รหัสโครงการ SUT7-705-48-12-68

รายงานการวิจัย

การออกแบบและพัฒนาเฟรมเวิร์คการเชื่อมต่อข้อมูลสำหรับลูกข่ายแบบบางของระบบกลุ่มแม่ข่ายโปรแกรมประยุกต์

Design and Development of the Data Connection Framework for the Application Server Cluster's Thin Client

คณะผู้วิจัย

หัวหน้าโครงการ

อาจารย์ ดร. ชาญวิทย์ แก้วกสิ

สาขาวิชาวิศวกรรมคอมพิวเตอร์

สำนักวิชาวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีสุรนารี

ได้รับทุนอุดหนุนการวิจัยจากมหาวิทยาลัยเทคโนโลยีสุรนารี ปีงบประมาณ พ.ศ. 2548

ผลงานวิจัยเป็นความรับผิดชอบของหัวหน้าโครงการวิจัยแต่เพียงผู้เดียว

กิตติกรรมประกาศ

งานวิจัยนี้สำเร็จลุล่วงด้วยดีเนื่องจากได้รับทุนอุดหนุนการวิจัย จากมหาวิทยาลัยเทคโนโลยีสุรนารี ประจำปีงบประมาณ 2548 ขอบคุณผู้ช่วยวิจัยทุกท่านที่ช่วยรวบรวมข้อมูล ศึกษาและทดลอง รวมทั้งช่วยดูแลงานด้านเอกสารทำให้งานวิจัยนี้สำเร็จลุล่วงไปได้ด้วยดี และขอขอบคุณคุณณิชชาภัทร สิทธิคุณ ที่ช่วยดูแลประสานงานและติดตามงานวิจัยมาโดยตลอด สุดท้ายขอขอบคุณอาจารย์และบุคคลากรสาขาวิศวกรรมคอมพิวเตอร์ทุกท่านที่ให้ความช่วยเหลือเป็นอย่างดี

ชาญวิทย์ แก้วกสิ

บทคัดย่อ

งานวิจัยนี้ได้ศึกษาและนำเอาเทคโนโลยีการโยงข้อมูลมาพัฒนาเฟรมเวิร์คการเชื่อมต่อข้อมูลสำหรับลูกข่ายแบบบางของระบบกลุ่มแม่ข่ายโปรแกรมประยุกต์ที่มีชื่อว่า Galvanium Framework เพื่อแก้ปัญหาความเข้ากันไม่ได้ระหว่างวัตถุบนแพล็ตฟอร์ม .NET และแพล็ตฟอร์ม Java เฟรมเวิร์คนี้ประกอบไปด้วยตัวเชื่อมต่อระหว่างลูกข่ายแบบบางบนแพล็ตฟอร์ม .NET กับระบบกลุ่มแม่ข่ายโปรแกรมประยุกต์ที่อยู่บนแพล็ตฟอร์ม Java ผ่านโปรโตคอล Hessian และตัวจำลองคุณสมบัติวัตถุสำหรับการโยงข้อมูลของ Java Bean ที่แปลงเป็นวัตถุ .NET โดยสนับสนุนทั้ง Data Object และ Business Object

จากการทดลองสร้างโปรแกรมลูกข่ายพบว่าสามารถสร้างลูกข่ายที่เชื่อมต่อกับโปรแกรมแม่-ข่าย ซึ่งเป็นระบบที่พัฒนาด้วย EJB ผ่านโปรโตคอล Hessian และสามารถทำงานได้อย่างปกติ โดยพบว่าเมื่อจำนวนข้อมูล 10,000 รายการ การแสดงผลของทั้งสองระบบใช้เวลาไม่แตกต่างกันอย่างมีนัยสำคัญ และเมื่อเพิ่มจำนวนวัตถุมากขึ้น การแสดงผลวัตถุด้วย Galvanium Framework จะช้ากว่าวัตถุปกติของ .NET แต่จะมีความเร็วใกล้เคียงกันอีกครั้งเมื่อจำนวนข้อมูล 50,000 รายการ อย่างไรก็ตาม Galvanium Framework ยังมีข้อจำกัด คือ ไม่สามารถทำงานกับคุณสมบัติของวัตถุที่ซ้อนกันหลายชั้นได้

Abstract

The work described in this report studied and applied data binding technology to develop a framework, Galvanium Framework, for interoperating .NET thin clients with Java application servers. This framework is designed to solve object incompatibility problems between .NET and Java platforms. This framework consists of a connector for .NET thin clients to Java application servers, and property emulators that can perform data-binding over Java Beans in the form of .NET objects. The property emulators work with both data objects and business objects.

From the experiment, it showed that a thin client developed using Galvanium Framework can work normally with an EJB system through the Hessian protocol. The experiment also showed that when displaying 10,000 entities, the performance of both systems, the normal .NET client and the client developed using Galvanium Framework, are not different significantly. However, when the number of displaying objects are increased, the Galvanium Framework client is slower than the normal .NET client. Performance of both clients converges to about the same speed when the number of displaying objects are 50,000. The current limitation of Galvanium Framework is that its property emulators have not supported nested properties.

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

สารบัญภาพ

บทที่ 1
บทนำ

1. ความสำคัญ ที่มาของปัญหาที่ทำการวิจัย

ในปัจจุบันการทำงานผ่านโปรแกรมลูกข่ายซึ่งเป็น Web Browser และการพัฒนาระบบงานในลักษณะที่เป็นโปรแกรมประยุกต์แบบเว็บ (Web Application) ได้รับความนิยมอย่างกว้างขวาง การพัฒนาระบบซึ่งแยกตัวแสดงผล ตัวประมวลผลเงื่อนไข และตัวจัดการข้อมูลออกจากกัน ที่เรียกว่า Multi-Tier นั้น จึงทำให้เกิดแนวคิดของโปรแกรมประเภทลูกข่ายแบบบาง (Thin Client) ขึ้น ซึ่ง Web Browser ก็สามารถพิจารณาได้ว่าเป็นลูกข่ายแบบบางของโปรแกรมประยุกต์แบบเว็บ อย่างไรก็ตาม การออกแบบระบบที่มีส่วนติดต่อกับผู้ใช้ (User Interface) ที่ซับซ้อนเพื่ออำนวยความสะดวกต่อผู้ใช้ระบบให้ได้มากที่สุดนั้น ยังไม่สามารถทำได้ดีด้วย Web Browser เมื่อเทียบกับโปรแกรมที่พัฒนาเป็นโปรแกรมประยุกต์ประเภทตั้งโต๊ะ (Desktop Application) และเพื่อที่จะให้ระบบงานสามารถรองรับผู้ใช้จำนวนมากได้นั้น การสร้างตัวเชื่อมข้อมูลจากลูกข่ายไปยัง กลุ่มเครื่องแม่ข่าย (Server Cluster) นั้น จึงเป็นสิ่งจำเป็น โดยผลลัพธ์ที่ได้คือความสามารถในการพัฒนาระบบงานซึ่งมีโปรแกรมลูกข่ายที่จะถูกออกแบบส่วนติดต่อกับผู้ใช้ให้ดีที่สุด และในขณะเดียวกันกับที่ระบบงานสามารถใช้ประโยชน์จากกลุ่มเครื่องแม่ข่าย เพื่อรองรับจำนวนผู้ใช้ให้ได้มากที่สุด

เฟรมเวิร์คในปัจจุบันที่ใช้เชื่อมต่อข้อมูลกับซอฟต์แวร์ประเภทแม่ข่ายโปรแกรมประยุกต์ของภาษา Java (Java Application Server) [1] นั้นมีอยู่แล้วสำหรับลูกข่ายที่พัฒนาด้วยโปรแกรมภาษา Java โดยใช้ Java Foundation Class และ Swing Component ในการพัฒนาโปรแกรมลูกข่าย แต่ปัญหาที่เกิดขึ้นก็คือ โปรแกรมที่พัฒนาในลักษณะ Graphical User Interface (GUI) ด้วยภาษา Java นั้น ต้องการทรัพยากรของเครื่องคอมพิวเตอร์ค่อนข้างมาก แม้ในปัจจุบันความเร็วของตัวประมวลผลของคอมพิวเตอร์จะสูงขึ้นมากก็ตาม เมื่อกล่าวถึงการพัฒนาโปรแกรมในลักษณะ GUI เพื่อใช้งานทั่วไป เครื่องมือและสภาวะแวดล้อมสำหรับการพัฒนาเช่น Microsoft Visual Basic, Borland Delphi และ Microsoft .NET Framework จะเป็นตัวเลือกที่นักพัฒนาระบบนึกถึงก่อน Java เสมอ แม้โปรแกรมที่พัฒนาด้วยเครื่องมือกลุ่มดังกล่าวไม่สามารถเชื่อมต่อกับ ซอฟต์แวร์ประเภทแม่ข่ายโปรแกรมประยุกต์สำหรับภาษา Java ได้โดยตรง แต่ก็สามรถทำผ่านตัวกลางอย่างเช่น Object Request Broker (ORB), Web Service หรือผ่านข้อตกลงเฉพาะ (Protocol) ที่พัฒนาขึ้นเองได้ อย่างไรก็ตาม นักพัฒนาก็อาจจะเสียเวลาในการพัฒนาเพื่อที่จะนำข้อมูลที่ดึงผ่านตัวกลางดังกล่าวไปแสดงผลผ่านส่วนประกอบสำหรับติดต่อผู้ใช้ (User Interface Component) ด้วยตนเอง วิธีการแสดงผลข้อมูลจาก

ระบบฐานข้อมูลหรือวัตถุข้อมูลที่นิยมใช้กันอยู่ในปัจจุบันคือ กลไกที่เรียกว่าการโยงข้อมูล (Data Binding) โดยเทคโนโลยีการโยงข้อมูลที่ทันสมัยที่สุดในปัจจุบันพบได้ใน Microsoft .NET Framework [2] ถึงแม้ว่าจะยังไม่มีรายการการวัดความยากง่ายของการใช้การโยงข้อมูลอย่างเป็นรูปธรรมแต่ด้วยการยอมรับจากกลุ่มผู้พัฒนาจำนวนมาก รวมทั้งมีแนวทางการพัฒนาระบบที่คล้ายกันจากด้าน Java จึงปฏิเสธได้ยากว่าเกิดข้อเปรียบเทียบในประเด็นของการใช้เวลาพัฒนาซอฟต์แวร์ขึ้นจริงระหว่างการพัฒนาโปรแกรมลูกข่ายด้วย Microsoft .NET และ Java แพล็ตฟอร์มด้วยเฟรมเวิร์คการเชื่อมต่อข้อมูลที่เหมาะสม จะทำให้สามารถพัฒนาระบบงานด้วยซอฟต์แวร์ประเภทแม่ข่ายโปรแกรมประยุกต์ของภาษา Java ซึ่งมีข้อดีเรื่องความมีเสถียรภาพ ร่วมกับการพัฒนาซอฟต์แวร์ลูกข่ายแบบบางบน Microsoft .NET ซึ่งมีความเร็วสูงและสะดวกต่อผู้ใช้ได้อย่างประสิทธิภาพ รวมทั้งสามารถลดเวลาในการพัฒนาระบบงานของผู้พัฒนาได้อีกด้วย

2. วัตถุประสงค์ของโครงการวิจัย

2.1 เพื่อออกแบบเฟรมเวิร์คการเชื่อมต่อข้อมูลสำหรับลูกข่ายแบบบางของระบบกลุ่มแม่ข่ายโปรแกรมประยุกต์

2.2 เพื่อพัฒนาต้นแบบของเฟรมเวิร์คการเชื่อมต่อข้อมูลสำหรับลูกข่ายแบบบางบนระบบปฏิบัติการ Windows รวมถึง Microsoft .NET Framework ให้สามารถติดต่อกับระบบกลุ่มแม่ข่ายโปรแกรมประยุกต์ภาษา Java

3. ขอบเขตของการวิจัย

3.1 การวิจัยครอบคลุมเฉพาะการสร้างเฟรมเวิร์คลูกข่ายบน Microsoft .NET Framework เพื่อใช้ความสามารถของการโยงข้อมูล (Data Binding) ที่มีใน Microsoft .NET Framework

3.2 ซอฟต์แวร์บนระบบเครื่องแม่ข่ายที่ใช้สร้างและทดสอบคือ JBoss Application Server [3]

3.3 การทดลองทำบนลูกข่ายเป็นระบบปฏิบัติการตระกูล Windows และแม่ข่ายเป็น RedHat Linux หรือ ระบบปฏิบัติการตระกูล Windows

4. ขั้นตอนการวิจัย

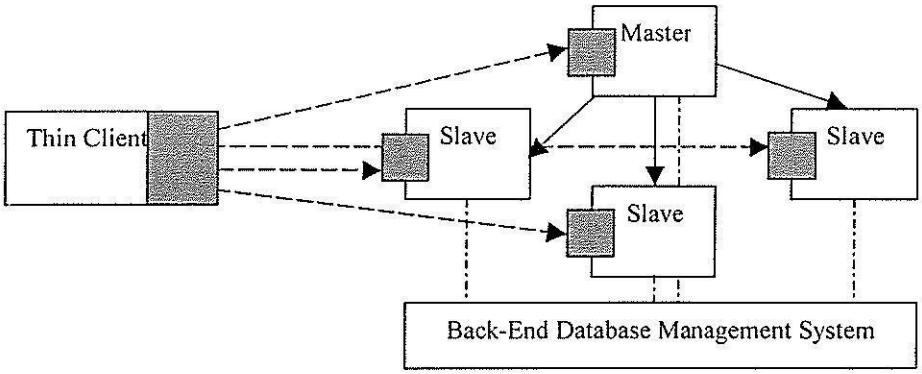
4.1 ศึกษาโครงสร้างของการส่ง/รับข้อมูลจาก EJB Application Server

4.2 ศึกษาโครงสร้างของการส่ง/รับข้อมูลจากลูกข่ายแบบบางที่มีลักษณะการเชื่อมต่อกับ GUI แบบการโยงข้อมูล (Data Binding)

4.3 พัฒนาเฟรมเวิร์คการเชื่อมต่อฝั่งลูกข่าย และพัฒนาตัวรับการเชื่อมต่อฝั่งแม่ข่าย

4.4 สร้างระบบงานสำหรับทดสอบที่ใช้ระบบการเชื่อมต่อที่พัฒนาขึ้น ตามรูปที่ 1

4.5 เปรียบเทียบประสิทธิภาพของระบบเชื่อมต่อแบบที่พัฒนาขึ้น กับระบบที่มีอยู่เดิม



รูปที่ 1. แบบจำลองการเชื่อมต่อลูกข่ายแบบบาง กับระบบแม่ข่าย
ส่วนที่แรเงาคือตัวเชื่อมต่อข้อมูล

รูปที่ 1 แสดงการเชื่อมต่อระหว่างลูกข่ายแบบบางกับระบบแม่ข่าย โดยระบบแม่ข่ายนั้นอยู่ในรูปแบบของ cluster ที่ประกอบไปด้วย โหนดหลัก(Master) และโหนดรอง(Slave) โดยทั้งหมดจะเชื่อมต่อกับระบบฐานข้อมูลเบื้องหลัง (Back-end Database Management System) ตัวเดียวกัน

5. ประโยชน์ที่คาดว่าจะได้รับ

5.1 เพื่อพัฒนาต้นแบบเฟรมเวิร์คลูกข่ายแบบบางที่สามารถเพิ่มประสิทธิภาพในการผลิตซอฟต์แวร์ระบบงาน

5.2 เพื่อเผยแพร่กลไกการพัฒนาและใช้งานเฟรมเวิร์คเป็นบริการความรู้แก่ภาคธุรกิจ

5.3 เพื่อนำไปสู่การผลิตเฟรมเวิร์คเชิงพาณิชย์

ในบทถัดไปจะกล่าวถึงพื้นฐานความรู้เกี่ยวกับเทคโนโลยีการโยงข้อมูลและ EJB ในบทที่ 3 จะกล่าวถึงการทำงานร่วมกันระหว่างแพล็ตฟอร์ม สำหรับบทที่ 4 จะอธิบายถึงการพัฒนาเฟรมเวิร์ค

และกระบวนการทดสอบประสิทธิภาพ บทที่ 5 แสดงผลการทดสอบประสิทธิภาพ และบทสุดท้าย บทที่ 6 เป็นการสรุป อภิปรายผลการทดสอบรวมทั้งข้อเสนอแนะ

บทที่ 2

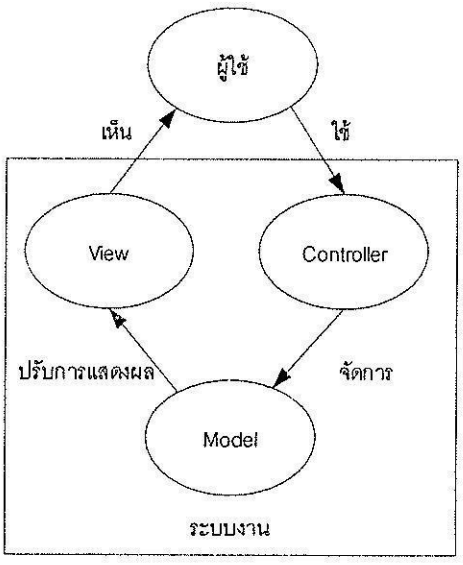
เทคโนโลยีการโยงข้อมูลและ Enterprise JavaBeans

1. เทคโนโลยีการโยงข้อมูล

เทคโนโลยีการโยงข้อมูลได้รับการพัฒนาและมีวิวัฒนาการตามการเปลี่ยนแปลงของเทคโนโลยีด้านวิศวกรรมซอฟต์แวร์โดยในยุคแรกมีการพัฒนาแบบจำลอง Model-View-Controller (MVC) [4] ขึ้นเพื่อใช้ในการจัดการส่วนติดต่อกับผู้ใช้ ต่อมาในยุคเริ่มต้นของ Java มีการนำเอา MVC มาปรับปรุงเพิ่มเติมซึ่งใช้ในการพัฒนา Java AWT และ Swing Components สำหรับอีกด้านหนึ่งกลุ่มผู้พัฒนาซอฟต์แวร์บนระบบปฏิบัติการ Windows ก็ได้มีการพัฒนาเทคโนโลยีการแสดงผลข้อมูลที่เรียกว่าการโยงข้อมูล (Data Binding) ตัวอย่างสภาวะแวดล้อมสำหรับการพัฒนา เช่น ในช่วงปี พ.ศ. 2538 ถึง 2543 คือ Microsoft Visual Basic และ Borland Delphi และในปัจจุบัน (พ.ศ. 2544 เป็นต้นมา) คือ Microsoft .NET Framework [2]

1.1 รูปแบบ Model-View-Controller

Model-View-Controller เป็นรูปแบบดั้งเดิมเพื่อใช้ในการออกแบบระบบติดต่อผู้ใช้ โดยแบ่งการทำงานออกเป็น 3 บทบาทคือ Model แทนข้อมูลที่ต้องการนำมาแสดงผล View แทนการแสดงผล และ Controller แทนตัวควบคุมการแสดงผล



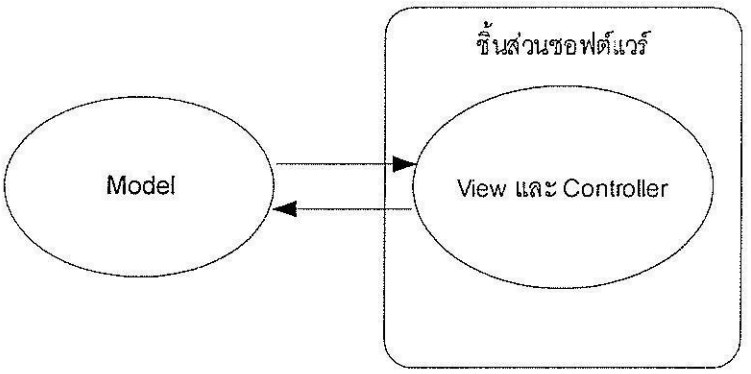
รูปที่ 2. แผนผังการทำงานของรูปแบบ MVC ที่กำลังติดต่อกับผู้ใช้

จากรูปที่ 2 เมื่อผู้ใช้เห็นการแสดงผลบนจอภาพซึ่งได้จากการปรับเปลี่ยนคุณสมบัติของส่วนประกอบซอฟต์แวร์สำหรับการแสดงผล เช่น ปุ่ม หรือ เมนู เป็นต้น ในที่นี่แสดงด้วย View ผู้ใช้อาจจะควบคุมระบบงานผ่านการใช้งานคีย์บอร์ดหรือเม้าส์ จากนั้นสัญญาณของการควบคุมจะไปกระตุ้นการทำงานของ Controller ที่รับผิดชอบส่วนประกอบซอฟต์แวร์บนจอภาพที่ผู้ใช้สั่งงาน ตัว Controller จะแปลความหมายการทำงานตามโปรแกรมและติดต่อไปยัง Model เมื่อรหัสโปรแกรมส่วนของ Model ได้รับคำสั่งจาก Controller และเกิดการเปลี่ยนแปลงก็จะปรับการแสดงผลไปยัง View ที่เกี่ยวข้อง จากนั้น View ก็จะแสดงผลโดยการวาดส่วนประกอบซอฟต์แวร์นั้น ๆ ใหม่บนจอภาพ

กลไกเบื้องหลังการทำงานของรูปแบบ MVC มักควบคุมไว้ด้วยรูปแบบอีกรูปแบบหนึ่งที่เรียกว่า Observer Pattern ซึ่งจะกล่าวเพิ่มเติมในส่วนถัดไป

1.2 Model-View-Controller ใน Java

ระบบติดต่อผู้ใช้ใน Java ได้รับอิทธิพลค่อนข้างสูงจาก MVC โดยมีการอิมพลีเมนต์รูปแบบ MVC อย่างชัดเจนใน Java AWT และ Swing Component อย่างไรก็ตาม MVC ที่อิมพลีเมนต์ใน Java เป็นรูปแบบที่ได้รับการปรับปรุงโดยนำเอาแนวคิดส่วนของ View และ Controller รวมกันไว้ในหน่วยชิ้นส่วนซอฟต์แวร์เดียวกัน และแยก Model ออกไว้ด้านนอก โดยทั่วไปจะเรียกสถาปัตยกรรมแบบนี้ว่า MVC แบบแก้ไข (Modified MVC)



รูปที่ 3. รูปแบบ MVC แบบแก้ไขที่พบใน Java

1.3 เทคโนโลยีการโยงข้อมูล

เทคโนโลยีการโยงข้อมูล (Data Binding) สามารถพิจารณาว่าคล้ายกับ V และ M ในรูปแบบ MVC โดยนำข้อมูลจาก Model มาแสดงใน View โดยตรง ด้วยการระบุค่าบางอย่างเช่น ชื่อของเขต

ข้อมูล หรือชื่อของคุณสมบัติของวัตถุ (Property) โดยลักษณะทั่ว ๆไปของเทคนิคการโยงข้อมูลจะเป็นการกำหนดให้คอมโพเน้นท์ที่มีหน้าที่เป็น View โยงไปหา Model

1.3.1 เทคโนโลยีการโยงข้อมูลในยุคแรก

เทคโนโลยีการโยงข้อมูลในยุคแรกจะกล่าวถึงเทคนิคที่พบใน Borland Delphi และ Microsoft Visual Basic

ใน Borland Delphi นั้น สถาปัตยกรรมด้านคอมโพเน้นท์ที่ติดต่อกับผู้ใช้จะอยู่บนเฟรมเวิร์คที่เรียกว่า Visual Component Library (VCL) โดยเทคโนโลยีการโยงข้อมูลก็ถูกพัฒนาขึ้นบน VCL เพื่อใช้สำหรับให้ผู้พัฒนาสามารถโยงคอมโพเน้นท์เข้ากับข้อมูลที่ดึงมาจากฐานข้อมูลได้ [5]

VCL ที่มีความสามารถด้านการโยงข้อมูลจะเป็น VCL สำหรับการแสดงผลชุดพิเศษที่ต่อกับฐานข้อมูลโดยเฉพาะโดยจะมีคุณสมบัติตัวหนึ่งชื่อว่า DataSource เพื่อโยงไปหาคอมโพเน้นที่เป็นแหล่งข้อมูลซึ่งคอมโพเน้นท์ตัวดังกล่าวก็จะโยงไปหาตัวเก็บข้อมูลที่ต่ออยู่กับระบบฐานข้อมูล สำหรับคุณสมบัติอีกค่าหนึ่งที่จะต้องระบุใน VCL เพื่อใช้ในการแสดงผลดังกล่าวคือคุณสมบัติชื่อ FieldName ซึ่งจะเป็นตัวระบุชื่อเขตข้อมูลที่ต้องการข้อมูลมาแสดงจาก DataSource

ตัวอย่างเช่น มีวัตถุ DBEdit1 เป็น TDBEdit เมื่อต้องการโยงการแสดงผลข้อมูลจาก ตารางชื่อ Student ที่มีเขตข้อมูล ID และ NAME ก็จะต้องทำการเชื่อมต่อ วัตถุ DataSource1 ที่เป็น TDataSource ไปยังคอมโพเน้นท์ที่สามารถต่อไปยังตาราง Student ได้ (เช่น TClientDataSet หรือ TADODataSet เป็นต้น [x]) จากนั้นเมื่อทำการตั้งค่าคุณสมบัติดังต่อไปนี้

```
DBEdit1.DataSource = DataSource1
DBEdit1.FieldName = 'NAME'
```

จะทำให้ DBEdit1 โยงหาข้อมูลจากเขตข้อมูล NAME ในตาราง Student ผ่านทาง DataSource1 เป็นต้น

ในทำนองเดียวกัน การโยงข้อมูลใน Microsoft Visual Basic นั้นอยู่บนสถาปัตยกรรม Component Object Model (COM) [6] และตัวคอมโพเน้นท์สำหรับแสดงผลนั้นมีความจำเป็นที่ต้องโยงหา DataSource ในลักษณะที่คล้ายกันกับ VCL ของ Borland Delphi ข้อแตกต่าง คือ คอมโพเน้นท์ของ Microsoft Visual Basic เช่น Textbox นั้น สามารถใช้งานเป็นทั้งคอมโพเน้นท์ปกติและคอมโพเน้นท์

ที่ใช้สำหรับโยงข้อมูล ในขณะที่ใน Borland Delphi จะมีคอมโพเน้นท์แยกเป็น 2 ตัว เช่น ตามตัวอย่างที่ผ่านมาจะมี TEdit สำหรับการแสดงผลปกติ และ TDBEdit สำหรับโยงข้อมูลกับระบบฐานข้อมูล

1.3.2 เทคโนโลยีการโยงข้อมูลใน OpenLaszlo ด้วย XML

การโยงข้อมูลอีกชนิดหนึ่งที่ใช้กันแพร่หลายมากขึ้น คือ การโยงตัวแสดงผลกับข้อมูลประเภท XML (eXtensible Markup Language) [7] จุดต่างของการโยงข้อมูลในลักษณะนี้กับเทคโนโลยีที่กล่าวถึงในหัวข้อ 1.3.1 ก็คือ ในหัวข้อ 1.3.1 ลักษณะข้อมูลจะเป็นข้อมูลโดยตรงที่ดึงมาจากระบบฐานข้อมูล แต่เทคโนโลยีที่จะกล่าวถึงในหัวข้อนี้นั้น ข้อมูลได้รับการประมวลผลมาครั้งหนึ่งแล้วผ่านตัวกลาง หรือตัวควบคุม (Controller) ซึ่งจะแปรสภาพข้อมูลจากฐานข้อมูลให้อยู่ในรูปแบบ XML ตามที่ตกลงกันระหว่างตัวควบคุม และตัวแสดงผล

โครงสร้าง XML ใน OpenLaszo [8] จะเก็บไว้ในคลาส dataset ในรูปแบบเชิงวัตถุ โดยสามารถเข้าถึงได้ด้วยคุณสมบัติ nodes ในแต่ละตัวแสดงผลของ OpenLaszlo จะมีคุณสมบัติชื่อ datapath หรือทางเดินข้อมูลไว้สำหรับอ้างอิงถึงข้อมูลในโครงสร้าง XML ที่ต้องการ โดย datapath จะต้องขึ้นต้นด้วยชื่อ dataset ที่ต้องการอ้างถึง ซึ่งค่าทางเดินข้อมูลนั้นจะเป็นไปตามข้อกำหนดมาตรฐาน Xpath ของ W3C ยกตัวอย่างเช่น

```
<text datapath="dsXML:/myXML/person[2]/firstname/text()"/>
```

หมายถึง ตัวแสดงผลชื่อ text โยงเข้ากับโครงสร้าง XML ของ dataset ชื่อ dsXML โดย dataset ดังกล่าวมีโหนดเริ่มต้นเป็น myXML และมีการโยงไปหาโหนดลูกชื่อ person ลำดับที่สอง และโยงต่อไปหาโหนดลูกชื่อ firstname และนำข้อความของโหนด firstname มาแสดงผล

โครงสร้าง XML อาจจะมีการกำหนด attribute ของแต่ละโหนดไว้แทนข้อความของโหนด ซึ่งสามารถนำมาแสดงผลได้โดยการระบุสัญลักษณ์ @ ไว้หน้า attribute ที่ต้องการอ้างถึง เช่น

```
<text datapath="dsXML:/myXML/person[2]/@id"/>
```

จะหมายถึงการแสดงข้อมูลจาก attribute ชื่อ id ของโหนด person ลำดับที่สองเป็นต้น

1.3.3 เทคโนโลยีการโยงข้อมูลใน Microsoft .NET

Microsoft .NET เป็นแพล็ตฟอร์มเสมือนสำหรับรันโค้ด Common Intermediate Language (CIL) [9] ที่สนับสนุนภาษาโปรแกรมหลายภาษา โดยภาษาหลักที่ได้รับความนิยมก็คือภาษา C# และ VisualBasic.NET สถาปัตยกรรมของ Microsoft .NET มีลักษณะคล้ายกับแพล็ตฟอร์ม Java อย่างไรก็ตามจุดเด่นที่มีการพัฒนาให้ดีกว่า Java ก็คือส่วนของการโยงข้อมูล

การโยงข้อมูลใน Microsoft .NET นั้นสามารถทำได้บนคอมโพเน้นท์สำหรับแสดงผลหรือเรียกว่า คอนโทรล (Control) ได้ทุกตัวในลักษณะเดียวกับการโยงข้อมูลใน Microsoft Visual Basic จุดที่ได้รับการพัฒนาขึ้นและเป็นการใช้ความสามารถของ Microsoft .NET ก็คือ การโยงข้อมูลไม่ได้จำกัดอยู่ที่ DataSource ในลักษณะเดียวกับ Microsoft Visual Basic หรือ Borland Delphi แต่สามารถโยงได้กับอะเรย์หรือรายการของวัตถุชนิดใด ๆ ก็ได้ นอกจากนั้นหากต้องการสร้างรายการของวัตถุแบบเฉพาะขึ้นมาเองก็สามารถสร้างคลาสที่อิมพลีเม้นต์ IList และอินเตอร์เฟสอื่น ๆ ที่ตัวแพล็ตฟอร์มรองรับไว้ได้ เป็นต้น สำหรับการระบุชื่อเขตข้อมูลเพื่อนำข้อมูลที่ต้องการมาแสดงผลนั้น ทำได้โดยใช้ชื่อของคุณสมบัติที่มีประกาศอยู่ในแต่ละคลาส รวมทั้งใช้คุณสมบัติซ้อนกันได้หลาย ๆ ระดับ โดยการระบุชื่อคุณสมบัติแล้วคั่นด้วยจุด เช่น Product.Vendor.Name เป็นการอ้างอิงถึงชื่อร้านค้าของสินค้าที่ระบุ เป็นต้น

1.3.4 เทคโนโลยีการโยงข้อมูลใน JavaFX Script

ภาษาโปรแกรม JavaFX Script [10] เป็นภาษาโปรแกรมแบบ Script ที่ถูกพัฒนาขึ้นโดยบริษัท Sun Microsystems JavaFX Script นี้ใช้ไวยากรณ์เชิงประกาศในการระบุคอมโพเน้นท์แบบ GUI ทำให้ผู้ใช้

สามารถเข้าใจการเขียนโปรแกรมได้ง่ายเมื่อเทียบกับการวางรูปแบบของ GUI จริงๆ JavaFX Script ยังเป็นภาษาแบบ Statically typed และมีคุณสมบัติโครงสร้างของข้อมูล ความสามารถในการนำกลับมาใช้ใหม่ และ Encapsulation เช่นเดียวกันกับภาษา Java นอกจากนี้ การโยงข้อมูลและการประเมินผลข้อมูลแบบ incremental เชิงประกาศนี้ ทำให้ผู้ใช้ภาษา JavaFX Script สามารถสร้างและปรับแต่งคอมโพเน้นท์ แต่ละอันได้ง่ายขึ้น โดยการ Synchronize ข้อมูลระหว่างโปรแกรมประยุกต์และคอมโพเน้นท์แบบ GUI อัตโนมัติ

การโยงข้อมูลใน JavaFX Script ใช้คีย์เวิร์ด bind สำหรับการโยงค่าของตัวแปรเข้ากับนิพจน์ที่โยง นิพจน์ที่โยงเข้ากับตัวแปรสามารถเป็นค่าของชนิดของตัวแปรพื้นฐาน วัตถุ ผลลัพธ์ของฟังก์ชั่นหรือผลลัพธ์ของนิพจน์ก็ได้

1.3.5 เทคโนโลยีการโยงข้อมูลใน ZK

ZK [11] เฟรมเวิร์คเป็นเฟรมเวิร์คที่มีขนาดเล็กที่ถูกออกแบบมาเพื่อเพิ่มประสิทธิภาพของการทำงานระดับเอ็นเตอร์ไพรส์โดยใช้ต้นทุนในการพัฒนาที่น้อยลง ZK ถูกพัฒนาขึ้นโดยบริษัท Potix Corporation ZK เฟรมเวิร์คมีระบบพื้นฐานของ AJAX ที่ครบถ้วน สามารถนำไปพัฒนาโปรแกรมประยุกต์แบบเวบได้ง่าย การโยงข้อมูลของ ZK สามารถทำได้โดยการกำหนดค่าของคุณสมบัติ bind และ dataSource ของ ZUL element คุณสมบัติ dataSource จะเชื่อมต่อกับวัตถุและชื่อของคุณสมบัติที่ต้องการโยงจะถูกกำหนดให้กับคุณสมบัติ bind เมื่อมีการเปลี่ยนแปลงค่าของ ZUL element แล้ว element ทั้งหมดที่ถูกโยงจะทำการอัพเดตข้อมูลอัตโนมัติ

2. Enterprise JavaBeans

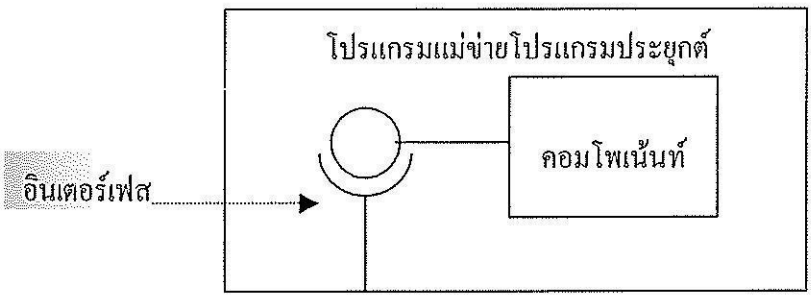
Enterprise JavaBeans (EJB) [12, 13] เป็นสถาปัตยกรรมคอมโพเน้นท์ฝั่งแม่ข่ายที่ช่วยให้กระบวนการพัฒนาโปรแกรมระดับเอ็นเตอร์ไพรส์ (Enterprise-class application) ที่ประกอบไปด้วยคอมโพเน้นท์แบบกระจาย (Distributed component) ทำได้ง่ายขึ้น EJB สร้างขึ้นมาเพื่อใช้สำหรับภาษา Java ประโยชน์ในการใช้งาน EJB คือทำให้สามารถสร้างโปรแกรมที่เชื่อถือได้ ปลอดภัย และพัฒนาต่อได้ง่าย โดยไม่มีความจำเป็นในการพัฒนาเฟรมเวิร์คที่ซับซ้อนเพื่อจัดการคอมโพเน้นท์แบบกระจายขึ้นเอง EJB นั้นตั้งใจจะมีไว้เพื่อให้สามารถพัฒนาคอมโพเน้นท์ฝั่งแม่ข่ายได้อย่างรวดเร็วโดยการพัฒนา EJB คอมโพเน้นท์ด้วยภาษา Java และสามารถนำระบบที่พัฒนาขึ้นไปใช้งานในระบบพื้นฐานเดียวกันที่เป็นโปรแกรมแม่ข่ายซึ่งรองรับมาตรฐาน EJB และด้วยการออกแบบ EJB ให้เป็นมาตรฐาน ทำให้ตัวคอมโพเน้นท์และระบบงานสามารถใช้งานได้โดยไม่ยึดติดกับโปรแกรมแม่ข่ายของบริษัทใดบริษัทหนึ่ง

ระบบแบบกระจาย (Distributed system) [13] คือระบบขนาดใหญ่ที่สามารถแบ่งส่วนประกอบในระบบออกจากกันเป็นชิ้นย่อย ๆ หรือ คอมโพเน้นท์ ได้ อีกทั้งยังแยกออกแบบระดับชั้น (Layer) ที่เป็นอิสระและไม่ขึ้นต่อกัน ความต้องการในการพัฒนาระบบแบบกระจายทำให้เกิดโปรแกรมแม่ข่ายโปรแกรมประยุกต์ (Application Server) หรือเรียกกันโดยทั่วไปว่า ตัวกลาง (Middleware – มิดเดิ้ลแวร์) ที่ผ่านมาองค์กรต่าง ๆ ต่างก็สร้างตัวกลางขึ้นมาใช้งานเอง อย่างไรก็ตามเมื่อจำนวนถูกข่ายเพิ่มมากขึ้นและต้องการระบบงานที่ใช้ตัวกลางกันในระดับคลัสเตอร์ ทำให้มีแนวคิดในการสร้างโปรแกรมแม่ข่ายโปรแกรมประยุกต์ที่เป็นมาตรฐานขึ้น เพื่อให้ผู้พัฒนาระบบไม่จำเป็นต้องกังวล

เกี่ยวกับสาธารณูปโภคพื้นฐานของแม่ข่ายโปรแกรมประยุกต์ จึงมีเวลามากขึ้นในการแก้ปัญหาเชิงตรรกทางธุรกิจให้กับระบบ

2.1 สถาปัตยกรรมของคอมโพเน้นท์

ปัญหาในยุคแรกของการพัฒนาระบบในโปรแกรมแม่ข่ายโปรแกรมประยุกต์ที่ยังไม่มีมาตรฐานคือ การพัฒนาระบบแล้วทำให้ต้นรหัสยึดติดกับตัวโปรแกรมแม่ข่ายสถาปัตยกรรมคอมโพเน้นท์ของ Java จึงออกแบบมาเพื่อให้ตัวคอมโพเน้นท์สามารถย้ายจากโปรแกรมแม่ข่ายของบริษัทหนึ่งไปยังอีกบริษัทหนึ่งได้โดยที่ไม่ต้องแก้ไขและคอมไพล์ต้นรหัสใหม่ แนวคิดดังกล่าวโยงไปหากระบวนการประกาศข้อตกลงในการโปรแกรมเชิงวัตถุ หรือ อินเตอร์เฟส (Interface) โดยจะกำหนดกลุ่มของอินเตอร์เฟสระหว่างคอมโพเน้นท์และโปรแกรมแม่ข่ายโปรแกรมประยุกต์ไว้ตั้งแต่แรก การตั้งข้อตกลงดังกล่าวทำให้สามารถย้ายคอมโพเน้นท์ไปทำงานบนโปรแกรมแม่ข่ายใด ๆ ก็ได้ที่มีมาตรฐานตามข้อตกลง และข้อตกลงนี้เองคือสถาปัตยกรรมคอมโพเน้นท์ของ EJB รูปที่ 4 แสดงภาพคอมโพเน้นท์ที่อยู่ในโปรแกรมแม่ข่ายโปรแกรมประยุกต์ซึ่งเชื่อมต่อกันอยู่ด้วยอินเตอร์เฟสตามข้อตกลง



รูปที่ 4. คอมโพเน้นท์ในโปรแกรมแม่ข่าย

2.2 การใช้ EJB เป็น คอมโพเน้นท์ระดับธุรกิจ

โดยทั่วไประบบงานจะแยกได้ออกเป็น 3 ระดับชั้นดังต่อไปนี้

1. ระดับแสดงผล (Presentation Layer)
2. ระดับตรรกเชิงธุรกิจ (Business Logic Layer)
3. ระดับวัตถุเชิงธุรกิจ (Business Object Layer)

โดย EJB จะรับทำหน้าที่ในส่วนที่ 2 และ 3 ในระดับแสดงผลนั้น เป็นการทำงานของระบบด้านลูกข่าย (client-side) ที่เกี่ยวข้องกับผู้ใช้โดยตรง เช่น โปรแกรมลูกข่าย หรือ ตัวแสดงหน้าเวบ โปรแกรมในกลุ่มนี้ที่รับผิดชอบเกี่ยวกับเวบอาจถูกเรียกว่า ระดับชั้นเวบ (Web-tier) ก็ได้ ในทางตรงข้าม EJB ซึ่งมีหน้าที่ในระดับที่ 2 และ 3 ไม่ได้ทำงานที่เกี่ยวข้องกับด้านลูกข่าย คอมโพเน้นท์ที่สร้างเป็น EJB จึงมักเรียกว่า คอมโพเน้นท์ด้านแม่ข่าย (Server-side) โดยหมายถึงการทำงานบนฝั่งแม่ข่าย มีหน้าที่เกี่ยวกับการประมวลผลที่ซับซ้อน หรือ ทำทรานแซ็คชั่นทางธุรกิจจำนวนมาก เป็นต้น คอมโพเน้นท์ด้านแม่ข่ายมักมีความจำเป็นที่ต้องทำงานได้ตลอดเวลา ทนต่อความผิดพลาด มีระบบจัดการทรานแซ็คชั่น สนับสนุนผู้ใช้หลายคน และต้องมีความปลอดภัยของการขนย้ายข้อมูล เป็นต้น โปรแกรมแม่ข่ายโปรแกรมประยุกต์จะเป็นตัวให้บริการนี้แก่คอมโพเน้นท์ EJB รวมถึงกระบวนการจัดการ EJB ขณะที่ระบบทำงานอยู่ นั่นหมายถึง EJB จะรับหน้าที่ในการแก้ปัญหาตรรกเชิงธุรกิจอย่างเดียวดังที่กล่าวไว้แล้ว และโดยปกติ EJB จะใช้ในงานต่อไปนี้

1. การทำงานตามตรรกเชิงธุรกิจ
2. การเข้าถึงข้อมูลในฐานข้อมูล
3. การเข้าถึงข้อมูลในระบบภายนอก

นั่นหมายความว่าการใช้งาน EJB ต้องการลูกข่ายประเภทใดประเภทหนึ่ง อย่างเช่น ลูกข่ายแบบหนา (Think Client) เวบชนิดพลวัต (Dynamic Web) หรือ ลูกข่ายเวบเซอร์วิส (Web Service Client) เป็นต้น

2.3 EJB พื้นฐาน

ส่วนนี้จะกล่าวถึงความแตกต่างกันของ EJB แต่ละประเภท และส่วนประกอบของ EJB ซึ่งได้แก่ คลาสของ EJB (Class of EJB), อินเตอร์เฟสรีโมท (Remote interface), อินเตอร์เฟสโลคอล (Local interface), วัตถุ EJB (EJB object), วัตถุโลคอล (Local object), อินเตอร์เฟสโฮม (Home interface), วัตถุโฮม (Home object), ตัวอธิบายการดีพลอย (Deployment descriptor) และ ไฟล์ EJB-jar

เทคโนโลยี EJB อยู่บนพื้นฐานของ เทคโนโลยี 2 อย่างคือ RMI-IIOP และ JNDI ในที่นี้จะอธิบายถึงพื้นฐานของเทคโนโลยี RMI-IIOP

2.3.1 Java RMI-IIOP

Java RMI-IIOP [13] มีชื่อเต็มว่า Java Remote Method Invocation over the Internet Inter-ORB Protocol เป็นกลไกพื้นฐานที่ใช้ใน J2EE เพื่อทำการเรียกใช้เมธอดระยะไกล ใช้สำหรับสร้างวัตถุแบบ

กระจาย ทำให้วัตถุติดต่อกันได้ทั้งในระดับหน่วยความจำเดียวกันต่างเวอร์ช่วลแมชชีน หรือต่างเครื่อง (ข้ามเครือข่ายหรือกระบวนการของเครือข่าย)

การเรียกกระบวนความระยะไกล (Remote Procedure Call) หรือ RPC เป็นการเรียกใช้กระบวนความจากโปรเซสซึ่งอยู่บนเครื่องหนึ่งไปยังอีกเครื่องหนึ่ง การใช้งาน RPC ทำให้คงตัวกระบวนความไว้ที่เครื่องใด ๆ แต่สามารถเรียกใช้กันข้ามโปรเซสหรือข้ามเครื่องได้

การเรียกใช้เมธอดระยะไกล (Remote Method Invocation) ในแพลตฟอร์ม Java ใช้แนวคิดของ RPC ดังกล่าวและปรับปรุงเพื่อให้สื่อสารได้ในระดับวัตถุ ซึ่งเป็นวัตถุแบบกระจาย การใช้ RMI-IIOP จึงทำให้ไม่เพียงแต่เรียกใช้กระบวนความอย่างเดียวเท่านั้น หากยังเรียกใช้เมธอดในวัตถุระยะไกลได้อีกด้วย ซึ่งทำให้สามารถพัฒนารหัสที่ต้องการใช้งานบนเครือข่ายด้วยแนวคิดเชิงวัตถุทั้งการสืบทอด การปิดซ่อน เป็นต้น RMI พัฒนาขึ้นโดยช่วยแก้ปัญหาพื้นฐานในการเรียกกระบวนความระยะไกลดังต่อไปนี้ [13]

1. การมาร์แชลและการอันมาร์แชล (Marshalling and unmarshalling)

กลไกของ RMI จะอนุญาตให้ส่งพารามิเตอร์ไปยังเมธอดระยะไกลได้ทั้งในรูปแบบของชนิดข้อมูลพื้นฐานและชนิดข้อมูลแบบวัตถุผ่านทางเครือข่าย แต่อาจเป็นไปได้ว่าเครื่องระยะไกลอาจใช้รูปแบบข้อมูลที่อยู่ในระบบที่ต่างกัน หรือแม้แต่มีกลไกการจัดการวัตถุด้วยวิธีที่ต่างกัน เช่น เครื่องระยะไกลมีระบบจัดการหน่วยความจำที่ต่างไปจากเครื่องต้นทาง เป็นต้น RMI จึงจำเป็นต้องพึ่งพากระบวนการ Marshalling และ Unmarshalling ซึ่งทำหน้าที่ประมวลผลพารามิเตอร์ของเมธอดเพื่อให้สามารถใช้ข้อมูลเหล่านั้นบนเครื่องระยะไกลที่มีสภาวะแวดล้อม (ระบบปฏิบัติการ, หน่วยประมวลผล, ฮาร์ดแวร์อื่น ๆ) ต่างจากเครื่องต้นทางได้

2. รูปแบบการส่งค่าพารามิเตอร์ (Parameter Passing Conventions)

การส่งค่าพารามิเตอร์แบ่งออกได้เป็น 2 ลักษณะใหญ่ ๆ คือ การส่งค่า และ การส่งค่าอ้างอิง เมื่อเกิดการส่งค่าแบบปกติ ค่าที่ส่งไปยังเมธอดระยะไกลคือสำเนาของค่าจากต้นทาง ซึ่งหมายถึงการเปลี่ยนแปลงค่าที่ปลายทางจะไม่เกิดผลกระทบแต่อย่างใดต่อค่าในเครื่องต้นทาง ในขณะที่พารามิเตอร์ที่ส่งไปด้วยการส่งค่าอ้างอิงนั้นจะเกิดการเปลี่ยนแปลงค่าได้เมื่อมีการแก้ไขค่าที่เครื่องระยะไกล

3. ผลกระทบจากความไม่มีเสถียรภาพของเครือข่ายและเวอร์ช่วลแมชชีน (Network or Virtual Machine Instability)

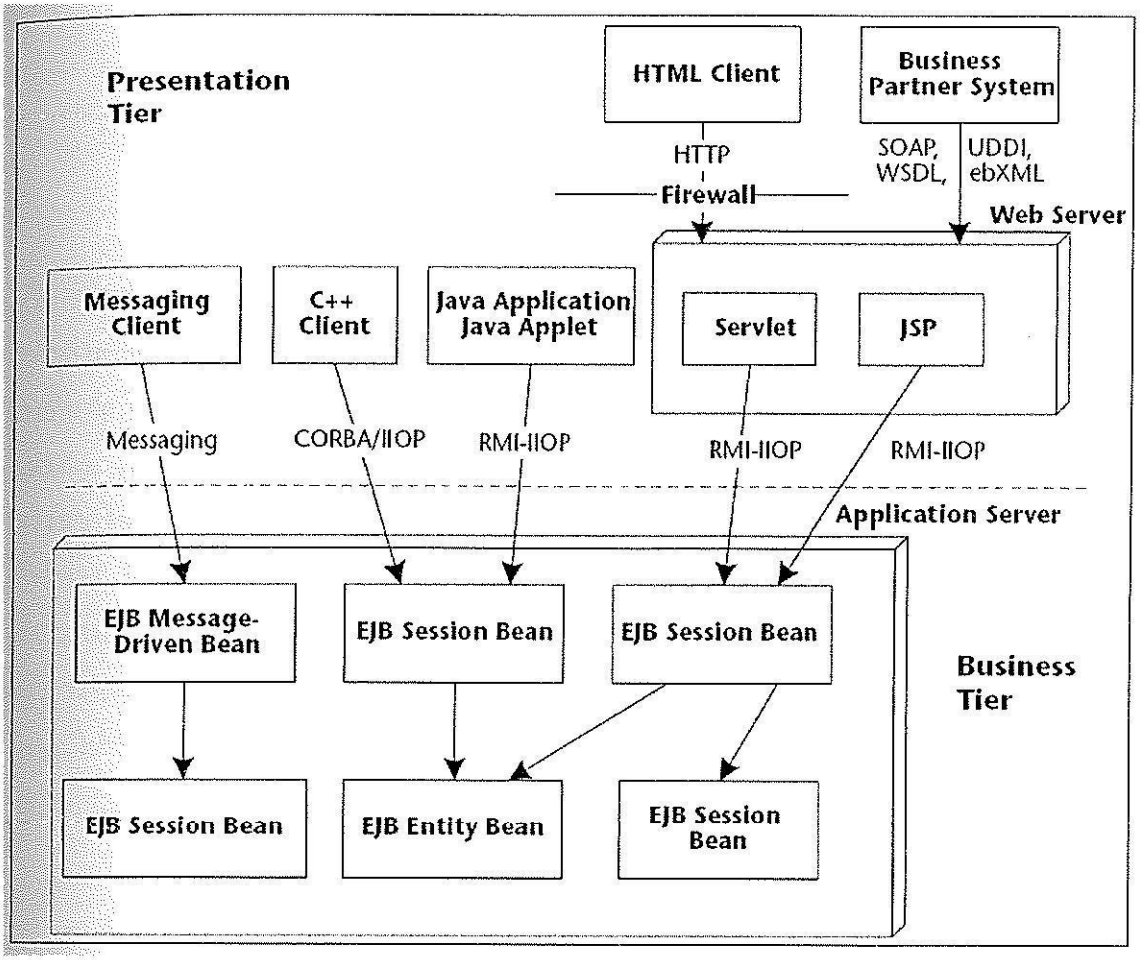
การพังของเวอร์ช่วลแมชชีนในการใช้งานในระบบเดียว ทำให้โปรแกรมประยุกต์ทั้งหมดที่ทำงานอยู่บนเวอร์ช่วลแมชชีนนั้นพังไปด้วย แต่ในกรณีของโปรแกรมประยุกต์แบบกระจาย ซึ่งมีเวอร์ช่วลแมชชีนหลายตัวทำงานอยู่ด้วยกันนั้น การพังของแมชชีนตัวใดตัวหนึ่งไม่ได้ทำให้ระบบล่มทั้งระบบ นั้นคือการติดต่อระหว่างแมชชีนด้วย RMI ย่อมมีผลกระทบจากความไม่เสถียรดังกล่าว อย่างไรก็ตาม RMI ได้มีการออกแบบให้ทำงานบนมาตรฐานซึ่งทำให้สามารถแจ้งข้อผิดพลาดที่เกี่ยวข้องกับความไม่เสถียรของระบบได้ โดยผู้พัฒนาไม่ต้องจัดการข้อผิดพลาดเหล่านั้นด้วยตนเอง

2.4 ประเภทของ EJB

ประเภทของ EJB แบ่งออกได้เป็น 3 ประเภท

1. Session Bean ใช้สำหรับจำลองแบบการะบวนการเชิงธุรกิจ ซึ่งมักจะอยู่ในรูปของการกระทำ เช่น การคำนวณ การเข้าถึงฐานข้อมูล การเรียกใช้บริการจากระบบงานอื่น หรือการเรียกใช้งาน Bean ตัวอื่น
2. Entity Bean ใช้สำหรับจำลองข้อมูลทางธุรกิจ เป็นวัตถุสำหรับเก็บข้อมูล กลไกการทำงานพื้นฐานของ Entity Bean คือการแคชข้อมูลจากระบบฐานข้อมูลเพื่อให้ลูกข่ายสามารถเข้าถึงข้อมูลได้อย่างมีประสิทธิภาพเพิ่มขึ้น ตัวอย่างเช่น ข้อมูลสินค้า, การสั่งซื้อ, พนักงาน เป็นต้น
3. Message-Driven Bean คล้ายกับ Session Bean ตรงที่เป็นการกระทำเหมือนกัน สิ่งที่ต่างกันก็คือ การเรียกใช้ Message-Driven Bean ทำได้โดยการส่งข้อความแทนการเรียกใช้โดยตรง

รูปที่ 5 แสดงการเรียกใช้ EJB ชนิดต่าง ๆ ในระบบคอมโพเน้นท์ EJB จากลูกข่ายต่างชนิดกัน เช่น HTML Client, Java Application, Java Applet และ Messaging Client



รูปที่ 5. แสดงการเรียกใช้ระบบคอมโพเน้นท์ EJB จากลูกข่าย [13]

บทที่ 3
การทำงานร่วมกันระหว่างแพล็ตฟอร์ม

การทำงานร่วมกันระหว่างแพล็ตฟอร์ม (Platform Interoperability) ในแง่ของซอฟต์แวร์จะเป็นความสามารถในการทำงานร่วมกันของโปรแกรมผ่านข้อตกลงหนึ่ง ๆ ระหว่างแพล็ตฟอร์มที่แตกต่างกันเพื่อให้งานพื้นฐานบางอย่างสามารถสำเร็จไปได้ ในบทนี้จะกล่าวถึงการทำงานร่วมกันของแพล็ตฟอร์ม Java และ Microsoft .NET โดยผ่านโปรโตคอลสำหรับเรียกใช้เมธอดระยะไกลที่ชื่อว่า Hessian [14] โดยมีฝั่งแม่ข่ายซึ่งเป็น Java และฝั่งลูกข่ายทำงานบน Microsoft .NET โดยปกติแล้วธรรมชาติของวัตถุบน Java และ .NET มีความแตกต่างกันและไม่สามารถเรียกใช้กันได้โดยตรง อย่างไรก็ตาม มีผู้พัฒนา IKVM [15] ซึ่งเป็นเครื่องจักรเสมือนสำหรับ Java (Java Virtual Machine) ที่สามารถทำงานได้บน Microsoft .NET ขึ้นมาโดยนอกจาก IKVM จะสามารถรัน Java แอพพลิเคชั่นบน Microsoft .NET ได้แล้วก็ยังมีคุณสมบัติพิเศษอีกประการหนึ่งก็คือการคอมไพล์ bytecode ภาษา Java ไปเป็น Common Intermediate Language (CIL) สำหรับ Microsoft .NET ด้วยคอมไพเลอร์ของ IKVM (ikvmc.exe)

1. แนวคิดทั่วไปในการทำงานข้ามแพล็ตฟอร์ม

แนวคิดของการทำงานข้ามแพล็ตฟอร์มที่ใช้ในงานวิจัยนี้สามารถพิจารณาได้เป็น 2 แนวคิดซึ่งใช้งานร่วมกัน คือ

1. การแปลงรหัสจากแพล็ตฟอร์มหนึ่งให้สามารถทำงานได้บนอีกแพล็ตฟอร์มหนึ่ง
2. การเรียกใช้เมธอดระยะไกลข้ามแพล็ตฟอร์ม

2. โปรโตคอล Hessian

ปัจจุบันการใช้งาน Web Service [16] ได้รับความนิยมอย่างกว้างขวาง สำหรับเป็นตัวกลางในการถ่ายโอนข้อมูลและการทำงานร่วมกันระหว่างแพล็ตฟอร์ม ด้วยข้อดีของ Web Service ซึ่งก็คือใช้ภาษา XML เป็นตัวสื่อนั้นทำให้มีการนำ Web Service ไปใช้บนหลาย ๆ ภาษา อย่างไรก็ตาม ข้อเสียจุดใหญ่ของ Web Service ในเรื่องประสิทธิภาพก็ได้รับการพูดถึงกันอย่างแพร่หลาย รวมทั้งการแลกเปลี่ยนข้อมูลที่ซับซ้อนตามมาตรฐานของ Web Service ที่เรียกว่า Complex Type นั้นไม่สนับสนุนการใช้งานชนิดข้อมูลประเภท List หรือ Set ซึ่งใช้กันมากในภาษา Java สำหรับการเก็บข้อมูลที่ดึงออกมาจากระบบฐานข้อมูล โดยมาตรฐานปัจจุบันของ Web Service (1.1) นั้น สนับสนุนอะเรย์ของวัตถุเพียง

อย่างเดียว ซึ่งทำให้ต้องแปลงจาก List หรือ Set ให้เป็นอะเรย์ก่อนส่งข้อมูลที่ต้องการข้ามเครือข่าย นอกจากจะมีผลเรื่องประสิทธิภาพแล้วการใช้งานร่วมกับ .NET ก็จะไม่สะดวก ในระยะเดียวกันนั้นก็ได้มีการพัฒนาโปรโตคอลอื่น ๆ ที่สามารถส่งข้อมูลบน HTTP ได้ในลักษณะเดียวกับ Web Service แต่มีการส่งข้อมูลในรูปของไบนารี่แทน XML ซึ่งเป็นข้อความ หนึ่งในโปรโตคอลเหล่านั้นก็คือ Hessian ซึ่งได้รับความนิยมพอสมควรบนแพล็ตฟอร์ม Java อย่างไรก็ตาม เมื่อมีความจำเป็นในการติดต่อข้ามแพล็ตฟอร์มระหว่าง Java และ .NET ตัว Hessian เองก็มีข้อจำกัดเนื่องจาก HessianC# [14] ซึ่งเป็นอิมพลีเม้นเทชั่นของโปรโตคอล Hessian บน .NET นั้นใช้ความสามารถในการเข้าถึงวัตถุแบบ .NET จึงไม่มีความสามารถในการจัดการกับวัตถุแบบ Java ได้

ในงานวิจัยนี้ผู้พัฒนาได้แก้ไขต้นรหัสของ Hessian ในภาษา Java และทำการแปลง Hessian ด้วย IKVM ในส่วนย่อยที่ 3.1 เพื่อให้การทำงานของตัวโปรแกรมยังเข้าถึงข้อมูลวัตถุตามรูปแบบที่กำหนดไว้เดิมของ Java แต่ทำงานได้บน .NET การแปลงในลักษณะนี้ส่งผลให้ตัวอิมพลีเม้นเทชั่นดังกล่าวเป็นตัวกลางในการให้แอพพลิเคชั่นทั้งบนฝั่ง .NET และ Java สามารถใช้ข้อมูลชนิดวัตถุร่วมกันได้

2.1 การใช้ Hessian เพื่อเรียกใช้เซอร์วิสระยะไกลโดยผ่านทาง HTTP [14]

Hessian ติดต่อกันผ่านทาง HTTP โดยใช้ servlet ประเภทเฉพาะ โดยเมื่อใช้ DispatcherServlet ของ Spring ก็จะสามารถเตรียมการให้ servlet ดังกล่าวทำงานเป็นเซอร์วิสได้ โดยเริ่มจากการกำหนด Servlet ชื่อ remoting ไว้ใน web.xml ดังต่อไปนี้

```
<servlet>
        <servlet-name>remoting</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
        <servlet-name>remoting</servlet-name>
        <url-pattern>/remoting/*</url-pattern>
</servlet-mapping>
```

จากนั้นจึงเตรียมไฟล์ชื่อ remoting-servlet.xml ขึ้นตามชื่อของ servlet ที่กำหนดไว้ใน web.xml เพื่อทำการระบุข้อมูลเกี่ยวกับ HessianServiceExporter โดยจะต้องวางไฟล์ remoting-servlet.xml นี้ไว้ในไดเร็คทอรี่ WEB-INF เช่นเดียวกับไฟล์ web.xml ดังนี้

```
<bean id="accountService" class="example.AccountServiceImpl">
</bean>
<bean name="/AccountService"
class="org.springframework.remoting.caucho.HessianServiceExporter">
    <property name="service" ref="accountService"/>
    <property name="serviceInterface" value="example.AccountService"/>
</bean>
```

เมื่อเชื่อมโยงเซอร์วิสที่เครื่องลูกข่าย Explicit handler mapping จะยังไม่ระบุ เมื่อ Mapping ส่งคำขอ URL ไปยังเซอร์วิส BeanNameUrlHandlerMapping จะถูกใช้ ดังนั้นเซอร์วิสจะถูก Export ที่ URL ที่ระบุชื่อ bean

2.2 Hessian 1.0.2 Specification [14]

เนื่องจากโปรโตคอล Hessian ถูกสร้างมาให้เป็นไบนารี่ที่เบาซึ่งต่างจากโปรโตคอลเว็บเซอร์วิสที่เป็น XML และโปรโตคอล Hessian สามารถใช้สำหรับเซอร์วิส EJB ได้

2.2.1 จุดมุ่งหมายของการออกแบบ

1. รองรับ XML เป็นวัตถุชั้นหนึ่ง
2. ไม่ต้องการการประกาศ interface หรือโครงสร้าง (Schema) จากภายนอก
3. สามารถแปลงวัตถุภาษา Java เป็นข้อมูลได้
4. รองรับการทำงานของ EJB
5. อนุญาตให้เครื่องลูกข่ายที่ไม่ใช่ภาษา Java ทำการใช้เว็บเซอร์วิสได้
6. เว็บเซอร์วิสสามารถใช้งานในรูปแบบของ Java Servlet ได้
7. โปรโตคอลไม่ซับซ้อน สามารถทำการทดสอบได้อย่างมีประสิทธิภาพ
8. มีประสิทธิภาพสูง
9. สามารถใช้งานได้ในบริบทของ Transaction

2.2.2 Serialization

Serialization ของ Hessian มีทั้งหมด 13 แบบ

1. null ซึ่งแทนพ้อยเตอร์ Null และไบท์ N แทนพ้อยเตอร์ Null นั้นๆ ค่าของ null จะอยู่ที่ของ string, xml, binary, list, map หรือ remote

   null ::= N

2. boolean ไบท์ F แทนความเป็นเท็จ และไบท์ T แทนความเป็นจริง

   boolean ::= T
   ::= F

3. int เป็นจำนวนเต็มแบบมีเครื่องหมาย 32 บิท จำนวนเต็มจะแทนตัวไบท์ I ตามด้วยจำนวนเต็ม 4 ไบท์ในลำดับ Big-endian

   int ::= I b32 b24 b16 b8

4. long เป็นจำนวนเต็มแบบมีเครื่องหมาย 64 บิท จำนวนเต็มจะแทนตัวไบท์ L ตามด้วยจำนวนเต็ม 8 ไบท์ในลำดับ Big-endian

   long ::= L b64 b56 b48 b40 b32 b24 b16 b8

5. double เป็นตัวเลขพ้อยเตอร์ทศนิยมแบบ IEEE 64 บิท

   double ::= D b64 b56 b48 b40 b32 b24 b16 b8

6. date แทนโดยมิลลิวินาทีแบบ long 64 บิทตั้งแต่เริ่มคริสตศักราช

   date ::= d b64 b56 b48 b40 b32 b24 b16 b8

7. string เป็นตัวอักษร Unicode 16 บิท เข้ารหัสแบบ UTF-8 string ซึ่งจะถูกเข้ารหัสเป็นกลุ่ม อักขระ S แทนกลุ่มสุดท้าย และ อักขระ s แทนกลุ่มเริ่มต้น แต่ละกลุ่มจะมีค่าของความยาว 16 บิท ส่วนความยาวจะเท่ากับจำนวนของตัวอักษร ซึ่งจะอาจจะแตกต่างกับจำนวนของไบท์

   string ::= (s b16 b8 utf-8-data)* S b16 b8 utf-8-data

8. XML เอกสารXML เข้ารหัสเป็น Unicode 16 บิท ตัวอักษรเข้ารหัสเป็น UTF-8 ข้อมูล XML จะถูกเข้ารหัสเป็นกลุ่ม อักขระ X แทนกลุ่มสุดท้าย และ อักขระ x แทนกลุ่มเริ่มต้น แต่ละกลุ่มจะมีค่าของความยาว 16 บิท ส่วนความยาวจะเท่ากับจำนวนของตัวอักษรซึ่งจะอาจจะแตกต่างกับจำนวนของไบท์

   xml ::= (x b16 b8 utf-8-data)* X b16 b8 utf-8-data

9. binary ข้อมูลไบนารี่จะถูกเข้ารหัสเป็นกลุ่ม อักขระ B แทนกลุ่มสุดท้าย และอักขระ b แทนกลุ่มเริ่มต้น แต่ละกลุ่มจะมีค่าของความยาว 16 บิท

   binary ::= (b b16 b8 binary-data)* B b16 b8 binary-data

10. list เป็นชนิดของ list ที่เรียงลำดับ มีลักษณะคล้ายอะเรย์ list ประกอบไปด้วย ข้อความระบุชนิด ความยาว รายการวัตถุ และปิดท้ายด้วยอักขระ z สำหรับข้อความระบุชนิดนั้นจะต้องเป็นข้อความแบบ UTF-8 ที่สามารถเข้าใจได้ด้วยเซอร์วิส (ซึ่งมักจะเป็นชื่อคลาสของ Java) ความยาวของ list อาจจะมีค่าเป็น -1 เพื่อแสดงว่าไม่จำกัดความยาว

    list ::= V *type*? *length*? *object** z

11. map แทนวัตถุที่ Serialized และ Maps และ type element จะหมายถึงประเภทของ map วัตถุอาจจะแทนได้ด้วย map จากชื่อของฟิลด์ไปยังค่าของมันและ type คือ class ของวัตถุนั้น

map ::= M t b16 b8 type-string (*object*, *object*)* z

12. ref เป็นการอ้างอิงถึง Instance ของ list หรือ map ก่อนหน้า เมื่อ list หรือ map ถูกอ่านจาก Stream จะมีการกำหนดค่าตำแหน่งเป็นจำนวนเต็มใน Stream นั้น เช่น list หรือ map แรกเป็น 0 ถัดไปเป็น 1

ref ::= R b32 b24 b16 b8

13. remote เป็นการอ้างถึงวัตถุระยะไกล remote มี type และ string แบบ UTF-8 แทนวัตถุที่เป็น URL

remote ::= r t b16 b8 type-name S b16 b8 url

2.2.3 การเรียกใช้

การเรียกใช้ Hessian เรียกเมธอดบนวัตถุพร้อมด้วยรายการของอาร์กิวเมนท์ วัตถุจะถูกระบุโดย Container และอาร์กิวเมนท์จะถูกระบุโดย Hessian serialization

call ::= c x01 x00 *header** m b16 b8 method-string (*object*)* z

3. เครื่องจักรเสมือน Java บน .NET

เครื่องจักรเสมือน Java เป็นโปรแกรมสำหรับรัน Java แอพพลิเคชั่นเพื่อให้ตัวแอพพลิเคชั่นสามารถทำงานบนแพล็ตฟอร์มระดับระบบปฏิบัติการใด ๆ ที่สนับสนุนโดย JVM อย่างไรก็ดี เมื่อเราพิจารณา Java และ Microsoft .NET เป็นแพล็ตฟอร์ม

3.1 IKVM

IKVM [15] เป็นเครื่องจักรเสมือน Java บน Microsoft .NET โดยมีความสามารถในการแปลง Java bytecode [17] เป็น CIL บน Microsoft .NET โดยสามารถแปลงได้ทั้งแบบ interpreted และ compiled mode IKVM อาศัยการจัดการแบบสะท้อน (Reflection) ใน .NET เพื่อสร้างรหัส CIL จากไฟล์คลาสของ Java ที่บรรจุ Java bytecode ไว้และทำงานในโหลด interpreter เมื่อใช้คำสั่ง

ikvmc.exe ก็จะสามารถแปลงรหัสจากไฟล์คลาสของ Java ให้เป็นไฟล์ DLL ของ .NET ที่เรียกว่า Assembly ไฟล์ได้

3.2 โครงสร้างของ IKVM

เนื่องจาก IKVM เป็นเครื่องจักรเสมือน Java ประเภทหนึ่ง ดังนั้นโครงสร้างหลักของ IKVM จึงเป็นไปตามมาตรฐานของการสร้างเครื่องจักรเสมือน Java โดยมี IKVM.Runtime.DLL เป็นระบบรันไทม์หลัก ภายใน IKVM.Runtime.DLL จะประกอบไปด้วยตัวอย่างไฟล์ .class ตามมาตรฐาน Java, ตัวคอมไพเลอร์ ซึ่งแปลง Java bytecode เป็น .NET CIL ด้วยกระบวนการ Emitting ตามชุดคำสั่งใน System.Reflection.Emit ของ .NET และ ระบบ Java Native Interface (JNI) เพื่อให้สามารถติดต่อกับชุดคำสั่งภาษาเครื่องตามรูปแบบที่ระบบปฏิบัติการกำหนดไว้ได้ โดยส่วนใหญ่ของ IKVM พัฒนาด้วยภาษา C# บางส่วนพัฒนาด้วย C และภาษา IL ซึ่งเป็นภาษาเครื่องเสมือนสำหรับ .NET

3.3 GNU Classpath

เนื่องจาก IKVM เป็นเครื่องจักรเสมือนเช่นเดียวกับ Java Runtime Environment (JRE) ของ Sun ดังนั้นจึงจำเป็นต้องมีไลบรารี่ที่สมบูรณ์เพื่อใช้รัน Java แอพพลิเคชั่น และ IKVM ก็ใช้ GNU Classpath [18] เป็นไลบรารี่

อย่างไรก็ตาม GNU Classpath ยังไม่มีความเข้ากันได้อย่างสมบูรณ์กับรหัสที่คอมไพล์ด้วยคอมไพเลอร์ของ IKVM และไลบรารี่สำหรับการเรียกใช้เมธอดระยะไกลอย่าง Hessian [14] จึงจำเป็นต้องมีการแก้ไข GNU Classpath ในส่วนของการติดต่อผ่านโปรโตคอล HTTP เพื่อให้การทำงานของ Hessian บน IKVM เป็นไปด้วยความราบรื่น

4. ความไม่เข้ากันของวัตถุ Java และ .NET

ในส่วนนี้จะกล่าวถึงความไม่เข้ากัน (Incompatibility) ของวัตถุบนฝั่ง Java และ .NET ซึ่งโครงสร้างโดยรวมเชิงสถาปัตยกรรมของทั้งสองระบบนั้นจะมีความคล้ายคลึงกัน อย่างไรก็ตามลักษณะของระบบชนิดข้อมูลและวัตถุในระดับรายละเอียด มีความแตกต่างกันระดับหนึ่ง

คุณสมบัติของวัตถุจะประกอบไปด้วยตัวรับและตัวส่งข้อมูล ใน Java จะใช้เมธอดที่ชื่อขึ้นต้นด้วย set แทนตัวรับและชื่อขึ้นต้นด้วย get แทนตัวส่ง ใน .NET จะมีการกำหนดตัวรับและตัวส่งด้วยคำสงวน get และ set

ตัวอย่างคลาส Person ในภาษา Java เป็นดังนี้

```
class Person {
 private String name;
 public String getName() { return name; }
 public void setName(String value) { name = value; }
}
```

ใน ตัวอย่างภาษา Java class Person จะมีฟิลด์ชื่อ name และมีคุณสมบัติตามข้อกำหนดของ Java Bean ชื่อ name เช่นเดียวกัน โดยคุณสมบัติ name มีตัวรับและตัวส่งข้อมูลคือเมธอด setName และ getName ตามลำดับ

สำหรับตัวอย่างคลาส Person ในภาษา C# เป็นดังนี้

```
class Person {
 private String name;
 public String Name {
    get { return name; }
    set { name = value;}
 }
}
```

ในตัวอย่างภาษา C# การประกาศคุณสมบัติของคลาสทำได้โดยใช้บล็อก { } และคีย์เวิร์ด get และ set โดยคลาส Person ในภาษา C# มีคุณสมบัติ Name ในลักษณะเดียวกันกับคลาส Person ของภาษา Java โดยมีข้อสังเกตคือ คุณสมบัติของภาษา Java จะมีคำแนะนำตามข้อกำหนดให้เริ่มด้วยตัวเล็ก แต่ในภาษา C# และ .NET Framework นั้นไม่มีเงื่อนไขดังกล่าว

ลักษณะภายในของคุณสมบัติ Name ของ C# เมื่อคอมไพล์เป็นภาษา CIL ของ .NET Framework มีดังนี้

```
.method public hidebysig specialname instance string
      get_Name() cil managed
{
 // Code size       12 (0xc)
 .maxstack  1
 .locals init ([0] string CS$1$0000)
 IL_0000:  nop
 IL_0001:  ldarg.0
 IL_0002:  ldfld      string Person::name
 IL_0007:  stloc.0
 IL_0008:  br.s       IL_000a
 IL_000a:  ldloc.0
 IL_000b:  ret
} // end of method Person::get_Name

.method public hidebysig specialname instance void
      set_Name(string 'value') cil managed
{
 // Code size       9 (0x9)
 .maxstack  8
 IL_0000:  nop
 IL_0001:  ldarg.0
 IL_0002:  ldarg.1
 IL_0003:  stfld      string Person::name
 IL_0008:  ret
} // end of method Person::set_Name
```

เมื่อ คอมไพล์โปรแกรมของ C# แล้วจะได้คลาสในลักษณะ binary เป็นภาษา CIL ที่แสดงไว้ข้างต้น โดยจะสังเกตได้ว่าคุณสมบัติ Name นั้นแปลงเป็นเมธอดสองเมธอด คือ get_Name และ set_Name ซึ่งเป็นเมธอดพิเศษที่ไม่สามารถเข้าถึงได้ตามปกติเพื่อใช้เป็นตัวตั้งค่าและส่งค่าของคุณสมบัติ Name เท่านั้น ใน .NET Framework ใช้เครื่องหมาย _ คั่นระหว่าง get และชื่อคุณสมบัติในการตั้งชื่อเมธอดคู่นี้ จึงเป็นเหตุผลให้สามารถข้ามข้อจำกัดที่กำหนดไว้ในภาษา Java เรื่องชื่อคุณสมบัติต้องเริ่มต้นด้วยอักษรตัวเล็กได้ เมื่อคอมไพล์เป็นภาษา CIL แล้วจะสามารถสังเกตได้ว่ามีคำสั่ง ldfld (Load Field) ในเมธอด get_Name และ stfld(Set Field) ในเมธอด set_Name เพื่อทำการส่งและรับค่ากับฟิลด์ name ของคลาส Person ซึ่งเป็นตัวแปรประเภท private ที่ใช้เก็บค่าตามการประกาศไว้

เมื่อต้องการนำวัตถุฝั่ง Java มาใช้งานบน .NET จึงมีความจำเป็นต้องทำการจำลองตัวรับและตัวส่งของแต่ละคุณสมบัติให้ตรงกัน งานวิจัยชิ้นนี้แก้ปัญหาดังกล่าวด้วยการใช้ข้อมูลไทพ์เชิงพลวัตร (Dynamic Type Information) ซึ่งสามารถใช้จำลองคุณสมบัติของวัตถุฝั่ง Java ผ่านกลไกการสะท้อน (Reflection)

5. การสะท้อนใน .NET

การเข้าถึงข้อมูลของวัตถุด้วยวิธีการ Reflection หรือการสะท้อน [19] เป็นกลไกการเข้าถึงสมาชิกของคลาสจากภายนอกโดยใช้สายอักขระเป็นตัวอ้างถึง สมาชิก จากตัวอย่างคลาส Person ในภาษา C# เมื่อสร้างวัตถุของคลาส Person ในภาษา C# บน .NET ด้วยคำสั่งต่อไปนี้

```
Person p = new Person()
```

จะได้วัตถุ p ของคลาส Person จากนั้นเราสามารถเรียกใช้เมธอด GetType บนวัตถุ p เพื่อดึงข้อมูลไทพ์ออกมาใช้อ้างอิงได้ เช่น

```
Type personType = p.GetType()
```

ตัวแปรวัตถุ personType เป็นตัวแทนไทพ์ของคลาส Person ที่สามารถใช้ในการประมวลผลการสะท้อนได้ และเมื่อใช้คำสั่ง

```
MethodInfo[] methods = personType.GetMethods();
```

จะได้อะเรย์ของข้อมูลเมธอด (MethodInfo) ของคลาส Person นั่นคือตามข้อกำหนดของภาษา Java ที่กำหนดไว้ว่าคุณสมบัติในภาษา Java จะประกอบด้วยคู่เมธอด get และ set เช่น คุณสมบัติ name จะประกอบไปด้วยคู่เมธอด getName และ setName ด้วยวิธีการดึงข้อมูลการสะท้อนในลักษณะนี้จะสามารถนำไปสร้างคุณสมบัติเชิงพลวัต

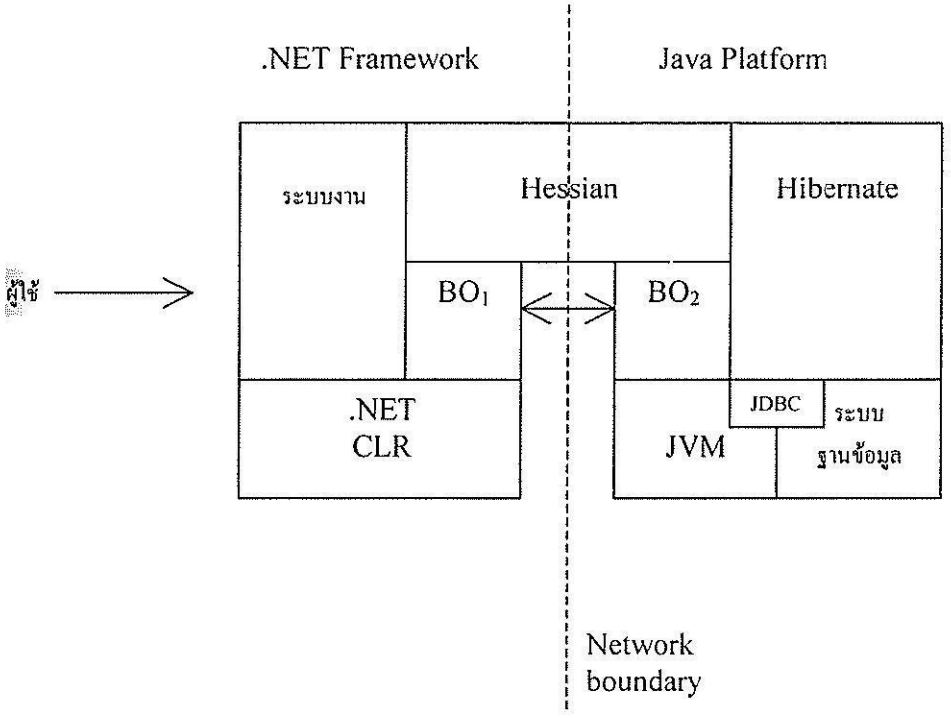
บทที่ 4

ระบบงาน การทดลองและวัดประสิทธิภาพ

ในบทนี้จะกล่าวถึง Galvanium Framework ซึ่งเป็นเฟรมเวิร์คที่ผู้วิจัยพัฒนาขึ้นเพื่อใช้ในการเชื่อมต่อข้อมูลสำหรับลูกข่ายแบบบางของระบบกลุ่มแม่ข่ายโปรแกรมประยุกต์ และเป็นเฟรมเวิร์คบน Microsoft .NET ที่ใช้ IKVM ในการทำงานร่วมกับรหัสภาษา Java บนแพล็ตฟอร์ม .NET framework โดยผ่านการเรียกใช้เมธอดระยะไกลร่วมกับกลไกการทำงานร่วมกันระหว่างแพล็ตฟอร์ม

1. Galvanium Framework

องค์ประกอบของ Galvanium Framework สามารถอธิบายได้ด้วยแผนภาพดังนี้



รูปที่ 6. องค์ประกอบของ Galvanium Framework

จากรูปตัวแสดงผลของระบบงานจะอยู่บนฝั่งแพล็ตฟอร์ม ในขณะที่บางส่วนของระบบงาน BO_1 และ BO_2 เป็นวัตถุธุรกิจ (Business Objects) ที่เป็นโปรแกรมชุดเดียวกันแต่คอมไพล์เป็นรหัสสองแบบ

ที่ต่างกันคือ รหัสสำหรับแพล็ตฟอร์ม .Net และรหัสสำหรับ Java ทางด้านซ้ายของ Network Boundary จะเป็นลูกข่ายสำหรับด้านขวาจะเป็นแม่ข่าย ซึ่งทั้งสองฝั่งเชื่อมต่อกันด้วย Hessian library

ส่วนประกอบของ Galvanium Framework

- .NET Client Proxy Factory สำหรับให้เครื่องแม่ข่ายและลูกข่ายติดต่อกันผ่าน Hessian Protocol

  class Suranaree.Galvanium.ClientProxyFactory
- .NET Bean Property Emulator สำหรับทำการโยงข้อมูลของ Java Bean ที่แปลงเป็นวัตถุสำหรับ .NET แล้วกับตัวแสดงผลบน .NET

  class Suranaree.Galvanium.BeanDescriptor (วัตถุเดี่ยว)

  class Suranaree.Galvanium.BeanCollection (รายการวัตถุ)

2. เครื่องมือในการพัฒนา

Galvanium Framework ถูกพัฒนาขึ้นเพื่อทำให้การจัดการข้อมูลบน .NET platform ที่ส่งมาจาก Java ทำได้ง่ายขึ้น ซึ่งใช้เครื่องมือหลายตัวช่วยในการทำงาน ดังต่อไปนี้

- Eclipse Development Platform สำหรับสนับสนุนการพัฒนา
- JBoss IDE สำหรับสนับสนุนการพัฒนา
- JBoss Application Server สำหรับใช้เป็นโปรแกรมแม่ข่าย
- Hibernate สำหรับแมพวัตถุไปยังข้อมูลเชิงสัมพันธ์
- Hessian สำหรับใช้เป็นตัวถ่ายโอนวัตถุข้ามเครือข่าย
- IKVM สำหรับแปลงรหัส Java เป็นรหัส .NET

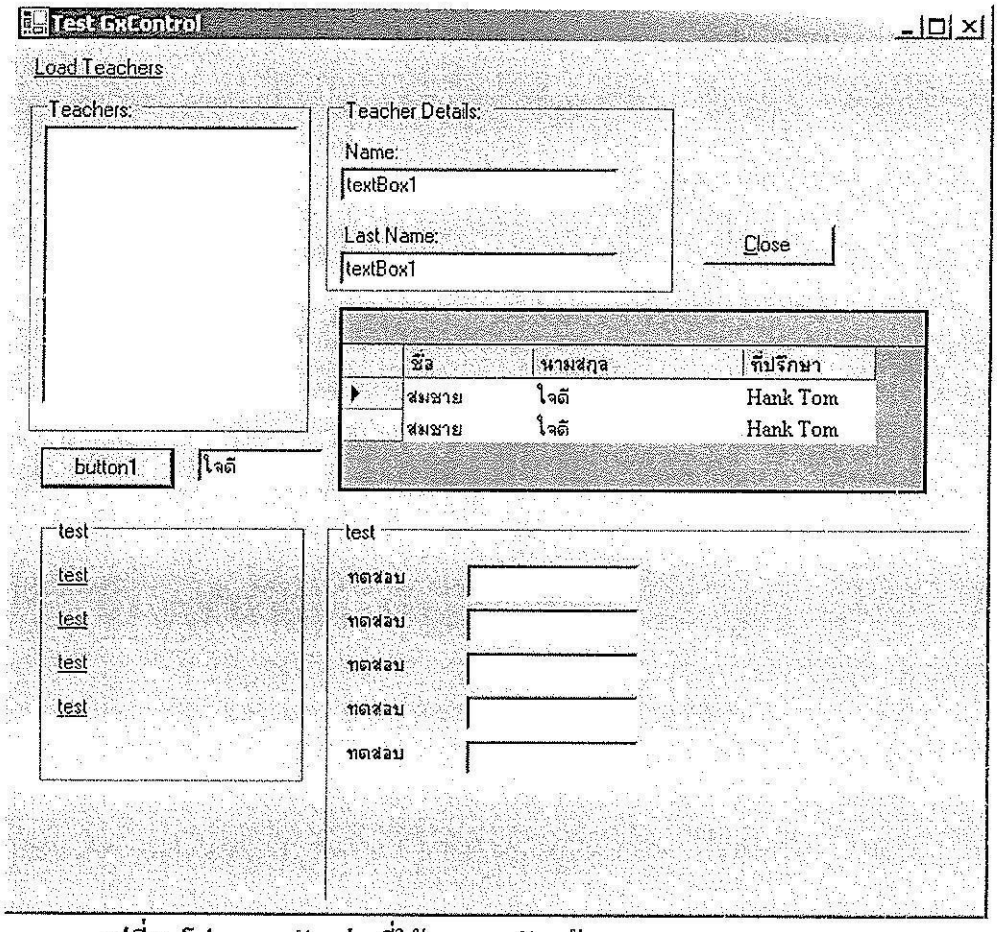
3. การใช้งาน

สำหรับในกระบวนการพัฒนาซอฟต์แวร์จะพิจารณาระบบงานเป็นแบบหลายชั้น (Multi-tier) โดยมีฐานข้อมูลเชิงสัมพันธ์เชื่อมต่อผ่านตัวเชื่อมต่อข้อมูลของ Java (JDBC) และมีชั้นของการแมพวัตถุไปยังข้อมูลเชิงสัมพันธ์ (Object/Relational Mapping – ORM) ด้วย Hibernate บนฝั่งแม่ข่าย โดยตรรกะทางธุรกิจ (Business logic) จะถูกเตรียมให้อยู่ในรูปแบบของ Session Bean ทั้งชนิดมีสถานะ (Stateful) และไร้สถานะ (Stateless) จากนั้นในอีกระดับชั้นจะมี Servlet ทำหน้าที่ส่งออกตัวบริการธุรกิจเพื่อให้สามารถเรียกใช้ระยะไกลผ่านตัวแปลงวัตถุของ Hessian ได้ ในขณะเดียวกันเมื่อแปลง

วัตถุธุรกิจที่เตรียมไว้จากการใช้ Hibernate (ได้จากแมพวัตถุฝั่ง Java ไปยังฐานข้อมูลเชิงสัมพันธ์ตามของตกลงของ Hibernate) ไปเป็นรหัส .NET ซึ่งการแปลงนี้ใช้ IKVM เป็นตัว compiler

เมื่อเตรียมการในระดับชั้นวัตถุธุรกิจเรียบร้อยแล้ว จะสามารถนำวัตถุธุรกิจดังกล่าวในรูปแบบของ .NET Assembly ใส่เข้าไปในโครงงานทางฝั่ง .NET ได้ โดยจะสามารถใช้ Galvanium Framework สร้าง client proxy เพื่อเรียกบริการของฝั่ง Java ผ่าน Hessian Client เมื่อได้ผลลัพธ์จากบริการมาในรูปแบบที่เป็นวัตถุธุรกิจแล้ว (ทั้งแบบ List หรือ Array หรือ Object ก็ตาม) จะสามารถใช้ BeanCollection (กรณีที่เป็นรายการวัตถุ) หรือ BeanDescriptor (กรณีที่เป็นวัตถุเดี่ยว) ครอบเพื่อจำลองคุณสมบัติแบบ Java Bean ไปเป็นคุณสมบัติแบบ .NET และนำไปโยงเข้ากับตัวแสดงผลใด ๆ ของ .NET ได้ตามต้องการ

ตัวอย่างการแสดงผลใน DataGrid



รูปที่ 7. โปรแกรมตัวอย่างที่ใช้แสดงผลวัตถุด้วย Galvanium Framework

ตัวอย่าง code

```
private void btbutton2_Click(object sender, System.EventArgs e)
{
        Student s = new Student();
        s.setFirstname("สมชาย");
        s.setLastname("ใจดี");
        s.setId(new java.lang.Integer(1));
        s.setAdvisor(
                new Teacher(new java.lang.Integer(1), "Tom","Hank")
        );
        Student[] ss = new Student[2];
        ss[0] = s;
        ss[1] = s;
        bc = new BeanCollection(ss, typeof(Student));

        dataGrid1.DataSource = bc;
        txtTestProp.DataBindings.Add("Text", bc, "Lastname");
}
```

4. การทดลอง

งานวิจัยชิ้นนี้เป็นการสร้างกระบวนการโยงข้อมูลแบบใหม่ที่สามารถให้ตัวแสดงผล .NET ไม่ว่าจะเป็น TextBox, ListBox หรือ DataGrid เป็นต้น สามารถโยงเข้ากับวัตถุที่สร้างมาจากภาษา Java ได้ ซึ่งวัตถุดังกล่าวจะต้องผ่านการแปลงจาก Java bytecode มาเป็น .NET IL ด้วยตัวแปลง IKVM มาแล้ว

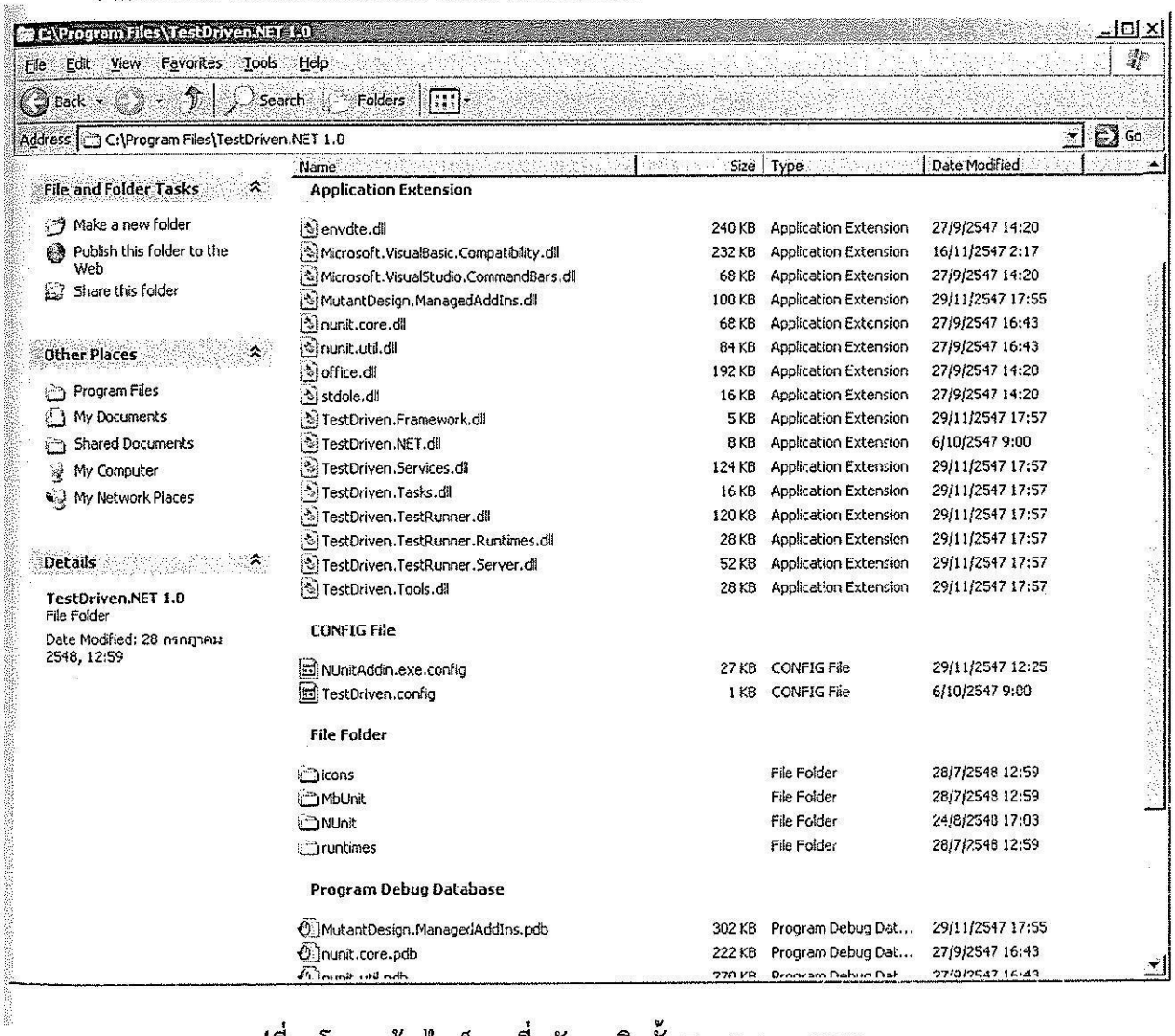
แนวทางการทดลองจะทำเพื่อทดสอบความแตกต่างของประสิทธิภาพ GUI ที่ใช้ในการแสดงผลและแก้ไขข้อมูล สำหรับวัตถุปกติของ .NET เทียบกับวัตถุที่แปลงมาจาก Java นั่นคือจะมีการจับเวลาโดยตัวจับเวลาเราจะใช้ TestDriven.NET และ AQTime ช่วยทำการรันโปรแกรมทดสอบและนำค่าเวลาการรันที่แจ้งออกมาเก็บเป็นข้อมูล และต้องมีการทดลองโดยใช้การวนรอบหลาย ๆ ครั้ง เพื่อให้ได้ค่าที่น่าเชื่อถือ

4. 1 การติดตั้งระบบ

1. ติดตั้ง Visual Studio.NET 2003 และทำการลง Visual C# .NET
2. download โปรแกรมที่จำเป็นต้องใช้ในการทดสอบระบบ มีดังต่อไปนี้

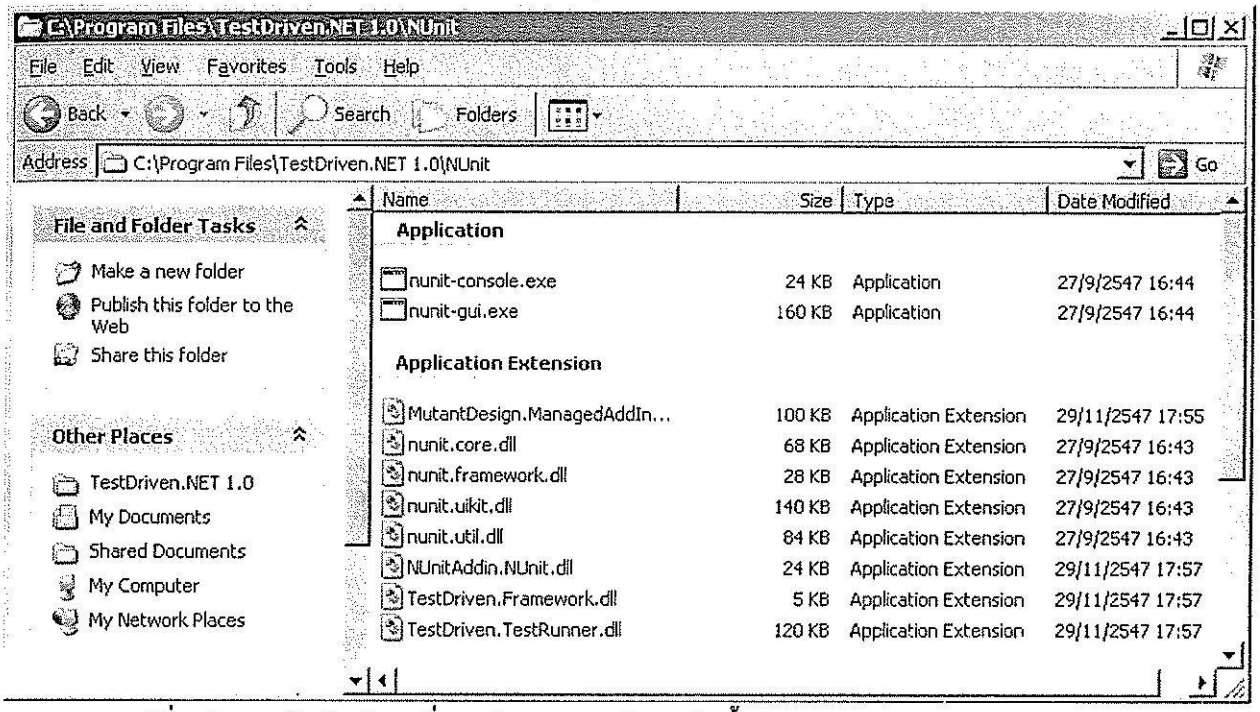
   TestDriven.NET-1.0.915d.zip

   ObjectViews.zip
3. ติดตั้ง TestDriven.NET v. 1.0 โดยจะทำการติดตั้ง NUnit ลงไปในระบบด้วย ถ้าตัวติดตั้งถามเกี่ยวกับการติดตั้ง NUnit ให้ตอบตกลง



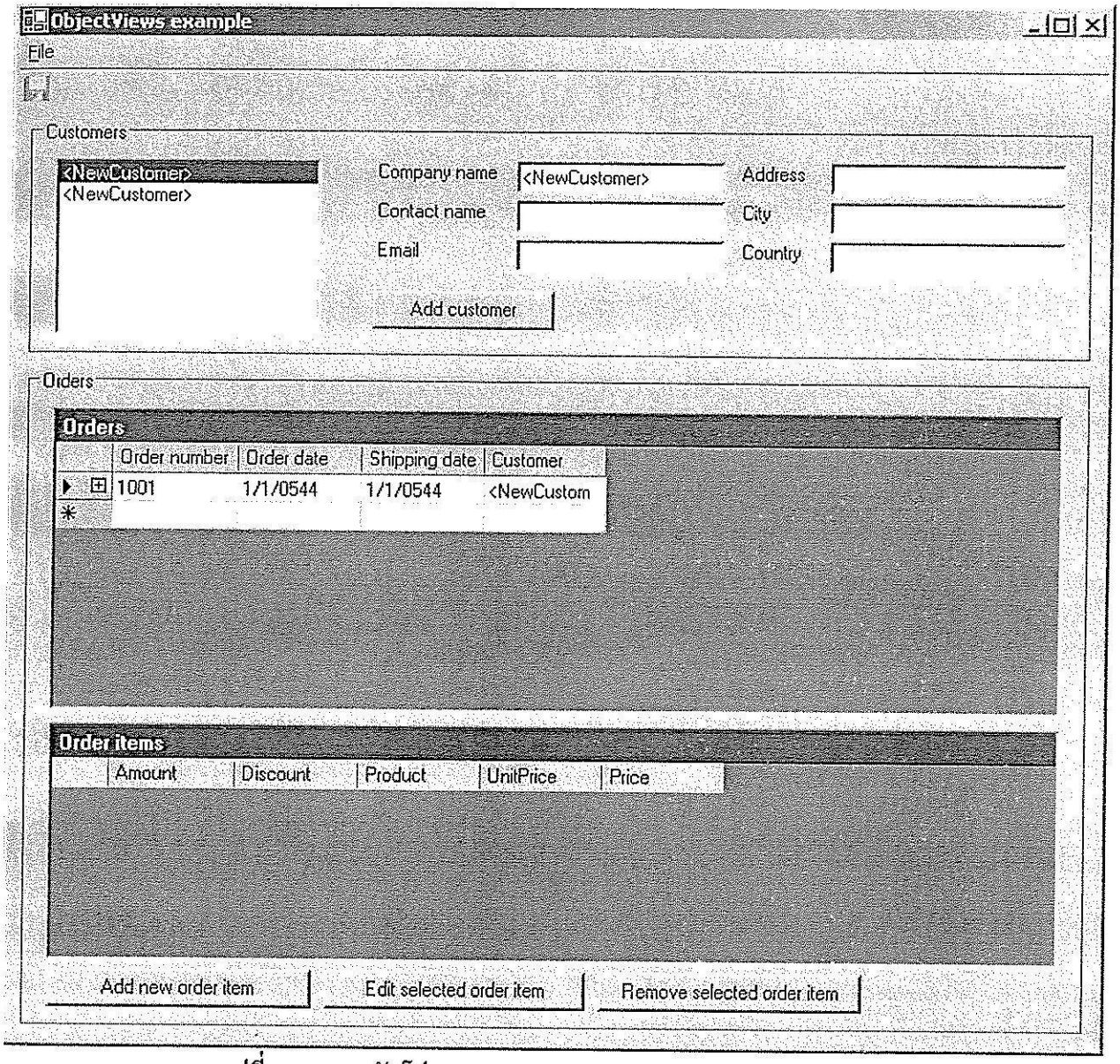
รูปที่ 8. โครงสร้างไดเร็คทอรี่หลังการติดตั้ง TestDriven.NET

ถ้าติดตั้งถูกต้องจะได้ไดเร็คทอรี่โครงสร้างตามรูปที่ 8 และในไดเร็คทอรี่ย่อย NUnit จะมีไฟล์จำนวนหนึ่งอยู่ ตามรูปที่ 9



รูปที่ 9. โครงสร้างไดเร็คทอรี่สำหรับ Nunit หลังการติดตั้ง TestDriven.NET

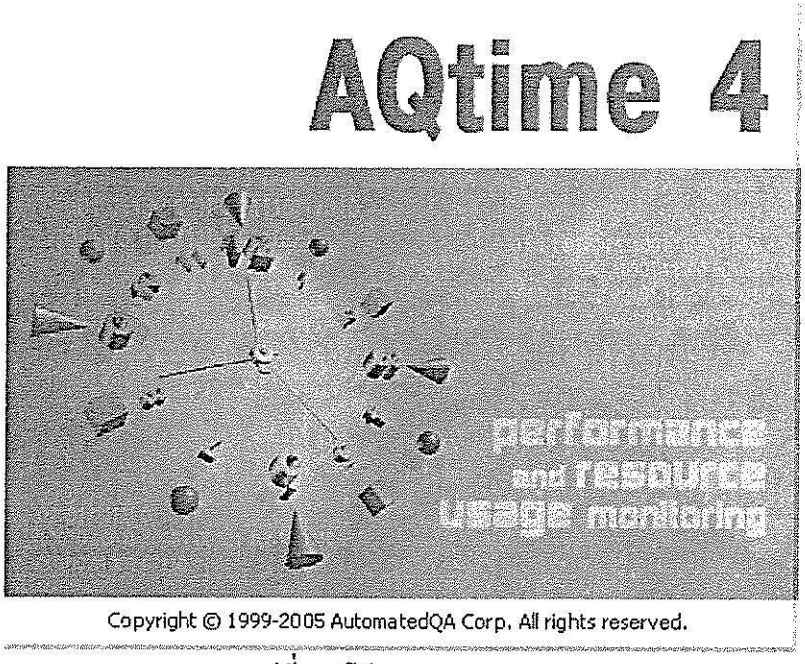
4. Extract ไฟล์จาก ObjectViews.zip ไว้ในไดเร็คทอรี่ เช่น c:\workspace\objectviews
5. เปิดไฟล์ ObjectViews.sln ใน c:\workspace\objectviews ด้วย VS.NET
6. ทดลองรันโปรเจ็คย่อยชื่อ ObjectViews.Example ดู ถ้ารันได้ถูกต้องแสดงว่าติดตั้ง ObjectViews ถูกต้องแล้ว ถ้ารันไม่ได้ ให้ตั้งค่า Reference ไปยังโปรเจ็คย่อยชื่อ ObjectViews แล้วทดลองรันดูอีกครั้ง ผลการรันจะได้โปรแกรมดังรูปที่ 10 แสดงขึ้นมา



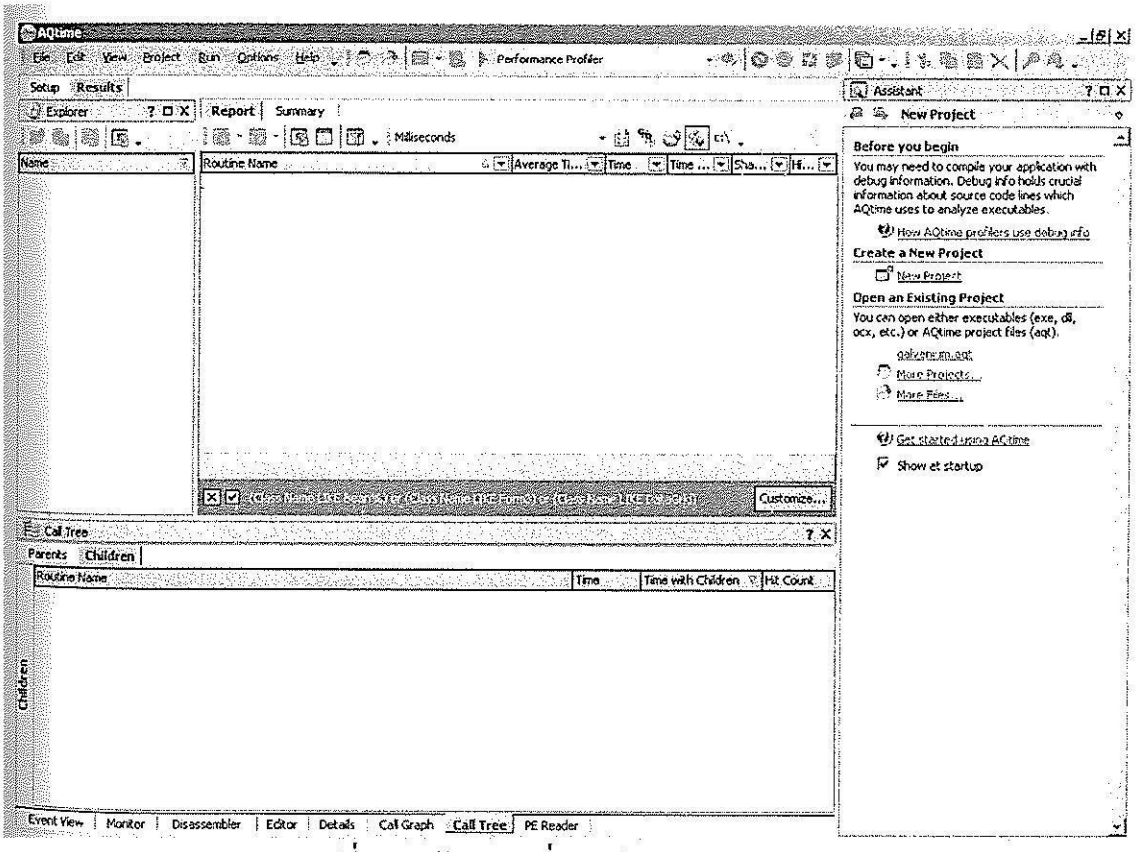
รูปที่ 10. ผลการรันโปรแกรม ObjectViews.Example

4.2 การวัดประสิทธิภาพ

การทดสอบและวัดประสิทธิภาพของ Galvanium Framework จะใช้ AQTime ในการวัดเวลาการทำงานของเฟรมเวิร์ค

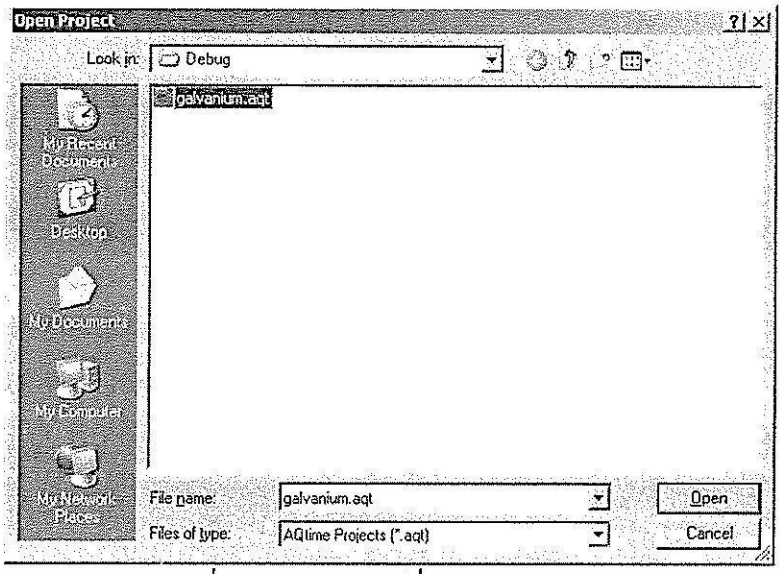


รูปที่ 11. โปรแกรม AQTime

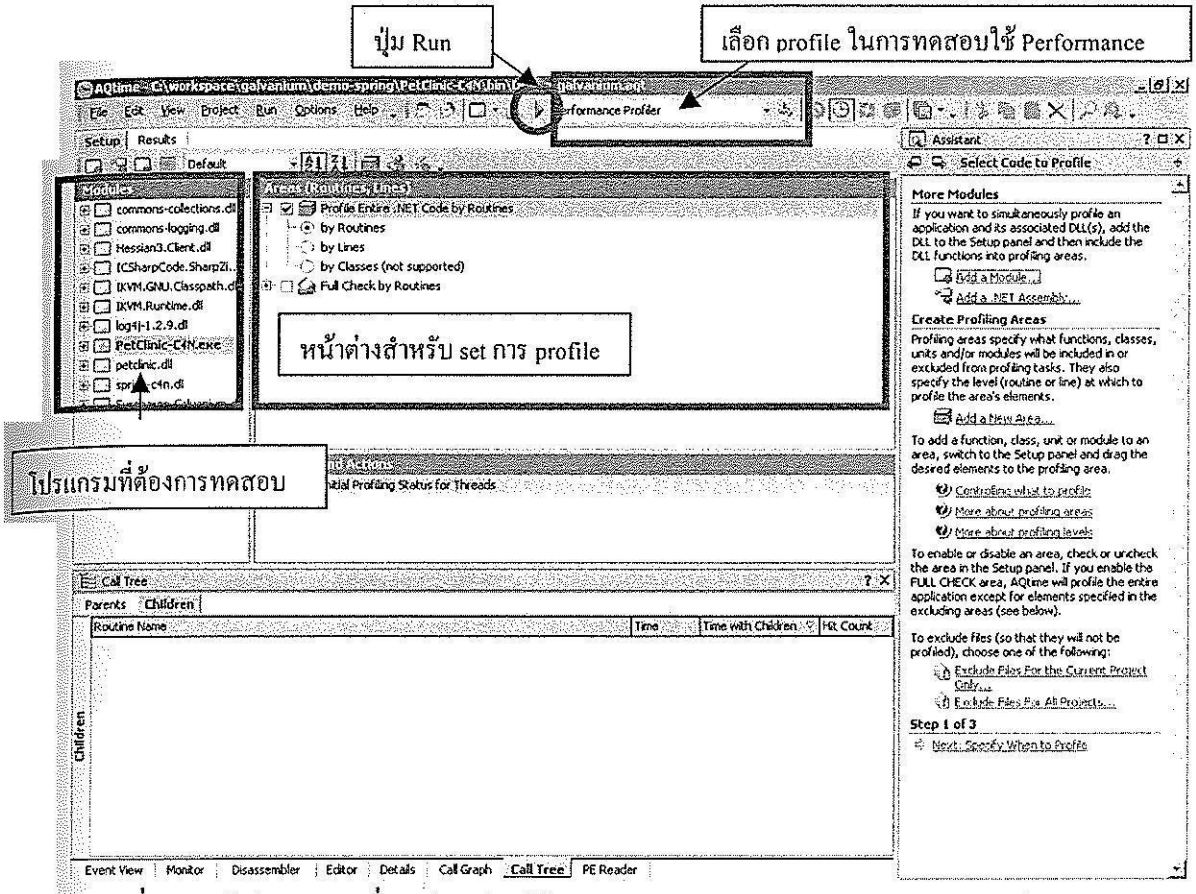


รูปที่ 12. หน้าจอแรกเริ่มของโปรแกรม AQTime

เมื่อเปิดโปรแกรม AQTime เรียบร้อยแล้วให้เลือกเมนู File | Open Project แล้วเลือกไฟล์ชื่อ galvanium.aqt ตามรูปที่ 13 เพื่อเปิดหน้าต่างการทำงานที่เตรียมไว้สำหรับวัดประสิทธิภาพของ Galvanium Framework สำหรับไฟล์ galvanium.aqt จะอยู่ในไดเร็คทอรี่ C:\workspace\galvanium\demo-spring\PetClinic-C4N\bin\Debug

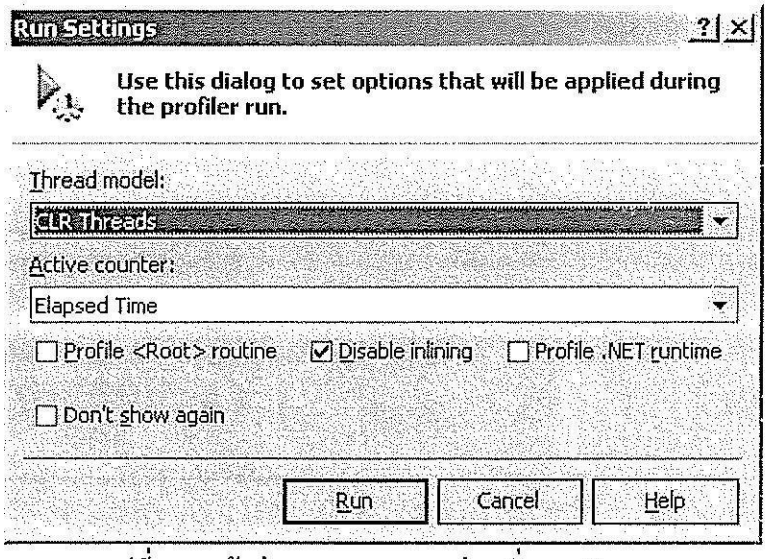


รูปที่ 13. การเลือกไฟล์ชื่อ galvanium.aqt



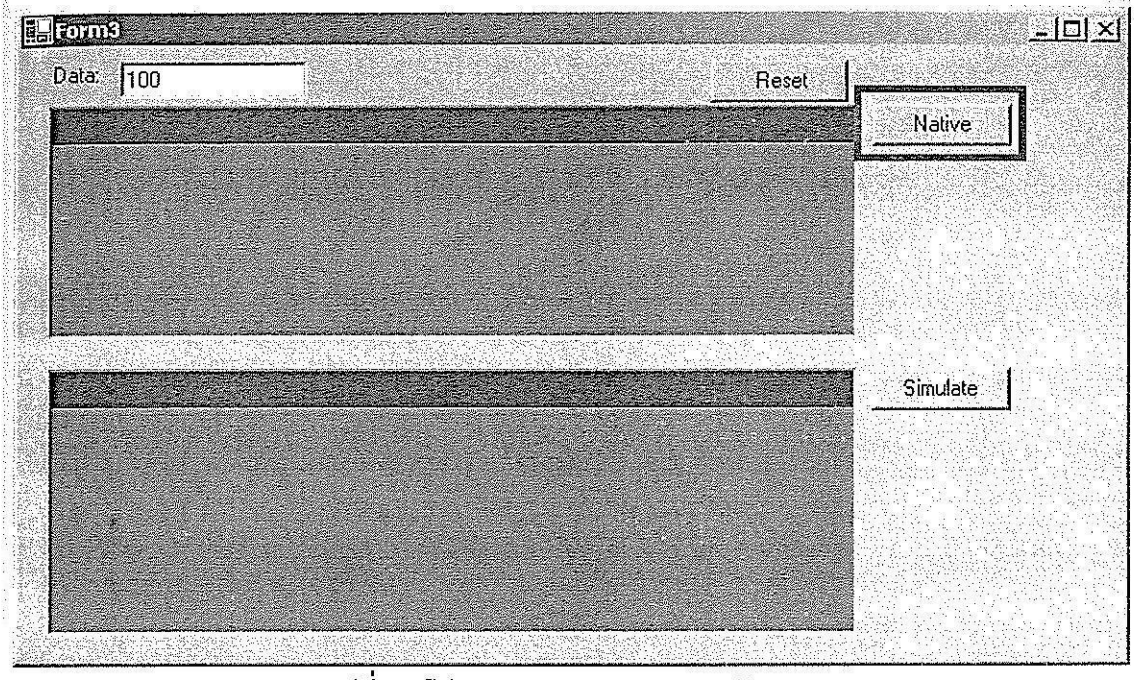
รูปที่ 14. หน้าต่างทำงานที่บันทึกเตรียมไว้สำหรับทดสอบ Galvanium Framework (ก่อนทดสอบ)

เมื่อต้องการเริ่มทดสอบให้กดปุ่ม  จากนั้นโปรแกรม AQTime จะแสดงหน้าต่าง Run Settings ขึ้นตามรูปที่ 15



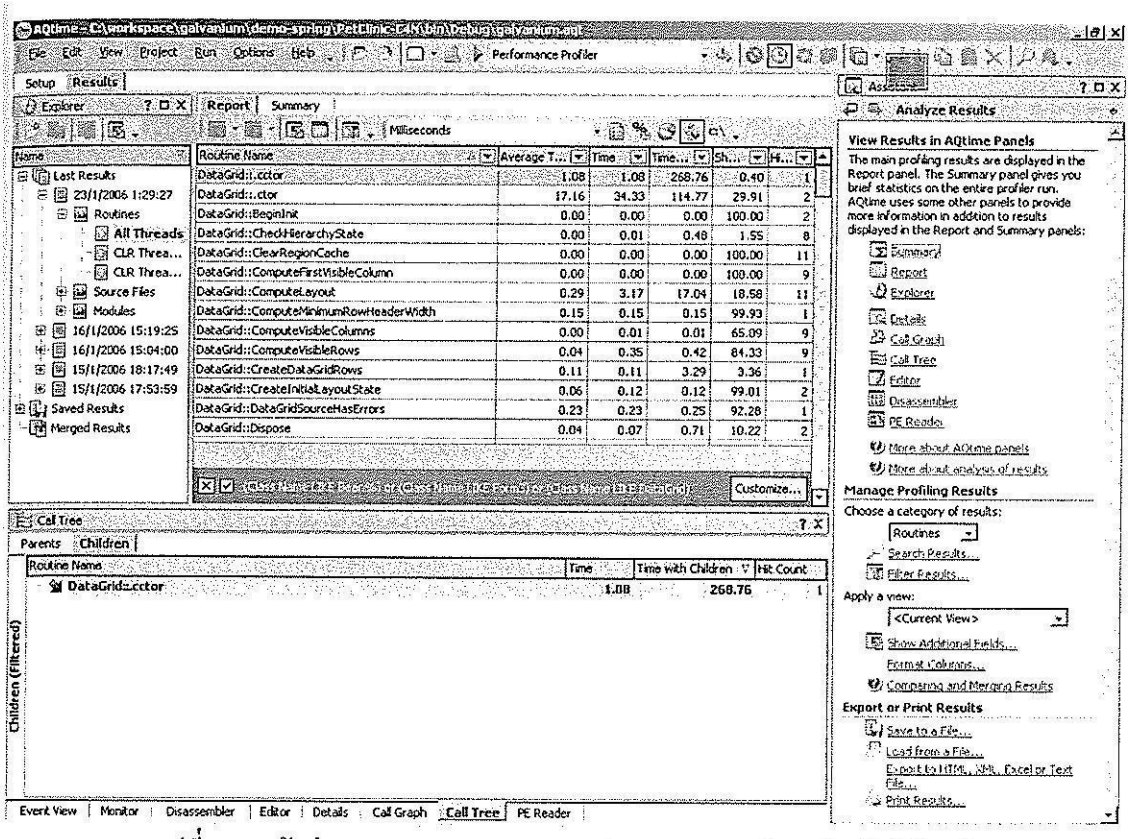
รูปที่ 15. หน้าต่าง Run Settings ก่อนเริ่มการจับเวลา

กดปุ่ม Run เพื่อเริ่มการทดสอบจะได้ตามรูปที่ 16

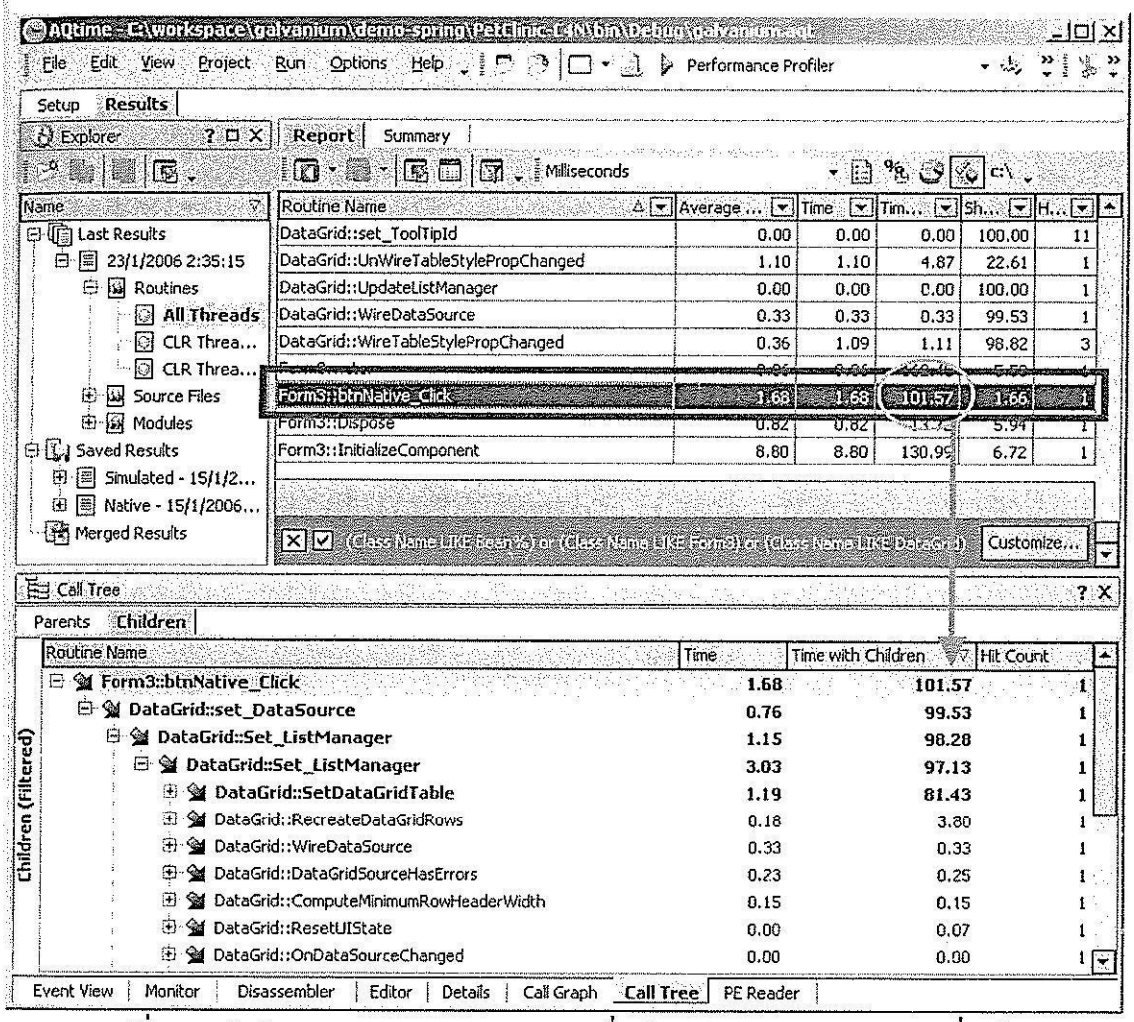


รูปที่ 16. โปรแกรมสำหรับการทดลองจับเวลา

หน้าจอโปรแกรมต้นแบบสำหรับทดสอบ Galvanium Framework กดปุ่ม Native 1 ครั้งเพื่อทำการทดสอบสำหรับ Data ขนาด 100 แถว จากนั้นปิดโปรแกรม จะได้ผลลัพธ์แสดงตามรูปที่ 12 โดยในการทดสอบจะทำการวัดเวลาการทำงานของระบบการโยงข้อมูลเดิมของ .NET (Native) เทียบกับ Galvanium Framework (Simulate) โดยใช้ข้อมูลขนาด 10,000, 20,000, 30,000, 40,000, 50,000 และ 60,000 ตามลำดับ



รูปที่ 17. หน้าต่าง AQTime แสดงผลการทำงานและเวลาในหน่วยมิลลิวินาที



รูปที่ 18. หน้าต่าง AQTime แสดงผลการรัน เมื่อเลือกดู Call Tree ของเมธอดที่สนใจ

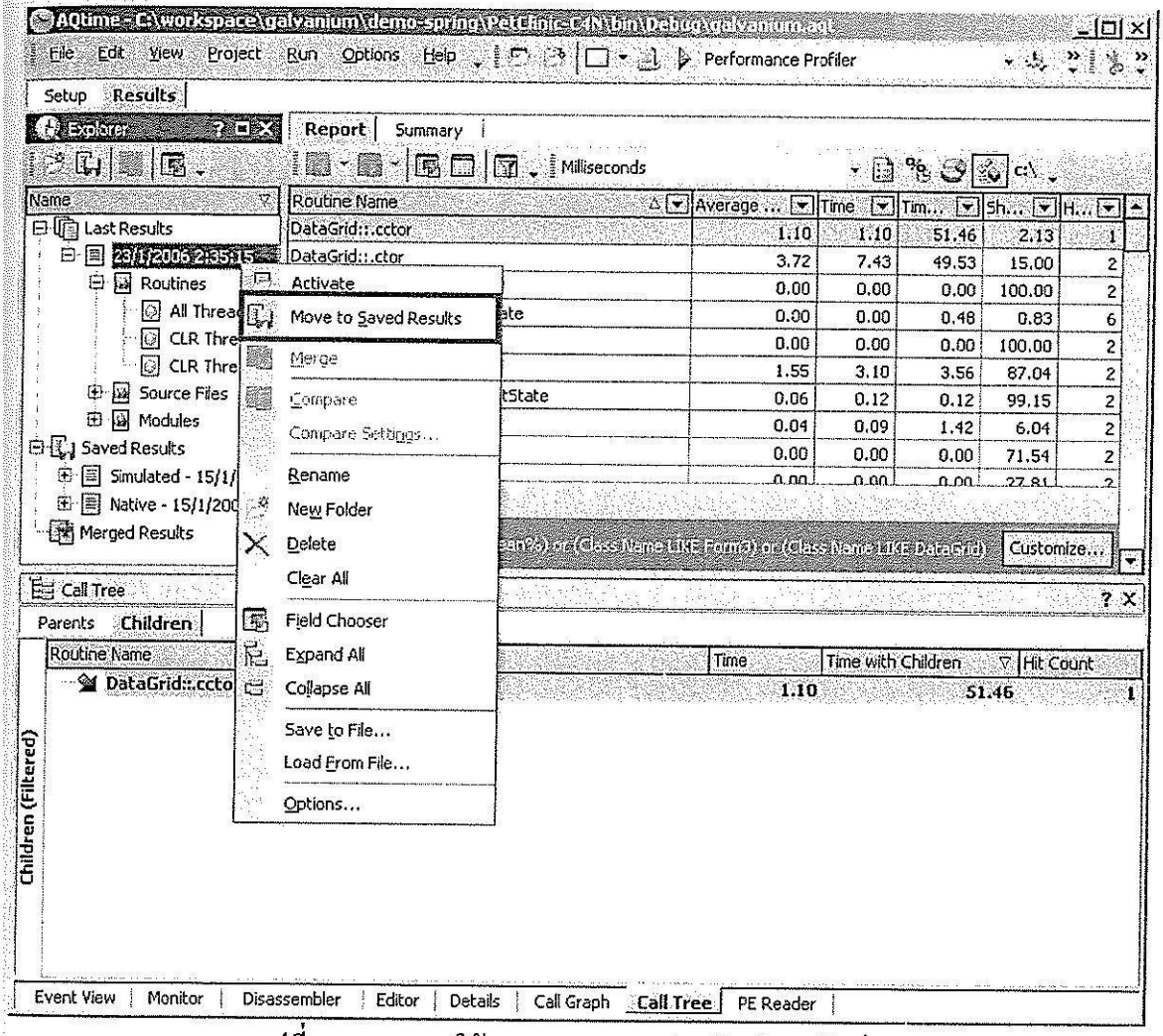
ตัวอย่างในกรอบสีแดงของรูปที่ 18 คือผลลัพธ์ของการรัน และกดปุ่ม Native เวลารวมคือ 101.57 มิลลิวินาที ซึ่งจะตรงกับ Call Tree ใน tab Children ด้านล่าง ส่วนของหน้าต่างตรง Tab ด้านล่างจะบอกรายละเอียดเวลาการทำงานเป็นชั้น ๆ ซ้อนลงไป โดยค่าที่สนใจคือ

1. Form3::btnNative_click
2. DataGrid::set_DataSource

สำหรับการทดลองของปุ่ม Simulate ก็เป็นเช่นเดียวกัน โดยค่าที่สนใจคือ

1. Form3::btnSimulate_click
2. DataGrid::set_DataSource

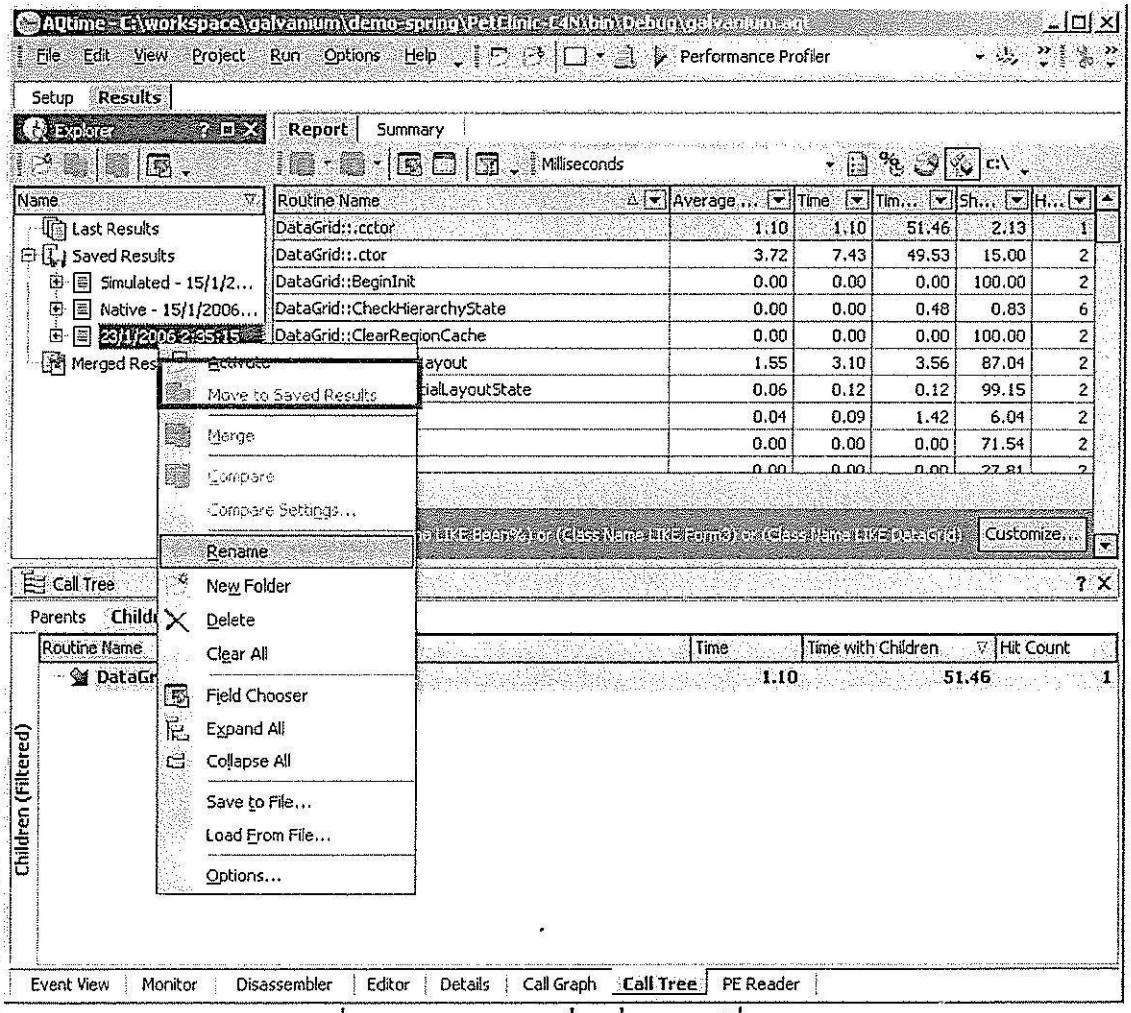
วิธีการเก็บผลลัพธ์



รูปที่ 19. แสดงการใช้ Context Menu สำหรับเก็บผลลัพธ์

คลิกขวา และเลือก Move to Saved Results เพื่อทำการเก็บผลลัพธ์เฉพาะที่ใช้อ้างอิง ตามรูปที่ 19

วิธีการตั้งชื่อการเก็บผลลัพธ์



รูปที่ 20. การเลือกและเปลี่ยนชื่อผลลัพธ์ที่เก็บแล้ว

จากนั้นทำการเปลี่ยนชื่อผลลัพธ์ที่ย้ายมาตามรูปที่ 20 ให้เป็นไปตามการตั้งชื่อต่อไปนี้

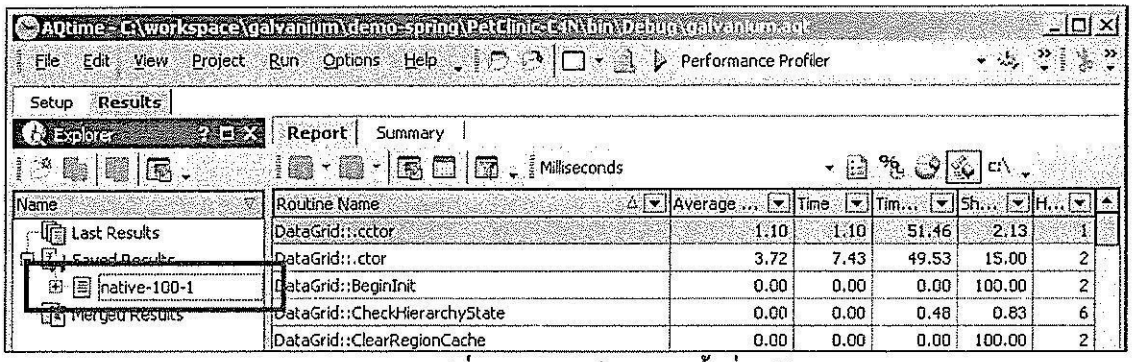
"วิธีการ-ขนาดข้อมูล-ครั้งที่ทดสอบ"

เช่น

native-1000-3 หมายถึงการทดสอบแบบ native ด้วย data size = 1,000 และเป็นการทดสอบครั้งที่ 3

simulate-20000-5 หมายถึงการทดสอบแบบ simulate ด้วย data size 20,000 และเป็นการทดสอบครั้งที่ 5

ซึ่งจะได้ตามรูปที่ 21



รูปที่ 21. ผลลัพธ์หลังการตั้งชื่อแล้ว

บทที่ 5

ผลการทดลอง

การทดลองพัฒนาระบบบนระบบปฏิบัติการ Windows XP โดยใช้ .NET framework 2.0 เป็นลูกข่าย เชื่อมต่อผ่านโปรโตคอล Hessian ไปยังแม่ข่ายที่มีระบบงานรันอยู่บน JBoss 4 และระบบปฏิบัติการ Linux Redhat

1. ผลการทดสอบการสร้างลูกข่ายแบบบาง

เมื่อสร้างระบบลูกข่ายเพื่อติดต่อไปยังแม่ข่ายผ่านโปรโตคอล Hessian แล้ว ปรากฏว่าสามารถเชื่อมต่อและถ่ายข้อมูลจากแม่ข่ายมายังลูกข่ายได้ถูกต้อง

2. ผลการทดลองการวัดประสิทธิภาพ

สร้างระบบทดสอบประสิทธิภาพเทียบการทำงานการโยงข้อมูลของวัตถุ .NET และวัตถุ Java Bean ผ่าน Galvanium Framework เพื่อวัดเวลาการทำงานของการตั้งค่า DataSource ของตัวแสดงผล DataGrid และเวลารวมของการกดปุ่มเพื่อตั้งค่า DataSource ซึ่งการทดลองจะทำการวัดเวลาการทำงานของระบบการโยงข้อมูลเดิมของ .NET (Native) เทียบกับ Galvanium Framework (Simulate) โดยใช้ข้อมูลขนาด 10,000, 20,000, 30,000, 40,000, 50,000 และ 60,000 ตามลำดับ ซึ่งจะได้ผลการทดลองดังนี้

ตารางที่ 1. แสดงเวลาการทำงานของ setDataSource (Native)

รอบ/จำนวนข้อมูล	10000	20000	30000	40000	50000	60000
ครั้งที่ 1	961.56	991.86	1017.82	1110.07	1590.30	1366.12
ครั้งที่ 2	662.83	1048.51	945.73	1171.04	1576.51	1806.43
ครั้งที่ 3	637.03	695.38	1180.48	1381.21	1223.81	1672.22
ครั้งที่ 4	647.83	804.73	912.44	1080.31	1425.32	1397.01
ครั้งที่ 5	555.92	899.59	911.45	1070.77	1382.76	1349.99
ครั้งที่ 6	768.18	916.40	1126.5	1290.08	1693.77	1823.10
เฉลี่ย	705.55	892.74	1015.73	1183.91	1482.07	1569.14

ตารางที่ 2. แสดงเวลาการทำงานของ btnNative_Click

รอบ/จำนวนข้อมูล	10000	20000	30000	40000	50000	60000
ครั้งที่ 1	1140.07	1461.09	1493.11	1728.51	2453.62	2350.58
ครั้งที่ 2	979.98	1413.47	1599.62	1854.33	2307.71	2632.31
ครั้งที่ 3	792.46	1197.03	1593.44	1960.33	2052.57	2595.16
ครั้งที่ 4	974.16	1188.65	1417.78	1760.22	2241.42	2341.93
ครั้งที่ 5	708.28	1233.44	1475.80	1673.10	2328.64	2388.10
ครั้งที่ 6	916.97	1213.10	1553.92	1963.74	2634.24	2613.70
เฉลี่ย	918.65	1284.46	1522.27	1823.372	2336.36	2486.96

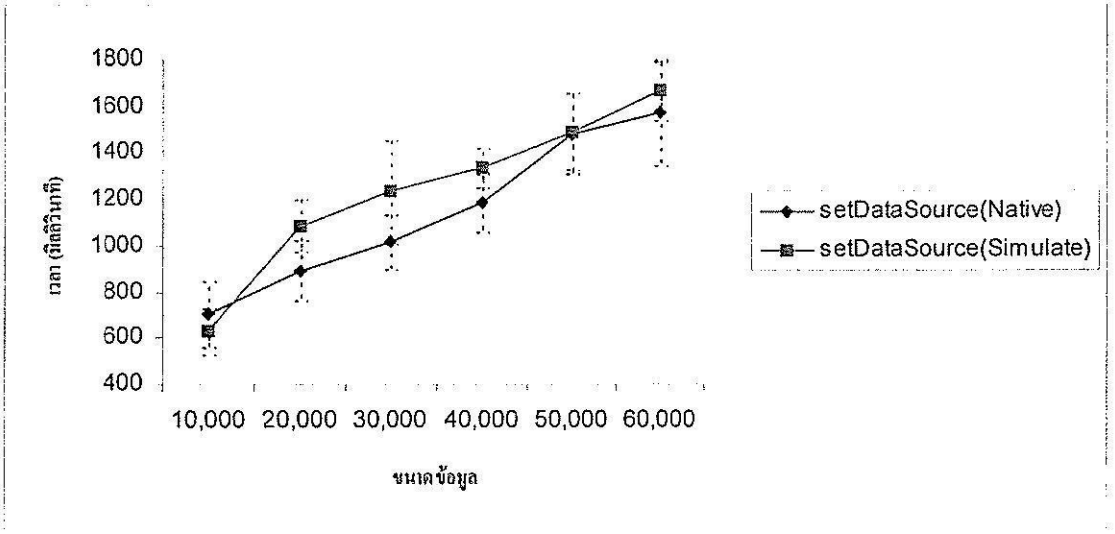
ตารางที่ 3. แสดงเวลาการทำงานของ setDataSource (Simulate)

รอบ/จำนวนข้อมูล	10000	20000	30000	40000	50000	60000
ครั้งที่ 1	562.08	1110.49	1403.84	1413.57	1413.57	1716.62
ครั้งที่ 2	506.37	1214.79	845.37	1288.15	1759.63	1795.52
ครั้งที่ 3	669.14	1192.29	1250.60	1435.95	1557.97	1609.67
ครั้งที่ 4	729.51	904.87	1430.95	1300.58	1357.80	1444.73
ครั้งที่ 5	558.00	1075.55	1362.97	1211.87	1311.94	1670.15
ครั้งที่ 6	757.04	1021.97	1095.11	1365.76	1535.32	1778.81
เฉลี่ย	630.35	1086.66	1231.47	1335.98	1489.37	1669.25

ตารางที่ 4. แสดงเวลาการทำงานของ btnSimulate_Click

รอบ/จำนวนข้อมูล	10000	20000	30000	40000	50000	60000
ครั้งที่ 1	977.20	1586.61	2104.64	2177.54	2177.54	2841.46
ครั้งที่ 2	935.82	1759.60	1643.68	2101.08	2789.47	2985.86
ครั้งที่ 3	1018.25	1691.37	1992.46	2190.58	2642.80	2821.66
ครั้งที่ 4	1031.83	1369.05	2086.22	2086.44	2544.63	2812.83
ครั้งที่ 5	861.37	1545.59	1965.51	2133.15	2335.72	2806.99
ครั้งที่ 6	1038.10	1465.36	1970.25	2222.27	2457.80	2881.47
เฉลี่ย	977.09	1569.59	1960.46	2151.84	2491.32	2858.37

จากผลการทดลองจะได้กราฟดังรูปที่ 22 และ 23 ซึ่งกราฟในรูปที่ 22 แสดงเวลาการทำงานของเมธอด DataGrid.setDataSource ในหน่วยมิลลิวินาทีของระบบแสดงผลปกติของ .NET เทียบกับระบบแสดงผลของ Galvanium Framework เมื่อใช้แสดงผลขนาดข้อมูล 10,000, 20,000, 30,000, 40,000, 50,000 และ 60,000 ตามลำดับ

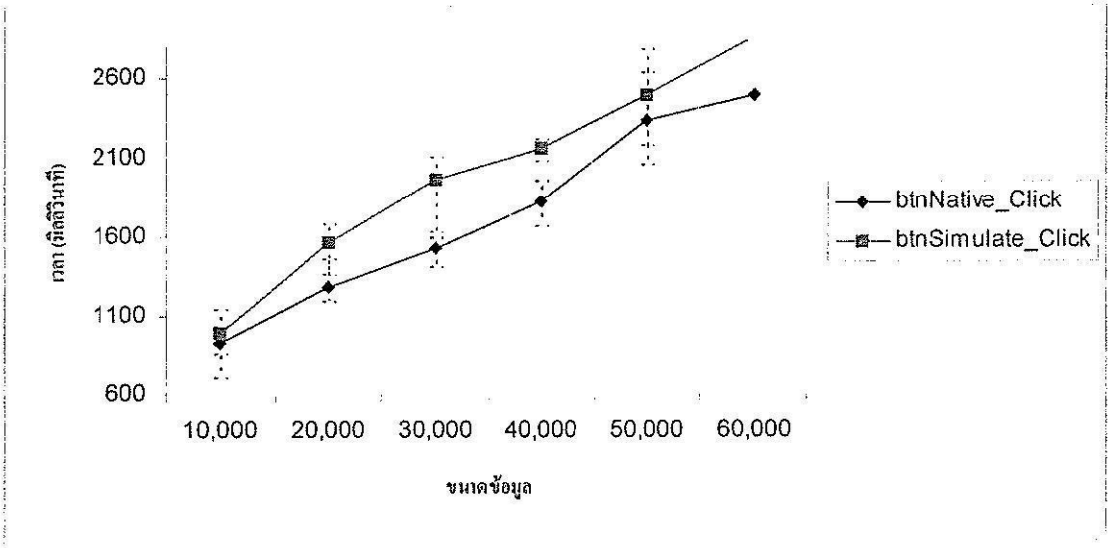


รูปที่ 22 กราฟแสดงการทำงานของเมธอด DataGrid.setDataSource

กราฟในรูปที่ 22 แสดงเวลาการทำงานของเมธอด DataGrid.setDataSource ในหน่วยมิลลิวินาทีของการแสดงรายการวัตถุปกติของ .NET เทียบกับการแสดงรายการวัตถุที่ได้จาก Galvanium Framework

เมื่อขนาดข้อมูลเป็น 10,000, 20,000, 30,000, 40,000, 50,000 และ 60,000 ตามลำดับ โดยเมื่อจำนวนวัตถุที่นำมาแสดงผลมีจำนวน 10000 สภาพแวดล้อมภายนอกจะมีผลทำให้ความเร็วในการแสดงผลวัตถุของ .NET และวัตถุที่ได้จาก Galvanium Framework มีค่าใกล้เคียงกัน ในขณะที่เมื่อข้อมูลเป็น 20,000, 30,000 และ 40,000 นั้น การแสดงผลโดยรวมของวัตถุจาก Galvanium Framework จะช้ากว่า แต่เมื่อข้อมูลเป็น 50,000 เวลาที่วัดได้ใกล้เคียงกันมาก เมื่อจำนวนวัตถุมากกว่า 50,000 จำนวนข้อมูลน่าจะมากเกินความสามารถของตัว Buffer สำหรับการแสดงผลจึงทำให้ความแตกต่างน้อยลง และความแตกต่างนี้ยังคงน้อยอยู่เมื่อเพิ่มจำนวนข้อมูลเป็น 60,000

สภาพแวดล้อมภายนอกมีผลต่อการทำงานในการแสดงผลของโปรแกรมประยุกต์ประเภท Desktop มาก เนื่องจากการแสดงผลหนึ่งครั้งจะเกี่ยวข้องกับโปรแกรมจำนวนมากรวมถึงการส่ง message ตอบรับไปยัง display subsystem ของระบบปฏิบัติการด้วย



รูปที่ 23 กราฟแสดงการทำงานของการกดปุ่มบนหน้าจอโปรแกรม

กราฟรูปที่ 23 แสดงเวลาการทำงานของการกดปุ่มบนหน้าจอโปรแกรม ซึ่งมีการทำงานในระดับที่สูงกว่าการทำงานของเมธอดที่แสดงในกราฟรูปที่ 22 โดยแสดงเวลาเป็นหน่วยมิลลิวินาที และเป็นการแสดงรายการวัตถุปรกติของ .NET เทียบกับการแสดงรายการวัตถุที่ได้จาก Galvanium Framework โดยข้อมูลที่ใช้จะเป็นข้อมูลชุดเดียวกันกับกราฟรูปที่ 22 ภาพรวมของกราฟเป็นไปตามลักษณะเดียวกันกับกราฟรูปที่ 22

จากกราฟรูปที่ 22 และ 23 พบว่า วัตถุที่สร้างจาก Galvanium Framework สามารถใช้ในการแสดงผลงานทั่วๆไปที่มีขนาดวัตถุ 10,000 ได้มีประสิทธิภาพในระดับเดียวกันกับวัตถุปรกติของ

.NET แต่จะอาจจะมีประสิทธิภาพลดลงบ้างเมื่อจำนวนของวัตถุเพิ่มขึ้น เหตุผลที่วัตถุจาก Galvanium Framework มีการแสดงผลได้ช้ากว่าวัตถุปรกติของ .NET เนื่องจากมี overhead เกิดขึ้นในการจำลองคุณสมบัติเชิงพลวัตรผ่านกระบวนการสะท้อน เพราะไม่ได้เข้าถึงวัตถุโดยตรงในลักษณะเดียวกับคลาสมาตรฐานของ .NET แต่ทำผ่านการเรียกใช้กลุ่มเมธอด get และ set ตามข้อตกลงของ JavaBeans อย่างไรก็ตามจุดเด่นของวัตถุที่สร้างจาก Galvanium Framework คือการเข้ากันได้กับวัตถุประเภท Java บนแพล็ตฟอร์ม JVM และในการใช้งานทั่วไปการแสดงผลข้อมูลขนาดมากกว่า 10,000 วัตถุในหน้าจอเดียวกันนั้นเกิดขึ้นน้อย Gaivanium Framework จึงสามารถใช้ทดแทนวัตถุปรกติของ .NET ได้ในกรณีที่ต้องการเชื่อมต่อกับระบบภายนอกประเภท Java

บทที่ 6
บทสรุป

1. สรุปผลการวิจัย

การทำงานร่วมกันระหว่างวัตถุบนแพล็ตฟอร์ม Java และแพล็ตฟอร์ม .NET ไม่สามารถทำได้สะดวก เนื่องจากความเข้ากันไม่ได้ของลักษณะระบบชนิดข้อมูลและวัตถุในระดับรายละเอียดโดยข้อตกลงของคุณสมบัติของวัตถุได้กำหนดมาให้มีความแตกต่างกันทำให้การสร้างระบบลูกข่ายขนาดบางที่เป็น .NET เพื่อเชื่อมต่อไปยังแม่ข่ายที่เป็น Java นั้น ไม่สามารถแสดงผลรายการวัตถุผลลัพธ์ได้ทันที สำหรับปัญหาของความแตกต่างระหว่างชนิดข้อมูลพื้นฐานที่ใช้ ได้แก้ไขโดยการใช้ตัวแปลงของ IKVM เข้ามาช่วย

งานวิจัยนี้ได้แก้ปัญหาความเข้ากันไม่ได้ระหว่างวัตถุบนสองแพล็ตฟอร์มนี้โดยการใช้ข้อมูลไทพ์เชิงพลวัตร ซึ่งจำลองคุณสมบัติของวัตถุฝั่ง Java ผ่านกลไกการสะท้อน (Reflection) บน .NET เพื่อให้สามารถนำวัตถุดังกล่าวไปโยงเข้ากับตัวแสดงผลบนแพล็ตฟอร์ม .NET ได้ทันที ผลลัพธ์ที่ได้จากการวิจัย คือ Galvanium Framework ซึ่งประกอบไปด้วยตัวเชื่อมต่อระหว่างเครื่องแม่ข่ายและลูกข่ายผ่านโปรโตคอล Hessian และตัวจำลองคุณสมบัติวัตถุสำหรับการโยงข้อมูลของ Java Bean ที่แปลงเป็นวัตถุ .NET ในการทดลองสร้างโปรแกรมลูกข่ายพบว่าสามารถสร้างลูกข่ายที่เชื่อมต่อกับโปรแกรมแม่-ข่าย ซึ่งเป็นระบบที่พัฒนาด้วย EJB ผ่านโปรโตคอล Hessian และสามารถทำงานได้อย่างปกติ ในการทดลองวัดประสิทธิภาพโดยวัดการสร้างวัตถุแบบ .NET เทียบกับการสร้างวัตถุแบบ Java Bean แล้วทำการโยงข้อมูลผ่าน Galvanium Framework พบว่าเมื่อจำนวนข้อมูล 10,000 รายการ การแสดงผลของทั้งสองระบบใช้เวลาไม่แตกต่างกันอย่างมีนัยสำคัญ และเมื่อเพิ่มจำนวนวัตถุมากขึ้น การแสดงผลวัตถุด้วย Galvanium Framework จะช้ากว่าวัตถุปกติของ .NET แต่จะมีความเร็วใกล้เคียงกันอีกครั้งเมื่อจำนวนข้อมูล 50,000 รายการ

2. ข้อเสนอแนะ

ในการวิจัยต่อไปสามารถปรับแต่งและพัฒนาให้การทำงานของ Galvanium Framework มีประสิทธิภาพสูงขึ้นได้ และสามารถทำให้รองรับเฟรมเวิร์คอื่นๆของ .NET ในเวอร์ชั่นที่แตกต่างกัน เช่น .NET 4.0, Mono หรือ Compact Framework เป็นต้น

รูปแบบการนำไปสู่การใช้งานเชิงพาณิชย์ของ Galvanium Framework นั้นสามารถทำได้โดยการแจกจ่ายตัวเฟรมเวิร์คในลักษณะลิขสิทธิ์คู่ GPL/Commercial และให้บริการในการพัฒนาระบบข้าม

แพล็ตฟอร์ม Java และ .NET ที่สร้างอยู่บน Galvanium Framework นี้ต่อไป

บรรณานุกรม

[1] Gosling, J., Joy, B., Steele, G., and Bracha, G. 2005. Java(tm) Language Specification, the (3rd Edition) (Java (Addison-Wesley)). Addison-Wesley Professional.

[2] Thai, T. and Lam, H. Q. 2003 .Net Framework Essentials. O'Reilly & Associates, Inc.

[3] Fleury, M. and Reverbel, F. 2003. The JBoss extensible server. In Proceedings of the ACM/IFIP/USENIX 2003 international Conference on Middleware (Rio de Janeiro, Brazil, June 16 - 20, 2003). M. Endler, Ed. Middleware Conference. Springer-Verlag New York, New York, NY, 344-373.

[4] Shan, Y. 1989. An event-driven model-view-controller framework for Smalltalk. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (New Orleans, Louisiana, United States, October 02 - 06, 1989). OOPSLA '89. ACM, New York, NY, 347-352.

[5] Teixeira, S. and Pacheco, X. 2001. Borland Delphi 6: Developer's Guide. Sams.

[6] Gray, D. N., Hotchkiss, J., LaForge, S., Shalit, A., and Weinberg, T. 1998. Modern languages and Microsoft's component object model. Commun. ACM 41, 5 (May. 1998), 55-65.

[7] World Wide Web Consortium. 2000. Extensible Markup Language (Xml) 1.0 Specifications: from the W3c Recommendations. iUniverse, Incorporated.

[8] Coremans, C. 2006. AJAX and Flash Development with Openlaszlo: a Tutorial. BrainySoftware.com.

[9] Kennedy, A. and Syme, D. 2001. Design and implementation of generics for the .NET Common language runtime. SIGPLAN Not. 36, 5 (May. 2001), 1-12.

[10] Weaver, J. L., Gao, W., Chin, S., and Iverson, D. 2009. Pro JavaFX Platform: Script, Desktop and Mobile RIA with Java Technology. 1st. Apress.

[11] Chen, H. and Cheng, R. 2007. ZK: Ajax Without the Javascript Framework. Apress.

[12] Denninger, S. and Peters, I. 2003. Enterprise Java Beans 2.1. APress L. P.

[13] Roman, E. 2005. Mastering Enterprise Java Beans, Wiley.

[14] Hightower, R. and Gradecki, J. D. 2003. Mastering Resin. 1. John Wiley & Sons, Inc.

[15] Dumbill, E. and Bornstein, N. M. 2004. Mono (Developer's Notebook). O'Reilly Media, Inc.

[16] Newcomer, E. 2002. Understanding Web Services: XML, WSDL, SOAP, and UDDI. Addison-Wesley Professional.

[17] Lindholm, T. and Yellin, F. 1999. Java Virtual Machine Specification. 2nd. Addison-Wesley Longman Publishing Co., Inc.

[18] The GNU Classpath Project. www.gnu.org/software/classpath/classpath.html.

[19] Ibrahim, M. H. 1992. Reflection and metalevel architectures in object-oriented programming. In Addendum To the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum) (Vancouver, British Columbia, Canada, October 18 - 22, 1992). J. L. Archibald and M. C. Wilkes, Eds. OOPSLA '92. ACM, New York, NY, 315-318.

ประวัติผู้วิจัย

อาจารย์ ดร. ชาญวิทย์ แก้วกสิ เป็นอาจารย์ประจำสาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี จบการศึกษาระดับปริญญาตรี วิศวกรรมคอมพิวเตอร์ (เกียรตินิยมอันดับ 1) จากมหาวิทยาลัยเทคโนโลยีสุรนารี และระดับปริญญาโท วิศวกรรมคอมพิวเตอร์ จากจุฬาลงกรณ์มหาวิทยาลัย ระดับปริญญาเอก Computer Science จาก University of Manchester มีความเชี่ยวชาญทางวิศวกรรมซอฟต์แวร์เชิงวัตถุ (object-oriented) และเชิงลักษณะ (aspect-oriented) มีผลงานวิจัยทางวิศวกรรมซอฟต์แวร์ได้รับการตีพิมพ์ในระดับนานาชาติ และได้พัฒนาเครื่องมือเพื่อช่วยลดระยะเวลาในการพัฒนาซอฟต์แวร์อย่างต่อเนื่อง