การพัฒนาโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต

นายวิโรจน์ สุจิรวรกุล

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2552

DEVELOPMENT OF A SUPPORTING FILE GENERATOR PROGRAM FOR NHIBERNATE

Mr.Wiroch Sujiraworakun

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering Program in Computer Engineer

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic Year 2009

หัวข้อวิทยานิพนธ์	การพัฒนาโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต
โดย	นายวิโรจน์ สุจิรวรกุล
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	รองศาสตราจารย์ ดร.วิวัฒน์ วัฒนาวุฒิ

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

..... คณบดีคณะวิศวกรรมศาสตร์
(รองศาสตราจารย์ ดร.บุญสม เลิศหิรัญวงศ์)

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(รองศาสตราจารย์ ดร.พรศิริ หมื่นไชยศรี)

..... อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก
(รองศาสตราจารย์ ดร.วิวัฒน์ วัฒนาวุฒิ)

..... กรรมการ
(รองศาสตราจารย์ ดร.ธาราทิพย์ สุวรรณศาสตร์)

..... กรรมการภายนอกมหาวิทยาลัย
(ดร.เดชานุชิต กตัญญูทวีทิพย์)

วิโรจน์ สุจิรวรกุล : การพัฒนาโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต. (DEVELOPMENT OF A SUPPORTING FILE GENERATOR PROGRAM FOR NHIBERNATE) อ.ที่ปรึกษาวิทยานิพนธ์หลัก : รองศาสตราจารย์ ดร.วิวัฒน์ วัฒนาวุฒิ, 109 หน้า.

วิทยานิพนธ์ฉบับนี้เป็นการพัฒนาโปรแกรมประยุกต์ทางธุรกิจขององค์กร โดยทั่วไปแล้วการปฏิบัติที่ดีที่สุดคือการใช้ประโยชน์จากกรอบงานที่มักจะกังวลกับการออกแบบเลเยอร์เพื่อให้มีความยืดหยุ่นมากที่สุดระหว่างการบำรุงรักษาโปรแกรม หนึ่งในความนิยมคือ เทคนิคการใช้งานเลเยอร์ถาวรซึ่งเป็นการแม็พแบบวัตถุเชิงสัมพันธ์ (โออาร์เอ็ม) และเอ็นไฮเบอร์เนตเป็นหนึ่งในเครื่องมือโออาร์เอ็มที่พัฒนาสำหรับภาษาซีชาร์ป

อย่างไรก็ตามการเตรียมไฟล์ประกอบสำหรับเอ็นไฮเบอร์เนตด้วยตัวเองซับซ้อนมากและเกิดข้อผิดพลาดขึ้น งานวิจัยนี้เสนอทางเลือกในการสร้างไฟล์ประกอบเอ็นไฮเบอร์เนต เอ็กซ์เอ็มแอลการตั้งค่า คลาสถาวร และเอ็กซ์เอ็มแอลในการแม็พอย่างอัตโนมัติจาก แผนภาพยูเอ็มแอลคลาสในรูปแบบเอ็กซ์เอ็มไอ ความสัมพันธ์ของคลาสส่วนมากที่นิยมถูกสนับสนุน หนึ่งต่อหนึ่ง หนึ่งต่อหลาย และหลายต่อหนึ่งผ่านทางสัญลักษณ์ตัวบ่งชี้

นอกจากนั้นไฟล์เอสคิวแอลสคริปต์ที่บรรจุชุดของนิยามข้อมูล (ดีดีแอล) จะถูกสร้างให้และตรวจสอบการสร้างสคีมาฐานข้อมูลเชิงสัมพันธ์ไปยังแผนภาพยูเอ็มแอลคลาสเดิม การทดสอบกรณีศึกษาพบว่าไฟล์ประกอบที่สร้างขึ้นเป็นไปตามความต้องการเดิมของเอ็นไฮเบอร์เนตโดยไม่มีปัญหาใด ๆ อย่างไรก็ตามผลการทดสอบยังแสดงให้เห็นกระบวนการสร้างใช้เวลาเพิ่มขึ้นเนื่องจากรายละเอียดของคลาสในรูปแบบเอ็กซ์เอ็มไอ

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต วิโรจน์ สุจิรวรกุล

สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่ออ.ที่ปรึกษาวิทยานิพนธ์หลัก วิวัฒน์ วัฒนาวุฒิ

ปีการศึกษา 2552

4970574321 : MAJOR COMPUTER ENGINEERING

KEYWORDS : ORM / NHibernate / Data Service Layer / Persistent Class / SQL Generator

WIROCH SUJIRAWORAKUN : DEVELOPMENT OF A SUPPORTING FILE GENERATOR PROGRAM FOR NHIBERNATE. THESIS ADVISOR : ASSOC. PROF. WIWAT VATANAWOOD, Ph.D., 109 pp.

This thesis aims to develop the common enterprise business applications, the best practice is to exploit application framework which typically concerns the layered design in order to gain the most flexibility during application maintenance. One of the popular Persistence Layer implementation techniques is the object-relational mapping (ORM) and NHibernate are one of ORM tools developed for C#.

However, to prepare the NHibernate assembly files manually is very complicated and error-prone. This paper proposes an alternative to generate NHibernate assembly files – XML configuration file, Persistent Class files and XML Mapping files, automatically right away from UML Class Diagram in XMI format. The mostly used class relation constraints are supported – one-to-one, one-to-many and many-to-one relation between classes, via multiplicity symbols.

Moreover, SQL script file containing the set of data definition language (DDL) is generated as well to provide and ensure the creation of the consistent relational database schema to the original UML Class Diagram. The test cases show that the generated assembly files are conform to the original NHibernate requirements without any problem. However, The results also show that the generating process consumes more time due to the details of class description in XMI format.

Department : Computer Engineering Student's Signature .....

Field of Study : Computer Engineering Advisor's Signature .....

Academic Year : 2009

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยความช่วยเหลืออย่างยิ่งของอาจารย์ที่ปรึกษาวิทยานิพนธ์ รองศาสตราจารย์ ดร.วิวัฒน์ วัฒนาวุฒิ ซึ่งท่านได้แนะนำ และให้ข้อคิดเห็นต่าง ๆ ในการวิจัยด้วยดีมาตลอด รวมทั้งตรวจแก้วิทยานิพนธ์ฉบับนี้อย่างละเอียด ผู้วิจัยขอกราบขอบพระคุณในความกรุณาจากอาจารย์เป็นอย่างสูง รวมถึงคณาจารย์ภาควิชาวิศวกรรมคอมพิวเตอร์ทุกท่าน ที่ประสิทธิประสาทวิชาความรู้ให้ผู้วิจัย

ขอขอบคุณ รองศาสตราจารย์ ดร.พรศิริ หมื่นไชยศรี ประธานกรรมการสอบวิทยานิพนธ์ รวมถึงกรรมการสอบอีกสองท่านได้แก่ รองศาสตราจารย์ ดร.ธาราทิพย์ สุวรรณศาสตร์ และ ดร.เดชานุชิต กตัญญูทวีทิพย์ ที่ช่วยสละเวลามาช่วยตรวจสอบ ดำเนินการสอบ และแก้ไขวิทยานิพนธ์ฉบับนี้ให้สมบูรณ์

ท้ายนี้ผู้วิจัยขอกราบขอบพระคุณบิดา มารดา และทุกคนที่คอยสนับสนุนในด้านการเรียน และให้กำลังใจแก่ผู้วิจัยเสมอจนสำเร็จการศึกษา

สารบัญ

หน้า

หน้า

สารบัญตาราง

สารบัญรูป

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

การพัฒนาโปรแกรมส่วนใหญ่เป็นการพัฒนาโปรแกรมเพื่อเชื่อมต่อกับฐานข้อมูล ผู้พัฒนาต้องทำการเขียนโปรแกรมรวมถึงต้องออกแบบและสร้างฐานข้อมูลขึ้น โดยปัจจุบันการพัฒนาโปรแกรมมีการนำแนวคิดการพัฒนาโปรแกรมเชิงวัตถุ (OOP: Object-Oriented Programming) ผู้พัฒนาโปรแกรมใช้แนวคิดการเขียนโปรแกรมเชิงวัตถุนี้จะเริ่มพัฒนาจากแผนภาพคลาส (Class Diagram) หลังจากนั้นทำการออกแบบแผนภาพแสดงความสัมพันธ์ระหว่างเอนทิตี (ERD: Entity-Relationship Diagram) ขึ้นเพื่อทำการกำหนดความสัมพันธ์ของข้อมูล อีกทั้งเมื่อมีการแก้ไขเปลี่ยนแปลงคลาส ส่งผลให้ต้องแก้ไขฐานข้อมูลตามไปด้วย ทำให้การพัฒนาโปรแกรมเป็นไปด้วยความล่าช้า อาจเกิดข้อผิดพลาดขึ้นขณะทำการแก้ไข และประสบปัญหาเพิ่มขึ้นเมื่อเกิดการแก้ไขจำนวนมาก สาเหตุเนื่องจากการพัฒนาโปรแกรมดังกล่าวเป็นการเขียนโปรแกรมขอใช้ข้อมูลจากฐานข้อมูลโดยตรงไม่ผ่านตัวกลาง จึงได้มีการนำแนวคิดที่จะพัฒนาตัวกลางเข้ามาทำหน้าที่ในการติดต่อกับฐานข้อมูลแทน ซึ่งเลเยอร์ตัวกลางที่นำมาใช้ถูกเรียกว่าเพอร์ซิสเทนต์เลเยอร์ (Persistence Layer) ดังที่กล่าวมาทั้งหมดเป็นแนวคิดของการแม็พแบบวัตถุ-เชิงสัมพันธ์ (O/R Mapping: Object-Relational Mapping)

ปัจจุบันแนวคิดของการแม็พแบบวัตถุ-เชิงสัมพันธ์ได้รับความนิยมเพิ่มสูงขึ้น ทำให้โปรแกรมที่พัฒนาตามแนวความคิดนี้มีอย่างแพร่หลาย เช่น Hibernate, ObJect Relational Bridge (OJB) และ IBatis เป็นต้น โดยที่ยกตัวอย่างมาทั้งหมดเป็นเครื่องมือการแม็พแบบวัตถุ-เชิงสัมพันธ์ด้วยภาษาจาวา (Java) สำหรับเครื่องมือการแม็พแบบวัตถุ-เชิงสัมพันธ์ด้วยเทคโนโลยีกรอบงานดอตเนต (.NET Framework) ได้แก่ NHibernate และ TierDeveloper เป็นต้น โปรแกรมเอ็นไฮเบอร์เนต (NHibernate) คือ โปรแกรมไฮเบอร์เนตที่ทำงานบนเทคโนโลยีกรอบงานดอตเนตได้รับความนิยมสำหรับใช้งานเป็นโอ-อาร์แม็พเปอร์ (O/R Mapper) เนื่องจากมีความสามารถและประสิทธิภาพค่อนข้างสูง อีกทั้งยังเป็นซอฟต์แวร์โอเพนซอร์ส (open source) ซึ่งสามารถนำไปพัฒนาได้อย่างอิสระไม่มีข้อจำกัดทางด้านลิขสิทธิ์ จึงมีการใช้งานกันกว้างขวาง

ปัญหาของการใช้โปรแกรมเอ็นไฮเบอร์เนต คือ ผู้ใช้งานโปรแกรมเอ็นไฮเบอร์เนตต้องเตรียมไฟล์สนับสนุนการใช้งานเอ็นไฮเบอร์เนตเองทั้ง 4 ไฟล์ด้วยการสร้างไฟล์ดังกล่าวด้วยตัวเอง ซึ่งได้แก่ เอ็กซ์เอ็มแอลตั้งค่าการใช้งาน (XML Configuration), คลาสถาวร (Persistent Class), เอ็กซ์เอ็มแอลแม็พไฟล์ (XML Mapping) และเอสคิวแอลสคริปต์ การเตรียมไฟล์สนับสนุนการใช้

งานทั้ง 4 ไฟล์ผู้ใช้งานต้องมีความรู้ ความสามารถ และความเข้าใจโปรแกรมเอ็นไฮเบอร์เนตอย่างดี การเตรียมไฟล์สนับสนุนการใช้งานดังกล่าวใช้เวลาในการเตรียมมากหรือน้อยขึ้นอยู่กับขนาดของฐานข้อมูลที่ต้องการทำการแม็พ ดังนั้นหากฐานข้อมูลที่ต้องการทำการแม็พมีขนาดของฐานข้อมูลใหญ่การเตรียมไฟล์สนับสนุนดังกล่าวด้วยตัวเองเป็นการทำงานที่ไม่สะดวกอย่างมาก เนื่องจากไฟล์สนับสนุนทั้ง 4 ไฟล์จะใช้เวลาในการเตรียมค่อนข้างมากตามไปด้วย และขณะที่ทำการเตรียมไฟล์สนับสนุนทั้ง 4 ไฟล์มีโอกาสเกิดข้อผิดพลาดขณะสร้างไฟล์ขึ้น ส่งผลให้เมื่อนำไปใช้งานกับโปรแกรมเอ็นไฮเบอร์เนตเกิดข้อผิดพลาดขึ้น ผู้ใช้งานต้องทำการตรวจสอบปัญหาและแก้ไขข้อผิดพลาดดังกล่าวส่งผลให้ผู้ใช้งานเสียเวลาในการตรวจสอบและแก้ไขปัญหาเพิ่มขึ้น จากปัญหาที่กล่าวมาทั้งหมดทำให้เกิดแนวความคิดในการพัฒนาโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนตขึ้นมาแก้ไขปัญหาดังกล่าว โดยโปรแกรมที่พัฒนาขึ้นจะทำหน้าที่สร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต

งานวิจัยนี้จึงออกแบบและพัฒนาโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต เพื่อลดเวลาและข้อผิดพลาดที่เกิดขึ้นระหว่างการสร้างไฟล์สนับสนุนการใช้งานโปรแกรมเอ็นไฮเบอร์เนตทั้ง 4 ไฟล์ และนำไปใช้งานร่วมกับโปรแกรมเอ็นไฮเบอร์เนต

1.2 วัตถุประสงค์ของการวิจัย

เพื่อออกแบบและพัฒนาโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต

1.3 ขอบเขตของการวิจัย

1) ใช้เทคโนโลยีของโปรแกรมเอ็นไฮเบอร์เนตเวอร์ชัน 1.2 เป็นอย่างน้อย
2) ใช้ไฟล์เอ็กซ์เอ็มแอลในรูปแบบเอ็กซ์เอ็มไอเป็นแฟ้มข้อมูลนำเข้า (Input file)
3) แผนภาพคลาสที่รองรับการนำเข้ารองรับความสัมพันธ์ระหว่างคลาส 3 ประเภท
    3.1) ความสัมพันธ์แบบหนึ่งต่อหนึ่ง (One to One)
    3.2) ความสัมพันธ์แบบหนึ่งต่อกลุ่ม (One to Many)
    3.3) ความสัมพันธ์แบบกลุ่มต่อหนึ่ง (Many to One)
4) สร้างไฟล์สนับสนุนการใช้งานเอ็นไฮเบอร์เนต ดังนี้
    4.1) ไฟล์เอ็กซ์เอ็มแอลตั้งค่าการใช้งาน (XML Configuration)
    4.2) ไฟล์เอ็กซ์เอ็มแอลในการแม็พแบบวัตถุ-เชิงสัมพันธ์ (XML Mapping)
    4.3) ไฟล์คลาสที่ใช้ในการแม็พแบบวัตถุ-เชิงสัมพันธ์

5) ไฟล์เอ็กซ์เอ็มแอลตั้งค่าการใช้งาน (XML Configuration) รองรับฐานข้อมูลได้ 3 ชนิด คือ

   5.1) ฐานข้อมูลไมโครซอฟท์เอสคิวแอลเซิร์ฟเวอร์ (Microsoft SQL Server)

   5.2) ฐานข้อมูลมายเอสคิวแอล (MySQL Database)

   5.3) ฐานข้อมูลโพสต์เกสท์เอสคิวแอล (PostgreSQL Database)

6) สร้างไฟล์เอสคิวแอลสคริปต์ เพื่อนำไปสร้างฐานข้อมูล เอสคิวแอลสคริปต์ (SQL script) ที่สร้างขึ้นเป็นแบบดีดีแอล (DDL: Data Definition Language) โดยไม่รวมถึงการพิสูจน์ตัวตน (Authentication) ของผู้ใช้งานฐานข้อมูล เอสคิวแอลสคริปต์ที่สร้างขึ้นรองรับคำสั่งที่สำคัญ คือ

   6.1) คำสั่ง CREATE (CREATE Statement)

   6.2) คำสั่ง DROP (DROP Statement)

   6.3) คำสั่ง ALTER (ALTER Statement)

1.4 คำจำกัดความที่ใช้ในการวิจัย

โอ-อาร์เลเยอร์, เลเยอร์การให้บริการข้อมูล, คลาสถาวร, ตัวสร้างเอสคิวแอล

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1) นำไปใช้เป็นโปรแกรมเครื่องมือสำหรับสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต

2) เพื่อลดเวลาและข้อผิดพลาดที่เกิดขึ้นระหว่างการสร้างไฟล์สนับสนุนการใช้งานโปรแกรมเอ็นไฮเบอร์เนตทั้ง 4 ไฟล์ ได้แก่ เอ็กซ์เอ็มแอลตั้งค่าการใช้งาน เอ็กซ์เอ็มแอลแม็พไฟล์ คลาสถาวร และเอสคิวแอลสคริปต์

1.6 วิธีดำเนินการวิจัย

1) ศึกษาทฤษฎีการแม็บแบบวัตถุ-เชิงสัมพันธ์

2) ศึกษาวิธีการใช้งานและรูปแบบของโปรแกรมเอ็นไฮเบอร์เนต

3) ศึกษารูปแบบภาษาเอสคิวแอบของฐานข้อมูลที่โปรแกรมต้องการเชื่อมต่อ

4) ศึกษาวิธีการเขียนและใช้งานภาษาเอ็กซ์เอ็มแอล

5) กำหนดขอบเขตการทำงานของโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต

6) ออกแบบหน้าจอใช้งานและวิธีการทดสอบโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต

7) พัฒนาโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนตตามที่ได้ออกแบบ

8) ทดสอบโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนตตามขอบเขตที่กำหนดไว้

9) สรุปผลการวิจัย การทดสอบและข้อเสนอแนะ

10) จัดทำวิทยานิพนธ์ฉบับสมบูรณ์

1.7 ผลงานตีพิมพ์

งานวิจัยนี้ได้รับการตีพิมพ์ในงานประชุมวิชาการ The 6th International Joint Conference on Computer Science and Software Engineering (JCSSE2009) ในวันที่ 13-15 พฤษภาคม 2552 จัดที่จังหวัดภูเก็ต ประเทศไทย ชื่อผลงานตีพิมพ์ Generate NHibernate Assembly Files ในเอกสารประกอบการประชุมวิชาการหน้าที่ 423-427 เอกสารแสดงในภาคผนวก ก

บทที่ 2
เอกสารและงานวิจัยที่เกี่ยวข้อง

2.1 แนวคิดและทฤษฎี

2.1.1 แบบจำลองเชิงวัตถุ (Object Modeling) [1]

เป็นแบบจำลองที่ใช้อธิบายระบบในแนวคิดเชิงวัตถุ การกำหนดสาระสำคัญในการเขียนโปรแกรม ได้แก่ เอกลักษณ์ (Identity), พฤติกรรม (Behavior) และ สถานะ (State) วัตถุคือการกำหนดสาระสำคัญนอกจากแบบชนิดข้อมูลนามธรรม (Abstract Data Types) โดยที่ข้อมูล (Data) และตัวแปร (Variable) ถูกนำมารวมเป็นแนวคิดซิงเกิลยูนิไฟอิ้ง (Single Unifying) ดังที่ได้กล่าวมาข้างต้นแบบจำลองเชิงวัตถุประกอบไปด้วยหลายแนวคิด ได้แก่ การกำหนดสาระสำคัญ (Abstraction), ความเหมือนกัน (Similarity), การห่อหุ้ม (Encapsulation), การสืบทอด (Inheritance) และความเป็นโมดูล (Modularity) เป็นต้น

แบบจำลองเชิงวัตถุมีความแตกต่างจากการใช้เทคนิคแบบจำลองอื่น ๆ มากเพราะแบบจำลองเชิงวัตถุ ได้รวมเอาแนวคิดของตัวแปร และแบบชนิดข้อมูลนามธรรมเข้าไว้ในแบบชนิดตัวแปรนามธรรม (Abstract Variable Type) ซึ่งก็คือวัตถุ วัตถุมีเอกลักษณ์ สถานะและพฤติกรรม แบบจำลองเชิงวัตถุสร้างระบบลักษณะนี้ให้ กับวัตถุเพื่อให้สามารถทำแบบจำลองเชิงวัตถุได้ง่ายขึ้น และมีแนวคิดของ ชนิด (Type), การสืบทอด (Inheritance), ความสัมพันธ์ (Association) และคลาส (Class) ถึงแม้ว่าแบบจำลองเชิงวัตถุจะเป็นเพียงขั้นตอนเล็ก ๆ จากการเขียนโปรแกรมเชิงชนิดข้อมูล แต่ก็ทำให้รู้สึกแตกต่างจากการเขียนโปรแกรมแบบโครงสร้าง แบบจำลองเชิงวัตถุให้ความสำคัญไปที่เอกลักษณ์และพฤติกรรมนั่นคือความแตกต่างอย่างสมบูรณ์จากแบบจำลองเชิงสัมพันธ์ที่ให้ความสำคัญกับสารสนเทศอื่น

2.1.2 แบบจำลองเชิงสัมพันธ์ (Relational Modeling) [1]

เป็นแบบจำลองที่อธิบายระบบตามหลักตรรกะ (Logic) และความเป็นจริง (Truth Statement) ฐานข้อมูลเชิงสัมพันธ์ (Relational Database) เป็นแบบจำลองตรรกะและบอกความเป็นจริงต่าง ๆ โดยจะเก็บค่าในรูปของตาราง จากสารสนเทศข้างต้นแบบจำลองฐานข้อมูลเชิงสัมพันธ์สามารถจดจำและคืนกลับสารสนเทศต้นแบบเหมือนกับการพิสูจน์ความจริงใหม่ ผู้ออกแบบฐานข้อมูลต้องมีความมั่นใจและเข้าใจว่าต้องการจะบอกอะไรกับฐานข้อมูลและ

ฐานข้อมูลต้องการจะบอกอะไรกับผู้ออกแบบ ดังนั้นผู้ออกแบบฐานข้อมูลต้องสามารถแปลระหว่างความรู้ของมนุษย์และแบบจำลองฐานข้อมูล

แบบจำลองเชิงสัมพันธ์แตกต่างจากแบบจำลองเชิงวัตถุมาก ในหลาย ๆ ด้านความแตกต่างนั้นเป็นประโยชน์เพราะส่วนมากแบบจำลองเชิงวัตถุและแบบจำลองเชิงสัมพันธ์มีความเกี่ยวข้องซึ่งกันและกัน จึงทำให้เกิดความแตกต่างอย่างสมบูรณ์

2.1.3 การแม็พแบบวัตถุ-เชิงสัมพันธ์ (O/R Mapping: Object-Relational Mapping) [1]

แบบจำลองเชิงวัตถุอธิบายระบบโดยใช้หลักการที่สำคัญของการเขียนโปรแกรมเชิงวัตถุ ได้แก่ เอกลักษณ์ (Identity), พฤติกรรม (Behavior) และสถานะ (State) ส่วนแบบจำลองเชิงสัมพันธ์อธิบายระบบโดยใช้ความสัมพันธ์ของข้อมูล การที่ทำให้แบบจำลองเชิงสัมพันธ์สามารถสนับสนุนแบบจำลองเชิงวัตถุต้องใช้ตัวกลางเข้ามาช่วย ตัวกลางที่กล่าวถึง คือ การแม็พแบบวัตถุ-เชิงสัมพันธ์ ดังนั้นการแม็พแบบวัตถุ-เชิงสัมพันธ์จึงเป็นการนำหลักการออกแบบเชิงวัตถุมาพัฒนาให้ใช้ได้จริงบนฐานข้อมูลเชิงสัมพันธ์ โดยหลักการแม็พแบบวัตถุ-เชิงสัมพันธ์ ประกอบด้วย 4 ส่วนสำคัญ [2] ดังนี้

- เอพีไอ (API: Application Programming Interface) ชุดคำสั่งสำหรับการใช้งานคำสั่งพื้นฐานของวัตถุ ได้แก่ ซีอาร์ยูดี (CRUD: Create Read Update Delete)
- ภาษาหรือเอพีไอสำหรับการคิวรี่ (Query) โดยอ้างถึงคลาสและคุณสมบัติของคลาส (Properties of classes)
- การอำนวยความสะดวกสำหรับการแม็พเมทาดาต้า (meta data)
- เทคนิคสำหรับการอิมพลีเมนท์โออาร์เอ็ม (O/RM: Object-Relational Mapping) เช่น การออพติไมฟังก์ชัน (Optimization Function) และการตรวจสอบทรานแซคชันระหว่างวัตถุ เป็นต้น

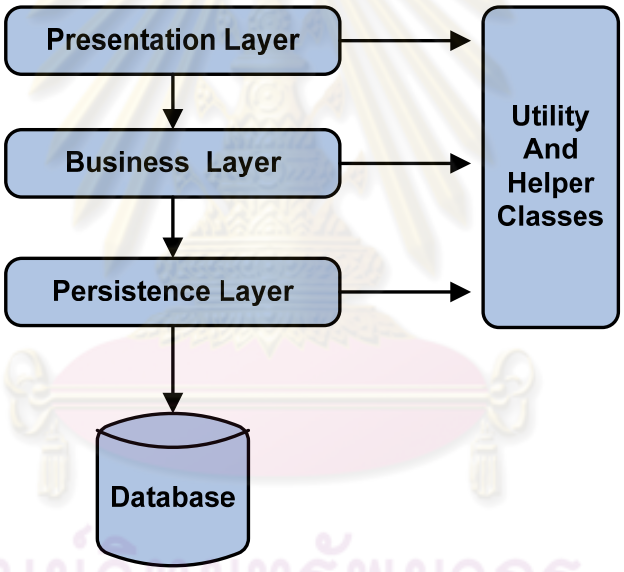
ข้อดีของหลักการแม็พแบบวัตถุ-เชิงสัมพันธ์ คือ ช่วยให้การออกแบบระบบมีความยืดหยุ่นและสามารถรองรับการเปลี่ยนแปลงในอนาคตได้ จึงเป็นการง่ายในการแม็พข้อมูลที่ต้องการให้ถูกต้อง มีประสิทธิภาพ เป็นมาตรฐานและดีกว่าการแม็พเองโดยการใช้คำสั่งเอสคิวแอล โดยตรง กระบวนการแม็พแบบวัตถุ-เชิงสัมพันธ์ต้องการความเข้าใจในเรื่องแบบจำลองเชิงวัตถุและแบบจำลองเชิงสัมพันธ์

2.1.4 สถาปัตยกรรมของเลเยอร์ (Layered Architecture) [2]

สถาปัตยกรรมของเลเยอร์ คือ อินเทอร์เฟสระหว่างการออกแบบที่ทำหน้าที่อนุญาตให้ทำการเปลี่ยนแปลงการทำงานที่ปราศจากนัยสำคัญในการกระจายของโค้ดในเลเยอร์อื่น ๆ เลเยอร์จะเป็นตัวกำหนดการพึ่งพิงระหว่างเลเยอร์ ซึ่งเป็นไปตามกฎดังนี้

- เลเยอร์ทำการติดต่อจากเลเยอร์ด้านบนลงสู่เลเยอร์ด้านล่าง และเลเยอร์จะมีการพึ่งพิงกัน (Dependency) กับเลเยอร์ด้านล่างโดยตรง
- เลเยอร์แต่ละเลเยอร์จะไม่มีผลต่อกันยกเว้นแต่เลเยอร์ด้านล่างเท่านั้น

ตามปกติแล้วโปรแกรมที่มีขนาดใหญ่เลเยอร์แต่ละเลเยอร์จะแบ่งแยกกันชัดเจน โดยปกติแล้วจะแบ่งเป็น 3 เลเยอร์ ได้แก่ เลเยอร์ที่นำเสนอข้อมูล (Presentation Layer), เลเยอร์กฎของธุรกิจ (Business Layer) และเพอร์ซิสเทนต์เลเยอร์ (Persistence Layer) ดังแสดงในรูปที่ 2.1



รูปที่ 2.1 Layered Architecture [2]

- **เลเยอร์นำเสนอข้อมูล** เป็นเลเยอร์ที่ติดต่อระหว่างผู้ใช้กับโปรแกรม ซึ่งเป็นเลเยอร์บนสุด โดยทำหน้าที่นำเสนอข้อมูลและควบคุมหน้าจอ
- **เลเยอร์กฎของธุรกิจ** เป็นเลเยอร์ที่อยู่ระหว่างเลเยอร์นำเสนอข้อมูลและเพอร์ซิสเทนต์เลเยอร์ มีทำหน้าที่ทำให้เกิดผลต่าง ๆ ตามกฎของธุรกิจหรือเป็นไปตามความต้องการของระบบ ส่วนหนึ่งจะถูกกำหนดจากผู้ใช้งาน แต่บางระบบหน้าจอการใช้งานจะ

ถูกสร้างขึ้นตามความต้องการของธุรกิจ เช่น แบบฟอร์มการสั่งซื้อสินค้า

- **เพอร์ซิสเทนต์เลเยอร์** เป็นเลเยอร์ที่ประกอบด้วยกลุ่มของคลาสและคอมโพเน้นท์ ซึ่งทำหน้าที่เข้าถึงฐานข้อมูล เพอร์ซิสเทนต์เลเยอร์อนุญาตให้แอพพลิเคชันและให้บริการสำหรับทำงานกับเซตของวัตถุที่ทำหน้าอ่านและเขียนข้อมูลลงในฐานข้อมูลเชิงสัมพันธ์ โดยเลเยอร์นี้ต้องออกแบบให้เหมาะสมกับแบบจำลองทางธุรกิจ
- **ฐานข้อมูล** ทำหน้าที่เก็บข้อมูลที่ได้รับมาตามแบบจำลองทางธุรกิจ
- **คลาสช่วยงาน** โดยปกติแอพพลิเคชันทุกแอพพลิเคชันจะมีคลาสช่วยงานที่ใช้ในการจัดการทุก ๆ เลเยอร์ในแอพพลิเคชัน เช่น คลาสที่ใช้ในการจัดการข้อผิดพลาดขณะทำการรันแอพพลิเคชัน

2.1.5 ไดนามิคลิงค์ไลบรารี (DLL: Dynamic-Link Library) [3]

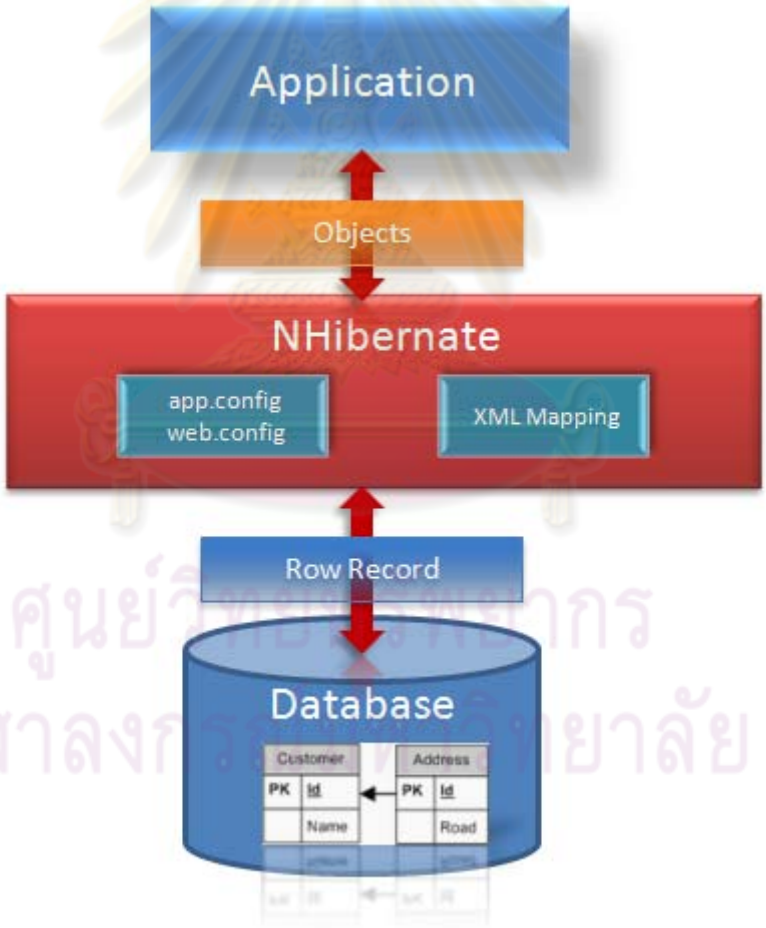
- ไดนามิคลิงค์ไลบรารีหรือดีแอลแอลเป็นโมดูลที่ประกอบไปด้วยฟังก์ชันและข้อมูลต่าง ๆ ที่สามารถถูกเรียกใช้โดยโมดูลอื่น เช่น โปรแกรมหรือดีแอลแอลไฟล์อื่น ๆ โดยจะเป็นไฟล์ที่มีนามสกุลเป็น *.dll
- ดีแอลแอลสามารถนิยามเป็นฟังก์ชันได้ 2 ประเภท คือ ฟังก์ชันแบบส่งออก (Export) และฟังก์ชันแบบภายใน (Internal) ฟังก์ชันแบบส่งออกมีจุดมุ่งหมายเพื่อถูกเรียกใช้งานโดยโมดูลอื่น ๆ เช่นเดียวกันสามารถถูกเรียกใช้งานบริเวณใดที่มีการประกาศโดยดีแอลแอลไฟล์อื่น ส่วนแบบฟังก์ชันภายในสามารถถูกเรียกใช้งานจากภายในดีแอลแอลที่ประกาศไว้เท่านั้น ถึงแม้ว่าดีแอลแอลสามารถทำการส่งออกข้อมูลได้ โดยปกติข้อมูลสามารถถูกใช้โดยเพียงฟังก์ชันเท่านั้น อย่างไรก็ตามไม่มีอะไรสามารถป้องกันการอ่านหรือเขียนข้อมูลจากโมดูลอื่นได้
- ดีแอลแอลให้บริการกับโมดูลของแอพพลิเคชันโดยฟังก์ชันสามารถทำการอัพเดทและถูกนำกลับมาใช้ใหม่ได้อย่างง่าย ดีแอลแอลสามารถช่วยลดการใช้งานหน่วยความจำเมื่อมีการเรียกใช้งานแอพพลิเคชันที่มีการทำงานซ้ำซ้อนในเวลาเดียวกัน เพราะว่าแต่ละแอพพลิเคชันได้รับการคัดลอกข้อมูลดีแอลแอลไปใช้งาน

- วินโดวส์เอพีไอ (API: Application Programming Interface) คือ กลุ่มของดีแอลแอลของวินโดวส์ที่ถูกรวมไว้เพื่อทำการเรียกใช้งาน โดยแต่ละกระบวนการของวินโดวส์เอพีไอจะเรียกใช้ดีแอลแอลเป็นส่วน ๆ ตามการใช้งาน ตัวอย่างไฟล์ดีแอลแอลที่สำคัญ เช่น kernal32.dll, shell32.dll และ user32.dll เป็นต้น

2.2 เอกสารและงานวิจัยที่เกี่ยวข้อง

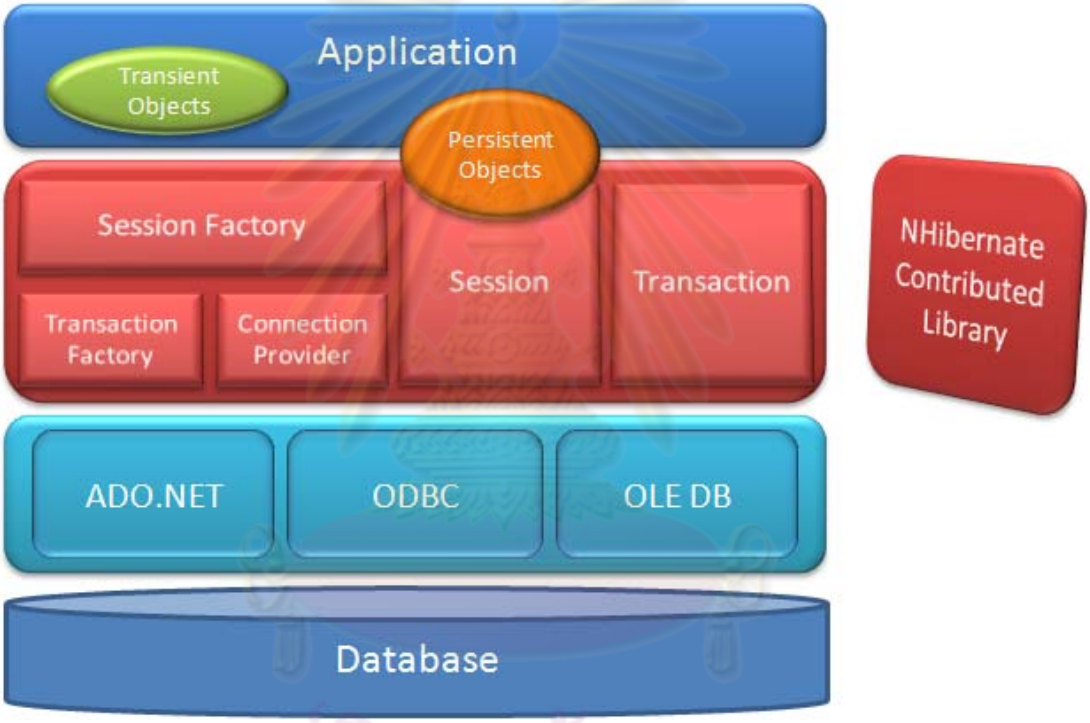
2.2.1 โปรแกรมเอ็นไฮเบอร์เนต “NHibernate Reference Documentation” [4]

โปรแกรมเอ็นไฮเบอร์เนต คือ โปรแกรมไฮเบอร์เนตที่ทำงานบนเทคโนโลยีกรอบงานดอตเนต เป็นโอ-อาร์แม็พเปอร์ (O/R Mapper) ที่มีความสามารถและประสิทธิภาพค่อนข้างสูง โดยสถาปัตยกรรมระดับสูงของเอ็นไฮเบอร์เนตแสดงในรูปที่ 2.2



รูปที่ 2.2 สถาปัตยกรรมระดับสูงของเอ็นไฮเบอร์เนต

จากรูปที่ 2.2 แสดงสถาปัตยกรรมระดับสูงของเอ็นไฮเบอร์เนต พบว่าเอ็นไฮเบอร์เนตจะทำหน้าที่เปลี่ยนลักษณะของวัตถุไปและกลับเข้ากับฐานข้อมูล แนวคิดและวิธีการของเอ็นไฮเบอร์เนตในการทำแม็พวัตถุเข้ากับตารางในฐานข้อมูลจะใช้เอ็กซ์เอ็มแอลแม็พ ลักษณะประจำหรือโปรแกรมช่วยงานอื่น เช่น Fluent NHibernate โดยผู้ใช้ต้องตั้งค่าระบุว่าคลาสใดในโดเมนของวัตถุนี้จะไปแม็พเข้ากับตารางใด และคุณสมบัตินี้จะไปแม็พเข้ากับสดมภ์ใดในตารางรวมไปถึงการกำหนดคุณสมบัติเพิ่มเติมได้ค่อนข้างยืดหยุ่น โดยในรูปที่ 2.3 แสดงสถาปัตยกรรมโดยละเอียดของเอ็นไฮเบอร์เนต



รูปที่ 2.3 สถาปัตยกรรมโดยละเอียดของเอ็นไฮเบอร์เนต

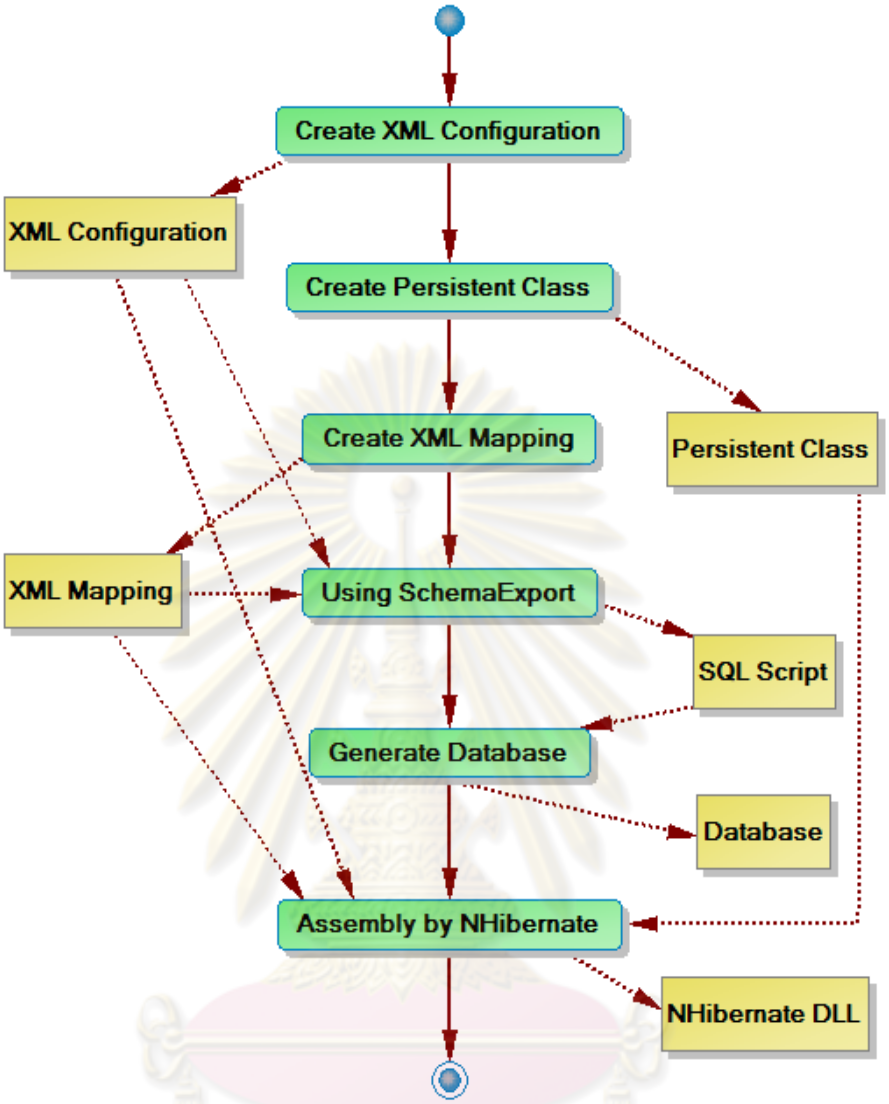
จากรูปที่ 2.3 สามารถอธิบายรายละเอียดของแต่ละส่วน คือ

1. **SessionFactory (NHibernate.ISessionFactory)** เป็นวัตถุที่ใช้เริ่มต้นการทำงานของเอ็นไฮเบอร์เนต โดยจะเปิดไฟล์แม็พและไฟล์การตั้งค่าต่าง ๆ เข้ามาตั้งต้นใช้งาน อีกทั้งยังเป็น factory สำหรับสร้าง ISession ซึ่งอาจจะมีแคชของข้อมูล ในกรณีที่มีนำข้อมูลกลับมาใช้งานในระดับของ transactions, process หรือ cluster-level.
2. **Session (NHibernate.ISession)** เป็นวัตถุที่ทำงานแบบเธรดเดี่ยว(Single Thread) มีช่วงการทำงานที่สั้น ซึ่งจะเป็นตัวที่ใช้ติดต่อระหว่างโปรแกรมกับส่วนเก็บข้อมูลของระบบ

เช่น ฐานข้อมูล เป็นต้น โดยการทำงานนั้นจะเป็นการใช้งานเอดีโอดอตเนต (ADO.NET) ในการติดต่อนั้นเอง ตลอดจนยังเป็น Factory ไว้สร้าง ITransaction เมื่อต้องการใช้งาน transaction ที่สำคัญยังมีการใช้แคชกับ Persistant Object ด้วย ช่วยให้เราสามารถทำงานกับ object graph และกรณีค้นหาวัตถุจากไอดี (ID) เป็นต้น

3. **Persistent Objects and Collections** เป็นวัตถุที่มีช่วงชีวิตสั้น ๆ โดยเป็นการทำงานของเธรดเดี่ยว(Single Thread) โดยจะเก็บค่า persistent state เอาไว้ด้วย และอาจจะรวมไปถึง business method บางตัวไว้ด้วย ซึ่งวัตถุนี้จะเป็นพิมพ์เดียวกับแบบจำลองโดเมนที่ออกแบบไว้ โดยจะคงคุณสมบัติของ POCO ไว้ได้ และที่สำคัญวัตถุนี้จะอยู่ใน Session ของ ISession และหาก Session ดังกล่าวปิดลง วัตถุเหล่านี้จะลอยตัวจากเอ็นไฮเบอร์เนตและส่งต่อไปยังเลเยอร์อื่น ๆ ได้ เช่น สามารถแม็พวัตถุไปยังดีทีโอ (DTO) เพื่อส่งต่อไปยังเลเยอร์นำเสนอข้อมูล
4. **Transient Objects and Collections** คือ วัตถุที่อยู่ในความจำของระบบ หรือวัตถุที่ถูกสร้างขึ้นใหม่ หรือเป็นวัตถุที่ไม่ได้อยู่ในการดูแลของ ISession

จากสถาปัตยกรรมของเอ็นไฮเบอร์เนตดังกล่าว ทำให้ทราบวิธีการทำงานของเอ็นไฮเบอร์เนตเบื้องต้น ดังนั้นเพื่ออธิบายให้เข้าใจถึงขั้นตอนการใช้งานเอ็นไฮเบอร์เนตเบื้องต้น จะอธิบายด้วยแผนภาพกิจกรรมในรูปที่ 2.4



รูปที่ 2.4 แผนภาพกิจกรรมแสดงขั้นตอนการใช้งานโปรแกรมเอ็นไฮเบอร์เนต

โดยขั้นตอนเริ่มต้นจาก

2.2.1.1 กิจกรรม Create XML Configuration เขียนไฟล์เอ็กซ์เอ็มแอลตั้งค่าการใช้งาน ให้เหมาะสมกับฐานข้อมูลที่ใช้งาน ตั้งชื่อไฟล์ app.config มีรายละเอียดดังนี้

- ใช้ฐานข้อมูลไมโครซอฟท์เอสคิวแอลเซิร์ฟเวอร์ (Microsoft SQL Server 2000)
- เซิร์ฟเวอร์ชื่อ = (local)
- ฐานข้อมูลชื่อ = QuickStart
- ชื่อผู้ใช้งาน = sa
- รหัสผู้ใช้ = ********

ตัวอย่างไฟล์เอ็กซ์เอ็มแอลตั้งค่าการใช้งานดังรูปที่ 2.5

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <configSections>
      <section
         name="hibernate-configuration"
         type="NHibernate.Cfg.ConfigurationSectionHandler, NHibernate" />
   </configSections>

    <hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
      <session-factory>
         <property name="dialect">NHibernate.Dialect.MsSql2000Dialect</property>
         <property name="connection.provider">NHibernate.Connection.Driver
                              ConnectionProvider
         </property>
         <property name="connection.connection_string">Server=(local);initial
                              catalog=quickstart;Integrated Security=SSPI</property>

         <mapping assembly="QuickStart" />
      </session-factory>
   </hibernate-configuration>

  <system.web>
       ..............
   </system.web>
</configuration>
```

รูปที่ 2.5 ตัวอย่างไฟล์เอ็กซ์เอ็มแอลที่ใช้ในการตั้งค่า (XML Configuration)

2.2.1.2 กิจกรรม Create Persistent Class เขียนคลาสที่ใช้ในการแม็พกับฐานข้อมูลโดยใช้ภาษาซีชาร์ปโดยตั้งชื่อไฟล์ user.cs ตัวอย่างไฟล์คลาสแสดงในรูปที่ 2.6

```
namespace QuickStart
{
   public class Cat
   {
      private string id;
      private string name;
      private char   sex;
      private float  weight;
      public Cat()
      {
      }
      public virtual string Id
      {
         get { return name; }
         set { id = value; }
      }
      public virtual string Name
      {
         get { return name; }
         set { name = value; }
      }
      public virtual char Sex
      {
         get { return sex; }
         set { sex = value; }
      }
```

รูปที่ 2.6 ตัวอย่างคลาสที่ใช้ในการแม็พ (Persistent Class)

```
public virtual float Weight
      {
         get { return weight; }
         set { weight = value; }
      }
   }
}
```

รูปที่ 2.6 ตัวอย่างคลาสที่ใช้ในการแม็พ (Persistent Class) (ต่อ)

2.2.1.3 กิจกรรม Create XML Mapping เขียนเอ็กซ์เอ็มแอลแม็พไฟล์เพื่อทำการแม็พกับฐานข้อมูล โดยตั้งชื่อไฟล์ user.hbm.xml ตัวอย่างไฟล์เอ็กซ์เอ็มแอลแม็พไฟล์แสดงในรูปที่ 2.7

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
   namespace="QuickStart" assembly="QuickStart">
   <class name="Cat" table="Cat">
      <id name="Id">
         <column name="CatId" sql-type="char(32)" not-null="true"/>
         <generator class="uuid.hex" />
      </id>
      <property name="Name">
         <column name="Name" length="16" not-null="true" />
      </property>
      <property name="Sex" />
      <property name="Weight" />
   </class>
</hibernate-mapping>
```

รูปที่ 2.7 ตัวอย่างไฟล์เอ็กซ์เอ็มแอลแม็พไฟล์ (XML Mapping)

2.2.1.4 กิจกรรม Using SchemaExport นำเอ็กซ์เอ็มแอลแม็พไฟล์มาเรียกใช้งานผ่านโปรแกรมเครื่องมือที่ชื่อว่า SchemaExport ซึ่งเป็นโปรแกรมช่วยงานของโปรแกรมเอ็นไฮเบอร์เนตที่ทำหน้าที่สร้างไฟล์เอสคิวแอลสคริปต์จากเอ็กซ์เอ็มแอลแม็พไฟล์ ตัวอย่างไฟล์แอสคิวแอลสคริปต์ที่ได้จากการใช้โปรแกรม SchemaExport ตั้งชื่อไฟล์ NHibernate.sql แสดงในรูปที่ 2.8

```
use QuickStart
go

CREATE TABLE Cat (
  CatID char(32) NOT NULL,
  Name nvarchar(16) NOT NULL,
  Sex nchar(1) default NULL,
  Weight real default NULL,
  PRIMARY KEY  (CatID)
)
Go
```

รูปที่ 2.8 ตัวอย่างไฟล์เอสคิวแอลสคริปต์ (SQL Script)

2.2.1.5 กิจกรรม Generate Database นำไฟล์เอสคิวแอลสคริปต์ที่ได้ไปสร้างฐานข้อมูลขึ้น โดยใช้คำสั่ง osql ในดอสคอมมานด์พร้อมต์

```
osql –E –i NHibernate.sql –o output.txt
```

2.2.1.6 กิจกรรม Assembly by NHibernate นำไฟล์เอ็กซ์เอ็มแอลที่ได้ทั้งสองไฟล์กับคลาสที่ใช้ในการแม็พที่ได้ และไลบรารีไฟล์ของเอ็นไฮเบอร์เนตมาทำการประกอบไฟล์โดยใช้โปรแกรมที่ชื่อว่า เอ็นแอนท์ (NAnt) โดยเริ่มต้นใช้โปรแกรมต้องมีการลงโปรแกรม Microsoft .NET Framework runtime 1.1, Microsoft .NET Framework runtime 2.0 และ Microsoft .NET

Framework SDK Version 1.1 ไว้ก่อนจึงทำการใช้งานได้ หลังจากนั้นทำการติดตั้งโปรแกรมเอ็นแอนท์ลงไป ขั้นตอนการประกอบไฟล์แสดงด้วยแผนภาพกิจกรรมรูปที่ 2.9

- กิจกรรม Extract file NHibernate-src.zip to folder ขยายไฟล์ NHibernate-src.zip ไปยังโฟลเดอร์ที่ต้องการทำการประกอบไฟล์ จะได้ไลบรารีของโปรแกรมเอ็นไฮเบอร์เนตที่ต้องการในโฟลเดอร์
- กิจกรรม Copy file to folder คัดลอกไฟล์ 3 ไฟล์ ได้แก่ ไฟล์ User.cs, ไฟล์ User.hbm.xml และไฟล์ app.config ไปยังโฟลเดอร์ที่เก็บไลบรารีของโปรแกรมเอ็นไฮเบอร์เนตในกิจกรรมแรก
- กิจกรรม Assembly with NAnt ทำการประกอบไฟล์โดยเรียกใช้ดอสคอมมานด์พร้อมต์ไปยังพาร์ทที่ทำการคัดลอกไฟล์ไว้แล้วพิมพ์คำสั่ง ดังนี้

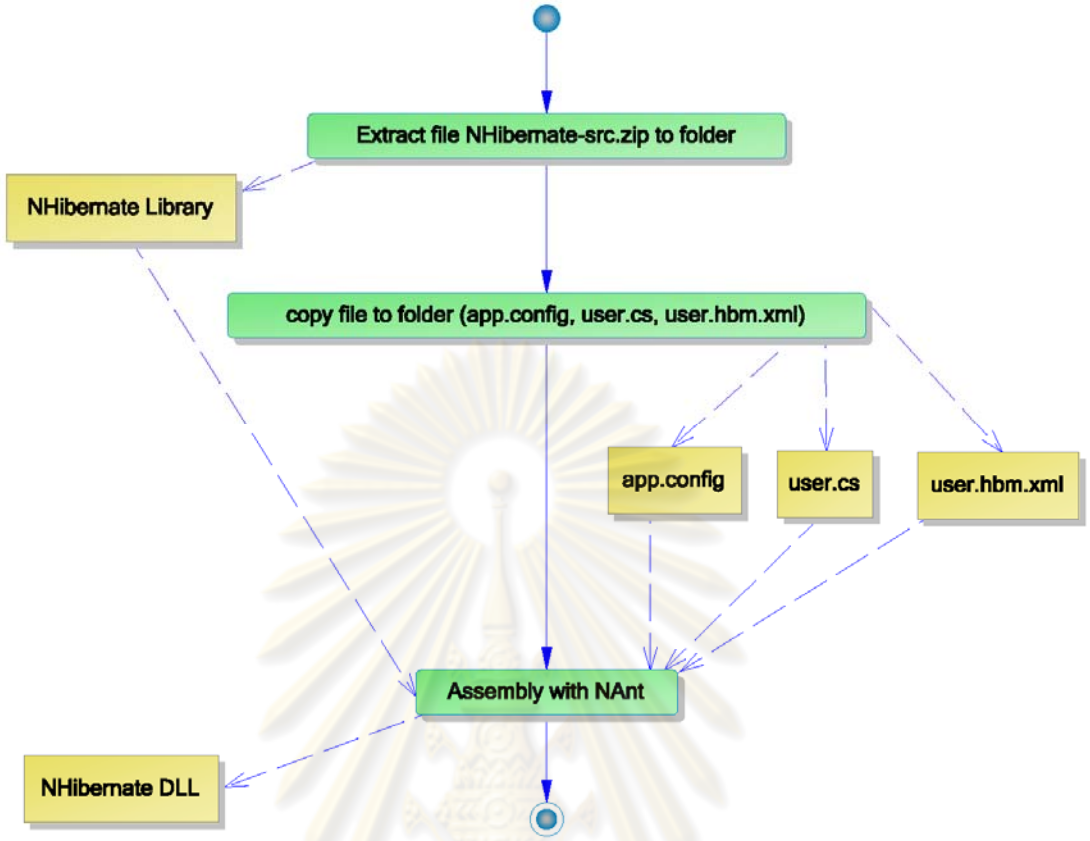
สำหรับโหมดดีบัก (Debug mode)

```
NAnt -D:project.config=debug clean build >output-debug-build.log
```

สำหรับโหมดรีลีส (Release mode)

```
NAnt -D:project.config=release clean build >output-release-build.log
```

ไฟล์ที่ได้จากการประกอบไฟล์จะอยู่ในโฟลเดอร์ที่ชื่อว่า Build ของโฟลเดอร์ที่ทำการคัดลอกไฟล์ทั้งสามไว้



รูปที่ 2.9 แผนภาพกิจกรรมแสดงการประกอบไฟล์ทั้งหมดด้วยโปรแกรมเอ็นแอนท์

ขั้นตอนทั้งหมดที่กล่าวมาเป็นขั้นตอนโดยละเอียดตั้งแต่การเตรียมไฟล์เพื่อใช้งานเอ็นไฮเบอร์เนตจนถึงการประกอบไฟล์ทั้งหมดด้วยโปรแกรมเอ็นแอนท์ จะเห็นได้ว่าการใช้งานทั้งหมดจนได้ไฟล์เอ็นไฮเบอร์เนตดีแอลแอลที่ต้องการ มีความยุ่งยาก ซับซ้อนหลายขั้นตอน ทำให้ผู้ใช้งานโปรแกรมเอ็นไฮเบอร์เนตต้องมีความรู้และความเข้าใจในการใช้งานเป็นอย่างดี รวมถึงต้องทำการเขียนโค้ดอย่างระมัดระวังข้อผิดพลาดที่อาจจะเกิดขึ้น ดังนั้นถ้าหากเกิดข้อผิดพลาดขึ้นในการเขียนโค้ดแล้วนำไปประกอบไฟล์ เมื่อมีการนำไฟล์เอ็นไฮเบอร์เนตไปเรียกใช้งานในแอพพลิเคชันอื่น ๆ หากเกิดข้อผิดพลาดขึ้นต้องทำการตรวจสอบหาข้อผิดพลาดหลายจุดด้วยกัน ตั้งแต่ข้อผิดพลาดที่อาจจะเกิดขึ้นจากตัวแอพพลิเคชันที่เรียกใช้เอง หรือข้อผิดพลาดที่เกิดขึ้นจากเขียนโค้ดในไฟล์ต่าง ๆ ที่ใช้ในประกอบไฟล์เป็นเอ็นไฮเบอร์เนตดีแอลแอล ซึ่งการตรวจสอบหาข้อผิดพลาดดังกล่าวใช้เวลาตรวจสอบมาก จึงเป็นสาเหตุหลักที่เป็นปัญหาของการใช้โปรแกรมเอ็นไฮเบอร์เนต

2.2.2 งานวิจัยเรื่อง “JWay-Model driven J2EE application framework” [5] โดย Sakowicz และคณะ ปี 2007

งานวิจัยนี้เป็นการพัฒนากรอบงานสำหรับการเขียนโปรแกรมด้วยภาษาจาวา จุดประสงค์ของงานวิจัยนี้คือ การพัฒนากรอบงานเพื่อสร้างเครื่องมือที่ทำให้เกิดการติดต่อได้อย่างต่อเนื่องกับไคล์เอ็นต์ (Client) JWay เป็นเว็บเทมเพลทแบบเอ็มวีซี (MVC: Model View Controller) ถูกออกแบบและเขียนด้วยภาษาจาวาโดยเป็นซอฟต์แวร์โอเพนซอร์ส กรอบงานและโปรแกรมที่ถูกใช้เป็นเทมเพลทถูกสร้างด้วยโปรแกรม Maven โมดูลโปรแกรมช่วยสร้างโค้ดถูกสร้างจาก XDoclet ในงานวิจัยนี้ขอเรียกว่า XWay โดย XWay ทำหน้าที่เหมือนกับดีเอโอ (DAO: Data Access Object) ส่วนกรอบงานของเพอร์ซิสเทนต์ที่ใช้ใน JWay คือ Hibernate ผลของงานวิจัยพบว่า JWay ประสบความสำเร็จตามจุดประสงค์คือ สามารถทำการติดต่ออย่างต่อเนื่องกับไคล์เอ็นต์ได้เป็นอย่างดี นอกจากนี้พบว่า Hibernate ได้รับความนิยมและมีประสิทธิภาพ จึงถูกนำมาใช้เป็นกรอบงานเพอร์ซิสเทนต์ แต่งานวิจัยที่ได้กล่าวมาการออกแบบและพัฒนากรอบงานยังอยู่ในส่วนของเทคโนโลยีของภาษาจาวาเท่านั้น

2.2.3 งานวิจัยเรื่อง “Object Oriented Application Cooperation Methods with Relational Database (ORM) based on J2EE Technology” [6] โดย Ziemniak และคณะ ปี 2007

งานวิจัยนี้เป็นการทดสอบประสิทธิภาพการทำงานเทคโนโลยีเพอร์ซิสเทนต์ทั้ง 3 เทคโนโลยี ได้แก่ JPA (Java Persistence API), JDO (JPOX: Java Persistent Objects) และ Hibernate โดยในการทดสอบใช้ ฟังก์ชันต่าง ๆ ของกรอบงานสปริง (Spring Framework) ได้แก่

- การทำการตั้งค่าโปรแกรมที่ใช้จาวาบีน (JavaBeans) ผ่าน Inversion-Of-Control
- เลเยอร์การนำเสนอข้อมูลเว็บตามแบบจำลองเอ็มวีซี (MVC: Model View Controller)
- ทดสอบการใช้งานร่วมกับฐานข้อมูล
- การบริหารจัดการทรานแซคชันโดยใช้เอโอพี (AOP: Aspect Oriented Programming)

ผลจากการทดสอบโดยใช้โปรแกรมตัวอย่างพบว่า JPA เป็นเทคโนโลยีที่มีสมรรถนะมากที่สุด และ Hibernate เป็นเทคโนโลยีที่ทรงพลังมากที่สุด และสุดท้าย JPOX เป็นเทคโนโลยีที่ใช้งานยากและผลที่ได้ไม่น่าพอใจเท่าที่ควร จากงานวิจัยนี้ทำให้เห็นว่าเทคโนโลยี JPA และ Hibernate เป็นเทคโนโลยีที่น่าสนใจนำมาใช้งาน เนื่องจากเทคโนโลยีทั้งสองเป็นเทคโนโลยีของภาษาจาวา แต่เทคโนโลยี Hibernate มีส่วนที่ทำงานกับกรอบงานดอตเนต (.NET Framework) ด้วยซึ่งถูกเรียกว่า NHibernate ผู้วิจัยจึงตัดสินใจเลือก NHibernate มาใช้งานในการพัฒนางานวิจัยนี้

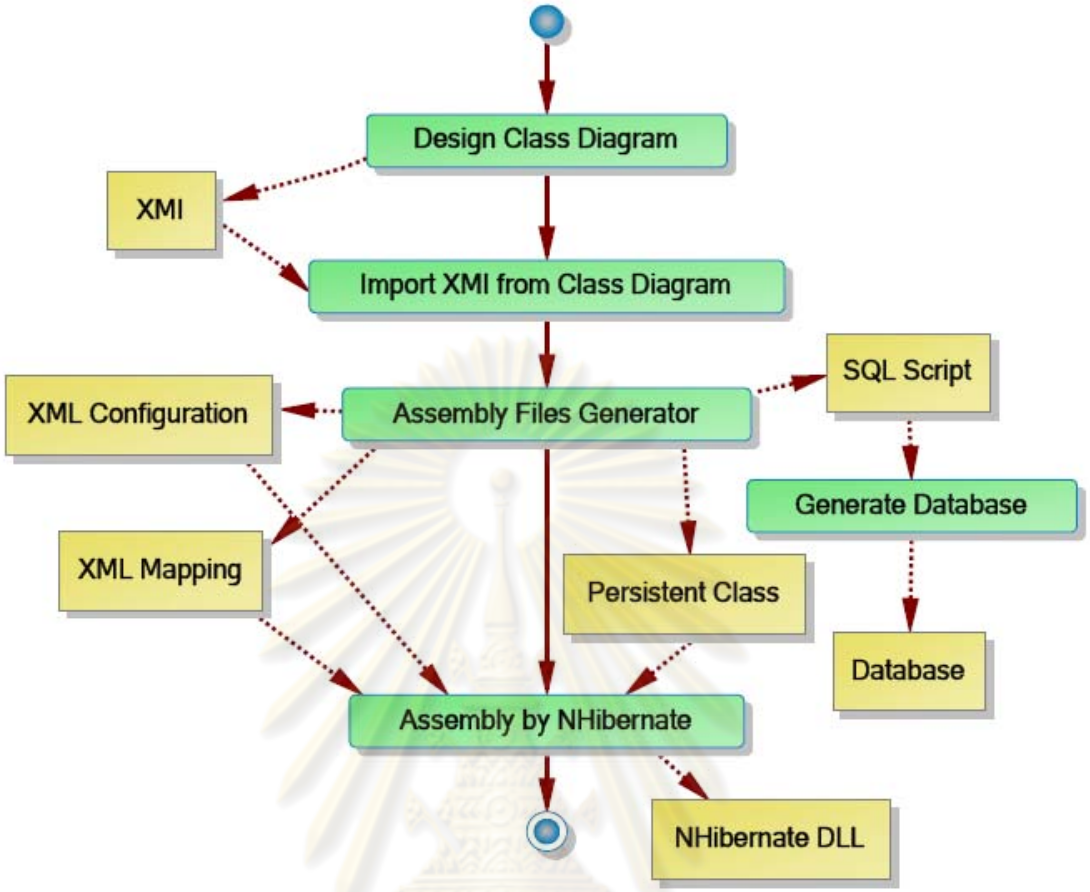
บทที่ 3
การวิเคราะห์และออกแบบโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต

ในงานวิจัยนี้ผู้วิจัยได้เลือกใช้ภาษายูเอ็มแอล (UML: Unified Modeling Language) เป็นเครื่องมือสำหรับสร้างหรือแสดงส่วนต่าง ๆ ของโปรแกรมโดยผู้วิจัยได้ทำการออกแบบโปรแกรมขึ้นมา

1) ภาพรวมของโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต
2) การออกแบบระบบงานสำหรับโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต
3) การออกแบบสถาปัตยกรรม (Architecture Design)
4) การออกแบบส่วนต่อประสานกราฟิกกับผู้ใช้ (Graphic User Interface Design)

3.1 ภาพรวมของโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต

จากที่ได้กล่าวมาข้างต้นในบทนำและขั้นตอนการใช้งานโปรแกรมเอ็นไฮเบอร์เนตในบทที่ 2 พบว่า ขั้นตอนการใช้งานโปรแกรมเอ็นไฮเบอร์เนตมีขั้นตอนการใช้งานหลายขั้นตอน แต่ขั้นตอนที่สำคัญ คือ ขั้นตอนการเตรียมไฟล์ 4 ไฟล์ ได้แก่ ไฟล์เอ็กซ์เอ็มแอลการตั้งค่า, ไฟล์เอ็กซ์เอ็มแอลการแม็พ, ไฟล์คลาสที่ใช้เพื่อทำการแม็พ และเอสคิวแอลสคริป เนื่องจากการเตรียมไฟล์ดังกล่าวผู้ใช้งานเอ็นไฮเบอร์เนตต้องทำการสร้างไฟล์ดังกล่าวขึ้นด้วยตัวเองส่งผลให้ใช้เวลาในการเตรียมไฟล์สูง ดังนั้นโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนตจึงต้องการช่วยลดเวลาจากการเตรียมไฟล์สนับสนุนดังกล่าว เพื่อให้สามารถนำไปใช้งานโปรแกรมเอ็นไฮเบอร์เบอร์เนตได้อย่างสะดวก และรวดเร็วขึ้น โดยเมื่อผู้ใช้งานโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนตแล้วจะมีขั้นตอนการทำงานดังอธิบายในแผนภาพกิจกรรมที่ 3.1



รูปที่ 3.1 แผนภาพกิจกรรมแสดงขั้นตอนการใช้งานโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต

จากแผนภาพกิจกรรมในรูปที่ 3.1จะเริ่มต้น ดังนี้

3.1.1 กิจกรรม Design Class Diagram

ผู้ใช้ต้องทำการออกแบบแผนภาพคลาสก่อน โดยในการออกแบบจะใช้โปรแกรม Visual Paradigm 7.2 ซึ่งเป็นโปรแกรมที่ใช้สำหรับวาดและออกแบบแผนภาพยูเอ็มแอล เมื่อทำการออกแบบแผนภาพคลาสที่ต้องการแล้วทำการส่งออกแผนภาพคลาสในรูปแบบของไฟล์เอ็กซ์เอ็มไอ

3.1.2 กิจกรรม Import XMI from Class Diagram

ผู้ใช้งานต้องนำเข้าแผนภาพคลาสที่อยู่ในรูปแบบของไฟล์เอ็กซ์เอ็มไอเข้ามายังโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต

3.1.3 กิจกรรม Assembly Files Generator

หลังจากโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนตได้นำเข้าไฟล์เอ็กซ์เอ็มไอแล้ว จะทำการแปลงข้อความที่อยู่ในรูปแบบเอ็กซ์เอ็มไอสร้างเป็นไฟล์สนับสนุนการใช้งานทั้ง 4 ไฟล์ คือ เอ็กซ์เอ็มแอลสำหรับการตั้งค่า (XML Configuration), เอ็กซ์เอ็มแอลที่ใช้ในการแม็พ (XML Mapping), คลาสที่ใช้สำหรับการแม็พ (Persistent Class) และเอสคิวแอลสคริปต์ (SQL Script)

3.1.4 กิจกรรม Generate Database

หลังจากที่ได้ไฟล์เอสคิวแอลสคริปต์ที่สร้างผ่านโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนตแล้ว จึงนำไปสร้างฐานข้อมูล

3.1.5 กิจกรรม Assembly by NHibernate

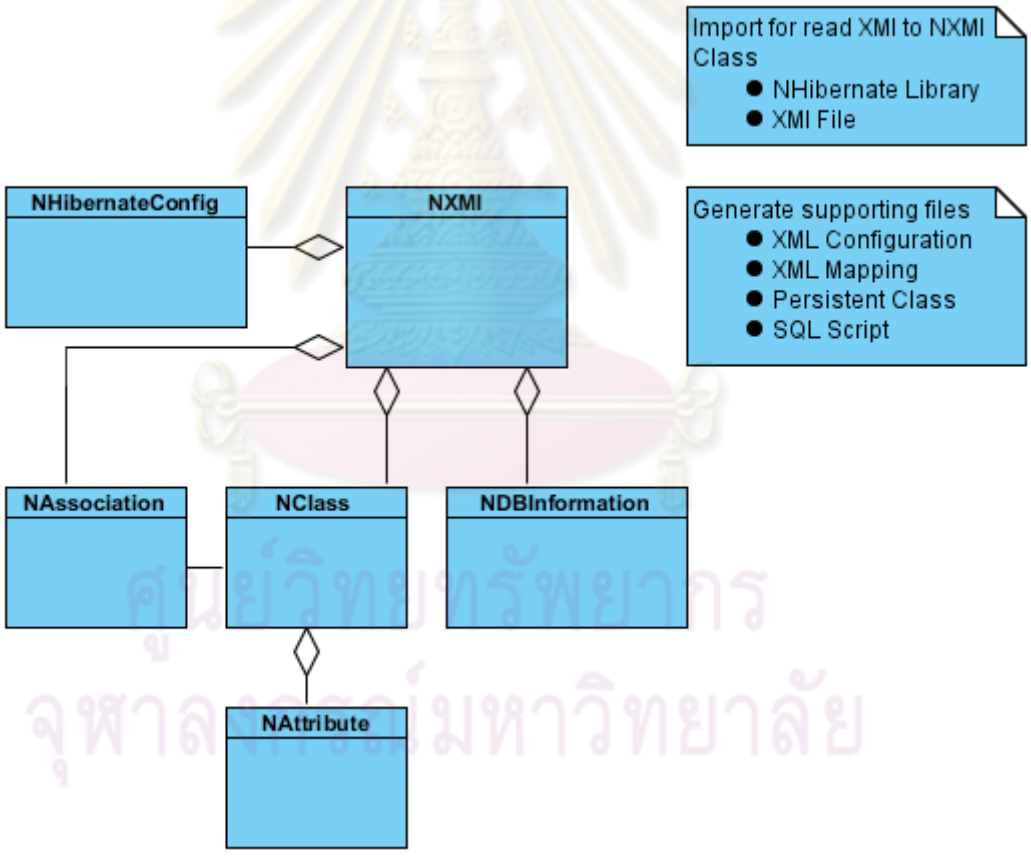
นำไฟล์สนับสนุนการใช้งานทั้ง 3 ไฟล์ คือ เอ็กซ์เอ็มแอลสำหรับการตั้งค่า (XML Configuration), เอ็กซ์เอ็มแอลที่ใช้ในการแม็พ (XML Mapping), คลาสที่ใช้สำหรับการแม็พ (Persistent Class) ไปทำการประกอบเป็นไฟล์ NHibernate.dll ด้วยโปรแกรม Microsoft Visual Studio .NET 2003 เพื่อนำไฟล์ NHibernate.dll ไปใช้งาน

3.2 การออกแบบระบบงานสำหรับโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต

การออกแบบระบบงานวิจัยจะใช้แผนภาพคลาส (Class Diagram) และแผนภาพกิจกรรม (Activity Diagram) เพื่ออธิบายถึงความสัมพันธ์ การเชื่อมโยงกันภายในระบบ และขั้นตอนวิธีการทำงานของโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต

3.2.1 แผนภาพคลาส

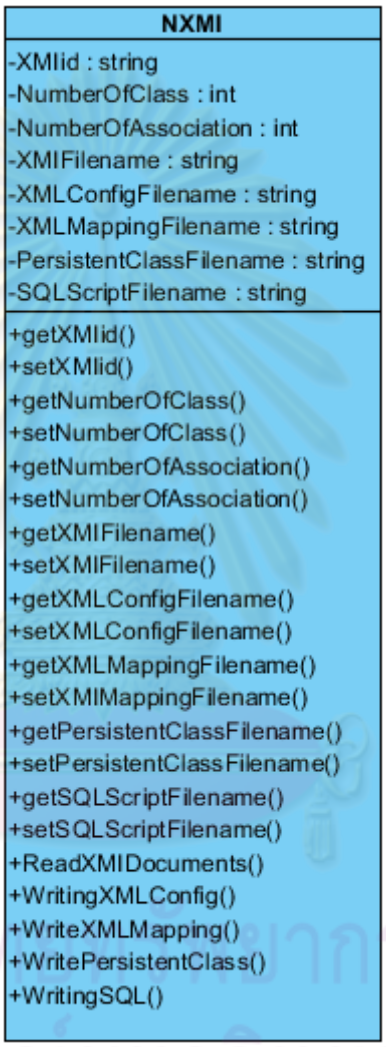
ในการออกแบบระบบงานเชิงวัตถุนั้น ขั้นตอนที่สำคัญ คือ ต้องค้นหาคลาสและวัตถุที่อยู่ภายใต้ขอบเขตของปัญหา ซึ่งผลลัพธ์ที่ได้สามารถแสดงแผนภาพคลาส รูปที่ 3.2



รูปที่ 3.2 แผนภาพคลาสของโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต

จากรูปที่ 3.3 ประกอบด้วยคลาสต่าง ๆ ดังนี้

- คลาส NXMI เป็นคลาสที่แสดงถึงคลาสที่ทำหน้าที่อ่านไฟล์เอ็กซ์เอ็มไอที่ส่งออกมาจากโปรแกรม Visual Paradigm 7.2 รูปที่ 3.3



รูปที่ 3.3 คลาส NXMI

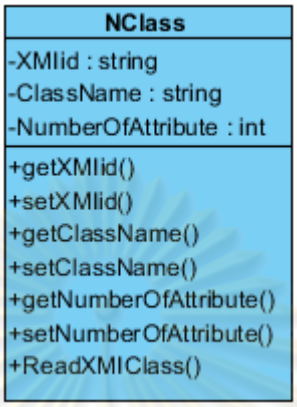
ตารางที่ 3.1 ลักษณะประจำของคลาส NXMI

ชื่อลักษณะประจำ	ความหมาย	ชนิดข้อมูล
XMIid	รหัสเอ็กซ์เอ็มไอ	string
NumberOfClass	จำนวนคลาส	int
NumberOfAssociation	จำนวนความสัมพันธ์	int
XMIFilename	ชื่อไฟล์เอ็กซ์เอ็มไอ	string
XMLConfigFilename	ชื่อไฟล์เอ็กซ์เอ็มแอลการตั้งค่า	string
XMLMappingFilename	ชื่อไฟล์เอ็กซ์เอ็มแอลที่ใช้ในการแม็พ	string
PersistentClassFilename	ชื่อไฟล์คลาสที่ใช้ในการแม็พ	string
SQLScriptFilename	ชื่อไฟล์เอสคิวแอลสคริป	string

ตารางที่ 3.2 บริการคลาส NXMI

ชื่อบริการ	ความหมาย
ReadXMIDocuments	อ่านไฟล์เอ็กซ์เอ็มไอ
WritingXMLConfig	เขียนไฟล์เอ็กซ์เอ็มแอลการตั้งค่า
WriteXMLMapping	เขียนไฟล์เอ็กซ์เอ็มแอลที่ใช้ในการแม็พ
WritePersistenceClass	เขียนไฟล์คลาสที่ใช้ในการแม็พ
WriteSQL	เขียนไฟล์เอสคิวแอลสคริป

- คลาส NClass เป็นคลาสที่แสดงถึงคลาสที่มีอยู่ในระบบ ซึ่งได้มาจากการอ่านไฟล์เอ็กซ์เอ็มไอ รูปที่ 3.4



รูปที่ 3.4 คลาส NClass

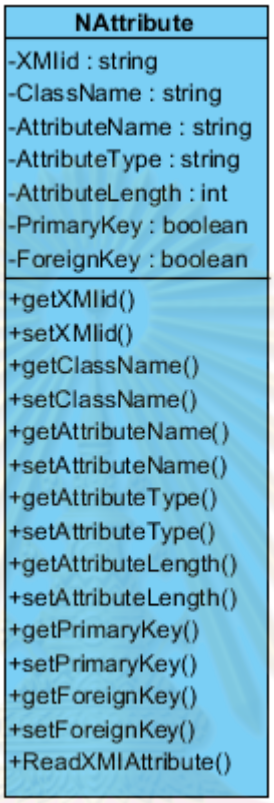
ตารางที่ 3.3 ลักษณะประจำของคลาส NClass

ชื่อลักษณะประจำ	ความหมาย	ชนิดข้อมูล
XMIid	รหัสเอ็กซ์เอ็มไอ	string
ClassName	จำนวนคลาส	string
NumberOfAttribute	จำนวนความสัมพันธ์	int

ตารางที่ 3.4 บริการคลาส NClass

ชื่อบริการ	ความหมาย
ReadXMIClass	อ่านคลาสจากไฟล์เอ็กซ์เอ็มไอ

- คลาส NAttribute เป็นคลาสที่แสดงถึงคุณสมบัติต่าง ๆ ของแต่ละคลาส ซึ่งแต่ละคลาสต้องมีคุณสมบัติอย่างน้อย 1 คุณสมบัติ รูปที่ 3.5



รูปที่ 3.5 คลาส NAttribute

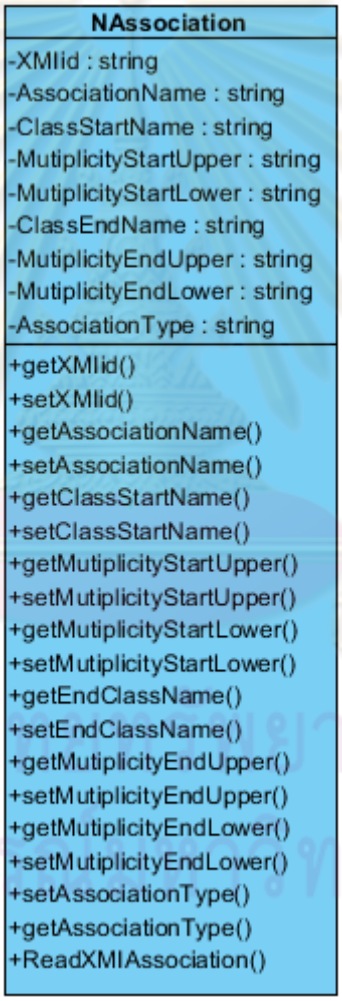
ตารางที่ 3.5 ลักษณะประจำของคลาส NAttribute

ชื่อลักษณะประจำ	ความหมาย	ชนิดข้อมูล
XMIid	รหัสเอ็กซ์เอ็มไอ	string
ClassName	ชื่อคลาส	string
AttributeName	ชื่อคุณลักษณะ	string
AttributeType	ประเภทของคุณลักษณะ	string
AttributeLength	ความยาวของคุณลักษณธ	int
PrimaryKey	เป็นคีย์หลัก	boolean
ForeignKey	เป็นคีย์ภายนอก	boolean

ตารางที่ 3.6 บริการคลาส NAttribute

ชื่อบริการ	ความหมาย
ReadXMIAttribute	อ่านคุณลักษณะจากไฟล์เอ็กซ์เอ็มไอ

- คลาส NAssociation เป็นคลาสที่แสดงถึงความสัมพันธ์ระหว่าง NClass แต่ละคลาส รูปที่ 3.6



รูปที่ 3.6 คลาส NAssociation

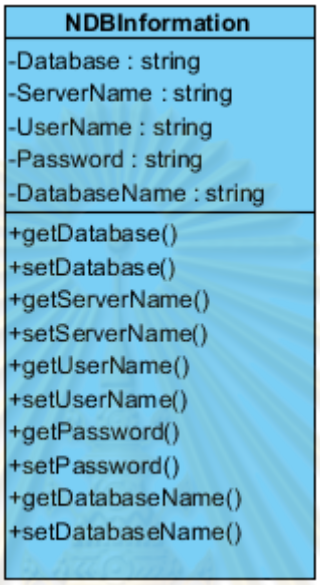
ตารางที่ 3.7 ลักษณะประจำของคลาส NAssociation

ชื่อลักษณะประจำ	ความหมาย	ชนิดข้อมูล
XMIid	รหัสเอ็กซ์เอ็มไอ	string
AssociationName	ชื่อความสัมพันธ์	string
ClassStartName	ชื่อคลาสเริ่มต้น	string
MultiplicityStartUpper		string
MultipilicityStartLower		string
ClassEndName	ชื่อคลาสสิ้นสุด	string
MultiplicityEndUpper		string
MultiplicityEndLower		string
AssociationType	ประเภทความสัมพันธ์	string

ตารางที่ 3.8 บริการคลาส NAssociation

ชื่อบริการ	ความหมาย
ReadXMIAssociation	อ่านความสัมพันธ์จากไฟล์เอ็กซ์เอ็มไอ

- คลาส NDBInformation เป็นคลาสที่ทำหน้าที่เก็บข้อมูลต่าง ๆ จากผู้ใช้งานในการทำการติดต่อระหว่างโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนตและฐานข้อมูล เช่น ชื่อฐานข้อมูล ชื่อผู้ใช้งาน รหัสผ่าน เป็นต้น รูปที่ 3.7

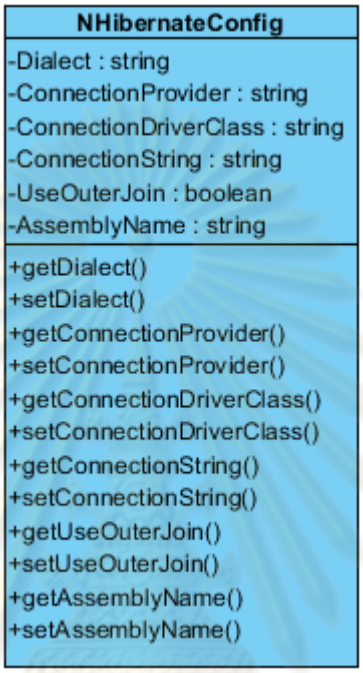


รูปที่ 3.7 คลาส NDBInformation

ตารางที่ 3.9 ลักษณะประจำของคลาส NDBInformation

ชื่อลักษณะประจำ	ความหมาย	ชนิดข้อมูล
Database	ชื่อชนิดฐานข้อมูล	string
ServerName	ชื่อเซิร์ฟเวอร์	string
UserName	ชื่อผู้ใช้	string
Password	รหัสผ่าน	string
DatabaseName	ชื่อฐานข้อมูล	string

- คลาส NHibernateConfig เป็นคลาสที่ทำหน้าที่แม็พข้อมูลที่ได้มาจากคลาส NHibernate ไปแม็พให้ตรงกับไลบรารี่ของโปรแกรมเอ็นไฮเบอร์เนตว่าฐานข้อมูลชนิดนี้ต้องมีการตั้งค่าสำหรับโปรแกรมเอ็นไฮเบอร์เนตอย่างไร รูปที่ 3.8



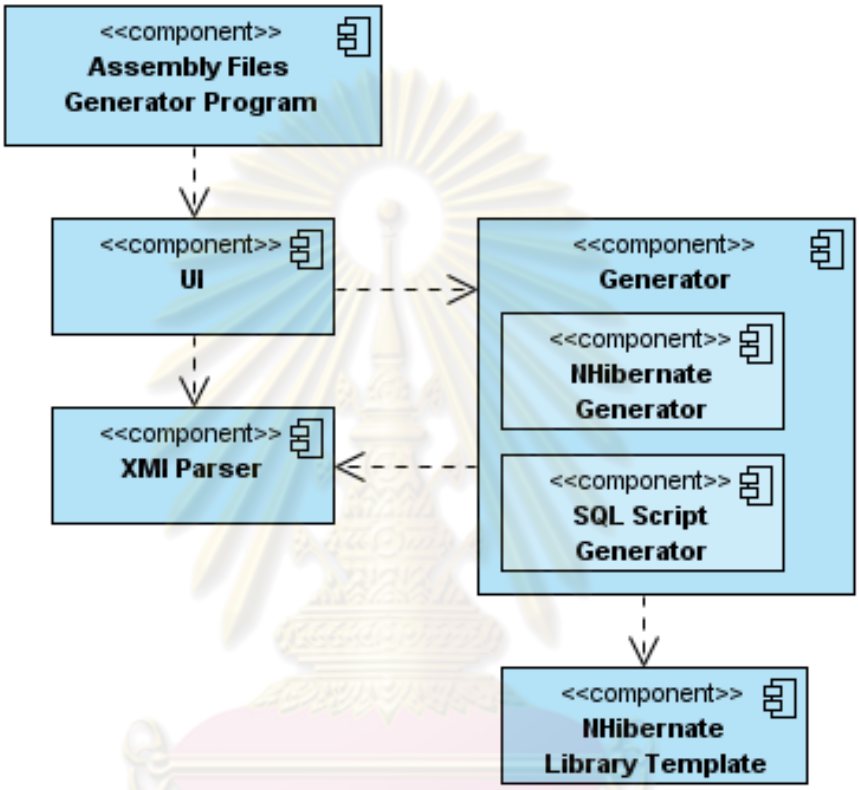
รูปที่ 3.8 คลาส NHibernateConfig

ตารางที่ 3.10 ลักษณะประจำของคลาส NHibernateConfig

ชื่อลักษณะประจำ	ความหมาย	ชนิดข้อมูล
Dialect	ชื่อ Dialect	string
ConnectionProvider	ชื่อผู้ให้บริการการเชื่อมต่อ	string
ConnectionDriverClass	คลาสไดรเวอร์การเชื่อมต่อ	string
UseOuterJoin	ใช้ความสัมพันธ์แบบ OuterJoin	string
AssemblyName	ชื่อไฟล์ประกอบ	string

3.3 การออกแบบสถาปัตยกรรม

สถาปัตยกรรมของโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต นั้นแสดงในแผนภาพส่วนประกอบรูปที่ 3.9 โดยมีรายละเอียดต่าง ๆ ดังนี้



รูปที่ 3.9 แผนภาพส่วนประกอบของโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต

- ส่วนประกอบ UI ทำหน้าที่ควบคุมและใช้งานโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต
- ส่วนประกอบ XMI Parser ทำหน้าที่จัดการและแปลงแผนภาพคลาสไดอะแกรมจาก Visual Paradigm 7.2 ให้อยู่ในรูปของไฟล์เอ็กซ์เอ็มไอ
- ส่วนประกอบ NHibernate Generator ทำหน้าที่รับและแปลงข้อมูลจากไฟล์เอ็กซ์เอ็มไอที่ได้รับจากส่วนประกอบ XMI Parser และทำการเปรียบเทียบกับแผ่นแบบไลบรารีของเอ็นไฮเบอร์เนต เพื่อทำการสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต

ได้แก่ ไฟล์เอ็กซ์เอ็มแอลตั้งค่าการใช้งาน, ไฟล์เอ็กซ์เอ็มแอลแม็พไฟล์ และคลาสที่ใช้ในการแม็พ

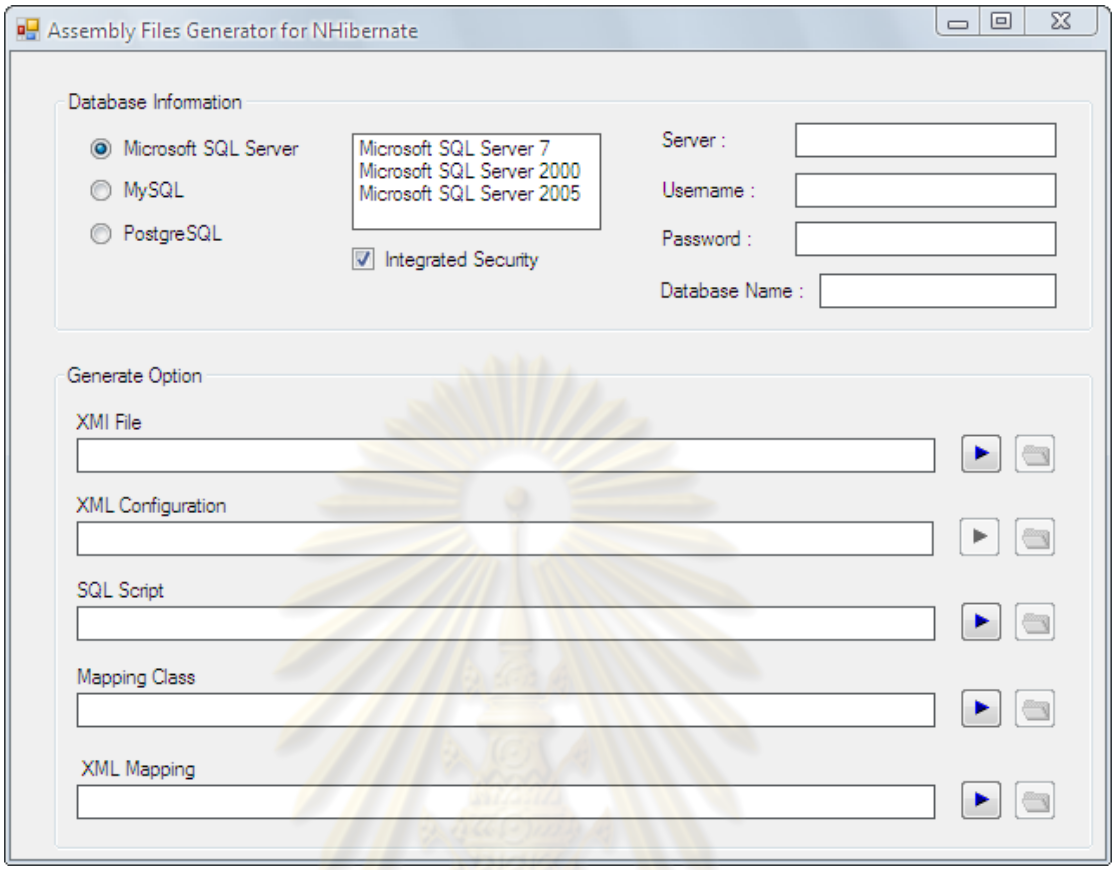
- ส่วนประกอบ SQL Script Generator ทำหน้าที่สร้างไฟล์เอสคิวแอลสคริปต์เพื่อนำไปใช้สร้างฐานข้อมูลร่วมกับการแม็พ

3.4 การออกแบบส่วนต่อประสานกราฟิกกับผู้ใช้

ผู้วิจัยได้ทำการออกแบบหน้าจอสำหรับการใช้งานโปรแกรม โดยจะอยู่ในรูปแบบของฟอร์ม (Form) โดยผ่านการรับข้อมูลจากผู้ใช้ ซึ่งจอภาพสามารถแสดงส่วนต่าง ๆ ที่ใช้งานได้ดังนี้

3.4.1 หน้าจอหลักของโปรแกรม

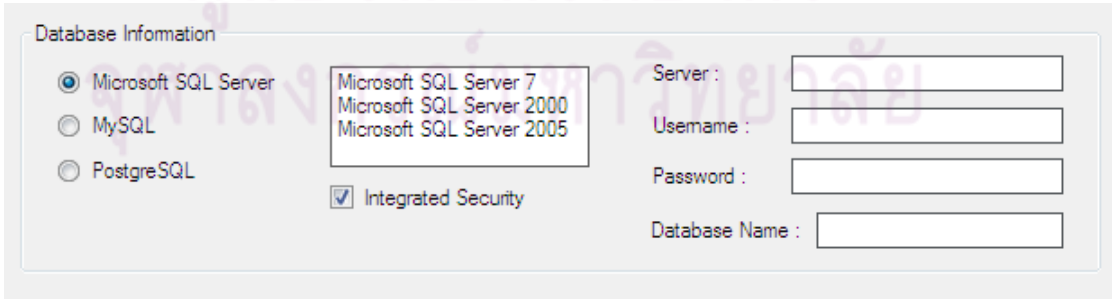
เป็นหน้าจอหลักของโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต ดังรูปที่ 3.10 และในส่วนของหน้าจอหลักนั้นสามารถแบ่งออกเป็น 2 ส่วน ได้แก่ 1.หน้าจอส่วนของการรับข้อมูลเกี่ยวกับฐานข้อมูล 2. หน้าจอส่วนของการนำเข้าไฟล์เอ็กซ์เอ็มไอและการสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต



รูปที่ 3.10 หน้าจอหลักของโปรแกรม

3.4.2 หน้าจอส่วนของการรับข้อมูลเกี่ยวกับฐานข้อมูล

ผู้ใช้ต้องป้อนข้อมูลที่ถูกต้องเข้าไป ได้แก่ ชนิดของฐานข้อมูล, เวอร์ชันของฐานข้อมูล, ชื่อฐานข้อมูล, ชื่อผู้ใช้งาน และรหัสผ่าน เป็นต้น ดังรูปที่ 3.11

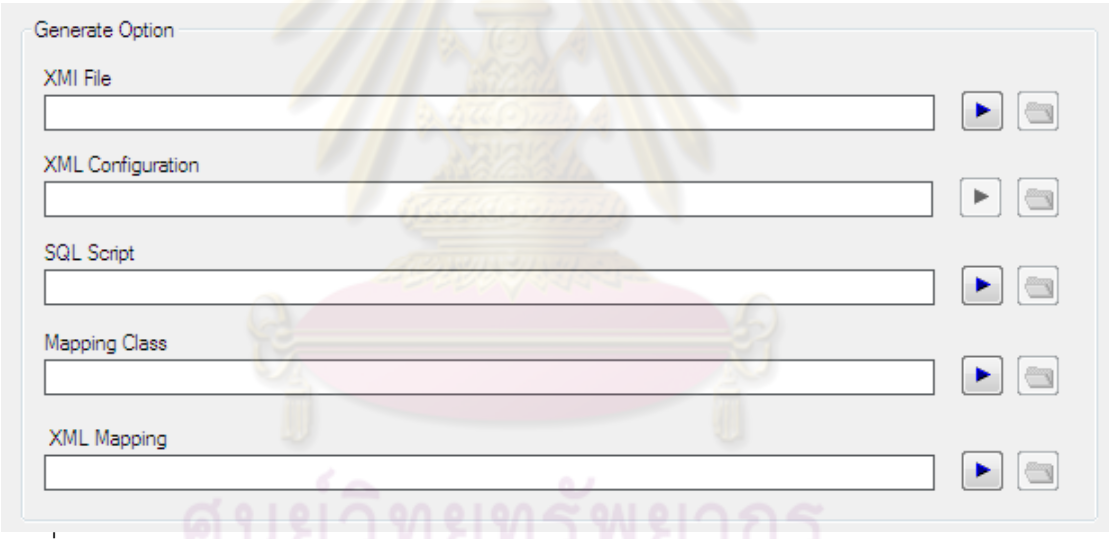


รูปที่ 3.11 หน้าจอส่วนของการรับข้อมูลเกี่ยวกับฐานข้อมูล

3.4.3 หน้าจอส่วนของการนำเข้าไฟล์เอ็กซ์เอ็มไอและการสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต

หน้าจอส่วนนี้ใช้สำหรับดำเนินการนำเข้าไฟล์เอ็กซ์เอ็มไอที่ได้จากการออกแบบผ่านโปรแกรม Visual Paradigm 7.2 ดังรูปที่ 3.12 โดยในหน้าจอดังกล่าวมีส่วนประกอบ ดังนี้

- กล่องข้อความ  ใช้สำหรับแสดงที่อยู่ของไฟล์ที่นำเข้าหรือสร้างขึ้น
- ข้อความแจ้งผลการดำเนินการ  ใช้สำหรับแจ้งผลการนำเข้าไฟล์หรือสร้างไฟล์
- ปุ่ม  ใช้สำหรับ สั่งให้โปรแกรมดำเนินการนำเข้าไฟล์หรือสร้างไฟล์
- ปุ่ม  ใช้สำหรับ ทำการเปิดโฟลเดอร์ที่ไฟล์ดังกล่าวเก็บไว้



รูปที่ 3.12 หน้าจอส่วนของการนำเข้าไฟล์หน้าจอส่วนของการนำเข้าไฟล์เอ็กซ์เอ็มไอและการสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต

บทที่ 4

การพัฒนาโปรแกรมและการทดสอบ

4.1 ฮาร์ดแวร์และซอฟต์แวร์ที่ใช้ในการพัฒนาโปรแกรม

การพัฒนาโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต ผู้วิจัยได้ใช้อุปกรณ์และเครื่องมือในการพัฒนาระบบดังนี้

4.1.1 รายละเอียดฮาร์ดแวร์ เครื่องไมโครคอมพิวเตอร์ มีดังนี้

- หน่วยประมวลผลกลางอินเทล เพนเทียมเอ็ม ความเร็ว 2.1 GHz.
- หน่วยความจำหลัก 4 GB.
- ฮาร์ดดิสก์ 320 GB

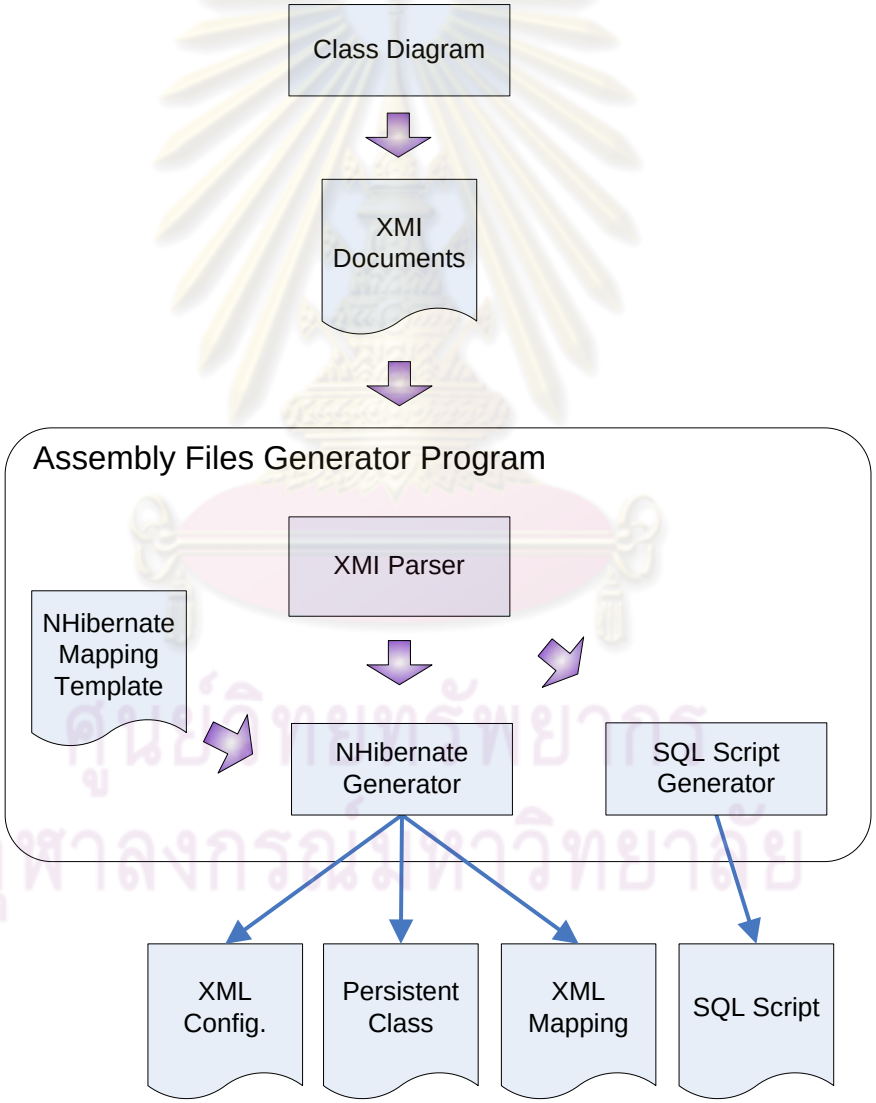
4.1.2 รายละเอียดซอฟต์แวร์ มีดังนี้

- ระบบปฏิบัติการ ได้แก่ Microsoft Windows XP Professional SP3
- Visual Paradigm 7.2 สำหรับออกแบบคลาสไดอะแกรมและเขียนแผนภาพยูเอ็มแอล
- ฐานข้อมูล ได้แก่ Microsoft SQL Server 2005, MySQL 5 และ PostgreSQL 8
- โปรแกรมเอ็นไฮเบอร์เนตเวอร์ชัน 1.2
- ชุดพัฒนาภาษาจาวา เวอร์ชัน 1.5 (Java Development Kit 1.5)
- ใช้โปรแกรม Microsoft Visual Studio .NET 2003, TestDriven.NET และ Microsoft .NET Framework SDK 1.1 สำหรับคอมไพล์และทดสอบไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต

- พัฒนาโปรแกรมด้วยด้วยภาษาซีชาร์ปดอทเน็ต (C#.NET) ภายใต้สภาพแวดล้อมของ Microsoft Visual Studio 2008 และ Microsoft .NET Framework 3.5 SP1

4.2 การพัฒนาโปรแกรม

งานวิจัยนี้ออกแบบและพัฒนาโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับไฮเบอร์เนต โดยได้นำเสนอการทำงานของโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต แบ่งออกเป็น 2 ส่วนสำคัญ ดังแสดงในรูปที่ 4.1 มีรายละเอียดดังนี้

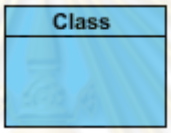


รูปที่ 4.1 แผนภาพแสดงการทำงานของโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต

4.2.1 ส่วนการนำเข้าไฟล์เอ็กซ์เอ็มไอผ่าน XMI Parser

ใช้โปรแกรมเครื่องมือที่ช่วยในการเขียนแผนภาพยูเอ็มแอล (UML: Unified Modeling Language) โดยงานวิจัยนี้เลือกใช้โปรแกรม Visual Paradigm for UML 7.2 ซึ่งโปรแกรม Visual Paradigm สามารถส่งออกข้อมูล (Export) จากแผนภาพคลาสออกมาเป็นไฟล์เอ็กซ์เอ็มแอลในรูปแบบของเอ็กซ์เอ็มไอ จากนำไฟล์เอ็กซ์เอ็มไอผ่าน XMI Parser เพื่อส่งต่อไปยัง NHibernate Generator และ SQL Script Generator โดย XMI Parser ยังรองรับสัญลักษณ์สำหรับแผนภาพคลาสบางส่วน ได้แก่

- สัญลักษณ์คลาส



รูปที่ 4.2 สัญลักษณ์คลาส

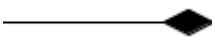
- สัญลักษณ์ Association

รูปที่ 4.3 สัญลักษณ์ Association

- สัญลักษณ์ Aggregation

รูปที่ 4.4 สัญลักษณ์ Aggregation

- สัญลักษณ์ Composition



รูปที่ 4.5 สัญลักษณ์ Composition

- สัญลักษณ์ Generalization

รูปที่ 4.6 สัญลักษณ์ Generalization

- สัญลักษณ์ Multiplicity แบบ 0, 1 และ * โดยถ้าหากมีการออกแบบเข้ามาในแบบอื่น XMI Parser จะแปลงให้เป็นแบบดังนี้ 0..1 เป็น 1 แบบ 0..* เป็น * และแบบ 1..* เป็น *

4.2.2 ส่วนการสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต

เป็นส่วนที่ทำหน้าที่รับไฟล์เอ็กซ์เอ็มไอที่ผ่าน XMI Parser เพื่อทำการสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นต์ไฮเบอร์เนตโดยมีการแปลงเปรียบเทียบกับเทมเพลทของเอ็นไฮเบอร์เนต โดยรายละเอียดของการสร้างไฟล์แต่ละไฟล์มีดังนี้

4.2.2.1 ส่วนของการสร้างไฟล์เอ็กซ์เอ็มแอลตั้งค่าใช้งาน ทำการรับค่าชนิดของฐานข้อมูลจากผู้ใช้ คือ การรับค่าชนิดของฐานข้อมูลที่ผู้ใช้งานต้องการนำไปใช้งานร่วมกับโปรแกรม NHibernate โดยมีให้เลือกอยู่ 3 ชนิด คือ ฐานข้อมูลไมโครซอฟท์เอสคิวแอลเซิร์ฟเวอร์ (Microsoft SQL Server) ฐานข้อมูลมายเอสคิวแอล (MySQL Database) และฐานข้อมูลโพสต์เกสท์เอสคิวแอล (PostgreSQL Database) การเลือกชนิดของฐานข้อมูลเพื่อที่โปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต จะได้ทำการกำหนดการตั้งค่าให้ออกมาอย่างถูกต้องตามรูปแบบของเอ็นไฮเบอร์เนต ซึ่งจะออกมาเป็นไฟล์เอ็กซ์เอ็มแอลตั้งค่าใช้งาน (XML configuration) ไฟล์เอ็กซ์เอ็มแอลที่สร้างขึ้นจะมีรายละเอียดการตั้งค่าใช้งาน ส่วนที่สำคัญ คือ เซสชั่นแฟกทอรี่ (session-factory) ซึ่งเป็นส่วนที่เก็บรายละเอียดในการตั้งค่าเกี่ยวกับการติดต่อกับฐานข้อมูล มีคุณสมบัติที่สำคัญดังนี้

1. **hibernate.connection.provider** เป็นคุณสมบัติ (property) ที่เก็บค่าเกี่ยวกับผู้ให้บริการ (provider) ปกติจะทำการกำหนดค่าเป็น NHibernate.Connection.Driver ConnectionProvider

2. **hibernate.dialect** เป็นคุณสมบัติ (property) เกี่ยวกับชนิดของฐานข้อมูลที่ทำการติดต่อ ตามตารางที่ 4.1

ตารางที่ 4.1 NHibernate SQL Dialects (hibernate.dialect)

RDBMS	Dialect
Microsoft SQL Server 2000	NHibernate.Dialect.MsSql2000Dialect
Microsoft SQL Server 2005	NHibernate.Dialect.MsSql2005Dialect
Microsoft SQL Server 7	NHibernate.Dialect.MsSql7Dialect
MySQL 3 or 4	NHibernate.Dialect.MySQLDialect
MySQL 5	NHibernate.Dialect.MySQL5Dialect
PostgreSQL	NHibernate.Dialect.PostgreSQLDialect
PostgreSQL 8.1	NHibernate.Dialect.PostgreSQL81Dialect
PostgreSQL 8.2	NHibernate.Dialect.PostgreSQL82Dialect

3. **hibernate.connection.driver_class** เป็นคุณสมบัติ (property) ที่เก็บชื่อคลาสของไดรเวอร์ที่ใช้ติดต่อกับฐานข้อมูลที่ทำการติดต่อ ดังตารางที่ xx.xx

ตารางที่ 4.2 NHibernate Connection Driver Class

RDBMS	Dialect
Microsoft SQL Server	NHibernate.Driver.SqlClientDriver
MySQL	NHibernate.Driver.MySqlDataDriver
PostgreSQL	NHibernate.Driver.NpgsqlDriver

4. **hibernate.connection.connection_string** เป็นคุณสมบัติ (property)ที่เก็บค่าข้อความการเชื่อมต่อกับฐานข้อมูล (connection string) ภายในจะระบุถึงรายละเอียด เช่น เซิร์ฟเวอร์ (server), ชื่อฐานข้อมูล (Database name), ชื่อผู้ใช้งาน (user id) และรหัสผ่าน (password)

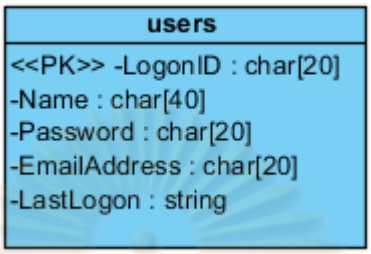
ตัวอย่างไฟล์เอ็กซ์เอ็มแอลที่ใช้ในการตั้งค่า ใช้ชื่อไฟล์ว่า app.config โดยรายละเอียด ดังนี้

- ใช้ฐานข้อมูลไมโครซอฟท์เอสคิวแอลเซิร์ฟเวอร์ (Microsoft SQL Server 2000)
- เซิร์ฟเวอร์ชื่อ = (local)
- ฐานข้อมูลชื่อ = NHibernateDB
- ชื่อผู้ใช้งาน = sa
- รหัสผู้ใช้ = ********

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
        <configSections>
        <section
                name="hibernate-configuration"
                type="NHibernate.Cfg.ConfigurationSectionHandler, NHibernate"
        />
        </configSections>

        <hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
        <session-factory>
        <propertyname="dialect">
                NHibernate.Dialect.MsSql2000Dialect
        </property>
        <propertyname="connection.provider">
                NHibernate.Connection.DriverConnectionProvider
        </property>
        <property name="connection.connection_string">
                Server=(local);
                initial catalog=NHibernateDB;
                Integrated Security=SSPI;
                User Id=sa;
                  Password=********
        </property>
        <mapping assembly="QuickStart" />
        </session-factory>
        </hibernate-configuration>
</configuration>
```

รูปที่ 4.7 ตัวอย่างไฟล์เอ็กซ์เอ็มแอลที่ใช้ในการตั้งค่า (XML Configuration)

4.2.2.2 คลาสที่ใช้ในการแม็พ (Persistent Class) เป็นคลาสที่ต้องการนำมาแม็พเข้ากับฐานข้อมูลด้วยไฟล์เอ็กซ์เอ็มแอล โดยภาษาที่เลือกใช้ในการเขียนคือ ซีชาร์ปดอตเนต (C#.NET) ตัวอย่างคลาสที่ต้องการนำมาแม็พโดยใช้ชื่อไฟล์ คือ User.cs ตัวอย่างของคลาสที่นำมาใช้อธิบาย



รูปที่ 4.8 คลาส users

```
using System;
namespace NHibernate.Examples.QuickStart
{
    public class User
    {
        private string id;
        private string userName;
        private string password;
        private string emailAddress;
        private DateTime lastLogon;

        public User()
        {
        }
        public string Id
        {
            get { return id; }
            set { id = value; }
        }
```

รูปที่ 4.9 ตัวอย่างคลาสที่ใช้ในการแม็พ (Persistent Class)

```
public string userName
        {
            get { return userName; }
            set { id = value; }
        }
         public string password
        {
            get { return password; }
            set { id = value; }
        }
         public string emailAddress
        {
            get { return emailAddress; }
            set { id = value; }
        }
         public string userName
        {
            get { return userName; }
            set { id = value; }
        }
         public string lastLogon
        {
            get { return lastLogon; }
            set { id = value; }
        }
}
```

รูปที่ 4.9 ตัวอย่างคลาสที่ใช้ในการแม็พ (Persistent Class) (ต่อ)

4.2.2.3 ส่วนการสร้างเอ็กซ์เอ็มแอลแม็พไฟล์ โดยส่วนนี้เป็นไฟล์เอ็กซ์เอ็มแอลแม็พไฟล์ จะนำไฟล์นี้ไปใช้งานร่วมกันคลาสที่ใช้ในการแม็พ โดยตัวอย่างไฟล์ใช้ชื่อว่า User.hbm.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
    namespace="QuickStart" assembly="QuickStart">
  <class name="NHibernate.Examples.QuickStart.User, NHibernate.Examples"
table="users">
    <id name="Id" column="LogonId" type="String" length="20">
      <generator class="assigned" />
    </id>
    <property name="UserName" column="Name" type="String" length="40"/>
    <property name="Password" type="String" length="20"/>
    <property name="EmailAddress" type="String" length="40"/>
    <property name="LastLogon" type="DateTime"/>
  </class>
</hibernate-mapping>
```

รูปที่ 4.10 ตัวอย่างไฟล์เอ็กซ์เอ็มแอลแม็พไฟล์ (XML Mapping)

4.2.2.4 ส่วนการสร้างเอสคิวแอลสคริปต์ ส่วนนี้จะมีการทำงานคล้ายกับส่วนของการสร้างไฟล์อื่น ๆ แต่จะทำการแปลงออกมาเป็นไฟล์เอสคิวแอลสคริปต์แทน ตัวอย่างไฟล์เอสคิวแอลสคริปต์ โดยรายละเอียด ดังนี้

- ใช้ฐานข้อมูลไมโครซอฟท์เอสคิวแอลเซิร์ฟเวอร์ (Microsoft SQL Server 2000)
- ฐานข้อมูลชื่อ = NHibernateDB
- ชื่อตาราง = users

ตารางที่ 4.3 รายละเอียดคอลัมน์ในตาราง users

คอลัมน์	ชนิดของข้อมูล	รายละเอียด
LogonID	nvarchar(20)	Primark Key, Default value = 0
Name	nvarchar(40)	NULL
Password	nvarchar(20)	NULL
EmailAddress	nvarchar(40)	NULL
LastLogon	DateTime	NULL

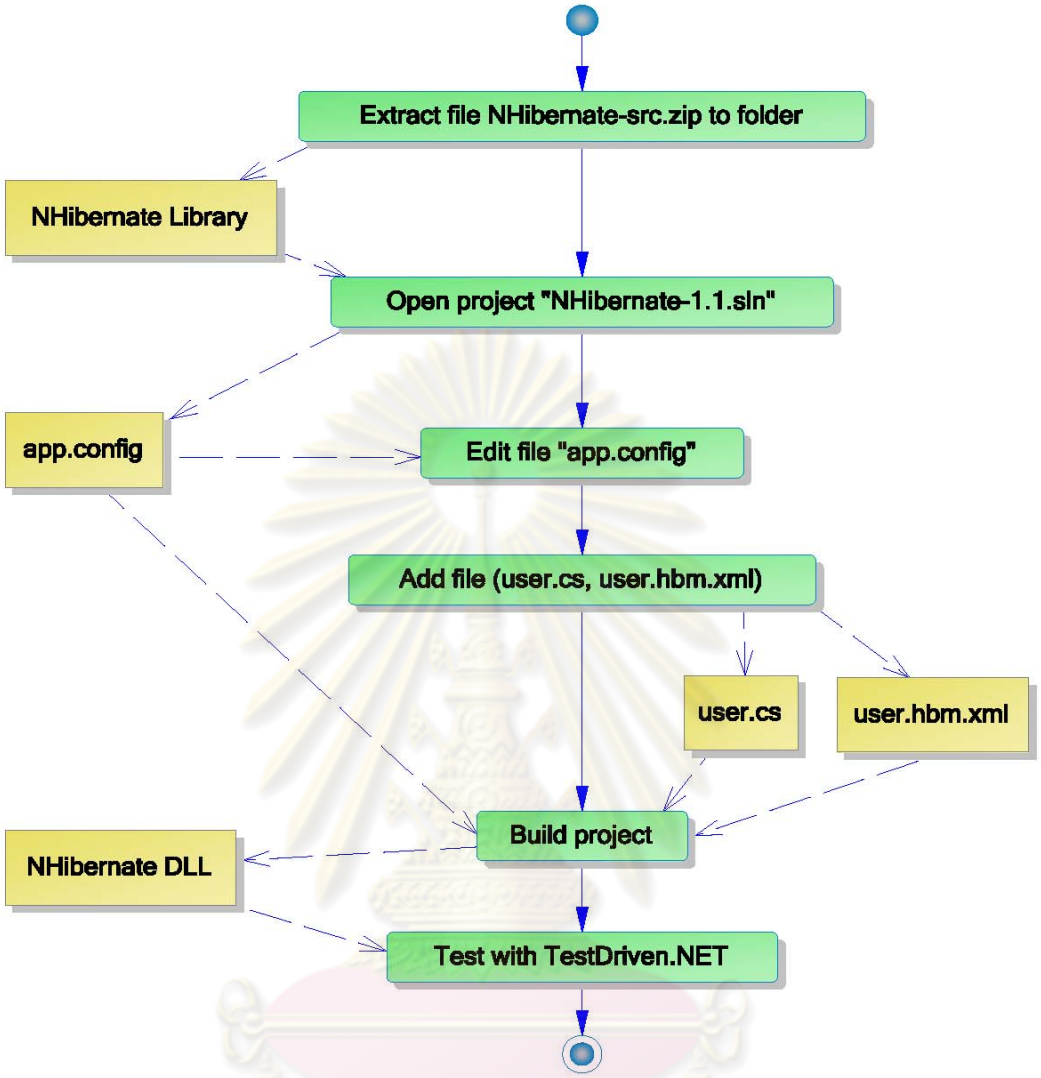
```
use NHibernateDB
go


CREATE TABLE users (
  LogonID nvarchar(20) NOT NULL default '0',
  Name nvarchar(40) default NULL,
  Password nvarchar(20) default NULL,
  EmailAddress nvarchar(40) default NULL,
  LastLogon datetime default NULL,
  PRIMARY KEY  (LogonID)
)
Go
```

รูปที่ 4.11 ตัวอย่างไฟล์เอสคิวแอลสคริปต์ (SQL Script)

4.3 การทดสอบโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต

ในการทดสอบโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต ต้องทำการประกอบไฟล์และทดสอบไฟล์สนับสนุนการใช้งานเอ็นไฮเบอร์เนต โดยการนำไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต 3 ไฟล์ได้แก่ เอ็กซ์เอ็มแอลตั้งค่าการใช้งาน, คลาสที่ใช้ในการแม็พ และเอ็กซ์เอ็มแอลแม็พไฟล์มาทำการประกอบไฟล์รวมกับไลบรารีของเอ็นไฮเบอร์เนต ส่วนเอสคิวแอลสคริปต์ที่ได้นำไปสร้างฐานข้อมูล หลังจากการประกอบไฟล์จะได้ไฟล์ NHibernate.dll ไฟล์หนึ่งซึ่งสามารถนำไปใช้พัฒนาแอพพลิเคชันอื่นต่อไปได้ ขั้นตอนการประกอบไฟล์สามารถทำได้การใช้โปรแกรม Visual Studio .NET 2003 และ TestDriven.NET ในการประกอบไฟล์และทดสอบไฟล์ ขั้นตอนดังกล่าวแสดงในแผนภาพกิจกรรมรูปที่ 4.12

- **กิจกรรม Extract file NHibernate-src.zip to folder** ขยายไฟล์ NHibernate-src.zip ไปยังโฟลเดอร์ที่ต้องการทำการประกอบไฟล์ จะได้ไลบรารีของโปรแกรมเอ็นไฮเบอร์เนตที่ต้องการในโฟลเดอร์
- **กิจกรรม Open project “NHibernate-1.1.sln”** เรียกใช้โปรแกรม Microsoft Visual Studio .NET 2003 เปิดโปรเจคไฟล์ชื่อ NHibernate-1.1.sln
- **กิจกรรม Edit file “app.config”** แก้ไขไฟล์ app.config ในโปรเจค
- **กิจกรรม Add file** เพิ่มไฟล์ User.cs และ User.hbm.xml เข้าไปในโปรเจค
- **กิจกรรม Build Project** เรียกใช้คำสั่ง Bulid ในโปรแกรม Microsoft Visual Studio .NET 2003เพื่อประกอบไฟล์
- **กิจกรรม Test with NUnit** โปรแกรม TestDriven.NET จะอินทิเกรตอยู่ใน Microsoft Visual Studio .NET 2003 โดยจะทำการทดสอบการประกอบไฟล์ขณะเรียกใช้คำสั่ง Build

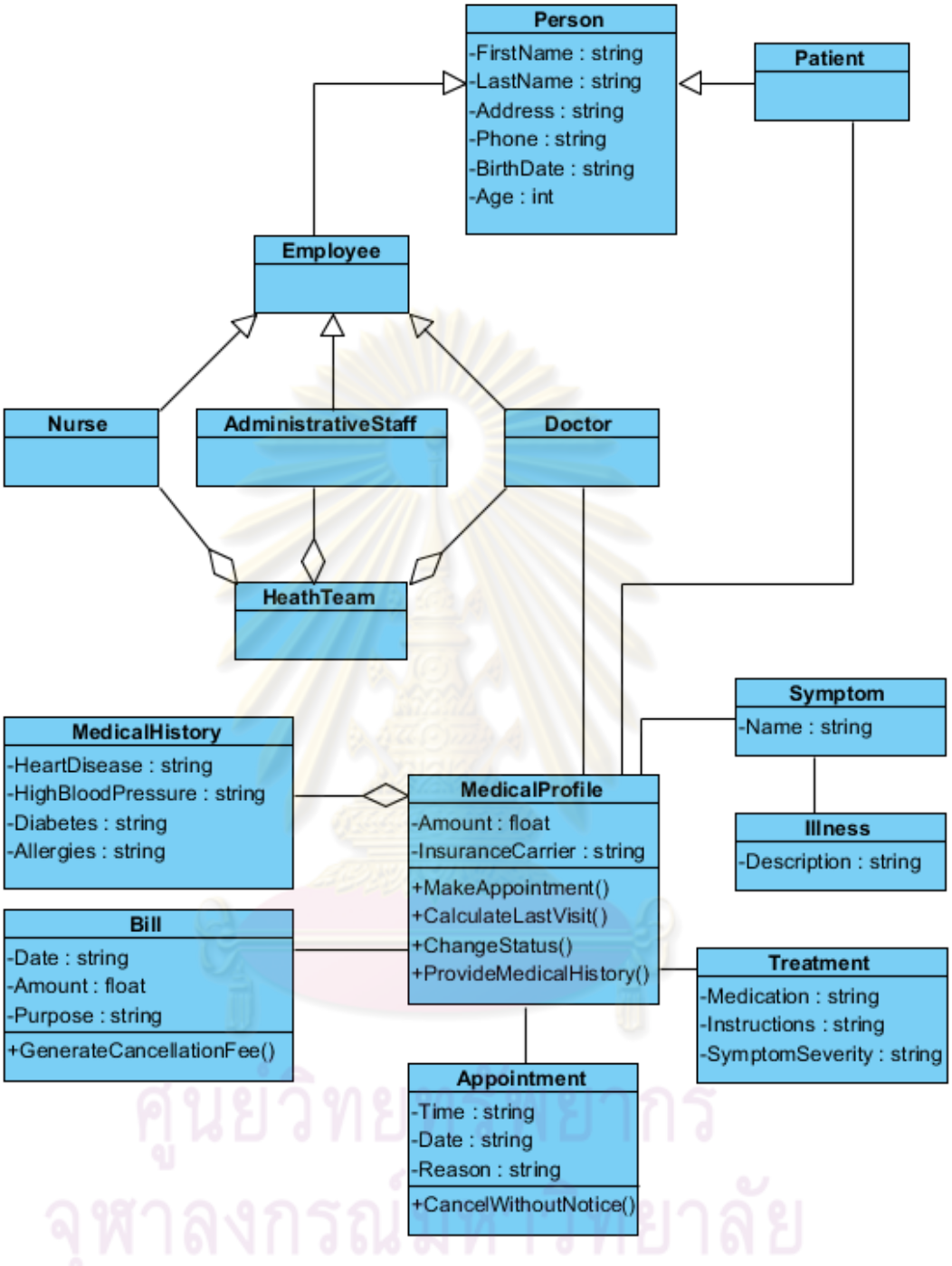


รูปที่ 4.12 แผนภาพกิจกรรมแสดงการประกอบไฟล์ด้วยโปรแกรม Visual Studio .NET 2003 และทดสอบด้วยโปรแกรม TestDriven.NET

วิธีการใช้งานจะทำการประกาศเรียกใช้ NHibernate.dll ที่ประกอบขึ้นในแอพพลิเคชันนั้น ทำให้แอพพลิเคชันที่พัฒนาขึ้นสามารถใช้เมธอดต่าง ๆ ของเอ็นไฮเบอร์เนตได้ เพื่อเป็นการทดสอบว่าไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนตที่สร้างขึ้นจากโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนตสามารถทำงานเป็นไปตามขอบเขตที่เรากำหนดไว้ โดยในการทดสอบนั้นผู้วิจัยได้เลือกระบบงาน 3 ระบบ ดังนี้

4.3.1 กรณีศึกษาระบบทะเบียนประวัติและการนัดหมายระหว่างแพทย์และคนไข้

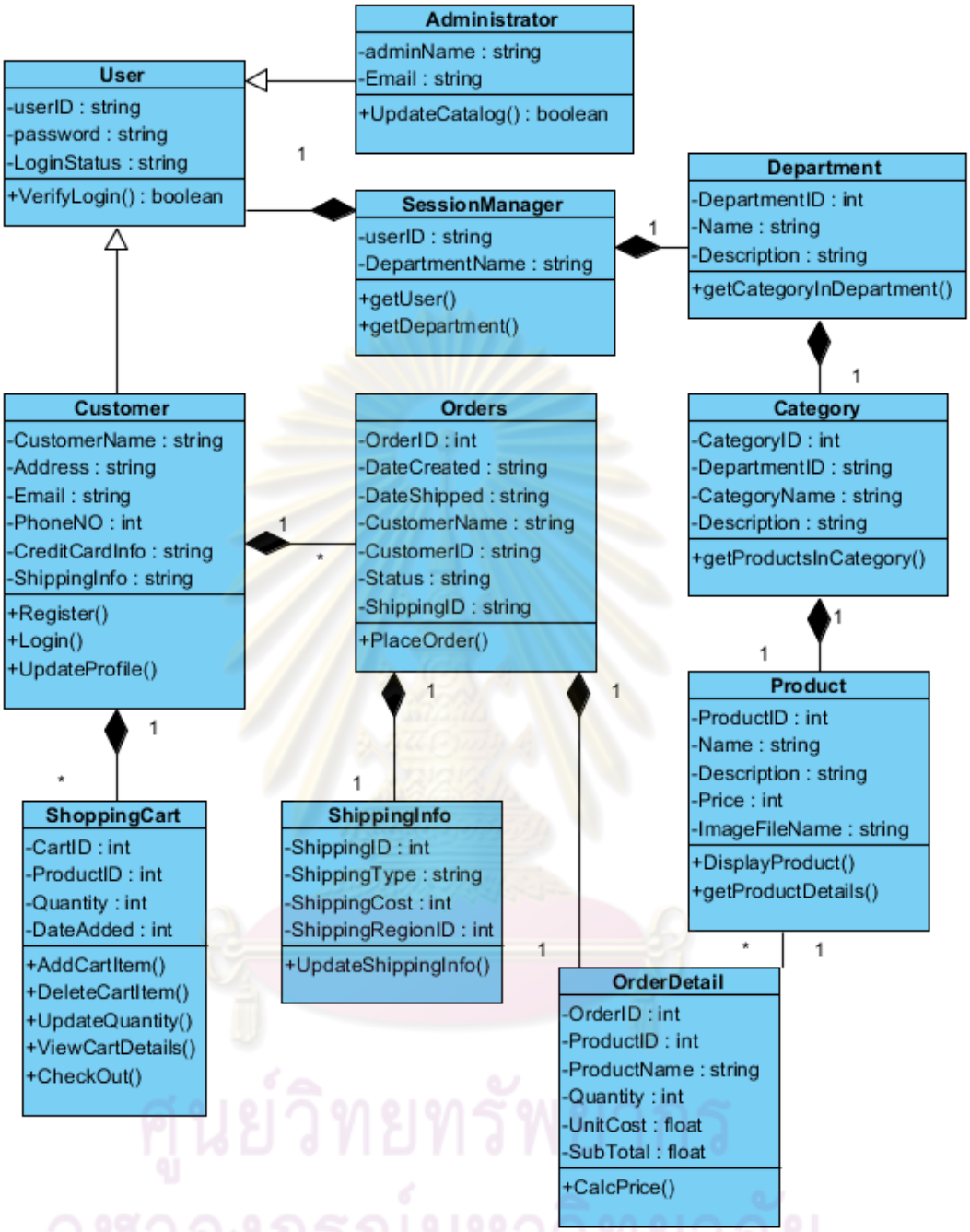
โดยระบบดังกล่าวเป็นระบบที่ประกอบด้วยคลาสจำนวน 14 คลาส และมีความสัมพันธ์ระหว่างคลาสแบบ Association จำนวน 7 คลาส ความสัมพันธ์แบบ Generalization จำนวน 5 คลาส และมีความสัมพันธ์แบบ Composition จำนวน 4 คลาส โดยระบบดังกล่าวเป็นระบบทะเบียนประวัติของคนไข้ที่เข้ามาทำการใช้บริการที่โรงพยาบาล และได้มีการทำการนัดหมายเพื่อพบแพทย์ในครั้งต่อไป ดังแสดงในแผนภาพคลาสรูปที่ 4.13 การทดสอบกับระบบทะเบียนประวัติและการนัดหมายระหว่างแพทย์และคนไข้ พบว่าโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต สามารถสร้างไฟล์สนับสนุนทั้ง 4 ไฟล์ได้อย่างไม่พบปัญหาใด ๆ ขณะทำการสร้างไฟล์สนับสนุน และเมื่อทำการทดสอบด้วยการนำไฟล์ไปสร้างฐานข้อมูล และนำประกอบไฟล์ด้วยโปรแกรม Visual Studio .NET 2003 สามารถประกอบไฟล์เป็นไฟล์ NHibernate.dll ได้โดยไม่พบปัญหา จากนั้นจึงทำการทดสอบด้วยโปรแกรม TestDriven.NET ไม่พบปัญหาเช่นกัน ไฟล์สนับสนุนที่สร้างขึ้นจะแสดงอยู่ในภาคผนวก ข



รูปที่ 4.13 แผนภาพคลาสของระบบทะเบียนประวัติและการนัดหมายระหว่างแพทย์และคนไข้

4.3.2 กรณีศึกษาระบบการสั่งซื้อสินค้า

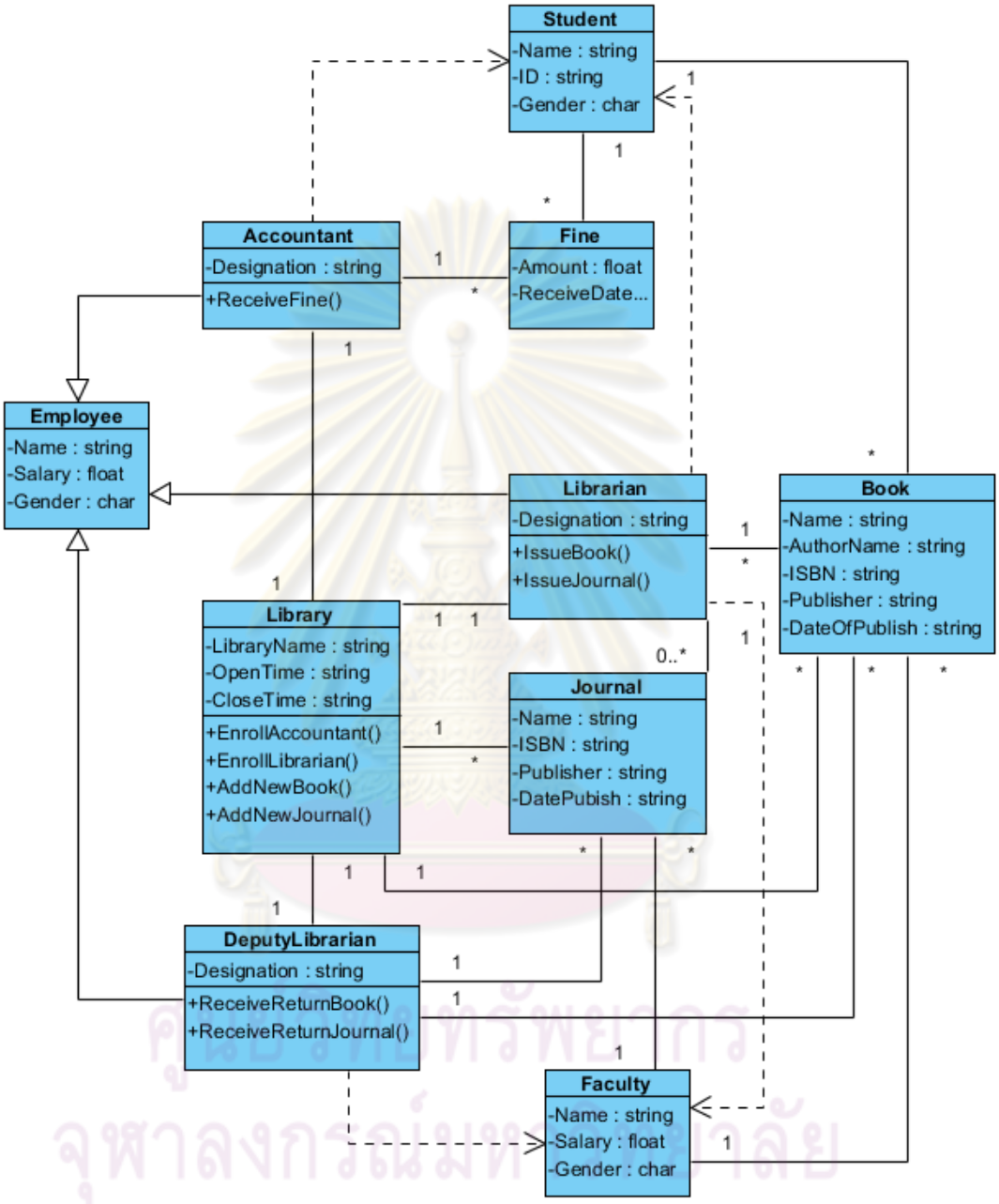
โดยระบบดังกล่าวเป็นระบบที่ประกอบด้วยคลาสจำนวน 11 คลาส และมีความสัมพันธ์ระหว่างคลาสแบบ Association จำนวน 1 คลาส ความสัมพันธ์แบบ Generalization จำนวน 2 คลาส และมีความสัมพันธ์แบบ Aggregation จำนวน 8 คลาส โดยระบบดังกล่าวเป็นระบบการสั่งซื้อสินค้าผ่านเว็บไซต์ ดังแสดงในแผนภาพคลาสรูปที่ 4.14 การทดสอบกับการสั่งซื้อสินค้าพบว่าโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต สามารถสร้างไฟล์สนับสนุนทั้ง 4 ไฟล์ได้อย่างไม่พบปัญหาใด ๆ ขณะทำการสร้างไฟล์สนับสนุน แต่พบข้อสังเกตคือขณะทำการนำเข้าไฟล์เอ็กซ์เอ็มไอระบบดังกล่าวใช้เวลาในการนำเข้าข้อมูลเป็นเวลานานพอสมควร และเมื่อทำการทดสอบด้วยการนำไฟล์ไปสร้างฐานข้อมูล และนำประกอบไฟล์ด้วยโปรแกรม Visual Studio .NET 2003 สามารถประกอบไฟล์เป็นไฟล์ NHibernate.dll ได้โดยไม่พบปัญหา จากนั้นจึงทำการทดสอบด้วยโปรแกรม TestDriven.NET ไม่พบปัญหาเช่นกัน ไฟล์สนับสนุนที่สร้างขึ้นจะแสดงอยู่ในภาคผนวก ค



รูปที่ 4.14 แผนภาพคลาสของระบบสั่งซื้อสินค้า

4.3.3 กรณีศึกษาระบบการยืมคืนหนังสือและวารสารของห้องสมุด

โดยระบบดังกล่าวเป็นระบบที่ประกอบด้วยคลาสจำนวน 10 คลาส และมีความสัมพันธ์ระหว่างคลาสแบบ Association จำนวน 11 คลาส ความสัมพันธ์แบบ Generalization จำนวน 3 โดยระบบดังกล่าวเป็นระบบการยืมคืนหนังสือและวารสารของห้องสมุด ดังแสดงในแผนภาพคลาสรูปที่ 4.15 การทดสอบกับการสั่งซื้อสินค้า พบว่าโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต สามารถสร้างไฟล์สนับสนุนทั้ง 4 ไฟล์ได้อย่างไม่พบปัญหาใด ๆ ขณะทำการสร้างไฟล์สนับสนุน แต่มีข้อสังเกตคือ มีการใช้สัญลักษณ์ Dependency ในแผนภาพคลาส ซึ่งเมื่อโปรแกรมพบก็ไม่เกิดปัญหาแต่จะไม่มีการสร้างความสัมพันธ์ของ Dependency และเมื่อทำการทดสอบด้วยการนำไฟล์ไปสร้างฐานข้อมูล และนำประกอบไฟล์ด้วยโปรแกรม Visual Studio .NET 2003 สามารถประกอบไฟล์เป็นไฟล์ NHibernate.dll ได้โดยไม่พบปัญหา จากนั้นจึงทำการทดสอบด้วยโปรแกรม TestDriven.NET ไม่พบปัญหาเช่นกัน ไฟล์สนับสนุนที่สร้างขึ้นจะแสดงอยู่ใน ภาคผนวก ง



รูปที่ 4.15 แผนภาพคลาสของระบบการยืมคืนหนังสือและวารสารของห้องสมุด

จากการทดสอบการทำงานของโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต ผู้ใช้งานสามารถนำแผนภาพคลาสที่ส่งออกมาในรูปของไฟล์เอ็กซ์เอ็มไอ และถูกนำเข้าไปยังโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต โปรแกรมสามารถสร้างไฟล์สนับสนุนการใช้งานได้ทั้ง 4 ไฟล์ตามความต้องการ โดยสัญลักษณ์ที่ใช้ในแผนภาพคลาสยังรองรับสัญลักษณ์ตามที่ได้อธิบายไว้ในหัวข้อ 4.2.1 เท่านั้น หากมีการใช้สัญลักษณ์นอกเหนือจากนั้นโปรแกรมจะไม่เกิดปัญหาขณะทำการทำงานแต่จะไม่สร้างรายละเอียดหรือความสัมพันธ์ที่ออกแบบไว้ในแผนภาพคลาสในส่วนของสัญลักษณ์นั้น ๆ และยังพบว่าในการทดสอบกรณีศึกษาระบบการสั่งซื้อสินค้า ซึ่งรายละเอียดในแผนภาพคลาสจะมีลักษณะประจำ (Attribute) และเมทอดจำนวนมากกว่ากรณีศึกษาอื่น ๆ จะมีผลต่อเวลาที่ใช้ในการนำเข้าข้อมูลของไฟล์เอ็กซ์เอ็มไอยิ่งมีจำนวนมากจะทำให้ผลการใช้งานขณะทำการนำเข้าไฟล์เอ็กซ์เอ็มไอมีความเร็วลดลง แต่จำนวนของคลาสที่มากขึ้นไม่ส่งผลความเร็วในการนำเข้าไฟล์เอ็กซ์เอ็มไออย่างชัดเจนนัก

บทที่ 5

สรุปผลการวิจัยและข้อเสนอแนะ

5.1 สรุปผลการวิจัย

การวิจัยครั้งนี้ได้ทำการพัฒนาโปรแกรมเครื่องมือสำหรับสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต เพื่อเป็นทางเลือกในการใช้งานโปรแกรมเอ็นไฮเบอร์เนตได้อย่าง มีประสิทธิภาพเพิ่มขึ้นโดยลดเวลาและข้อผิดพลาดในการเตรียมไฟล์สนับสนุนการใช้งานโปรแกรมเอ็นไฮเบอร์เนตจากปกติที่ผู้ใช้งานต้องสร้างไฟล์ดังกล่าวขึ้นเอง โดยเมื่อผู้ใช้งานออกแบบแผนภาพคลาสและส่งออกแผนภาพคลาสให้อยู่ในรูปของไฟล์เอ็กซ์เอ็มไอเสร็จเรียบร้อย ผู้ใช้ต้องนำเข้าไฟล์เอ็กซ์เอ็มไอดังกล่าวเข้าสู่โปรแกรมโดยโปรแกรมจะทำการสร้างไฟล์สนับสนุนทั้ง 4 ไฟล์ขึ้นโดยอัตโนมัติ ได้แก่ ไฟล์เอ็กซ์เอ็มแอลตั้งค่าการใช้งาน เอ็กซ์เอ็มแอลแม็พไฟล์ คลาสที่ใช้ในการแม็พและเอสคิวแอลสคริปต์ โปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนตที่ได้รับการพัฒนาเพื่อสนับสนุนเอ็นไฮเบอร์เนต เวอร์ชัน 1.2 และรองรับความสัมพันธ์ระหว่างวัตถุแบบหนึ่งต่อหนึ่ง (one to one) หนึ่งต่อหลาย (one to many) และหลายต่อหนึ่ง (many to one)

5.2 ปัญหาและข้อจำกัดที่พบจากการวิจัย

เกิดความล่าช้าขณะทำการพัฒนาโปรแกรมดังกล่าวขึ้น เนื่องจากโปรแกรม Visual Paradigm 6.3 เป็นเวอร์ชันที่ผู้วิจัยได้ทำการออกแบบไว้ในช่วงแรก แต่ขณะทำการพัฒนาโปรแกรมมาทาง Visual Paradigm ได้มีการปรับปรุงโปรแกรมดังกล่าวเป็นเวอร์ชัน 7.0 เป็นต้นมา ได้มีการเปลี่ยนรูปแบบของไฟล์เอ็กซ์เอ็มไอที่ได้มีการเปลี่ยนรูปแบบจากเดิมไปมาก ทำให้เสียเวลาเพื่อแก้ไขปรับปรุงโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนตรองรับไฟล์เอ็กซ์เอ็มไอในรูปแบบของเวอร์ชัน 7.0 เป็นต้นมา

5.3 ข้อเสนอแนะ

ผู้วิจัยพบว่าระบบจะมีประสิทธิภาพ และนำไปใช้ประโยชน์มากขึ้นหากมีการพัฒนาและการวิจัยในส่วนต่าง ๆ ดังนี้

1) โปรแกรมสามารถเพิ่มความสามารถในการรองรับความสัมพันธ์ระหว่างวัตถุแบบ many to many

2) งานวิจัยนี้สามารถพัฒนาเพิ่มเติมในส่วนของการรองรับชนิดฐานข้อมูลได้โดยการแก้ไขไฟล์ app.config ของโปรแกรม แต่ต้องทำการพัฒนาในส่วนของ GUI เพื่อให้รองรับกับชนิดของฐานข้อมูลที่เพิ่มขึ้น

3) โปรแกรมสามารถรองรับเอ็นไฮเบอร์เนตในเวอร์ชัน 2.0 ซึ่งขณะที่ผู้พัฒนากำลังพัฒนาโปรแกรมเอ็นไฮเบอร์เนตในเวอร์ชันดังกล่าว เนื่องจากโปรแกรมเอ็นไฮเบอร์เนตจะมีความสามารถในการใช้งาน และประสิทธิภาพสูงขึ้นค่อนข้างมาก หากเทียบกับโปรแกรมเอ็นไฮเบอร์เนตในเวอร์ชันปัจจุบัน

รายการอ้างอิง

[1] Fussell, M.L. Foundations of Object Relational Mapping, 1997.

[2] Bauer, C. and G. King. Hibernate in Action. 2nd ed. Manning Publications, 2005.

[3] Microsoft Developer Network. Dynamic-Link Libraries. [online] http://msdn2.microsoft.com/en-us/library/ms682589(VS.85).aspx [2009, March 31].

[4] Hibernate. NHibernate Reference Documentation, 2008.

[5] Sakowicz, B., et al. JWay - Model driven J2EE application framework. 15th International Conference Mixed Design of Integrated Circuits and Systems. Poland, 2007.

[6] Ziemniak, P., B. Sakowicz, and A. Napieralski. Object Oriented Application Cooperation Methods with Relational Database (ORM) based on J2EE Technology. 9th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics. Ukraine, 2007.

[7] Hibernate. NHibernate Documentation, [online] http://www.hibernate.org [2007, October 12].

[8] Dennis, A., B. H. Wixom, et al. System Analysis and Design with UML Version 2.0. 2nd ed. John Wiley & Sons, 2005.

[9] Larman, C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process. 2nd ed. Prentice Hall PTR, 2001.

[10] Boggs, W., Boggs, M. Mastering UML with Rational Rose. Sybex, 1999.

ภาคผนวก

ภาคผนวก ก

ผลงานตีพิมพ์

Generating NHibernate Assembly Files

Wiroch Sujiraworakun and Wiwat Vatanawood
Department of Computer Engineering, Chulalongkorn University
Bangkok, 10330, Thailand
Email: wiroch.s@student.chula.ac.th wiwat@chula.ac.th

Abstract

To develop the common enterprise business applications, the best practice is to exploit application framework which typically concerns the layered design in order to gain the most flexibility during application maintenance. One of the popular Persistence Layer implementation techniques is the object-relational mapping (ORM) and NHibernate is one of ORM tools developed for C#.

However, to set up the NHibernate assembly files manually is very complicate and error prone. This paper proposes an alternative to generate NHibernate assembly files – XML configuration file, Persistent Class files and XML Mapping files, automatically right away from UML Class Diagram metafile in XMI format. The mostly used class relation constraints are supported – one-to-one, one-to-many and many-to-one relations between classes, via multiplicity symbols.

Moreover, SQL script file containing the set of data definition language (DDL) is generated as well to provide and ensure the creation of the consistent relational database schema to the original UML Class Diagram.

Keywords: ORM, NHibernate, Data Service Layer, Persistent Class, SQL Generator

1. Introduction

Nowadays many enterprise applications have been developed with ORM technology to support the flexibility and maintainability of the whole systems. The developer has reduced development time by 20%-50%. With the ORM technology, the specific designed codes are efficiently reused in the next development procedure [1]. Currently there exist many object-relational mapping tools with .NET framework technology such as NHibernate, Microsoft Language Integrated Query (LINQ) and Component Scalable Logical Architecture .NET (CSLA.NET). Especially, NHibernate comes popular in the very short time because it has been proved and ported from Hibernate Core Tools for java and NHibernate API is very similarly to that of Hibernate. The Hibernate is the most powerful technology [2] although there is open source software which everyone can continuously develop with legal software license. However, one of the difficult tasks for NHibernate developers is a set of the assembly files must be prepared manually. They must have understood NHibernate API and its configuration very clearly in order to do that.

2. Object-relational mapping

The Object-relational mapping (ORM) is a programming technique for converting data between incompatible type systems in relational databases and object-oriented programming languages. In principle, ORM consists of 4 components [3,4]:

- An API for performing basic CRUD operations on objects of persistent classes
- A language or API for specifying queries that refer to classes and properties of classes
- A facility for specifying mapping metadata
- A technique for the ORM implementation to interact with transactional objects to perform dirty checking, lazy association fetching, and other optimization functions

The advantages of ORM are helping design system or applications more flexible and can be modified without problem with applications and relational database. The most important in ORM technology is persistence layer [4].

2.1 Persistent Layer

The persistence layer is a group of classes and components responsible for data storage to, and retrieval from, one or more data stores. This layer necessarily includes a model of the business domain entities. Figure 1 illustrates layered architecture [4].

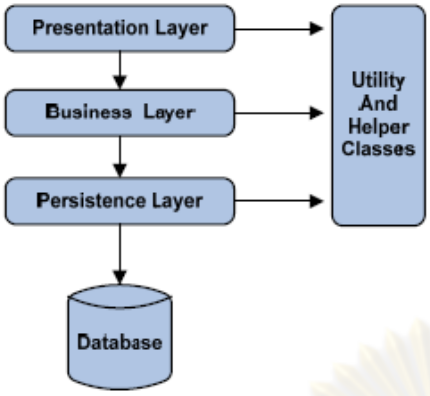


Figure 1. A persistence layer is the basis in a layered architecture [4]

2.2 Mapping Object Relationships

In the Object-Oriented principle, we need to understand and design how the interactions among objects – so called object relationships. There are three types of object relationships: association, aggregation, and composition. The ORM should provide the mapping tools for the mentioned object relationships. There are two categories of object relationships that you need to be concerned with when mapping. [5]

2.2.1 Based on multiplicity

a) **One-to-one:** This is a relationship where the maximum of each of its multiplicities is one, an example of which holds relationship between Employee and Position in figure 2. An employee holds one and only one position and a position may be held by one employee.

b) **One-to-many:** Also known as a many-to-one relationship, this occurs when the maximum of one multiplicity is one and the other is greater than one. An example is the works in relationship between Employee and Division. An employee works in one division and any given division has one or more employees working in it.

c) **Many-to-many:** This is a relationship where the maximum of both multiplicities is greater than one, an example of which is the assigned relationship between Employee and Task. An employee is assigned one or more tasks and each task is assigned to zero or more employees.

2.2.2 Based on direction

a) Uni-directional: A uni-directional relationship when an object knows about the object it is related to but the other objects do not know of the original object.

b) Bi-directional: A bi-directional relationship exists when the objects on both end of the relationship know of each other. [5]

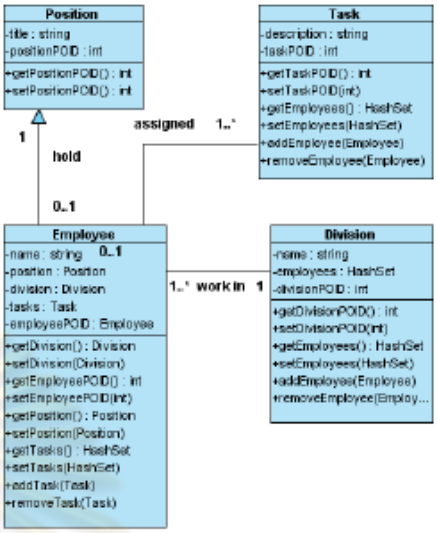


Figure 2. Relationships between objects [5]

3. NHibernate

NHibernate is Hibernate on .NET Framework technology. It handles persisting plain .Net objects to and form underlying relational database. Figure 3 illustrates high-level view of the NHibernate architecture.

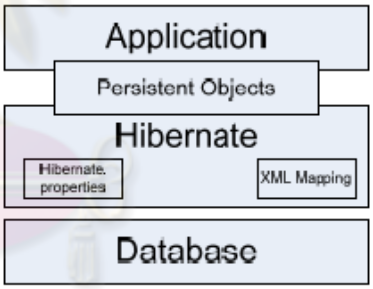


Figure 3. High-level view of the NHibernate architecture [6]

NHibernate is an object-relational mapping tool which has more efficiency and high performance so the procedures to develop applications with NHibernate can explain in figure 4.

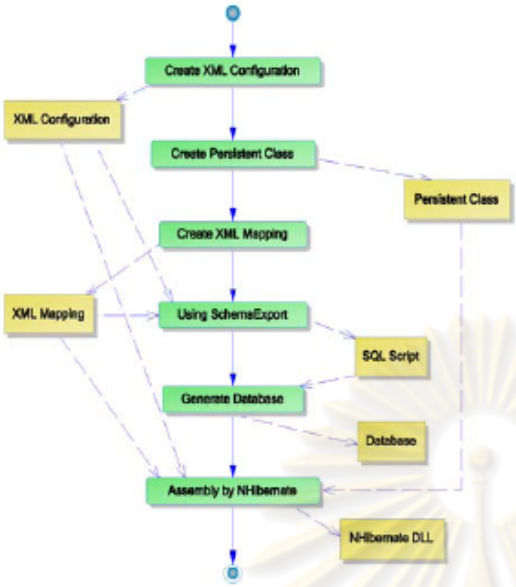


Figure 4. Activity diagram procedures develop application with NHibernate

3.1 XML Configuration File

Define information about database configuration e.g. database class, database provider, connection string, database name, etc.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section
      name="hibernate-configuration"
      type="NHibernate.Cfg.ConfigurationSectionHandler,
NHibernate"/>
  </configSections>

  <hibernate-configuration xmlns="urn:nhibernate-
configuration-2.2">
    <session-factory>
      <property
name="dialect">NHibernate.Dialect.MsSql2000Dialect</pro
perty>
      <property
name="connection.provider">NHibernate.Connection.Driver
ConnectionProvider</property>
      <property
name="connection.connection_string">Server=(local);initial
catalog=quickstart;Integrated Security=SSPI</property>

      <mapping assembly="QuickStart" />
    </session-factory>
  </hibernate-configuration>
</configuration>
```

Figure 5. XML Configuration File [6]

3.2 Persistent Class File

Class defines properties or operations for each attributes in relational database with C# language e.g. get or set.

```
namespace QuickStart
{
  public class Cat
  {
    private string id;
    private string name;
    private char  sex;
    private float  weight;
    public Cat()
    {
    }
    public virtual string Id
    {  get { return id; }
      set { id = value; }
    }
    public virtual string Name
    {  get { return name; }
      set { name = value; }
    }
    public virtual char Sex
    {  get { return sex; }
      set { sex = value; }
    }
    public virtual float Weight
    {  get { return weight; }
      set { weight = value; }
    }
  }
}
```

Figure 6. Persistent Class File [6]

3.3 XML Mapping File

Define each attributes in detail to map between attributes in persistent class and attributes in relation database.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
  namespace="QuickStart" assembly="QuickStart">
  <class name="Cat" table="Cat">
    <id name="Id">
      <column name="CatId" sql-type="char(32)" not-
null="true"/>
      <generator class="uuid.hex" />
    </id>
    <property name="Name">
      <column name="Name" length="16" not-null="true"
/>
    </property>
    <property name="Sex" />
    <property name="Weight" />
  </class>
</hibernate-mapping>
```

Figure 7. XML Mapping File [6]

3.4 SQL Script File

Use for create tables and attributes in relational database.

```
use QuickStart
go

CREATE TABLE Cat (
  CatID char(32) NOT NULL,
  Name nvarchar(16) NOT NULL,
  Sex nchar(1) default NULL,
  Weight real default NULL,
  PRIMARY KEY  (CatID)
)
Go
```

Figure 8. SQL Script [6]

The figure 4 illustrates the typical activity diagram for mapping a single persistent object/class to correspondent a single table in relational database. It happens to be more complicate and inconvenient if we have to a single persistent object/class referring to more than one table in relational database. The developers have more manual work to do with the NHibernate assembly files.

4. Our proposed solution

We intend to propose the automatic software tool to generating the essential NHibernate assembly files as mentioned earlier for the ORM mapping configuration. We propose the direct transformation feature from UML Class diagram into NHibernate assembly files automatically. Our software tool will take the input UML Class diagram in XMI file format and perform the following (shown in figure 9):

- Generate 4 assembly files from XMI file
- SQL Script for database data manipulation such as create table etc.
- Support 3 relational databases - MSSQL, MySQL and PostgreSQL
- Support NHibernate version 1.2 or later

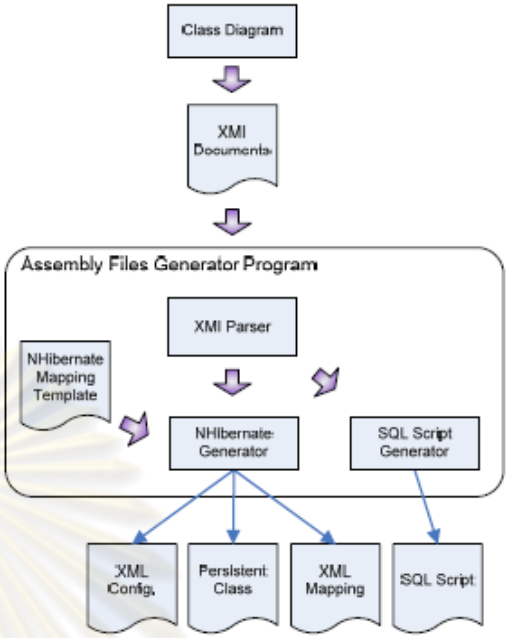


Figure 9. Application procedures

5. The architechture of our software tool

We experiment our software tool with the following components:

- Microsoft Visual Studio 2003, 2008
- Visual Paradigm 6.3
- Microsoft SQL Server 2005 (MSSQL 2005), MySQL 5 and PostgreSQL 8
- NHibernate 1.2 [6]
- Java JDK 1.5
- Microsoft .NET Framework 1.1, 2.0
- Microsoft .NET Framework SDK 1.1

The assembly files generator component illustrates in Figure 10.

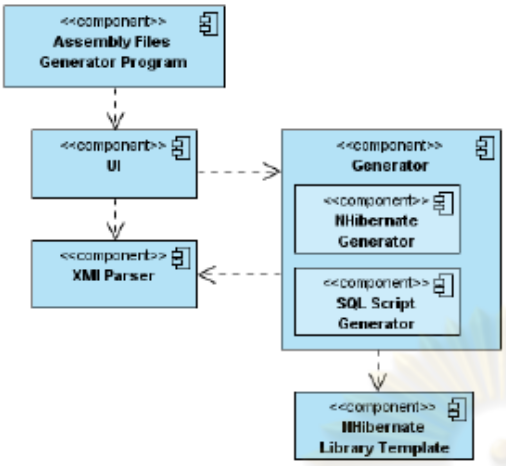


Figure 10. Component diagram of Assembly Files Generator Program

- UI component performs the control and navigation of the software tools.
- Component XMI Parser manipulates and interprets the class diagram in XMI file format
- NHibernate Generator component obtains the data from the given XMI Parser and compare with NHibernate Library Template component to generate XML configuration, persistent class and XML mapping
- SQL Script Generator component operates in similar to NHibernate Generator component but generating related SQL script

The new activity diagram is proposed to generate NHibernate assembly files as shown in figure 11.

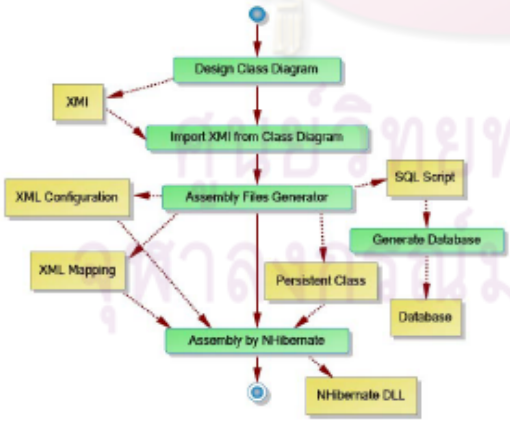


Figure 11. The new activity diagram using our proposed solution to generate assembly files

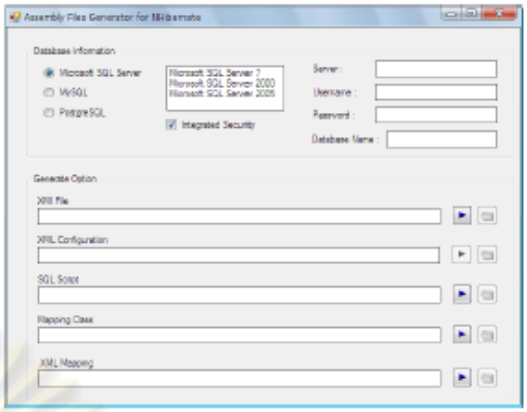


Figure 12. UI of assembly files generator program

6. Type-style and fonts

For an English paper, Times New Roman is used. For a Thai paper, Angsana New is used. Do not use bit-mapped fonts.

7. Conclusion

In conclusion we propose an alternative to prepare NHibernate assembly files automatically. After class designers finish their designs the class diagrams would be interpreted and the correspondent assembly files are generated with ease. The assembly files are XML Configuration file, Persistent Class file, XML Mapping and SQL script file. At the moment, the software tool has been developed to support NHibernate 1.2. The one-to-one, one-to-many and many-to-one object relationships are currently supported.

8. References

[1] I. Khan, "Five Reasons for using an O/R mapping tool," 2005.

[2] P. Ziemniak, B. Sakowicz and A. Napieralski, "Object Oriented Application Cooperation Methods with Relational Database (ORM) based on J2EE Technology," Proc. CAD Systems in Microelectronics, 2007. CADSM '07. 9th International Conference - The Experience of Designing and Applications of, 2007, pp. 327-330.

[3] M.L. Fussell, Foundations of Object Relational Mapping, 1997.

[4] C. Bauer and G. King, Hibernate in Action, Manning Publications Co., 2005.

[5] S.W. Ambler, Agile Database Techniques, John Wiley & Sons, 2003.

[6] Hibernate, NHibernate Reference Documentation Version 1.2.0, 2006.

ภาคผนวก ข

กรณีศึกษาระบบทะเบียนประวัติและการนัดหมายระหว่างแพทย์และคนไข้

เนื่องจากไฟล์ที่ได้จากผลการทดสอบมีจำนวนมาก จึงขอตัดมาเพียงส่วนหนึ่งของแต่ละไฟล์เพื่อมานำเสนอ

ไฟล์ XMLConfig.xml

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section
      name="hibernate-configuration"
      type="NHibernate.Cfg.ConfigurationSectionHandler, NHibernate" />
  </configSections>
  <hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
    <session-factory>
      <property
        name="dialect">NHibernate.Dialect.MsSql2005Dialect</property>
      <property
          name="connection.provider">NHibernate.Connection.DriverConnectionProvider
          </property>
      <property
        name="connection.driver_class">NHibernate.Driver.SqlClientDriver</property>
      <property
        name="connection.connection_string">Server=(local);initial
catalog=TestDB;Integrated Security=SSPI;User Id=sa;Password=1234</property>
      <property
        name="use_outer_join">true</property>
      <mapping
        assembly="QuickStart" />
    </session-factory>
  </hibernate-configuration>
</configuration>
```

ไฟล์ MappingClass.cs

```
namespace QuickStart
{
   public class Person
   {
    private string _FirstName;
    private string _LastName;
    private string _Address;
    private string _Phone;
    private string _BirthDate;
    private int _Age;

    public Person()
    {
    }
    public virtual string FirstName
    {
     get{return _FirstName;}
     set{_FirstName = value;}
    }
    public virtual string LastName
    {
     get{return _LastName;}
     set{_LastName = value;}
    }
    public virtual string Address
    {
     get{return _Address;}
     set{_Address = value;}
    }
```

```
 public virtual string Phone
 {
  get{return _Phone;}
  set{_Phone = value;}
 }
 public virtual string BirthDate
 {
  get{return _BirthDate;}
  set{_BirthDate = value;}
 }
 public virtual int Age
 {
  get{return _Age;}
  set{_Age = value;}
 }
}

public class Employee
{
 private string _FirstName;
 private string _LastName;
 private string _Address;
 private string _Phone;
 private string _BirthDate;
 private int _Age;

 public Employee()
 {
 }
 public virtual string FirstName
```

```
{
 get{return _FirstName;}
 set{_FirstName = value;}
}
public virtual string LastName
{
 get{return _LastName;}
 set{_LastName = value;}
}
public virtual string Address
{
 get{return _Address;}
 set{_Address = value;}
}
public virtual string Phone
{
 get{return _Phone;}
 set{_Phone = value;}
}
public virtual string BirthDate
{
 get{return _BirthDate;}
 set{_BirthDate = value;}
}
public virtual int Age
{
 get{return _Age;}
 set{_Age = value;}
}
}
```

ไฟล์ SQLScript.sql

```
use TestDB
go

CREATE TABLE Person
(
FirstName char(50),
LastName char(50),
Address char(50),
Phone char(50),
BirthDate char(50),
Age int,
)

CREATE TABLE Employee
(
FirstName char(50),
LastName char(50),
Address char(50),
Phone char(50),
BirthDate char(50),
Age int,
)

CREATE TABLE MedicalProfile
(
Amount float,
InsuranceCarrier char(50),
)
```

```
CREATE TABLE Nurse
(
FirstName char(50),
LastName char(50),
Address char(50),
Phone char(50),
BirthDate char(50),
Age int,
)

CREATE TABLE AdministrativeStaff
(
FirstName char(50),
LastName char(50),
Address char(50),
Phone char(50),
BirthDate char(50),
Age int,
)

CREATE TABLE Doctor
(
FirstName char(50),
LastName char(50),
Address char(50),
Phone char(50),
BirthDate char(50),
Age int,
)
```

```
CREATE TABLE HeathTeam
(
CREATE TABLE MedicalHistory
(
HeartDisease char(50),
HighBloodPressure char(50),
Diabetes char(50),
Allergies char(50),
)

CREATE TABLE Appointment
(
Time char(50),
Date char(50),
Reason char(50),
)

CREATE TABLE Bill
(
Date char(50),
Amount float,
Purpose char(50),
)

CREATE TABLE Symptom
(
Name char(50),
)
```

ไฟล์ XMLMapping.xml

```
<?xml version="1.0" encoding="utf-8"?>
<hibernate-mapping namespace="QuickStart" assembly="QuickStart"
xmlns="urn:nhibernate-mapping-2.2">
  <class name="Person" table="Person">
    <property name="FirstName" type="string" length="50" />
    <property name="LastName" type="string" length="50" />
    <property name="Address" type="string" length="50" />
    <property name="Phone" type="string" length="50" />
    <property name="BirthDate" type="string" length="50" />
    <property name="Age" type="int" />
  </class>
  <class name="Employee" table="Employee">
    <property name="FirstName" type="string" length="50" />
    <property name="LastName" type="string" length="50" />
    <property name="Address" type="string" length="50" />
    <property name="Phone" type="string" length="50" />
    <property name="BirthDate" type="string" length="50" />
    <property name="Age" type="int" />
  </class>
  <class name="MedicalProfile" table="MedicalProfile">
    <property name="Amount" type="float" />
    <property name="InsuranceCarrier" type="string" length="50" />
  </class>
  <class name="Nurse" table="Nurse">
    <property name="FirstName" type="string" length="50" />
    <property name="LastName" type="string" length="50" />
    <property name="Address" type="string" length="50" />
    <property name="Phone" type="string" length="50" />
```

```
  <property name="BirthDate" type="string" length="50" />
  <property name="Age" type="int" />
</class>
<class name="AdministrativeStaff" table="AdministrativeStaff">
  <property name="FirstName" type="string" length="50" />
  <property name="LastName" type="string" length="50" />
  <property name="Address" type="string" length="50" />
  <property name="Phone" type="string" length="50" />
  <property name="BirthDate" type="string" length="50" />
  <property name="Age" type="int" />
</class>
<class name="Doctor" table="Doctor">
  <property name="FirstName" type="string" length="50" />
  <property name="LastName" type="string" length="50" />
  <property name="Address" type="string" length="50" />
  <property name="Phone" type="string" length="50" />
  <property name="BirthDate" type="string" length="50" />
  <property name="Age" type="int" />
</class>
<class name="HeathTeam" table="HeathTeam" />
<class name="MedicalHistory" table="MedicalHistory">
  <property name="HeartDisease" type="string" length="50" />
  <property name="HighBloodPressure" type="string" length="50" />
  <property name="Diabetes" type="string" length="50" />
  <property name="Allergies" type="string" length="50" />
</class>
<class name="Appointment" table="Appointment">
  <property name="Time" type="string" length="50" />
  <property name="Date" type="string" length="50" />
  <property name="Reason" type="string" length="50" />
```

```
  </class>
  <class name="Bill" table="Bill">
    <property name="Date" type="string" length="50" />
    <property name="Amount" type="float" />
    <property name="Purpose" type="string" length="50" />
  </class>
  <class name="Symptom" table="Symptom">
    <property name="Name" type="string" length="50" />
  </class>
  <class name="Illness" table="Illness">
    <property name="Description" type="string" length="50" />
  </class>
  <class name="Treatment" table="Treatment">
    <property name="Medication" type="string" length="50" />
    <property name="Instructions" type="string" length="50" />
    <property name="SymptomSeverity" type="string" length="50" />
  </class>
  <class name="Patient" table="Patient">
    <property name="FirstName" type="string" length="50" />
    <property name="LastName" type="string" length="50" />
    <property name="Address" type="string" length="50" />
    <property name="Phone" type="string" length="50" />
    <property name="BirthDate" type="string" length="50" />
    <property name="Age" type="int" />
  </class>
</hibernate-mapping>
```

ภาคผนวก ค

กรณีศึกษาระบบการสั่งซื้อสินค้า

ไฟล์ XMLConfig.xml

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section
      name="hibernate-configuration"
      type="NHibernate.Cfg.ConfigurationSectionHandler, NHibernate" />
  </configSections>
  <hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
    <session-factory>
      <property
        name="dialect">NHibernate.Dialect.MsSql2005Dialect</property>
      <property

name="connection.provider">NHibernate.Connection.DriverConnectionProvider</prope
rty>
      <property
        name="connection.driver_class">NHibernate.Driver.SqlClientDriver</property>
      <property
        name="connection.connection_string">Server=(local);initial
catalog=TestDB;Integrated Security=SSPI;User Id=sa;Password=1234</property>
      <property
        name="use_outer_join">true</property>
      <mapping
        assembly="QuickStart" />
    </session-factory>
  </hibernate-configuration>
</configuration>
```

ไฟล์ MappingClass.cs

```
namespace QuickStart
{
   public class User
   {
    private string _userID;
    private string _password;
    private string _LoginStatus;

    public User()
    {
    }
    public virtual string userID
    {
     get{return _userID;}
     set{_userID = value;}
    }
    public virtual string password
    {
     get{return _password;}
     set{_password = value;}
    }
    public virtual string LoginStatus
    {
     get{return _LoginStatus;}
     set{_LoginStatus = value;}
    }
    public void VerifyLogin()
    {
```

```
 }
}

public class SessionManager
{
 private string _userID;
 private string _DepartmentName;

 public SessionManager()
 {
 }
 public virtual string userID
 {
  get{return _userID;}
  set{_userID = value;}
 }
 public virtual string DepartmentName
 {
  get{return _DepartmentName;}
  set{_DepartmentName = value;}
 }
 public void getUser()
 {
 }
 public void getDepartment()
 {
 }
}

public class Customer
```

```
{
 private string _CustomerName;
 private string _Address;
 private string _Email;
 private int _PhoneNO;
 private string _CreditCardInfo;
 private string _ShippingInfo;
 private string _userID;
 private string _password;
 private string _LoginStatus;

 public Customer()
 {
 }
 public virtual string CustomerName
 {
  get{return _CustomerName;}
  set{_CustomerName = value;}
 }
 public virtual string Address
 {
  get{return _Address;}
  set{_Address = value;}
 }
 public virtual string Email
 {
  get{return _Email;}
  set{_Email = value;}
 }
 public virtual int PhoneNO
```

```
{
 get{return _PhoneNO;}
 set{_PhoneNO = value;}
}
public virtual string CreditCardInfo
{
 get{return _CreditCardInfo;}
 set{_CreditCardInfo = value;}
}
public virtual string ShippingInfo
{
 get{return _ShippingInfo;}
 set{_ShippingInfo = value;}
}
public void Register()
{
}
public void Login()
{
}
public void UpdateProfile()
{
}
public virtual string userID
{
 get{return _userID;}
 set{_userID = value;}
}
public virtual string password
{
```

```
    get{return _password;}
    set{_password = value;}
   }
   public virtual string LoginStatus
   {
    get{return _LoginStatus;}
    set{_LoginStatus = value;}
   }
  }
}
```

ไฟล์ SQLScript.sql

```
use TestDB
go

CREATE TABLE User
(
userID char(50),
password char(50),
LoginStatus char(50),
)

CREATE TABLE SessionManager
(
userID char(50),
DepartmentName char(50),
)

CREATE TABLE Customer
(
CustomerName char(50),
Address char(50),
Email char(50),
PhoneNO int,
CreditCardInfo char(50),
ShippingInfo char(50),
userID char(50),
password char(50),
LoginStatus char(50),
)
```

```
CREATE TABLE ShoppingCart
(
CartID int,
ProductID int,
Quantity int,
DateAdded int,
)

CREATE TABLE Orders
(
OrderID int,
DateCreated char(50),
DateShipped char(50),
CustomerName char(50),
CustomerID char(50),
Status char(50),
ShippingID char(50),
)

CREATE TABLE Administrator
(
adminName char(50),
Email char(50),
userID char(50),
password char(50),
LoginStatus char(50),
)

CREATE TABLE ShippingInfo
```

```
(
ShippingID int,
ShippingType char(50),
ShippingCost int,
ShippingRegionID int
)

CREATE TABLE OrderDetail
(
OrderID int,
ProductID int,
ProductName char(50),
Quantity int,
UnitCost float,
SubTotal float
)

CREATE TABLE Department
(
DepartmentID int,
Name char(50),
Description char(50),
)
```

ไฟล์ XMLMappings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<hibernate-mapping namespace="QuickStart" assembly="QuickStart"
xmlns="urn:nhibernate-mapping-2.2">
  <class name="User" table="User">
    <property name="userID" type="string" length="50" />
    <property name="password" type="string" length="50" />
    <property name="LoginStatus" type="string" length="50" />
  </class>
  <class name="SessionManager" table="SessionManager">
    <property name="userID" type="string" length="50" />
    <property name="DepartmentName" type="string" length="50" />
  </class>
  <class name="Customer" table="Customer">
    <property name="CustomerName" type="string" length="50" />
    <property name="Address" type="string" length="50" />
    <property name="Email" type="string" length="50" />
    <property name="PhoneNO" type="int" />
    <property name="CreditCardInfo" type="string" length="50" />
    <property name="ShippingInfo" type="string" length="50" />
    <property name="userID" type="string" length="50" />
    <property name="password" type="string" length="50" />
    <property name="LoginStatus" type="string" length="50" />
  </class>
  <class name="ShoppingCart" table="ShoppingCart">
    <property name="CartID" type="int" />
    <property name="ProductID" type="int" />
    <property name="Quantity" type="int" />
    <property name="DateAdded" type="int" />
```

```
</class>
<class name="Orders" table="Orders">
  <property name="OrderID" type="int" />
  <property name="DateCreated" type="string" length="50" />
  <property name="DateShipped" type="string" length="50" />
  <property name="CustomerName" type="string" length="50" />
  <property name="CustomerID" type="string" length="50" />
  <property name="Status" type="string" length="50" />
  <property name="ShippingID" type="string" length="50" />
</class>
<class name="Administrator" table="Administrator">
  <property name="adminName" type="string" length="50" />
  <property name="Email" type="string" length="50" />
  <property name="userID" type="string" length="50" />
  <property name="password" type="string" length="50" />
  <property name="LoginStatus" type="string" length="50" />
</class>
<class name="ShippingInfo" table="ShippingInfo">
  <property name="ShippingID" type="int" />
  <property name="ShippingType" type="string" length="50" />
  <property name="ShippingCost" type="int" />
  <property name="ShippingRegionID" type="int" />
</class>
<class name="OrderDetail" table="OrderDetail">
  <property name="OrderID" type="int" />
  <property name="ProductID" type="int" />
  <property name="ProductName" type="string" length="50" />
  <property name="Quantity" type="int" />
  <property name="UnitCost" type="float" />
  <property name="SubTotal" type="float" />
```

```
  </class>
  <class name="Department" table="Department">
    <property name="DepartmentID" type="int" />
    <property name="Name" type="string" length="50" />
    <property name="Description" type="string" length="50" />
  </class>
  <class name="Category" table="Category">
    <property name="CategoryID" type="int" />
    <property name="DepartmentID" type="string" length="50" />
    <property name="CategoryName" type="string" length="50" />
    <property name="Description" type="string" length="50" />
  </class>
  <class name="Product" table="Product">
    <property name="ProductID" type="int" />
    <property name="Name" type="string" length="50" />
    <property name="Description" type="string" length="50" />
    <property name="Price" type="int" />
    <property name="ImageFileName" type="string" length="50" />
  </class>
  <class name="Illness" table="Illness">
    <property name="Description" type="string" length="50" />
  </class>
  <class name="Treatment" table="Treatment">
    <property name="Medication" type="string" length="50" />
    <property name="Instructions" type="string" length="50" />
    <property name="SymptomSeverity" type="string" length="50" />
  </class>
</hibernate-mapping>
```

ภาคผนวก ง

กรณีศึกษาระบบการยืมคืนหนังสือและวารสารของห้องสมุด

ไฟล์ XMLConfig.xml

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section
      name="hibernate-configuration"
      type="NHibernate.Cfg.ConfigurationSectionHandler, NHibernate" />
  </configSections>
  <hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
    <session-factory>
      <property
        name="dialect">NHibernate.Dialect.MsSql2005Dialect</property>
      <property
          name="connection.provider">NHibernate.Connection.DriverConnectionProvider
          </property>
      <property
        name="connection.driver_class">NHibernate.Driver.SqlClientDriver</property>
      <property
        name="connection.connection_string">Server=(local);initial
catalog=TestDB;Integrated Security=SSPI;User Id=sa;Password=1234</property>
      <property
        name="use_outer_join">true</property>
      <mapping
        assembly="QuickStart" />
    </session-factory>
  </hibernate-configuration>
</configuration>
```

ไฟล์ MappingClass.cs

```
namespace QuickStart
{
   public class Employee
   {
    private string _Name;
    private float _Salary;
    private string _Gender;

    public Employee()
    {
    }
    public virtual string Name
    {
     get{return _Name;}
     set{_Name = value;}
    }
    public virtual float Salary
    {
     get{return _Salary;}
     set{_Salary = value;}
    }
    public virtual string Gender
    {
     get{return _Gender;}
     set{_Gender = value;}
    }
   }
```

```
public class Accountant
{
 private string _Designation;
 private string _Name;
 private float _Salary;
 private string _Gender;

 public Accountant()
 {
 }
 public virtual string Designation
 {
  get{return _Designation;}
  set{_Designation = value;}
 }
 public void ReceiveFine()
 {
 }
 public virtual string Name
 {
  get{return _Name;}
  set{_Name = value;}
 }
 public virtual float Salary
 {
  get{return _Salary;}
  set{_Salary = value;}
 }
 public virtual string Gender
 {
```

```
  get{return _Gender;}
  set{_Gender = value;}
 }
}

public class Librarian
{
 private string _Designation;
 private string _Name;
 private float _Salary;
 private string _Gender;

 public Librarian()
 {
 }
 public virtual string Designation
 {
  get{return _Designation;}
  set{_Designation = value;}
 }
 public void IssueBook()
 {
 }
 public void IssueJournal()
 {
 }
 public virtual string Name
 {
  get{return _Name;}
  set{_Name = value;}
```

```
    }
    public virtual float Salary
    {
     get{return _Salary;}
     set{_Salary = value;}
    }
    public virtual string Gender
    {
     get{return _Gender;}
     set{_Gender = value;}
    }
   }
}
```

ไฟล์ SQLScript.sql

```
use
go

CREATE TABLE Employee
(
Name char(50),
Salary float
Gender char(50),
)

CREATE TABLE Accountant
(
Designation char(50),
Name char(50),
Salary float
Gender char(50),
)

CREATE TABLE Librarian
(
Designation char(50),
Name char(50),
Salary float
Gender char(50),
)

CREATE TABLE DeputyLibrarian
(
```

```
Designation char(50),
Name char(50),
Salary float
Gender char(50),
)

CREATE TABLE Fine
(
Amount float
ReceiveDate char(50),
)

CREATE TABLE Library
(
LibraryName char(50),
OpenTime char(50),
CloseTime char(50),
)

CREATE TABLE Journal
(
Name char(50),
ISBN char(50),
Publisher char(50),
DatePubish char(50),
)

CREATE TABLE Book
(
Name char(50),
```

```
AuthorName char(50),
ISBN char(50),
Publisher char(50),
DateOfPublish char(50),
)

CREATE TABLE Student
(
Name char(50),
ID char(50),
Gender char(50),
)

CREATE TABLE Faculty
(
Name char(50),
Salary float
Gender char(50),
)

CREATE TABLE Symptom
(
Name char(50),
)

CREATE TABLE Illness
(
Description char(50),
)
```

ไฟล์ XMLMapping.xml

```
<?xml version="1.0" encoding="utf-8"?>
<hibernate-mapping namespace="QuickStart" assembly="QuickStart"
xmlns="urn:nhibernate-mapping-2.2">
  <class name="Employee" table="Employee">
    <property name="Name" type="string" length="50" />
    <property name="Salary" type="float" />
    <property name="Gender" type="string" length="50" />
  </class>
  <class name="Accountant" table="Accountant">
    <property name="Designation" type="string" length="50" />
    <property name="Name" type="string" length="50" />
    <property name="Salary" type="float" />
    <property name="Gender" type="string" length="50" />
  </class>
  <class name="Librarian" table="Librarian">
    <property name="Designation" type="string" length="50" />
    <property name="Name" type="string" length="50" />
    <property name="Salary" type="float" />
    <property name="Gender" type="string" length="50" />
  </class>
  <class name="DeputyLibrarian" table="DeputyLibrarian">
    <property name="Designation" type="string" length="50" />
    <property name="Name" type="string" length="50" />
    <property name="Salary" type="float" />
    <property name="Gender" type="string" length="50" />
  </class>
  <class name="Fine" table="Fine">
    <property name="Amount" type="float" />
```

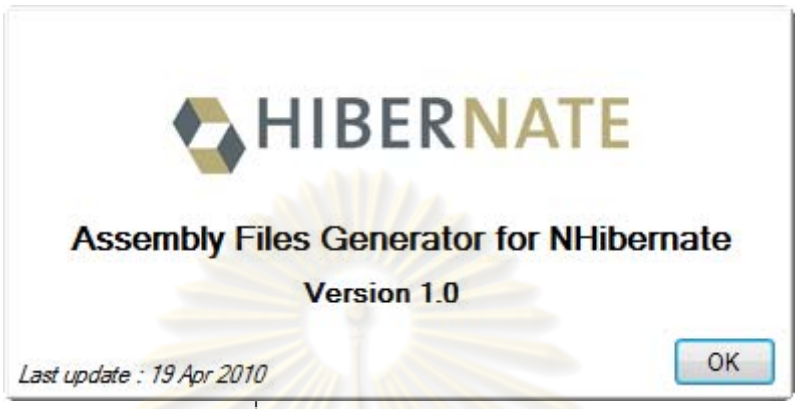
```
  <property name="ReceiveDate" type="string" length="50" />
</class>
<class name="Library" table="Library">
  <property name="LibraryName" type="string" length="50" />
  <property name="OpenTime" type="string" length="50" />
  <property name="CloseTime" type="string" length="50" />
</class>
<class name="Journal" table="Journal">
  <property name="Name" type="string" length="50" />
  <property name="ISBN" type="string" length="50" />
  <property name="Publisher" type="string" length="50" />
  <property name="DatePubish" type="string" length="50" />
</class>
<class name="Book" table="Book">
  <property name="Name" type="string" length="50" />
  <property name="AuthorName" type="string" length="50" />
  <property name="ISBN" type="string" length="50" />
  <property name="Publisher" type="string" length="50" />
  <property name="DateOfPublish" type="string" length="50" />
</class>
<class name="Student" table="Student">
  <property name="Name" type="string" length="50" />
  <property name="ID" type="string" length="50" />
  <property name="Gender" type="string" length="50" />
</class>
<class name="Faculty" table="Faculty">
  <property name="Name" type="string" length="50" />
  <property name="Salary" type="float" />
  <property name="Gender" type="string" length="50" />
</class>
```

```
  <class name="Symptom" table="Symptom">
    <property name="Name" type="string" length="50" />
  </class>
  <class name="Illness" table="Illness">
    <property name="Description" type="string" length="50" />
  </class>
  <class name="Treatment" table="Treatment">
    <property name="Medication" type="string" length="50" />
    <property name="Instructions" type="string" length="50" />
    <property name="SymptomSeverity" type="string" length="50" />
  </class>
  <class name="Patient" table="Patient">
    <property name="FirstName" type="string" length="50" />
    <property name="LastName" type="string" length="50" />
    <property name="Address" type="string" length="50" />
    <property name="Phone" type="string" length="50" />
    <property name="BirthDate" type="string" length="50" />
    <property name="Age" type="int" />
  </class>
  <class name="User" table="User">
    <property name="userID" type="string" length="50" />
    <property name="password" type="string" length="50" />
    <property name="LoginStatus" type="string" length="50" />
  </class>
  <class name="SessionManager" table="SessionManager">
    <property name="userID" type="string" length="50" />
    <property name="DepartmentName" type="string" length="50" />
  </class>
</hibernate-mapping>
```

ภาคผนวก จ

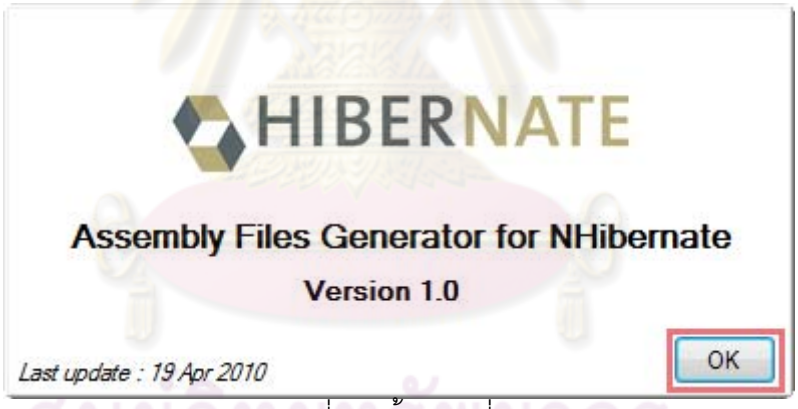
วิธีการใช้งานโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต

1. ทำการเปิดโปรแกรม Assembly Files Generator For NHibernate
2. เมื่อเปิดโปรแกรมขึ้นมาแล้วจะพบหน้าต่างแสดงชื่อโปรแกรม ดังรูป



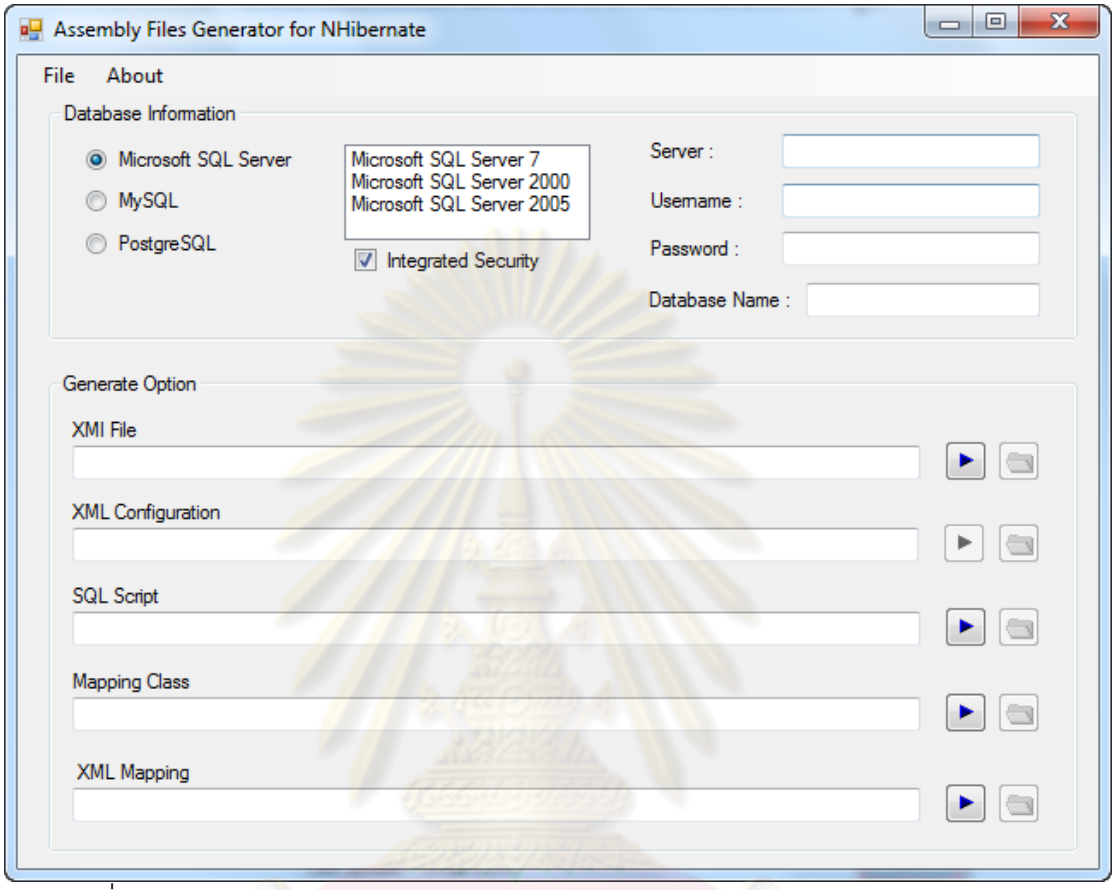
รูปที่ จ.1 หน้าจอ Splash Screen

3. คลิกที่ปุ่ม OK



รูปที่ จ.2 ขั้นตอนที่ 3

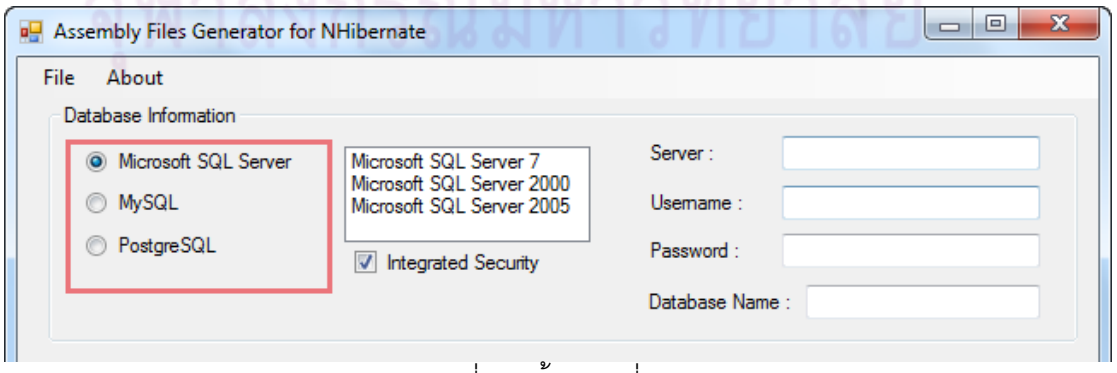
4. โปรแกรมจะเข้าสู่หน้าจอหลัก



รูปที่ จ.3 หน้าจอหลักของโปรแกรมสร้างไฟล์สนับสนุนการใช้งานสำหรับเอ็นไฮเบอร์เนต (ขั้นตอนที่ 4)

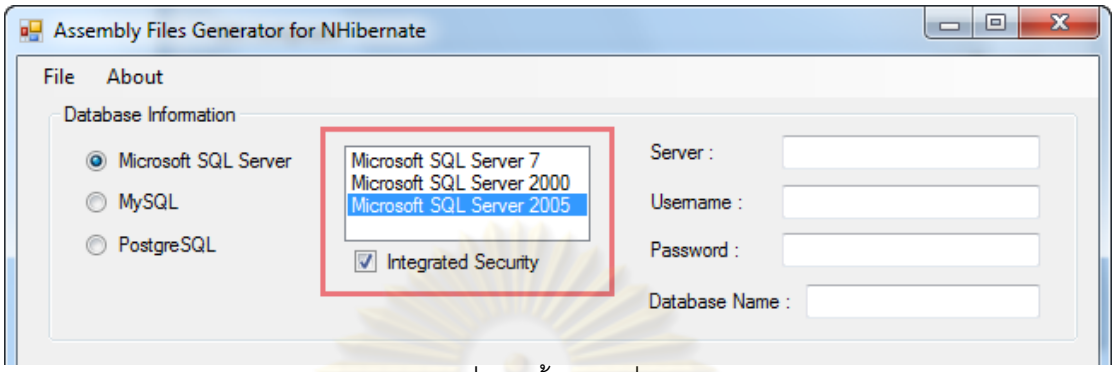
5. เริ่มต้นผู้ใช้งานต้องป้อนข้อมูลเกี่ยวกับฐานข้อมูลที่ต้องการทำการแม็พ

   5.1 เลือกชนิดของฐานข้อมูลที่ใช้



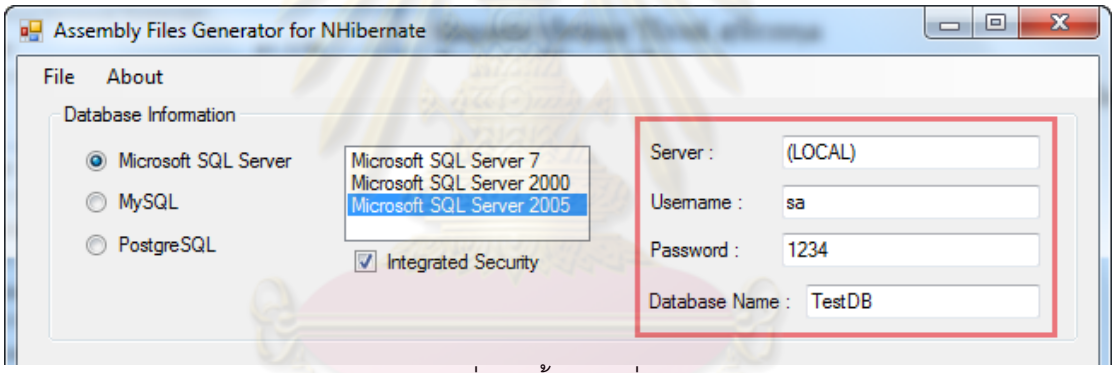
รูปที่ จ.4 ขั้นตอนที่ 5.1

5.2 เลือกเวอร์ชั่นของฐานข้อมูล



รูปที่ จ.5 ขั้นตอนที่ 5.2

5.3 ป้อนรายละเอียดของฐานข้อมูล



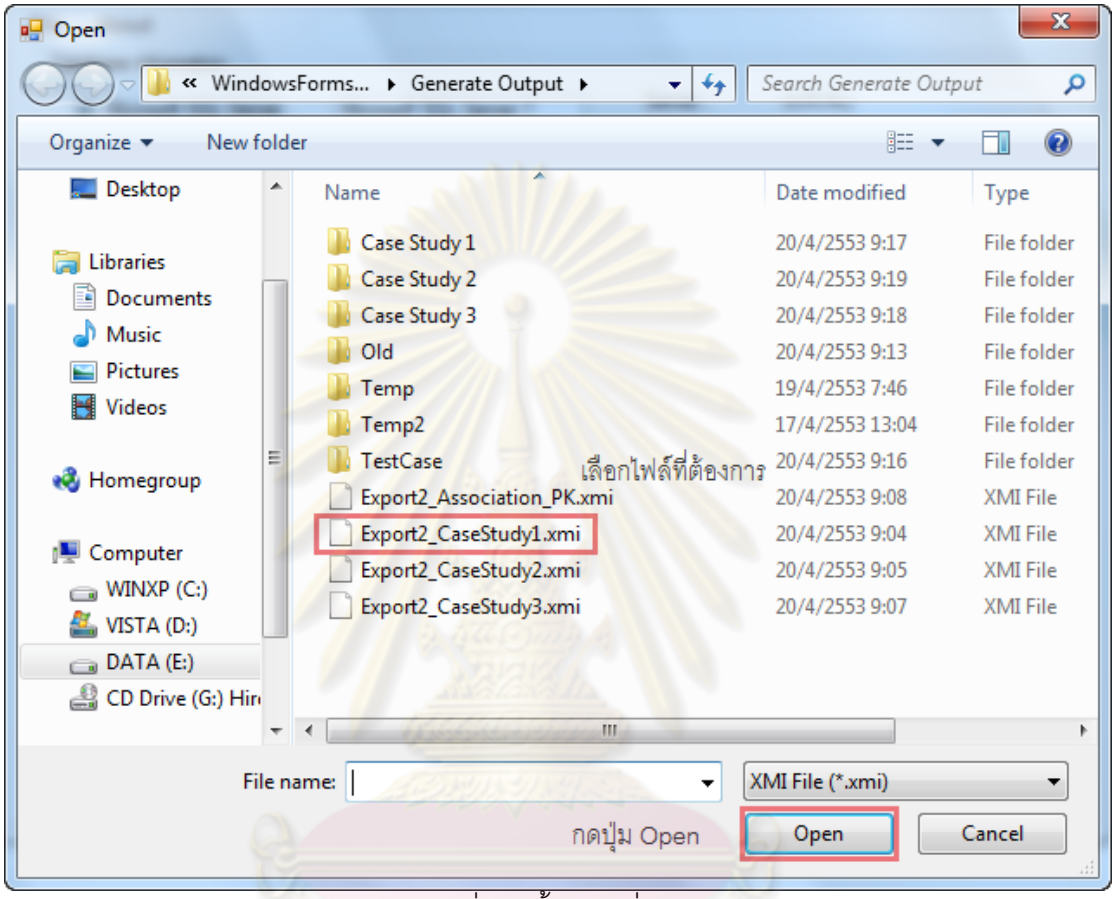
รูปที่ จ.6 ขั้นตอนที่ 5.3

6. หลังจากป้อนข้อมูลเสร็จ ผู้ใช้ต้องทำการนำเข้าไฟล์เอ็กซ์เอ็มไอ

6.1 กดที่ปุ่ม 

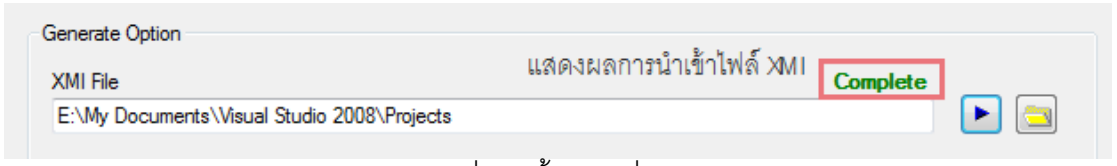
รูปที่ จ.7 ขั้นตอนที่ 6.1

6.2 โปรแกรมจะมีหน้าต่างที่ใช้ในการนำเข้าไฟล์เอ็กซ์เอ็มไอ ให้ผู้ใช้เลือกไฟล์เอ็กซ์เอ็มไอที่ได้จากการส่งออกจากโปรแกรม Visual Paradigm 7.2 แล้วกดปุ่ม Open



รูปที่ จ.8 ขั้นตอนที่ 6.2

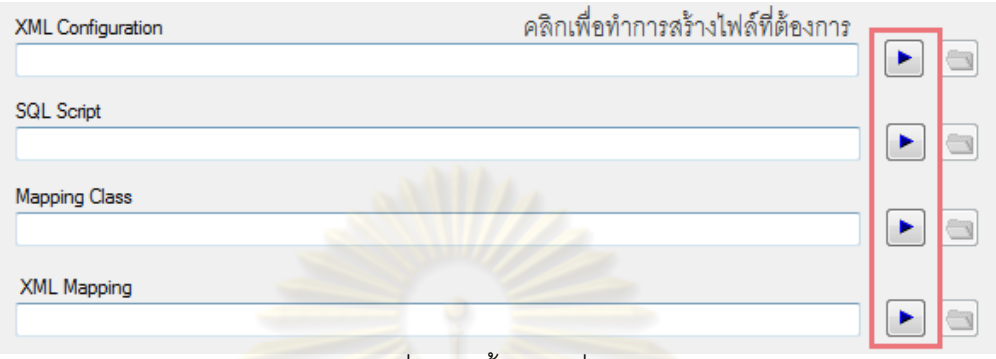
6.3 โปรแกรมจะแสดงสถานะหลังการนำเข้าไฟล์เอ็กซ์เอ็มไอว่าสามารถนำเข้าได้หรือไม่ โดยจะแสดงคำว่า Complete หากไม่มีปัญหาขณะทำการนำเข้าและแสดงคำว่า Fail ถ้าหากเกิดปัญหาขณะทำการนำเข้า



รูปที่ จ.9 ขั้นตอนที่ 6.3

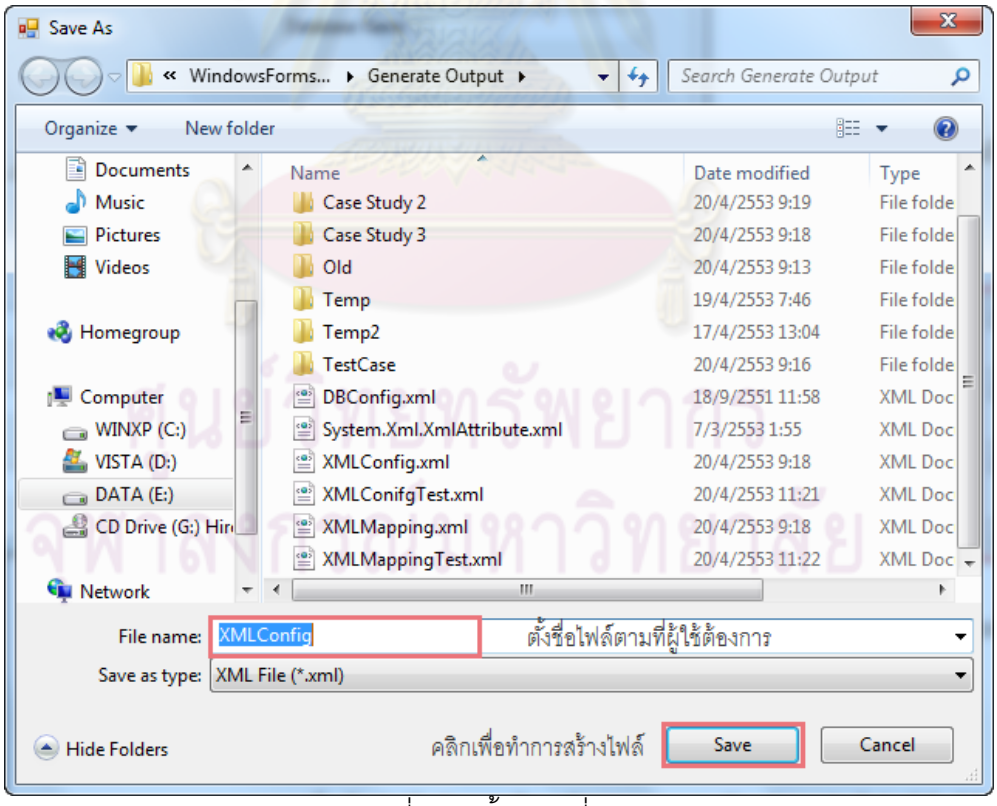
7. เมื่อทำการนำเข้าไฟล์เอ็กซ์เอ็มไอแล้ว ผู้ใช้สามารถสร้างไฟล์สนับสนุนทั้ง 4 ไฟล์ได้

7.1 โดยการกดปุ่ม ▶ ที่ได้ตามตำแหน่งดังกล่าว



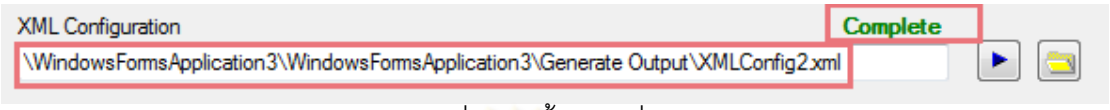
รูปที่ จ.10 ขั้นตอนที่ 7.1

7.2 โปรแกรมจะแสดงหน้าต่างเพื่อให้ทำการเก็บไฟล์ดังกล่าว โดยผู้ใช้สามารถตั้งชื่อไฟล์ได้ตามต้องการ จากนั้นกดปุ่ม Save



รูปที่ จ.11 ขั้นตอนที่ 7.2

7.3 เมื่อสร้างไฟล์สนับสนุนแล้วจะมีสถานะบอกเช่นเดียวกับการนำเข้าไฟล์เอ็กซ์เอ็มไอ โดยในส่วนของ [ ] จะแสดงที่อยู่ของไฟล์ และมีปุ่ม  เพื่อให้ผู้ใช้สามารถไปยังที่เก็บไฟล์ที่ได้สร้างไว้



รูปที่ จ.12 ขั้นตอนที่ 7.3

8. เมื่อทำการใช้งานโปรแกรมเสร็จแล้วสามารถปิดโปรแกรมได้ทันที

ประวัติผู้เขียนวิทยานิพนธ์

นายวิโรจน์ สุจิรวรกุล เกิดเมื่อวันที่ 12 ตุลาคม 2526 สำเร็จการศึกษาวิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมเครื่องกล จากสถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ ในปีการศึกษา 2548 เข้าศึกษาต่อระดับวิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัยเมื่อ พ.ศ.2549