึ่งงานวิจัยนี้ได้นำเสนอทั้งลักษณะเฉพาะที่ทำงานในระบบ (Stereotype) บทบาท (Role) รวมทั้งสัญลักษณ์ภาพความสัมพันธ์ (Relation) และการส่งข้อความของการทำงานในระบบ (Message) ด้วย เป็นต้น

Description Techniques	Mobile Agent	Region	Agent System	Agency	Move	Remote Execution	Clone	Role Change
Class Diagram			√					
Object Diagram								
Sequence Diagram								
Collaboration Diagram								
Statechart Diagram					√	√	√	√
Activity Diagram					√			

รูปที่ 2.16 การเพิ่มความสามารถของ UML สำหรับการทำงานของระบบเคลื่อนที่

การแลกเปลี่ยนข้อมูลของกลุ่มข้อมูลโดย XML (XML Metadata Interchange, XMI)

XMI นั้นเป็นสำหรับการทำการแลกเปลี่ยนกลุ่มของข้อมูลโดยใช้ XML เป็นตัวกลางการติดต่อส่งผ่านเอกสารข้อมูล ซึ่งสามารถถูกแสดงได้อยู่ในรูปกลุ่มของวัตถุอำนวยความสะดวกพื้นฐาน (Meta-Object Facility, MOF) โดยส่วนมากเราจะใช้ XMI ในลักษณะตัวกลางการแลกเปลี่ยนสำหรับภาษาออกแบบเชิงแบบจำลอง (UML) ถึงแม้ว่า XMI จะสามารถถูกนำไปใช้สำหรับเป็นตัวกลางการแลกเปลี่ยนของแบบจำลอง หรือภาษาอื่น หรือกลุ่มของข้อมูลอื่น ๆ ได้

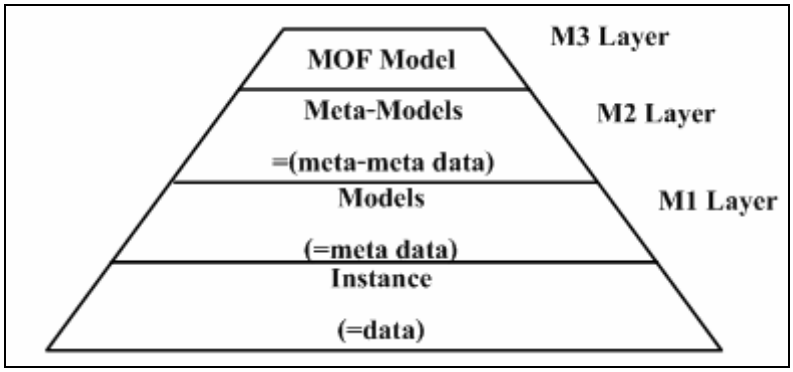
XMI นั้นถูกออกแบบมาเพื่อให้ใช้ได้ง่ายเพื่อใช้สำหรับการแลกเปลี่ยนข้อมูลระหว่างเครื่องมือสำหรับออกแบบภาษาออกแบบเชิงแบบจำลอง และที่เก็บข้อมูลต่าง ๆ ของกลุ่มของวัตถุอำนวยความสะดวกพื้นฐานในสิ่งแวดล้อมต่าง ๆ กัน โดยส่วนมาก XMI จะถูกใช้เป็นตัวกลางระหว่างเครื่องมือสร้างแบบจำลอง และเครื่องมือสร้างซอฟต์แวร์ ซึ่งเป็นส่วนหนึ่งของวิศวกรรมการขับเคลื่อนด้วยแบบจำลอง (Model-Driven Engineering)

XMI นั้นสร้างขึ้นมาจากการผสมผสานมาตรฐานของอุตสาหกรรมซอฟต์แวร์ ดังนี้

1) XML - Extensible Markup Language, มาตรฐานของ W3C
2) UML - Unified Modeling Language, มาตรฐานของ OMG
3) MOF - Meta Object Facility, มาตรฐานของOMG เพื่อระบุกลุ่มของข้อมูล
4) MOF - ทำการจับคู่ระหว่างความสัมพันธ์ของ XMI

กลุ่มของวัตถุอำนวยความสะดวกพื้นฐาน (Meta-Object Facility, MOF)

กลุ่มของวัตถุอำนวยความสะดวกพื้นฐานเป็นสิ่งที่ OMGได้ทำการคิดขึ้นเพื่อเป็นมาตรฐานสำหรับวิศวกรรมการขับเคลื่อนด้วยแบบจำลอง ซึ่งทาง OMG ต้องการสถาปัตยกรรมของกลุ่มแบบจำลองเพื่อมาระบุการทำงานของภาษาออกแบบเชิงแบบจำลอง โดยได้ทำการแบ่งชั้นของการทำงานของกลุ่มของวัตถุอำนวยความสะดวกพื้นฐานออกเป็น 4 ชั้น ดังรูป 2.17



รูปที่ 2.17 สถาปัตยกรรมของกลุ่มแบบจำลอง

ปัจจุบันได้รับมาตรฐานระดับนานาชาติ ซึ่งก็คือ ISO/IEC 19502:2005 Information technology -- Meta Object Facility (MOF)

2.2.2 เอกสารแบบการจำลองวัตถุ (Document Object Model, DOM)

เอกสารแบบการจำลองวัตถุเป็นระบบ และภาษาที่ไม่ขึ้นต่อมาตรฐานของแบบจำลองเชิงวัตถุ สำหรับแสดงเอกสารที่เป็น HTML หรือ XML และเอกสารในรูปแบบที่ใกล้เคียงโดยเฉพาะอย่างยิ่งเอกสารที่ถูกใช้ในเว็บเบราว์เซอร์ (Web Browsers) โดยเอกสารแบบจำลองวัตถุนั้นถูกสร้างมาจาก World Wide Web Consortium

ทาง W3C ได้ทำการระบุมาตรฐานของเอกสารแบบจำลองวัตถุออกเป็น ระดับหลาย ๆ ระดับ โดยในแต่ละระดับนั้นจะประกอบไปด้วยความต้องการที่จำเป็นต่อการทำงาน และคุณสมบัติพิเศษ ดังนั้นถ้าต้องการการทำงานของระบบที่เกื้อหนุนกันในทุก ๆ ส่วน ระบบการทำงานที่สร้างนั้นต้องสร้างตามที่มาตรฐานระบุตั้งแต่ต้น แต่บางที่อาจมีบางระบบที่จะได้รับการสนับสนุนจากผู้พัฒนารายอื่น ๆ ซึ่งได้เพิ่มเติมเข้ามา ซึ่งก็จะไม่ซ้ำซ้อนกับมาตรฐานของ W3C โดยได้ทำการระบุมาตรฐานของ DOM 4 ระดับ ดังนี้

ระดับ 0 เป็นระบบที่ทำการสนับสนุนการทำงานของ DOM โดยจะมีอยู่ทันที่เมื่อทำการสร้าง DOM Level 1 ซึ่ง Level 0 นั้นไม่ได้ถูกระบุอย่างเป็นทางการจาก W3C ในทางตรงกันข้าม Level 0 นั้นเป็นส่วนที่เกิดก่อนกระบวนการกำหนดมาตรฐาน

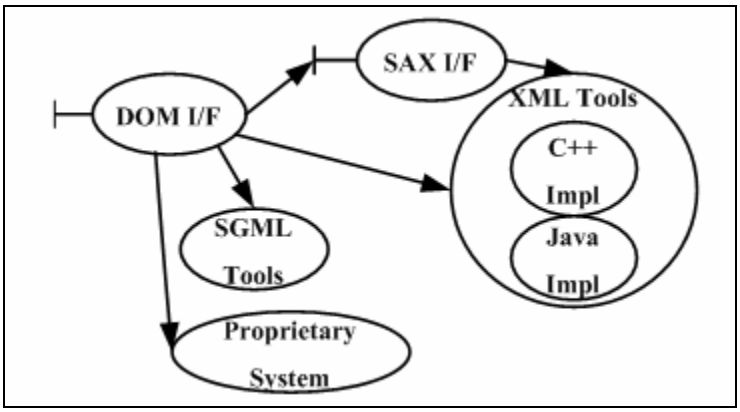
ระดับ 1 เป็นการระบุเอกสารของ DOM และการจัดการเอกสาร

ระดับ 2 สนับสนุน XML Namespace รวมถึงกรองการร้องขอและเหตุการณ์

ระดับ 3 ประกอบไปข้อกำหนด 6 ข้อ ได้แก่

1) DOM Level 3 Core
2) DOM Level 3 Load and Save
3) DOM Level 3 XPath
4) DOM Level 3 Views and Formatting
5) DOM Level 3 Requirements
6) DOM Level 3 Validation

ซึ่งข้อกำหนดสำคัญทั้งหมดนั้นมีอยู่ในเอกสารของการระบุมาตรฐานของ DOM และอยู่ที่ W3C



รูปที่ 2.18 แสดงการติดต่อโดยมี DOM เป็นตัวกลาง

Suzuki and Yamamoto (1999) ได้นำเสนองานวิจัยในการเปลี่ยนรูปซึ่งเป็นมาตรฐานของภาษาออกแบบเชิงแบบจำลอง และนำเสนอของการเปลี่ยนรูปแบบของ UML ในแบบที่กระทำได้ โดยได้เสนอการแลกเปลี่ยนข้อมูลโดยให้อยู่ในรูปแบบ XML เรียกว่า UXF (UML eXchange Format) และได้ทำการพัฒนาความสามารถในการจัดการแบบจำลองสำหรับภาษาออกแบบเชิงแบบจำลอง โดยระบบนี้จะเป็นผลดีต่อกลุ่มของนักพัฒนาระบบ เพราะสามารถนำงาน

ที่ออกแบบกลับมาใช้ใหม่ได้ และเครื่องมือในการพัฒนานั้นสามารถทำการเปลี่ยนแปลงระหว่างข้อมูลและแบบจำลองได้โดยผ่าน XML ซึ่งถูกออกแบบให้อยู่บนมาตรฐานของเอกสารแบบการจำลองวัตถุ (DOM) ซึ่ง DOM นั้นจะเป็นตัวกลางที่ทำหน้าที่ติดต่อระหว่างมาตรฐานของระบบ และภาษาของการเขียนโปรแกรมโดยจะช่วยจัดการเนื้อหา โครงสร้าง และรูปแบบของเอกสาร ดังรูปที่ 2.18

2.3 การเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming, OOP)

การเขียนโปรแกรมเชิงวัตถุ เป็นวิธีการเขียนโปรแกรมรูปแบบหนึ่ง โดยใช้ความหมายวัตถุในการออกแบบระบบงาน และเป็นสิ่งที่ทำให้คอมพิวเตอร์ทำงานตามที่กำหนด ซึ่งการเขียนโปรแกรมเชิงวัตถุได้นำประโยชน์และส่วนดีจากวิธีการเขียนโปรแกรมแบบก่อน ๆ มาใช้ โดยจะประกอบไปด้วย การสืบทอดคุณสมบัติ (Inheritance) การประกอบด้วยหน่วยแยกต่าง ๆ ที่สามารถรวมกันได้ (Modularity) การฟ้องรูป (Polymorphism) และ การหุ้มห่อ (Encapsulation)

การเขียนโปรแกรมเชิงวัตถุบางทีอาจถูกมองได้ว่าเป็นกลุ่มของวัตถุที่ทำงานร่วมกัน ซึ่งถ้ามองเป็นภาพการเขียนโปรแกรมแบบก่อน ๆ ก็คือ กลุ่มการทำงานของฟังก์ชั่น (Function) หรือรายการของคำสั่งการทำงานของเครื่องคอมพิวเตอร์

2.3.1 แนวคิดขั้นพื้นฐานของการเขียนโปรแกรมเชิงวัตถุ (Fundamental Concepts)

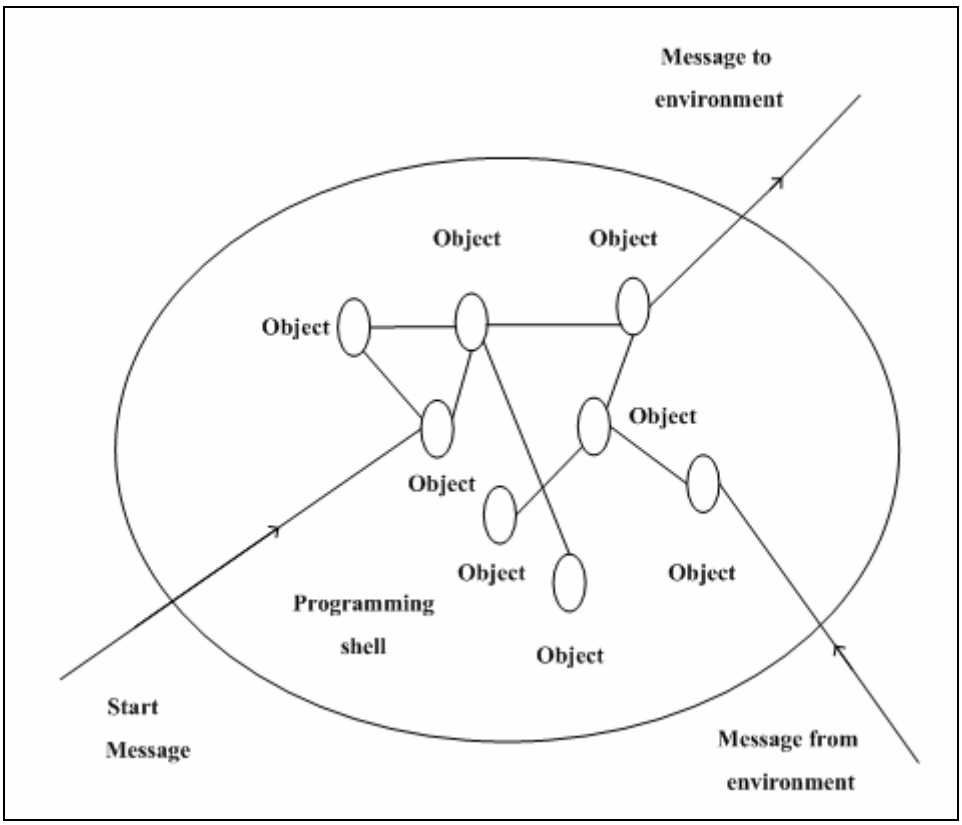
Armstrong and Deborah (2006) ได้ทำการนิยามคุณสมบัติพื้นฐาน (Quarks) หรือแนวคิดขั้นพื้นฐานของการเขียนโปรแกรมเชิงวัตถุ ไว้ดังนี้

คลาส (Class) คลาสนั้นเป็นการระบุลักษณะของแนวคิดในของการทำงานในรูปนามธรรม (Abstract) โดยรวมเอาลักษณะเฉพาะของแนวคิด (Attributes or Properties) และสิ่งที่ลักษณะของแนวคิดนั้นทำ (Behaviors or Methods or Features) ซึ่ง คุณสมบัติ (Properties) และเมธอด (Methods) นั้นถูกเรียกได้ว่าเป็นส่วนหนึ่งของคลาส

วัตถุ (Object) เป็นส่วนประกอบส่วนหนึ่งที่ทำหน้าที่ตามที่ได้ถูกระบุไว้ในคลาส

เมธอด (Methods) เป็นหน่วยของโปรแกรม เพื่อใช้แก้ปัญหาหรือทำงานอย่างหนึ่งสำหรับเรียกใช้เมื่อต้องการ เมธอดนั้นจะช่วยให้สามารถแบ่งงานทั้งหมดออกเป็นส่วนย่อย ๆ ซึ่งง่ายต้องการสร้าง และจัดการมากกว่าจะเขียนเป็นหน่วยเดียวทั้งหมด ซึ่งถ้าจะแก้ไขการทำงานหรือรายละเอียดแต่ละเมธอดก็ทำได้ง่ายกว่า โดยที่จะไม่ส่งผลกระทบต่อหน่วยอื่น ๆ

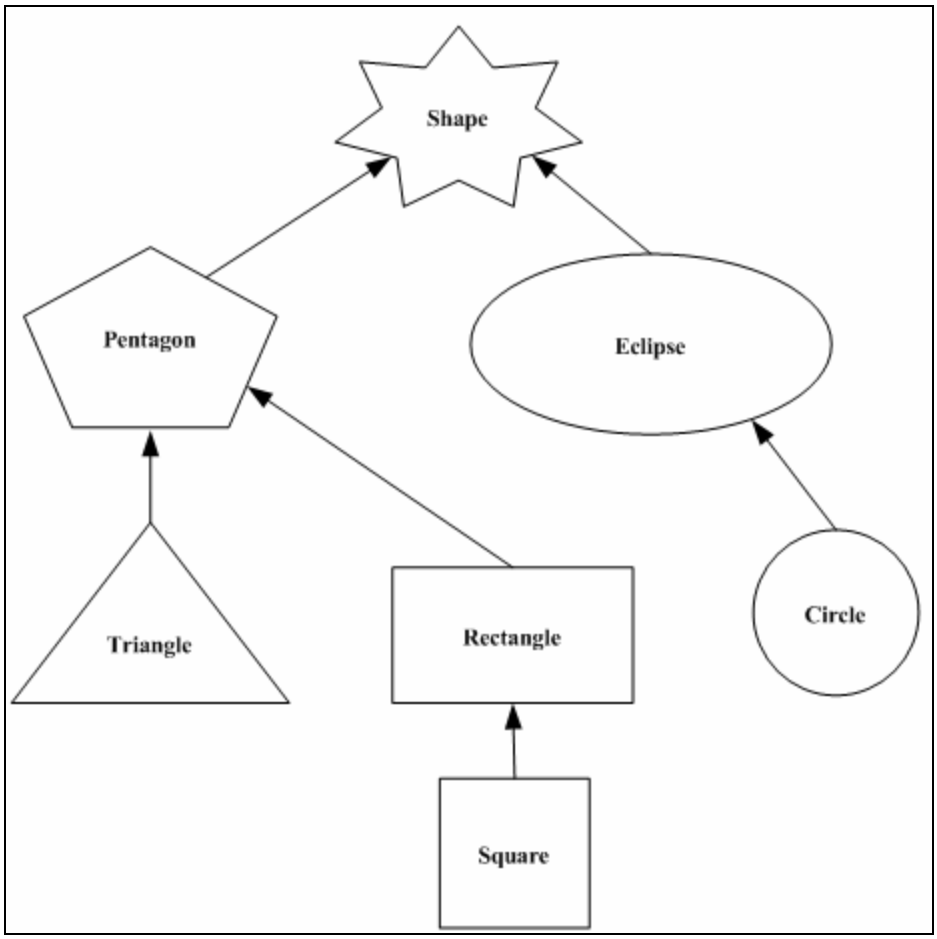
การส่งข้อความ (Message Passing) เป็นกระบวนการที่วัตถุทำการส่งข้อมูลไปให้อีกวัตถุหนึ่ง หรือเป็นการส่งค่าไปถามเพื่อขอใช้เมธอด ดังรูปที่ 2.19



รูปที่ 2.19 แสดงการส่งข้อความของวัตถุ

การสืบทอดคุณสมบัติ (Inheritance) บางคลาสมีโอกาสที่เป็นคลาสย่อยของคลาสอื่น โดยคลาสที่เป็นคลาสย่อยนั้นสามารถทำงานในส่วนของคลาสหลักได้ และยังสามารถทำงานในส่วนอื่นซึ่งคลาสหลักนั้นทำไม่ได้ด้วย ซึ่งเรียกกลไกลการสร้างคลาสอย่างนี้ว่าการสืบทอดคุณสมบัติ (Inheritance) ซึ่งเป็นกลไกลหนึ่งที่เป็นเงื่อนไขของภาษาเชิงวัตถุ ซึ่งมีลักษณะดังรูปที่ 2.20

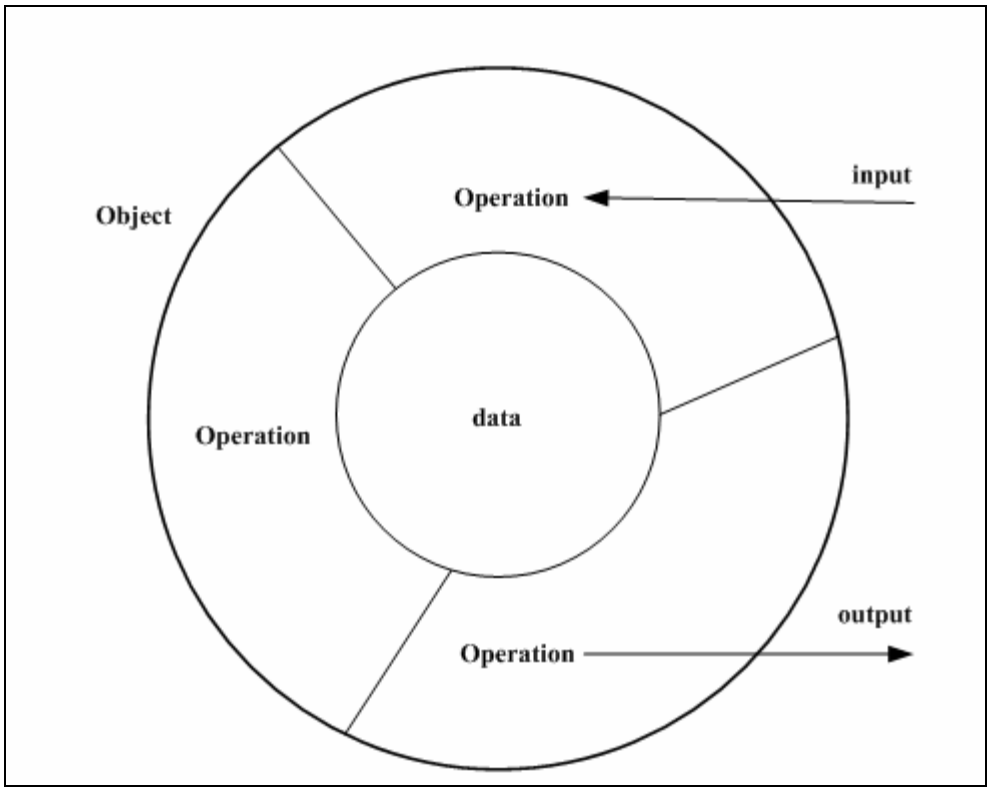
การหุ้มห่อ (Encapsulation) เป็นคุณสมบัติที่หุ้มห่อโครงสร้างของคลาสซึ่งโครงสร้างของคลาสนั้นจะประกอบไปด้วย คุณสมบัติ และ เมธอด และการจะใช้งานคลาสนั้นจะถูกออกแบบให้เข้าถึงเฉพาะบางส่วนเท่านั้นตามแต่ที่คลาสนั้นระบุไว้ โดยการใช้งานนั้นผู้ใช้ไม่จำเป็นที่ต้องเข้าใจโครงสร้างของข้อมูลที่อยู่ในนั้น ก็สามารถใช้งานข้อมูลเหล่านั้นทางเมธอดที่ได้ถูกระบุไว้ ซึ่งมีลักษณะดังรูปที่ 2.21



รูปที่ 2.20 คุณสมบัติการสืบทอด

นามธรรม (Abstraction) เป็นการออกแบบให้การแก้ปัญหาซึ่งเกิดจากซับซ้อนของงานให้ง่ายขึ้นโดยออกแบบการทำงานของคลาสให้เหมาะสม และทำการแก้ปัญหาให้เหมาะสมกับระดับของการสืบทอดของคลาส

การพ้องรูป (Polymorphism) ภาษาเชิงวัตถุมีกลไกการสืบทอดคุณสมบัติ และ Dynamic Binding ทำให้สามารถเมธอดที่มีลักษณะพ้องรูป คือ เมธอดที่สามารถจัดการกับการทำงานของตัวแทนของคลาสที่ต่างกันได้ โดยมีเงื่อนไขว่าคลาสนั้นต้องเป็นคลาสที่ขยายมาจากคลาสหนึ่งที่กำหนดให้เป็นพารามิเตอร์ของเมธอดนั้น



รูปที่ 2.21 คุณสมบัติการหุ้มห่อ

2.3.2 การวิเคราะห์และออกแบบระบบเชิงวัตถุ (Object-Oriented Analysis and Design)

การวิเคราะห์และออกแบบระบบเชิงวัตถุเป็นการนำเอาแบบจำลองของวัตถุมาประยุกต์ใช้ให้เข้ากับบริบทของความต้องการ ซึ่งส่วนมากการวิเคราะห์และวิธีการออกแบบเชิงวัตถุนั้นคือใช้ขอบข่ายการทำงานของระบบเป็นตัวขับเคลื่อน ไม่ว่าจะเป็นการทำการเก็บความต้องการ (Requirement), การออกแบบระบบ (Design), การทำการสร้างระบบงาน (Implementation), การทำสอบระบบ (Testing) และการส่งมอบระบบ (Deployment)

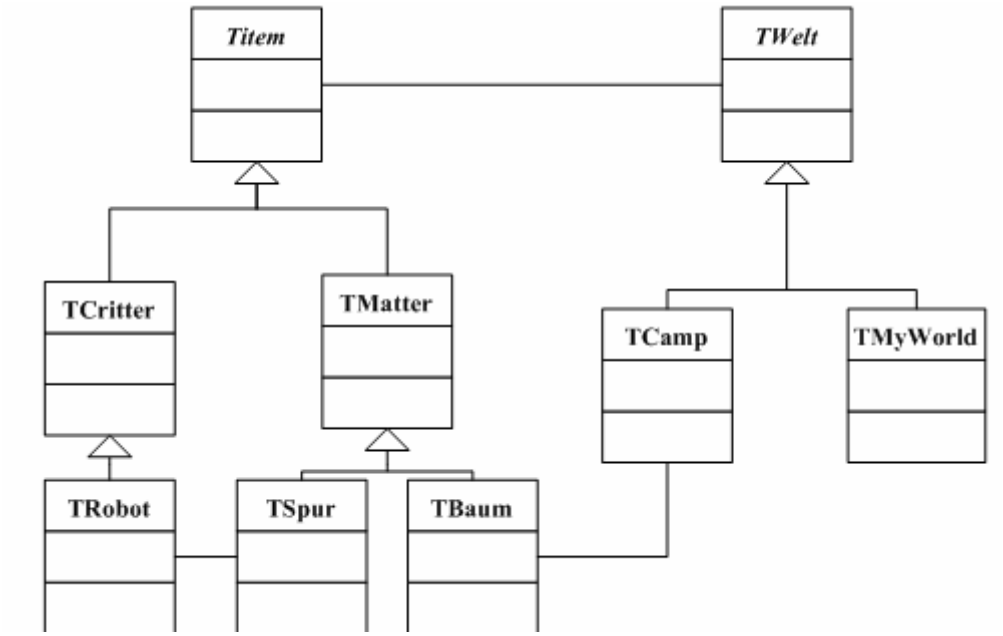
1) ระบบเชิงวัตถุ (Object-Oriented Systems)

ระบบเชิงวัตถุนั้นจะประกอบไปด้วยพฤติกรรมของระบบที่ทำงานร่วมกันระหว่างวัตถุที่อยู่ในระบบเหล่านี้ และสถานะของระบบนั้นจะประกอบไปด้วยสถานะโดยรวมของวัตถุที่อยู่ในระบบด้วยเช่นกัน การทำงานร่วมกันของวัตถุนั้นเกี่ยวพันกับการส่งข้อความ ข้อมูลหรือสัญญาณไปให้ส่วนอื่น ๆ ดังรูปที่ 2.19

2) การวิเคราะห์เชิงวัตถุ (Object-Oriented Analysis, OOA)

การวิเคราะห์เชิงวัตถุนั้นเป็นการมุ่งเน้นเพื่อออกแบบจำลองของปัญหาที่สนใจ (Problem Domain) ซึ่งปัญหาที่มุ่งเน้นนั้นจะถูกแก้ปัญหาด้วยวิธีทางระบบเชิงวัตถุ โดยการเริ่มต้นในการวิเคราะห์นี้นั้นจะเริ่มจากการทำการเก็บความต้องการ และการทำแบบจำลองของการวิเคราะห์นั้นจะทำโดยไม่ต้องไปสนใจปัญหาของขั้นตอนการสร้าง และพัฒนาว่าระบบนั้นจะต้องถูกพัฒนาขึ้นมาด้วยอะไรและอย่างไร แบบจำลองของระบบนั้นสามารถถูกแบ่งออกมาได้เป็นหลาย ๆ ความสนใจ ซึ่งในแต่ละความสนใจนั้นเป็นการวิเคราะห์ระบบ และแสดงให้เห็นถึงการทำงานของระบบ หรือเทคโนโลยี และเทคนิคของการใช้พัฒนาระบบ หรือจะเป็นขอบเขตของแนวคิดของระบบ

ผลลัพธ์ของการวิเคราะห์เชิงวัตถุนั้นคือการอธิบายสิ่งที่ต้องการจะสร้างว่าต้องการจะสร้างอะไร การใช้แนวคิดและความสัมพันธ์ระหว่างแนวคิดนั้น และหลายครั้งอาจถูกเรียกว่าเป็นแบบจำลองของแนวคิด (Conceptual Model) ซึ่งเงื่อนไขในขั้นตอนของการสร้างและพัฒนานั้นจะถูกอธิบายโดยกระบวนการออกแบบเชิงวัตถุต่อไป ซึ่งรูปที่ 2.22 เป็นตัวอย่างของแบบจำลองที่ได้จากการวิเคราะห์เชิงวัตถุ



รูปที่ 2.22 แบบจำลองของการวิเคราะห์เชิงวัตถุ

3) การออกแบบเชิงวัตถุ (Object-Oriented Design, OOD)

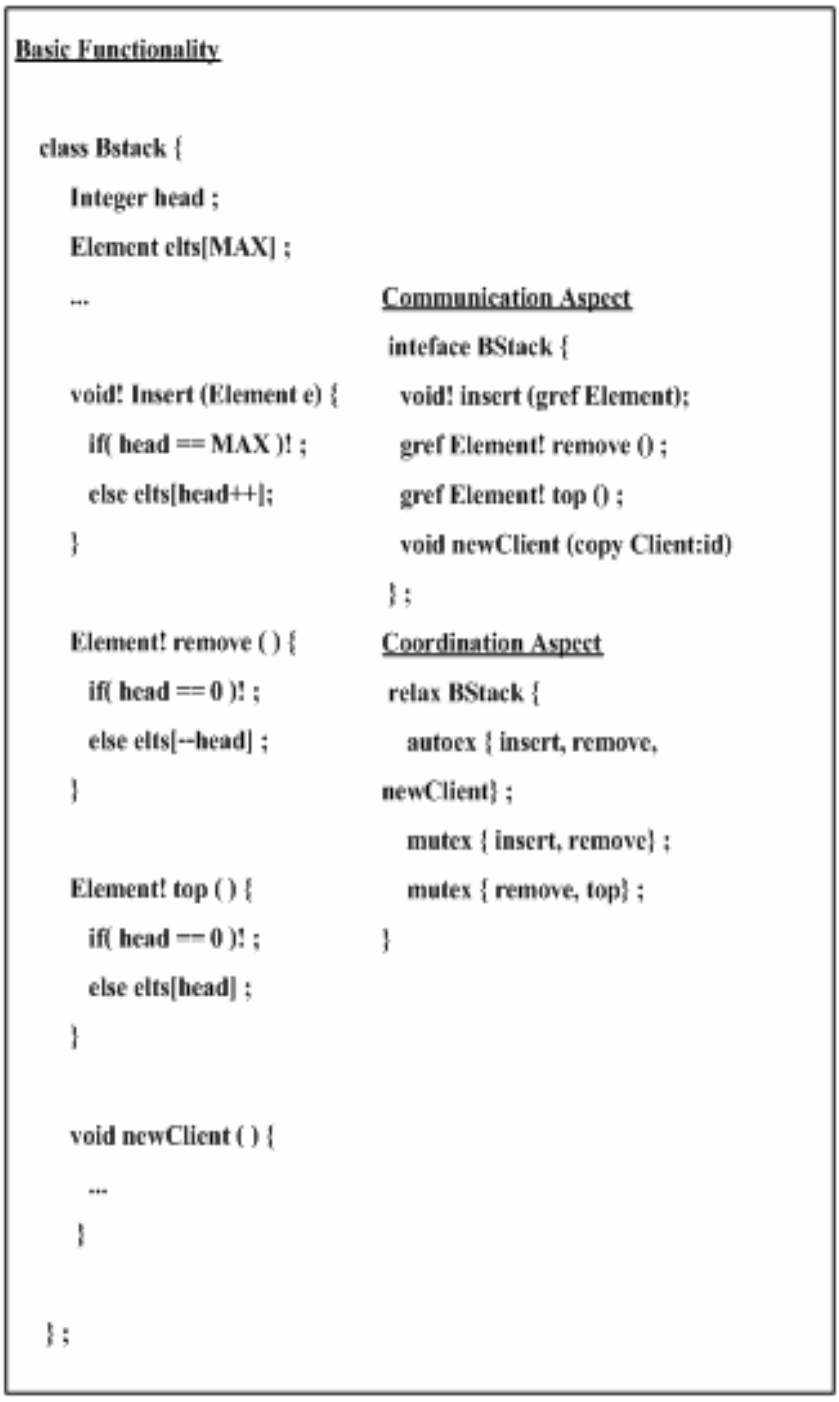
การออกแบบเชิงวัตถุเป็นกิจกรรมที่ทำการสร้างแนวคิดที่จะแก้ไขปัญหาด้วยวิธีไหน การออกแบบเชิงวัตถุเป็นการนำแบบจำลองของแนวคิดที่ได้มาจากขั้นตอนการวิเคราะห์เชิงวัตถุมาและเพิ่มเงื่อนไขของการพัฒนาระบบเข้าไป ซึ่งได้แก่สิ่งแวดล้อมของระบบที่จะทำการพัฒนา ภาษาที่ใช้ในการพัฒนา และเครื่องมือที่ใช้ในการพัฒนา โดยทั้งหมดนั้นต้องอยู่บนรากฐานของสมมติฐานที่ออกแบบ โดยแนวคิดในแบบจำลองของแนวคิดนั้นจะถูกโยงเข้ากับคลาสที่เป็นรูปธรรม (Concrete Class) คลาสที่เป็นนามธรรม (Abstract Class) และบทบาทซึ่งวัตถุจะเปลี่ยนไปเมื่ออยู่ในสถานะต่าง ๆ กัน ผลลัพธ์ที่ได้จากการออกแบบเชิงวัตถุนั้นคือการอธิบายรายละเอียดว่าระบบนั้นจะถูกสร้างขึ้นมาอย่างไร และวัตถุอะไร

2.4 การเขียนโปรแกรมเชิงลักษณะ (Aspect-Oriented Programming, AOP)

การเขียนโปรแกรมเชิงลักษณะเป็นการวิธีการเขียนโปรแกรมสมัยใหม่ที่ได้ทำการพัฒนาเพื่อขจัดข้อบกพร่องที่ปรากฏอยู่จากวิธีการเขียนโปรแกรมเชิงวัตถุโดยใช้วิธีการดึงเอาส่วนที่มีความสัมพันธ์หรือเกี่ยวข้องกันออกมา (Separation of Concerns) และทำการระบุการทำงานของส่วนที่ดึงออกมาโดยเรียกส่วนที่ถูกดึงนี้ออกมาว่าจุดตัด (Cross-cutting Concerns) ซึ่งการทำงานนั้นจะใช้ภาษาเชิงลักษณะเป็นสำคัญ ในขณะที่การพัฒนาโปรแกรมเชิงลักษณะ (Aspect-Oriented Software Development, AOSD) นั้นใช้เพื่อสำหรับนำเอาภาษาอื่น หรืองานอื่น ๆ เข้ามาทำงานร่วมกันกับการทำงานของภาษาเชิงลักษณะดังรูปที่ 2.23

2.4.1 การแยกความสัมพันธ์ (Separation of Concerns)

การแยกความสัมพันธ์ของส่วนที่เกี่ยวข้องกันนั้นอาจจะส่งผลซึ่งไม่อาจหลีกเลี่ยงได้ให้แก่ระบบ ซึ่งอย่างน้อยก็จะเกิดในส่วนของการทำงานที่ซ้อนทับกัน วิธีการเขียนโปรแกรมทุกอย่าง ซึ่งรวมถึงการเขียนโปรแกรมแบบเป็นขั้นตอน (Procedural Programming) และการเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming) นั้นสนับสนุนการทำงานในการแยกส่วน และการคลอบคลุมหุ้มห่อบางส่วนอยู่แล้วในระดับหนึ่ง อย่างเช่นการทำงานของคลาสแม่ และคลาสลูก โดยการทำงานที่มีลักษณะเหมือนกันจะถูกรวบเอาไว้เรียกว่าการจัดหมู่ในแนวตั้ง (Vertical Grouping) แต่อาจมีบางกรณีที่ไม่สามารถจัดการด้วยการจัดหมู่ในแนวตั้งได้ โดยทางวิศวกรรมซอฟต์แวร์เรียกว่าความสัมพันธ์ของจุดที่ตัดกัน (Crosscutting Concerns) เพราะว่าการตัดกันนั้นเกิดขึ้นในหลาย ๆ ส่วนของระบบ



รูปที่ 2.23 กลไกการทำงานของการเขียนโปรแกรมเชิงลักษณะ

Suzuki and Yamamoto (1999) ได้ทำการเสนอแนวคิด และงานวิจัยที่พยายามนำลักษณะการทำงานของการเขียนโปรแกรมเชิงลักษณะมาทำงานร่วมกับภาษาออกแบบเชิง

แบบจำลอง โดยได้เพิ่มเติมการทำงานของการเขียนโปรแกรมเชิงลักษณะลงไปให้กับภาษาออกแบบเชิงโมเดล

จากที่ได้กล่าวมาทั้งหมดนั้น จะเห็นได้ว่าวิธีการพัฒนาซอฟต์แวร์ไม่ว่าจะเป็นเทคนิคการพัฒนา และเทคโนโลยีที่จำนำมาใช้ในการพัฒนานั้นสามารถนำไปปรับใช้ให้เข้ากันกับการทำงานของแต่ละกระบวนการของวัฏจักรการพัฒนาซอฟต์แวร์ (Software Development Life Cycle) ได้ซึ่งทั้งนี้การจะเลือกใช้วัฏจักรเป็นในการพัฒนานั้นขึ้นอยู่กับแต่ละปัญหาที่ได้เกิดขึ้นในแต่ละระบบ ดังนั้นถ้าจะมีปรับปรุงเทคนิค และรายละเอียดของการพัฒนานั้นอาจจะส่งผลกระทบกระบวนการทำงานของแต่ละวัฏจักรบ้าง แต่คงไม่ร้ายแรงมากนักเมื่อนำมาเทียบกับประโยชน์ที่ได้จากการปรับปรุง ทั้งนี้การนำเอาความสามารถของสถาปัตยกรรมการขับเคลื่อนด้วยแบบจำลอง (MDA) และเอกสารแบบการจำลองวัตถุ (DOM) มาประยุกต์ใช้จะช่วยลดผลกระทบให้น้อยลงตามไปด้วย เพราะการทำงานของสถาปัตยกรรมเหล่านี้นั้นจะแยกแนวคิดออกจากเทคนิคการพัฒนา และ เทคโนโลยีอยู่แล้ว และการทำการเพิ่มเติมความสามารถของการทำงานของสถาปัตยกรรมการขับเคลื่อนด้วยแบบจำลองนั้นก็เพื่อเป็นการปรับกระบวนการให้เข้ากันกับการทำงาน และปัญหาที่เกิดขึ้นในปัจจุบันนี้ โดยขั้นตอนและวิธีการดำเนินการวิจัยนั้นจะได้กล่าวถึงในบทต่อไป

บทที่ 3
วิธีการดำเนินการวิจัย

งานวิจัยนี้มุ่งหมายในการออกแบบและพัฒนาเครื่องมือในการสร้างแบบจำลองของภาษาออกแบบเชิงแบบจำลองสำหรับการพัฒนาเพื่อนำไปใช้เพื่อขั้นตอนการวิเคราะห์ และออกแบบระบบซอฟต์แวร์สำหรับทุกกระบวนการการพัฒนาซอฟต์แวร์ และทุกวัฏจักรของการพัฒนาซอฟต์แวร์ที่เป็นการพัฒนาซอฟต์แวร์เชิงวัตถุ เพื่อประโยชน์ในการนำกลับมาใช้ใหม่, การแก้ไข ตรวจสอบ และลดภาระของผู้ทำการวิเคราะห์และออกแบบ รวมทั้งผู้พัฒนาระบบอีกด้วย โดยในงานวิจัยนี้จะศึกษาถึงวิธีการในการวิเคราะห์และออกแบบเชิงวัตถุ ที่ได้มาจากข้อกำหนดของการวิเคราะห์และออกแบบซอฟต์แวร์ซึ่งอ้างอิงมาจาก UML Specification 1.5 จาก OMG, มาตรฐานการทำงานของ DOM จาก W3C และการทำงานของการเขียนโปรแกรมเชิงลักษณะ มาเพิ่มมาตรฐานของการออกแบบด้วยภาษา UML โดยการทำงานของ UML ที่ได้ทำการเพิ่มความสามารถเข้าไปแล้วนั้นจะทำงานได้ยืดหยุ่นขึ้นและตรงตามเทคนิคการพัฒนาซอฟต์แวร์ในปัจจุบันมากขึ้นด้วย โดยรายละเอียดในบทที่ 3 นี้จะเริ่มจากระเบียบวิธีวิจัยดังแสดงในหัวข้อที่ 3.1 ส่วนในหัวข้อที่ 3.2 จะเป็นการนำเสนอการเพิ่มความสามารถภาษาออกแบบเชิงแบบจำลองด้วยเทคนิคการเขียนโปรแกรมเชิงลักษณะ หัวข้อที่ 3.3 จะเป็นการวิเคราะห์และออกแบบเครื่องมือสำหรับแสดงการวิเคราะห์และออกแบบภาษา UML ในหัวข้อ 3.4 นั้นจะเป็นการออกแบบของการเปลี่ยนแปลงรูปร่างระหว่างแบบจำลอง และเอกสาร XML และส่วนของการนำเอกสาร XML ที่ได้มานั้นเปลี่ยนเป็นโค้ดจริง ในหัวข้อ 3.5 เป็นการออกแบบส่วนแสดงผลกราฟิกจากการออกแบบด้วยภาษาออกแบบเชิงแบบจำลองที่เพิ่มความสามารถของเทคนิคการเขียนโปรแกรมเชิงลักษณะ และส่วนติดต่อกับผู้ใช้งาน

3.1 ระเบียบวิธีวิจัย

1) ศึกษาค้นคว้าและรวบรวมงานวิจัยที่เกี่ยวข้อง
2) ศึกษาวิธีการวิเคราะห์ และออกแบบสำหรับ UML
3) ศึกษาวิธีการเขียนโปรแกรมเชิงลักษณะ
4) ศึกษาการทำงานของเอกสารแบบการจำลองวัตถุ (DOM)

5) ศึกษาการใช้งานระบบเครื่องมือสำหรับทำการออกแบบภาษาออกแบบเชิงแบบจำลอง (UML) แบบต่าง ๆ ทั้งที่เป็นส่วนของที่เป็นลิขสิทธิ์ที่เป็นสำหรับการค้า และที่เป็นของโอเพนซอร์ส (Open Source) เพื่อหารูปแบบ และเทคโนโลยีที่เหมาะสมสำหรับนำมาพัฒนา
6) ออกแบบและพัฒนารูปแบบการนำภาษาออกแบบเชิงแบบจำลอง, การเขียนโปรแกรมเชิงลักษณะ พร้อมทั้งนำเอกสารแบบการจำลองวัตถุมาใช้เป็นส่วนติดต่อกับแบบจำลองและผลลัพธ์
7) ทดสอบผลการทำงานของเครื่องมือ และทำการแก้ไขข้อผิดพลาด
8) วิเคราะห์ และสรุปผลการทำวิจัย

3.2 การเพิ่มความสามารถภาษาออกแบบเชิงแบบจำลองด้วยเทคนิคการเขียนโปรแกรมเชิงลักษณะ

ในส่วนนี้จะเป็นการนำเสนอการเพิ่มความสามารถภาษาออกแบบเชิงแบบจำลองด้วยเทคนิคการเขียนโปรแกรมเชิงลักษณะ (Extensible Unified Modeling Language with Aspect-Oriented Programming) ซึ่งเป็นการนำเสนอแนวทางการทำงานของภาษาออกแบบเชิงแบบจำลองเพื่อให้เหมาะสมสำหรับการเขียนโปรแกรมเชิงลักษณะ ซึ่งจะสนับสนุนทั้งการทำงานแบบการเขียนโปรแกรมเชิงวัตถุซึ่งเป็นความสามารถก่อนนำมาเพิ่มเติมด้วย

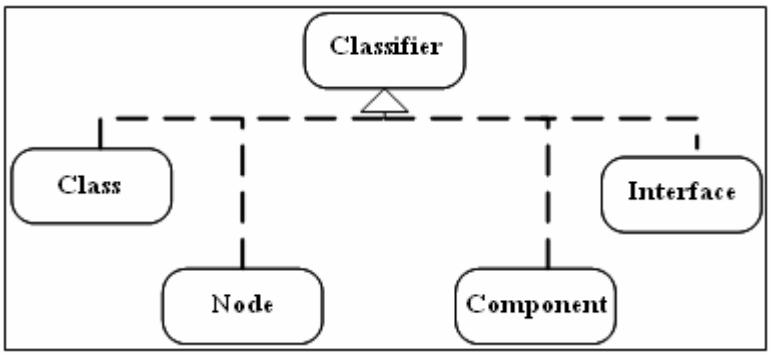
สำหรับการเพิ่มเติมความสามารถของภาษาออกแบบเชิงแบบจำลองนั้น จะทำการพัฒนาบนมาตรฐานเดิมของภาษาออกแบบเชิงแบบจำลอง ซึ่งจะสามารถทำงานแบบต่าง ๆ ตามที่ได้ถูกออกแบบมาตั้งแต่ต้น การเพิ่มความสามารถด้วยเทคนิคการเขียนโปรแกรมเชิงลักษณะนั้นจะถูกเพิ่มลงไปในส่วนของแผนภาพคลาส (Class Diagram) ซึ่งเรียกว่าแผนภาพคลาสแบบลักษณะ (Aspect Class Diagram) และทำการอธิบายโครงสร้างและการทำงานของแผนภาพนี้ด้วยเอกสารแบบการจำลองวัตถุ

3.2.1 การวิเคราะห์ และออกแบบการทำงานของภาษาออกแบบเชิงแบบจำลองด้วยเทคนิคการเขียนโปรแกรมเชิงลักษณะ

ในกระบวนการออกแบบเพื่อเพิ่มเติมความสามารถของเทคนิคการเขียนโปรแกรมเชิงลักษณะลงไปนั้น จะต้องเกิดขึ้นในส่วนของกลุ่มของแบบจำลองของภาษาออกแบบเชิงแบบจำลอง (UML Metamodel) เพราะจะทำให้สนับสนุนการทำงานแบบเดิม ซึ่งส่วนของกลุ่มของแบบจำลองของภาษาออกแบบเชิงแบบจำลองก่อนจะทำการเพิ่มความสามารถลงไปในแผนภาพและแผนภาพ จะนั้นประกอบไปด้วยข้อมูลดังนี้

- Classifier
- Class
- Interface
- Node
- Component

องค์ประกอบของกลุ่มของแบบจำลองของภาษาออกแบบเชิงแบบจำลองนั้นจะเป็นการบอกถึงพฤติกรรมการทำงานและลักษณะโครงสร้างของภาษา ซึ่งรูปที่ 3.1 แสดงถึงโครงสร้างการทำงานของภาษาออกแบบเชิงแบบจำลอง

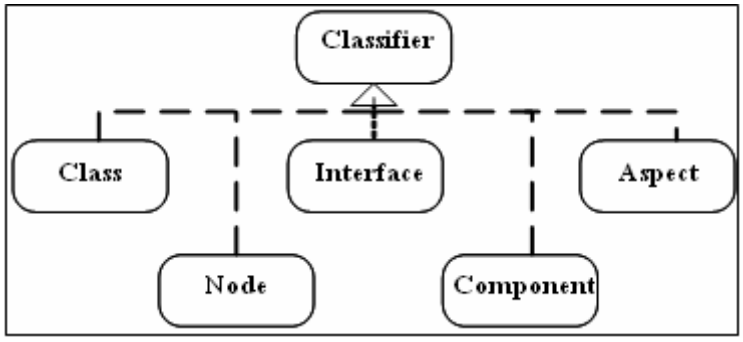


รูปที่ 3.1 โครงสร้างการทำงานของภาษาออกแบบเชิงแบบจำลอง

ดังนั้นการเพิ่มมาตรฐานของภาษาออกแบบเชิงแบบจำลองให้สนับสนุนการทำงานของเทคนิคการเขียนโปรแกรมเชิงลักษณะนั้นต้องเริ่มจากส่วนนี้ และองค์ประกอบของภาษาออกแบบเชิงแบบจำลองหลังจากได้เพิ่มเทคนิคการเขียนโปรแกรมเชิงลักษณะนั้นจะมีโครงสร้างเพิ่มขึ้นดังนี้

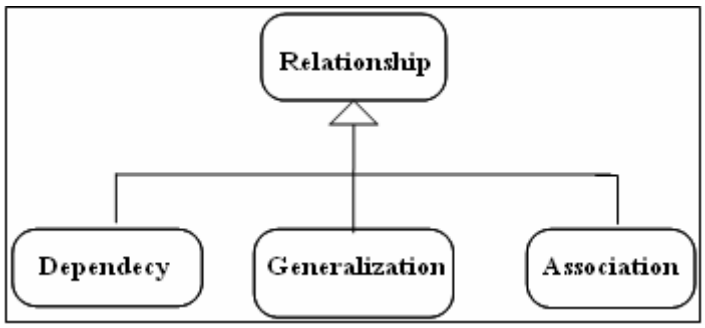
- Classifier
- Class
- Aspect
- Interface
- Node
- Component

ซึ่งรูปที่ 3.2 แสดงถึงโครงสร้างการทำงานของภาษาออกแบบเชิงแบบจำลองที่ได้เพิ่มความสามารถของเทคนิคการเขียนโปรแกรมเชิงลักษณะเข้าไป



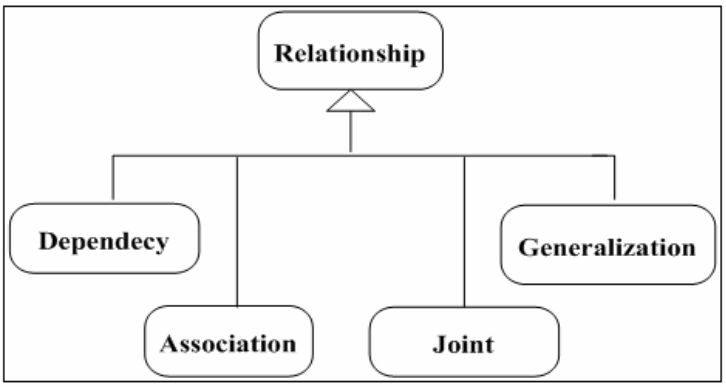
รูปที่ 3.2 โครงสร้างการทำงานของภาษาออกแบบเชิงแบบจำลองที่เพิ่มความสามารถของการเขียนโปรแกรมเชิงลักษณะ

นอกจากจะทำการเพิ่มเติมความสามารถของภาษาออกแบบเชิงแบบจำลองในส่วนที่เป็น Classifier แล้วจะต้องมีการเพิ่มความสัมพันธ์ (Relationship) ระหว่างคลาสกับคลาส โดยความสัมพันธ์ระหว่างคลาสเดิมนั้นมีลักษณะดังรูปที่ 3.3



รูปที่ 3.3 โครงสร้างความสัมพันธ์ระหว่างคลาส

ความสัมพันธ์ของมาตรฐานเดิมนั้นสนับสนุนเพียงการทำงานของการเขียนโปรแกรมเชิงวัตถุ ซึ่งไม่ได้สนับสนุนการทำงานของการเขียนโปรแกรมเชิงลักษณะ ดังนั้นการทำให้มาตรฐานความสัมพันธ์ของภาษาออกแบบเชิงแบบจำลองสนับสนุนการทำงานของการเขียนโปรแกรมเชิงลักษณะนั้นจะต้องเพิ่มความสัมพันธ์ในเชิงลักษณะเรียกว่าความสัมพันธ์ร่วม (Joint) ซึ่งรูปที่ 3.4 จะแสดงให้เห็นถึงลักษณะความสัมพันธ์ที่ถูกเพิ่มเข้ามากับความสัมพันธ์เดิม



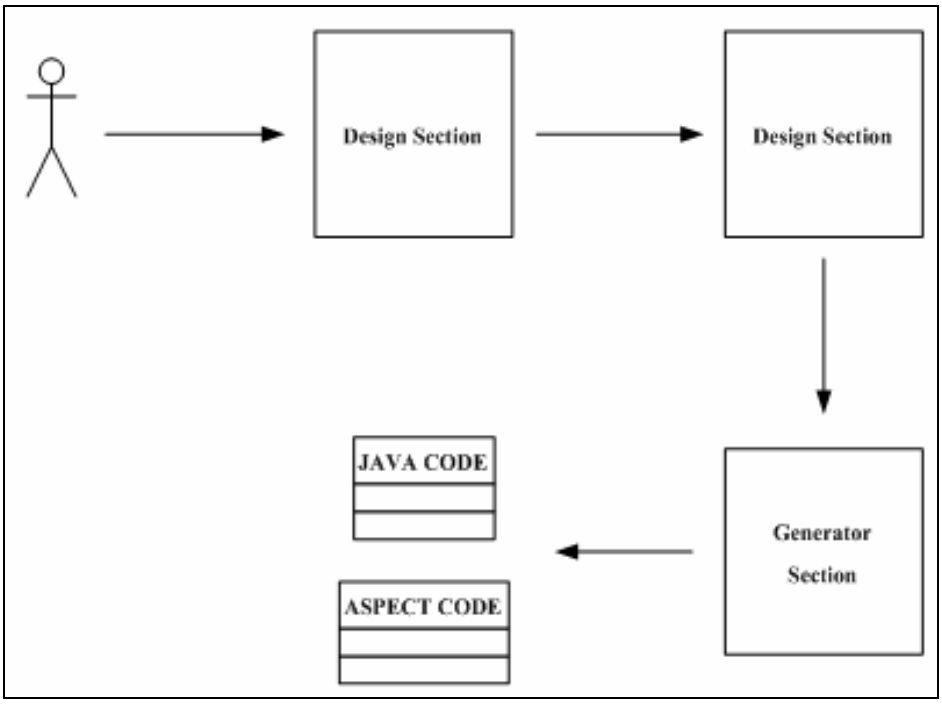
รูปที่ 3.4 โครงสร้างความสัมพันธ์ระหว่างคลาสเชิงลักษณะ

การเพิ่มเติมคุณสมบัติของการเขียนโปรแกรมเชิงลักษณะลงไปในภาษาออกแบบเชิงแบบจำลองนั้น เพื่อทำให้เกิดการสนับสนุนและสามารถรองรับกับการทำงานของเทคนิคของการเขียนโปรแกรมเชิงลักษณะ ซึ่งจะสามารถช่วยลดและจัดการกับปัญหาที่อาจจะเกิดขึ้นจากระบบของงาน หรือเทคนิคการออกแบบ และพัฒนาโปรแกรมแบบเดิมได้ โดยในหัวข้อถัดไปจะเป็นการนำเอาแนวคิดของการเพิ่มความสามารถของการเขียนโปรแกรมเชิงลักษณะลงไปในภาษาออกแบบเชิงแบบจำลองมาพัฒนาเป็นเครื่องมือซึ่งจะช่วยทำให้เกิดการนำไปใช้ได้อย่างสะดวก และรวดเร็วขึ้น

3.3 การวิเคราะห์ และออกแบบเครื่องมือสำหรับช่วยในการวิเคราะห์ และออกแบบภาษาออกแบบเชิงแบบจำลอง ที่เพิ่มความสามารถของการเขียนโปรแกรมเชิงลักษณะ

ในกระบวนการวิเคราะห์ และออกแบบระบบซอฟต์แวร์เชิงวัตถุ (Object-Oriented Analysis and Design) ผู้ที่ทำการวิเคราะห์และออกแบบระบบต้องสามารถนำเอาความต้องการของระบบ (Requirement) มาสร้างเป็นสัญลักษณ์ที่สามารถอธิบายการทำงานของระบบได้ สำหรับการพัฒนาซอฟต์แวร์เชิงวัตถุแล้วภาษาออกแบบเชิงแบบจำลองเป็นมาตรฐานที่ถูกนำมาใช้กันอย่างแพร่หลาย ดังนั้นการออกแบบระบบงานให้ตรงตามวัตถุประสงค์และความสามารถของภาษาออกแบบเชิงแบบจำลองที่ได้เพิ่มความสามารถของการเขียนโปรแกรมเชิงลักษณะจึงเป็นเรื่องที่ทำได้ยาก ซึ่งเครื่องมือที่พัฒนาขึ้นมานั้นสามารถช่วยให้การวิเคราะห์และออกแบบระบบนั้นเป็นระเบียบ,จัดการระบบ และนำกลับมาใช้ใหม่ได้สะดวกขึ้น ซึ่งจะเป็นการลดภาระของผู้วิเคราะห์และออกแบบระบบ

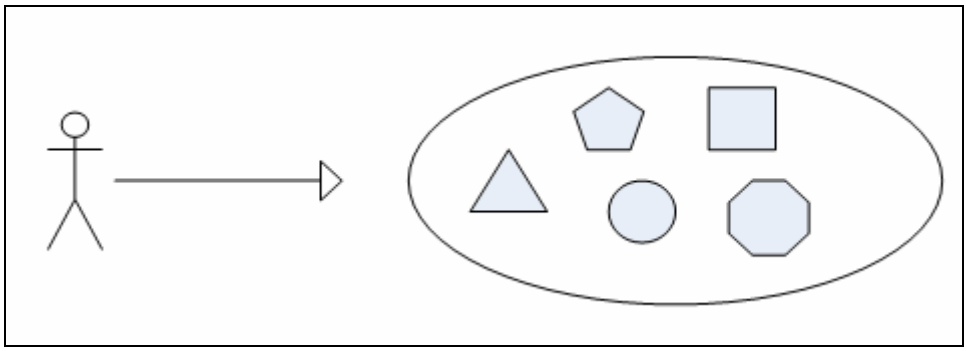
เครื่องมือที่ทำการพัฒนามีองค์ประกอบหลักอยู่ 3 ส่วน คือ ส่วนของการทำการออกแบบด้วยภาษาออกแบบเชิงแบบจำลอง ส่วนของการเปลี่ยนรูปข้อมูล และส่วนของการสร้างโค้ดทำงานจากแบบจำลอง ดังตัวอย่างที่แสดงในรูปที่ 3.5



รูปที่ 3.5 แสดงการทำงานของเครื่องมือ

3.3.1 ส่วนของการทำการออกแบบด้วยภาษาออกแบบเชิงแบบจำลอง (UML)

การทำงานของส่วนนี้จะเป็นส่วนติดต่อของเครื่องมือกับผู้วิเคราะห์และออกแบบระบบ โดยหน้าที่หลักของส่วนนี้ คือวาดแผนภาพของภาษาออกแบบเชิงแบบจำลองที่เป็นกราฟิกของสัญลักษณ์ในแบบต่าง ๆ รวมถึงความสัมพันธ์ (Relationship) ต่าง ๆ ก็จะถูกนำมาประกอบเป็นแผนภาพ (Diagram) นี้ด้วย ซึ่งรูปที่ 3.6 นั้นจะแสดงให้เห็นถึงการทำงานระหว่างผู้วิเคราะห์และออกแบบระบบกับเครื่องมือ

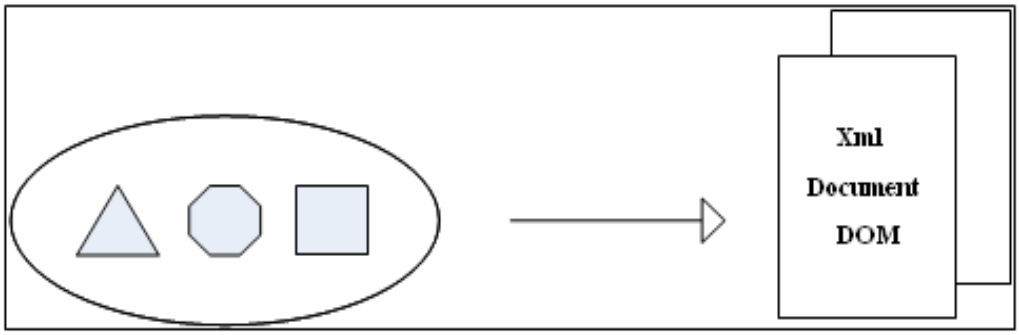


รูปที่ 3.6 แสดงการทำงานของเครื่องมือส่วนของการออกแบบ

ผู้ที่ทำการวิเคราะห์และออกแบบระบบสามารถทำการแก้ไขสัญลักษณ์ หรือความสัมพันธ์เพื่อให้เกิดการอธิบายทำงานให้ตรงกับความต้องการของระบบ โดยใช้สัญลักษณ์ที่มีอยู่สำหรับกำหนดความต้องการของระบบ

3.3.2 ส่วนของการเปลี่ยนรูปของข้อมูล (Transform Section)

ในส่วนนี้จะรับข้อมูลมาจากส่วนของการทำการออกแบบด้วยภาษาออกแบบเชิงแบบจำลอง โดยข้อมูลที่ได้มานั้นจะเป็นกราฟิกของสัญลักษณ์ในแบบต่าง ๆ รวมถึงความสัมพันธ์ (Relationship) และเมื่อการทำงานมาถึงส่วนนี้แล้วจะทำการเปลี่ยนรูปกราฟิกเหล่านั้นให้เป็นข้อมูลของเอกสารแบบการจำลองวัตถุ (DOM) ซึ่งเป็นรูปแบบที่ผู้วิเคราะห์และออกแบบระบบสามารถปรับเปลี่ยนและแก้ไขได้ง่าย เนื่องจากเป็นรูปแบบที่สามารถอ่านและเข้าใจได้โดยง่าย ซึ่งรูปที่ 3.7 จะเป็นการแสดงให้เห็นถึงการทำงานของระบบส่วนนี้



รูปที่ 3.7 แสดงการทำงานของเครื่องมือส่วนของการเปลี่ยนรูปข้อมูล

ลักษณะกราฟิก และความสัมพันธ์ต่าง ๆ ของสัญลักษณ์ของภาษาออกแบบเชิงแบบจำลองนั้นจะถูกแปลงให้อยู่ในรูปของเอกสารแบบการจำลองวัตถุ ซึ่งใช้ XML เป็นโครงสร้างหลักของการทำงาน โดยข้อมูลของเอกสารที่ได้นั้นจะถูกกำหนดให้อยู่ในรูปแบบตามมาตรฐานของเครื่องมือ ซึ่งจะมีลักษณะของโครงสร้างเป็นดังรูปที่ 3.8

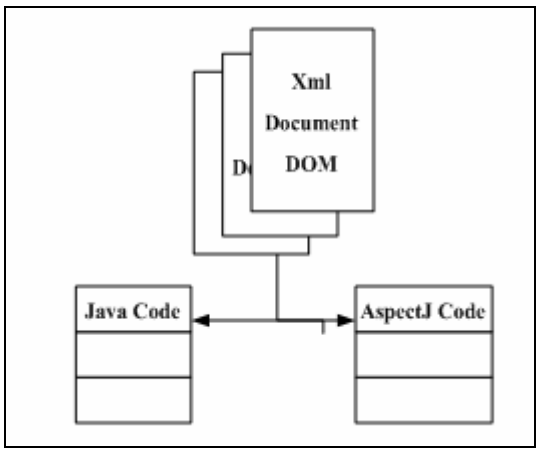
```
-<Object-oreinted>
 -<class>
   <name></name>
   -<attribute>
     -<list>
        <value></value>
     </list>
   </attribute>
   -<operation>
     -<list>
        <value></value>
     </list>
   </operation>
   -<relation>
     <type></type>
     <class></class>
   </relation>
 </class>
</Object-oriented>

-<Aspect-oriented>
  -<class>
    <name></name>
    -<joint-point>
      -<class name="">
        -<method name="" return-type="">
          -<arg>
            <value></value>
          </arg>
        </method>
      </class>
    </joint-point>
  </class>
</Aspect-oriented>
```

รูปที่ 3.8 แสดงรูปแบบของเอกสารแบบการจำลองวัตถุ

3.3.3 ส่วนของการสร้างโครงสร้างของโค้ด (Generator section)

ส่วนนี้เป็นส่วนที่ติดต่อระหว่างส่วนของการวิเคราะห์และออกแบบระบบ (System Analysis and Design) และส่วนการสร้างระบบ (Implementation) ซึ่งหน้าที่หลักของส่วนนี้จะเป็นการนำกราฟิก และความสัมพันธ์ต่าง ๆ ของสัญลักษณ์ของภาษาออกแบบเชิงแบบจำลองซึ่งถูก

แปลงเป็นเอกสารแบบการจำลองวัตถุ มาทำให้อยู่ในรูปของโครงสร้างของโค้ดเพื่อนำมาสร้างระบบ โดยโครงสร้างของโค้ดนั้นจะถูกแบ่งออกเป็น 2 ลักษณะได้แก่ โครงสร้างของการเขียนโปรแกรมเชิงวัตถุ และโครงสร้างของการเขียนโปรแกรมเชิงลักษณะ ซึ่งจะถูกแบ่งแยกกันอย่างชัดเจน โดยโครงสร้างของคลาสที่เป็นการเขียนโปรแกรมเชิงวัตถุนั้นจะถูกสร้างในลักษณะของโครงสร้างของภาษาจาวา (JAVA) และโครงสร้างของคลาสที่เป็นการเขียนโปรแกรมเชิงลักษณะนั้นถูกสร้างในลักษณะของโครงสร้างของภาษาจาวาเชิงลักษณะ (AspectJ) ซึ่งรูปที่ 3.9 จะแสดงให้เห็นถึงการทำงานของส่วนของการสร้างโครงสร้างของโค้ดนี้

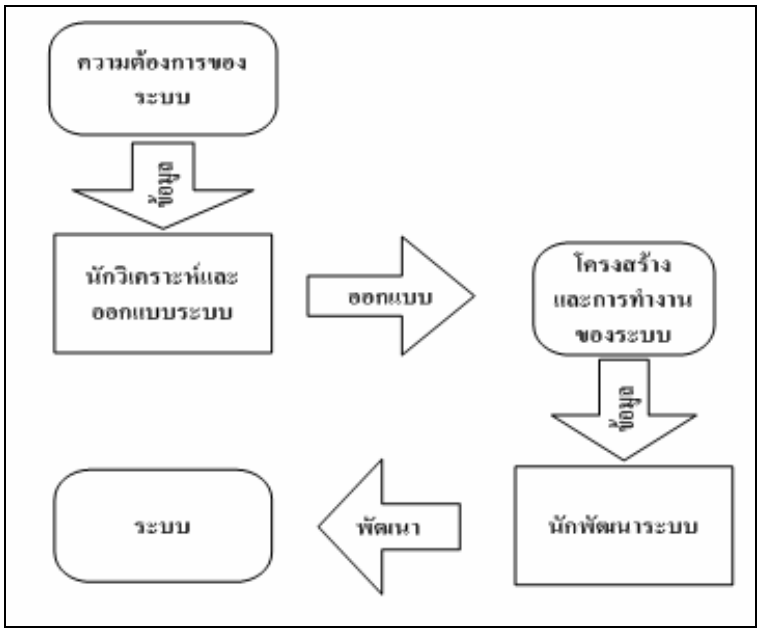


รูปที่ 3.9 แสดงการทำงานของส่วนสร้างโครงสร้าง

3.3.4 การวิเคราะห์และออกแบบเครื่องมือ

ในการวิเคราะห์และออกแบบระบบซอฟต์แวร์ ผู้ที่ทำการวิเคราะห์และออกแบบระบบต้องนำเอาความต้องการของระบบ (Requirement) มาวิเคราะห์การทำงานให้ตรงกับความต้องการของระบบ เพื่อให้การออกแบบนั้นครอบคลุมการทำงานของระบบผู้วิเคราะห์และออกแบบต้องเข้าใจถึงปัญหาของงาน และความสามารถของภาษาที่ใช้สำหรับออกแบบ เครื่องมือจะสนับสนุนทางสัญลักษณ์ของภาษาออกแบบเชิงแบบจำลอง เพื่อลดปัญหาที่อาจเกิดจากการออกแบบ และทำการสนับสนุนให้ผู้วิเคราะห์และออกแบบระบบสามารถใช้ความสามารถของภาษาออกแบบเชิงแบบจำลองได้อย่างมีประสิทธิภาพ

จากการวิเคราะห์และออกแบบเครื่องมือ ความสัมพันธ์ระหว่างส่วนต่าง ๆ ของเครื่องมือกับข้อมูลที่ใช้ และไหลเวียนอยู่ในระบบนั้นสามารถสร้างเป็นแผนภาพของกระแสการไหลของข้อมูลได้ดังรูปที่ 3.10



รูปที่ 3.10 แผนภาพกระแสข้อมูลของเครื่องมือ

จากแผนภาพกระแสข้อมูลของเครื่องมือ มีกระบวนการที่สำคัญก็คือ การออกแบบโครงสร้าง และการทำงานของระบบ และพัฒนาระบบ

การออกแบบโครงสร้าง และการทำงานของระบบเป็นส่วนที่จะนำเอาความต้องการของระบบมาสร้างเป็นโครงสร้าง และกระบวนการทำงานของระบบ และจากนั้นนำข้อมูลที่ได้จากการออกแบบไปใช้ประกอบในการสร้าง และพัฒนาระบบ

การพัฒนาระบบ ทำหน้าที่นำโครงสร้างซึ่งได้มาจากส่วนของการออกแบบโครงสร้างและพัฒนาระบบ เพื่อมาสร้างระบบ

3.4 การออกแบบ และพัฒนาเครื่องมือช่วยในการวิเคราะห์ และออกแบบภาษาออกแบบเชิงแบบจำลอง ที่เพิ่มความสามารถของการเขียนโปรแกรมเชิงลักษณะ

การออกแบบและพัฒนาเครื่องมือช่วยวิเคราะห์และออกแบบภาษาออกแบบเชิงแบบจำลองนี้แสดงถึงโครงสร้างของการทำงานที่นำเอาแนวคิดของการเพิ่มความสามารถของภาษาออกแบบเชิงแบบจำลองด้วยเทคนิคการเขียนโปรแกรมเชิงลักษณะ มาออกแบบและพัฒนา และใช้ภาษาจาวาเป็นภาษาหลักในการพัฒนา และใช้ Eclipse IDE เป็นเครื่องมือหลักในการพัฒนา และใช้งาน การออกแบบนั้นสามารถแบ่งการทำงานออกเป็น 3 ส่วนดังที่กล่าวมาแล้วในหัวข้อ 3.3 ซึ่งได้แก่

1) ส่วนของการทำการออกแบบด้วยภาษาออกแบบเชิงแบบจำลอง

2) ส่วนของการเปลี่ยนรูปข้อมูล

3) ส่วนของการสร้างโค้ดทำงานจากแบบจำลอง

การออกแบบนี้นั้นจะแสดงเป็นข้อมูลของคลาส ซึ่งในแต่ละคลาสนั้นจะถูกแสดงอยู่ในรูปแบบโครงสร้าง และอธิบายรายละเอียดการทำงานที่สำคัญของแต่ละคลาส

3.4.1 คลาส AspectModel

คลาส AspectModel เป็นคลาสที่ทำหน้าที่กำหนดลักษณะของคลาสเชิงลักษณะ (Aspect Class) โดย รายละเอียดของลักษณะดังกล่าวนั้นจะปรากฏอยู่ในตารางที่ 3.1

ตารางที่ 3.1 รายละเอียดข้อมูลของคลาส AspectModel

ชื่อข้อมูล	ชนิดข้อมูล	ค่าเริ่มต้น	หมายเหตุ
name	String	ไม่มี	ชื่อของคลาสเชิงลักษณะ
isAbstract	boolean	false	บอกลักษณะของคลาสเชิงลักษณะว่าเป็นนามธรรมหรือไม่

การทำงานของคลาส AspectModel นี้จะเป็นการแสดงให้เห็นถึง โครงสร้างและรูปแบบของคลาสเชิงลักษณะ

3.4.2 คลาส JointPointModel

คลาส JointPointModel เป็นคลาสทีทำหน้าที่คอยกำหนดพฤติกรรมการทำงานของคลาสเชิงลักษณะ โดยรายละเอียดการทำงานของพฤติกรรมของคลาสนั้นจะปรากฏอยู่ที่ตารางที่ 3.2

ตารางที่ 3.2 รายละเอียดข้อมูลของคลาส JointPointModel

ชื่อข้อมูล	ชนิดข้อมูล	ค่าเริ่มต้น	หมายเหตุ
name	String	ไม่มี	ชื่อของจุดตัดของคลาส
isAbstract	boolean	false	บอกลักษณะของจุดตัดระหว่างคลาสทั้ง 2 อย่างว่าเป็นนามธรรมหรือไม่

ตารางที่ 3.2 รายละเอียดข้อมูลของคลาส JointPointModel (ต่อ)

ชื่อข้อมูล	ชนิดข้อมูล	ค่าเริ่มต้น	หมายเหตุ
callMethod	String	ไม่มี	เก็บชื่อของเมธอดที่ทำงานบนจุดตัดนี้
type	String	void	ลักษณะการคืนค่าของเมธอดที่ทำงานบนจุดตัดนี้

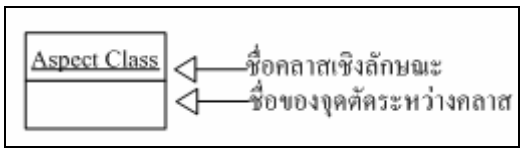
การทำงานของคลาส JointPointModel เป็นการทำงานโดยการกำหนดค่าของจุดตัดระหว่างคลาสเชิงลักษณะ กับคลาสเชิงวัตถุ

3.4.3 คลาส JointModel

คลาส JointModel เป็นคลาสที่ใช้กำหนดลักษณะของความสัมพันธ์แบบจุดตัด (Joint Relation) ระหว่างคลาสเชิงวัตถุและคลาสเชิงลักษณะ โดยความสัมพันธ์นี้เป็นความสัมพันธ์ระหว่างคลาสเชิงวัตถุ และคลาสเชิงลักษณะเท่านั้น ไม่สามารถมีความสัมพันธ์แบบจุดตัดระหว่างคลาสเชิงวัตถุกับคลาสเชิงวัตถุ หรือ คลาสเชิงลักษณะกับคลาสเชิงลักษณะได้ ซึ่งคลาสนี้เป็นคลาสที่คอยกำหนดพฤติกรรมการทำงานเท่านั้น จึงไม่จำเป็นต้องมีข้อมูลของคลาส ดังนั้นจึงไม่มีตารางมาอธิบายรายละเอียดของคลาส JointModel นี้

3.4.4 คลาส AspectClassFigure

คลาส AspectClassFigure เป็นคลาสใช้สำหรับการกำหนดลักษณะของคลาสเชิงลักษณะที่แสดงเป็นผลแบบกราฟิก โดยแสดงให้เห็นถึงชื่อของคลาสเชิงลักษณะ และชื่อของจุดตัดซึ่งมีลักษณะดังรูปที่ 3.11



รูปที่ 3.11 ลักษณะของคลาสเชิงลักษณะ

โดยรายละเอียด และลักษณะโครงสร้างการทำงานของคลาส AspectClassFigure นี้ปรากฏอยู่ในตารางที่ 3.3

ตารางที่ 3.3 รายละเอียดข้อมูลของคลาส AspectClassFigure

ชื่อข้อมูล	ชนิดข้อมูล	ค่าเริ่มต้น	หมายเหตุ
name	String	ไม่มี	ชื่อที่แสดงบนกราฟิกของคลาสเชิงลักษณะ
icon	Image	ไม่มี	รูปลัญลักษณ์ของคลาสเชิงลักษณะ

3.4.5 คลาส AspectOperationLabel

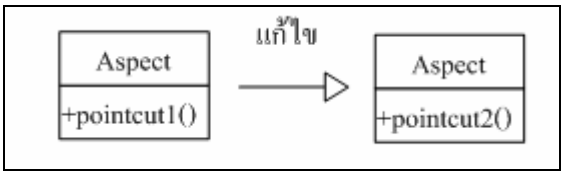
คลาส AspectOperationLable เป็นคลาสที่ใช้ควบคุมการทำงานของลักษณะของตัวหนังสือที่ปรากฏอยู่ในคลาสเชิงลักษณะดังรูปที่ 3.13 คลาส AspectOperationLable จะทำงานและแสดงผลร่วมกันกับคลาส AspectClassFigure ลักษณะและรายละเอียดข้อมูลของคลาส AspectOperationLable นั้นแสดงอยู่ในตารางที่ 3.4

ตารางที่ 3.4 รายละเอียดข้อมูลของคลาส AspectOperationLable

ชื่อข้อมูล	ชนิดข้อมูล	ค่าเริ่มต้น	หมายเหตุ
underline	boolean	false	กำหนดว่าลักษณะของตัวหนังสือนั้นต้องมีการขีดเส้นใต้ไว้หรือไม่
icon	Image	ไม่มี	รูปของเมธอดของคลาสเชิงลักษณะ

3.4.6 คลาส JointPointEditPart

คลาส JointPointEdirPart เป็นคลาสที่ไว้สำหรับทำงานส่วนที่ทำการแก้ไข ซึ่งอาจเกิดข้อผิดพลาดขึ้นได้ในขณะสร้างจุดตัดบนคลาสเชิงลักษณะ โดยหน้าที่หลักของส่วนนี้คือรับข้อมูลที่ใส่เข้าไปให้ใหม่ และแสดงผลดังรูปที่ 3.12



รูปที่ 3.12 แสดงการทำงานของคลาส JointPointEditPart

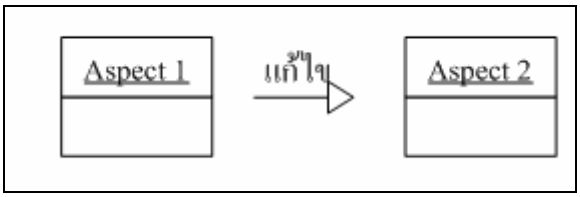
ซึ่งคลาส JointPointEdirPart จะทำการแสดงผลข้อมูลทุกอย่างที่ทำงานบนส่วนของจุดตัดโดยจะทำงานร่วมกันกับคลาส AspectOperationLable ซึ่งจัดการรูปแบบของอักษร ตารางที่ 3.5 แสดงถึงข้อมูลภายในคลาส JointPointEdirPart

ตารางที่ 3.5 รายละเอียดข้อมูลของคลาส JointPointEdirPart

ชื่อข้อมูล	ชนิดข้อมูล	ค่าเริ่มต้น	หมายเหตุ
normalic	Font	ไม่มี	กำหนดว่าลักษณะตัวหนังสือของจุดตัด
italic	Font	ไม่มี	กำหนดว่าลักษณะตัวหนังสือของจุดตัด

3.4.7 คลาส AspectEditPart

คลาส AspectEditPart เป็นคลาสที่ไว้สำหรับทำงานส่วนที่ทำการแก้ไข ซึ่งอาจเกิดข้อผิดพลาดขึ้นได้ในขณะสร้างคลาสเชิงลักษณะ โดยหน้าที่หลักของส่วนนี้คือรับข้อมูลที่ใส่เข้าไปให้ใหม่ และแสดงผลดังรูปที่ 3.13



รูปที่ 3.13 แสดงการทำงานของคลาส AspectEditPart

โดยคลาส AspectEditPart นี้จะเป็นตัวรับข้อมูลจากคลาส AspectClassFigure และ คลาส JointPointEditPart มาแสดงผลทั้งรูปและตัวอักษร ซึ่งรายละเอียดและลักษณะการทำงานของคลาสของ AspectEditPart แสดงอยู่ในตารางที่ 3.6

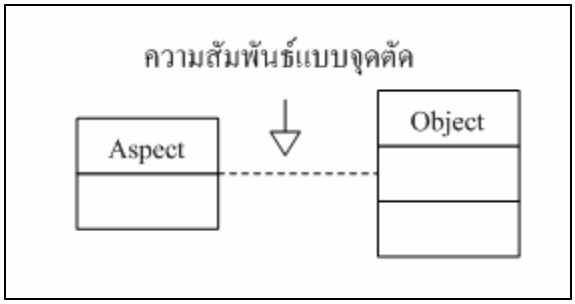
ตารางที่ 3.6 รายละเอียดข้อมูลของคลาส AspectEditPart

ชื่อข้อมูล	ชนิดข้อมูล	ค่าเริ่มต้น	หมายเหตุ
normalic	Font	ไม่มี	กำหนดว่าลักษณะตัวหนังสือคลาสเชิงลักษณะ
italic	Font	ไม่มี	กำหนดว่าลักษณะตัวหนังสือคลาสเชิงลักษณะ

การทำงานของคลาส AspectEditPart นั้นอยู่ในรูปแบบของกระบวนการภายในระบบ โดยนำข้อมูลจากคลาสอื่น ๆ มาประกอบเข้าด้วยกัน และนำไปแสดงผล

3.4.8 คลาส JointConnectionFigure

คลาส JointConnectionFigure เป็นคลาสที่ทำงานในส่วนแสดงลักษณะเส้นความสัมพันธ์แบบจุดตัด (Joint Relation) ซึ่งจะมีลักษณะเป็นเส้นประ และใช้แสดงความสัมพันธ์ระหว่างคลาสเชิงลักษณะและคลาสเชิงวัตถุ ซึ่งรูปที่ 3.14 แสดงให้เห็นถึงลักษณะของเส้นความสัมพันธ์แบบจุดตัด และความสัมพันธ์แบบจุดตัดระหว่างทั้ง 2 คลาส



รูปที่ 3.14 แสดงความสัมพันธ์แบบจุดตัดระหว่างคลาสเชิงลักษณะและคลาสเชิงวัตถุ

คลาส JointConnectionFigure และคลาส JointModel เป็นการนำแนวคิดของการเพิ่มความสามารถของภาษาออกแบบเชิงแบบจำลองด้วยเทคนิคการเขียนโปรแกรมเชิงลักษณะ มาสร้างเป็นรูปธรรม ซึ่งการทำงานในส่วนนี้จะเป็นการเพิ่มเติมความสามารถของความสัมพันธ์ระหว่างคลาส

3.4.9 คลาส JointEditPart

คลาส JointEditPart เป็นคลาสที่คอยทำหน้าที่แก้ไขและปรับเปลี่ยนให้เส้นของความสัมพันธ์ที่แสดงออกมานั้นถูกต้อง เมื่อมีการเคลื่อนย้ายคลาสเชิงลักษณะ หรือคลาสเชิงวัตถุ

3.4.10 คลาส AssociationModel

คลาส AssociationModel เป็นคลาสที่ทำหน้าที่กำหนดลักษณะความสัมพันธ์แบบเกี่ยวพันกัน (Associate) โดยความสัมพันธ์นี้จะเป็นความสัมพันธ์ของมาตรฐานเดิมของภาษาออกแบบเชิงแบบจำลอง คือ ความสัมพันธ์ระหว่างคลาสเชิงวัตถุกับคลาสเชิงวัตถุ โดยข้อกำหนดและรายละเอียดของข้อมูลภายในคลาสนั้นแสดงอยู่ในตารางที่ 3.7

ตารางที่ 3.7 รายละเอียดข้อมูลของคลาส AssociationModel

ชื่อข้อมูล	ชนิดข้อมูล	ค่าเริ่มต้น	หมายเหตุ
stereoType	String	ไม่มี	สำหรับกำหนดลักษณะและอธิบายความสัมพันธ์
fromMultiplicity	String	ไม่มี	กำหนดรูปแบบของความสัมพันธ์จากคลาสหนึ่งมายังคลาสหนึ่ง
toMultiplicity	String	ไม่มี	กำหนดรูปแบบของความสัมพันธ์จากคลาสหนึ่งไปยังคลาสหนึ่ง

รูปแบบของความสัมพันธ์จากคลาสเชิงวัตถุหนึ่งไปยังอีกคลาสเชิงวัตถุอีกอันหนึ่งนั้น เป็นการบ่งบอกถึงความเกี่ยวข้องกันของคลาสว่ามีความเกี่ยวข้องกันอย่างไรโดยรูปแบบของความสัมพันธ์นั้นประกอบไปด้วยลักษณะดังนี้

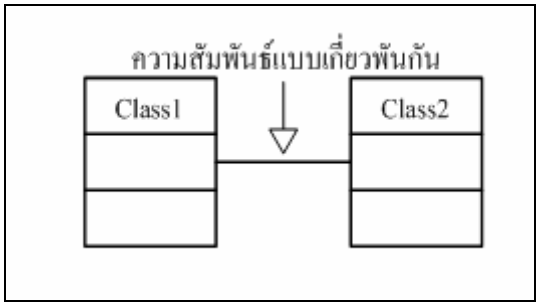
- Zero
- One
- Zero or One

- One or More
- Many

โดยการนำความสัมพันธ์มาใช้เพื่ออธิบายการทำงานของระบบในขั้นตอนการวิเคราะห์และออกแบบระบบ

3.4.11 คลาส AssociationConnectionFigure

คลาส AssociationConnectionFigure เป็นคลาสที่ทำงานในส่วนแสดงลักษณะเส้นความสัมพันธ์แบบเกี่ยวพันกัน (Association Relation) โดยจะมีลักษณะเป็นเส้นตรง และใช้แสดงความสัมพันธ์ระหว่างคลาสเชิงวัตถุกับคลาสเชิงวัตถุ ซึ่งรูปที่ 3.15 แสดงให้เห็นถึงลักษณะของเส้นความสัมพันธ์แบบจุดตัด



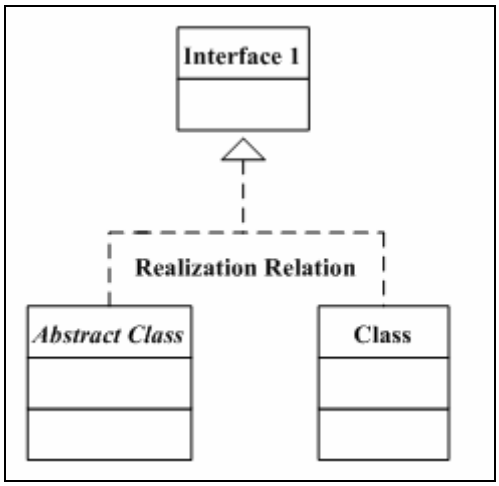
รูปที่ 3.15 แสดงความสัมพันธ์แบบเกี่ยวพันกัน

3.4.12 คลาส RealizationModel

คลาส RealizationModel เป็นคลาสที่ทำหน้าที่กำหนดลักษณะรูปแบบความสัมพันธ์ระหว่างอินเตอร์เฟส (Interface) กับคลาสที่เป็นนามธรรม (Abstract Class) หรือคลาสเชิงวัตถุทั่วไป

3.4.13 คลาส RealizationConnectionFigure

คลาส RealizationConnectionFigure เป็นคลาสที่แสดงลักษณะที่ถูกกำหนดของความสัมพันธ์ระหว่างอินเตอร์เฟส (Interface) กับคลาสที่เป็นนามธรรม (Abstract Class) หรือคลาสเชิงวัตถุทั่วไป โดยรูปที่ 3.16 แสดงให้เห็นถึงความสัมพันธ์แบบ Realization



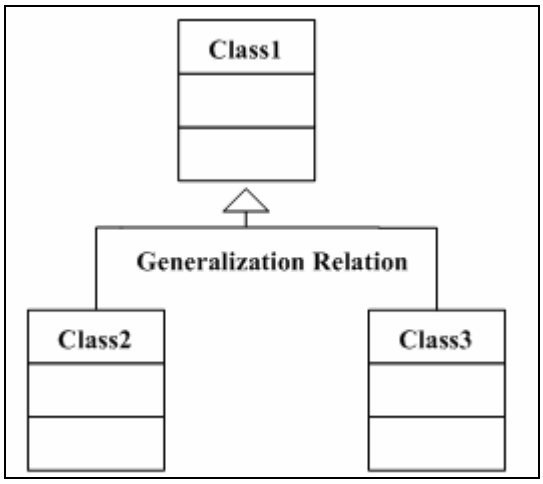
รูปที่ 3.16 แสดงความสัมพันธ์แบบ Realization

3.4.14 คลาส GeneralizationModel

คลาส GeneralizationModel เป็นคลาสที่ทำหน้าที่กำหนดลักษณะของความสัมพันธ์ระหว่างคลาส (Class) กับคลาสที่เป็นคลาสย่อยหรือคลาสลูก (Sub Class)

3.4.15 คลาส GeneralizationConnectionFigure

คลาส GeneralizationConnectionFigure เป็นคลาสที่แสดงลักษณะที่ถูกกำหนดรูปแบบความสัมพันธ์ระหว่างคลาส (Class) กับคลาสที่เป็นคลาสย่อยหรือคลาสลูก (Sub Class) โดยรูปที่ 3.17 แสดงให้เห็นถึงความสัมพันธ์แบบ Generalization



รูปที่ 3.17 แสดงความสัมพันธ์แบบ Generalization

3.4.16 คลาส ModelGeneratorDelegation

คลาส ModelGeneratorDelegation เป็นคลาสที่ทำหน้าที่เป็นตัวแทนติดต่อระหว่างผู้วิเคราะห์และออกแบบระบบกับการทำงานของเครื่องมือในส่วนของการแปลงข้อมูล (Generation Section) โดยการทำงานของคลาสนี้จะทำให้กราฟิกสัญลักษณ์ของภาษาออกแบบเชิงแบบจำลองที่เพิ่มเติมความสามารถของการเขียนโปรแกรมเชิงลักษณะกลายไปเป็นข้อมูลของเอกสารแบบการจำลองวัตถุซึ่งรายละเอียดข้อมูลภายในคลาสแสดงอยู่ในตารางที่ 3.8

ตารางที่ 3.8 รายละเอียดข้อมูลของคลาส ModelGeneratorDelegation

ชื่อข้อมูล	ชนิดข้อมูล	ค่าเริ่มต้น	หมายเหตุ
attr_key	int	ศูนย์	เก็บข้อมูลจำนวนของคุณสมบัติของคลาสเชิงวัตถุจากกราฟิกของภาษาออกแบบเชิงแบบจำลอง
opt_key	int	ศูนย์	เก็บข้อมูลจำนวนของเมธอดของคลาสเชิงวัตถุจากกราฟิกของภาษาออกแบบเชิงแบบจำลอง
point_key	int	ศูนย์	เก็บข้อมูลจำนวนของฟังก์ชั่นของคลาสเชิงลักษณะจากกราฟิกของภาษาออกแบบเชิงแบบจำลอง
oo_root	Element	ไม่มี	เก็บข้อมูลทั้งหมดของคลาสเชิงวัตถุจากกราฟิกของภาษาออกแบบเชิงแบบจำลอง ให้อยู่ในรูปของเอกสารแบบการจำลองวัตถุ
ao_root	Element	ไม่มี	เก็บข้อมูลทั้งหมดของคลาสเชิงลักษณะจาก

ตารางที่ 3.8 รายละเอียดข้อมูลของคลาส ModelGeneratorDelegation (ต่อ)

ชื่อข้อมูล	ชนิดข้อมูล	ค่าเริ่มต้น	หมายเหตุ
			กราฟิกของภาษาออกแบบเชิงแบบจำลองให้อยู่ในรูปของเอกสารแบบการจำลองวัตถุ
selection	ISelection	ไม่มี	สำหรับเก็บข้อมูลของทุกไฟล์ในเครื่องมือ

3.4.17 คลาส DocumentGeneratorDelegation

คลาส DocumentGeneratorDelegation เป็นคลาสที่ทำหน้าที่แปลงข้อมูลจากเอกสารแบบการจำลองวัตถุ มาทำให้อยู่ในรูปแบบของคลาส ซึ่งจะนำไปใช้ในกระบวนการพัฒนาระบบซอฟต์แวร์ โดยรายละเอียดข้อมูลภายในคลาสแสดงอยู่ในตารางที่ 3.9

ตารางที่ 3.9 รายละเอียดข้อมูลของคลาส DocumentGeneratorDelegation

ชื่อข้อมูล	ชนิดข้อมูล	ค่าเริ่มต้น	หมายเหตุ
path	String	ไม่มี	สำหรับเก็บข้อมูลที่ตั้งของไฟล์กราฟิกสัญลักษณ์ของภาษาออกแบบเชิงแบบจำลอง
package_name	String	ไม่มี	สำหรับเก็บข้อมูลของโฟลเดอร์สำหรับเก็บไฟล์เอกสารแบบการจำลองวัตถุ
selection	ISelection	ไม่มี	สำหรับเก็บข้อมูลของทุกไฟล์ในเครื่องมือ

3.4.18 คลาส DocumentToObject

เก็บไว้ และนำไปสร้างเป็นโค้ดของการพัฒนาโปรแกรมเชิงวัตถุ เพื่อนำไปใช้ในกระบวนการพัฒนาในส่วนถัดไป ตารางที่ 3.10 เป็นรายละเอียดข้อมูลของคลาส

ตารางที่ 3.10 รายละเอียดข้อมูลของคลาส DocumentToObject

ชื่อข้อมูล	ชนิดข้อมูล	ค่าเริ่มต้น	หมายเหตุ
name	String	ไม่มี	ข้อมูลที่เป็นชื่อของคลาสเชิงวัตถุ
type	String	ไม่มี	ข้อมูลที่เป็นลักษณะของคลาส
operation	List	ไม่มี	ข้อมูลของเมธอดทุก ๆ ตัว โดยแต่ละตัวถูกเก็บอยู่ในรูปของข้อมูลคลาส OperationDocument
attribute	List	ไม่มี	ข้อมูลของคุณสมบัติทุก ๆ ตัว โดยแต่ละตัวถูกเก็บอยู่ในรูปของข้อมูลคลาส AttributeDocument
inherited	List	ไม่มี	ข้อมูลของคลาสแม่ทุก ๆ ตัว โดยแต่ละตัวนั้นถูกเก็บอยู่ในรูปของข้อมูลคลาส InheritedDocument
relationType	List	ไม่มี	ข้อมูลของลักษณะของความสัมพันธ์ทุก ๆ ตัว
relationTo	List	ไม่มี	ข้อมูลความสัมพันธ์จากคลาส ไปยังอีกคลาส

3.4.19 คลาส DocumentToAspect

คลาส DocumentToAspect เป็นคลาสที่ทำหน้าที่เก็บข้อมูล โดยได้จากการนำเอาเอกสารแบบการจำลองวัตถุของคลาสเชิงลักษณะมาทำการสกัดข้อมูลซึ่งถูกเก็บไว้ตามคลาสต่าง ๆ มาเก็บไว้ และนำไปสร้างเป็นโค้ดของการพัฒนาโปรแกรมเชิงลักษณะ เพื่อนำไปใช้ในกระบวนการพัฒนาในส่วนถัดไป ตารางที่ 3.11 เป็นรายละเอียดข้อมูลของคลาส

ตารางที่ 3.11 รายละเอียดข้อมูลของคลาส DocumentToAspect

ชื่อข้อมูล	ชนิดข้อมูล	ค่าเริ่มต้น	หมายเหตุ
name	String	ไม่มี	ข้อมูลที่เป็นชื่อของคลาสเชิงลักษณะ
callMethod	เอกสารข้อมูลคลาส PointCutDocument	ไม่มี	ข้อมูลของเมธอดของคลาสเชิงลักษณะ โดยถูกเก็บอยู่ในรูปของข้อมูลคลาส PointCutDocument
callClass	String	ไม่มี	ข้อมูลที่เป็นชื่อของคลาสซึ่งถูกคลาสเชิงลักษณะนั้นไปทำงานด้วย
return_type	String	ไม่มี	ลักษณะของข้อมูลที่ถูกส่งกลับจากเมธอดของคลาสเชิงลักษณะ

3.4.20 คลาส DocumentModel

คลาส DocumentModel เป็นคลาสที่บอกลักษณะของเอกสาร ซึ่งอยู่ในรูปของเอกสารกลาง โดยทั้งคลาส DocumentToAspect และคลาส DocumentToObject ต้องนำเอาลักษณะของคลาสนี้ไปเป็นบรรทัดฐานเพื่อจะสามารถนำมาสร้างเป็นโค้ดของการเขียนโปรแกรมเชิงวัตถุและเชิงลักษณะ

3.4.21 คลาส AttributeDocument

คลาส AttributeDocument เป็นคลาสที่ทำหน้าที่เก็บข้อมูลที่ได้จากการนำเอาเอกสารแบบการจำลองวัตถุมาทำการสกัดข้อมูลของคุณสมบัติออกมา และนำมาเก็บไว้ เพื่อนำไปสร้างเป็นโค้ดเพื่อใช้ในการสร้างโค้ดเพื่อใช้ในส่วนต่อไป ตารางที่ 3.12 เป็นรายละเอียดข้อมูลของคลาส

ตารางที่ 3.12 รายละเอียดข้อมูลของคลาส AttributeDocument

ชื่อข้อมูล	ชนิดข้อมูล	ค่าเริ่มต้น	หมายเหตุ
name	String	ไม่มี	ข้อมูลที่เป็นชื่อของคุณสมบัติ

ตารางที่ 3.12 รายละเอียดข้อมูลของคลาส AttributeDocument (ต่อ)

ชื่อข้อมูล	ชนิดข้อมูล	ค่าเริ่มต้น	หมายเหตุ
varibleType	String	ไม่มี	ข้อมูลส่วนที่เป็นชนิดของคุณสมบัติ
accessType	String	ไม่มี	ข้อมูลส่วนที่เป็นลักษณะการเข้าถึงคุณสมบัติ

3.4.22 คลาส OperationDocument

คลาส OperationDocument เป็นคลาสที่ทำหน้าที่เก็บข้อมูลที่ได้จากการนำเอาเอกสารแบบการจำลองวัตถุมาทำการสกัดข้อมูลของเมธอดของคลาสเชิงวัตถุออกมา และนำมาเก็บไว้ เพื่อนำไปสร้างเป็นโค้ดเพื่อใช้ในการสร้างโค้ดเพื่อใช้ในส่วนต่อไป ตารางที่ 3.13 เป็นรายละเอียดข้อมูลของคลาส

ตารางที่ 3.13 รายละเอียดข้อมูลของคลาส OperationDocument

ชื่อข้อมูล	ชนิดข้อมูล	ค่าเริ่มต้น	หมายเหตุ
name	String	ไม่มี	ข้อมูลส่วนที่เป็นชื่อของเมธอด
returnType	String	ไม่มี	ข้อมูลส่วนที่เป็นลักษณะของการคืนค่ากลับของเมธอด
accessType	String	ไม่มี	ข้อมูลส่วนที่เป็นลักษณะของการเข้าถึงของเมธอด
argName	String	ไม่มี	ข้อมูลส่วนที่เป็นชื่อของค่าที่รับเข้ามาในเมธอด
argType	String	ไม่มี	ข้อมูลส่วนที่เป็นลักษณะและชนิดของค่าที่รับเข้ามาในเมธอด

3.4.23 คลาส PointCutDocument

คลาส PointCutDocument เป็นคลาสที่ทำหน้าที่เก็บข้อมูลที่ได้จากการนำเอาเอกสารแบบการจำลองวัตถุมาทำการสกัดข้อมูลของเมธอดของคลาสเชิงลักษณะออกมา และนำมาเก็บไว้ เพื่อนำไปสร้างเป็นโค้ดเพื่อใช้ในการสร้างโค้ดเพื่อใช้ในส่วนต่อไป ตารางที่ 3.14 เป็นรายละเอียดข้อมูลของคลาส

ตารางที่ 3.14 รายละเอียดข้อมูลของคลาส PointCutDocument

ชื่อข้อมูล	ชนิดข้อมูล	ค่าเริ่มต้น	หมายเหตุ
name	String	ไม่มี	ข้อมูลส่วนที่เป็นชื่อของเมธอดของคลาสเชิงลักษณะ
returnType	String	ไม่มี	ข้อมูลส่วนที่เป็นลักษณะของการคืนค่ากลับของเมธอดของคลาสเชิงลักษณะ
methodCall	String	ไม่มี	ข้อมูลส่วนที่เป็นชื่อของคลาสที่ถูกเมธอดของคลาสเชิงลักษณะเรียกใช้

3.4.24 คลาส InheritedDocument

คลาส InheritedDocument เป็นคลาสที่บอกลักษณะเอกสารของความสัมพันธ์แบบการสืบทอด ซึ่งอยู่ในรูปของความสัมพันธ์กลาง โดยทั้งคลาส GeneralizeDocument และคลาส RealizeDocument ต้องนำเอาลักษณะของคลาสนี้ไปเป็นบรรทัดฐานเพื่อจะนำไปใช้สร้างเอกสารของความสัมพันธ์ระหว่างคลาส

3.4.25 คลาส GeneralizeDocument

คลาส GeneralizeDocument เป็นคลาสที่บอกลักษณะเอกสารของความสัมพันธ์แบบคลาส กับคลาสย่อย (Sub Class) ตารางที่ 3.15 เป็นรายละเอียดข้อมูลของคลาส

ตารางที่ 3.15 รายละเอียดข้อมูลของคลาส GeneralizeDocument

ชื่อข้อมูล	ชนิดข้อมูล	ค่าเริ่มต้น	หมายเหตุ
name	String	ไม่มี	ข้อมูลส่วนที่เป็นชื่อของความสัมพันธ์

3.4.26 คลาส RealizaionDocument

คลาส RealizaionDocument เป็นคลาสที่บอกลักษณะเอกสารของความสัมพันธ์แบบคลาส กับอินเตอร์เฟส ตารางที่ 3.16 เป็นรายละเอียดข้อมูลของคลาส

ตารางที่ 3.16 รายละเอียดข้อมูลของคลาส RealizeDocument

ชื่อข้อมูล	ชนิดข้อมูล	ค่าเริ่มต้น	หมายเหตุ
name	String	ไม่มี	ข้อมูลส่วนที่เป็นชื่อของความสัมพันธ์

3.4.27 คลาส Visiblility

คลาส Visibility ทำหน้าที่กำหนดลักษณะของการเข้าถึงข้อมูลทุก ๆ อย่างภายในคลาส ซึ่งจะประกอบไปด้วยการเข้าถึง 4 ลักษณะ ได้แก่

- public
- protected
- private
- package

3.4.28 คลาส JarResources

คลาส JarResources เป็นคลาสที่ทำหน้าที่หารูปแบบของ (Template) โค้ดที่กำหนดไว้ในไฟล์เอกสารจาวา (Java Archive, Jar) เพื่อนำมาสร้างเป็นโครงสร้างของโค้ดของคลาสเชิงวัตถุ และคลาสเชิงลักษณะ โดยข้อมูลภายในคลาสแสดงอยู่ในตารางที่ 3.17

ตารางที่ 3.17 รายละเอียดข้อมูลของคลาส JarResources

ชื่อข้อมูล	ชนิดข้อมูล	ค่าเริ่มต้น	หมายเหตุ
jarFileName	String	ไม่มี	ข้อมูลส่วนที่เป็นชื่อของไฟล์เอกสารภาษาจาวา
htJarContents	กลุ่มของข้อมูล	ไม่มี	เก็บไฟล์เอกสารของภาษาจาวาทุก ๆ ไฟล์
htSize	กลุ่มของข้อมูล	ไม่มี	เก็บขนาดของไฟล์เอกสารของภาษาจาวาทุก ๆ ไฟล์

3.4.29 คลาส AssociationEditPart

คลาส AssociationEditPart เป็นคลาสที่ทำหน้าที่แก้ไขและปรับเปลี่ยนให้เส้นของความสัมพันธ์ที่เกี่ยวพันกันนั้นแสดงออกมาอย่างถูกต้อง เมื่อมีการเคลื่อนย้ายคลาสเชิงคลาสเชิงวัตถุ

3.4.30 คลาส GeneralizationEditpart

คลาส GeneralizationEditpart เป็นคลาสที่ทำหน้าที่แก้ไขและปรับเปลี่ยนให้เส้นของความสัมพันธ์ระหว่างคลาสกับคลาสย่อย (Sub Class) นั้นแสดงออกมาอย่างถูกต้อง เมื่อมีการเคลื่อนย้ายคลาสเชิงคลาสเชิงวัตถุ หรือคลาสเชิงลักษณะ

3.4.31 คลาส RealizaionEditPart

คลาส RealizaionEditPart เป็นคลาสที่ทำหน้าที่แก้ไขและปรับเปลี่ยนให้เส้นของความสัมพันธ์ระหว่างคลาสกับอินเตอร์เฟส (Interface) นั้นแสดงออกมาอย่างถูกต้อง เมื่อมีการเคลื่อนย้ายคลาสเชิงคลาสเชิงวัตถุ หรือคลาสเชิงลักษณะ

3.4.32 คลาส RoleModel

คลาส RoleModel ทำหน้าที่กำหนดลักษณะของบทบาทของความสัมพันธ์แบบเกี่ยวพันกัน (Association Relation) ระหว่างคลาสเชิงวัตถุ

3.5 การรวบรวมการ ทำงานของส่วนแสดงผลกราฟิกและการออกแบบส่วนติดต่อกับผู้ใช้งาน

เป็นการรวบรวมการทำงานของส่วนต่าง ๆ ทั้งหมดในระบบเพื่อมาทำงานร่วมกันและรวมถึงการนำข้อมูลจากภายนอกเข้ามาภายในระบบด้วย โดยสามารถแยกออกแบบ 2 ส่วน คือส่วนแสดงผลกราฟิก และ ส่วนติดต่อกับผู้ใช้งาน

3.5.1 ส่วนแสดงผลกราฟิก

ส่วนแสดงผลกราฟิกนี้เป็นการนำเอาคลาสที่ได้ออกแบบ และพัฒนาจากหัวข้อที่ 3.4 นั้นมารวมเข้าด้วยกัน เพื่อใช้ในการสนับสนุนและทำงานร่วมกัน โดยจะถูกรวมเข้าเป็นคลาสที่ใช้สำหรับการออกแบบแผนภาพของภาษาออกแบบเชิงแบบจำลองที่ได้เพิ่มความสามารถของการเขียนโปรแกรมเชิงลักษณะลงไป ซึ่งเรียกว่าแผนภาพคลาสเชิงลักษณะ (Aspect Class Diagram) โดยแผนภาพดังกล่าวนี้จะสนับสนุนการทำงานของภาษาออกแบบเชิงแบบจำลองในส่วนที่เป็นการเขียนโปรแกรมเชิงวัตถุด้วย

1) คลาส AspectClassDiagramEditor

คลาส AspectClassDiagramEditor ทำหน้าที่รวบรวมคลาสอื่น ๆ เข้ามาไว้ด้วยกัน เพื่อทำงานร่วมกัน โดยตารางที่ 3.18 รายละเอียดข้อมูลของคลาส AspectClassDiagramEditor

ตารางที่ 3.18 ข้อมูลของคลาส AspectClassDiagramEditor

ชื่อข้อมูล	ชนิดข้อมูล	ค่าเริ่มต้น	หมายเหตุ
addAttributeAction	ข้อมูลคลาส Attibute	ไม่มี	ข้อมูลส่วนที่เป็นการทำงานของคุณสมบัติ
addOperationAction	ข้อมูลคลาส Operation	ไม่มี	ข้อมูลส่วนที่เป็นการทำงานของเมธอด
addAspectOperationAction	ข้อมูลคลาส Operation	ไม่มี	ข้อมูลส่วนที่เป็นการทำงานของเมธอดของคลาสเชิงลักษณะ

3.5.2 ส่วนติดต่อกับผู้ใช้งาน

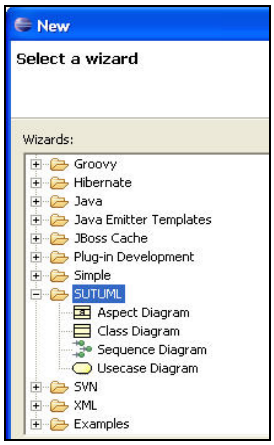
ส่วนติดต่อกับผู้ใช้งานนี้ มีหน้าจอหลักอยู่ 5 ส่วน คือ หน้าจอเมนูหลัก, หน้าจอการสร้างแผนภาพ, หน้าจอการวาดสัญลักษณ์กราฟิกสัญลักษณ์ของภาษาออกแบบเชิงแบบจำลอง, หน้าจอการแปลงรูปสัญลักษณ์กราฟิกสัญลักษณ์ของภาษาออกแบบเชิงแบบจำลองให้เป็นเอกสารแบบการจำลองวัตถุและหน้าจอการแปลงรูปของเอกสารแบบการจำลองวัตถุให้อยู่เป็นโครงสร้างโค้ด

1) หน้าจอเมนูหลัก

ส่วนนี้เป็นส่วนเริ่มต้นการใช้งานเครื่องมือ โดยเครื่องมือนี้จะอยู่ในรูปแบบของปลักอิน (Plug-in) ของ Eclipse IDE ซึ่งจะทำงานโดยอาศัยทรัพยากรจาก Eclipse ดังรูป 3.18

รูปที่ 3.18 Eclipse IDE

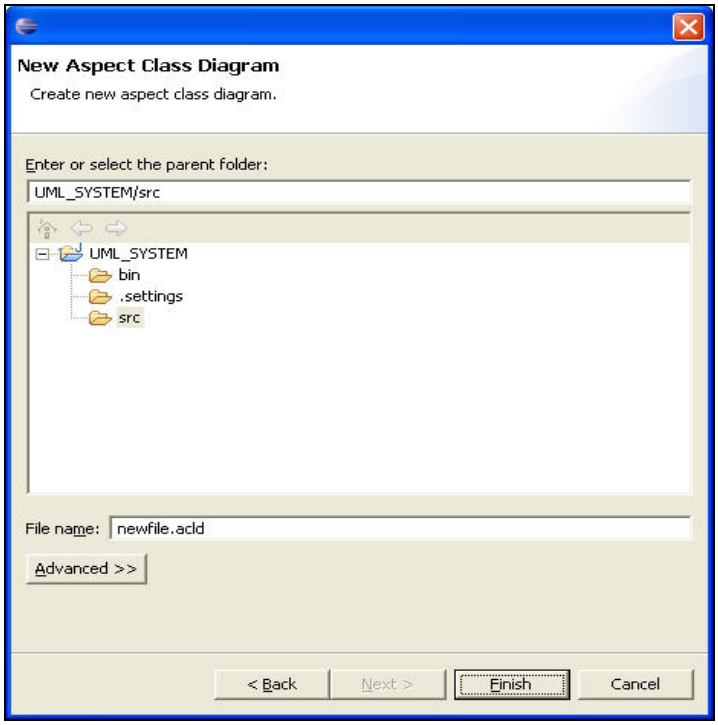
เมื่อเข้าใช้งาน Eclipse แล้วจะมีเครื่องมือย่อยให้เลือกใช้งาน ซึ่งเครื่องมือสำหรับวาดกราฟิกสัญลักษณ์ของภาษาออกแบบเชิงแบบจำลองเป็นหนึ่งในเครื่องมือย่อย ดังรูปที่ 3.19



รูปที่ 3.19 Plug-in SUTUML

2) หน้าจอการสร้างแผนภาพ

หน้าจอนี้เป็นหน้าจอเริ่มต้นที่จะทำการสร้างกราฟิกสัญลักษณ์ของภาษาออกแบบเชิงแบบจำลอง ดังแสดงในรูปที่ 3.20



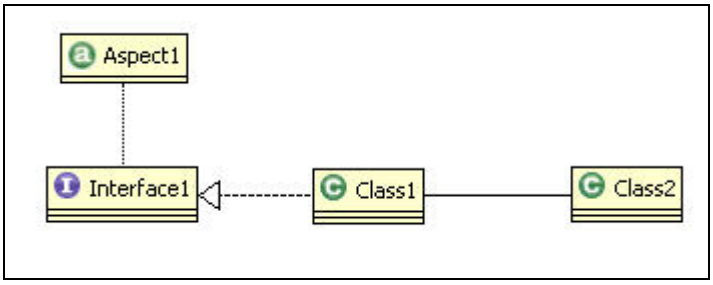
รูปที่ 3.20 หน้าจอสร้างแผนภาพเชิงลักษณะ

3) หน้าจอการวาดสัญลักษณ์กราฟิกสัญลักษณ์ของภาษาออกแบบเชิงแบบจำลอง

หน้าจอนี้เป็นหน้าจอของการออกแบบและสร้างกราฟิกสัญลักษณ์ของภาษาออกแบบเชิงแบบจำลอง ดังแสดงในรูปที่ 3.21

รูปที่ 3.21 หน้าจอการวาดสัญลักษณ์กราฟิกของแผนภาพเชิงลักษณะ

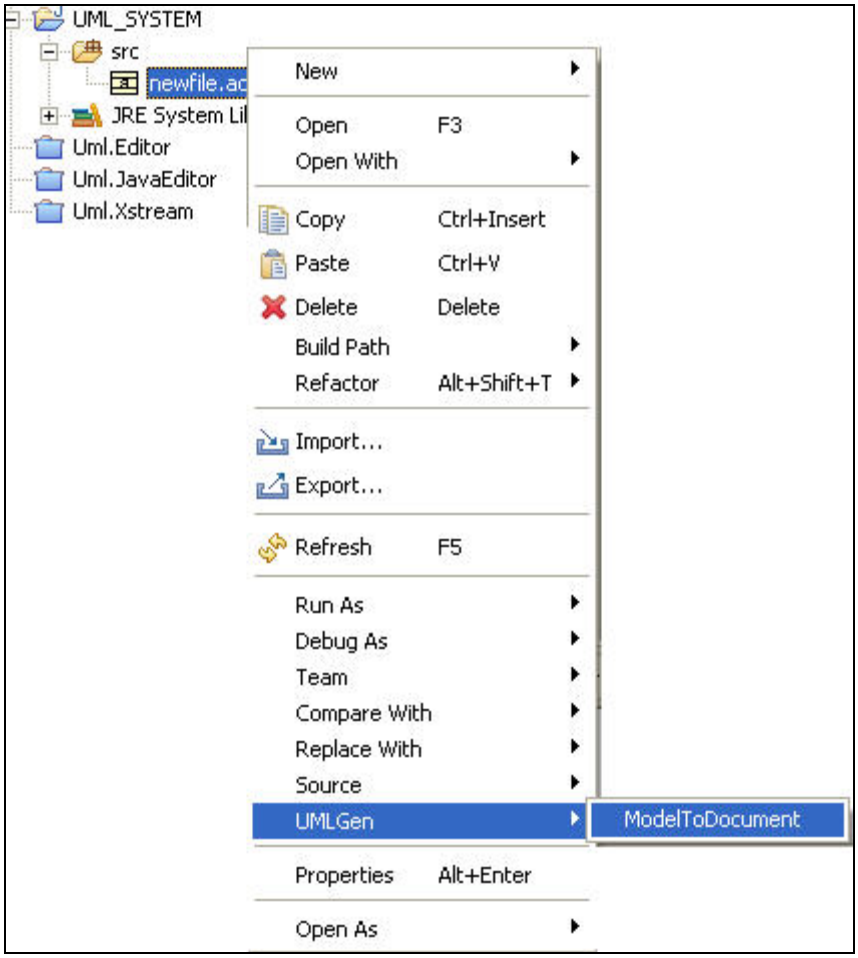
ซึ่งลักษณะของการออกแบบสัญลักษณ์กราฟิกของแผนภาพเชิงลักษณะจะปรากฏเป็นดังรูปที่ 3.22



รูปที่ 3.22 ลักษณะการออกแบบสัญลักษณ์

4) หน้าจอการแปลงรูปสัญลักษณ์กราฟิกสัญลักษณ์ของภาษาออกแบบเชิงแบบจำลองให้เป็นเอกสารแบบการจำลองวัตถุ

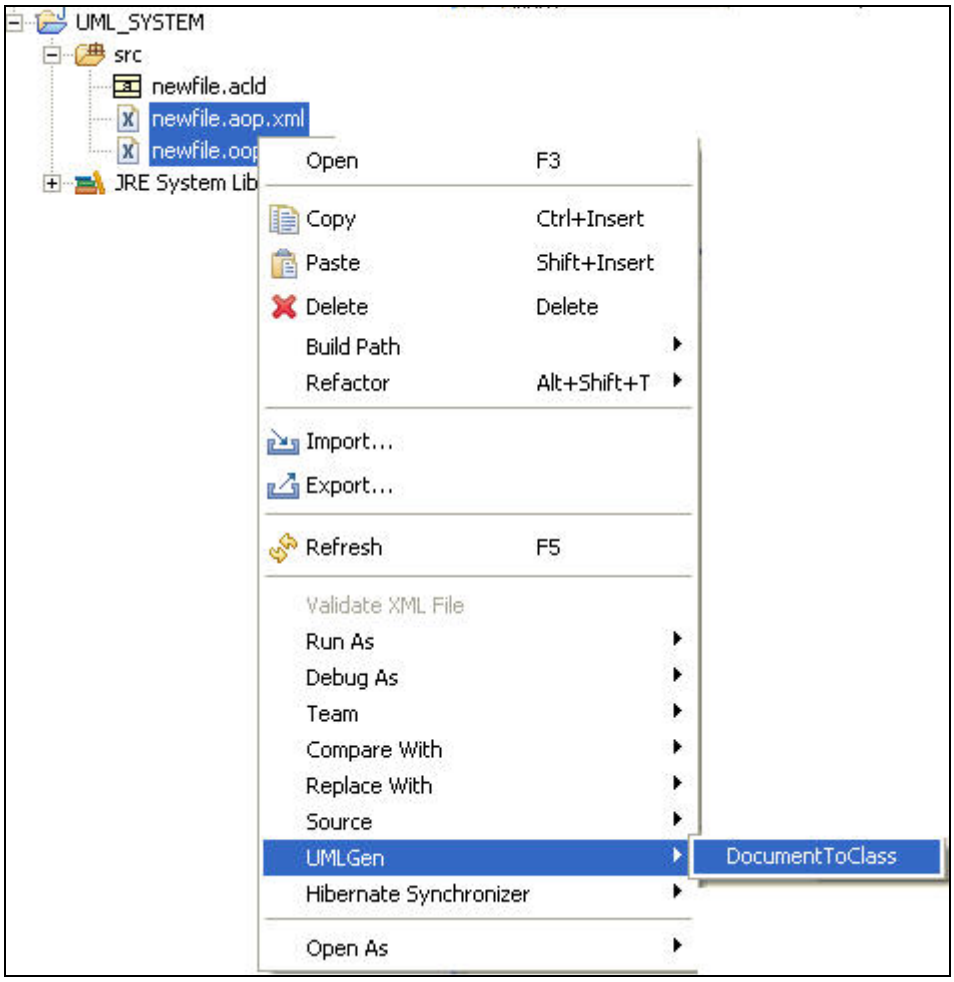
หน้าจอนี้เป็นการแสดงการนำเอากราฟิกสัญลักษณ์ของภาษาออกแบบเชิงโมเดลที่ได้เพิ่มเติมเทคนิคการเขียนโปรแกรมเชิงลักษณะซึ่งอยู่ในรูปแผนภาพมาทำการเปลี่ยนแปลงรูปแบบให้อยู่ในรูปของเอกสารแบบการจำลองวัตถุ ดังแสดงในรูปที่ 3.43



รูปที่ 3.23 หน้าจอการเปลี่ยนรูปข้อมูล

5) หน้าจอการแปลงรูปของเอกสารแบบการจำลองวัตถุให้อยู่เป็นโครงสร้างโค้ด

หน้าจอนี้เป็นการแสดงการนำเอาข้อมูลซึ่งอยู่ในรูปของแบบการจำลองวัตถุมาทำการเปลี่ยนแปลงรูปแบบให้อยู่ในรูปโครงสร้างของคลาสเชิงวัตถุ และคลาสเชิงลักษณะ ดังแสดงในรูปที่ 3.24



รูปที่ 3.24 หน้าจอการเปลี่ยนรูปข้อมูล

บทที่ 4
ผลการวิเคราะห์ข้อมูลและการอภิปรายผล

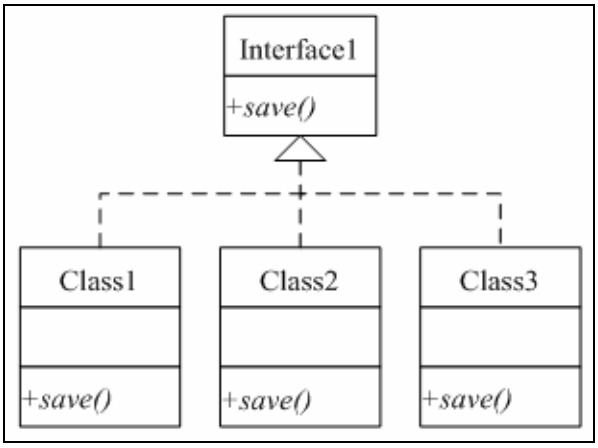
การทดลองในงานวิจัยนี้จะมุ่งเน้นไปที่การทำงานของภาษาออกแบบเชิงแบบจำลองที่เพิ่มความสามารถของการเขียนโปรแกรมเชิงลักษณะ ว่าสามารถเข้ากันได้กับภาษาออกแบบเชิงแบบจำลองแบบเดิมได้โดยจะทำการทดสอบกับข้อมูลจำลอง อีกส่วนหนึ่งจะเป็นการทำสอบการทำงานของเครื่องมือว่าสามารถรองรับภาษาออกแบบเชิงแบบจำลองที่เพิ่มความสามารถของการเขียนโปรแกรมเชิงลักษณะ รวมถึงการแปลงกราฟิกของภาษาออกแบบเชิงแบบจำลองให้เป็นเอกสารแบบการจำลองวัตถุ และโครงสร้างของคลาส

สำหรับเนื้อหาของบทนี้จะเป็นการนำเสนอผลการทดลองจากการทดสอบเครื่องมือสำหรับออกแบบภาษาออกแบบเชิงแบบจำลองที่เพิ่มความสามารถของการเขียนโปรแกรมเชิงลักษณะ โดยจะนำเสนอตามลำดับดังต่อไปนี้

4.1 อธิบายการทำงานของภาษาออกแบบเชิงแบบจำลองที่เพิ่มความสามารถของ การเขียนโปรแกรมเชิงลักษณะ
4.2 สภาพแวดล้อมที่ใช้พัฒนาเครื่องมือสำหรับออกแบบภาษาออกแบบเชิงแบบจำลองที่เพิ่มความสามารถของการเขียนโปรแกรมลักษณะ
4.3 โครงสร้างของเครื่องมือ
4.4 สภาพแวดล้อมที่ใช้ทดสอบเครื่องมือ
4.5 กรณีศึกษาที่ใช้ทดสอบเครื่องมือ
4.6 ขั้นตอนการทดสอบ
4.7 ผลการทดสอบเครื่องมือ
4.8 สรุปผลการทดสอบเครื่องมือ

4.1 อธิบายการทำงานของภาษาออกแบบเชิงแบบจำลองที่เพิ่มความสามารถของการเขียนโปรแกรมเชิงลักษณะ

สำหรับในส่วนนี้จะเป็นการอธิบายการทำงานร่วมกันของภาษาออกแบบเชิงแบบจำลอง ที่เป็นส่วนของการทำงานเชิงวัตถุ และเชิงลักษณะรูปที่ 4.1 แสดงการทำงานของการบันทึกการทำงาน



รูปที่ 4.1 แสดงการทำงานของกลุ่มการบันทึกการทำงาน

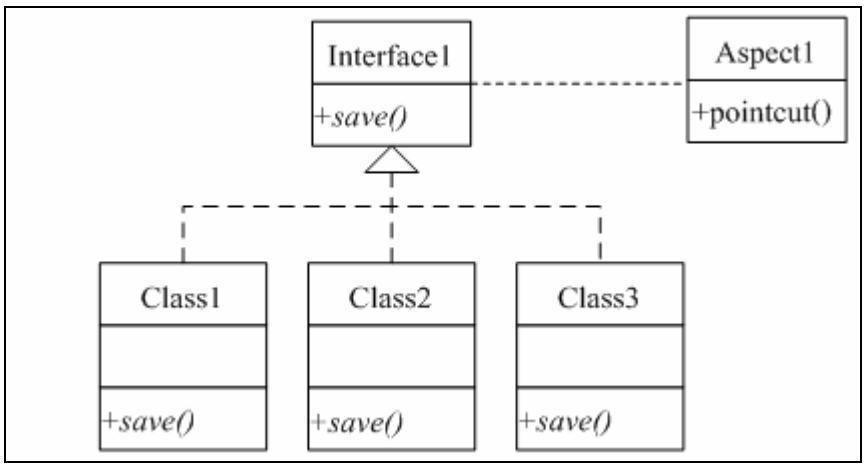
การทำงานของระบบที่ทำการแสดงให้เห็นข้างบนนั้นจะเป็นการบันทึกข้อมูลของระบบ ๆ หนึ่งลงไปในไฟล์ หรือฐานข้อมูลก็ตาม ซึ่งทุกครั้งที่ทำการบันทึกนั้นต้องทำการบันทึกข้อมูลของการบันทึกด้วยทุกครั้ง โดยรูปที่ 4.2 จะเป็นการแสดงให้เห็นถึงการทำงานของเมธอด save()

```
public void save(){

    // do something

    logging("record All Info");

}
```

รูปที่ 4.2 การทำงานของเมธอด save()

จากรูปที่ 4.2 แสดงให้เห็นว่าหลังจากได้มีการทำงานต่าง ๆ แล้วก่อนจะออกจากการทำงานภายในเมธอด save นั้นจะต้องมีการทำการบันทึกข้อมูลต่าง ๆ ก่อน ซึ่งทุก ๆ คลาสที่ได้ทำการอิมพลีเมนท์ Interface1 นั้นจะมีเมธอด save และทุก ๆ เมธอด save จะต้องมีการทำ log ทุกครั้งไป และถ้ายิ่งมีจำนวนคลาสมาทำการอิมพลีเมนท์ Interface1 มากขึ้นก็จะทำให้เกิดกระบวนการทำงานแบบนี้ซ้ำ ๆ กันมากขั้นเรื่อย ๆ

รูป 4.1 ได้แสดงให้เห็นถึงการออกแบบการทำงานของระบบเชิงวัตถุ ซึ่งการออกแบบด้วยความสามารถเชิงวัตถุนั้นไม่เพียงพอต่อการอธิบายทำงานได้ ดังนั้นถ้านำเอาระบบเดิมนี้ไปออกแบบด้วยภาษาออกแบบเชิงแบบจำลองที่เพิ่มเติมความสามารถของการเขียนโปรแกรมเชิงลักษณะลงไปจะมีลักษณะของแผนภาพดังรูปที่ 4.3



รูปที่ 4.3 แสดงการทำงานของกลุ่มการบันทึกการทำงานด้วยการเพิ่มเติมคุณสมบัติการเขียนโปรแกรมเชิงลักษณะ

การทำงานของรูปที่ 4.3 นั้นจะเหมือนกับรูปที่ 4.1 คือจะทำการบันทึกข้อมูลบางอย่างของระบบลงไปในไฟล์ หรือฐานข้อมูล โดยจะแตกต่างกันตรงที่ วิธีการทำงานภายในเมธอดที่ปรากฏในรูป 4. 2 ซึ่งการทำงานในส่วนเขียนข้อมูลลงบันทึกนั้นจะถูกยกไปทำที่คลาสเชิงลักษณะแทน โดยลักษณะการทำงานภายในเมธอดของคลาสเชิงลักษณะนั้นแสดงอยู่ในรูปที่ 4.4

```
pointcut pointcut1 ( ) : call (void Interface1.save( ) ) ;
        after ( ) : pointcut1 ( ) {
            logging("record all Info")
        }
```

รูปที่ 4.4 การทำงานของเมธอดภายในคลาสเชิงลักษณะ

การทำงานที่เกิดขึ้นภายในแต่ละเมธอดของคลาสที่ไปอิมพลีเมนท์ Interface1 นั้นจะถูกยกมาทำที่เมธอดของคลาสเชิงลักษณะที่เดียว ดังนั้นสิ่งที่เหลืออยู่ภายในเมธอดของคลาสที่มาอิมพลีเมนท์ Interface1 นั้นคือการทำงานภายในอย่างเดียวเท่านั้น ดังรูปที่ 4.5

```
public void save(){
    // do something
}
```

รูปที่ 4.5 การทำงานของเมธอด save() หลังจากใช้เทคนิคการเขียนโปรแกรมเชิงลักษณะ

การทำงานของส่วนเพิ่มเติมความสามารถการเขียนโปรแกรมเชิงลักษณะในภาษาออกแบบเชิงแบบจำลองนั้นไม่สามารถทำงานด้วยตัวมันเองได้ แต่จะคอยสนับสนุนและช่วยเหลือการทำงานของภาษาออกแบบเชิงแบบจำลองในส่วนของการพัฒนาระบบเชิงวัตถุ เพื่อนำมาออกแบบและอธิบายการทำงานของระบบที่ต้องการพัฒนาเท่านั้น ดังนั้นผู้ที่ทำการวิเคราะห์และออกแบบระบบต้องรู้ด้วยตัวเองว่าจะนำการทำงานของการเขียนโปรแกรมเชิงลักษณะมาเพื่อใช้อธิบายการทำงานตรงส่วนไหนของระบบ

4.2 สภาพแวดล้อมที่ใช้พัฒนาเครื่องมือสำหรับออกแบบภาษาออกแบบเชิงแบบจำลองที่เพิ่มความสามารถของการเขียนโปรแกรมลักษณะ

เครื่องคอมพิวเตอร์ที่ใช้ในการพัฒนามีรายละเอียดดังนี้

1) ฮาร์ดแวร์
   - คอมพิวเตอร์ส่วนบุคคล (PC) Pentium IV 3.2 กิกะเฮิร์ส
   - หน่วยความจำ 512 เมกกะไบต์
   - ฮาร์ดดิสความจุ 80 กิกะไบต์
2) ซอฟต์แวร์
   - ระบบปฏิบัติการ ไมโครซอฟต์วินโดวส์ เอ็กซ์พี เซอร์วิสแพ็ค 2
   - จาวา J2SE 1.4.2
   - Eclipse IDE 3.1
   - Graphical Editing Framework (GEF) 2.0

- Eclipse Java Development Tools

4.3 โครงสร้างของเครื่องมือ

โครงสร้างหลักของเครื่องมือจะประกอบไปด้วยส่วนหลัก ๆ อยู่ 3 ส่วนดังแสดงในรูปที่ 4.6

รูปที่ 4.6 แสดงส่วนประกอบหลักของเครื่องมือ

การทำงานของเครื่องมือนั้นจะถูกแบ่งออกเป็นส่วน ๆ ดังที่ปรากฏอยู่ในรูปที่ 4.6 โดยแต่ละส่วนนั้นจะมีระบบย่อย เพื่อสนับสนุนการทำงานของแต่ละส่วน ซึ่งจะทำการอธิบายการทำงานในหัวข้อย่อยถัดไป

4.3.1 ส่วนการวาดแผนภาพกราฟิก

การทำงานของเครื่องมือนั้นจะต้องผ่านส่วนแรกสุดก่อน ซึ่งก็คือ ส่วนของการออกแบบ และจะได้แบบจำลองของภาษาออกแบบเชิงแบบจำลอง เพื่อนำไปใช้ในส่วนต่อไปของเครื่องมือ ซึ่งในส่วนของการออกแบบจะสามารถแบ่งได้เป็น 3 ส่วน

1) ส่วนการวาดแผนภาพกราฟิก

ส่วนนี้จะเป็นส่วนที่วาดกราฟิกสัญลักษณ์ของภาษาออกแบบเชิงแบบจำลอง ซึ่งจะประกอบไปด้วยสัญลักษณ์ของ คลาสเชิงวัตถุ อินเตอร์เฟส และ คลาสเชิงลักษณะ นอกจากนี้ยังมีความสัมพันธ์ต่าง ๆ อย่างเช่น ความสัมพันธ์ที่เกี่ยวข้องกัน (Association) และความสัมพันธ์แบบจุดตัด (Joint) เป็นต้น

2) ส่วนการกำหนดค่าคุณสมบัติ

การทำงานในส่วนนี้จะเป็นการปรับค่าต่าง ๆ ที่ปรากฏบนกราฟิกของคลาสและบนเส้นของความสัมพันธ์ระหว่างคลาส ก็ต้องมาปรับในส่วนของการกำหนดค่าคุณสมบัตินี้ นอกจากนั้น ยังรวมไปถึงลักษณะของคลาสว่าเป็นคลาสแบบรูปธรรม หรือนามธรรม

3) ส่วนภาพร่าง

ส่วนของภาพร่างนี้จะทำการแสดงโครงสร้างทั้งหมดของแผนภาพกราฟิกของภาษาออกแบบเชิงแบบจำลอง เพื่อจะได้ทำการกำหนด และจัดการขนาดของรูปกราฟิกต่าง ๆ ที่แสดงผลออกมาให้เหมาะสม

4.3.2 ส่วนของการเปลี่ยนรูปของข้อมูล

หน้าที่หลักของการทำงานของส่วนของการเปลี่ยนรูปของข้อมูล คือ นำแผนภาพกราฟิกของภาษาออกแบบเชิงแบบจำลองที่ถูกออกแบบแล้ว มาทำให้อยู่ในสภาพของเอกสารแบบการจำลองวัตถุเพื่อจะได้นำไปใช้ในส่วนของการสร้างโครงสร้างโค้ด

4.3.3 ส่วนของการสร้างโครงสร้างของโค้ด

การทำงานของส่วนนี้จะเป็นการนำเอกสารแบบการจำลองวัตถุมาทำการสร้างโครงสร้างของโค้ดเพื่อนำไปใช้ต่อในส่วนของการพัฒนา โดยเอกสารแบบการจำลองวัตถุที่นำมาใช้สร้างนั้นไม่จำเป็นต้องผ่านมาจากส่วนของการเปลี่ยนรูปของข้อมูลก็ได้ ซึ่งอาจจะมาจากผู้วิเคราะห์และออกแบบระบบได้ทำการสร้างและเขียนไฟล์ขึ้นมาเอง แต่มีข้อแม้ว่าต้องอยู่ในรูปแบบที่ทางระบบได้กำหนดไว้ ซึ่งถ้าไม่มีความชำนาญในการเขียนเอกสารเองสามารถให้ส่วนก่อนหน้าทำการสร้างให้ได้

4.4 สภาพแวดล้อมที่ใช้ทดสอบเครื่องมือ

สภาพแวดล้อมที่ใช้ทดสอบเครื่องมือเป็นสภาพแวดล้อมเดียวกันกับที่ใช้ในการพัฒนาเครื่องมือ

4.5 กรณีศึกษาที่ใช้ทดสอบเครื่องมือ

กรณีศึกษาที่ใช้ทดสอบเครื่องมือ เป็นกรณีศึกษาของระบบจัดการงานด้านบุคลากรของศูนย์ราชการ

การวิเคราะห์และออกแบบการทำงานของระบบ เพื่ออธิบายการทำงาน และความสัมพันธ์ระหว่างคลาส โดยระบบจัดการงานด้านบุคลากรของศูนย์ราชการสามารถทำการวิเคราะห์ลักษณะออกมาได้เป็นดังต่อไปนี้

4.5.1 คลาสข้อมูล (Entity Class)

เป็นกลุ่มคลาสที่ทำหน้าที่เป็นตัวเก็บคุณสมบัติทั่ว ๆ ไปของระบบ โดยจะเป็นตัวบอกคุณสมบัติ และพฤติกรรมของแต่ละคลาส ซึ่งสามารถแบ่งออกมาได้เป็นดังนี้

- คลาสเจ้าหน้าที่ เป็นคลาสที่อธิบายลักษณะของเจ้าหน้าที่ซึ่งทำงานอยู่ในศูนย์งานราชการ
- คลาสพนักงาน เป็นคลาสที่อธิบายลักษณะของพนักงานที่ทำงานอยู่ในศูนย์งานราชการ โดยคลาสดังกล่าวมีลักษณะทั่วไปเหมือนเจ้าหน้าที่ แต่มีบางคุณสมบัติที่เข้ามาเพิ่มเพื่อบ่งบอกถึงลักษณะเฉพาะตัวของคลาส
- คลาสลูกจ้าง เป็นคลาสที่อธิบายลักษณะของลูกจ้างที่ทำงานอยู่ในศูนย์งานราชการ โดยคลาสดังกล่าวมีลักษณะทั่วไปเหมือนเจ้าหน้าที่ แต่มีบางคุณสมบัติที่เข้ามาเพิ่มเพื่อบ่งบอกถึงลักษณะเฉพาะตัวของคลาส
- คลาสลูกจ้างประจำ เป็นคลาสที่อธิบายลักษณะของลูกจ้างประจำที่ทำงานอยู่ในศูนย์งานราชการ โดยคลาสดังกล่าวมีลักษณะทั่วไปเหมือนลูกจ้าง แต่มีบางคุณสมบัติที่เข้ามาเพิ่มเพื่อบ่งบอกถึงลักษณะเฉพาะตัวของคลาส
- คลาสลูกจ้างชั่วคราว เป็นคลาสที่อธิบายลักษณะของลูกจ้างชั่วคราวที่ทำงานอยู่ในศูนย์งานราชการ โดยคลาสดังกล่าวมีลักษณะทั่วไปเหมือนลูกจ้าง แต่มีบางคุณสมบัติที่เข้ามาเพิ่มเพื่อบ่งบอกถึงลักษณะเฉพาะตัวของคลาส
- คลาสเลขที่ตำแหน่ง เป็นคลาสที่อธิบายลักษณะของตำแหน่งที่เจ้าหน้าที่คนนั้นมีความเกี่ยวข้องด้วย
- คลาสประวัติตำแหน่งและอัตราเงินเดือน เป็นคลาสที่อธิบายประวัติการทำงานและอัตราเงินเดือนของเจ้าหน้าที่
- คลาสบัญชีอัตราเงินเดือน เป็นคลาสที่อธิบายรายละเอียดของเงินเดือน
- คลาสวันลา เป็นคลาสที่อธิบายรายละเอียดของวันลาของเจ้าหน้าที่
- คลาสประวัติการศึกษา เป็นคลาสที่อธิบายรายละเอียดของการศึกษาของเจ้าหน้าที่
- คลาสการอบรม เป็นคลาสที่อธิบายรายละเอียดของการอบรม และสัมมนาของเจ้าหน้าที่
- คลาสความผิดทางวินัย เป็นคลาสที่อธิบายรายละเอียดของความผิดทางวินัยของเจ้าหน้าที่
- คลาสปริมาณงาน เป็นคลาสที่อธิบายรายละเอียดของปริมาณงานที่ได้ปฏิบัติของเจ้าหน้าที่
- คลาสสายงาน เป็นคลาสที่อธิบายรายละเอียดของลักษณะของสายงานที่มีความสัมพันธ์ต่อเลขที่ตำแหน่ง

- คลาสสำนัก เป็นคลาสที่ทำการอธิบายรายละเอียดของลักษณะของสำนักงานที่มีความสัมพันธ์ต่อสายงาน
- คลาสกลุ่มที่เกื้อกูลกัน เป็นคลาสที่อธิบายรายละเอียดของลักษณะของเกื้อกูลกันที่มีความสัมพันธ์ต่อสายงาน
- อินเทอร์เฟสเลื่อนขั้นได้ เป็นส่วนที่อธิบายรายละเอียด และลักษณะของการเลื่อนขั้น
- อินเทอร์เฟสเลื่อนขั้นได้ เป็นส่วนที่อธิบายรายละเอียด และลักษณะของการรับเครื่องราชอิสริยาภรณ์
- คลาสโปรโมต เป็นคลาสที่อธิบายรายละเอียดของเจ้าหน้าที่ที่ได้รับการคัดเลือกให้เลื่อนระดับ
- คลาสประวัติเครื่องราชอิสริยาภรณ์ เป็นคลาสที่อธิบายประวัติการได้รับพระราชทานเครื่องราชอิสริยาภรณ์
- คลาสประวัติการขอเครื่องราชอิสริยาภรณ์ เป็นคลาสที่อธิบายประวัติการขอรับพระราชทานเครื่องราชอิสริยาภรณ์
- คลาสเครื่องราชอิสริยาภรณ์ เป็นคลาสที่อธิบายรายละเอียดของราชอิสริยาภรณ์
- คลาสสหกรณ์ เป็นคลาสที่อธิบายของสหกรณ์

4.5.2 คลาสควบคุม (Controller Class)

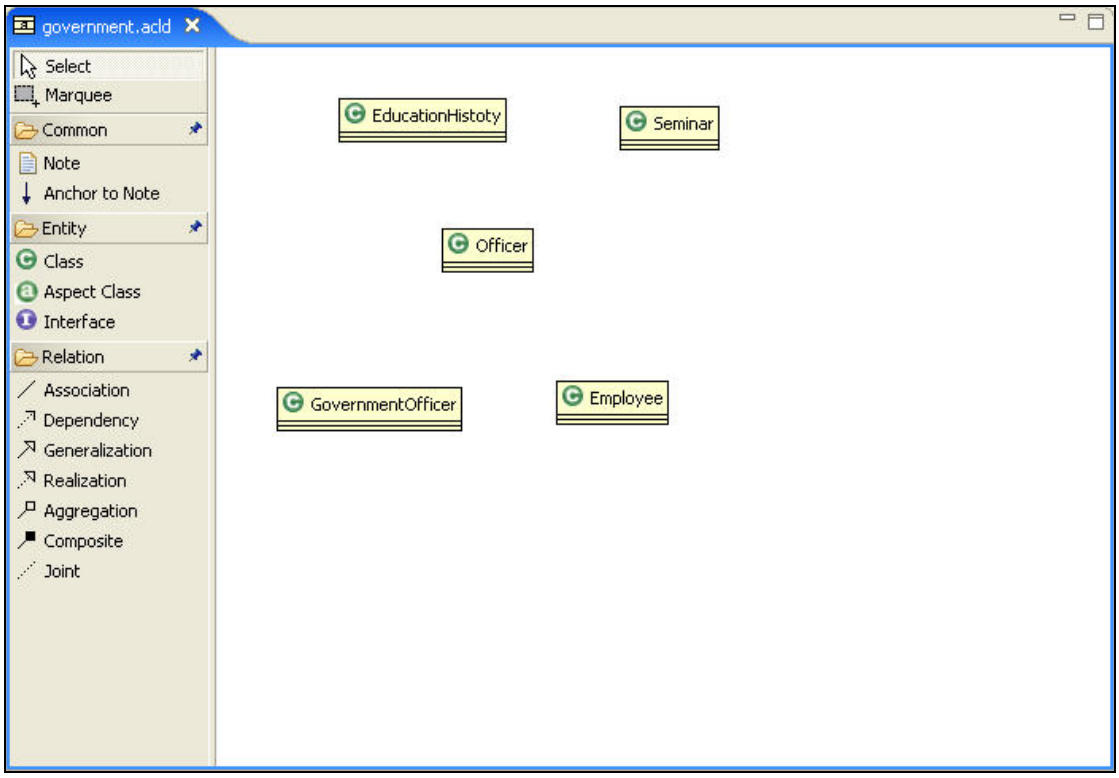
เป็นกลุ่มคลาสที่ทำหน้าที่เป็นตัวควบคุมการทำงานของระบบ โดยจะทำหน้าที่เป็นตัวกลางคอยติดต่อระหว่างระบบอินเตอร์เฟส และคลาสข้อมูล ซึ่งสามารถแบ่งออกมาได้เป็นดังนี้

- คลาส DisciplineDAO เป็นคลาสที่ควบคุมการทำงาน และการเข้าถึงข้อมูลของคลาสความผิดทางวินัย
- คลาส HornorAbleDAO เป็นคลาสที่ควบคุมการทำงาน และการเข้าถึงข้อมูลของคลาสรับเครื่องราชอิสริยาภรณ์ได้
- คลาส HornorDAO เป็นคลาสที่ควบคุมการทำงาน และการเข้าถึงข้อมูลของคลาสเครื่องราชอิสริยาภรณ์
- คลาส HornorHistoryDAO เป็นคลาสที่ควบคุมการทำงาน และการเข้าถึงข้อมูลของคลาสประวัติการได้รับพระราชทานเครื่องราชอิสริยาภรณ์
- คลาส HornorRequestingDAO เป็นคลาสที่ควบคุมการทำงาน และการเข้าถึงข้อมูลของคลาสประวัติการขอรับพระราชทานเครื่องราชอิสริยาภรณ์

- คลาส HornorHandler เป็นคลาสที่ควบคุมการทำงานและจัดการข้อมูลของคลาสเครื่องราชอิสริยาภรณ์
- คลาส OfficeDAO เป็นคลาสที่ควบคุมการทำงาน และช่วยในการการเข้าถึงข้อมูลของคลาสเจ้าหน้าที่
- คลาส PositionDAO เป็นคลาสที่ควบคุมการทำงาน และช่วยในการเข้าถึงข้อมูลของคลาสเลขที่ตำแหน่ง
- คลาส PromoteAble เป็นคลาสที่ควบคุมการทำงาน และการเข้าถึงข้อมูลของกลุ่มคลาสที่อิมพลีเมนต์อินคลาสเลขที่ตำแหน่ง
- คลาส PromoteHistoryDAO เป็นคลาสที่ควบคุมการทำงาน และข้อมูลของคลาสประวัติการเลื่อนขั้น
- คลาส SalaryDAO เป็นคลาสที่ควบคุมการทำงาน และการช่วยในการจัดการเข้าถึงข้อมูลของคลาสอัตราเงินเดือน
- คลาส SalaryHandler เป็นคลาสที่ควบคุมการทำงาน และใช้ข้อมูลของคลาสอัตราเงินเดือน
- คลาส SeminarDAO เป็นคลาสที่ควบคุมการทำงาน และการเข้าถึงข้อมูลของคลาสประวัติการอบรม

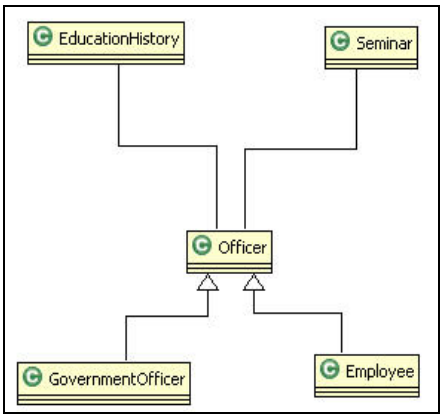
4.6 ขั้นตอนการทดสอบ

1) ทำการเปิดโปรแกรม และเลือกเข้าใช้งานโปรแกรม เพื่อทำการสร้างแผนภาพของภาษาออกแบบเชิงแบบจำลอง
2) ทำการเลือกสัญลักษณ์กราฟิกของภาษาออกแบบเชิงแบบจำลองมาทำการออกแบบตามที่ได้วิเคราห์ไว้ ดังรูปที่ 4.7



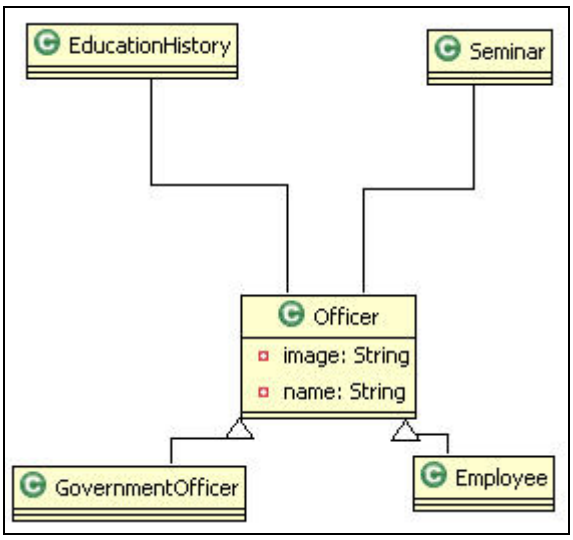
รูปที่ 4.7 การออกแบบแผนภาพของภาษาออกแบบเชิงแบบจำลอง

3) ทำการเลือกสัญลักษณ์ความสัมพันธ์ของภาษาออกแบบเชิงแบบจำลองมาทำการออกแบบตามที่ได้วิเคราะห์ไว้ ดังรูปที่ 4.8



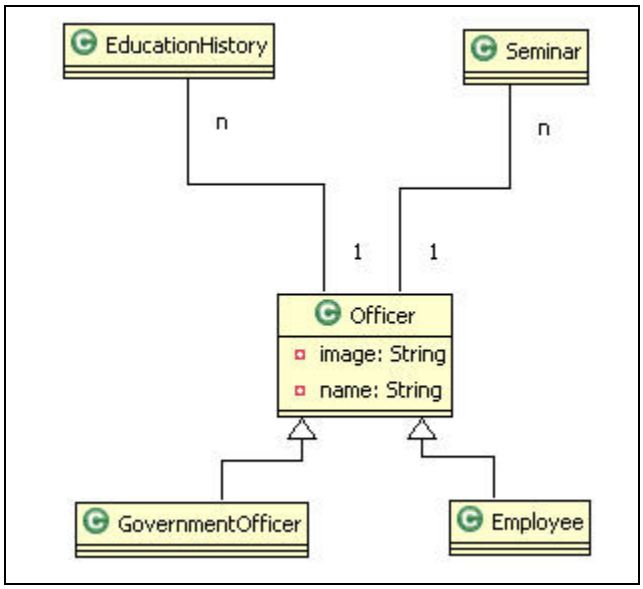
รูปที่ 4.8 การออกแบบแผนภาพของภาษาออกแบบเชิงแบบจำลอง

4) ทำการเพิ่มคุณสมบัติ (attribute) โดยการคลิกขวา → add attribute ดังรูปที่ 4.9



รูปที่ 4.9 การออกแบบแผนภาพของภาษาออกแบบเชิงแบบจำลอง

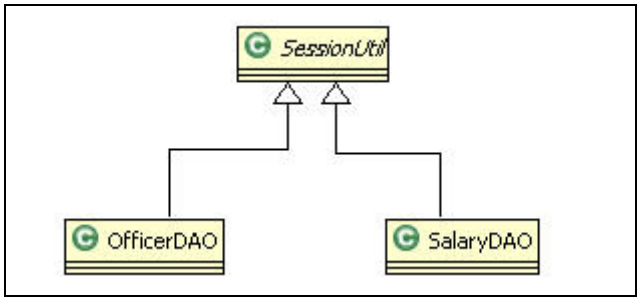
5) ทำการเพิ่มความสัมพันธ์ระหว่างคลาสข้อมูลโดยการเลือกตัวเลือกที่หน้าต่างกำหนดค่าคุณสมบัติ ดังรูปที่ 4.10



รูปที่ 4.10 การออกแบบแผนภาพของภาษาออกแบบเชิงแบบจำลอง

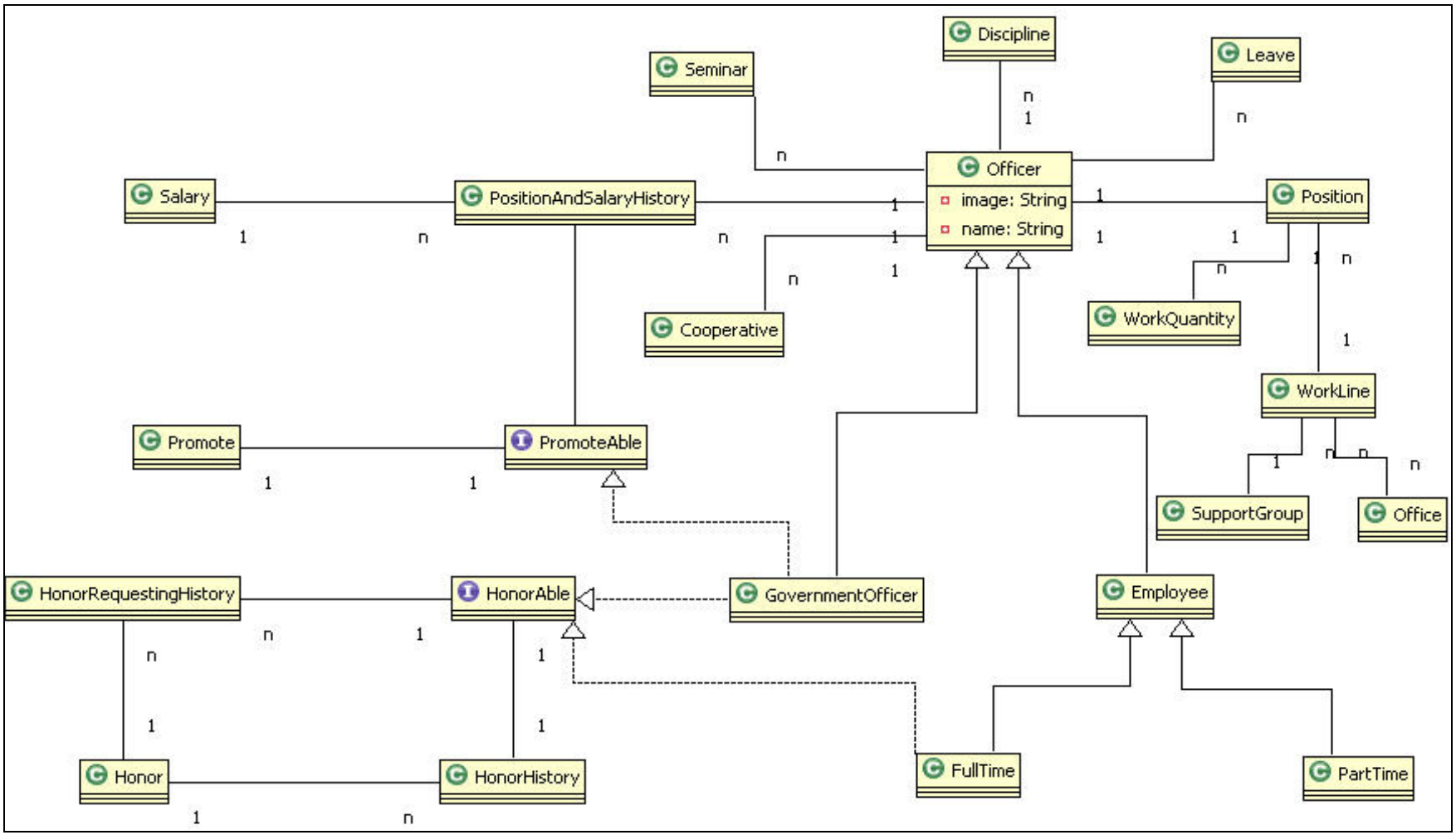
เมื่อได้ทำการออกแบบตามที่ได้วิเคราะห์คลาสข้อมูลทั้งหมด และได้ทำการนำมาออกแบบแผนภาพของภาษาออกแบบเชิงแบบจำลองด้วยเครื่องมือแล้วจะได้คลาสของความสัมพันธ์ทั้งหมดเป็นดังรูปที่ 4.12

6) ทำการออกแบบคลาสควบคุม (Controller Class) ด้วยเครื่องมืออีกครั้ง ดังรูปที่ 4.11



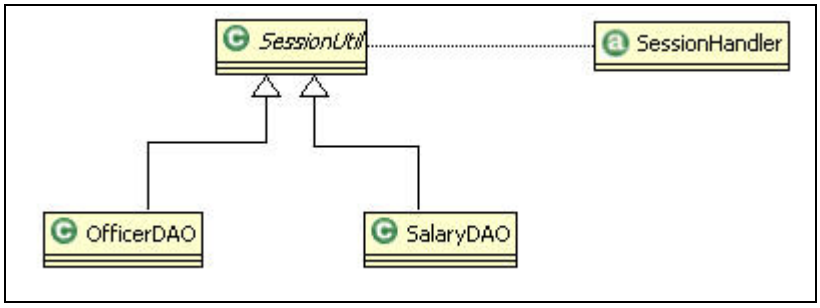
รูปที่ 4.11 การออกแบบแผนภาพของภาษาออกแบบเชิงแบบจำลอง

คลาส SessionUtil เป็นคลาสที่ถูกออกแบบมาแบบพิเศษเพื่อใช้ความสามารถของการเขียนโปรแกรมเชิงลักษณะมาจัดการปัญหาที่เกิดจากการเรียกใช้ Session ซ้ำ ๆ กันของทุก ๆ คลาส



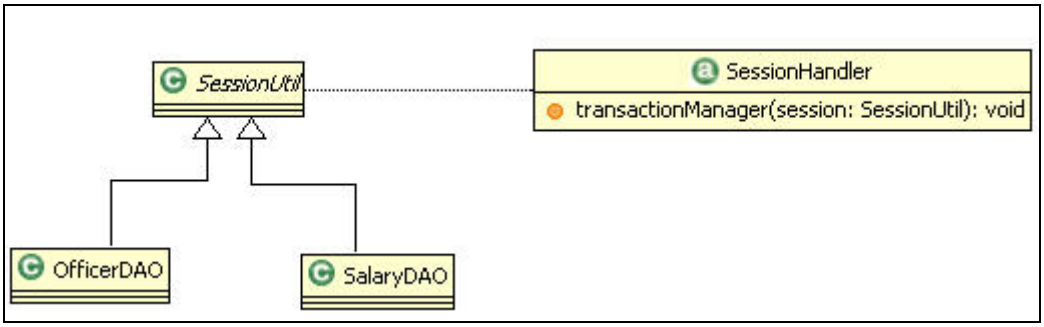
รูปที่ 4.12 การออกแบบแผนภาพของภาษาออกแบบเชิงแบบจำลอง

7) ทำการออกแบบคลาสเชิงลักษณะเพื่อมาจัดการคลาส Session ดังรูปที่ 4.13



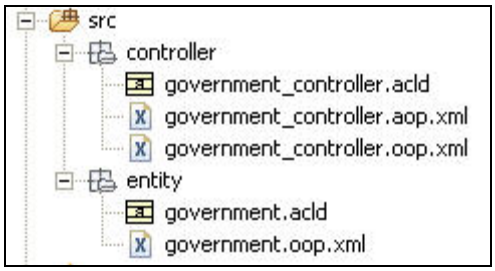
รูปที่ 4.13 การออกแบบแผนภาพของภาษาออกแบบเชิงแบบจำลองที่เพิ่มความสามารถของการเขียนโปรแกรมเชิงลักษณะ

8) ทำการเพิ่มคุณสมบัติ (Property or Attribute) โดยการคลิกขวา → add attribute หรือถ้าเป็นคลาสเชิงลักษณะจะทำการเพิ่มคุณสมบัติ โดยการคลิกขวา → add jointpoint แทน ดังรูปที่ 4.14 และ รูปที่ 4.16 แสดงความสัมพันธ์ทั้งหมดของการออกแบบคลาสควบคุม



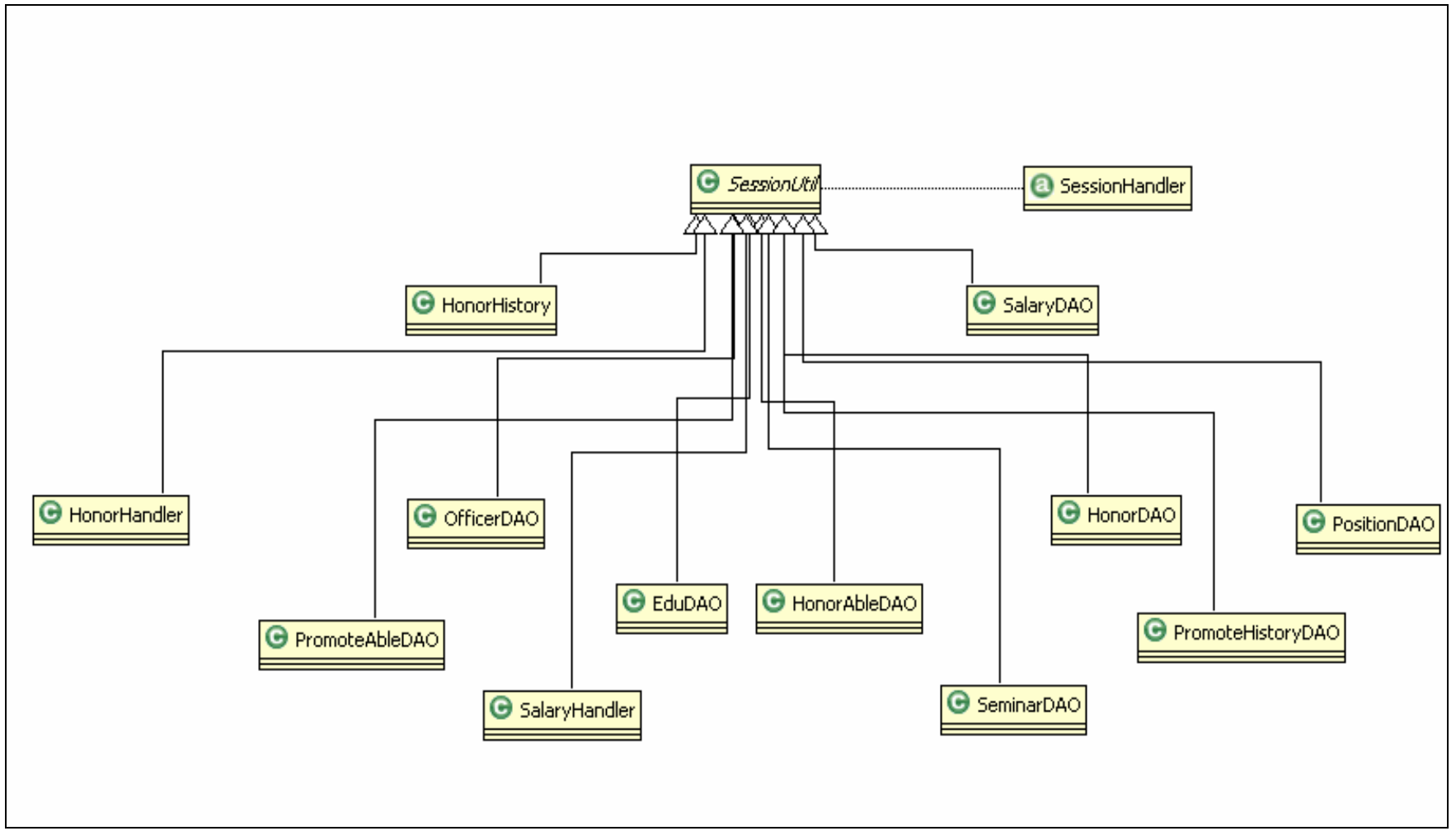
รูปที่ 4.14 การออกแบบแผนภาพของภาษาออกแบบเชิงแบบจำลอง

9) ทำการแปลงรูปแบบของแผนภาพกราฟิกของภาษาออกแบบเชิงแบบจำลองให้อยู่ในรูปของเอกสารแบบจำลองวัตถุ โดยการคลิกขวาที่ไฟล์นามสกุล .acld → UMLGen → ModelToDocument จะทำให้ได้ไฟล์เอกสารดังรูปที่ 4.16



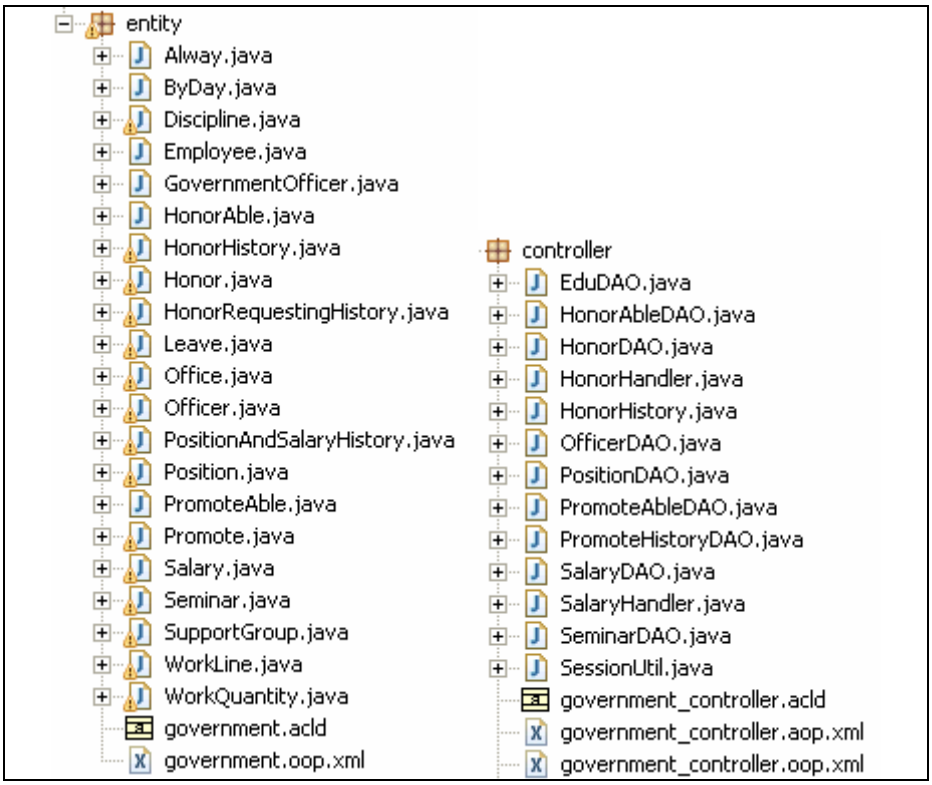
รูปที่ 4.15 เอกสารแบบจำลองวัตถุ

ไฟล์เอกสารแบบจำลองวัตถุที่ได้จากการสร้างด้วยเครื่องมือดังรูปที่ 4.16 นั้นจะมีอยู่ 2 นามสกุล คือ นามสกุล .aop.xml และ .oop.xml โดยนามสกุล .aop.xml นั้นจะเป็นเอกสารแบบจำลองวัตถุของคลาสเชิงลักษณะ ส่วน นามสกุล .oop.xml นั้นจะเป็นเอกสารแบบจำลองวัตถุของคลาสเชิงวัตถุ ซึ่งเครื่องมือจะทำการตรวจสอบเองว่าในแผนภาพที่ทำการออกแบบนั้นมีคลาสแบบใดได้ถูกออกแบบอยู่ในนั้นบ้าง



รูปที่ 4.16 การออกแบบแผนภาพของภาษาออกแบบเชิงแบบจำลองที่เพิ่มการเทคนิคของการเขียนโปรแกรมเชิงลักษณะ

10) ทำการแปลงเอกสารแบบจำลองวัตถุให้อยู่ในรูปแบบของโครงสร้างของคลาสเชิงวัตถุและคลาสเชิงลักษณะ โดยการคลิกขวาที่ไฟล์นามสกุล .aop.xml หรือ .oop.xml → UMLGen → DocumentToClass จะทำให้ได้ไฟล์เอกสารดังรูปที่ 4.17



รูปที่ 4.17 ไฟล์ของคลาสเชิงวัตถุ และคลาสเชิงลักษณะ

ไฟล์ของโครงสร้างคลาสที่ได้จากการสร้างด้วยเครื่องมือดังรูปที่ 4.17 นั้นจะมีอยู่ 2 นามสกุล เช่นเดียวกันกับของเอกสารแบบจำลองวัตถุ ซึ่งได้แก่ นามสกุล .aj (Aspect J) และ .java โดยนามสกุล .aj นั้นจะเป็นโครงสร้างของคลาสเชิงลักษณะ ส่วน นามสกุล .java นั้นจะเป็นโครงสร้างของคลาสเชิงวัตถุ

หมายเหตุ ผลทดสอบสำหรับกรณีทดสอบที่ 1 แสดงในภาคผนวก

4.7 ผลการทดสอบเครื่องมือ

ในส่วนนี้จะแยกผลการทดสอบเครื่องมือออกแบบ 3 ส่วน ตามการทำงานของเครื่องมือซึ่งมี 3 ส่วนเท่ากัน โดยจะแยกผลของการทดสอบเป็นส่วน ๆ ดังนี้ คือ ผลการทดสอบของส่วนออกแบบภาษาออกแบบเชิงแบบจำลอง , ผลการทดสอบของส่วนแปลงข้อมูล และ ผลการทดสอบของส่วนสร้างโครงสร้างคลาสเชิงลักษณะ และคลาสเชิงวัตถุ

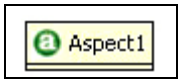
4.7.1 ผลการทดสอบส่วนออกแบบภาษาออกแบบเชิงแบบจำลองที่ได้เพิ่มเติมความสามารถของการเขียนโปรแกรมเชิงลักษณะ

ผลการทดสอบของส่วนออกแบบนี้จะเริ่มจากการนำแบบจำลองทุกชนิดที่อยู่ในเครื่องมือมาทำการวาดลงเครื่องมือ และใส่ความสัมพันธ์ทุกอย่างลงไปในแบบจำลอง ซึ่งลักษณะแบบจำลองบางชนิดจะใช้กับความสัมพันธ์บางอย่างไม่ได้ เช่น จะใช้ความสัมพันธ์แบบเกี่ยวข้องกัน (Association) ระหว่างคลาสเชิงวัตถุกับคลาสเชิงลักษณะ หรือ ระหว่างอินเทอร์เฟสกับคลาสเชิงลักษณะไม่ได้ จะต้องใช้ความสัมพันธ์แบบจุดตัด (Joint) เท่านั้น โดยจะทำการอธิบายผลการทดสอบความสามารถของเครื่องมือแบบลำดับดังนี้

- สัญลักษณ์ c ที่อยู่บนคลาสเป็นตัวบ่งบอกว่าคลาสนั้นเป็นคลาสเชิงวัตถุ ดังรูปที่ 4.18

รูปที่ 4.18 สัญลักษณ์ของคลาสเชิงวัตถุ

- สัญลักษณ์ a ที่อยู่บนคลาสเป็นตัวบ่งบอกว่าเป็นคลาสเชิงลักษณะ ดังรูปที่ 4.19



รูปที่ 4.19 สัญลักษณ์ของคลาสเชิงลักษณะ

- สัญลักษณ์ I ที่อยู่บนคลาสเป็นตัวบ่งบอกว่าคลาสนั้นเป็นอินเทอร์เฟส ดังรูปที่ 4.20

รูปที่ 4.20 สัญลักษณ์ของอินเทอร์เฟส

- ไม่สามารถใช้ความสัมพันธ์แบบเกี่ยวข้องกัน (Association) ระหว่างคลาสเชิงวัตถุกับคลาสเชิงลักษณะ หรือ ระหว่างอินเทอร์เฟสกับคลาสเชิงลักษณะได้
- ไม่สามารถใช้ความสัมพันธ์แบบร่วม (Joint) ระหว่างคลาสเชิงวัตถุกับคลาสเชิงวัตถุ หรือ ระหว่างอินเทอร์เฟสกับคลาสเชิงวัตถุได้
- ไม่สามารถใช้ความสัมพันธ์แบบร่วม (Joint) ระหว่างคลาสเชิงลักษณะกับคลาสเชิงลักษณะได้
- สามารถสร้างคลาสเชิงวัตถุ และเพิ่มจำนวนของคลาสเชิงวัตถุได้
- สามารถสร้างคลาสเชิงลักษณะ และเพิ่มจำนวนของคลาสเชิงลักษณะได้
- สามารถสร้างอินเทอร์เฟส และเพิ่มจำนวนของอินเทอร์เฟสได้
- สามารถ แก้ไข/ลบ คลาสเชิงวัตถุได้
- สามารถ แก้ไข/ลบ คลาสเชิงลักษณะได้
- สามารถ แก้ไข/ลบ อินเทอร์เฟสได้
- สามารถเพิ่มคุณสมบัติ (Property or Attribute) ให้กับคลาสเชิงวัตถุได้
- สามารถเพิ่มพฤติกรรม(Behavior or Operation) ให้กับคลาสเชิงวัตถุได้
- สามารถเพิ่มคุณสมบัติ (Property or Attribute) ให้กับอินเทอร์เฟสได้
- สามารถเพิ่มพฤติกรรม(Behavior or Operation) ให้กับอินเทอร์เฟสได้
- คุณสมบัติ และพฤติกรรมคลาสเชิงวัตถุนั้นสามารถใช้ความสามารถของการถูกห่อหุ้ม (Encapsulation) ได้
- สามารถเพิ่มจุดตัด (Pointcut) ให้กับคลาสเชิงลักษณะได้
- สามารถ แก้ไข/ลบ คุณสมบัติ (Property or Attribute) ให้กับคลาสเชิงวัตถุได้
- สามารถ แก้ไข/ลบ พฤติกรรม (Behavior or Operation) ให้กับคลาสเชิงวัตถุได้
- สามารถ แก้ไข/ลบ คุณสมบัติ (Property or Attribute) ให้กับอินเทอร์เฟสได้
- สามารถ แก้ไข/ลบ พฤติกรรม (Behavior or Operation) ให้กับอินเทอร์เฟสได้
- สามารถ แก้ไข/ลบ จุดตัด (Pointcut) ให้กับคลาสเชิงลักษณะได้

- สามารถเพิ่มลักษณะของความสัมพันธ์ (0, 0...1, 1...n, n) ให้กับความสัมพันธ์แบบเกี่ยวข้องกันได้
- สามารถ แก้ไข/ลบ ลักษณะของความสัมพันธ์ (0, 0...1, 1...n, n) ให้กับความสัมพันธ์แบบเกี่ยวข้องกันได้
- เส้นความสัมพันธ์ทุกชนิดสามารถทำการขยาย และย่อตัวได้เมื่อเกิดการเปลี่ยนแปลงตำแหน่งของคลาสเชิงวัตถุ, อินเทอร์เฟส และคลาสเชิงลักษณะ

4.7.2 ผลการทดสอบส่วนแปลงข้อมูล

ผลการทดสอบข้อมูลของส่วนแปลงข้อมูลนี้จะเริ่มด้วยการนำคลาสลักษณะต่างมาแสดงผลการทดสอบ และแสดงผลการทดสอบด้วยความสัมพันธ์ระหว่างคลาส

4.7.2.1 ผลการทดสอบการแปลงข้อมูล ของคลาสเชิงวัตถุและอินเทอร์เฟส

การแปลงข้อมูลของคลาสเชิงวัตถุนี้จะครอบคลุมไปถึงในส่วนของอินเทอร์เฟสด้วย ซึ่งรายละเอียดของส่วนการแปลงข้อมูลนั้นจะอธิบายเป็นลำดับดังนี้

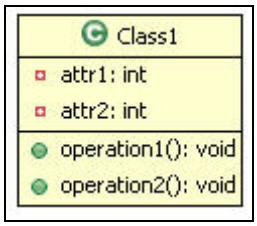
- เครื่องมือสามารถทำการแปลงข้อมูลของคลาสเชิงวัตถุได้ ถึงแม้จะไม่มีความสัมพันธ์ก็ตาม
- คลาสเชิงวัตถุ เมื่อไม่มีคุณสมบัติ และพฤติกรรม ดังรูปที่ 4.21 นั้นสามารถแปลงออกมาเป็นเอกสารแบบการจำลองวัตถุได้เป็นดังรูปที่ 4.22

รูปที่ 4.21 คลาสเชิงวัตถุ

```
<Object-Oriented>
  <Class name = "Class1"/>
</Object-Oriented>
```

รูปที่ 4.22 เอกสารแบบการจำลองวัตถุของคลาสเชิงวัตถุ

- คลาสเชิงวัตถุ เมื่อมีคุณสมบัติ และพฤติกรรม ดังรูปที่ 4.23 นั้นสามารถแปลงออกมาเป็นเอกสารแบบการจำลองวัตถุได้เป็นดังรูปที่ 4.24

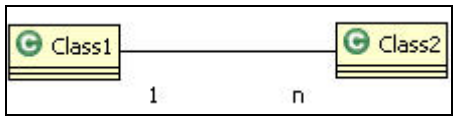


รูปที่ 4.23 คลาสเชิงวัตถุเมื่อมีคุณสมบัติ และพฤติกรรม

```
<Object-Oriented>
  <Class name = "Class1"/>
    <attribute name="attr1" type="int" accessType="private" isStatic="false" value=''1'/>
    <attribute name="attr2" type="int" accessType="private" isStatic="false" value=''2'/>
    <operation name="operation1" returnType="void" accessType="public" isAbstract="false" value="1";/>
    <operation name="operation2" returnType="void" accessType="public" isAbstract="false" value="2";/>
</Object-Oriented>
```

รูปที่ 4.24 เอกสารแบบการจำลองวัตถุของคลาสเชิงวัตถุเมื่อมีคุณสมบัติ และพฤติกรรม

- คลาสเชิงวัตถุ เมื่อมีความสัมพันธ์แบบเกี่ยวข้องกัน (Association) ดังรูปที่ 4.25 นั้นสามารถแปลงออกมาเป็นเอกสารแบบการจำลองวัตถุได้เป็นดังรูปที่ 4.26

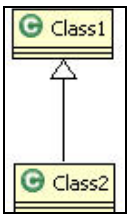


รูปที่ 4.25 คลาสเชิงวัตถุที่มีความสัมพันธ์แบบเกี่ยวข้องกัน

```
<Class name = “Class2”>
    <many-to-one class= ”class1”/>
</Class>
<Class name = “Class1”>
    <one-to-many class= ”class2”/>
</Class>
```

รูปที่ 4.26 เอกสารแบบการจำลองวัตถุของคลาสเชิงวัตถุที่มีความสัมพันธ์แบบเกี่ยวข้องกัน

- คลาสเชิงวัตถุ เมื่อมีความสัมพันธ์แบบสืบทอด (Generalization) ดังรูปที่ 4.27 นั้นสามารถแปลงออกมาเป็นเอกสารแบบการจำลองวัตถุได้เป็นดังรูปที่ 4.28

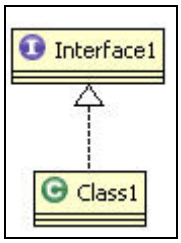


รูปที่ 4.27 เอกสารแบบการจำลองวัตถุของคลาสเชิงวัตถุที่มีความสัมพันธ์แบบเกี่ยวข้องกัน

```
<Object-Oriented>
  <Class name = “Class2”>
      <generalize>class1</generalize>
  </Class>
  <Class name = “Class1”/>
</Object-Oriented>
```

รูปที่ 4.28 เอกสารแบบการจำลองวัตถุของคลาสเชิงวัตถุที่มีความสัมพันธ์แบบ Generalization

- คลาสเชิงวัตถุ เมื่อมีความสัมพันธ์แบบเกี่ยวข้องกัน (Realization) ดังรูปที่ 4.29 นั้นสามารถแปลงออกมาเป็นเอกสารแบบการจำลองวัตถุได้เป็นดังรูปที่ 4.30



รูปที่ 4.29 คลาสเชิงวัตถุที่มีความสัมพันธ์แบบ Realization

```
<Object-Oriented>
  <Inteface name = "Inteface1">
  <Class name = "Class1">
    <realize>Inteface1</realize>
  </Class>
</Object-Oriented>
```

รูปที่ 4.30 เอกสารแบบการจำลองวัตถุของคลาสเชิงวัตถุที่มีความสัมพันธ์แบบ Realization

4.7.1.2 ผลการทดสอบการแปลงข้อมูล ของคลาสเชิงลักษณะ

รายละเอียดของส่วนการแปลงข้อมูลของคลาสเชิงลักษณะนั้นจะอธิบายเป็นลำดับดังนี้

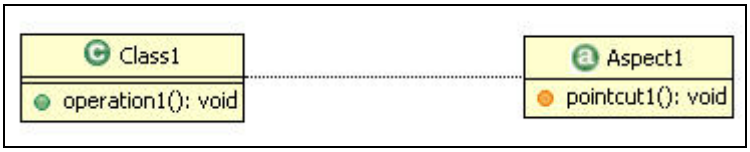
- เครื่องมือจะไม่ทำการแปลงข้อมูลของคลาสเชิงลักษณะถ้าไม่มีความสัมพันธ์แบบร่วมเกิดขึ้นระหว่างคลาสเชิงวัตถุกับคลาสเชิงลักษณะ
- คลาสเชิงลักษณะ เมื่อมีความสัมพันธ์แบบร่วม (Joint) ดังรูปที่ 4.31 นั้น สามารถแปลงออกมาเป็นเอกสารแบบการจำลองวัตถุได้เป็นดังรูปที่ 4.32

รูปที่ 4.31 คลาสเชิงลักษณะที่มีความสัมพันธ์แบบร่วม

```
<Aspect-Oriented>
  <Aspect-Class name="Aspect1" callClass="Class1">
     <pointcut values="1" />                          <Object-Oriented>
  </Aspect-Class>                                        <Class name="Class1" />
</Aspect-Oriented>                                    </Object-Oriented>
```

รูปที่ 4.32 เอกสารแบบการจำลองวัตถุของคลาสเชิงลักษณะที่มีความสัมพันธ์แบบร่วม

- คลาสเชิงลักษณะ เมื่อมีจุดตัด(Pointcut) กับคลาสเชิงวัตถุ ดังรูปที่ 4.33 นั้น สามารถแปลงออกมาเป็นเอกสารแบบการจำลองวัตถุได้เป็นดังรูปที่ 4.34



รูปที่ 4.33 คลาสเชิงลักษณะที่มีความสัมพันธ์แบบร่วมเมื่อมีจุดตัด

```
<Aspect-Oriented>
 <Aspect-Class name="Aspect1" callClass="Class1">
  <pointcut name="pointcut1" callMethod="operation1()" returnType="void" values="1" />
 </Aspect-Class>
</Aspect-Oriented>

<Object-Oriented>
 <Class name="Class1">
  <operation name="operation1" returnType="void" accessType="public" isAbstract="false" values="1" />
 </Class>
</Object-Oriented>
```

รูปที่ 4.34 เอกสารแบบการจำลองวัตถุของคลาสเชิงลักษณะที่มีความสัมพันธ์แบบร่วมเมื่อมีจุดตัดและมีพฤติกรรม

4.7.3 ผลการทดสอบของส่วนสร้างโครงสร้างของคลาสเชิงลักษณะ และคลาสเชิงวัตถุ

รายละเอียดของส่วนการสร้างโครงสร้างของคลาสนั้นจะอธิบายเป็นตาราง ดังต่อไปนี้

ตารางที่ 4.1 แสดงผลการทดสอบเครื่องมือส่วนการสร้างโครงสร้าง

ลักษณะของคลาส	ลักษณะโครงสร้างคลาส
คลาสเชิงวัตถุ ดังรูปที่ 4.21	public class Class1 { }

คลาสเชิงวัตถุ ดังรูปที่ 4.23	public class Class1 {
	private int attr1 ;
	private int attr2 ;
	public void operation1(){ }
	public void operation2(){ }
	}

คลาสเชิงวัตถุ ดังรูปที่ 4.25	public class Class1 {
	private java.util.Set class2;
	}public class Class2 {
	private Class1 class1;
	}

คลาสเชิงวัตถุ ดังรูปที่ 4.27	public class Class1 { } public class Class2 extends Class1 { }
คลาสเชิงวัตถุ ดังรูปที่ 4.29	public interface Interface1 { } public class Class1 implements Interface1 { }
คลาสลักษณะและเชิงวัตถุดังรูปที่ 4.31	public class Class1 { } public aspect Aspect1 { pointcut ( ) : call(Class1.);}

คลาสลักษณะและเชิงวัตถุดังรูปที่ 4.33	public class Class1 {
	public void operation1(){ }
	}public aspect Aspect1 {
	pointcut pointcut1 ( ) : call(void Class1.);}

4.8 สรุปผลการทดสอบเครื่องมือ

ในการทดสอบในส่วนของการออกแบบแผนภาพกราฟิกของภาษาออกแบบเชิงแบบจำลองที่เพิ่มเติมความสามารถของการเขียนโปรแกรมเชิงลักษณะนั้น พบว่าเครื่องมือสามารถทำการออกแบบสัญลักษณ์ต่าง ๆ ได้อย่างถูกต้อง ซึ่งรวมไปถึงรูปแบบของความสัมพันธ์แบบต่าง ๆ ด้วย โดยในการทดสอบนี้จะเน้นเฉพาะความสัมพันธ์แบบเกี่ยวข้องกัน (Association), ความสัมพันธ์แบบเจเนอร์รัลไรเซชั่น (Generalization), ความสัมพันธ์แบบเรียลไรเซชั่น (Realization) และความสัมพันธ์แบบร่วมเท่านั้น (Joint)

ในการทดสอบเครื่องมือส่วนของการแปลงข้อมูลให้อยู่ในรูปของเอกสารแบบการจำลองวัตถุนั้น พบว่าเครื่องมือจะทำงานได้อย่างถูกต้องก็ต่อเมื่อผู้ที่ทำการออกแบบนั้นได้ทำการออกแบบสัญลักษณ์ของแผนภาพกราฟิกของภาษาออกแบบเชิงแบบ จำลองมาอย่างถูกต้องเท่านั้น โดยผู้ที่ทำการออกแบบนั้นต้องรู้ลักษณะการทำงานของสัญลักษณ์แต่ละชนิดมาก่อน โดยถ้าเกิดการใช้สัญลักษณ์ไม่ถูกตามที่ภาษาออกแบบเชิงแบบจำลองได้นิยามเอาไว้ (นิยามโดย OMG) ทางเครื่องมืออาจเกิดการทำงานอย่างผิดพลาดหรือไม่ถูกต้องได้ โดยเครื่องมือจะไม่ไปทำการเปลี่ยนแปลงไฟล์และกราฟิกสัญลักษณ์ที่อยู่ภายในไฟล์อย่างเด็ดขาด

ในการทดสอบเครื่องมือส่วนของการสร้างโครงสร้าง ผลจะคล้ายเหมือนกันกับในส่วนของการทดสอบเครื่องมือส่วนของการแปลงข้อมูล ซึ่งก็คือเครื่องมือจะทำงานได้อย่างถูกต้องก็ต่อเมื่อผู้ที่ทำการสร้างเอกสารแบบการจำลองวัตถุนั้นถูกต้อง ซึ่งการจะสร้างเอกสารแบบการจำลองวัตถุนั้นทำได้ 2 วิธี คือ ใช้เครื่องมือสร้าง และผู้ใช้สร้างเอง โดยการสร้างเองนั้นผู้ใช้ต้องสร้างให้ถูกวากยสัมพันธ์ (Syntax) ของเอกสาร

หมายเหตุ วากยสัมพันธ์ (Syntax) ของเอกสารแสดงในภาคผนวก

บทที่ 5
สรุปผลการวิจัยและข้อเสนอแนะ

กระบวนการวิเคราะห์และออกแบบระบบซอฟต์แวร์เป็นส่วนหนึ่งของกระบวนการพัฒนาซอฟต์แวร์ โดยในขั้นตอนของกระบวนการวิเคราะห์และออกแบบระบบซอฟต์แวร์นี้สามารถนำเอาวิธีการออกแบบระบบรูปแบบต่าง ๆ เข้ามาช่วยวิเคราะห์และออกแบบได้ ซึ่งการวิเคราะห์และออกแบบนี้สามารถแบ่งออกได้เป็น 2 ลักษณะ คือ วิธีการวิเคราะห์และออกแบบเชิงโครงสร้าง (Structured Methodology) โดยประกอบไปด้วยวิธีการมากมาย และหลายแนวทางปฏิบัติ ซึ่ง การวิเคราะห์ และออกแบบระบบแบบการไหลของกระแสข้อมูล (Data Flow Diagram) นั้นเป็นส่วนหนึ่งของวิธีการนี้ด้วย อีกลักษณะหนึ่งของการวิเคราะห์และออกแบบ คือ วิธีการวิเคราะห์และออกแบบเชิงวัตถุ (Object-Oriented Methodology) โดยวิธีนี้ประกอบไปด้วยหลาย ๆ ส่วนด้วยกัน ซึ่งการวิเคราะห์เชิงวัตถุ (Object-Oriented Analysis) และการออกแบบเชิงวัตถุ (Object-Oriented Design) เป็นส่วนหนึ่งด้วยเช่นกัน ภาษาออกแบบเชิงแบบจำลอง (Unified Modeling Language, UML) นั้นเป็นภาษาที่เป็นมาตรฐานที่ใช้วิเคราะห์และออกแบบระบบด้วยรูปแบบของกราฟิกสัญลักษณ์ในรูปแบบต่าง ๆ หลักการและแนวคิดของภาษาออกแบบเชิงแบบจำลองนั้นต้องการแยกเทคโนโลยี และเทคนิคการพัฒนาระบบแบบต่าง ๆ ออกมาจากการทำงานของระบบที่พัฒนา แนวคิดที่ว่านี้ถูกเรียกว่าสถาปัตยกรรมขับเคลื่อนด้วยแบบจำลอง (Model Driven Architecture, MDA) การที่แยกแนวคิดของการพัฒนาระบบ ออกมาจากเทคโนโลยีต่าง ๆ นั้น ทำให้นักวิเคราะห์และออกแบบสามารถทำการวิเคราะห์และออกแบบระบบได้อย่างไม่มีข้อกังวลเกี่ยวกับผลกระทบจากส่วนอื่น ๆ และในส่วนของเทคโนโลยี และเทคนิคการพัฒนาระบบแบบต่าง ๆ ก็ได้รับประโยชน์จากแนวคิดนี้ด้วยเช่นกัน เพราะทำให้สามารถนำเทคโนโลยีแบบต่าง ๆ ไม่ว่าจะเก่าหรือใหม่มาปรับใช้กับการพัฒนาระบบ ซึ่งรวมไปถึงเทคนิคและภาษาที่จะนำมาพัฒนาระบบด้วย ด้วยปัญหาที่พบในการพัฒนาระบบซอฟต์แวร์ และรูปแบบของการเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming) ที่นำมาใช้พัฒนาในทุกวันนี้สามารถตอบสนองต่อการวิเคราะห์และออกแบบ รวมไปถึงการพัฒนาระบบได้เพียงส่วนมาก แต่ยังไม่ครอบคลุมทั้งหมดของปัญหาซึ่งอาจทำให้เกิดผลกระทบตามมาในภายหลัง สถาปัตยกรรมขับเคลื่อนด้วยแบบจำลองนั้นเอื้อประโยชน์ต่อการนำเทคนิคและวิธีการที่เหมาะสมมาปรับใช้ให้เกิดประโยชน์ต่อการพัฒนาโปรแกรมอย่างสูงสุด ทำให้สามารถนำเอาเทคนิคการเขียนโปรแกรมเชิงลักษณะ (Aspect-Oriented Programming)

มาปรับใช้ให้เข้ากันกับภาษาออกแบบเชิงแบบจำลอง เพื่อให้เกิดการอธิบายการทำงานอย่างครอบคลุมต่อปัญหาทั้งหมด

งานวิจัยนี้มุ่งที่จะศึกษาค้นคว้า ตลอดจนนำเสนอมาตรฐานและพัฒนาเครื่องมือที่เหมาะสมเพื่อใช้ช่วยในขั้นตอนของการวิเคราะห์และออกแบบระบบซอฟต์แวร์เชิงวัตถุ และใช้ความสามารถของการเขียนโปรแกรมเชิงลักษณะมาช่วยเพิ่มประสิทธิภาพของการทำงานของซอฟต์แวร์เชิงวัตถุให้ดีขึ้นด้วย ฉะนั้นวัตถุประสงค์ของงานวิจัยนี้คือเพื่อเพิ่มวิธีการ และเทคนิคของการวิเคราะห์และออกแบบระบบให้สามารถรองรับกับปัญหาต่าง ๆ ที่เกิดขึ้นจากการวิเคราะห์ , ออกแบบ, พัฒนา รวมไปถึงซ่อมบำรุง และการดูแลรักษาซอฟต์แวร์

ขั้นตอนการดำเนินงานวิจัยนี้จะแบ่งออกเป็นสองส่วน โดยส่วนแรกคือ การศึกษามาตรฐานการทำงานของภาษาออกแบบเชิงแบบจำลอง เพื่อวิเคราะห์หาลักษณะ และวิธีการออกแบบระบบซอฟต์แวร์เชิงวัตถุด้วยภาษาออกแบบเชิงแบบจำลอง หลังจากได้วิเคราะห์การทำงานของภาษาออกแบบเชิงแบบจำลองจนได้ลักษณะเด่น และด้อยแล้วจากนั้นนำลักษณะการทำงานของการเขียนโปรแกรมเชิงลักษณะเข้าไปประกอบเพื่อนำไปลดการทำงานในส่วนที่เป็นข้อด้อยของ ภาษาออกแบบเชิงแบบจำลอง โดยได้ศึกษาความเป็นไปได้ที่จะนำการเขียนโปรแกรมเชิงลักษณะมาผสมผสานกับการเขียนโปรแกรมเชิงวัตถุ ในส่วนของภาษาออกแบบเชิงแบบจำลองที่เป็นคลาสโครงสร้าง (Structure Class) ซึ่งก็คือแผนภาพของคลาส (Class Diagram) และนำเสนอเป็นมาตรฐานของภาษาออกแบบเชิงแบบจำลอง สำหรับการวิเคราะห์และออกแบบซอฟต์แวร์เชิงวัตถุที่ได้รองรับการทำงานของการเขียนโปรแกรมเชิงลักษณะ

การวิจัยในส่วนที่สองคือการพัฒนา และทดสอบประสิทธิภาพของเครื่องมือสำหรับการวิเคราะห์ และออกแบบระบบซอฟต์แวร์ด้วยภาษาออกแบบเชิงแบบจำลองที่ได้เพิ่มเติมความสามารถของการเขียนโปรแกรมเชิงลักษณะ โดยเครื่องมือที่ได้พัฒนาขึ้นนั้นถูกสร้างขึ้นมาเพื่อสนับสนุนสำหรับการนำภาษาออกแบบเชิงแบบจำลองไปใช้ได้อย่างถูกต้อง และมีประสิทธิภาพ โดยจะสนับสนุนการทำงานของภาษาออกแบบเชิงแบบจำลอง ในส่วนของการเขียนโปรแกรมเชิงวัตถุซึ่งเป็นมาตรฐานเดิมก่อนที่จะได้ทำการเพิ่มเติมขึ้นได้เหมือนเดิม เครื่องมือสำหรับการวิเคราะห์และออกแบบระบบซอฟต์แวร์ด้วยภาษาออกแบบเชิงแบบจำลองนี้ จะสนับสนุนการออกแบบของการเขียนโปรแกรมเชิงวัตถุ และเชิงลักษณะ ในส่วนของภาษาออกแบบเชิงแบบจำลองที่เป็นคลาสโครงสร้าง ตามที่ได้นำเสนอตามมาตรฐานที่ได้กล่าวมาในส่วนของการพัฒนาในส่วนแรก ทั้งนี้ทางเครื่องมือจะสามารถสนับสนุนการออกแบบ และรวมไปถึงการสร้างโค้ดสำหรับพัฒนาซึ่งจะในไปใช้ในกระบวนการถัดไปได้อย่างถูกต้องก็ต่อเมื่อทางผู้

วิเคราะห์ และออกแบบระบบนั้น ได้ทำการออกแบบแผนภาพของภาษาออกแบบเชิงแบบจำลองได้อย่างถูกต้องตามมาตรฐาน

ในการวิจัยนี้จะทดสอบประสิทธิภาพและความถูกต้องของการทำงานของเครื่องมือ โดยจะอาศัยชุดของกรณีศึกษาของการออกแบบระบบซอฟต์แวร์ที่เสมือนจริงแต่ได้มีการปรับปรุง และเปลี่ยนแปลงลักษณะการทำงาน และข้อมูลบางชนิด เพื่อวิเคราะห์ผลลัพธ์ของการออกแบบระบบด้วยภาษาออกแบบเชิงแบบจำลองที่ได้เพิ่มเติมความสามารถของการเขียนโปรแกรมเชิงลักษณะ

5.1 สรุปผลการวิจัย

5.1.1 สรุปการนำเสนอมาตรฐานเพิ่มเติมของภาษาออกแบบเชิงแบบจำลองด้วยเทคนิคการเขียนโปรแกรมเชิงลักษณะ (Enhancing Aspect-Oriented for Unified Modeling Language)

การเพิ่มเติมมาตรฐานและสัญลักษณ์ของภาษาออกแบบเชิงลักษณะ ได้ถูกแบ่งออกเป็น 2 ส่วน ได้แก่

1) การนำเสนอมาตรฐานเพิ่มเติมของคลาสเชิงลักษณะมารวมกับมาตรฐานของคลาสเชิงวัตถุเดิม ซึ่งลักษณะของคลาสที่ได้เพิ่มเติมนั้นเป็นแบบคลาสโครงสร้าง เรียกว่าแผนภาพของคลาสเชิงวัตถุ ซึ่งหลังจากได้นำเอาคลาสเชิงลักษณะเข้าไปเพื่อทำงานร่วมกัน ทำให้เกิดเป็นแผนภาพใหม่ซึ่งเรียกว่าแผนภาพของคลาสเชิงลักษณะ (Aspect Class Diagram) โดยแผนภาพของคลาสเชิงลักษณะนั้นจะยังสามารถใช้ความสามารถของภาษาออกแบบเชิงโมเดลเดิมได้ตามปกติ
2) การนำเสนอมาตรฐานของความสัมพันธ์ที่จะนำความสามารถของคลาสเชิงลักษณะออกมาใช้ทำงานให้เข้ากันกับคลาสเชิงวัตถุ และความสัมพันธ์ที่เกิดขึ้นระหว่างคลาสเชิงลักษณะนั้นจะถูกเรียกว่าความสัมพันธ์ร่วม (Joint Relation) โดยมีสัญลักษณ์เป็นเส้นประ ความสัมพันธ์แบบร่วมนั้นจะเป็นตัวบอกว่าคลาสเชิงลักษณะนั้นจะทำงานร่วมกันกับคลาส หรืออินเทอร์เฟสใด ๆ

5.1.2 สรุปผลการทดสอบประสิทธิภาพและการทำงานของเครื่องมือสำหรับออกแบบภาษาออกแบบเชิงแบบจำลองที่เพิ่มเติมความสามารถของการเขียนโปรแกรมเชิงลักษณะ

การทดสอบการทำงานของเครื่องมือสำหรับออกแบบภาษาออกแบบเชิงแบบจำลองนี้สามารถแบ่งการสรุปออกได้เป็น 3 ส่วนด้วนกันได้ แก่ ส่วนออกแบบ , ส่วนการแปลงรูปข้อมูล และ ส่วนการสร้างโครงสร้างคลาส

1) ส่วนออกแบบ ทางเครื่องมือได้เตรียมสัญลักษณ์ออกแบบกราฟิกของภาษาออกแบบเชิงแบบจำลองไว้ให้ โดยมีสัญลักษณ์ของคลาสดังต่อไป นี้
   - สัญลักษณ์ของคลาสเชิงวัตถุ ซึ่งภายในจะประกอบไปด้วยคุณสมบัติ (Property or Attribute) และ การทำงานหรือพฤติกรรม (Behavior or Method)
   - สัญลักษณ์ของอินเทอร์เฟส ซึ่งภายในจะประกอบไปด้วยลักษณะของการทำงานหรือพฤติกรรมของอินเทอร์เฟส
   - สัญลักษณ์ของอินเทอร์เฟส ซึ่งภายในจะประกอบไปด้วย การทำงานของจุดตัด (Pointcut) ที่เกิดจากรอยต่อ (Joint Point) ระหว่างคลาสเชิงวัตถุและคลาสเชิงลักษณะ
   - สัญลักษณ์ของความสัมพันธ์ โดยจะมีความสัมพันธ์ต่าง ๆ ระหว่างคลาสเชิงวัตถุกับคลาสเชิงวัตถุเอง และจะมีความสัมพันธ์ระหว่างคลาสเชิงลักษณะกับคลาสเชิงวัตถุ คือความสัมพันธ์ร่วมอยู่ด้วย
2) ส่วนของการแปลงข้อมูล นั้นจะทำการแปลงกราฟิกสัญลักษณ์ของภาษาออกแบบเชิงโมเดลให้อยู่ในรูปเอกสารแบบการจำลองวัตถุ (Document Object Model, DOM) เพื่อที่จะได้สามารถปรับปรุง แก้ไข และนำไปใช้ในส่วนถัดไปได้อย่างถูกต้องตามความต้องการของผู้ออกแบบ

ส่วนการแปลงข้อมูลนั้นจะแปลงกราฟิกตามที่ได้ถูกออกแบบมาจากส่วนออกแบบ โดยไม่สามารถบอกได้ว่าสัญลักษณ์ที่ถูกออกแบบมานั้นถูกต้องตามที่ภาษาออกแบบเชิงแบบจำลองนั้นได้นิยามหรือไม่ ดังนั้นเพื่อความถูกต้องของการแปลงเอกสารควรตรวจสอบสัญลักษณ์ว่าได้ถูกนำมาใช้ได้อย่างถูกต้อง

3) ส่วนของการแปลงเอกสารแบบการจำลองวัตถุให้เป็นโครงสร้างคลาสนั้น จะทำการนำเอาเอกสารที่ถูกต้องตามรูปแบบที่เครื่องมือได้กำหนดมาสร้างเป็นโครงสร้างของคลาสทั้ง 2 แบบ คือ คลาสเชิงวัตถุ และคลาสเชิงลักษณะ ซึ่งเครื่องมือจะสามารถทำงานได้อย่างถูกต้องก็ต่อเมื่อเอกสารแบบการจำลองวัตถุนั้นอยู่ในรูปแบบที่กำหนดเอาไว้

จากข้อสรุปที่ได้กล่าวไปทั้งหมดแล้วนั้น จะเห็นได้ว่า ภาษาออกแบบเชิงแบบจำลองที่เพิ่มความสามารถของการเขียนโปรแกรมเชิงลักษณะนั้นสามารถช่วยให้ผู้ที่วิเคราะห์และออกแบบระบบซอฟต์แวร์มีทางเลือก และสามารถแก้ปัญหาต่าง ๆ ที่เกิดขึ้นจากการพัฒนาซอฟต์แวร์ได้มากขึ้น และเครื่องมือที่พัฒนาขึ้นมานั้นเป็นตัวช่วยที่ทำให้ผู้ที่วิเคราะห์และออกแบบ

ระบบนั้นสามารถจัดการกับระบบได้อย่างสะดวก และตรงตามวัตถุประสงค์ของสัญลักษณ์ ซึ่งจะช่วยให้ระบบที่ออกแบบมานั้นสามารถทำงานได้อย่างถูกต้อง

5.2 การประยุกต์ผลการวิจัย

ภาษาออกแบบเชิงแบบจำลองเป็นภาษาที่ใช้สำหรับการวิเคราะห์และออกแบบระบบ ซึ่งเป็นส่วนหนึ่งของกระบวนการพัฒนาซอฟต์แวร์ (Software Process) หรือวัฏจักรการพัฒนาซอฟต์แวร์ (Software Development Life Cycle, SDLC) อยู่แล้ว ดังนั้นเพื่อที่จะแก้ปัญหาที่อาจจะเกิดขึ้นจากการพัฒนาซอฟต์แวร์ ภาษาออกแบบเชิงแบบจำลองที่เพิ่มความสามารถของการเขียนโปรแกรมเชิงลักษณะเป็นทางเลือกหนึ่งที่จะแก้ปัญหาที่เกิดจากกระบวนการ หรือจากการวิเคราะห์และออกแบบซอฟต์แวร์นั้น แต่ภาษาออกแบบเชิงแบบจำลองเป็นภาษาที่ใช้สำหรับวิเคราะห์และออกแบบซอฟต์แวร์เชิงวัตถุ เพราะฉะนั้นการที่จะใช้ประโยชน์จากภาษาออกแบบเชิงแบบจำลองที่เพิ่มความสามารถของการเขียนโปรแกรมเชิงลักษณะให้ได้อย่างเต็มที่นั้น ทางผู้ที่พัฒนาระบบหรือผู้ที่วิเคราะห์และออกแบบระบบก็ควรจะใช้วิธีการหรือกระบวนการพัฒนาซอฟต์แวร์เชิงวัตถุด้วย

5.3 ข้อเสนอแนะในการวิจัยต่อไป

สำหรับงานทางด้านวิเคราะห์และออกแบบระบบซอฟต์แวร์เชิงวัตถุ มักจะพบว่าเกิดปัญหาบางอย่างที่เกิดจากหลักการของการพัฒนาซอฟต์แวร์เชิงวัตถุเอง อีกทั้งปัญหานั้นไม่อาจจะแก้ได้ด้วยเทคนิคของการพัฒนาซอฟต์แวร์เชิงวัตถุ ดังนั้นทุก ๆ แผนภาพของภาษาออกแบบเชิงแบบจำลองซึ่งถูกระบุ และสร้างมาเพื่ออธิบายการทำงานของการพัฒนาซอฟต์แวร์เชิงวัตถุนั้นก็มักจะพบปัญหาที่เกิดจากตัวหลักการที่ว่านี้ด้วย เทคนิคการเขียนโปรแกรมเชิงลักษณะเป็นการเขียนโปรแกรมที่ถูกสร้างขึ้นมาเพื่อแก้ปัญหาของการเขียนโปรแกรมเชิงวัตถุ และจะต้องทำงานร่วมกับกับการเขียนโปรแกรมเชิงวัตถุนี้ด้วย โดยการเขียนโปรแกรมเชิงลักษณะไม่สามารถนำไปพัฒนาระบบซอฟต์แวร์ด้วยตัวมันเองได้ ซึ่งจะเห็นได้ว่าเทคนิคการเขียนโปรแกรมเชิงลักษณะเป็นเพียงเทคนิคสนับสนุนของเทคนิคการพัฒนาซอฟต์แวร์เชิงวัตถุ สำหรับลักษณะแนวทางวิจัยที่จะพัฒนาต่อไปนั้นจะสามารถแบ่งออกได้เป็นหลายแนวทางดังนี้

- วิเคราะห์แผนภาพต่าง ๆ ของภาษาออกแบบเชิงแบบจำลอง เพื่อหาข้อบกพร่องที่เป็นของเทคนิคการพัฒนาซอฟต์แวร์เชิงวัตถุ
- นำเทคนิคของการเขียน และพัฒนาซอฟต์แวร์เชิงลักษณะมาทำการปรับปรุงแผนภาพต่าง ๆ ของภาษาออกแบบเชิงแบบจำลอง ซึ่งลักษณะของแผนภาพของภาษาออกแบบ

เชิงแบบจำลองนั้นมีหลายประเภท ดังนั้นการที่จะปรับปรุงนั้นควรดูความเหมาะสม และหน้าที่ของแผนภาพแต่ละประเภทด้วย

- การพัฒนาสัญลักษณ์ประเภทต่าง ๆ ที่นำมาใช้อธิบายการทำงานของการเขียนโปรแกรมเชิงลักษณะในแต่ละแผนภาพ
- การพัฒนาเครื่องมือสำหรับขั้นตอนการพัฒนา ซึ่งจะเป็นเครื่องมือสำหรับใช้แนะนำว่าจุดใดของการเขียนโปรแกรมเชิงวัตถุควรจะใช้เทคนิคของการเขียนโปรแกรมเชิงลักษณะเข้ามาช่วย

รายการอ้างอิง

Kiczales,G. et.al. (1997). Aspect-Oriented Programming. InProceedings of the European Conference on Object-Oriented Programming (ECOOP), LNCS 1241, Springer-Verlag.

Lopes,C. V., and Kiczales,G. (1998). Recent Developments in AspectJ. In ECOOP'98 Workshop Reader, LNCS 1543,Springer-Verlag.

Suzuki, J. and Yamamoto, Y. (1998). Managing the Software Design Documents with XML. In Proceedings of ACM SIGDOC'98, Quebec City, Canada.

Suzuki , J. and Yamamoto ,Y. (1999). Extending (UML) with Aspects: Aspect Support in the Design Phase. (ECOOP) Workshops'99.

De Win ,B., Vanhaute ,B., and De Decker ,B. (2001). Towards an open weaving process. In OOPSLA 2001 Workshop on Advanced Separation of Concerns in Object-Oriented System.

Mellor, S.J., Scott, K., Uhl A., Weise and D. (2004). MDA Distilled: Principles of Model-Driven Architecture. Addison Wesley.

Sauer ,S.,Engels, G. (1999).Extending UML for Modeling of Multimedia Application.

Klien ,C. Rausch ,A.Shiling ,M.Wen ,Z. (2001). Extension of Unified Modeling Language for mobile agents. In Siau K. and Halpin T. (Eds) UML. Sytsem Analysis , Design and Development Issues. Idea Group Publishing.

OMG. (2002). Meta Object Facility (MOF) Specification Version 1.4

QVT-Partners. (2003). Revised Submission for MOF 2.0 Query / Views / Transformation RFP.

Object Management Group, Unified Modeling Language Specification version 1.5.

Kiczales,G AspectJ .(2002). http://www.eclipse.org/aspectj.

W3C. (2005). Document Object Model (DOM). http://www.w3.org/DOM.

OMG.(2002).Model Driven Architecture .http://www.omg.org/mda.

ภาคผนวก ก

บทความผลงานวิจัยที่นำเสนอในการประชุมวิชาการวิทยาศาสตร์และเทคโนโลยีแห่งประเทศไทย ครั้งที่ 32

การเพิ่มความสามารถเชิงลักษณะใน UML

Enhancing aspect-oriented for UML

ศิวดล เสถียรพัฒนากูล, ผศ.ดร พิชโยทัย มหัทธนาภิวัฒน์

Siwadol Sateanpattanakul, Pichayothai Mahatthanapiwat

School of Computer Engineering, Suranaree University of Technology, Nakhon Ratchasima 30000, Thailand, E-mail address: sidol.sat@gmail.com, pmh@sut.ac.th

บทคัดย่อ: ในการพัฒนาซอฟต์แวร์ในยุคปัจจุบัน วิธีการพัฒนาซอฟต์แวร์เชิงวัตถุถือว่าเป็นหนึ่งในวิธีที่ได้รับความนิยมเป็นอย่างมาก ทั้งยังมีกระบวนการอื่น ๆ ที่ถูกสร้างมาสนับสนุนการพัฒนาซอฟต์แวร์เชิงวัตถุนี้ด้วย Unified Modeling Language (UML) เป็นอีกหนึ่งมาตรฐานซึ่งถูกออกแบบมาให้เหมาะสมกับการใช้งานร่วมกันกับการพัฒนาซอฟต์แวร์เชิงวัตถุนี้ แต่ข้อจำกัดของการพัฒนาซอฟต์แวร์เชิงวัตถุนั้นยังมีอยู่ ดังนั้นการนำเอาการพัฒนาซอฟต์แวร์แบบ aspect มาช่วยจะทำให้แก้ปัญหาของการพัฒนาซอฟต์แวร์เชิงวัตถุได้ การวิจัยนี้ได้เพิ่มความสามารถของ aspect เข้าไปกับ UML ที่เป็นมาตรฐาน รวมทั้งได้นำเอา XML มาใช้งานร่วมในฐานะเอกสารข้อมูล โดยทำหน้าที่เป็นตัวกลางระหว่างโมเดลและโค้ด

Abstract: Object-oriented paradigm is one of the most widely used techniques for software development at present. Unified Modeling Language (UML) is a standard used for object-oriented software development. However, there are still some limitations in object-oriented software development. The aspect-oriented paradigm will solve these limitations. This research will extend the capability of the UML by including the characteristic of the aspect paradigm. Moreover, XML will be used as a document to transfer data between UML and source code.

Introduction: Java programming is widely used in the development of many applications in object-oriented paradigm. Moreover, it is appropriate for Software Development Life Cycle (SDLC) that uses UML in process.

Although, java con be used for the abstraction principle making less complexity, there are still some problems in code management. Aspect-oriented paradigm is the way to solve this problem because it has the solution to manage cross-cutting problem that reduces the problem of code tangling and make software structure clean and configurable.

This paper addresses the aspect-oriented in design that leads to the implement phase. The aspect-oriented is included in the UML to support the design phase. Then we used XML to change from the model to the source code. This system consists of 3 sections as follows: Design section, Transformation section and Generator section.

Methodology: The following is 3 section of the system**.**

1. **Design section:** It is used as a user interface. UML notation will be drawn to analyze the software system. UML notations consist of class diagram, relation and aspect diagram that extend from UML.

2. Transform section: This section will extract entities and relation from UML notation in Design section. We can divide this section to 3 subsections as follows.

2.1 Entity extractor: This part will extract UML entities that are Class, Interface, Component and Aspect these entities are kind of Classifier.

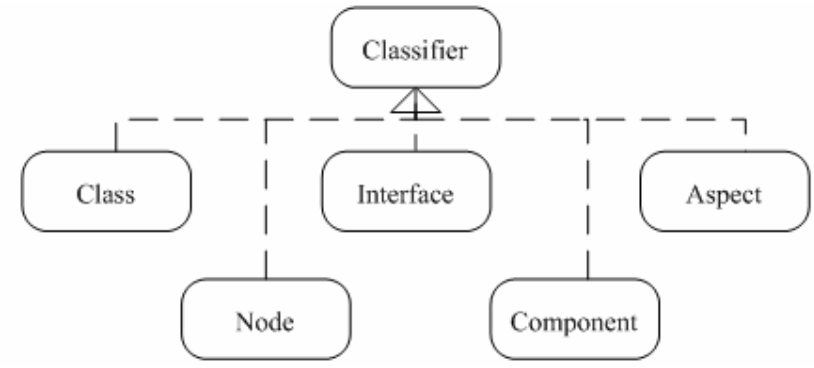


Figure 1: Aspect as a metamodel element derived from Classifier

2.2 Relation extractor: This section will extract UML relation between each UML Entity.

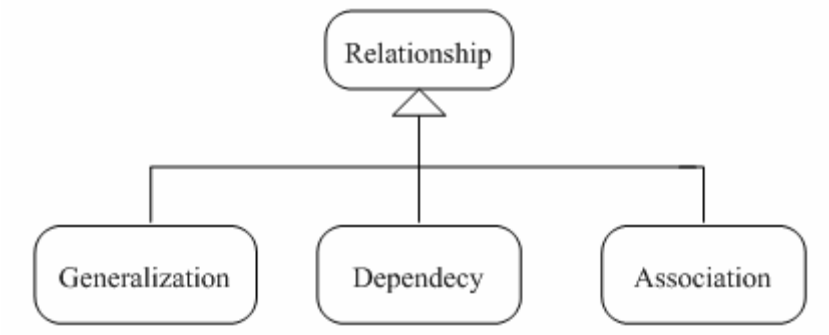


Figure 2: Some kind of Relationship element in the UML metamodel

2.3 Xml builder: Xml builder will get values that are extracted from above sub-section and write them into xml file with specified format.

```
-<Object-orcinted>
 -<class>
    <name></name>
    -<attribute>
      -<list>
         <value></value>
       </list>
     </attribute>
     -<operation>
       -<list>
          <value></value>
        </list>
      </operation>
      -<relation>
        <type></type>
        <class></class>
      </relation>
  </class>
</Object-oriented>
```

```
-<Aspect-oriented>
  -<class>
     <name></name>
     -<joint-point>
       -<class name="">
          -<method name="" return-type="">
             -<arg>
                <value></value>
              </arg>
            </method>
        </class>
      </joint-point>
   </class>
</Aspect-oriented>
```

Figure 3: Xml format file for java and aspectJ

3. Generator section: This section will generate xml file to Java and AspectJ source code. We can divide generator section to 2 subsections.

3.1 Xml parser: Xml parser will parse value from xml file from xml builder section.

3.2 Code generator: This is the final section that gets parsed value from XML parser.

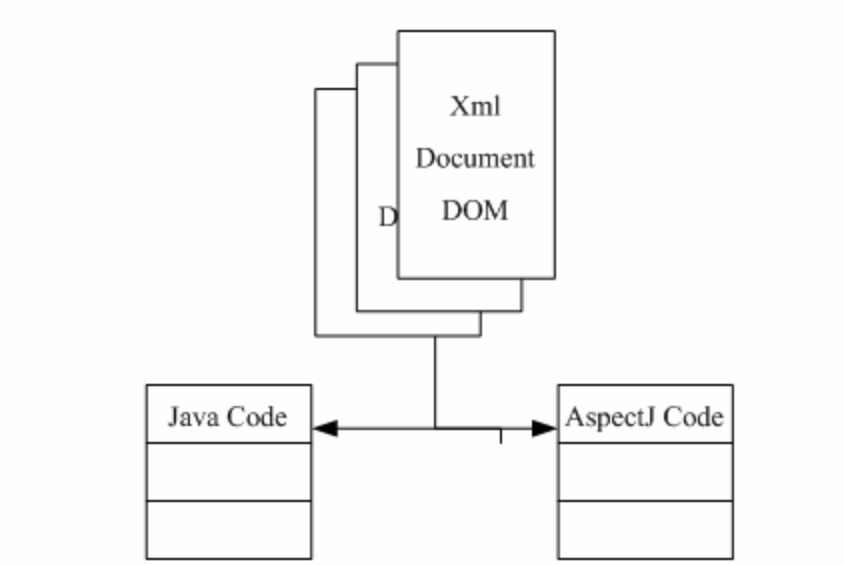


Figure 4: Source code output after Xml file pass Generator section

Conclusion: This system addressed design phase and implementation phase, and the aspect extended to UML. Then, this work proposed UML description language and Aspect description language based on XML. Furthermore, XML will be automatically generated to Java and AspectJ source code for used in implementation phase. Thus, this system will be useful for system analysts in the design phase and programmer in the implementation phase.

References:

1. G. Kiczales et.at. Aspect-Oriented Programming. In*Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, LNCS 1241,Springer-Verlag, 1997.

2. C. V. Lopes and Gregor Kiczales. Recent Developments in AspectJ. In *ECOOP'98 Workshop Reader*, Springer-Verlag LNCS 1543, 1998.

3. J. Suzuki and Y. Yamamoto. Managing the Software Design Documents with XML. In *Proceedings of ACM SIGDOC'98*, Quebec City, Canada, September 1998.

4. http://www.eclipse.org/aspectj/

5. http://www.omg.org

6. http ://www.w3.org/DOM/

Keywords: Unified Modeling Language (UML), AspectJ, XML, Aspect Oriented Programming (AOP), Software development lift cycle (SDLC).

ภาคผนวก ข

รูปแบบของเอกสารแบบการจำลองวัตถุที่ใช้ในเครื่องมือ และผลการทดสอบเครื่องมือ

รูปแบบของไฟล์เอกสารแบบการจำลองวัตถุของการเขียนโปรแกรมเชิงวัตถุ

เป็นการนำเอาเอกสารแบบการจำลองวัตถุของคลาสเชิงวัตถุมาอธิบายรายละเอียดของ Tag และรวมไปถึงความหมายต่าง ๆ ที่ใช้ในเครื่องมือ โดยมีรายละเอียดดังต่อไปนี้

```
<?xml version="1.0" encoding="UTF-8"?>
```

เป็นการเขียนบอกว่าเป็นเอกสารแบบ XML ที่มีการเข้ารหัสแบบ UTF-8

```
<Object-Oriented>
```

เป็นการบอกค่าเริ่มต้นว่าเป็นการไฟล์ของการทำงานแบบเชิงวัตถุ

```
<Interface name="Interface2">
```

Interface เป็น Tag ของอินเทอร์เฟส และ คุณลักษณะ name ที่อยู่ข้างในนั้นเป็นตัวบ่งบอกชื่อของ อินเทอร์เฟส

```
<generalize>Interface1</generalize>
```

generalize เป็น Tag ของความสัมพันธ์ระหว่างคลาสแม่กับคลาสลูก

```
</Interface>
```

เป็นการปิด Tag ของอินเทอร์เฟส

```
<Class name="Class2">
```

Class เป็น Tag ของคลาสเชิงวัตถุ และ คุณลักษณะ name ที่อยู่ข้างในนั้นเป็นตัวบ่งบอกชื่อของ คลาส

```
<generalize>Class1</generalize>
```

generalize เป็น Tag ของความสัมพันธ์ระหว่างคลาสแม่กับคลาสลูก

```
</Class>
```

เป็นการปิด Tag ของคลาสเชิงวัตถุ

```
<attribute name="attr1" type="int" accessType="private" isStatic="false" values="1" />
```

attribute เป็น Tag ของคุณลักษณะที่อยู่ภายในคลาส หรือ อินเทอร์เฟส ซึ่งจะมีค่าต่าง ๆ ที่บ่งบอกลักษณะของคุณลักษณะ ดังนี้

- name เป็นชื่อของคุณลักษณะ หรือคุณสมบัติ
- type เป็นชนิดของคุณลักษณะ หรือคุณสมบัติ
- accessType เป็นลักษณะการเข้าถึงของคุณลักษณะ หรือคุณสมบัติ
- isStatic เป็นการบอกลักษณะการเปลี่ยนแปลงค่าของคุณลักษณะ หรือคุณสมบัติ
- value นั้นเป็นการบอกจำนวนของคุณลักษณะ หรือคุณสมบัติซึ่งจะมีหรือไม่มีก็ได้

```
<operation name="operation1" returnType="void" accessType="public"
 isAbstract="false"values="1" />
```

operation เป็น Tag ของการทำงานหรือพฤติกรรมของคลาส หรือ อินเทอร์เฟส ซึ่งจะมีค่าต่าง ๆ ที่บ่งบอกการทำงานหรือพฤติกรรม ดังนี้

- name เป็นชื่อของการทำงานหรือพฤติกรรม
- returnType เป็นลักษณะของการส่งค่าคืนของการทำงานหรือพฤติกรรม
- accessType เป็นลักษณะการเข้าถึงของการทำงานหรือพฤติกรรม
- isAbstract เป็นการบอกลักษณะของการทำงานหรือพฤติกรรมว่าเป็นนามธรรมหรือไม่
- value นั้นเป็นการบอกจำนวนของการทำงานหรือพฤติกรรมซึ่งจะมีหรือไม่มีก็ได้

```
<realize>Interface1</realize>
```

realize เป็น Tag ของความสัมพันธ์ระหว่าง อินเทอร์เฟส และคลาส

```
</Object-Oriented>
```

เป็นการปิด Tag ไฟล์ของการทำงานแบบเชิงวัตถุ

รูปแบบของไฟล์เอกสารแบบการจำลองวัตถุของการเขียนโปรแกรมเชิงลักษณะ

เป็นการนำเอาเอกสารแบบการจำลองวัตถุของคลาสเชิงลักษณะมาอธิบายรายละเอียดของ Tag และรวมไปถึงความหมายต่าง ๆ ที่ใช้ในเครื่องมือ โดยมีรายละเอียดดังต่อไปนี้

```
<?xml version="1.0" encoding="UTF-8"?>
```

เป็นการเขียนบอกว่าเป็นเอกสารแบบ XML ที่มีการเข้ารหัสแบบ UTF-8

```
<Aspect-Oriented>
```

เป็นการบอกค่าเริ่มต้นว่าเป็นการไฟล์ของการทำงานแบบเชิงลักษณะ

```
<Aspect-Class name="Aspect1" callClass="Interface1">
```

Aspect-Class เป็น tag ของคลาสเชิงลักษณะ โดยภายในจะมีคุณลักษณะดังนี้

- name เป็นชื่อของคลาสเชิงลักษณะ
- callClass เป็นคลาส หรือ อินเทอร์เฟสที่คลาสเชิงลักษณะมีปฏิสัมพันธ์ด้วย

```
<pointcut name="pointcut1" callMethod="" returnType="void" values="1" />
```

pointcut เป็น tag ของการทำงานหรือพฤติกรรมของคลาสเชิงลักษณะ โดยจะเรียกว่าจุดตัด โดยภายในจะมีคุณลักษณะดังนี้

- name เป็นชื่อของจุดตัด
- callMethod เป็นชื่อของคลาสเชิงวัตถุที่จุดตัดของคลาสเชิงลักษณะไปทำงาน และจัดการการทำงานแบบต่าง ๆ

```
</Aspect-Class>
```

เป็นการปิด Tag ของคลาสเชิงลักษณะ

```
</Aspect-Oriented>
```

เป็นการปิด Tag ไฟล์ของการทำงานแบบเชิงลักษณะ

ผลการทดสอบเครื่องมือจากกรณีศึกษา

จากการที่ได้นำกรณีศึกษาของหน่วยงานของส่วนราชการมาทำการออกแบบด้วยภาษาออกแบบเชิงแบบจำลองที่เพิ่มเติมความสามารถของการเขียนโปรแกรมเชิงลักษณะดังรูปที่ 4.11 และ 4.15 นั้นเมื่อได้นำแผนภาพมาผ่านกระบวนการแปลงข้อมูลแล้วปรากฏว่าได้ผลออกมาเป็นไฟล์ ดังรูป 4.16 และรายละเอียดของแต่ละไฟล์เป็นดังต่อไปนี้

- ไฟล์ government.oop.xml มีโครงสร้างดังนี้

```
<?xml version="1.0" encoding="UTF-8"?>
<Object-Oriented>
 <Class name="PartTime">
  <generalize>Employee</generalize>
 </Class>
 <Class name="Employee">
  <generalize>Officer</generalize>
 </Class>
 <Class name="FullTime">
  <generalize>Employee</generalize>
  <realize>HonorAble</realize>
 </Class>
 <Class name="HonorHistory">
  <one-to-one class="HonorAble" />
  <many-to-one class="Honor" />
 </Class>
 <Class name="Honor">
  <one-to-many class="HonorHistory" />
  <one-to-many class="HonorRequestingHistory" />
 </Class>
```

```
  <Class name="HonorRequestingHistory">
    <many-to-one class="Honor" />
 <many-to-one class="HonorAble" />
</Class>
<Interface name="HonorAble">
 <one-to-many class="HonorRequestingHistory" />
</Interface>
<Class name="GovernmentOfficer">
 <generalize>Officer</generalize>
 <realize>HonorAble</realize>
 <realize>PromoteAble</realize>
</Class>
<Class name="Promote">
 <one-to-one class="PromoteAble" />
</Class>
<Interface name="PromoteAble">
 <one-to-one class="Promote" />
 <one-to-many class="PositionAndSalaryHistory" />
</Interface>
<Class name="Salary">
 <one-to-many class="PositionAndSalaryHistory" />
</Class>
<Class name="PositionAndSalaryHistory">
 <many-to-one class="PromoteAble" />
 <many-to-one class="Salary" />
```

```
  <many-to-one class="Officer" />
</Class>
<Class name="WorkQuantity">
  <many-to-one class="Position" />
</Class>
<Class name="Office">
  <many-to-many class="WorkLine" />
</Class>
<Class name="SupportGroup">
  <one-to-many class="WorkLine" />
</Class>
<Class name="WorkLine">
  <many-to-many class="Office" />
  <many-to-one class="SupportGroup" />
  <one-to-many class="Position" />
</Class>
<Class name="Position">
  <one-to-many class="WorkQuantity" />
  <many-to-one class="WorkLine" />
  <one-to-one class="Officer" />
</Class>
<Class name="Seminar">
  <many-to-one class="Officer" />
</Class>
<Class name="Discipline">
```

```
    <many-to-one class="Officer" />
  </Class>
  <Class name="Leave">
    <many-to-one class="Officer" />
  </Class>
  <Class name="Cooperative">
    <many-to-one class="Officer" />
  </Class>
  <Class name="Officer">
    <one-to-many class="PositionAndSalaryHistory" />
    <one-to-one class="Position" />
    <one-to-many class="Seminar" />
    <one-to-many class="Discipline" />
    <one-to-many class="Leave" />
    <one-to-many class="Cooperative" />
    <attribute name="image" type="String" accessType="private" isStatic="false" values="1"
/>
    <attribute name="name" type="String" accessType="private" isStatic="false" values="2" />
  </Class>
</Object-Oriented>
```

- ไฟล์ government_controller.oop.xml มีโครงสร้างดังนี้

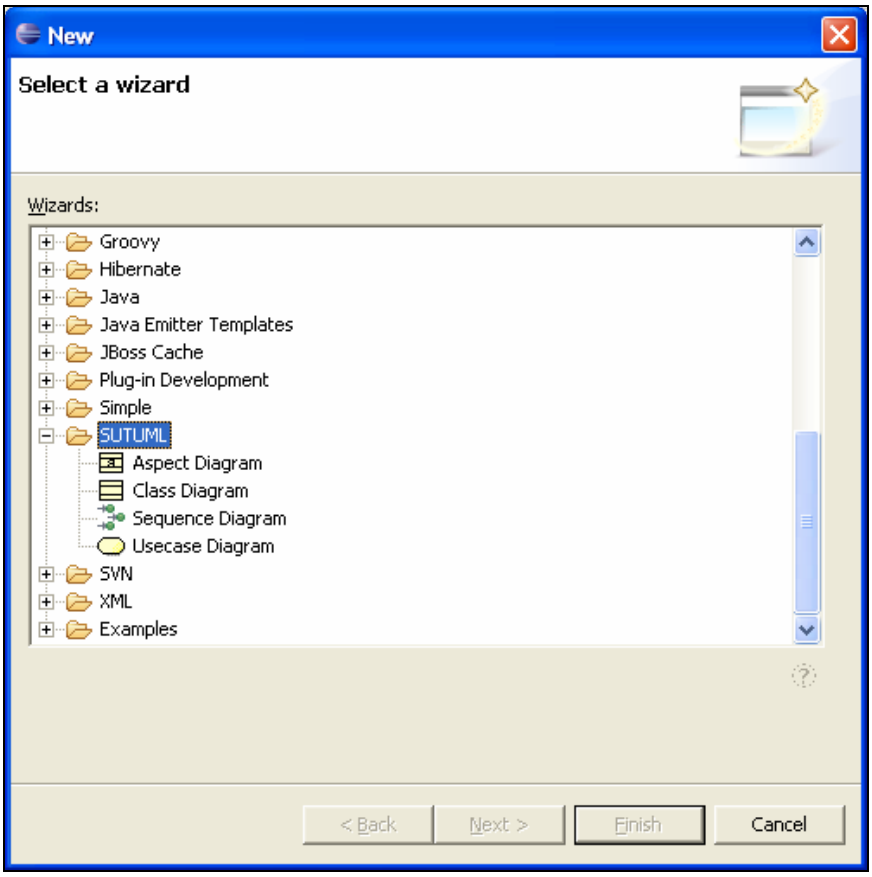
```
<?xml version="1.0" encoding="UTF-8"?>
<Object-Oriented>
 <Class name="HonorHandler">
  <generalize>SessionUtil</generalize>
 </Class>
 <Class name="HonorHistory">
  <generalize>SessionUtil</generalize>
 </Class>
 <Class name="PromoableDAO">
  <generalize>SessionUtil</generalize>
 </Class>
 <Class name="SalaryDAO">
  <generalize>SessionUtil</generalize>
 </Class>
 <Class name="OfficerDAO">
  <generalize>SessionUtil</generalize>
 </Class>
 <Class name="HornorAbleDAO">
  <generalize>SessionUtil</generalize>
 </Class>
```

คู่มือการใช้งาน

วิธีการติดตั้ง SUTUML

- ทำการดาวน์โหลด Eclipse IDE มาจากเวบไซต์ http://eclipse.org จากนั้นทำการแกะไฟล์ที่ถูกบีบอัดให้อยู่ในโฟลเดอร์ eclipse
- ทำการติดตั้งส่วนเพิ่มเติมของ Eclipse IDE ซึ่งได้แก่ EMF and UML2 จากเวบไซต์ http://eclipse.org
- นำไฟล์ Jar ของ SUTUML ไปวางในโฟลเดอร์ plugin ซึ่งอยู่ในโฟลเดอร์ eclipse ที่ได้ถูกแกะออก

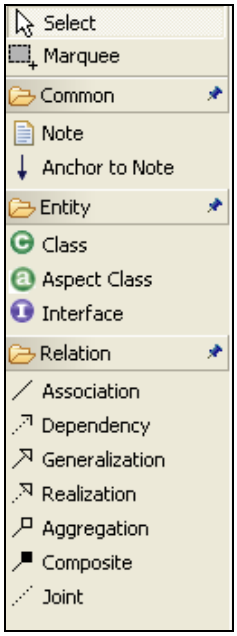
โดยหลังจากติดตั้งสำเร็จจะปรากฏ plugin ของ SUTUML ขึ้นใน Eclipse IDE โดยจะเป็นดังรูป



plugin ของ SUTUML เมื่อถูกติดตั้งสำเร็จ

วิธีการใช้งาน SUTUML

- เปิด Eclipse IDE และทำการเลือก "File"->"New"->"Other"->"SUTUML"->"Aspect Class diagram" หลังจากนั้นไฟล์คลาสเชิงลักษณะ (*.acld) จะถูกสร้างขึ้น
- เปิดส่วนการแก้ไขของคลาสเชิงลักษณะ (Aspect Class Diagram Editor) และทำการใช้งาน pallete เพื่อทำการสร้างกราฟิกสัญลักษณ์ของภาษาออกแบบเชิงแบบจำลอง ดังรูป



ส่วนของ pallete สำหรับสร้างกราฟิกของ UML

- ถ้าต้องการเปลี่ยนรูปของ UML ที่ได้ออกแบบไว้ให้ทำการคลิกขวาที่ไฟล์นามสกุล .acld จากนั้นเลือก "UMLGen"->"ModelToDocument" โดยจะได้ไฟล์ที่เป็นเอกสารการจำลองวัตถุออกมา
- และถ้าต้องการเปลี่ยนรูปของนเอกสารการจำลองวัตถุ ให้ทำการคลิกขวาที่ไฟล์นามสกุล .xml จากนั้นเลือก "UMLGen"->"DocumentToClass" โดยจะได้ไฟล์ที่เป็นโครงสร้างของคลาสที่ได้ออกแบบไว้ออกมา

ประวัติผู้เขียน

นายศิวดล เสถียรพัฒนากูล เกิดเมื่อวันที่ 8 มิถุนายน พ.ศ. 2524 เริ่มเข้าศึกษาระดับปริญญาตรีที่สาขาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี จังหวัดนครราชสีมา สำเร็จการศึกษาเมื่อปี พ.ศ. 2546 ภายหลังจากสำเร็จการศึกษาได้เข้าทำงานกับสาขาวิชาวิศวกรรมคอมพิวเตอร์ในตำแหน่งผู้ช่วยสอน สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี จากการทำงานเกี่ยวกับด้านการสอนและงานทางด้านวิศวกรรมซอฟต์แวร์ ซึ่งทำให้ผู้วิจัยได้รู้จักตัวเองมากขึ้น จึงได้เกิดแรงจูงใจที่จะศึกษาต่อในระดับปริญญาโททางด้านวิศวกรรมซอฟต์แวร์เพื่อเป็นการพัฒนาความรู้ความสามารถให้กับตัวเอง ดังนั้นในปี พ.ศ. 2548 จึงได้เข้าศึกษาต่อในสาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี ในปี พ.ศ. 2548