โปรแกรมตัวสร้างรหัสคำสั่งจากแผนภาพสถานะ

STATE DIAGRAM CODE GENERATOR

โดย

นัฐวุธ พุ่มพวง

NATAWUT PHUMPHUANG

อาจารย์ที่ปรึกษา

ผศ.ดร.ภัทรชัย ลลิตโรจน์วงศ์

รายงานนี้เป็นส่วนหนึ่งของวิชาโครงการพัฒนาระบบงาน

หลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ภาคเรียนที่ 1 ปีการศึกษา 2551

STATE DIAGRAM CODE GENERATOR

NATAWUT PHUMPHUANG

A SYSTEM DEVELOPMENT PROJECT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF SCIENCE PROGRAM IN INFORMATION TECHNOLOGY

FACULTY OF INFORMATION TECNOLOGY

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

1/ 2008

หัวข้อ	โปรแกรมตัวสร้างรหัสคำสั่งจากแผนภาพสถานะ
นักศึกษา	นายนัฐวุธ พุ่มพวง
รหัสนักศึกษา	46066739
ปริญญา	วิทยาศาสตรมหาบัณฑิต
สาขาวิชา	เทคโนโลยีสารสนเทศ
แขนงวิชา	วิทยาการสารสนเทศ
ปีการศึกษา	2551
อาจารย์ที่ปรึกษา	ผศ.ดร.ภัทรชัย ลลิตโรจน์วงศ์

บทคัดย่อ

แผนภาพสถานะ ที่ใช้ในการวิเคราะห์และออกแบบระบบ จะใช้ในการแสดงสถานะของวัตถุหนึ่ง รวมไปถึงเหตุการณ์ต่างๆที่สามารถทำให้สถานะของวัตถุนั้นเปลี่ยนแปลงไป และการกระทำที่เกิดขึ้นเมื่อสถานะของระบบเปลี่ยนไป ซึ่งส่วนมาก การพัฒนาโปรแกรมให้เป็นไปตามแผนภาพสถานะนั้น จะเป็นไปในลักษณะเดียวกันซ้ำๆหลายๆครั้ง เพื่อเป็นการลดความซ้ำซ้อนและข้อผิดพลาดที่อาจเกิดขึ้นจากการพัฒนาโปรแกรมที่อ้างอิงจากแผนภาพสถานะ โครงการนี้จะนำเสนอการออกแบบ และพัฒนาโปรแกรมต้นแบบที่ใช้สำหรับอ่านข้อมูลแผนภาพสถานะที่อยู่ในรูปแบบ XML Metadata Interchange (XMI) ที่ได้จากโปรแกรม Rational Rose เพื่อแปลงเป็นแม่แบบของรหัสคำสั่งในรูปแบบภาษา Java สำหรับใช้ในการพัฒนาโปรแกรมต่อไป

Title	State Diagram Code Generator
Student	Mr. Natawut Phumphuang
Student ID.	46066739
Degree	Master of Science
Programme	Information Science
Academic Year	2008
Advisor	Asst. Prof. Dr. Pattarachai Lalitrojwong

ABSTRACT

State diagrams are used to describe the behavior of a system. They describe all of the possible states of an object as events occur. Each diagram usually represents objects of a single class and tracks the different states of its objects through the system. State diagram generally uses in implementation of repeating actions for several times again and again. For reduce the redundancy and mistakes in source code during the development, this project presents a solution generating draft source code from a state diagram in XML Metadata Interchange (XMI) format from Rational Rose Software, for future detail coding in Java syntax.

กิตติกรรมประกาศ

โครงการนี้สำเร็จลุล่วงได้ด้วยการได้รับความช่วยเหลือและความกรุณาจากบุคคลต่างๆ เหล่านี้

1. ขอขอบพระคุณ บิดา มารดา ที่สนับสนุนการเรียนในทุกๆด้าน
2. ขอขอบพระคุณ ผศ.ดร.ภัทรชัย ลลิตโรจน์วงศ์ อาจารย์ที่ปรึกษาโครงการ ที่คอยให้ความรู้ คำปรึกษา คำแนะนำ และกำลังใจด้วยความ อบอุ่นและเต็มใจ รวมถึงให้การสนับสนุนในทุกด้านเป็นอย่างดีที่สุด
3. ขอขอบพระคุณอาจารย์ผู้สอนในคณะเทคโนโลยีสารสนเทศทุกท่าน ที่ได้ให้ความรู้ต่างๆ เพื่อนำมาใช้เป็นส่วนช่วยในการพัฒนาโครงการนี้
4. ขอขอบพระคุณ คุณมานะชัย อุดมดี หัวหน้างานที่สนับสนุน และให้โอกาสในการศึกษาหลักสูตรดังกล่าวจนสำเร็จ
5. ขอขอบคุณเพื่อนๆ Stat 17 ที่คอยให้กำลังใจ และเป็นห่วงเสมอมา
6. ขอขอบคุณเพื่อนๆ IS 16.1 ที่คอยช่วยเหลือด้านข้อมูลและข่าวสารอย่างดียิ่ง
7. ขอขอบคุณเพื่อนๆ และน้องๆ ที่บริษัท ซอฟท์ซวีท จำกัด ที่ช่วยเป็นกำลังใจ และช่วยเหลือในด้านต่างๆด้วยความเต็มใจ

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญรูป

รูปที่ หน้า

สารบัญรูป (ต่อ)

รูปที่ หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาของปัญหา

ซอฟท์แวร์ในปัจจุบันมีการพัฒนาให้มีความซับซ้อน สามารถครอบคลุมการทำงานที่หลากหลายได้ ในขั้นตอนการสร้างสรรซอฟท์แวร์ การออกแบบถือเป็นสิ่งสำคัญในการกำหนดทิศทาง รูปแบบ รวมไปถึงขอบเขตการทำงานของตัวซอฟท์แวร์ เครื่องมือในการออกแบบ และเอกสารการออกแบบต่างๆ จึงมีอยู่มากมายสำหรับสื่อถึงสิ่งที่ผู้ออกแบบต้องการให้ซอฟท์แวร์เป็นได้อย่างถูกต้องอย่างละเอียดครบถ้วน และเพื่อให้ซอฟท์แวร์สามารถทำงานได้ตามที่ผู้ออกแบบได้ออกแบบไว้ กระบวนการหรือขั้นตอนในการพัฒนาซอฟท์แวร์ จึงเป็นส่วนสำคัญอีกส่วนหนึ่ง ที่จะทำให้ซอฟท์แวร์นั้นทำงานได้ถูกต้อง ดังนั้น จึงมีการสร้างเครื่องมือต่างๆ ที่ช่วยในการพัฒนาซอฟท์แวร์ทำได้อย่างรวดเร็ว และผิดพลาดน้อยที่สุด

การออกแบบขั้นตอนการทำงานที่อยู่ในรูปของแผนภาพเครื่องสถานะจำกัด (Finite State Machine Diagram) เป็นการออกแบบขั้นตอนของซอฟท์แวร์ที่มีการทำงานในลักษณะที่เป็นการเปลี่ยนสถานะตามเงื่อนไขต่างๆที่กำหนด และสามารถนำมาประยุกต์กับซอฟท์แวร์ที่มีลักษณะการตรวจสอบเงื่อนไขในแต่ละลำดับขั้นก่อนการทำงานใดๆได้ โดยที่แผนภาพเครื่องสถานะจำกัดสามารถเขียนโดยใช้รูปแบบและสัญลักษณ์ของแผนภาพสถานะ (State Diagram) ตามมาตรฐานของยูเอ็มแอล ซึ่งในการพัฒนาซอฟท์แวร์ให้มีการทำงานเป็นไปตามแผนภาพสถานะนั้น อาจต้องมีการทำซ้ำ และมีการตรวจสอบเงื่อนไขต่างๆหลายจุด หากแผนภาพสถานะนั้นมีการออกแบบไว้อย่างซับซ้อน การพัฒนาซอฟท์แวร์ก็จะมีความซับซ้อนตามไปด้วย จึงทำให้มีความเสี่ยงที่จะเกิดข้อผิดพลาดในระหว่างขั้นตอนกระบวนการการพัฒนาซอฟท์แวร์ได้

โปรแกรมตัวสร้างรหัสคำสั่ง (Code Generator) เป็นเครื่องมือที่ช่วยให้ผู้พัฒนาลดขั้นตอนการทำงานในการเขียนรหัสคำสั่ง ตามเอกสารหรือแผนภาพใดๆ และยังช่วยลดความผิดพลาดที่อาจเกิดขึ้นระหว่างการเขียนรหัสคำสั่ง เพื่อให้เป็นไปตามที่ผู้ออกแบบซอฟท์แวร์ได้ออกแบบไว้ ในโครงการนี้จะเสนอแนวทางในการพัฒนาโปรแกรมตัวสร้างรหัสคำสั่งจากแผนภาพสถานะที่ออกแบบโดยใช้โปรแกรม Rational Rose โดยจะใช้รายละเอียดจากไฟล์ข้อมูลที่อยู่ในรูปของ XMI ที่ถูกสร้างโดยโปรแกรม Rational Rose นำมาแปลงให้เป็นโครงสร้างรหัสคำสั่งภาษา Java เพื่อให้ผู้พัฒนาสามารถนำไปใช้พัฒนาซอฟท์แวร์ให้มีความสมบูรณ์ และถูกต้องตามที่ออกแบบไว้ต่อไปได้

1.2 วัตถุประสงค์ของโครงงาน

โปแกรมสร้างรหัสคำสั่งจากแผนภาพสถานะนี้ได้แนวคิดจากโปรแกรมแปลงภาษาไทยให้เป็นอักษรเบลล์ ซึ่งการแปลงภาษาไทยเป็นอักษรเบลล์นั้นจะมีกฎต่างๆในการแปลงลักษณะคล้ายการเข้ารหัส มากกว่าการแปลภาษาโดยตรง ซึ่งโปรแกรมที่ได้ทดลองศึกษาใช้การแปลงภาษาโดยการเทียบกับพจนานุกรม ดังนั้นข้อผิดพลาดที่อาจเกิดขึ้นได้คือ เมื่อมีการพิมพ์ผิดเล็กน้อย หรือคำศัพท์เป็นคำศัพท์ใหม่ หรือพจนานุกรมมีคำศัพท์ไม่ครบ โปรแกรมแปลงนี้จะไม่สามารถแปลงอักษรเบลล์ออกมาได้อย่างถูกต้อง

เนื่องจากการแปลงภาษาไทยให้เป็นอักษรเบลล์นั้นมีกฎการแปลงที่สามารถระบุได้ตายตัว สามารถเขียนในลักษณะของแผนภาพสถานะจำกัดได้ แต่เนื่องจากกฎเกณฑ์ต่างๆที่มีอยู่มากมาย ทำให้การเขียนโปรแกรมตมกฎให้ครบถ้วนอาจต้องใช้เวลาและเกิดข้อผิดพลาดขึ้นได้ง่าย ดังนั้น โครงงานนี้จึงถูกพัฒนาขึ้นเพื่อช่วยให้การเขียนโปรแกรมแปลงภาษาไทยเป็นอักษรเบลล์หรือโปรแกรมอื่นๆในลักษณะเดียวกันนี้ สามารถทำได้ง่ายขึ้น ลดข้อผิดพลาด ลดการเขียนรหัสคำสั่งที่ซ้ำๆกันลง และช่วยให้การทำงานของโปรแกรมเป็นไปได้อย่างถูกต้องตามที่ออกแบบไว้ในรูปแบบของแผนภาพสถานะ

1.3 ขอบเขตของการพัฒนาโครงงาน

1. ออกแบบ และพัฒนาโปรแกรมที่ใช้ในการสร้างรหัสคำสั่งจากแผนภาพสถานะ ที่อยู่ในรูปของไฟล์ XMI ซึ่งถูกส่งออกมาจากโปรแกรม Rational Rose ซึ่งประกอบไปด้วยส่วนเฉพาะต่างๆดังนี้
   - ส่วนในการอ่านข้อมูลจากไฟล์ XMI
   - ส่วนของวัตถุที่บันทึกข้อมูลสำหรับใช้ในการสร้างรหัสคำสั่ง
   - ส่วนในการสร้างรหัสคำสั่ง
2. พัฒนาโปรแกรมให้สามารถสร้างรหัสคำสั่งในรูปแบบของโครงสร้างภาษา Java เพื่อให้ผู้พัฒนาสามารถพัฒนาซอฟท์แวร์ต่อไปให้สมบูรณ์ได้อย่างถูกต้องตามที่ออกแบบไว้ในแผนภาพสถานะ
3. ออกแบบโปรแกรมเพื่อรองรับการเพิ่มเติมคุณสมบัติในการอ่านข้อมูลจากไฟล์ XMI ที่ได้จากโปรแกรมอื่นๆ นอกเหนือจาก Rational Roseในอนาคต

1.4 ขั้นตอนในการพัฒนาโครงงาน

1. ศึกษาการออกแบบโดยใช้แผนภาพสถานะจำกัด
2. ศึกษาการเขียนแผนภาพสถานะ

3. ศึกษาการใช้งานโปรแกรม Rational Rose
4. ศึกษาการส่งออกข้อมูลแผนภาพสถานะโดยโปรแกรม Rational Rose
5. ศึกษารายละเอียดของมาตรฐาน XML
6. ศึกษารายละเอียดของมตรฐาน XMI
7. ศึกษารูปแบบไฟล์ XMI ที่ได้จากการส่งออกจากโปรแกรม Rational Rose
8. ศึกษาการพัฒนาโปรแกรมสร้างรหัสคำสั่ง
9. ศึกษาการพัฒนาโปรแกรมบนวินโดว์ด้วย Visual Studio.Net 2005
10. ศึกษาการพัฒนาแอพลิเคชันด้วยภาษา C#
11. ศึกษารูปแบบโครงสร้างของภาษา Java
12. วิเคราะห์ ออกแบบโปรแกรมสร้างรหัสคำสั่ง
13. ออกแบบโปรแกรมอย่างง่าย โดยใช้แผนภาพสถานะ
14. ทำการสร้างแผนภาพสถานะ และส่งออกโดยโปรแกรม Rational Rose
15. พัฒนาโปรแกรมสร้างรหัสคำสั่ง
16. ทดสอบการใช้งาน และปรับปรุงแก้ไข
17. จัดทำเอกสารประกอบโครงการ

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1. ลดข้อผิดพลาดที่เกิดจากการเขียนรหัสคำสั่งของโปรแกรมที่ออกแบบด้วยแผนภาพสถานะที่ซับซ้อน
2. ลดระยะเวลาที่ใช้ในการพัฒนาโปรแกรม
3. นำแผนภาพที่ใช้ในการออกแบบมาช่วยการพัฒนาโปรแกรมได้มากขึ้น

1.6 เครื่องมือที่ใช้ในการพัฒนาโครงงาน

- Rational Rose 8.0

  สำหรับออกแบบแผนภาพสถานะ และสร้างไฟล์ XMI สำหรับสร้างรหัสคำสั่ง
- Microsoft Visual Studio.Net 2005

  สำหรับพัฒนาโปรแกรมสร้างรหัสคำสั่ง

บทที่ 2
ทฤษฎีที่เกี่ยวข้อง

2.1 ยูเอ็มแอล

ยูเอ็มแอล หรือ Unified Modeling Language (UML) เป็นรูปแบบมาตรฐานที่ใช้หลักการออกแบบที่เป็นชิงวัตถุ ใช้ในการกำหนดมุมมอง รายละเอียด สร้างระบบงาน และสร้างเอกสารอ้างอิงสำหรับระบบงาน

ยูเอ็มแอล ถูกใช้อย่างแพร่หลายในการออกแบบโปรแกรมเชิงวัตถุ เนื่องจากมีคุณสมบัติที่ครบถ้วนและมี ข้อดีในการใช้งานหลายข้อ

ประโยชน์ของยูเอ็มแอลในการนำมาใช้ในการออกแบบคือ (ชาลี วรกุลพิพัฒน์. 2544)

1. ยูเอ็มแอล รวมแบบจำลองต่างๆไว้ เช่น แบบจำลองข้อมูล แบบจำลองธุรกิจ แบบจำลองวัตถุ แบบจำลองส่วนประกอบ แผนภาพสถานะ

2. เป็นมาตรฐานเปิดของทุกภาษาในปัจจุบัน ไม่ว่าจะเป็น Java, J2EE, VisualAge ล้วนแต่สนับสนุน UML

3. ยูเอ็มแอล ครอบคลุมทุกขั้นตอนในวงจรชีวิตของการพัฒนาระบบ ตั้งแต่ขั้นตอนของการหาความต้องการของระบบ การออกแบบระบบ การนำไปใช้งาน การติดตั้งระบบ ไปจนถึงขั้นตอนการจัดทำเอกสาร

4. เป็นภาษาที่มีความสมดุลในแง่ของความเรียบง่ายและความซับซ้อน คือ ง่ายต่อการเรียนรู้และการนำมาใช้งานจริง และสามารถนำไปใช้กับงานที่ซับซ้อนมากๆได้

5. มีบริษัทชั้นนำและอุตสาหกรรมต่างๆ ให้การยอมรับและสนับสนุน

2.2 เครื่องสถานะจำกัด

เครื่องสถานะจำกัด (Finite State Machine) หรือเรียกอีกชื่อว่า Finite State Automation (FSA) เป็นแบบจำลองอย่างง่ายที่ใช้ในการแสดงพฤติกรรมต่างๆของระบบ หรือวัตถุที่มีควาวมซับซ้อน โดยใช้เงื่อนไขหรือรูปแบบที่จำกัด โดยรูปแบบจะมีการเปลี่ยนแปลงเมื่อเกิดเหตุการณ์ใดๆขึ้น

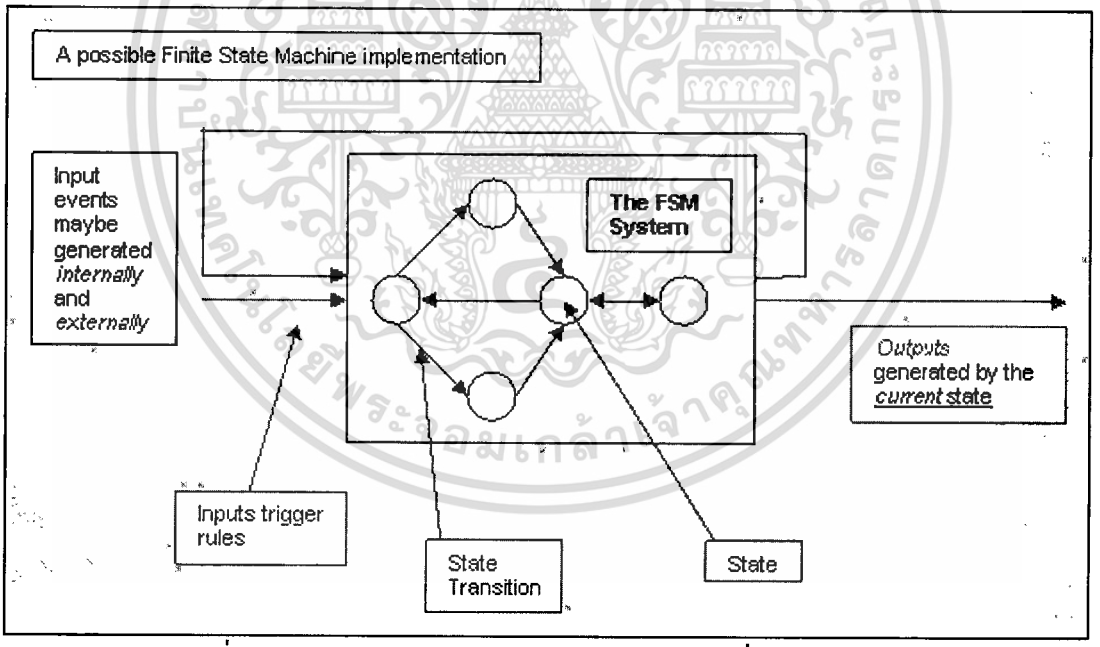
เครื่องสถานะจำกัดประกอบด้วยองค์ประกอบหลัก 4 ส่วนคือ (Brownlee. 2008)

- สถานะ (States) สำหรับแสดงพฤติกรรมซึ่งอาจจะมีการกระทำเกิดขึ้นในสถานะ

- การเปลี่ยนสภานะ (State Transitions) คือการเปลี่ยนแปลงจากสถานะหนึ่งไปยังสถานะหนึ่ง
- หลักเกณฑ์หรือเงื่อนไข (Rules or Conditions) ที่ทำให้เกิดการเปลี่ยนแปลงสถานะ
- เหตุการณ์นำเข้า (Input Events) คือเหตุการณ์ที่อาจเกิดขึ้นจากภายใน หรือภายนอก ซึ่งจะเป็นตัวกระตุ้นหลักเกณฑ์และนำไปสู่การเปลี่ยนแปลงสถานะ

เครื่องสถานะจำกัดจะต้องเริ่มด้วยสถานะเริ่มต้นสำหรับแสดงจุดเริ่มต้น และสถานะปัจจุบันสำหรับจดจำผลที่ได้จากเปลี่ยนสถานะล่าสุด หลังจากรับเหตุการณ์นำเข้าซึ่งทำหน้าที่กระตุ้นทำให้เกิดการประมวลผลของหลักเกณฑ์บางอย่างที่ทำหน้าที่ควบคุมการเปลี่ยนแปลงสถานะจากสถานะปัจจุบันไปยังสถานะอื่นๆ (Brownlee. 2008)

การนำเสนอเครื่องสถานะจำกัดที่ดีที่สุดนั้นสามารถใช้แผนผังการไหล (Flow Chart) ในการนำเสนอหรือสามารถใช้กราฟของสถานะโดยตรงได้ ซึ่งเป็นวิธีการที่สามารถแสดงแบบจำลองนามธรรมได้อย่างเที่ยงตรง (Brownlee. 2008)



รูปที่ 2.1 ตัวอย่างการเขียนแผนผังการไหลแสดงเครื่องสถานะจำกัด

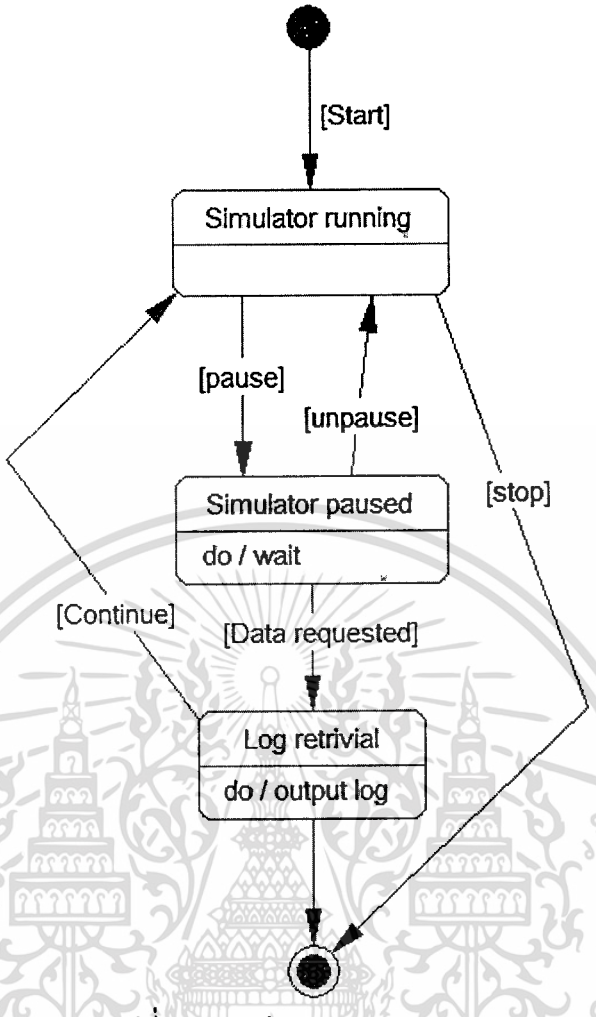
อย่างไรก็ดีเครื่องสถานะจำกัดนั้นยังสามารถแสดงโดยใช้แบบจำลองที่มีอยู่ในยูเอ็มแอลได้เช่นเดียวกัน โดยใช้แผนภาพสถานะซึ่งมีองค์ประกอบที่ใกล้เคียงกัน

2.3 แผนภาพสถานะ

แผนภาพสถานะ (State Diagram) เป็นแผนภาพที่แสดงสถานะของวัตถุ และเหตุการณ์หรือเงื่อนไขที่ทำให้เกิดการเปลี่ยนแปลงสถานะ รวมถึงการกระทำที่เกิดขึ้นเมื่อมีการเปลี่ยนแปลงสถานะ ช่วยในการอธิบายพฤติกรรมของระบบที่มีการเปลี่ยนแปลงไปตามเวลา แผนภาพสถานะประกอบด้วย สถานะ ซึ่งแสดงสถานะของวัตถุ และ การเปลี่ยนสถานะ ที่แสดงการเชื่อมโยงระหว่างแต่ละสถานะ โดยจะมีเงื่อนไขในการเปลี่ยนสถานะ และการกระทำ มีการกระทำเมื่อมีการเปลี่ยนแปลงสถานะ กำกับไว้ที่เส้นการเปลี่ยนสถานะ ซึ่งจะระบุทั้งเงื่อนไขและ การกระทำได้ แต่ต้องระบุเพียงตัวใดตัวหนึ่งก็ได้ แต่ต้องมีอย่างน้อยหนึ่งตัว (Martin and Kendall. 2000)

สัญลักษณ์	ชื่อ	คำอธิบาย
	สถานะ (State)	สถานะของวัตถุ
	การเปลี่ยนสถานะ (Transition)	เหตุการณ์หรือการกระทำที่ทำให้สถานะมีการเปลี่ยนแปลง
	สถานะเริ่มต้น (Initial State)	จุดเริ่มต้นของการทำงาน
	สถานะสิ้นสุด (End State)	จุดสิ้นสุดของการทำงาน

รูปที่ 2.2 สัญลักษณ์ที่ใช้ในแผนภาพสถานะ



รูปที่ 2.3 ตัวอย่างแผนภาพสถานะ

จากตัวอย่างแผนภาพสถานะในรูปที่ 2.3 ซึ่งเป็นการแสดงการเปลี่ยนสถานะของโปรแกรมจำลองโปรแกรมหนึ่ง เมื่อเริ่มทำงาน ระบบจะอยู่ในสถานะ Simulator running ซึ่งจะมีการเปลี่ยนสถานะเมื่อผู้ใช้ทำการ Stop หรือ ทำการ Pause โดยเมื่อมีการ Pause สถานะจะถูกเปลี่ยนไปสู่ Simulator paused และเมื่อมีการ Stop สถานะจะถูกเปลี่ยนไปสู่ Log retrieval และหากมีการ Stop จะจบการทำงาน

เมื่อระบบอยู่ในสถานะ Simulator paused หากมีการ Unpause สถานะจะกลับไปสู่ Simulator running และ ถ้ามี Data request สถานะจะเปลี่ยนไปเป็น Log retrieval

เมื่อระบบอยู่ในสถานะ Log retrieval หากมีการ Continue สถานะจะกลับไปสู่ Simulator running ถ้าไม่มีการทำงานใดๆ ระบบก็จะจบการทำงาน

2.4 XML Metadata Interchange

XML Metadata Interchange (XMI) คือมาตรฐานในการแลกเปลี่ยนโครงสร้างข้อมูลโดยใช้รูปแบบ XML เป็นมาตรฐาน

2.4.1 ความเป็นมา

XMI ถูกรับรองโดย OMG (Object Management Group) เผยแพร่ครั้งแรกเมื่อปี ค.ศ. 2001 โดมีจุดประสงค์ที่จะใช้เป็นสื่อกลางในการแลกเปลี่ยนข้อมูลของระบบต่างๆ เพื่อให้มีความถูกต้องและเป็นไปตามมาตรฐาน XMI นั้น สามารถใช้กับโครงสร้างใดๆที่จัดโครงสร้างให้อยู่ในรูปแบบของ Meta-Object Facility (MOF) ได้ ซึ่งโดยทั่วไปนั้น XMI จะถูกใช้ในการแลกเปลี่ยนข้อมูลแบบจำลองของ UML นั่นเอง จึงมีผู้พัฒนาซอฟต์แวร์หลายรายมีการใส่การทำงานในส่วนนี้ลงไปในตัวซอฟต์แวร์ เช่น Rational Rose เป็นต้น

แนวคิดของ OMG มีการมองข้อมูลของแบบจำลองออกเป็น 2 ส่วน คือ แบบจำลองรูปธรรม (Concrete Models) และแบบจำลองเต็มนามธรรม (Abstract Models) โดยแบบจำลองรูปธรรม จะแสดงถึงความสัมพันธ์ของข้อมูลต่างๆในแผนภาพ และแบบจำลองนามธรรมนั้นจะใช้บอกลักษณะการวางของแผนภาพ

อย่างไรก็ดี ถึงแม้ XMI จะออกมาเป็นมาตรฐาน แต่การนำไปใช้ของผู้พัฒนาแต่ละรายอาจมีรายละเอียดปลีกย่อยที่เป็นลักษณะจำเพาะของแต่ละราย ดังนั้น หากต้องมีการส่งข้อมูลระหว่างซอฟต์แวร์ที่พัฒนาโดยผู้พัฒนาคนละราย อาจต้องมีการแปลงรูปแบบของ XMI ให้อยู่ในรูปแบบมาตรฐานก่อน จึงจะสามารถส่งข้อมูลถึงกันได้อย่างถูกต้อง (Object Management Group. 2005)

2.4.2. กฎเกณฑ์ทั่วไปของ XMI

ในการสร้างไฟล์ XMI เพื่อให้เป็นไปตามมาตรฐานจะต้องเป็นไปตามกฎเกณฑ์หลัก 3 ข้อดังนี้

1. ต้องมีการประกาศข้อมูลพื้นฐานของ XML ข้อกำหนดนี้ถูกระบุไว้ใน เนมสเปซ http://schema.omg.org/spec/XMI/version โดย version ด้านท้ายใช้แทนรุ่นของ XMI ที่ออกเป็นมาตรฐาน

2. การแสดงแบบจำลองของคลาส คือ ให้ทุกๆแบบจำลองของคลาสมีอยู่ในแบบเค้าร่างโดยใช้เอลิเมนต์ name แทนชื่อของคลาสนั้นๆ เช่นเดียวกับ complexType ที่ใช้ name แทนชื่อของคลาส ส่วนการประกาศคุณสมบัติของคลาสนั้น โดยปกติแล้ว เนื้อหาของแบบจำลอง XML นั้น มีความตรงกันกับ แบบจำลองของคลาสอยู่แล้ว

3. กลไกลส่วนขยายแบบจำลอง (Model Extension Mechanism) เค้าร่าง XMI ทุกเค้าร่างจะประกอบด้วย กลไกสำหรับขยายแบบจำลองของคลาสอยู่แล้ว ซึ่งเอลิเมนต์ของส่วนขยายอาจจะมีหรือไม่มีก็ได้ โดยจะอยู่ในส่วนเนื้อหาของแบบจำลองของแต่ละคลาสนั่นเอง

2.4.3 แบบจำลอง XMI

2.4.3.1 เค้าร่างและโครงสร้างของเอกสาร XMI

เค้าร่างของ XMI ทุกเค้าร่างจะต้องประกอบด้วยการประกาศต่อไปนี้ (Object Management Group. 2005)

- รุ่นของ XML เช่น

```
<? XML version="1.0"?>
```

- ทางเลือกในการประกาศการเข้ารหัสของเอกสารซึ่งเป็นไปตามมาตรฐาน ISO-10646 (หรือเรียกว่ายูนิโค้ดส่วนขยาย) เช่น

```
<? XML version="1.0" ENCODING="UCS-2"?>
```

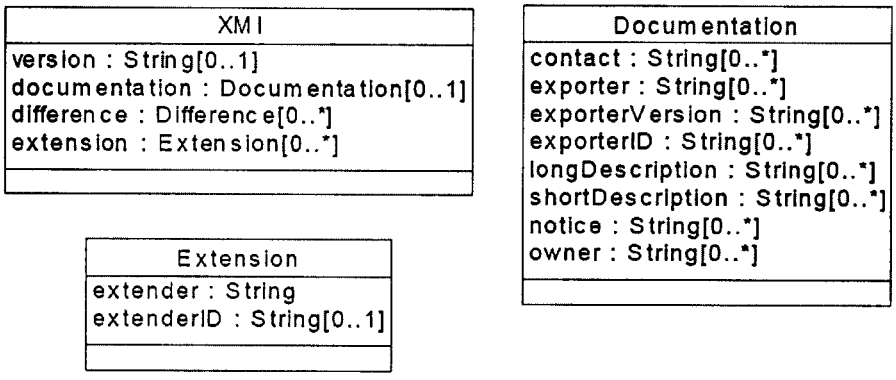
- ส่วนต่างๆที่ถูกต้องตามกระบวนการดำเนินการของ XML
- เค้าร่าง เอลิเมนต์ของ XML
- การนำเข้าของเอลิเมนต์ XML สำหรับ XMI namespace
- การประกาศแบบจำลองที่มีลักษณะเฉพาะ

เอกสาร XMI ทุกชิ้น จะต้องประกอบไปด้วยการประกาศดังต่อไปนี้ ยกเว้นกรณีที่มีการฝังข้อมูล XMI ลงไปในเอกสาร XML (Object Management Group. 2005)

- ขั้นตอนการประมวลผล XML
- ทางเลือกในการประกาศลักษณะการเข้ารหัสของชุดตัวอักษรที่ใช้
- กระบวนการอื่นๆที่เป็นไปตามลักษณะของ XML

2.4.3.2 คลาสแบบจำลอง XMI

คลาส XMI โดยทั่วไปจะเป็นส่วนสำหรับบรรจุเอกสารโครงสร้าง XMI และเนื้อหา โดยลักษณะของคลาส XMI ประกอบไปด้วย รุ่นของ XMI การจัดการเอกสาร การเปลี่ยนแปลง และส่วนขยาย ในคลาสการจัดการเอกสารนั้น จะประกอบไปด้วยหลายๆส่วนที่ใช้สำหรับอธิบายเอกสารที่ไม่ใช้ในการคำนวน คลาสส่วนขยายประกอบไปด้วยโครงสร้างสำหรับข้อมูลที่เป็นส่วนขยาย (Object Management Group. 2005)



รูปที่ 2.4 เอลิเมนต์ XMI เอกสารประกอบและ เอลิเมนต์ส่วนขยาย

2.4.3.3 XMI

ระดับบนสุดของเอกสาร XMI จะประกอบด้วยข้อมูลที่เป็นเอลิเมนต์ของ XMI เท่านั้น โดยประกาศในลักษณะดังนี้ (Object Management Group. 2005)

```
<xsd:complexType name="XMI">
 <xsd:choice minOccurs="0" maxOccurs="unbounded">
  <xsd:any processContents="strict"/>
 </xsd:choice>
 <xsd:attribute ref="id"/>
 <xsd:attributeGroup ref="IdentityAttribs"/>
 <xsd:attributeGroup ref="LinkAttribs"/>
 <xsd:attribute name="type" type="xsd:QName" use="optional"
form="qualified"/>
 <xsd:attribute name="version" type="xsd:string" use="optional"
form="qualified"/>
</xsd:complexType>

<xsd:element name="XMI" type="XMI"/>
```

ถ้ามีการกำหนดคุณลักษณะของรุ่นของ XMI แล้วนั้น จะหมายถึงเอกสารนี้จะอ้างอิงโครงสร้างตาม XMI รุ่นดังกล่าว โดยแต่ละรุ่นของ XMI จะมี เนมสเปซ ที่แตกต่างกันซึ่งถูกแสดงอยู่ในรูป http://schema.omg.org/spec/XMI/[version]

2.4.3.4 ส่วนขยาย

คลาสส่วนขยายนั้นออกแบบเพื่อใช้สำหรับรองรับข้อมูลส่วนขยายต่างๆที่อยู่นอกเหนือแบบจำลองของผู้ใช้ ในส่วนขยายนี้จะมีหลายคุณลักษณะของคลาส XMI และอาจจะฝังอยู่ในพื้นที่ที่กำหนดไว้ของ เอกสาร XMI เค้าร่างของส่วนขยายจะเป็นดังนี้ (Object Management Group. 2005)

```
<xsd:complexType name="Extension">
 <xsd:choice minOccurs="0" maxOccurs="unbounded">
  <xsd:any processContents="lax"/>
 </xsd:choice>
 <xsd:attribute ref="id"/>
```

```
 <xsd:attributeGroup ref="ObjectAttribs"/>
 <xsd:attribute name="extender" type="xsd:string" use="optional"/>
 <xsd:attribute name="extenderID" type="xsd:string" use="optional"/>
</xsd:complexType>

<xsd:element name="Extension" type="Extension"/>
```

แอตทริบิวต์ extender

แอตทริบิวต์ extender จะบอกว่า เครื่องมือตัวใดเป็นผู้สร้างส่วนขยายนี้

2.4.3.5 เอกสารประกอบ

คลาสเอกสารส่วนประกอบ จะประกอบไปด้วยข้อมูลที่เกี่ยวกับเอกสาร XMI หรือการถ่ายโอนเริ่มดำเนินการ ใช้สำหรับบันทึกข้อมูลของเจ้าของเอกสาร ผู้ติดต่อ คำอธิบายทั้งแบบสั้น และยาว เครื่องมือที่ทำการนำออกที่เป็นผู้สร้างเอกสาร รุ่นของเครื่องมือ ข้อความแสดงลิขสิทธิ์ หรือประกาศอื่นๆที่เกี่ยวกับเอกสารนั้นๆ โดยรูปแบบของข้อมูลในเอกสารประกอบทั้งหมดจะเป็นตัวอักษร มีเค้าร่าง XML ดังนี้ (Object Management Group. 2005)

```
<xsd:complexType name="Documentation">
 <xsd:choice minOccurs="0" maxOccurs="unbounded">
  <xsd:element name="contact" type="xsd:string"/>
  <xsd:element name="exporter" type="xsd:string"/>
  <xsd:element name="exporterVersion" type="xsd:string"/>
  <xsd:element name="longDescription" type="xsd:string"/>
  <xsd:element name="shortDescription" type="xsd:string"/>
  <xsd:element name="notice" type="xsd:string"/>
  <xsd:element name="owner" type="xsd:string"/>
  <xsd:element ref="Extension"/>
 </xsd:choice>
 <xsd:attribute ref="id"/>
 <xsd:attributeGroup ref="ObjectAttribs"/>
 <xsd:attribute name="contact" type="xsd:string" use="optional"/>
 <xsd:attribute name="exporter" type="xsd:string" use="optional"/>
 <xsd:attribute name="exporterVersion" type="xsd:string"
       use="optional"/>
 <xsd:attribute name="longDescription" type="xsd:string"
       use="optional"/>
 <xsd:attribute name="shortDescription" type="xsd:string"
       use="optional"/>
 <xsd:attribute name="notice" type="xsd:string" use="optional"/>
 <xsd:attribute name="owner" type="xsd:string" use="optional"/>
</xsd:complexType>

<xsd:element name="Documentation" type="Documentation"/>
```

รายละเอียดต่างๆ ของเอกสารจะถูกแสดงในเอลิเมนต์

2.4.3.6 เพิ่ม แทนที่ และลบ

คลาสเพิ่ม (Add) ใช้สำหรับการเพิ่มกลุ่มของวัตถุลงในเอกสารนี้ หรือเอกสารอื่นๆ แอตทริบิวต์ position จะเป็นตัวกำหนดว่า ตำแหน่งใดที่จะใช้ในการเพิ่มข้อมูลดังกล่าว โดยอ้างอิงจาก เอลิเมนต์ของ XML อื่นๆ แอตทริบิวต์ addition จะอ้างถึงกลุ่มของวัตถุที่ต้องการเพิ่ม โดยทั้งสองคุณลักษณะนี้ จะต้องถูกกำหนดให้เป็น “true”

คลาสแทนที่ (Replace) ใช้สำหรับการลบกลุ่มของวัตถุ ณ ตำแหน่งปลายทาง และเพิ่มวัตถุที่กำหนดลงไปในตำแหน่งดังกล่าว

คลาสลบ (Delete) ใช้สำหรับการลบกลุ่มของวัตถุที่กำหนดออกไปจากเอกสารนี้ หรือเอกสารอื่นๆ

คลาส Difference เป็นซูเปอร์คลาสของทั้งคลาสเพิ่ม แทนที่ และลบ โดยมีการประกาศคลาส เป็นดังนี้ (Object Management Group. 2005)

```
<xsd:complexType name="Difference">
 <xsd:choice minOccurs="0" maxOccurs="unbounded">
  <xsd:element name="target">
   <xsd:complexType>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
     <xsd:any processContents="skip"/>
    </xsd:choice>
    <xsd:anyAttribute processContents="skip"/>
   </xsd:complexType>
   </xsd:element>
  <xsd:element name="difference" type="Difference"/>
  <xsd:element name="container" type="Difference"/>
  <xsd:element ref="Extension"/>
 </xsd:choice>
 <xsd:attribute ref="id"/>
 <xsd:attributeGroup ref="ObjectAttribs"/>
 <xsd:attribute name="target" type="xsd:IDREFS" use="optional"/>
 <xsd:attribute name="container" type="xsd:IDREFS" use="optional"/>
</xsd:complexType>

<xsd:element name="Difference" type="Difference"/>

<xsd:complexType name="Add">
<xsd:complexContent>
 <xsd:extension base="Difference">
  <xsd:attribute name="position" type="xsd:string" use="optional"/>
  <xsd:attribute name="addition" type="xsd:IDREFS" use="optional"/>
 </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="Add" type="Add"/>

<xsd:complexType name="Replace">
 <xsd:complexContent>
  <xsd:extension base="Difference">
   <xsd:attribute name="position" type="xsd:string" use="optional"/>
   <xsd:attribute name="replacement" type="xsd:IDREFS"
       use="optional"/>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="Replace" type="Replace"/>

<xsd:complexType name="Delete">
 <xsd:complexContent>
  <xsd:extension base="Difference"/>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="Delete" type="Delete"/>
```

มีคลาส Difference เป็นซูเปอร์คลาส

มีคลาส Difference เป็นซูเปอร์คลาส

มีคลาส Difference เป็นซูเปอร์คลาส

2.5 Microsoft.NET

Microsoft.NET หรือที่เรียกว่า .NET เป็นแพลตฟอร์มที่ทางไมโครซอฟต์ออกแบบและพัฒนาเพื่อรองรับการทำงานบนอินเทอร์เน็ต ในรูปแบบของเว็บฟอร์ม รวมทั้งที่ไม่ใช่เว็บฟอร์ม ทำ

ได้ง่าย รวดเร็ว และมีประสิทธิภาพสูง ย้ายจากโมเดลที่มีไคลเอนท์เป็นจุดศูนย์กลางไปสู่โมเดลที่ใช้เน็ตเวิร์คเป็นศูนย์กลาง ซึ่งก็คืออินเทอร์เน็ตนั่นเอง (ไพศาล โมลิสกุลมงคล. 2545)

2.6 ความเป็นมาของ .NET

.NET เป็นพื้นฐานของการพัฒนา .NET Framework ซึ่งรวบรวมภาษาที่ใช้งานจริงและแพลตฟอร์มที่สามารถเอ็กซีคิวต์ได้ รวมทั้งคลาสไลบรารีพื้นฐานเป็นจำนวนมากและฟังก์ชันที่มีประสิทธิภาพมากมาย ที่แก่นของ .NET Framework ล้อมรอบด้วย XML และ SOAP เพื่อสนับสนุนการรวมตัวของแอพพลิเคชันบนอินเทอร์เน็ต

.NET Framwork เป็นโมเดลใหม่ที่สมบูรณ์แบบในการเขียนโปรแกรมและพัฒนาแอพพลิเคชันสำหรับอินเทอร์เน็ตบนวินโดว์ คำว่า .NET เป็นสิ่งที่มีความสำคัญในการทำตลาดของไมโครซอฟต์เพราะป้าย .NET นี้จะติดอยู่กับผลิตภัณฑ์ที่สนับสนุน .NET สำหรับ .NET Enterprise Server ไม่ว่าจะเป็น BizTalk Server 2000, Application Center2000, Commerce Server 2000, Exchange 2000 รวมถึง SQL Server 2000 (ไพศาล โมลิสกุลมงคล. 2545)

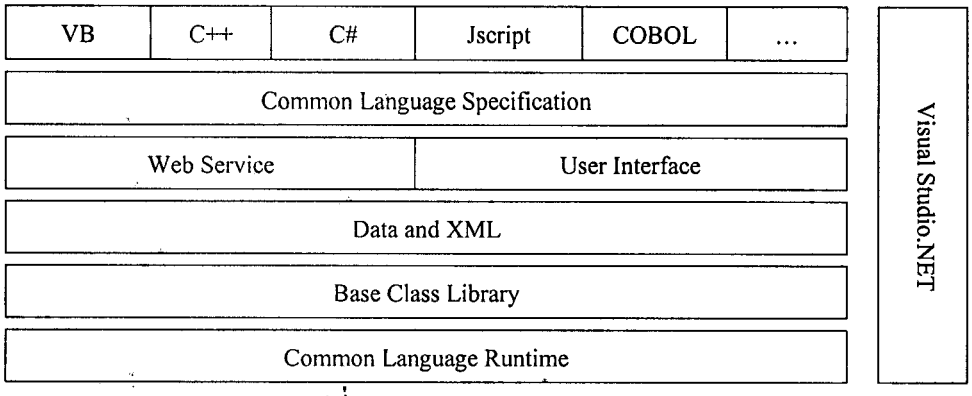
2.6.1 สถาปัตยกรรมของ .NET

สถาปัตยกรรมของ .NET มีส่วนต่างๆ ดังรูปที่ 2.5 ซึ่งประกอบไปด้วยองค์ประกอบดังต่อไปนี้ (ไพศาล โมลิสกุลมงคล. 2545)

- ชุดของเซอร์วิสพื้นฐานที่สามารถนำไปใช้งานในโปรแกรมภาษาต่างๆ
- เซอร์วิสเหล่านี้ถูกเอ็กซิคิวต์ในรูปแบบของ Intermediate Code ที่ไม่ขึ้นอยู่กับสถาปัตยกรรมใดๆ ทำให้สามารถรันบนสถาปัตยกรรมที่หลากหลาย
- เซอร์วิสจะทำงานในขณะที่รัน (Common Language Runtime) ซึ่งจะจัดการทรัพยากรและติดตามการเอ็กซิคิวต์ของแอพพลิเคชัน

เป้าหมายหลักของ .NET คือการสนับสนุนนักพัฒนาในการสร้างแอพพลิเคชันโดยใช้เว็บเซอร์วิสเพื่อใช้งานบนเทอร์มินัลต่างๆ ไม่ว่าจะเป็น เครื่องคอมพิวเตอร์พีซี พีดีเอ โทรศัพท์มือถือ หรือ อื่นๆ

- สถาปัตยกรรม .NET แสดงให้เห็นว่า .NET สนับสนุนโปรแกรมภาษาที่หลากหลาย โดยให้โปรแกรมเมอร์ที่ใช้โปรแกรมภาษาต่างๆ สามารถทำงานทันทีโดยไม่ต้องเสียเวลาศึกษาโครงสร้างภาษาใหม่ โดย .NET สนับสนุนการใช้โปรแกรมภาษาที่หลากหลายและมีคอมไพเลอร์สำหรับภาษาเหล่านั้น ภาษาต่างๆ ที่สามารถนำมาใช้งานนอกจากยกตัวอย่างไว้ในรูป ยังมีอีกมากมาย เช่น APL, Java, Delphi, Fortran, Component Pascal, Pascal, Perl, RPG และอีกมากมาย



รูปที่ 2.5 สถาปัตยกรรมของ .NET

- เนื่องจากภาษาคอมพิวเตอร์มีมากมาย ปกติแล้วภาษาใหม่ มักถูกสร้างขึ้นมาเพื่อตอบสนองความต้องการด้านต่างๆ เช่น แก้ปัญหาทางคณิตศาสตร์ คำนวณสำหรับงานวิจัย หรือความต้องการให้แอพพลิเคชันมีความน่าเชื่อถือ และการรักษาความปลอดภัย ด้วยเหตุนี้ทำให้ภาษาที่มีอยู่มีความแตกต่างกัน อาจจะเป็นในด้านขั้นตอน ด้านเชิงวัตถุ ด้านพารามิเตอร์ หรือการกำหนดตัวแปร แต่สำหรับแพลตฟอร์ม .NET สนับสนุนนั้นจะต้องมีชุดของโครงสร้างตามรายการที่กำหนดไว้ในคุณสมบัติพื้นฐานทางภาษา ที่ชื่อว่า Common Language Specification (CLS) เพื่อให้สามารถคอมไพล์เป็น MSIL เพื่อให้การทำงานเป็นไปตามข้อกำหนด

- ส่วนที่เป็นแอพพลิเคชันในแพลตฟอร์ม .NET จะเห็นได้ว่า มี 3 แบบ คือ Web Service, Web Form, Windows Form สำหรับ Web Service เราคุ้นเคยกันมาบ้างแล้ว ซึ่งก็คือการสร้างคอมโพเนนต์หรือโปรแกรมเพื่อให้บริการผ่านโพรโทคอล SOAP/HTTP นั่นเอง ส่วน Web Form และ Windows Form ถือว่าเป็นของใหม่ Web Form ก็คือการพัฒนาเว็บแบบใหม่ โดยเราสามารถสร้างส่วนต่อประสานกับผู้ใช้ ออกมาได้อย่างง่ายๆ เพียงแค่ลากเม้าส์ในลักษณะ drag and drop เหมือนกับการพัฒนาโปรแกรม Visual Basic ทั้งนี้เนื่องจากไมโครซอฟต์พยายามทำให้การพัฒนาเว็บง่ายขึ้น ส่วน Windows Form เป็นการสร้างโปรแกรมที่ทำงานในเครื่องพีซี คล้ายๆ กับฟอร์ม Visual Basic แบบเดิม หรือสรุปให้ชัดๆ ก็คือ Web Form เป็นการสร้างเว็บเพจ ส่วน Windows Form คือการสร้างโปรแกรมสำหรับทำงานในเครื่องพีซีนั่นเอง

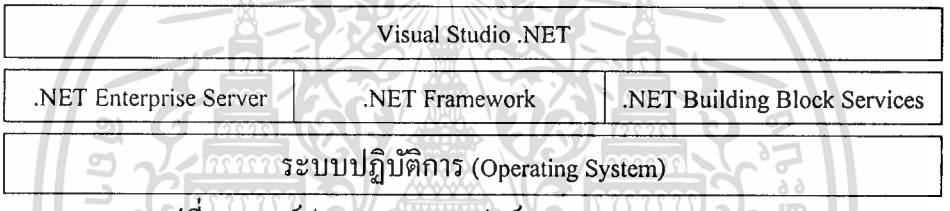
- โปรแกรมที่พัฒนาขึ้นด้วย .NET จะมีการเรียกใช้ข้อมูลที่เป็นประเภทเดียวกันทั้งหมด ไม่ว่าจะถูกพัฒนาขึ้นมาโดยใช้ภาษาใดๆ ก็ตาม ประเภทข้อมูลเหล่านั้นจะอยู่ในกลุ่มของคลาส Data และ XML เพื่อใช้ในกรเรียกใช้และจัดการฐานข้อมูลหรือข้อมูลในรูปแบบ XML ตัวอย่างเช่น คลาส SQL, ADO .NET, XML เป็นต้น

- ลำดับชั้นของชุดคลาสจะมีเซอร์วิสและ API ที่จำเป็นสำหรับการพัฒนาแอพพลิเคชันทำให้การพัฒนาเป็นไปในทิศทางเดียวกัน และใช้เวลาในการพัฒนาน้อยลงเป็นอย่างมาก

- เป็นแอพพลิเคชันที่ไม่ขึ้นกับฮาร์ดแวร์ โดยโค้ดที่ออกแบบทั้งหมดจะถูกคอมไพล์ผ่านทาง Intermediate Binary Code ไปเป็นภาษาที่เรียกว่า MSIL (Microsoft Intermediate Language) หลังจากนั้น MSIL จะถูกเอ็กซิคิวต์เป็นโค้ดเครื่อง (Machine Code) ใน Common Language Runtime (CLR) ซึ่งมีพื้นฐานการทำงานเหมือน JVM (Java Virtual Machine) บนแพลตฟอร์มจาวา โดย CLR คอมไพเลอร์ Just in Time (JIT) ซึ่งเป็นการคอมไพล์ที่ไม่ทำรวดเดียวตั้งแต่ต้นจนจบ แต่จะคอมไพล์เฉพาะส่วนที่จะใช้งานก่อนแล้วจึงรัน หลังจากนั้น ถ้าต้องการใช้งานส่วนใดก็จะคอมไพล์เพิ่มเติม โดยไม่คอมไพล์ซ้ำส่วนที่คอมไพล์ไปแล้วทำให้การทำงานเร็วขึ้นมาก

2.6.2 แพลตฟอร์ม .NET

องค์ประกอบของแพลตฟอร์ม .NET มีลักษณะดังรูปที่ 2.6 ซึ่งออกแบบเป็น 3 ชั้น ต่อไปนี้ (สราวุธ อ้อยศรีสกุล. 2544)



รูปที่ 2.6 องค์ประกอบแพลตฟอร์มของ Microsoft .NET

ชั้นล่างสุดคือระบบปฏิบัติการ (Operating System) ในเครื่องเซิร์ฟเวอร์ เครื่องเดสก์ทอป หรืออุปกรณ์ใดๆ ที่โปรแกรมทำงาน ถัดขึ้นมาในชั้นที่สอง แบ่งเป็นองค์ประกอบ 3 ส่วน คือ .NET Enterprise Server, .NET Framework และ .NET Building Block Services ส่วนชั้นสุดท้ายคือ Visual Studio.NET ซึ่งเป็นเครื่องมือที่ช่วยในการพัฒนาโปรแกรมหรือแอพพลิเคชันให้เป็นไปอย่างง่ายดาย

สำหรับ .NET Enterprise Server ที่อยู่ในชั้นที่สอง ก็คือ ผลิตภัณฑ์ต่างๆ เช่น ฐานข้อมูล SQL Server 2000, BizTalk Server 2000, Exchange 2000 เป็นต้น ผลิตภัณฑ์เหล่านี้ถูกออกแบบมาเพื่อรองรับการพัฒนาแอพพลิเคชันในระดับองค์กรไม่เฉพาะในยุค .NET เท่านั้น เพราะก่อนหน้านี้ก็มีผลิตภัณฑ์บางตัวออกมาให้ใช้ก่อนแล้วในแพลตฟอร์ม Window DNA ผลิตภัณฑ์ชุดนี้นับว่าเป็นชุดต่อจาก Microsoft Back Office โดยจุดหลักที่เปลี่ยนไปจากเดิม คือการรองรับข้อมูลในรูปแบบ XML นั่นเอง

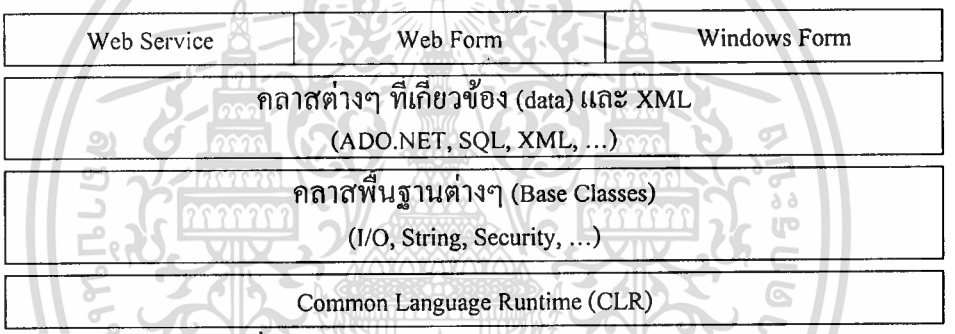
ส่วน .NET Building Block Services ก็คือบริการ Web Service ที่ไมโครซอฟต์สร้างขึ้นเพื่อให้นักพัฒนาใช้ ในช่วงระยะแรกๆ จะมีบริการ 2 อย่างที่มักได้ยิน ได้แก่ Microsoft Passport ก็คือการที่ผู้ใช้ กรอกข้อมูลบางอย่างเช่นชื่อผู้ใช้และรหัสผ่านเพียงครั้งเดียว ก็สามารถเข้านอกออกใน

เว็บไซต์ต่างๆ ที่ใช้ระบบ Microsoft Passport ได้ดี ในขณะที่ HailStorm เป็นอีกบริการหนึ่งที่ช่วยอำนวยความสะดวกในการท่องเว็บไซต์ให้กับผู้ใช้ในลักษณะที่เรียกว่า Personalization คือมีการเก็บข้อมูลต่างๆ ของผู้ใช้แต่ละคนไว้ รวมถึงสิทธิในการเข้าถึงข้อมูลต่างๆ ในอินเทอร์เน็ตด้วย

.NET Framework แบ่งออกเป็นระดับต่างๆ ตามลักษณะของ การพัฒนาซอฟต์แวร์ คือ ตั้งแต่ระดับระบบปฏิบัติการ, ระดับคอมโพเนนต์ที่ช่วยในการพัฒนาซอฟแวร์ (เรียกว่า คลาสพื้นฐานหรือ Base Classes) ไปจนถึงระดับการแสดงผล อย่าง Web Service, Web Form และ Windows Form นอกจากนี้ .NET Framework ยังต้องอาศัยเทคโนโลยีอย่างภาษา XML และโพรโทคอล SOAP ในการติดต่อ และเรียกใช้งานโปรแกรมอื่นๆ ผ่านระบบอินเทอร์เน็ตด้วย

2.6.3 .NET Framework

.Net Framework มีส่วนประกอบพื้นฐาน ดังรูปที่ 2.7 (สราวุธ อ้อยศรีสกุล. 2544)



รูปที่ 2.7 องค์ประกอบของ .NET Framework

ชั้นแรกคือส่วนที่เป็นแอพพลิเคชันในแพลตฟอร์ม .NET มี 3 แบบ คือ Web Service ซึ่งก็คือการสร้างคอมโพเนนต์หรือโปรแกรมเพื่อให้บริการผ่านโปรโตคอล SOAP/HTTP นั่นเอง ส่วน Web Form ก็คือ การพัฒนาเว็บเพจรูปแบบใหม่ ส่วน Windows Form เป็นการสร้างโปรแกรมที่ทำงานในเครื่องพีซี

ชั้นถัดมา คือ ชั้น Data และ XML เป็นกลุ่มคลาสที่ใช้ในการจัดการและเรียกใช้ข้อมูลต่างๆ เช่น จากฐานข้อมูล

ชั้น Base Class เป็นกลุ่มของคลาสที่ใช้งานทั่วๆ ไป

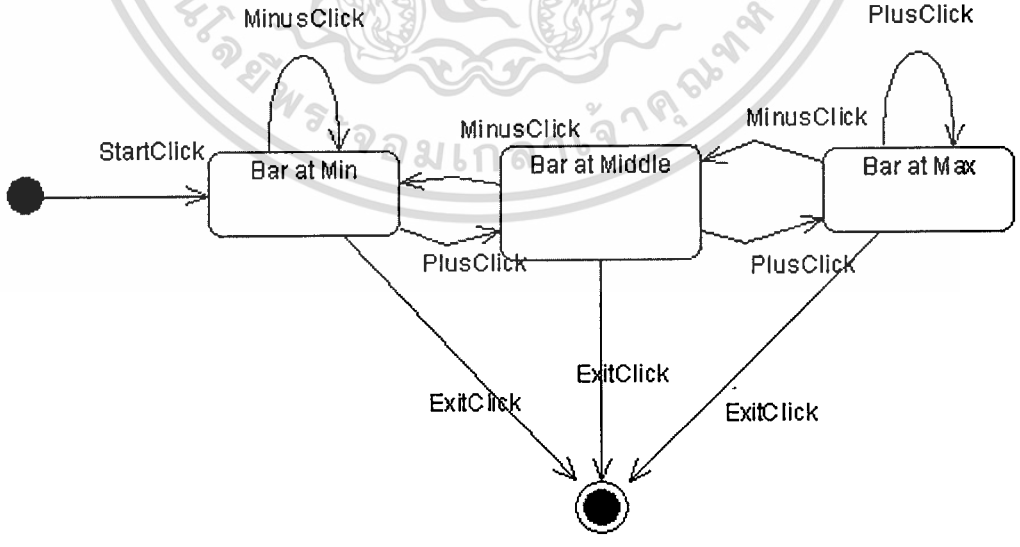
ชั้นล่างสุด คือ CLR เป็นสภาวะแวดล้อมแบบรันไทม์ในการจัดการโค้ดที่คอมไพล์แล้ว

บทที่ 3
การวิเคราะห์และออกแบบระบบ

โปรแกรมตัวสร้างรหัสคำสั่งจากแผนภาพสถานะ เป็นโปรแกรมที่แปลงข้อมูลของแผนภาพสถานะ ที่อยู่ในรูปแบบ XMI ให้เป็นรหัสคำสั่งภาษา Java โดยโปรแกรมจะสร้างรหัสคำสั่งต้นแบบที่มีเงื่อนไขต่างๆตามแผนภาพสถานะที่ออกแบบไว้ สำหรับให้นักพัฒนานำไปเป็นแนวทางในการพัฒนาเพิ่มเติมให้โปรแกรมทำงานได้สมบูรณ์ เพื่อช่วยให้ผู้พัฒนา สามารถพัฒนาโปรแกรมโดยช่วยการสร้างเงื่อนไขต่าง ลดการพิมพ์รหัสคำสั่งที่ซ้ำซ้อน และลดความผิดพลาดที่อาจจะเกิดขึ้น

ขั้นตอนการทำงานของโปรแกรม คือ หลังจากที่รับแฟ้มข้อมูล XMI โดยผู้ใช้งานระบุตำแหน่งที่อยู่ของแฟ้มแล้ว โปรแกรมจะทำการอ่านแฟ้มข้อมูล XMI และจัดเก็บอยู่ในรูปของวัตถุ หลังจากทำการตรวจสอบความถูกต้องของข้อมูลแล้วโปรแกรมจะทำการวิเคราะห์ข้อมูลโครงสร้างของ XMI ทำการตรวจสอบข้อมูล เพื่ออ่านกลุ่มข้อมูลเอลิเมนต์สถานะ และเอลิเมนต์การเปลี่ยนสถานะ จัดเก็บลงหน่วยความจำ และทำการสร้างรหัสคำสั่งโดยผู้ใช้สามารถเขียนรหัสคำสั่งเบื้องต้นก่อนทำการสร้างได้

ผู้เขียนได้ทำการออกแบบโปรแกรมตัวอย่างสำหรับใช้ในการทดสอบ โดยโปรแกรมตัวอย่างที่ออกแบบจำลองการทำงานของแถบสถานะที่แสดงสถานะ 3 สถานะคือ สูงสุด กึ่งกลาง และน้อยสุด โดยมีปุ่มให้เพิ่มค่าและลดค่า ซึ่งเขียนแผนภาพสถานะได้ดังนี้



รูปที่ 3.1 ตัวอย่างแผนภาพสถานะ

การทำงานของแผนภาพสถานะในรูปที่ 3.1 อธิบายได้ดังนี้

- เมื่อเริ่มทำงานสถานะของบาร์จะอยู่ในสถานะ Bar at Min

- เมื่ออยู่ในสถานะ Bar at Min หากมีการกดลบ (MinusClick) สถานะจะไม่เปลี่ยนแปลง หากมีการกดบวก (PlusClick) สถานะจะเปลี่ยนเป็น Bar at Middle และหากมีการกดออก (ExitClick) โปรแกรมจะจบการทำงาน

- เมื่ออยู่ในสถานะ Bar at Mid หากมีการกดลบ (MinusClick) สถานะจะเปลี่ยนเป็น Bar at Min หากมีการกดบวก (PlusClick) สถานะจะเปลี่ยนเป็น Bar at Max และหากมีการกดออก (ExitClick) โปรแกรมจะจบการทำงาน

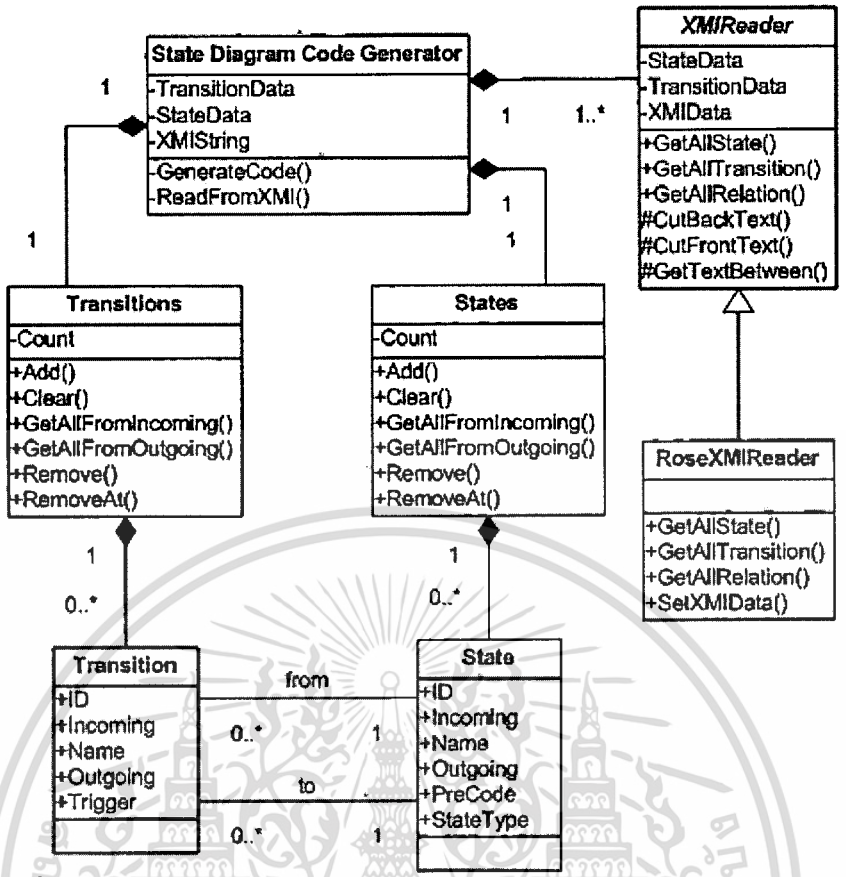
- เมื่ออยู่ในสถานะ Bar at Min หากมีการกดลบ (MinusClick) สถานะจะเปลี่ยนเป็น Bar at Middle หากมีการกดบวก (PlusClick) สถานะจะไม่เปลี่ยนแปลง และหากมีการกดออก (ExitClick) โปรแกรมจะจบการทำงาน

3.1 คลาสไดอะแกรม

คลาสไดอะแกรมของโปรแกรมสร้างรหัสคำสั่งประกอบด้วย 7 คลาสหลัก คือ

1. State Diagram Code Generator คลาสการทำงานหลักของโปรแกรม
2. XMIReader คลาสต้นแบบในการสร้างตัวอ่านข้อมูลจากไฟล์ XMI สำหรับรองรับการอ่านไฟล์ XMI จากโปรแกรมอื่นๆในอนาคต
3. RoseXMIReader คลาสที่พัฒนาสืบทอดจากคลาส XMIReader สำหรับอ่านข้อมูลจากไฟล์ XMI ที่สร้างโดยโปรแกรม Rational Rose
4. State คลาสของสถานะ 1 สถานะ
5. States คลาสรวมสำหรับเป็นคลาสบรรจุของคลาส State
6. Transition คลาสการเปลี่ยนสถานะ 1 เหตุการณ์
7. Transitions คลาสรวมสำหรับเป็นคลาสบรรจุของคลาส Transition

แสดงคลาสไดอะแกรมได้ดังนี้

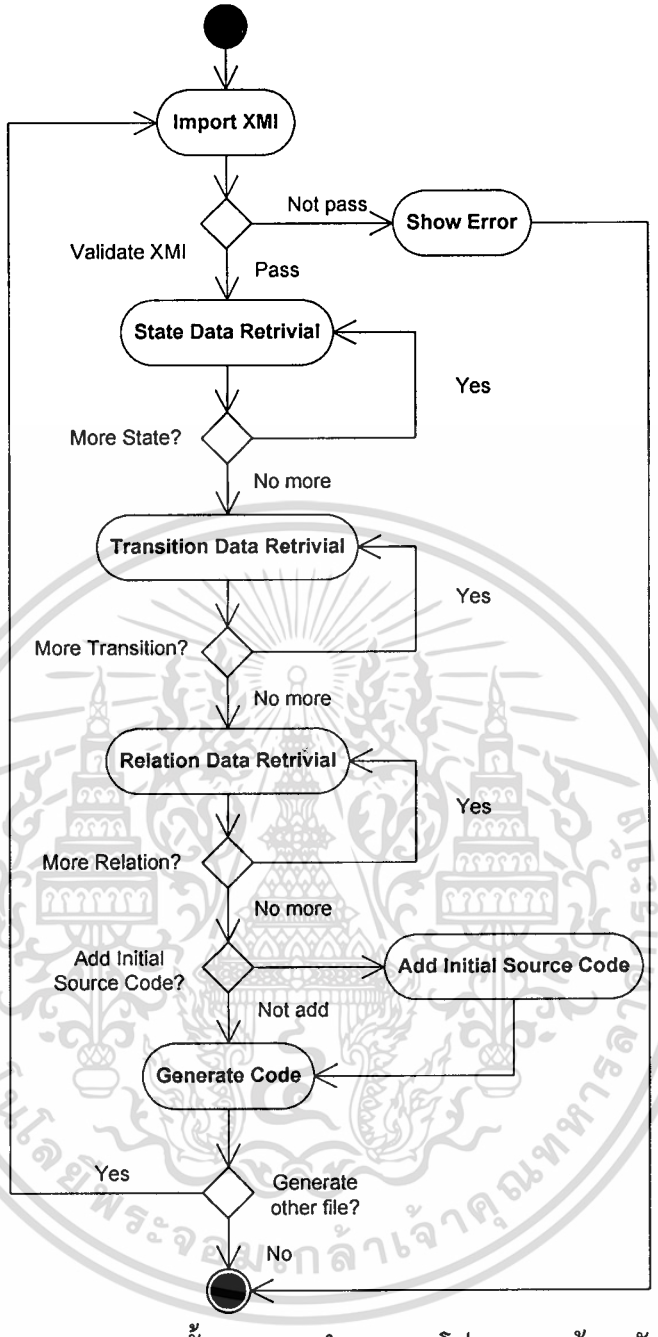


รูปที่ 3.2 คลาสไดอะแกรมของโปรแกรมสร้างรหัสคำสั่งจากแผนภาพสถานะ

หลังจากที่โปรแกรมทำการอ่านข้อมูลจากแฟ้ม XMI แล้วจะทำการแยกเอลิเมนต์ต่างๆ และจัดเก็บอยู่ในคลาส State, Transition เพื่อใช้ในการสร้างรหัสคำสั่ง

3.2 แอกทิวิตีไดอะแกรม

การทำงานของโปรแกรม จะเริ่มจากการนำเข้าข้อมูลแฟ้ม XMI หลังจากโหลดรายละเอียดแฟ้ม XMI เข้าสู่หน่วยความจำ โปรแกรมจะทำการสร้างวัตถุ State, Transition ขึ้นตามลำดับ จากนั้นจะทำการเชื่อมโยงแต่ละวัตถุตามแผนภาพ เมื่อทำการสร้างวัตถุจนครบทุกสถานะ ผู้ใช้สามารถแก้ไขรหัสคำสั่งเบื้องต้นได้ แล้วจึงให้โปรแกรมทำการสร้างรหัสคำสั่งต้นแบบต่อไป โดยแสดงแอกทิวิตีไดอะแกรมของโปรแกรมได้ดังนี้



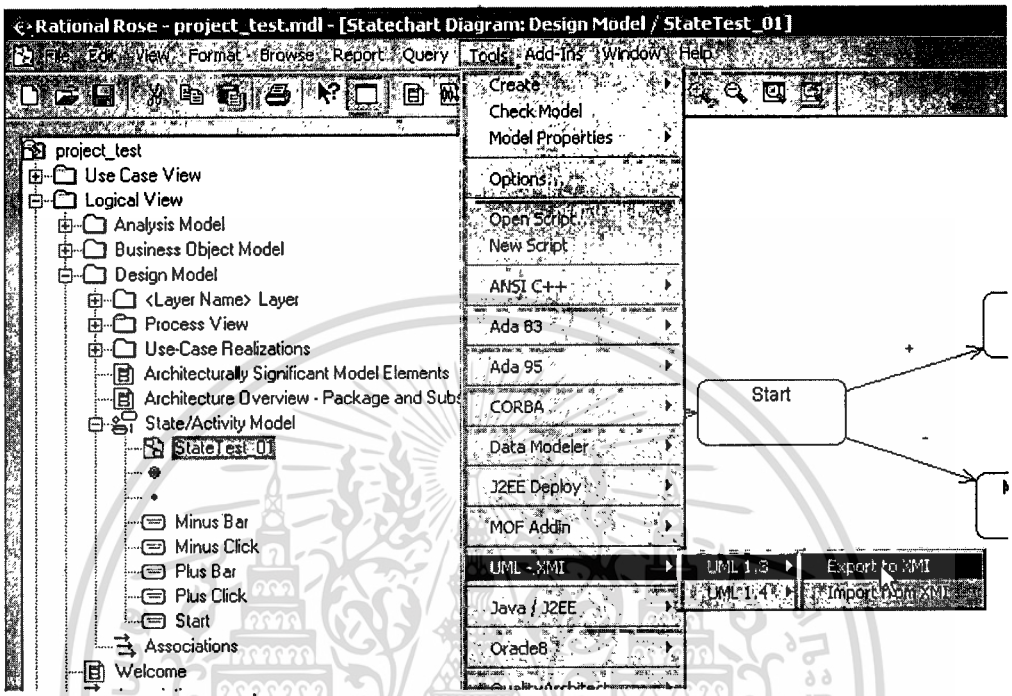
รูปที่ 3.3 แอกทิวิตีไดอะแกรมแสดงขั้นตอนการทำงานของโปรแกรมสร้างรหัสคำสั่งจาก แผนภาพสถานะ

3.3 วิเคราะห์รูปแบบไฟล์ XMI ที่สร้างโดยโปรแกรม Rational Rose

ตามขอบเขตของโครงงานนี้ เลือก Rational Rose เป็นกรณีศึกษา เนื่องจากเป็นเครื่องมือที่มีประสิทธิภาพ และนิยมใช้กันอย่างแพร่หลาย

3.3.1 การสร้างไฟล์ XMI จากโปรแกรม Rational Rose

การสร้างไฟล์ XMI จากโปรแกรม Rational Rose สามารถสร้างได้จากเมนู Tools / UML - XMI / UML 1.3 เลือก Export to XMI ดังแสดงในรูปที่ 3.3



รูปที่ 3.4 การสร้าง XMI จากโปรแกรม Rational Rose

3.3.2 รูปแบบไฟล์ XMI ที่ได้จากโปรแกรม Rational Rose

ไฟล์ XMI ที่สร้างโดยโปรแกรม Rational Rose นั้นจะประกอบด้วยข้อมูลทั้งหมดของงานที่ทำ เช่น กลุ่มของแบบจำลอง รายละเอียดแบบจำลองต่างๆ และ การแสดงผล เป็นต้น โดยส่วนของแผนภาพสถานะจะแสดงอยู่ในเอลิเมนต์ StateMachine แบ่งออกเป็นส่วนแบบจำลองรูปธรรมแสดงอยู่ในเอลิเมนต์ StateMachine.top และแบบจำลองนามธรรมแสดงในเอลิเมนต์ StateMachine.transitions โดยมีโครงสร้างดังนี้

```
<UML:StateMachine>
   <UML:StateMachine.top>
      <UML:CompositeState>
         <UML:CompositeState.subvertex>
            ... ข้อมูลสถานะในแผนภาพ
         </UML:CompositeState.subvertex>
      </UML:CompositeState>
   </UML:StateMachine.top>
   <UML:StateMachine.transitions>
      ... ข้อมูลการเปลี่ยนสถานะในแผนภาพ
   </UML:StateMachine.transitions>
</UML:StateMachine>
<UML:SignalEvent/>
<UML:Signal/>
<UML:Comment/>
```

ข้อมูลวัตถุในแผนภาพสถานะ ประกอบด้วย

- สถานะเริ่มต้น แทนด้วยเอลิเมนต์ Pseudostate
- สถานะสิ้นสุด แทนด้วยเอลิเมนต์ FinalState
- สถานะทั่วไป แทนด้วยเอลิเมนต์ SimpleState

ข้อมูลการเปลี่ยนสถานะทั้งหมดจะถูกแสดงในเอลิเมนต์ Transition

ข้อมูลเหตุการณ์ทั้งหมดจะถูกแสดงในเอลิเมนต์ Signal และ SignalEvent

ข้อมูลคำอธิบายจะถูกแสดงในเอลิเมนต์ Comment

เอลิเมนต์ที่แสดงสถานะเริ่มต้นจะแสดงในเอลิเมนต์ UML:Pseudostate โดยมีแอตทริบิวต์ที่สนใจคือ xmi.id แสดงรหัสอ้างอิง และ outgoing แสดง ID ของการเปลี่ยนสถานะทั้งหมด ที่เชื่อมโยงออกจากสถานะเริ่มต้น แสดงให้เห็นในตัวอย่างดังนี้

```
<UML:Pseudostate xmi.id = 'G.2'
  name = '' visibility = 'public' isSpecification = 'false'
  kind = 'initial'
  outgoing = 'G.3' />
```

เอลิเมนต์ที่แสดงสถานะสิ้นสุดจะแสดงในเอลิเมนต์ UML:FinalState โดยมีแอตทริบิวต์ที่สนใจคือ xmi.id แสดงรหัสอ้างอิง และ incoming แสดง ID ของการเปลี่ยนสถานะทั้งหมดที่เชื่อมโยงมาสู่สถานะสุดท้าย แสดงให้เห็นในตัวอย่างดังนี้

```
<UML:FinalState xmi.id = 'G.1'
  name = '' visibility = 'public' isSpecification = 'false'
  incoming = 'G.10 G.12 G.16 G.21' />
```

เอลิเมนต์ที่แสดงสถานะทั่วไปจะแสดงในเอลิเมน์ UML:SimpleState ประกอบด้วยแอตทริบิวต์ที่สนใจต่างๆดังนี้ xmi.id คือรหัสอ้างอิง name เป็นชื่อของสถานะ incoming แสดง ID ของการเปลี่ยนสถานะต้นทางที่เชื่อมโยงมาถึงสถานะนี้ และ outgoing แสดง ID ของการเปลี่ยนสถานะทั้งหมดที่เชื่อมโยงต่อออกจากสถานะนี้

```
<UML:SimpleState xmi.id = 'G.4'
  name = 'Start' visibility = 'public' isSpecification = 'false'
  outgoing = 'G.5 G.7' incoming = 'G.3' />
```

เอลิเมนต์ที่แสดงการเปลี่ยนสถานะจะแสดงในเอลิเมนต์ UML:Transition ประกอบด้วยแอตทริบิวต์ที่สนใจต่างๆดังนี้ xmi.id คือรหัสอ้างอิง source แสดง ID ของสถานะต้นทางที่เกิดการเปลี่ยนสถานะนี้ target แสดง ID สถานะปลายทางของการเปลี่ยนสถานะนี้ และ trigger แสดงรหัสอ้างอิงของเอลิเมนต์ SignalEvent ซึ่งเก็บชื่อเหตุการณ์ของการเปลี่ยนสถานะไว้ แสดงให้เห็นในตัวอย่างดังนี้

```
<UML:Transition xmi.id = 'G.5'
  name = '' visibility = 'public' isSpecification = 'false'
  trigger = 'S.258.0329.14.35' source = 'G.4' target = 'G.13' />
```

เอลิเมนต์ที่แสดงชื่อของเหตุการณ์ของการเปลี่ยนสถานะ จะถูกแสดงในเอลิเมนต์ UML:SignalEvent ประกอบด้วยแอตทริบิวต์ที่สนใจคือ xmi.id แสดงรหัสอ้างอิง และ name แสดงชื่อของเหตุการณ์ ดังตัวอย่าง

```
<UML:SignalEvent xmi.id = 'S.278.1134.02.5'
  name = 'StartClick' visibility = 'public' isSpecification = 'false'
  signal = 'S.278.1134.02.6' />
```

```
<!-- ==================== project_test::Design Model::State/Activity Model    [StateMachine] ====================
<UML:StateMachine xmi.id = 'S.278.1134.01.4'
  name = 'State/Activity Model' visibility = 'public' isSpecification = 'false'
  context = 'S.278.1134.01.3' >
  <UML:StateMachine.top>
    <!-- ==================== project_test::Design Model::State/Activity Model::{top}    [CompositeState] ======
    <UML:CompositeState xmi.id = 'XX.5.1134.2.16'
      name = '{top}' visibility = 'public' isSpecification = 'false'
      isConcurrent = 'false' >
      <UML:CompositeState.subvertex>
        <!-- ==================== project_test::Design Model::State/Activity Model::{top}::    [FinalState] ====
        <UML:FinalState xmi.id = 'G.1'
          name = '' visibility = 'public' isSpecification = 'false'
          incoming = 'G.8 G.17 G.22' />
        <!-- ==================== project_test::Design Model::State/Activity Model::{top}::    [Pseudostate] ===
        <UML:Pseudostate xmi.id = 'G.2'
          name = '' visibility = 'public' isSpecification = 'false'
          kind = 'initial'
          outgoing = 'G.3' />
        <!-- ==================== project_test::Design Model::State/Activity Model::{top}::Bar at Min    [Simple
        <UML:SimpleState xmi.id = 'G.5'
          name = 'Bar at Min' visibility = 'public' isSpecification = 'false'
          outgoing = 'G.6 G.8' incoming = 'G.3 G.11' />
        <!-- ==================== project_test::Design Model::State/Activity Model::{top}::Bar at Middle    [Si
        <UML:SimpleState xmi.id = 'G.10'
          name = 'Bar at Middle' visibility = 'public' isSpecification = 'false'
          outgoing = 'G.11 G.13 G.15 G.16 G.17' incoming = 'G.6 G.15 G.16 G.20' />
        <!-- ==================== project_test::Design Model::State/Activity Model::{top}::Bar at Max    [Simple
        <UML:SimpleState xmi.id = 'G.19'
          name = 'Bar at Max' visibility = 'public' isSpecification = 'false'
          outgoing = 'G.20 G.22' incoming = 'G.13' />
      </UML:CompositeState.subvertex>
    </UML:CompositeState>
  </UML:StateMachine.top>
  <UML:StateMachine.transitions>
    <UML:Transition xmi.id = 'G.3'
      name = '' visibility = 'public' isSpecification = 'false'
```

รูปที่ 3.5 ตัวอย่างไฟล์ XMI ที่ได้จากโปรแกรม Rational Rose

3.4 คลาสข้อมูลสถานะ และการเปลี่ยนสถานะ

ข้อมูลที่อ่านได้จากไฟล์ XMI จะถูกบันทึกลงโปรแกรมในรูปของคลาสสถานะ และคลาสการเปลี่ยนสถานะ โดยมีคลาสสำหรับใช้ในการบันทึกข้อมูลทั้งหมด 4 คลาสดังนี้

3.4.1 คลาส State

เป็นคลาสสำหรับเก็บข้อมูลสถานะ 1 สถานะ

มีแอตทริบิวต์ดังนี้

- ID รหัสอ้างอิงของสถานะ

- Name ชื่อของสถานะ
- Incoming การเปลี่ยนแปลงสถานะที่นำมาสู่สถานะนี้
- Outgoing การเปลี่ยนสถานะที่สามารถเกิดขึ้นได้จากสถานะนี้
- StateType ประเภทของสถานะ เริ่มต้น สิ้นสุด หรือสถานะทั่วไป
- PreCode เก็บรหัสคำสั่งเบื้องต้นของสถานะ

3.4.2 คลาส States

เป็นคลาสบรรจุสำหรับเก็บคลาสสถานะรวมกัน

- Count แสดงจำนวนของสถานะทั้งหมดในคลาส

มีฟังก์ชันการทำงานดังนี้

- Add() สำหรับเพิ่มคลาสสถานะลงในคลาสบรรจุ
- Clear() ทำการลบคลาสสถานะที่บรรจุอยู่ทั้งหมด
- GetAllFromIncoming() ค้นหาคลาสสถานะทั้งหมดที่มีคลาสการเปลี่ยนสถานะ Incoming เป็น ID ที่ระบุ โดยจะได้เป็นคลาสบรรจุใหม่
- GetAllFromOutgoing() ค้นหาคลาสสถานะทั้งหมดที่มีคลาสการเปลี่ยนสถานะใน Outgoing เป็น ID ที่ระบุ โดยจะได้เป็นคลาสบรรจุใหม่
- Remove() ลบคลาสสถานะที่ระบุออกจากคลาสบรรจุ
- RemoveAt() ลบคลาสสถานะในตำแหน่งที่ระบุออกจากคลาสบรรจุ

3.4.3 คลาส Transition

เป็นคลาสสำหรับเก็บข้อมูลการเปลี่ยนสถานะ 1 เหตุการณ์

มีแอตทริบิวต์ดังนี้

- ID รหัสอ้างอิงของสถานะ
- Name ชื่อของสถานะ
- Incoming สถานะต้นทางที่ก่อนเกิดการเปลี่ยนสถานะนี้
- Outgoing สถานะปลายทางเมื่อนเกิดการเปลี่ยนสถานะนี้
- Trigger เก็บรหัสของ Trigger เพื่อใช้ในการเชื่อมโยงกับเอลิเมนต์เหตุการณ์

3.4.4 คลาส Transitions

เป็นคลาสบรรจุสำหรับสำหรับเก็บคลาสการเปลี่ยนสถานะรวมกัน

มีแอตทริบิวต์ดังนี้

- Count แสดงจำนวนของการเปลี่ยนสถานะทั้งหมดในคลาส

มีฟังก์ชันการทำงานดังนี้

- Add() สำหรับเพิ่มคลาสการเปลี่ยนสถานะลงในคลาสบรรจุ
- Clear() ทำการลบคลาสการเปลี่ยนสถานะที่บรรจุอยู่ทั้งหมด
- GetAllFromIncoming() ค้นหาคลาสการเปลี่ยนสถานะทั้งหมดที่มีคลาสสถานะใน Incoming เป็น ID ที่ระบุ โดยจะได้เป็นคลาสบรรจุใหม่
- GetAllFromOutgoing() ค้นหาคลาสการเปลี่ยนสถานะทั้งหมดที่มีคลาสสถานะใน Outgoing เป็น ID ที่ระบุ โดยจะได้เป็นคลาสบรรจุใหม่
- Remove() ลบคลาสการเปลี่ยนสถานะที่ระบุออกจากคลาสบรรจุ
- RemoveAt() ลบคลาสการเปลี่ยนสถานะในตำแหน่งที่ระบุออกจากคลาสบรรจุ

3.5 การสร้างรหัสคำสั่ง

ขั้นตอนการสร้างรหัสคำสั่ง เริ่มจากการนำเข้าข้อมูลจากไฟล์ XMI สู่โปรแกรม รายละเอียดในไฟล์ XMI จะถูกจัดเก็บอยู่ในตัวแปร XMIString ในรูปแบบของข้อความ จากนั้น ขั้นตอนการสร้างรหัสคำสั่งจะแบ่งออกเป็น 3 ขั้นตอนคือ

3.5.1 แปลงข้อความจากไฟล์ XMI ให้อยู่ในรูปของคลาส States และ Transitions และจัดเก็บลงสู่ตัวแปร StateData และ TransitionData ของโปรแกรม

3.5.2 การเพิ่มรหัสคำสั่งเบื้องต้นเพื่อช่วยในการสร้างรหัสคำสั่งให้มีความสมบูรณ์ยิ่งขึ้น

3.5.3 ทำการสร้างรหัสคำสั่งจากข้อมูลที่เก็บอยู่ในคลาส States และ Transitions

3.5.1 แปลงข้อความจากไฟล์ XMI ให้อยู่ในรูปคลาส

3.5.1.1 คลาสต้นแบบสำหรับอ่านข้อมูลจากไฟล์ XMI

เพื่อให้การพัฒนาการอ่านไฟล์ XMI จากโปรแกรมอื่นในอนาคตเป็นไปในทิศทางเดียวกัน จึงได้สร้างคลาสต้นแบบสำหรับการอ่านข้อมูลจาก XMI ขึ้น คือคลาส XMIReader

คลาสต้นแบบ XMIReader นี้หน้าที่หลักคือใช้ในการแปลงข้อความที่อ่านได้จากไฟล์ XMI ให้เป็นข้อมูลในรูปแบบของคลาส State และ Transition ประกอบด้วยฟังก์ชันต้นแบบที่ต้องทำการพัฒนาอยู่ 3 ฟังก์ชันคือ

1. GetAllState() สำหรับทำการอ่านข้อมูลสถานะทั้งหมดและสร้างคลาส States บันทึกลงตัวแปร StateData ในโปรแกรม

2. GetAllTransition() สำหรับทำการอ่านข้อมูลการเปลี่ยนสถานะทั้งหมด และสร้างคลาส Transitions บันทึกลงตัวแปร TransitionData ในโปรแกรม

3. GetAllRelation() สำหรับทำการอ่านข้อมูลการเชื่อมโยง และทำการเชื่อมโยงสถานะกับการเปลี่ยนสถานะเข้าด้วยกัน และปรับปรุงข้อมูลปลายทางของคลาส State ใน StateData

เพื่อความสะดวกในการพัฒนาเพิ่มเติม คลาส XMIReader จะมีฟังก์ชันสำหรับใช้งานในการประมวลผลข้อความอยู่ 3 ฟังก์ชันคือ

1. CutFrontString(AllText, cutPoint*[, startPos]*) return String

สำหรับค้นหาตำแหน่งคำที่ระบุและตัดข้อมูลส่วนต้นจนถึงตำแหน่งอ้างอิงออก เพื่อให้มีการเก็บข้อมูลในหน่วยความจำน้อยลงระหว่างทำการประมวลผลข้อความ มีพารามิเตอร์ต่างๆดังนี้

- AllText คือข้อมูลข้อความทั้งหมด
- cutPoint เป็นข้อความที่ต้องการระบุตำแหน่งอ้างอิงสำหรับตัดข้อมูล
- startPos ตำแหน่งที่เริ่มค้นหาตำแหน่งอ้างอิง กรณีที่ไม่ต้องการค้นหาตั้งแต่เริ่มต้นหากไม่ระบุจะทำการค้นหาตั้งแต่เริ่มต้นข้อความ

ข้อมูลที่ได้จากฟังก์ชันคือข้อความที่ถูกตัดแล้ว

ตัวอย่างการใช้งาน

```
XMIData = CutFrontText(XMIData, "<UML:StateMachine.top>");
```

2. CutBackString(AllText, cutPoint*[, startPos]*) return String

ลักษณะเหมือนกับฟังก์ชัน CutFrontString() คือค้นหาตำแหน่งคำที่ระบุ แต่จะเป็นการตัดข้อมูลตั้งแต่ตำแหน่งอ้างอิงจนถึงตำแหน่งสุดท้ายทิ้งไป

ตัวอย่างการใช้งาน

```
XMIData = CutBackText(XMIData, "</UML:StateMachine.top>");
```

3. GetTextBtween(AllText, textStart, textEnd*[, startIdxText][, startPos]*) return String

สำหรับอ่านค่าของข้อความที่อยู่ระหว่างข้อความที่กำหนดมีพารามิเตอร์ต่างๆดังนี้

- AllText ข้อมูลข้อควาทั้งหมดที่ต้องการค้นหา
- textStart ข้อความที่ใช้เริ่มต้นค้นหาข้อความที่ต้องการ
- textEnd ข้อความที่ปรากฏต่อจากข้อความที่ต้องการค้นหา
- startIdxText ข้อความที่ใช้เป็นจุดอ้างอิงสำหรับการค้นหา หากไม่ระบุจะค้นหาตั้งแต่เริ่มต้น
- startPos ตำแหน่งตัวอักษรที่ใช้เป็นจุดอ้างองสำหรับการค้นหา หากมีการระบุพร้อมกับ startIdxText จะอ้างอิงตามตำแหน่งของ startIdxText หากไม่ระบุ จะยึดตามตำแหน่งอ้างอิงของ startIdxText

ตัวอย่างการใช้งาน

```
s.Name = GetTextBetween(XMIData, "name = ", "\"", "", 0);
```

3.5.1.2 คลาสสำหรับอ่านข้อมูลจากไฟล์ XMI ที่ได้จากโปรแกรม Rational Rose

ตามขอบเขตของโครงงานนี้ จะทำการพัฒนาโปรแกรมที่สามารถอ่านข้อมูลจากไฟล์ XMI ที่สร้างโดยโปรแกรม Rational Rose โดยใช้ชื่อคลาสว่า RoseXMIReader ซึ่งเป็นคลาสที่พัฒนาสืบทอดจากคลาสต้นแบบ XMIReader ในข้อ 3.4.1.1 โดยมีขั้นตอนการพัฒนาดังนี้

1. เตรียมข้อมูล XMI เพื่อการสร้างคลาส

จากการศึกษาโครงสร้างของไฟล์ XMI ที่ได้จากโปรแกรม RationalRose พบว่า ข้อมูลที่เก็บรายละเอียดของแผนภาพสถานะจะเริ่มตั้งแต่เอลิเมนต์ UML:StateMachine.top ดังนั้นขั้นแรกจะทำการตัดข้อมูลข้อความส่วนต้นออกเมื่อมีการกำหนดค่า XMIData ให้กับ Class โดยใช้คำสั่ง

```
XMIData = CutFrontText(XMIData, "<UML:StateMachine.top>");
```

2. ออกแบบเพื่อพัฒนาฟังก์ชัน GetAllState()

เป็นฟังก์ชันสำหรับสร้างคลาส State จากข้อมูล XMI มีรหัสคำสั่งลำลองแสดงการทำงานดังนี้

```
1       string tmpStr;
2       int tmpIdx;
3
4       tmpIdx = GetNextPositionOf("<UML") from begin;
5       tmpStr = GetNextElementName;
6       while ( tmpStr in StateElementGroup ) {
7          if ( isStateElement ) {
8             s = new State Class;
9             s.ID = GetTextBetween(AllText, "xmi.id = '", "'", tmpIdx);
10            s.Name = GetTextBetween(AllText, "name = '", "'", tmpIdx);
11            if ( tmpStr = pseudostate ) s.Type = StartState;
12            if ( tmpStr = finalstate ) s.Type = FinalState;
13            if ( tmpStr = simplestate ) s.Type = SimpleState;
14
15            StateData.Add( s );
16         }
17
18         tmpIdx = GetNextPositionOf("<UML") from tmpIdx;
19         tmpStr = GetNextElementName;
20      }
```

จากรหัสคำสั่งข้างต้นอธิบายการทำงานได้ดังนี้

1: สร้างตัวแปร tmpStr ในรูปแบบข้อความ

2: สร้างตัวแปร tmpIdx ในรูปแบบตัวเลข

4: ค้นหาตำแหน่งของข้อความ "<UML" และเก็บลงตัวแปร tmpIdx สำหรับใช้เป็นตำแหน่งอ้างอิง

5: ค้นหาชื่อเอลิเมนต์จากตำแหน่ง tmpIdx ล่าสุด

6: ทำงานซ้ำจนกว่าจะพบเอลิเมนต์ที่ไม่ใช่กลุ่มเอลิเมนต์ของสถานะ

7: ตรวจสอบ หากเป็นเอลิเมนต์สถานะให้สร้างคลาส

8: สร้างคลาส State ใหม่

9: อ่านค่ารหัสอ้างอิงจากข้อความ "xmi.id = 'xxx'" และบันทึกลงแอตทริบิวต์ ID

10: อ่านค่าชื่อสถานะจากข้อความ “name = ‘xxx’” และบันทึกลงแอตทริบิวต์ Name

11-13: ตรวจสอบประเภทของสถานะว่าเป็น สถานะเริ่มต้น สิ้นสุด หรือทั่วไป

15: เพิ่มคลาส State ใหม่เข้าสู่คลาสบรรจุ StateData

18: ค้นหาตำแหน่งของแอตทริบิวต์ถัดไปจากข้อความ “<UML” โดยอ้างอิงตำแหน่งเดิม

19: ค้นหาชื่อเอลิเมนต์จากตำแหน่ง tmpIdx ล่าสุด

3. ออกแบบเพื่อพัฒนาฟังก์ชัน GetAllTransition()

เป็นฟังก์ชันสำหรับสร้างคลาส Transition จากข้อมูล XMI มีรหัสคำสั่งลำลองแสดงการทำงานดังนี้

```
1       string tmpStr;
2       int tmpIdx;
3
4       tmpIdx = GetNextPositionOf("transition") from begin;
5       tmpIdx = GetNextPositionOf("<UML") from tmpIdx;
6       tmpStr = GetNextElementName;
7       while ( tmpStr in TransitionElementGroup ) {
8          if ( isTransitionElement ) {
9             t = new Transition Class;
10            t.ID = GetTextBetween(AllText, "xmi.id = '", "'", tmpIdx);
11            t.Name = GetTextBetween(AllText, "name = '", "'", tmpIdx);
12            t.Trigger = GetTextBetween(AllText, "trigger = '", "'", tmpIdx);
13            t.Incoming = StateData[ID = GetTextBetween(AllText, "source = '", "'",
        tmpIdx)];
14            t.Outgoing = StateData[ID = GetTextBetween(AllText, "target = '", "'",
        tmpIdx)];
15
16            TransitionData.Add( t );
17         }
18
19         tmpIdx = GetNextPositionOf("<UML") from tmpIdx;
20         tmpStr = GetNextElementName;
21      }
```

จากรหัสคำสั่งข้างต้นอธิบายการทำงานได้ดังนี้

1: สร้างตัวแปร tmpStr ในรูปแบบข้อความ

2: สร้างตัวแปร tmpIdx ในรูปแบบตัวเลข

4: ค้นหาตำแหน่งเริ่มต้นของกลุ่ม Transition จากข้อความ “transition”

5: ค้นหาตำแหน่งของข้อความ “<UML” และเก็บลงตัวแปร tmpIdx สำหรับใช้เป็นตำแหน่งอ้างอิง

6: ค้นหาชื่อเอลิเมนต์จากตำแหน่ง tmpIdx ล่าสุด

7: ทำงานซ้ำจนกว่าจะพบว่าเอลิเมนต์ไม่ใช่กลุ่มเอลิเมนต์ของการเปลี่ยนสถานะ

8: ตรวจสอบ ว่าเป็นเอลิเมนต์การเปลี่ยนสถานะหรือไม่

9: สร้างคลาส Transition ใหม่

10: อ่านค่ารหัสอ้างอิงจากข้อความ “xmi.id = ‘xxx’” และบันทึกลงแอตทริบิวต์ ID

11: อ่านค่าชื่อสถานะจากข้อความ “name = ‘xxx’” และบันทึกลงแอตทริบิวต์ Name

12: อ่านค่ารหัสของ Trigger จากข้อความ “trigger = ‘xxx’” และบันทึกลงแอตทริบิวต์ Trigger

13: อ่านค่ารหัสอ้างอิงของสถานะต้นทางจากข้อความ “source = ‘xxx’” จากนั้นนำ ID หาคลาส State จากคลาสบรรจุสถานะที่ทำการสร้างไว้ก่อนหน้านี้บันทึกลงในแอตทริบิวต์ Incoming

14: อ่านค่ารหัสอ้างอิงของสถานะปลายทางจากข้อความ “target = ‘xxx’” จากนั้นนำ ID หาคลาส State จากคลาสบรรจุสถานะที่ทำการสร้างไว้ก่อนหน้านี้บันทึกลงในแอตทริบิวต์ Outgoing

16: เพิ่มคลาสการเปลี่ยนสถานะใหม่เข้าสู่คลาสบรรจุ TransitionData

19: ค้นหาตำแหน่งของแอตทริบิวต์ถัดไปจากข้อความ “<UML” โดยอ้างอิงตำแหน่งเดิม

20: ค้นหาชื่อเอลิเมนต์จากตำแหน่ง tmpIdx ล่าสุด

4. ออกแบบเพื่อพัฒนาฟังก์ชัน GetAllRelation()

เป็นฟังก์ชันสำหรับสร้างการเชื่อมโยงของคลาสสถานะ และคลาสการเปลี่ยนสถานะจากข้อมูล XMI มีรหัสคำสั่งลำลองแสดงการทำงานดังนี้

```
1       for ( s = each State in StateData ) {
2           s.Incoming = TransitionData.GetAllFromOutGoing(s.ID);
3           s.Outgoing = TransitionData.GetAllFromIncoming(s.ID);
4       }
```

จากรหัสคำสั่งข้างต้นอธิบายการทำงานได้ดังนี้

1: ทำการเลือกสถานะทั้งหมดในคลาสบรรจุ StateData เพื่อทำการปรับปรุง

2: บันทึกข้อมูลการเปลี่ยนสถานะที่มีการเชื่อมโยงมาสู่คลาสสถานะลงในแอตทริบิวต์ Incoming โดยใช้ฟังก์ชัน GetAllFromOutGoing จากคลาส Transitions

3: บันทึกข้อมูลการเปลี่ยนสถานะที่มีการเชื่อมโยงจากคลาสสถานะลงในแอตทริบิวต์ Outgoing โดยใช้ฟังก์ชัน GetAllFromIncoming จากคลาส Transitions

3.5.1.4 ตัวอย่างการจัดเก็บข้อมูล

การแปลงข้อมูลเอลิเมนต์ในไฟล์ XMI และจัดเก็บในรูปแบบของคลาสโดยใช้คลาส RoseXMIReader แสดงตามตัวอย่างดังต่อไปนี้

ข้อมูลจากไฟล์ XMI:

```
<UML:SimpleState xmi.id = 'G.19'
  name = 'Bar at Max' visibility = 'public' isSpecification = 'false'
  outgoing = 'G.20 G.22' incoming = 'G.13' />
```

จากข้อมูลเอลิเมนต์ข้างต้นพบว่าเป็นเอลิเมนต์ของสถานะทั่วไป มีรหัสอ้างอิงคือ G.19 ชื่อเอลิเมนต์คือ Bar at Max มีทางออก 2 ทางคือไปยังการเปลี่ยนสถานะรหัส G.20 G.22 และมีการเปลี่ยนสถานะเข้าสู่สถานะ 1 ทางคือ รหัส G.13 ซึ่งจะสร้างคลาส State โดยมีข้อมูลจัดเก็บได้ดังนี้

คลาส State และข้อมูลที่จัดเก็บ:

```
        State() {
        Name = "Bar at Max"
        ID = "G.19"
        Incoming = Transition() { ID = "G.13" }
        Outgoing = Transition() { ID = "G.20" } ,
            Transition() { ID = "G.22" }
PreCode = ""
        }
```

3.5.2 การเพิ่มรหัสคำสั่งเบื้องต้น

ในการเพิ่มรหัสคำสั่งเบื้องต้นสามารถเพิ่มได้ 2 ส่วนคือ

3.5.2.1 เพิ่มคำสั่งเบื้องต้นในทุกสถานะ หรือเงื่อนไข

เป็นส่วนของรหัสที่จะถูกสร้างขึ้นทุกครั้งที่มีการสร้างฟังก์ชันสถานะ หรือฟังก์ชันเงื่อนไข

3.5.2.2 เพิ่มคำสั่งเบื้องต้นเฉพาะสถานะที่ต้องการ

เป็นส่วนของรหัสที่จะถูกสร้างขึ้นในสถานะที่กำหนด โดยข้อมูลของรหัสคำสั่งจะถูกเก็บไว้ยังแอตทริบิวต์ PreCode ในคลาส State ที่บรรจุอยู่ภายในคลาสบรรจุของระบบ

3.5.3 ทำการสร้างรหัสคำสั่งจากคลาสสำหรับจัดเก็บข้อมูล

เมื่อได้คลาสสถานะแล้ว โปรแกรมจะตรวจสอบชนิดของสถานะนั้น และทำการสร้างรหัสคำสั่งของฟังก์ชันต้นแบบของแต่ละสถานะ โดยในการสร้างแต่ละสถานะจะมีการตรวจสอบแอตทริบิวต์ Outgoing ว่ามี Transition อยู่จำนวนเท่าใด ซึ่งหมายถึงมีจำนวนทางออกจากสถานะนั้นเท่าใดเพื่อทำการสร้างรหัสคำสั่งในส่วนของเงื่อนไขดังนี้

- ถ้ามีทางออกทางเดียว ทำการเรียกฟังก์ชันของสถานะปลายทางทันที

- ถ้ามีทางออกมากกว่า 1 ทาง สร้างรหัสคำสั่งเงื่อนไข switch-case สำหรับการทำงานสถานะต่อไป และสร้างฟังก์ชันเงื่อนไขขึ้น เพื่อให้ผู้ใช้ทำการพัฒนาฟังก์ชันเงื่อนไขให้คืนค่าที่กำหนดไว้ตามที่ต้องการ

รหัสคำสั่งต้นแบบของสถานะ G.19 (Bar at Max):

```
// method for work of State Bar at Max
private void State_Bar_at_Max()
{

    // Code to run every state function


    // *** condition for Bar at Max:
    switch (Cond_Bar_at_Max()) {
    case "MinusClick":
        State_Bar_at_Middle();
        break;
    case "ExitClick":
        State_End();
        break;
    case "PlusClick":
        State_Bar_at_Max();
        break;

    }

}
```

ส่วนของฟังก์ชันเงื่อนไขที่ใช้ในการเลือกการทำงานสถานะต่อไป

รหัสคำสั่งต้นแบบฟังก์ชันเงื่อนไขของสถานะ G.19 (Bar at Max)

```
private string Cond_Bar_at_Max() {
    string returnVal = "";

    // Code to run every condition function

    // ***** Code to work for return result for condition in state Bar_at_Max

    // returnVal = "MinusClick";
    // returnVal = "ExitClick";
    // returnVal = "PlusClick";

    return returnVal;
}
```

แสดงค่าของตัวแปรที่ต้องส่งคืนทั้งหมดในรูปแบบของข้อคิดเห็น

โดยในการสร้างรหัสคำสั่งต้นแบบเงื่อนไขของสถานะจะต้องมีค่าของตัวแปรที่ฟังก์ชันต้องส่งคืนให้ เพื่อความสะดวกในการพัฒนาต่อไป

ตัวอย่างรหัสคำสั่งต้นแบบที่จะได้ จากแผนภาพสถานะในรูปที่ 3.1

```
// method for work of State End
private void State_End()
{

    // Code to run every state function


    //End
}
```

```
// method for work of State Start
private void State_Start()
{

    // Code to run every state function


    // One Transition Just Call Next Function
    State_Bar_at_Min();

}

// method for work of State Bar at Min
private void State_Bar_at_Min()
{

    // Code to run every state function


    // *** condition for Bar at Min
    switch (Cond_Bar_at_Min()) {
    case "PlusClick":
        State_Bar_at_Middle();
        break;
    case "ExitClick":
        State_End();
        break;
    case "MinusClick":
        State_Bar_at_Min();
        break;

    }

}

// method for work of State Bar at Middle
private void State_Bar_at_Middle()
{

    // Code to run every state function


    // *** condition for Bar at Middle
    switch (Cond_Bar_at_Middle()) {
    case "MinusClick":
        State_Bar_at_Min();
        break;
    case "PlusClick":
        State_Bar_at_Max();
        break;
    case "ExitClick":
        State_End();
        break;

    }

}

// method for work of State Bar at Max
private void State_Bar_at_Max()
{

    // Code to run every state function


    // *** condition for Bar at Max
    switch (Cond_Bar_at_Max()) {
    case "MinusClick":
        State_Bar_at_Middle();
        break;
    case "ExitClick":
        State_End();
        break;
```

สถานะที่มีทางออกเดียว จะทำการเรียกสถานะถัดไปโดยไม่มีเงื่อนไข

สถานะที่มีทางออกมากกว่า 1 ทาง จะเรียกใช้ฟังก์ชันเงื่อนไขเพื่อใช้ในการกำหนดการทำงานสถานะถัดไป

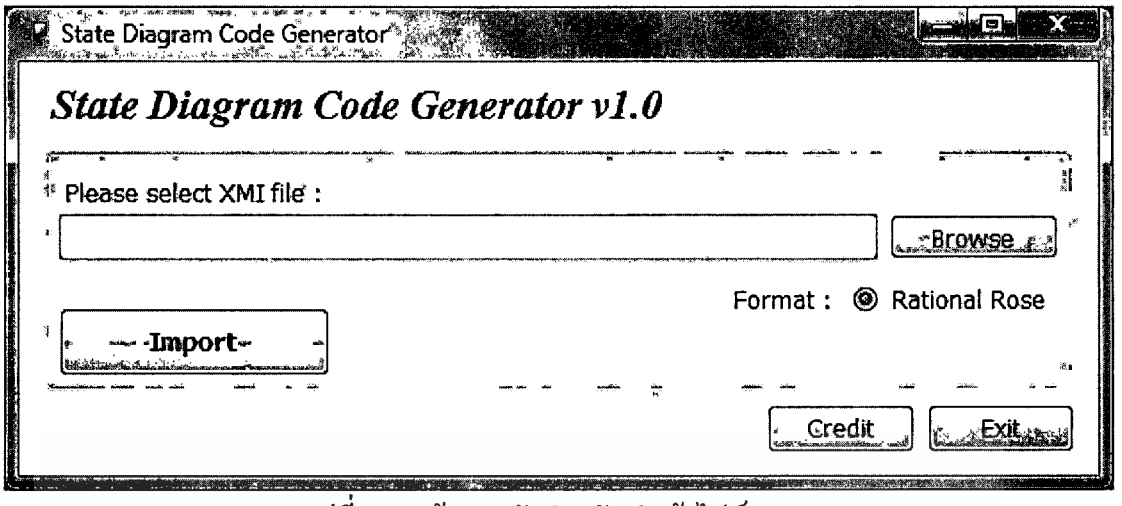
```
    case "PlusClick":
        State_Bar_at_Max();
        break;

    }
}
```

ส่วนของฟังก์ชันเงื่อนไขที่ใช้ในการเลือกการทำงานสถานะต่อไป

```
// ********************************************************************************
// ********************  Condition Function NEED TO IMPLEMENT  ********************
// ********************************************************************************

private string Cond_Bar_at_Min() {
    string returnVal = "";

    // Code to run every condition function

    // ***** Code to work for return result for condition in state Bar_at_Min

    // returnVal = "PlusClick";
    // returnVal = "ExitClick";
    // returnVal = "MinusClick";

    return returnVal;
}

private string Cond_Bar_at_Middle() {
    string returnVal = "";

    // Code to run every condition function

    // ***** Code to work for return result for condition in state Bar_at_Middle

    // returnVal = "MinusClick";
    // returnVal = "PlusClick";
    // returnVal = "ExitClick";

    return returnVal;
}

private string Cond_Bar_at_Max() {
    string returnVal = "";

    // Code to run every condition function

    // ***** Code to work for return result for condition in state Bar_at_Max

    // returnVal = "MinusClick";
    // returnVal = "ExitClick";
    // returnVal = "PlusClick";

    return returnVal;
}
```

แสดงเงื่อนไขทั้งหมดที่ต้องทำการส่งค่ากลับเพื่อให้ทำงานได้ตรงตามเงื่อนไข

3.6 การออกแบบส่วนต่อประสานกับผู้ใช้



รูปที่ 3.6 หน้าจอหลักสำหรับนำเข้าไฟล์ XMI

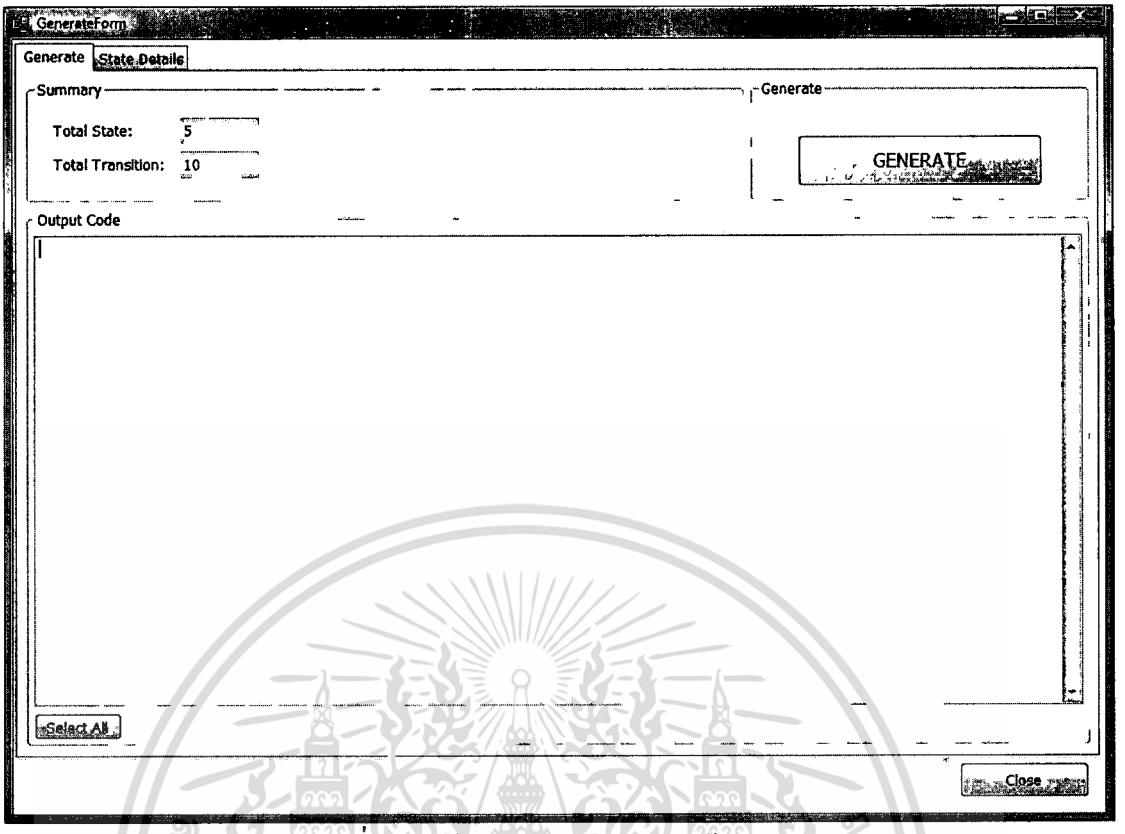
3.6.1 หน้าจอหลัก สำหรับนำเข้าข้อมูลจากไฟล์ XML ที่อยู่ในโครงสร้างของ XMI

- ปุ่ม Browse สำหรับแสดงรายชื่อไฟล์ในเครื่องเพื่อนำเข้า
- ปุ่ม Import สำหรับนำเข้าข้อมูล XML หลังจากเลือกไฟล์แล้ว
- ปุ่ม Credit แสดงข้อมูลผู้พัฒนา
- ปุ่ม Exit สำหรับออกจากโปรแกรม

3.6.2 หน้าจอหลักสำหรับการสร้างรหัส จะแบ่งเป็น 2 แท็บคือ

3.6.2.1 แท็บ Generate แสดงรายละเอียดของไฟล์ XMI ที่นำเข้า แสดงจำนวนสถานะและการเปลี่ยนสถานะที่พบ รวมถึงรหัสคำสั่งที่ถูกสร้างจะแสดงในกล่องข้อความ

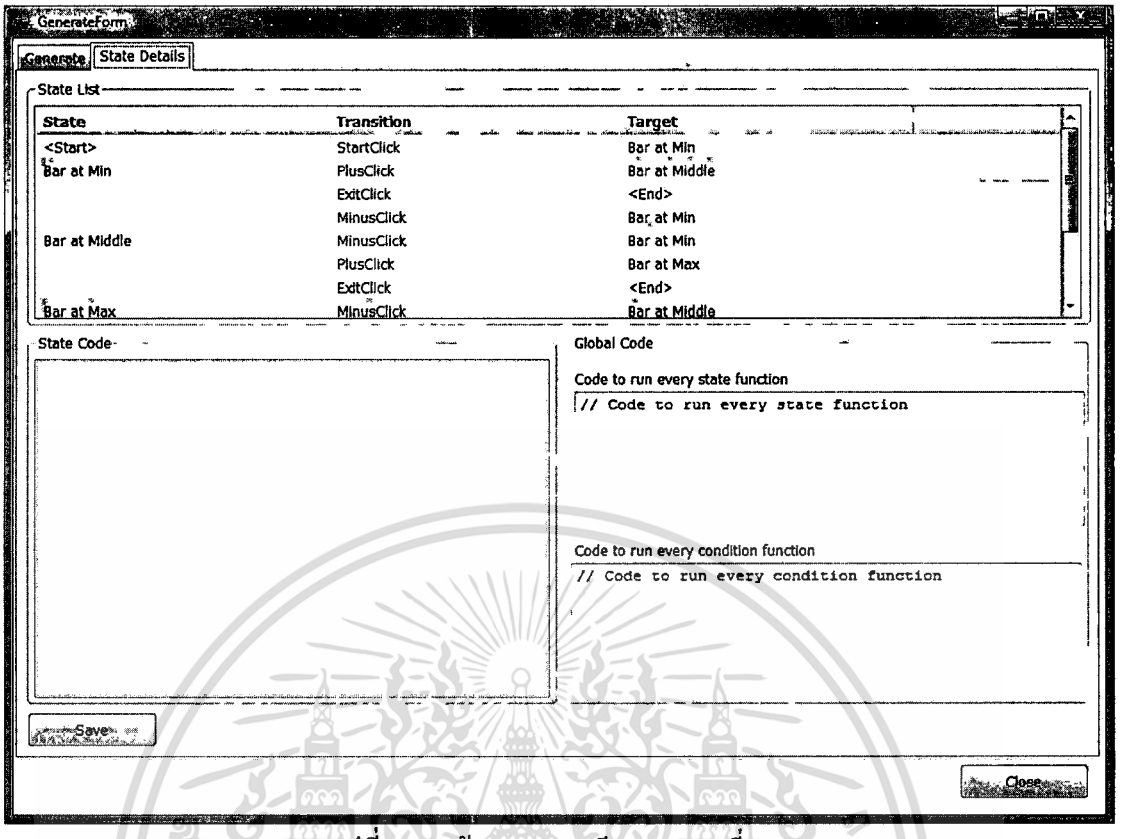
- ปุ่ม Close เพื่อยกเลิกการทำงานและกลับไปยังหน้าหลัก
- ปุ่ม Select All เพื่อทำการเลือกรหัสคำสั่งในช่องข้อความทั้งหมด



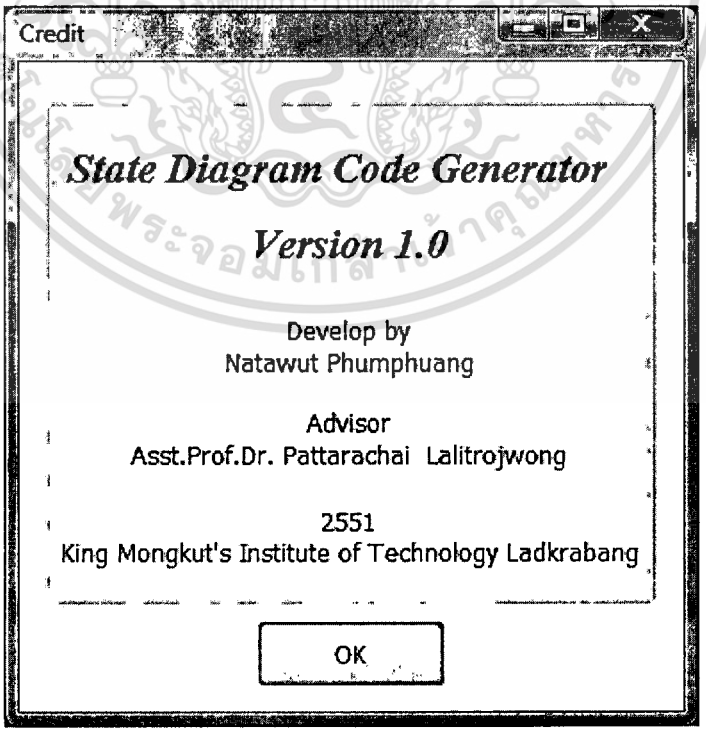
รูปที่ 3.7 หน้าจอหลักสำหรับการสร้างรหัส

3.6.2.2 หน้าจอการสร้างรหัส แท็บรายละเอียดสถานะ

- แสดงรายละเอียดของสถานะจากไฟล์ XML ในกล่องรายการ
- เลือกรายการที่ต้องการเขียนรหัสคำสั่งเบื้องต้นจากตารางด้านบน
- เขียนรหัสคำสั่งเบื้องต้นลงของสถานะที่เลือกลงในกล่องข้อความด้านซ้าย
- ปุ่ม Save เพื่อบันทึกรหัสที่เพิ่ม สำหรับใช้ในการสร้างรหัส
- กล่องข้อความด้านขวาสำหรับใส่รหัสคำสั่งที่ต้องการให้มีอยู่ในทุกฟังก์ชัน ของสถานะหรือทุกฟังก์ชันของเงื่อนไขการเปลี่ยนสถานะ



รูปที่ 3.8 หน้าจอรายละเอียดสถานะที่พบ



รูปที่ 3.9 หน้าจอข้อมูลผู้พัฒนา

3.6.3 หน้าจอแสดงข้อมูลผู้พัฒนา

- แสดงรายละเอียดของผู้พัฒนา
- ปุ่ม OK กด เพื่อปิดหน้าจอ

บทที่ 4
การพัฒนาโปรแกรม

โปรแกรมการสร้างรหัสคำสั่งจากแผนภาพสถานะนี้ทำการพัฒนาโดยใช้ภาษา C# โดยใช้เครื่องมือพัฒนาคือ Visual Studio.Net 2005 ในการพัฒนา

โดยการพัฒนาแบ่งออกเป็นสองส่วนหลักคือ ส่วนการแปลงข้อมูลรูปแบบ XML สู่วัตถุข้อมูล และส่วนการสร้างรหัสคำสั่งจาก วัตถุข้อมูล เพื่อรองรับการแปลงรหัสคำสั่งจากไฟล์ XMI ที่นำออกจากโปรแกรมอื่นๆนอกเหนือจาก Rational Rose ซึ่งอาจมีรูปแบบของไฟล์ XML ที่แตกต่างออกไป

4.1 คลาสสำหรับจัดเก็บข้อมูล

คลาสสำหรับจัดเก็บข้อมูลที่อ่านได้จากไฟล์ XMI ที่ใช้จะแบ่งเป็นสองกลุ่มคือ

4.1.1 กลุ่มคลาสสำหรับบันทึกข้อมูลสถานะ

ประกอบด้วย คลาส State สำหรับเก็บข้อมูล 1 สถานะ และ States สำหรับเก็บข้อมูลสถานะทั้งหมด

4.1.1.1 คลาส State สำหรับเก็บข้อมูลสถานะ 1 สถานะ

```
public class State
    {
        string _Name = "";
        string _ID = "";
        StateTypes _StateType = StateTypes.Simple;
        Transitions _Incoming;
        Transitions _OutGoing;
        string _PreCode = "";

        public string Name
        {
            get { return _Name; }
            set { _Name = value; }
        }

        public string ID
        {
            get { return _ID; }
            set { _ID = value; }
        }

        public StateTypes StateType
        {
            get { return _StateType; }
            set { _StateType = value; }
        }

        public Transitions Incoming
        {
            get { return _Incoming; }
            set { _Incoming = value; }
```

```
    }

    public Transitions Outgoing
    {
        get { return _OutGoing; }
        set { _OutGoing = value; }
    }

    public bool Equals(State st)
    {
        if (this.ID.Equals(st.ID) && this.Name.Equals(st.Name))
            return true;

        return false;
    }

    public string PreCode
    {
        get { return _PreCode; }
        set { _PreCode = value; }
    }

    public enum StateTypes
    {
        Psuedoe,
        Simple,
        Final
    }
```

4.1.1.2 คลาส States สำหรับเป็นคลาสบรรจุสถานะทั้งหมด

```
public class States
{
    private ArrayList arrList;

    public States()
    {
        arrList = new ArrayList();
    }

    public void Add(State sData)
    {
        for (int i = 0; i < arrList.Count; i++)
        {
            if (((State)arrList[i]).ID == sData.ID)
                throw new Exception("ID already exists");
        }
        arrList.Add(sData);
    }

    public virtual void Remove(State sData)
    {
        for (int i = 0; i < arrList.Count; i++)
        {
            if (sData.Equals((State)arrList[i]))
            {
                arrList.RemoveAt(i);
                break;
            }
        }
    }

    public void RemoveAt(int Index)
    {
        arrList.RemoveAt(Index);
    }

    public int Count
    {
        get { return arrList.Count; }
```

```
        }

        public void Clear()
        {

        }

        public State this[string StateID]
        {
            get
            {
                int idx = IndexOfKey(StateID);

                if (idx != -1)
                    return (State)arrList[idx];
                else
                    return null;
            }
        }

        public State this[int Index]
        {
            get { return (State)arrList[Index]; }
        }

        private int IndexOfKey(string zKey)
        {
            int retIdx = -1;
            for (int i = 0; i < arrList.Count; i++)
            {
                if (((State)arrList[i]).ID == zKey)
                {
                    retIdx = i;
                    break;
                }
            }
            return retIdx;
        }
    }
```

โดยคลาส States ซึ่งเป็นคลาสบรรจุ ออกแบบให้สามารถเรียกสถานะได้จาก ID ที่สร้างขึ้นโดยโปรแกรม Case Tool

4.1.2 กลุ่มคลาสสำหรับบันทึกการเปลี่ยนสถานะ

กลุ่มที่สอง ประกอบด้วยคลาส Transition และ คลาส Transitions สำหรับบรรจุคลาส Transition ทั้งหมด

4.1.2.1 คลาส Transition สำหรับเก็บข้อมูลการเปลี่ยนสถานะ 1 เหตุการณ์

```
public class Transition
    {
        string _Name;
        string _ID;
        string _Trigger;
        State _Incomong;
        State _Outgoing;

        public string Name
        {
```

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

        public string ID
        {
            get { return _ID; }
            set { _ID = value; }
        }

        public State Incoming
        {
            get { return _Incomong; }
            set { _Incomong = value; }
        }

        public State Outgoing
        {
            get { return _Outgoing; }
            set { _Outgoing = value; }
        }

        public string Trigger
        {
            get { return _Trigger; }
            set { _Trigger = value; }
        }

    }
```

4.1.2.2 คลาส Transitions สำหรับเก็บข้อมูลการเปลี่ยนสถานะทั้งหมด

```
public class Transitions
    {
        private ArrayList arrTrans;

        public Transitions()
        {
            arrTrans = new ArrayList();
        }

        public void Add(Transition TransitionObj)
        {
            for (int i = 0; i < arrTrans.Count; i++)
            {
                if (((Transition)arrTrans[i]).ID == TransitionObj.ID)
                    throw new Exception("ID already exists");
            }
            arrTrans.Add(TransitionObj);
        }

        public void Remove(Transition TransitionObj)
        {
            for (int i = 0; i < arrTrans.Count; i++)
            {
                if (TransitionObj.Equals((Transition)arrTrans[i]))
                {
                    arrTrans.RemoveAt(i);
                    break;
                }
            }
        }

        public void RemoveAt(int Index)
        {
            arrTrans.RemoveAt(Index);
        }

        public void Clear()
        {
            arrTrans.Clear();
        }
```

```
        public int Count
        {
            get { return arrTrans.Count; }
        }

        public Transition this[string ID]
        {
            get
            {
                int idx = IndexOfKey(ID);

                if (idx != -1)
                    return (Transition)arrTrans[idx];
                else
                    return null;
            }
            set
            {
                int idx = IndexOfKey(ID);

                if (idx != -1)
                    arrTrans[idx] = value;
            }
        }

        public Transition this[int Index]
        {
            get { return (Transition)arrTrans[Index]; }
        }

        private int IndexOfKey(string zKey)
        {
            int retIdx = -1;
            for (int i = 0; i < arrTrans.Count; i++)
            {
                if (((Transition)arrTrans[i]).ID == zKey)
                {
                    retIdx = i;
                    break;
                }
            }
            return retIdx;
        }

        public Transitions GetAllFromIncoming(string IncomingID)
        {
            Transitions t = new Transitions();
            for (int i = 0; i < arrTrans.Count; i++)
            {
                if (((Transition)arrTrans[i]).Incoming.ID == IncomingID)
                    t.Add((Transition)arrTrans[i]);
            }
            return t;

        }

        public Transitions GetAllFromOutgoing(string OutgoingID)
        {
            Transitions t = new Transitions();
            for (int i = 0; i < arrTrans.Count; i++)
            {
                if (((Transition)arrTrans[i]).Outgoing.ID == OutgoingID)
                    t.Add((Transition)arrTrans[i]);
            }
            return t;
        }
    }
```

4.2 คลาสสำหรับอ่านข้อมูลจาก XMI

เพื่อการพัฒนาต่อในอนาคต คลาสสำหรับอ่านข้อมูลจากไฟล์ XMI ซึ่งไฟล์ XMI ที่สร้างจากโปรแกรมที่แตกต่างกัน อาจมีรายละเอียดแตกต่างกัน ดังนั้น ผู้พัฒนาจึงสร้างคลาสอ่านข้อมูลเป็นลักษณะคลาสต้นแบบ (abstract)

4.2.1 คลาสต้นแบบ สำหรับสร้างคลาสการอ่านข้อมูล XMI

```
public abstract class XMIReader
{
    protected string XMIData;
    States StateData;
    Transitions TransitionData;

    public XMIReader(string XMITextData)
    {
        XMIData = XMITextData;
        StateData = new States();
        TransitionData = new Transitions();
    }

    public XMIReader()
    {
        StateData = new States();
        TransitionData = new Transitions();
    }

    public abstract void GetAllState();
    public abstract void GetAllTransition();
    public abstract void GetAllRelation();
```

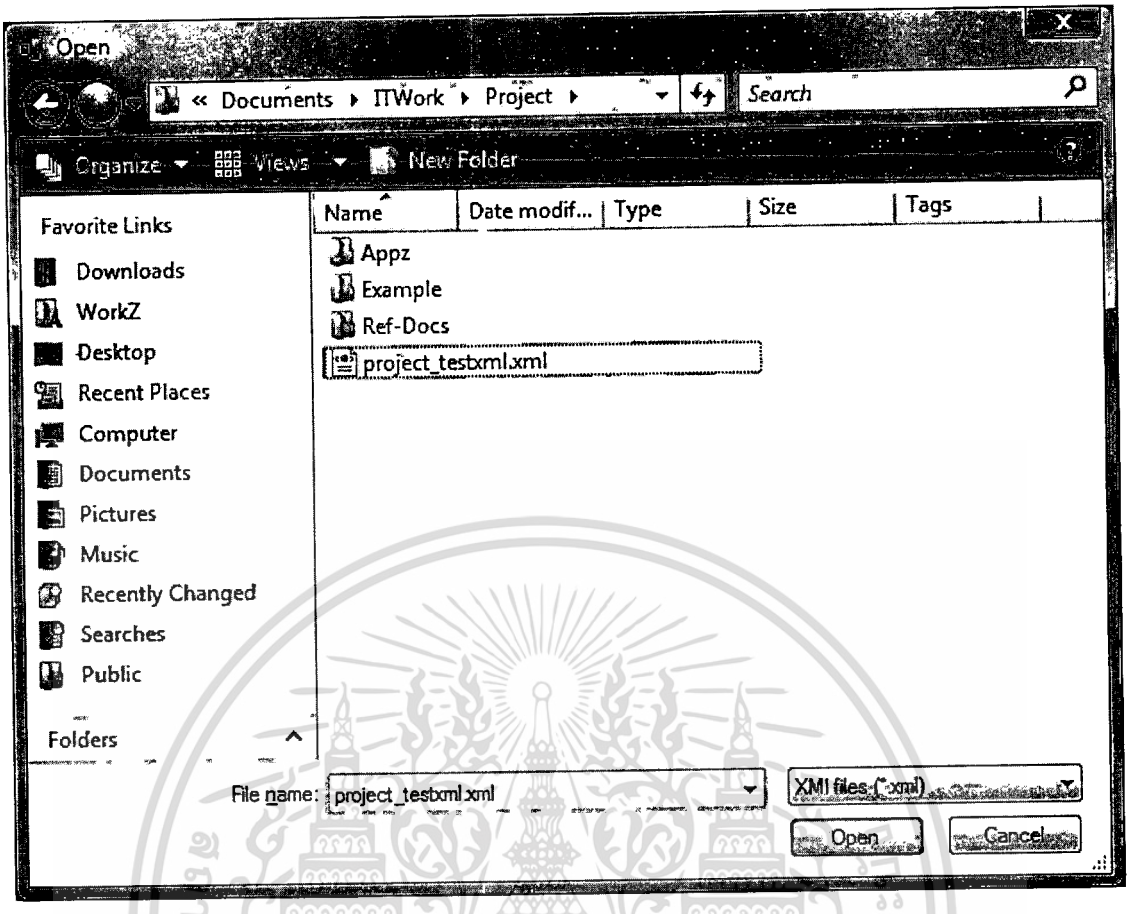
4.2.2 คลาสสำหรับอ่านข้อมูลที่สร้างโดยโปรแกรม Rational Rose

```
public class RoseXMIReader : XMIReader
{
    public RoseXMIReader(string xmiData)
    {
        SetXMIData(xmiData);
    }
    public void SetXMIData(string xmiData)
    {
        XMIData = xmiData;
        PrepareString();

        Program.tmpStr = XMIData; // for debug
    }

    void PrepareString()
    {
        XMIData = CutFrontText(XMIData, "<UML:StateMachine.top>");
    }

    public override void GetAllState()
    {
        string tmpStr;
        int tmpIdx;

        tmpIdx = XMIData.IndexOf("<UML:");
        tmpStr = GetTextBetween(XMIData, "<UML:", " ", tmpIdx).ToLower();
        while (tmpStr.Contains("state"))
        {
            if (!tmpStr.Contains("composite") && !tmpStr.Contains("statemachine"))
            {
```

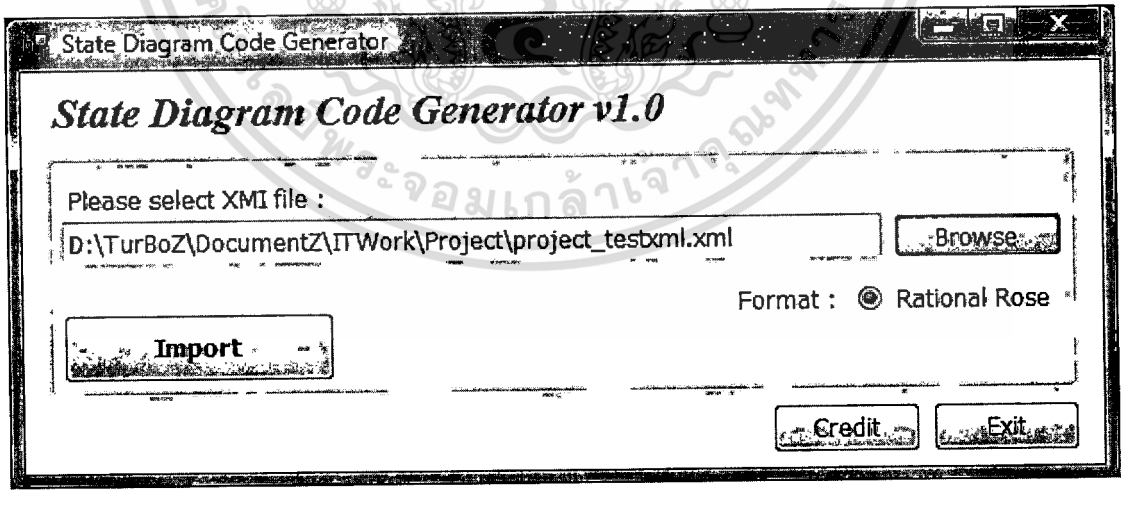
```
                    State s = new State();
                    s.ID = GetTextBetween(XMIData, "xmi.id = '", "'", tmpIdx);
                    s.Name = GetTextBetween(XMIData, "name = '", "'", tmpIdx);
                    if (tmpStr.Contains("pseudo")) s.StateType =
State.StateTypes.Psuedoe;
                    if (tmpStr.Contains("final")) s.StateType =
State.StateTypes.Final;
                    if (tmpStr.Contains("simple")) s.StateType =
State.StateTypes.Simple;

                    Program.StateData.Add(s);
                }

                tmpIdx = XMIData.IndexOf("<UML:", tmpIdx + 4);
                tmpStr = GetTextBetween(XMIData, "<UML:", " ", tmpIdx).ToLower();


            }
        }

        public override void GetAllTransition()
        {
            string tmpStr;
            int tmpIdx;

            tmpIdx = XMIData.IndexOf(".transitions");
            tmpIdx = XMIData.IndexOf("<UML:", tmpIdx);
            tmpStr = GetTextBetween(XMIData, "<UML:", " ", tmpIdx).ToLower();
            while (tmpStr == "transition")
            {
                Transition t = new Transition();
                t.ID = GetTextBetween(XMIData, "xmi.id = '", "'", tmpIdx);
                t.Trigger = GetTextBetween(XMIData, "trigger = '", "'", tmpIdx);
                t.Incoming = Program.StateData[GetTextBetween(XMIData, "source = '",
"'", tmpIdx)];
                t.Outgoing = Program.StateData[GetTextBetween(XMIData, "target = '",
"'", tmpIdx)];

                t.Name = GetTextBetween(XMIData, "name = '", "'",
XMIData.IndexOf("SignalEvent xmi.id = '" + t.Trigger + "'"));

                Program.TransitionData.Add(t);

                tmpIdx = XMIData.IndexOf("<UML:", tmpIdx + 4);
                tmpStr = GetTextBetween(XMIData, "<UML:", " ", tmpIdx).ToLower();
                }


        }

        public override void GetAllRelation()
        {
            for (int i = 0; i < Program.StateData.Count; i++)
            {
                State s = Program.StateData[i];
                s.Incoming = Program.TransitionData.GetAllFromOutgoing(s.ID);
                s.Outgoing = Program.TransitionData.GetAllFromIncoming(s.ID);
            }

        }
    }
```

4.3 การทำงานของโปรแกรมสร้างรหัสคำสั่ง

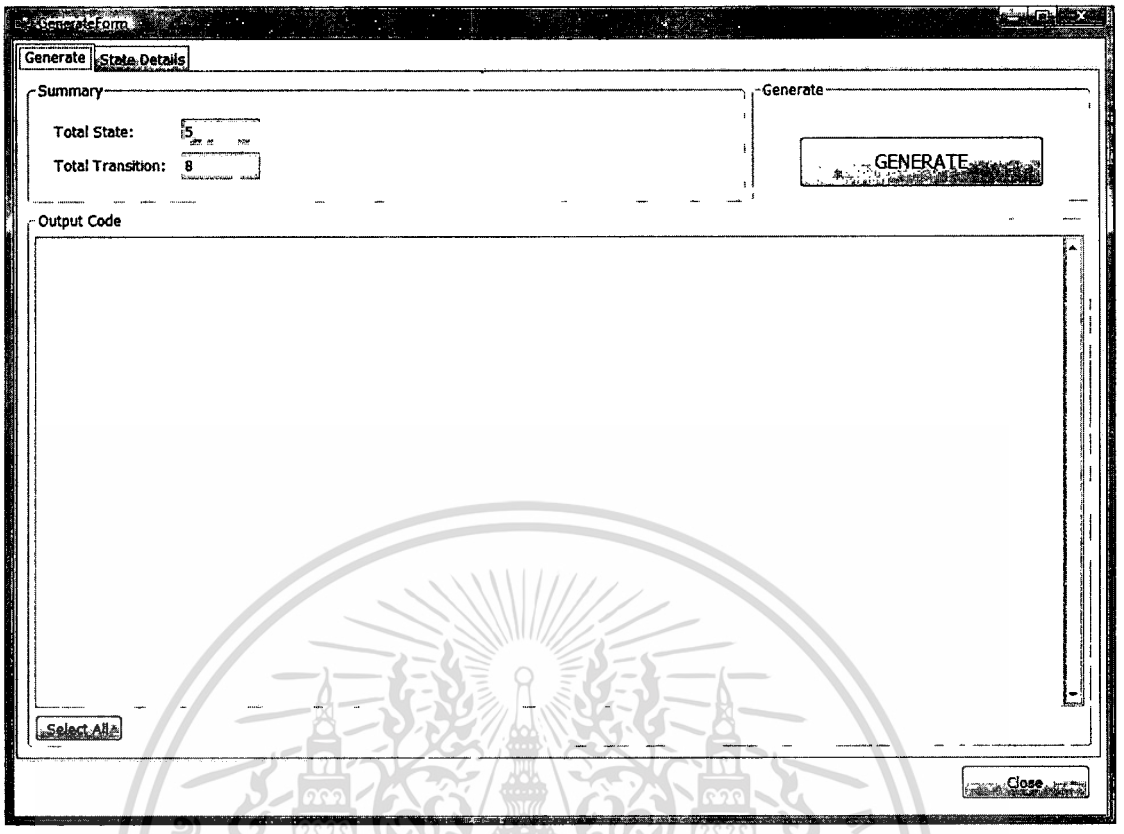
ตัวอย่างขั้นตอนการใช้งานโปรแกรม



รูปที่ 4.1 เลือกไฟล์ xml ที่ต้องการสร้างรหัสคำสั่ง

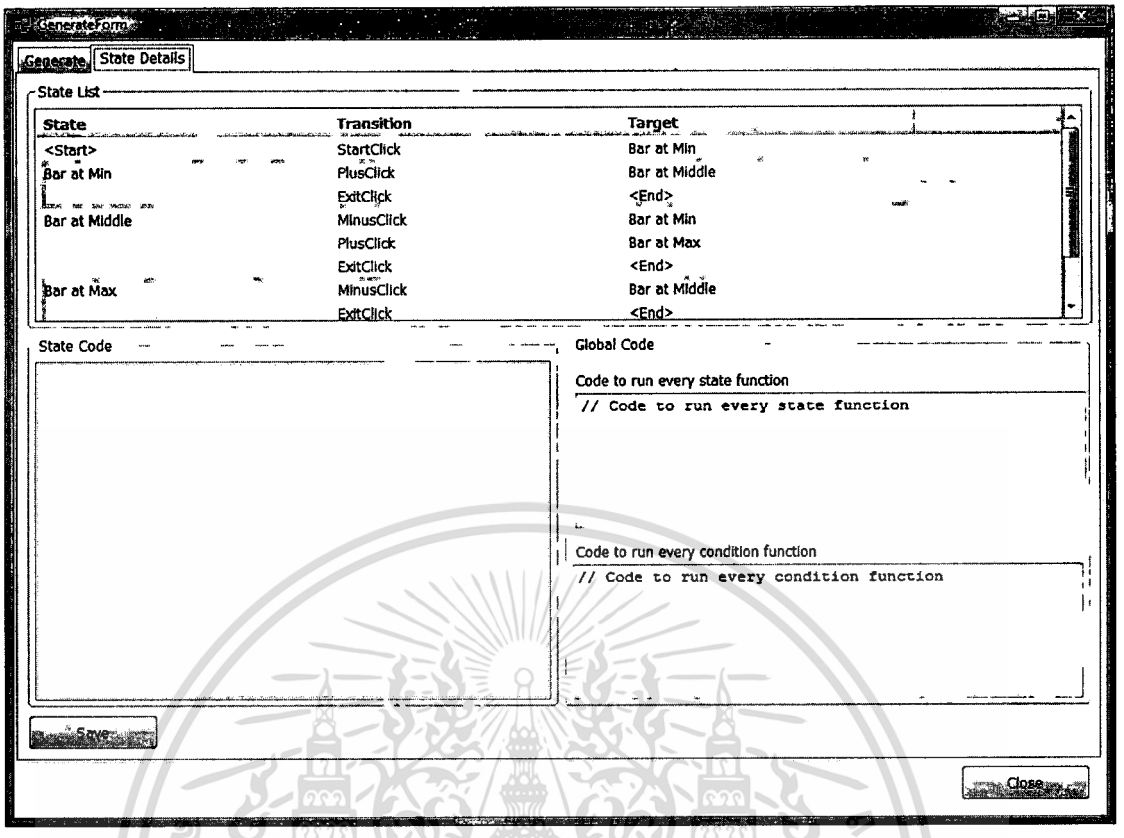


รูปที่ 4.2 เมื่อเลือกไฟล์ที่ต้องการกด Import

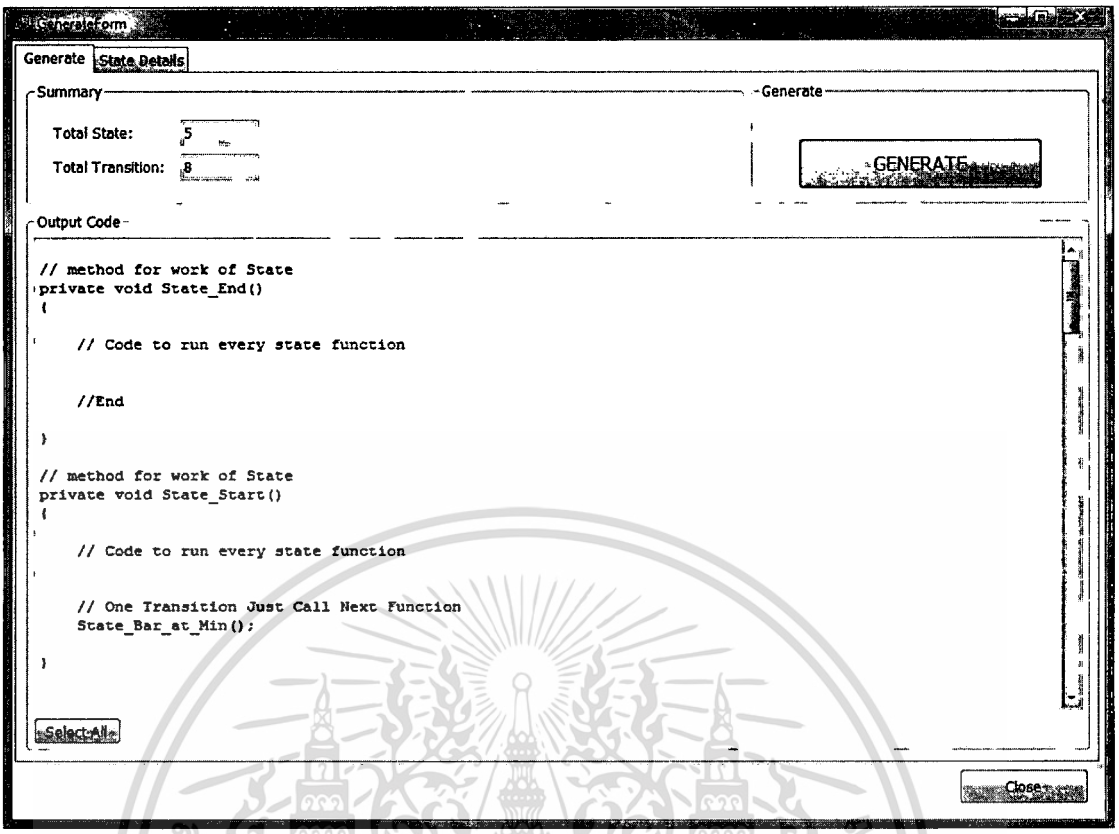


รูปที่ 4.3 หน้าจอสร้างรหัสคำสั่ง

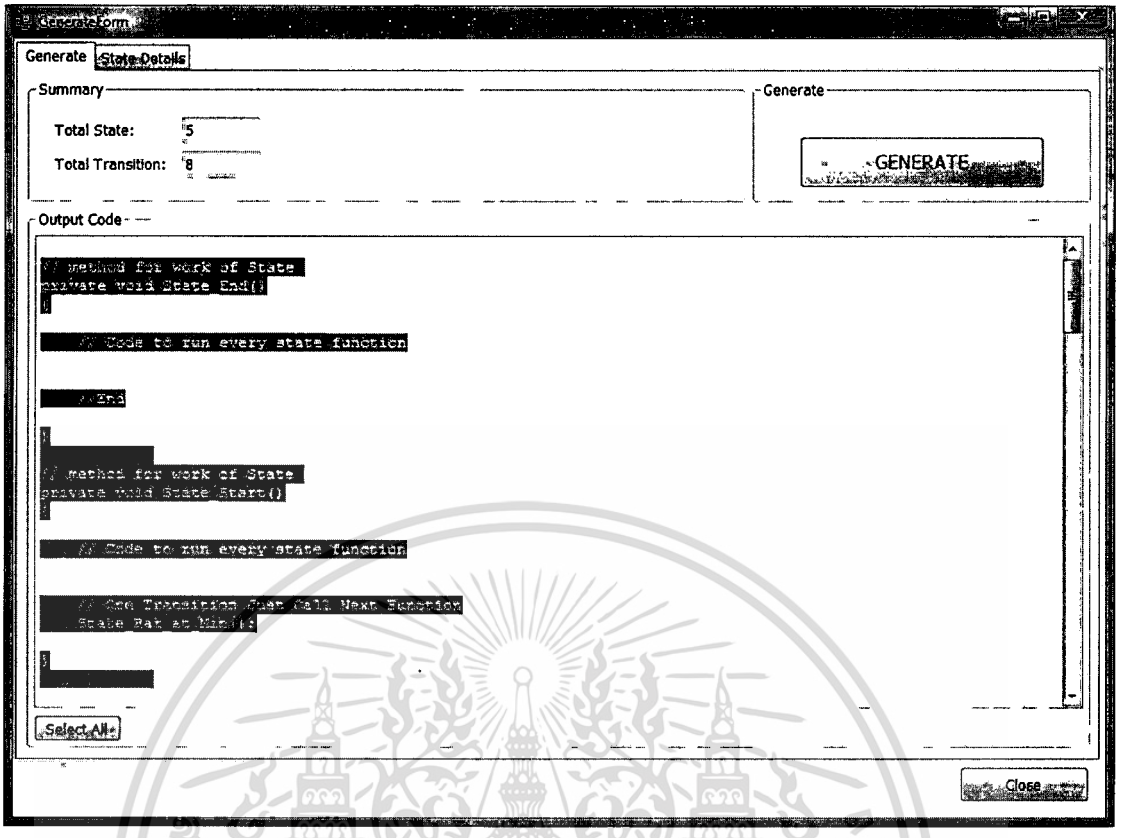
โปรแกรมจะแสดงจำนวนสถานะ และการเปลี่ยนสถานะที่อ่านได้จากไฟล์ XMI หากไม่ต้องการแก้ไขรหัสเบื้องต้น ผู้ใช้สามารถกด GENERATE เพื่อทำการสร้างรหัสคำสั่งมาตรฐานได้ทันที



รูปที่ 4.4 หน้าจอแท็บ State Detail แสดงรายละเอียดสถานะ และสามารถเพิ่มรหัสคำสั่งได้



รูปที่ 4.5 หน้าจอหลังจากกด Generate จะได้รหัสคำสั่งในช่อง Output Code



รูปที่ 4.6 หน้าจอการทำงานเมื่อกดปุ่ม Select All

4.4 การทดสอบการนำไปใช้งาน

ในตัวอย่างแรกจะแสดงให้เห็นการพัฒนารหัสคำสั่งให้โปรแกรมสามารถทำงานอย่างสมบูรณ์โดยใช้โปรแกรมที่ออกแบบแผนภาพสถานะในรูปที่ 3.1 โดยขั้นตอนในการทำงานทั้งหมดอย่างละเอียดจะแสดงในตัวอย่างที่ 2

เนื่องจากผู้เขียนมีความถนัดการพัฒนาด้วยภาษา C# และโครงสร้างภาษาของ Java และ C# มีความใกล้เคียงกันมาก ผู้เขียนจึงใช้การทดสอบโปรแกรมด้วยโปรแกรมภาษา C#

4.4.1 โปรแกรมทดสอบที่ 1 โปรแกรมเปลี่ยนแปลงตำแหน่งบาร์

จากโปรแกรมที่ออกแบบไว้ในรูปที่ 3.1 เป็นโปรแกรมง่ายๆสำหรับปรับตำแหน่งบาร์ได้ 3 ระดับคือ น้อยสุด (Min) ปานกลาง (Middle) และสูงสุด (Max) จากรหัสคำสั่งที่โปรแกรมได้สร้างขึ้น เมื่อทำการพัฒนาเพิ่มเติมจนโปรแกรมที่ออกแบบไว้ทำงานได้สมบูรณ์จะมีรหัสคำสั่งดังนี้

```
// method for work of State
private void State_End()
{

    // Code to run every state function
    [illegible]

    //End

}

// method for work of State
private void State_Start()
{

    // Code to run every state function
    [illegible]

    // One Transition Just Call Next Function
    State_Bar_at_Min();

}

// method for work of State Bar at Min
private void State_Bar_at_Min()
{

    // Code to run every state function
    [illegible]


    // *** condition for Bar at Min
    switch (Cond_Bar_at_Min())
    {
        case "PlusClick":
            State_Bar_at_Middle();
            break;
        case "ExitClick":
            State_End();
            break;
        case "MinusClick":
            State_Bar_at_Min();
            break;

    }

}

// method for work of State Bar at Middle
private void State_Bar_at_Middle()
{
```

การทำงานเมื่อโปรแกรมอยู่ในสถานะ Min

```
        // Code to run every state function
        [redacted]
        [redacted]
        [redacted]


        // *** condition for Bar at Middle
        switch (Cond_Bar_at_Middle())
        {
            case "MinusClick":
                State_Bar_at_Min();
                break;
            case "PlusClick":
                State_Bar_at_Max();
                break;
            case "ExitClick":
                State_End();
                break;

        }

    }

    // method for work of State Bar at Max
    private void State_Bar_at_Max()
    {

        // Code to run every state function
        [redacted]
        [redacted]
        [redacted]


        // *** condition for Bar at Max
        switch (Cond_Bar_at_Max())
        {
            case "MinusClick":
                State_Bar_at_Middle();
                break;
            case "ExitClick":
                State_End();
                break;
            case "PlusClick":
                State_Bar_at_Max();
                break;

        }

    }


    // ************************************************************************
    // ********************  Condition Function NEED TO IMPLEMENT  ****************
    // ************************************************************************

    private string Cond_Bar_at_Min()
    {
        string returnVal = "";

        // Code to run every condition function

        // ***** Code to work for return result for condition in state Bar_at_Min
        [redacted]

        // returnVal = "PlusClick";
        // returnVal = "ExitClick";
        // returnVal = "MinusClick";

        return returnVal;
    }

    private string Cond_Bar_at_Middle()
    {
        string returnVal = "";

        // Code to run every condition function
```

การทำงานเมื่อโปรแกรมอยู่ในสถานะ Middle

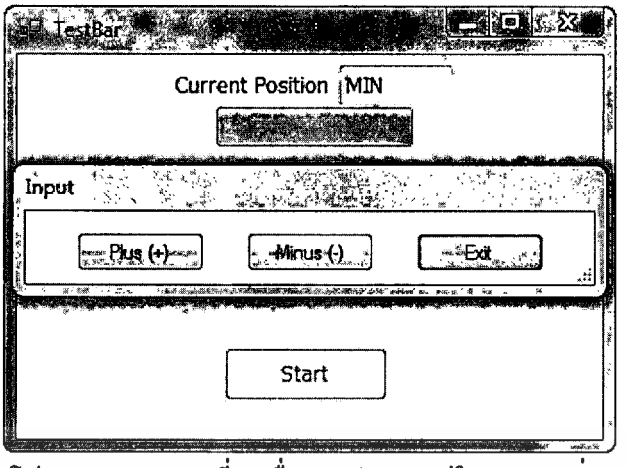
การทำงานเมื่อโปรแกรมอยู่ในสถานะ Max

ฟังก์ชันในการกำหนดเงื่อนไข

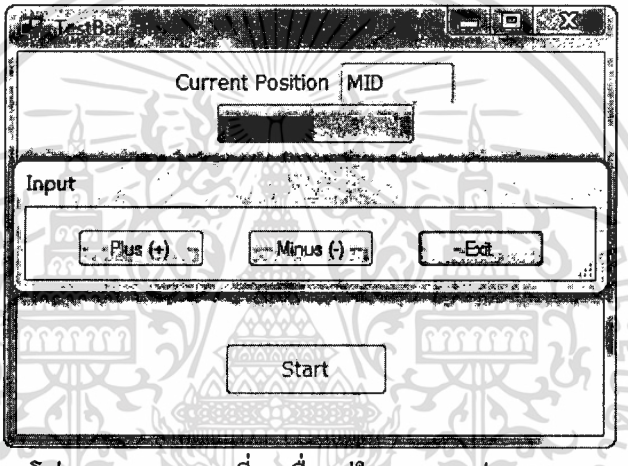
```
        // ***** Code to work for return result for condition in state Bar_at_Middle
        [redacted]

        // returnVal = "MinusClick";
        // returnVal = "PlusClick";
        // returnVal = "ExitClick";

        return returnVal;
    }

    private string Cond_Bar_at_Max()
    {
        string returnVal = "";

        // Code to run every condition function

        // ***** Code to work for return result for condition in state Bar_at_Max
        [redacted]

        // returnVal = "MinusClick";
        // returnVal = "ExitClick";
        // returnVal = "PlusClick";

        return returnVal;
    }
```

รหัสคำสั่งที่ทำแถบสีคือรหัสคำสั่งที่เพิ่มเพื่อให้การทำงานสมบูรณ์ ผลการทำงานของโปรแกรมแสดงดังรูปต่อไปนี้

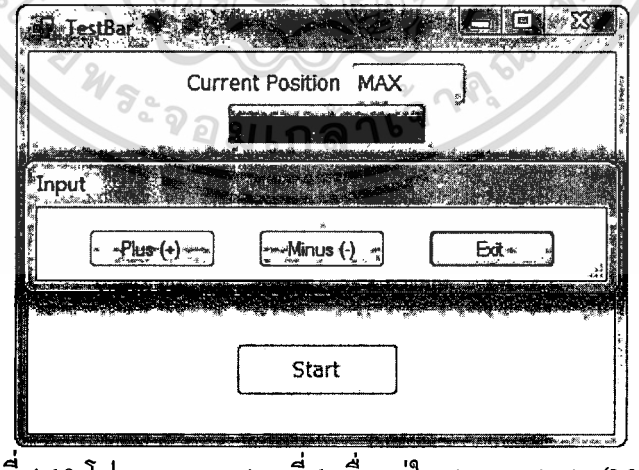
รูปที่ 4.7 หน้าจอโปรแกรมทดสอบ 1 เมื่อเปิดโปรแกรม



รูปที่ 4.8 โปรแกรมทดสอบที่ 1 เมื่อการทำงานอยู่ในสถานะต่ำสุด (Min)



รูปที่ 4.9 โปรแกรมทดสอบที่ 1 เมื่ออยู่ในสถานะปานกลาง (Middle)



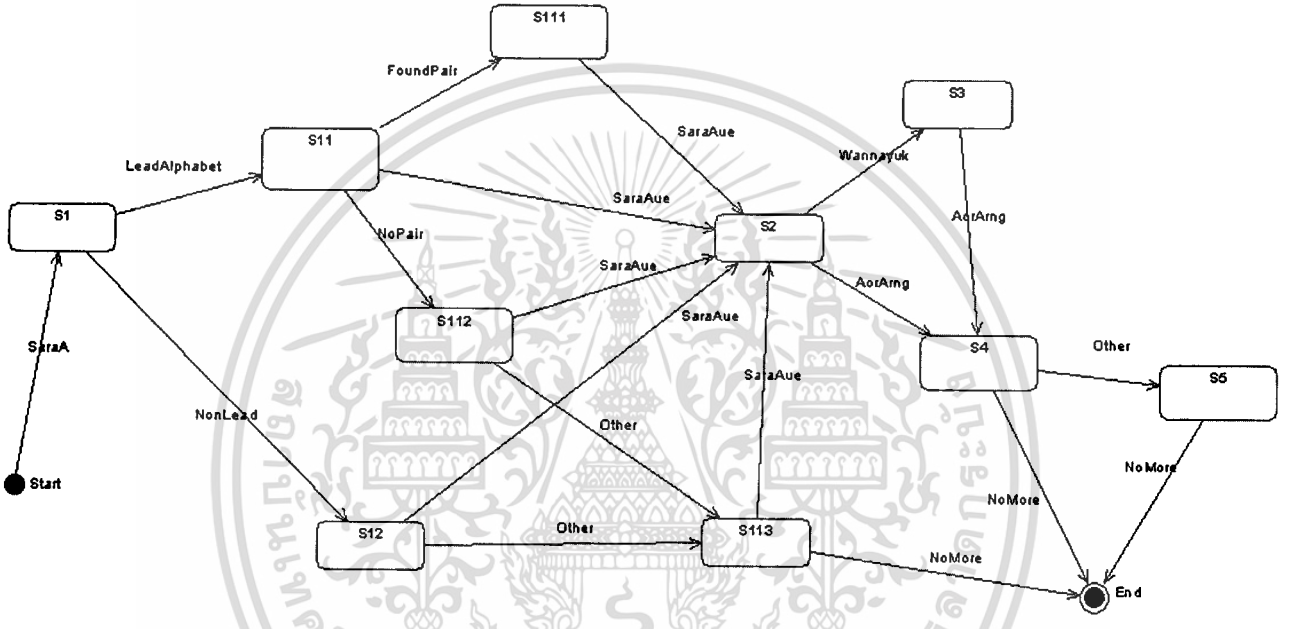
รูปที่ 4.10 โปรแกรมทดสอบที่ 1 เมื่ออยู่ในสถานะสูงสุด (Max)

4.4.2 โปรแกรมทดสอบที่ 2 จำลองโปรแกรมแปลงภาษาไทยให้เป็นอักษรเบลล์

โปรแกรมทดสอบที่ 2 นี้ จะทำการออกแบบจาก ส่วนหนึ่งของการแปลงภาษาไทยให้เป็นอักษรเบลล์ โดยเลือกเฉพาะส่วนของการแปลงสระ "เอือ" ซึ่งมีกลุ่มอักษรนำดังนี้

ก-ล, ข-ม|ย, ค-ร|ล, ถ-ล, ป-ร|ล, พ-ร, ม-ล, ส-ม, ห-ง|น|ม|ย|ล (Lalitrojwong and Mongkhon. 2003)

จากข้อมูลเบื้องต้น นำมาเขียนแผนภาพสถานะได้ดังนี้



รูปที่ 4.11 แผนภาพสถานะของโมดูลแปลงอักษรเบลล์สำหรับสระเอือ

เนื่องจากชื่อของสถานะ และการเปลี่ยนสถานะจะถูกนำไปสร้างเป็นชื่อฟังก์ชัน ดังนั้นจึงต้องเขียนแผนภาพสถานะในภาษาอังกฤษ และหลีกเลี่ยงอักขระพิเศษต่างๆ

จากรูปที่ 4.11 มีขั้นตอนการทำงานอธิบายได้ดังนี้

- เริ่มการทำงานจากการตรวจสอบคำภาษาไทยที่ถูกตัดคำมาเรียบร้อยแล้ว เมื่อพบสระเอ การทำงานจะเข้าสูสถานะ S1 เพื่อทำการตรวจหาสระ "เอือ" ต่อไป

- จากสถานะ S1 หากพบว่าตัวต่อไปอยู่ในกลุ่มอักษรนำตัวแรกของสระเอือจะไปสู่สถานะ S11 หากตัวต่อไปไม่ใช่อักษรนำ จะไปสู่สถานะ S12

- จากสถานะ S11 ถ้าอักขระตัวถัดไปเป็นตัวอักษรที่เป็นคู่กับอักษรนำจะไปสู่สถานะ S111 หากเป็นตัวอักษรที่ไม่เข้าคู่กับอักษรนำจะไปสู่สถานะ S112 และหากอักขระตัวถัดไปเป็นสระอือจะไปสู่สถานะ S2

- จากสถานะ S111 เมื่อพบอักขระตัวถัดไปว่าเป็นสระอือ จะเข้าสู่สถานะ S2

- จากถานะ S12 และ S112 หากพบว่าอักขระตัวถัดไปเป็นสระอือจะเข้าสู่สถานะ S2 แต่หากเป็นอักขระตัวอื่นซึ่งหมายถึงจะไม่เป็นไปตามกฎของสระเอือ การทำงานจะเข้าสู่สถานะ S113

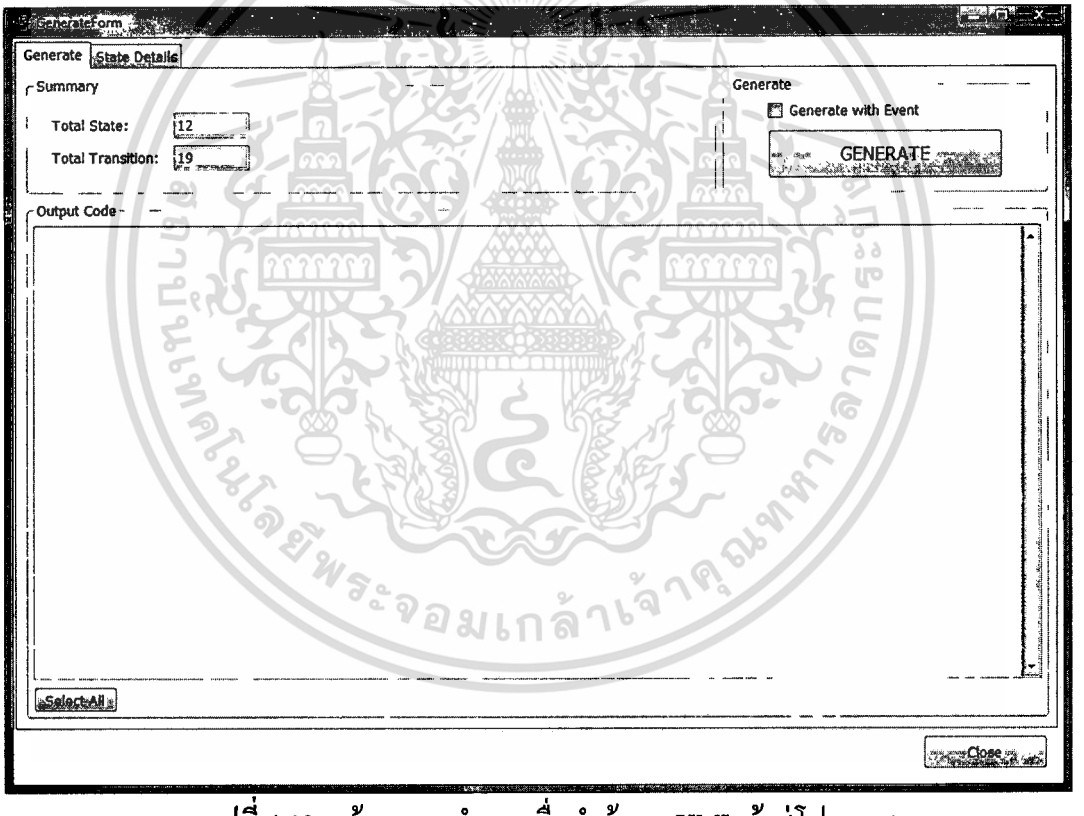
- จากสถานะ S113 หากพบว่าอักขระตัวถัดไปเป็นสระอือจะเข้าสู่สถานะ S2 หากไม่ใช่สระอือจะเข้าสู่สถานะจบการทำงาน

- จากสถานะ S2 หากพบว่าอักขระตัวถัดไปเป็นวรรณยุกต์จะเข้าสู่สถานะ S3 หรือถ้าอักขระตัวถัดไปเป็นอักษร "อ" จะเข้าสู่สถานะ S4

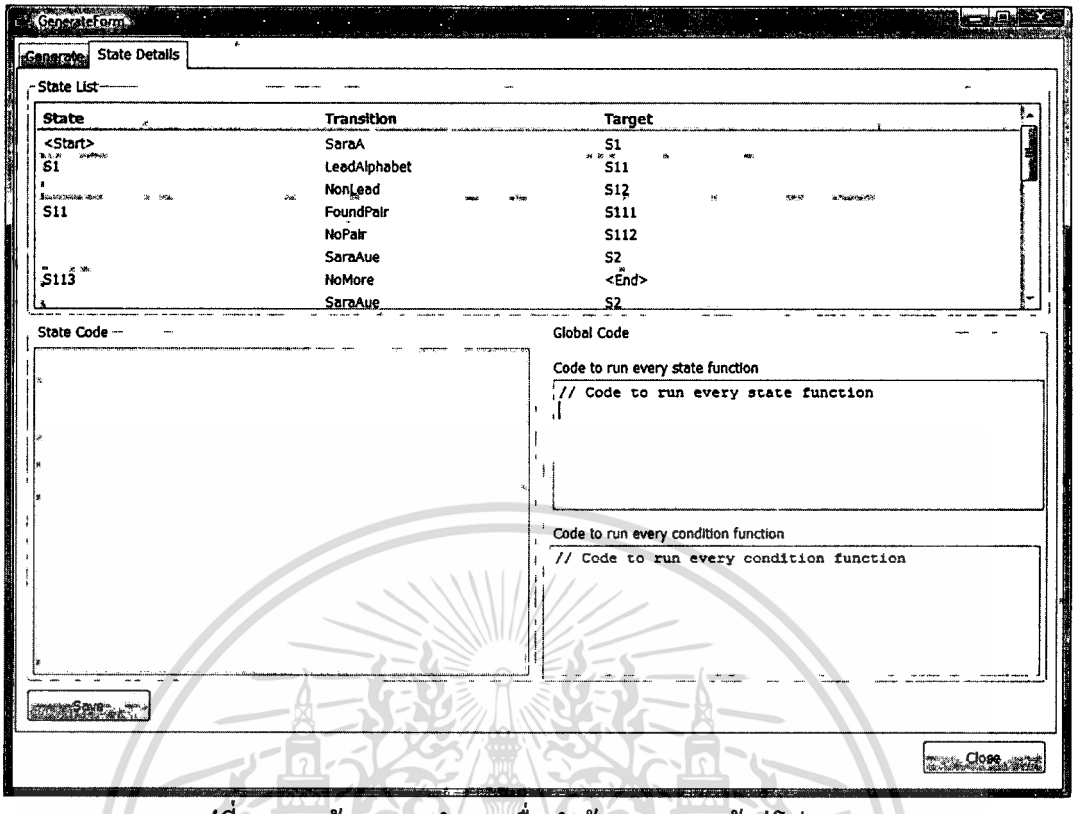
- จากสถานะ S3 หากอักขระตัวถัดไปเป็นอักษร "อ" จะเข้าสู่สถานะ S4

- จากสถานะ S4 หากพบว่ามีอักขระตัวถัดไปเป็นตัวอักษรจะเข้าสู่สถานะ S5 เพื่อพิมพ์ตัวสะกด และจบการทำงาน หรือหากไม่มีตัวอักขระถัดไปก็จะเข้าสู่สถานะจบการทำงานทันที

หลังจากสร้างไฟล์ XMI แล้ว นำเข้าสู่โปรแกรมสร้างรหัสคำสั่ง

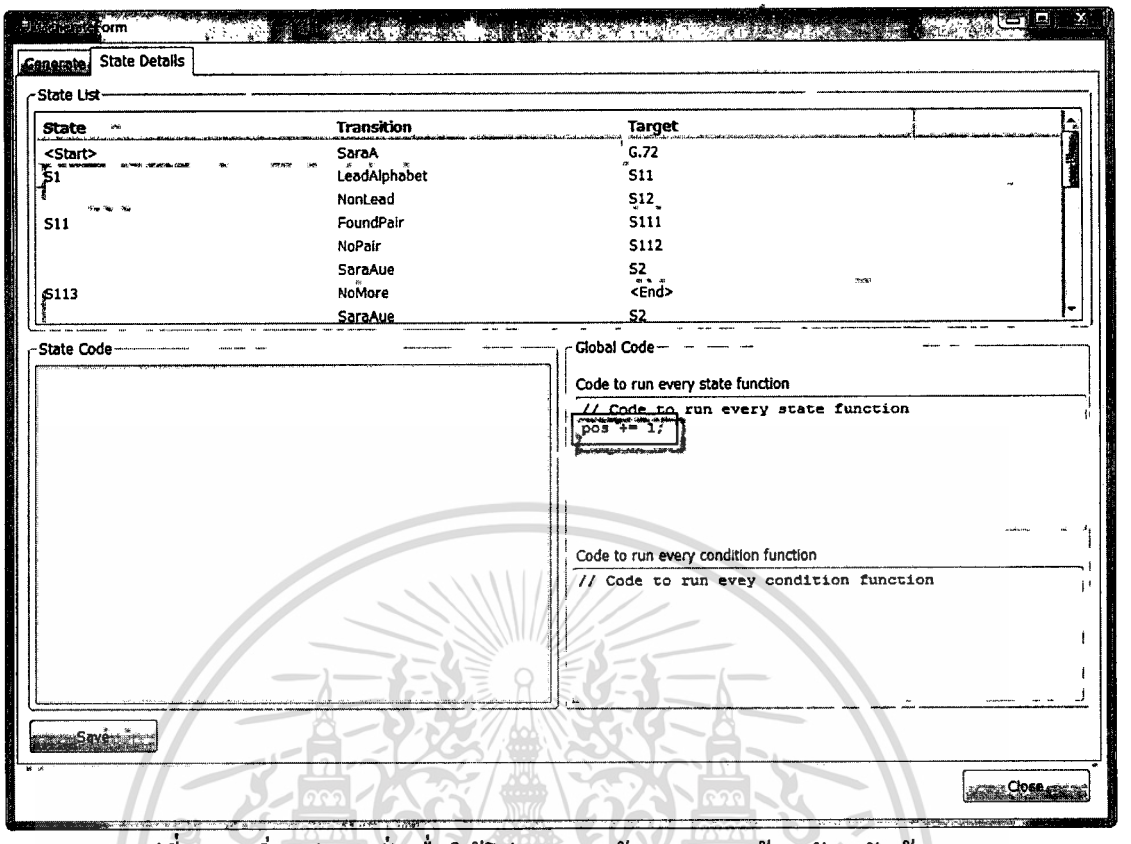


รูปที่ 4.12 หน้าจอการทำงานเมื่อนำข้อมูล XMI เข้าสู่โปรแกรม



รูปที่ 4.13 หน้าจอการทำงานเมื่อนำข้อมูล XMI เข้าสู่โปรแกรม

จากรูปที่ 4.12 จะเห็นได้ว่า โปรแกรมตรวจพบสถานะ 12 สถานะ และการเปลี่ยนสถานะ 19 เหตุการณ์ โดยจะทำการเพิ่มรหัสเบื้องต้นเพื่อใช้ในการทำงานของโปรแกรม



รูปที่ 4.14 เพิ่มรหัสคำสั่งเพื่อให้โปรแกรมสร้างออกมาพร้อมกับรหัสต้นแบบ

ทำการเพิ่มรหัสคำสั่ง เพื่อใช้ในการเลื่อนตำแหน่งของอักขระที่ตรวจสอบในการทำงานของแต่ละสถานะ

```
pos += 1;
```

หลังจากนั้นให้โปรแกรมทำการสร้างรหัสคำสั่ง จะได้รหัสคำสั่งดังนี้

```
// method for work of State Start
private void State_Start()
{

    // Code to run every state function
pos += 1;

    // One Transition Just Call Next Function
    State_S1();

}

// method for work of State End
private void State_End()
{

    // Code to run every state function
pos += 1;

    //End
```

ฟังก์ชันการทำงานของสถานะเริ่มต้น

ฟังก์ชันการทำงานของสถานะสุดท้าย

```
// method for work of State S1
private void State_S1()
{

    // Code to run every state function
pos += 1;


    // *** condition for S1
    switch (Cond_S1()) {
    case "LeadAlphabet":
        State_S11();
        break;
    case "NonLead":
        State_S12();
        break;

    }

}

// method for work of State S11
private void State_S11()
{

    // Code to run every state function
pos += 1;


    // *** condition for S11
    switch (Cond_S11()) {
    case "FoundPair":
        State_S111();
        break;
    case "NoPair":
        State_S112();
        break;
    case "SaraAue":
        State_S2();
        break;

    }

}

// method for work of State S113
private void State_S113()
{

    // Code to run every state function
pos += 1;


    // *** condition for S113
    switch (Cond_S113()) {
    case "NoMore":
        State_End();
        break;
    case "SaraAue":
        State_S2();
        break;

    }

}

// method for work of State S111
private void State_S111()
{

    // Code to run every state function
pos += 1;

    // One Transition Just Call Next Function
    State_S2();
}
```

ฟังก์ชันการทำงานของสถานะ S1

ฟังก์ชันการทำงานของสถานะ S11

ฟังก์ชันการทำงานของสถานะ S113

ฟังก์ชันการทำงานของสถานะ S111

```
// method for work of State S112
private void State_S112()
{

    // Code to run every state function
pos += 1;


    // *** condition for S112
    switch (Cond_S112()) {
    case "SaraAue":
        State_S2();
        break;
    case "Other":
        State_S113();
        break;

    }

}

// method for work of State S3
private void State_S3()
{

    // Code to run every state function
pos += 1;

    // One Transition Just Call Next Function
    State_S4();

}

// method for work of State S5
private void State_S5()
{

    // Code to run every state function
pos += 1;

    // One Transition Just Call Next Function
    State_End();

}

// method for work of State S4
private void State_S4()
{

    // Code to run every state function
pos += 1;


    // *** condition for S4
    switch (Cond_S4()) {
    case "Other":
        State_S5();
        break;
    case "NoMore":
        State_End();
        break;

    }

}

// method for work of State S2
private void State_S2()
{

    // Code to run every state function
pos += 1;


    // *** condition for S2
    switch (Cond_S2()) {
```

ฟังก์ชันการทำงานของสถานะ S112

ฟังก์ชันการทำงานของสถานะ S3

ฟังก์ชันการทำงานของสถานะ S5

ฟังก์ชันการทำงานของสถานะ S4

ฟังก์ชันการทำงานของสถานะ S2

```
    case "Wannayuk":
        State_S3();
        break;
    case "AorArng":
        State_S4();
        break;

    }
}

// method for work of State S12
private void State_S12()
{

    // Code to run every state function
pos += 1;


    // *** condition for S12
    switch (Cond_S12()) {
    case "Other":
        State_S113();
        break;
    case "SaraAue":
        State_S2();
        break;

    }
}

// ********************************************************************
// ********************  Condition Function NEED TO IMPLEMENT  ********
// ********************************************************************

private string Cond_S1() {
    string returnVal = "";

    // Code to run every condition function

    // ***** Code to work for return result for condition in state S1

    // returnVal = "LeadAlphabet";
    // returnVal = "NonLead";

    return returnVal;
}

private string Cond_S11() {
    string returnVal = "";

    // Code to run every condition function

    // ***** Code to work for return result for condition in state S11

    // returnVal = "FoundPair";
    // returnVal = "NoPair";
    // returnVal = "SaraAue";

    return returnVal;
}

private string Cond_S113() {
    string returnVal = "";

    // Code to run every condition function

    // ***** Code to work for return result for condition in state S113

    // returnVal = "NoMore";
    // returnVal = "SaraAue";

    return returnVal;
}

private string Cond_S112() {
    string returnVal = "";
```

ฟังก์ชันการทำงานของสถานะ S12

ฟังก์ชันการทำงานเพื่อกำหนดเงื่อนไขการทำงานถัดไปของสถานะ S1

ฟังก์ชันการทำงานเพื่อกำหนดเงื่อนไขการทำงานถัดไปของสถานะ S11

ฟังก์ชันการทำงานเพื่อกำหนดเงื่อนไขการทำงานถัดไปของสถานะ S113

ฟังก์ชันการทำงานเพื่อกำหนดเงื่อนไขการทำงานถัดไปของสถานะ S112

```
    // Code to run every condition function

    // ***** Code to work for return result for condition in state S112

    // returnVal = "SaraAue";
    // returnVal = "Other";

    return returnVal;
}

private string Cond_S4() {
    string returnVal = "";

    // Code to run every condition function

    // ***** Code to work for return result for condition in state S4

    // returnVal = "Other";
    // returnVal = "NoMore";

    return returnVal;
}

private string Cond_S2() {
    string returnVal = "";

    // Code to run every condition function

    // ***** Code to work for return result for condition in state S2

    // returnVal = "Wannayuk";
    // returnVal = "AorArng";

    return returnVal;
}

private string Cond_S12() {
    string returnVal = "";

    // Code to run every condition function

    // ***** Code to work for return result for condition in state S12

    // returnVal = "Other";
    // returnVal = "SaraAue";

    return returnVal;
}
```

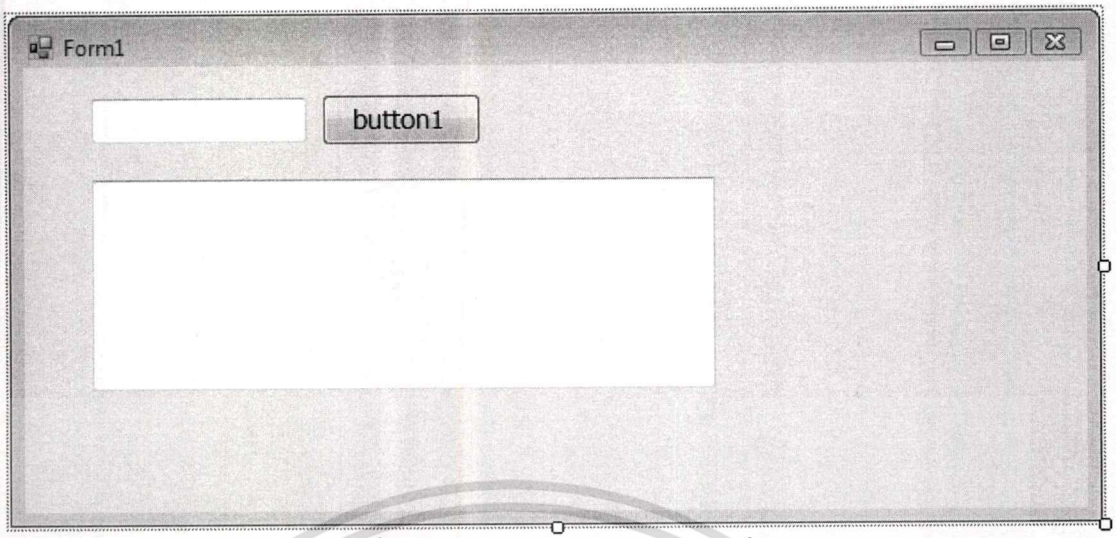
ฟังก์ชันการทำงานเพื่อกำหนดเงื่อนไขการทำงานถัดไปของสถานะ S4

ฟังก์ชันการทำงานเพื่อกำหนดเงื่อนไขการทำงานถัดไปของสถานะ S2

ฟังก์ชันการทำงานเพื่อกำหนดเงื่อนไขการทำงานถัดไปของสถานะ S12

รหัสคำสั่งที่สร้างโดยโปรแกรม

จะเห็นได้ว่า รหัสคำสั่งที่ผู้ใช้เพิ่มไปจะถูกสร้างออกมาพร้อมกับรหัสทั้งหมด ขั้นตอนต่อไปทำการสร้าง โปรแกรมสำหรับทดสอบขึ้น



รูปที่ 4.15 หน้าจอโปรแกรมทดสอบที่ 2

จากนั้นทำการพัฒนาโปรแกรมให้สมบูรณ์

```
public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            textBox2.Text = "";
            pos = -1;
            wan = "";
            aue = true;
            State_Start();
        }

        int pos = 0;
        string wan = "";
        bool aue = true;


        // method for work of State Start
        private void State_Start()
        {

            // Code to run every state function
            pos += 1;
            // One Transition Just Call Next Function
            State_S1();

        }

        // method for work of State End
        private void State_End()
        {

            // Code to run every state function
            pos += 1;
            //End

        }

        // method for work of State S1
        private void State_S1()
        {
```

ฟังก์ชันการทำงานของปุ่ม Button1 มีการกำหนดค่าเบื้องต้นให้กับตัวแปรที่ใช้

ตัวแปรที่ใช้ในการทำงาน

```
        // Code to run every state function
        pos += 1;

        // *** condition for S1
        switch (Cond_S1())
        {
            case "LeadAlphabet":
                State_S11();
                break;
            case "NonLead":
                State_S12();
                break;

        }

    }

    // method for work of State S11
    private void State_S11()
    {

        // Code to run every state function
        pos += 1;

        // *** condition for S11
        switch (Cond_S11())
        {
            case "FoundPair":
                State_S111();
                break;
            case "NoPair":
                State_S112();
                break;
            case "SaraAue":
                State_S2();
                break;

        }

    }
    // method for work of State S113
    private void State_S113()
    {

        // Code to run every state function
        pos += 1;
        aue = false;
        for (int i = 0; i < pos - 2; i++)
        {
            textBox2.Text += textBox1.Text[i].ToString() + "|";
        }
        // *** condition for S113
        switch (Cond_S113())
        {
            case "NoMore":
                State_End();
                break;
            case "SaraAue":
                State_S2();
                break;

        }

    }

    // method for work of State S111
    private void State_S111()
    {

        // Code to run every state function
        pos += 1;
        // One Transition Just Call Next Function
        State_S2();

    }
```

เพิ่มการทำงานในสถานะ S112 ระบุว่าหากการทำงานมาสู่สถานะนี้แสดงว่าไม่ใช่สระ “เอือ” และทำการพิมพ์อักขระก่อนหน้าตำแหน่งปัจจุบันลงในกล่องผลลัพธ์

```
// method for work of State S112
private void State_S112()
{

    // Code to run every state function
    pos += 1;

    string t = textBox1.Text;
    textBox2.Text += t[0] + "|";
    aue = false;

    // *** condition for S112
    switch (Cond_S112())
    {
        case "SaraAue":
            State_S2();
            break;
        case "Other":
            State_S113();
            break;

    }

}

// method for work of State S3
private void State_S3()
{

    // Code to run every state function
    pos += 1;

    wan = "ว" + textBox1.Text[pos - 1].ToString() + "|";
    // One Transition Just Call Next Function
    State_S4();

}

// method for work of State S5
private void State_S5()
{

    // Code to run every state function
    pos += 1;

    textBox2.Text += textBox1.Text[pos - 1].ToString() + "|";
    // One Transition Just Call Next Function
    State_End();

}

// method for work of State S4
private void State_S4()
{

    // Code to run every state function
    pos += 1;

    textBox2.Text += (aue ? "เอือ|" : "อือ|");
    textBox2.Text += wan;
    // *** condition for S4
    switch (Cond_S4())
    {
        case "Other":
            State_S5();
            break;
        case "NoMore":
            State_End();
            break;

    }

}

// method for work of State S2
private void State_S2()
```

เพิ่มการทำงานในสถานะ S112 ระบุว่าหากการทำงานมาสู่สถานะนี้แสดงว่าไม่ใช่สระ “เอือ” และทำการพิมพ์สระ "เอ” ลงในกล่องผลลัพธ์

การทำงานในสถานะ S3 กำหนดตัวแปร wan เป็นวรรณยุกต์ที่พบ

การทำงานในสถานะ S5 ทำการพิมพ์ตัวสะกดสู่กล่องผลลัพธ์

การทำงานในสถานะ S4 ในสถานะนี้จะตรวจสอบว่าข้อความที่ผ่านมาเป็น สระ “อือ” หรือ สระ “เอือ” และทำการพิมพ์ลงกล่องผลลัพธ์ ตามด้วยวรรณยุกต์ (ถ้ามี)

```
{
    // Code to run every state function
    pos += 1;

    string t = textBox1.Text;

    for (int i = 1; i < pos - 1; i++)
        textBox2.Text += t[i].ToString() + "|";
    // *** condition for S2
    switch (Cond_S2())
    {
        case "Wannayuk":
            State_S3();
            break;
        case "AorArng":
            State_S4();
            break;

    }

}

// method for work of State S12
private void State_S12()
{

    // Code to run every state function
    pos += 1;

    // *** condition for S12
    switch (Cond_S12())
    {
        case "Other":
            State_S113();
            break;
        case "SaraAue":
            State_S2();
            break;

    }
}


// ************************************************************
// ***** Condition Function NEED TO IMPLEMENT
// ************************************************************

private string Cond_S1()
{
    string returnVal = "";
    // ***** Code to work for return result for condition in state S1

    switch (textBox1.Text[pos])
    {
        case 'ก':
        case 'ด':
        case 'ม':
        case 'ค':
        case 'ป':
        case 'ข':
        case 'พ':
        case 'ส':
        case 'ห':
            returnVal = "LeadAlphabet";
            break;
        default:
            returnVal = "NonLead";
            break;
    }
```

การทำงานในสถานะ S2
ทำการพิมพ์อักขระก่อนหน้าทั้งหมด
ยกเว้นสระ “เอ”

ฟังก์ชันเงื่อนไขตรวจสอบว่าเป็นกลุ่ม
อักษรนำสำหรับสระ “เอือ” หรือไม่

```
            return returnVal;
        }

        private string Cond_S11()
        {
            string returnVal = "";
            // ***** Code to work for return result for condition in state S11

            string t = textBox1.Text;

            if (t[pos] == 'อ')
            {
                returnVal = "SaraAue";
            }
            else
            {

                switch (t[pos - 1])
                {
                    case 'ก':
                    case 'ถ':
                    case 'ฝ':
                        if (t[pos] == 'ถ') returnVal = "FoundPair";
                        else returnVal = "NoPair";
                        break;
                    case 'ค':
                    case 'ป':
                        if (t[pos] == 'ร' || t[pos] == 'ล') returnVal = "FoundPair";
                        else returnVal = "NoPair";
                        break;
                    case 'ง':
                        if (t[pos] == 'ง' || t[pos] == 'ย') returnVal = "FoundPair";
                        else returnVal = "NoPair";
                        break;
                    case 'จ':
                        if (t[pos] == 'ว') returnVal = "FoundPair";
                        else returnVal = "NoPair";
                        break;
                    case 'ช':
                        if (t[pos] == 'ม') returnVal = "FoundPair";
                        else returnVal = "NoPair";
                        break;
                    case 'ท':
                        if (t[pos] == 'ร' || t[pos] == 'น' || t[pos] == 'ม' || t[pos]
== 'ย' || t[pos] == 'ล') returnVal = "FoundPair";
                        else returnVal = "NoPair";
                        break;
                }
            }
            // returnVal = "FoundPair";
            // returnVal = "NoPair";

            return returnVal;
        }

        private string Cond_S12()
        {
            string returnVal = "";
            // ***** Code to work for return result for condition in state S12

            if (textBox1.Text[pos] == 'อ')
                returnVal = "SaraAue";
            else
                returnVal = "Other";

            return returnVal;
        }

        private string Cond_S112()
        {
            string returnVal = "";
            // ***** Code to work for return result for condition in state S112
```

ฟังก์ชันเงื่อนไขตรวจสอบว่าเป็นสระ "อือ" หรือไม่

```
        if (textBox1.Text[pos] == 'ื')
            returnVal = "SaraAue";
        else
            returnVal = "Other";

        return returnVal;
    }

    private string Cond_S113()
    {
        string returnVal = "";
        // ***** Code to work for return result for condition in state S113

        string t = textBox1.Text;

        if (t[pos] == 'ื')
            returnVal = "SaraAue";
        else
            returnVal = "NoMore";

        return returnVal;
    }

    private string Cond_S4()
    {
        string returnVal = "";
        // ***** Code to work for return result for condition in state S4

        if (textBox1.Text.Length <= pos)
            returnVal = "NoMore";
        else
            returnVal = "Other";

        return returnVal;
    }

    private string Cond_S2()
    {
        string returnVal = "";
        // ***** Code to work for return result for condition in state S2

        string t = textBox1.Text;
        switch (Convert.ToInt32(t[pos]))
        {
            case 3656:
            case 3657:
            case 3658:
            case 3659:
                returnVal = "Wannayuk";
                break;
            default:
                returnVal = "AorArng";
                break;
        }

        return returnVal;
    }
}
```

ฟังก์ชันเงื่อนไขตรวจสอบว่าเป็นสระ “อือ” หรือไม่

ฟังก์ชันเงื่อนไขตรวจสอบว่าเป็นสระ “อือ” หรือไม่

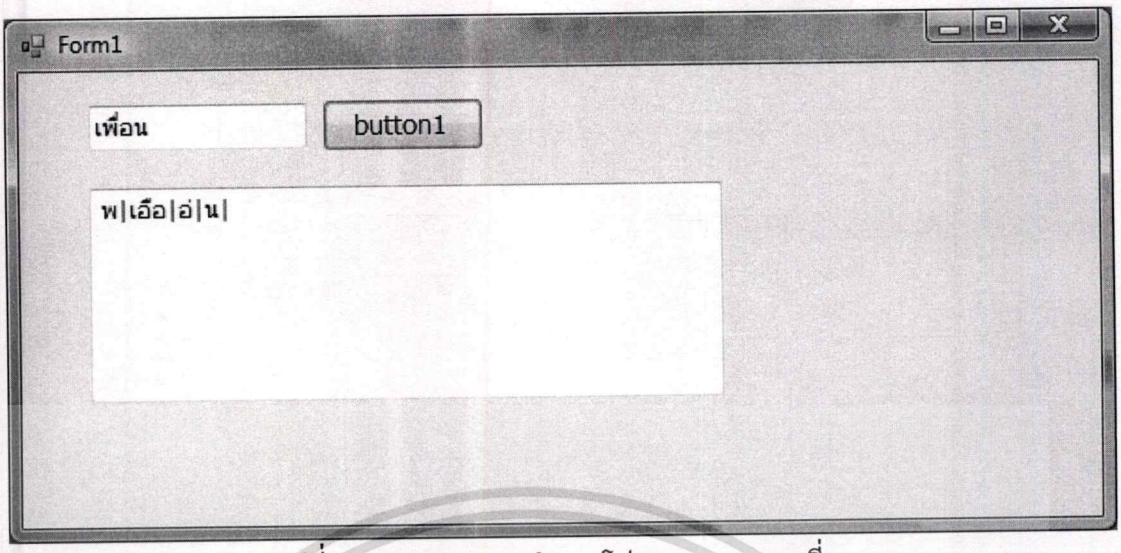
ฟังก์ชันเงื่อนไขตรวจสอบว่า มีตัวสะกดหรือไม่

ฟังก์ชันแงื่อนไขตรวจสอบว่าเป็นวรรณยุกต์หรือเป็นตัวอักษร “อ”

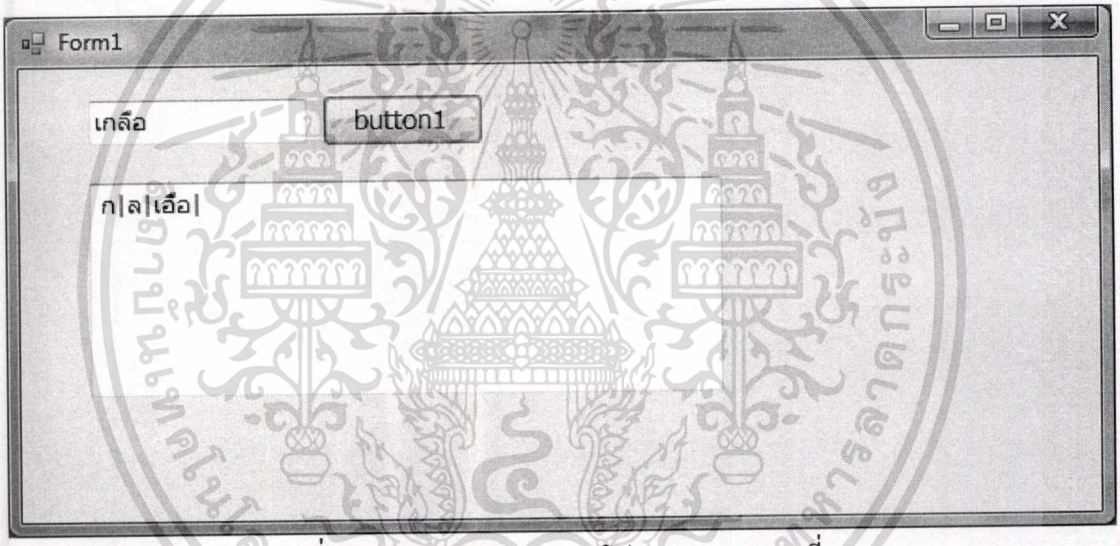
รหัสคำสั่งที่ทำการพัฒนาการทำงานทั้งหมดจากรหัสเบื้องต้นเรียบร้อยแล้ว

รหัสที่ทำแถบสี คือรหัสที่เพิ่มจากรหัสต้นแบบที่ถูกสร้างโดยโปรแกรม

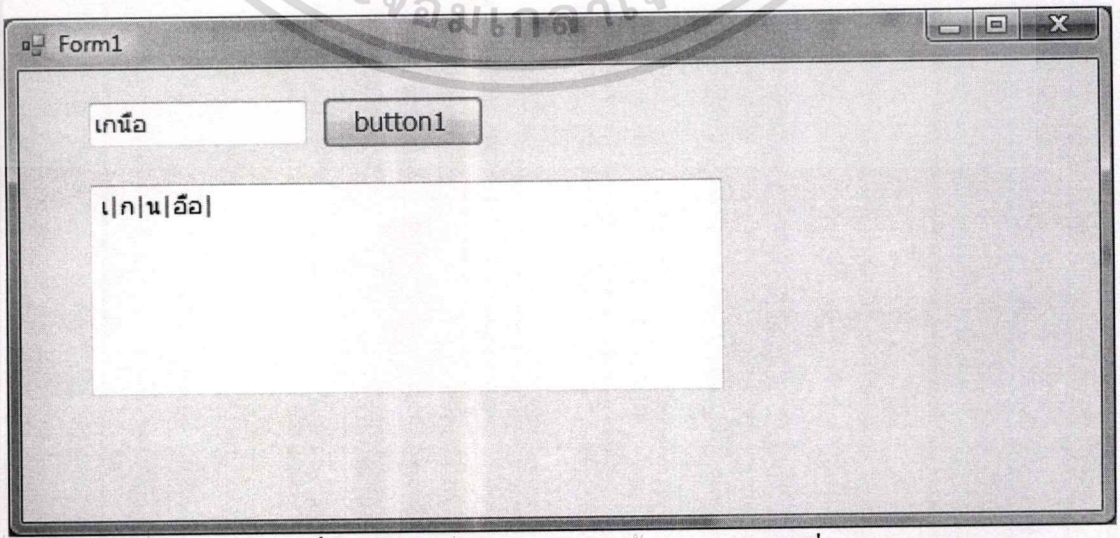
จากการทดสอบ การทำงานเป็นไปตามที่ออกแบบแผนภาพสถานะไว้ ดังรูป



รูปที่ 4.16 ทดสอบการทำงานโปรแกรมทดสอบที่ 2



รูปที่ 4.17 ทดสอบการทำงานโปรแกรมทดสอบที่ 2



รูปที่ 4.18 ทดสอบการทำงานโปรแกรมทดสอบที่ 2

บทที่ 5
บทสรุปและข้อเสนอแนะ

ในบทนี้เป็นการสรุปผลภาพโดยรวมของการพัฒนาระบบ ประโยชน์ที่ได้รับจากการพัฒนาระบบ ปัญหาและอุปสรรคระหว่างการออกแบบ ข้อจำกัด ข้อเสนอแนะและแนวทางในการพัฒนาระบบในอนาคต

5.1 สรุปผลโครงการ

โครงการพัฒนาโปรแกรมสร้างรหัสคำสั่งจากแผนภาพสถานะถูกพัฒนาขึ้นจากแนวคิดที่ต้องการลดขั้นตอนการพัฒนาโปรแกรมที่สามารถออกแบบในรูปแบบของแผนภาพสถานะ รวมถึงการลดความผิดพลาดที่อาจเกิดขึ้นจากการออกแบบที่ซับซ้อน และเพื่อให้ได้รหัสคำสั่งที่เป็นมาตรฐานและทำงานได้ถูกต้อง จากผลการทดสอบโปรแกรม สามารถลดขั้นตอนการเขียนรหัสคำสั่งได้ในระดับหนึ่ง และรหัสคำสั่งยังถูกสร้างมาในโครงสร้างที่เป็นมาตรฐาน ช่วยให้การพัฒนาโปรแกรมต่อยอดเป็นไปอย่างมีประสิทธิภาพมากขึ้น

5.2 ประโยชน์ที่ได้รับจากการพัฒนาระบบ

1. มีความเข้าใจการออกแบบการทำงานของระบบด้วยแผนภาพสถานะ
2. มีเครื่องมือต้นแบบที่ช่วยในการพัฒนาโปรแกรม และสามารถพัฒนาต่อยอดเพื่อเพิ่มคุณสมบัติต่างๆ ได้ในอนาคต
3. ลดความผิดพลาดที่อาจเกิดขึ้นจากการเขียนรหัสคำสั่งจากการออกแบบที่ซับซ้อน

5.3 ปัญหาและอุปสรรคระหว่างการออกแบบและพัฒนาระบบ

1. เนื่องจากผู้เขียนพึ่งศึกษาการเขียนโปรแกรมด้วยภาษา Java จึงมีปัญหาในเรื่องของไวยากรณ์ของภาษาอยู่บ้าง
2. โปรแกรมที่พัฒนาเป็นเครื่องมือที่ใช้สำหรับผู้พัฒนาซอฟท์แวร์ ทำให้ผู้ใช้งานโปรแกรมนี้จะต้องมีความรู้ความเข้าใจในการพัฒนาซอฟท์แวร์เบื้องต้นแล้ว

5.4 ข้อจำกัดของระบบ

1. เนื่องจากเป็นโปรแกรมต้นแบบ รูปแบบของไฟล์ที่นำเข้าสามารถใช้ได้กับไฟล์ XMI ที่ส่งออกจากโปรแกรม Rational Rose รุ่น 8.0 เท่านั้น

2. การสร้างรหัสคำสั่งสามารถสร้างได้เพียงภาษาเดียว

3. เนื่องจากชื่อของสถานะ และการเปลี่ยนสถานะจะถูกนำมาสร้างเป็นรหัสคำสั่ง ดังนั้นจึงควรหลีกเลี่ยงอักขระพิเศษ และภาษาไทย ซึ่งอาจทำให้ความง่ายในการเข้าใจแผนภาพสถานะ ลดลง

5.5 ข้อเสนอแนะและแนวทางในการพัฒนาระบบ

1. เพิ่มการรองรับรูปแบบไฟล์ XMI จากโปรแกรมที่หลากหลายมากขึ้น
2. เพิ่มรูปแบบของภาษาของรหัสคำสั่งที่สามารถสร้างได้
3. พัฒนาลักษณะการทำงานให้มีการใช้งานที่ง่ายขึ้น
4. พัฒนาความสามารถของโปรแกรมให้มีคุณสมบัติต่างๆ เช่นบันทึกงานที่ทำไว้ได้
5. อาจศึกษาโครงสร้างและการสร้างรหัสคำสั่งจากแผนภาพอื่นๆที่ใช้ในการออกแบบ

บรรณานุกรม

ชาลี วรกุลพิพัฒน์ และ เทพฤทธิ์ บัณฑิตวัฒนาวงศ์. 2544. **UML ภาษามาตรฐานเพื่อผู้พัฒนาซอฟต์แวร์.** กรุงเทพฯ: ซีเอ็ดยูเคชั่น.

ไพศาล โมลิสกุลมงคล. 2545. **Microsoft Visual C#.net.** กรุงเทพฯ: ดวงกมลสมัย.

วีระศักดิ์ ซึงถาวร. 2546. **JAVA Programming volume I.** กรุงเทพฯ: ซีเอ็ดยูเคชั่น.

สราวุธ อ้อยศรีสกุล. 2544. **ถอดรหัส .net + Web Services.** กรุงเทพฯ: วิตตี้ กรุ๊ป.

สุรพรรษ์ เพ็ญจำรัส. 2546. **เรียนลัด C# และการเขียนโปรแกรม .NET.** กรุงเทพฯ: โปรวิชั่น

Brownlee, J. 2008. **Finite State Machine**. [Online]. Available: http://ai-depot.com/FiniteStateMachines/ .

Martin, F and Kendall, S. 2000. **UML Distilled**. Addison-Wesley.

Object Management Group. 2005. **MOF 2.0/XMI Mapping Version 2.1.1**. [Online]. Available: http://www.omg.org/spec/XMI/2.1/PDF.

Lalitrojwong, L and Triwatthanachaikun, M. 2003. **Thai to Braille Translation.** Proceedings of the Third International Symposium on Communications and Information Technologies (ISCIT 2003), Songkhla, Thailand, pp. 386-390.

State Diagram. 2006. [Online]. Available: http://en.wikipedia.org/wiki/State_diagram.

ประวัติผู้เขียน

ชื่อผู้เขียน นายนัฐวุธ พุ่มพวง

วัน-เดือน-ปีเกิด 2 พฤษภาคม 2523

สถานที่เกิด กรุงเทพมหานคร

ประวัติการศึกษา

ปริญญาตรี	วิทยาศาสตร์ สาขาสถิติประยุกต์
สถานที่สำเร็จการศึกษา	สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีที่สำเร็จการศึกษา	2545