กรอบงานการผูกความสัมพันธ์ข้อมูลเชิงวัตถุในชั้นการแสดงผลข้อมูลนามธรรมและข้อมูลเชิงสัมพันธ์

นายวุฒิพล หมัดเส็น

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์
มหาวิทยาลัยเทคโนโลยีสุรนารี
ปีการศึกษา 2551

FRAMEWORK OF OBJECT DATA BINDING IN PRESENTATION LAYER BETWEEN ABSTRACT DATA AND RELATIONAL DATA

Wuttipol Madsen

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 2008

กรอบงานการผูกความสัมพันธ์ข้อมูลเชิงวัตถุในชั้นการแสดงผลข้อมูลนามธรรม และข้อมูลเชิงสัมพันธ์

สภามหาวิทยาลัย มหาวิทยาลัยเทคโนโลยีสุรนารี อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

คณะกรรมการสอบวิทยานิพนธ์

_____

(รศ.ดร. กิตติศักดิ์ เกิดประสพ)
ประธานกรรมการ

_____

(ผศ.ดร. พิชโยทัย มหัทธนาภิวัฒน์)
อาจารย์ที่ปรึกษา

_____

(รศ. ดร. นิตยา เกิดประสพ)
กรรมการ

_____

(ดร. ปรเมศวร์ ห่อแก้ว)
กรรมการ

_____

(ศ. ดร. ไพโรจน์ สัตยธรรม)
รองอธิการบดีฝ่ายวิชาการ

_____

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

วุฒิพล หมัดเส็น : กรอบงานการผูกความสัมพันธ์ข้อมูลเชิงวัตถุในชั้นการแสดงผลข้อมูลนามธรรมและข้อมูลเชิงสัมพันธ์ (FRAMEWORK OF OBJECT DATA BINDING IN PRESENTATION LAYER BETWEEN ABSTRACT DATA AND RELATIONAL DATA) อาจารย์ที่ปรึกษา : ผู้ช่วยศาสตราจารย์ ดร.พิชโยทัย มหัทธนาภิวัฒน์, 131 หน้า

การวิเคราะห์และออกแบบเชิงวัตถุ (Object Oriented Analysis and Design) ได้ถูกนำมาใช้พัฒนาโปรแกรมประยุกต์ซึ่งนับวันเวลาผ่านไปยิ่งทวีความซับซ้อนมากยิ่งขึ้น โดยอาศัยแนวคิดเชิงวัตถุอันประกอบด้วย การสืบทอด (Inheritance) โมดูลาริตี (Modularity) การพ้องรูป (Polymorphism) และ การห่อหุ้มข้อมูล (Encapsulation) ทำให้การวิเคราะห์และออกแบบกระทำได้ง่าย มีประสิทธิภาพและรวดเร็วยิ่งขึ้น

ด้วยเหตุที่แต่เดิมนั้นใช้การออกแบบเชิงโครงสร้าง (Structured Analysis Approaches) อย่างการวิเคราะห์การไหลของข้อมูล (Data Flow Analysis) และการออกแบบข้อมูลเชิงสัมพันธ์ (Entity-Relation Modeling) จึงทำให้การออกแบบและโครงสร้างของฐานข้อมูลเป็นแบบฐานข้อมูลเชิงสัมพันธ์ (Relational Database) ปัญหาในการนำข้อมูลที่ถูกออกแบบให้มีความสัมพันธ์เชิงวัตถุเข้าไปเก็บในฐานข้อมูลเชิงสัมพันธ์ และการนำข้อมูลเชิงวัตถุนั้นไปแสดงผลในตารางแสดงผลเชิงสัมพันธ์จึงเกิดขึ้น

ในงานวิจัยนี้ได้นำเสนอกรอบงาน (Framework) ที่ใช้สำหรับการแสดงค่าหรือคุณสมบัติ (Properties) ของข้อมูลที่มีความสัมพันธ์เชิงวัตถุในตารางข้อมูล (JTable) รูปแบบปกติและแสดงในรูปแบบลักษณะประกอบ (JComponent) ต่าง ๆ นอกจากนี้การนำเสนอยังคงตรงตามแนวคิดของแบบจำลองโมเดล-วิว-คอนโทรล กล่าวคือหากมีการแก้ไขข้อมูลผ่านตารางข้อมูล คุณสมบัติของข้อมูลเหล่านั้นย่อมเปลี่ยนไปทั้งหมด ทั้งข้อมูลเชิงนามธรรมที่ถูกบรรจุอยู่ในสภาพแวดล้อมภายในโปรแกรมประยุกต์ (Object Container) เชื่อมโยงโดยตรงกระทั่งถึงฐานข้อมูลเชิงสัมพันธ์ โดยกรอบงาน (Framework) นี้ได้สนับสนุนสภาพแวดล้อม (Application Context) ต่าง ๆ ให้ทำงานได้โดยอัตโนมัติผ่านการกำหนดคุณสมบัติต่าง ๆ ทางเอกสารเฉพาะ (XML File) ซึ่งทำให้การพัฒนาโปรแกรมนั้นมีคุณภาพมากยิ่งขึ้นและตรงตามแบบแนวคิดเชิงวัตถุ

สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนักศึกษา_____

ปีการศึกษา 2551 ลายมือชื่ออาจารย์ที่ปรึกษา_____

WUTTIPOL MADSEN : FRAMEWORK OF OBJECT DATA BINDING IN PRESENTATION LAYER BETWEEN ABSTRACT DATA AND RELATIONAL DATA. THESIS ADVISOR : ASST. PROF. PICHAYOTAI MAHATTHANAPIWAT, Ph.D., 131 PP.

OBJECT – ORIENTED FRAMEWORK / OBJECT BINDING

The object oriented paradigm, which consists of the idea of inheritance, modularity, polymorphism and encapsulation, enables analysts and developers to rapidly produce the high performance and complicated applications.

Traditionally, domain analysis has been based on structured analysis approaches such as data flow analysis and entity-relation modeling. Consequently the problems have been occurred when presenting object oriented entities in presentation layer and storing them into relational database.

This thesis aims to construct the flexible framework to provide the model-view-control mechanisms and bind object-oriented data on the presentation layer, object instances in the container placed in the application context, through database layer without losing the object relationship. Furthermore, it allows domain application developers to present associated objects in JTable and automatically binds its properties into the table column in various JComponent. Furthermore, its hooks are defined by XML file. Consequently, it provides less coupling, supports implementing the high performance application and preserves the object oriented paradigm.

School of Computer Engineering Student's Signature_____

Academic Year 2008 Advisor's Signature_____

กิตติกรรมประกาศ

วิทยานิพนธ์นี้สำเร็จลุล่วงด้วยดี ผู้วิจัยขอกราบขอบพระคุณคณาจารย์ และบุคคลต่าง ๆ ที่กรุณาได้ให้คำปรึกษา แนะนำ ช่วยเหลือ อย่างดียิ่ง ทั้งในด้านวิชาการ และด้านการดำเนินการวิจัย ดังนี้

ผศ. ดร. พิชโยทัย มหัทธนาภิวัฒน์ อาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่ได้ประสิทธ์ประสาทวิชาความรู้ต่าง ๆ ในด้านวิศวกรรมซอฟต์แวร์ อีกทั้งยังให้คำปรึกษาแนะนำแนวทางการดำเนินงาน ตรวจทาน และแก้ไขวิทยานิพนธ์นี้จนเสร็จสมบูรณ์

ผศ. ดร. คะชา ชาญศิลป์ที่ได้ให้ความรู้และแนวคิดเกี่ยวกับการพัฒนาโปรแกรมประยุกต์โดยอาศัยเทคโนโลยีพื้นฐานเว็บ

อาจารย์ชาญวิทย์ แก้วกสิ ที่ให้คำปรึกษาและแนวคิดเกี่ยวกับความรู้ด้านวิชาการเกี่ยวกับการพัฒนาโปรแกรมประยุกต์โดยอาศัยแนวคิดเชิงวัตถุและการพัฒนากรอบงานเชิงวัตถุ

สุดท้ายนี้ ผู้วิจัยขอขอบคุณคณาจารย์ทุกท่านทั้งในอดีตและปัจจุบันที่ได้ประสิทธิ์ประสาทวิชาความรู้ต่าง ๆ กราบขอบพระบิดาคุณ มารดาที่คอยเป็นกำลังใจ เป็นที่ปรึกษาทั้งด้านสังคมและด้านการศึกษามาเป็นอย่างดีโดยตลอด และขอขอบคุณผู้มีพระคุณทุกท่านที่มิอาจกล่าวถึงได้ทั้งหมด ซึ่งเป็นแรงบันดาลใจให้ผู้วิจัยประสบความสำเร็จด้านการศึกษาเรื่อยมา

วุฒิพล หมัดเส็น

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญรูป

รูปที่ **หน้า**

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มาของปัญหาการวิจัย

ในยุคปัจจุบันเป็นยุคแห่งข้อมูลข่าวสารและโลกาภิวัตน์ ภาคธุรกิจ ภาคอุตสาหกรรมนั้นมีการแข่งขันในการเพิ่มอัตราการผลิต คุณภาพของผลิตภัณฑ์ สิ่งเหล่านี้เป็นตัวผลักดันให้เกิดข้อมูลจำนวนมากขึ้นและต้องการจัดการที่ดียิ่ง ระบบจัดการการข้อมูลแบบอัตโนมัติหรือซอฟต์แวร์ที่ทำงานด้วยระบบคอมพิวเตอร์นั้นถูกนำเข้ามาใช้ในการจัดการข้อมูลที่เกิดขึ้นอย่างรวดเร็วและต้องการการสังเคราะห์เพื่อเปลี่ยนจากข้อมูลดิบให้เป็นข้อมูลที่ใช้ในการวิเคราะห์ได้ ยิ่งโลกแห่งข้อมูลข่าวสารมีความซับซ้อนยิ่งขึ้นทำให้ซอฟต์แวร์ที่ใช้นั้นมีความซับซ้อนมากขึ้นและยากในการพัฒนา

Abadi and Cardelli (1996) และ Eliëns (2006) กล่าวถึงแนวคิดเชิงวัตถุ (Object Oriented Paradigm) นั้นได้รับการคิดค้นขึ้นโดยอาศัยพื้นฐานของการห่อหุ้ม (Encapsulation) สภาพส่วนจำเพาะ (Modularity) ภาวะพหุสัณฐาน (Polymorphism) และการสืบทอด (Inheritance) หลักการเชิงวัตถุเหล่านี้ทำให้กระบวนการพัฒนาซอฟต์แวร์เป็นไปอย่างรวดเร็ว เพิ่มอัตราการผลิตซอฟต์แวร์ (Productivity) เพิ่มความสามารถในการเพิ่มเติมซอฟต์แวร์ (Extensibility) และความสามารถในการนำกลับมาใช้ใหม่ (Reusability)

Krasner and Pope (1988) ได้ทำการยกตัวอย่างกระบวนการพัฒนาซอฟต์แวร์ ซึ่งประกอบไปด้วย การเก็บความต้องการในเชิงซอฟต์แวร์ (Software Requirement) การวิเคราะห์และออกแบบ (Software Analysis and Design) การพัฒนาโปรแกรม (Implementation) การทดสอบโปรแกรม (Testing) และการส่งมอบ (Deployment) โดยการวิเคราะห์และการออกแบบเชิงวัตถุ (Object Oriented Analysis and Design) แนวคิดอย่างโมเดล-วิว-คอลโทรล (Model-View-Control: MVC) นั้น อาศัยแนวคิดเชิงวัตถุในการวิเคราะห์และออกแบบซอฟต์แวร์ และถูกใช้ในการแบ่งแยกความเกี่ยวพันของความต้องการเชิงซอฟต์แวร์ (Separation of Concerns) ออกเป็นชั้น ๆ ได้แก่ ส่วนติดต่อกับผู้ใช้ (User Interface) ส่วนตรรกะในการทำงาน (Business Logic) และส่วนข้อมูล (Data Layer) วัตถุเชิงนามธรรม (Abstract Object) ที่ได้รับการออกแบบในขั้นการวิเคราะห์และออกแบบเชิงวัตถุนั้นถูกแทนที่ด้วยวัตถุที่อยู่ในแต่ละชั้น แนวคิดในการแบ่งความสัมพันธ์ออกเป็นชั้นนี้ ทำให้การพัฒนาซอฟต์แวร์เป็นไปอย่างมีประสิทธิภาพ เนื่องจาก

อย่างไรก็ตามปัญหาในการพัฒนาซอฟต์แวร์โดยอาศัยแนวคิดแบบ MVC นี้ก่อให้เกิดปัญหาขึ้นสองส่วนด้วยกันได้แก่

1. การเก็บวัตถุนามธรรมที่อยู่ในบริบทของโปรแกรมประยุกต์ (Application Context) เข้าในฐานข้อมูลเชิงสัมพันธ์ (Relational Database) โดยปราศจากการทำให้คุณสมบัติและความสัมพันธ์เชิงวัตถุนั้นหายไป เนื่องจากการเขียนโปรแกรมในเชิงวัตถุ (Object Oriented Programming) นั้นมักใช้แนวคิดในการออกแบบและเขียนโปรแกรมโดยพิจารณาข้อมูลเป็นวัตถุซึ่งเป็นปริมาณที่ไม่ใช่ปริมาณสเกลาร์ (Non Scalar Value) แต่อย่างไรก็ตาม ระบบจัดการฐานข้อมูลที่นิยมหลายตัวนั้นจัดการกับข้อมูลที่เป็นสเกลาร์เท่านั้น นักพัฒนาต้องเสียเวลามากในการเปลี่ยนค่าเชิงวัตถุให้อยู่ในรูปแบบกลุ่มของค่าอย่างง่ายก่อนเก็บค่าเหล่านั้นเข้าสู่ฐานข้อมูลเชิงสัมพันธ์
2. การนำข้อมูลวัตถุนามธรรมนั้นแสดงผลในส่วนแสดงผล (Presentation Layer) โดยรักษาลักษณะเชิงวัตถุไว้

แนวคิดของ Object-Oriented Mapping ถูกคิดขึ้นเพื่อแก้ปัญหาดังที่กล่าวมาแล้วในข้อแรก อีกทั้งมีผู้คิดส่วนโปรแกรมโดยอาศัยแนวคิดนี้จำนวนมาก ยกตัวอย่างเช่น Java Data Object (JDO), Enterprise Java Beans 3.0 (EJB3) และ Hibernate Technology ในส่วนของปัญหาที่สองนั้นยังคงมิได้มีการแก้ไขอย่างจริงจังจึงทำให้การพัฒนาโปรแกรมประยุกต์ในเชิงวัตถุนั้นยังคงต้องเสียเวลาในการแสดงผลวัตถุนามธรรมที่อยู่ในบริบทของโปรแกรมประยุกต์เช่นเคย

อนึ่งปัจจุบันมีผู้พัฒนาส่วนโปรแกรมที่สามารถเชื่อมคุณสมบัติเชิงวัตถุทั้งสามชั้นอย่าง JGoodies และการพัฒนาตามมาตรฐาน JSR295 ใน java. beans นั้นยังคงมีความซับซ้อน ยากแก่การเข้าใจ อีกทั้งยังใช้จำนวนรหัสคำสั่ง (Code) เป็นจำนวนมากอีกด้วย

งานวิจัยนี้จึงมุ่งเน้นในการสร้างกรอบงานเพื่อแก้ปัญหาทั้งสองข้อไปพร้อมกัน อีกทั้งยังได้อาศัยความสามารถของกรอบงานในการเพิ่มความยืดหยุ่น (Flexibility) ลดความยึดติด (Decoupling) และเพิ่มความเชื่อมแน่น (Cohesion) ให้กับซอฟต์แวร์หรือโปรแกรมประยุกต์ที่ทำงานภายใต้กรอบงานนี้ โดยกรอบงานเป็นส่วนของโปรแกรมที่ออกแบบและพัฒนาไว้เป็นแบบทั่วไป (Generic Component) พร้อมนำกลับมาใช้ใหม่สำหรับพัฒนาโปรแกรมประยุกต์ที่เฉพาะเจาะจง (Specific Domain) ในหลาย ๆ โปรแกรมโดยใช้กรอบงานเพียงกรอบงานเดียว กรอบงานนั้นถูกออกแบบด้วยส่วนโปรแกรมที่นำกลับมาใช้ใหม่ (Reusable Component) หลายส่วนประกอบไปด้วยคลังรหัสคำสั่ง (Code Libraries) รูปแบบเชิงซอฟต์แวร์ (Software Pattern) ภาษาเชิงสคริปท์ (Scripting Language) และอื่น ๆ กรอบงานถูกกำหนดส่วนที่เป็น API, Slot และ Hook ซึ่งสนับสนุน

1.2 วัตถุประสงค์การวิจัย

1) เพื่อศึกษาการสร้างกรอบงานเชิงวัตถุจากส่วนโปรแกรมต่าง ๆ เพื่อแก้ปัญหาในการแสดงผลและเก็บข้อมูลที่เป็นค่าเชิงวัตถุเข้าฐานข้อมูลเชิงสัมพันธ์โดยไม่ทำให้คุณสมบัติและความสัมพันธ์เชิงวัตถุของข้อมูลเหล่านั้นสูญเสียไป
2) เพื่อศึกษาการสร้างกรอบงานโดยกำหนดส่วนต่อประสาน (Interface) ให้มีความยืดหยุ่น สนับสนุนการเชื่อมต่อส่วนโปรแกรม (Component) ใหม่ที่อาจเกิดขึ้นโดยไม่สามารถคาดเดาได้
3) ศึกษาการสร้างกลไกให้กับกรอบงานในการเชื่อมต่อข้อมูลเชิงวัตถุที่อยู่ในส่วนแสดงผล ส่วนควบคุมและส่วนชั้นข้อมูลให้มีค่าที่สอดคล้องกัน (Synchronization)
4) ศึกษาการใช้ Software Pattern, Code Libraries และ Expression Language ในการสร้างกรอบงานเชิงวัตถุ
5) ศึกษาหลักการของ Object-Relational Mapping ในการเก็บและดึงข้อมูลเชิงวัตถุในฐานข้อมูลเชิงสัมพันธ์
6) ศึกษาการกำหนด Hook ให้กับกรอบงานเพื่อรองรับการกำหนดค่าตัวแปรเสริมผ่านทางเอกสารประเภท Extensible Markup Language (XML file)

1.3 ขอบเขตการวิจัย

1) สร้างกรอบงานจากส่วนโปรแกรมต่าง ๆ เพื่อแก้ปัญหาในการแสดงผลและเก็บข้อมูลที่เป็นค่าเชิงวัตถุเข้าฐานข้อมูลเชิงสัมพันธ์ โดยไม่ทำให้คุณสมบัติและความสัมพันธ์เชิงวัตถุของข้อมูลเหล่านั้นสูญเสียไป โดยอาศัยหลักการเชิงวัตถุ
2) สร้างกรอบงานโดยการกำหนดส่วนต่อประสานให้มีความยืดหยุ่น เพื่อสนับสนุนการเชื่อมต่อส่วนโปรแกรมใหม่ที่อาจเกิดขึ้นโดยไม่สามารถคาดเดาได้ โดยกรอบงานในงานวิจัยนี้กำหนดส่วนต่อประสานดังต่อไปนี้
    - ส่วนต่อประสาน (Interface) สำหรับเชื่อมต่อส่วนโปรแกรมที่ใช้สำหรับแสดงผลในตารางข้อมูล
    - ส่วนต่อประสานสำหรับเชื่อมต่อส่วนโปรแกรม ที่ใช้สำหรับติดต่อฐานข้อมูลเชิงสัมพันธ์

- ส่วนต่อประสานสำหรับเชื่อมต่อส่วนโปรแกรม ที่มีการกำหนดค่า และรูปร่างลักษณะของตารางข้อมูล และการเชื่อมต่อค่าเชิงวัตถุที่ถูกสร้างในบริบทของโปรแกรมประยุกต์

3) พัฒนากรอบงานด้วยภาษาจาวาซึ่งเป็นภาษาเชิงวัตถุเชื่อมข้อมูลที่มีความสัมพันธ์เชิงวัตถุกับฐานข้อมูลเชิงสัมพันธ์โดยงานวิจัยนี้ทดสอบกับฐานข้อมูล MySQL

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1) ได้กรอบงานเชิงวัตถุที่สามารถแก้ปัญหาในการแสดงผลและเก็บข้อมูลที่เป็นค่าเชิงวัตถุเข้าฐานข้อมูลเชิงสัมพันธ์โดยไม่ทำให้คุณสมบัติและความสัมพันธ์เชิงวัตถุของข้อมูลเหล่านั้นสูญเสียไปโดยอาศัยหลักการเชิงวัตถุ
2) การพัฒนาโปรแกรมประยุกต์ภายใต้กรอบงานเชิงวัตถุที่พัฒนาขึ้นจากงานวิจัยนี้สามารถเพิ่มอัตราการพัฒนาโปรแกรมประยุกต์ได้สูงขึ้น (Productivity)
3) การพัฒนาโปรแกรมประยุกต์ภายใต้กรอบงานเชิงวัตถุที่พัฒนาขึ้นจากกงานวิจัยนี้เพิ่มความยืดหยุ่นให้กับกระบวนการพัฒนาโปรแกรมประยุกต์ (Flexibility)
4) การพัฒนาโปรแกรมประยุกต์ภายใต้กรอบงานเชิงวัตถุที่พัฒนาขึ้นจากกงานวิจัยนี้ทำให้ความยึดติดของโปรแกรมประยุกต์นั้นลดลง (Decoupling)
5) กรอบงานเชิงวัตถุที่พัฒนาขึ้นจากกงานวิจัยนี้ทำให้ลดจำนวนและความซับซ้อนของรหัสคำสั่งในกระบวนการพัฒนาโปรแกรมประยุกต์
6) กรอบงานเชิงวัตถุที่พัฒนาขึ้นจากงานวิจัยนี้ทำให้กระบวนการพัฒนาโปรแกรมประยุกต์เป็นการพัฒนาเชิงวัตถุโดยสมบูรณ์

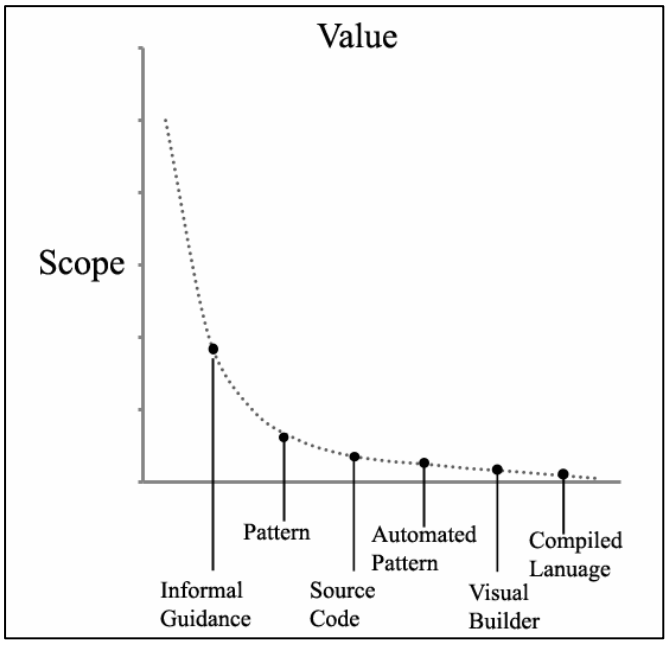
บทที่ 2
ปริทัศน์วรรณกรรมและงานวิจัยที่เกี่ยวข้อง

2.1 แนวคิดและเทคโนโลยีเชิงวัตถุ

Abadi and Cardelli. (1996) และ Eliëns (2006) กล่าวถึงแนวคิดเชิงวัตถุ (Object-Oriented Paradigm) ถูกสร้างขึ้นเพื่อแก้ไขปัญหาในการวิเคราะห์ ออกแบบ และพัฒนาซอฟต์แวร์ที่ทวีความซับซ้อนมากยิ่งขึ้น แนวคิดเชิงวัตถุนั้นนิยามกรอบในเชิงนามธรรม (Abstraction) สำหรับใช้ในการวิเคราะห์และจำแนกปัญหาต่าง ๆ ให้อยู่ในรูปแบบวัตถุอุปมาอุปไมยกับสิ่งที่มีอยู่จริง แนวคิดนี้เปลี่ยนจากการแบ่งแยกข้อมูล (Data) ออกจากการประมวลผล (Processing) กลับมารวมทั้งสองแบบเข้าด้วยกันโดยมีแนวคิดเชิงนามธรรมถึงวัตถุ (Object) หนึ่งวัตถุใดควรประกอบไปด้วยสถานะของวัตถุ (State) และพฤติกรรมของวัตถุ (Behavior) นั้น แนวคิดพื้นฐานในการเขียนโปรแกรมเชิงวัตถุนั้นประกอบไปด้วย

2.1.1 นามธรรม (Abstraction)

Abstraction กล่าวถึงแนวคิดเชิงวัตถุพยายามมุ่งเน้นการวิเคราะห์และออกแบบโดยการพิจารณาปัญหาในเชิงนามธรรมหรือพิจารณาในรูปแบบของสาระสำคัญเป็นหลัก พยายามไม่พิจารณาปัญหาต่าง ๆ ในรายละเอียด แนวคิดนี้ทำให้ปัญหาและความต้องการในเชิงซอฟต์แวร์ถูกวิเคราะห์และหาคำตอบในรูปแบบของ Generic Component ที่พร้อมนำกลับมาใช้ใหม่หรือแก้ปัญหาที่เกิดขึ้นใหม่แต่มีขอบเขตทำนองเดียวกัน



รูปที่ 2.1 แสดงขอบเขตของความเป็นนามธรรม

การเพิ่มระดับของความเป็นนามธรรมนั้นเป็นหนึ่งในกลวิธีทางกระบวนการพัฒนาซอฟต์แวร์โดยเป็นการนำความรู้ได้จากกระบวนการแก้ไขปัญหาแล้วนั้นมาใช้แก้ปัญหาเดิมที่เกิดขึ้นบ่อย ๆ จากรูปที่ 2.1 ซึ่งถูกนำเสนอโดย Pfleeger (1998) นั้นแสดงขอบเขตและคุณภาพของความเป็นนามธรรมอันเกิดจากการใช้กลวิธีแตกต่างกัน กราฟแสดงขอบเขตของความเป็นนามธรรมที่สูงจากการใช้ Informal Guidance และรูปแบบเชิงซอฟต์แวร์ (Software Pattern) โดยกลวิธีเหล่านี้ถูกเรียกว่า Black-Box Abstraction เนื่องจากผู้ใช้ไม่จำเป็นต้องเข้าใจถึงรายละเอียดว่าวิธีการเหล่านั้นแก้ปัญหาได้อย่างไรเพียงแต่ทราบถึงปัญหาใดที่จะถูกแก้และประยุกต์วิธีเหล่านั้นกับปัญหาที่เผชิญอย่างไร ในทางตรงกันข้าม Visual Builder และ Compiled Language นั้นมีขอบเขตและคุณค่าของความเป็นนามธรรมซึ่งสะท้อนถึงความสามารถในการนำกลับมาใช้ใหม่ที่น้อยอีกด้วย ด้วยวิธีการเหล่านี้ผู้ใช้หรือนักพัฒนาต้องทราบรายละเอียดในการแก้ปัญหา ดังนั้นกลวิธีเหล่านี้จึงถูกเรียกว่า White-Box Abstraction

2.1.2 การห่อหุ้ม (Encapsulation) หรือการปิดบังข้อมูล (Information Hiding)

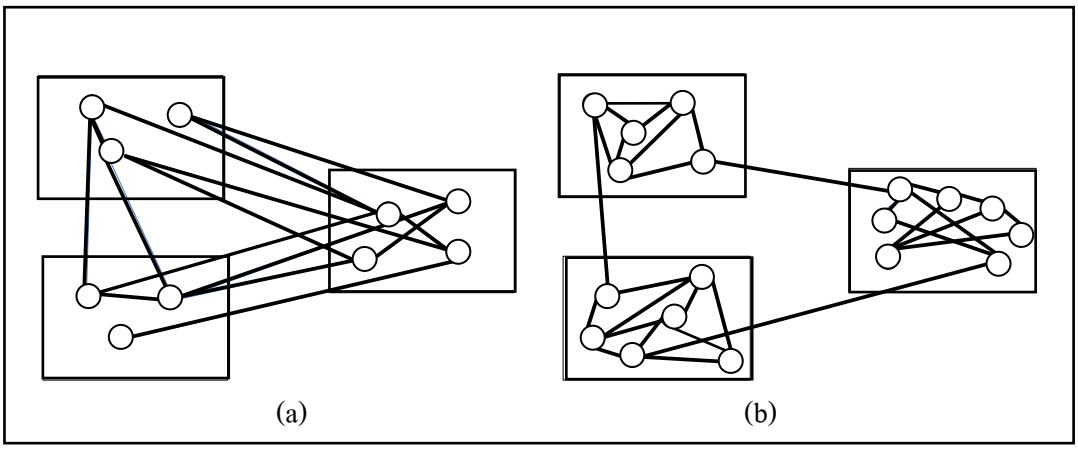
แนวคิดนี้สนับสนุนการวิเคราะห์และออกแบบโปรแกรมออกเป็นส่วน ๆ ในระดับคลาส กลุ่มของคลาส (Package) คลังของส่วนโปรแกรม (Software Libraries) และกรอบงาน (Software Framework) หลักการห่อหุ้มนี้สนับสนุนหลักการวิเคราะห์และออกแบบซอฟต์แวร์ให้มีประสิทธิภาพในส่วนการออกแบบโปรแกรมให้มีลักษณะเป็นส่วนจำเพาะ (Modularity) แนวคิดใน

```
class cell is
        var contents: Integer :=0;
        method get(): Integer is
                return selft.contents;
        end;
        method set(n: Integer) is
                if(n>0) then self.contents :=n;
       end;
end;
```

รูปที่ 2.2 แสดงการเขียนคลาส (Class) ของวัตถุ Cell

รูปที่ 2.2 แสดงตัวอย่างการเขียนคลาสตามหลักการห่อหุ้มโดยคลาส Cell มีตัวแปร contents แสดงคุณสมบัติโดยการส่งและกำหนดค่าให้กับตัวแปรของคลาสนั้นมีข้อกำหนดให้กระทำผ่าน Get และ Set Method ตามลำดับ จากตัวอย่างแสดงให้เห็นว่าค่าของตัวแปร contents นั้นต้องเป็นค่าที่ไม่เป็นลบเท่านั้น

อนึ่งการนิยามคลาสหรือส่วนโปรแกรมที่ถูกสร้างโดยกลุ่มของคลาสโดยหลักการห่อหุ้มและสภาพส่วนจำเพาะนั้นยังต้องคำนึงถึง Cohesion และ Coupling ด้วย



รูปที่ 2.3 แสดงการอธิบาย Cohesion และ Coupling เชิงรูปภาพ (a) แสดงถึงโครงสร้างของ Couple ที่สูง (b) แสดงโครงสร้างของ Cohesion ที่สูงแต่มี Coupling ที่ต่ำ

Ghezzi, Jazayeri, and Mandrioli (2002) กล่าวว่าในการวิเคราะห์และออกแบบส่วนโปรแกรมออกเป็นโมดูล (Module) นั้นเพื่อให้ได้ประสิทธิภาพสูงสุดควรออกแบบให้มี Cohesion ที่สูงและมี Coupling ต่ำ โดย Cohesion เป็นคุณสมบัติภายในของโมดูลซึ่งแสดงถึงการทำงานร่วมกันของส่วนต่าง ๆ ที่อยู่ภายในโมดูล หากส่วนเหล่านั้นทำงานร่วมกันได้อย่างมีประสิทธิภาพแล้วดังแสดงในภาพที่ 2.3(b) นั้นแสดงถึง Cohesion ที่สูง ในขณะที่ Coupling นั้นแสดงถึงคุณสมบัติของโมดูลในการเป็นอิสระต่อโมดูลอื่น ซึ่งหากมีการติดต่อหรือทำงานร่วมกันระหว่างโมดูลที่ต่ำแสดงถึงการอิสระต่อกันสูงดังภาพ 2.3(b) ซึ่งทำให้ง่ายในดูแลรักษาและทนทานต่อความต้องการเชิงซอฟต์แวร์ที่เปลี่ยนแปลงไป

2.1.3 การสืบทอด (Inheritance)

Abadi and Cardelli (1996) และ Simons (2003) นำเสนอแนวคิดในการสืบทอดและเพิ่มความสามารถทั้งคุณสมบัติและพฤติกรรมให้กับวัตถุซึ่งแสดงโดยพีชคณิตดังนี้

Subtype= Basetype U Extension และนิยามกฎเกี่ยวกับการสืบทอด 3 ข้อได้แก่

1) If c' is a subclass of c, and o': InstanceTypeOf(c'), then o':InstanceTypeOf(c)

2) If a: A and A<:B, then a:B

3) InstanceTypeOf(c')<:InstanceTypeOf(c) if and only if c' is subclass of c.

โดยเครื่องหมาย <: หมายถึงข้อมูลที่อยู่ฝั่งซ้ายของเครื่องหมายเป็นคลาสย่อย (Sub Class) หรือเซตย่อย (Sub Set) ของข้อมูลที่อยู่ฝั่งขวาของเครื่องหมาย

โดยนิยามข้อแรกกล่าวว่าหาก c' เป็นคลาสย่อยของ c แล้ว o' ซึ่งเป็น Instance ของ c' จะเป็น Instance ของ c ด้วย ข้อสองกล่าวว่าหาก a มีชนิด (Type) เป็น A และชนิด A เป็นคลาสย่อยของ B แล้ว a มีชนิดเป็น B ด้วย ในข้อสุดท้ายกล่าวไว้ว่า Instance ของ c' เป็นคลาสย่อยของ Instance ของ c ก็ต่อเมื่อ c' เป็นคลาสย่อยของ c เท่านั้น ด้วยนิยามนี้ทำให้เกิดความสามารถในการสืบทอดขึ้นดังตัวอย่าง

```
subclass reCell of cell is
        var backup: Integer :=0;
        override set(n: Integer) is
                if (n>0) then
                    self.backup :=self.contents;
                    super.set(n);
        end;
        method restore() is
                self.contents :=self.backup;
        end;
end;
```

รูปที่ 2.4 แสดงคลาส reCell ซึ่งสืบทอดมาจากคลาส Cell

จากรูปที่ 2.4 ตัวอย่างการสืบทอดคลาส cell ด้วยคลาส reCell (Restorable cell) โดยมีความคิดในเชิงนามธรรมถึงการเพิ่มพฤติกรรมให้กับคลาส cell ในคลาส reCell โดยพฤติกรรมดังกล่าวนั้นคือ การนำสถานะที่ผ่านมาของวัตถุกลับคืน การเพิ่มพฤติกรรมนี้ประกอบไปด้วยการเพิ่มเมท็อด restore และการเปลี่ยนแปลงพฤติกรรมของเมท็อดใหม่โดยการเพิ่มการเก็บค่าสถานะของวัตถุไว้ก่อนการกำหนดสถานะใหม่

Hueni, Johnson, and Engel (1995) นิยามคุณสมบัติของการสืบทอดในรูปของสมการพิชคณิตได้ดังต่อไปนี้

$$\frac{\forall \tau.GenSub[\tau] <: GenSuper[\tau]}{\forall t.t <: GenSub[t] \Rightarrow t <: GenSuper[t]} \quad (2.1)$$

จากสมการที่ (1) นั้นอธิบายถึง GenSub และ GenSuper ซึ่งเป็นคลาสซึ่งมีความสัมพันธ์ในเชิงสืบทอดกันกล่าวคือ GenSub เป็นคลาสลูกของ GenSuper หรือในทางนามธรรมนั้นกล่าวได้ว่า GenSub ได้รับการสืบทอดสถานะและพฤติกรรมมาจากคลาส GenSuper ดังนั้นและ Instance ซึ่งเกิดจากกระบวนการ Instantiation จากคลาส GenSub นั้นย่อมเป็นคลาสลูกของ GenSuper ด้วยเช่นกัน นอกจากนี้แล้วการพิจารณาชนิดข้อมูล หรือคลาสในรูปแบบของเซต (Set) นั้นยังช่วยอธิบายความหมายของการสืบทอดได้อีกด้วย ดังตัวอย่างเช่น GenSuper = {A, B} และ GenSub = GenSuper + {B, C} = {A, B, C}

จากตัวอย่าง GenSuper เป็นคลาสซึ่งประกอบไปด้วย A ซึ่งเป็นคุณสมบัติเชิงข้อมูล (Data attribute) และ B ซึ่งเป็นคุณสมบัติเชิงฟังก์ชัน (Function Attribute) การสืบทอด GenSuper โดย GenSub นั้นเห็นได้ชัดว่ายังคงไว้ซึ่งคุณสมบัติ A และเพิ่มคุณสมบัติ C ซึ่งตรงตามหลักการแห่งการสืบทอด และในที่นี้ยังคงพิจารณาได้อีกว่า B นั้นอาจถูกซ้อนทับโดยการซ้อนทับ (Override) ซึ่งเป็นการเปลี่ยนพฤติกรรมของคลาสแม่ในคลาสที่สืบทอดของคลาสลูก จากสมการต่อไปนี้แสดงการนิยามฟังก์ชันย่อย (Sub-Function)

$$\frac{D_X <: D_{Y,} C_y <: C_x}{f : D_y \to C_y <: f : D_x \to C_x} \quad (2.2)$$

จากสมการ (2) ซึ่งเป็นสมการเชิงพีชคณิตนั้น $f : D_y \to C_y$ หมายถึงฟังก์ชันซึ่ง มีโดเมนเป็น D_y และมีโดเมนร่วมเกี่ยว (Co-domain) กล่าวคือ C_y ในเชิงการเขียนโปรแกรมนั้นโดเมนหมายถึงอาร์กิวเมนต์ที่ส่งเข้าสู่ฟังก์ชัน และโดเมนร่วมเกี่ยวคือผลที่ได้จากฟังก์ชัน จากสมการ (2) นั้นกล่าวว่า $f : D_y \to C_y$ เป็นฟังก์ชันย่อยของ $f : D_X \to C_X$ ก็ต่อเมื่อ D_X เป็นแบบชนิดย่อย (Sub-Type) หรือสืบทอดจาก D_y และ C_y เป็นแบบชนิดย่อยหรือสืบทอดจาก C_X หลักแห่งการสืบทอดนั้นทำให้การพัฒนาซอฟต์แวร์เชิงวัตถุมีประสิทธิภาพมากยิ่งขึ้น ดังจะได้กล่าวเพื่อแสดงเห็นต่อไป

2.1.4 ภาวะพหุสัณฐาน (Polymorphism)

ภาวะพหุสัณฐานอาศัยความแนวคิดเชิงวัตถุพื้นฐานในเรื่อง Dynamic Binding และการสืบทอดเพื่อสนับสนุนให้เกิดกลไกของโปรแกรมทำงานอย่างเป็นพลวัตในขณะดำเนินการ (Run-Time)

```
var myCell: InstanceTypeOf(cell) := new cell;
var myReCell: InstanceTypeOf(reCell) := new reCell;
myCell :=myreCell;
```

รูปที่ 2.5 แสดงภาวะพหุสัณฐานในการอ้างถึงวัตถุที่ถูกสร้างขึ้นโดยการสืบทอด

จากรูปที่ 2.5 ตัวอ้างอิง myCell อ้างอิงวัตถุที่ถูกสร้างจากคลาส cell และตัวอ้างอิง myReCell อ้างอิงวัตถุที่ถูกสร้างจากคลาส reCell ในบรรทัดสุดท้ายเป็นการกำหนดให้ตัวอ้างอิง myCell อ้างอิงวัตถุที่ถูกสร้างจากคลาสแม่ (Super Class) จากขั้นตอนกระบวนการดังที่ได้กล่าวแล้วนั้นเป็นการผิดกฎในเชิงภาษาทั้งสิ้น หากใช้ในภาษาอย่าง Pascal แต่กลับถูกต้องในการใช้ในภาษาที่ถูกออกแบบให้ทำงานภายใต้กลไกเชิงวัตถุ

```
procedure g(x: InstnaceTypeOf(cell))is
        x.set(3);
end;
g(myCell)
g(myReCell);
```

รูปที่ 2.6 แสดงตัวอย่างภาวะพหุสัณฐานในขณะเกิด Dynamic Binding

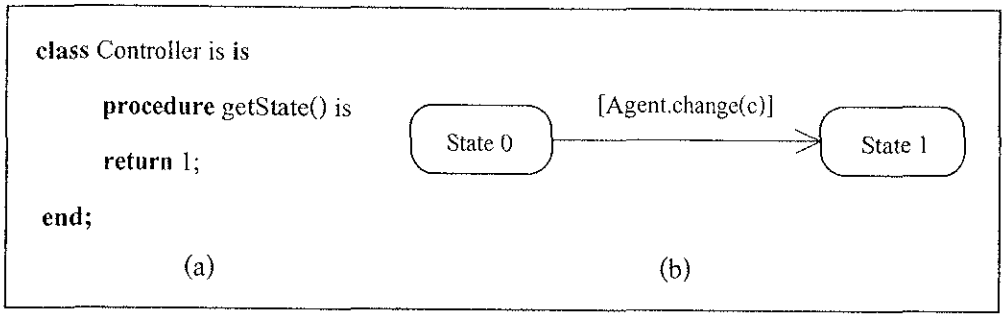
จากรูปที่ 2.6 นั้นเมท็อด g นั้นถูกกำหนดให้รับค่าที่มีชนิดเป็น cell โดยการทำงานในเมท็อดนั้นจะเรียกให้เมท็อด set ของตัวอ้างอิง x ทำงาน ในกลไกลของภาษาเชิงวัตถุนั้นยินยอมให้ตัวอ้างอิง x นั้นอ้างถึงวัตถุที่ถูกสร้างจากคลาส cell และคลาสที่สืบทอด

ในเชิงสถานะของวัตถุหรือโปรแกรมนั้น ภาวะพหุสัณฐานซึ่งเกิดจากกลไก ของ Dynamic Binding และการสืบทอดนั้นสามารถเปลี่ยนสถานะของวัตถุหรือโปรแกรมให้ทำงานแตกต่างออกไปได้โดยขึ้นอยู่กับวัตถุที่ถูกทำงานจริง (Concrete Object) ในขณะโปรแกรมทำงาน

```
class Agentis
        var status:Integer:=0;
        procedure change(x:InstacneTypeOf(Controller)) is
                status:=x.getState();
        end;
end;
```

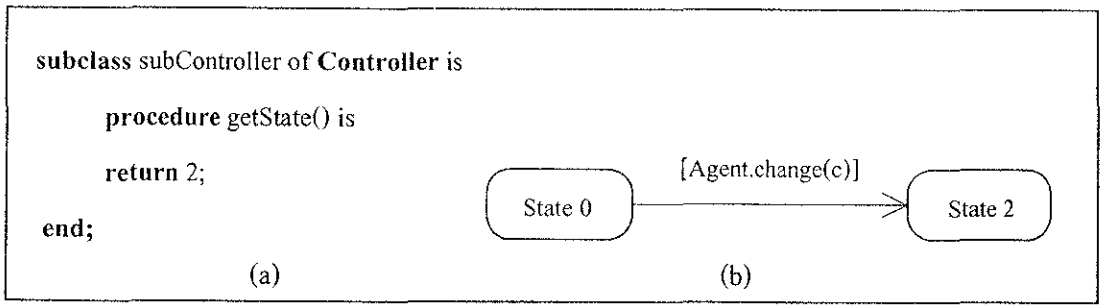
รูปที่ 2.7 แสดงสถานะของคลาส Agent

ดังแสดงในรูปที่ 2.7 นั้นคลาส Agent มีตัวแปรเชิงคุณสมบัติที่บ่งชี้ถึงสถานะโดยแรกเริ่ม ซึ่งมีสถานะเป็น 0 และมีความสามารถเชิงพฤติกรรมที่ถูกสะท้อนโดยเมท็อด change ซึ่งจะเปลี่ยนแปลงสถานะของวัตถุ (Instance) อันเกิดจากคลาส Agent โดยเมท็อด change นั้นรับอาร์กิวเมนต์ซึ่งเป็นวัตถุที่ถูกสร้างโดยคลาส Controller (รูปที่ 2.8 a) โดยมีหน้าที่ควบคุมสถานะของวัตถุ โดยนักวิเคราะห์และออกแบบสามารถออกแบบคลาส Controller อย่างง่ายเพื่อให้เข้าใจหลักการของภาวะพหุสัณฐานได้ดังต่อไปนี้



รูปที่ 2.8 (a) แสดงโครงสร้างของคลาส Controller
(b) แสดงสถานะของวัตถุที่ถูกสร้างจาก Agent

รูปที่ 2.8 b แสดงหลักการของภาวะพหุสัณฐานนั้นสามารถเปลี่ยนแปลงพฤติกรรมหรือสถานะของโปรแกรม หรือเพิ่มความสามารถของโปรแกรมได้โดยอาศัยหลักการของการสืบทอดสร้างคลาสซึ่งเป็นคลาสลูกของคลาส Controller และมีเมท็อด getState เมื่อปรับเปลี่ยนเพื่อส่งค่าสถานะกลับ



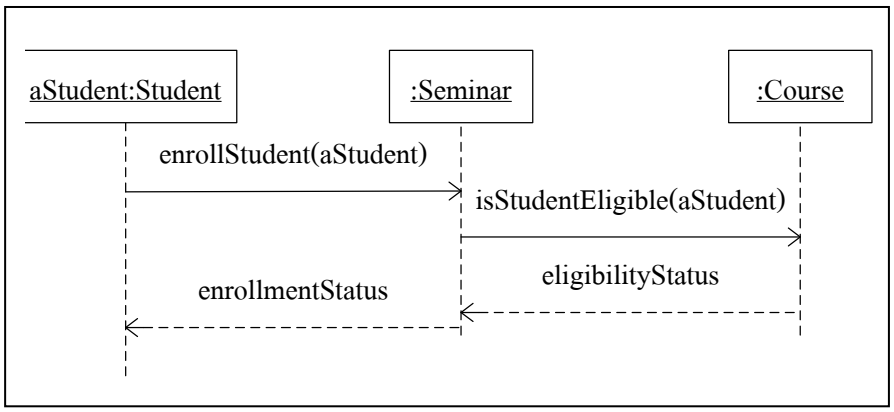
รูปที่ 2.9 (a) แสดงการสืบทอดคลาส Controller (b) แสดงสถานะของ Agent

รูปที่ 2.9 (a) และ 2.9 (b) แสดงคลาส subController ซึ่งสืบทอดมาจากคลาส Controller และมีเมท็อด getState() เป็นเมท็อดที่ซ้อนทับเพื่อปรับเปลี่ยนพฤติกรรมในคลาสแม่โดยอาศัยกลไก Dynamic binding และทำการส่งวัตถุที่ถูกสร้างจากคลาส subController ให้กับวัตถุของคลาส Agent สถานะของโปรแกรมหรือ Agent เปลี่ยนไป ดังที่แสดงจากตัวอย่างทั้งสองแล้วนั้น ภาวะพหุสัณฐานนั้นก่อให้เกิดความยืดหยุ่นและการเพิ่มความสามารถให้ความส่วนโปรแกรมในการกระบวนพัฒนาโปรแกรมประยุกต์

2.1.5 การสื่อสารด้วยข้อความ (Message communication)

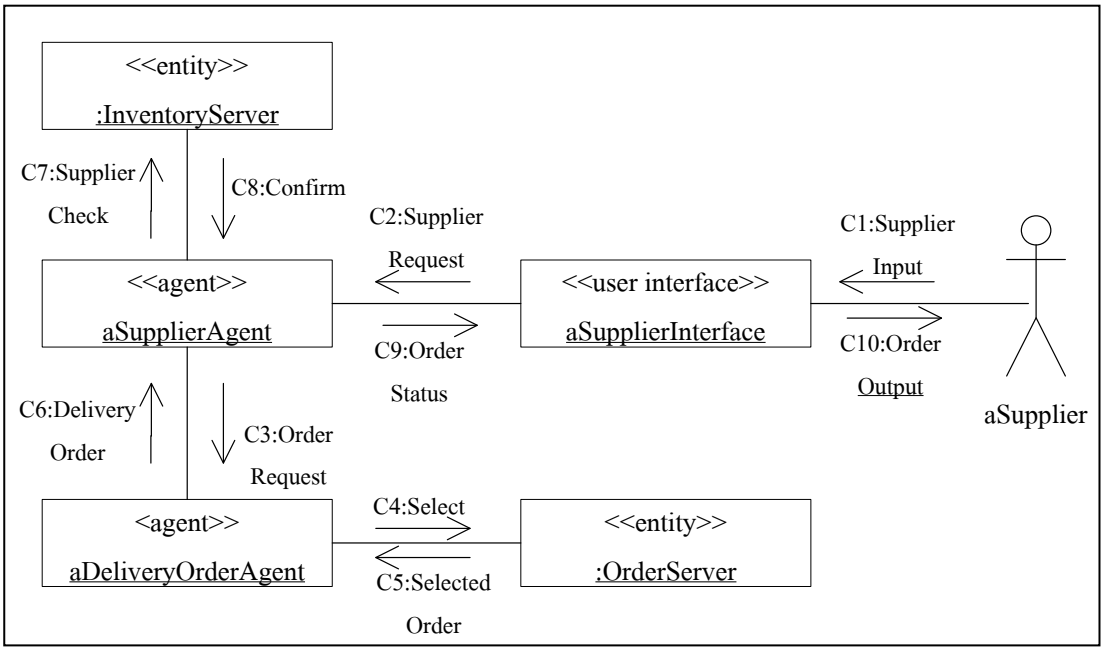
เป็นแนวคิดที่แสดงถึงการสื่อสารระหว่างวัตถุหนึ่งวัตถุใดกับวัตถุอื่น โดยการสื่อสารระหว่างวัตถุทั้งสองนั้นแสดงได้โดยการเรียกเมท็อด (Method call) ของเป้าหมายให้ทำงานซึ่งการส่งค่าตัวแปรเสริม (Parameter) ในการเรียกด้วยก็ได้

Gomaa (2004) แสดงแนวคิดเกี่ยวกับการสื่อสารกันนี้ได้เป็นส่วนหนึ่งในออกแบบในแผนภาพเชิงพฤติกรรม (Behavior Diagram) บนมาตรฐาน UML (Unified Modeling Language) ยกตัวอย่างเช่น Communication Diagram และ Sequence Diagram



รูปที่ 2.10 แสดง Sequence Diagram

จากรูปที่ 2.10 แสดงตัวอย่างของ Sequence Diagram ซึ่งใช้แนวคิดในการส่งสาร (Messages) ระหว่างวัตถุที่อยู่ในระบบ โดย Sequence Diagram นั้นแสดงถึงการติดต่อสื่อสารกันระหว่าวัตถุอย่างเป็นลำดับขั้น ดังแสดงในตัวอย่างนั้นตัวอ้างอิง aStudent นั้นเป็นวัตถุที่ถูกสร้างจากคลาส Student เรียกเมท็อดชื่อ enrollStudent ที่อยู่ในวัตถุซึ่งถูกสร้างจากคลาส Seminar ซึ่งเมท็อด enrollStudent นั้นแสดงถึงพฤติกรรมของ Seminar ซึ่งมีความสามารถในการจัดการการลงทะเบียน นอกจากนี้ยังมีการส่งสารในการตอบสนองคำร้องขอ ดังแสดงในรูปนั้นวัตถุที่เกิดจากคลาส Seminar ส่งสถานะของการลงทะเบียนให้กับวัตถุ aStudent ซึ่งตามมาตรฐาน UML นั้นแสดงโดยใช้เส้นประ



รูปที่ 2.11 แสดง Communication Diagram

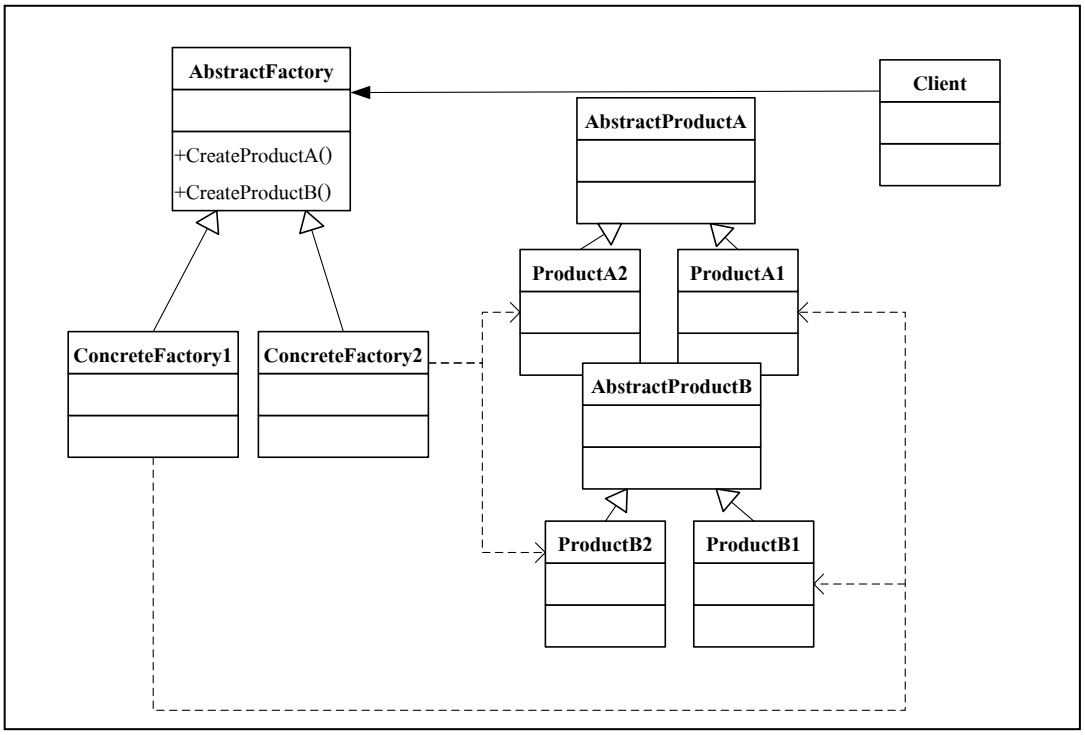
จากรูปที่ 2.11 แสดง Communication Diagram ซึ่งเป็นแผนภาพที่ใช้ในการอธิบายการติดต่อสื่อสารระหว่างวัตถุที่อยู่ในบริบทของโปรแกรมประยุกต์ (Application Context) เท่านั้นไม่มุ่งเน้นลำดับการทำงานของโปรแกรมซึ่งแต่ต่างจาก Sequence Diagram ซึ่งแสดงขั้นตอนการสื่อสารกันระหว่างวัตถุมากกว่า

2.1.6 ความสัมพันธ์ (Associations)

เป็นแนวคิดเกี่ยวกับความสัมพันธ์กันระหว่างวัตถุที่ถูกออกแบบเพื่อเป็นคำตอบ ของความต้องการเชิงซอฟต์แวร์

2.1.7 การนำกลับมาใช้ใหม่ (Reusability)

แนวคิดเกี่ยวกับการนำมาใช้ใหม่นั้นถูกกล่าวถึงมากในกระบวนการวิเคราะห์และออกแบบเนื่องจากช่วยบรรเทาขีดจำกัดด้านเวลาและค่าใช้จ่ายมากยิ่งขึ้น แนวคิดเชิงวัตถุเกี่ยวกับการห่อหุ้ม ภาวะพหุสัณฐาน และแนวคิดในเชิงนามธรรม (Abstraction) นั้นสนับสนุนแนวคิดในการนำกลับมาใช้ใหม่ (Reusability) ทั้งในรูปแบบของแนวคิดตัวอย่างเช่นรูปแบบเชิงซอฟต์แวร์และรูปแบบของส่วนโปรแกรมที่พัฒนาขึ้นจริง (Implementation) ในรูปแบบของคลังส่วนโปรแกรม (Code Libraries) หรือกรอบงานตัวอย่างเช่น Hibernate3, Spring Framework, EJB3, JPO, JPA, JPOX, XUI (Java and XML Rich Application Framework, Xoptrope, www, 2008a)



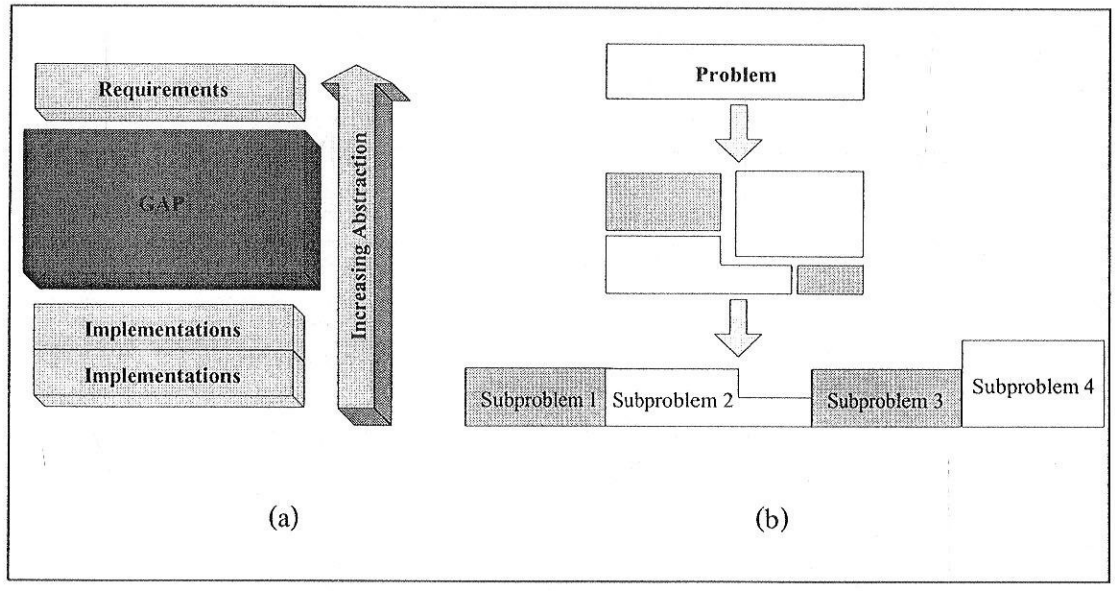
รูปที่ 2.12 แสดง Class Diagram ของ Abstract Factory Pattern

รูปที่ 2.12 เป็นการแสดงการออกแบบ Abstract Factory Pattern โดย Gamma, Helm, Johnson, and Vlissides (1995) ซึ่งเป็นกาสนับสนุนแนวคิดในการนำกลับมาใช้ใหม่ โดยการสร้างรูปแบบเชิงซอฟต์แวร์ให้มีความยืดหยุ่นในรูปแบบทั่วไปซึ่งจะถูกนำมาใช้ในการแก้ปัญหาที่มีขอบเขตเฉพาะเจาะจงต่อไป การออกแบบรูปแบบเชิงซอฟต์แวร์นั้นเป็นรูปแบบแนวคิดที่ถูกออกแบบโดยไม่ขึ้นตรงกับภาษาที่ใช้ในการพัฒนา ภาษาเชิงวัตถุอย่าง Java หรือ C#

2.2 วิศวกรรมซอฟต์แวร์เชิงวัตถุ (Object-Oriented Software Engineering)

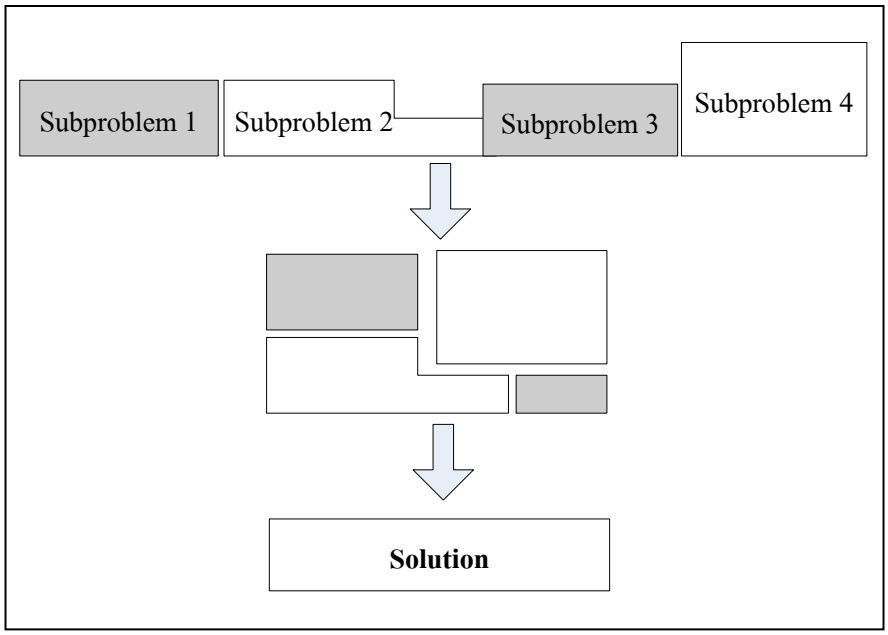
วิศวกรรมซอฟต์แวร์นั้นเป็นกระบวนที่ถูกคิดค้นเพื่อพัฒนาซอฟต์แวร์อย่างเป็นระบบ ใช้ความรู้ เครื่องมือ วิธีการและกระบวนการในด้านการเก็บความต้องการ การวิเคราะห์ ออกแบบ การสร้างหรือพัฒนา การทดสอบ และการบำรุงรักษาเข้าประยุกต์ใช้กับกระบวนการพัฒนาซอฟต์แวร์ วิศวกรรมซอฟต์แวร์มีความสัมพันธ์กับศาสตร์ที่เกี่ยวข้องอันได้แก่วิทยาการคอมพิวเตอร์ วิศวกรรมคอมพิวเตอร์ การจัดการ คณิตศาสตร์ การควบคุมจัดการคุณภาพและวิศวกรรมระบบ วิศวกรรมซอฟต์แวร์เชิงวัตถุเป็นการนำกระบวนการโดยทั่วไปของวิศวกรรมซอฟต์แวร์ประยุกต์ใช้เข้ากับหลักการเชิงวัตถุเพื่อจุดประสงค์ในการพัฒนาซอฟต์แวร์

ในกระบวนการวิเคราะห์และออกแบบซอฟต์แวร์นั้น เริ่มต้นด้วยปัญหาหรือความต้องการเชิงซอฟต์แวร์ซึ่งมีความเป็นนามธรรมสูงดังรูปที่ 2 และก่อให้เกิดช่องว่างระหว่างความเป็นนามธรรมนั้นกับรายละเอียดการปฏิบัติ (Refinement) ในการพัฒนาซอฟต์แวร์ดังแสดงในรูป (a)



รูปที่ 2.13 (a) แสดงระดับความเป็นนามธรรม (b) แสดงแนวทางการวิเคราะห์ปัญหา

รูปที่ 2.13 (b) แสดงการวิเคราะห์ปัญหา โดยปัญหาที่เกิดจากการเก็บความต้องการเชิงซอฟต์แวร์นั้นถูกวิเคราะห์และแบ่งออกเป็นส่วนย่อยซึ่งสามารถเข้าใจและจัดการได้โดยง่าย โดยในแต่ละปัญหาย่อยเหล่านี้ต้องถูกรำลึกถึงความเกี่ยวพันกันเสมอ



รูปที่ 2.14 แสดงกระบวนการสร้างซอฟต์แวร์จากส่วนโปรแกรมย่อย ๆ

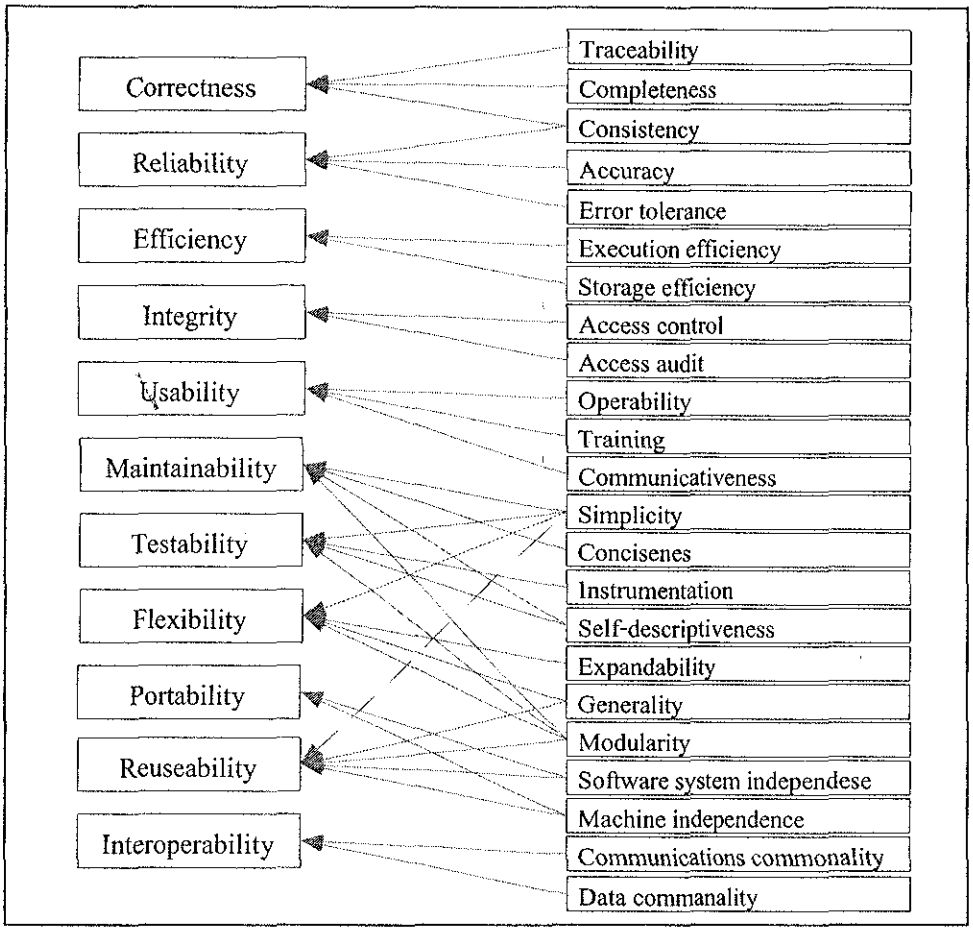
จากรูปที่ 2.14 แสดงถึงกระบวนการสังเคราะห์ (Synthesis) โดยเป็นการใช้กระบวนการทางวิศวกรรมซอฟต์แวร์พัฒนาซอฟต์แวร์จากส่วนโปรแกรมย่อย ส่วนโปรแกรมเหล่านี้เกิดจากการจำแนกปัญหาหรือความต้องการในเชิงซอฟต์แวร์ออกเป็นส่วน ๆ ดังกล่าวไว้ข้างต้น

กล่าวโดยสรุปคือ วิศวกรรมซอฟต์แวร์เชิงวัตถุคือการใช้ความรู้ด้านวิศวกรรมในกระบวนการวิเคราะห์ปัญหาหรือความต้องการเชิงซอฟต์แวร์ออกเป็นส่วนย่อย และสังเคราะห์ส่วนย่อยออกเป็นส่วนโปรแกรมซึ่งอาจถูกสร้างด้วยความรู้ กลวิธี กระบวนการแตกต่างกัน และถูกนำมารวมกันเพื่อให้ตรงตามความต้องการเชิงซอฟต์แวร์

2.2.1 คุณภาพของผลผลิตที่เกิดจากกระบวนการพัฒนาซอฟต์แวร์

หากกล่าวถึงความหมายหรือนิยามของคุณภาพในเชิงซอฟต์แวร์ที่ดีนั้นคำตอบที่ได้รับมักจะมีความแตกต่างกันโดยขึ้นอยู่กับใครเป็นผู้ตอบคำถาม ลูกค้าหรือผู้ใช้มักตอบในมุมมองคุณสมบัติภายนอกของซอฟต์แวร์ ยกตัวอย่างเช่น ความใช้ง่าย (Usability) ก็ดีหรือความน่าเชื่อถือ (Reliability) ในการใช้งานก็ดี หากแต่ผู้ที่มีหน้าที่สร้างซอฟต์แวร์โดยตรงนั้นกลับมีความเห็นแตกต่างกันโดยมองคุณสมบัติภายในของซอฟต์แวร์นั้น ๆ เป็นหลัก ตัวอย่างเช่น ความยืดหยุ่น (Flexibility) ความสามารถในการนำกลับมาใช้ใหม่ (Reusability) ความสามารถในการบำรุงรักษา (Maintainability) และอื่น ๆ เป็นต้น

แบบจำลองความสัมพันธ์ระหว่างปัจจัยและเกณฑ์ถูกสร้างขึ้นเพื่อใช้ในการวัดคุณภาพของซอฟต์แวร์ดังรูปที่ 2.15



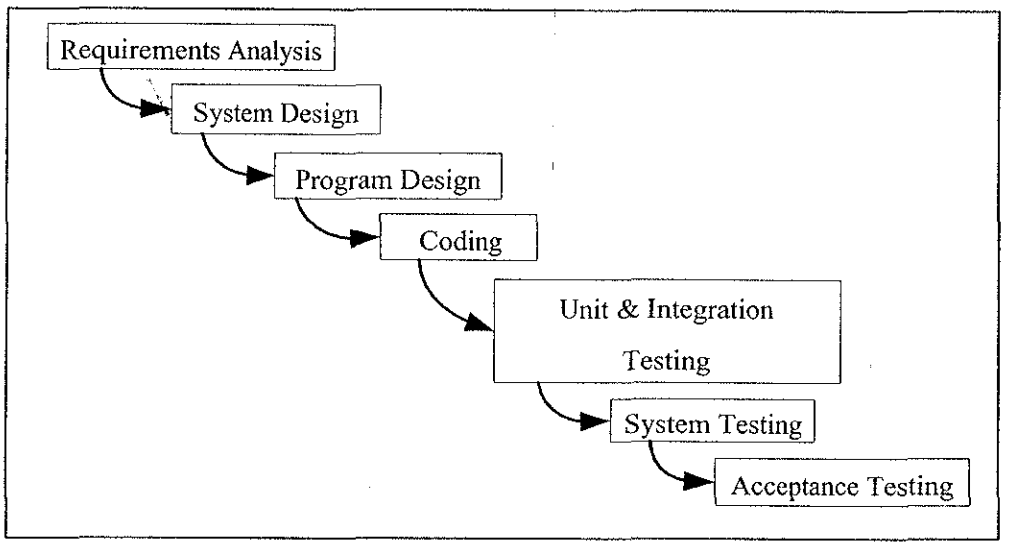
รูปที่ 2.15 คุณภาพเชิงซอฟต์แวร์

งานวิจัยชิ้นนี้มุ่งเน้นคุณภาพในกระบวนการพัฒนาซอฟต์แวร์ในส่วนของความยืดหยุ่นและความสามารถในการกลับมาใช้ใหม่ โดยจากแบบจำลองนั้นเกณฑ์ที่ใช้วัดความยืดหยุ่นประกอบไปด้วย การออกแบบซอฟต์แวร์โดยง่ายไม่ซับซ้อน (Simplicity) การออกแบบโดยหลักทั่วไป (Generality) ความเป็นโมดูลาร์หรือสภาพเป็นส่วนจำเพาะ (Modularity) ในส่วนของการนำกลับมาใช้ใหม่นั้นมีเกณฑ์บ่งชี้อันได้แก่ การออกแบบโดยง่าย สภาพโมดูลาร์ ความไม่ขึ้นกับซอฟต์แวร์ระบบที่ใช้ และความไม่ขึ้นกับเครื่องที่ใช้ดำเนินการ

2.2.2 แบบจำลองกระบวนการพัฒนาซอฟต์แวร์ (Software development process model)

ในกระบวนการพัฒนาซอฟต์แวร์นั้นมีหลากหลายแบบจำลองซึ่งถูกคิดขึ้นเพื่อวางแนวทางในการพัฒนาซอฟต์แวร์เพื่อให้ได้มาซึ่งคุณภาพสูงสุดต่อผลิตภัณฑ์ แม้ว่าแต่ละแบบจำลองนั้นมีรายละเอียดที่กล่าวถึงกระบวนการพัฒนาที่แตกต่างกันแต่ทุก ๆ แบบจำลองนั้นยังคงประกอบไปด้วยความต้องการในเชิงซอฟต์แวร์ซึ่งเปรียบเสมือนข้อมูลเข้า (Input) สู่กระบวนการและมีซอฟต์แวร์เป็นผลิตภัณฑ์ออก (Output) จากกระบวนการพัฒนา

1) แบบจำลองน้ำตก (Waterfall Model)

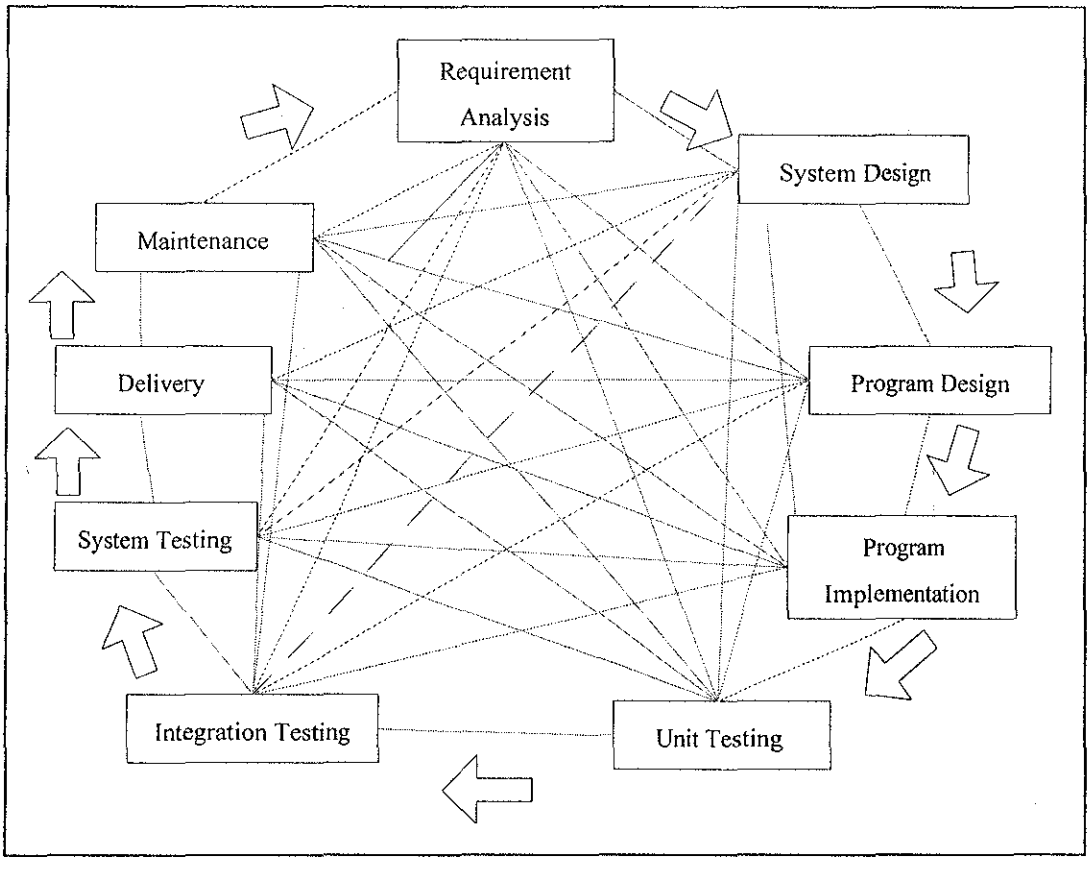


รูปที่ 2.16 แบบจำลอง Waterfall

จากรูปที่ 2.16 แสดงแบบจำลองน้ำตกซึ่งอธิบายถึงขั้นตอนกระบวนการพัฒนาซอฟต์แวร์นั้นประกอบไปด้วย

1) การเก็บความต้องเชิงซอฟต์แวร์และการวิเคราะห์ระบบ
2) การออกแบบระบบ
3) การออกแบบโปรแกรม
4) การเขียนโปรแกรม
5) การทดสอบในระดับส่วนย่อยและการทดสอบเพื่อมีการรวมส่วนย่อย
6) การทดสอบระบบ
7) การทดสอบขั้นการอนุมัติซอฟต์แวร์
8) การดูแลบำรุงรักษาซอฟต์แวร์

แบบจำลองนี้ตั้งอยู่บนสมมติฐานที่ความต้องการเชิงซอฟต์แวร์ของระบบนั้นถูกต้องสมบูรณ์และมั่งคงไม่มีการเปลี่ยนแปลง ดังนั้นขั้นกระบวนการในแต่ละส่วนนั้นสามารถดำเนินการให้เสร็จสมบูรณ์ในแต่ละขึ้นก่อนที่จะเริ่มการดำเนินการในขั้นถัดไป ยกตัวอย่างเช่น การเขียนโปรแกรมนั้นจะเริ่มได้ต่อเมื่อมีการออกแบบโปรแกรมทั้งหมดเรียบร้อยแล้ว และเมื่อการเขียนโปรแกรมเสร็จสมบูรณ์จึงเริ่มขั้นตอนกระบวนการทดสอบโปรแกรม หากแต่ Pfleeger (1998) กล่าวถึงในกระบวนการพัฒนาซอฟต์แวร์ที่เกิดขึ้นจริงนั้นมิได้เป็นอย่างทฤษฎีในอุดมคติกล่าวว่าความต้องการเชิงซอฟต์แวร์นั้นมักไม่สมบูรณ์ ไม่ชัดเจน กำกวม ซ้ำซ้อนและมีการเปลี่ยนแปลงในภายหลังเสมอ อันมีสาเหตุจากการเก็บความต้องการไม่สมบูรณ์แต่เริ่มแรกของกระบวนการการพัฒนาซอฟต์แวร์ หรือแม้กระทั่งความไม่เข้าใจในความต้องการของผู้ให้ความต้องการเชิงซอฟต์แวร์นั้น ๆ เอง ดังนั้นแล้วการจัดการพัฒนาซอฟต์แวร์ตามแบบจำลองน้ำตกนั้นจึงไม่สามารถทำให้ซอฟต์แวร์ที่ได้นั้นมีความสมบูรณ์ตรงตามความต้องการภายใต้ค่าใช้จ่ายและเวลาที่กำหนดได้

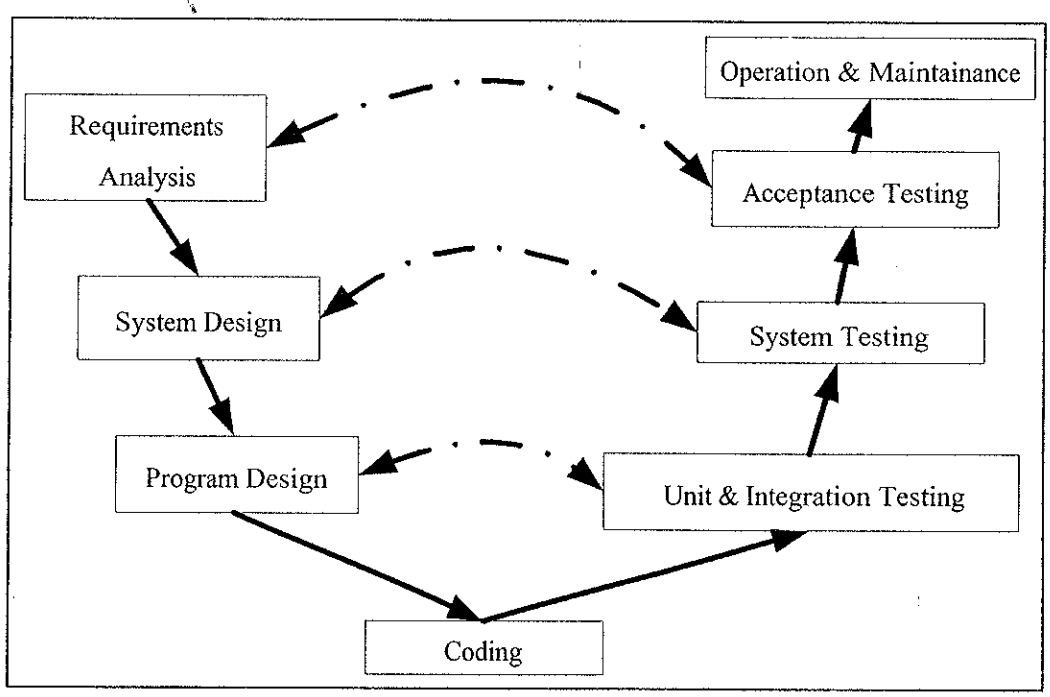


รูปที่ 2.17 แสดงกระบวนการพัฒนาซอฟต์แวร์ที่เกิดจริง

จากรูปที่ 2.17 แสดงถึงกระบวนการพัฒนาซอฟต์แวร์ที่เกิดขึ้นจริง ซึ่งขัดแย้งกับแบบจำลองน้ำตกเนื่องจากมีการย้อนกลับดำเนินการในขั้นตอนที่ผ่านมา ซึ่งเกิดเนื่องจากความไม่สมบูรณ์ของการทำงานในขั้นตอนที่ผ่านมาหรืออาจเกิดขึ้นจากความเปลี่ยนแปลงไปของความต้องการ เชิงซอฟต์แวร์ดังได้กล่าวไว้ข้างต้น เหตุการณ์ดังภาพเป็นภาพจำลองเมื่อกระบวนการในการพัฒนาซอฟต์แวร์ไม่สามารถควบคุมให้เป็นไปตามกระบวนการวางแผนไว้ได้

2) แบบจำลองรูปตัว V (V-model)

เนื่องจากการดำเนินการตามแบบจำลองน้ำตกนั้นไม่สามารถทำให้กระบวนพัฒนาซอฟต์แวร์มีประสิทธิภาพสูงสุด แบบจำลองรูปตัว V อยู่บนพื้นฐานเดิมของแบบจำลองน้ำตกและได้กล่าวถึงขั้นตอนทดสอบความถูกต้องดังแสดงในรูปที่ 2.18



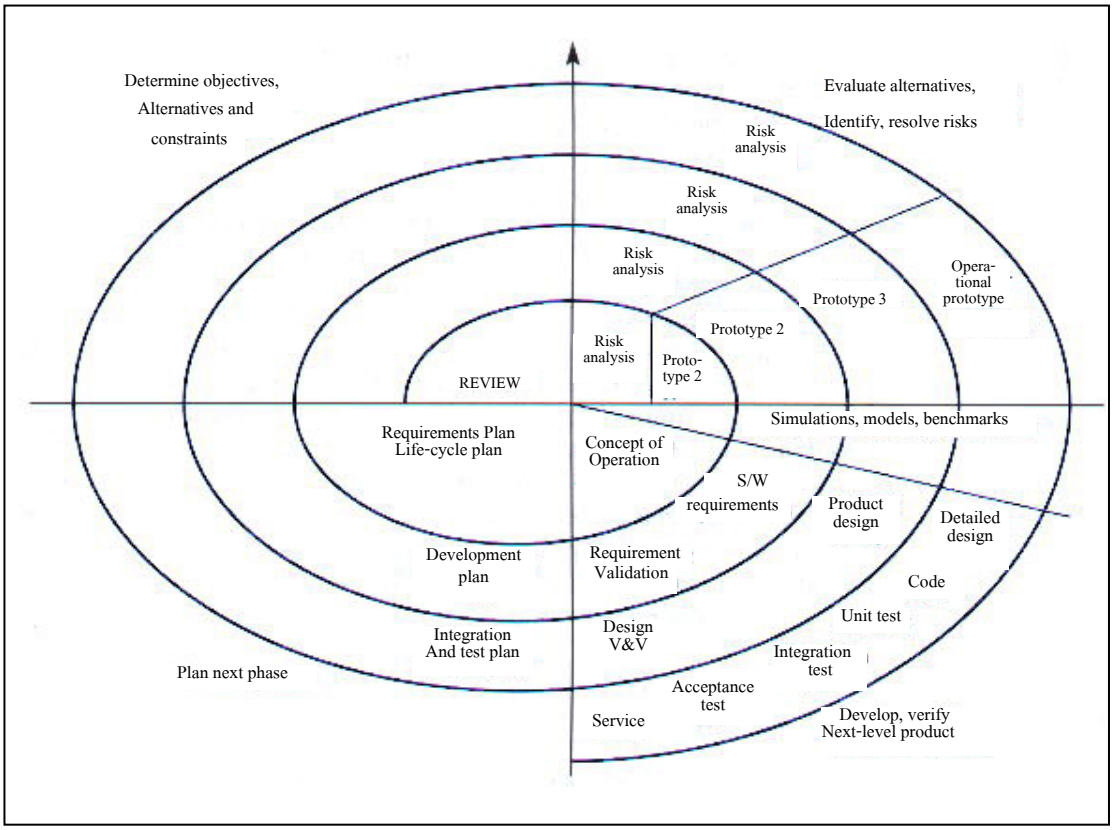
รูปที่ 2.18 แสดงแบบจำลองรูปตัว V

จากแบบจำลองที่แสดงดังรูปที่ 2.18 นั้นอธิบายการทดสอบความถูกต้องโดยอธิบายถึงกระบวนการทดสอบแบบหน่วยย่อยและการทดสอบการรวมกันของหน่วยย่อยนั้นใช้ในการทดสอบและยืนยันความถูกต้องของการออกแบบโปรแกรม การทดสอบระบบใช้ในการยืนยันความถูกต้องในส่วนของการออกแบบระบบ และท้ายที่สุดการทดสอบเพื่อยอมรับในขั้นสุดท้ายเป็นการยืนยันความถูกต้องของการวิเคราะห์ความต้องการเชิงซอฟต์แวร์

แบบจำลองรูปตัว V นี้แบ่งส่วนของขั้นตอนการพัฒนาอยู่ฝั่งซ้ายและการทดสอบยืนยันความถูกต้องของกระบวนการพัฒนาอยู่ในฝั่งขวา และมีเส้นประเชื่อมความสัมพันธ์เพื่อกล่าวเป็นนัยว่า หากพบปัญหาความผิดพลาดเพื่อใช้กระบวนการทดสอบในฝั่งขวาของแบบจำลองเมื่อใดจำเป็นต้องย้อมกลับไปจัดการปัญหาที่เกิดขึ้นในกระบวนการพัฒนาที่อยู่ในฝั่งซ้ายใหม่ แบบจำลองนี้แสดงถึงการทดสอบที่สอดคล้องกับกระบวนการต่าง ๆ ในการพัฒนาซอฟต์แวร์

3) แบบจำลองก้นหอย (Spiral Model)

Pfleeger (1998) ผู้ซึ่งมีความเห็นถึงกระบวนการพัฒนาซอฟต์แวร์ที่มีความเสี่ยงเข้ามาเกี่ยวข้อง ดังนั้นการนำเสนอแบบจำลองก้นหอยซึ่งเป็นการรวมแนวคิดของกระบวนการพัฒนาซอฟต์แวร์เข้ากับการจัดการความเสี่ยงที่อาจเกิดขึ้นในระหว่างกระบวนการพัฒนาโดยการควบคุม และจัดการให้ความเสี่ยงเหล่านั้นเกิดขึ้นน้อยที่สุด



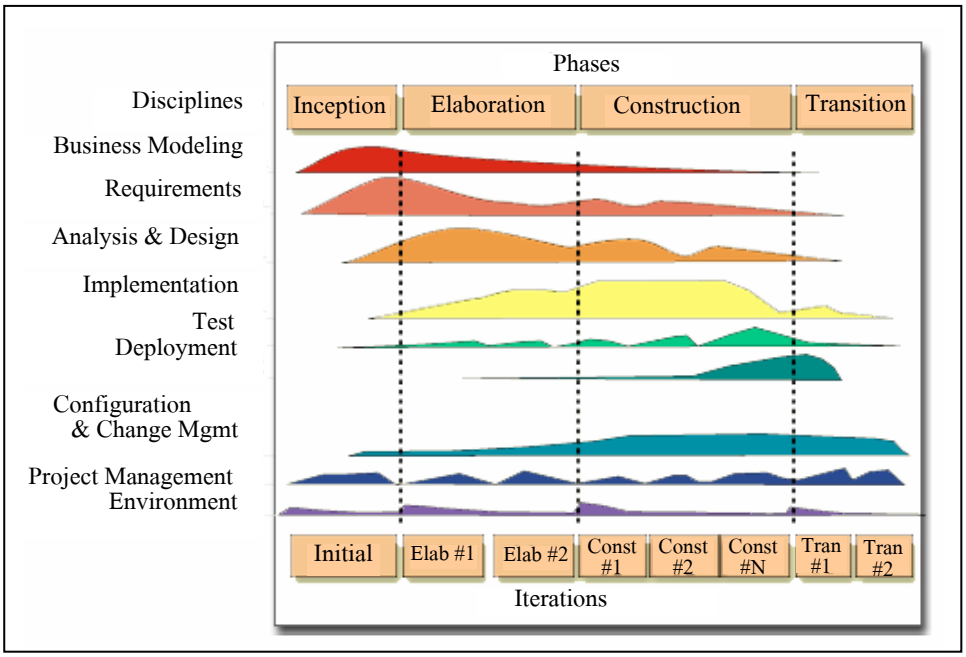
รูปที่ 2.19 แสดงแบบจำลองก้นหอย

ในกระบวนการพัฒนาซอฟต์แวร์โดยใช้แบบจำลองก้นหอยดังแสดงในรูป 2.19 นั้นเริ่มต้นด้วยการเก็บความต้องการเชิงซอฟต์แวร์และการวางแผนในขั้นต้นสำหรับพัฒนา

กระบวนการพัฒนาซอฟต์แวร์นี้เป็นกระบวนการพัฒนาแบบวนรอบ (Iteration) โดยในแต่ละรอบนั้นการออกแบบต่าง ๆ การพัฒนาซอฟต์แวร์ (Implementation) หรือแม้กระทั่งกระบวนการทดสอบจะถูกวิเคราะห์ประเมินความเสี่ยงและแก้ไขซ้ำแล้วซ้ำเล่าโดยหากพบความเสี่ยงที่ชัดเจนในระหว่างที่ดำเนินตามขั้นตอนการพัฒนาแบบก้นหอยนี้ผู้ควบคุมโครงการ (Project Manager) นั้นเป็นผู้ตัดสินต่อการจัดการความเสี่ยงที่เกิดขึ้นนั้น ๆ

4) แบบจำลอง Rational Unified Process (RUP)

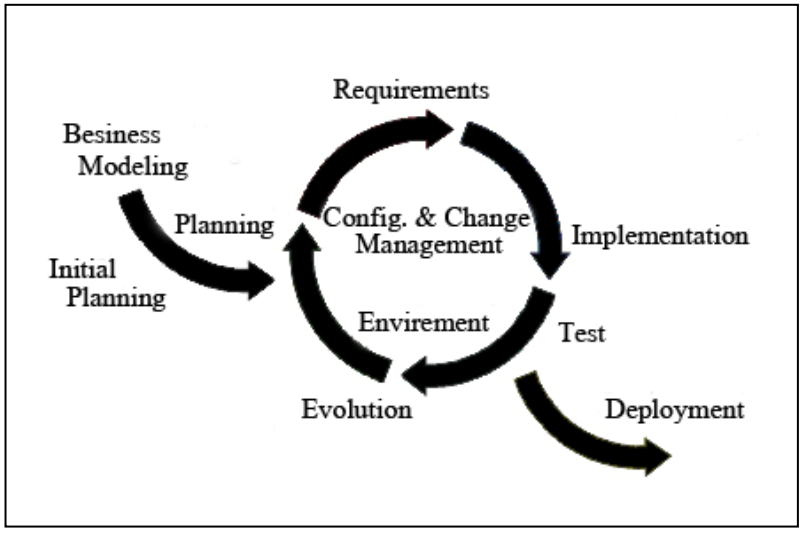
Rational Unified Process เป็นแบบจำลองขั้นตอนพัฒนาซอฟต์แวร์ของบริษัท Rational Software ซึ่งอยู่ในเครือ IBM แบบจำลองนี้ได้รับแนวคิดและการปรับปรุงมาจากแบบจำลองก้นหอยโดยการมีรูปแบบการพัฒนาซอฟต์แวร์แบบวนรอบเช่นกันแต่แตกต่างกันในรายละเอียดการพัฒนาดังรูปที่ 2.20



รูปที่ 2.20 แสดงแบบจำลอง Rational Unified Process

จากรูปที่ 2.20 แสดงให้เห็นโดยชัดแจ้งถึงขั้นตอนกระบวนการพัฒนาซอฟต์แวร์โดยใช้ RUP โดยการกระบวนการพัฒนานั้นแบ่งส่วนได้แก่การจำลองความต้องการเชิงซอฟต์แวร์ในระดับสูง (Business Modeling) การเก็บความต้องการเชิงซอฟต์แวร์ การวิเคราะห์และออกแบบการลงรายละเอียดในการเขียนโปรแกรม การทดสอบและส่งมอบซอฟต์แวร์ กระบวนการจัดการการเปลี่ยนแปลงเชิงซอฟต์แวร์ การจัดการโครงการและการจัดการสภาพแวดล้อมของซอฟต์แวร์

นอกจาก RUP นำเสนอการพัฒนาออกเป็นส่วนแล้วยังนำเสนอการวนรอบในการพัฒนาซอฟต์แวร์ออกเป็นขั้น (Phase) ประกอบไปด้วย Inception, Elaboration, Construction และ Transition โดยในแต่ขั้นนั้นจะมีดำเนินในทุก ๆ ส่วนการพัฒนาแต่จะมีระดับที่แตกต่างกันดังแสดงในกราฟ ยกตัวอย่างเช่น ในขั้นของ Inception จะต้องมีการจำลองความต้องการในระดับสูง มีการเก็บความต้องการเชิงซอฟต์แวร์ การวิเคราะห์และออกและอื่น ๆ ดังที่ได้กล่าวไว้ข้างต้น แต่ระดับกิจกรรมที่ดำเนินการในส่วนของการวิเคราะห์และออกแบบจะต่ำดังแสดงในกราฟ อีกตัวอย่างขอกล่าวถึงในขั้นของ Construction นั้นมุ่งเน้นในส่วนของการลงรายละเอียดในการเขียนรหัสคำสั่งดังนั้นจากรูปจึงเห็นได้ว่ากราฟจากรูป (แถบ Implementation) นั้นมีระดับสูงส่วนในส่วนของวิเคราะห์แลออกแบบนั้นเป็นลำดับรองลงมา เนื่องจากขั้นตอนนี้ยังคงต้องอาศัยการวิเคราะห์และออกแบบอยู่ อีกประเด็นที่ชี้ให้เห็นได้จากรูปที่ 2.20 และรูปที่ 2.21 รูปนั้นคือมีการวนรอบของกิจกรรมแต่ละส่วนซึ่งเห็นได้ว่าในส่วนของ Construction แบ่งเป็น 3 ส่วน อันได้แก่ Const#1, Const#2 และ Const#3



รูปที่ 2.21 แสดงการวนรอบของ RUP

นอกจากนี้แล้ว RUP ยังเป็นกระบวนการพัฒนาซอฟต์แวร์เชิงวัตถุโดยมีกิจกรรมที่สนับสนุนการเก็บความต้องการ วิเคราะห์ออกแบบและเขียนรหัสคำสั่งในเชิงวัตถุ ที่เห็นได้ชัดเจนคือ RUP นั้นนำเสนอ Unified Modeling Language เป็นแนวทางในการวิเคราะห์และออกแบบ

2.3 กรอบงานเชิงวัตถุ (Object-Oriented Application Framework)

กรอบงานเชิงวัตถุคือเทคโนโลยีที่ถูกสร้างขึ้นเพื่อทำให้การลดค่าใช้จ่ายและการเพิ่มประสิทธิภาพในกระบวนการพัฒนาซอฟต์แวร์เกิดขึ้นเป็นรูปธรรม กรอบงานนั้นคือโปรแกรมประยุกต์ที่ไม่สมบูรณ์ (Semi-Application) ถูกสร้างด้วยส่วนต่าง ๆ อันประกอบไปด้วยรูปแบบเชิงซอฟต์แวร์ คลังรหัสคำสั่งเชิงคลาส (Class Libraries) และส่วนประกอบเชิงซอฟต์แวร์ (Software Component) เพื่อสนับสนุนการนำกลับมาใช้ใหม่ในการพัฒนาโปรแกรมที่เฉพาะเจาะจงสำหรับลูกค้า โดยประโยชน์หลักที่ได้จากการใช้กรอบงานเชิงวัตถุ ประกอบไปด้วย สภาพเป็นส่วนจำเพาะ (Modularity) ความสามารถในการนำกลับมาใช้ใหม่ (Reusability) ความสามารถในด้านการเพิ่มความสามารถ (Extensibility) และ Inversion of Control

กรอบงานเชิงวัตถุเพิ่มสภาพเป็นส่วนจำเพาะโดยการห่อหุ้ม (Encapsulation) รายละเอียดของส่วนโปรแกรมให้อยู่เบื้องหลังส่วนต่อประสานที่ชัดเจน สภาพเป็นส่วนจำเพาะของกรอบงานนั้นสามารถช่วยเพิ่มประสิทธิภาพของซอฟต์แวร์โดยการจำกัดผลกระทบอันเกิดเนื่องมาจากการเปลี่ยนแปลงของการออกแบบและการพัฒนา ซึ่งการเปลี่ยนแปลงที่เกิดขึ้นเหล่านี้ทำให้เกิดความยากลำบาก ในการดูแลซอฟต์แวร์ที่ได้รับการพัฒนาเสร็จสิ้นแล้ว

ส่วนต่อประสานที่ชัดเจนซึ่งได้รับจัดเตรียมไว้โดยกรอบงานเชิงวัตถุนั้นเพิ่มความสามารถในการนำกลับมาใช้ใหม่โดยการกำหนดส่วนโปรแกรมที่รองรับการสร้างโปรแกรมประยุกต์ขึ้นใหม่ ซึ่งความสามารถในการนำกลับมาใช้ใหม่นี้ ทำให้ประสบการณ์หรือความรู้ของนักพัฒนาโปรแกรมประยุกต์นั้นไม่สูญเสียไปโดยไร้ประโยชน์ ทำให้หลีกเลี่ยงการสร้างหรือแก้ไขปัญหาเพื่อตอบสนองต่อความต้องการเชิงซอฟต์แวร์ (Software Requirement) จากเดิมที่เคยได้รับการแก้ไขแล้ว เช่นเดียวกันความสามารถในการนำกลับมาใช้ใหม่นั้น ทำให้นักพัฒนาโปรแกรมพัฒนาโปรกรมได้อย่างมีประสิทธิภาพ เพิ่มอัตราการผลิต อีกทั้งยังก่อให้เกิดความน่าเชื่อถือต่อซอฟต์แวร์ที่ได้รับการพัฒนาอีกด้วย กรอบงานเพิ่มความสามารถในด้านการเพิ่มความสามารถ (Extensibility) มากขึ้นโดยอาศัยหลักการของภาวะพหุสัณฐานการสืบทอดและการนิยาม ฮุคซึ่งรองรับการขยายหรือเชื่อมต่อผ่านส่วนต่อประสานที่ถูกกำหนดไว้อย่างชัดเจนจึงทำให้กรอบงานยังคงมีความเสถียรและทำงานได้อย่างมีประสิทธิภาพภายใต้กาลเวลาและความต้องการที่เปลี่ยนแปลงไป

กรอบงานเชิงวัตถุนั้นถูกออกแบบและกำหนดคุณสมบัติต่าง ๆ ด้วย “Inversion of Control” ซึ่งเป็นสถาปัตยกรรมที่มีกลไกในการสนับสนุนให้โปรแกรมเฉพาะที่ถูกพัฒนาภายใต้กรอบงาน ถูกกำหนดค่าได้อย่างพลวัตในขณะดำเนินงาน (Run-time) ด้วยเหตุนี้จึงทำให้กรอบงานเชิงวัตถุที่ถูกสร้างด้วยกลไก Inversion of Control มีความยืดหยุ่น และมีความสามารถในด้านการเพิ่มความสามารถ (Extensibility) สูง

2.3.1 ภาพรวมการใช้งานกรอบงานโดยทั่วไป

กรอบงานเชิงวัตถุนั้นถูกใช้ในการพัฒนาโปรแกรมเฉพาะเป็นระยะเวลาหลายปี กรอบงานยุคแรก (ตัวอย่างเช่น MacApp และ Interviews) นั้นถูกสร้างเพื่อสนับสนุนการทำงานด้านกราฟฟิคของส่วนติดต่อกับผู้ใช้ (User Interface) Microsoft Foundation Classes (MFC) เป็นกรอบงานในยุคเดียวกันซึ่งกลายเป็นมาตรฐานเชิงอุตสาหกรรมในการสร้างโปรแกรมประยุกต์สำหรับงานด้านกราฟฟิคสำหรับคอมพิวเตอร์ส่วนบุคคลในเวลาต่อมา นอกจากนี้ยังคงมีการพัฒนาโปรแกรมประเภทอื่นที่ความซับซ้อนมากกว่าอย่าง Telecommunication, Distributed Medical Imaging และ Real-Time Avionics แต่ขาดการใช้กรอบงานหรือใช้กรอบงานที่ไม่มีประสิทธิภาพเพียงพอ ทำให้การพัฒนาโปรแกรมประเภทนี้เกิดต้นทุนและใช้เวลาในการพัฒนาและบำรุงรักษาที่สูง

2.3.2 การจำแนกกรอบงานสำหรับโปรแกรมประยุกต์

กรอบงานเชิงวัตถุนั้นสามารถจำแนกออกเป็นประเภทต่าง ๆ โดยขึ้นอยู่กับเกณฑ์ของการจำแนกซึ่งโดยทั่วไปแล้วประกอบไปด้วย

จำแนกจากจุดประสงค์ในการสร้าง

1) System Infrastructure Framework เป็นกรอบงานที่ถูกพัฒนาขึ้นสำหรับสนับสนุนโปรแกรมประยุกต์ในด้านโครงสร้างพื้นฐาน (Infrastructure) ของโปรแกรมนั้น ๆ ตัวอย่างเช่น Operating System, Communication Framework, Language Processing Tools และกรอบงานเกี่ยวกับส่วนติดต่อกับผู้ใช้ (User Interface) ซึ่งกรอบงานเหล่านี้ถูกสร้างให้เป็นองค์ประกอบภายในโปรแกรมประยุกต์จึงไม่ถูกขายให้กับลูกค้าโดยตรง
2) Middleware Integration Framework เป็นกรอบงานที่ถูกสร้างขึ้นเพื่อเชื่อมการทำงานของ distribute component ในโปรแกรมประยุกต์ กรอบงานประเภทนี้ช่วยเพิ่มความสามารถในการพัฒนาในด้านสภาพเป็นส่วนจำเพาะความสามารถในด้านการเพิ่มความสามารถ (Extensibility) และความสามารถในการนำกลับมาใช้ใหม่ของโครงสร้างพื้นฐานของกรอบงานให้ทำงานกับ Distributed Environment ได้ดีและราบรื่นมากยิ่งขึ้น

3) Enterprise Application Framework กรอบงานประเภทนี้มักถูกออกแบบเพื่อสนับสนุนการพัฒนาโปรแกรมประยุกต์ที่มีการใหญ่และมีความซับซ้อนอย่างโปรแกรมประยุกต์ที่ใช้ในด้านการสื่อสาร วิทยาศาสตร์ อุตสาหกรรมการผลิต และวิศวกรรมด้านการเงิน เมื่อเปรียบเทียบแล้ว Enterprise Application Framework นั้นใช้ต้นทุนในการผลิตมากกกว่า System Infrastructure Framework และ Middleware Integration Framework แต่อย่างไรก็ตาม กรอบงานประเภทนี้ให้ผลกลับคืนที่ดีกว่า เนื่องจากสนับสนุนการทำงานต่อผู้ใช้ปลายทางโดยตรง

นอกจากการการจำแนกประเภทของกรอบงานข้างต้นแล้วยังสามารถจำแนกกรอบงานโดยการพิจารณาจากเทคนิคที่ใช้ในการสืบทอดกรอบงานได้แก่ White Box Framework และ Black Box Framework

จำแนกจากเทคนิคที่ใช้ในการสร้างกรอบงาน

1) White Box Framework นั้นอาศัยแนวคิดเชิงวัตถุอย่างการสืบทอด (Inheritance) และ Dynamics Binding เพื่อให้เพิ่มความสามารถของกรอบงานจาก Functionality เดิมที่มีอยู่ในกรอบงานนั้นถูกนำกลับมาใช้และเพิ่มเติมโดยการ (1) สืบทอดจากคลาส (Classes) ที่ถูกเตรียมไว้ในกรอบงาน (2) ซ้อนทับโดยใช้ฮุคที่กรอบงานนั้น ๆ เตรียมไว้
2) Black Box Framework เป็นกรอบงานที่สนับสนุนความสามารถในด้านการเพิ่มความสามารถ (Extensibility) โดยการกำหนดส่วนต่อประสาน (Interface) สำหรับการนำส่วนโปรแกรมเชิงวัตถุที่ถูกพัฒนาขึ้นใหม่ต่อ (Plug) เข้าสู่กรอบงาน ความสามารถที่มีอยู่เดิมของ กรอบงานนั้นถูกนำกลับมาใช้ใหม่โดย (1) การกำหนดส่วนโปรแกรมตามส่วนต่อประสาน (Interface) และ (2) รวมส่วนโปรแกรมเหล่านั้นเข้ากับกรอบงานโดยใช้รูปแบบเชิงซอฟต์แวร์อย่าง Strategy และ Functor

ในการพัฒนาโปรแกรมประยุกต์ภายใต้กรอบงานแบบ White box นั้นนักพัฒนาโปรแกรมประยุกต์จำเป็นต้องทราบโครงสร้างทางสถาปัตยกรรมของกรอบงานประเภทนี้เป็นอย่างดี แต่ในทางกลับกันการพัฒนาโปรแกรมประยุกต์โดยใช้กรอบงานแบบ Black Box นั้นไม่จำเป็นต้องทราบโครงสร้างทางสถาปัตยกรรมของกรอบงาน เพียงแต่ต้องทราบวิธีการเชื่อมต่อกับส่วนต่อประสานที่กำหนดไว้อย่างดีแล้วเท่านั้น

2.3.3 ข้อดีและข้อด้อยของกรอบงานสำหรับโปรแกรมประยุกต์

ในการพัฒนากรอบงานเชิงวัตถุนั้นถูกสร้างด้วยการนำแนวคิดและเทคโนโลยีหลากหลายมารวมเข้าด้วยกันได้แก่ รูปแบบเชิงซอฟต์แวร์ คลังรหัสคำสั่งเชิงคลาสและส่วนประกอบเชิงซอฟต์แวร์ จึงทำให้กรอบงานเชิงวัตถุนั้นเพิ่มอัตราการผลิดซอฟต์แวร์และลดขั้นตอนการพัฒนา อย่างไรก็ตามด้วยเหตุผลเดียวกัน การที่กรอบงานเชิงวัตถุนั้นประกอบไปด้วยหลายแนวคิดและองค์ประกอบ ทำให้เกิดความซับซ้อน ความพยายามในการนำกรอบงานเชิงวัตถุขนาดใหญ่มาใช้นั้นจึงมักล้มเหลว เว้นแต่มีกระบวนการจัดการในด้านขั้นตอนการพัฒนา ระยะเวลาในการศึกษาการใช้กรอบงาน การจัดการองค์ประกอบที่ถูกนำมารวมกันในกรอบงาน การบำรุงรักษา การตรวจสอบความถูกต้อง การจัดการความผิดพลาด สมรรถภาพและความไม่มีมาตรฐานของการสร้างกรอบงาน

1) การพัฒนาซอฟต์แวร์ที่มีความซับซ้อนให้มีคุณภาพสูงภายใต้กรอบงานที่มีความซับซ้อนนั้นเป็นสิ่งที่ทำได้ยากเป็นอย่างยิ่ง จำเป็นต้องใช้นักพัฒนาที่มีความเชี่ยวชาญสูง
2) การพัฒนาซอฟต์แวร์ภายใต้กรอบงานเชิงวัตถุนั้นจำเป็นต้องเสียเวลาส่วนหนึ่งในการเรียนรู้วิธีการใช้กรอบงานนั้น ๆ ตัวอย่างเช่นการเรียนรู้การใช้งานกรอบงานสำหรับซอฟต์แวร์ในเชิงกราฟฟิคอย่าง MFC หรือ MacApp ใช้เวลาโดยทั่วไปประมาณ 6 – 12 เดือนทั้งนี้ทั้งนั้นขึ้นอยู่กับประสบการณ์ของนักพัฒนาด้วย เว้นแต่การศึกษากรอบงานนั้นสามารถนำมาใช้ในการพัฒนาซอฟต์แวร์ได้หลายโครงการในคราวเดียวซึ่งย่อมถือเป็นการคุ้มค่าต่อต้นทุนและเวลา
3) การใช้กรอบงานหลายกรอบงานทำงานร่วมกัน (Integration of Multiple Framework) นั้นทำให้กระบวนการพัฒนาซอฟต์แวร์นั้นดีขึ้น (ตัวอย่างเช่นการทำงานร่วมกันของกรอบงานเกี่ยวกับ GUI, Communication และ Database) อย่างไรก็ตามกรอบงานนั้นถูกออกแบบในเริ่มแรกให้ทำงานร่วมกันภายในเป็นหลัก ไม่ได้ถูกออกแบบให้ทำงานร่วมกับส่วนโปรแกรมภายนอกที่อยู่ภายในกรอบงานอื่นจึงทำให้เกิดปัญหาเกี่ยวกับ Concurrency/Distribution Architecture และ Event Dispatching Model
4) ความต้องการในเชิงซอฟต์แวร์ของโปรกรมแกรมประยุกต์โดยทั่วไปจะถูกเปลี่ยนแปลงอยู่บ่อย ๆ เช่นเดียวกัน ความต้องการหลักของกรอบงานย่อมถูกเปลี่ยนได้เช่นกัน จึงทำให้เกิดการปรับปรุงเปลี่ยนแปลงกรอบงาน เมื่อกรอบ

5) แม้ว่าการออกแบบกรอบงานเป็นส่วนย่อย (Module) นั้นทำให้จำกัดขอบเขตของผลกระทบอันเกิดจากข้อบกพร่องของซอฟต์แวร์ การตรวจสอบและการแก้ไขข้อบกพร่องที่เกิดขึ้น แต่ยังคงมีข้อเสียอื่น ๆ อีกด้วย ได้แก่ ส่วนโปรแกรมที่ได้รับการออกแบบให้มีความยืดหยุ่นสูง (Generic Component) นั้นยากในการตรวจสอบในเชิงนามธรรม เนื่องจากส่วนโปรแกรมเหล่านี้ได้รับการออกแบบโดยปราศจากรายละเอียด ส่วนโปรแกรมเหล่านี้จะทำงานได้ต่อเมื่อมีการสร้าง ส่วนโปรแกรมที่เป็นรูปธรรม (Concrete Component) แล้วเท่านั้น ขณะที่การออกแบบ Generic Component นั้นก่อให้เกิดความยืดหยุ่นสูงแต่ทำให้เกิดความยากและซับซ้อนในการทดสอบในระดับโปรแกรมย่อย (Module Testing) เนื่องมาจากส่วนโปรแกรมไม่สามารถทำงานได้ด้วยคำสั่งที่เกิดขึ้นจริงจากคลาสย่อยของ Generic Component นอกจากนี้แล้วปัญหายังเกิดกับกรอบงานที่ใช้แนวคิดของ Inversion of Control ในการออกแบบกรอบงาน ซึ่งเป็นแนวคิดที่กลับการไหลของการควบคุมโปรแกรมทำให้ยากในการแก้ไขข้อผิดและยากต่อการเข้าใจพฤติกรรมของกรอบงานนั้น

2.3.4 แนวโน้มในอนาคต

ในช่วงหลายปีที่ผ่านมานี้นักวิจัยและนักพัฒนาพยาพยายามวิเคราะห์และออกแบบให้ได้มาซึ่งกระบวนการจัดการซึ่งเกี่ยวเนื่องกับกรอบงานดังต่อไปนี้

1) Reducing Framework Development Effort โดย Fayad and Schmidt (1997) กล่าวว่าแต่เดิมนั้นกรอบงานซึ่งนำกลับมาใช้ใหม่ได้ถูกพัฒนามาจากโปรแกรมระบบหรือประยุกต์ ที่มีอยู่ก่อนแล้ว แต่กระบวนการพัฒนานี้กลับเป็นกระบวนการพัฒนาที่ช้าและขาดความแน่นอน ไม่สามารถทำนายได้เนื่องจากรูปแบบเชิงซอฟต์แวร์และหลักการออกแบบและแก่นของกรอบงานนั้นเป็นแบบ “Bottom-Up” แต่อย่างไรก็ตามขณะนี้กรอบงานได้ถูกพัฒนาขึ้นเป็นตัวอย่าง เป็นจำนวนมาก นักวิจัยและนักพัฒนาคาดหมายว่าการพัฒนากรอบงานในยุคถัดมาจะใช้ความรู้ที่มีอยู่เดิมเพื่อให้ได้มาซึ่งการออกแบบและพัฒนากรอบงานาที่มีประสิทธิภาพสูงอย่างรวดเร็ว

2) Grater focus on Domain-Specific Enterprise Framework ในที่นี้ Campbell and Islam (1993) และ Schmidt (1997) กล่าวถึงกรอบงานที่มีอยู่ในปัจจุบันนั้น

ถูกสร้างเพื่อสนับสนุนโปรแกรมประยุกต์ในส่วนของโครงสร้างพื้นฐาน ตัวอย่างเช่นส่วนติดต่อผู้ใช้และระบบสื่อสาร อย่างไรก็ตามความรู้และประสบการณ์ในการพัฒนากรอบงานได้เพิ่มมากขึ้นตามลำดับ การพัฒนากรอบงานเริ่มมุ่งไปสู่การพัฒนาให้เป็นกรอบงานที่เฉพาะเจาะจงมากขึ้นโดยขอบเขตของความสนใจนี้มุ่งไปที่ขอบเขตเชิงธุรกิจเป็นหลัก ยกตัวอย่างเช่น ภาคธุรกิจการผลิต การธนาคาร ภาคธุรกิจการประกัน และระบบการแพทย์ เป็นต้น

3) Black-box Framework ผู้เชี่ยวชาญ ได้แก่ Campbell and Islam (1993) และ Johnson and Foote (1988) มักกล่าวแสดงความเห็นในควรออกแบบกรอบงานในรูปแบบ Black-Box Framework มากกว่า White-Box Framework เนื่องจากกรอบงานแบบ Black-Box นั้นเน้นความสัมพันธ์เชิงวัตถุแบบพลวัต (โดยอาศัยรูปแบบเชิงซอฟต์แวร์อย่าง Factory, Bridge และ Strategy) มากกว่าความสัมพันธ์แบบสถิต (Static Class Relationship) ซึ่งทำให้สามารถเพิ่มเติมและกำหนดค่าให้กับกรอบงานแบบ Black-Box อย่างพลวัต

4) Framework Documentation การอธิบายกรอบงานด้วยเอกสารนั้นมีความจำเป็นอย่างยิ่งที่ใช้ในกรอบงานซึ่งมีขนาดใหญ่และมีความซับซ้อนสูง แต่อย่างไรก็ตามการสร้างเอกสารในการอธิบายกรอบงานนั้นใช้ต้นทุนที่สูงและใช้เวลามาก อีกประเด็นที่เกี่ยวข้องคือเอกสารที่มีอยู่นั้นอธิบายในรายละเอียดเชิงลึก (Low-Level) มากเกินไปซึ่งไม่สามารถอธิบายความสัมพันธ์ระหว่างส่วนโปรแกรมได้ ดังนั้นแล้ว Fayad and Schmidt (1997) แสดงความเห็นเกี่ยวกับกรอบงานในอนาคตว่าควรใช้ Reverse-Engineering กับโครงสร้างของคลาสในกรอบงาน ที่มีความซับซ้อนเพื่อช่วยในการสร้างเอกสารที่มีความแม่นยำและมีประโยชน์ซึ่งควรอธิบาย ในรูปแบบนามธรรมขั้นสูง (High-Level Abstraction) อย่างรูปแบบเชิงซอฟต์แวร์

5) Process for Managing Framework Development กรอบงานนั้นถูกสร้างให้เป็นรูปแบบทั่วไป มีความเป็นนามธรรมสูงซึ่งถูกเตรียมสำหรับการสร้างโปรแกรมประยุกต์ ที่เฉพาะเจาะจงขึ้นจริง จึงเปรียบกรอบงานนั้นเป็นผลิตภัณฑ์ที่เป็นนามธรรมดังนั้นจึงยากที่จะจัดการคุณภาพและผลผลิตที่เกิดขึ้นโดยกรอบงาน อย่างไรก็ตาม Fayad and Schmidt (1997) เชื่อว่าการทำ Prototyping นั้นสามารถลดความเสี่ยงและนำมาซึ่งความสำเร็จในกระบวนการพัฒนากรอบงานได้

6) Framework Economics ตัวชี้วัดทางเศรษฐศาตร์นั้นสามารถนำมาใช้ในกระบวนการพัฒนากรอบงานได้ดังนี้

- Determining Effective Framework Cost Metrics คือตัวชี้วัดที่บ่งชี้ความสามารถในการนำกลับมาใช้ใหม่ของกรอบงาน และการสร้างโปรแกรมประยุกต์ตั้งแต่กระบวนการแรกเริ่ม
- Cost Estimation คือตัวชี้วัดที่ใช้ในการประเมินหรือทำนายการซื้อการสร้างและการปรับปรุงกรอบงาน
- Investment Analysis and Justification ใช้ในการชี้วัดผลประโยชน์ที่กลับคืนมาหลังจากการลงทุนสร้างกรอบงาน

7) Fayad and Schmidt (1997) มีความเชื่อถึงตัวชี้วัดทางเศรษฐศาสตร์นี้เป็นตัวเชื่อมช่องว่างระหว่างเทคนิค การจัดการและการเงินในการสร้าง การซื้อและการปรับปรุงกรอบงาน

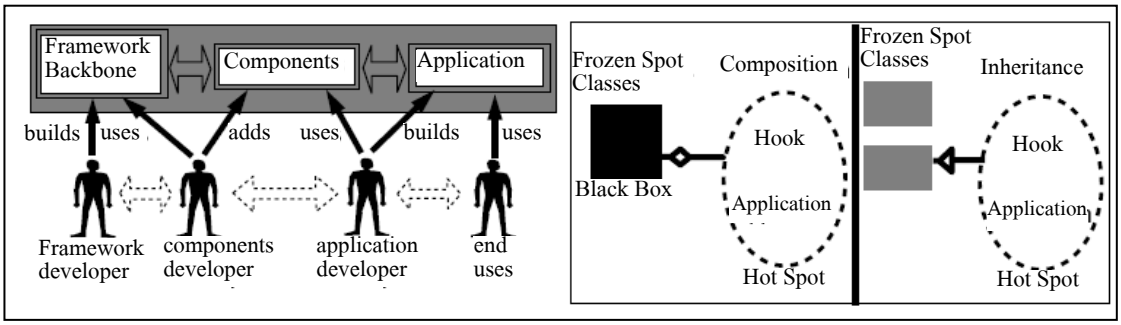
8) Framework Standard กรอบงานมีความซับซ้อนและได้รับการยอมรับมากยิ่งขึ้นดังนั้นจึงมีความจำเป็นอย่างยิ่งในการกำหนดมาตรฐานในสำหรับกรอบงาน อันได้แก่ มาตรฐานเกี่ยวกับการพัฒนากรอบงาน (Framework Development) การปรับปรุงกรอบงาน (Framework Adaptation) การรวมและการสร้างความสัมพันธ์ในการทำงานร่วมกันของกรอบงาน (Framework Interoperability and Integration) โดยมาตรฐานนี้ทำให้กรอบงานมีรูปแบบที่ชัดเจนและสามารถช่วยลดต้นทุนในการพัฒนากรอบงานอีกด้วย

2.4 กรอบงานสำหรับการออกแบบกรอบงานเชิงวัตถุ

กรอบงานเชิงวัตถุถูกสร้างสำรับการนำกลับมาใช้ใหม่ในขอบเขตเฉพาะ (Specific Domain) กระบวนการพัฒนากรอบงานมักเริ่มต้นด้วยการเป็นกรอบงานแบบเปิด (Open Framework) และก้าววิวัฒนาการไปสู่โปรแกรมประยุกต์แบบปิด (Close Application) ในบทความ Parsons, Rashid, Speck, and Telea (1999) นำเสนอแนวทางแบบ Component-Based สำหรับการออกแบบกรอบงานอย่างยืดหยุ่นโดยเน้นไปที่ส่วนต่อประสาน (Interface) ของกรอบงานซึ่งถูกใช้สำหรับเพื่อเติม (Plugging-in) ส่วนโปรแกรมใหม่ ๆ ในภายหลัง และยังนำเสนอมุมมองบทบาทของผู้ที่เกี่ยวข้องกับกรอบงานใหม่ โดยกล่าวถึงการวิเคราะห์บทบาทของผู้ที่สัมพันธ์กับกรอบงานแบบเดิมนั้นอยู่ในรูปแบบ Developer/End-user ซึ่งไม่มีความยืดหยุ่น การมองบทบาทในรูปแบบของ “Actor” ซึ่งสามารถปรับเปลี่ยนพฤติกรรมของกรอบงานได้นั้นทำให้แนวคิดเกี่ยวกับกรอบงาน

2.4.1 Flexible Elements of Object Oriented Framework

Parsons et al. (1999) เป็นบุคคลที่กล่าวถึงระหว่างกระบวนการพัฒนากรอบงานนั้นมักถูกสร้างเป็นแบบ White Box Framework ในขั้นเริ่มต้นก่อนถูกเปลี่ยนไปเป็น Black Box Framework ซึ่งในบทความได้นำเสนอวิธีการภายใต้ Component Driven ซึ่งสนับสนุนทั้ง White Box และ Black Box Framework



รูปที่ 2.22 แสดง Hot Spot ใน Black Box และ White Box Framework (ขวา)
บทบาทของผู้ใช้กรอบงาน (ซ้าย)

1) ตัวละครและบทบาท (Actor and Their Roles)

ในระหว่างกระบวนพัฒนาโปรแกรมประยุกต์ภายใต้พื้นฐานของกรอบงานนั้น ผู้ที่มีความเกี่ยวข้องสามารถถูกจำแนกเป็นหลายประเภท ได้แก่ Framework Developer (FD), Component Developer (CD), Application Developer (AD) และ End User (EU) ดังรูปที่ 2.22 (ซ้าย) โดยนักพัฒนากรอบงานมีหน้าที่ในการพัฒนาแกน (Core) ของกรอบงานและเตรียมเครื่องมือหรือส่วนต่อประสาน (Interface) ในการเข้าถึงหรือการเชื่อมต่อเพิ่มความสามารถ (Plugging-in) ให้กับกรอบงาน นักพัฒนาส่วนโปรแกรม (Component Developer: CD) นั้นมีหน้าที่พัฒนาส่วนโปรแกรมใหม่ๆ และนำไปรวมเข้าสู่กรอบงานโดยผ่านส่วนต่อประสานที่นักพัฒนากรอบงานได้สร้างไว้ นักพัฒนาโปรแกรมประยุกต์ (Application Developer: AD) นำส่วนโปรแกรมนั้น เข้ากับโปรแกรมซึ่งพัฒนาตามความต้องการของผู้ใช้ (End-User: EU)

กระบวนการนำกลับมาใช้ใหม่เกิดได้ใน 2 ส่วน กล่าวคือ ส่วนแรก นักพัฒนา ส่วนโปรแกรมสามารถนำรหัสต้นฉบับ (Source Code) กลับมาใช้ใหม่ได้โดยอาศัย

ความสามารถพื้นฐานของที่มีอยู่แล้วในภาษา (Programming Language) ที่ใช้พัฒนาอย่างการสืบทอดหรือการทำแม่แบบ (Template) ส่วนที่ 2 นั้นนักพัฒนาโปรแกรมประยุกต์สามารถนำความสามารถในการนำกลับมาใช้ใหม่โดยผ่านกลไกที่มีอยู่แล้วของกรอบงานในการสร้างโปรแกรมประยุกต์ที่เฉพาะเจาะจงตรงตามความต้องการเชิงซอฟต์แวร์ของลูกค้า

จากรูปที่ 2.22 (ซ้าย) นั้นดังได้กล่าวไปส่วนหนึ่งแล้ว แสดงให้เห็นโดยชัดแจ้งว่าบทบาทของตัวละครที่มีต่อกรอบงานนั้นมีความยืดหยุ่นสูง แนวคิดของกรอบงานตามบทความนี้ทำให้บทบาทของตัวละครถูกจำแนกออกและเป็นอิสระต่อกันอย่างชัดเจน งานต่าง ๆ ที่เกิดขึ้นโดยที่ตัวละครสร้างขึ้นมาไม่มีผลกระทบซึ่งกันและกัน ยิ่งไปกว่านั้นแล้วส่วนต่อประสานที่ถูกพัฒนาโดยนักพัฒนากรอบงานนั้นยังทำให้กรอบงานสามารถถูกเชื่อมต่อเพิ่มส่วนโปรแกรมใหม่ ๆ ได้อีกด้วย

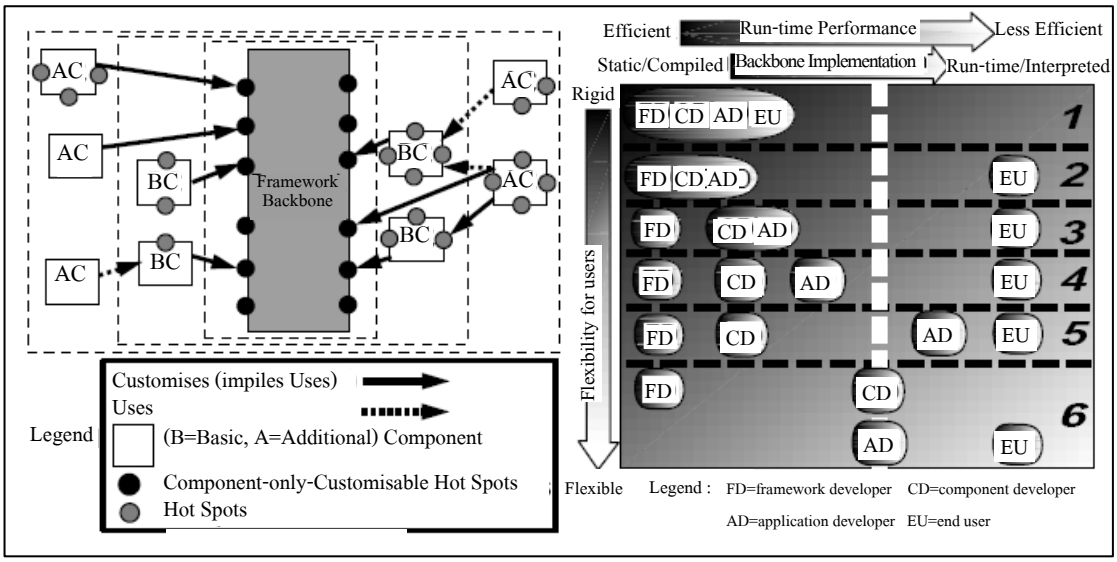
2) A Component-Driven Approach to Framework Design

ในบทความของ Parsons et al. (1999) นั้นยังคงเห็นด้วยในการพัฒนาส่วนโปรแกรมให้มี Cohesive ที่สูง แต่มี Coupling ที่ต่ำ โดยส่วนโปรแกรมที่ถูกรวมเข้ากับกรอบงานนั้นจะถูกจัดการโดยแกนของกรอบงานที่ถูกเรียกว่า "Backbone" ซึ่งได้รับการออกแบบให้อยู่ในขอบเขตของความต้องการเชิงซอฟต์แวร์ที่ชัดเจนและมีความสามารถในการช่วยสนับสนุนการสื่อสาร การแลกเปลี่ยนข้อมูล กลไกการทำงานร่วมกันของส่วนโปรแกรมต่าง ๆ และ Hot Spot สำหรับการเชื่อมต่อส่วนโปรแกรมใหม่ ๆ

ในกระบวนการการพัฒนากรอบงานนั้นหลังจากได้ออกแบบและพัฒนาแก่นของกรอบงานเสร็จแล้ว กระบวนการในขั้นถัดมาคือการพัฒนาส่วนโปรแกรมพื้นฐานโดยนักพัฒนากรอบงานหรือนักพัฒนาส่วนโปรแกรมซึ่งจะทำให้ได้กรอบงานที่สมบูรณ์ ส่วนโปรแกรมใหม่ที่ถูกสร้างขึ้นโดยนักพัฒนาส่วนโปรแกรมหรือนักพัฒนาโปรแกรมประยุกต์เพื่อสนับสนุนความต้องการในเชิงซอฟต์แวร์ที่เกิดขึ้นใหม่นั้นจะไม่ถูกเรียกว่า "Backbone" หากแต่เพียงเป็น "ส่วนโปรแกรมใหม่ (Plugged-in Component)" ที่ทำงานร่วม Backbone และส่วนโปรแกรมพื้นฐานเท่านั้น

Parsons et al. (1999) กล่าวถึงกระบวนการสร้างกรอบงานไว้ 4 ขั้นตอนดังนี้

1) Application Domain and Role Definition ขั้นตอนแรกนั้นต้องระบุขอบเขตของกรอบงานให้ชัดเจนโดยการจำแนกความต้องการเชิงซอฟต์แวร์และบทบาทของผู้ที่มีความเกี่ยวข้องกับกระบวนการพัฒนากรอบงาน



รูปที่ 2.23 แสดงแกนของกรอบงานและ Plugged-in Component (ซ้าย)
และกระบวนการพัฒนากรอบงาน (ขวา)

2) Backbone Development เป็นขั้นตอนที่แกนของกรอบงานถูกพัฒนาโดยนักพัฒนากรอบงาน

3) Basic Component Development โดยขั้นที่ 3 นี้เป็นกระบวนการการพัฒนาส่วนโปรแกรมพื้นฐานและ Hot spot ของกรอบงาน โดยส่วนโปรแกรมต่าง ๆ ที่พัฒนาขึ้นนั้นควรอยู่ภายใต้ความต้องเชิงซอฟต์แวร์หรือขอบเขตของกรอบงานที่ได้กำหนดไว้ในขั้นตอนแรก

4) Additional Components Development ในท้ายที่สุดนักพัฒนาส่วนโปรแกรมหรือนักพัฒนาโปรแกรมประยุกต์เริ่มพัฒนาส่วนโปรแกรมเพิ่มเติมจากส่วนโปรแกรมพื้นฐานและนำไปเชื่อมต่อกับกรอบงานนั้นโดยผ่านตัวประสานหรือ Hot Spot ที่ได้เตรียมไว้ ทั้งนี้ทั้งนั้นส่วนโปรแกรมที่นำมาเพิ่มนั้นต้องตรงตามขอบเขตของกรอบงานที่กำหนดไว้ก่อนแล้วด้วย

3) Backbone Architecture and Flexibility Requirement

จากบทความ Parsons et al. (1999) แบ่งกรอบงานออกเป็น 6 ประเภทหลัก ดังต่อไปนี้

1) Monolithic Systems กรอบงานประเภทนี้เป็นรูปแบบดั้งเดิมโดยการเปรียบกรอบงานเหมือนหินก้อนใหญ่เพียงก้อนเดียวเพราะกรอบงานประเภทนี้ถูกพัฒนาโดยการเก็บความต้องการของระบบทั้งหมดและพัฒนากรอบงานให้เสร็จเพียงครั้งเดียว ข้อดีของกรอบงานประเภทนี้คือส่วนโปรแกรมต่าง ๆ ทำงานร่วมกันได้อย่างมีประสิทธิภาพ ในทางกลับกันหากกรอบงานประเภทนี้มีความอ่อนไหวต่อความต้องการเชิงซอฟต์แวร์ใหม่หรือความต้องการที่เปลี่ยนแปลงไปมาก
2) Modular System กรอบงานนี้แบ่งผู้ใช้กรอบงานออกจากผู้พัฒนากรอบงานซึ่งผู้ใช้กรอบงานนั้นใช้กรอบงานโดยผ่านทางส่วนต่อประสานที่ถูกเตรียมไว้ แต่ยังคงมีข้อเสียในเรื่องของการเพิ่มส่วนโปรแกรมเมื่อมีความต้องการเชิงซอฟต์แวร์เพิ่มขึ้นอยู่เช่นเดิม
3) Application Libraries เป็นกรอบงานแบบแรกที่ใช้แนวคิดของการนำกลับมาใช้ใหม่โดยอาศัยหลักการเชิงวัตถุ (Object Oriented Paradigm) ซึ่งถูกออกแบบเพียงครั้งเดียวโดยนักพัฒนากรอบงาน แต่ผู้พัฒนาโปรแกรมประยุกต์สามารถนำไปใช้ในการพัฒนาโปรแกรมประยุกต์ต่าง ๆ กันได้หลาย ๆ โปรแกรม ทว่ากรอบงานประเภทนี้ยังคงแบ่งแยกบทบาทของผู้เกี่ยวข้องอย่างนักพัฒนาส่วนโปรแกรมและนักพัฒนาโปรแกรมประยุกต์ยังไม่ชัดเจนเพียงพอ
4) Compiled Component-Based Framework เป็นกรอบงานที่มีลักษณะคล้าย กับ Application Libraries กรอบงานนี้ถูกแบ่งออกเป็น 2 ส่วนได้อย่างชัดเจนคือส่วนที่เป็น Fix Part และ Variant Part โดย Fix part นั้นถูกพัฒนาเพียงครั้งแรกและกลายเป็นแก่นของกรอบงาน (Framework Backbone) ในที่สุด อีกส่วนหนึ่งนั้นคือ Variant Part นี้เป็นส่วนที่รอรับการเชื่อมต่อส่วนโปรแกรมใหม่อันเกิดจากความต้องการเชิงซอฟต์แวร์ที่มากยิ่งขึ้น
5) Dynamic Component-Based Framework เป็นกรอบงานที่ได้รับการพัฒนาต่อจากกรอบงานที่ได้รับการคอมไพล์แล้วโดยการเพิ่มส่วน

โปรแกรมเข้าไปในกรอบงานที่กำลังถูกใช้งานโดยโปรแกรมประยุกต์ที่ถูกสร้างบนพื้นฐานของกรอบงานนั้น ๆ

6) Single Language Framework เป็นกรอบงานที่เกิดจากการพัฒนาโดยใช้ภาษาในการพัฒนาเพียงภาษาเดียวในทุกส่วนและในทุกช่วงเวลาในการพัฒนากรอบงาน นักพัฒนากรอบงาน นักพัฒนาส่วนโปรแกรมและนักพัฒนาโปรแกรมประยุกต์นั้นต่างพัฒนาส่วนโปรแกรมของตนเองโดยใช้ภาษาเดียวกัน ซึ่งทำให้การทำงานของกรอบงานนั้นเป็นไปอย่างมีประสิทธิภาพสูงสุด ลดช่องว่างที่เกิดขึ้นจากการพัฒนากรอบงานโดยใช้หลากหลายภาษา

2.5 การเลือกกรอบงานเชิงวัตถุ (Choosing an Object-Oriented Domain Framework)

ในการตัดสินใจว่ากรอบงานใด ๆ เหมาะสมและนำมาซึ่งความสามารถพื้นฐานในการพัฒนาโปรแกรมประยุกต์หรือไม่นั้นเป็นหนึ่งในการตัดสินใจสำคัญของนักพัฒนาที่จะต้องกระทำโดยเฉพาะอย่างยิ่งการตัดสินใจกับกรอบงานที่เพิ่งถูกสร้างขึ้นใหม่ โดยปกติประสบการณ์ในการใช้กรอบงานเป็นสิ่งที่ช่วยในการตัดสินใจ หากแต่นักพัฒนาเกิดความไม่แน่ใจถึงความสามารถและไม่สามารถรับรู้ได้ถึงความเหมาะสมของกรอบงานกับโปรแกรมประยุกต์ที่กำลังพัฒนาอยู่จนกว่าจะได้เข้าสู่กระบวนการการพัฒนาแล้ว ส่งผลให้โครงการนั้น ๆ ต้องถูกละทิ้งไปหรือต้องกลับไปสู่ขั้นตอนการพัฒนาแรกเริ่มใหม่ในที่สุด ดังนั้นนักพัฒนาจึงต้องการพื้นฐานสำหรับทำการตัดสินใจเบื้องต้นเพื่อพิจารณาความเหมาะสมที่จะลงทุนด้วยเวลาในการทำความใจและใช้กรอบงานนั้น ๆ ในบทความ Froehlich, Hoover, and Sorenson (2000) นำเสนอขั้นตอนสำหรับพิจารณาถึงความเหมาะสมของกรอบงานและความเข้ากันได้ระหว่างกรอบงานกับความต้องการเชิงซอฟต์แวร์ได้แก่ขั้นตอนในการค้นหาขีดจำกัด สุดและจำนวนของความไม่แน่นอน หรือความกำกวมในการใช้กรอบงาน

2.5.1 กระบวนการตัดสินใจ

โดยทั่วไปแล้วมี 3 ขั้นตอนในการตัดสินใจว่ากรอบงานเหมาะสมสำหรับโปรแกรมประยุกต์นั้น ๆ หรือไม่ ได้แก่

1) พิจารณาว่ากรอบงานควรถูกปฏิเสธโดยทันทีหรือไม่โดยการตรวจสอบว่าขีดจำกัดของกรอบงานนั้นขัดต่อความต้องการเชิงซอฟต์แวร์ของโปรแกรมประยุกต์โดยชัดแจ้งหรือไม่
2) พิจารณาว่ากรอบงานเหมาะสมกับโปรแกรมประยุกต์ที่กำลังพัฒนาอย่างเห็นได้ชัดหรือไม่โดยการเปรียบเทียบและจับคู่ความต้องการเชิงซอฟต์แวร์ของโปรแกรมประยุกต์เข้ากับฮุคซึ่งเป็นตัวกำหนดและอธิบายกรอบงาน โดยฮุคเป็นสิ่งที่อธิบายความสามารถของกรอบงานและยังอธิบายวิธีการในการเพิ่มความสามารถให้กับกรอบงานเพื่อให้ตรงกับความต้องการของโปรแกรมประยุกต์ที่ได้รับซึ่งหากมีฮุคที่สนับสนุนสำหรับทุก ๆ ความต้องการของโปรแกรมประยุกต์นั่นแสดงถึงความเหมาะสมของกรอบงานในการพัฒนาโปรแกรมประยุกต์นั้น
3) ประเมินระดับของความไม่แน่นอน ถ้าหากโปรแกรมประยุกต์มีความต้องการใดซึ่งฮุคมิได้กำหนดไว้นั้นหมายถึงความไม่แน่นอนของการใช้กรอบงานนั้น ถึงแม้ภายหลังกรอบงานอาจจะเหมาะสำหรับโปรแกรมประยุกต์แต่ยังคงไม่สามารถรับประกันได้ถึงความสามารถเหล่านั้นโดยยิ่งมีความไม่แน่นอนหรือกำกวมมากเท่าใดนักพัฒนาย่อมทำงานในการพัฒนาโปรแกรมประยุกต์จากกรอบงานมากขึ้นเท่านั้น

2.5.2 ขีดจำกัด-สิ่งที่กรอบงานไม่สามารถกระทำได้

ทุก ๆ กรอบงานจะต้องมีขีดจำกัดซึ่งเกิดจากการตัดสินใจในการออกแบบของนักพัฒนากรอบงานนั้นเอง โดยทั่วไปแล้วขีดจำกัดของกรอบงานนั้นจะไม่สามารถถูกเปลี่ยนหรือมองข้ามไปโดยง่าย อีกทั้งยังเป็นสาเหตุสำคัญที่ทำให้โครงการไม่สามารถบรรลุเป้าหมายหากนักพัฒนาละเลยต่อขีดจำกัดเหล่านั้น นอกจากนี้กรอบงานมักจะไม่มีเอกสารเกี่ยวกับขีดจำกัดของกรอบงานนั้นจึงมักต้องใช้การทดสอบโดยการใช้กรอบงานนั้นหรือศึกษาจากผู้เชี่ยวชาญ ในบทความ Froehlich et al. (2000) เสนอถึงความสำคัญของ รูปแบบเชิงซอฟต์แวร์ซึ่งมักถูกใช้ในการออกแบบกรอบงานดังนั้นแล้วขีดจำกัดส่วนหนึ่งของกรอบงานที่มีนั้นจึงขึ้นอยู่กับขีดจำกัดของรูปแบบเชิงซอฟต์แวร์ที่ถูกออกแบบในนั่นเอง

2.5.3 ฮุค – สิ่งที่อธิบายความสามารถของกรอบงาน

เทคนิคอย่างคุคบุค (Cookbooks), แพตเทิร์น (Patterns) และ ฮุคถูกนำมาพัฒนาเพื่อช่วยในการอธิบายถึงวิธีการใช้และความสามารถของกรอบงานโดยฮุคซึ่งเป็นวิธีการที่นิยมใช้กันมากที่สุดนั้นสามารถกำหนดได้ 3 รูปแบบซึ่งจะกล่าวต่อไป ฮุคนั้นถูกกำหนดอยู่ที่ Hot Spot ของกรอบงานโดยแต่ละฮุคนั้นจะสนับสนุนความต้องการเฉพาะเจาะจงโดยได้ถูกอธิบายไว้ในเอกสารซึ่งกล่าวถึงวิธีการทำงานและเพิ่มความสามารถของกรอบงานเหล่านั้น

ฮุคนั้นจำแนกได้ 3 รูปแบบต่อไปนี้

1) Option คือส่วนโปรแกรมซึ่งถูกสร้างให้ใช้งานอย่างง่ายครบถ้วนและถูกกำหนดไว้ในกรอบงาน
2) Pattern คือส่วนโปรแกรมซึ่งถูกกำหนดค่าด้วยตัวแปรเสริม (Parameters) หรือการเชื่อมต่อด้วยวิธีการซึ่งถูกกำหนดไว้เป็นอย่างดี หรือมีรูปแบบชัดเจน
3) Open คือส่วนของกรอบงานที่ถูกกำหนดให้มีรองรับการเพิ่มความสามารถได้โดยการเพิ่มส่วนโปรแกรมใหม่ ๆ เข้าไปในกรอบงาน

2.5.4 การทำงานภายใต้ความไม่แน่นอน

เมื่อกรอบงานซึ่งคิดว่าเหมาะสมแต่กลับปรากฏในภายหลังว่ากรอบงานนั้นมีความผิดพลาดในการสนับสนุนความต้องการในทางซอฟต์แวร์บางประการก่อให้เกิดความไม่ชัดเจนในการพัฒนาขึ้น บ่อยครั้งที่นักพัฒนาโปรแกรมประยุกต์มีความรู้ไม่เพียงพอในการตัดสินใจว่ากรอบงานนั้นเหมาะสมหรือไม่ซึ่งอาจทำให้นักพัฒนาต้องปรับเปลี่ยนหรือแก้ไขกรอบงานในบางประการทั้งที่ความสามารถเหล่านั้นได้ถูกกำหนดไว้ในกรอบงานอยู่ก่อนแล้วแต่มิได้ถูกอธิบายไว้โดยเอกสาร ด้วยเหตุที่เป็นไปไม่ได้ในการกำจัดความไม่แน่นอนที่เกิดขึ้นเหล่านั้นนักพัฒนาจึงจำเป็นต้องรู้จักวิธีการจัดการความไม่แน่นอนหรือกำกวมที่เกิดขึ้น ภายใต้ความกำกวมนักพัฒนาโปรแกรมประยุกต์ต้องมีความรู้ความเข้าใจในเชิงลึกเกี่ยวกับสถาปัตยกรรมและรายละเอียดการสร้างกรอบงานที่ถูกเลือก อีกทั้งควรมีเอกสารที่ชัดเจนอย่างรูปแบบเชิงซอฟต์แวร์หรือ Architecture Description Language นอกจากนี้แล้วเครื่องมือ (Tools) ที่ช่วยเพิ่มความเข้าใจในการทำงานของกรอบงานอย่างเช่น Exploring Exemplar นั้นช่วยให้นักพัฒนาเข้าใจกรอบงานที่สนใจอย่างลึกซึ้ง เครื่องมือในกลุ่มของ Rapid Development of Application อย่าง Interactive Development Environment (IDE) ซึ่งประกอบไปด้วยส่วนต่อประสานกราฟิกกับผู้ใช้ (GUI) จำนวนมากช่วยให้พัฒนาต้นแบบ (Prototype) ของโปรแกรมประยุกต์เป็นอย่างรวดเร็วอีกทั้งยังทำให้เข้าใจขั้นตอนการใช้กรอบงาน ค้นพบฮุคใหม่ ๆ และเข้าถึงขีดจำกัดของกรอบงานอีกด้วย

2.5.5 สรุป

ในการใช้กรอบงานเชิงวัตถุนั้น สิ่งหนึ่งที่ใช้ในการตัดสินใจคือการพิจารณาว่ากรอบงานที่ถูกเลือกนั้น ๆ เหมาะสมกับความต้องการเชิงซอฟต์แวร์หรือไม่ เนื่องด้วยกรอบงานนั้นมีความซับซ้อน ต้องการความรู้ความเข้าใจที่ลึกซึ้ง ดังนั้นจึงเป็นการเสียเวลาพอสมควรในการตัดสินใจการใช้กรอบงานเหล่านั้น การเข้าถึงข้อมูลเกี่ยวกับความเหมาะสมในการใช้งานกรอบงานเป็นสิ่งสำคัญ ขีดจำกัดและโครงสร้างหรือรูปแบบการออกแบบของกรอบงานนั้นแสดงให้เห็นถึงความสามารถของกรอบงานว่าทำสิ่งใดไม่ได้ การอธิบายด้วยฮุคใช้ในการแสดงให้เห็นว่ากรอบงานทำสิ่งใดได้ ท้ายที่สุดไม่ว่ากรอบงานใด ๆ คงประกอบไปด้วยความกำกวมถึงความเหมาะสมในการพัฒนาโปรแกรมประยุกต์หนึ่งดังที่กล่าวไว้ข้างต้น

บทที่ 3
วิธีดำเนินการวิจัย

ในบทความส่วนนี้เป็นการนำเสนอวิธีดำเนินการวิจัยโดยการนำความรู้ แนวคิดและวิธีต่าง ๆ ทางด้านวิศวกรรมซอฟต์แวร์เพื่อใช้ในการพัฒนากรอบงานเชิงวัตถุ โดยมีรายการนำเสนอได้แก่การศึกษาและวิเคราะห์ การออกแบบระบบ การพัฒนาระบบและการใช้งานระบบ

3.1 การศึกษาและการวิเคราะห์

จากการศึกษาในเบื้องต้นนั้นพบว่าการสร้างกรอบงานเชิงวัตถุนั้นประกอบไปด้วยวิธีและแนวคิดต่าง ๆ ดังต่อไปนี้

3.1.1 Unified Modeling Language

ในกระบวนการพัฒนาซอฟต์แวร์โดยทั่วไปนั้นจำเป็นต้องมีมาตรฐาน รูปแบบของเอกสารและแผนภาพ (Diagram) ต่าง ๆ เกิดขึ้นในกระบวนการเพื่อเป็นตัวกลางในการสื่อสารระหว่างทีมที่พัฒนาซอฟต์แวร์สำหรับวิเคราะห์ความต้องการของในเชิงซอฟต์แวร์ระบบการออกแบบระบบ การเขียนรหัสคำสั่ง การทดสอบระบบและในท้ายที่สุดนั้นคือการบำรุงรักษาระบบ ตัวอย่างเช่น การใช้แผนภาพการไหลของข้อมูล (Data Flow Diagram) เพื่อวิเคราะห์และอธิบายขั้นตอนการทำงานและการส่งข้อมูลกันระหว่างส่วนโปรแกรม นอกจากนี้แล้วการใช้แผนภาพหรือการจำลองโปรแกรมโดยใช้สัญลักษณ์นั้นเป็นการเพิ่มระดับความเป็นนามธรรมในการวิเคราะห์และออกแบบทำให้ช่วยปิดบังข้อมูลในรายละเอียดเพื่อให้ง่ายต่อการวิเคราะห์และการนำกลับมาใช้ใหม่

กระบวนการพัฒนาซอฟต์แวร์เชิงวัตถุนั้นใช้ภาษาเชิงสัญลักษณ์อย่าง Unified Modeling Language (UML) ซึ่งประกอบด้วยแผนภาพที่แสดงแนวคิดในมุมมองความต้องการหลัก (Functional Requirement View) มุมมองแบบโครงสร้างสถิต (Static Structural View) และมุมมองแบบพฤติกรรมพลวัตของระบบ (Dynamic Behavior View) นั้นถูกนำมาใช้ในขั้นตอนต่าง ๆ ดังต่อไปนี้

1) การจำลองความต้องการเชิงซอฟต์แวร์ (Requirement Modeling)

ในระหว่างขั้นตอนการจำลองความต้องการของระบบนั้นแบบจำลองถูกกำหนดในขอบเขตของผู้ใช้ระบบ (Actor) และงาน (Use Case) ที่ถูกกระทำโดยตัวผู้ใช้ระบบนั้น ๆ แผนภาพ Use Case นี้ถูกนำมาใช้อธิบายความต้องการในเชิงซอฟต์แวร์ทั้งหมดของระบบ อีกทั้งเป็น

เครื่องมือที่ใช้ในการกำหนดขอบเขตและการลำดับความสำคัญของความต้องการเป็นอย่างดี ทั้งนี้ทั้งนั้นหากแผนภาพ Use Case นี้ยังไม่มีความชัดเจนเพียงพออาจใช้การจำลองส่วนติดต่อผู้ใช้เพื่อตรวจสอบความต้องการนั้น ๆ อีกทางหนึ่งได้

2) การจำลองการวิเคราะห์ระบบ (Analysis Modeling)

ในขั้นตอนการวิเคราะห์นั้นแบบจำลองสถิตและพลวัต (Static and Dynamic Model) ถูกพัฒนาขึ้นโดยแบบจำลองสถิตนั้นใช้ในการกำหนดความสัมพันธ์ในรูปแบบให้กับโครงสร้างของคลาส (Class Diagram) ส่วนแบบจำลองพลวัตนั้นถูกจำลองจากแผนภาพ Use Case และแสดงความสัมพันธ์เชิงพฤติกรรมระหว่างวัตถุที่อยู่ในกระบวนการวิเคราะห์

3) การจำลองการออกแบบระบบ (Design Modeling)

เมื่อกระบวนการผ่านขั้นตอนการจำลองความต้องการเชิงซอฟต์แวร์และการวิเคราะห์แล้วนั้น สถาปัตยกรรมของซอฟต์แวร์จึงถูกกำหนดขึ้นในแบบจำลองการออกแบบแนวคิดเชิงวัตถุอย่างการซ่อนข้อมูล คลาสและการสืบทอดนั้นถูกนำมาใช้เพื่อออกแบบระบบเป็นระบบย่อย (Subsystem)

เนื่องจากงานวิจัยชิ้นนี้ใช้แนวคิดในเชิงวัตถุในการสร้างกรอบงาน จึงมีความจำเป็นอย่างยิ่งในการใช้ UML ซึ่งสนับสนุนรูปแบบของแนวคิดเชิงวัตถุเช่นกัน โดยการใช้ UML นี้เพื่อแสดงแนวคิดของกรอบงาน การวิเคราะห์และการออกแบบกรอบงาน การจำลองสถาปัตยกรรมของกรอบงานตลอดจนการจำลองขั้นตอนและพฤติกรรมของเอนทิตีต่าง ๆ ในบริบทของกรอบงาน

ดังได้กล่าวพอสังเขปไว้ข้างต้นถึงลักษณะแนวคิดในการจำลองของ UML นั้นประกอบไปด้วยการจำลองในมุมมองของความต้องการหลัก มุมมองแผนภาพสถิตและมุมมองในลักษณะแผนภาพเชิงพลวัตทำให้แผนภาพของ UML จำแนกเป็น 3 ประเภทหลักดังต่อไปนี้

1) แผนภาพเชิงพฤติกรรม (Behavior Diagram)
   - Activity Diagram
   - State Machine Diagram
   - Use Case Diagram
2) แผนภาพเชิงปฏิสัมพันธ์ (Interactive Diagram)
   - Communication Diagram
   - Interaction overview Diagram
   - Sequence Diagram
   - Timing Diagram
3) แผนภาพเชิงโครงสร้าง
   - Class Diagram

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

3.1.2 รูปแบบเชิงซอฟต์แวร์ (Software Pattern)

Froehlich et al. (2000) กล่าวถึงความสำคัญในการอธิบายคุณสมบัติและพฤติกรรมของกรอบงานเชิงวัตถุโดยการใช้รูปแบบเชิงซอฟต์แวร์ซึ่งถูกคิดขึ้นสำหรับวิเคราะห์และออกแบบซอฟต์แวร์เชิงวัตถุ

Gamma et al. (1995) หรือที่รู้จักในชื่อกลุ่ม Gang of Four (GoF) ได้คิดค้นและเขียนเป็นหนังสือเกี่ยวกับการรูปแบบมาตรฐานรูปแบบเชิงซอฟต์แวร์ดังต่อไปนี้

1) Creational Patterns
   - Abstract Factory
   - Builder
   - Factory Method
   - Prototype
   - Singleton

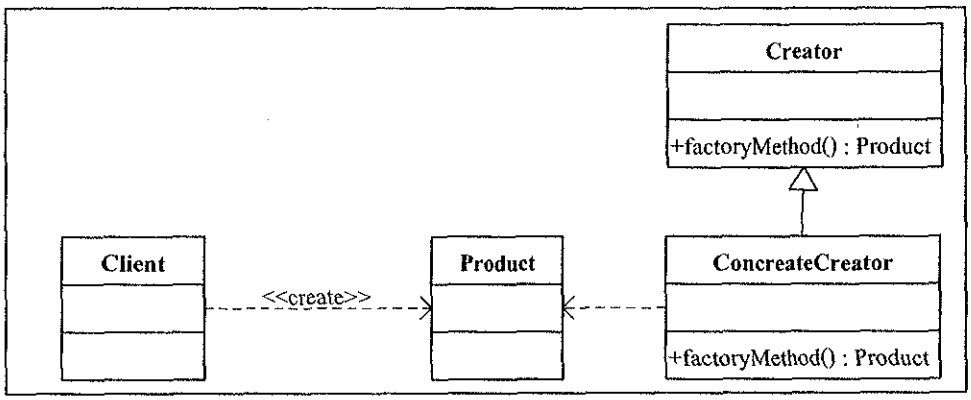
2) Structural Patterns
   - Adapter
   - Bridge
   - Composite
   - Decorator
   - Façade
   - Flyweight
   - Proxy

3) Behavioral Pattern
   - Chain of Responsibility
   - Command
   - Interpreter

- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template method
- Visitor

ในงานวิจัยชิ้นนี้เลือกใช้เฉพาะรูปแบบเชิงซอฟต์แวร์ที่เกี่ยวข้องกับการสร้างกรอบงานเท่านั้น ซึ่งจะนำเสนอดังต่อไปนี้

1) Factory Method Pattern

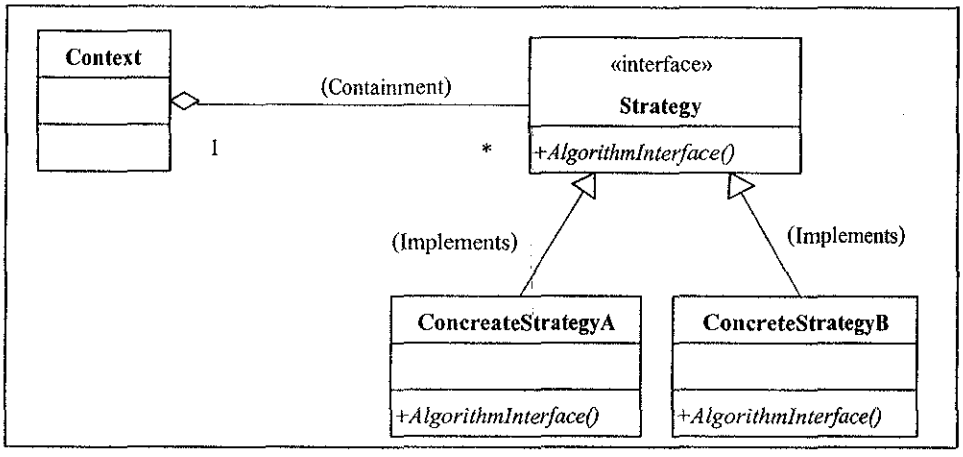


รูปที่ 3.1 แสดง Factory Method Pattern

จากรูปที่ 3.1 เป็นแผนภาพของคลาสที่แสดง Factory Method Pattern ซึ่งใช้ในการแก้ปัญหาในการสร้างวัตถุในบริบทของโปรแกรมโดยมีวัตถุหนึ่งซึ่งเรียกว่า Creator สร้างวัตถุใดวัตถุหนึ่งซึ่งจากแผนภาพถูกเรียกว่า Product ให้กับวัตถุใดวัตถุหนึ่งในที่นี้ ได้แก่ Client

Factory Method Pattern นั้นทำให้เกิดความยืดหยุ่นมากขึ้นและทำให้ความเชื่อมติด (Couple) ลดลงในการใช้ตัว Creator เนื่องจาก Client ไม่ได้เรียกเมธอดจาก Creator ซึ่งเป็นคลาสแม่โดยตรงแต่เรียกจากคลาสลูกและในกลไกการเชื่อมความสัมพันธ์แบบพลวัต (จากรูปที่ 3.1 คือคลาส ConcreateCreator)

2) Strategy Pattern

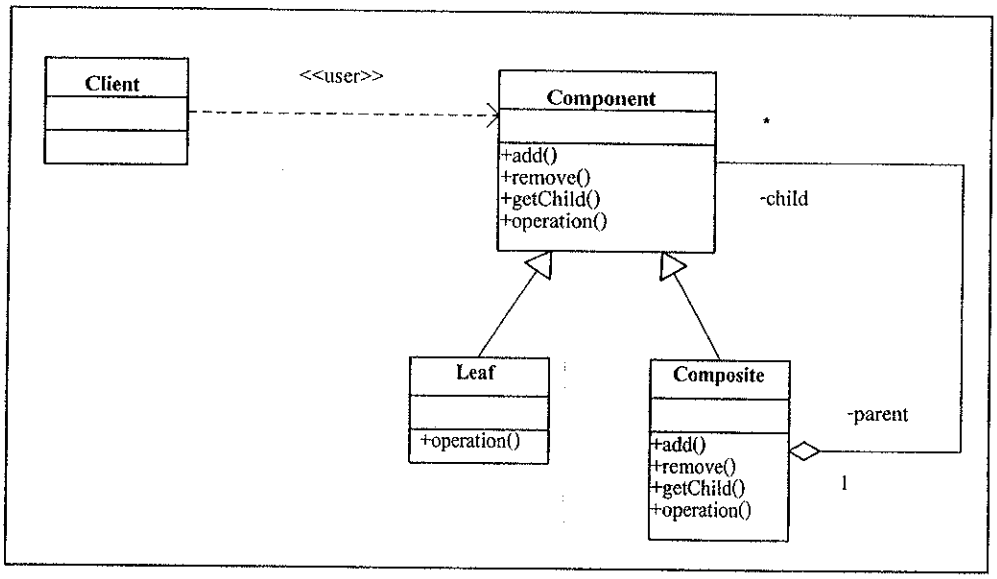


รูปที่ 3.2 แสดง Strategy Pattern

Strategy Pattern ดังแสดงในรูปที่ 3.2 ใช้หลักการของภาวะพหุสัณฐานในการสนับสนุนการทำงานให้วัตถุรูปธรรม (Concrete Object) ซึ่งได้รับการสืบทอดจากคลาสแม่ซึ่งมีความเป็นนามธรรมมากกว่าสามารถถูกเรียกให้ทำงานในขณะโปรแกรมกำลังทำงาน

แบบอย่างนี้เป็นแบบอย่างที่เหมาะสมอย่างยิ่งในสถานการณ์ที่ต้องการปรับเปลี่ยนขั้นตอนวิธี (Algorithm) ของการทำงานอย่างพลวัตในขณะโปรแกรมทำงานทำให้ระบบมีความยืดหยุ่นสูง

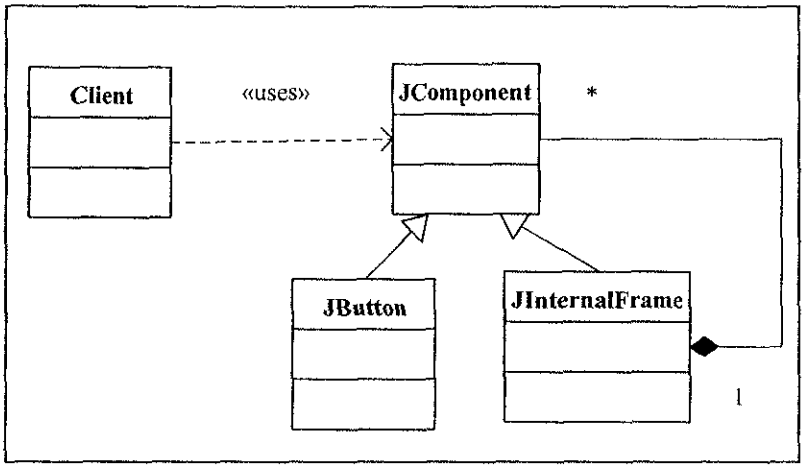
3) Composite Pattern



รูปที่ 3.3 แสดง Composite Pattern

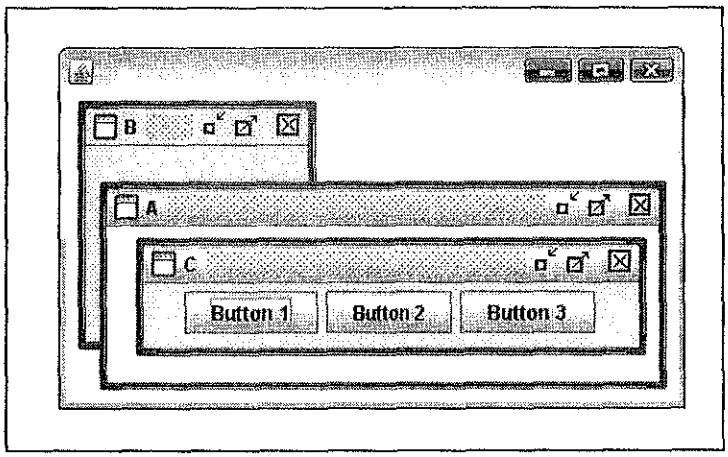
Composite Pattern ดังแสดงในรูปที่ 3.3 นั้นใช้ในการแก้ปัญหาเมื่อพบโครงสร้างของเอนทิตีมีลักษณะเป็นต้นไม้ (Tree Structure Data) หรือเป็นลำดับชั้น (Hierarchy Model) โดยจากรูปนั้นเอนทิตี Client ใช้งานเอนทิตี Component ซึ่งมีองค์ประกอบย่อยที่ซับซ้อน อีกทั้งองค์ประกอบย่อยเหล่านั้นยังมีคุณสมบัติเช่นเดียวกับองค์ประกอบหลัก

ตัวอย่างที่เห็นได้ชัดเจน ได้แก่ การออกแบบส่วนติดต่อผู้ใช้แบบกราฟิก (Graphic User Interface) โดยทั่วไปหรือการออกแบบ Swing Framework



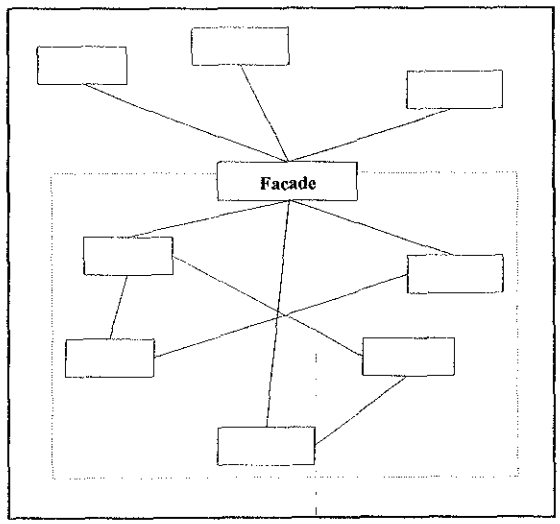
รูปที่ 3.4 แสดงแผนภาพในการออกแบบ Swing Framework

จากรูปที่ 3.4 คลาสแม่นั้นมีคุณสมบัติของพฤติกรรมพื้นฐานเกี่ยวกับการแสดงผลของส่วนที่ใช้ติดต่อกับผู้ใช้ ส่วนคลาสลูกที่สืบทอดนั้นเพิ่มคุณสมบัติเฉพาะเจาะจงเกี่ยวกับการแสดงผลหน้าต่าง (Frame) ส่วนติดต่อกับผู้ใช้ โดยในคลาส JFrame นั้นสามารถบรรจุคลาสลูกที่สืบทอดจาก JComponent อย่าง JButton ได้ด้วย ตัวอย่างดังแสดงในรูปที่ 3.5



รูปที่ 3.5 แสดงตัวอย่างการใช้ Composite Pattern

4) Facade Pattern



รูปที่ 3.6 แสดงแผนภาพของ Facade Pattern

รูปที่ 3.6 นั้นแสดง Facade Pattern ซึ่งเป็นรูปแบบของการออกแบบแผนภาพของคลาสและการเขียนรหัสคำสั่งในภาษาเชิงวัตถุ Facade Pattern นั้นใช้แก้ปัญหาในการเข้าถึงบริการของคลาสต่าง ๆ ที่อยู่ในคลังของส่วนโปรแกรม (Package) ต่างกัน จึงทำการสร้างคลาส ที่ถูกเรียกว่า Facade เพื่อเป็นตัวกลางในการเข้าถึงบริการต่าง ๆ เหล่านั้นทำให้เกิดความสะดวกต่อนักพัฒนาโปรแกรมเนื่องจากสามารถเข้าถึงบริการต่าง ๆ ได้ นอกจากนี้แล้ว Facade Pattern ยังใช้ในแผนภาพที่แสดงแนวคิดหลักในการออกแบบระบบย่อย (Subsystem) อีกด้วย

3.2 การออกแบบระบบ

งานวิจัยนี้มุ่งเน้นในการสร้างกรอบงานเพื่อสนับสนุนคุณสมบัติในกระบวนการพัฒนาซอฟต์แวร์เกี่ยวกับความยืดหยุ่นและความสามารถในการนำกลับมาใช้ใหม่

ในกระบวนการพัฒนากรอบงานนั้นใช้ภาษาจาวาซึ่งเป็นภาษาที่ถูกคิดขึ้นในมีหลักเกณฑ์การเขียนรหัสคำสั่งในเชิงวัตถุและใช้ Unified Modeling Language ในการออกแบบเนื่องจากสนับสนุนแนวคิดในเชิงวัตถุเช่นกัน ส่วนการออกแบบนั้นนำเสนอดังต่อไปนี้

3.2.1 กำหนดความต้องการของเชิงซอฟต์แวร์ของกรอบงาน

ในบทความของ Parsons et al. (1999) กล่าวถึงขั้นตอนกระบวนการในการสร้างกรอบงานประกอบไปด้วย 4 ขั้นตอนอันได้แก่

1) กำหนดความต้องการเชิงซอฟต์แวร์และบทบาทต่าง ๆ ในการพัฒนากรอบงาน
2) พัฒนาแก่นของกรอบงาน
3) พัฒนาส่วนโปรแกรมพื้นฐานให้กับกรอบงาน
4) พัฒนาส่วนโปรแกรมเสริมให้กับกรอบงาน

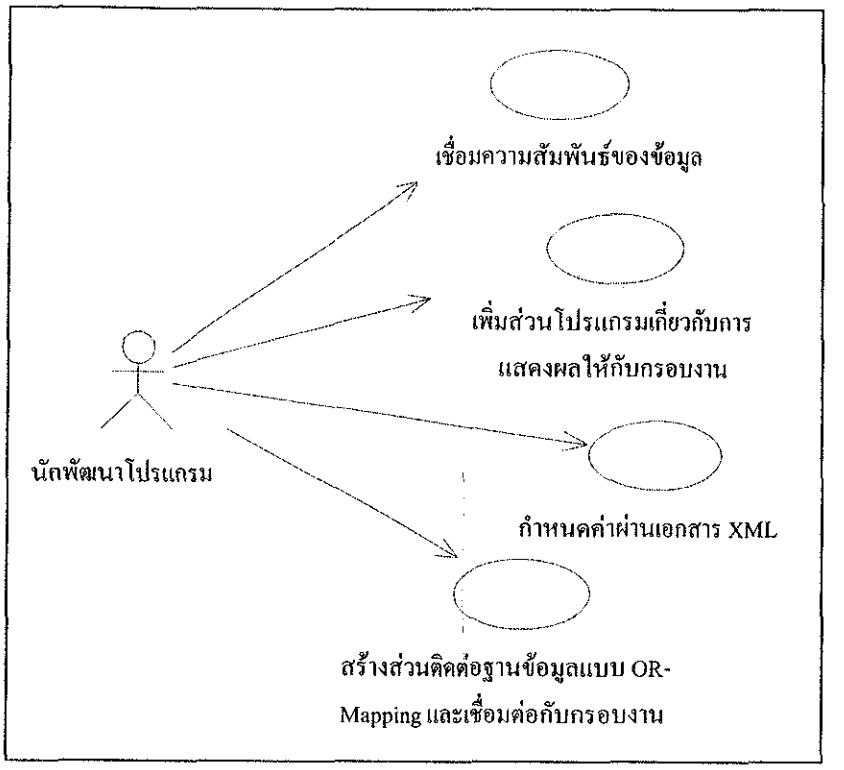
ดังนั้นแล้วในส่วนนี้จึงใช้แนวคิดของดังกล่าวและทำการนำเสนอการกำหนดความต้องการเชิงซอฟต์แวร์ของกรอบงานดังนี้

เชื่อมข้อมูลซึ่งเป็นตัวแทนของเอนติทีในชั้นการแสดงผลสู่ข้อมูลนามธรรมที่อยู่ในบริบทของโปรแกรม

- เชื่อมข้อมูลนามธรรมที่อยู่ในบริบทของโปรแกรมกับข้อมูลในแบบจำลองหรือฐานข้อมูลเชิงสัมพันธ์ (Relational Data Model)
- สร้างกลไกให้กับกรอบงานในการเชื่อมต่อข้อมูลเชิงวัตถุที่อยู่ในส่วนแสดงผล ส่วนควบคุมและส่วนชั้นข้อมูลให้มีค่าที่สัมพันธ์กัน (Synchronization)
- กำหนดส่วนเชื่อมต่อหรือฮุคให้กับกรอบงานสำหรับการรอรับการสร้างส่วนโปรแกรมสำหรับแสดงผล และส่วนโปรแกรมสำหรับเชื่อมต่อฐานข้อมูลโดยใช้แนวคิดของ Object-Relational Mapping ที่จะถูกกำหนดโดยนักพัฒนาโปรแกรมสร้างความยืดหยุ่นและทำให้กรอบงานสามารถขยายความสามารถของโปรแกรมประยุกต์ออกไปได้
- สร้างกลไกของกรอบงานที่รักษาแนวคิด Plain Old Java Object (POJO) เพื่อให้เอนทิตีหรือคลาสที่เขียนโดยภาษาจาวานั้นมีรูปแบบเรียบง่ายและพร้อมในการนำกลับมาใช้ใหม่ในขอบเขต (Domain) อื่น ๆ

ในขั้นตอนถัดไปดังที่ Parsons et al. (1999) ได้กล่าวไว้นั้นคือส่วนของการพัฒนากรอบงานตั้งแต่แก่นของกรอบงานและส่วนโปรแกรมอื่น ๆ หากแต่กระบวนการเหล่านั้นจะเกิดขึ้นได้ต่อเมื่อได้รับการออกแบบแล้ว ดังนั้นจึงนำเสนอถึงส่วนของการออกแบบกรอบงานต่อไป

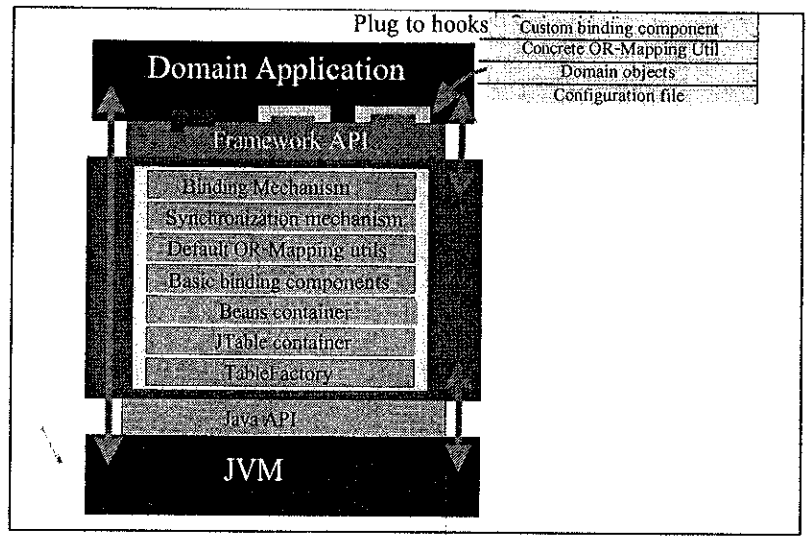
นอกจากนี้หากพิจารณาโดยหลักการออกแบบเชิงวัตถุโดยใช้ Unified Modeling Language นั้นสามารถจำลองแผนภาพเชิงความต้องการของซอฟต์แวร์ด้วยแผนภาพ Use Case ในมุมมองของนักพัฒนาโปรแกรมที่ใช้กรอบงานได้ดังต่อไปนี้



รูปที่ 3.7 แสดงแผนภาพ Use Case ของกรอบงาน

จากรูปที่ 3.7 แสดงแผนภาพ Use Case ของกรอบงานในมุมมองของนักพัฒนาโปรแกรมซึ่งมีบทบาทเป็น Actor ของแผนภาพนี้ โดยแผนภาพ Use Case นั้นใช้อธิบายถึงความสัมพันธ์ระหว่างผู้ที่กระทำต่อซอฟต์แวร์ (Actor) และกิจกรรม (Use Case) ต่าง ๆ ซึ่งเกิดขึ้นกับซอฟต์แวร์ที่สนใจ ใช้อธิบายเพียงว่าใครทำอะไรกับซอฟต์แวร์แต่ไม่อธิบายถึงขั้นตอนของการทำงานในแต่ละกิจกรรม ดังแสดงในรูปที่ 3.7 นั้นแสดงถึงความสามารถของกรอบงานที่ให้ประโยชน์ต่อนักพัฒนาโปรแกรมเฉพาะ

3.2.2 สถาปัตยกรรมของกรอบงาน



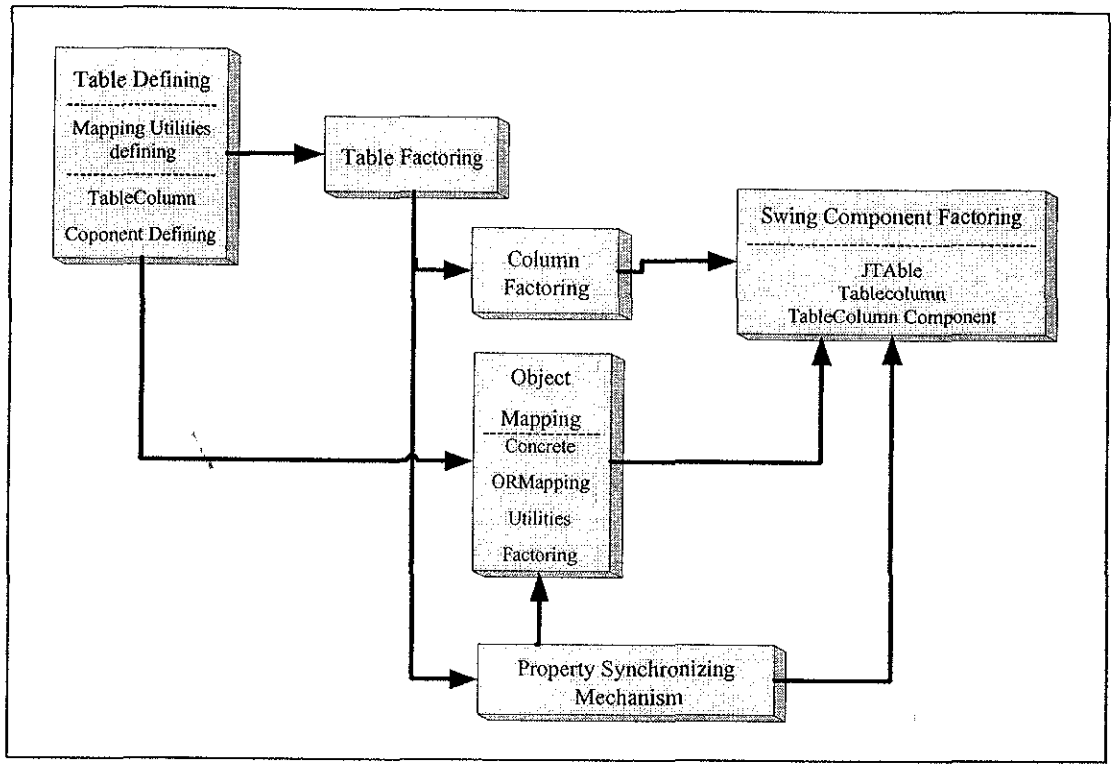
รูปที่ 3.8 แผนภาพแสดงสถาปัตยกรรมของกรอบงาน

จากรูปที่ 3.8 แสดงสถาปัตยกรรมของกรอบงานซึ่งถูกออกแบบให้ทำงานภายใต้ Java Application Interface (Java API) จากรูปนั้นแสดงให้เห็นว่าโปรแกรมประยุกต์ที่พัฒนาภายใต้กรอบงานสามารถใช้บริการของกรอบงานได้ อีกทั้งยังคงใช้บริการของ Java Application Interface (Java API) ได้เช่นกัน

ในส่วนของกรอบงานเองนั้นได้กำหนดให้ประกอบไปด้วยเครื่องมือ กลไกล และส่วนโปรแกรมต่าง ๆ ดังต่อไปนี้

1) กลไกการเชื่อมข้อมูลเชิงวัตถุกับส่วนแสดงผล (Binding Mechanism)
2) กลไกการเชื่อมคุณสมบัติของวัตถุให้ตรงกัน (Synchronization Mechanism)
3) ส่วนโปรแกรมพื้นฐานที่ใช้สำหรับเชื่อมต่อฐานข้อมูลเชิงสัมพันธ์โดยใช้แนวคิด OR-Mapping (Default OR-Mapping Utilities)
4) ส่วนโปรแกรมพื้นฐานต่าง ๆ ที่สร้างกรอบงานเตรียมไว้สำหรับนักพัฒนาโปรแกรม (Basic Binding Components)
5) ส่วน Container ที่ใช้เก็บ Instances ของวัตถุที่อยู่ในบริบทของโปรแกรม (Beans Container)
6) ส่วน Container ที่ใช้เก็บตารางแสดงผล (JTable Container)
7) ส่วนโปรแกรมที่มีหน้าที่สร้างตารางแสดงผล (Table Factory)

3.2.3 การออกแบบกลไกการทำงานของกรอบงาน



รูปที่ 3.9 แสดงกลไกการทำงานของกรอบงาน

กลไกการทำงานของกรอบงานนั้นถูกออกแบบให้มีขั้นตอนกระบวนการดังต่อไปนี้

1) ขั้นตอนการกำหนดคุณสมบัติของเครื่องมือและคุณลักษณะของตารางข้อมูล (Table Defining) ในขั้นตอนนี้นักพัฒนาโปรแกรมประยุกต์ต้องกำหนดค่าตัวแปรเสริมและส่วนโปรแกรมอื่น ๆ ที่จำเป็นต่อกรอบงานดังต่อไปนี้

- คุณลักษณะของตารางข้อมูลต่าง ๆ นั้นถูกกำหนดผ่านเอกสาร XML โดยนักพัฒนาโปรแกรมต้องกำหนดตามนิยามของเอกสารชื่อ mapping.xsd ดังแสดงในรูปที่ 3.10
- สร้างส่วนโปรแกรมที่ใช้แสดงผลในคอลัมน์ของตารางข้อมูลและเชื่อมส่วนโปรแกรมนั้นเข้าสู่กรอบงาน
- สร้างส่วนโปรแกรมที่ใช้ในการเชื่อมต่อฐานข้อมูลด้วยเทคโนโลยี Object-Relational Mapping

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/xmlschema"
                elementFormDefault="qualified"
                xmlns="urn:nonstandard:mapping"
                targetNamespace="urn:nonstandard:mapping">

    <xsd:element name="table-mapping" type="Table-Mapping" />
    <xsd:complexType name="Table-Mapping" />
          <xsd:sequence >
               <xsd:element name="table" type="Table" minoccurs="1" maxoccurs="unbounded"/>
          <xsd:sequence >
    </xsd:complexType >

    <xsd:complexType  name="Table">

          <xsd:sequence >
               <xsd:element name="property" type="Property" minoccurs="1" maxoccurs="unbounded"/>
          </xsd:sequence >
          <xsd:attribute name="class" type="xsd:string" use="required" />
          <xsd:attribute name="name" type="xsd:string" use="required" />
          <xsd:attribute name="delible" type="xsd:boolean" use="required" />
    </xsd: complexType >

    <xsd:complexType  name="Property">
          <xsd:attribute name="name" type="xsd:string" use="required" />
          <xsd:attribute name="col-name" type="xsd:string" use="required" />
          <xsd:attribute name="type" type="xsd: string" use="required" />
          <xsd:attribute name="bind" type="xsd: string" />

    </xsd: complexType >
</xsd:schema>
```

รูปที่ 3.10 แสดงเอกสาร XSD ซึ่งเป็นเอกสารที่ใช้นิยามโครงสร้างของเอกสาร XML

รูปที่ 3.10 เอกสาร mapping.xsd เป็นเอกสารชนิด XML Schema Definition (XSD) ซึ่งใช้นิยามโครงสร้างของเอกสาร XML ซึ่งเป็นเอกสารที่ใช้กำหนดคุณลักษณะของตารางข้อมูล โดยนักพัฒนาโปรแกรมต้องกำหนดเอกสาร XML ที่มีนามสกุลเป็น .db.xml และกำหนดให้กรอบงานอ่านเอกสารเหล่านั้น ตัวอย่างดังแสดงในรูปที่ 3.11

```
<?xml version="1.0" encoding="tis620" ?>
  <table-mapping xmlns="urn:nonstandard:mapping"
       xmlns:xsi="http://www.w3.org/2001/xmlschema-instance"
       xsi: schemaLocation="urn:nonstandard:mapping
       file:./mapping.xsd">

      <table class="spccificdomain.entity.Advisor" name="advisor" delible="false">
              <property name="name" col-name="name" type="string"   />
              <property name="${major.name}" col-name="major" type="string"   />
          <property name="quantification" col-name="quantification" type="string"   />
          <property name="salary" col-name="salary" type="Double"   />
      </table>

      <table class="spccificdomain.entity.Register" name="register" delible="true">
              <property name="id" col-name="id" type="String"   />
              <property name="${student.fname}" col-name="name" type="string"   />
              <property name="${subject.name}" col-name="course" type="string"   />
              <property name="score" col-name="score" type="Double" bind="grade"   />
      </table>
</table-mapping>
```

รูปที่ 3.11 แสดงตัวอย่างการกำหนดคุณลักษณะของเอกสาร XML

2) ขั้นตอนการสร้างตารางข้อมูล (Table Factoring)

ในขั้นตอนที่ 2 นี้เป็นกระบวนการที่เกิดขึ้นโดยกรอบงานในช่วงดำเนินการของรหัสคำสั่ง (Run time) ประกอบด้วยขั้นตอนย่อย ๆ ดังต่อไปนี้

- การสร้างสดมภ์ของตารางข้อมูล (Column Factoring)
- การจับคู่เพื่อเชื่อมคุณสมบัติระหว่างสดมภ์ของตารางข้อมูลกับคุณสมบัติของวัตถุที่อยู่บริบทของโปรแกรมประยุกต์ (Object Mapping)
- การสร้างกลไกลการเชื่อมโยงข้อมูลในชั้นการแสดงผล Instances ของวัตถุที่อยู่ในบริบทของโปรแกรมประยุกต์และเอนทีตีฐานข้อมูลเชิงสัมพันธ์

3) ขั้นตอนการสร้างตารางข้อมูล

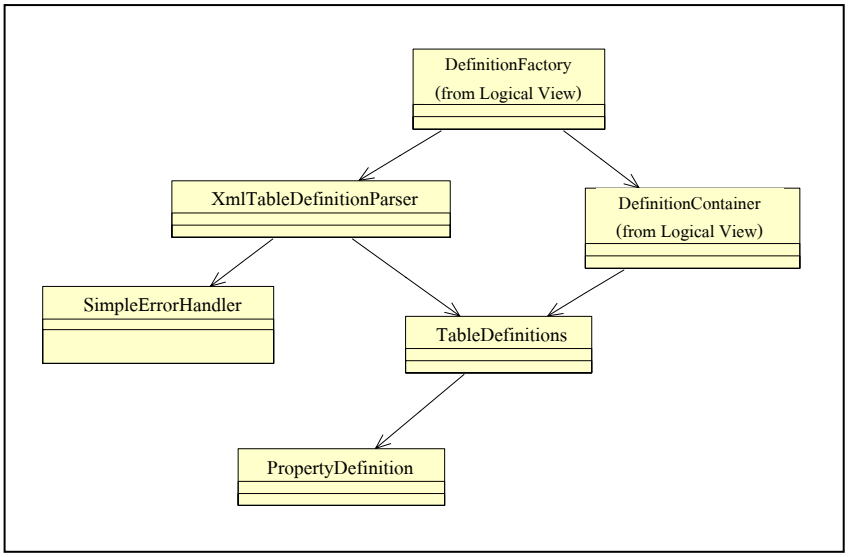
หลังจากได้ผ่านกระบวนการในข้อ 2 แล้ว กรอบงานเริ่มทำการสร้างตารางเพื่อแสดงผลของข้อมูล โดยใช้ส่วนของโปรแกรมแสดงผลและกลไกต่าง ๆ ที่ได้ถูกสร้างไว้แล้วในขั้นตอนที่ 2

3.2.4 การออกแบบส่วนต่าง ๆ ของกรอบงาน

1) ส่วนโปรแกรมสำหรับอ่านค่าที่กำหนดโดยเอกสาร XML

ส่วนนี้มีหน้าที่สำหรับอ่านค่าที่นักพัฒนาโปรแกรมได้กำหนดในเอกสาร XML เพื่อกำหนดคุณสมบัติต่าง ๆ ของตารางข้อมูล และการเชื่อมกรอบงานกับส่วนโปรแกรมที่นักพัฒนาโปรแกรมนั้นกำหนดขึ้นเอง

- แผนภาพเชิงคลาสของระบบย่อย



รูปที่ 3.12 แสดงแผนภาพเชิงคลาสของส่วนอ่านเอกสาร XML

รูปที่ 3.12 แสดงแผนภาพเชิงคลาสของส่วนอ่านค่าเอกสาร XML ที่ใช้สำหรับกำหนดค่าตัวแปรเสริมต่าง ๆ ที่จำเป็นต่อกรอบงาน

- คลาสและหน้าที่รับผิดชอบของแต่ละคลาส

คลาส DefinitionFactory เป็นคลาสซึ่งทำหน้าที่เป็น Boundary Class ของระบบกล่าวคือนักพัฒนาโปรแกรมเข้าถึงส่วนการอ่านนิยามของตารางข้อมูลเอกสาร XML ผ่านคลาสนี้เพียงคลาสเดียว และมีหน้าที่ออกคำสั่งให้ Instance ของคลาสอื่น ๆ ทำงาน

คลาส XmlDefinitionParser เป็นคลาสทำหน้าที่เป็นตัวดำเนินการอ่านค่าเอกสาร XML ในแต่ละโหนดทั้งเอกสาร และทำการส่งโหนดที่มีนิยาม <table> ให้กับ Instance ของคลาส TableDefinitions

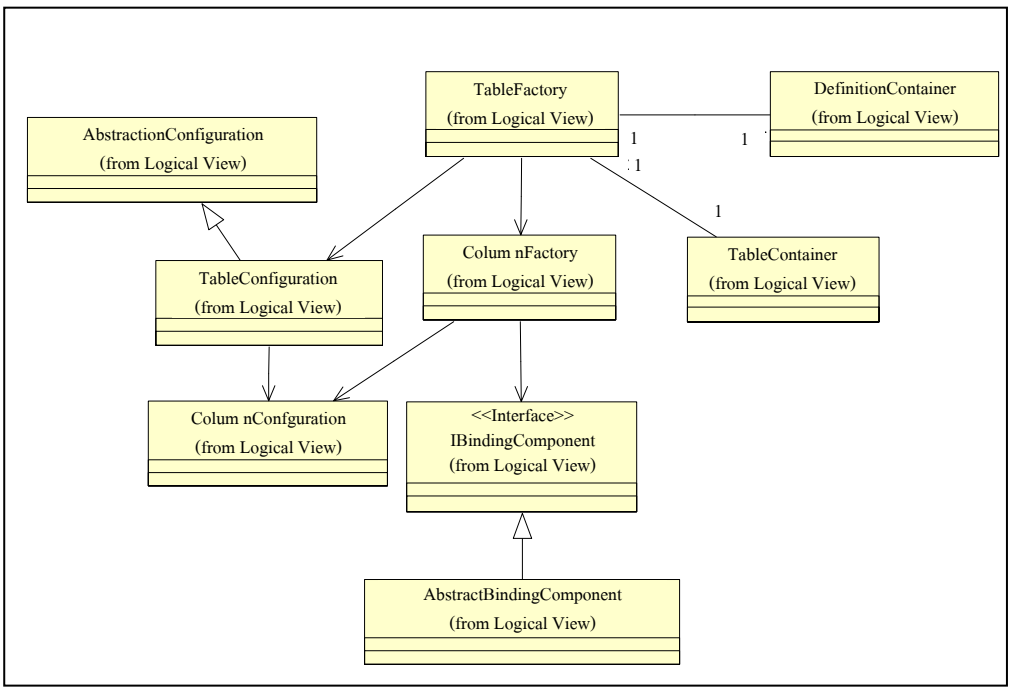
คลาส DefinitionContainer เป็นคลาสซึ่งทำหน้าที่เป็น Container มีหน้าที่เก็บรายการ (List) ของ Iinstance ของคลาส TableDefinitions

คลาส TableDefinitions เป็นคลาสที่ใช้เก็บรายการของ Instance ของคลาส PropertyDefinition และเก็บคุณสมบัติของตารางข้อมูลอย่างชื่อตารางข้อมูล, คลาสที่ตารางข้อมูล ทำการผูกความสัมพันธ์ และการกำหนดให้ตารางข้อมูลสามารถลบข้อมูลได้

คลาส ProperyDefinitions เป็นคลาสที่ใช้คุณสมบัติของแต่ละสดมภ์ของตารางข้อมูลอันได้แก่ชื่อสดมภ์ ชนิดของข้อมูลที่แสดงผล ชนิดของส่วนโปรแกรมที่ใช้ผูกความสัมพันธ์ของข้อมูล (Binding Component) และคุณสมบัติ (Property) ของตารางข้อมูลที่จะถูกโยงเข้ากับคลาส

2) ส่วนสร้างตารางข้อมูล

- แผนภาพเชิงคลาสของระบบย่อย



รูปที่ 3.13 แสดงแผนภาพเชิงคลาสของส่วนสร้างตารางข้อมูล

- คลาสและหน้าที่รับผิดชอบของแต่ละคลาส

คลาส TableFactory เป็นคลาสที่ทำหน้าที่เป็น Boundary Class ของระบบย่อยนี้มีเมธอดที่มีชื่อ createTable(name:String):JTable ทำหน้าที่เป็น Boundary Method ในการสร้างตารางข้อมูล

คลาส ColumnFactory เป็นคลาสที่ทำหน้าที่สร้างสดมภ์ของตารางข้อมูล

คลาส TableConfiguration เป็นคลาสที่อ่านนิยามจาก Instance ของคลาส TableDefinitions ของตารางเป้าหมายและกำหนดค่าส่วนโปรแกรมที่จำเป็นต่าง ๆ ให้กับระบบย่อย

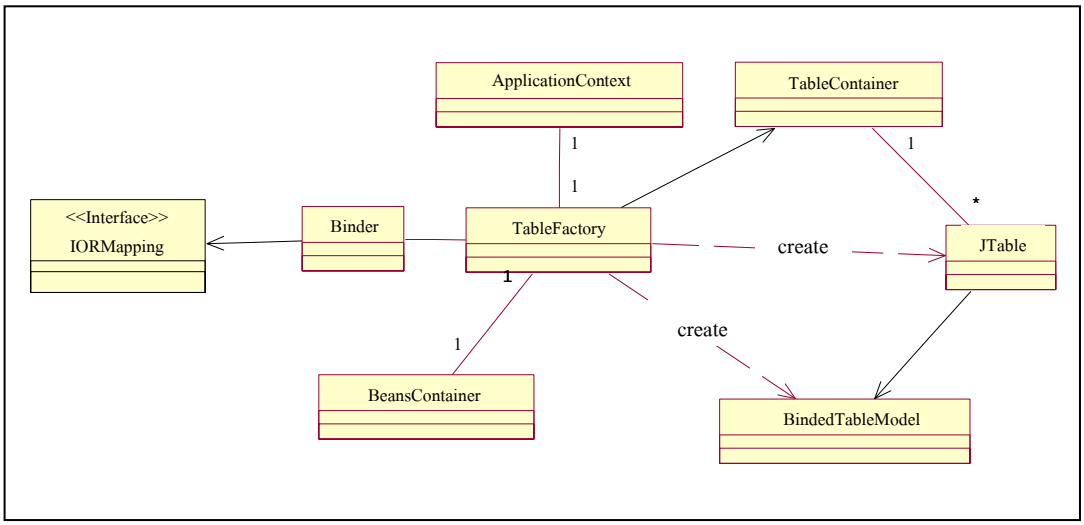
คลาส ColumnConfiguration เป็นคลาสที่อ่านนิยามจาก Instance ของคลาส PropertyDefinitions ของตารางเป้าหมายและกำหนดค่าส่วนโปรแกรมที่จำเป็นต่าง ๆ ให้กับระบบย่อย

คลาส AbstractConfiguration เป็นคลาสที่ทำหน้าที่เป็นแม่แบบในการแสดงพฤติกรรมของการเป็นคลาสประเภทกำหนดตัวแปรเสริม (Configuration)

คลาส TableContainer เป็นคลาสสำหรับเก็บบัญชีของ Instance ของคลาส JTable

3) ส่วนผูกความสัมพันธ์ของตารางข้อมูลกับ Instance ของคลาสที่อยู่ในบริบทของโปรแกรมประยุกต์

- แผนภาพเชิงคลาสของระบบย่อย



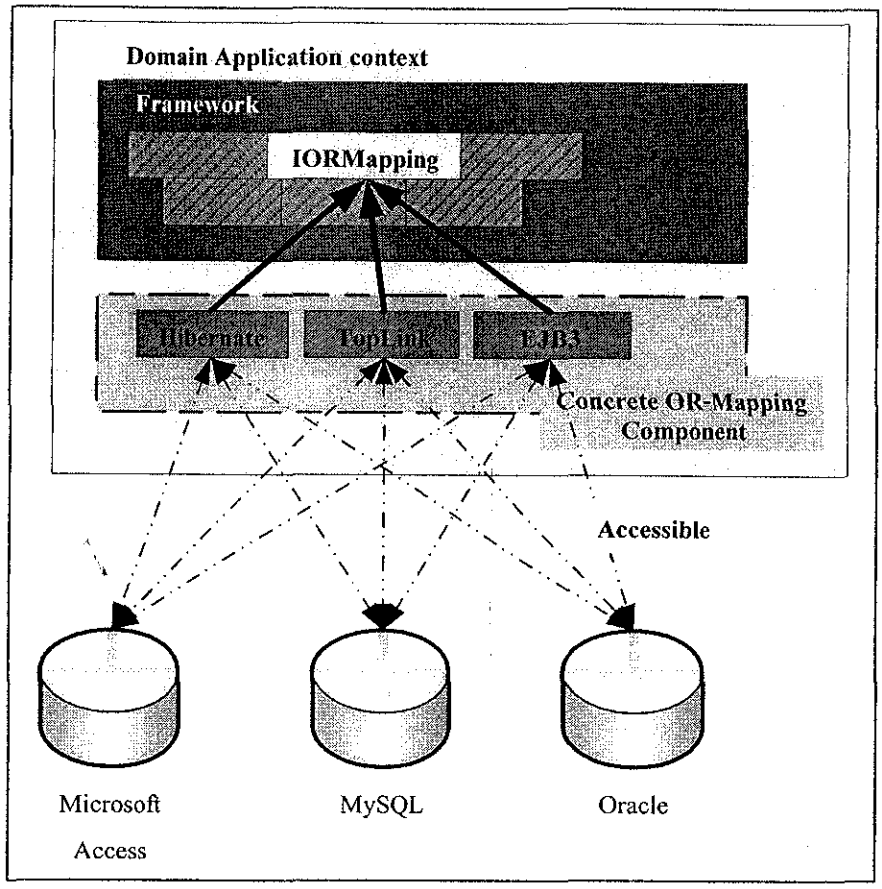
รูปที่ 3.14 แสดงแผนภาพเชิงคลาสของส่วนผูกความสัมพันธ์

- คลาสและหน้าที่ความรับผิดชอบของแต่ละคลาส

คลาส Binder ทำหน้าที่ในการผูกความสัมพันธ์ระหว่างคุณสมบัติต่าง ๆ ของแต่ละ Instance ของคลาสเข้ากับสดมภ์ของตารางข้อมูล อีกทั้งยังเพิ่มตัวเฝ้ามอง (Listener) สถานะของ Instance ในชั้นการแสดงผล (Presentation Layer) ชั้น Business Logic และชั้นแบบจำลองข้อมูล (Data model) และท้ายสุดหากมีการเปลี่ยนแปลงสถานะของ Instance ใด ๆ ในบริบทของโปรแกรมประยุกต์นั้น instance ของคลาส Binder นี้จะทำการปรับให้ค่าของ Instance ของ Domain Object นั้นมีค่าตรงกัน (Synchronization)

คลาส BeansContainer ถูกออกแบบให้มีหน้าที่เป็นตัวบรรจุ Instance ของคลาสต่าง ๆ ที่อยู่ในบริบทของโปรแกรมประยุกต์

คลาส IORMapping เป็นอินเตอร์เฟสซึ่งสนับสนุนการออกแบบกรอบงานให้มีความยืดหยุ่นสำหรับนักพัฒนาโปรแกรมประยุกต์ หรือส่วนของโปรแกรมที่จะนำเข้ามาเชื่อมต่อ เพื่อเปลี่ยนพฤติกรรมหรือเพิ่มประสิทธิภาพของกรอบงาน ในการกระทำการตามเป้าหมายดังกล่าวนั้น จำเป็นต้องทำการ Implement เพื่อสร้างคลาสลูกของอินเตอร์เฟสนี้ โดย Concrete Class ต่าง ๆ ที่ ณ Implement นั้น ทำหน้าที่ในการเป็นตัวกลางในการติดต่อฐานข้อมูลเชิงสัมพันธ์ ซึ่งการติดต่อดังกล่าวนี้ใช้แนวคิดของ Object-Relational Mapping ดังนั้นแล้วกรอบงานนี้จึงมีความยืดหยุ่นสูงและพร้อมรอบรับการเชื่อมโยงฐานข้อมูลที่แตกต่างกัน หรือแม้กระทั่งส่วนของ OR-Mapping ที่ถูกพัฒนาขึ้นอย่างแตกต่างกัน อาทิเช่น Hibernate3, TopLink, EJB3 และอื่นๆ ดังรูปที่ 3.15

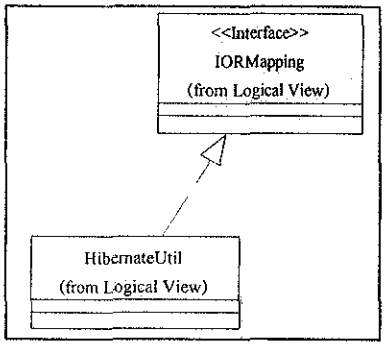


รูปที่ 3.15 แสดงการสร้าง Concrete OR-Mapping Component

ดังแสดงในรูปที่ 3.15 นั้นเป็นการบ่งชี้อย่างชัดเจนถึงความสามารถในการรองรับส่วนโปรแกรมที่อาจถูกพัฒนาขึ้นโดยนักพัฒนาส่วนโปรแกรมย่อยในอนาคต (Unforeseen Component) หรืออาจกล่าวได้อีกในมุมมองหนึ่งตามแนวคิดของการสร้างกรอบงานนั้น IOMapping ทำหน้าที่เป็นเสมือนสุดของกรอบงานรองรับการเชื่อมต่อของส่วนโปรแกรมจากพัฒนา โดยสุดทำให้กรอบงานมีความยืดหยุ่นอย่างยิ่งยวด

4) ส่วนเชื่อมโยงฐานข้อมูลเชิงสัมพันธ์

- แผนภาพเชิงคลาสของระบบย่อย



รูปที่ 3.16 แสดงแผนภาพเชิงคลาสของส่วนเชื่อมโยงฐานข้อมูล

- คลาสและหน้าที่รับผิดชอบของแต่ละคลาส

คลาส HibernateUtil นี้เป็น Concrete Class ที่มีอยู่แล้วในกรอบงานซึ่งเป็นการ Implement มาจาก IORMapping โดยเป็นคลาสที่ทำหน้าที่เป็นตัวกลางในการทำ Object-Relational Mapping ระหว่าง Instance ที่มีความสัมพันธ์เชิงวัตถุกับเอนทีตีในฐานข้อมูลเชิงสัมพันธ์ โดยคลาสนี้ได้นำส่วนโปรแกรมชื่อ Hibernate3 ซึ่งเป็นส่วนโปรแกรมที่ถูกพัฒนาขึ้นเพื่อทำงานด้านนี้โดยเฉพาะ

การทำงานของ HibernateUtil นั้นจำเป็นต้องสร้างตัวกำหนดตัวแปรเสริม (Configuration Component) ซึ่งอาจอยู่ในรูปแบบ 2 รูปแบบดังนี้

- Class ที่จะถูกสร้างเป็น Instance และเชื่อมต่อมายัง Instance ของ HibernateUtil
- ไฟล์เอกสารที่อยู่ในรูปแบบ XML File โดยมีตัวอย่างดังนี้

```
<hibernate-configuration>
<session-factory >
        <property name="hibernate.connection.url">jdbc:mysql://localhost/ooweb</property>
        <property name="hibernate.connection.driver_class">org.gjt.mm.mysql.Driver
        </property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">root</property>
        <!-- property name="hibernate.connection.pool_size"></property -->
        <!-- dialect for MySQL -->
        <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
        <mapping resource="Student.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

จากตัวอย่างข้างต้นนั้นเป็นการกำหนดค่าผ่านเอกสาร XML เป็นการกำหนดที่อยู่ของฐานข้อมูล และค่าที่จำเป็นอื่นอย่าง Driver ของตัวติดต่อฐานข้อมูล ชื่อผู้ใช้และรหัสผ่านจากตัวอย่างนั้นเป็นการกำหนดให้เชื่อมต่อกับฐานข้อมูล MySQL นอกจากนี้แล้วยังสามารถปรับเปลี่ยนชนิดของฐานข้อมูลที่ต้องการเชื่อมต่อได้โดยการเปลี่ยนแปลงค่าตัวแปรเสริมในเอกสาร XML ใหม่

อนึ่งคลาส HibernateUtil นี้ได้ใช้รูปแบบเชิงซอฟต์แวร์ชื่อ Delegate Pattern โดยการออกแบบให้ HibernateUtil เป็นตัวกลาง (Delegator) ระหว่างกรอบงานกับ Hibernate3 โดยทำการส่งต่อการทำงานของเมธอดที่ต่าง ๆ ที่จำเป็นจากการ Implement อินเตอร์เฟส IORMaping อันได้แก่

- update(Object):void;
- insert(Object):void;
- delete(Object):void;
- findAll(Object):List;
- findAll(Class):List;

ต่อไปนี้เป็นการแสดงตัวอย่างการส่งต่อการทำงานของเมธอด findAll (Class):List ไปสู่ Hibernate3

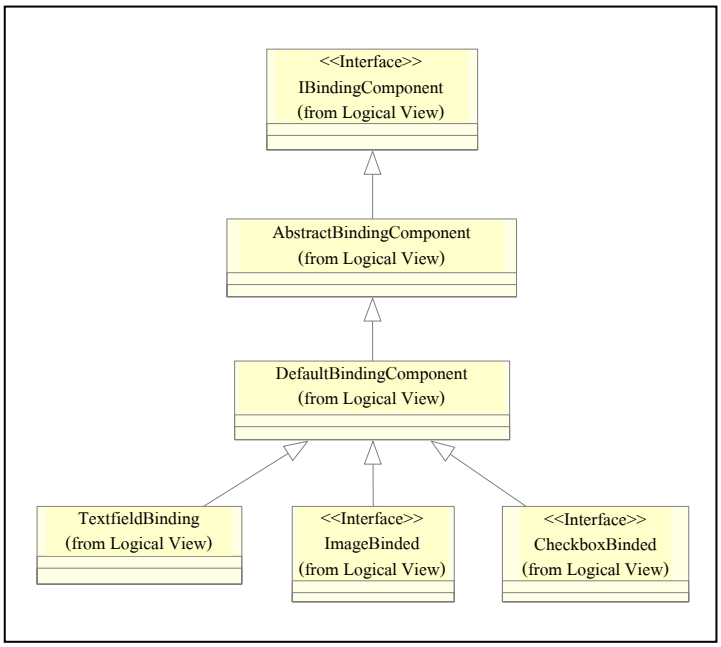
```
public List findAll(Class clazz) {
        Session session = HibernateUtil.getSession();
        session.beginTransaction();
        List list = session.createCriteria(clazz).list();
        return list;
}
```

และนอกจากทำ Delegation แล้วนั้นคลาส HibernateUtil ยังจำเป็นต้องสร้างสภาพแวดล้อมเริ่มต้นสำหรับการทำงานของ Hibernate อีกด้วย

5) ส่วนโปรแกรมในการสร้าง Binding Component

การออกแบบระบบย่อยนี้เพื่อรองรับการสร้างส่วนโปรแกรมที่ใช้ในการแสดงผลบนสดมภ์ของตารางข้อมูลและเชื่อมต่อ (Plug-in) เข้ากับฮุคของกรอบงาน

- แผนภาพเชิงคลาสของระบบย่อย



รูปที่ 3.17 แสดงแผนภาพเชิงคลาสของส่วนสร้าง Binding Component

- คลาสและหน้าที่รับผิดชอบของแต่ละคลาส

อินเตอร์เฟส IBindingComponent ทำหน้าที่เป็นอินเตอร์เฟสของระบบย่อยนี้ กำหนดเมธอดสำคัญต่าง ๆ สำหรับการสืบทอด 2 เมธอดได้แก่

1) getTableCellRenderer():TableCellRenderer

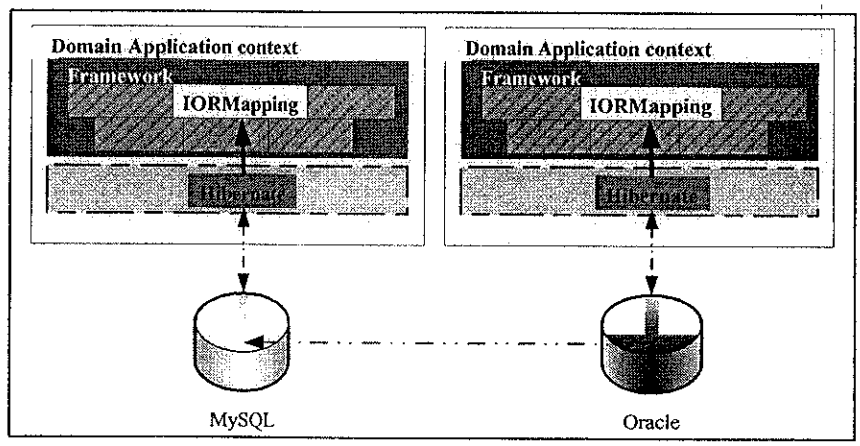
2) getTableCellEditor():TableCellEditor

คลาส AbstractBindingComponent เป็นคลาสที่ทำหน้าที่กำหนดสภาพแวดล้อมต่าง ๆ ที่จำเป็นให้กับระบบ

คลาส DefaultBindingComponent เป็น Concrete Class พื้นฐานที่ใช้เป็นส่วนแสดงผลได้จริงในรูปแบบของ TextField

3.2.5 ส่วนการเชื่อมต่อฐานข้อมูลโดยใช้กลวิธี Object-Rational Mapping

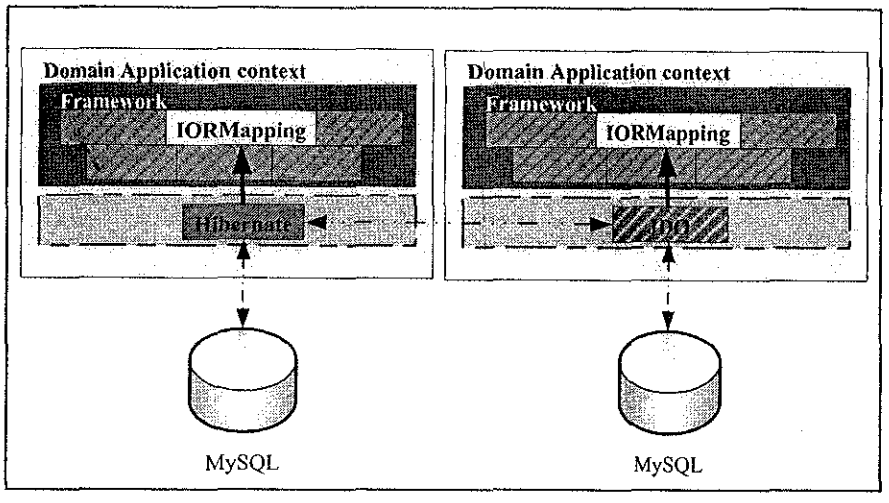
แนวคิดเกี่ยวกับ Object-Relational Mapping หรือมักเรียกในชื่อว่า OR-Mapping (Wikipedia, www, 2008a)นั้นเป็นแนวคิดหนึ่งที่ถูกนำมาใช้อย่างแพร่หลายเพื่อช่วยแก้ไขปัญหาของ Data Access Object Pattern ดังกล่าว ส่วนโปรแกรมหรือโมดูลที่ถูกพัฒนาขึ้นเพื่อสนับสนุนแนวคิดของ Object-Relational Mapping นั้นมักถูกออกแบบให้สามารถทำงานได้โดยไม่ขึ้นตรงต่อตัวจัดการฐานข้อมูลและไม่มีขีดจำกัดของแบบจำลองเอนทิตี อีกทั้งยังรับภาระในเรื่องการรักษาคุณสมบัติเชิงวัตถุในการเปลี่ยนแปลง Instance ของเอนทิตีคลาสเข้าสู่ฐานข้อมูลเชิงสัมพันธ์ ทำให้นักพัฒนาโปรแกรมประยุกต์มีความสะดวกมากยิ่งขึ้นในการพัฒนาโปรแกรมหนึ่งโปรแกรมใดขึ้นมาเพื่อให้ตรงตามความต้องการของลูกค้าอีกทั้งยังรองรับความต้องการที่อาจเปลี่ยนแปลงไปอีกด้วย



รูปที่ 3.18 แสดงการปรับเปลี่ยน DBMS จากการใช้ส่วนโปรแกรมของ OR-Mapping

รูปที่ 3.18 แสดงตัวอย่างการใช้ส่วนโปรแกรม OR-Mapping ที่ชื่อ Hibernate3 ซึ่งถูกออกแบบให้ทำงานได้โดยไม่ขึ้นตรงต่อตัวจัดการฐานข้อมูล นักพัฒนาโปรกรมจึงเกิดความสะดวกมากยิ่งขึ้นในการพัฒนาโปรแกรมประยุกต์ต่าง ๆ ดังได้กล่าวไว้แล้วข้างต้น

ในงานวิจัยนี้นั้นได้ออกแบบให้กรอบงานมีสุดซึ่งรองรับการนำส่วนโปรแกรม OR-Mapping ต่าง ๆ แบบพลวัตหรือกล่าวคือเป็น Pluggable Component เพื่อรองรับการเปลี่ยนแปลงไปของความต้องการเชิงซอฟต์แวร์ที่มักเกิดขึ้นเสมอในอนาคตดังแสดงในรูปที่ 3.19



รูปที่ 3.19 แสดงการปรับเปลี่ยนการใช้ OR-Mapping component

รูปที่ 3.19 นั้นแสดงถึงความสามารถของกรอบงานที่สนับสนุนการเปลี่ยนแปลง OR-Mapping Component แบบพลวัต ตัวอย่างดังรูปนั้นเห็นได้ว่า Hibarnate3 (กรอบเส้นประด้านซ้าย) เป็นส่วนประกอบอย่างหนึ่งซึ่งสามารถถูกเปลี่ยนเป็น Java Data Object (กรอบเส้นประด้านซ้าย) ซึ่งเป็น Component ที่ถูกพัฒนาโดย Third-Party โดยพลวัต

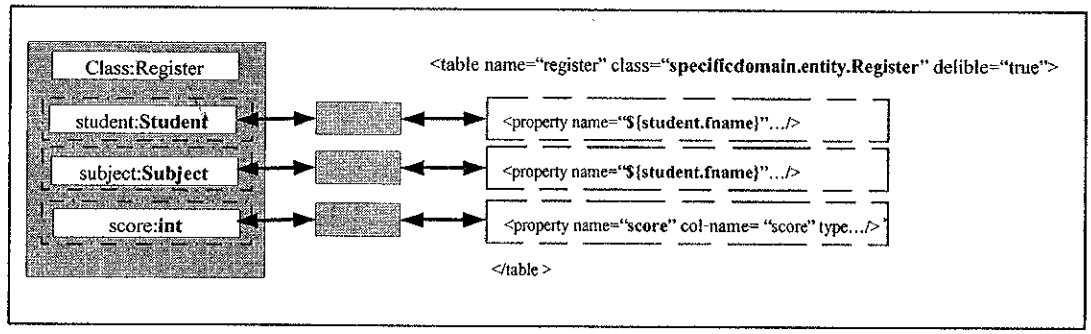
3.2.6 การเชื่อมโยงตัวแทนของวัตถุที่อยู่ในบริบทของโปรแกรมต่าง ๆ

ในการพัฒนาโปรแกรมประยุกต์ภายใต้แนวคิดของหลักการเชิงวัตถุนั้นความต้องการเชิงซอฟต์แวร์ของระบบจะได้รับการวิเคราะห์และออกแบบให้อยู่ในรูปของแผนภาพเชิงคลาส (Class diagram) เพื่อแสดงแบบจำลองเอนทีตี จากนั้นเอนทีตีต่าง ๆ จะถูกจำลองเป็นตัวแทนอยู่ในชั้นต่าง ๆ ตามหลักการแบ่งแยกความเกี่ยวพัน (Separation of Concern)

ในชั้นการแสดงผลเอนทิตีใด ๆ อาจถูกจำลองด้วยตัวแปรประเภทแถว (Array) สำหรับการแสดงผลในตารางข้อมูลหรือคุณสมบัติบางประการของเอนทิตีอาจถูกจำลองด้วย Text Field และถูกเชื่อมโยงเข้ากับ Instance ของเอนทิตีคลาสนั้น ๆ

ในแบบจำลองเชิงข้อมูล (Data Model) นั้น Instance ของเอนทิตีคลาสใด ๆ ในบริบทของโปรแกรมประยุกต์ถูกเปลี่ยนเป็นแถว (Record) ในตารางฐานข้อมูล

ดังได้กล่าวมาแล้วข้างต้น งานวิจัยชิ้นนี้ได้มีแนวคิดในการผูกความสัมพันธ์คุณสมบัติต่าง ๆ ของตัวแทน Instance ของเอนทิตีคลาสเหล่านั้นให้มีค่าที่ตรงกัน โดยการออกแบบวิธีเกี่ยวกับการผูกความสัมพันธ์ (Binding Mechanism) ให้กับกรอบงาน อีกทั้งยังเพิ่มความสะดวกและความยืดหยุ่นให้กับนักพัฒนาโปรแกรมประยุกต์โดยการกำหนดให้กำหนดตัวแปรเสริมที่ต้องการผูกความสัมพันธ์ผ่านทางเอกสาร XML



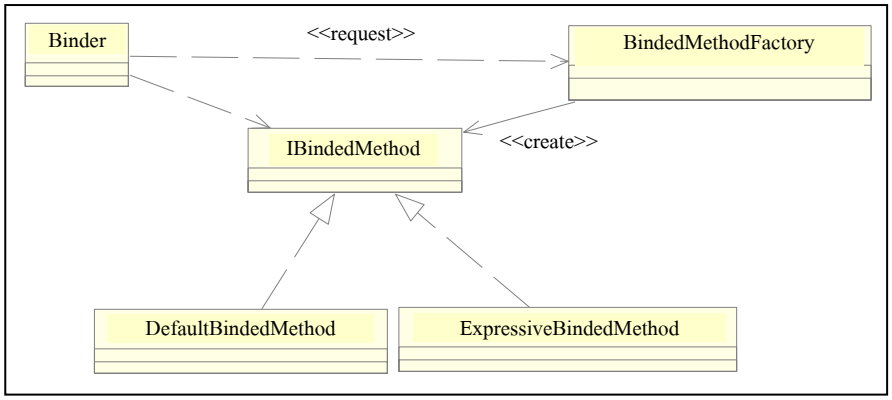
รูปที่ 3.20 แสดงตัวอย่างความสัมพันธ์ระหว่างเอนทิตีคลาสกับเอกสาร XML

รูปที่ 3.20 เป็นการแสดงตัวอย่างความสัมพันธ์ของคลาส Register กับเอกสาร XML ที่มีโหนด (Node) ชื่อ <table>[properties]</table> โดยภายโหนด <table> นี้ยังมีโหนดลูกชื่อ <property></property> ซึ่งเป็นค่าตัวแปรเสริมให้กับตัวสร้างตารางข้อมูลของกรอบ ในทางเทคนิคนั้นตัวแปรเชิงคลาส (Class Variable) ของคลาส Register ซึ่งเปรียบเสมือนตัวแทนคุณสมบัติหรือสถานะของ instance แต่ละ Instance ที่ถูกสร้างขึ้นจากคลาส Register นี้และจะถูกกำหนดค่าต่าง ๆ ผ่านทางเมธอดที่ใช้กำหนดค่าโดยเฉพาะ (Setter Method) และจะส่งค่าผ่านทางเมธอดสำหรับส่งค่ากลับโดยเฉพาะ (Getter Method) ดังนั้นแล้วกรอบงานจึงถูกออกแบบให้อ่านค่าการกำหนดค่าตัวแปรเสริมจากเอกสาร XML จากโหนด <property /> และสร้าง Instance ของคลาสควบคุมห่อหุ้มการทำงานในการผูกความสัมพันธ์คุณสมบัติของ Instance ของคลาส Register กับตารางข้อมูลชื่อ Register

กรอบงานได้ถูกออกแบบให้พิจารณาเมธอดที่ถูกจับคู่กับโหนด <property /> ในเอกสาร XML ออกเป็น 2 ประเภท และสร้างตัวควบคุมทำงานกับเมธอดทั้งสองประเภทนี้ ได้แก่

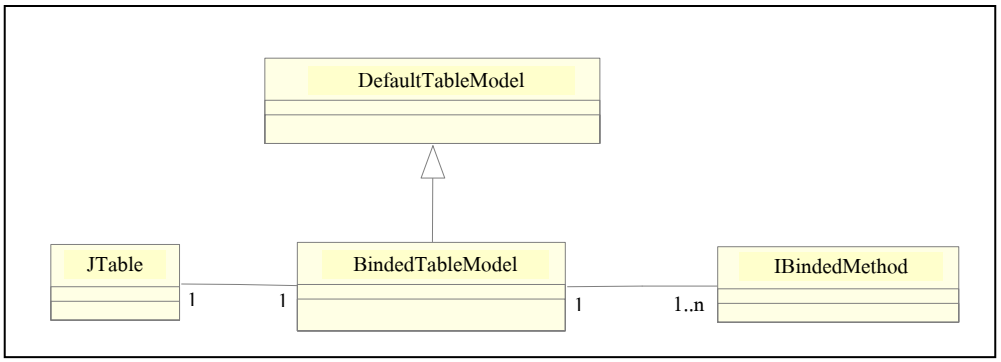
1) DefaultBindedMethod ทำหน้าที่ควบคุมค่าที่เปลี่ยนแปลงไปของ Getter และ Setter method ที่ใช้กับตัวแปรคลาสของคลาสนั้นเองหรืออาจเปรียบได้ว่าคุณสมบัติของ Instance นั้น ๆ เอง ดังตัวอย่างที่แสดงในรูปที่ 3.20 นั้นแสดงโดยชัดเจนด้วยค่าตัวแปรที่ชื่อ score ซึ่งเป็นคุณสมบัติพื้นฐาน (Primitive type) ของ Instance ของคลาส Register

2) ExpressiveBindedMethod เป็นตัวควบคุมที่ถูกใช้กับค่าของโหนด <property/> ที่ถูกกำหนดในลักษณะของ Expression Language ซึ่งใช้ในการอ้างถึงเมธอดของ Instance ที่มีความสัมพันธ์ในเชิงวัตถุกับ Instance ของคลาสที่สนใจ ดังตัวอย่างที่แสดงในรูปที่ 3.21 นั้นการกำหนดด้วย ${student.name} เป็นการอ้างถึง Getter และ Setter method ของ Instance ของคลาส Student ซึ่งมีความสัมพันธ์ในเชิงวัตถุกับคลาส Register ซึ่งเป็นคลาสหลักแบบ N: 1



รูปที่ 3.21 แสดง Factory Pattern สำหรับสร้างตัวควบคุมการเรียกเมธอด

รูปที่ 3.21 แสดงการใช้ Factory Pattern สำหรับสร้าง Instance ของคลาส DefaultBindedMethod และ ExpressivebindedMethod โดยในขั้นตอนของการกำหนดค่าเริ่มต้นให้กับระบบนั้น กรอบงานจะอ่านค่าจากเอกสาร XML และทำการสร้าง Instance ของคลาสลูกของ IBindedMethod โดยอัตโนมัติ โดยตัวควบคุมเหล่านี้จะถูกบรรจุไว้กับแบบจำลองของตารางข้อมูลดังแสดงในรูปที่ 3.22



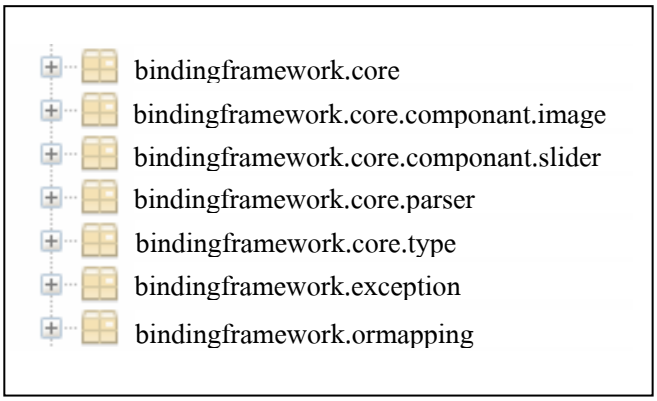
รูปที่ 3.22 แสดงความสัมพันธ์ระหว่างตัวควบคุมการเรียกใช้เมธอด
กับแบบจำลองของตารางข้อมูล

จากรูปที่ 3.22 นั้นแสดงความสัมพันธ์ระหว่างตัวควบคุมการเรียกใช้เมธอดในที่นี้ถูกแทนด้วยอินเตอร์เฟสชื่อ IBinedMethod กับแบบจำลองของตารางข้อมูลซึ่งถูกแทนด้วยคลาส BindedTableModel โดยในกลไกการทำงานของ Java Foundation Classes (JFC) ซึ่งเป็นกรอบงานในการควบคุมการทำงานของส่วนแสดงผลนั้นจะทำการเรียกให้คลาสที่ทำหน้าที่เฝ้าระวังเหตุการณ์ (Listener) ให้ทำงานหากเกิดเหตุการณ์ต่าง ๆ ขึ้น ดังนั้นแล้วหากมีการแก้ไขค่าต่าง ๆ ผ่านทางตารางข้อมูลจึงทำให้เมธอดที่ทำหน้าที่เป็น Getter และ Setter ของคลาสซึ่งเป็นเอนทิตีถูกทำงานและทำการเปลี่ยนแปลงสถานะของ Instance ต่าง ๆ ที่อยู่ในบริบทของโปรแกรม

3.3 การพัฒนาระบบ

นอกจากการออกแบบวิธีและกลไกต่าง ๆ ในการพัฒนากรอบงานนี้แล้ว งานวิจัยนี้นั้นได้พัฒนาในรูปแบบของส่วนโปรแกรมที่สามารถนำกลับมาใช้ใหม่ได้ (Reusable Component) โดยผู้วิจัยได้ออกแบบให้พัฒนาบนสภาพแวดล้อมของภาษา Java (Java Platform) โดยผลลัพธ์จากการพัฒนานั้นจะได้ส่วนโปรแกรมที่อยู่ในรูป Java Archive File (Jar File) ซึ่งนักพัฒนาโปรแกรมประยุกต์สำหรับการใช้งานเฉพาะด้านใด ๆ นั้นสามารถอ้างอิงส่วนของโปรแกรมจากงานวิจัยนี้ได้โดยง่าย

เนื่องจากงานวิจัยชิ้นนี้ได้ถูกออกแบบให้สนับสนุนการพัฒนาโปรแกรมประยุกต์บนสภาพแวดล้อมของ Java ดังนั้นแล้วการออกแบบจึงจำเป็นต้องพัฒนาส่วนโปรแกรมต่าง ๆ ให้อยู่ในรูปแบบของกลุ่มหรือคลัง (Package) ของส่วนโปรแกรม โดยกรอบงานนี้ได้ออกแบบคลังของส่วนโปรแกรมต่าง ๆ ดังแสดงในรูปที่ 3.23



รูปที่ 3.23 แสดงโครงสร้างของส่วนโปรแกรมต่าง ๆ ในการสร้างกรอบงาน

จากรูปที่ 3.23 นั้นแสดงคลังส่วนโปรแกรมต่าง ๆ ดังต่อไปนี้

1) bindingframework.core เป็นคลังที่เปรียบเสมือนเป็นแก่นของกรอบงานเก็บคลาสต่าง ๆ ทั้งหมด 24 คลาส โดยแก่นของกรอลบงานนี้ทำหน้าที่ในการสร้างสภาพแวดล้อมที่เหมาะสมให้แก่กรอบงานสำหรับการสร้างตารางข้อมูลและเชื่อมโยงความสัมพันธ์ต่าง ๆ ของส่วนโปรแกรมที่มีอยู่แล้วและส่วนโปรแกรมที่อาจกำหนดเพิ่มโดยนักพัฒนาส่วนโปรแกรมในอนาคต
2) bindingframework.core.component เป็นคลังของส่วนโปรแกรมที่ใช้สร้างส่วนแสดงผลต่าง ๆ อาทิเช่น รูปภาพ และแถบเลื่อน เป็นต้น
3) bindingframewrok.core.component.image เป็นคลังเก็บส่วนแสดงผลแบบรูปภาพ
4) bindingframework.core.component.sliderเป็นคลังเก็บส่วนแสดงผลแบบแถบเลื่อน
5) bindingframework.core.parser ใช้สำหรับเก็บส่วนโปแกรมต่าง ๆ ที่สนับสนุนการอ่านเอกสาร XML ซึ่งจะถูกนำมาใช้ในการกำหนดสภาพแวดล้อมที่เหมาะสมให้กับกรอบงาน
6) bindingframework.type ใช้เป็นคลังของส่วนโปรแกรมที่ทำหน้าที่เก็บชนิดของตัวแปรในภาษา Java ที่มีอยู่เดิมและอาจถูกเพิ่มโดยนักพัฒนาโปรแกรมเฉพาะทาง
7) bindingframework.exception ใช้ในการเก็บคลาสที่ทำหน้าที่จัดการความผิดพลาดที่เกิดขึ้นในขณะโปแกรมคอมไพล์และดำเนินการ (Exception Handling)
8) bindingframework.ormapping เป็นคลังส่วนโปรแกรมที่ทำหน้าที่จัดการเกี่ยวกับ Object-Relation Mapping ซึ่งเป็นส่วนสำคัญในการเข้าถึงข้อมูลที่มีความสัมพันธ์ในเชิงวัตถุและแบบจำลองข้อมูลเชิงสัมพันธ์

ส่วนรหัสคำสั่งในคลังส่วนโปรแกรมต่าง ๆ นั้นถูกเขียนอยู่ในรูปแบบของเอกสารซึ่งมีนามสกุลเป็น .java และจะถูกแปลงเป็นรหัสคำสั่งแบบไบนารี (Byte Code) ในนามสกุล .class และในท้ายที่สุดรหัสคำสั่งทั้งหมดที่อยู่ในคลังส่วนโปรแกรมต่าง ๆ จะถูกบีบอัดเป็น Java Archive File (Jar File) ดังได้กล่าวมาแล้วในข้างต้นเพื่อให้นักพัฒนาโปรแกรมเฉพาะใช้งานต่อไป

บทที่ 4
การทดสอบและอภิปรายผล

การพัฒนากรอบงานเชิงวัตถุนั้นเป็นการพัฒนาส่วนโปรแกรมซึ่งสนับสนุนการนำกลับมาใช้ใหม่โดยมีแนวคิดและวัตถุประสงค์เพื่อช่วยเพิ่มประสิทธิภาพในการพัฒนาโปรแกรมประยุกต์เฉพาะด้านภายใต้กรอบงานนั้น ดังนั้นวิทยานิพนธ์ส่วนนี้จึงนำเสนอวิธีการและอภิปรายผลจากการทดสอบการทำงานและประสิทธิภาพของกรอบงานด้วยวิธีต่าง ๆ อันได้แก่

1) วิเคราะห์จำนวนรหัสคำสั่งที่ใช้ในโปรแกรมประยุกต์เฉพาะทาง
2) วิเคราะห์ความยึดติด (Coupling) ของโปรแกรมประยุกต์เฉพาะทาง
3) วิเคราะห์ความยืดหยุ่น (Flexibility) ของโปรแกรมประยุกต์เฉพาะทาง
4) ทดสอบเวลาที่ใช้ในการสร้างตารางข้อมูลและผูกความสัมพันธ์ของวัตถุนามธรรม

การวิเคราะห์และทดสอบทั้ง 4 ประเภทนั้นสามารถใช้ในการอธิบายประสิทธิภาพของกรอบงานที่ถูกนำมาใช้ในการสนับสนุนนักพัฒนาในการพัฒนาโปรแกรมประยุกต์เฉพาะทาง การวิเคราะห์จำนวนรหัสคำสั่งที่อาจเพิ่มขึ้นหรือลดลงนั้น เป็นการวิเคราะห์ความเป็นไปได้ที่กรอบงานจะช่วยลดระยะเวลาและความซับซ้อนของรหัสคำสั่งในการพัฒนาโปรแกรมประยุกต์เฉพาะทาง ในการวิเคราะห์ความยึดติดและความยืดหยุ่นของโปรแกรมประยุกต์ในการใช้กรอบงานนั้นเป็นสิ่งบ่งชี้ถึงความสามารถในปรับเปลี่ยนและแก้ไขโปรแกรมประยุกต์เฉพาะทาง และในท้ายที่สุดเป็นการทดสอบเวลาที่ใช้ในการสร้างตารางข้อมูลและผูกความสัมพันธ์ของวัตถุนามธรรมโดยการทดสอบนี้เป็นการหาระยะเวลาที่ต้องใช้จากการที่กรอบงานต้องอ่านค่าจากตัวแปรเสริมที่อยู่ในเอกสาร XML และทำการกำหนดสภาพแวดล้อมที่เหมาะสมต่าง ๆ ให้กับโปรแกรมประยุกต์

4.1 วิเคราะห์จำนวนรหัสคำสั่งที่ใช้ในโปรแกรมประยุกต์เฉพาะทาง

4.1.1 การเขียนรหัสคำสั่งโดยไม่ใช้กรอบงาน

ต่อไปนี้จะเป็นการแสดงตัวอย่างของการแสดงรหัสคำสั่ง ที่ใช้ในการแสดงผลข้อมูลนามธรรม และการผูกความสัมพันธ์ของข้อมูลเหล่านั้น

```
1.  Hibernateutil util = new HibernateUtil();
2.   String colname[] = {"fname", "lname", "advisor"};
3.   DefaultTableModel model = new DefaultTableModel (colname, 0);
4.   JTable table1 = new JTable(model);
5.   List<Student> list = util.findAll (Student.class);
6.   for (Student x:list){
7.        vector data=new Vector ();
8.        data.add(x.getFname());
9.        data.add(x.getLname());
10.         data.add(x.getAdvisor().getName());
11.         model.addRow(data);
12.  }
```

รูปที่ 4.1 แสดงตัวอย่างการเขียนรหัสแสดงผลข้อมูลนามธรรม

จากรูปที่ 4.1 ในบรรทัดที่ 5 นั้นเป็นการใช้ Object-Relational Mapping ในการตั้งข้อคำถามเพื่อนำข้อมูลต่าง ๆ ของนักศึกษามาแสดงผล บรรทัดที่ 6 -12 นั้นเป็นการดึงข้อมูลจากแต่ละคุณสมบัติของ Instance เพื่อนำมาใส่ในแต่ละสดมภ์ของแบบจำลองตารางข้อมูล (Table Model) ดังนั้นแล้วจากตัวอย่างการเขียนรหัสคำสั่งอย่างง่ายข้างต้นนั้นจำนวนบรรทัดของละรหัสคำสั่ง (Line of Code) หรือย่อด้วยสัญลักษณ์ LoC นั้นสามารถคำนวณได้ดังนี้ LoC(m) = 7 + n + C

จากสมการข้างต้นนั้น 7 บรรทัดแรกใช้ในการกำหนดค่าเริ่มต้นที่ใช้ในการสร้างตารางข้อมูลและการอ่านข้อมูลจากตัว OR-mapping component ซึ่งในที่นี้ใช้ Hibernate ซึ่ง n เป็นจำนวนของค่าคุณสมบัติของข้อมูลที่ต้องการแสดงผล m เป็นสัญลักษณ์แทนโมดูลที่พิจารณา และ C เป็นค่าคงที่ของจำนวนบรรทัดในส่วนอื่นของโปรแกรมที่ไม่เกี่ยวข้องกับแสดงผลข้อมูลนามธรรมและการผูกความสัมพันธ์ของข้อมูล

โดยตัวอย่างที่แสดงในรูปที่ 4.1 นั้นเป็นเพียงแต่การนำคุณสมบัติของข้อมูลนามธรรมแสดงผลเท่านั้นยังมิได้มีกลไกการผูกความสัมพันธ์ของข้อมูลในชั้นต่าง ๆ อันได้แก่ ชั้นแสดงผล ชั้นตรรกะในการทำงาน (Business Logic) และเอนทิตีในแบบจำลองฐานข้อมูลเชิงสัมพันธ์ โดยในการผูกความสัมพันธ์นั้นสามารถเขียนรหัสคำสั่งต่อหนึ่งคุณสมบัติโดยง่ายดังต่อไปนี้

```
1. Property fnameProperty=BeanProperty.create(“fname”)
2. fnameProperty.addpropertyStateListener(x, new PropertyStateListener(){
3.       public void propertyStateChanged(propertyStateEvent pse){
4.             // codes
5. });
```

รูปที่ 4.2 แสดงตัวอย่างการเพิ่มตัวเฝ้าระวัง (Listener) ในการผูกความสัมพันธ์ของข้อมูลนามธรรม

จากรูปที่ 4.2 นั้นแสดงให้เห็นโดยพิจารณาได้ว่าในการผูกความสัมพันธ์คุณสมบัติหนึ่ง ๆ ของข้อมูลนามธรรมนั้นใช้ LoC > 5 เนื่องจากจำเป็นต้องเขียนรหัสคำสั่งที่เฉพาะเจาะจงเพิ่มในบรรทัดที่ 3-5 เพิ่มขึ้น

ดังนั้นแล้วการแสดงผลข้อมูลนามธรรมทั้งหมดนั้นมีค่าจำนวนบรรทัดของรหัสคำสั่งเป็นดังต่อไปนี้ $LoC(m) > 7 + n + 5*n + C$ หรือ $LoC(m) > 7 + 6n + C$

นอกจากนี้แล้วนั้นในการแสดงผลในลักษณะของ JComponent ต่าง ๆ นั้น ตัวอย่างเช่น การระบุให้คะแนนมากกว่า 50 เป็นสีเขียวและน้อยกว่า 50 เป็นสีแดงนั้นยังต้องเพิ่มจำนวนบรรทัดของรหัสคำสั่งเข้าไปอีกด้วย อีกทั้งหากจำเป็นต้องสร้างตารางข้อมูลจำนวนมากกว่า 1 ตารางข้อมูลแล้วนั้นทำให้จำนวนบรรทัดของรหัสคำสั่งเป็น $LoC(m) > \sum_{i=1}^{x}(7 + 6n_i) + C$

โดย x เป็นจำนวนของตารางข้อมูลและ n_i เป็นจำนวนสดมภ์ของแต่ละตารางข้อมูลที่ i ซึ่งถูกผูกความสัมพันธ์เข้ากับแต่ละคุณสมบัติของข้อมูลนามธรรม และ C เป็นค่าคงที่ของจำนวนบรรทัดในส่วนอื่นของโปรแกรมที่ไม่เกี่ยวข้องกับแสดงผลข้อมูลนามธรรมและการผูกความสัมพันธ์ของข้อมูล

4.1.2 การเขียนรหัสคำสั่งโดยใช้กรอบงาน

กรอบงานนั้นถูกสร้างขึ้นมาเพื่ออำนวยความสะดวกให้กับนักพัฒนาโปรแกรมเฉพาะทางให้สามารถทำงานผ่านกรอบงานได้โดยลดความซับซ้อนของการเขียนรหัสคำสั่งและหลีกเลี่ยงการเขียนรหัสคำสั่งเพื่อให้โปรแกรมประยุกต์สามารถทำงานตามกลไกการผูกความสัมพันธ์ของข้อมูลได้ (Synchronization) โดยหากนักพัฒนาโปรแกรมประยุกต์สามารถเขียนรหัสคำสั่งโดยการใช้กรอบงานได้ดังตัวอย่างที่แสดงในรูปที่ 4.3

```
ApplicationContext.newDefinition(new file(“Student.db.xml”));
ApplicationContext.newDefinition(new file(“TableDefinition.db.xml”));
ApplicationContext context=new ApplicationContext (“001”);
Context.addBindingComponent(“grade”,new GradeComponent());
table1 = context.createTable(“register”);
table1 = context.createTable(“student”);
```

รูปที่ 4.3 แสดงตัวอย่างการเขียนรหัสคำสั่งโดยใช้ความสามารถของกรอบงาน

จากรูปนั้นกรอบงานช่วยลดความซับซ้อนและจำนวนของการเขียนรหัสคำสั่งได้อย่างชัดเจน ซึ่งอาจเขียนเป็นสมการได้ดังต่อไปนี้ LoC(m) = 1+ d + x + C โดยที่ d เป็นจำนวนของเอกสาร XML ที่ใช้ในการกำหนดตัวแปรเสริม X เป็นจำนวนของตารางข้อมูลที่ต้องการสร้าง M เป็นสัญลักษณ์แทนโมดูลที่พิจารณาและ C เป็นค่าคงที่ของจำนวนบรรทัดในส่วนอื่นของโปรแกรมที่ไม่เกี่ยวข้องกับแสดงผลข้อมูลนามธรรมและการผูกความสัมพันธ์ของข้อมูล

4.1.3 เปรียบเทียบการเขียนรหัสคำสั่งโดยใช้กรอบงานและไม่ใช้กรอบงาน

กรณีทดสอบที่ 1 หากต้องการสร้างตารางข้อมูลจำนวน 1 ตารางข้อมูล ตารางข้อมูลละ 10 สดมภ์และผูกความสัมพันธ์ของชั้นการแสดงผลไปกระทั่งชั้นฐานข้อมูลเชิงสัมพันธ์ โดยต่อไปนี้ กำหนดให้ C เป็นค่าคงที่ของจำนวนบรรทัดในส่วนอื่นของโปรแกรมที่ไม่เกี่ยวข้องกับแสดงผลข้อมูลนามธรรมและการผูกความสัมพันธ์ของข้อมูล หากไม่ใช้กรอบงานจะได้ค่า LoC(m) > 7 + (6 × 10) + C หรือ LoC(m) > 67 + C บรรทัด หากพัฒนาโดยใช้กรอบงานและใช้เอกสาร XML ในการกำหนดค่า 1 เอกสาร LoC(m) = 1 + 1 + 1 + C หรือ LoC(m) = 3 + C บรรทัด

กรณีทดสอบที่ 2 หากต้องการสร้างตารางข้อมูลจำนวน 2 ตารางข้อมูล แบ่งเป็นตารางข้อมูลละ 10 สดมภ์ และผูกความสัมพันธ์ของชั้นการแสดงผลไปกระทั่งชั้นฐานข้อมูลเชิง

$$\sum_{i=1}^{2}(7+6n_i)$$

หากพัฒนาโดยใช้กรอบงานและใช้เอกสาร XML ในการกำหนดค่า 1 เอกสารLoC(m) = 1 + 1 + 2 + C หรือ LoC(m) = 4 + C บรรทัด

กรณีทดสอบที่ 2 หากต้องการสร้างตารางข้อมูลจำนวน 3 ตารางข้อมูล แบ่งเป็นตารางข้อมูลละ 10 สดมภ์ และผูกความสัมพันธ์ของชั้นการแสดงผลไปกระทั่งชั้นฐานข้อมูลเชิงสัมพันธ์ หากไม่ใช้กรอบงานจะได้ค่า LoC(m) > $\sum_{i=1}^{3}(7+6n_i)$ + C หรือ LoC(m) > (7 + (6 × 10)) + (7 + (6 × 10)) + (7 + (6 × 10)) + C จะได้ว่า LoC(m) > 201 + C บรรทัด

หากพัฒนาโดยใช้กรอบงานและใช้เอกสาร XML ในการกำหนดค่า 1 เอกสารLoC(m) = 1 + 1 + 3 + C หรือ LoC(m) = 5 + C บรรทัด

ดังนั้นจากกรณีทดสอบทั้งสามสามารถกล่าวโดยสรุปได้อย่างเห็นชัดเจนว่า ในส่วนของจำนวนรหัสคำสั่งและความซับซ้อนที่ใช้ในโปรแกรมประยุกต์ลดน้อยลงเป็นจำนวนมาก จึงเป็นการเพิ่มประสิทธิภาพในการพัฒนาโปรแกรมประยุกต์

4.2 วิเคราะห์ความยึดติดของโปรแกรมประยุกต์เฉพาะทาง

ในการวัดความยึดติด (Coupling Measuring) ส่วนโปรแกรมต่าง ๆ ที่ได้พัฒนาด้วยเทคโนโลยีเชิงวัตถุนั้นมีการกล่าวถึงอยู่ในหลายงานวิจัยและหลากหลายแนวคิดในบทความเชิงวิชาการ อาทิเช่น

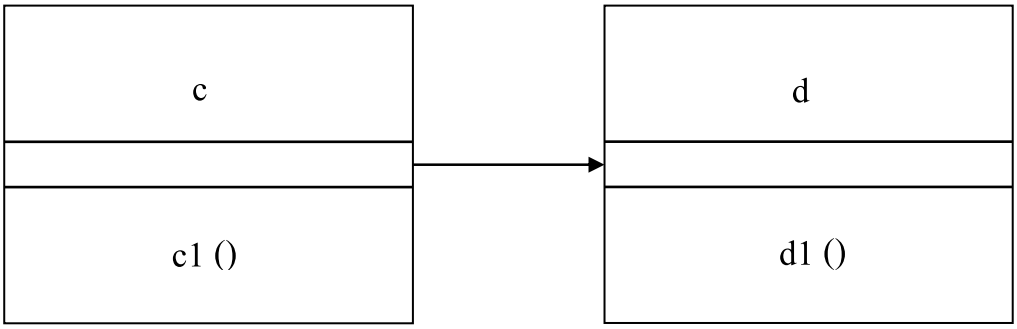
Briand, Daly, and Wust (1998) และ Ghassemi and Mourant (2000) ได้กล่าวถึงสมการที่ใช้ในการวัดค่าความยึดติดของระบบซึ่งประกอบไปด้วยคลาสต่าง ๆ จำนวนมาก โดยนิยามวิธีการคำนวณค่า Coupling factor (CoF) ดังนี้

$$COF(C)=\frac{\sum_{c\in C}\left|\left\{d \middle| d\in C-(\{c\}\cup Ancestors(c))\wedge uses(c,d,i)\right\}\right|}{|C|^2-|C|-\left(2\sum_{c\in C}|Descendents(c)|\right)}$$

โดย Ghassemi and Mourant (2000) ได้ให้นิยามของเพรดิเคต (Predicate) uses(c, d, i) ดังนี้

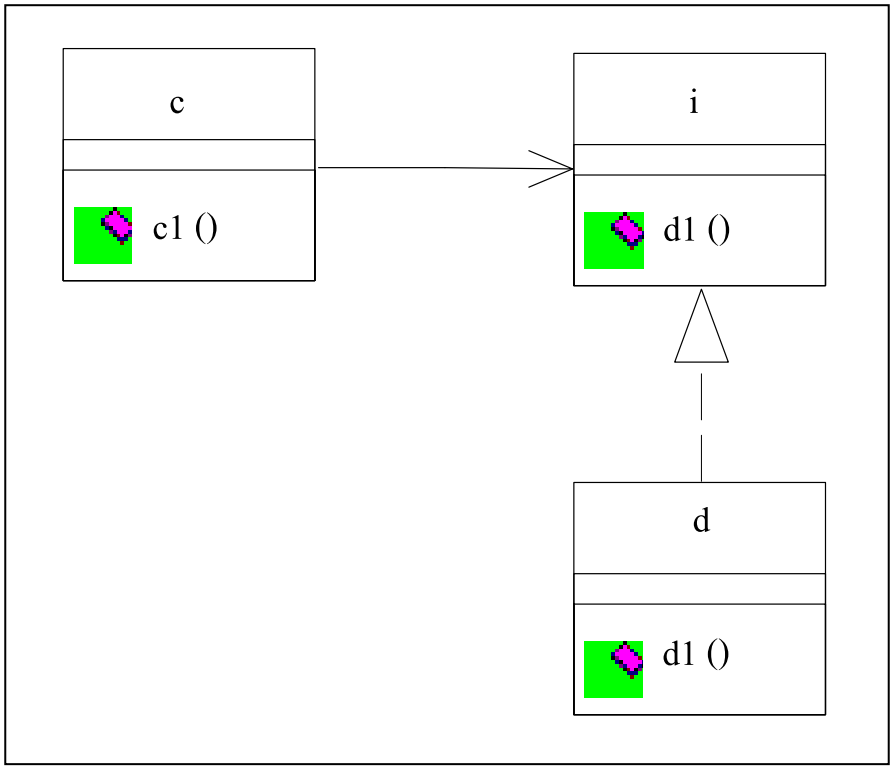
$$uses(c,d,i)\Leftrightarrow((\exists m\in M_I(c):\exists m'\in M_I(d):m'\in PIM(m))\vee$$
$$(\exists m\in M_I(c):\exists a\in A_i(d):a\in AR(m)))\wedge$$
$$(\exists m'\in M_I(d):m'\notin Md(i))$$

จากทั้งสองสมการนั้นกล่าวโดยสรุปได้ว่าการหา CoF นั้นสามารถคำนวณได้การหาผลรวมของ uses(c, d, i) โดย c, d เป็นสมาชิกของคลาสในระบบ $(c \in C, d \in C)$ และ i เป็นสมาชิกของอินเตอร์เฟสในระบบ $(i \in I)$ โดย uses(c, d, i) ของคลาส c และ d คู่ใด ๆ ในระบบจะมีค่าเป็น 1 ก็ต่อเมื่อคลาส c มีการเรียกใช้เมธอดหรือแอทิบิวต์ (Attribute) ของคลาส d โดยเมธอดของคลาส d ที่ถูกเรียกใช้นั้นไม่ได้ถูกเรียกผ่านอินเตอร์เฟส i โดยอาศัยหลักการของสภาวะพหุสัณฐานยกตัวอย่างได้ ดังรูปที่ 4.4



รูปที่ 4.4 แสดงความสัมพันธ์เชิงวัตถุของคลาส c และ d

จากรูปที่ 4.4 นั้นจะได้ uses(c, d, i) เป็น 1 เนื่องจากเมธอด c1 ของคลาส c เรียกใช้เมธอด d1 ของคลาส d จึงถือได้ว่ามี Coupling

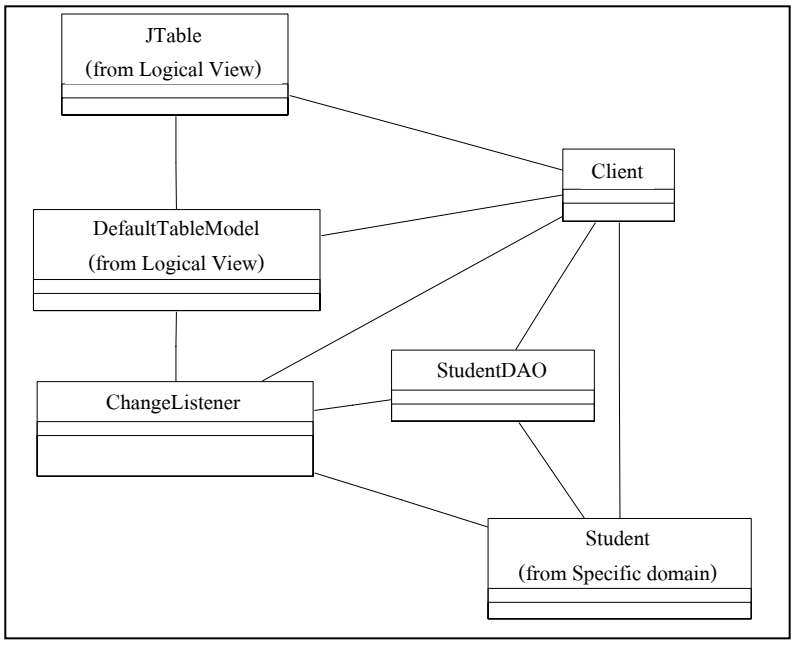


รูปที่ 4.5 แสดงความสัมพันธ์เชิงวัตถุผ่านอินเตอร์เฟส

จากรูปที่ 4.5 นั้นจะได้ uses(c, d, i) เป็น 0 เนื่องจากเมธอด c1 ของคลาส c นั้นเรียกใช้เมธอด d1() ของ d โดยอาศัยสถาวะพหุสัณฐานและการเชื่อมโยงแบบพลวัต (Dynamics Binding) ผ่านทางอินเตอร์เฟส i ซึ่งการที่ผลลัพธ์ของเพรดิเคต uses(c, d, i) เป็น 0 นั้น เนื่องมาจากการใช้อินเตอร์เฟสซึ่งเป็นตัวกลางระหว่างคลาสที่ขอรับบริการ (Client) และคลาสที่ให้บริการ (Server) ทำให้ความยึดติดนั้นต่ำลง

ดังได้กล่าวถึงการคำนวณค่า CoF ข้างต้นแล้วนั้น ต่อไปนี้ เป็นการวิเคราะห์เพื่อเปรียบเทียบการหาค่าความยึดติดของโปรแกรมประยุกต์เฉพาะทางในกรณีที่ถูกพัฒนาภายใต้กรอบงานและไม่ใช้ความสามารถของกรอบงาน

1) วิเคราะห์ค่า CoF ของโปรแกรมประยุกต์ที่ถูกพัฒนาขึ้นโดยไม่ใช้กรอบงาน



รูปที่ 4.6 แสดงการแสดงผลข้อมูลนามธรรมโดยไม่ใช้กรอบงาน

จากรูปที่ 4.6 นั้นเป็นการแสดงวิธีการนำข้อมูลของนักศึกษาทุกคนจากฐานข้อมูลเชิงสัมพันธ์โดยผ่านการใช้ข้อคำถาม (Query) โดยคลาส StudentDAO และแปลงข้อมูลที่อยู่ในรูปแบบกลุ่มของ Primitive Type ให้เปลี่ยนเป็นข้อมูลเชิงวัตถุในที่นี้คือ Instance ของคลาส Student และนำข้อมูลเหล่านั้นแสดงในตารางข้อมูลโดยผ่านแบบจำลองตารางข้อมูล (DefaultTableModel) โดยมีตัวเฝ้ามอง (ChangeListener) การเปลี่ยนแปลงค่าของข้อมูลเหล่านั้น หากมีการเปลี่ยนแปลงข้อมูลในสดมภ์หนึ่งสดมภ์ ตัวเฝ้ามองจะทำการเปลี่ยนแปลงค่าเหล่านั้นทั้งใน Instance และค่าในฐานข้อมูลเชิงสัมพันธ์ จากแผนภาพเชิงคลาสข้างต้นนั้นแสดงการคำนวณค่า CoF ได้ดังนี้

กำหนดให้

A แทนคลาส JTable

B แทนคลาส DefaultTableModel

C แทนคลาส ChangeListener

D แทนคลาส Client

E แทนคลาส StudentDAO

F แทนคลาส Student

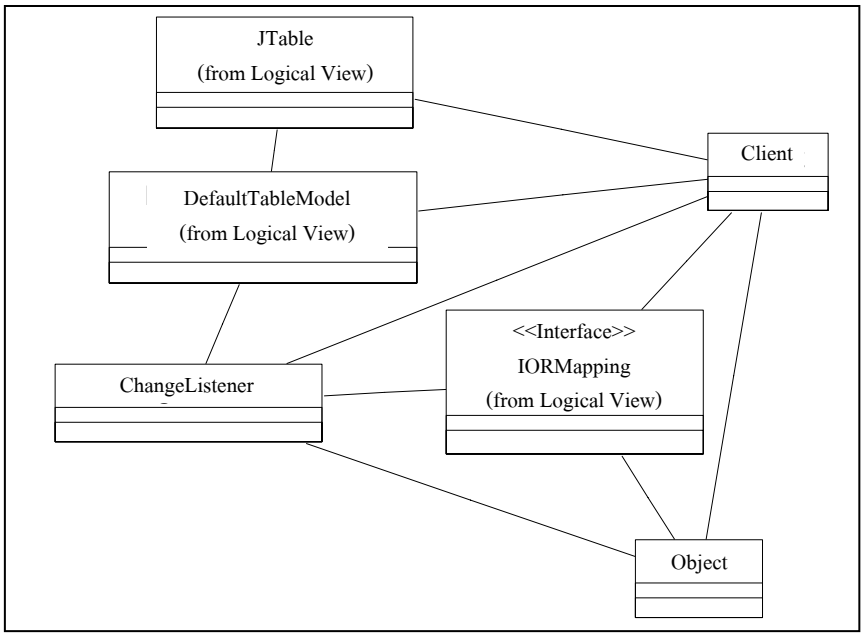
X แทนจำนวนคลาสทั้งหมดที่อยู่ในระบบ = 6

ตารางแสดงความสัมพันธ์ระหว่างคลาสแต่ละคู่ในระบบ

-	A	B	C	D	E	F
A	-	1	0	1	0	0
B	0	-	1	1	0	0
C	0	0	-	1	1	1
D	0	0	0	-	1	1
E	0	0	0	0	-	1
F	0	0	0	0	0	-

$$\text{CoF} = \frac{10}{6^2 - 6 - 0} = \frac{10}{30}$$

2) วิเคราะห์ค่า CoF ของโปรแกรมประยุกต์ที่ถูกพัฒนาขึ้นภายใต้กรอบงาน



รูปที่ 4.7 แสดงการพัฒนาโปรแกรมประยุกต์ภายใต้กรอบงาน

จากรูปที่ 4.7 นั้นแสดงถึงการพัฒนาโปรแกรมประยุกต์เฉพาะทางภายใต้กรอบงานซึ่งสร้างกลไกโดยการแทนคลาสซึ่งเป็น Data Access Object (DAO) ด้วยอินเตอร์เฟส IORMapping เพื่อลดให้ความยึดติดของระบบต่ำลง (Low Coupling) จากรูปที่ 4.7 เชิงคลาสข้างต้น แสดงการคำนวณค่า CoF ได้ดังนี้

กำหนดให้

A แทนคลาส JTable

B แทนคลาส DefaultTableModel

C แทนคลาส ChangeListener

D แทนคลาส Client

E แทนอินเตอร์เฟส IORMapping

F แทนคลาส Object

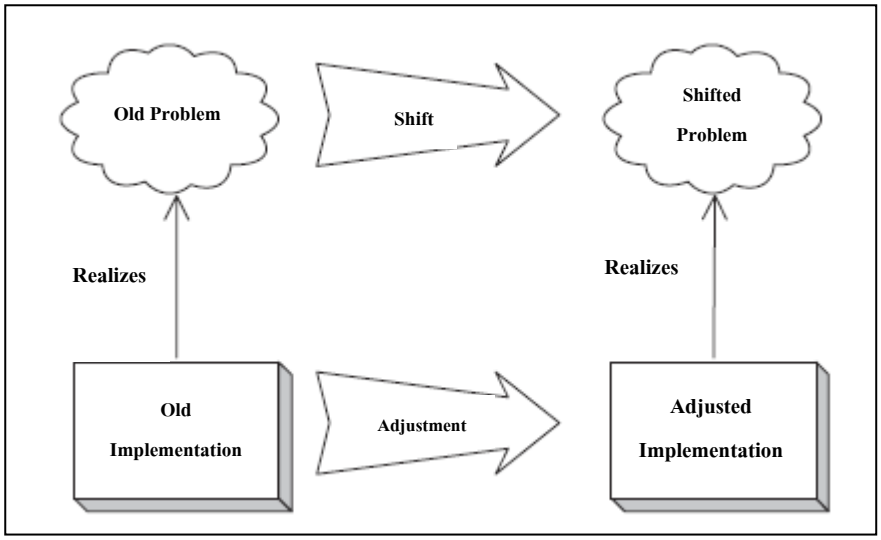
X แทนจำนวนคลาสทั้งหมดที่อยู่ในระบบ = 6

ตารางแสดงความสัมพันธ์ระหว่างคลาสแต่ละคู่ในระบบ

-	A	B	C	D	E	F
A	-	1	0	1	0	0
B	0	-	1	1	0	0
C	0	0	-	1	0	1
D	0	0	0	-	0	1
E	0	0	0	0	-	1
F	0	0	0	0	0	-

$$\text{CoF} = \frac{8}{6^2 - 6 - 0} = \frac{8}{30}$$ จากการเปรียบเทียบค่า Coupling Factor ของการพัฒนาโปรแกรมประยุกต์โดยการใช้กรอบงานและไม่ใช้กรอบงานนั้น จะเห็นได้ว่าการใช้กรอบงานนั้นมีค่า Coupling Factor ที่ต่ำกว่า โดยค่า Coupling Factor ที่ถูกลดไปนั้นเป็นส่วนของการติดต่อฐานข้อมูล จึงวิเคราะห์ได้ว่าการลดลงไปของค่า Coupling Factor ในส่วนนี้ทำให้การติดต่อฐานข้อมูลมีความยืดหยุ่นมากยิ่งขึ้น

Eden and Mens (2006) กล่าวถึงวิวัฒนาการเชิงซอฟต์แวร์ (Evolution Step) ซึ่งเป็นแนวคิดที่ใช้ในการวัดค่าความยืดหยุ่น (Flexibility) ดังแสดงในรูปที่ 4.8



รูปที่ 4.8 แสดง Evolution step

จากรูปที่ 4.8 นั้นกล่าวถึงปัญหาในเชิงความต้องการเชิงซอฟต์แวร์นั้นถูกแก้ด้วยการพัฒนา (Implementation) ระบบหนึ่งและเมื่อความต้องการเปลี่ยนไปเป็นการสะท้อนปัญหาที่มีอยู่เดิมให้เปลี่ยนแปลงไปเช่นเดียวกัน ปัญหาใหม่นั้นก็ได้รับการแก้ไขด้วยการปรับปรุงระบบเดิมที่มีอยู่ก่อน โดยจากรูปที่ 4.8 นั้นสามารถแสดงในรูปของ Evolution Function ได้ดังต่อไปนี้

$\varepsilon(p_{old}, p_{shifted}, i_{old}) = i_{adjusted}$ หรือ $\varepsilon : P \times P \times I \to I$ โดยในการวัดค่าความยืดหยุ่นนั้นได้กำหนดนิยามของ Evolution Cost ($C^{\mu}_{\mathrm{mod}ules}$) ซึ่งหมายถึงจำนวนผลกระทบที่วัดโดยมาตรวัด μ ที่ส่งผลต่อ Modules โดยตัวแปร μ นี้สามารถถูกแทนด้วยมาตรวัดใด ๆ ที่สนใจ อาทิเช่น Line of Code (LoC) เป็นต้น และตัวแปร modules นั้นสามารถถูกแทนด้วยหน่วยที่สนใจ เช่นคลาส เมธอด หรือ Package โดยอาจยกตัวอย่างได้ดังนี้

$C^{LoC}_{classes}(\varepsilon)$ หมายถึง จำนวนของคลาสที่จำนวนของรหัสคำสั่งถูกเปลี่ยนแปลงไปเนื่องมาจากการเปลี่ยนแปลงความต้องการเชิงซอฟต์แวร์หรือ Evolution step โดยยิ่งค่า Evolution Cost นี้ยิ่งน้อยนั้นแสดงถึงความยืดหยุ่นยิ่งมากขึ้น

จากนิยามของค่า Evolution Step ดังได้กล่าวมาแล้วข้างต้นนั้นจึงสรุปความยืดหยุ่นของการใช้กรอบงานดังต่อไปนี้

1) การเปลี่ยนแปลงการติดต่อฐานข้อมูลนั้นทำให้ค่า $C_{classes}^{LoC}(\varepsilon) = 0$ เนื่องจากนักพัฒนาโปรแกรมประยุกต์นั้นสามารถสร้างคลาสซึ่ง Implements อินเตอร์เฟส IORMapping และทำการเชื่อมต่อกับกรอบงานโดยไม่มีผลกระทบต่อรหัสคำสั่งของโปรแกรมแต่อย่างใด
2) การเปลี่ยนแปลงชนิดของส่วนโปรแกรม Object-Relational Mapping นั้นทำให้ค่า $C_{classes}^{LoC}(\varepsilon) = 0$ ด้วยเหตุผลเดียวกันกับข้อ 3.1
3) การเปลี่ยนแปลงลักษณะของตารางข้อมูลภายหลังจากที่โปรแกรมได้ส่งมอบอันได้แก่จำนวนสดมภ์ของตารางข้อมูล ค่าภายในสดมภ์ที่ต้องการแสดงผลและลักษณะ Component ที่แสดงผลทำให้ค่า $C_{classes}^{LoC}(\varepsilon) = 0$ เนื่องจากนักพัฒนาสามารถเปลี่ยนแปลงการกำหนดค่าโดยผ่านเอกสาร XML ไม่จำเป็นต้องแก้ไขรหัสคำสั่งในส่วนของโปรแกรมหลักแต่อย่างใด

4.4 ทดสอบเวลาที่ใช้ในการสร้างตารางข้อมูลและผูกความสัมพันธ์ของวัตถุนามธรรม

เนื่องมาจากการสร้างกรอบงานเพื่อสนับสนุนการพัฒนาโปรแกรมประยุกต์เฉพาะทางของนักพัฒนาโปรแกรมให้มีประสิทธิภาพในด้านเวลาที่พัฒนา เพิ่มความยืดหยุ่นและลดความซับซ้อนของโปรแกรมลง ดังนั้นการพัฒนากรอบงานนั้นจึงจำเป็นต้องสร้างกลไกในการสร้างสภาพแวดล้อมที่เหมาะสมต่าง ๆ โดยกรอบงานต้องอ่านค่าจากเอกสาร XML และสร้างตัวควบคุมที่เหมาะสมจำนวนมากซึ่งทำให้สูญเสียทรัพยากรของระบบส่วนหนึ่งไปดังนั้นจึงได้ทำการตรวจสอบระยะเวลาต่าง ๆ ดังต่อไปนี้

4.4.1 ทรัพยากรที่ใช้

ฮาร์ดแวร์

- คอมพิวเตอร์แบบพีซี Pentium IV 3.2 กิกะเฮิร์ส
- หน่วยความจำ 512 เมกกะไบต์
- ฮาร์ดดิสก์ 80 กิกะไบต์

ซอฟต์แวร์

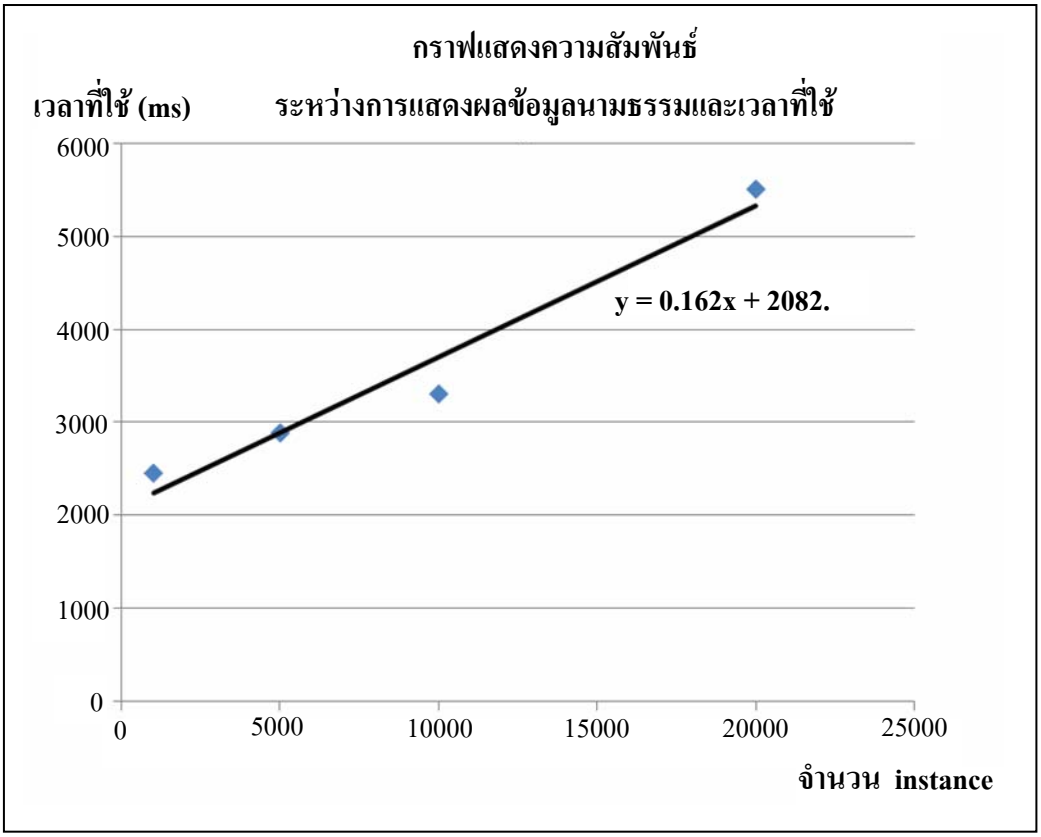
- ระบบปฏิบัติการ ไมโครซอฟต์วินโดวส์ เอ็กซ์พี
- เครื่องมือพัฒนาชื่อ NetBeans IDE 6.0

4.4.2 ทดสอบการสร้าง Instance จากคลาส 1 คลาสด้วยกรอบงาน

ตารางที่ 4.1 แสดงจำนวนของ Instance และเวลาที่ใช้ในการสร้าง Instance จากคลาส 1 คลาส

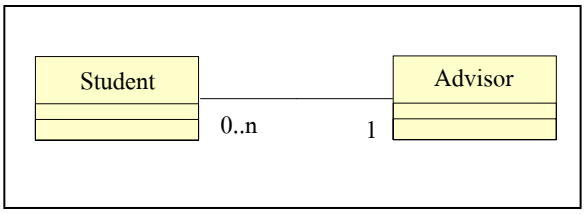
จำนวน (Instance)	เวลาที่ใช้ในแต่ละครั้งที่ทดสอบ (milliseconds)					ค่าเฉลี่ย
	1	2	3	4	5	
1000	3828	2609	1984	1953	1907	2456.2
5000	2640	2640	3375	2954	2844	2890.6
10000	3297	3281	3321	3390	3266	3311
20000	7047	5953	4843	4891	4860	5518.8

จากตารางที่ 4.1 นั้นสามารถแสดงเป็นภาพเส้นได้ดังต่อไปนี้



รูปที่ 4.9 แสดงกราฟความสัมพันธ์ระหว่างการแสดงผลข้อมูลนามธรรมและเวลาที่ใช้จากการสร้าง Instance ของคลาส 1 คลาสด้วยกรอบงาน

4.4.3 ทดสอบการสร้างและผูกความสัมพันธ์ของวัตถุนามธรรม 2 คลาส

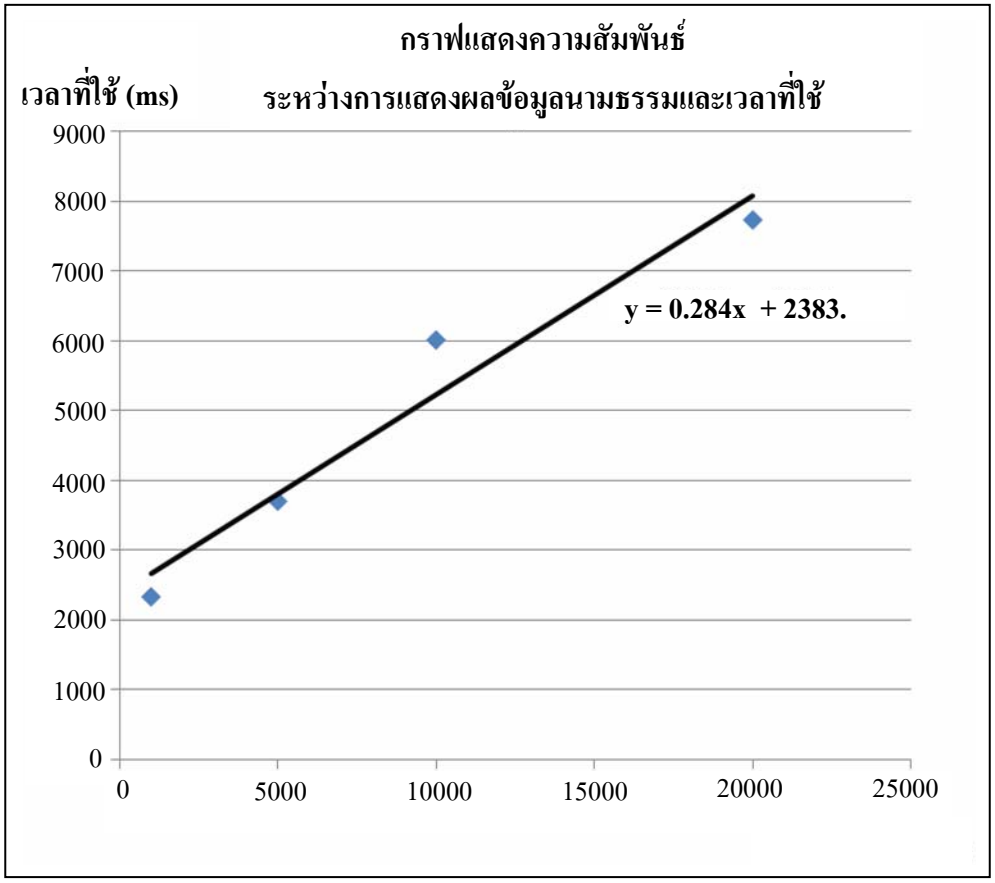


รูปที่ 4.10 แสดงแผนภาพเชิงคลาสที่ให้ความสัมพันธ์ระหว่างคลาสแบบ 1- n ที่ใช้ในการทดสอบการสร้าง Instance ของคลาส Student ภายใต้สภาพแวดล้อมของกรอบงาน

ตารางที่ 4.2 แสดงจำนวนของ Instance และเวลาที่ใช้ในการสร้าง Instance จากคลาสที่มีความสัมพันธ์ กัน 2 คลาสเหล่านั้น

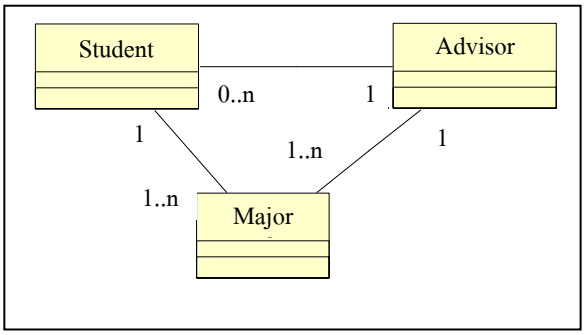
จำนวน (Instance)	เวลาที่ใช้ในแต่ละครั้งที่ทดสอบ (milliseconds)					ค่าเฉลี่ย
	1	2	3	4	5	
1000	2157	2625	2344	2281	2281	2337.6
5000	4203	3578	3609	3547	3578	3703.6
10000	9156	6032	5078	4922	4844	6006.4
20000	7985	7640	7656	7672	7656	7721.8

จากตารางที่ 4.2 นั้นสามารถแสดงเป็นภาพเส้นได้ดังต่อไปนี้



รูปที่ 4.11 แสดงกราฟความสัมพันธ์ระหว่างการแสดงผลข้อมูลนามธรรมและเวลาที่ใช้จากการสร้าง Instance ของคลาส 2 คลาสด้วยกรอบงาน

4.4.4 ทดสอบการสร้างและผูกความสัมพันธ์ของวัตถุนามธรรม 3 คลาส

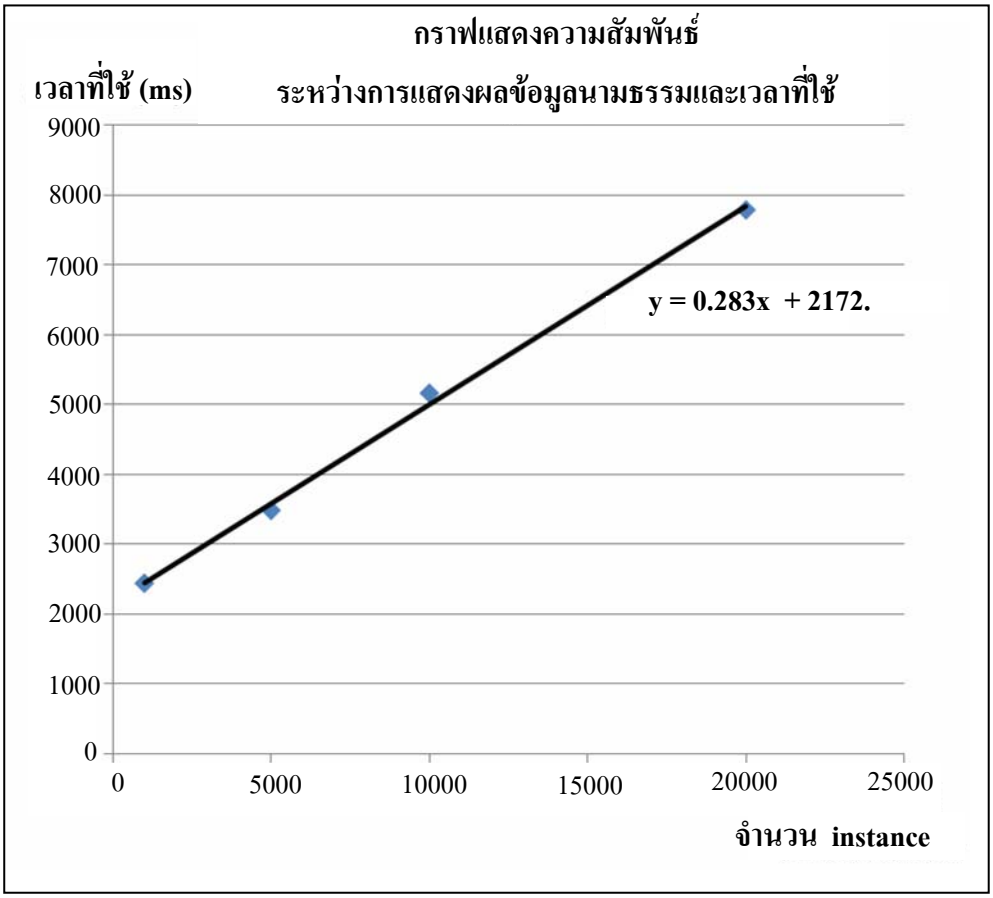


รูปที่ 4.12 แสดงแผนภาพเชิงคลาสที่ให้ความสัมพันธ์ระหว่างคลาสแบบ 1- n จำนวน 3 คลาส และใช้ ในการทดสอบการสร้าง Instance ของคลาส Student ภายใต้สภาพแวดล้อมของกรอบงาน

ตารางที่ 4.3 แสดงจำนวนของ Instance และเวลาที่ใช้ในการสร้าง Instance จากคลาสที่มีความสัมพันธ์กัน 3 คลาส

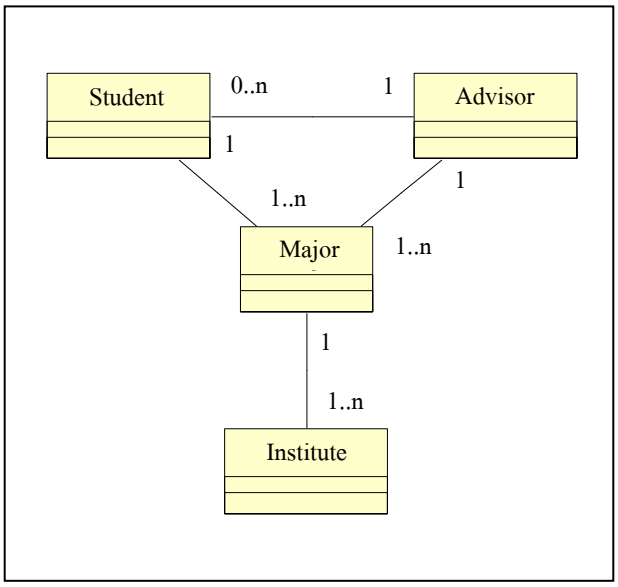
จำนวน (Instance)	เวลาที่ใช้ในแต่ละครั้งที่ทดสอบ (milliseconds)					ค่าเฉลี่ย
	1	2	3	4	5	
1000	3219	2328	2234	2234	2235	2450
5000	3516	3484	3484	3484	3485	3490.6
10000	6016	4922	4954	4969	4969	5166
20000	7953	7750	7735	7766	7718	7784.4

จากตารางที่ 4.3 นั้นสามารถแสดงเป็นภาพเส้นได้ดังต่อไปนี้



รูปที่ 4.13 แสดงกราฟความสัมพันธ์ระหว่างการแสดงผลข้อมูลนามธรรมและเวลาที่ใช้จากการสร้าง Instance ของคลาส 3 คลาสด้วยกรอบงาน

4.4.5 ทดสอบการสร้างและผูกความสัมพันธ์ของวัตถุนามธรรม 4 คลาส

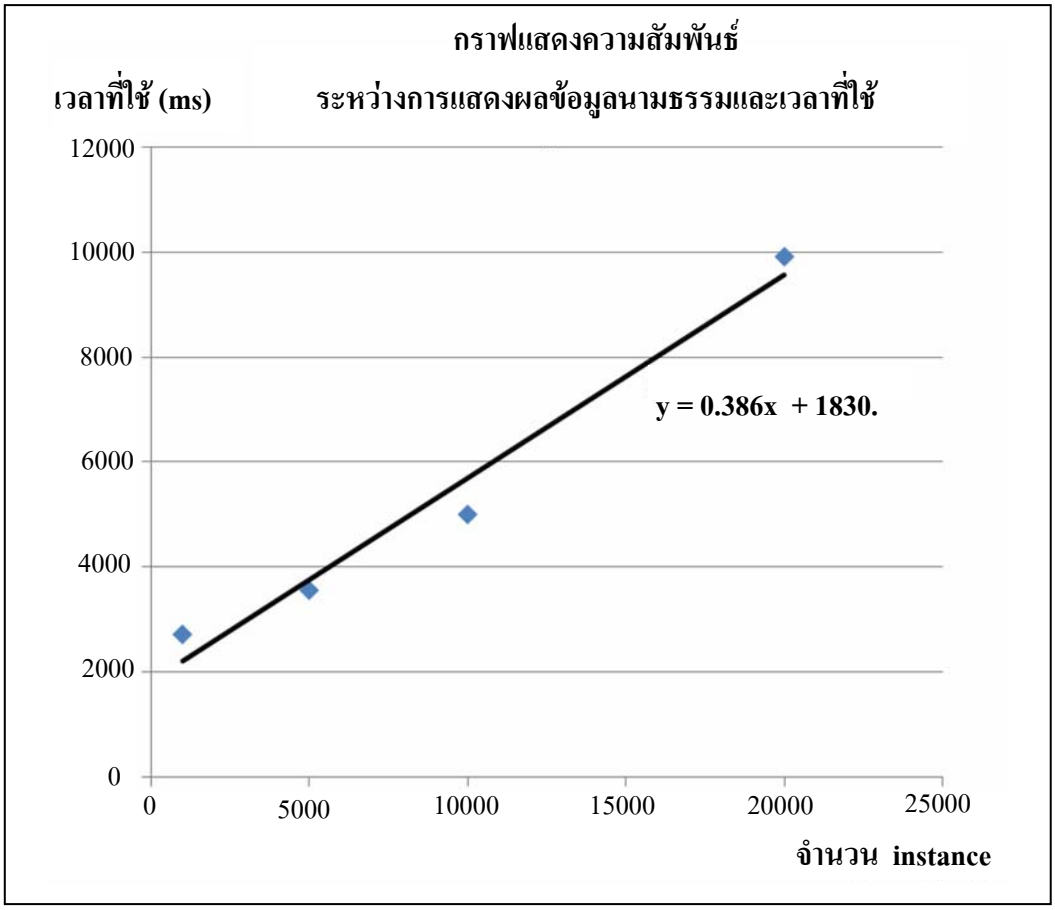


รูปที่ 4.14 แสดงแผนภาพเชิงคลาสที่ให้ความสัมพันธ์ระหว่างคลาสแบบ 1- n จำนวน 4 คลาส และใช้ในการทดสอบการสร้าง Instance ของคลาส Student ภายใต้สภาพแวดล้อมของกรอบงาน

ตารางที่ 4.4 แสดงจำนวนของ Instance และเวลาที่ใช้ในการสร้าง Instance จากคลาสที่มีความสัมพันธ์กัน 4 คลาส

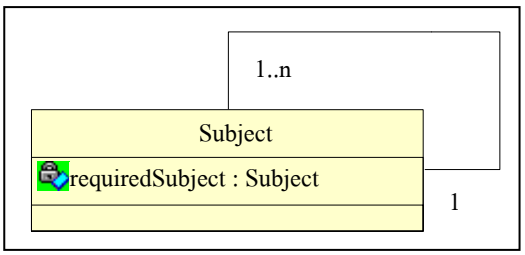
จำนวน (Instance)	เวลาที่ใช้ในแต่ละครั้งที่ทดสอบ (milliseconds)					ค่าเฉลี่ย
	1	2	3	4	5	
1000	4297	2484	2282	2281	2297	2728.2
5000	3578	3563	3532	3625	3547	3569
10000	5234	4938	4954	4984	4954	5012.8
20000	8750	10735	9969	8687	11468	9921.8

จากตารางที่ 4.4 นั้นสามารถแสดงเป็นภาพเส้นได้ดังต่อไปนี้



รูปที่ 4.15 แสดงกราฟความสัมพันธ์ระหว่างการแสดงผลข้อมูลนามธรรมและเวลาที่ใช้จากการสร้าง Instance ของคลาส 4 คลาสด้วยกรอบงาน

4.4.6 ทดสอบการสร้างและผูกความสัมพันธ์ของวัตถุนามธรรม n คลาสโดยใช้ Instance ของคลาส Student จำนวน 15000 Instance



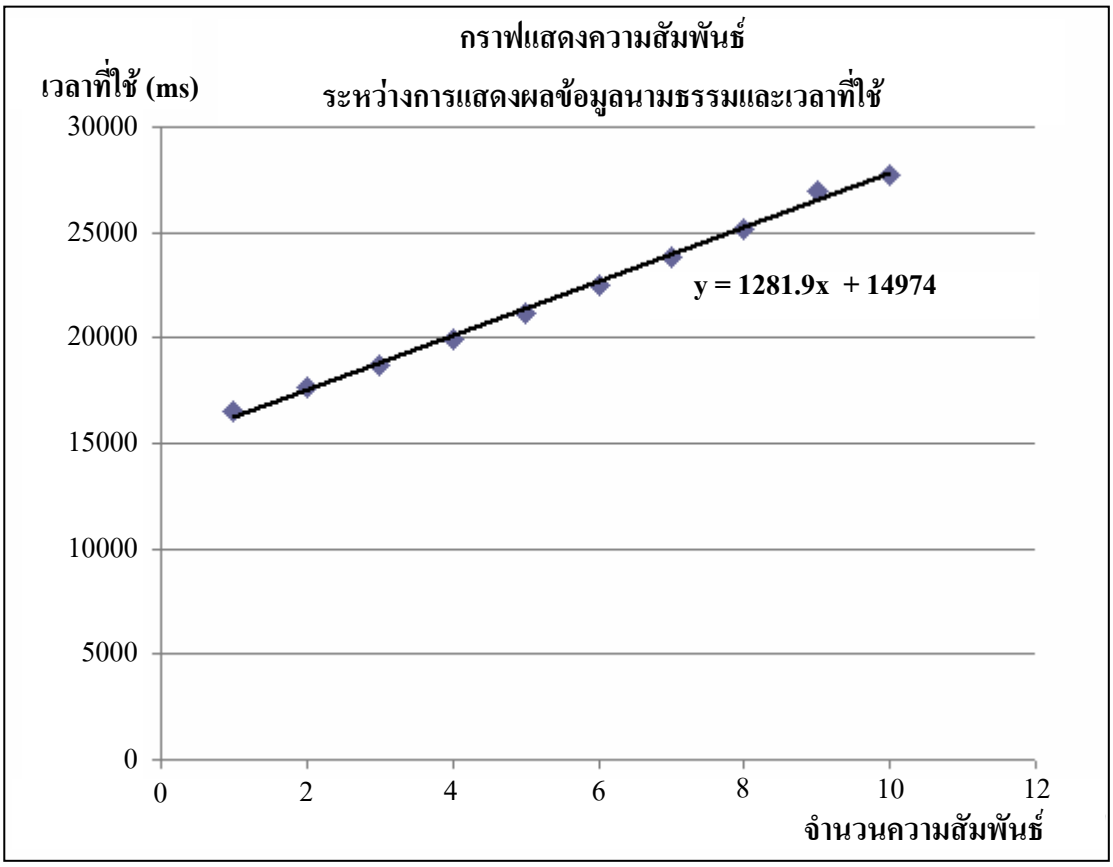
รูปที่ 4.16 แสดงแผนภาพเชิงคลาสที่ให้ความสัมพันธ์แบบ Recursive
และใช้ในการทดสอบการสร้าง Instance ของคลาส
Subject นี้ ภายใต้สภาพแวดล้อมของกรอบงาน

จากรูปที่ 4.16 นั้นคลาส Subject มีความสัมพันธ์แบบ Recursive ซึ่งมักพบเห็นได้บ่อยใน Hierarchy Pattern โดยในความหมายเชิงนามธรรมของแผนภาพเชิงคลาสนี้หมายถึงวิชาหนึ่งวิชาใดนั้นต้องการผ่านรายวิชาอื่นมาก่อนจึงสามารถลงทะเบียนเรียนในรายวิชานั้น ๆ ได้

ตารางที่ 4.5 แสดงจำนวนของ Instance และเวลาที่ใช้ในการสร้าง Instance แปรผันตาม จำนวนความสัมพันธ์ของคลาส

จำนวน ความสัมพันธ์ n คลาส	เวลาที่ใช้ในแต่ละครั้งที่ทดสอบ (milliseconds)					ค่าเฉลี่ย
	1	2	3	4	5	
ไม่มีความสัมพันธ์	15672	16062	15969	15156	15250	15621.8
1	16609	16360	16578	16516	16406	16493.8
2	17907	17734	17484	17875	17453	17690.6
3	18688	18734	18782	18735	18688	18725.4
4	19907	19953	19937	19937	19953	19937.4
5	21078	21094	21297	21109	21219	21159.4
6	22359	22578	22438	22640	22469	22496.8
7	24093	23641	23860	23922	23797	23862.6
8	25406	24890	25140	25375	24938	25149.8
9	26406	29297	26515	26437	26266	26984.2
10	27829	27766	27687	27547	27875	27740.8

จากตารางที่ 4.5 นั้นสามารถแสดงเป็นภาพเส้นได้ดังต่อไปนี้



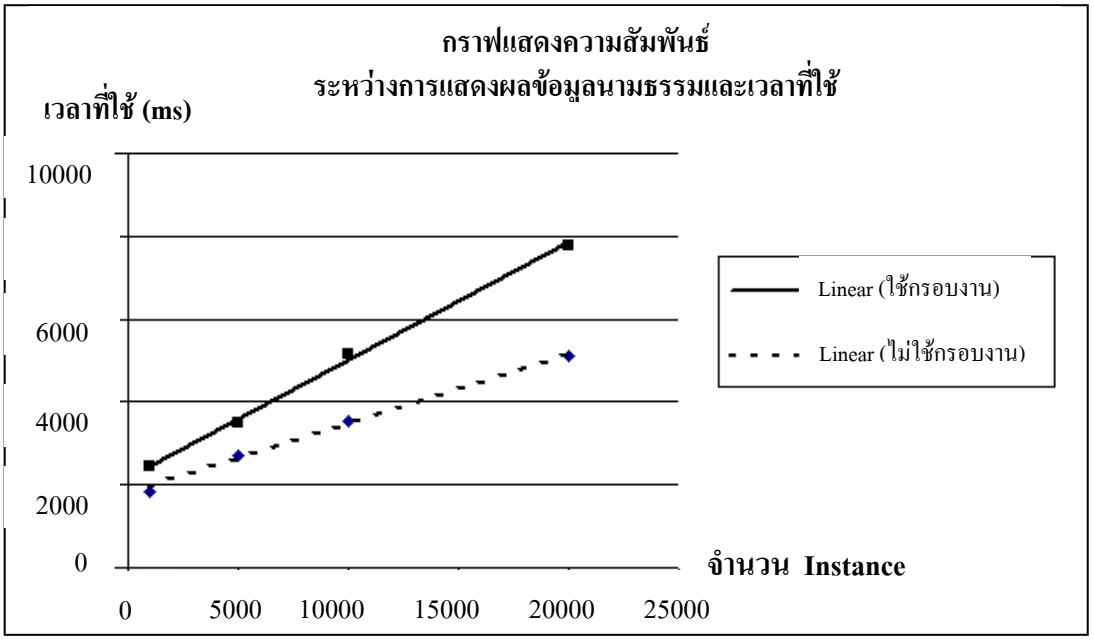
รูปที่ 4.17 กราฟแสดงจำนวนของ Instance และเวลาที่ใช้ในการสร้าง Instance แปรผันตามจำนวนความสัมพันธ์ของคลาส

4.4.7 ทดสอบการสร้างและผูกความสัมพันธ์ของวัตถุนามธรรมซึ่งมีความสัมพันธ์กัน จำนวน 3 คลาสโดยไม่ใช้กรอบงาน

ตารางที่ 4.6 แสดงจำนวนของ Instance และเวลาที่ใช้ในการสร้าง Instance จากคลาสที่มีความสัมพันธ์กัน 3 คลาสโดยไม่ใช้กรอบงาน

จำนวน (Instance)	ครั้งที่ทดสอบ					ค่าเฉลี่ย
	1	2	3	4	5	
1000	1797	1953	1813	1797	1797	1831.4
5000	2625	2625	2906	2640	2812	2721.6
10000	3735	3547	3437	3437	3453	3521.8
20000	5500	5031	5000	4984	5000	5103

จากตารางที่ 4.3 และ 4.6 สามารถนำเสนอภาพเส้นที่เปรียบเทียบการสร้างและผูกความสัมพันธ์ของวัตถุนามธรรมซึ่งมีความสัมพันธ์กันจำนวน 3 คลาสโดยใช้และไม่ใช้กรอบงาน



รูปที่ 4.18 กราฟแสดงจำนวนของ Instance และเวลาที่ใช้ในการสร้าง Instance จากคลาสที่มีความสัมพันธ์กัน 3 คลาสโดยไม่ใช้กรอบงาน

จากกราฟในรูปที่ 4.18 นั้นจะเห็นได้ว่าการใช้กรอบงานในการสร้างและผูกความสัมพันธ์ของวัตถุนามธรรมนั้นทำให้เสียเวลาในการประมวลผลมากยิ่งขึ้นเมื่อเปรียบเทียบกับการสร้างและผูกความสัมพันธ์ของวัตถุนามธรรมโดยไม่ใช้กรอบงาน โดยเวลาที่เสียไปนี้เกิดจากการอ่านค่าตัวแปรเสริมและการกำหนดค่าสภาพแวดล้อมให้กับกรอบงานเพื่อให้กรอบงานนั้นทำงานได้อย่างพลวัต

4.5 สรุปความสามารถของกรอบงาน

ในที่นี้ขอกล่าวโดยสรุปเพื่อให้เห็นความสามารถของกรอบงานที่ได้รับการพัฒนาขึ้นภายใต้งานวิจัยชิ้นนี้ ดังตารางต่อไปนี้

รายการ	สามารถทำได้	ไม่สามารถทำได้
เชื่อมโยงข้อมูลในชั้นแสดงผล, ตรรกะในการทำงานและเอนทิตีในฐานข้อมูล	X	
กำหนดค่าตัวแปรเสริมสำหรับกรอบงานผ่านเอกสาร XML	X	
เปลี่ยนแปลงส่วน Object-Relational Mapping Component อย่างพลวัต	X	
แสดงสดมภ์ของตารางข้อมูลในรูปแบบ JComponent ต่าง ๆ	X	
เพิ่มประเภทของส่วนแสดงผล	X	
ผูกความสัมพันธ์ของ Domain Object กับส่วนแสดงผลนอกตารางข้อมูล		X
แสดงข้อมูลอย่างมีเงื่อนไขซับซ้อน		X

4.6 อภิปรายผล

จากการวิเคราะห์และทดสอบเพื่อเวลาที่กรอบงานใช้ในการสร้างสภาพแวดล้อมที่เหมาะสมต่อโปรแกรมประยุกต์สามารถอภิปรายผลได้ดังต่อไปนี้

- กรอบงานลดจำนวนของรหัสคำสั่งที่ถูกเขียนเพื่อแสดงผลของตารางข้อมูล
- กรอบงานมีความค่าความยึดติดต่อการติดต่อฐานข้อมูลต่ำลง
- กรอบงานมีความยืดหยุ่นในการปรับปรุงและคงทนต่อการเปลี่ยนแปลงไปของความต้องการเชิงซอฟต์แวร์
- เวลาที่ใช้ในการกำหนดสภาพแวดล้อมให้กับโปรแกรมประยุกต์มีขนาดที่สามารถยอมรับได้
- การใช้กรอบงานในการสร้างและผูกความสัมพันธ์ของวัตถุนามธรรมนั้นทำให้เสียเวลาในการประมวลผลมากยิ่งขึ้นเมื่อเปรียบเทียบกับการสร้างและผูกความสัมพันธ์ของวัตถุนามธรรมโดยไม่ใช้กรอบงาน

บทที่ 5
บทสรุป

จากการศึกษา วิจัย และพัฒนากรอบงานเชิงวัตถุสามารถสรุปผลที่ได้รับ ปัญหา และข้อจำกัดที่พบได้ดังต่อไปนี้

5.1 สรุปผลการวิจัย

ปัจจุบันการพัฒนาโปรแกรมประยุกต์ถูกพัฒนาขึ้นเพื่อเพิ่มประสิทธิภาพให้กับกระบวนการผลิตหรือกลุ่มอาชีพต่าง ๆ จำนวนมาก การพัฒนาโปรแกรมประยุกต์ใด ๆ นั้นเป็นสิ่งจำเป็นอย่างยิ่งในการพัฒนาโปรแกรมประยุกต์ที่สนใจให้มีประสิทธิภาพในด้านต่าง ๆ อาทิเช่นความถูกต้องตามความต้องการเชิงซอฟต์แวร์ ความยืดหยุ่นและความคงทนต่อการเปลี่ยนแปลงความต้องเชิงซอฟต์แวร์ ความสวยงานและง่ายต่อการใช้ เป็นต้น โดยการพัฒนาซอฟต์แวร์ให้มีประสิทธิภาพดังกล่าวนั้นมักถูกจำกัดอยู่ภายใต้ค่าใช้จ่ายและระยะเวลาอันมีขีดจำกัด อีกทั้งโดยปกติความต้องการเชิงซอฟต์แวร์มักถูกเปลี่ยนแปลงในขณะที่โปรแกรมประยุกต์อยู่ในขั้นตอนการพัฒนาหรือได้พัฒนาจนกระทั่งเสร็จสิ้น

กรอบงานเชิงวัตถุซึ่งถูกพัฒนาขึ้นภายใต้แนวคิดและเทคโนโลยีเชิงวัตถุนั้นเปรียบเสมือนโปรแกรมประยุกต์ที่เกือบสมบูรณ์และมีความยืดหยุ่นคงทนต่อการเปลี่ยนแปลงเชิงซอฟต์แวร์ดังนั้นจึงเหมาะอย่างยิ่งในการใช้กรอบงานเพื่อสนับสนุนการทำงานของนักพัฒนาโปรแกรมประยุกต์ให้พัฒนาโปรแกรมเฉพาะทางต่าง ๆ ให้มีประสิทธิภาพภายใต้เงื่อนไขที่กำหนด

หากกล่าวโดยสรุปกรอบงานการผูกความสัมพันธ์ข้อมูลเชิงวัตถุในชั้นการแสดงผลข้อมูลนามธรรมและข้อมูลเชิงสัมพันธ์เป็นกรอบงานที่มีประสิทธิภาพสูงดังต่อไปนี้

1) กรอบงานลดจำนวนของรหัสคำสั่งที่ถูกเขียนเพื่อแสดงผลของตารางข้อมูลได้
2) กรอบงานมีค่าความยึดติดต่อการติดต่อฐานข้อมูลต่ำลง
3) กรอบงานมีความยืดหยุ่นในการปรับปรุงและคงทนต่อการเปลี่ยนแปลงไปของความต้องการเชิงซอฟต์แวร์พอสมควร
4) เวลาที่ใช้ในการกำหนดสภาพแวดล้อมให้กับโปรแกรมประยุกต์มีขนาดที่ยอมรับได้

5.2 ประโยชน์ของกรอบงานการผูกความสัมพันธ์ข้อมูลเชิงวัตถุในชั้นการแสดงผลข้อมูลนามธรรมและข้อมูลเชิงสัมพันธ์

1) เชื่อมโยงข้อมูลในชั้นแสดงผล ชั้นตรรกะในการทำงานและข้อมูลในแบบจำลองฐานข้อมูลเชิงสัมพันธ์
2) สามารถกำหนดค่าตัวแปรเสริมสำหรับกรอบงานผ่านเอกสาร XML ซึ่งทำให้คงทนต่อความต้องการเชิงซอฟต์แวร์ที่เปลี่ยนแปลงไปภายหลังการส่งมอบโปรแกรมประยุกต์ที่ถูกพัฒนาขึ้นภายใต้กรอบงานได้
3) สามารถเปลี่ยนแปลงส่วนโปแกรมทำหน้าที่ Object-Relational Mapping ได้อย่างพลวัต
4) แสดงสดมภ์ของตารางข้อมูลในรูปแบบ JComponent ต่าง ๆ
5) เพิ่มประเภทของส่วนแสดงผลอย่างพลวัต

5.3 ข้อจำกัดของกรอบงานที่สร้างขึ้น

1) กรอบงานที่พัฒนาขึ้นนี้ใช้ได้กับโปรแกรมประยุกต์ที่ถูกพัฒนาขึ้นด้วยภาษาจาวาเท่านั้น
2) กรอบงานที่พัฒนาขึ้นนี้ไม่สามารถผูกความสัมพันธ์ของ Domain Object กับส่วนแสดงผลภายนอกตารางข้อมูลได้
3) กรอบงานที่พัฒนาขึ้นนี้ไม่สามารถแสดงข้อมูลที่มีเงื่อนไขซับซ้อนได้

5.4 แนวทางในการพัฒนาต่อ

เนื่องจากในปัจจุบันการพัฒนาซอฟต์แวร์ต่าง ๆ มีแนวโน้มในการใช้ที่เปลี่ยนแปลงการวิเคราะห์และออกแบบโดยวิธีเชิงโครงสร้าง (Structured analysis approaches) ไปสู่การวิเคราะห์และออกแบบเชิงวัตถุเนื่องมาจากแนวคิดและเทคโนโลยีเชิงวัตถุนั้นสนับสนุนการนำกลับมาใช้ใหม่และมีความยืดหยุ่นอย่างสูง หากแต่ยังคงประสบปัญหาในการเปลี่ยนแปลงข้อมูลต่าง ๆ ที่อยู่ในรูปแบบที่ยากต่อการวิเคราะห์และออกแบบเชิงวัตถุเช่นข้อมูลที่อยู่ในรูปแบบเอกสาร XML ข้อมูลที่อยู่ในฐานข้อมูลเชิงสัมพันธ์หรือแม้กระทั่งข้อมูลมีลักษณะเป็น spread sheet อย่างเอกสาร Microsoft Excel ดังนั้นแล้วแนวทางการพัฒนาต่อนั้นสามารถเป็นงานวิจัยที่ศึกษาวิธีกำหนดมาตรฐานกลางของข้อมูลต่าง ๆ ที่จะเกิดขึ้นในอนาคตให้สามารถเปลี่ยนแปลงและวิเคราะห์ในเชิงวัตถุได้ สามารถเปลี่ยนแปลงแก้ไขข้อมูลเหล่านั้นโดยปราศจากการสูญเสียความสัมพันธ์

รายการอ้างอิง

Abadi, M. and Cardelli, L. (1996). **A Theory of objects**. Berlin: Springer

Arthur, J. and Azadegan, S. (2005)**.** Spring framework for rapid open source J2EE Web application development: a case study . SNPD/SAWN 2005. **Sixth International Conference on 23-25 May 2005** (pp.90 – 95)

Briand, L. C., Daly, J. W., and Wust, J. K. (1998). A Unified Framework for Coupling Measurement in Object-Oriented Systems, **Journal of IEEE Transactions on Software Engineering** (pp.91-121)

Campbell, R. H. and Islam, N. (1993). A technique for documenting the framework of an object-oriented system. **Proceeding of Object Orientation in Operating System**: p288-30

Eden, A. H. and Mens, T. (2006). Measuring software flexibility. **IEEE Proceedings - Volume 153**(No.3, pp.113 – 125)

Eliëns, A. (2000). **Principle of object oriented development**. England: Addison-Wesley.

Fayad, M. and Schmidt, D. C. (1997). Object-oriented application framework. **Communication of ACM**(pp.32-38.)

Fischer, G. (1987). Cognitive view of reuse and redesign. **IEEE Software**: 60-72

Froehlich, G., Hoover J. H., and Sorenson, P. G. (2000). Choosing an object-oriented domain framework. **ACM Computing Surveys (CSUR)**: 17

Gamma, E., Helm, R. Johnson, R., and Vlissides, J. M. (1995), **Design Patterns: Elements of Reusable Object-Oriented Software**. England:Addison-Wesley.

Ghassemi, M. D. and Mourant, R. R. (2000)**.** Evaluation of coupling in the context of Java interfaces (poster session)**. Conference on Object Oriented Programming Systems Languages and Applications Addendum** (pp.47 - 48)

Ghezzi, C., Jazayeri, M., and Mandrioli, D. (2002). **Fundamentals of Software Engineering**. New Jersy: Addison-Wesley

Gomaa, H. (2004). **Designing software product lines with UML from Use case to pattern-based software architectures**. Addison-Wesley

Greenfiled, J. and Sort, K. (2004). Software Facories: Assembling Applications with Patterns, Model, Frameworks, and Tools. **Conference on Object Oriented Programming Systems Languages and Application** (pp.6-27)

Hueni, H., Johnson, R., and Engel R. (1995). A framework for network protocol software. **Proceedings of OOPSLA'95.**

Johnson, R. E. and Foote, B. (1988). Designing reusable classes. **Journal of Object-Oriented Programming** (pp.22-35)

Krasner, G. E. and Pope, S. T. (1988). A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. **Journal of Object-Oriented Programming** (pp. 26-49)

Parsons, D., Rashid, A., Speck, A., and Telea, A. (1999). A Framework for Object Oriented Frameworks Design. **Technology of Object-Oriented Languages and Systems** (p.141)

Pfleeger, S. L. (1998). **Software Engineering**. London: Prentice-Hall International (UK) Limited.

Pree, W. (1994). Meta patterns - a means for capturing the essentials of reusable object-oriented design**. Proceedings of the ECOOP** (pp.150-162)

Sakowicz, B., Murlewski, J., Labus A., and Napieralski**,,** A. (2007)**.** JWay - Model-Driven J2EE Application Framework. **Mixed Design of Integrated Circuits and Systems**. MIXDES '07 14th International Conference (pp.703 – 706.)

Schmidt, D. C. (1997). **Applying design patterns and frameworks to develop object-oriented communication software**. Handbook of Programming Languages, Volume I, MacMillan: Computer Publishing.

Simons, A J. H. (2003). **Classification and Inheritance** [On-line] , Avaiable: http://www.jot.fm /issues/issue_2003_07/column4/

Sommerville, I. (1996). Software Process Model. **ACM Computing Survey Volume 28** (pp. 269 – 271). New York: ACM

Wegnerm, P., Stanley, B. Zdonik S. B. (1998). Inheritance as an incremental medication mechanisms or what like is and isn't like. **Procedding of the ECOOP'88** (pp.56-77)

Wikipedia. (2008a). **Object-relational mapping** [On-line]. Available: http://en.wikipedia/wiki /Object-relational_mapping

Wikipedia. (2008b). **Application Framework** [On-line]. Available: http://en.wikipedia/wiki /Application_framewrok

Xoptrope. (2008). **XUI** [On-line]. Available: http://www.xoetrope.com/wiki/HomePage

Ziemniak, P., Sakowicz, B., and Napieralski, A. (2007). Object Oriented Application Cooperation Methods with Relational Database (ORM) based on J2EE Technology. **CAD Systems in Microelectronics** (pp. 327 – 330)

ภาคผนวก ก

บทความที่ได้รับการตีพิมพ์เผยแพร่

The 4th National Conference on Computing and Information Technology

มหาวิทยาลัยราชภัฏมหาสารคาม

23 – 24 พฤษภาคม 2551

The Framework of the Data Object Binding in the Presentation Layer between Abstract Data and the Relational Data Model

Wuttipol Madsen and Pichayotai Mahatthanapiwat
Computer Engineering, Suranaree University of Technology
Maung, Nakhon Ratchasima, 30000, Thailand
Email: southwizard@hotmail.com

Abstract

The object oriented analysis and design methodology enables analysts and developers to rapidly produce complicated applications. Traditionally, domain analysis has been based on structured analysis approaches therefore problems have been occurred in presenting and storing the object oriented entities. We describe our flexible framework to bind object-oriented data on the presentation layer, object instances in the container placed in the application context through database layer without losing the object relationship. Furthermore, it allows domain application developers to present associated objects in JTable and automatically binds its properties into the table column in various JComponent, which reduces the connection to the database, and keep properties of three objects in synchronization. However our framework is constructed similar to JGoodies and Beans Binding implementation of JSR295. Its hooks are defined by XML file. Consequently, it provides less coupling. Furthermore, developer can create their own binding JComponent and OR-Mapping Utilities by implementing and plugging it into the framework's hooks.

Key Words: object-oriented framework, object binding.

1. Introduction

Recently, large number of complex applications are efficiently analyzed and designed by model-view-control methodology supported by object-oriented paradigm [1,2]. Such paradigm provides new programming concept such as encapsulation, inheritance, modularity, reusability, etc. This methodology separates system into three layers as follows: the user interface layer, the business logic layer and the data layer. Traditionally, domain analysis and design has been based on structured approaches such as data flow analysis and ER modeling. Usually, many database systems are in relational form. There are two gaps occurred when translating abstract objects in the application context in the relational database elements without losing their properties and object relations. Furthermore, it occurs when translating abstract objects in the application context into the presentation layer (user interface) like JTable or data grid. Object-relational mapping components such as Hibernate, EJB or JDO are constructed to solve first problem, however, another one still occurs. There are components such as JGoodies, JSR295 implementation and other public libraries created to solve both problems at the same time although its source code is complicated and inflexible.

We describe our flexible black box framework to support the following activities:

- To bind object data in the presentation layer, and to bean the data object in the container placed in the application context through entities on the database layer.
- To present associated objects in JTable and to bind their properties into the table columns in various JComponent forms automatically. Furthermore, their object relationships are presented in multi-cross joined table view.
- Preserve Pain Old Java Object's idea [3].
- To keep the properties of three objects synchronized.

However, our framework is constructed similar to JGoodies and Beans Binding implementation of JSR295, these framework's hooks are defined by XML files. Consequently, it provides highe

r degree of loosely coupling. Furthermore, developer can create their custom binding components and OR-Mapping Utilities

by implementing and plugging them into the framework's hooks.

2. The Flexibility of Framework

Object-oriented application frameworks are promising tools to verify proven software designs and implementation in order to reduce the cost and improve the quality of software.

A framework is reusable, "semi-complete" application that can be specialized to produce custom applications [4, 5]. In contrast to earlier Object Oriented reuse techniques based on the class libraries, frameworks are targeted at the particular business units and application domains. A framework enhances extensibility by providing explicit API, slot and hook methods [6] that allow applications to extend its stable interface. The framework extensibility is essential to ensure timely customization services and features. Based on the customizable characteristics, Objet-oriented frameworks fall into two main categories [7,9].

- **White-Box Frameworks:** Components and application developers need to know the white box framework's architecture to adapt the framework to the concrete application. The hotspots are usually limited to inheritances. When system hotspots are cleared, building white-box framework is relatively easy.
- **Black Box Frameworks:** Black box frameworks, in contrast, hide their internal structures. Users know only general framework descriptions and its hotspots rather than the detailed architecture. Using black-box framework is thus easier than using white-box framework. The black-box frameworks, however, are harder to build than the white-box framework.

Framework's hooks fall into three categories of use [7,9].

- Option: components are provided with the framework that can be easily used to gather within the overall framework architecture.
- Pattern: components are configured with parameters and/or connected together in well-defined ways.
- Open: the framework is extended in well-defined ways, generally by providing new components.

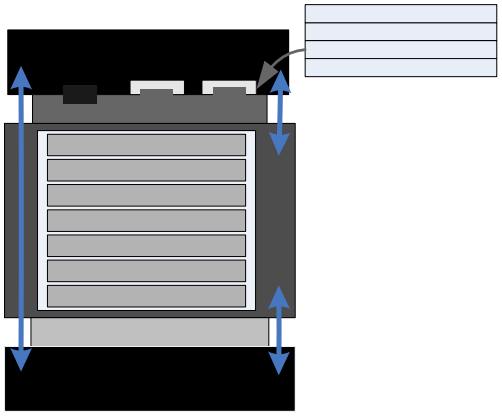


Figure 1. Framework architecture

Figure 1 demonstrates our flexible framework including the following elements:

- Binding Mechanism
- Synchronization mechanism
- Default OR-Mapping utilities
- Basic binding components
- Beans container
- JTable container
- Table Factory

Additionally, hooks are defined to allow the following components to be plugged into the framework:.

- Custom binding components.
- Concrete OR-Mapping Util.
- Domain objects.
- Configuration files.

Domain application developers can rapidly produce specific application supported by our framework. Furthermore, this architecture brings more flexible development process, less coupling design as well as cleans codes in the implementation.

Alternatively, domain application developer can collaborate to produce their domain application by directly using the Java APIs but the development time will increase and the ambiguity would occur.

3. Loosely-coupling with XML configuration

The Extensible Markup Language (XML) is a general-purpose markup language which allows its user to define their custom elements. This framework utilizes the advantages of the XML to define the presentation prototype. It brings more flexible and minimizes source code to bind abstract objects in the application context into the presentation view such as JTable. We define our schema to standardize the configuration form as shown in Figure 2.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified"
      xmlns="urn:nonstandard:mapping"
      targetNamespace="urn:nonstandard:mapping">

  <xsd:element name="table-mapping" type="Table-Mapping" />
  <xsd:complexType name="Table-Mapping">
    <xsd:sequence >
      <xsd:element name="class" type="Class" />
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Class">

    <xsd:sequence >
      <xsd:element name="property" type="Property" />
    </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="required" />
        <xsd:attribute name="table" type="xsd:string" use="required" />
        <xsd:attribute name="delible" type="xsd:boolean" use="required" />
  </xsd:complexType>

  <xsd:complexType name="Property">
        <xsd:attribute name="name" type="xsd:string" use="required" />
        <xsd:attribute name="col-name" type="xsd:string" use="required" />
        <xsd:attribute name="type" type="xsd:string" use="required" />
        <xsd:attribute name="bind" type="xsd:string" />

  </xsd:complexType>
</xsd:schema>
```

Figure 2. XML schema for binding configuration.

In the example, presentation view can be defined as Figure 3.

```
<?xml version="1.0" encoding="tis620" ?>

<table-mapping xmlns="urn:nonstandard:mapping"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:nonstandard:mapping
  file:/mapping.xsd">

  <class name="specificdomain.Student" table="student" delible="true" >
    <property name="fname" col-name="name" type="string" />
    <property name="lname" col-name="last name" type="string" />
    <property name="passed" col-name="passed?" type="boolean" />
    <property name="image" col-name="image" type="icon" bind="image" />
    <property name="${advisor.name}" col-name="advisor" type="string" />
    <property name="${advisor.major.name}" col-name="advisor's major" type="string"
  </class>

</table-mapping>
```

Figure 3. Configuration example

Defining JTable's characteristics by the XML methodology reduces line of codes in the developer's program and it is easy to reconfigure presentation layer after the domain application has been built and deployed. Additionally, it helps developers to easily represent their presentation layer's appearances.

4. Lossess domain object relationships in the presentation layer conversion

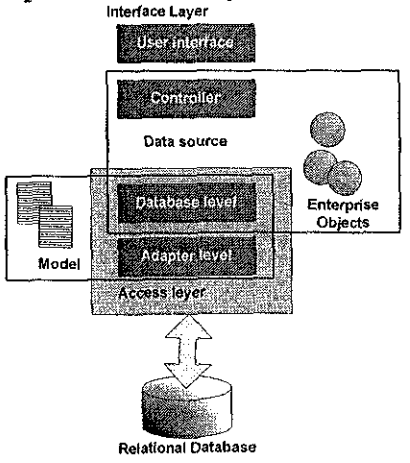
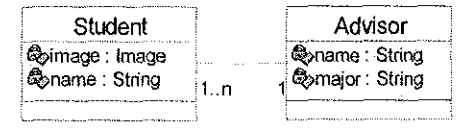


Figure 4. System architecture

Model-View-Control pattern and distributed model [8] maintain clear separation between the user interface layer, the business logic layer (business objects) and the database layer as displayed in Figure 4 .

Two gaps are occurred in storing domain objects into relational database and presenting those objects in the presentation layer (user interface) without losing their properties and object relationships. First problem is solved by OR-Mapping components such as EJB, Hibernate, and Oracle's TopLink. However, another still occurs. Figure 5 illustrates significant problem of the associate objects presentation.




```
Object name[]=new String[]{"advisor name","student name","image","major"};
DefaultTableModel model=new DefaultTableModel(name,0);
Object obj[]=new Object[4];
List<Advisor> adList=AdvisorDAO.findAll();

for(int i=0;i<adList.size();i++){
  Advisor tmp=(Advisor)adList.get(i);
  obj[0]=tmp.getName();
  obj[1]=tmp.getMajor();
  for(Student st:tmp.getStudentList()){
    obj[2]=st.getName();
    obj[3]=st.getImage();
    model.addRow(obj);
  }
}
```

Figure 5. Problem of the domain object presentation.

As the figure shown above, AdvisorDAO is easily constructed by the OR-Mapping component to preserve the object behaviors. However, developers have to spend much of their time to translate list of advisory objects and their advisee's properties to the presentation view.

Our framework is able to eliminate both problems. It provides interface or hooks for plugging custom OR-Mapping components used to resolve the mapping between the domain objects and the database data. Consequently, it is not dependent on unforeseen mapping components and the database management systems. Furthermore, our framework provides mechanism to preserve object relationships in the presentation layer. Framework's mechanism allows domain application developer to defined JTable's appearance by using Unified Expression Language.

Unified Expression Language started as part of JSTL that enables domain application developer to simply present the associated objects in the presentation layer without using multi-inner loops as showed as figure 5. In the example, presenting major of student' advisor can be defined as "${advisor.major.name}" previously shown in Figure 3.

5. Object properties synchronization

Object-oriented software designers tend to design software by using separation of concern concept. Model-View-Control is a significant pattern which is modeled to support that concept. Using separation of concern concept make benefits as follows:

-reduce complexity.

-allow the domain application developers to build multiple representations of single domain objects.

-be able to reuse views and business objects.

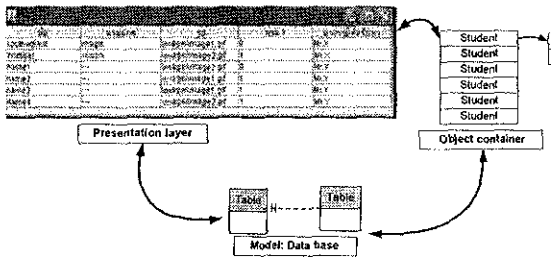


Figure 6. Model-View-Control Pattern

Figure 6 illustrates that the system is separated into the user interface layer, the business objects layer and the data model by using Model-View-Model concept. Domain objects are represented as JTable's records in presentation view, objects (instances) in beans container placed in the application context and finally objects in the database. Framework's mechanism allows three representative object's properties to keep synchronized. It provides a listener named "PropertyChangeListener" to monitor an event named "PropertyChangeEvent" fired from the source component in the presentation view when its properties are edited. Furthermore, it provides mechanism to trigger the object's properties synchronization process in the presentation view, in the objects container and in the database then object's properties in the object container are changed.

6. Framework mechanism and domain application development process

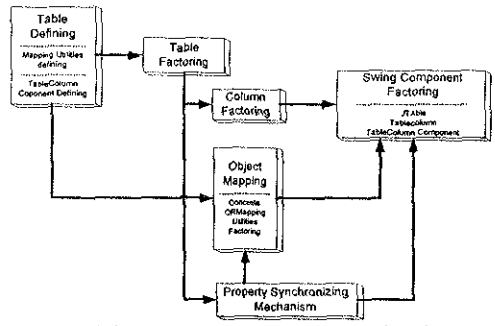


Figure 7 . Framework mechanism

Domain application developers can rapidly produce domain application by using our framework which includes the following processes.

1) Domain application developers define presentation appearances and these components:

1.1) **Object-relational mapping utilities defining:** Application developer can create their custom OR-Mapping components or use public libraries such as Hibernate3, Java Data Object (JDO), EJB3 and etc. and plug those components into the framework.

1.2) **TableColumn component defining:** Application developer can create custom JComponent such as Image, Icon, Slider bar, JTextfield, JLabel or other components which are subclass of JComponent and plug those components to the framework interface.

1.3) **Table mapping defining:** XML configuration method is used to map domain object's properties into the JTable's columns. XML schema illustrated in figure 2 is used to standardize the configuration document. As we described above, the developers can use unified expression language to access relationship link's properties.

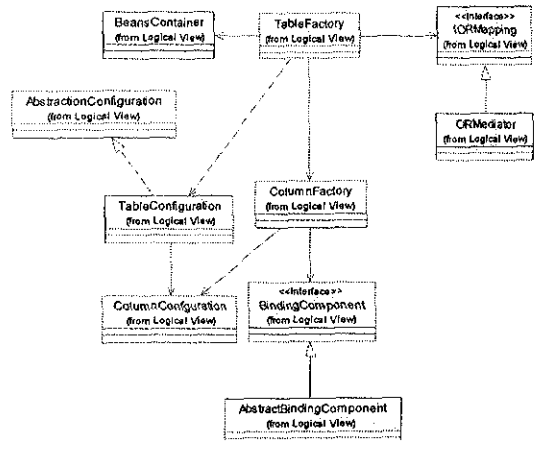


Figure 8. TableFactoring's class diagram.

2) When domain application based on our framework is executed, the presentation view represented as JTable and its characteristics are created by Table and columns factoring mechanism (its class diagram is shown as figure 8) by using configuration parameters form the first step. The listener's instances are created and live in the domain application life cycle to monitor JTable's properties.

```
DefinitionFactory definitionFactory=new DefinitionFactory();
definitionFactory.newDefinition(new File("Student.db.xml"));
definitionFactory.newDefinition(new File("TableDef.db.xml"));
IORMapping ormapping=new HibernateUtil();
TableFactory factory=new TableFactory(ormapping);
JTable table=factory.createTable("student");
```

Figure 9. Example of program's implementation.

An example of simple program based on our framework is illustrated in Figure 9. Firstly, developer must create the definition factory's instance and add configuration files as shown in Figure 3 into that instance. Secondary, customized object-relational mapping utilities are instantiated in order. Finally, the presentation layer as table can be simply provided by calling method named "createTable" from the table factory instance.

3) Framework's Object-Relational Mapping mechanism starts to bind list of database elements into the domain objects in container placed in the application context by using Object-Relational Mapping utilities and mapping parameters defined in the first step.

4) Finally, custom concrete binding components defined in the first step are created and the object's properties in the application containers are synchronized to those component models.

Figure 10 b and c illustrate information in database including list of student associated to their advisor. Figure (a) demonstrates the appearance of presentation layer as JTable embedded in JFrame. Our Framework's mechanism binds student's information in relational data model to object data in application context and present those instances on presentation layer by preserving their relationships. Two advisor's columns in Figure 10 (a) are created as the configuration in Figure 3 by object-relational and our framework's mechanism. Image's column is rendered as custom binding component in configuration file.

Summary

In the highly competitive arena of the application development, it is important to develop the applications rapidly and efficiently under time and cost constraints. Object-Oriented concepts such as reusability, polymorphism, and modularity are used in the software development process. Model-View-Controller pattern is used to separate concerns to the user interface layer, the business object layer and the database layer. Problems are occurred when translating and synchronizing the properties of the object entities in three layers without losing object relationships. Our framework is constructed to solve both gaps. Furthermore, it provides services, slots and hooks to allow domain application developer to produce the domain application rapidly. Consequently, this minimizes domain application source code, results in more flexibility and improving productivity.

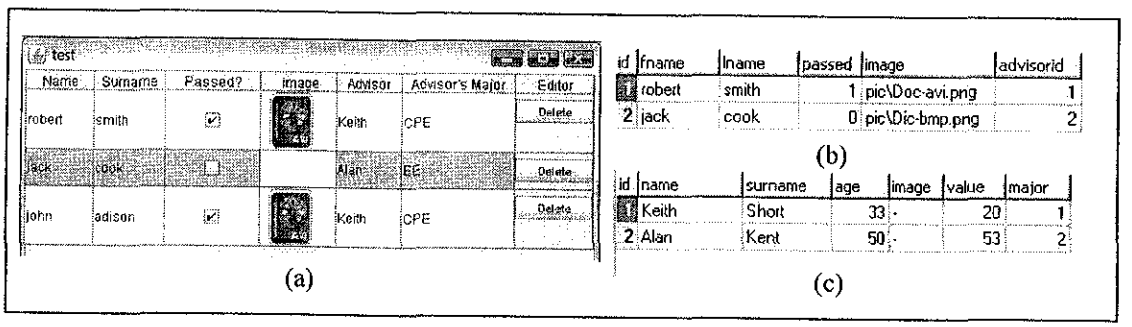


Figure 10. (a) Presentation layer 's appearrance (b) student's data in database (c) advisor's data table

References

[1] Abadi M. and Cardelli L. (1996), A Theory of objects, Spring-Verlag
[2] Anton Eliëns. (2000), Principle of object oriented development. Harlow, England: Addison-Wesley.
[3] Plain Old Java Object. Accessed on 5 January 2008. http://en.wikipedia/wiki/POJO
[4] John, R.E. and Foore, B. Designing reusable classes.
"J. Object-Oriented Programming 1", 5 (June/July 1988), 22-35
[5] Fayad, M.E., Schmidt, D.C. , and Johnson, R.E. "Object-Oriented Application Framework: Problems
and Perspectives". Wiley, NY, 1997, to appear.
[6] Pree, W. "Design Patterns for Object-Oriented Software Development". Addi-son-Wesley, Reading, Mass. 1994
[7] Garry Froehlich, H. James Hoover,Ling Liu and Paul G. Sorenson. "Choosing an Object-Oriented Domain Framework". ACM computing surveys. Vol 32. No1. March 2000.
[8] Charly Kleissner, (1995). "Enterprise Object™ Framework : A Second Generation Object-Retional Enalbler ". ACM proceeding.
[9] David Parsons, Awais Rashid, Andreas Speck, Alexandru Telea, (1999). "A framework for object oriented frame work design", ACM proceeding.

ภาคผนวก ข

รหัสคำสั่ง

1. คลาส ApplicationContext

```
package bindingframework.core;
import bindingframework.exception.NonConfigurationException;
import bindingframework.ormapping.IORMapping;
import java.io.File;
import java.util.HashMap;
import java.util.List;
import javax.swing.JTable;
import specificdomain.HibernateUtil;

public class ApplicationContext {
    private static HashMap<String,ApplicationContext> map;
    static{
            map==new HashMap<String, ApplicationContext>();
    }
    private TableFactory factory;
   private  static void putContext(String key,ApplicationContext context){
       map.put(key, context);
    }
     public   static void putContext(String key,TableFactory factory){
       map.put(key, new ApplicationContext(factory));
     }
    public  static ApplicationContext getContext(String key){
       return map.get(key);
    }
    public ApplicationContext(String appId){
          this(appId,new DefaultORMappingUtil());
    }
```

```
    public ApplicationContext(TableFactory factory){
        this.factory=factory;
    }
     public ApplicationContext(String appId,IORMapping orUtility){
        factory=new TableFactory(appId,orUtility);
     }
     public TableFactory getFactory(){
         return factory;
    }
    public static TableFactory getFactory(String key){
        return map.get(key).getFactory();
    }
     public  List getBeanList(Class key ){
        return factory.getBeanList(key);
    }
     public void synchronize(Class clazz){
        factory.updateBeans(clazz);
        factory.refreshAllTable();
    }
    public JTable createTable(String name) throws NonConfigurationException{
        return factory.createTable(name);
    }
    public static void newDefinition(File file){
        DefinitionFactory.getInstance().newDefinition(file);
    }
   public void addBindingComponent(String name,IBindingComponent component){
        ComponentFactory.addBindingComponent(name, component);
    }

}
```

2. คลาส TableFactory

```
package bindingframework.core;
import bindingframework.core.parser.TableDefinitions;
import bindingframework.exception.NonConfigurationException;
import bindingframework.ormapping.IORMapping;
import java.util.List;
import javax.swing.JTable;

public class TableFactory {
    private BeansContainer container = null;
    private TableContainer tc = new TableContainer();
    private Binder binder;
    private IORMapping orUtility;
    public TableFactory(String factoryId) {
        this(factoryId,new DefaultORMappingUtil());
    }
   public TableFactory(String factoryId,IORMapping orUtility){
        this.orUtility =orUtility;
        container = new BeansContainer();
        binder=new Binder(this);
        ApplicationContext.putContext(factoryId, this);
    }
    public TableContainer getTableContainer(){
        return tc;
    }
    public void setOrUtility(IORMapping orutil) {
        this.orUtility = orutil;
    }
```

```
public JTable createTable(String tableName) throws NonConfigurationException {
    TableDefinitions def = DefinitionContainer.getDefinition(tableName);
    TableConfiguration conf = new TableConfiguration(def);
    JTable table = null;
    if (tc.containsKey(tableName)) {
        JTable bufferTable = tc.get(tableName);
        table = new JTable(bufferTable.getModel(), bufferTable.getColumnModel());
    } else {
        table = conf.getTable();
        Class clazz=Utilities.loadClass(def.getClassName());
        loadBeans(clazz);
        binder.bindModel(table);
        binder.setChangeListener(table, def);
        tc.put(tableName, table);
   }
    return table;
}
 private void loadBeans(Class key){
    if(!container.hasBean(key)){
        List buff=orUtility.findAll(key);
        container.addBeanList(key, buff);
    }
}
 public  List getBeanList(Class key){
    if(container.hasBean(key)){
        return container.getBeanList(key);
    }else{
        return null;
    }
}
```

```
    public void removeBean(Object obj){
        orUtility.delete(obj);
    }
    public BeansContainer getBeansContainer() {
        if (container == null) {
            return new BeansContainer();
        } else {
            return container;
        }
    }
    public IORMapping getOrUtility(){
        return orUtility;
    }
    public void refreshAllTable(){
        binder.refreshAllTable();
    }
    public void updateBeans(Class clazz){
        List list=container.getBeanList(clazz);
        for(int i=0;i<list.size();i++){
            Object o=list.get(i);
            orUtility.update(o);
        }
    }
}
```

2. คลาส Binder

```
package bindingframework.core;
import bindingframework.core.parser.PropertyDefinitions;
import bindingframework.core.parser.TableDefinitions;
import bindingframework.ormapping.IORMapping;
import java.util.ArrayList;
import java.util.List;
import java.util.Vector;
import javax.swing.JTable;
import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;
import javax.swing.table.DefaultTableModel;

public class Binder {
    private IORMapping orUtility;
    private TableFactory factory;
    public Binder(TableFactory factory) {
        this.factory = factory;
        orUtility = factory.getOrUtility();
    }
```

```
public void bindModel(JTable table) {
    BindedTableModel model = (BindedTableModel) table.getModel();
    TableDefinitions def = model.getTableDefinition();
    Class clazz = Utilities.loadClass(def.getClassName());
    List list = factory.getBeansContainer().getBeanList(clazz);
    List<IBindedMethod> bmethodList = new ArrayList<IBindedMethod>();
    for (PropertyDefinitions pdef : def.getPropertiesList()) {
        IBindedMethod bmethod = BindedMethodFactory.createMethod(pdef);
        bmethodList.add(bmethod);
        model.addBindedMethod(bmethod);
    }
    for (int i = 0; i < list.size(); i++) {
        Object ob = list.get(i);
        Vector data = new Vector();
        for (IBindedMethod x : bmethodList) {
            data.add(x.getValue(ob));
        }
        model.addRow(data);
    }
}
public void setChangeListener(JTable table, TableDefinitions def) {
    DefaultTableModel model = (DefaultTableModel) table.getModel();
    model.addTableModelListener(new ModelListener());
}
public void refreshAllTable(){
    factory.getTableContainer().refreshAllTable(this);
}
class ModelListener implements TableModelListener {
    public ModelListener() {
}
```

```
        public void tableChanged(TableModelEvent e) {
            Class clazz = null;
            if (e.getType() == e.UPDATE) {
                BindedTableModel model = (BindedTableModel) e.getSource();
                TableDefinitions def = model.getTableDefinition();
                clazz = Utilities.loadClass(def.getClassName());
                List list = factory.getBeanList(clazz);
                int row = e.getFirstRow();
                int col = e.getColumn();
                Object target = list.get(row);
                Object data = model.getValueAt(row, col);
                IBindedMethod bmethod = model.getBindedMethodList().get(col);
                bmethod.setValue(target, data);
                orUtility.update(target);
                refreshAllTable();
            }
        }
    }
}
```

3. คลาส TableConfiguration

```
package bindingframework.core;
import bindingframework.core.parser.PropertyDefinitions;
import bindingframework.core.parser.TableDefinitions;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.EventObject;
import java.util.List;
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JTable;
import javax.swing.event.CellEditorListener;
import javax.swing.table.DefaultTableColumnModel;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableCellEditor;
import javax.swing.table.TableCellRenderer;
import javax.swing.table.TableColumn;

public class TableConfiguration extends AbstractConfiguration{
    private List<ColumnConfiguration> columnList;
    private int columnNum;
    private int rowNum;
    private TableDefinitions def;
    private DefaultTableModel model;
    private DefaultTableColumnModel cModel;
```

```
    public TableConfiguration(TableDefinitions def) {
        this.def=def;
        rowNum=0;
        columnNum=def.getPropertiesList().size();
        columnList=new ArrayList<ColumnConfiguration>();
        model=new BindedTableModel(0,columnNum,def);
        cModel=new DefaultTableColumnModel();
    }
    public JTable getTable(){
        JTable table=new JTable(model,cModel);
        int i=0;
        for(PropertyDefinitions px:def.getPropertiesList()){
            ColumnConfiguration conf=new ColumnConfiguration(px,i++);
            TableColumn column=conf.getTableColumn();
            cModel.addColumn(column);
        }
       if(def.isDelible()){
            TableColumn delCol=new TableColumn();
            delCol.setHeaderValue("Editor");
            delCol.setCellRenderer(new DeleteRender());
            delCol.setCellEditor(new DeleteEditor());
            delCol.setMaxWidth(80);
            cModel.addColumn(delCol);
       }
        return table;
    }
}
```

4. คลาส ColumnConfiguration

```
package bindingframework.core;
import bindingframework.core.parser.PropertyDefinitions;
import bindingframework.exception.NonConfigurationException;
import javax.swing.table.TableColumn;

public class ColumnConfiguration{
   private PropertyDefinitions pdef;
   private TableColumn column;
   private int columnIndex;
   private String bindingType;
   public ColumnConfiguration(PropertyDefinitions pdef,int i) {
      this.pdef=pdef;
      this.columnIndex=i;
      bindingType=pdef.getBindingType();
      column=new TableColumn(columnIndex); }
      public TableColumn getTableColumn(){
      column.setHeaderValue(pdef.getHeader());
      if(bindingType!=null){
         try {
            IBindingComponent component =
ComponentFactory.createBindingConmponent(bindingType);
            column.setCellEditor(component.getTableCellEditor());
            column.setCellRenderer(component.getTableCellRenderer());
         } catch (NonConfigurationException ex) {
            System.out.println(ex.getMessage());
         }
      }
      return column;
   }
}
```

5. คลาส DefaultTableMappingUtil

```
package bindingframework.core;
import bindingframework.ormapping.IORMapping;
import java.util.List;
import org.hibernate.*;
import org.hibernate.cfg.*;

public class DefaultORMappingUtil implements IORMapping {
   private static final SessionFactory sessionFactory;
   private static final Session session;
   static {
      try {
         sessionFactory = new Configuration().configure().buildSessionFactory();
         session = sessionFactory.openSession();
      } catch (Throwable ex) {
         System.err.println("Initial SessionFactory creation failed." + ex);
         throw new ExceptionInInitializerError(ex);
      }
   }
   public static SessionFactory getSessionFactory() {
      return sessionFactory;
   }
   public static Session getSession() {
      return session;
   }
```

```
public List findAll(Class clazz) {
    Session session = DefaultORMappingUtil.getSession();
    session.beginTransaction();
    List list = session.createCriteria(clazz).list();
    return list;
}
public void insert(Object st) {
    Session session = DefaultORMappingUtil.getSession();
    session.beginTransaction();
    session.save(st);
    session.getTransaction().commit();
}
public static List findByName(String name, String tableName) {
    Session session = DefaultORMappingUtil.getSession();
    Query q = session.createQuery("from " + tableName + " st where st.fname = :name");
    q.setString("name", name);
    List list = q.list();
    session.getTransaction().commit();
    return list;
}
public void update(Object obj) {
    Session session = DefaultORMappingUtil.getSession();
    session.beginTransaction();
    session.update(obj);
    session.flush();
    session.getTransaction().commit();
}
```

```
    public void delete(Object obj) {
        try {
            Session session = DefaultORMappingUtil.getSession();
            session.beginTransaction();
            session.delete(obj);
            session.flush();
            session.getTransaction().commit();
        } catch (Exception ex) {


        }
    }
    public List findAll(Object obj) {
        return findAll(obj.getClass());
    }
    public void save(Object obj) {
        Session session = DefaultORMappingUtil.getSession();
        session.beginTransaction();
        session.save(obj);
        session.flush();
        session.getTransaction().commit();
    }
}
```

6. คลาส BindedMethodFactory

```
package bindingframework.core;
import bindingframework.core.parser.PropertyDefinitions;
public class BindedMethodFactory {
   public BindedMethodFactory(Class clazz) {

   }
   public static IBindedMethod createMethod(PropertyDefinitions pdef) {
      IBindedMethod bmethod=null;
      if (!pdef.getName().startsWith("${")) {
         bmethod = new DefaultBindedMethod(pdef);

      } else {
         bmethod = new ExpressiveBindedMethod(pdef);
      }
      return bmethod;
   }
}
```

7. คลาส DefaultBindedMethod

```
package bindingframework.core;
import bindingframework.core.parser.PropertyDefinitions;
import java.lang.reflect.Method;
import java.util.Collection;
import java.util.Set;
import java.util.logging.Level;
import java.util.logging.Logger;
import specificdomain.Main;
public class DefaultBindedMethod implements IBindedMethod {
   private PropertyDefinitions pdef;
   private Class clazz;
   public DefaultBindedMethod(Class clazz, PropertyDefinitions pdef) {
      this.pdef = pdef;
      this.clazz = clazz;
   }

   public DefaultBindedMethod(PropertyDefinitions pdef) {
      this.pdef = pdef;
      String className = pdef.getTableDefinitions().getClassName();
      this.clazz = Utilities.loadClass(className);
   }
```

```
public Object getValue(Object target) {
    String methodName = getMethodName("get", pdef.getName());
    try {
        Method methoz = clazz.getMethod(methodName);
        Object ob = methoz.invoke(target);
        return ob;
    } catch (Exception ex) {
        ex.printStackTrace();
    }
    return null;
}

private String getMethodName(String type, String name) {
    char c[] = name.toCharArray();
    c[0] = Character.toUpperCase(c[0]);
    return type + new String(c);
}

public void setValue(Object target,Object value) {
    try {
        String methodName = getMethodName("set", pdef.getName());
        Class paramType = FrameworkConfiguration.getTypes(pdef.getType());
        Method methoz = getResolvedMothod(clazz, methodName, paramType);
        Object ob = methoz.invoke(target, value);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

```
    private Method getResolvedMothod(Class target,String methodName,Class paramType){
        Method md = null;
      try {
        md = target.getMethod(methodName, paramType);
      } catch (NoSuchMethodException ex) {
        Collection<Class> lx=FrameworkConfiguration.getTypes();
        for(Class cx:lx){
          try {
            md = target.getMethod(methodName, cx);
            if(md!=null){
              return md;
            }
          } catch (Exception ex1) {

          }
        }
      } catch (SecurityException ex) {
      }

      return md;
     }

    public String getName() {
      return pdef.getName();
    }
}
```

8. คลาส ExpressiveBindedMethod

```
package bindingframework.core;
import bindingframework.core.parser.PropertyDefinitions;
import de.odysseus.el.util.SimpleContext;
import javax.el.ExpressionFactory;
import javax.el.ValueExpression;
import specificdomain.entity.Student;

public class ExpressiveBindedMethod implements IBindedMethod {
   private PropertyDefinitions pdef;
   private Class clazz;
   public ExpressiveBindedMethod(Class clazz, PropertyDefinitions pdef) {
      this.pdef = pdef;
      this.clazz = clazz;
   }
   public ExpressiveBindedMethod(PropertyDefinitions pdef) {
      this.pdef = pdef;
      String className = pdef.getTableDefinitions().getClassName();
      this.clazz = Utilities.loadClass(className);

   }
   private String getContent(String text) {
      String tmp = text.substring(text.indexOf("${") + 2);
      return "${tmp." + tmp;
   }
```

```
    public void setValue(Object target, Object value) {
        String content = getContent(pdef.getName());
        ExpressionFactory factory = new de.odysseus.el.ExpressionFactoryImpl();
        SimpleContext context = new SimpleContext();
        context.setVariable("tmp", factory.createValueExpression(target, clazz));
        ValueExpression expr = factory.createValueExpression(context, content, String.class);
        expr.setValue(context, value);
    }
    public Object getValue(Object target) {
        String content = getContent(pdef.getName());
        ExpressionFactory factory = new de.odysseus.el.ExpressionFactoryImpl();
        SimpleContext context = new SimpleContext();
        context.setVariable("tmp", factory.createValueExpression(target, clazz));
        ValueExpression expr = factory.createValueExpression(context, content, String.class);
        return expr.getValue(context);
    }
    public String getName() {
        return pdef.getName();
    }
}
```

ประวัติผู้เขียน

นายวุฒิพล หมัดเส็น เกิดเมื่อวันที่ 4 เมษายน พ.ศ. 2526 ณ อ.ท่าแพร จ.สตูล เริ่มการศึกษาระดับประถมศึกษาที่โรงเรียนวัดหนองคัน (ไจพิทยาคาร) ระดับมัธยมศึกษาที่โรงเรียนท่าใหม่ (พูลสวัสดิ์ราษฎร์นุกูล) จ.จันทบุรี และในปีการศึกษา 2549 สำเร็จการศึกษาวิศวกรรมศาสตรบัณฑิต (เกียรตินิยมอันดับ 1) สาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี จ.นครราชสีมา และได้รับทุนการศึกษาสำหรับผู้มีผลการเรียนดีเด่นเข้าศึกษาต่อในระดับปริญญาโทสาขาวิชาวิศวกรรมคอมพิวเตอร์ ณ มหาวิทยาลัยเดิม

ระหว่างศึกษาในระดับปริญญาโทได้ปฏิบัติหน้าที่เป็นผู้ช่วยสอนในรายวิชาต่าง ๆ จำนวน 4 รายวิชาดังต่อไปนี้

1. การโปรแกรมคอมพิวเตอร์ (Computer Programming)
2. เทคโนโลยีเชิงวัตถุ (Object-Oriented Technology)
3. การวิเคราะห์และออกแบบระบบงาน (System Analysis and Design)
4. วิศวกรรมซอฟต์แวร์ (Software Engineering)

และได้ทำการนำเสนอบทความเชิงวิชาการ 1 บทความเรื่อง *The Framework of the Data Object Binding in the Presentation Layer between Abstract Data and the Relational Data Model* ในการประชุมวิชาการ The 4th National Conference on Computing and Information Technology ณ มหาวิทยาลัยราชภัฏมหาสารคาม จ.มหาสารคาม และได้รับอนุมัติให้ทำวิทยานิพนธ์เรื่องกรอบงานการผูกความสัมพันธ์ข้อมูลเชิงวัตถุในชั้นการแสดงผลข้อมูลนามธรรมและข้อมูลเชิงสัมพันธ์ (FRAMEWORK OF OBJECT DATA BINDING IN PRESENTATION LAYER BETWEEN ABSTRACT DATA AND RELATIONAL DATA)

ตลอดช่วงเวลาที่เข้ารับการศึกษานั้นมุ่งมั่นศึกษาหาความรู้ต่าง ๆ ทั้งในด้านวิศวกรรมพื้นฐานและความรู้ขั้นประยุกต์เกี่ยวกับวิศวกรรมคอมพิวเตอร์ และมีความสนใจเป็นพิเศษเกี่ยวกับการพัฒนาซอฟแวร์โดยใช้เทคโนโลยีเชิงวัตถุ วิศวกรรมซอฟต์แวร์ การพัฒนากรอบงาน และภาษาเชิงลักษณะ (Aspect Oriented Programming) ซึ่งจะได้ทำการศึกษาวิจัยต่อไป