บทที่ 2

ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

ในบทนี้จะกล่าวถึงทฤษฎีและงานวิจัยที่เกี่ยวข้อง โดยประกอบด้วยหลักการพัฒนาซอฟต์แวร์แบบ Model-Driven Architecture (MDA) นิยามของความปลอดภัยสำหรับซอฟต์แวร์ การรักษาความปลอดภัยของเว็บเซอร์วิส (WS-Security) การรักษาความปลอดภัยของจาวาเซิร์ฟเล็ต (Java Servlet) และงานวิจัยที่เกี่ยวข้อง

2.1 การพัฒนาซอฟต์แวร์แบบ MDA

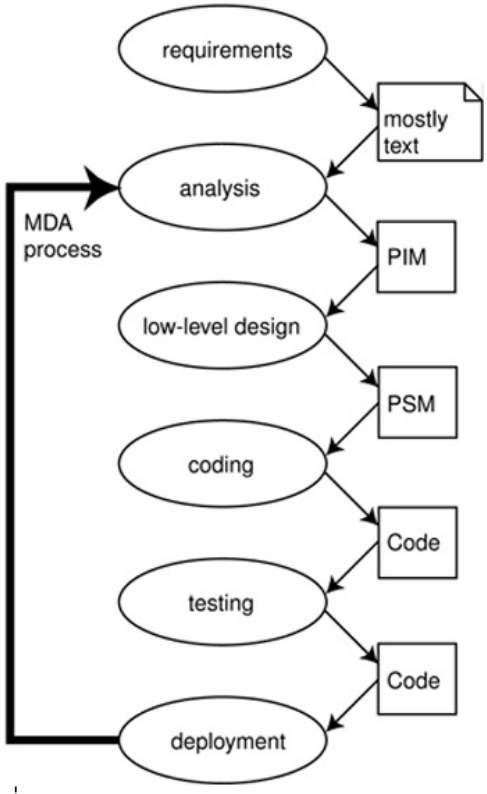
2.1.1 แนวคิดการพัฒนาซอฟต์แวร์แบบ MDA

Model-Driven Architecture หรือ MDA คือกระบวนการพัฒนาซอฟต์แวร์ที่กำหนดขึ้นโดยองค์กร Object Management Group (OMG) (Miller & Mukerji, 2003) โดยมีหลักการคือการใช้แบบจำลอง (Model) เป็นตัวขับเคลื่อนการพัฒนาซอฟต์แวร์ กล่าวคือใช้แบบจำลองในการอธิบายความเข้าใจของระบบ การออกแบบ การสร้าง การติดตั้ง การดูแลรักษา และตลอดจนปรับปรุงแก้ไขระบบ แสดงในภาพที่ 2.1 โดยวงจรชีวิตการพัฒนาซอฟต์แวร์แบบ MDA เริ่มต้นจากการนำความต้องการของระบบเข้าสู่กระบวนการวิเคราะห์โดยแบบจำลองที่นักวิเคราะห์สร้างขึ้นแทนระบบที่ต้องการพัฒนาคือแบบจำลองที่อิสระจากแพลตฟอร์ม (Platform Independent Model : PIM) ถัดมาคือขั้นตอนการออกแบบ ในการออกแบบมีการเลือกแพลตฟอร์มที่ต้องการใช้ ดังนั้นการออกแบบจะใช้วิธีการนำแบบจำลอง PIM เข้าสู่กระบวนการแปลงแบบจำลองเพื่อสร้างแบบจำลองที่เฉพาะเจาะจงกับแพลตฟอร์ม (Platform Specific Model : PSM) ส่วนในขั้นตอนการโปรแกรมมิ่งก็จะนำแบบจำลอง PSM สร้างเป็นโค้ดเพื่อนำไปทดสอบและติดตั้ง ตามลำดับ (Kleppe, Warmer & Bast, 2003)

หลักการสำคัญในการกำหนดแบบจำลองในกระบวนการพัฒนาซอฟต์แวร์แบบ MDA นั้นจะใช้มุมมอง (Viewpoint) ของระบบในการกำหนดแบบจำลอง (Miller & Mukerji, 2003) โดยมีการแบ่งมุมมองของระบบที่แตกต่างกัน 3 มุมมองเพื่อใช้สำหรับอธิบายถึงระบบที่ต้องการพัฒนา ดังนี้

ภาพที่ 2.1

วงจรชีวิตการพัฒนาซอฟต์แวร์แบบ MDA



ที่มา: “MDA Explained: The Model Driven Architecture: Practice and Promise”, Wesley, 2003

- Computation Independent Viewpoint

เป็นมุมมองที่เน้นในส่วนของสภาพแวดล้อมของระบบ และความต้องการของระบบเท่านั้น ไม่สนใจรายละเอียดของโครงสร้าง และการประมวลผลของระบบ

- Platform Independent Viewpoint

เป็นมุมมองที่เน้นในเรื่องของการประมวลผลของระบบ โดยที่ไม่มีการระบุรายละเอียดที่เกี่ยวข้องกับแพลตฟอร์มที่จะเลือกมาใช้งาน โดยในมุมมองนี้จะแสดงข้อกำหนด (Specification) ของระบบที่สมบูรณ์แล้ว ซึ่งจะไม่มีการเปลี่ยนแปลงแม้จะมีการเลือกใช้แพลตฟอร์มที่แตกต่างกัน

- Platform Specific Viewpoint

เป็นมุมมองของระบบที่ได้ทำการรวมเอามุมมองของ Platform Independent Viewpoint มารวมกับรายละเอียดของแพลตฟอร์มที่เลือกใช้ในการพัฒนาระบบ

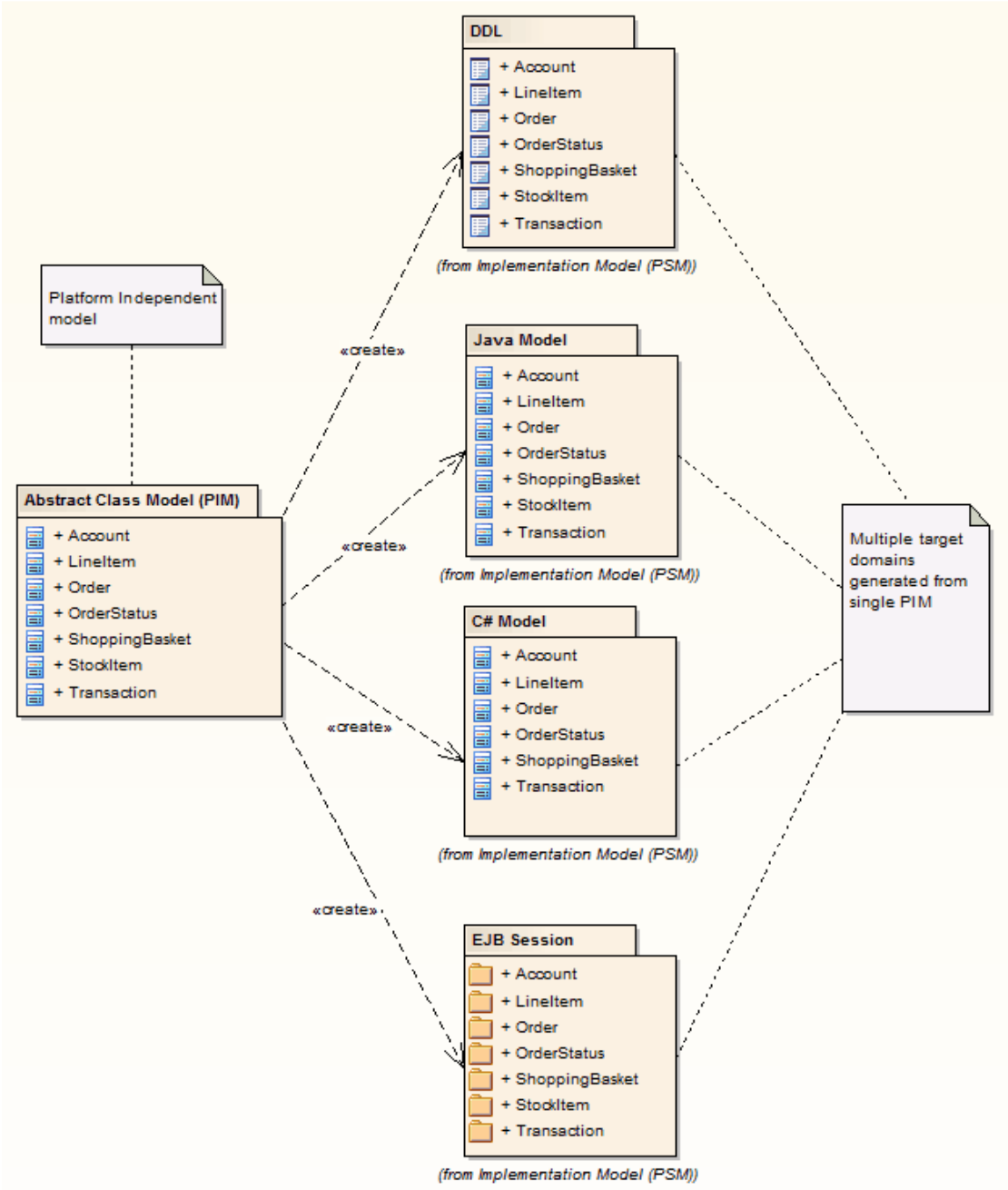
จากมุมมองของระบบทั้ง 3 มุมมองที่กล่าวสามารถสร้างแบบจำลองได้ 3 แบบ คือ Computation Independent Model (CIM) Platform Independent Model (PIM) และ Platform Specific Model (PSM)

แบบจำลอง CIM เป็นแบบจำลองที่ใช้แทนถึงระบบที่พัฒนาในมุมมอง Computation Independent Viewpoint ที่แสดงถึงความต้องการของระบบ สามารถเรียกแบบจำลองนี้ได้อีกอย่างว่า แบบจำลองโดเมน (Domain Model) หรือแบบจำลองทางธุรกิจ (Business Model) โดยที่แบบจำลอง CIM เป็นแบบจำลองที่มีบทบาทสำคัญในการเป็นตัวเชื่อมช่องว่างระหว่าง ผู้เชี่ยวชาญเกี่ยวกับโดเมนและความต้องการของระบบกับผู้เชี่ยวชาญในการออกแบบที่จะทำหน้าที่สร้างชิ้นงาน (Artifacts) ที่ใช้อธิบายความต้องการในโดเมนนั้น โดยส่วนใหญ่แล้วแบบจำลอง CIM จะถูกนำเสนอรายละเอียดในรูปแบบของข้อความ (Kleppe, Warmer & Bast, 2003)

แบบจำลอง PIM เป็นแบบจำลองที่ใช้อธิบายถึงระบบที่ต้องการพัฒนา ตามมุมมองของ Platform Independent Viewpoint ซึ่งแบบจำลองในระดับนี้ไม่ได้มีการระบุรายละเอียดแพลตฟอร์มที่เลือกใช้ นั่นคือเป็นแบบจำลองของระบบที่สามารถนำไปประยุกต์ใช้กับแพลตฟอร์มที่แตกต่างกันได้ ซึ่งในการออกแบบแบบจำลอง PIM นักวิเคราะห์ระบบ (System Analyst) มีการวิเคราะห์ความต้องการทั้งที่เป็นความต้องการเชิงฟังก์ชันหลัก (Functional Requirement) และความต้องการที่ไม่ใช่เชิงฟังก์ชันหลัก (Non-Functional Requirement) แต่ในการพัฒนาซอฟต์แวร์จริงก็จะพบปัญหาเกี่ยวกับการวิเคราะห์ความต้องการที่ไม่ใช่เชิงฟังก์ชันหลักที่จะส่วนใหญ่ถูกพัฒนาหลังจากที่พัฒนาแอพพลิเคชั่นเสร็จแล้ว ซึ่งก่อให้เกิดค่าใช้จ่ายในการพัฒนาที่เพิ่มขึ้น (Devanbu & Stubbleine, 2000) ปัจจุบันจึงมาหลายงานวิจัย (Rodrigues, Roberts, Emmerich & Skene, 2001) ที่ได้เสนอแนวทางและวิธีการในการนำความต้องการที่ไม่ใช่เชิงฟังก์ชันหลักของระบบอย่างเช่นความน่าเชื่อถือ (Reliability) ประสิทธิภาพ (Performance) รวมถึงความปลอดภัย (Security) เพื่ออำนวยความสะดวกให้นักวิเคราะห์สามารถเพิ่มความต้องการดังกล่าวในการออกแบบแบบจำลอง PIM ได้ โดยตัวอย่างแบบจำลอง PIM แสดงในภาพที่ 2.2

ภาพที่ 2.2

แบบจำลอง PIM และแบบจำลอง PSM



ที่มา: http://www.sparxsystems.com/enterprise_architect_user_guide/software_development/mdastyletransforms.html

แบบจำลอง PSM เป็นแบบจำลองที่นำแบบจำลอง PIM มาเพิ่มเติมรายละเอียดที่เกี่ยวข้องกับเพลตฟอร์มหรือเทคโนโลยีที่เลือกใช้ในการพัฒนาระบบ โดยแบบจำลองที่ได้จากการวิเคราะห์และออกแบบตามมุมมองของ Platform Specific Viewpoint ตัวอย่างแบบจำลอง PSM แสดงในภาพที่ 2.2 จากแบบจำลอง PIM ซึ่งอยู่ในรูปของคลาส (Class) สามารถที่สร้างแบบจำลอง PSM ได้หลายแบบจำลองขึ้นอยู่กับแพลตฟอร์มที่เลือกใช้ เช่นแพลตฟอร์มจาวา แพลตฟอร์ม C# หรือ Enterpise JavaBeans (EJB)

โดยที่หัวใจของการพัฒนาซอฟต์แวร์แบบ MDA คือแนวคิดของการแปลงแบบจำลอง (Model Transformation) จากแบบจำลอง PIM เป็นแบบจำลอง PSM (Swithinbank et al., 2005) หลักการของการแปลงแบบจำลองสามารถแบ่งออกได้เป็น 2 ระดับคือการแปลงแบบจำลองระดับ Model-to-Model และ Model-to-Text หรือ Model-to-Code ดังแสดงในภาพที่ 2.3 รายละเอียดของการแปลงแบบจำลองมีดังนี้

การแปลงแบบจำลองแบบ Model-to-Model คือการแปลงแบบจำลองหนึ่งเป็นอีกแบบจำลองหนึ่ง ซึ่งแบบจำลองที่ใช้วิธีการแบบนี้เป็นแบบจำลองที่อยู่ในระดับที่เป็นนามธรรม (Abstract level) อย่างเช่นการแปลงแบบจำลองจากแบบจำลอง PIM เป็นแบบจำลอง PSM หรือการแปลงแบบจำลองในระดับเดียวกันแต่ต้องการให้แบบจำลองปลายทางมีรายละเอียดที่มากขึ้น

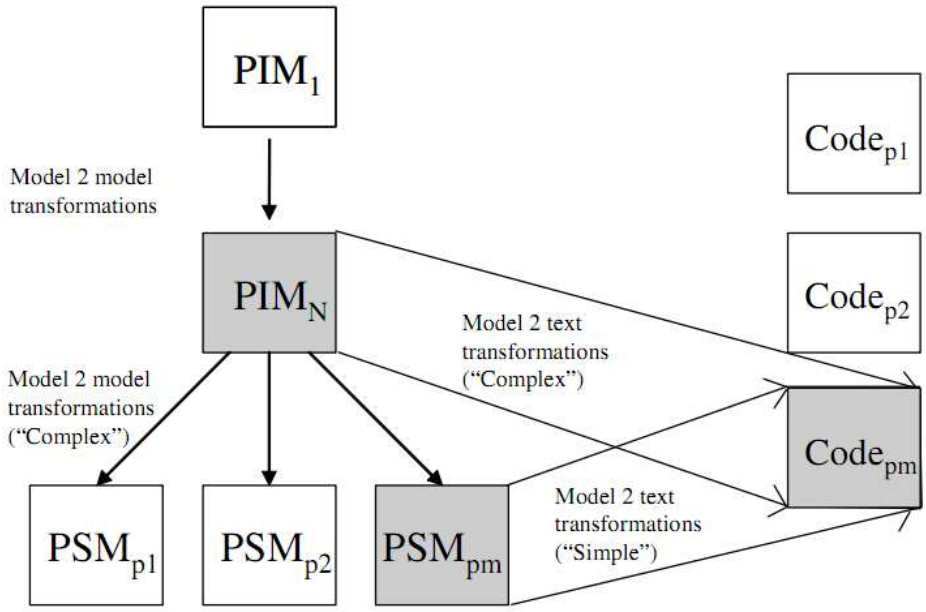
การแปลงแบบจำลองแบบ Model-to-Text คือการแปลงแบบจำลองที่เป็นแบบจำลองในระดับนามธรรม อย่างเช่นแบบจำลอง PIM หรือ แบบจำลอง PSM เป็นโค้ด

ในการพัฒนาซอฟต์แวร์แบบ MDA นั้นได้ใช้หลักการแปลงแบบจำลองทั้งแบบ Model-to-Model และ Model-to-Text ซึ่งส่วนใหญ่แล้วหลักการพัฒนาซอฟต์แวร์แบบ MDA จะเริ่มต้นที่การพัฒนาแบบจำลอง PIM แล้วแปลงเป็นแบบจำลอง PSM ที่ระบุแพลตฟอร์มที่เลือกใช้ ขั้นตอนนี้เรียกว่าการแปลงแบบจำลองแบบ Model-to-Model แล้วหลังจากนั้นก็นำแบบจำลอง PSM ที่สร้างขึ้นแปลงเป็นซอร์สโค้ด เป็นขั้นตอนการแปลงแบบจำลองแบบ Model-to-Text วิธีการแบบนี้มีข้อดีคือสามารถสร้างแบบจำลอง PSM ที่มีลักษณะเป็นทั่วไป (Common) ของแพลตฟอร์มนั้นไว้ก่อนได้ แล้วค่อยนำแบบจำลอง PSM ไปสร้างเป็นซอร์สโค้ด หลักการนี้ทำให้สามารถนำแบบจำลอง PSM ไปใช้งานซ้ำได้ (Reuse) กรณีแพลตฟอร์มนั้นสามารถที่จะเลือกพัฒนาโดยใช้เทคโนโลยีหรือเฟรมเวิร์กที่มีมากกว่าหนึ่งได้ โดยที่แต่ละเทคโนโลยีหรือเฟรมเวิร์กที่เลือกใช้มีการสร้างซอร์สโค้ดหรือไฟล์คอนฟิคกูเรชั่นที่แตกต่างกัน ซึ่งถ้าใช้วิธีการแบบ Model-to-Text ทำการแปลงจากแบบจำลอง PIM เป็นซอร์สโค้ดเลยนั้น ทำให้การแปลงแบบจำลองมีความ

ซับซ้อนกว่า และถ้าต้องการเปลี่ยนเทคโนโลยีหรือเฟรมเวิร์กก็จะไม่มีแบบจำลองที่สามารถจะนำกลับมาใช้ใหม่ได้ ต้องเริ่มพัฒนาในส่วนของการสร้างซอร์สโค้ดใหม่ตั้งแต่ต้น

ภาพที่ 2.3

การแปลงแบบจำลองในระดับ Model-to-Model และ Model-to-Text



ที่มา: Oldevik, Neple & Aagedal (2004)

2.1.2 **ภาษามาตรฐาน** (Standard Language)

การพัฒนาซอฟต์แวร์แบบ MDA ที่แบบจำลองแต่ละระดับของมีรายละเอียดของมุมมองที่แตกต่างกัน เพื่อให้การสร้างแบบจำลองเป็นไปได้อย่างเหมาะสมองค์กร OMG จึงได้กำหนดภาษามาตรฐาน (Standard Language) สำหรับใช้ในการกำหนดและสร้างแบบจำลองขึ้น ซึ่งภาษามาตรฐานที่เหมาะสมในการนำไปใช้อธิบายการทำงานของระบบในแต่ละระดับของแบบจำลอง มีดังนี้ (Miller & Mukerji, 2003)

- Meta-Object Facility (MOF) คือภาษาที่อยู่ในระดับนามธรรม (Abstract) สำหรับนำไปใช้ในการกำหนด สร้าง และจัดการกับเมตาโมเดล (Metamodel) ตัวอย่างการ

นำ MOF ไปกำหนดเป็นภาษาที่ใช้อธิบายโมเดล เช่น UML และ CWM (Common Warehouse Metamodel)

- Unified Modeling Language (UML) คือภาษามาตรฐานในการสร้างแบบจำลองสำหรับใช้ในการสร้างเอกสารที่ใช้อธิบายถึงระบบที่ต้องการพัฒนา ซึ่งบทบาทของ UML ในการพัฒนาซอฟต์แวร์แบบ MDA คือการนำมาใช้สร้างแบบจำลอง
- XML Metadata Interchange (XMI) คือวิธีการในการบันทึกแบบจำลองที่สร้างด้วยภาษา UML ให้อยู่ในรูปแบบของภาษา XML ซึ่ง UML มีพื้นฐานมาจากเมตาโมเดลที่กำหนดโดย MOF โดยสามารถกล่าวได้อีกทางหนึ่งว่า XMI คือการนำเมตาโมเดลที่กำหนดโดย MOF มาจัดเก็บให้อยู่ในโครงสร้างแบบ XML โดยที่ XMI เป็นข้อกำหนดที่นำไปใช้ในการจับคู่ (Mapping) ระหว่างเมตาโมเดลที่กำหนดด้วย MOF กับ XML DTDs และเอกสาร XML
- Object Constraint Language (OCL) คือภาษาที่ใช้สำหรับกำหนดเงื่อนไข และวิธีการเข้าถึงอิลิเมนต์ของแบบจำลอง หรือเมตาโมเดล
- Query/View/Transformation (QVT) คือภาษามาตรฐานที่ใช้สำหรับสร้างข้อกำหนดหรือกฎในการแปลงแบบจำลองระหว่างเมตาโมเดลของต้นทางและปลายทาง การแปลงแบบจำลองจะมีการใช้เอนจินต์ QVT (QVT Engine) ช่วยในการแปลงแบบจำลองต้นทางไปเป็นแบบจำลองปลายทางด้วยเงื่อนไขที่กำหนดไว้ในกฎหรือนิยามของการแปลง (Transformation Rule หรือ Transformation Definition)

2.1.3 **แนวทางการแปลงแบบจำลอง** (Model Transformation Approaches)

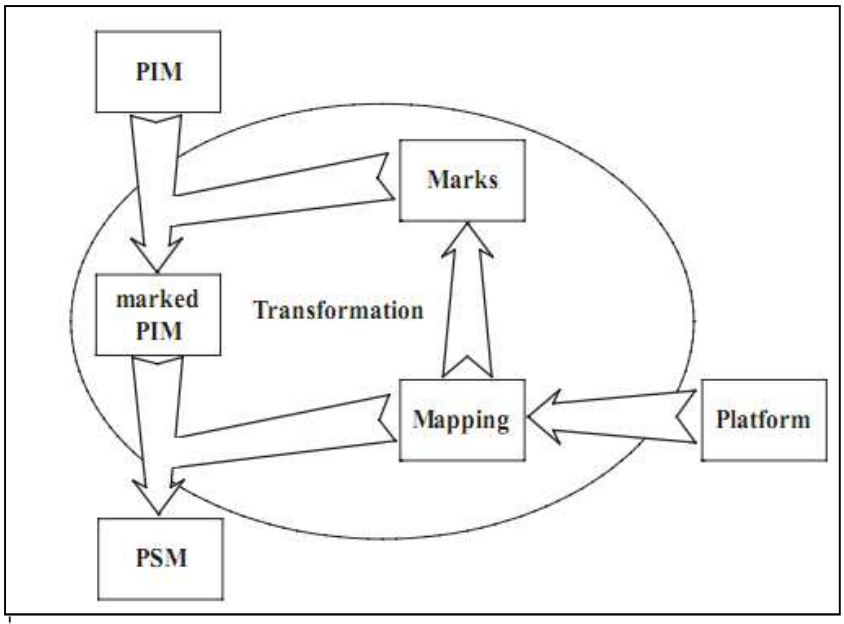
การแปลงแบบจำลอง (Model Transformation) คือกระบวนการในการแปลงแบบจำลองจากแบบจำลองหนึ่งไปเป็นแบบจำลองอีกแบบหนึ่งสำหรับระบบเดียวกัน (Miller & Mukerji, 2003) โดยการแปลงแบบจำลองแบ่งออกเป็นหลายวิธี เช่นการมาร์ค (Marking) การแปลงแบบจำลองโดยใช้เมตาโมเดล (Metamodel Transformation) โดยมีรายละเอียดแต่ละแบบดังนี้

1. **การมาร์ค** (Marking)

การมาร์ค เป็นแนวทางการแปลงแบบจำลองโดยการกำหนดแท็ก (Tag) พิเศษขึ้นสำหรับนำไปเพิ่มไว้ในแบบจำลอง เพื่อใช้เป็นแนวทางในการแปลงแบบจำลองจาก PIM เป็น PSM ซึ่งในการแปลงแบบจำลองนี้จะต้องเลือกเพลตฟอร์มที่ต้องการก่อน จากนั้นสร้างชุดแท็กพิเศษขึ้นมาเพื่อใช้แทนถึงแพลตฟอร์มที่เลือก และกำหนดการจับคู่ (Mapping) ระหว่างชุดแทกพิเศษกับแพลตฟอร์ม วิธีการแปลงแบบจำลองกระทำหลังจากเพิ่มแท็กพิเศษและเชื่อมโยงกับอิลิเมนต์ของ PIM เรียบร้อยแล้ว จากนั้นขั้นตอนการแปลงแบบจำลองจะทำการอ่านแบบจำลอง PIM ที่ได้มาร์คด้วยแท็กพิเศษ และทำการอ่านข้อกำหนดที่ได้เก็บไว้ในกำหนดการจับคู่เพื่อเรียกใช้งานแพลตฟอร์มที่ได้จับคู่ไว้มาสร้างเป็น PSM ที่เป็นแบบจำลองปลายทาง (Miller & Mukerji, 2003) ขั้นตอนดังกล่าวแสดงในภาพที่ 2.4

ภาพที่ 2.4

การแปลงแบบจำลองแบบการมาร์ค

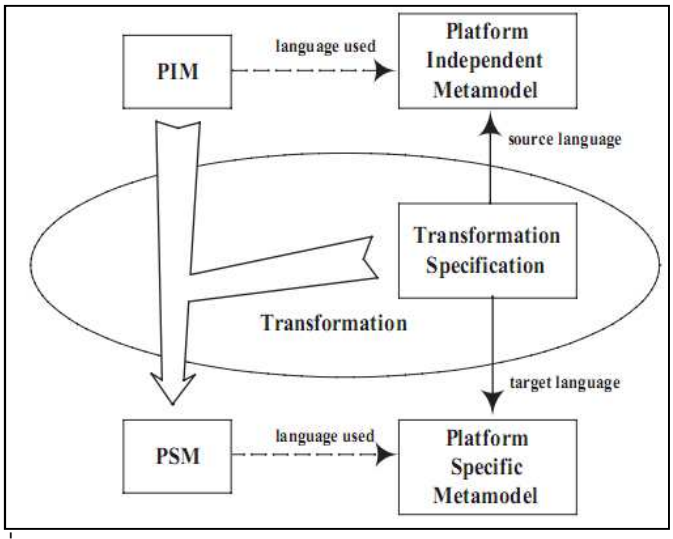


ที่มา: Miller & Mukerji (2003)

2. **การแปลงแบบจำลองโดยใช้เมตาโมเดล** (Metamodel Transformation)

การแปลงแบบจำลองโดยใช้เมตาโมเดล เป็นการแปลงแบบจำลองจากแบบจำลองต้นทางไปเป็นแบบจำลองปลายทางที่ถูกสร้างขึ้นโดยเมตาโมเดลของแต่ละแบบจำลอง อาจจะเป็นภาษาที่แตกต่างกัน ในขั้นตอนการแปลงแบบจำลองวิธีการนี้มีการกำหนดข้อกำหนดการแปลงแบบจำลอง (Transformation Specification) โดยการจับคู่ระหว่างเมตาโมเดลของแบบจำลองต้นทาง และเมตาโมเดลของแบบจำลองปลายทาง ซึ่งเมื่อสร้างแบบจำลอง PIM ขึ้นมาแล้ว การแปลงแบบจำลองสามารถทำได้โดยเข้าไปประมวลผลตามข้อกำหนดการแปลงแบบจำลองที่มีการกำหนดไว้ และท้ายสุดแบบจำลอง PSM ก็จะถูกสร้างขึ้นมาเป็นแบบจำลองปลายทาง (Miller & Mukerji, 2003) วิธีการแปลงแบบจำลองแสดงในภาพที่ 2.5

ภาพที่ 2.5
การแปลงแบบจำลองโดยใช้เมตาโมเดล



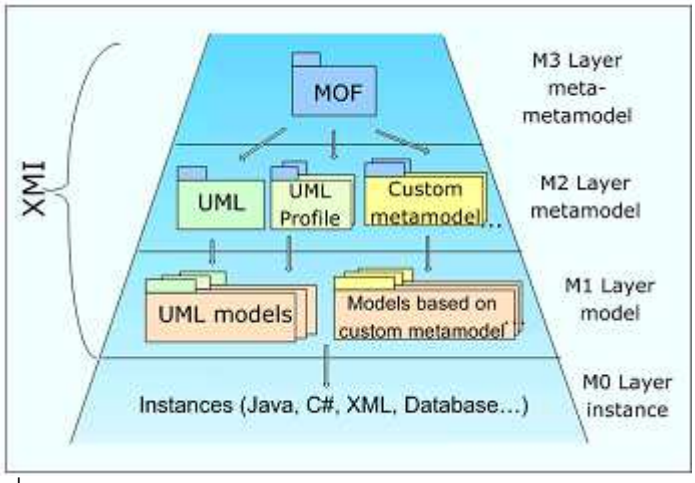
ที่มา: Miller & Mukerji (2003)

ซึ่งข้อแตกต่างระหว่างการแปลงแบบจำลองทั้ง 2 แบบนี้คือการแปลงแบบจำลองโดยการมาร์คแบบจำลอง PIM และแบบจำลอง PSM ถูกสร้างขึ้นมาด้วยภาษาเดียวกัน แต่สำหรับการแปลงแบบจำลองโดยใช้แบบเมตาโมเดล แบบจำลองของ PIM และแบบจำลอง PSM จะถูกสร้างขึ้นโดยใช้ภาษาที่แตกต่างกัน

2.1.4 แนวทางการสร้างแบบจำลอง

สถาปัตยกรรมของ MDA นั้นเรียกว่าเป็น สถาปัตยกรรมแบบประกอบจากชั้นเมตาโมเดล 4 ชั้น (4-Layer Architecture) (Đuric, 2004) ดังรูปที่ 2.6 ประกอบด้วยเลเยอร์ดังนี้ M3 เมตา-เมตาโมเดล (Meta-Metamodel) M2 เลเยอร์เมตาโมเดล (Metamodel) M1 เลเยอร์ model และ M0 เลเยอร์อินสแตนท์ (Instance) เรียงตามลำดับ โดยเลเยอร์ MOF เป็นเลเยอร์ M3 สำหรับใช้ในการกำหนดภาษาสำหรับใช้สร้างแบบจำลอง อย่างเช่นภาษา UML ซึ่งอยู่ในเลเยอร์ M2 และเป็นภาษาที่ใช้กำกับแบบจำลอง และถูกนำไปใช้ในการสร้างแบบจำลองของระบบที่ต้องการพัฒนาในเลเยอร์ M1 และเลเยอร์ท้ายสุดเป็นอินสแตนท์ของแบบจำลองที่ถูกสร้างขึ้นเพื่อใช้งานจริง

ภาพที่ 2.6
สถาปัตยกรรม 4 ชั้นของการสร้างเมตาดาต้าของ MOF



ที่มา: Đuric (2004)

2.1.5 วิธีการเพิ่มขยายและสร้างภาษาของแบบจำลอง (Extending and Creating Model Language)

การเพิ่มขยาย UML และสร้างภาษาสำหรับแบบจำลอง สามารถทำได้ 2 แนวทาง คือการเพิ่มขยาย UML โดยใช้กลไกของ UML Profile และการเพิ่มขยาย UML โดยใช้ MOF ซึ่งเรียกว่าเมตาโมเดล (Frankel, 2003) ซึ่งมีรายละเอียดการนิยามแต่ละแบบดังนี้

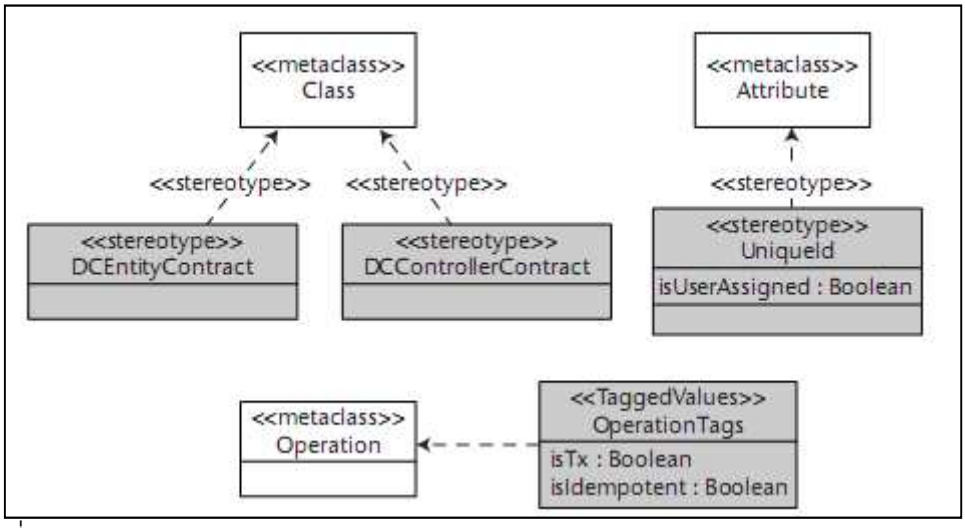
1. การเพิ่มขยาย UML โดยใช้กลไกของ UML Profile

Profile คือชุดของส่วนเพิ่มขยายที่จำเป็นสำหรับการนำมาประกอบเป็นภาษา UML ที่มีความเฉพาะเจาะจง (Dialect of UML) สามารถกล่าวได้ว่า UML ไม่ได้เป็นภาษาเพียงอย่างเดียวแต่เป็นหลักการของภาษาสำหรับแทนถึงภาษาในตระกูลที่มีพื้นฐานของภาษา UML (UML-Based Language) ในการพัฒนาซอฟต์แวร์แบบ MDA มีการใช้งานกลไกการเพิ่มขยาย (Extension Mechanism) นี้อย่างกว้างขวางเพราะต้องการสร้างแบบจำลองที่รองรับมุมมองของระบบ (System Aspects) และระดับนามธรรมของระบบ (Abstract Level) ที่แตกต่างกัน ซึ่งกลไกในการสร้างส่วนเพิ่มขยายนี้ได้กำหนดให้อยู่ในรูปของ Stereotype และ Tagged value โดยมีรายละเอียดดังนี้

การนิยาม Stereotype คือการกำหนดว่าอิลิเมนต์ของแบบจำลองนั้นมีการใช้งานที่พิเศษ โดยการกำหนด Stereotype จะแสดงอยู่ภายใต้เครื่องหมาย <<ชื่อ Stereotype>> Stereotype สามารถเพิ่มขยายมาจากอิลิเมนต์ของ UML เมตาโมเดลอิลิเมนต์ใดก็ได้ ซึ่งในตัวอย่าง Stereotype <<DCEntityContract>> และ <<DCControllerContract>> เพิ่มขยายสำหรับอิลิเมนต์ Class ส่วน Stereotype <<UniqueId>> เพิ่มขยายสำหรับอิลิเมนต์ Attribute และในส่วน Tagged Value ชื่อ EnterpriseOperation เป็นการเพิ่มข้อมูลให้กับอิลิเมนต์ Operation ซึ่งมีการกำหนด Property เพิ่มเติม 2 ค่า คือ isTx และ isIdempotent ดังแสดงในภาพที่ 2.7

ภาพที่ 2.7

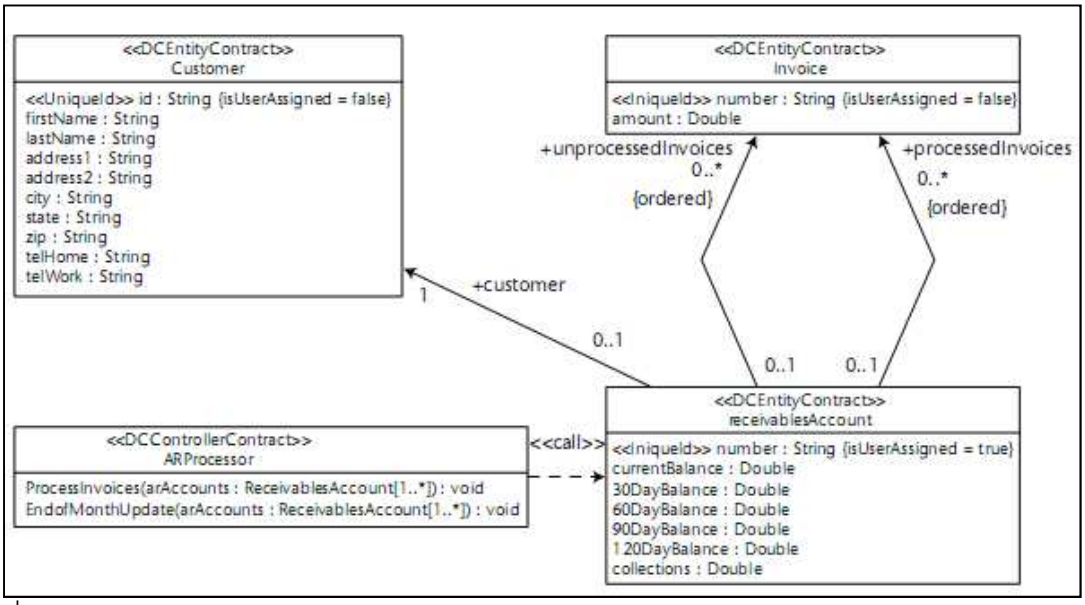
ตัวอย่างการกำหนด Profile เพื่อเพิ่มขยายแบบจำลอง UML



ที่มา: Frankel (2003)

ภาพที่ 2.8

ตัวอย่างการใช้งาน Stereotype และ Tagges Value



ที่มา: Frankel (2003)

Tagged Values คือกลไกสำหรับการเตรียมข้อมูลพิเศษที่เกี่ยวข้องกับ Stereotype โดยการกำหนดTagged Value จะกำหนดอยู่ในรูปแบบของแท็ก (Tag) และค่าของแท็ก โดยไวยากรณ์ในการกำหนดแท็กคือ <tag name>"="<tagged value> ในตัวอย่างภาพที่ 2.8 Stereotype <<UniqueId>> มี Tagged Value ชื่อ isUserAssigned และมีค่าของแท็กที่สามารถกำหนดได้ 2 ค่าคือ True หรือ False

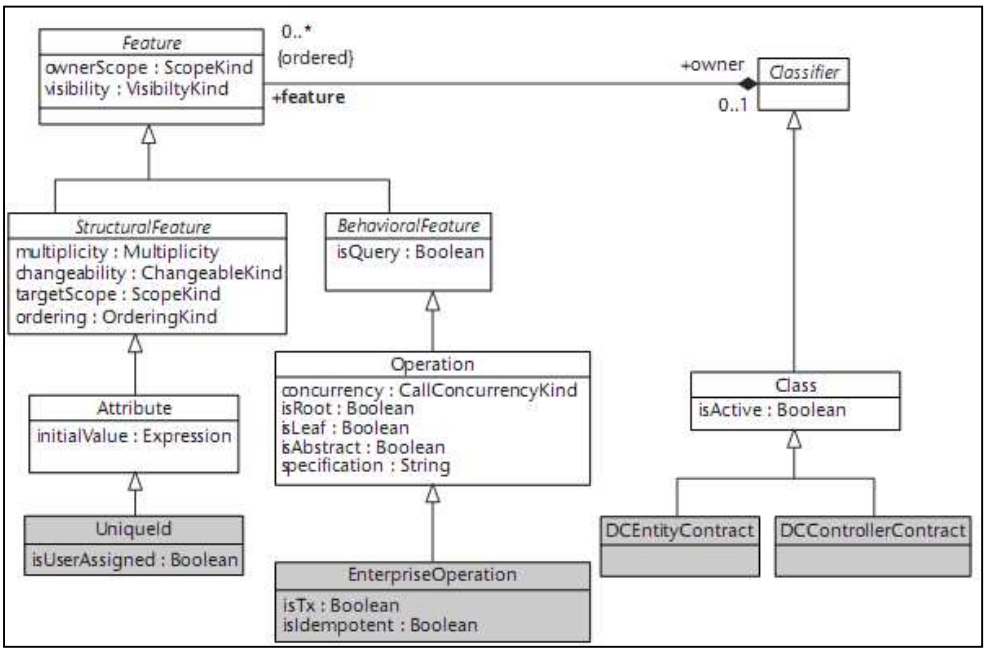
อิลิเมนต์หนึ่งสามารถมีการประยุกต์ใช้ Stereotype ได้หลาย Stereotype โดยภายใต้ Stereotype หนึ่งสามารถกำหนด Tagged Value ของ Stereotype นั้นได้หลายแท็ก

2. การเพิ่มขยาย UML โดยใช้ MOF

การเพิ่มขยาย UML โดยใช้ MOF สามารถเรียกได้อีกชื่อว่า "Heavyweight UML Metamodel Extension" ซึ่ง MOF จะมีเมตาโมเดลที่ใกล้เคียงกับโครงสร้างของคลาสของ UML วิธีการเพิ่มขยาย UML สำหรับอิลิเมนต์หนึ่งๆ โดยใช้ MOF ทำได้โดยการสร้างส่วนขยายเป็นซับคลาส (Subclass) ของอิลิเมนต์ในเมตาโมเดลของ UML ที่ต้องการ ตัวอย่างการเพิ่มขยาย UML แสดงในภาพที่ 2.9 แสดงการนิยาม DCEntityConstract และ DCControllerContact เป็นซับคลาสของอิลิเมนต์ Class ในเมตาโมเดลของ UML และนิยาม UniqueId เป็นซับคลาสของอิลิเมนต์ Attribute และนิยาม EnterpriseOperation เป็นซับคลาสของอิลิเมนต์ Operation และกำหนดให้มี Property คือ isTx และ isIdempotent

ภาพที่ 2.9

ตัวอย่างการเพิ่มขยาย UML โดยใช้ MOF



ที่มา: Frankel (2003)

3. การเปรียบเทียบการใช้งานระหว่าง UML Profile และการเพิ่มขยายแบบจำลองโดยใช้ MOF

แนวทางการเพิ่มขยาย UML โดยใช้ Profile เป็นวิธีที่มีคุณสมบัติสามารถใช้ในกรณีทั่วไปได้ (Generic) คือสามารถที่จะนำไปใช้กับเครื่องมือ UML ใดๆ ได้ ซึ่งนักออกแบบที่ต้องการใช้ส่วนเพิ่มขยายนี้ สามารถนำ Profile ไปใช้งานได้โดยไม่ขึ้นกับเครื่องมือที่ต่างผู้ผลิตกัน แต่วิธีการนี้มีข้อจำกัดในการนิยามความหมาย (Semantics) ในการสร้างแบบจำลองคลาสของ Object-Oriented ซึ่งในขณะที่ MOF สามารถสนับสนุนการนิยามความหมายดังกล่าวได้อย่างเต็มที่

ในขณะที่การเพิ่มขยาย UML โดยใช้ MOF หรือ Heavyweight Extensions นั้น นักออกแบบจะมีอิสระในการสร้างชุดของภาษาที่จะนำมาใช้ในการสร้างแบบจำลอง และประกอบกับเครื่องมือของ MDA สนับสนุนในการสร้างเมตาโมเดลสำหรับแบบจำลองแบบใหม่ได้อย่างเต็มที่ แต่จะมีข้อจำกัดในเรื่องไม่สามารถนำส่วนเพิ่มขยายนี้ไปใช้ในเครื่องมือ UML ทั่วไปได้

ดังนั้นหากการเพิ่มขยายที่มีการนิยามความหมาย (Semantics) ควรใช้วิธีการสร้างเมตาโมเดลโดยใช้ MOF แต่หากไม่มีการเพิ่มความหมายที่อธิบายได้ยากโดยใช้แบบจำลอง Object-Oriented ควรใช้ UML Profile

นอกจากนี้ ในการสร้างกฎการแปลงแบบจำลอง (Transformation Rule) โดยเปรียบเทียบการสร้างแบบจำลอง PIM และ PSM ของทั้ง 2 แบบ (Bezivin, Hammoudi, Lopesm & Jouault, 2004) กล่าว่าการใช้ UML Profile ในการสร้างเมตาโมเดลของ PIM และ PSM นั้นมีข้อเสียบางประการ ดังนี้ 1) ทำให้กฎการแปลงแบบจำลองที่สร้างขึ้นมามีขนาดใหญ่เนื่องจากการสร้างตัวกรอง (Filter) และคำสั่งสำหรับใช้ควบคุมโฟล์ว (Flow) เช่นคำสั่ง if-then-else และลำดับถัดมา 2) การใช้งาน Profile จะไม่ได้รับประโยชน์ของภาษาที่ใช้ในการสร้างกฎการแปลงแบบจำลอง อย่างเช่นภาษา ATL หรือ QVT ได้อย่างเต็มที่

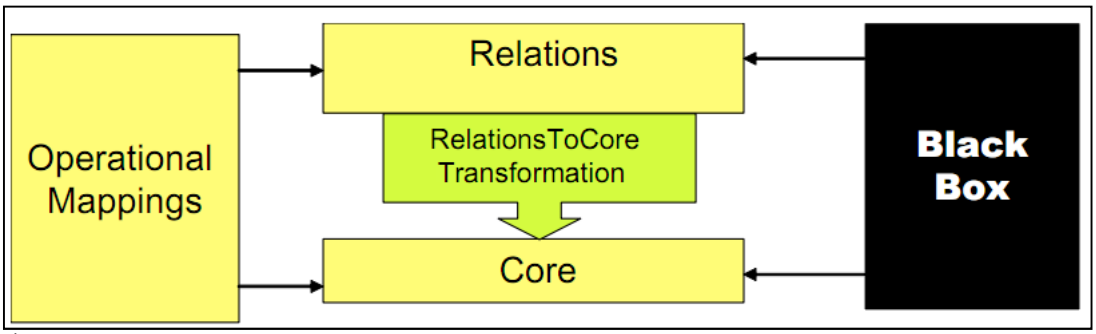
2.1.7 ภาษา Query/View/Transformation

ภาษา Query/View/Transformation คือภาษามาตรฐานสำหรับใช้ในการกำหนดความสัมพันธ์ระหว่างแบบจำลองที่กำหนดขึ้นโดยองค์กร OMG ภาษา QVT ทำหน้าที่ในการกำหนดความสัมพันธ์และการแปลง (Transformation) ระหว่างแบบจำลองอย่างอัตโนมัติ โครงสร้างของภาษา QVT เป็นแบบผสมระหว่างวิธีการแบบ Declarative และ Imperative ดังปรากฏในสถาปัตยกรรมของภาษา QVT ในภาพที่ 2.10

โครงสร้างของภาษาแบบ Declarative เป็นสถาปัตยกรรมแบบสองระดับ (Two-Level Declarative Architecture) ประกอบด้วยภาษา Relations และภาษา Core ซึ่งมีความแตกต่างกันคือภาษา Relations เป็นภาษาระดับสูง (Higher Abstraction) ที่ใช้งานง่าย (User-Friendly) ภาษา Relations ใช้ในการนิยามกฎการแปลงระหว่างแบบจำลอง เป็นภาษาที่รองรับการสร้างแพทเทิร์นของออปเจ็กต์ที่ซับซ้อน ในการค้นหาตำแหน่ง สร้าง หรือปรับปรุงอิลิเมนต์ของแบบจำลองที่เกี่ยวข้องในกระบวนการแปลงได้ กฎการแปลงที่นิยามขึ้นด้วยภาษา Relations จะมีกลไกภายในในการสร้างคลาสสำหรับตามรอย (Trace) และอินสแตนท์ต่าง ๆ เพื่อใช้บันทึกการจับคู่ระหว่างอิลิเมนต์ของแบบจำลองต้นทางและแบบจำลองปลายทางที่เกิดขึ้นในระหว่างที่มีการประมวลผลการแปลงแบบจำลอง ส่วนภาษา Core เป็นภาษาที่อยู่ในระดับล่าง (Lower Level) ที่มีพื้นฐานจากภาษา EMOF และ OCL (Object Constraint Language) เป็นภาษาที่มีความเรียบง่ายกว่าภาษา Relations

ภาพที่ 2.10

สถาปัตยกรรมของภาษา QVT



ที่มา: OMG (2007)

ส่วน Imperative สร้างขึ้นมาเพื่อแก้ปัญหาในกรณีที่วิธีการนิยามกฎแบบ Declarative ไม่สามารถแก้ปัญหาได้ ดังนั้นจึงมีการพัฒนาภาษา Operational Mapping และสร้างกลไกในการเรียกใช้ฟังก์ชันการแปลงที่เรียกว่า Black Box ขึ้น ซึ่งภาษา Operational Mapping ได้ต่อยอดมาจากภาษา Relations และภาษา Core โดยมีแนวคิดคือแพทเทิร์นของออปเจ็กต์ที่นิยามใน relation จะถูกสร้างขึ้นโดยวิธีการแบบ Imperative และไวยากรณ์ที่ใช้ในภาษา Operational Mapping จะมีโครงสร้างของภาษา Imperative เช่น ลูป หรือเงื่อนไข ในส่วนของ Black Box คือกลไกที่อนุญาตให้ประมวลผลโค้ดภายนอกในระหว่างที่ดำเนินการแปลงแบบจำลองได้ ซึ่ง Black Box นี้สามารถพัฒนาโดยใช้ภาษาโปรแกรมมิ่งใดๆ ก็ได้

เพื่อให้เกิดความเข้าใจในโครงสร้างสถาปัตยกรรมของภาษา QVT ได้ชัดเจนขึ้น สามารถเปรียบเทียบสถาปัตยกรรมของภาษา QVT กับสถาปัตยกรรมของภาษาจาวาได้ดังนี้

- ภาษา Relations เทียบได้กับภาษาจาวา (Java) และการนิยามกฎโดยใช้ภาษา Relations เทียบได้กับการสร้างจาวาโค้ด (Java Code)
- ภาษา Core เทียบได้กับจาวาไบท์โค้ด (Java Byte Code)
- การแปลง RelationToCore เทียบได้กับ Java Compiler
- การทำงานของ Core สามารถเทียบได้กับพฤติกรรมของ Java Virtual Machine
- การเรียกใช้ Operational Mapping หรือ Black Box เทียบได้กับการสร้าง JNI (Java Native Interface)

โครงสร้างภาษา Relations

ภาษา Relations เป็นภาษาที่ใช้กำหนดการแปลงระหว่างแบบจำลอง กฎการแปลงใช้วิธีการกำหนดผ่านกลุ่มของ relation โดยในหนึ่ง relation จะประกอบด้วยส่วนสำคัญ 2 ส่วนคือ กลุ่มของโดเมนต้นทาง และกลุ่มของโดเมนปลายทาง การประกาศกลุ่มของโดเมนต้นทางใช้วิธีการกำหนดเป็นแพทเทิร์นของแบบจำลอง (Model Pattern) ซึ่งในกระบวนการแปลงจะมีการตรวจสอบว่าอิลิเมนต์ของแบบจำลองต้นทางนั้นตรงกับแพทเทิร์นของแบบจำลองที่เป็นกลุ่มของโดเมนต้นทางหรือไม่ ถ้าตรงกันก็จะเข้าสู่ขั้นตอนการสร้าง (Generate) กลุ่มของโดเมนปลายทางที่ได้ระบุใน relation นอกจากนี้ในภาษา Relations มีการกำหนดนิพจน์ (Expression) สำหรับใช้กำหนดเงื่อนไขภายใต้ relation ซึ่งประกอบด้วยนิพจน์ when และ where โดยนิพจน์ when ใช้กำหนดเงื่อนไขที่ relation นั้นจะต้องพิจารณาก่อน (Precondition) ส่วนนิพจน์ where ใช้ในการกำหนดเงื่อนไขว่า relation นั้นจะต้องทำตามเงื่อนไขที่ระบุใน where หลังจากที่ดำเนินการตาม relation แล้ว หรือสามารถเรียก where ว่าเป็นเงื่อนไขหลัง (Postcondition) การกำหนดทิศทางของการแปลงทำได้โดยใช้ checkonly หรือ enforce การใช้ checkonly หมายถึง relation นั้นจะถูกเข้าถึงและทำการตรวจสอบในเงื่อนไขที่กำหนดแต่จะไม่มีการทำการแก้ไขใดๆ กับโดเมนนั้น ส่วน enforce จะมีบังคับให้การแก้ไขแบบจำลองตามแพทเทิร์นที่ระบุในโดเมน

โดยตัวอย่าง relation แสดงในภาพที่ 2.11 เป็นส่วนหนึ่งของการแปลงแบบจำลอง UML เป็นแบบจำลอง RDBMS โดย relation ชื่อ ClassToTable มีจุดประสงค์ในการแปลง Class ของ UML เพื่อสร้างเป็น Table ใน RDBMS relation นี้ได้กำหนดโดเมนที่เกี่ยวข้อง 2 โดเมนคือ uml และ rdbms ซึ่งทั้งสองโดเมนจะอ้างอิงไปยังเมตาโมเดล SimpleUML และ SimpleRDBMS ตามลำดับ การอ้างอิงทำได้โดยกำหนดภายใต้ transformation ส่วนของโดเมน uml มีแพทเทิร์นคือ Class จะต้องเป็นประเภท Persistent มีชื่อ และมีการระบุ Package และเช่นเดียวกับโดเมน rdbms กำหนดให้มีแพทเทิร์น Table ที่ต้องระบุชื่อและ Schema โดยในเงื่อนไข when มี relation PackageToSchema หมายถึง relation ClassToTable นี้จะทำการแปลงก็ต่อเมื่อใน Package มีอิลิเมนต์ Class และ ใน Schema มีอิลิเมนต์ Table ส่วนเงื่อนไข where มี relation AttributeToColumn หมายถึงเมื่อมีการแปลง ClassToTable ก็จะต้องแปลง relation AttributeToColumn ด้วย

ภาพที่ 2.11

ตัวอย่าง relation สำหรับการแปลงแบบจำลอง UML เป็นแบบจำลอง RDBMS

```
transformation umlRdbms (uml : SimpleUML, rdbms : SimpleRDBMS){

	relation ClassToTable { // map each persistent class to a table
		cn : String;
		checkonly domain uml c:Class {
			namespace = p:Package {};
			kind = 'Persistent';
			name = cn;
		};
		enforce domain rdbms t:Table {
			schema = s:Schema {};
			name = cn;
		};

		when {
			PackageToSchema(p, s);
		}
		where {
			AttributeToColumn(c, t);
		}
	}
}
```

2.2 คุณสมบัติความปลอดภัยของซอฟต์แวร์

2.2.1 นิยามของความปลอดภัย

NCSC-TG-004 (1988) หรือ National Computer Security Center ได้ประมวลนิยามคำศัพท์ที่เกี่ยวข้องกับความปลอดภัยบนคอมพิวเตอร์ ได้ให้คำจำกัดความการรักษาความปลอดภัยดังนี้

- การรักษาความลับ (Confidentiality) คือแนวคิดสำหรับการควบคุมข้อมูลที่มีความสำคัญให้อยู่ในสถานะที่ไว้ใจได้ โดยที่จะถูกจำกัดอยู่ในกลุ่มคนหรือองค์กรที่มีการจัดสรรไว้เท่านั้น “The concept of holding sensitive data in confidence, limited to an appropriate set of individuals or organizations”
- การรักษาความสมบูรณ์ (Integrity) คือคุณสมบัติของข้อมูลที่มีคุณภาพตามที่ได้คาดหวังไว้ “The property that data meet an a priori expectation of quality.”

- การพิสูจน์ตัวจริง (Authentication) คือการอนุญาตให้ผู้ใช้งานเข้าถึงโปรแกรมหรือกระบวนการถูกต้องตามสิทธิ์ที่ได้รับ “The granting of access rights to a user, program, or process.”

ISO/IEC-17799 (2005) ได้นิยามการรักษาความลับ (Confidentiality) คือการที่แน่ใจได้ว่าสารสนเทศถูกเข้าถึงได้เฉพาะผู้ที่มีสิทธิในการเข้าถึงได้เท่านั้น “ensuring that information is accessible only to those authorized to have access”

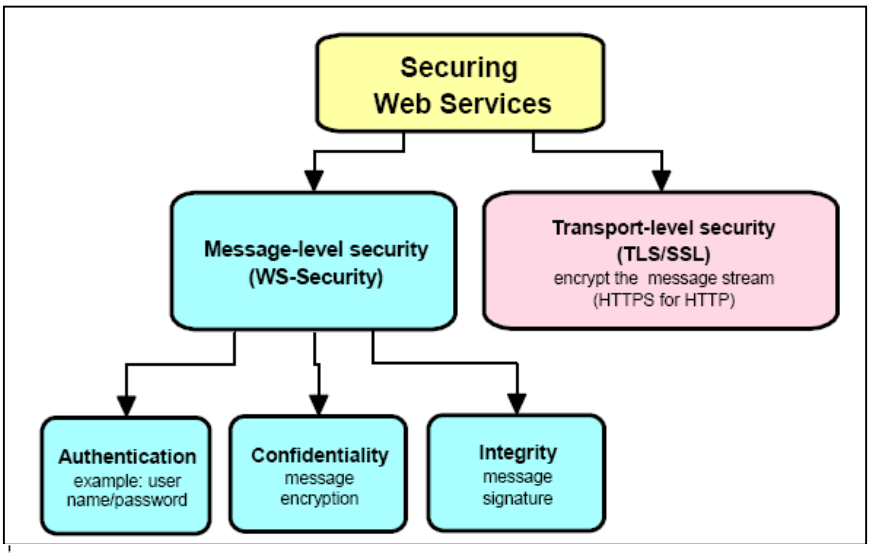
2.2.2 **การรักษาความปลอดภัยของเว็บเซอร์วิส** (Web Service Security : WS-Security)

การรักษาความปลอดภัยบนเว็บเซอร์วิสสามารถแบ่งออกได้เป็น 2 ระดับคือการรักษาความปลอดภัยในระดับข้อความโซ๊ป (SOAP) และการรักษาความปลอดภัยระดับการสื่อสาร (Transport) (Wahli, Kjaer, Robertson et al., 2004) โดยการรักษาความปลอดภัยในระดับข้อความโซ๊ปซึ่งมีมาตรฐานที่ใช้ในการรักษาความปลอดภัยบนเว็บเซอร์วิซโดยมีพื้นฐานบนข้อความโซ๊ปที่เรียกว่า Web Service Security (WS-Security) ได้กำหนดขึ้นในปี ค.ศ. 2002 โดยบริษัทไอบีเอ็มและบริษัทไมโครซอฟต์ และได้มีการปรับปรุงจนออกเป็นมาตรฐานในปี ค.ศ. 2004 โดยองค์กร OASIS (องค์กรที่ดูแลมาตรฐานของเว็บเซอร์วิส) สามารถแบ่งรูปแบบการรักษาความปลอดภัยได้เป็น 3 รูปแบบคือ การพิสูจน์ตัวจริง (Authentication) การรักษาความลับ (Confidentiality) และการรักษาความสมบูรณ์ (Integrity) ดังแสดงในภาพที่ 2.12

โดยในงานวิจัยนี้มุ่งความสนใจการรักษาความปลอดภัยบนเว็บเซอร์วิสโดยใช้มาตรฐาน WS-Security เนื่องจาก WS-Security มีข้อกำหนดในการรักษาความปลอดภัยที่ครอบคลุมในประเด็นความปลอดภัยทั้ง 3 ด้านที่กล่าวถึงในข้อ 2.2.1 ซึ่งจะนำวิธีการรักษาความปลอดภัยทั้ง 3 รูปแบบดังกล่าวมาเป็นข้อกำหนดในการรักษาความปลอดภัยของแบบจำลอง PSM ซึ่งเป็นแบบจำลองปลายทาง

ภาพที่ 2.12

การรักษาความปลอดภัยของเว็บเซอร์วิส



ที่มา: Wahli, Kjaer, Robertson et al. (2004)

การกำหนดความปลอดภัยลงไปในเว็บเซอร์วิส กำหนดโดยระบุอิลิเมนต์ลงไปในข้อความโซ๊ป โดยรูปแบบของการรักษาความปลอดภัยแยกออกเป็น 3 แบบดังนี้

- การใช้ Security Token เพื่อใช้ในใช้ในการรับรองสิทธิ์และการตรวจสอบสิทธิ์ เช่นใช้ระบุความเป็นตัวตนของผู้ใช้งานว่าเป็นผู้ที่มีสิทธิ์ใช้งานเว็บเซอร์วิสนั้นหรือไม่ ซึ่งการใช้ Security Token สามารถแบ่งรูปแบบการพัฒนาได้หลายแบบเช่นใช้ UsernameToken หรือ BinarySecurityToken
- การใช้ XML Encryption คือการเข้ารหัสข้อความ ซึ่งเป็นข้อกำหนดที่พัฒนาขึ้นมาโดยองค์กร W3C (World Wide Web Consortium) เพื่อช่วยเพิ่มความสามารถในการรักษาความลับ โดยการเข้ารหัสข้อมูลในข้อความโซ๊ปซึ่งเป็นวิธีการในการป้องกันการอ่านข้อความจากผู้ใช้งานที่ไม่มีสิทธิ์ หมายถึงถ้าไม่รหัสที่ใช้ในการถอดรหัสก็จะไม่สามารถอ่านข้อความได้ โดยที่การเข้ารหัสอาจจะทำได้ทั้งหมดหรือเป็นการเข้ารหัสแค่บางส่วนของข้อความก็ได้
- การใช้ XML Signature คือการใช้ลายเซ็นดิจิทัล (Digital Signature) ซึ่งมีจุดประสงค์ข้อ 1) เพื่อใช้ตรวจสอบเอกสารสิทธิ์ที่ได้รับมา อย่างเช่นใบรับรองอิเล็กทรอนิกส์ของ X.509 ว่าข้อความไม่ได้ถูกแก้ไขในระหว่างที่รับส่งเพื่อให้มั่นใจได้ว่าข้อความนั้นมี

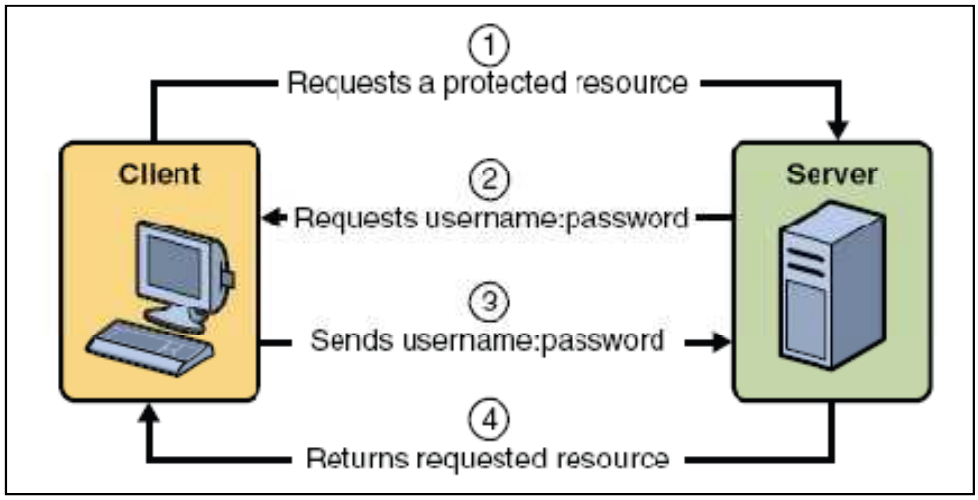
ความถูกต้องสมบูรณ์ โดยในการตรวจสอบเอกสารสิทธิ์จะใช้ลายเซ็นรวมกันกับใบรับรองอิเล็กทรอนิกส์ และข้อ 2) เพื่อการตรวจสอบว่าผู้ใช้งานที่อ้างถึงนั้นเป็นใคร ซึ่งข้อกำหนดนี้ถูกกำหนดขึ้นมาโดยองค์กร W3C เช่นเดียวกัน

2.2.3 **การรักษาความปลอดภัยของจาวาเซิร์ฟเล็ต** (Java Servlet)

จาวาเซิร์ฟเล็ต (Java Servlet) เป็นเว็บคอมโพเนนท์ (Web Component) หนึ่งตามข้อกำหนดของแพลตฟอร์ม J2EE สำหรับใช้พัฒนาเว็บแอพพลิเคชั่น การรักษาความปลอดภัยของจาวาเซิร์ฟเล็ตครอบคลุมความต้องการด้านการพิสูจน์ตัวจริง การควบคุมการเข้าถึงทรัพยากร (Access Control) การรักษาความสมบูรณ์ และการรักษาความลับ

ข้อกำหนดการจัดการเรื่องความปลอดภัยของเซิร์ฟเล็ต ทำได้โดยเพิ่มอิลิเมนต์ที่เป็นส่วนของการจัดการด้านความปลอดภัยกำกับในไฟล์ Deployment Descriptor หรือ web.xml ตัวอย่างกลไกการพิสูจน์ตัวจริงของเซิร์ฟเล็ต คือเมื่อไคล์เอนท์ต้องการเข้าถึงทรัพยากรใดๆ ก็ส่งการร้องขอมายังเว็บเซิร์ฟเวอร์ เซิร์ฟเวอร์ก็จะส่งการร้องขอรหัสผู้ใช้และรหัสผ่านมายังไคล์เอนท์เพื่อให้ไคล์เอนท์ระบุผู้ใช้งานแล้วส่งข้อมูลกลับมาให้เซิร์ฟเวอร์ตรวจสอบอีกครั้ง ซึ่งถ้าตรวจสอบแล้วผู้ใช้งานนั้นมีสิทธิ์ ไคล์เอนท์ก็สามารถเรียกใช้งานทรัพยากรนั้นได้ ตัวอย่างขั้นตอนในการพิสูจน์ตัวจริงแบบพื้นฐานของโปรโตคอล HTTP แสดงดังภาพที่ 2.13

ภาพที่ 2.13
ตัวอย่างการพิสูจน์ตัวจริงโดยใช้มาตรฐานพื้นฐานของโปรโตคอล HTTP



ที่มา: “The Java EE 5 Tutorial”, Oracle, 2010, pp. 856

2.3 งานวิจัยที่เกี่ยวข้อง

ในส่วนนี้กล่าวถึงงานวิจัยที่เกี่ยวข้อง สามารถแบ่งงานวิจัยที่เกี่ยวข้องได้คือการเพิ่มคุณสมบัติความปลอดภัยในแบบจำลอง และการแปลงแบบจำลองเชิงฟังก์ชันของระบบที่ครอบคลุมคุณสมบัติด้านความปลอดภัย โดยมีรายละเอียดดังต่อไปนี้

2.3.1 การเพิ่มคุณสมบัติความปลอดภัยในแบบจำลอง

งานวิจัยที่เกี่ยวข้องกับการเพิ่มคุณสมบัติด้านความปลอดภัยของแบบจำลอง มีหลายงานวิจัยที่ได้มุ่งความสนใจทั้งในเรื่องของวิธีการในการสร้างแบบจำลองเพื่อให้มีคุณสมบัติด้านความปลอดภัย และประเด็นสนใจในประเด็นของความปลอดภัยที่ต้องการจะให้แบบจำลองนั้นรองรับ โดยมีรายละเอียดของงานวิจัยที่เกี่ยวข้องดังนี้

1. Jurjens (2002) ได้เสนอ UMLSec เป็น UML Profile ที่มีคุณสมบัติในด้านความปลอดภัยโดยเน้นไปที่การรักษาความปลอดภัยสำหรับใช้ในระบบแบบกระจาย (Distributed System) ที่กำหนดคุณสมบัติความปลอดภัยโดยใช้ UML Profile ผ่านทางการสร้าง Stereotype และ Tagged Value ซึ่งการขยายแบบจำลอง UML ให้มีคุณสมบัติด้านความปลอดภัยสำหรับระบบแบบกระจายของ Jurjens นั้น ได้เน้นที่ความปลอดภัยบนเส้นทางการสื่อสาร ซึ่งโครงสร้างของ UMLsec ที่นำเสนอ แสดงในภาพที่ 2.14 อย่างเช่น กำหนด Stereotype ชื่อ encrypted สำหรับใช้อธิบายว่าการเชื่อมโยง (Connection) นั้นมีการเข้ารหัสด้วย จุดประสงค์ของงานวิจัยเพื่อต้องการให้นักวิเคราะห์และออกแบบระบบสามารถที่จะนำความต้องการที่ไม่ใช่เชิงฟังก์ชั่นหลักของระบบ (Non-Functional Requirement) เพิ่มเติมลงไปในแบบจำลองต่างๆ ได้ เช่นแผนภาพดีพลอยเม็นท์ (Deployment Diagram) หรือแผนภาพซีเควนซ์ (Sequence Diagram) ทำให้เกิดประโยชน์แก่นักพัฒนาในการที่พัฒนาแอพพลิเคชั่นได้ครอบคลุมประเด็นของความต้องการที่ไม่ใช่เชิงฟังก์ชันหลักด้วย

โดยในงานวิจัยนี้ได้นำแนวคิดของ Jurjens ในการขยายแบบจำลองให้มีคุณสมบัติความปลอดภัยด้านอื่นๆ นอกเหนือจากที่งานวิจัยของ Jurjens ได้นำเสนอไปแล้ว โดยแบบจำลองที่มีการขยายให้มีคุณสมบัติด้านความปลอดภัยเข้าสู่กระบวนการพัฒนาซอฟต์แวร์แบบ MDA ซึ่ง

จะแตกต่างจากงานวิจัยของ Jurjens ตรงที่ Jurjens นำเสนอเฉพาะในส่วนของ UML Profile สำหรับใช้ในการออกแบบแบบจำลองเท่านั้น

ภาพที่ 2.14

UMLsec Stereotype

Stereotype	Base Class	Tags	Constraints	Description
Internet	link			Internet connection
encrypted	link			encrypted connection
LAN	link			LAN connection
secure links	subsystem		dependency security matched by links	enforces secure communication links
secrecy	dependency			assumes secrecy
secure dependency	subsystem		« call », « send » respect data security	structural interaction data security
critical	object	secret		critical object
no down-flow	subsystem		prevents down-flow	information flow
data security	subsystem		provides secrecy	basic datasec requirements
fair exchange	package	start,stop	after start eventually reach stop	enforce fair exchange

ที่มา: Jurjens (2002)

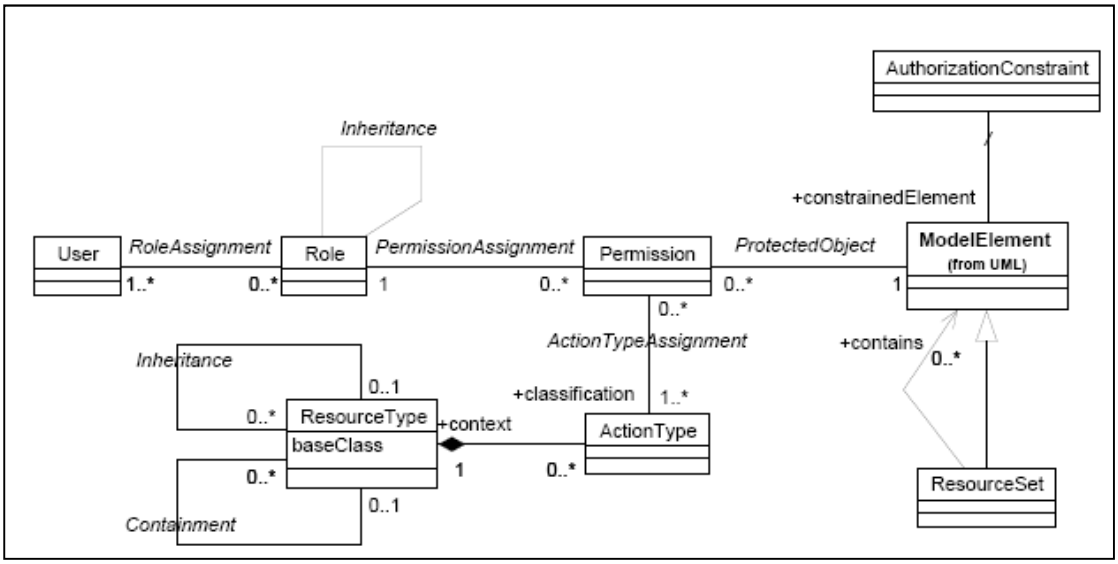
2. Lodderstedt et al. (2002) ได้เสนอ SecureUML เป็นการกำหนดเมตาโมเดล (Metamodel) และ UML Profile สำหรับจัดการเรื่อง Role-Based Access Control (RBAC) แนวคิดของงานวิจัยนี้เหมือนกับแนวคิดในงานวิจัยของ Jurjens (2002) คือต้องการขยายแบบจำลอง UML ให้มีคุณสมบัติด้านความปลอดภัย เพื่อให้นักออกแบบสามารถนำไปใช้งานพัฒนาแอพพลิเคชั่นที่มีคุณสมบัติความปลอดภัยได้ แต่งานของ Lodderstedt et al. และ Jurjens ต่างกันตรงที่งานของ Lodderstedt et al. เน้นที่ความปลอดภัยเรื่อง RBAC และได้นำเสนอวิธีการนำเมตาโมเดลและ UML Profile ไปใช้งานในกระบวนการพัฒนาซอฟต์แวร์แบบ MDA ด้วย ในขณะที่ Jurjens เสนอเฉพาะการนำ UMLsec ไปออกแบบแบบจำลอง การพัฒนากรณีศึกษาเพื่อแสดงตัวอย่างการนำ SecureUML ไปใช้ในการพัฒนาแอพพลิเคชั่นเป็นแพลตฟอร์ม EJB (Enterprise JavaBeans)

กระบวนการในการสร้างเมตาโมเดลของ Lodderstedt et al. ใช้วิธีการวิเคราะห์โดยนำหลักการของ RBAC มาวิเคราะห์และออกแบบเมตาคลาส จนได้โครงสร้างเมตาโมเดลของ SecureUML ดังแสดงในภาพที่ 2.15 ดังนั้นในงานวิจัยนี้ได้ใช้แนวคิดแบบเดียวกันนี้ในการสร้างเมตาโมเดลทั้งหมดที่ใช้ในงานวิจัย และได้นำหลักการของ OOAD มาร่วมวิเคราะห์และออกแบบ

ด้วย นอกจากนี้ในงานวิจัยของ Lodderstedt et al. ได้แสดงตัวอย่างการพัฒนาแอพพลิเคชั่นที่ใช้แนวคิดของการแปลงแบบจำลอง Model-to-Model ด้วย แต่ในงานวิจัยนี้ก็มีข้อจำกัดอยู่ในเรื่องรองรับความปลอดภัยเฉพาะ RBAC เท่านั้น ดังนั้นในงานวิจัยนี้จึงนำเสนอความปลอดภัยด้านอื่นๆ เช่นการพิสูจน์ตัวจริง การรักษาความลับ และการรักษาความสมบูรณ์

ภาพที่ 2.15

เมตาโมเดลของ SecureUML



ที่มา: Lodderstedt et al. (2002)

2.3.2 การแปลงแบบจำลองที่ครอบคลุมคุณสมบัติด้านความปลอดภัย

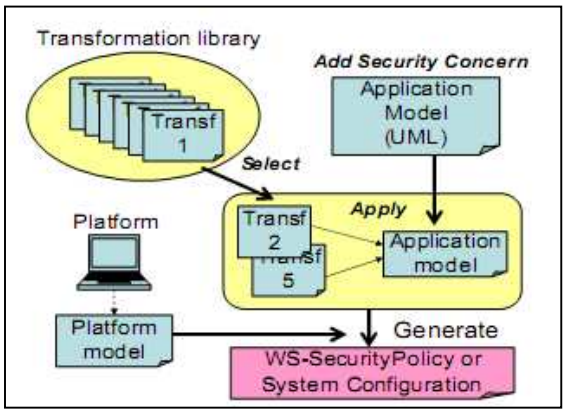
1. Nakamura et al. (2005) ได้เสนอเฟรมเวิร์คในการสร้างไฟล์คอนฟิคกูเรชั่นที่เป็นส่วนของการรักษาความปลอดภัยบนเว็บเซอร์วิส โดยใช้หลักการพัฒนาซอฟต์แวร์แบบ MDA ภาพรวมของเครื่องมือแสดงในภาพที่ 2.16 Nakamura et al. ได้เสนอวิธีการพัฒนาซอฟต์แวร์ด้วยแนวคิดของ MDA พร้อมกับเสนอวิธีการในการเพิ่มคุณสมบัติด้านความปลอดภัยสำหรับแอพพลิเคชั่นเว็บเซอร์วิส โดยทำการเพิ่มคุณสมบัติด้านความปลอดภัยโดยใช้วิธีการเพิ่มบางอิลิเมนต์ของ UML ในแบบจำลองของแอพพลิเคชั่น จากนั้นก็นำเข้าสู่กระบวนการแปลงแบบจำลองซึ่งผลลัพธ์ก็จะได้ไฟล์คอนฟิคกูเรชั่นต่างๆ สำหรับเว็บเซอร์วิส

โดยงานวิจัยของ Nakamura et al. แตกต่างจากงานวิจัยนี้ตรงที่งานของ Nakamura et al. ใช้แนวคิดการแปลงแบบจำลองแบบ Model-to-Code ซึ่งทำให้แบบจำลองและชุดไลบรารี่ที่ใช้ในการแปลงแบบจำลองนั้นมีจุดด้อยตรงที่ค่อนข้างยึดติดกับแพลตฟอร์มและเทคโนโลยีที่เลือกใช้ โดยเฟรมเวิร์กที่ Nakamura et al. ได้นำเสนอนั้นจะเน้นไปที่การสร้างไฟล์คอนฟิคกูเรชั่นที่จัดการเรื่องการรักษาความปลอดภัยของเว็บเซอร์วิสที่เฉพาะเจาะจงกับ Websphere Application Server (WAS) ดังนั้นเพื่อให้แบบจำลองที่สร้างขึ้นสามารถนำไปใช้งานซ้ำได้และไม่ยึดติดกับเฟรมเวิร์กใดๆ งานวิจัยนี้จึงได้นำเสนอการสร้างแบบจำลอง และเลือกวิธีการแปลงแบบจำลองแบบ Model-to-Model

นอกจากนี้ ประเด็นเรื่องความปลอดภัยที่ Nakamura et al.ได้นำเสนอนั้น เป็นความปลอดภัยพื้นฐานที่อ้างอิงมาตรฐานมาจากการรักษาความปลอดภัยของเว็บเซอร์วิสเช่นเดียวกันกับในงานวิจัยนี้ แต่ Nakamura et al.ได้กำหนดเฟรมเวิร์กการแปลงแบบจำลองในระดับ Model-to-Code ดังนั้นจึงไม่ได้มีการสร้างเมตาโมเดลสำหรับความปลอดภัยในระดับของ PIM แต่จะใช้วิธีการนำคุณสมบัติความปลอดภัยเพิ่มเติมลงไปในคลาสไดอะแกรมในแบบจำลองระดับ PSM เลย โดยใช้วิธีการกำหนดผ่าน Stereotype แล้วจึงแปลงเป็นซอร์สโค้ดและไฟล์คอนฟิคกูเรชั่นต่างๆ ดังนั้นประเด็นของความปลอดภัยในงานวิจัยของ Nakamura et al. ก็จะแตกต่างกับกับงานวิจัยของ Lodderstedt et al. และ Jurjens ตรงที่เป็นความปลอดภัยคนละด้านและคนละระดับกัน ซึ่งในงานของ Lodderstedt et al. งานวิจัยของ Jurjens รวมถึงในงานวิจัยนี้ได้เน้นไปที่มุมมองความปลอดภัยในระดับของการออกแบบแบบจำลอง

ภาพที่ 2.16

แสดงภาพรวมของเฟรมเวิร์ก



ที่มา: Nakamura et al. (2005)