วิธีการสกัดรูปแบบการใช้งานของอ็อบเจกต์โดยการค้นคืน ด้วยชนิดของบริบทที่หลากหลาย

นางสาวปารัช สุพงษ์พันธุ์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมซอฟต์แวร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2553

OBJECT USAGE PATTERN EXTRACTION USING VARIOUS TYPES OF CONTEXT SENSITIVE RETRIEVAL

Miss Parat Supongpan

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science Program in Software Engineering

Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University

Academic Year 2010

หัวข้อวิทยานิพนธ์	วิธีการสกัดรูปแบบการใช้งานของอ็อบเจกต์โดยการค้นคืนด้วยชนิดของบริบทที่หลากหลาย
โดย	นางสาว ปารัช สุพงษ์พันธุ์
สาขาวิชา	วิทยาศาสตร์คอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	รองศาสตราจารย์ ดร.พรศิริ หมื่นไชยศรี
อาจารย์ที่ปรึกษาวิทยานิพนธ์ร่วม	ดร.นัยนา สหเวชชภัณฑ์

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

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

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(รองศาสตราจารย์ ดร.วันชัย ริ้วไพบูลย์)

..... อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก
(รองศาสตราจารย์ ดร.พรศิริ หมื่นไชยศรี)

..... อาจารย์ที่ปรึกษาวิทยานิพนธ์ร่วม
(ดร.นัยนา สหเวชชภัณฑ์)

..... กรรมการ
(รองศาสตราจารย์ ดร.วิวัฒน์ วัฒนาวุฒิ)

..... กรรมการภายนอกมหาวิทยาลัย
(ผู้ช่วยศาสตราจารย์ ดร.ทรงศักดิ์ รองวิริยะพานิช)

ปารัช สุพงษ์พันธุ์ : วิธีการสกัดรูปแบบการใช้งานของอ็อบเจกต์โดยการค้นคืนด้วยชนิดของบริบทที่หลากหลาย. (OBJECT USAGE PATTERN EXTRACTION USING VARIOUS TYPES OF CONTEXT SENSITIVE RETRIEVAL) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: รศ. ดร.พรศิริ หมื่นไชยศรี, อ.ที่ปรึกษาวิทยานิพนธ์ร่วม ดร.นัยนา สหเวชภัณฑ์ 156 หน้า.

ในปัจจุบัน นักพัฒนาซอฟต์แวร์นิยมใช้กรอบงานและไลบรารีมาช่วยพัฒนาซอฟต์แวร์ให้เสร็จสิ้นตามระยะเวลาที่กำหนด แต่กรอบงานและไลบรารีเหล่านี้เรียนรู้และใช้งานยาก เนื่องจากเอกสารการใช้งานไม่ได้อธิบายวิธีการใช้อย่างละเอียด อีกทั้งรูปแบบการใช้งานของกรอบงานและไลบรารีจะซับซ้อน และประกอบไปด้วยคลาสจำนวนมาก นักพัฒนาซอฟต์แวร์จึงนิยมใช้โค้ดตัวอย่างเพื่อช่วยลดระยะเวลาในการเรียนรู้การใช้งาน และเพื่อเป็นแนวทางในการพัฒนาซอฟต์แวร์ วิทยานิพนธ์ฉบับนี้จึงนำเสนอวิธีการสกัดรูปแบบการใช้งานโค้ดจากคลังข้อมูล เพื่อแนะนำตัวอย่างรูปแบบการใช้งานที่มีบริบทโค้ดใกล้เคียงกับโปรแกรมที่กำลังพัฒนาให้แก่นักพัฒนาซอฟต์แวร์

ประโยชน์ที่ได้รับจากวิทยานิพนธ์นี้คือ (1) วิธีการสกัดรูปแบบการใช้งานอ็อบเจกต์ (2) ประเภทของการสืบค้นรูปแบบการใช้งานอ็อบเจกต์โดยใช้บริบทโค้ดชนิดต่างๆ ช่วยคัดกรองผลลัพธ์ จากการทดลองพบว่า วิธีการที่นำเสนอสามารถตอบโจทย์ได้ครบถ้วน และสามารถเรียงอันดับผลลัพธ์ตามความสอดคล้องกับซอฟต์แวร์ที่กำลังพัฒนาได้เป็นอย่างดี

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต ปารัช สุพงษ์พันธุ์

สาขาวิชาวิศวกรรมคอมพิวเตอร์ ลายมือชื่อ อ. ที่ปรึกษาวิทยานิพนธ์หลัก

ปีการศึกษา 2553 ลายมือชื่อ อ. ที่ปรึกษาวิทยานิพนธ์ร่วม

5070347721 : MAJOR COMPUTER SCIENCE

KEYWORDS : CODE REUSE / CODE QUERY TOOL / USAGE PATTERN

PARAT SUPONGPAN : OBJECT USAGE PATTERN EXTRACTION USING VARIOUS TYPES OF CONTEXT SENSITIVE RETRIEVAL. ADVISOR : ASSOC.PROF. PORNSIRI MUENCHAISRI, Ph.D. Co - ADVISOR : NAIYANA SAHAVECHAPAN, Ph.D., 156 pp.

Today, software developers rely on frameworks and libraries to create highly qualified and full-featured applications on-time. These frameworks and libraries, however, cause a steep learning curve due to the sheer number of classes and complex APIs. It is thus common practice for programmers to use code samples to guide their software development effort. To assist programmers, in this work, I have extended previous work to enable programmers to query a sample code repository for usage patterns relevant to the programming task at hand.

In particular, this thesis has two contributions (i) the innovative extraction algorithms that accommodate the range of queries and constrains the extraction process as needed to answer the usage patterns of a given object type; and (ii) the various degrees of context-sensitive queries for providing best-fit usage patterns. The experiment has shown that the methodology has significant potential to solve the programming tasks, and to perform well coverage of tasks and rankings for best-fit usage patterns.

Department : Computer Engineering
Field of Study : Computer Science
Academic Year : 2010

Student's Signature : Parat Supongpan
Advisor's Signature : Pornsiri Muenchaisri
Co-Advisor's Signature : Naiyana Sahavechaphan .

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ด้วยดี เนื่องมาจากความช่วยเหลืออย่างดียิ่งของท่านรองศาสตราจารย์ ดร.พรศิริ หมื่นไชยศรี และดร.นัยนา สหเวชภัณฑ์ อาจารย์ที่ปรึกษาวิทยานิพนธ์ ทั้งสองท่านที่สละเวลาให้คำปรึกษาแนะนำแนวทางเกี่ยวกับวิทยานิพนธ์อย่างดีตลอดมาจนเสร็จสมบูรณ์ และผู้วิจัยขอกราบขอบพระคุณคณะกรรมการสอบวิทยานิพนธ์ ได้แก่ รองศาสตราจารย์ ดร.วันชัย ริ้วไพบูลย์ รองศาสตราจารย์ ดร. วิวัฒน์ วัฒนาวุฒิ รองศาสตราจารย์ ดร. พรศิริ หมื่นไชยศรี และผู้ช่วยศาสตราจารย์ ดร.ทรงศักดิ์ รองวิริยะพานิช ที่ได้ให้คำแนะนำข้อคิดเห็น ข้อเสนอแนะ และแนวทางในการพัฒนาวิทยานิพนธ์

ขอขอบคุณ พี่ธุรการภาคฯ และเพื่อนๆ ในแล็ปทุกคนที่ช่วยอำนวยความสะดวกในการทำงานและช่วยตักเตือนแนะนำสิ่งดีๆ เสมอมา

สุดท้ายนี้ ขอกราบขอบพระคุณทุกคนในครอบครัวที่คอยให้กำลังใจ ช่วยเหลือและสนับสนุนในด้านการศึกษาเป็นอย่างดีเสมอมา

สารบัญ

หน้า

บทคัดย่อภาษาไทย ..... ง
บทคัดย่อภาษาอังกฤษ ..... จ
กิตติกรรมประกาศ ..... ฉ
สารบัญ ..... ช
สารบัญตาราง ..... ฌ
สารบัญภาพ ..... ญ
บทที่ 1 บทนำ 1
1. ที่มาและความสำคัญของปัญหา 1
2. วัตถุประสงค์ของวิทยานิพนธ์ 4
3. ขอบเขตของวิทยานิพนธ์ 4
4. วิธีการดำเนินงานวิจัย 5
5. ประโยชน์ที่คาดว่าจะได้รับ 6
บทที่ 2 ทฤษฎีและงานวิจัยที่เกี่ยวข้อง 7
1. ทฤษฎีที่เกี่ยวข้อง 7
2. งานวิจัยที่เกี่ยวข้อง 40
บทที่ 3 วิธีออกแบบและพัฒนาการสกัดรูปแบบการใช้งานอ็อบเจกต์ 46
1. ข้อมูลนำเข้าของระบบ 47
2. การสร้างคำร้องขอ (Query Formulation) 64
3. การสกัดรูปแบบการใช้งาน (Usage Pattern Extraction) 67
4. การจัดอันดับรูปแบบการใช้งาน (Usage Pattern Ranking) 71
5. การแปลงผลลัพธ์ให้อยู่ในรูปแบบของโค้ด (Code Format Transformation) 79
บทที่ 4 การออกแบบและพัฒนาเครื่องมือ 81
1. สภาพแวดล้อมที่ใช้พัฒนาเครื่องมือ 81
2. การออกแบบและพัฒนาเครื่องมือ 81
3. การออกแบบหน้าจอเครื่องมือ 98
บทที่ 5 การทดสอบเครื่องมือ 101
1. ตัวอย่างโจทย์โปรแกรมที่ใช้ทดสอบ 101
2. การทดสอบเครื่องมือ 133

หน้า

บทที่ 6 สรุปผลการวิจัย 136
1. สรุปผล 136
2. ข้อจำกัดของเครื่องมือและแนวทางการพัฒนาต่อ..... 136
3. ผลงานที่เกี่ยวข้องกับวิทยานิพนธ์ 137
รายการอ้างอิง..... 138
ภาคผนวก
ภาคผนวก ก แบบจำลองโค้ดที่ใช้ในวิทยานิพนธ์..... 143
1. แบบจำลองโหนด (Node Model)..... 144
2. แบบจำลองเอดจ์ (Edge Model)..... 148
ภาคผนวก ข คู่มือการใช้งาน..... 154
1. สภาพแวดล้อมที่จำเป็นในการติดตั้งเครื่องมือ XSNIPPETUSAGE 154
2. การติดตั้งเครื่องมือ 154
ประวัติผู้เขียนวิทยานิพนธ์ 156

สารบัญตาราง

หน้า

ตารางที่ 2.1 ค่าของข้อมูลที่เป็นไปได้แต่ละชนิดข้อมูล 9
ตารางที่ 2.2 ค่าเริ่มต้นสำหรับแต่ละชนิดข้อมูล 9
ตารางที่ 3.1 คำอธิบายอิลิเมนท์ในเอ็กซ์เอ็มแอลสกีมาสำหรับจัดเก็บซอร์สโค้ด 53
ตารางที่ 3.1 คำอธิบายอิลิเมนท์ในเอ็กซ์เอ็มแอลสกีมาสำหรับจัดเก็บซอร์สโค้ด (ต่อ) 54
ตารางที่ 3.1 คำอธิบายอิลิเมนท์ในเอ็กซ์เอ็มแอลสกีมาสำหรับจัดเก็บซอร์สโค้ด (ต่อ) 55
ตารางที่ 3.1 คำอธิบายอิลิเมนท์ในเอ็กซ์เอ็มแอลสกีมาสำหรับจัดเก็บซอร์สโค้ด (ต่อ) 56
ตารางที่ 3.2 ตารางสรุปหมายเลขดิวอีของชนิดข้อมูลในรูปที่ 1.1 62
ตารางที่ 3.2 ตารางสรุปหมายเลขดิวอีของชนิดข้อมูลในรูปที่ 1.1 (ต่อ) 63
ตารางที่ 3.3 บริบทชนิดต่างๆ ในรูปที่ 3.2 65
ตารางที่ 3.4 จำนวนรูปแบบการใช้งานอ็อบเจกต์ที่พบในคลังข้อมูลของในรูปที่ 1.1 73
ตารางที่ 3.5 จำนวนคำสั่งในรูปแบบการใช้งานอ็อบเจกต์ในรูปที่ 1.1 73
ตารางที่ 3.6 สรุปค่าวิทยาการสำนึกและอันดับผลลัพธ์ของรูปแบบการใช้งานอ็อบเจกต์ 79
ตารางที่ 4.1 รายละเอียดยูสเคสเรียกดูรูปแบบการใช้งานอ็อบเจกต์ 83
ตารางที่ 4.2 รายละเอียดยูสเคสสกัดบริบทโค้ด 84
ตารางที่ 4.3 รายละเอียดยูสเคสเรียกข้อมูลแบบจำลองโค้ด 85
ตารางที่ 4.4 รายละเอียดยูสเคสสกัดรูปแบบการใช้งาน 86
ตารางที่ 4.5 รายละเอียดยูสเคสจัดอันดับรูปแบบการใช้งาน 87
ตารางที่ 4.6 รายละเอียดยูสเคสสร้างคลังข้อมูล 88
ตารางที่ 5.1 ตารางสรุปคุณลักษณะของโจทย์โปรแกรมที่ใช้ทดสอบทั้งหมด 19 โจทย์ 102
ตารางที่ 5.3 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ ASTRewrite 105
ตารางที่ 5.3 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ ASTRewrite (ต่อ) 106
ตารางที่ 5.3 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ ASTRewrite (ต่อ) 107
ตารางที่ 5.4 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ IWorkingCopyManager 115
ตารางที่ 5.4 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ IWorkingCopyManager (ต่อ) 116
ตารางที่ 5.5 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ IWorkingCopyManager 122
ตารางที่ 5.5 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ IWorkingCopyManager (ต่อ) 123
ตารางที่ 5.5 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ IWorkingCopyManager (ต่อ) 124
ตารางที่ 5.6 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ ITextEditor 127

สารบัญภาพ

หน้า

รูปที่ 1.1 รูปแบบการใช้งานแบบต่างๆ ของอ็อบเจกต์ ASTRewrite 3
รูปที่ 1.1 รูปแบบการใช้งานแบบต่างๆ ของอ็อบเจกต์ ASTRewrite (ต่อ) 4
รูปที่ 2.1 ไวยากรณ์ชนิดข้อมูล 7
รูปที่ 2.2 ไวยากรณ์สำหรับการประกาศตัวแปรโดยใช้ชนิดข้อมูลแบบพื้นฐาน 8
รูปที่ 2.3 ไวยากรณ์สำหรับการประกาศตัวแปรโดยใช้ชนิดข้อมูลแบบอ้างอิง 11
รูปที่ 2.4 ไวยากรณ์สำหรับการนิยามไอเด็นติฟายเออร์ 12
รูปที่ 2.5 ตัวอย่างการประกาศตัวแปรโดยใช้ชนิดข้อมูลแบบอ้างอิง 12
รูปที่ 2.6 ตัวอย่างการประกาศตัวแปรโดยใช้ชนิดข้อมูลแบบอ้างอิงด้วยอาเรย์ 13
รูปที่ 2.7 ตัวอย่างของการใช้งานอ็อบเจกต์ในภาษาจาวา 14
รูปที่ 2.8 ตัวอย่างของการใช้งานอ็อบเจกต์ในภาษาจาวา 15
รูปที่ 2.9 ตัวอย่างของการใช้งานอ็อบเจกต์ในภาษาจาวา 16
รูปที่ 2.10 ตัวอย่างของประเภทของตัวแปรประเภทต่างๆ 17
รูปที่ 2.11 ไวยากรณ์ภาษาจาวาสำหรับการนิยามคลาส 19
รูปที่ 2.11 ไวยากรณ์ภาษาจาวาสำหรับการนิยามคลาส (ต่อ) 20
รูปที่ 2.11 ไวยากรณ์ภาษาจาวาสำหรับการนิยามคลาส (ต่อ) 21
รูปที่ 2.11 ไวยากรณ์ภาษาจาวาสำหรับการนิยามคลาส (ต่อ) 22
รูปที่ 2.11 ไวยากรณ์ภาษาจาวาสำหรับการนิยามคลาส (ต่อ) 23
รูปที่ 2.11 ไวยากรณ์ภาษาจาวาสำหรับการนิยามคลาส (ต่อ) 24
รูปที่ 2.11 ไวยากรณ์ภาษาจาวาสำหรับการนิยามคลาส (ต่อ) 25
รูปที่ 2.11 ไวยากรณ์ภาษาจาวาสำหรับการนิยามคลาส (ต่อ) 26
รูปที่ 2.12 ตัวอย่างรูปแบบการนิยามคลาสโดยทั่วไปตามไวยากรณ์ภาษาจาวาแบบง่าย 27
รูปที่ 2.13 ตัวอย่างการนิยามคลาสโดยกำหนดค่า Class modifiers เป็น abstract 28
รูปที่ 2.14 ตัวอย่างการนิยามอินเนอร์คลาส 29
รูปที่ 2.15 ตัวอย่างการนิยามคลาสโดยการสืบทอดคุณสมบัติจากคลาสอื่น 30
รูปที่ 2.16 ตัวอย่างการนิยามคลาสโดยการสืบทอดคุณสมบัติจากคลาสและอินเตอร์เฟสคลาส. 31
รูปที่ 2.17 ตัวอย่างการนิยามอินเตอร์เฟสและการสืบทอดคุณสมบัติอินเตอร์เฟส 32
รูปที่ 2.17 ตัวอย่างการนิยามอินเตอร์เฟสและการสืบทอดคุณสมบัติอินเตอร์เฟส (ต่อ) 33

หน้า

รูปที่ 2.18 ตัวอย่างลำดับชั้นของคลาสและหมายเลขดิวอี 37
รูปที่ 2.20 อัลกอริทึมของจับคู่ชนิดข้อมูล 38
รูปที่ 2.20 อัลกอริทึมของจับคู่ชนิดข้อมูล (ต่อ)..... 39
รูปที่ 2.21 อัลกอริทึมของไทป์แมทช์เฮลเปอร์..... 39
รูปที่ 2.21 อัลกอริทึมของไทป์แมทช์เฮลเปอร์ (ต่อ) 40
รูปที่ 2.22 แสดงการเปรียบเทียบระหว่างโค้ดแบบลำดับและแบบพาร์เซียลออเดอร์..... 42
รูปที่ 3.3 เอ็กซ์เอ็มแอลสกีมาสำหรับจัดเก็บซอร์สโค้ด..... 50
รูปที่ 3.3 เอ็กซ์เอ็มแอลสกีมาสำหรับจัดเก็บซอร์สโค้ด (ต่อ) 51
รูปที่ 3.3 เอ็กซ์เอ็มแอลสกีมาสำหรับจัดเก็บซอร์สโค้ด (ต่อ) 52
รูปที่ 3.5 ตัวอย่างไฟล์เอ็กซ์เอ็มแอลที่จัดเก็บซอร์สโค้ด 58
รูปที่ 3.5 ตัวอย่างไฟล์เอ็กซ์เอ็มแอลที่จัดเก็บซอร์สโค้ด (ต่อ)..... 59
รูปที่ 3.6 หมายเลขดิวอีของชนิดข้อมูลในรูปที่ 1.1..... 60
รูปที่ 3.7 หมายเลขดิวอีของอินเตอร์เฟซในรูปที่ 1.1..... 60
รูปที่ 3.8 หมายเลขดิวอีของชนิดข้อมูล ICompilationUnit 61
รูปที่ 3.9 หมายเลขดิวอีของชนิดข้อมูล IMember 61
รูปที่ 3.10 หมายเลขดิวอีของชนิดข้อมูล IType..... 62
รูปที่ 3.11 อัลกอริทึมของการสกัดรูปแบบการใช้งาน..... 69
รูปที่ 3.11 อัลกอริทึมของการสกัดรูปแบบการใช้งาน (ต่อ) 70
รูปที่ 3.12 โค้ดโมเดลแสดงตัวอย่างการสกัดรูปแบบการใช้งาน โดยมี tq = ASTRewrite..... 72
รูปที่ 3.13 ตัวอย่างขั้นตอนการแปลงแบบจำลองโค้ดให้กลายเป็นโค้ด 79
รูปที่ 3.13 ตัวอย่างขั้นตอนการแปลงแบบจำลองโค้ดให้กลายเป็นโค้ด (ต่อ)..... 80
รูปที่ 4.1 แผนภาพยูสเคส 82
รูปที่ 4.2 แพ็กเกจไดอะแกรมแสดงการเชื่อมต่อระหว่างแพ็กเกจ..... 89
รูปที่ 4.3 แผนภาพคลาสไดอะแกรม (ไม่ระบุแอททริบิวท์และเมท็อด)..... 90
รูปที่ 4.4 แผนภาพคลาสไดอะแกรมอย่างละเอียด..... 91
รูปที่ 4.6 Retrieve Code Model Sequence Diagram..... 97
รูปที่ 4.7 หน้าจอแสดงเครื่องมือเรียกดูรูปแบบการใช้งานอ็อบเจกต์ 98
รูปที่ 4.8 หน้าจอแสดงผลลัพธ์โดยละเอียดจากการทำงานของโปรแกรม 100

หน้า

หน้า

รูปที่ 5.1 สภาพแวดล้อมสำหรับโจทย์การใช้งานอ็อบเจกต์ ASTRewrite 103
รูปที่ 5.2 ผลลัพธ์ที่ต้องการสำหรับตัวอย่างโจทย์การใช้งานอ็อบเจกต์ ASTRewrite..... 104
รูปที่ 5.3 สภาพแวดล้อมสำหรับตัวอย่างโจทย์การใช้งานอ็อบเจกต์ IWorkingCopyManager. 112
รูปที่ 5.4 ผลลัพธ์ที่ต้องการของโจทย์การใช้งานอ็อบเจกต์ IWorkingCopyManager..... 113
รูปที่ 5.5 สภาพแวดล้อมสำหรับตัวอย่างโจทย์การใช้งานอ็อบเจกต์ IWorkingCopyManager. 119
รูปที่ 5.6 ผลลัพธ์ที่ต้องการสำหรับโจทย์การใช้งานอ็อบเจกต์ IWorkingCopyManager 120
รูปที่ 5.7 สภาพแวดล้อมสำหรับตัวอย่างโจทย์การใช้งานอ็อบเจกต์ ITextEditor 126
รูปที่ 5.8 ผลลัพธ์ที่ต้องการสำหรับตัวอย่างโจทย์การใช้งานอ็อบเจกต์ ITextEditor 126
รูปที่ 5.9 ร้อยละของจำนวนโจทย์ที่การสืบค้นแต่ละประเภทสามารถหาคำตอบได้ 134
รูปที่ 5.10 การแจกแจงแบบสะสมของอันดับผลลัพธ์ที่เหมาะสมที่สุด 134
รูปที่ 5.11 ผลลัพธ์จากการทดลองเครื่องมือ MAPO 135
รูปที่ ก.1 ลำดับเมท็อดอินโวเคชันของเมท็อด showPerspective() 143
รูปที่ ก.2 คลาส JavaMetricView แสดงการทำงานของเมท็อด createPartControl() 144
รูปที่ ก.3 ประเภทของแบบจำลองโหนด 145
รูปที่ ก.4 แสดงการแปลงจากโค้ดไปเป็นแบบโหนดชนิดข้อมูล..... 145
รูปที่ ก.5 ประเภทของแบบจำลองเอดจ์..... 146
รูปที่ ก.6 ประเภทของแบบจำลองเอดจ์..... 147
รูปที่ ก.7 การแปลงจาวาโค้ดให้กลายเป็นโหนดเมท็อด..... 148
รูปที่ ก.8 ประเภทของแบบจำลองเอดจ์..... 148
รูปที่ ก.9 ตัวอย่างการแปลงจาวาโค้ดไปเป็นโหนดที่เชื่อมต่อกันด้วยความสัมพันธ์แบบต่างๆ... 149
รูปที่ ก.10 ตัวอย่างการแปลงจากจาวาโค้ดไปเป็นเอดจ์เมท็อด..... 150
รูปที่ ก.11 การแปลงจากจาวาโค้ดเป็นเอดจ์การให้ค่า 152
รูปที่ ก.12 การแปลงจาวาโค้ดให้กลายเป็นเอดจ์พารามิเตอร์ 152
รูปที่ ข.1 ไฟล์ XSNIPPETUSAGE.jar ในไดเร็กทอรีที่ลงโปรแกรม Eclipse 154
รูปที่ ข.2 หน้าจอ Command Prompt 155
รูปที่ ข.3 หน้าจอ Command Prompt (2) 155
รูปที่ ข.4 แสดงหน้าจอการเลือก XSnippetUsage View..... 155
รูปที่ ข.5 หน้าจอแสดงการเรียกใช้งาน XSnippetUsage 155

บทที่ 1
บทนำ

1. ที่มาและความสำคัญของปัญหา

นักพัฒนาซอฟต์แวร์นิยมใช้กรอบงาน (Framework) และคลาสไลบรารี (Class Library) มาช่วยพัฒนาซอฟต์แวร์ให้เสร็จสิ้นตามระยะเวลาที่กำหนด การใช้งานกรอบงานและคลาสไลบรารีประกอบไปด้วย การสร้างซับคลาส (Subclass) ให้ถูกต้องตามที่กำหนดไว้ การสร้างอ็อบเจกต์ตามวิธีที่กำหนดเพื่อให้ใช้งานได้ตามต้องการ การเรียกใช้เมท็อดตามลำดับให้ถูกต้อง และการใช้งานอ็อบเจกต์ให้ตรงตามวัตถุประสงค์การใช้งาน

รูปที่ 1.1 แสดงรูปแบบการใช้งานต่างๆ ของอ็อบเจกต์ rewrite ที่มีชนิดเป็น ASTRewrite ซึ่งเป็นคลาสพื้นฐานที่อยู่ภายใต้อีคลิปไลบรารี (Eclipse Library) และมีรูปแบบการใช้งานที่หลากหลาย ยกตัวอย่างเช่น รูปแบบ A เป็นรูปแบบการใช้งานของอ็อบเจกต์ rewrite ในการเพิ่มตัวอักษรลงในอีคลิปเอดิเตอร์ (Eclipse Editor) แบบต่อท้ายข้อความที่ต้องการ รูปแบบ B เป็นรูปแบบการใช้งานสำหรับเพิ่มตัวอักษรแบบต่อด้านหน้าข้อความที่ต้องการ ส่วนรูปแบบ C และ D เป็นรูปแบบสำหรับการเขียนค่าทับโมดิฟายเออร์ (Modifier) ของเมท็อดลงในอีคลิปเอดิเตอร์ จากตัวอย่างเห็นได้ว่า การใช้งานอ็อบเจกต์หนึ่งๆ ในแต่ละวัตถุประสงค์ส่งผลให้รูปแบบการใช้งานแตกต่างกันไป

ปัญหาการใช้งานอ็อบเจกต์ที่หลากหลายเป็นปัญหาที่นักพัฒนาซอฟต์แวร์มักพบเป็นประจำเมื่อใช้งานไลบรารีที่ไม่ชำนาญ เนื่องจากกรอบงานและไลบรารีเหล่านี้เรียนรู้และใช้งานยาก เอกสารการใช้งานไม่ได้อธิบายวิธีการใช้อย่างละเอียด อีกทั้งรูปแบบการใช้งานของกรอบงานและไลบรารีมักจะซับซ้อนและประกอบไปด้วยคลาสจำนวนมาก นักพัฒนาซอฟต์แวร์ส่วนใหญ่จึงหาโค้ดตัวอย่างมาเป็นแนวทางในการใช้งานกรอบงานหรือคลาสไลบรารี

งานวิจัยหลายงาน [1-6] นำเสนอวิธีการค้นหาตัวอย่างโค้ดหรือการทำเหมืองโค้ดมาช่วยแก้ไขปัญหาดังกล่าว งานวิจัยของ Reid, H. [1] นำเสนอวิธีสกัดตัวอย่างโค้ดโดยนำข้อมูลบริบทเชิงโครงสร้าง (Structural Context) เช่น พาเรนท์ของคลาส หรือชนิดข้อมูลภายในคลาสที่กำลังพัฒนามาสร้างเป็นวิทยาการศึกษาสำนึก (Heuristic) แล้วนำไปช่วยค้นหาโค้ดจากคลังข้อมูล ผลลัพธ์ที่ได้คือโค้ดในคลังข้อมูลที่มีบริบทโค้ด (Code Context) ตรงกับบริบทที่พบในคลาสที่กำลังพัฒนา และมีการใช้งานบ่อยที่สุด แต่งานวิจัยนี้เอาบริบทที่ไม่จำเป็นเข้าไปคัดกรองผลลัพธ์มากเกินไป จนทำให้ผลลัพธ์ที่ได้มีจำนวนมาก หรือบางครั้งก็ไม่มีผลลัพธ์เลย

งานวิจัยของ Tao, X., and Jian, P [5] นำเสนอวิธีการทำเหมืองข้อมูลโค้ด โดยนำโค้ดเสิร์จเอนจินเข้ามาช่วยค้นหาคลาสที่เกี่ยวข้องกับคีย์เวิร์ดที่ต้องการ แล้วนำคลาสที่ได้มาหารูปแบบการใช้งานเอพีไอ (API Usage Pattern) ที่พบมากในฐานข้อมูล ถึงแม้ว่าวิธีการเหล่านี้จะสามารถช่วยแก้ไขปัญหาดังกล่าวได้ แต่ผลลัพธ์ที่ได้มักไม่มีคุณภาพ เนื่องจากผลลัพธ์ที่ได้มีจำนวนมากและไม่สอดคล้องกับความต้องการของนักพัฒนาซอฟต์แวร์

งานวิจัยของ Naiyana.S [4] นำเสนอวิธีการค้นหาโค้ดตัวอย่างในการสร้างอ็อบเจกต์ (Object Instantiation) จากชนิดข้อมูลของอ็อบเจกต์ (Object Type) โดยใช้บริบทของคลาสที่กำลังพัฒนาเข้าช่วยคัดเลือกและจัดอันดับผลลัพธ์ หากผลลัพธ์มีบริบทโค้ดตรงกับบริบทของคลาสที่กำลังพัฒนา ผลลัพธ์นั้นจะได้รับการคัดเลือกและจัดอันดับให้อยู่ในอันดับที่ดีขึ้น ซึ่งจะทำให้นักพัฒนาซอฟต์แวร์สามารถพบผลลัพธ์ได้รวดเร็วขึ้น อย่างไรก็ตามงานวิจัยนี้สามารถค้นหาได้เฉพาะการสร้างอ็อบเจกต์ แต่ไม่สามารถหารูปแบบการใช้งานของอ็อบเจกต์ได้ และการสร้างอ็อบเจกต์จะเป็นประโยชน์กับไลบรารีขนาดใหญ่ที่มีวิธีการสร้างอ็อบเจกต์ซับซ้อนเท่านั้น

วิทยานิพนธ์นี้พัฒนาต่อมาจากงานวิจัย [4] โดยเปลี่ยนจากการค้นหาวิธีการสร้างอ็อบเจกต์มาเป็นการสกัดรูปแบบการใช้งานของอ็อบเจกต์ ซึ่งมีรายละเอียดดังนี้

1.1. การออกแบบประเภทของการร้องขอข้อมูล โดยแบ่งออกเป็น 2 ประเภท ได้แก่ การสอบถามแบบทั่วไป (Generalized Query) ซึ่งเป็นการสอบถามที่ไม่ใช้บริบทโค้ด (Code Context) เหมาะสมสำหรับกรณีที่ไม่พบผลลัพธ์ที่มีบริบทตรงกับบริบทโค้ดที่กำลังพัฒนา และการร้องขอแบบเฉพาะเจาะจง (Specialized Query) ซึ่งใช้ข้อมูลบริบทเข้าไปช่วยคัดกรอง เพื่อให้ได้ผลลัพธ์ที่ตรงตามความต้องการมากที่สุด

1.2. การออกแบบวิธีการสกัดรูปแบบการใช้งานของอ็อบเจกต์ โดยใช้อัลกอริทึมแบบค้นหาแบบลึกก่อน (Depth First Search) ซึ่งวิธีการที่นำเสนอในวิทยานิพนธ์นี้สามารถช่วยให้สกัดรูปแบบการใช้งานอ็อบเจกต์ได้ถูกต้องและครบถ้วน

การทดลองจะทดสอบโดยสร้างโจทย์โปรแกรม (Programming Task) ขึ้นมา แต่ละโจทย์จะมีทรัพยากรต่างๆ เช่น ไฟล์ที่จำเป็นต่อการทำงานหรือคลาสไลบรารีที่ต้องใช้ในการทำงานไว้ครบถ้วน โดยเว้นที่ว่างในโค้ดไว้เพื่อตรวจสอบว่าวิธีการสามารถให้คำตอบที่เติมเต็มส่วนที่หายไปได้ครบถ้วนและถูกต้องหรือไม่ รวมทั้งตรวจสอบอันดับของผลลัพธ์ที่ได้ด้วย

A.
```
FieldDeclaration fd; AST node; ICompilationUnit unit, Map option;
Initializer initializer = ast.newInitializer();
RefactoringASTParser rap = new RefactoringASTParser(AST.JLS3);
CompilationUnit root = rap.parse(unit, true, new ProgressMonitor(monitor, 1));
ASTRewrite rewrite = ASTRewrite.create(root.getAST());
ListRewrite listRewrite = rewrite.getListRewrite(node, null);
listRewrite.insertAfter(initializer, fd, option);
```

B.
```
Block block; AST ast; ICompilationUnit cu;
ASTRewrite rewrite = ASTRewrite.create(ast);
ListRewrite listRewrite = rewrite.getListRewrite
(block, Block.STATEMENTS PROPERTY);
listRewrite.insertFirst(st, null);
TextEdit res = rewrite.rewriteAST();
IDocument document = new Document(cu.getSource());
res.apply(document);
IBuffer buffer = cu.getBuffer();
buffer.setContents(document.get());
```

C.
```
ICompilationUnit cu; int indent; IType ct; String fcmt;
ISourceRange r = ct.getSourceRange();
IBuffer buf = cu.getBuffer();
String oritxt = buf.getText(r.getOffset(), r.getLength());
IDocument document = buf.getDocument();
ASTRewrite rewrite = new ASTRewrite.create(null);
TextEdit textEdit = rewrite.rewriteAST(document, null);
textEdit.apply(document);
buf.replace(range.getOffset(), range.getLength(),fmct);
```

รูปที่ 1.1 รูปแบบการใช้งานแบบต่างๆ ของอ็อบเจกต์ ASTRewrite

```
D.  IMember im;
    ICompilationUnit icu = im.getCompilationUnit();
    IBuffer bf = icu.getBuffer();
    IDocument id = new Document(content);
    ASTRewrite rewrite = ASTRewrite.create(null);
    TextEdit textEdit = rewrite.rewriteAST(id, null);
    textEdit.apply(id);
    bf.replace(0,bf.getLength(),bf.getContent());
```

รูปที่ 1.1 รูปแบบการใช้งานแบบต่างๆ ของอ็อบเจกต์ ASTRewrite (ต่อ)

2. วัตถุประสงค์ของวิทยานิพนธ์

2.1 เพื่อออกแบบประเภทของคำร้องขอเพื่อใช้ในการค้นหารูปแบบการใช้งานอ็อบเจกต์

2.2 เพื่อออกแบบวิธีการสกัดรูปแบบการใช้งานอ็อบเจกต์

2.3 สร้างเครื่องมือสนับสนุนวิธีการในข้อ 2.1 และ 2.2

3. ขอบเขตของวิทยานิพนธ์

3.1 การวิเคราะห์โค้ดจะเป็นแบบสถิต (Static Analysis) ไม่คำนึงถึงโครงสร้างแบบกระแสควบคุม (Control Flow Structure e.g., if-else) และแบบวนซ้ำ (Iteration e.g.; while, for)

3.2 ออกแบบวิธีการสกัดรูปแบบการใช้งานอ็อบเจกต์บนพื้นฐานของภาษาจาวา (Java Programming Language) ซึ่งเป็นภาษาแบบอ็อบเจกต์ (Object-Oriented Language) แต่สามารถนำแนวคิดไปประยุกต์ใช้กับภาษาอ็อบเจกต์อื่นๆ ได้ เช่น C++ C# หรือ Perl เป็นต้น

3.3 พัฒนาวิธีการสกัดรูปแบบการใช้งานอ็อบเจกต์เป็น plug-in ของ Eclipse Integrated Development Environment (IDE) ซึ่งจะสามารถดึงข้อมูลบริบทของโค้ดในขณะที่กำลังเขียนโปรแกรมบน Eclipse ได้

3.4 ข้อมูลโค้ดในฐานข้อมูลได้มาจากแหล่งข้อมูล 2 แหล่ง ได้แก่

- จากโค้ดเสิร์ชเอนจิน โดยใช้ประเภทของอ็อบเจกต์ที่ต้องการร้องขอ t_q ของแต่ละโจทย์โปรแกรมเพื่อดาวน์โหลดซอร์สโค้ดจากโค้ดเสิร์จเอนจินที่เกี่ยวข้องกับอ็อบเจกต์ t_q และจะเลือกผลลัพธ์ 30 อันดับแรกมาจัดเก็บในฐานข้อมูล

- จากโอเพนซอร์สที่มีการใช้งานไลบรารีของแต่ละโจทย์โปรแกรม เช่น ไลบรารีพื้นฐานของอีคลิปปลั๊กอิน (Standard Eclipse Plug-in Library)

3.5 การทดสอบวิธีการจะกระทำโดยสร้างโจทย์โปรแกรม (Programming Task) ขึ้นมา 19 โจทย์ โจทย์โปรแกรมแต่ละข้อจะมีทรัพยากรต่างๆ ที่จำเป็น เช่น ไลบรารีที่ติดตั้งคลาสพาทเรียบร้อยแล้ว หรือไฟล์ที่เกี่ยวข้องในการทำงานของโปรแกรม ในการทำงานของโจทย์นั้นๆ ยกเว้นรูปแบบการใช้งานของอ็อบเจกต์ที่ต้องการซึ่งจะเว้นว่างไว้เพื่อทดสอบว่าวิธีการสามารถตอบโจทย์นั้นได้ครบถ้วนและถูกต้องหรือไม่ การวัดประสิทธิภาพของวิธีการจะวัดจากจำนวนโจทย์ที่เครื่องมือสนับสนุนสามารถตอบโจทย์ได้ และอันดับของผลลัพธ์ที่เครื่องมือสนับสนุนแนะนำ

นอกจากนี้ ได้ออกแบบโจทย์โปรแกรมเพื่อตรวจสอบว่าวิธีการสามารถนำไปประยุกต์ใช้ได้กับไลบรารีหลายประเภท โดยโจทย์โปรแกรมที่นำมาทดสอบสามารถแบ่งออกเป็น 3 ประเภทตามประเภทของไลบรารีได้ดังนี้

- สำหรับทดสอบไลบรารีพื้นฐานของจาวา (J2SE Standard Library)
- สำหรับทดสอบไลบรารีของอีคลิป (Eclipse Library)
- สำหรับทดสอบไลบรารีอื่นๆ (Other Library)

4. วิธีการดำเนินงานวิจัย

4.1 ศึกษาวิธีการสกัดข้อมูลประเภทต่างๆ โดยเฉพาะที่นำมาประยุกต์ใช้กับคลังข้อมูลโค้ด

4.2 ศึกษาโครงสร้างของโค้ด ในที่นี้จะศึกษาในภาษาจาวา

4.3 ศึกษางานวิจัยที่เกี่ยวข้อง

4.4 ศึกษาขั้นตอนวิธีการต่างๆ ที่จำเป็นต้องใช้ในการพัฒนางานวิจัย

4.5 กำหนดขอบเขตของงาน

4.6 ออกแบบโครงสร้างของวิทยานิพนธ์ รวมทั้งกำหนดวิธีการในการพัฒนา

4.7 สร้างเครื่องมือสนับสนุนโดยใช้วิธีการที่ได้ออกแบบไว้

4.8 นำระบบไปทดสอบว่าสามารถตอบโจทย์ได้ครบถ้วนหรือไม่และสามารถแนะนำรูปแบบการใช้งานให้กับผู้ใช้งานหรือไม่

4.9 สรุปผลการวิจัย และข้อเสนอแนะ

4.10 จัดทำรูปเล่มวิทยานิพนธ์

5. ประโยชน์ที่คาดว่าจะได้รับ

5.1 ได้วิธีการในการสกัดรูปแบบการใช้งานอ็อบเจกต์ เพื่อให้ผู้ใช้ลดภาระในการค้นหาตัวอย่างการใช้งานของอ็อบเจกต์ด้วยตนเอง ซึ่งจะใช้เวลามากและอาจจะไม่ได้ผลลัพธ์ตามต้องการ และช่วยสนับสนุนการนำโค้ดกลับมาใช้ใหม่ เนื่องจากผู้ใช้สามารถใช้งานไลบรารีหรือกรอบงานจากโอเพนซอร์สภายนอกได้สะดวกมากขึ้น

5.2 ได้เครื่องมือสนับสนุนการแนะนำรูปแบบการใช้งานของอ็อบเจกต์ และสามารถจัดลำดับความสำคัญของผลลัพธ์ที่ต้องการได้

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

1. ทฤษฎีที่เกี่ยวข้อง

ในหัวข้อนี้จะกล่าวถึงทฤษฎีที่เกี่ยวข้องกับวิทยานิพนธ์ ซึ่งได้แก่ภาษาจาวา และมาตรวัดความเหมือนกันของบริบท

1.1. ภาษาจาวา (Java Programming Language)

โครงสร้างพื้นฐานของภาษาจาวาจะใช้เป็นความรู้ในการทำความเข้าใจบริบทโค้ดที่ใช้ภายในวิทยานิพนธ์ เนื่องจากการออกแบบประเภทของบริบทโค้ดเลียนแบบมาจากโครงสร้างของภาษาจาวา รายละเอียดของโครงสร้างภาษาจาวามีดังนี้

ภาษาจาวาเป็นภาษาระดับสูงที่รองรับการพัฒนาซอฟต์แวร์ในรูปแบบภาษาโปรแกรมเชิงวัตถุ (Object-Oriented Language) ซึ่งไวยากรณ์ของภาษาจาวามีลักษณะคล้ายคลึงกับภาษา C และภาษา C++ เป็นส่วนใหญ่ อย่างไรก็ตาม ภาษาจาวานำเสนอรูปแบบการทำงานของซอฟต์แวร์ที่แตกต่างกับทั้งสองภาษาดังกล่าว คือ ภาษาจาวาจะคอมไพล์ (Compile) ซอร์สโค้ดเป็นรหัสไบต์ (Byte Code) และผู้ใช้งานสามารถนำรหัสไบต์ที่ได้ไปใช้งานร่วมกับเจวีเอ็ม (JVM: Java Virtual Machine) โดยไม่ขึ้นอยู่กับระบบปฏิบัติการของเครื่องคอมพิวเตอร์ที่ซอฟต์แวร์ทำงาน ซึ่งเป็นข้อแตกต่างกับภาษาอื่นๆ ทำให้ในปัจจุบันภาษาจาวาได้รับความนิยมแพร่หลายในการนำมาพัฒนาซอฟต์แวร์

โครงสร้างและรายละเอียดของภาษาจาวา มีรายละเอียดพอสังเขปดังนี้

1.1.1 ชนิดข้อมูล (Types) รูปที่ 2.1 แสดงไวยากรณ์ชนิดข้อมูล ประกอบด้วย ชนิดข้อมูลแบบพื้นฐานและชนิดข้อมูลแบบอ้างอิง โดยมีรายละเอียด ดังนี้

Type:

PrimitiveType

ReferenceType

รูปที่ 2.1 ไวยากรณ์ชนิดข้อมูล

1.1.1.1 ชนิดข้อมูลแบบพื้นฐาน (Primitive type) คือ ชนิดข้อมูลแบบพื้นฐานที่กำหนดโดยภาษาจาวา ซึ่งชนิดข้อมูลแบบนี้จะสามารถส่งผ่านโดยค่า (Pass by value) เท่านั้น รูปที่ 2.2 แสดงไวยากรณ์สำหรับการประกาศตัวแปรโดยใช้ชนิดข้อมูลแบบพื้นฐาน ซึ่งชนิดข้อมูลแบบพื้นฐาน ประกอบด้วย

- ชนิดข้อมูลแบบตรรกะ ได้แก่ boolean
- ชนิดข้อมูลจำนวนเต็ม ได้แก่ byte short int และ long
- ชนิดข้อมูลอักขระ ได้แก่ char
- ชนิดข้อมูลทศนิยม ได้แก่ float และ double

PrimitiveType:
  NumericType
  boolean

NumericType:
  IntegralType
  FloatingPointType

IntegralType: one of
  byte short int long char

FloatingPointType: one of
  float double

รูปที่ 2.2 ไวยากรณ์สำหรับการประกาศตัวแปรโดยใช้ชนิดข้อมูลแบบพื้นฐาน

โดยช่วงค่าของข้อมูลและขนาดของข้อมูลแต่ละชนิดข้อมูล จะแสดงดังตารางที่ 2.1 ตัวอย่างการประกาศตัวแปร เช่น int currPos;

ตัวแปรชื่อ currPos จะมีการเก็บค่าข้อมูลเป็นจำนวนเต็ม โดยค่าข้อมูลของตัวแปร currPos จะอยู่ในช่วง –2147483648 ถึง 2147483647 หากนักพัฒนา

ซอฟต์แวร์มีการกำหนดค่าให้กับตัวแปร currPos มากกว่า 2147483647 หรือน้อยกว่า -2147483648 จะทำให้ค่าที่เก็บในตัวแปรนี้มีความผิดพลาด ตัวอย่างเช่น

currPos = Integer.MAX_VALUE + 1

จากนิพจน์ข้างต้น ค่าของ currPos ควรจะมีค่าเท่ากับ 2147483648 แต่ค่าที่ได้จากการทำงานของโปรแกรมจะมีค่าเท่ากับ – 2147483648 ซึ่งเป็นค่าที่ไม่ถูกต้อง

นอกจากนั้นภาษาจาวาได้กำหนดค่าเริ่มต้นของชนิดข้อมูลแต่ละชนิด ดังตารางที่ 2.2 ฉะนั้น หากนักพัฒนาซอฟต์แวร์ไม่กำหนดค่าเริ่มต้นของตัวแปร ภาษาจาวาจะกำหนดค่าเริ่มต้นให้กับตัวแปรอัตโนมัติตามตารางที่ 2.2

ตารางที่ 2.1 ค่าของข้อมูลที่เป็นไปได้แต่ละชนิดข้อมูล

ชนิดข้อมูล	ค่าของข้อมูลที่เป็นไปได้		ขนาด (บิต)
Byte	- 128	- 127	8
Short	- 32768	- 32767	16
Int	- 2147483648	- 2147483647	32
Long	- 9223372036854775808	- 9223372036854775807	64
Char	0	- 65535	16
Float	2^{-149}	- (2-2-23) x 2127	32
Double	2^{-1074}	- (2-2-52) x 21023	64
Boolean	True หรือ False		1

ตารางที่ 2.2 ค่าเริ่มต้นสำหรับแต่ละชนิดข้อมูล

ชนิดข้อมูล	ค่าเริ่มต้น
byte	0
short	0
int	0
long	0L
char	\u0000
float	0.0f
double	0.0d
boolean	False

1.1.1.2 ชนิดข้อมูลแบบอ้างอิง (Reference type) คือ ชนิดข้อมูลที่ประกอบด้วย ประเภทคลาส (class types) ประเภทอินเตอร์เฟส (interface types) และประเภทอาเรย์ (array types) โดยชนิดข้อมูลแบบอ้างอิงจะสามารถส่งผ่านโดยการอ้างอิงเท่านั้น (Pass by reference) เท่านั้น รูปที่ 2.3 แสดงไวยากรณ์สำหรับการประกาศตัวแปรโดยใช้ชนิดข้อมูลแบบอ้างอิง จะพบว่า ประเภทคลาสและประเภทอินเตอร์เฟสประกอบด้วยสองส่วน คือ ตัวระบุการประกาศชนิดข้อมูล (type declaration specifier, TypeDeclSpecifier) และไทป์อากิวเมนต์ (type arguments,$TypeArguments_{opt}$) โดยนักพัฒนาซอฟต์แวร์สามารถเลือกกำหนดค่าไทป์อากิวเมนต์หรือไม่ก็ได้ ซึ่งตัวระบุการประกาศชนิดข้อมูลอาจจะเป็นชื่อประเภท (type name, TypeName) หรือชื่อคลาส หรือชื่ออินเตอร์เฟส ในที่นี้ชื่อประเภทแบ่งออกได้เป็น 2 ประเภท ได้แก่ ชื่อประเภทแบบง่าย (Simple Type Names) และชื่อประเภทแบบมีเงื่อนไข (Qualified Type Names) โดยชื่อประเภทแบบง่าย คือ ชื่อของชนิดข้อมูลที่ปรากฏภายในขอบเขตของโปรแกรมที่นักพัฒนา ซอฟต์แวร์สามารถเรียกใช้งานได้ทันที ส่วนชื่อประเภทแบบมีเงื่อนไข คือ ชื่อของชนิดข้อมูลที่จำเป็นต้องระบุแพ็คเกจ (Package) โดยมีชื่อของชนิดข้อมูลที่นิยามไว้ภายในไอเด็นติฟายเออร์ (Identifier) เป็นสายอักขระที่สามารถใช้สำหรับตั้งชื่อคลาส ชื่อตัวแปร ชื่อเมท็อด และชื่อของค่าคงที่ ซึ่งไวยากรณ์สำหรับการนิยามไอเด็นติฟายเออร์แสดงดังรูปที่ 2.4 โดยไอเด็นติฟายเออร์ต้องเริ่มต้นด้วยอักขระ A-Z, a-z, _ หรือ $ เท่านั้น และต้องไม่เป็นคำสงวนของภาษาจาวา

รูปที่ 2.5 แสดงตัวอย่างการประกาศตัวแปรโดยใช้ชนิดข้อมูลแบบอ้างอิง ซึ่ง java.util.Date แสดงตัวอย่างการประกาศตัวแปรโดยใช้ชนิดข้อมูลแบบอ้างอิงด้วย qualified type name ที่อธิบายก่อนหน้า จะเห็นว่า Date คือไอเด็นติฟายเออร์ และ java.util คือแพ็คเกจที่นิยามคลาส Date และรูปที่ 2.6 แสดงตัวอย่างการประกาศตัวแปรโดยใช้ชนิดข้อมูลแบบอ้างอิงด้วยอาเรย์ และการประกาศคลาสและอินเตอร์เฟส เพื่อใช้กำหนดชนิดของข้อมูล

```
Point thePoint    = new Point()
IMove theIMove = new MovingBox()
```

จากนิพจน์ข้างต้น ตัวแปร thePoint มีชนิดข้อมูลคือ คลาส Point โดยมีสมาชิกภายในคลาส คือ ตัวแปร metrics ซึ่งมีค่าของข้อมูลเป็นอาเรย์ของตัวเลขจำนวนเต็ม (Integer) และตัวแปร theIMove จะมีชนิดข้อมูล คือ IMove แต่ข้อมูลภายในตัวแปรจะเป็นข้อมูล MovingBox ซึ่งเป็นคุณสมบัติพอลลีมอร์ฟิสซึม (Polymorphism)

ReferenceType:

ClassOrInterfaceType

TypeVariable

ArrayType

ClassOrInterfaceType:

ClassType

InterfaceType

ClassType:

TypeDeclSpecifier TypeArguments$_{opt}$

InterfaceType:

TypeDeclSpecifier TypeArguments$_{opt}$

TypeDeclSpecifier:

TypeName

ClassOrInterfaceType . Identifier

TypeName:

Identifier

TypeName . Identifier

TypeVariable:

Identifier

ArrayType:

Type [ ]

รูปที่ 2.3 ไวยากรณ์สำหรับการประกาศตัวแปรโดยใช้ชนิดข้อมูลแบบอ้างอิง

Identifier:

IdentifierChars but not a Keyword or *BooleanLiteral* or *NullLiteral*

IdentifierChars:

JavaLetter

IdentifierChars JavaLetterOrDigit

JavaLetter:

Any Unicode character that is a Java letter

JavaLetterOrDigit:

Any Unicode character that is a Java letter-or-digit

BooleanLiteral: one of

true false

NullLiteral:

null

รูปที่ 2.4 ไวยากรณ์สำหรับการนิยามไอเด็นติฟายเออร์

```
package wnj.test;
class Test
{
   public static void main(String[] args)
   {
      java.util.Date date = new java.util.Date(System.currentTimeMillis());
      System.out.println(date.toLocaleString());
   }
}
```

รูปที่ 2.5 ตัวอย่างการประกาศตัวแปรโดยใช้ชนิดข้อมูลแบบอ้างอิง

```
class Point
{
   int[] metrics;
}

interface IMove
{
    void move(int deltax, int deltay);
}
```

รูปที่ 2.6 ตัวอย่างการประกาศตัวแปรโดยใช้ชนิดข้อมูลแบบอ้างอิงด้วยอาเรย์

1.1.2. อ็อบเจกต์ (Object)

อ็อบเจกต์ คือ อินแสตนซ์ของคลาสหรืออาเรย์ ซึ่งค่าของข้อมูลจะเป็นค่าแบบอ้างอิง (Reference values) หรือพอยเตอร์ (Pointer) ไปหาอ็อบเจกต์

รูปที่ 2.7 - 2.9 แสดงตัวอย่างของการใช้งานอ็อบเจกต์ในภาษาจาวา รูปที่ 2.7 ในบรรทัดที่ 1 – 24 เป็นการนิยามคลาส Point ที่จะใช้เป็นชนิดของค่าของข้อมูลในคลาส Test ในตัวอย่างของรูปที่ 2.8 และรูปที่ 2.9 ซึ่งคลาส Test นั้นจะเป็นคลาสที่โปรแกรมเรียกทำงาน เนื่องจากมีฟังก์ชัน main นิยามไว้ภายในคลาส รูปที่ 2.8 ในบรรทัดที่ 31 จะพบว่าตัวแปร thePoint เป็นอ็อบเจกต์ที่มีชนิดข้อมูลเป็นคลาส Point โดยมีค่าเท่ากับนัล (null) และในบรรทัดที่ 35 จะกำหนดค่าให้กับตัวแปร thePoint โดยใช้อินแสตนซ์ของคลาส Point ที่นิยามขึ้นมา ดังนั้นตัวแปร thePoint จะประกอบด้วยสมาชิกหรือแอททริบิวต์ของคลาส Point ได้แก่ ตัวแปร x และตัวแปร y ซึ่งมีชนิดข้อมูลเป็นจำนวนเต็ม และรูปที่ 2.9 เป็นอีกหนึ่งตัวอย่างของการใช้งาน อ็อบเจกต์ในภาษาจาวา ซึ่งในบรรทัดที่ 1 – 24 เป็นการนิยามคลาส Point เช่นเดียวกับรูปที่ 2.7 ซึ่งในตัวอย่างนี้จะพบอ็อบเจกต์สองอ็อบเจกต์ ได้แก่ ตัวแปร thePoints และตัวแปร theString โดยตัวแปร thePoints จะเป็นอาเรย์ที่มีชนิดข้อมูลเป็นคลาส Point และตัวแปร theString จะเป็นอาเรย์ที่มีชนิดข้อมูลเป็นคลาส String ซึ่งทั้งสองตัวแปรจะเป็นอาเรย์หนึ่งมิติที่ขนาดเท่ากับสอง

1.1.3. ตัวแปร (Variable)

ตัวแปรเป็นหน่วยพื้นฐานที่ใช้เก็บค่าข้อมูลในภาษาจาวา ซึ่งภาษาจาวานั้นเป็นภาษาที่เป็น Strongly typed language ซึ่งทำให้ทุกตัวแปรหรือทุกนิพจน์ (Expression) ที่นิยาม

ในโปรแกรมที่พัฒนาด้วยภาษาจาวานั้นจำเป็นต้องระบุชนิดข้อมูลอย่างชัดเจน โดยรูปแบบการนิยามตัวแปรเป็นดังนี้

type identifier [= value][, identifier [= value] ...] ;

```
1   class Point
2   {
3     int x, y;
4
5     Point()
6     {
7       System.out.println("default");
8     }
9
10    Point(int x, int y)
11    {
12      this.x = x;
13      this.y = y;
14    }
15
16    // A Point instance is explicitly created at class initialization time:
17    static Point origin = new Point(0,0);
18
19    // A String can be implicitly created by a + operator:
20    public String toString()
21    {
22      return "(" + x + "," + y + ")";
23    }
24  }
```

รูปที่ 2.7 ตัวอย่างของการใช้งานอ็อบเจกต์ในภาษาจาวา

ซึ่งภาษาจาวาได้แบ่งประเภทตัวแปรออกเป็น 7 ประเภท มีรายละเอียดพอสังเขปดังนี้

- ตัวแปรคลาส (Class variable) คือ ตัวแปรที่นิยามภายในคลาสด้วยคีย์เวิร์ด static หรือตัวแปรที่นิยามภายในอินเตอร์เฟสด้วยคีย์เวิร์ด static หรือไม่ใช้คีย์เวิร์ด static ซึ่งตัวแปรชนิดนี้จะสร้างขึ้นและกำหนดค่าเริ่มต้น เมื่อจัดเตรียมคลาสหรืออินเตอร์เฟสสำหรับเริ่มใช้งาน รูปที่ 2.10 ในบรรทัดที่ 3 แสดงตัวอย่างการนิยามตัวแปรคลาส ได้แก่ ตัวแปร numPoint ซึ่งชนิดข้อมูลเป็นจำนวนเต็ม

```
1   class Point
2   {
        …
24  }
25
26  class Test
27  {
28    public static void main(String[] args)
29    {
30      // A Point is explicitly created using newInstance:
31      Point thePoint = null;
32
33      try
34      {
35        thePoint = (Point)Class.forName("Point").newInstance();
36      } catch (Exception e)
37      {
38        System.out.println(e);
39      }
40  }
```

รูปที่ 2.8 ตัวอย่างของการใช้งานอ็อบเจกต์ในภาษาจาวา

```
1   class Point
2   {
        …
24  }
25
26  class Test
27  {
28    public static void main(String[] args)
29    {
30      // An array is implicitly created by an array constructor:
31      Point thePoints[] = { new Point(0,0), new Point(1,1) };
32
33      // Strings are implicitly created by + operators:
34      System.out.println("thePoints: { " + thePoints [0] + ", " + thePoints [1] + " }");
35      // An array is explicitly created by an array creation expression:
36      String theString[] = new String[2];
37      theString [0] = "he";
38      theString [1] = "llo";
39      System.out.println(theString [0] + theString [1]); // ”Hello”
40    }
```

รูปที่ 2.9 ตัวอย่างของการใช้งานอ็อบเจกต์ในภาษาจาวา

- ตัวแปรอินแสตนซ์ (Instance variable) คือ ตัวแปรที่นิยามภายในคลาสโดยไม่ใช้คีย์เวิร์ด static หรือตัวแปรที่นิยามภายในอินเตอร์เฟสด้วยคำว่า static หรือไม่ใช้คำว่า static ซึ่งค่าของตัวแปรคลาสจะกำหนดค่าเริ่มต้น เมื่อสร้างคลาสหรืออินเตอร์เฟสขึ้น รูปที่ 2.10 ในบรรทัดที่ 4 แสดงตัวอย่างการนิยามตัวแปรอินแสตนซ์ ได้แก่ ตัวแปร x และตัวแปร y ซึ่งชนิดข้อมูลเป็นจำนวนเต็ม
- อาเรย์คอมโพเนนท์ คือ ตัวแปรที่เป็นส่วนประกอบภายในอาเรย์ รูปที่ 2.10 บรรทัดที่ 5 จะพบว่า ตัวแปร w[0] จะเป็นอาเรย์คอมโพเนนท์ โดยชนิดข้อมูล

เป็นจำนวนเต็ม ซึ่งอาเรย์คอมโพเนนท์จะสร้างและกำหนดค่าเริ่มต้นเมื่อสร้างอ็อบเจกต์ที่เป็นอาเรย์ขึ้น

- เมท็อดพารามิเตอร์ (Method parameters) คือ ตัวแปรที่นิยามขึ้นสำหรับใช้เป็นอากิวเมนต์ของเมท็อด รูปที่ 2.10 บรรทัดที่ 5 จะพบว่า ตัวแปร x เป็นเมท็อดพารามิเตอร์ ซึ่งชนิดข้อมูลเป็นจำนวนเต็ม
- ตัวแปรโลคอล (Local variables) คือ ตัวแปรที่นิยามขึ้นภายในเมท็อดหรือ control statement รูปที่ 2.10 บรรทัดที่ 9 จะพบว่าตัวแปร oldx เป็น เมท็อดพารามิเตอร์ ซึ่งชนิดข้อมูลเป็นจำนวนเต็ม
- คอนสตัคเตอร์พารามิเตอร์ (Constructor parameters) คือ ตัวแปรที่นิยามขึ้นสำหรับใช้เป็นอากิวเมนต์ของคอนสตัคเตอร์
- พารามิเตอร์ที่ใช้จัดการสิ่งผิดปรกติ (Exception parameters) จะนิยามขึ้นเพื่อใช้จัดการสิ่งผิดปรกติต่างๆ ที่เกิดขึ้นระหว่างโปรแกรมทำงาน

```
1   class Point
2   {
3     static int numPoints;   // numPoints is a class variable
4     int x, y;               // x and y are instance variables
5     int[] w = new int[10];  // w[0] is an array component
6
7     int setX(int x)         // x is a method parameter
8     {
9       int oldx = this.x;    // oldx is a local variable
10      this.x = x;
11      return oldx;
12    }
13  }
```

รูปที่ 2.10 ตัวอย่างของประเภทของตัวแปรประเภทต่างๆ

1.1.4. คลาส (Class)

คลาสเป็นชนิดข้อมูลแบบอ้างอิงชนิดหนึ่งที่นักพัฒนาซอฟต์แวร์สามารถนิยามข้อมูลต่างๆ ภายในคลาสได้ด้วยตนเอง ซึ่งทำให้เกิดความยืดหยุ่นในการพัฒนาซอฟต์แวร์มากยิ่งขึ้น นอกจากนั้นคลาสยังมีประโยชน์ต่างๆ สำหรับการพัฒนาซอฟต์แวร์ เช่น การห่อหุ้ม (Encapsulation) ซึ่งทำให้นักพัฒนาซอฟต์แวร์สามารถปิดบังข้อมูลภายในคลาสที่ไม่ต้องการให้คลาสอื่นมองเห็นหรือเรียกใช้งานได้โดยตรง จึงทำให้สามารถจัดการข้อมูลภายในคลาสได้อย่างมีประสิทธิภาพ

รูปที่ 2.11 แสดงไวยากรณ์ภาษาจาวาสำหรับการนิยามคลาส ซึ่งในภาษาจาวาแบ่งการนิยามคลาสออกเป็น 2 แบบ คือ การนิยามคลาสโดยทั่วไป (Normal class declaration) และการนิยามอีนัม (Enum declaration)

1.1.4.1. การนิยามคลาสโดยทั่วไป ประกอบด้วย 3 ส่วนที่สำคัญ ได้แก่ คีย์เวิร์ด class, ไอเด็นติฟายเออร์และ คลาสบอดี้ (Class body) รวมทั้งการกำหนดค่าเริ่มต้นสำหรับตัวแปรที่เป็นสมาชิกของคลาส นอกจากนั้นนักพัฒนาซอฟต์แวร์สามารถกำหนดค่าคุณสมบัติอื่น ๆ ที่ใช้นิยามคลาส ได้แก่ คลาสโมดิฟายเออร์ (Class modifiers) ไทป์พารามิเตอร์ (Type parameters) ซุปเปอร์คลาส (Super classes) และอินเตอร์เฟสคลาส (Interface classes) ซึ่งแต่ละส่วนประกอบในการนิยามคลาส โดยมีรายละเอียดพอสังเขป ดังนี้

- คีย์เวิร์ด class ใช้สำหรับระบุว่าเป็นการนิยามคลาส
- Identifier ใช้สำหรับกำหนดชื่อของคลาสที่ต้องการนิยาม
- Class body ใช้สำหรับนิยามคอนสตัคเตอร์ของสมาชิกคลาสหรือแอททริบิวต์และเมท็อด นอกจากนั้นนักพัฒนาซอฟต์แวร์สามารถนิยามคลาสและอินเตอร์เฟสเพิ่มเติมภายในคลาสได้อีกด้วย
- Class modifiers ใช้สำหรับกำหนดระดับการเข้าใช้งานของคลาส (public, protected, private) การอนุญาตสืบทอดคลาส (final) และการนิยามคลาสนั้นไม่มีความสมบูรณ์ (abstract)
- Super classes ใช้สำหรับระบุคลาสที่ต้องการสืบทอดคุณสมบัติประกอบด้วยสมาชิกของคลาสและเมท็อด เพื่อนำมาใช้นิยามคลาสใหม่ตามที่ต้องการ
- Interface classes ใช้สำหรับระบุคลาสที่ต้องการสืบทอดคุณสมบัติ ประกอบด้วยสมาชิกของคลาสและเมท็อด เพื่อนำมาใช้นิยามคลาสใหม่ตามที่ต้องการเช่นเดียวกับซุปเปอร์คลาส แต่การ

สืบทอดคุณสมบัติของเมท็อดนั้น จำเป็นต้องเขียนโปรแกรมในส่วนการทำงานของเมท็อดเอง ซึ่งจะแตกต่างจากซุปเปอร์คลาส

ClassDeclaration:

NormalClassDeclaration

EnumDeclaration

InterfaceDeclaration:

NormalInterfaceDeclaration

AnnotationTypeDeclaration

NormalClassDeclaration:

$ClassModifiers_{opt}$ class *Identifier* $TypeParameters_{opt}$ $Super_{opt}$ $Interfaces_{opt}$ *ClassBody*

ClassModifiers:

ClassModifier

ClassModifiers ClassModifier

ClassModifier: one of

Annotation public protected private abstract static final strictfp

TypeParameters:

< *TypeParameter* {, *TypeParameter*} >

TypeParameter:

Identifier [extends *Bound*]

Bound:

Type {& *Type*}

ClassBodyDeclarations:

ClassBodyDeclaration

ClassBodyDeclarations ClassBodyDeclaration

รูปที่ 2.11 ไวยากรณ์ภาษาจาวาสำหรับการนิยามคลาส

ClassBodyDeclaration:

ClassMemberDeclaration

InstanceInitializer

StaticInitializer

ConstructorDeclaration

ClassMemberDeclaration:

FieldDeclaration

MethodDeclaration

ClassDeclaration

InterfaceDeclaration

;

Block:

{ $BlockStatements_{opt}$ }

BlockStatements:

BlockStatement

BlockStatements BlockStatement

MethodDeclaration:

MethodHeader MethodBody

BlockStatement:

LocalVariableDeclarationStatement

ClassDeclaration

Statement

MethodHeader:

$MethodModifiers_{opt}$ *TypeParameters*$_{opt}$ *ResultType MethodDeclarator* $\underline{Throws}_{opt}$

รูปที่ 2.11 ไวยากรณ์ภาษาจาวาสำหรับการนิยามคลาส (ต่อ)

MethodDeclarator:

Identifier (FormalParameterList$_{opt}$)

MethodModifiers:

MethodModifier

MethodModifiers MethodModifier

MethodModifier: one of

Annotation public protected private abstract static final synchronized native strictfp

LocalVariableDeclarationStatement:

LocalVariableDeclaration;

LocalVariableDeclaration:

VariableModifiers Type VariableDeclarators

ResultType:

Type

void

VariableDeclarators:

VariableDeclarator

VariableDeclarators , VariableDeclarator

VariableDeclarator:

VariableDeclaratorId

VariableDeclaratorId = VariableInitializer

StaticInitializer:

static *Block*

รูปที่ 2.11 ไวยากรณ์ภาษาจาวาสำหรับการนิยามคลาส (ต่อ)

VariableDeclaratorId:

Identifier

VariableDeclaratorId [ ]

VariableInitializer:

Expression

ArrayInitializer

ArrayInitializer:

{ *[VariableInitializer* {, *VariableInitializer*} *[,]]* }

ConstructorDeclaration:

ConstructorModifiers$_{opt}$ *ConstructorDeclarator Throws*$_{opt}$ *ConstructorBody*

ConstructorDeclarator:

TypeParameters$_{opt}$ SimpleTypeName (FormalParameterList$_{opt}$)

Throws:

throws *ExceptionTypeList*

ExceptionTypeList:

ExceptionType

ExceptionTypeList , ExceptionType

ExceptionType:

ClassType

TypeVariable

ConstructorBody:

{ ExplicitConstructorInvocation$_{opt}$ *BlockStatements*$_{opt}$ }

รูปที่ 2.11 ไวยากรณ์ภาษาจาวาสำหรับการนิยามคลาส (ต่อ)

ConstructorModifiers:

ConstructorModifier

ConstructorModifiers ConstructorModifier

ConstructorModifier: one of

Annotation public protected private

FieldDeclaration:

FieldModifiers$_{opt}$ *Type VariableDeclarators*;

AbstractMethodDeclaration:

AbstractMethodModifiers$_{opt}$ *TypeParameters*$_{opt}$ *ResultType MethodDeclarator Throws*$_{opt}$;

FieldModifiers:

FieldModifier

FieldModifiers FieldModifier

AbstractMethodModifiers:

AbstractMethodModifier

AbstractMethodModifiers AbstractMethodModifier

AbstractMethodModifier: one of

Annotation public abstract

EnumDeclaration:

ClassModifiers$_{opt}$ enum *Identifier Interfaces*$_{opt}$ *EnumBody*

EnumBody:

{ *EnumConstants*$_{opt}$,$_{opt}$ *EnumBodyDeclarations*$_{opt}$ }

รูปที่ 2.11 ไวยากรณ์ภาษาจาวาสำหรับการนิยามคลาส (ต่อ)

FieldModifier: one of

Annotation public protected private static final transient volatile

NormalInterfaceDeclaration:

InterfaceModifiers$_{opt}$ interface *Identifier TypeParameters*$_{opt}$ *ExtendsInterfaces*$_{opt}$ *InterfaceBody*

ExtendsInterfaces:

extends *InterfaceType*

ExtendsInterfaces , InterfaceType

InterfaceType:

TypeDeclSpecifier TypeArguments$_{opt}$

InterfaceModifier: one of

Annotation public protected private abstract static strictfp

InterfaceModifiers:

InterfaceModifier

InterfaceModifiers InterfaceModifier

InterfaceBody:

{ *InterfaceMemberDeclarations*$_{opt}$ }

Super:

extends ClassType

InterfaceMemberDeclarations:

InterfaceMemberDeclaration

InterfaceMemberDeclarations InterfaceMemberDeclaration

รูปที่ 2.11 ไวยากรณ์ภาษาจาวาสำหรับการนิยามคลาส (ต่อ)

InterfaceMemberDeclaration:

ConstantDeclaration

AbstractMethodDeclaration

ClassDeclaration

InterfaceDeclaration

;

EnumConstants:

EnumConstant

EnumConstants , EnumConstant

EnumConstant:

Annotations Identifier $Arguments_{opt}$ $ClassBody_{opt}$

Arguments:

(ArgumentListopt)

EnumBodyDeclarations:

$ClassBodyDeclarations_{opt}$

ConstantDeclaration:

$ConstantModifiers_{opt}$ *Type VariableDeclarators;*

ConstantModifiers:

ConstantModifier

ConstantModifier ConstantModifers

ConstantModifier: one of

Annotation public static final

InstanceInitializer:

Block

รูปที่ 2.11 ไวยากรณ์ภาษาจาวาสำหรับการนิยามคลาส (ต่อ)

EnumConstants:

Interfaces:

implements *InterfaceTypeList*

InterfaceTypeList:

InterfaceType

InterfaceTypeList , InterfaceType

ClassBody:

{ $ClassBodyDeclarations_{opt}$ }

รูปที่ 2.11 ไวยากรณ์ภาษาจาวาสำหรับการนิยามคลาส (ต่อ)

รูปที่ 2.12 แสดงตัวอย่างรูปแบบการนิยามคลาสโดยทั่วไปตามไวยากรณ์ภาษาจาวาแบบง่าย ซึ่งในตัวอย่างนี้จะมีการนิยามคลาสที่ชื่อ Classname โดยมีแอททริบิวต์และเมท็อดอย่างละ N ซึ่ง N คือ จำนวนเต็มบวกของลำดับที่เป็นไปได้สำหรับแอททริบิวต์และเมท็อด

รูปที่ 2.13 แสดงตัวอย่างการนิยามคลาสโดยกำหนดค่าคลาสโมดิฟายเออร์ (Class modifiers) เป็น abstract โดยคลาส Point เป็นคลาสเริ่มต้นหรือคลาสแม่ที่สืบทอดคุณสมบัติด้วยคลาส ColoredPoint ดังนั้นคลาส ColoredPoint จะมีคุณสมบัติต่างๆ เช่นเดียวกับคลาส Point นอกจากนั้นคลาส ColoredPoint ยังมีการนิยามแอททริบิวต์ ได้แก่ ตัวแปร color และคลาส SimplePoint สืบทอดคุณสมบัติจากคลาส Point เช่นเดียวกัน แต่มีการเพิ่มส่วนของการทำงานของเมท็อด alert ในคลาสนี้ ซึ่งจากตัวอย่างดังกล่าวจะพบว่า นักพัฒนาซอฟต์แวร์ไม่สามารถสร้างอ็อบเจกต์หรืออินแสตนซ์ของคลาสโดยใช้ ColoredPoint หรือ Point ดังแสดงนิพจน์ดังนี้

```
Point thePoint = new Point()
ColoredPoint theColoredPoint = new ColoredPoint()
```

ซึ่งภาษาจาวาจะแสดงข้อความผิดพลาดในขณะที่คอมไพล์ซอร์สโค้ด

นักพัฒนาซอฟต์แวร์สามารถสร้างอ็อบเจกต์ด้วยคลาส SimplePoint ดังแสดงนิพจน์ดังนี้

Point thePoint = new SimplePoint ()

SimplePoint the SimplePoint = new SimplePoint ()

เนื่องจากคลาส SimplePoint ไม่ได้นิยามเป็นคลาสแบบ abstract

```
class Classname
{
   type instance-variable1;
   type instance-variable2;
   // ...
   type instance-variableN;

   type methodname1(parameter-list)
   {
      // body of method
   }

   type methodname2(parameter-list)
   {
      // body of method
   }
   // ...

   type methodnameN(parameter-list)
   {
      // body of method
   }
}
```

รูปที่ 2.12 ตัวอย่างรูปแบบการนิยามคลาสโดยทั่วไปตามไวยากรณ์ภาษาจาวาแบบง่าย

รูปที่ 2.14 แสดงตัวอย่างการนิยามอินเนอร์คลาส (Inner class) หรือนิยามคลาสที่อยู่ภายในคลาส ซึ่งคลาส WithDeepNesting เป็นคลาสที่อยู่ภายนอกสุด โดยมีแอททริบิวต์เป็นตัวแปร toBe ที่เป็นชนิดข้อมูลตรรกะ และมีอินเนอร์คลาสคือ คลาส Nested ส่วนคลาส Nested มีแอททริบิวต์เป็นตัวแปร theQuestion ที่ชนิดข้อมูลตรรกะ และมีอินเนอร์คลาสคือคลาส DeeplyNested ซึ่งอินเนอร์คลาสสามารถเรียกใช้งานหรือเข้าถึงข้อมูลแอททริบิวต์ของคลาสภายนอกได้ เช่น theQuestion = toBe || !toBe;

```
abstract class Point
{
  int x = 1, y = 1;

   void move(int dx, int dy)
  {
     x += dx;
    y += dy;
    alert();
   }
    abstract void alert();
}

abstract class ColoredPoint extends Point
{
  int color;
}

class SimplePoint extends Point
{
  void alert() {… }
}
```

รูปที่ 2.13 ตัวอย่างการนิยามคลาสโดยกำหนดค่า Class modifiers เป็น abstract

```
class WithDeepNesting
{
  boolean toBe;

  WithDeepNesting(boolean b)
  {
    toBe = b;
  }

  class Nested
  {
    boolean theQuestion;
    class DeeplyNested
    {
      DeeplyNested()
      {
        theQuestion = toBe || !toBe;
      }
    }
  }
}
```

รูปที่ 2.14 ตัวอย่างการนิยามอินเนอร์คลาส

รูปที่ 2.15 แสดงตัวอย่างการนิยามคลาสโดยการสืบทอดคุณสมบัติจากคลาสอื่น ซึ่งคลาส Point จะเป็นซุปเปอร์คลาสที่ถ่ายทอดคุณสมบัติไปที่คลาส ColoredPoint และคลาส ColoredPoint ถ่ายทอดคุณสมบัติไปที่คลาส Colored3dPoint รวมทั้งคุณสมบัติของคลาส Point ด้วย โดยคลาส Colored3dPoint จะไม่สามารถถ่ายทอดคุณสมบัติไปคลาสใดๆ ได้อีก เนื่องจากมีการกำหนดค่า Class modifiers เป็น final

```
class Point
{
   int x, y;
}

class ColoredPoint extends Point
{
   int color;
}

final class Colored3dPoint extends ColoredPoint
{
   int z;
}
```

รูปที่ 2.15 ตัวอย่างการนิยามคลาสโดยการสืบทอดคุณสมบัติจากคลาสอื่น

รูปที่ 2.16 แสดงตัวอย่างการนิยามคลาสโดยการสืบทอดคุณสมบัติจากคลาสและอินเตอร์เฟสคลาส โดยคลาส ColoredPoint จะสืบทอดคุณสมบัติจากคลาส Point และอินเตอร์เฟสคลาส Colorable ซึ่งคลาส ColoredPoint จำเป็นต้องนิยามเมท็อด setColor และเมท็อด getColor ที่สืบทอดมาจากอินเตอร์เฟสคลาส Colorable และต้องเพิ่มส่วนการทำงานของทั้งสองเมท็อดด้วย หากนักพัฒนาซอฟต์แวร์ไม่ได้นิยามเมท็อดดังกล่าว ภาษาจาวาจะแสดงข้อผิดพลาดในระหว่างการคอมไพล์

1.1.4 อินเตอร์เฟส

อินเตอร์เฟสเป็นชนิดข้อมูลแบบอ้างอิงอีกชนิดหนึ่ง โดยเมท็อดที่นิยามภายในอินเตอร์เฟสจะไม่มีการทำงานของเมท็อด รูปที่ 2.11 แสดงไวยากรณ์ของอินเตอร์เฟส โดยรูปแบบการนิยามอินเตอร์เฟสมีรายละเอียดคล้ายคลึงกับการนิยามคลาส แต่การนิยามอินเตอร์เฟสไม่สามารถเขียนส่วนการทำงานของเมท็อดได้ ดังนั้นนักพัฒนาซอฟต์แวร์จะไม่สามารถใช้ชนิดข้อมูลอินเตอร์เฟสมาสร้างอ็อบเจกต์ได้ดังนี้

```
IPoint theIPoint = new IPoint();
```

จากนิพจน์ข้างต้น IPoint คืออินเตอร์เฟสที่นิยามขึ้นมา และตัวแปร theIPoint มีชนิดข้อมูลเป็นอินเตอร์เฟส IPoint ซึ่งเมื่อทำการคอมไพล์ซอร์สโค้ด ภาษาจาวาจะแสดงข้อผิดพลาดในระหว่างการคอมไพล์ เนื่องจากชนิดข้อมูลอินเตอร์เฟสไม่สามารถนำมาสร้างอ็อบเจกต์ได้ อย่างไรก็ตาม นักพัฒนาซอฟต์แวร์สามารถใช้คลาส Point ซึ่งได้สืบทอดคุณสมบัติมาจากอินเตอร์เฟส IPoint เพื่อสร้างอ็อบเจ็กต์ดังนี้

```
IPoint theIPoint = new Point();
```

```
interface Colorable
{
   void setColor(int color);
   int getColor();
}

class Point
{
   int x, y;
}

class ColoredPoint extends Point implements Colorable
{
   int color;
   void setColor(int color) {…}
   int getColor() {…}
}
```

รูปที่ 2.16 ตัวอย่างการนิยามคลาสโดยการสืบทอดคุณสมบัติจากคลาสและอินเตอร์เฟสคลาส

รูปที่ 2.17 แสดงตัวอย่างการนิยามอินเตอร์เฟสและการสืบทอดคุณสมบัติอินเตอร์เฟส โดยคลาส FixedStack เป็นคลาสที่สืบทอดคุณสมบัติจากอินเตอร์เฟส IntStack ซึ่งอินเตอร์เฟส IntStack ประกอบด้วยสองเมท็อด ได้แก่ เมท็อด push และเมท็อด pop ดังนั้นคลาส FixedStack

จำเป็นต้องกำหนดส่วนของการทำงานสำหรับเมท็อด push และเมท็อด pop ที่สืบทอดคุณสมบัติมา หากไม่กำหนดส่วนของการทำงานสำหรับทั้งสองเมท็อด ภาษาจาวาจะแสดงข้อผิดพลาดในระหว่างการคอมไพล์

```
// Define an integer stack interface.
interface IntStack
{
   void push(int item); // store an item
   int pop(); // retrieve an item
}

// An implementation of IntStack that uses fixed storage.
class FixedStack implements IntStack
{
   private int stck[];
   private int tos;

   // allocate and initialize stack
   FixedStack(int size)
   {
      stck = new int[size];
      tos = -1;
   }

   // Push an item onto the stack
```

รูปที่ 2.17 ตัวอย่างการนิยามอินเตอร์เฟสและการสืบทอดคุณสมบัติอินเตอร์เฟส

```
  public void push(int item)
  {
    if(tos==stck.length-1) // use length member
      System.out.println("Stack is full.");
    else
      stck[++tos] = item;
  }

  // Pop an item from the stack
  public int pop()
  {
    if(tos < 0)
    {
      System.out.println("Stack underflow.");
      return 0;
    }
    else
    {
      return stck[tos—];
    }
  }
}
```

รูปที่ 2.17 ตัวอย่างการนิยามอินเตอร์เฟสและการสืบทอดคุณสมบัติอินเตอร์เฟส (ต่อ)

1.2. มาตรวัดความเหมือนกันของบริบท

มาตรวัดความเหมือนกันของบริบทเป็นมาตรวัดที่นิยามขึ้นในงานวิจัยของนัยนา [4] ซึ่งเป็นงานวิจัยที่วิทยานิพนธ์นี้พัฒนาเพิ่มเติม มาตรวัดความเหมือนกันจะใช้ในขั้นตอนของการจัดอันดับผลลัพธ์ ในส่วนของการคำนวนค่าวิทยาการสำนึกแบบบริบท โดยรายละเอียดของมาตรวัดมีดังนี้

มาตรวัดความเหมือนกันของบริบทจะวัดความเหมือนกันระหว่างบริบทโค้ดที่กำลังพัฒนาและบริบทโค้ดในคลังข้อมูล มาตรวัดความเหมือนกันของบริบทสามารถแทนด้วยสัญลักษณ์ $M_{CT}(Q, s)$ โดยจะวัดค่าความเหมือนกันระหว่าง

- บริบทแบบพาเรนท์ CP_s และบริบทแบบชนิดข้อมูล CT_s ซึ่งทั้ง CP_s และ CT_s เป็นบริบทที่หาได้จากโค้ดในคลังข้อมูล
- บริบทแบบพาเรนท์ CP_q และบริบทแบบชนิดข้อมูล CT_q ที่หาได้จากโค้ดที่กำลังพัฒนา และระบุไว้ภายในคำร้องขอ Q

มาตรวัดความเหมือนกันของบริบท สามารถเขียนเป็นสมการได้ดังสมการที่ (1)

$$M_{CT}(Q, s) = \frac{M_P(Q,s) + M_{VT}(Q,s)}{2} \quad \text{_____ (1)}$$

เมื่อ

Q = คำร้องขอซึ่งจะประกอบไปด้วยชนิดของอ็อบเจกต์ t_q และ บริบทโค้ด CT

s = รูปแบบการใช้งานของอ็อบเจกต์ t_q

$M_P(Q, s)$ = มาตรวัดเชิงปริมาณที่ใช้วัดค่าความเหมือนกันระหว่าง พาเรนท์ CP_q และ CP_s ของคำร้องขอ Q และ s ตามลำดับ

$M_{VT}(Q, s)$ = มาตรวัดเชิงปริมาณระหว่างชนิดข้อมูลวิซิเบิล (Visible Type) ที่อยู่ใน Q และ s โดยชนิดข้อมูลวิซิเบิล หมายถึงชนิดข้อมูลที่อยู่ในคลาสและชนิดข้อมูลที่อยู่ในเมท็อดที่กำลังพัฒนา ได้แก่ แอททริบิวต์ของคลาส ชนิดข้อมูลในเมท็อดซิกเน-เจอร์ และตัวแปรโลคอลที่ประกาศในเมท็อดที่กำลังสนใจ

มาตรวัดความเหมือนกันระหว่างพาเรนท์ $M_P(Q, s)$ คือค่าเฉลี่ยของความเหมือนกันระหว่างซุปเปอร์คลาส $M_S(Q, s)$ และความเหมือนกันระหว่างอินเตอร์เฟส $M_I(Q, s)$ ของ Q และ s ตามลำดับ $M_P(Q, s)$ สามารถเขียนเป็นสมการได้ดังสมการที่ (2)

$$M_P(Q, s) = \frac{M_S(Q,s) + M_I(Q,s)}{2} \quad \text{_____ (2)}$$

โดยที่

$$M_S(Q, s) = M_T\big(\text{superclass}(Q), \text{superclass}(s)\big) \quad \text{_____ (3)}$$

$$M_I(Q, s) = \frac{I+S}{|\text{intf}(Q)|+|\text{intf}(s)|} \quad \text{_____ (4)}$$

$$I = \sum_{i_q \in \text{intf}(Q)} \left[\max_{i_s \in \text{intf}(s)} M_T(i_q, i_s)\right] \quad \text{_____ (5)}$$

$$S = \sum_{i_s \in \text{intf}(s)} \left[\max_{i_q \in \text{intf}(Q)} M_T(i_s, i_q)\right] \quad \text{_____ (6)}$$

เมื่อ

$superclass(Q), superclass(s)$ = กลุ่มของซูเปอร์คลาสที่อยู่ใน Q และ s ตามลำดับ

$intf(Q),\ intf(s)$ = กลุ่มของอินเตอร์เฟสที่อยู่ใน Q และ s ตามลำดับ

$i_q \in intf(Q)$ = อินเตอร์เฟสที่อยู่ในกลุ่ม $intf(Q)$

$i_s \in intf(s)$ = อินเตอร์เฟสที่อยู่ในกลุ่ม $intf(s)$

$M_T(i_q, i_s)$ = ค่าความเหมือนกันระหว่าง i_q และ i_s

$M_I(Q, s)$ = ค่าเฉลี่ยของความเหมือนกันระหว่างอินเตอร์เฟสสองกลุ่ม ได้แก่ $intf(Q)$ และ $intf(s)$

ค่าความเหมือนกัน $M_{VT}(Q, s)$ จะคำนวณจากชนิดข้อมูลวิชิเบิลที่อยู่ในคำร้องขอ Q และชนิดข้อมูลวิชิเบิลที่อยู่ในโค้ด s ซึ่ง $M_{VT}(Q, s)$ สามารถเขียนเป็นสมการได้ดังนี้

$$M_{VT}(Q, s) = \frac{\sum_{t_c \in type(Q)} [\max_{t_s \in type(s)} M_T(t_c, t_s)]}{|type(Q)|} \quad \text{_____ (7)}$$

เมื่อ

$type(Q), type(s)$ = กลุ่มของบริบทที่อยู่ใน Q และ s ตามลำดับ

$t_c \in type(Q)$ = บริบทที่อยู่ในกลุ่ม $type(Q)$

$t_s \in type(s)$ = บริบทที่อยู่ในกลุ่ม $type(s)$

$M_T(t_c, t_s)$ = ค่าความเหมือนกันระหว่าง t_c และ t_s

$M_{VT}(Q, s)$ จัดเป็นมาตรวัดแบบไม่สมมาตร (Asymmetric) เนื่องจากจะคำนวณหาค่าเฉลี่ยของความเหมือนกันที่มีค่ามากที่สุดของแต่ละ t_c และ t_s ในมุมมองของคำร้องขอ Q เป็นหลัก

ค่าความเหมือนกันของชนิดข้อมูล (Type Match) $M_T(t_c, t_s)$ ในสมการที่ (3) - (6) จะคำนวณโดยอัลกอริทึมชนิดข้อมูลแมทช์ที่จะอธิบายในส่วนถัดไป

1.3. อัลกอริทึมจับคู่ชนิดข้อมูล (Type Match Algorithm)

อัลกอริทึมจับคู่ชนิดข้อมูลจะคำนวณค่าความเหมือนกันระหว่างสองชนิดข้อมูล โดยใช้หมายเลขดิวอีเปรียบเทียบ หมายเลขดิวอีคือหมายเลขที่แสดงค่าความสัมพันธ์เชิงลำดับชั้นระหว่างชนิดข้อมูล โดยแต่ละหมายเลขดิวอีจะบ่งบอกความยาวของเส้นทางระหว่างชนิดข้อมูลของดิวอีไปยังพาเรนท์ของชนิดข้อมูลนั้น

ตัวอย่างที่ 1 ชนิดข้อมูลประเภทคลาส (Class Types)

รูปที่ 2.18 แสดงตัวอย่างลำดับชั้นของคลาส โดยที่แต่ละคลาสจะมีหมายเลขดิวอีกำกับด้านล่าง คลาส Object ซึ่งเป็นต้นตระกูลของทุกๆ คลาสจะมีหมายเลขดิวอีเป็น 1.1 โดยที่ เลข 1 ตัวแรกหมายถึงประเภทของคลาส และเลข 1 ตัวถัดมาหมายถึงลำดับชั้นของชนิดข้อมูล ซึ่งเลข 1 จะถือว่าเป็นเลขที่ให้ค่าความเจเนริก (Generic) มากที่สุด

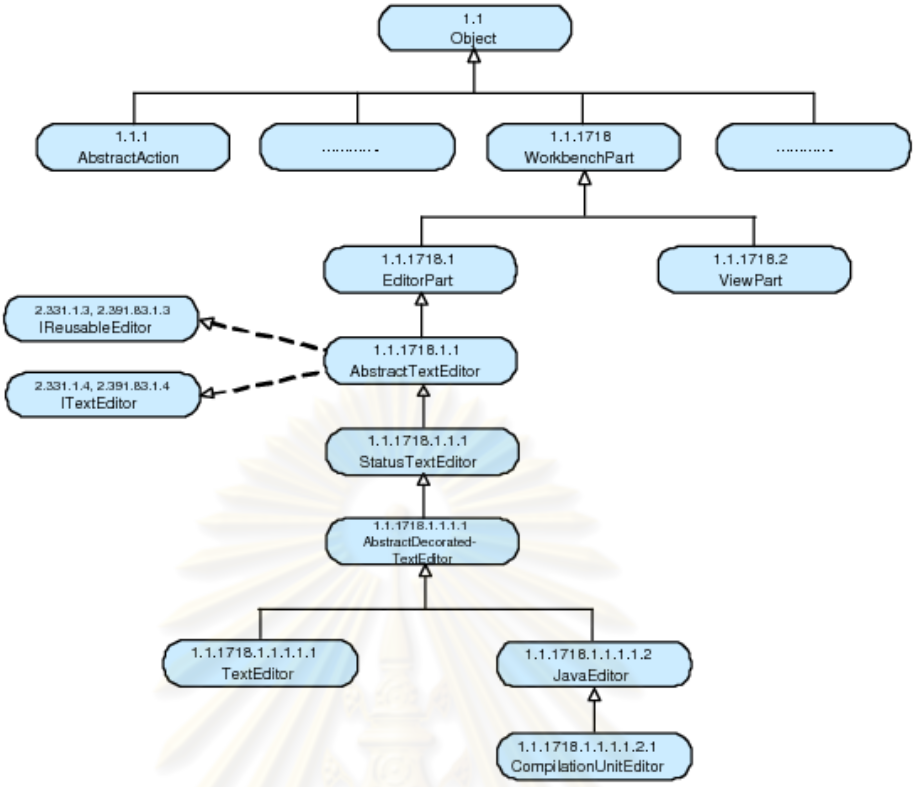
หมายเลขดิวอีสำหรับซับคลาสของคลาส Object จะคำนวณโดยส่งต่อหมายเลขดิวอีของพาเรนท์ให้กับซับคลาส โดยที่หลังสัญลักษณ์ "." ให้ตามด้วยหมายเลขลำดับซิบลิงค์ (Sibling Order Number) ยกตัวอย่างเช่น หมายเลขดิวอีของคลาส AbstractAction และ WorkbenchPart โดยที่ทั้งสองคลาสเป็นซับคลาสของคลาส Object จะมีหมายเลขดิวอีเป็น 1.1.1 และ 1.1.1718 ตามลำดับ โดยที่

- เลข 1.1 หมายถึงซุปเปอร์คลาส ซึ่งก็คือ คลาส Object
- เลข 1 และ 1718 หมายถึงลำดับของคลาสที่เป็นลูกของคลาส Object
- จำนวนของ "." ที่อยู่ในหมายเลขดิวอีจะแสดงถึงความยาวของเส้นทาง (Path Length) ของชนิดข้อมูลในลำดับชั้น ยกตัวอย่างเช่น ในหมายเลขดิวอี 1.1.1718 มีจำนวน "." สองตัว ดังนั้นเส้นทางของ WorkbenchPart จึงมีความยาวเท่ากับ 2

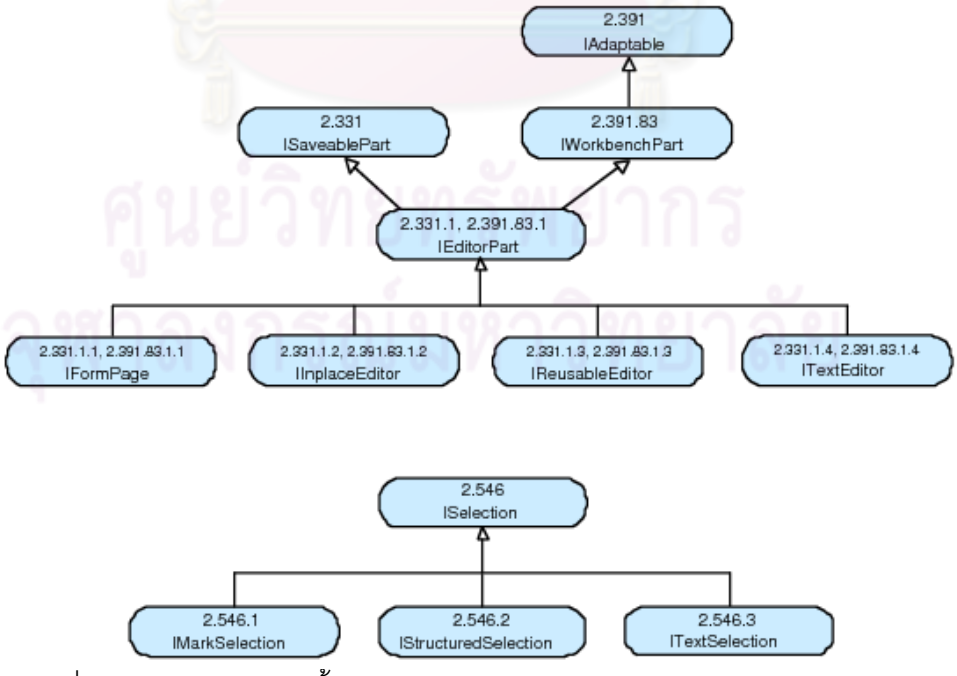
ตัวอย่างที่ 2 ชนิดข้อมูลประเภทอินเตอร์เฟส (Interface Types)

รูปที่ 2.19 แสดงลำดับชั้นของคลาส IEditorPart และ ISelection หมายเลขดิวอีของแต่ละอินเตอร์เฟสจะคำนวณเหมือนชนิดข้อมูลประเภทคลาส ยกเว้นเลขหน้าสุดจะกลายเป็นเลข 2 เพื่อระบุว่าเป็นชนิดข้อมูลประเภทอินเตอร์เฟส

นอกจากนี้คลาสอาจจะอิมพลีเมนท์มาจากหลายอินเตอร์เฟส ซึ่งคลาสเหล่านั้นอาจมีหมายเลขดิวอีได้หลายหมายเลข ยกตัวอย่างเช่น รูปที่ 2.19 อินเตอร์เฟส IEditorPart อิมพลีเมนท์มาจากซุปเปอร์อินเตอร์เฟส (Super-Interface) ISaveablePart และ IWorkbenchPart ดังนั้นอินเตอร์เฟส IEditorPart จึงมีหมายเลขดิวอีสองหมายเลข ได้แก่ 2.331.1 และ 2.391.83.1 ซึ่งมาจากอินเตอร์เฟสที่ต่างกันสองอินเตอร์เฟส อินเตอร์เฟส ITextEditor ก็มีหมายเลขดิวอี 2 หมายเลขเช่นเดียวกัน ได้แก่ 2.331.1.4 และ 2.391.83.1.4



รูปที่ 2.18 ตัวอย่างลำดับชั้นของคลาสและหมายเลขดิวอีของ
คลาส AbstractAction และ WorkbenchPart



รูปที่ 2.19 ตัวอย่างลำดับชั้นของคลาสและหมายเลขดิวอีของคลาส IEditorPart

ตัวอย่างที่ 3 ชนิดข้อมูลประเภทคลาสและอินเตอร์เฟส (Class and Interface Types)

สมมติให้ อินเตอร์เฟส I อิมพลีเมนท์โดยคลาส C หมายเลขดิวอีของอินเตอร์เฟส I จะส่งต่อให้แก่คลาส C หมายเลขดิวอีของคลาส C จะคำนวณคล้ายกับชนิดข้อมูลประเภทคลาสหรือชนิดข้อมูลประเภทอินเตอร์เฟส แต่หมายเลขสุดท้ายที่ต่อท้ายหมายเลขดิวอีจะลดลงตั้งแต่ -1 เป็นต้นไป โดยค่าจะลดลงตามลำดับของคลาสที่อิมพลีเมนท์อินเตอร์เฟส I ยกตัวอย่างเช่น ในรูปที่ 2.18 คลาส AbstractTextEditor ที่สืบทอดมาจากคลาส EditorPart และอิมพลีเมนท์อินเตอร์เฟส IReusableEditor และ ITextEditor หมายเลขดิวอีที่สืบทอดให้คลาส AbstractTextEditor มีดังต่อไปนี้

- 1.1.1718.1.1 ได้มาจากหมายเลขดิวอี 1.1.1718.1 ของคลาส EditorPart
- 2.331.1.3.-1 และ 2.391.83.1.3.-1 ได้มาจากหมายเลขดิวอี 2.331.1.3 และ 2.391.83.1.3 ของอินเตอร์เฟส IReusable
- 2.331.1.4.-1 และ 2.391.83.1.4.-1 ได้มาจากหมายเลขดิวอี 2.331.1.4 และ 2.391.83.1.4 ของคลาส ITextEditor

เช่นเดียวกันกับคลาส IStatusTextEditor ซึ่งสืบทอดมาจากคลาส AbstractTextEditor ก็จะได้รับหมายเลขดิวอีมาทั้งหมด 5 หมายเลข ได้แก่ 1.1.1718.1.1.1 2.331.1.3.-1.1 2.391.83.1.3.-1.1 2.331.1.4.-1.1 และ 2.391.83.1.4.-1.1

รูปที่ 2.20 และรูปที่ 2.21 แสดงอัลกอริทึมของจับคู่ชนิดข้อมูล โดยที่แต่ละชนิดข้อมูลสามารถมีหมายเลขดิวอีได้มากกว่าหนึ่งหมายเลข โดยอัลกอริทึมจะให้ผลลัพธ์เป็นค่าความเหมือนกันที่มีค่ามากที่สุดระหว่างสองชนิดข้อมูล

```
float typeMatch (String: t_q, String t_s)
{
  ArrayList deweyList_q  =  deweyInstance.lookupDewey(t_q);
  ArrayList deweyList_s  =  deweyInstance.lookupDewey(t_s);

  double max = 0.0;
   for each dewey_q ∈ deweyList_q
  {
    for each dewey_s ∈ deweyList_s
```

รูปที่ 2.20 อัลกอริทึมของจับคู่ชนิดข้อมูล

```
        double weight = typeMatchHelper(t_q, deweyq, t_s, dewey_s);
        if (weight > max) max = weight;
      }
    }
    return max;
}
```

รูปที่ 2.20 อัลกอริทึมของจับคู่ชนิดข้อมูล (ต่อ)

```
float typeMatchHelper (String: t_q, String: dewey_q, String: t_s, String: dewey_s)
{
    if (dewey_q == dewey_s)
        return 1.0;
    else if (isGeneralized(dewey_q, dewey_s))
    {
        length_q = pathLength(dewey_q);
        length_s = pathLength(dewey_s);
        return (length_q * 2) / (length_q + length_s);
    }
    else if (isSpecialized(dewey_q, dewey_s))
    {
        length_q = pathLength(dewey_q);
        length_s  = pathLength(dewey_s);
        return (length_s * 2) / (length_q + length_s);
    }
    else if (shareParent(dewey_q, dewey_s))
    {
        t_p = latestParent(dewey_q, dewey_s);
        length_p = lastestParent(dewey_s);
```

รูปที่ 2.21 อัลกอริทึมของไทป์แมทช์เฮลเปอร์

```
    length_q = pathLength(dewey_q);
    length_s  = pathLength(dewey_s);
    return (length_p * 2) / (length_q + length_s);
  }
  return 0.0;
}
```

รูปที่ 2.21 อัลกอริทึมของไทป์แมทช์เฮลเปอร์ (ต่อ)

2. งานวิจัยที่เกี่ยวข้อง

ในปัจจุบันมีงานวิจัยหลากหลายงานที่นำเสนอการวิเคราะห์โค้ดในคลังข้อมูล โดยสามารถแบ่งออกเป็น 2 ประเภท ได้แก่ การวิเคราะห์โค้ดแบบไดนามิก (Dynamic Analysis) [7-11] และแบบสถิต (Static Analysis)

การวิเคราะห์โค้ดแบบไดนามิกเป็นการวิเคราะห์เส้นทางการทำงานโค้ดระหว่างที่ซอฟต์แวร์กำลังทำงาน ซึ่งทำให้ผู้วิเคราะห์ไม่จำเป็นต้องหาเส้นทางการทำงานของซอฟต์แวร์ด้วยตนเอง เนื่องจากสามารถตรวจสอบเส้นทางการทำงานของซอฟต์แวร์ได้ในระหว่างที่ซอฟต์แวร์กำลังทำงานโดยตรง เช่น วิเคราะห์เส้นทางของโค้ดโดยการดีบั๊คโปรแกรม เป็นต้น แต่การวิเคราะห์แบบไดมามิกก็มีข้อจำกัด เพราะการวิเคราะห์แบบไดนามิกต้องกระทำในขณะที่ซอฟต์แวร์กำลังทำงาน ดังนั้นโค้ดที่นำมาวิเคราะจะต้องครบถ้วนจึงต้องรันโปรแกรมได้ นอกจากนี้ยังต้องเตรียมข้อมูลนำเข้าให้ครอบคลุมทุกกรณี เพื่อให้ซอฟต์แวร์สามารถทำงานในทุกเส้นทางที่เป็นไปได้

การวิเคราะห์โค้ดแบบสถิตเป็นการวิเคราะห์เส้นทางของโค้ดโดยไม่จำเป็นต้องมีการทำงานของซอฟต์แวร์เกิดขึ้น ซึ่งการวิเคราะห์แบบนี้จะวิเคราะห์จากตัวอักษรของโค้ดโดยตรง ทำให้สามารถวิเคราะห์เพียงบางส่วนของซอฟต์แวร์ โดยไม่จำเป็นต้องใช้ซอฟต์แวร์ทั้งหมดที่พัฒนาขึ้นในโครงการ นอกจากนั้นซอฟต์แวร์ที่นำมาวิเคราะห์ไม่จำเป็นต้องทำงานได้จริงในขณะที่ทำการวิเคราะห์ และในปัจจุบันมีโอเพนซอร์สไลบรารีที่สนับสนุนการวิเคราะห์โค้ดแบบสถิตมากมาย เช่น PMD หรือ Abstract Syntax Tree (AST) เป็นต้น

วิทยานิพนธ์นี้เลือกใช้วิธีวิเคราะห์โค้ดแบบสถิต โดยโค้ดที่นำมาวิเคราะห์ไม่จำเป็นต้องทำงานได้ และยังช่วยลดเวลาในการสร้างข้อมูลนำเข้าให้หลากหลายเพื่อให้โค้ดได้ใช้งานทุกส่วน

งานวิจัยที่ใช้การวิเคราะห์แบบสถิตมีอยู่หลายรูปแบบ เช่น การวิเคราะห์ความเปลี่ยนแปลงของซอฟต์แวร์ (Software Change Analysis) [12-15] การตรวจจับข้อผิดพลาดในซอฟต์แวร์ (Software Defect Detection) [15-22] การค้นคืนข้อมูล (Information Retrieval) [7, 23, 24] การ

ทำเหมืองรูปแบบที่เกิดขึ้นเป็นประจำ (Frequent pattern mining) [5, 12, 15, 22] ในงานวิจัย [25, 26] ได้กล่าวสรุปงานวิจัยที่เกี่ยวข้องกับการวิเคราะห์โค้ดในคลังข้อมูลไว้ โดยวิทยานิพนธ์นี้จะจัดอยู่ในประเภทการวิเคราะห์โค้ดเพื่อแนะนำโค้ดตัวอย่างให้แก่ผู้ใช้ งานวิจัยที่จัดอยู่ในประเภทดังกล่าวมีดังต่อไปนี้

2.1. งานวิจัย Approximate Structural Context Matching: An Approach to Recommend Relevant Examples โดย Reid, H. [1]

งานวิจัยนี้นำเสนอวิธีค้นหาโค้ดตัวอย่างจากคลังข้อมูล โดยนำคีย์เวิร์ดที่ต้องการหาโค้ดตัวอย่างและบริบทโค้ดของโปรแกรมที่กำลังพัฒนาไปเปรียบเทียบกับโค้ดในคลังข้อมูล หากบริบทโค้ดและโค้ดตัวอย่างมีความคล้ายคลึงกันก็จะเลือกขึ้นมาเป็นผลลัพธ์

ข้อจำกัดของงานวิจัยนี้ มีรายละเอียดดังนี้

- งานวิจัยนี้ให้ผลลัพธ์ทั้งเมท็อดที่มีโค้ดที่เกี่ยวข้องเท่านั้น แต่เนื่องจากในเมท็อดหนึ่งๆ อาจมีโค้ดบางส่วนที่ไม่เกี่ยวข้องกับสิ่งที่ต้องการ การแสดงผลลัพธ์ทั้งเมท็อดอาจทำให้ผู้ใช้สับสนว่าโค้ดส่วนใดในเมท็อดที่เป็นผลลัพธ์ที่ต้องการ
- งานวิจัยนี้นำบริบทโค้ดไปใช้มากเกินความจำเป็น ซึ่งอาจจะทำให้ได้ผลลัพธ์จำนวนมากที่ไม่เกี่ยวข้องกับความต้องการ หรือบางครั้งจำนวนผลลัพธ์ที่ได้อาจมีน้อยเกินไปจนไม่มีผลลัพธ์ใดๆ

2.2. งานวิจัย MAPO: mining API usages from open source repositories โดย Tao, X., and Jian, P [5]

งานวิจัยนี้นำเสนอวิธีการทำเหมืองรูปแบบการใช้งาน โดยมีการติดต่อโค้ดเสิร์จเอนจินเพื่อใช้เป็นฐานข้อมูล ส่วนการทำเหมืองจะใช้อัลกอริทึมแบบการทำเหมืองลำดับเหตุการณ์แบบปิดที่เกิดขึ้นเป็นประจำ (Mining Closed Frequent Sequence) โดยจะให้ผลลัพธ์เป็นรูปแบบการใช้งานของเอพีไอที่เกิดขึ้นมากที่สุด (Frequent API usage patterns) ในคลังข้อมูล

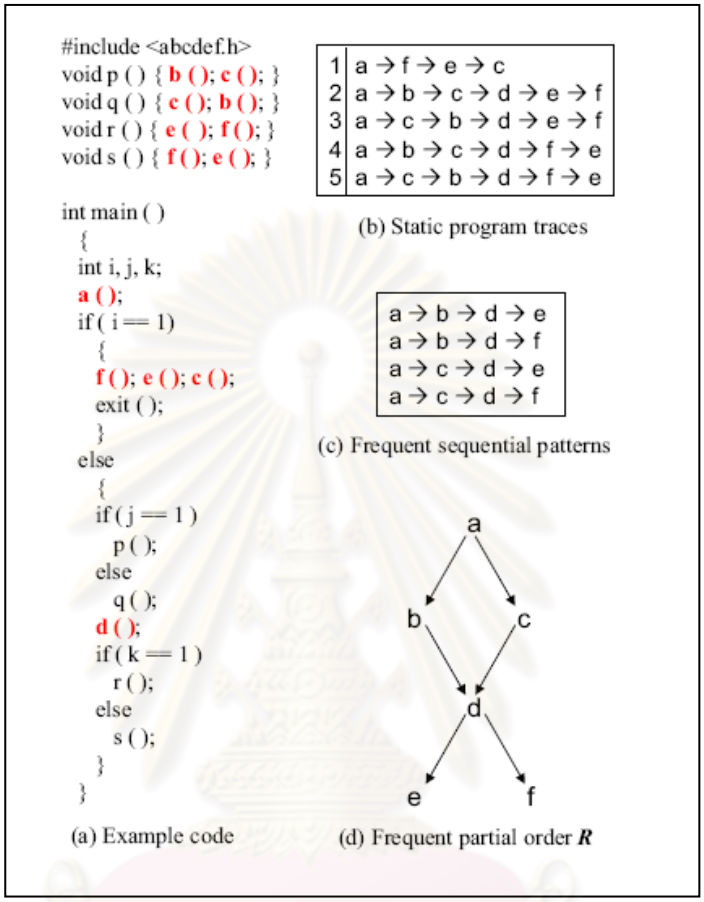
ข้อจำกัดของงานวิจัยนี้ มีรายละเอียดดังนี้

- งานวิจัยนี้พิจารณาความถี่ของโค้ดที่เกิดขึ้นเท่านั้น ไม่ได้พิจารณาการนำบริบทโค้ดเข้าไปสกัดข้อมูลหรือการจัดอันดับว่าผลลัพธ์ใดคือผลลัพธ์ที่ใกล้เคียงกับโค้ดที่ผู้ใช้กำลังพัฒนามากที่สุด
- ผลลัพธ์แสดงเพียงชื่อคลาสและชื่อเมท็อดเท่านั้น ไม่ได้แสดงพารามิเตอร์ของเมท็อด

2.3. งานวิจัย Mining API patterns as partial orders from source code: from usage scenarios to specifications โดย Mithun, A., Tao, X., Jian, P., and Jun, X.[6]

งานวิจัยนี้นำเสนอวิธีทำเหมืองข้อมูลโค้ดให้อยู่ในรูปแบบของพาร์เชียลออเดอร์ (Partial Order) ดังรูปที่ 2.22 แสดงการเปรียบเทียบระดับการแสดงผลลัพธ์ของโค้ดแบบลำดับและแบบ

พาร์เชียลออเดอร์ โดยผลลัพธ์ของโค้ดที่มีรูปแบบพาเชียลออเดอร์ (d) จะสามารถอธิบายภาพรวมของการใช้งานของเอพีไอได้ดีกว่าแบบลำดับธรรมดา (b และ c)



รูปที่ 2.22 แสดงการเปรียบเทียบระหว่างโค้ดแบบลำดับและแบบพาร์เชียลออเดอร์

ข้อจำกัดของงานวิจัยมีรายละเอียดดังนี้

- นักพัฒนาซอฟต์แวร์สามารถนำผลลัพธ์ของโค้ดที่มีรูปแบบลำดับไปใช้ได้ทันที ในขณะที่โค้ดแบบพาร์เชียลออเดอร์เหมาะสมสำหรับการแสดงภาพรวมของโค้ดมากกว่า
- วิทยานิพนธ์นี้ออกแบบบนพื้นฐานของภาษาซี ทำให้ไม่มีการพิจารณาหลักการของอ็อบเจกต์เข้ามาใช้ในการสกัดข้อมูล
- ไม่นำบริบทโค้ดเข้ามาช่วยในการกรองและจัดอันดับผลลัพธ์ พิจารณาเฉพาะความถี่ในการเกิดรูปแบบพาร์เชียลออเดอร์ของโค้ดในคลังข้อมูลเท่านั้น

2.4. งานวิจัย Parseweb: a programmer assistant for reusing open source code on the web โดย Suresh, T., and Tao, X. [3]

งานวิจัยนี้นำเสนอวิธีวิเคราะห์ลำดับการเรียกใช้เมท็อดในการสร้างอ็อบเจกต์ โดยเรียกใช้โค้ดเสิร์จเอนจินเพื่อเป็นฐานข้อมูล จากนั้นจึงนำโค้ดที่ได้มาวิเคราะห์ลำดับของการเรียกเมท็อด (Method Invocation Sequence) จาก AST (Abstract Syntax Tree) และกราฟแบบไดเร็กอไซคลิก (Directed Acyclic Graph) การวิเคราะห์ด้วยกราฟดังกล่าวจะสามารถทำให้วิเคราะห์คอนโทรลโฟลว (Control Flow) ของโค้ดได้ นอกจากนี้งานวิจัยนี้ได้นำเสนอการจัดอันดับผลลัพธ์ โดยใช้วิทยาการศึกษาสำนึกแบบความยาว (Length Heuristic) และแบบความถี่ (Frequency Heuristic) เข้าช่วย ทำให้ผลการทดลองให้ผลลัพธ์ที่ตรงตามความต้องการมากกว่างานวิจัยอื่นๆ และให้ผลดีเทียบเท่ากับงานวิจัย [4] อย่างไรก็ตาม งานวิจัยนี้ไม่ได้ทดสอบกับโจทย์ที่เกี่ยวข้องกับบริบทเนื่องจากวิธีการที่นำเสนอไม่ได้วิเคราะห์บริบทในโค้ด

2.5. งานวิจัย Jungloid mining: helping to navigate the API jungle โดย David, M., Lin, X., Rastislav, B., and Doug, K. [2]

งานวิจัยนี้นำเสนอวิธีวิเคราะห์ลำดับโค้ดจากจังกลอยกราฟ (Jungloid Graph) โดยวิเคราะห์ข้อมูลคำร้องขอที่อยู่ในรูปอ็อบเจกต์ชนิดข้อมูลที่เป็นข้อมูลนำเข้า T_{in} และข้อมูลออก T_{out} จังกลอยกราฟสร้างจากข้อมูลโค้ดในคลังข้อมูลและเมท็อดซิกเนเจอร์ (Method Signature) ในเอกสารเอพีไอ (API Document) โดยจะสกัดการเชื่อมต่อของอ็อบเจกต์ (Object Connection) ผ่านทางการเรียกใช้เมท็อด (Method Call) มาสร้างเป็นกราฟ และการหาผลลัพธ์หาได้จากการท่องตามเส้นทาง (Path Traversal) จากอ็อบเจกต์ชนิดข้อมูล T_{out} ไปยังอ็อบเจกต์ชนิดข้อมูล T_{in} ซึ่งผลลัพธ์ที่ได้คือเส้นทางในการสร้างอ็อบเจกต์ชนิดข้อมูล T_{out} ซึ่งสร้างมาจากอ็อบเจกต์ชนิดข้อมูล T_{in}

ข้อจำกัดของงานวิจัยนี้ มีรายละเอียดดังนี้

- การจัดอันดับผลลัพธ์จะกระทำโดยใช้วิทยาการศึกษาสำนึกแบบความยาวเท่านั้น ซึ่งทำให้การจัดอันดับไม่มีประสิทธิภาพเพียงพอ เนื่องจากผลลัพธ์ที่ต้องการไม่ได้ขึ้นกับความยาวของผลลัพธ์เพียงเท่านั้น แต่ยังขึ้นอยู่กับปัจจัยอื่นๆ ด้วย เช่น ความถี่ที่เกิดขึ้นของผลลัพธ์ หรือความตรงกันระหว่างบริบทโค้ดที่กำลังพัฒนากับโค้ดในคลังข้อมูล

2.6. งานวิจัย XSnippet: mining for sample code โดย Naiyana, S., and Kajal, C [4]

งานวิจัยนี้นำเสนอระเบียบวิธีในการค้นหาตัวอย่างโค้ดสำหรับการสร้างอ็อบเจกต์โดยใช้บริบทโค้ดเข้าช่วย ข้อมูลนำเข้าของระบบคือประเภทอ็อบเจกต์ที่ต้องการหาตัวอย่างโค้ด งานวิจัยนี้นำเสนอประเด็นสำคัญต่างๆ โดยมีรายละเอียดดังนี้

2.6.1. ประเภทของคำร้องขอในงานวิจัย XSnippet

งานวิจัยนี้นำเสนอประเภทของคำร้องขอเพื่อค้นหารูปแบบการสร้างอ็อบเจกต์โดยใช้ประเภทของโค้ดบริบทในการแบ่งประเภทของคำร้องขอดังนี้

2.6.1.1. คำร้องขอแบบพิเศษ (Specialized Query) คำร้องขอประเภทนี้จะนำบริบทโค้ดเข้าไปกรองไฟล์คลาสที่ได้จากคลังข้อมูลเพื่อกำจัดผลลัพธ์ที่ไม่เกี่ยวข้องออก เพื่อให้ได้ผลลัพธ์ที่ตรงกับความต้องการ (Better-fit) มากกว่าเนื่องจากได้กำจัดผลลัพธ์ที่ไม่เกี่ยวข้องออกไปแล้ว โดยคำร้องขอประเภทนี้สามารถแบ่งประเภทได้อีก 2 ประเภท คือ คำร้องขอที่ขึ้นกับชนิดข้อมูล (Type-Based Query) และคำร้องขอที่ขึ้นกับพาเรนท์ (Parent-Based Query)

- คำร้องขอที่ขึ้นกับชนิดข้อมูล จะให้ผลลัพธ์ที่มีการอ้างอิงอ็อบเจกต์ที่ร้องขอและบริบทแบบชนิดข้อมูล ตัวอย่างของบริบทแบบชนิดข้อมูล ได้แก่ ชนิดข้อมูลของตัวแปรที่ประกาศไว้ในคลาส (Field Declaration) ชนิดข้อมูลของตัวแปรที่ประกาศไว้ในเมท็อด (Local Variable Declaration) ชนิดข้อมูลของพารามิเตอร์ (Method Parameter) เป็นต้น
- คำร้องขอที่ขึ้นกับพาเรนท์ จะให้ผลลัพธ์ที่มีการอ้างอิงอ็อบเจกต์ที่ร้องขอและบริบทแบบพาเรนท์ ตัวอย่างของบริบทแบบพาเรนท์ ได้แก่ พาเรนท์ของคลาสหรือซุปเปอร์คลาส (Parent or Superclass) อินเตอร์เฟส (Interface) ที่อิมพลีเมนท์คลาสนั้น เป็นต้น

2.6.1.2 คำร้องขอทั่วไป (Generalized Query) จะให้ผลลัพธ์ทั้งหมดที่มีการอ้างอิงอ็อบเจกต์ที่ร้องขอ ซึ่งคำร้องขอประเภทนี้จะเหมาะกับการร้องขอที่ไม่ต้องการใช้บริบทเข้าช่วย หรือผู้ร้องขอไม่ทราบว่าจะใช้บริบทใด และผลลัพธ์ที่ได้จะครอบคลุมมากกว่า (Better-coverage) นั่นคือสามารถให้ผลลัพธ์ที่ครบถ้วนและตรงตามต้องการมากที่สุด เนื่องจากไม่มีการใช้บริบทตัดผลลัพธ์ออก

2.6.2. การจัดอันดับผลลัพธ์ในงานวิจัย XSnippet

การจัดอันดับผลลัพธ์ที่ได้หลังจากการสกัดข้อมูลโค้ด XSnippet ซึ่งวิทยานิพนธ์นี้ได้นำเสนอวิทยาการศึกษาสำนึกออกเป็น 3 ประเภทเพื่อช่วยในการจัดอันดับผลลัพธ์ ดังนี้

2.6.2.1. วิทยาการศึกษาสำนึกแบบความยาว (Length Heuristic)

วิทยาการศึกษาสำนึกแบบความยาวจะนับจำนวนบรรทัด (Line of Codes) ของผลลัพธ์แล้วนำมาเปรียบเทียบกัน หากผลลัพธ์ใดมีจำนวนบรรทัดน้อยแสดงว่ามี

ความยาวน้อย ซึ่งผลลัพธ์นั้นจะมีการจัดอันดับในลำดับที่สูง วิทยาการศึกษาสำนึกชนิดนี้เรียกอีกอย่างหนึ่งว่าวิทยาการศึกษาสำนึกแบบเส้นทางที่สั้นที่สุด (Shortest Path Heuristic)

2.6.2.2. วิทยาการศึกษาสำนึกแบบความถี่ (Frequency Heuristic)

วิทยาการศึกษาสำนึกแบบความถี่จะมีการนำไปใช้งานบ่อยที่สุด [2,3,5] โดยวิทยาการศึกษาสำนึกประเภทนี้จะนับจำนวนรูปแบบโค้ดในคลังข้อมูลโค้ดที่เหมือนกันมาให้ค่าความถี่แล้วนำมาเปรียบเทียบกัน โดยรูปแบบที่พบมากหรือมีค่าความถี่มากก็จะมีความน่าจะเป็นว่าเป็นรูปแบบที่มีผู้ใช้มาก และลำดับก็จะจัดอยู่ในอันดับที่สูงกว่ารูปแบบที่พบน้อย

2.6.2.3. วิทยาการศึกษาสำนึกแบบบริบท (Context Heuristic)

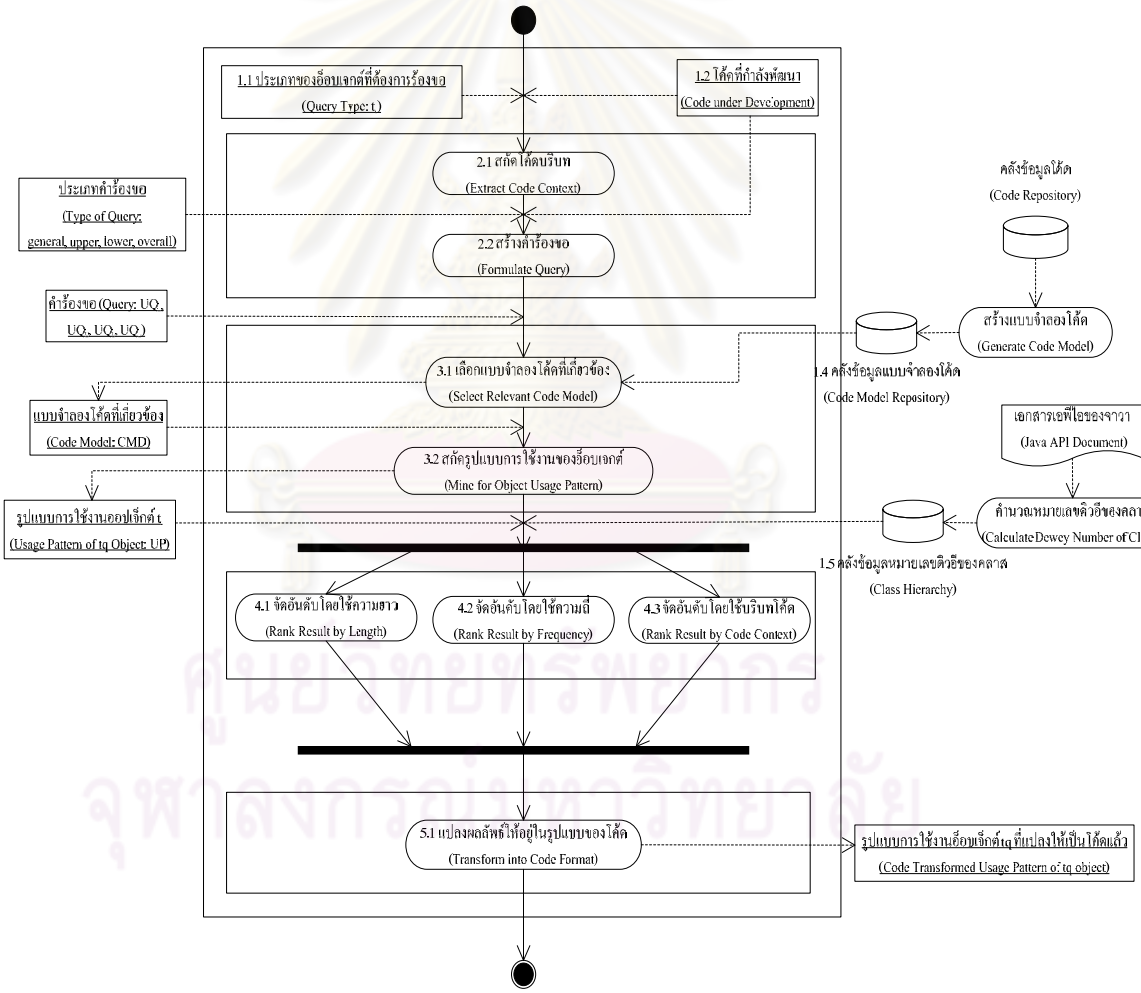
วิทยาการศึกษาสำนึกแบบบริบทจะนำบริบทโค้ดที่อยู่รอบข้างมาช่วยจัดอันดับ โดยมีสมมติฐานว่าหากผลลัพธ์มีบริบทโค้ดคล้ายคลึงกับบริบทโค้ดที่กำลังพัฒนาอยู่มาก ผลลัพธ์นั้นจะมีโอกาสเป็นคำตอบที่ถูกต้องสำหรับการร้องขอนั้นๆ ซึ่งวิทยาการศึกษา-สำนึกแบบบริบทจะคำนวณเปรียบเทียบค่าความเหมือนกัน (Context Match Value) ระหว่างกลุ่มของบริบทของอ็อบเจกต์ที่ร้องขอและกลุ่มของบริบทของรูปแบบการใช้งานของอ็อบเจกต์ หากค่าที่คำนวณได้มีค่ามาก รูปแบบการใช้งานอ็อบเจกต์นั้นก็จะอยู่ในอันดับสูง ค่าความเหมือนกันจะถูกคำนวณโดยมาตรวัดความเหมือนกันของบริบท (Context Match Measure) ซึ่งเป็นมาตรวัดความเหมือนกันระหว่างบริบทโค้ดในบริเวณอ็อบเจกต์ที่ต้องการและบริบทของรูปแบบการใช้งานที่ได้จากระบบ โดยที่ผลลัพธ์ที่มีค่าความเหมือนกันสูง จะถูกจัดให้อยู่ในอันดับที่สูง รายละเอียดของมาตรวัดความเหมือนกันของบริบทแสดงในบทที่ 2 หัวข้อมาตรวัดความเหมือนกันของบริบท

การจัดอันดับผลลัพธ์ในวิทยานิพนธ์นี้จะใช้ทั้งสามวิทยาการสำนึก โดยจะให้ความสำคัญของวิทยาการสำนึกแบบบริบท > ความถี่ > ความยาว ผลลัพธ์จะจัดเรียงด้วยค่าความเหมือนกันของบริบทก่อน หากผลลัพธ์มีค่าเท่ากันจึงใช้ความถี่เรียงลำดับ และหากผลลัพธ์มีค่าความถี่เท่ากันอีก ก็จะใช้ความยาวเรียงลำดับเป็นอันดับสุดท้าย

อย่างไรก็ตามตัวอย่างการสร้างอ็อบเจกต์จะมีประโยชน์กับการพัฒนาซอฟต์แวร์ที่ใช้ไลบรารีขนาดใหญ่ ซึ่งจะมีการสร้างอ็อบเจกต์ที่ซับซ้อน และอาจมีหลากหลายรูปแบบเท่านั้น ในขณะที่รูปแบบการใช้งานอ็อบเจกต์สามารถเกิดขึ้นได้กับไลบรารีทั้งขนาดเล็กและใหญ่ ดังนั้นวิทยานิพนธ์นี้ได้จึงนำเสนอวิธีการสกัดรูปแบบการใช้งานอ็อบเจกต์ และออกแบบประเภทของการเรียกดูผลลัพธ์ซึ่งจะขึ้นกับบริบทโค้ดในขณะที่กำลังพัฒนาซอฟต์แวร์ และนำวิทยาการศึกษาสำนึกใน [4] มาช่วยเรียงอันดับผลลัพธ์ ซึ่งทำให้นักพัฒนาซอฟต์แวร์สามารถนำโค้ดตัวอย่างไปใช้งานได้อย่างมีประสิทธิภาพ และตรงตามความต้องการมากที่สุด

บทที่ 3
วิธีออกแบบและพัฒนาการสกัดรูปแบบการใช้งานอ็อบเจกต์

วิทยานิพนธ์นี้นำเสนอวิธีการสกัดรูปแบบการใช้งานของอ็อบเจกต์ ซึ่งเป็นส่วนที่พัฒนาเพิ่มเติมจากงานวิจัย [4] ที่สามารถค้นหาโค้ดตัวอย่างรูปแบบการสร้างอ็อบเจกต์ได้เท่านั้น รูปที่ 3.1 แสดงภาพรวมของวิทยานิพนธ์ซึ่งประกอบไปด้วย 4 ส่วนหลัก ได้แก่ การสร้างคำร้องขอ (Query Formulation) การสกัดรูปแบบการใช้งาน (Usage Pattern Extraction) การจัดอันดับรูปแบบการใช้งาน (Usage Pattern Ranking) และการแปลงผลลัพธ์ให้อยู่ในรูปแบบของโค้ด (Code Format Transformation)



รูปที่ 3.1 ภาพรวมของวิทยานิพนธ์

1. ข้อมูลนำเข้าของระบบ

1.1. ประเภทของอ็อบเจกต์ที่ต้องการร้องขอ (Query Type:t_q)

ประเภทของอ็อบเจกต์ที่ต้องการร้องขอ t_q คือประเภทของอ็อบเจกต์ที่ต้องการหารูปแบบการใช้งาน เช่น ในรูปที่ 1.1 เป็นรูปแบบการใช้งานต่างๆ ของประเภทอ็อบเจกต์ t_q = ASTRewrite

1.2. โค้ดที่กำลังพัฒนา (Code under Development)

โค้ดที่กำลังพัฒนาคือโค้ดส่วนต่างๆที่นักพัฒนาซอฟต์แวร์กำลังพัฒนาซอฟต์แวร์ ณ ขณะที่ร้องขอ ในรูปที่ 3.2 แสดงโค้ดที่นักพัฒนาซอฟต์แวร์กำลังพัฒนาซอฟต์แวร์ในขณะที่เรียกใช้ระบบโดยมี t_q = ASTRewrite โค้ดที่กำลังพัฒนานั้นจะนำมาวิเคราะห์ในขั้นตอนที่ 2.1 การสกัดบริบทโค้ด เพื่อสกัดบริบทโค้ดและรูปแบบการใช้งาน และสามารถนำไปใช้ในการคัดเลือกแบบจำลองโค้ด

```
public class MakeIMemberPrivateAction extends AbstractChangeMemberModifierAction
{
   private ASTParser parser;
   public void performAction(IBuffer buffer)
   {
      String content = buffer.getContents();
      IDocument doc = new Document(content);
      ASTRewrite rewrite = ASTRewrite.create(null);

      ……..current development……..
      buffer.replace(0, buffer.getLength(), doc.get());
   }
}
```

รูปที่ 3.2 โค้ดที่กำลังพัฒนาขณะเรียกใช้ระบบ โดยที่ t_q = ASTRewrite

1.3. ประเภทของคำร้องขอ (Type of Queries)

ในวิทยานิพนธ์นี้ได้แบ่งประเภทของคำร้องขอไว้ทั้งหมด 4 ประเภทเพื่อเพิ่มความยืดหยุ่นให้กับผู้ใช้ในการเลือกคำร้องขอ ประเภทของคำร้องขอแบ่งออกเป็น 2 ประเภทหลัก ได้แก่ การร้องขอแบบทั่วไป (Generalized Query) และการร้องขอแบบพิเศษ (Specialized Query)

- การร้องขอแบบทั่วไปจะไม่คำนึงถึงบริบทโค้ด การคัดเลือกแบบจำลองโค้ดในขั้นตอน 2.1 เลือกแบบจำลองโค้ดที่เกี่ยวข้องจะไม่มีการกรองแบบจำลองโค้ดออก แต่จะเลือกทุกแบบจำลองที่มีส่วนเกี่ยวข้องกับ t_q
- การร้องขอแบบพิเศษจะนำบริบทโค้ดเข้าไปช่วยในกระบวนการทำงานของระบบด้วย การร้องขอแบบพิเศษสามารถแบ่งออกได้อีก 3 ประเภท ขึ้นกับตำแหน่งของ t_q เทียบกับบริบทโค้ด ได้แก่ การร้องขอโดยใช้บริบทด้านบน (Upper Context) บริบทด้านล่าง (Lower Context) และบริบททั้งหมด (Overall Context)

รายละเอียดของแต่ละประเภทของคำร้องขอ จะอธิบายโดยละเอียดอีกครั้งในขั้นตอน 2.1 สร้างคำร้องขอ

1.4. คลังข้อมูลแบบจำลองโค้ด (Code Model Repository)

การเตรียมคลังข้อมูลโค้ดจะใช้วิธีดาวน์โหลดซอร์สโค้ดจากแหล่งโอเพนซอร์สทั่วไปและจากโค้ดเสิร์ชเอนจิน จากนั้นจึงแปลงให้อยู่ในรูปของไฟล์เอ็กซ์เอ็มแอล แต่ละเอ็กซ์เอ็มแอลไฟล์จะแทนไฟล์ซอร์สโค้ดหนึ่งไฟล์ ซึ่งรูปแบบของเอ็กซ์เอ็มแอลจะเป็นไปตามเอ็กซ์เอ็มแอลสกีมา (XML Schema) ดังรูปที่ 3.3 โดยมีรายละเอียดดังตารางที่ 3.1 และรูปที่ 3.4 แสดงการแปลงซอร์สโค้ดให้กลายเป็นแบบจำลองโค้ดโดยใช้โครงสร้างของเอ็กซ์เอ็มแอลสกีมาที่ออกแบบไว้

ในคลังข้อมูลแบบจำลองโค้ดแปลงมาจากคลาสในภาษาจาวารวมทั้งสิ้นกว่า 9,000 คลาส โดยที่คลาสเหล่านี้มาจากไลบรารีจาวามาตรฐานประมาณ 3,000 ไฟล์ ไลบรารีของอีคลิปปลั๊กอินประมาณ 5,000 ไฟล์ และไลบรารีอื่นๆ เช่น lowagie หรือ apache ประมาณ 1,000 ไฟล์ ซึ่งตัวอย่างไฟล์เอ็กซ์เอ็มแอลของซอร์สโค้ดแสดงดังรูปที่ 3.5

1.5. คลังข้อมูลของจาวาชนิดข้อมูล (Java Type Repository)

หมายเลขดิวอีสามารถคำนวนได้ (Dewey Number) ดังที่ได้อธิบายแล้วในบทที่ 2 ในหัวข้ออัลกอริทึมจับคู่ชนิดข้อมูล เพื่อนำไปใช้ในขั้นตอน 3.1 จัดอันดับโดยใช้บริบทโค้ด

ในบทที่ 1 รูปที่ 1.1 สามารถคำนวณหมายเลขดิวอีของชนิดข้อมูลได้ดังรูปที่ 3.6 - 3.10 และสามารถสรุปหมายเลขดิวอีได้ดังตารางที่ 3.2

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xsd:element name="classModel" type="ClassModel"/>

  <xsd:complexType name="ClassModel">
    <xsd:sequence>
      <xsd:element name="package" type="xsd:string" minOccurs="0"/>
      <xsd:element name="className" type="xsd:string"/>
      <xsd:element name="superclass" type="xsd:string"/>
      <xsd:element name="qualifiedName" type="xsd:string" minOccurs="0" maxOccurs="1" />
      <xsd:element ref="interfaces" minOccurs="0"/>
      <xsd:element ref="declaredMethods" minOccurs="0"/>
      <xsd:element name="plugin" type="xsd:string" minOccurs="0"/>
      <xsd:element ref="declaredFields" minOccurs="0"/>
      <xsd:element ref="declaredClasses" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="return" type="xsd:string"/>

  <xsd:element name="parameters">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="parameter" type="NodeElement" minOccurs="0"
          maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

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

รูปที่ 3.3 เอ็กซ์เอ็มแอลสกีมาสำหรับจัดเก็บซอร์สโค้ด

```
<xsd:element name="method">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="parameters" minOccurs="0"/>
      <xsd:element ref="return"    minOccurs="0"/>
      <xsd:element ref="behaviors" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="modifier" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="interfaces">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="name" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="edge">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="parameters" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="type" type="xsd:string"/>
    <xsd:attribute name="localNumber" type="xsd:string"/>
    <xsd:attribute name="label" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

รูปที่ 3.3 เอ็กซ์เอ็มแอลสกีมาสำหรับจัดเก็บซอร์สโค้ด (ต่อ)

```
<xsd:element name="declaredMethods">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="method" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="declaredFields">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="field" minOccurs="0"maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="field">
  <xsd:complexType>
    <xsd:attribute name="classType" type="xsd:string"/>
    <xsd:attribute name="objectName" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="behaviors">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="behavior" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

รูปที่ 3.3 เอ็กซ์เอ็มแอลสกีมาสำหรับจัดเก็บซอร์สโค้ด (ต่อ)

```
  <xsd:element name="behavior">
    <xsd:complexType>
      <xsd:sequence minOccurs="0">
        <xsd:element name="inputNode" minOccurs="0" maxOccurs="1"
          type="NodeElement"/>
        <xsd:element ref="edge" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="outputNode" minOccurs="0" maxOccurs="1"
          type="NodeElement"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:complexType name="NodeElement">
    <xsd:attribute name="objectName" type="xsd:string"/>
    <xsd:attribute name="caseid" type="xsd:string"/>
    <xsd:attribute name="nodeType" type="xsd:string"/>
    <xsd:attribute name="classType" type="xsd:string"/>
    <xsd:attribute name="value" type="xsd:string"/>
  </xsd:complexType>

  <xsd:element name="declaredClasses">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="classModel" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

รูปที่ 3.3 เอ็กซ์เอ็มแอลสกีมาสำหรับจัดเก็บซอร์สโค้ด (ต่อ)

ตารางที่ 3.1 คำอธิบายอิลิเมนท์ในเอ็กซ์เอ็มแอลสกีมาสำหรับจัดเก็บซอร์สโค้ด

ลำดับที่	เอกซ์เอ็มแอลอิลิเมนท์	คำอธิบาย
1	classModel	คลาสในภาษาจาวาหนึ่งคลาส โดยอิลิเมนท์นี้จะเป็นอิลิเมนท์หลักที่จะอ้างอิงถึงอิลิเมนท์ทุกตัวในเอกซ์เอ็มแอลสกีมา
2	package	แพ็กเกจหรือโฟลเดอร์ที่เก็บ classModel
3	plugin	ปลั๊กอินที่ classModel ขยายมา
4	className	ชื่อคลาสของ classModel
	qualifiedName	ชื่อเต็มของ className (ชื่อแพ็กเกจและคลาสรวมกัน)
5	superclass	ชื่อซุปเปอร์คลาสที่ classModel สืบทอด สามารถเทียบได้กับ superclass ตามมาตรฐานจาวา
6	interfaces	ชื่อของอินเตอร์เฟสที่ classModel อิมพลีเมนท์
7	declaredClasses	กลุ่มของอินเนอร์คลาส (Inner Class) ภายใน classModel
8	declaredFields	กลุ่มแอททริบิวท์ของคลาส
9	declaredMethods	กลุ่มเมท็อดของคลาส
10	field	แอททริบิวท์ของคลาส สามารถเทียบได้กับ attribute หรือ field ตามมาตรฐานจาวา ซึ่งเอกซ์เอ็มแอลอิลิเมนท์ field มีสามารถมีแอททริบิวต์ ดังนี้ - classType หมายถึง ชนิดข้อมูลของแอททริบิวต์ - objectName หมายถึง ชื่อออบเจ็กต์
11	return	ชนิดข้อมูลที่เมท็อดรีเทิร์นผลลัพธ์ออกมา
12	name	ชื่อของอิลิเมนท์ชนิดใดก็ได้ แล้วแต่ว่าอิลิเมนท์ใดจะอ้างอิงมา

ตารางที่ 3.1 คำอธิบายอิลิเมนท์ในเอ็กซ์เอ็มแอลสกีมาสำหรับจัดเก็บซอร์สโค้ด (ต่อ)

ลำดับที่	เอกซ์เอ็มแอลอิลิเมนท์	คำอธิบาย
13	method	เมท็อดของคลาส สามารถเทียบได้กับ method ตามมาตรฐานจาวา ซึ่งเอกซ์เอ็มแอลอิลิเมนท์ method มีแอททริบิวท์ดังนี้ - name หมายถึง ชื่อเมท็อด - modifier หมายถึง โมดิฟายเออร์ของคลาส แบ่งออกเป็น 12 ประเภทตามมาตรฐานจาวา และมีค่าคงที่ในแต่ละประเภทดังนี้ ABSTRACT 1024 FINAL 16 INTERFACE 512 NATIVE 256 PRIVATE 2 PROTECTED 4 PUBLIC 1 STATIC 8 STRICT 2048 SYNCHRONIZED 32 TRANSIENT 128 VOLATILE 64

ตารางที่ 3.1 คำอธิบายอิลิเมนท์ในเอ็กซ์เอ็มแอลสกีมาสำหรับจัดเก็บซอร์สโค้ด (ต่อ)

ลำดับที่	เอกซ์เอ็มแอลอิลิเมนท์	คำอธิบาย
14	nodeElement	อ็อบเจกต์ที่ใช้งานภายในเมท็อด สามารถเป็นได้ทั้งโหนดอินพุทและโหนดเอาท์พุท
15	behaviors	กลุ่มของ behavior
14	behavior	คำสั่งภายใน method สามารถเปรียบเทียบได้กับ statement ในมาตรฐานจาวา ยกตัวอย่างเช่น ในรูปที่ 1.1 ทั้งคำสั่ง TextEdit te = rewrite.rewriteAST(id, null); จะคือ behavior
15	inputNode	อ็อบเจกต์ที่เรียกใช้ behavior เปรียบเสมือนข้อมูลเข้าของ behavior ซึ่งชนิดข้อมูลของ inputNode คือ NodeElement ยกตัวอย่างเช่น คำสั่งในรูปที่ 1.1 TextEdit te = rewrite.rewriteAST(id, null); ในที่นี้ inputNode คือ rewrite
16	outputNode	อ็อบเจกต์ที่รีเทิร์นมาจาก behavior เปรียบเสมือนข้อมูลออกจาก behavior ซึ่งชนิดข้อมูลของ outputNode คือ NodeElement ยกตัวอย่างเช่น คำสั่งในรูปที่ 1.1 TextEdit te = rewrite.rewriteAST(id, null); ในที่นี้ outputNode คือ te
17	parameter	พารามิเตอร์ของเมท็อด สามารถเทียบได้กับ parameter ตามมาตรฐานจาวา ยกตัวอย่างเช่น คำสั่งในรูปที่ 1.1 TextEdit te = rewrite.rewriteAST(id, null); ในที่นี้ parameter คือ id และ null
18	parameters	กลุ่มของ parameter

ตารางที่ 3.1 คำอธิบายอิลิเมนท์ในเอ็กซ์เอ็มแอลสกีมาสำหรับจัดเก็บซอร์สโค้ด (ต่อ)

ลำดับที่	เอกซ์เอ็มแอลอิลิเมนท์	คำอธิบาย
19	edge	ตัวเชื่อมต่อระหว่าง inputNode และ outputNode หรืออีกนัยหนึ่งก็คือคำสั่งที่ inputNode เรียกใช้ยกตัวอย่างเช่น คำสั่งในรูปที่ 1.1 TextEdit te = rewrite.rewriteAST(id, null); ในที่นี้ edge คือ rewriteAST(id, null) ซึ่งเอกซ์เอ็มแอลอิลิเมนท์ Edge มีแอททริบิวท์ดังนี้ - type หมายถึง ชนิดของเอดจ์ - localNumber หมายถึง ลำดับของเอดจ์ที่เกิดขึ้นภายใน method - label หมายถึง ชื่อของเอดจ์

JAVA CODE

```
import org.eclipse.jdt.core.dom.rewrite.ASTRewrite;
import org.eclipse.jface.text.Document;
...
public class MakeIMemberPrivateAction {
  private IMember im;
  protected boolean performAction()
  {
    ICompilationUnit icu = im.getCompilationUnit();
    IBuffer bf = icu.getBuffer();
    String content = bf.getContents();
    int i = bf.getLength();
    IDocument id = new Document(content);

    String s = id.get();
    ASTRewrite astr = ASTRewrite.create(null);

                ////Current Development////

    bf.replace(0, i, s);
    return true;
  }
}
```

+

XML SCHEMA

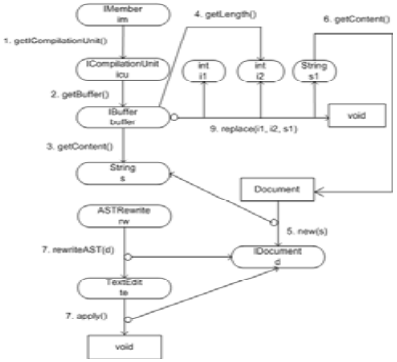
```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
    <xsd:element name="classModel" type="ClassModel"/>
    <xsd:complexType name="ClassModel">
        <xsd:sequence>
            <xsd:element name="package" type="xsd:string" minOccurs="0"/>
            <xsd:element name="className" type="xsd:string"/>
            <xsd:element name="superclass" type="xsd:string"/>
            <xsd:element name="qualifiedName" type="xsd:string" minOccurs="0" maxOccurs="1" />
            <xsd:element ref="interfaces" minOccurs="0"/>
            <xsd:element ref="declaredMethods" minOccurs="0"/>
            <xsd:element name="plugin" type="xsd:string" minOccurs="0"/>
            <xsd:element ref="declaredFields" minOccurs="0"/>
            <xsd:element ref="declaredClasses" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="return" type="xsd:string"/>
    <xsd:element name="parameters">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="parameter" type="NodeElement" minOccurs="0"
                    maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
...
<xsd:schema>
```

=

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<classModel>
    <package>com.ibm.jdg2e.jdt.extras2</package>
    <className>MakeIMemberPrivateAction</className>
    <superclass>com.ibm.jdg2e.jdt.extras2.AbstractChangeMemberModifierAction</superclass>
    <declaredMethods>
        <method name="isChecked" modifier="4">
            <behaviors>
                <behavior>
                    <inputNode objectName="member" nodeType="object"

                        classType="org.eclipse.jdt.core.IMember"/>
                    <edge type="method" localNumber="1" label="getFlags">
                    <parameters>
                    <parameter objectName="member" nodeType="object"
                            classType="org.eclipse.jdt.core.IMember"/>
                        <parameter objectName="AccPublic" nodeType="object" classType="int"/>
                        <parameter objectName="TokenNamepublic" nodeType="object"
                            classType="int"/>
                    </parameters>
                    </edge>
                    <outputNode objectName="flags" nodeType="object" classType="int"/>
                </behavior>
            </behaviors>
        </method>
...
</xml>
```

Code Model in
XML Representation

=



Code Model in
Graphical Representation

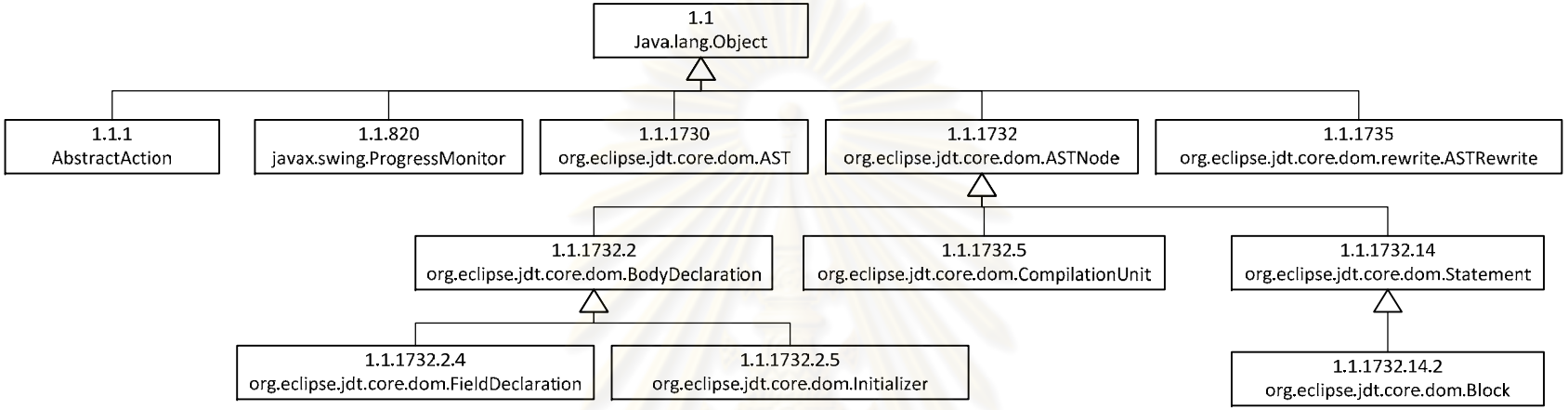
รูปที่ 3.4 แสดงการแปลงซอร์สโค้ดให้กลายเป็นแบบจำลองโค้ด

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<classModel>
  <package>com.ibm.jdg2e.jdt.extras2</package>
  <className>MakeIMemberPrivateAction</className>
  <superclass>com.ibm.jdg2e.jdt.extras2.AbstractChangeMemberModifierAction</superclass>
  <declaredMethods>
    <method name="isChecked" modifier="4">
      <behaviors>
        <behavior>
          <inputNode objectName="member" nodeType="object"
            classType="org.eclipse.jdt.core.IMember"/>
          <edge type="method" localNumber="1" label="getFlags">
            <parameters>
              <parameter objectName="member" nodeType="object"
                classType="org.eclipse.jdt.core.IMember"/>
              <parameter objectName="AccPublic" nodeType="object" classType="int"/>
              <parameter objectName="TokenNamepublic" nodeType="object"
                classType="int"/>
            </parameters>
          </edge>
          <outputNode objectName="flags" nodeType="object" classType="int"/>
        </behavior>
      </behaviors>
    </method>
    <method name="canPerformAction" modifier="4">
      <behaviors>
        <behavior>
          <edge type="case" localNumber="case1.c1-start" label="caseEdge">
            <parameters/>
          </edge>
        </behavior>
```

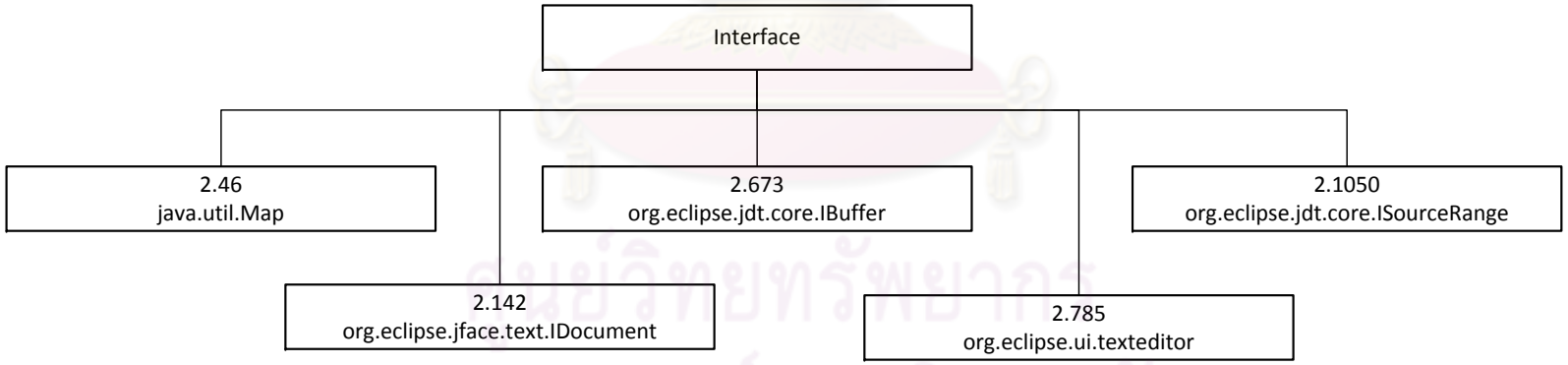
รูปที่ 3.5 ตัวอย่างไฟล์เอ็กซ์เอ็มแอลที่จัดเก็บซอร์สโค้ด

```
            <behavior>
              <inputNode objectName="member" nodeType="object"
                classType="org.eclipse.jdt.core.IMember"/>
              <edge type="method" localNumber="1" label="getDeclaringType">
                <parameters/>
              </edge>
              <outputNode objectName="type" nodeType="object"
                classType="org.eclipse.jdt.core.IType"/>
            </behavior>
            <behavior>
              <edge type="case" localNumber="case1.c1-end" label="caseEdge">
                <parameters/>
              </edge>
            </behavior>
          </behaviors>
        </method>
    </declaredMethods>
    <declaredClasses>
      <classModel/>
    </declaredClasses>
</classModel>
```

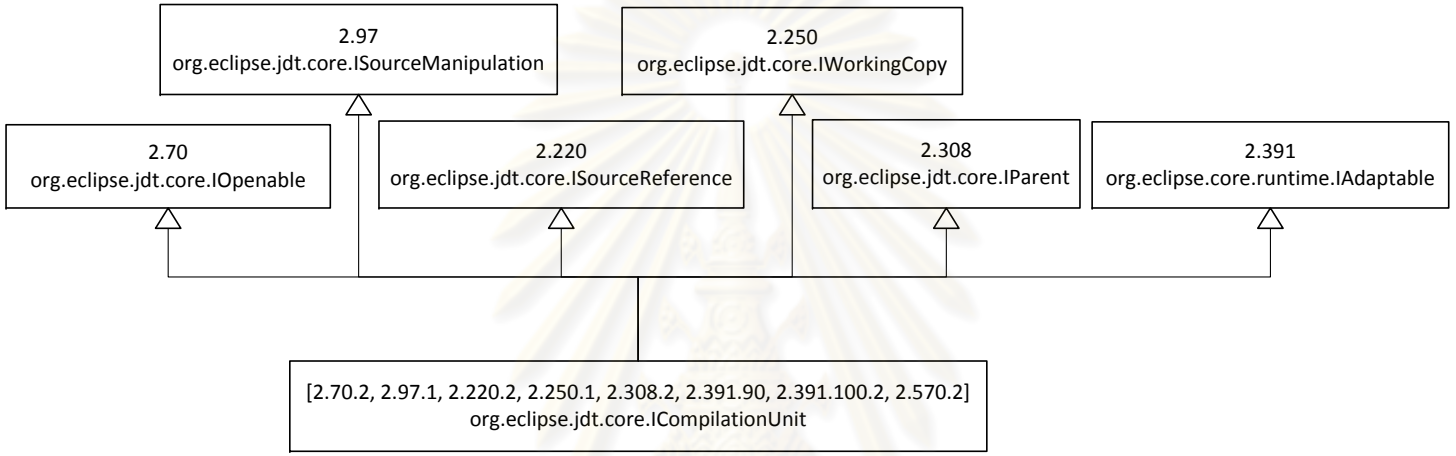
รูปที่ 3.5 ตัวอย่างไฟล์เอ็กซ์เอ็มแอลที่จัดเก็บซอร์สโค้ด (ต่อ)



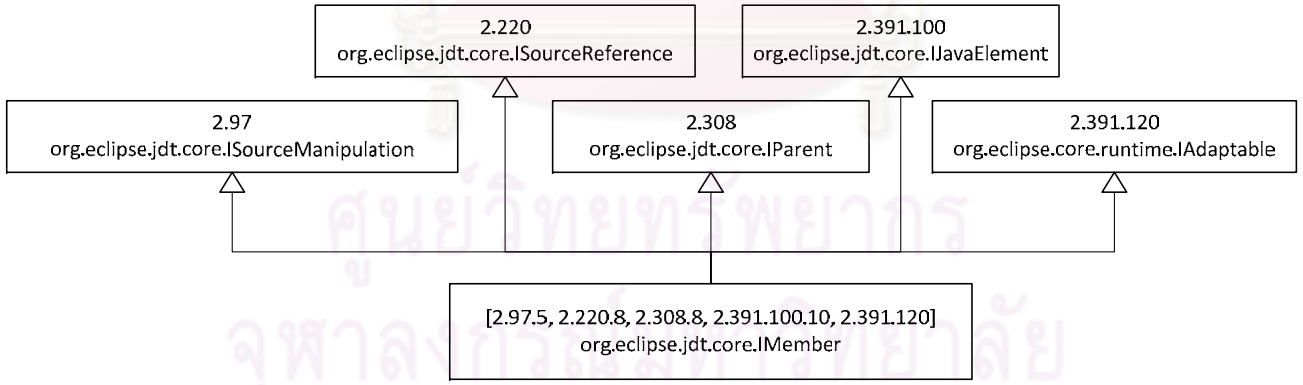
รูปที่ 3.6 หมายเลขดิวอีของชนิดข้อมูลในรูปที่ 1.1



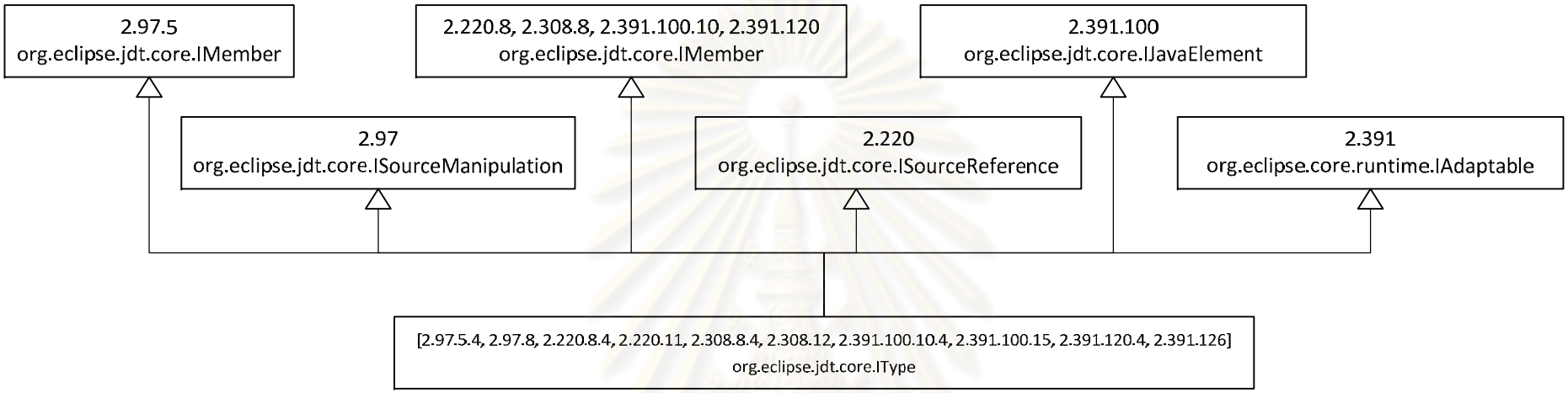
รูปที่ 3.7 หมายเลขดิวอีของอินเตอร์เฟซในรูปที่ 1.1



รูปที่ 3.8 หมายเลขดิวอีของชนิดข้อมูล ICompilationUnit



รูปที่ 3.9 หมายเลขดิวอีของชนิดข้อมูล IMember



รูปที่ 3.10 หมายเลขดิวอีของชนิดข้อมูล IType

ตารางที่ 3.2 ตารางสรุปหมายเลขดิวอีของชนิดข้อมูลในรูปที่ 1.1

ชนิดข้อมูล	หมายเลขดิวอี
org.eclipse.jdt.core.dom.FieldDeclaration	1.1.1732.2.4
org.eclipse.jdt.core.dom.AST	1.1.1730
org.eclipse.jdt.core.ICompilationUnit	[2.70.2, 2.97.1, 2.220.2, 2.250.1, 2.308.2, 2.391.90, 2.391.100.2, 2.570.2]
java.util.Map	2.46
org.eclipse.jdt.core.dom.Initializer	1.1.1732.2.5
org.eclipse.jdt.core.dom.CompilationUnit	1.1.1732.5

ตารางที่ 3.2 ตารางสรุปหมายเลขดิวอีของชนิดข้อมูลในรูปที่ 1.1 (ต่อ)

ชนิดข้อมูล	หมายเลขดิวอี
javax.swing.ProgressMonitor	[1.1.820, 2.1219.2]
org.eclipse.jdt.core.dom.rewrite.ListRewrite	1.1.1781
org.eclipse.jdt.core.dom.Block	1.1.1732.14.2
org.eclipse.jdt.core.dom.rewrite.ASTRewrite	1.1.1735
org.eclipse.ui.texteditor	2.785
org.eclipse.jface.text.IDocument	2.142
org.eclipse.jdt.core.IBuffer	2.673
org.eclipse.jdt.core.IType	[2.97.5.4, 2.97.8, 2.220.8.4, 2.220.11, 2.308.8.4, 2.308.12, 2.391.100.10.4, 2.391.100.15, 2.391.120.4, 2.391.126]
org.eclipse.jdt.core.ISourceRange	2.1050
org.eclipse.jdt.core.IMember	[2.97.5, 2.220.8, 2.308.8, 2.391.100.10, 2.391.120]

2. การสร้างคำร้องขอ (Query Formulation)

ในขั้นตอนการสร้างคำร้องขอ จะรวบรวมข้อมูลเข้าทั้งหมดเพื่อสร้างเป็นคำร้องขอและส่งไปยังขั้นตอน 3 เพื่อสกัดรูปแบบการใช้งานต่อไป การสร้างคำร้องขอสามารถแบ่งออกเป็น 2 กระบวนการได้ดังนี้

2.1. สกัดบริบทโค้ด (Extract Code Context)

บริบทโค้ดเป็นแหล่งข้อมูลที่สำคัญที่จะช่วยให้ระบบสามารถค้นหารูปแบบการใช้งานที่ตรงกับความต้องการได้มากขึ้น หากผลลัพธ์ใดตรงกับบริบทโค้ดมาก ผลลัพธ์นั้นจะมีโอกาสเป็นคำตอบที่ต้องการมากขึ้น ในบทที่ 1 รูปที่ 1.1 ถ้าบริบทโค้ดมีการเรียกใช้เมท็อด replace() ของ IBuffer อ็อบเจกต์ รูปแบบการใช้งาน C, D อาจเป็นผลลัพธ์ที่เกี่ยวข้องกับความต้องการมากที่สุด เนื่องจากมีการเรียกใช้เมท็อด replace() ของอ็อบเจกต์ IBuffer เหมือนกัน

จาวาคลาส (Java Class) ประกอบไปด้วย

(1) พาเรนท์ ซุปเปอร์คลาส อินเตอร์เฟสที่คลาสนั้นสืบทอดมา

(2) แอททริบิวต์ ของคลาสหรือตัวแปรที่ประกาศไว้ภายในเมท็อด

(3) เมท็อด ซึ่งเป็นพฤติกรรมของคลาส

วิทยานิพนธ์นี้ใช้โครงสร้างพื้นฐานของจาวาคลาสมานิยามประเภทต่างๆ ของบริบทโค้ดสำหรับเมท็อด m ในคลาส C หนึ่งๆ จะสามารถนิยามประเภทของบริบทโค้ดได้เป็นบริบทโค้ดแบบพาเรนท์ (Parent Context: CP(m)) บริบทโค้ดแบบชนิดข้อมูล (Type Context: CT(m)) และบริบทโค้ดแบบการเรียกใช้เมท็อด (Method Invocation Context: CM(m)) บริบทโค้ดแบบพาเรนท์และแบบชนิดข้อมูลได้นิยามไว้แล้วใน [4]

<u>บริบทโค้ดแบบการเรียกใช้เมท็อด</u>

การเรียกใช้เมท็อดขณะอยู่ในเมท็อด m หรือ CM(m) คือกลุ่มของการเรียกใช้เมท็อดที่เรียกใช้ในเมท็อด m แต่ละการเรียกใช้เมท็อด cm สามารถนิยามได้ดังนี้

(cl; l; r; p)

เมื่อ

cl = คลาสที่เรียกใช้เมท็อด cm

l = ชื่อเมท็อด cm

r = รีเทิร์นชนิดข้อมูล (Return Type) ของเมท็อด cm

p = กลุ่มของพารามิเตอร์ของเมท็อด cm

บริบทโค้ดแบบการเรียกใช้เมท็อดแบ่งประเภทโดยใช้ตำแหน่งประเภทอ็อบเจกต์ t_q เทียบกับตำแหน่งบริบทโค้ด สามารถแบ่งออกเป็น 2 ประเภท ได้แก่

- บริบทโค้ดแบบบน $CM_U(m)$ คือบริบทโค้ดตั้งแต่ต้นเมท็อด m จนถึงบริบทโค้ดเหนือตำแหน่งประเภทอ็อบเจกต์ t_q
- บริบทโค้ดแบบล่าง $CM_L(m)$ คือบริบทโค้ดตั้งแต่ตำแหน่งประเภทอ็อบเจกต์ ลงไปจนสิ้นสุดเมท็อด m

โดย $CM = CM_U(m) \cup CM_L(m)$ ยกตัวอย่างเช่น ตารางที่ 3.3 แสดงการเรียกใช้เมท็อดของเมท็อด m = performAction() ในคลาส MakeIMemberPrivateAction ที่แสดงในรูปที่ 3.2 โดย t_q = ASTRewrite

ตารางที่ 3.3 บริบทชนิดต่างๆ ในรูปที่ 3.2

ประเภทของบริบท	บริบท
Parent Context (CP(m))	AbstractChangeMemberModifierAction
Type Context (CT (m))	ASTParser, IBuffer, String, IDocument, Document, int
Upper Method Context (CMU(m))	(IBuffer, getContents(), String, {}) (IDocument, new(), Document, {String})
Lower Method Context (CML(m))	(IBuffered, replace(), void, {int, int, String})

ระบบจะสกัดบริบททุกประเภท แต่การสกัดบริบทแบบการเรียกใช้เมท็อดจะขึ้นอยู่กับประเภทของคำร้องขอที่ผู้ใช้เลือก เช่น หากคำร้องขอที่เลือกคือบริบทแบบบน การสกัดบริบทโค้ดจะเลือกเฉพาะบริบทแบบบนเท่านั้น ในทางเดียวกัน หากคำร้องขอที่เลือกคือบริบททั้งหมด การสกัดบริบทโค้ดก็จะเลือกทั้งบริบทด้านบนและบริบทด้านล่าง เป็นต้น

2.2. สร้างคำร้องขอ (Formulate Query)

การสร้างคำร้องขอจะใช้ข้อมูลเข้าทั้งหมด 3 ประเภท ได้แก่ ประเภทของอ็อบเจกต์ที่ต้องการร้องขอ t_q ประเภทของคำร้องขอที่ผู้ใช้เลือก และบริบทโค้ดที่สกัดได้ตามหัวข้อ 2.1 คำร้องขอจะแบ่งประเภทจากคำร้องขอทั่วไป (Generalize Usage Pattern Query: UQ_G) ไปยังคำร้องขอแบบพิเศษ โดยคำร้องขอแบบพิเศษสามารถแบ่งประเภทได้อีก 3 ประเภท ได้แก่ คำร้องขอที่ใช้บริบททั้งหมด (Overall Usage Pattern Query : UQ_A) คำร้องขอที่ใช้เฉพาะบริบทด้านบน

(Upper Usage Pattern Query : $\mathrm{UQ_U}$) และคำร้องขอที่ใช้เฉพาะบริบทด้านล่าง (Lower Usage Pattern Query : $\mathrm{UQ_L}$)

2.2.1. คำร้องขอทั่วไป ($\mathrm{UQ_G}$) ข้อมูลเข้าของคำร้องขอ คือ ประเภทของอ็อบเจกต์ที่ต้องการ ($\mathrm{t_q}$) และให้ผลลัพธ์เป็นกลุ่มของรูปแบบการใช้งานทั้งหมดในคลังข้อมูล (UP) ที่มีการอ้างอิงประเภทอ็อบเจกต์ $\mathrm{t_q}$ อยู่ภายใน คำร้องขอ ($\mathrm{UQ_G}$) สามารถเขียนอยู่ในรูปสัญลักษณ์ทางคณิตศาสตร์ได้ดังนี้

$$\mathrm{UQ_G(t_q)} = \forall \mathrm{up} \in \mathrm{UP}\text{: up contains in the usage pattern of } \mathrm{t_q}$$

คำร้องขอ $\mathrm{UQ_G}$ จะเป็นประโยชน์เมื่อ

(1) รูปแบบการใช้งานของประเภทอ็อบเจกต์ $\mathrm{t_q}$ ไม่เกี่ยวข้องกับบริบทโค้ด

(2) ไม่มีรูปแบบการใช้งานที่ตรงกับบริบทโค้ด ใน หาก $\mathrm{UQ_G}$ ($\mathrm{t_q}$ = ASTRewrite) คำร้องขอนี้จะให้ผลลัพธ์ทุกรูปแบบตั้งแต่ A-D ในบทที่ 1 รูปที่ 1.1

2.2.2. คำร้องขอที่ใช้เฉพาะบริบทด้านบน ($\mathrm{UQ_U}$) ข้อมูลเข้าของคำร้องขอคือ ประเภทของอ็อบเจกต์ที่ต้องการ $\mathrm{t_q}$ และบริบทโค้ดด้านบน $\mathrm{CM_U}$ ผลลัพธ์ของการร้องขอคือกลุ่มของรูปแบบการใช้งานในคลังข้อมูล (UP) โดยที่ $\mathrm{up} \in \mathrm{UP}$ มีการอ้างอิงถึงการเรียกใช้เมท็อด $\mathrm{cm} \in \mathrm{CM_U(m)}$ คำร้องขอ $\mathrm{UP_U}$ สามารถเขียนเป็นสัญลักษณ์ทางคณิตศาสตร์ได้ดังนี้

$$\mathrm{UQ_U}\left(\mathrm{t_q}, \mathrm{CM_U(m)}\right) = \exists \mathrm{up} \in \mathrm{UQ_G}\text{: } \mathrm{MI(up)} \cap \mathrm{CM_U(m)}$$

โดยที่ MI หมายถึงการเรียกใช้เมท็อดใน up คำร้องขอ ($\mathrm{UQ_U}$) จะเป็นประโยชน์ก็ต่อเมื่อรูปแบบการใช้งาน up ของประเภทอ็อบเจกต์ $\mathrm{t_q}$ มีบริบทโค้ดอยู่ใน $\mathrm{CM_U}$ ในรูปที่ 1.1 ถ้าคำร้องขอคือ $\mathrm{UQ_U}$($\mathrm{t_q}$ = ASTRewrite, $\mathrm{CM_U}$ = (performAction()) แล้วคำร้องขอจะให้ผลลัพธ์เป็นรูปแบบ B, C

2.2.3. คำร้องขอที่ใช้เฉพาะบริบทด้านล่าง ($\mathrm{UQ_L}$) ข้อมูลเข้าของคำร้องขอคือ ประเภทของอ็อบเจกต์ที่ต้องการ $\mathrm{t_q}$ และบริบทโค้ดด้านบน $\mathrm{CM_L}$ ผลลัพธ์ของการร้องขอคือกลุ่มของรูปแบบการใช้งาน (UP) ในคลังข้อมูล โดยที่ $\mathrm{up} \in \mathrm{UP}$ มีการอ้างอิงถึงการเรียกใช้บางเมท็อด $\mathrm{cm} \in \mathrm{CM_L(m)}$ คำร้องขอ $\mathrm{UQ_L}$ สามารถเขียนเป็นสัญลักษณ์ทางคณิตศาสตร์ได้ดังนี้

$$\mathrm{UQ_L(t_q, CM_L(m))} = \exists \mathrm{up} \in \mathrm{UQ_G}\text{: } \mathrm{MI(up)} \cap \mathrm{CM_L(m)}$$

โดยที่ MI หมายถึงการเรียกใช้เมท็อดใน up คำร้องขอ UQ_L จะเป็นประโยชน์ก็ต่อเมื่อรูปแบบการใช้งาน up ของประเภทอ็อบเจกต์ t_q มีบริบทโค้ดอยู่ใน CM_L ในรูปที่ 1.1 ถ้าคำร้องขอคือ UQ_L (t_q = ASTRewrite, CM_L (performAction()) แล้วคำร้องขอจะให้ผลลัพธ์เป็นรูปแบบ C, D

คำร้องขอที่ใช้บริบททั้งหมด (UQ_A) จะรวมข้อดีของสองคำร้องขอด้านบนไว้ด้วยกัน ข้อมูลเข้าของ UQ_A ได้แก่ ประเภทของอ็อบเจกต์ที่ต้องการ t_q บริบทโค้ดด้านบน CM_U และบริบทโค้ดด้านล่าง CM_L ผลลัพธ์ของการร้องขอคือกลุ่มของรูปแบบการใช้งาน (UP) ในคลังข้อมูล โดยที่ $up \in UP$ มีการอ้างอิงถึงการเรียกใช้บางเมท็อด $cm \in CM_U(m)$ และ $cm \in CM_L(m)$ คำร้องขอ UQ_A สามารถเขียนเป็นสัญลักษณ์ทางคณิตศาสตร์ได้ดังนี้

$$UQ_A(t_q, CM_U(m), CM_L(m)) = UQ_U \cap UQ_L$$

2.2.4. คำร้องขอ UQ_A จะเป็นประโยชน์ก็ต่อเมื่อรูปแบบการใช้งาน up ของประเภทอ็อบเจกต์ t_q มีบริบทโค้ดอยู่ใน CM_U และ CM_L ในรูปที่ 1.1 หากคำร้องขอ UQ_A (t_q = ASTRewrite, CM_U (performAction()), CM_L (performAction())) แล้วคำร้องขอจะให้ผลลัพธ์เป็นรูปแบบ D

เมื่อสร้างคำร้องขอได้แล้ว คำร้องขอจะส่งต่อไปยังขั้นตอนถัดไป เพื่อสกัดรูปแบบการใช้งาน

3. **การสกัดรูปแบบการใช้งาน** (Usage Pattern Extraction)

การสกัดรูปแบบการใช้งานจะเริ่มจากการคัดเลือกแบบจำลองโค้ดจากคลังข้อมูล จากนั้นจึงส่งไปสกัดรูปแบบการใช้งานต่อ ในหัวข้อนี้สามารถแบ่งกระบวนการออกเป็น 2 กระบวนการ ดังนี้

3.1. เลือกแบบจำลองที่เกี่ยวข้อง (Select Relevant Code Model)

การเลือกแบบจำลองที่เกี่ยวข้องจะนำเอาประเภทอ็อบเจกต์ที่ต้องการ t_q และบริบทโค้ดแบบการเรียกใช้เมท็อด CM เข้าไปช่วยคัดเลือกแบบจำลองโค้ดในคลังข้อมูล การคัดเลือกจะขึ้นอยู่กับประเภทของคำร้องขอ (UQ_G, UQ_U, UQ_L, UQ_A) โดยมีรายละเอียดดังนี้

- ในคำร้องขอ UQ_G จะคัดเลือกแบบจำลองโค้ดทั้งหมดที่มีการอ้างอิงถึงอ็อบเจกต์ประเภท t_q
- ในคำร้องขอ UQ_U จะคัดเลือกแบบจำลองโค้ดที่มีการอ้างอิงถึงอ็อบเจกต์ประเภท t_q และแบบจำลองนั้นจะต้องมีการอ้างอิงถึงบริบทโค้ด CM_U ด้วย

- ในคำร้องขอ UQ_L จะคัดเลือกแบบจำลองโค้ดที่มีการอ้างอิงถึงอ็อบเจกต์ประเภท t_q และแบบจำลองนั้นจะต้องมีการอ้างอิงถึงบริบทโค้ด CM_L ด้วย
- ในคำร้องขอ UQ_A จะคัดเลือกแบบจำลองโค้ดที่มีการอ้างอิงถึงอ็อบเจกต์ประเภท t_q และแบบจำลองนั้นจะต้องมีการอ้างอิงถึงบริบทโค้ด CM_U และ CM_L ด้วย

จากนั้นแบบจำลองที่คัดเลือกก็จะส่งต่อให้กับกระบวนการถัดไป

3.2. สกัดเพื่อหารูปแบบการใช้งานของอ็อบเจกต์ (Extract for Object Usage Pattern)

เป้าหมายของการสกัดรูปแบบการใช้งานอ็อบเจกต์คือ เพื่อหารูปแบบการใช้งานของอ็อบเจกต์ที่เป็นไปตามคำร้องขอ UQ รูปที่ 3.11 แสดงอัลกอริทึมของเมท็อด usagePatternExtraction() กำหนดให้ข้อมูลนำเข้าของเมท็อด usagePatternExtraction() คือ ประเภทของอ็อบเจกต์ที่ต้องการ t_q บริบทแบบเมท็อดอินโวเคชัน CM และแบบจำลองโค้ด CMD

อัลกอริทึมของเมท็อด usagePatternExtraction() มีกระบวนการทำงานดังนี้

1) ระบุกลุ่มของโหนด $NQ = \{n_q\}$ โดยที่โหนด $n_q \in cmd_i$ และประเภทของโหนด n_q คือ t_q
2) เมท็อด usagePatternExtraction() จะเรียก เมท็อด forwardExtraction() เพื่อทำหน้าที่ท่องไปตามเส้นทางของแต่ละโหนด $n_q \in NQ$ อย่างเวียนบรรจบ (recursively) ทั้งแบบไปข้างหน้าและแบบย้อนกลับ (Forward and Backward Traversal) โดยจะท่องไปทุกๆ เอดจ์ (Edge: e) ei,...,ej ที่เกิดขึ้นบนโหนด n_q
3) การท่องเส้นทางแบบไปข้างหน้าจะจบลงเมื่อเกิดเหตุการณ์ใดเหตุการณ์หนึ่งดังนี้ (1) โหนด $n_q \in cmd_i$ ไม่มีเอดจ์เชื่อมต่ออีก หรือ (2) พบการเรียกใช้เมท็อด $m_i \in CM_L$
4) ส่วนการท่องเส้นทางแบบย้อนกลับจะจบลงเมื่อเกิดเหตุการณ์ใดเหตุการณ์หนึ่งดังนี้ (1) โหนด $n_q \in cmd_i$ ไม่มีเอดจ์เชื่อมต่ออีก หรือ (2) พบการเรียกใช้เมท็อด $m_i \in CM_U$
5) นอกจากนี้หากบนเอดจ์พบพารามิเตอร์ param เมท็อดของอัลกอริทึม backwardExtraction() จะท่องเส้นทางการสร้างพารามิเตอร์ (แบบย้อนกลับ: Backward Traversal) ต่อไป

```
function UP usagePatternExtraction(Type t_q, CMD cmd_i, Method Invocation Context CM(m))
{
   UP allUPs;
   for each method m_i ∈ cmd_i.getMethods(tq)
   {
      for each node n_input ∈ m_i.getNodes(t_q)
      {
         UP up = forwardExtraction(n_input, CM);
         up.sortMethodOrdering();
         allUPs.addPaths(up);
      }
   }
   return allUPs;
}

function UP forwardExtraction(Node n_input, Method Invocation Context CM)
{
   UP up;
   for each bi ∈ n_input.getBehavior()
   {
      Edge e = bi.getEdge();
      MethodInvocation m_i = e.getMethod();
      if (m_i ∈ CM) return;
      Node noutput = e.getOutputNode();
      Path p = createPath(n_input,e,n_output);
      if (n_output != null) up.addAll(mineUsagePattern(n_output));
      if (e have parameter)
      {
         Parameter p = e.getParameter();
```

รูปที่ 3.11 อัลกอริทึมของการสกัดรูปแบบการใช้งาน

```
        for each parameter pn ∈ e.getParameter()
            up.addAll(backwardExtraction(p_n));
      }
      up.add(p);
   }
   return up;
}

function UP backwardExtraction(Node n_output, Method Invocation Context CM)
{
   UP up;
   for each b_i ∈ n_output.getBackwardBehavior()
   {
      Edge e = b_i.getEdge();
      MethodInvocation m_i = e.getMethod();
      if (m_i ∈ CM) return;
      Node n_input = e.getInput();
      Path p = createPath(param,e,n_output);
      if (n_input != null) up.addAll(forwardExtraction(n_input));
      if (n_output != null) up.addAll(backwardExtraction(n_output));
      if (e have parameter)
      {
         Parameter p = e.getParameter();
         for each parameter p_n ∈ e.getParameter()
            up.addAll(backwardExtraction(p_n));
         ups.add(p);
      }
   }
   return up;
}
```

รูปที่ 3.11 อัลกอริทึมของการสกัดรูปแบบการใช้งาน (ต่อ)

รูปที่ 3.12 (a) แสดงกระบวนการทำงานของอัลกอริทึมของเมท็อด usagePatternExtraction() ในรูปแบบของแบบจำลองโค้ด โดยมี $\mathrm{UQ_G} = (\mathrm{t_q} = \mathrm{ASTRewrite})$ และมีโค้ดที่เป็นต้นแบบของแบบจำลองโค้ดในบทที่ 1 รูปที่ 1.1(d)

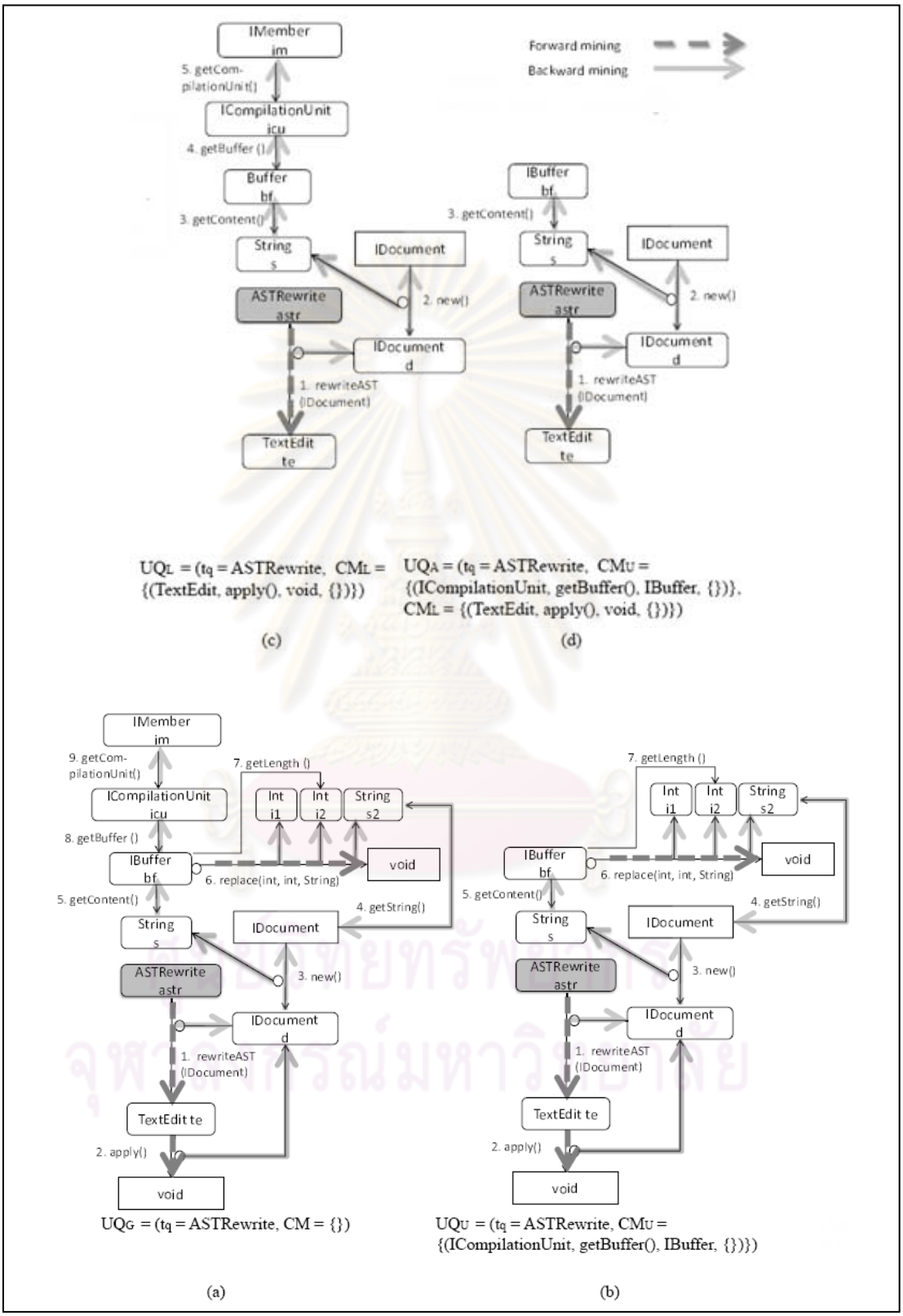
1) อัลกอริทึมจะระบุกลุ่มของโหนด $\mathrm{NQ} = \{\mathrm{n_q}\}$ โดยที่โหนด $\mathrm{n_q} \epsilon \mathrm{cmd_i}$ และประเภทของโหนด $\mathrm{n_q}$ = ASTRewrite ในตัวอย่างนี้ NQ = {ASTRewrite astr} โดยที่ $\mathrm{n_q}$ = ASTRewrite astr การท่องเส้นทางการเชื่อมต่อจะเริ่มที่โหนดนี้
2) โหนด $\mathrm{n_q}$ มีเอดจ์ที่เชื่อมต่อ 1 เอดจ์ นั่นคือ rewriteAST() อัลกอริทึมของเมท็อด forwardExtraction() จะท่องเส้นทางไปตามเอดจ์ rewriteAST() ไปยังโหนด TextEdit te
3) ที่โหนด TextEdit te อัลกอริทึมของเมท็อด forwardExtraction() จะทำงานแบบวนซ้ำจนกระทั่งพบโหนด void จึงหยุดทำงาน
4) ในการท่องตามทั้งสองเอดจ์ที่ผ่านมาจะพบพารามิเตอร์ระหว่างทาง ดังนั้น เมท็อด backwardExtraction() จะท่องไปตามเส้นทางการสร้างพารามิเตอร์ด้วย การท่องตามเส้นทางพารามิเตอร์จะเริ่มที่เอดจ์ rewriteAST() เมื่อพบพารามิเตอร์โหนด IDocument d อัลกอริทึมของเมท็อด backwardExtraction() จะเริ่มต้นทำงานโดยท่องไปตามเอดจ์ new() ที่เชื่อมต่อกับโหนด IDocument d แล้วท่องตามเอดจ์ของโหนดถัดไปเรื่อยๆ จนกระทั่งถึงโหนด IMember m จึงสิ้นสุดการทำงาน เนื่องจากไม่พบเอดจ์ที่เชื่อมต่อกับโหนด IMember m อีกแล้ว

ส่วนรูปที่ 3.12 (b) (c) และ (d) แสดงกระบวนการทำงานของอัลกอริทึมของเมท็อด usagePatternExtraction() ในรูปแบบของแบบจำลองโค้ด โดยใช้คำร้องขอแบบพิเศษ $\mathrm{UQ_U}$ $\mathrm{UQ_L}$ และ $\mathrm{UQ_A}$ ตามลำดับ โดยกระบวนการจะหยุดทำงานก่อนที่จะท่องไปครบทุกเอดจ์เมื่อพบการเรียกใช้เมท็อด $\mathrm{m_i} \epsilon \mathrm{CM}$

4. การจัดอันดับรูปแบบการใช้งาน (Usage Pattern Ranking)

หลังจากผ่านขั้นตอน 3 สกัดรูปแบบการใช้งานแล้วจะได้กลุ่มของรูปแบบการใช้งานอ็อบเจกต์ $\mathrm{t_q}$ ที่ต้องการ แต่เนื่องจากผลลัพธ์ที่ได้อาจมีจำนวนมาก และผลลัพธ์ที่ต้องการอาจอยู่ในอันดับล่างๆ โดยปกตินักพัฒนาซอฟต์แวร์จะนิยมใช้ผลลัพธ์จากอันดับที่ไม่เกิน 10-20 อันดับแรกเท่านั้น ดังนั้นการจัดอันดับรูปแบบการใช้งานอ็อบเจกต์จึงมีความสำคัญที่จะช่วยให้นักพัฒนาซอฟต์แวร์สามารถพบผลลัพธ์ที่ต้องการได้เร็วขึ้น ในงานวิจัย [4] ได้ทำการทดลองแล้วว่า วิทยาการศึกษาสำนึกแบบผสม (Combination Heuristic) เป็นวิธีการที่นำมาจัดอันดับโค้ดแล้วให้ผลลัพธ์ที่ตรงกับ

ความต้องการ ในวิทยานิพนธ์นี้จึงได้นำวิทยาการสำนึกแบบผสมเข้ามาช่วยจัดอันดับผลลัพธ์ สามารถดูรายละเอียดของวิทยาการศึกษาสำนึกแบบผสม



รูปที่ 3.12 โค้ดโมเดลแสดงตัวอย่างการสกัดรูปแบบการใช้งาน โดยมี t_q = ASTRewrite

รูปแบบการใช้งานอ็อบเจกต์ในรูป 1.1 สามารถจำลองการจัดอันดับในแต่ละประเภทได้ดังนี้

4.1. จัดอันดับด้วยวิทยาการสำนึกแบบความถี่ พบจำนวนรูปแบบการใช้งานอ็อบเจกต์แบบ A-D ในคลังข้อมูลโค้ดดังตารางที่ 3.4

ตารางที่ 3.4 จำนวนรูปแบบการใช้งานอ็อบเจกต์ที่พบในคลังข้อมูลของในรูปที่ 1.1

รูปแบบการใช้งานอ็อบเจกต์ในรูปที่ 1.1	จำนวนรูปแบบการใช้งานที่พบในคลังข้อมูล
A	5
B	1
C	8
D	4

4.2. จัดอันดับด้วยวิทยาการสำนึกแบบความยาว สามารถนับจำนวนคำสั่งของรูปแบบการใช้งานอ็อบเจกต์แบบ A-D ได้ดังตารางที่ 3.5

ตารางที่ 3.5 จำนวนคำสั่งในรูปแบบการใช้งานอ็อบเจกต์ในรูปที่ 1.1

รูปแบบการใช้งานอ็อบเจกต์ในรูปที่ 1.1	จำนวนคำสั่ง
A	8
B	10
C	8
D	13

*หมายเหตุ: บรรทัดที่เป็นการประกาศอ็อบเจกต์ เช่น FieldDeclaration fd จะไม่นับรวมเป็นหนึ่งคำสั่ง

4.3. จัดอันดับด้วยวิทยาการสำนึกแบบบริบท

ในหัวข้อนี้จะอธิบายวิธีการคำนวณค่าวิทยาการสำนึกแบบบริบทในรูปแบบการใช้งานอ็อบเจกต์แบบ D ในรูปที่ 1.1 เทียบกับโค้ดบริบทที่ผู้ใช้กำลังพัฒนาในรูปที่ 3.2 ดังนี้

กำหนดให้

Q หมายถึง โค้ดบริบทที่ผู้ใช้กำลังพัฒนาในรูปที่ 3.2

s หมายถึง โค้ดบริบทในรูปแบบการใช้งานอ็อบเจกต์แบบ D ในรูปที่ 1.1

4.3.1. มาตรวัดค่าความเหมือนกันระหว่างพาเรนท์ $M_P(Q, s)$ สามารถเขียนเป็นสมการได้ดังนี้

$$M_P(Q, s) = \frac{M_s(Q,s) + M_I(Q,s)}{2} \quad \text{_____ (8)}$$

โดยที่

$$M_s(Q, s) = M_T\big(\text{superclass}(Q), \text{superclass}(s)\big) \quad \text{_____ (9)}$$

$$M_I(Q, s) = \frac{I+S}{|\text{intf}(Q)|+|\text{intf}(s)|} \quad \text{_____ (10)}$$

$$I = \sum_{i_q \in \text{intf}(Q)} \left[\max_{i_s \in \text{intf}(s)} M_T(i_q, i_s)\right] \quad \text{_____ (11)}$$

$$S = \sum_{i_s \in \text{intf}(s)} \left[\max_{i_q \in \text{intf}(Q)} M_T(i_s, i_q)\right] \quad \text{_____ (12)}$$

เมื่อ

$\text{superclass}(Q), \text{superclass}(s)$ = กลุ่มของซูเปอร์คลาสที่อยู่ใน Q และ s ตามลำดับ

$\text{intf}(Q),\ \text{intf}(s)$ = กลุ่มของอินเตอร์เฟสที่อยู่ใน Q และ s ตามลำดับ

$i_q \in \text{intf}(Q)$ = อินเตอร์เฟสที่อยู่ในกลุ่ม $\text{intf}(Q)$

$i_s \in \text{intf}(s)$ = อินเตอร์เฟสที่อยู่ในกลุ่ม $\text{intf}(s)$

$M_T(i_q, i_s)$ = ค่าความเหมือนกันระหว่าง i_q และ i_s

$M_I(Q, s)$ = ค่าเฉลี่ยของความเหมือนกันระหว่างอินเตอร์เฟสสองกลุ่ม ได้แก่ $\text{intf}(Q)$ และ $\text{intf}(s)$

จากสมการข้างต้น สามารถเทียบค่าได้ดังนี้

$\text{superclass}(Q)$ = [AbstractChangeMemberModifierAction]

$\text{superclass}(s)$ = [AbstractChangeMemberModifierAction]

เนื่องจากทั้ง Q และ s มีซุปเปอร์คลาสเพียงค่าเดียวและเป็นชนิดเดียวกัน ดังนั้นค่าความเหมือนกันจึงเท่ากับ 1

$$M_s(Q, s) = 1 \quad \text{_____ (13)}$$

$\text{intf}(Q)$ = []

$\text{intf}(s)$ = []

เนื่องจากทั้ง Q และ s ไม่ได้อิมพลีเมนท์อินเตอร์เฟสเหมือนกัน ดังนั้นค่าความเหมือนกันจึงเท่ากับ 1

$$M_I(Q, s) = 1 \quad \text{_____ (14)}$$

ดังนั้น

$$M_P(Q, s) = \frac{1+1}{2}$$
$$= 1 \qquad \text{_____ (15)}$$

4.3.2 มาตรวัดค่าความเหมือนกันระหว่างบริบทแบบชนิดข้อมูล

ค่าความเหมือนกัน $M_{VT}(Q, s)$ จะคำนวณจากชนิดข้อมูลวิซิเบิลที่อยู่ในคำร้องขอ Q และ ชนิดข้อมูลวิซิเบิลที่อยู่ในโค้ด s

$M_{VT}(Q, s)$ สามารถเขียนเป็นสมการได้ดังนี้

$$M_{VT}(Q, s) = \frac{\sum_{t_c \in type(Q)}[\max_{t_s \in type(s)} M_T(t_c, t_s)]}{|type(Q)|} \qquad \text{_____ (16)}$$

เมื่อ

$type(Q), type(s)$ = กลุ่มของบริบทที่อยู่ใน Q และ s ตามลำดับ

$t_c \in type(Q)$ = บริบทที่อยู่ในกลุ่ม $type(Q)$

$t_s \in type(s)$ = บริบทที่อยู่ในกลุ่ม $type(s)$

$M_T(t_c, t_s)$ = ค่าความเหมือนกันระหว่าง t_c และ t_s

จากสมการข้างต้น สามารถเทียบสมการได้ดังนี้

$type(Q)$ = {ASTRewrite, null, Integer, int, void, BodyDeclaration, SimplePropertyDescriptor, TextEdit, IDocument, UndoEdit, IBuffer, String}

$type(s)$ = {IMember, boolean, ICompilationUnit, IOpenable, Buffer, String, int, IDocument}

เนื่องจากชนิดข้อมูลหนึ่งๆ สามารถมีหมายเลขดิวอีได้หลายหมายเลข โดยสามารถคำนวณ $M_T(t_c, t_s)$ ของแต่ละคู่ชนิดข้อมูลได้ดังนี้

$M_T(t_c, t_s) = M_T(\text{IMember, ASTRewrite})$

= (2.97.5, 1.1.1735) = 0.0 (ไม่มีความเหมือนกัน)

= (2.220.8, 1.1.1735) = 0.0

= (2.97.5, 1.1.1735) = 0.0

= (2.220.8, 1.1.1735) = 0.0

= (2.308.8, 1.1.1735) = 0.0

= (2.391.100.10, 1.1.1735) = 0.0

= (2.391.120, 1.1.1735) = 0.0

= (2.308.8, 1.1.1735) = 0.0

= (2.391.100.10, 1.1.1735) = 0.0

= (2.391.120, 1.1.1735) = 0.0

ดังนั้น ค่ามากที่สุดของ $M_T(t_s, t_c)$ = 0.0

$M_T(t_s, t_c)$ = M_T(IMember, null) = 0.0

$M_T(t_s, t_c)$ = M_T(IMember, Integer)

= (2.97.5, 1.1.722.8) = 0.0

= (2.97.5, 2.230.-2) = 0.0

= (2.97.5, 2.852.-2.1) = 0.0

= (2.220.8, 1.1.722.8) = 0.0

= (2.220.8, 2.230.-2) = 0.0

= (2.220.8, 2.852.-2.1) = 0.0

= (2.308.8, 1.1.722.8) = 0.0

= (2.308.8, 2.230.-2) = 0.0

= (2.308.8, 2.852.-2.1) = 0.0

= (2.391.100.10, 1.1.722.8) = 0.0

= (2.391.100.10, 2.230.-2) = 0.0

= (2.391.100.10, 2.852.-2.1) = 0.0

= (2.391.120, 1.1.722.8) = 0.0

= (2.391.120, 2.230.-2) = 0.0

= (2.391.120, 2.852.-2.1) = 0.0

ดังนั้น ค่ามากที่สุดของ $M_T(t_s, t_c)$ = 0.0

$M_T(t_s, t_c)$ = M_T(IMember, int) = 0.0

$M_T(t_s, t_c)$ = M_T(IMember, void) = 0.0

$M_T(t_s, t_c)$ = M_T(IMember, BodyDeclaration) = 0.0

$M_T(t_s, t_c)$ = M_T(IBuffer, IBuffer) = 1.0

$M_T(t_s, t_c)$ = M_T(String, ASTRewrite)

= (1.1.992, 1.1.1735) = 0.5

= (2.712.-2, 1.1.1735) = 0.0

= (2.230.-2, 1.1.1735) = 0.0

= (2.852.-2, 1.1.1735) = 0.0

ดังนั้น ค่ามากที่สุดของ $M_T(t_s, t_c)$ = 0.5

$M_T(t_s, t_c)$ = M_T(String, null) = 0.0

$M_T(t_s, t_c)$ = M_T(String, Integer)

= (1.1.992, 1.1.722.8) = 0.4

= (1.1.992, 2.230.-2) = 0.0

= (1.1.992, 2.852.-2.1) = 0.0

= (2.712.-2, 1.1.722.8) = 0.0

= (2.712.-2, 2.230.-2) = 0.0

= (2.712.-2, 2.852.-2.1) = 0.0

= (2.230.-2, 1.1.722.8) = 0.0

= (2.230.-2, 2.230.-2) = 0.0

= (2.230.-2, 2.852.-2.1) = 0.0

= (2.852.-2, 1.1.722.8) = 0.0

= (2.852.-2, 2.230.-2) = 0.0

= (2.852.-2, 2.852.-2.1) = 0.8

ดังนั้น ค่ามากที่สุดของ $M_T(t_s, t_c)$ = 0.8

$M_T(t_s, t_c)$ = M_T(IDocument, String)

= (2.142, 1.1.992) = 0.0

= (2.142, 2.712.-2) = 0.0

= (2.142, 2.230.-2) = 0.0

= (2.142, 2.852.-2) = 0.0

ดังนั้น ค่ามากที่สุดของ $M_T(t_s, t_c)$ = 0.0

ในทำนองเดียวกัน เมื่อเปรียบเทียบหมายเลขดิวอีของแต่ละชนิดข้อมูลเสร็จเรียบร้อยแล้ว จะนำมาหาค่ามากที่สุดของแต่ละชุดการเปรียบเทียบค่าความเหมือนกัน ซึ่งสามารถคำนวณได้ดังนี้

max (0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0) = 0

max (0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0) = 0

max (0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0) = 0

max (0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0) = 0

max (0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0) = 1.0

max (0.5,0.0,0.8,0.0,0.0,0.4,0.4,0.5,0.0,0.4,0.0,1.0) = 1.0

max (0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0) = 1.0

max (0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0) = 1.0

ดังนั้น

$$M_{VT}(Q,s) = \frac{\sum_{t_c \in type(Q)}[\max_{t_s \in type(s)} M_T(t_c,t_s)]}{|type(Q)|}$$

$$= \frac{1.0 + 1.0 + 1.0 + 1.0}{|8 + 2|}$$

$$= 0.5 \qquad \text{_____ (17)}$$

4.3.3. มาตรวัดความเหมือนกันระหว่างบริบท สามารถเขียนเป็นสมการได้ดังนี้

$$M_{CT}(Q,s) = \frac{M_P(Q,s) + M_{VT}(Q,s)}{2}$$

$$= ((1.0+1.0)/2.0 + 0.5)/2 = 1.5/2$$

$$= 0.75 \qquad \text{_____ (18)}$$

เพราะฉะนั้น ค่าความเหมือนกันระหว่างบริบทในรูปแบบการใช้งานอ็อบเจกต์แบบ D ในรูปที่ 1.1 เทียบกับโค้ดบริบทที่ผู้ใช้กำลังพัฒนาในรูปที่ 3.2 จึงมีค่าเท่ากับ 0.75

ตารางที่ 3.6 สรุปค่าวิทยาการสำนึกและอันดับการจัดเรียงผลลัพธ์ของแต่ละรูปแบบการใช้งานอ็อบเจกต์ในรูปที่ 1.1 โดยการจัดอันดับจะให้ความสำคัญกับค่าความเหมือนกันระหว่าง

บริบท > ความถี่ > ความยาว เพราะฉะนั้นลำดับที่ผู้ใช้ควรเลือกเป็นคำตอบมากที่สุด จึงเรียงจาก D > C > A > B ตามลำดับ

ตารางที่ 3.6 สรุปค่าวิทยาการสำนึกและอันดับผลลัพธ์ของรูปแบบการใช้งานอ็อบเจกต์

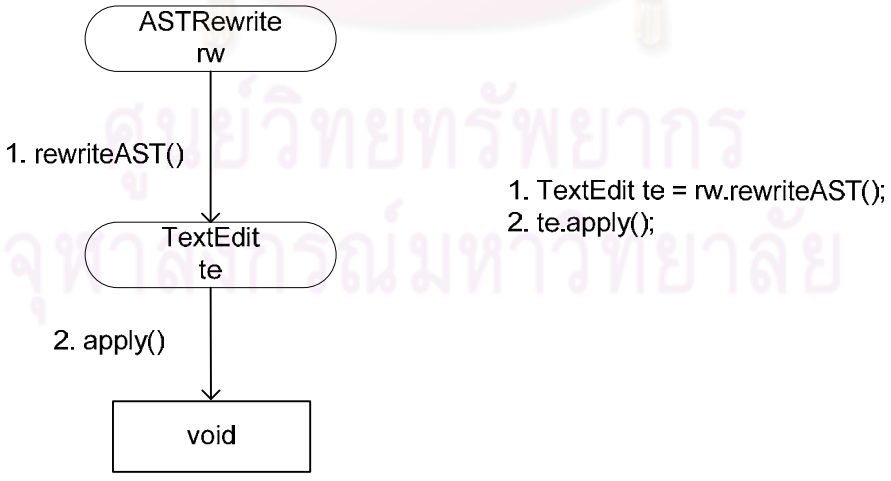
รูปแบบการใช้งาน	ค่าความเหมือนกันของบริบท	ค่าความถี่	ค่าความยาว	ลำดับที่ได้
A	0.63	5	8	3
B	0.54	1	10	4
C	0.75	4	8	2
D	0.75	8	13	1

*หมายเหตุ: ลำดับที่ได้จะเรียงอันดับเฉพาะผลลัพธ์ในสี่แบบนี้เท่านั้น

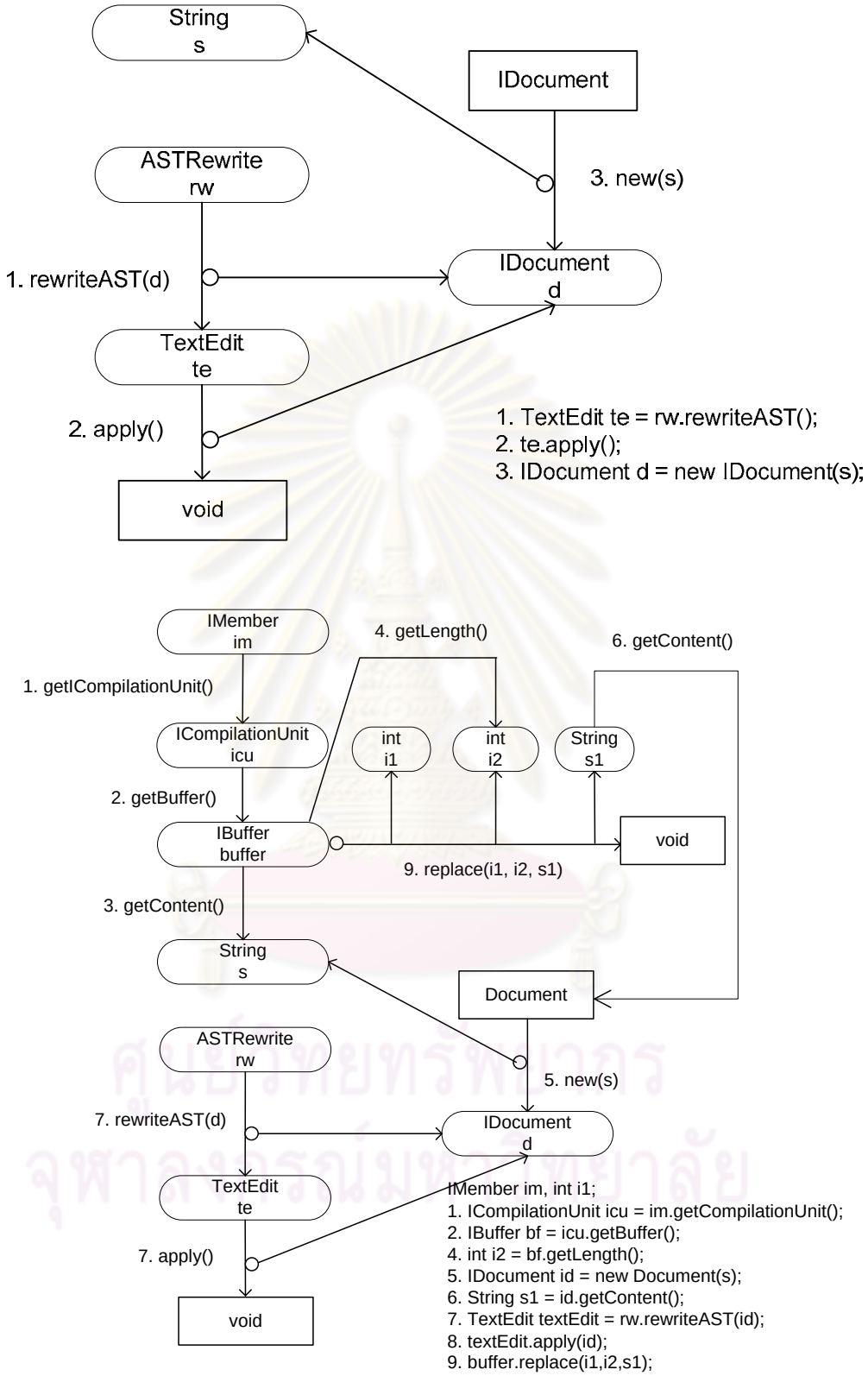
5. **การแปลงผลลัพธ์ให้อยู่ในรูปแบบของโค้ด** (Code Format Transformation)

หลังจากที่แต่ละผลลัพธ์ผ่านการจัดอันดับแล้วจะส่งผ่านมาแปลงให้กลายเป็นรูปแบบของโค้ดที่สามารถนำไปใช้งานได้ทันที จากนั้นจึงนำผลลัพธ์ทั้งหมดแสดงให้กับผู้ใช้งาน

รูปที่ 3.13 แสดงตัวอย่างขั้นตอนการแปลงแบบจำลองโค้ดให้กลายเป็นโค้ด โดยโค้ดที่นำมาแสดงในรูปมารูปแบบการใช้งานอ็อบเจกต์แบบ D ในรูปที่ 1.1 ทั้งนี้ สามารถศึกษาการแปลงผลลัพธ์ให้อยู่ในรูปแบบโค้ดโดยละเอียดได้ในภาคผนวก ก



รูปที่ 3.13 ตัวอย่างขั้นตอนการแปลงแบบจำลองโค้ดให้กลายเป็นโค้ด



รูปที่ 3.13 ตัวอย่างขั้นตอนการแปลงแบบจำลองโค้ดให้กลายเป็นโค้ด (ต่อ)

บทที่ 4
การออกแบบและพัฒนาเครื่องมือ

1. สภาพแวดล้อมที่ใช้พัฒนาเครื่องมือ

1.1. ฮาร์ดแวร์

ฮาร์ดแวร์ที่ใช้พัฒนาเครื่องมือ ประกอบด้วย

1.1.1. หน่วยประมวลผลอินเทล ความเร็ว 1.79 กิกะเฮิร์ซ (GHz)

1.1.2. หน่วยความจำหลัก 2 กิกะไบต์ (GB)

1.1.3. ฮาร์ดดิสก์ (Hard disk) 150 กิกะไบต์

1.2. ซอฟต์แวร์

ซอฟต์แวร์ที่ใช้พัฒนาเครื่องมือ ประกอบด้วย

1.2.1. ระบบปฏิบัติการไมโครซอฟท์วินโดวส์เอกซ์พี

1.2.2. พัฒนาเครื่องมือด้วยภาษาจาวา โดยใช้โปรแกรมอีคลิปรุ่น 3.5.0 (Eclipse 3.5.0)

1.2.3. JRE (Java Runtime Environment)

2. การออกแบบและพัฒนาเครื่องมือ

2.1. แผนภาพยูสเคสและรายละเอียดยูสเคส (Use Case Diagram และ Use Case Description)

จากบทที่ 3 สามารถแสดงแผนภาพยูสเคส เพื่ออธิบายความต้องการเชิงหน้าที่ของซอฟต์แวร์ (Functional requirements) ดังรูปที่ 4.1 โดยมีรายละเอียดต่างๆ ดังนี้

2.1.1. ยูสเคสเรียกดูรูปแบบการใช้งานอ็อบเจกต์ (Query Usage Pattern Use Case) มีหน้าที่สกัดรูปแบบการใช้งานอ็อบเจกต์จากโค้ดในคลังข้อมูลให้ผู้ใช้ โดยมีการนำบริบทโค้ดเข้าไปช่วยในกระบวนการเพื่อให้ผลลัพธ์ออกมาตรงตามความต้องการของผู้ใช้ ยูสเคสนี้จะเรียกใช้ยูสเคสสกัดบริบทโค้ด ยูสเคสเรียกข้อมูลแบบจำลองโค้ด ยูสเคสสกัดรูปแบบการใช้งาน และยูสเคสจัดอันดับรูปแบบการใช้งาน รายละเอียดของยูสเคสแสดงดังตารางที่ 4.1

2.1.2. ยูสเคสสกัดบริบทโค้ด (Extract Code Context Use Case) มีหน้าที่สกัดบริบทโค้ดในคลาสที่ผู้ใช้กำลังพัฒนา โดยการสกัดบริบทโค้ดจะมีอยู่สามประเภท ได้แก่ การสกัดโค้ด

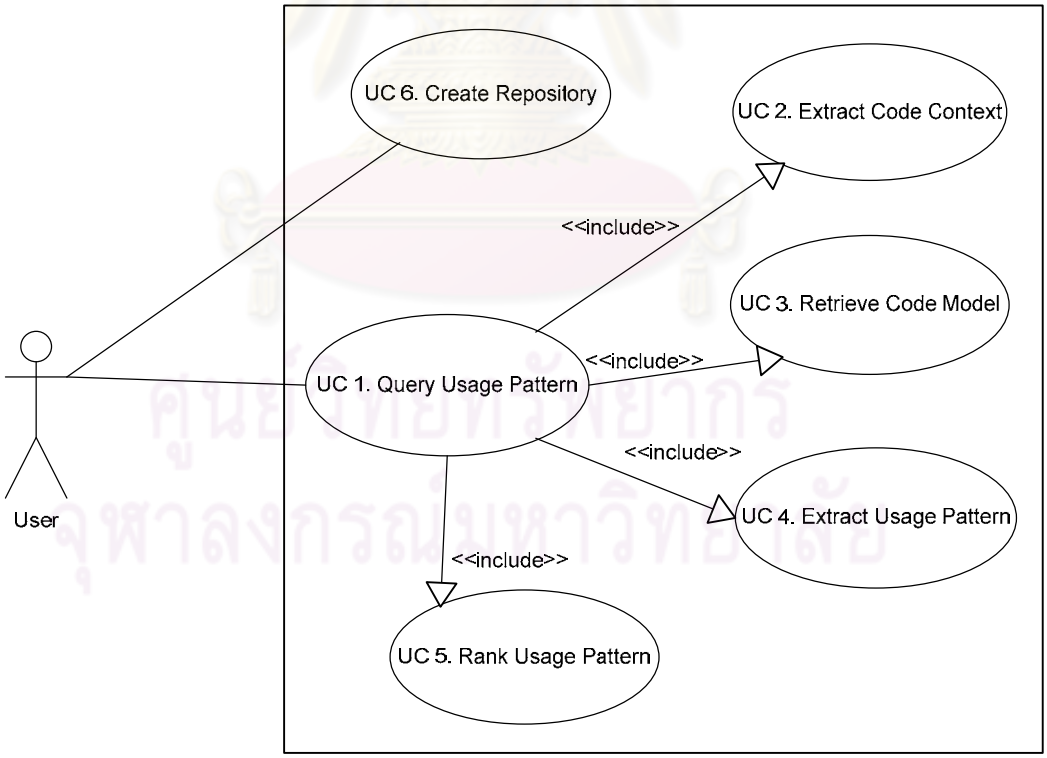
แบบไม่ใช้บริบท การสกัดแบบใช้บริบทด้านบน การสกัดแบบใช้บริบทด้านล่าง การสกัดแบบใช้บริบททั้งหมด รายละเอียดของยูสเคสแสดงดังตารางที่ 4.2

2.1.3. ยูสเคสเรียกข้อมูลแบบจำลองโค้ด (Retrieve Code Model Use Case) มีหน้าที่เรียกข้อมูลแบบจำลองโค้ดของคลังข้อมูล ผลลัพธ์ที่ได้คือไฟล์เอกซ์เอ็มแอลที่มีอิลิเมนท์ถูกต้องตามสกีมาของแบบจำลองโค้ด รายละเอียดของยูสเคสแสดงดังตารางที่ 4.3

2.1.4. ยูสเคสสกัดรูปแบบการใช้งาน (Extract Usage Pattern Use Case) เป็นยูสเคสที่มีหน้าที่สกัดรูปแบบการใช้งานของอ็อบเจกต์ที่ผู้ใช้ต้องการ รายละเอียดของยูสเคสแสดงดังตารางที่ 4.4

2.1.5. ยูสเคสจัดอันดับรูปแบบการใช้งาน (Rank Usage Pattern Use Case) เป็นยูสเคสที่มีหน้าที่จัดอันดับรูปแบบการใช้งานอ็อบเจกต์ โดยใช้วิทยาการสำนึกความยาว ความถี่ และบริบทโค้ดเข้าช่วย เพื่อให้ได้ผลลัพธ์ที่ตรงตามความต้องการของผู้ใช้ รายละเอียดของยูสเคสแสดงดังตารางที่ 4.5

2.1.6. ยูสเคสสร้างคลังข้อมูล (Create Repository Use Case) เป็นยูสเคสที่มีหน้าที่สร้างคลังแบบจำลองโค้ด รายละเอียดของยูสเคสแสดงดังตารางที่ 4.6



รูปที่ 4.1 แผนภาพยูสเคส

ตารางที่ 4.1 รายละเอียดยูสเคสเรียกดูรูปแบบการใช้งานอ็อบเจกต์

Use Case No:	1	
Use Case Name:	Query Usage Pattern	
Brief Description:	ยูสเคสนี้อธิบายการเรียกดูการใช้งานอ็อบเจกต์	
Primary Actor:	User	
Relationships:	Association: - Include: 1. Extract Code Context Use Case 2. Retrieve Code Model Use Case 3. Extract Usage Pattern Use Case 4. Rank Usage Pattern Use Case Extend: - Generalization: -	
Pre-condition :	-	
Post-condition :	-	
Normal Flows:	Step	Action
	1	ผู้ใช้เลือกประเภทการเรียกดูการใช้งานอ็อบเจกต์
	2	ผู้ใช้เลือกอ็อบเจ็กต์ที่ต้องการเรียกดูรูปแบบการใช้งาน
	3	ผู้ใช้เรียกดูรูปแบบการใช้งานอ็อบเจกต์
	4	UC2
	5	UC3
	6	UC4
	7	UC5
	8	เครื่องมือแสดงรูปแบบการใช้งานอ็อบเจกต์

ตารางที่ 4.2 รายละเอียดยูสเคสสกัดบริบทโค้ด

Use Case No:	2	
Use Case Name:	Extract Code Context	
Brief Description:	ยูสเคสนี้ใช้อธิบายการสกัดบริบทโค้ด	
Primary Actor:	UC1	
Relationships:	Association: - Include: - Extend: - Generalization: -	
Pre-condition :	ผู้ใช้ต้องเลือกประเภทของการเรียกดูรูปแบบการใช้งาน	
Post-condition :	-	
Normal Flows:	Step	Action
	1	เครื่องมือวิเคราะห์โค้ดที่ผู้ใช้กำลังพัฒนา
	2	เครื่องมือสกัดบริบทโค้ดที่สอดคล้องกับประเภทของการเรียกดูรูปแบบการใช้งาน
	3	เครื่องมือสกัดชื่อเต็ม (Qualified Name) ของประเภทอ็อบเจกต์ที่ผู้ใช้ต้องการเรียกดูรูปแบบการใช้งาน

ตารางที่ 4.3 รายละเอียดยูสเคสเรียกข้อมูลแบบจำลองโค้ด

Use Case No:	3	
Use Case Name:	Retrieve Code Model	
Brief Description:	ยูสเคสนี้ใช้อธิบายการเรียกข้อมูลแบบจำลองโค้ด	
Primary Actor:	UC1	
Relationships:	Association: - Include: - Extend: - Generalization: -	
Pre-condition :	- ไฟล์เอกซ์เอ็มแอลที่อยู่ในคลังข้อมูลต้องมีรูปแบบตามเอกซ์เอ็มแอลสกีมาดังที่ระบุไว้ในรูปที่ 3.3 - ต้องผ่าน UC1 และ UC2 มาก่อน	
Post-condition :	-	
Normal Flows:	Step	Action
	1	เครื่องมือนำประเภทอ็อบเจกต์ที่ได้จาก UC2 ไปเป็นดัชนีในการเรียกไฟล์เอกซ์เอ็มแอลจากคลังข้อมูล
	2	เครื่องมือคัดกรองไฟล์ที่ได้ด้วยบริบทโค้ดที่ได้จาก UC2
	3	เครื่องมือแปลงไฟล์ให้อยู่ในรูปอ็อบเจกต์ของแบบจำลองโค้ด

ตารางที่ 4.4 รายละเอียดยูสเคสสกัดรูปแบบการใช้งาน

Use Case No:	4	
Use Case Name:	Extract Usage Pattern	
Brief Description:	ยูสเคสนี้ใช้อธิบายการสกัดรูปแบบการใช้งาน	
Primary Actor:	UC1	
Relationships:	Association: - Include: - Extend: - Generalization: -	
Pre-condition :	ต้องผ่าน UC1 UC2 และ UC3 มาก่อน	
Post-condition :	-	
Normal Flows:	Step	Action
	1	เครื่องมือระบุตำแหน่งอ็อบเจกต์ที่ต้องการหารูปแบบการใช้งานภายในเมท็อด
	2	เครื่องมือท่องไปตามเส้นทางที่อ็อบเจกต์มีการเรียกใช้งานไปเรื่อยๆ จนกว่าจะสิ้นสุดเมท็อด หรือพบบริบทโค้ดที่ได้สกัดใน UC2
Alternate Flows:	1	ถ้าระหว่างการสกัดรูปแบบการใช้งานอ็อบเจกต์พบพารามิเตอร์ภายในเมท็อด เครื่องมือจะสกัดการสร้างพารามิเตอร์ด้วย
	2	ถ้าระหว่างเส้นทางการสกัดการสร้างพารามิเตอร์พบพารามิเตอร์อีกก็จะวนไปทำ 1 เรื่อยๆ จนกว่าจะสิ้นสุดเมท็อด หรือพบบริบทโค้ดที่ได้สกัดใน UC2
	3	ย้อนกลับไปที่ Basic Flow 2

ตารางที่ 4.5 รายละเอียดยูสเคสจัดอันดับรูปแบบการใช้งาน

Use Case No:	5	
Use Case Name:	Rank Usage Pattern	
Brief Description:	ยูสเคสนี้ใช้อธิบายการจัดอันดับรูปแบบการใช้งานอ็อบเจกต์	
Primary Actor:	-	
Relationships:	Association: - Include: - Extend: - Generalization: -	
Pre-condition :	ต้องผ่าน UC1 UC2 UC3 และ UC4	
Post-condition :	-	
Normal Flows:	Step	Action
	1	เครื่องมือวัดจำนวนรูปแบบการใช้งานอ็อบเจกต์ที่พบในฐานข้อมูล
	2	เครื่องมือวัดความยาว หรือจำนวนคำสั่งในรูปแบบการใช้งานอ็อบเจกต์
	3	เครื่องมือวัดความเหมือนกันระหว่างบริบทของรูปแบบการใช้งานอ็อบเจกต์ที่ได้จาก UC 5 และบริบทที่ได้จาก UC 2
	4	เครื่องมือนำค่าที่ได้จากข้อ 1 2 และ 3 มาคำนวณและจัดอันดับผลลัพธ์
	5	เครื่องมือแสดงรูปแบบการใช้งานอ็อบเจกต์

ตารางที่ 4.6 รายละเอียดยูสเคสสร้างคลังข้อมูล

Use Case No:	6	
Use Case Name:	Create Repository	
Brief Description:	ยูสเคสนี้ใช้อธิบายการสร้างคลังข้อมูลแบบจำลองโค้ด	
Primary Actor:	User หรือ Administrator	
Relationships:	Association: - Include: - Extend: - Generalization: -	
Pre-condition :	-	
Post-condition :	-	
Normal Flows:	Step	Action
	1	ผู้ใช้วางโค้ดลงในโฟลเดอร์ที่ระบุ
	2	ผู้ใช้เลือกสร้างคลังข้อมูล
	3	เครื่องมือสร้างคลังข้อมูล และอัพเดตดัชนีของประเภทอ็อบเจกต์ในโค้ดที่ผู้ใช้นำมาวางใหม่

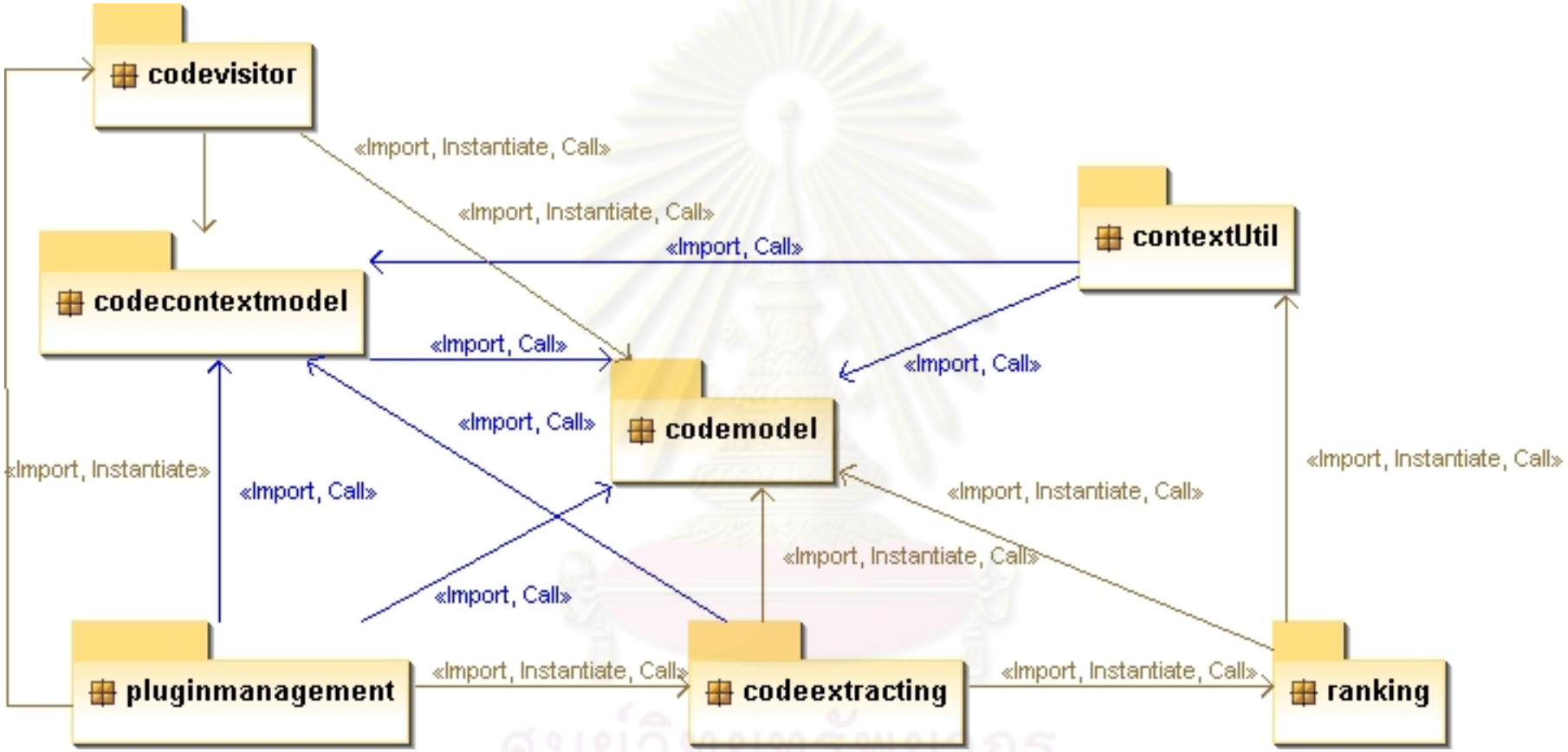
2.2. แผนภาพคลาส (Class Diagram)

เครื่องมือสนับสนุนวิทยานิพนธ์สามารถออกแบบเป็นคลาสไดอะแกรม ซึ่งมีแพ็กเกจคลาส และความสัมพันธ์ดังแสดงในรูปที่ 4.2 และรูปที่ 4.3 ซึ่งมีเฉพาะแพ็กเกจ ranking" เท่านั้นที่นำมาจากงานวิจัย [4] โดยมีรายละเอียดดังนี้

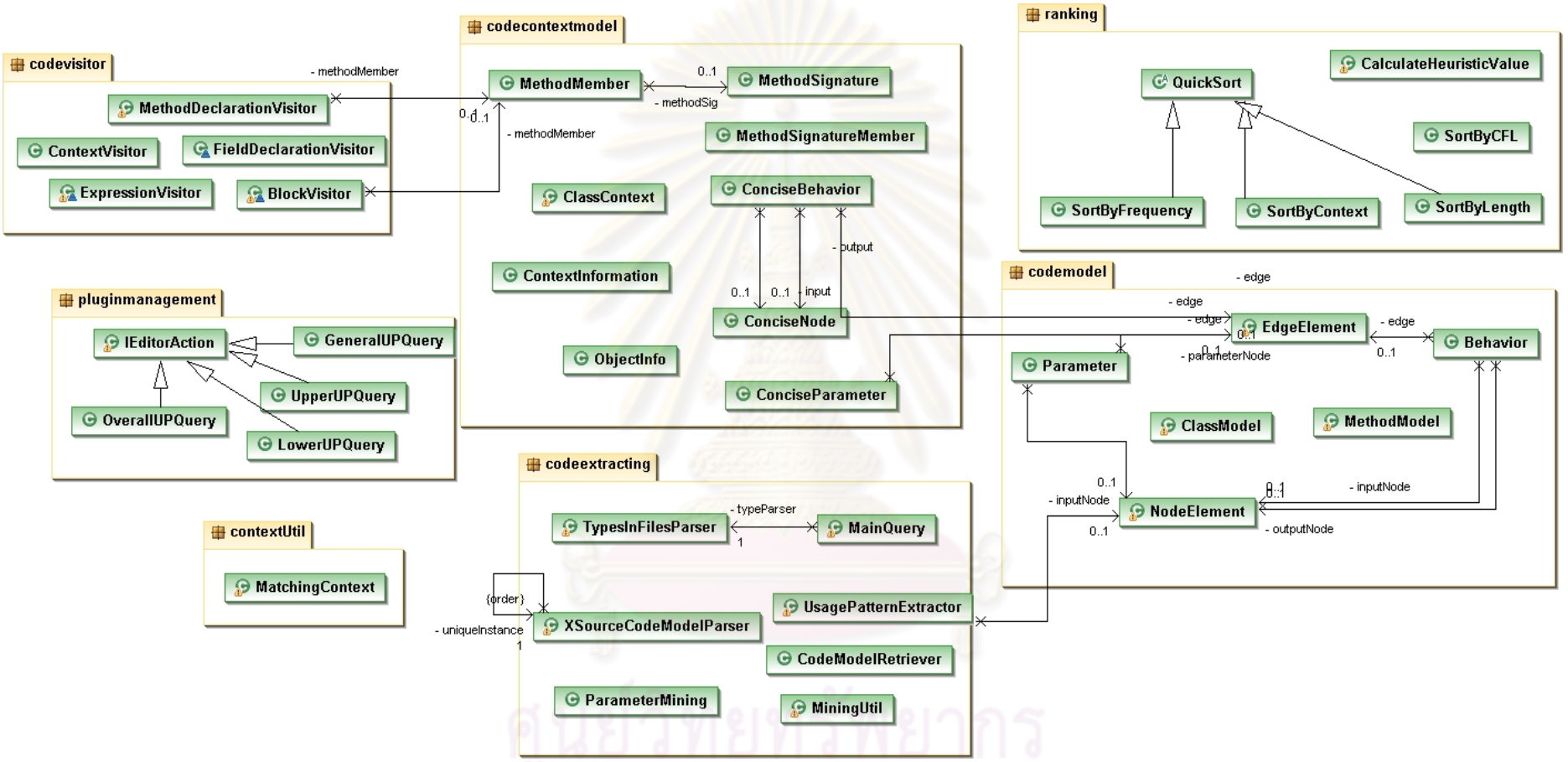
2.2.1. Codemodel Package

คลาสที่อยู่ใน Codemodel Package จะทำหน้าที่เก็บข้อมูลโค้ด เพื่อให้กลุ่มคลาสใน Codeextracting Package สกัดรูปแบบการใช้งานอ็อบเจกต์ได้ โดยลักษณะโครงสร้างของคลาสที่อยู่ในแพ็กเกจนี้จะเลียนแบบโครงสร้างคลาสตามมาตรฐานจาวา นั่นคือในคลาสจะประกอบด้วยชื่อคลาส แอททริบิวต์และเมท็อด เป็นต้น ใน Codemodel Package มีคลาสทั้งสิ้น 7 คลาส โดยมีรายละเอียดดังนี้

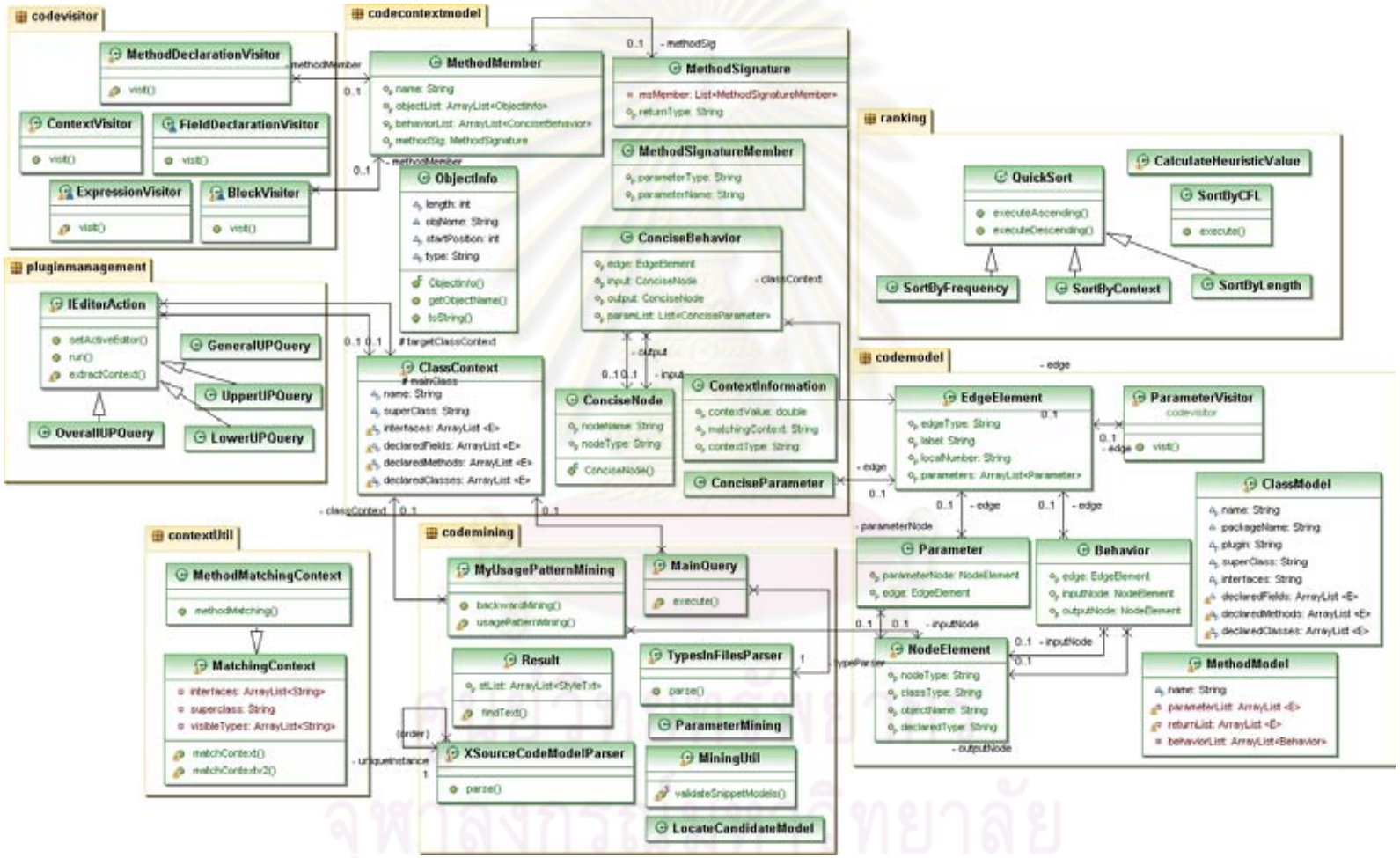
2.2.1.1. ClassModel Class ทำหน้าที่เก็บข้อมูลโค้ดของคลาสหนึ่งคลาส สามารถเทียบได้กับ Class ตามมาตรฐานจาวา



รูปที่ 4.2 แพ็กเกจไดอะแกรมแสดงการเชื่อมต่อระหว่างแพ็กเกจ



รูปที่ 4.3 แผนภาพคลาสไดอะแกรม (ไม่ระบุแอททริบิวท์และเมท็อด)



รูปที่ 4.4 แผนภาพคลาสไดอะแกรมอย่างละเอียด

2.2.1.2. MethodModel Class ทำหน้าที่เก็บข้อมูลโค้ดของเมท็อดหนึ่งเมท็อด มีความสัมพันธ์แบบ Compostion กับ ClassModel Class สามารถเทียบได้กับ Method Declaration ตามมาตรฐานจาวา

2.2.1.3. Behavior Class ทำหน้าที่เก็บข้อมูลโค้ดของคำสั่งโปรแกรมหนึ่งคำสั่ง โดยจะประกอบไปด้วย Input Output และ Operation ของคำสั่งนั้น สามารถเทียบได้กับ Statement ตามมาตรฐานจาวา

2.2.1.4. EdgeElement Class ทำหน้าที่เก็บข้อมูลโค้ดของ operation ใน Behavior Class สามารถเปรียบเทียบได้กับ Method ตามมาตรฐานจาวา

2.2.1.5, Parameter Class ทำหน้าที่เก็บข้อมูลโค้ดของพารามิเตอร์ สามารถเปรียบเทียบได้กับ Parameter ตามมาตรฐานจาวา

2.2.1.6, NodeElement Class ทำหน้าที่เก็บข้อมูลโค้ดของตัวแปรในคำสั่งโปรแกรม สามารถเปรียบเทียบได้กับ Variable ตามมาตรฐานจาวา

2.2.2. Codecontextmodel Package

คลาสที่อยู่ใน Codecontextmodel Package จะทำหน้าที่เก็บข้อมูลโค้ดบริบทเพื่อให้คลาสในกลุ่ม Codeextracting Package สกัดรูปแบบการใช้งานของอ็อบเจกต์ และเพื่อให้คลาสในกลุ่ม Ranking Package สามารถนำข้อมูลโค้ดบริบทไปใช้งานได้ ใน Codecodetextmodel Package มีคลาสทั้งสิ้น 9 คลาส โดยมีรายละเอียดดังนี้

2.2.2.1. ClassContext Class ทำหน้าที่เก็บข้อมูลโค้ดบริบทของคลาสหนึ่งคลาส สามารถเทียบได้กับ Class ตามมาตรฐานจาวา

2.2.2.2. MethodMember Class ทำหน้าที่เก็บข้อมูลโค้ดบริบทชนิดเมท็อด สามารถเทียบได้กับ Method Declaration ตามมาตรฐานจาวา

2.2.2.3. MethodSignature Class ทำหน้าที่เก็บและจัดการข้อมูลโค้ดบริบทชนิดเมท็อดซิกเนเจอร์ สามารถเทียบได้กับ Method Signature ตามมาตรฐานจาวา

2.2.2.4. MethodSignatureMember Class ทำหน้าที่เก็บข้อมูลโค้ดบริบทชนิดเมท็อดซิกเนเจอร์ระดับย่อย

2.2.2.5. ConciseBehavior Class ทำหน้าที่เก็บข้อมูลโค้ดบริบทชนิดคำสั่งโปรแกรม สามารถเทียบได้กับ Statement ตามมาตรฐานจาวา

2.2.2.6. ConciseNode Class ทำหน้าที่เก็บข้อมูลโค้ดบริบทชนิดตัวแปรคำสั่ง สามารถเทียบได้กับ Variable ตามมาตรฐานจาวา

2.2.2.7. ConciseParameter Class ทำหน้าที่เก็บข้อมูลโค้ดบริบทชนิดพารามิเตอร์ สามารถเทียบได้กับ Parameter ตามมาตรฐานจาวา

2.2.2.8. ContextInformation Class ทำหน้าที่เก็บข้อมูลโค้ดบริบทอื่นๆ นอกเหนือไปจากคลาสข้างต้น

2.2.3 Codevisitor Package

คลาสที่อยู่ใน Codevisitor Package จะทำหน้าที่ visit ข้อมูลโค้ดที่เก็บอยู่ใน ASTParser คลาสทั้งหมดจะออกแบบให้สอดคล้องกับ Visitor Pattern ที่ ASTParser วางโครงสร้างให้ผู้ใช้สามารถสร้างคลาสเพื่อ visit ข้อมูลภายในได้ ใน Codevisitor Package มีทั้งสิ้น 5 คลาส โดยมีรายละเอียดดังนี้

2.2.3.1. ContextVisitor Class ทำหน้าที่ visit บริบทโค้ด

2.2.3.2. BlockVisitor Class ทำหน้าที่ visit โค้ดส่วนที่เป็น Block เช่น Block ของ MethodDeclaration หรือ Block ย่อยภายใน MethodDeclaration

2.2.3.3. ExpressionVisitor Class ทำหน้าที่ visit โค้ดที่เป็น Expression

2.2.3.4. MethodDeclarationVisitor Class ทำหน้าที่ visit โค้ดที่เป็น MethodDeclaration

2.2.3.5. FieldDeclarationVisitor Class ทำหน้าที่ visit โค้ดที่เป็น FieldDeclaration

2.2.4. Codeextracting Package

คลาสที่อยู่ใน Codeextracting Package จะทำหน้าที่สกัดรูปแบบการใช้งานของอ็อบเจกต์จากข้อมูลที่เก็บไว้ในคลาสในกลุ่ม Codemodel Package ซึ่งคลาสที่อยู่ใน Codeextraction Package มีทั้งสิ้น 6 คลาส โดยมีรายละเอียดดังนี้

2.2.4.1. UsagePatternExtractor Class ทำหน้าที่สกัดรูปแบบการใช้งาน อ็อบเจกต์

2.2.4.2. MainQuery Class ทำหน้าที่เป็นคลาสที่เชื่อมต่อระหว่างส่วนประสานงานของผู้ใช้กับคลาสที่ทำงานเบื้องหลัง และเป็นคลาสที่ควบคุมการทำงานของโปรแกรมโดยรวม

2.2.4.3. TypeInFilesParser Class ทำหน้าที่เลือกไฟล์เอ็กซ์เอ็มแอลที่เกี่ยวข้องกับอ็อบเจกต์ที่ต้องการหารูปแบบการใช้งานขึ้นมาจากคลังข้อมูลโค้ด

2.2.4.4. XSourceCodeModelParser Class ทำหน้าที่แปลงไฟล์เอ็กซ์เอ็มแอลที่ได้จากคลาส TypeInFilesParser ให้เป็นแบบจำลองโค้ด

2.2.4.5. MiningUtil Class ทำหน้าที่ช่วยเหลือการสกัดรูปแบบของอ็อบเจกต์

2.2.4.6. ParameterMining Class ทำหน้าที่สกัดรูปแบบการใช้งานอ็อบเจกต์เฉพาะที่ต้องสกัดผ่านพารามิเตอร์ของโค้ดเท่านั้น

2.2.5. ContextUtil Package

คลาสที่อยู่ใน ContextUtil Package จะทำหน้าที่ช่วยเหลือโปรแกรมจัดการบริบทโค้ด เช่น ตรวจสอบความเหมือนกันระหว่างบริบท เป็นต้น ในคลาสที่อยู่ใน ContextUtil Package มีทั้งสิ้น 1 คลาส โดยมีรายละเอียดดังนี้

2.2.5.1. MatchingContext Class ทำหน้าที่เปรียบเทียบความเหมือนกันระหว่างประเภทของอ็อบเจกต์

2.2.6. Ranking Package

คลาสที่อยู่ใน Ranking Package จะทำหน้าที่เรียงอันดับรูปแบบการใช้งานของอ็อบเจกต์ ใน Ranking Package มีทั้งสิ้น 6 คลาส โดยมีรายละเอียดดังนี้

2.2.6.1. QuickSort Class ทำหน้าที่เรียงอันดับรูปแบบการใช้งานของอ็อบเจกต์ โดยใช้วิธีเรียงแบบ Quick Sort

2.2.6.2. SortByFrequency Class ทำหน้าที่เรียงอันดับรูปแบบการใช้งานอ็อบเจกต์ตามความถี่ของรูปแบบการใช้งานที่เกิดขึ้น

2.2.6.3. SortByContext Class ทำหน้าที่เรียงอันดับรูปแบบการใช้งานอ็อบเจกต์ตามความเหมือนกันระหว่างบริบทโค้ดในคลังข้อมูลและบริบทโค้ดที่กำลังพัฒนา

2.2.6.4. SortByLength Class ทำหน้าที่เรียงอันดับรูปแบบการใช้งานอ็อบเจกต์ตามความยาวของรูปแบบการใช้งาน หรือนับตามจำนวนคำสั่งโปรแกรมในหนึ่งรูปแบบการใช้งาน

2.2.6.5. SortByCFL Class ทำหน้าที่เรียงอันดับรูปแบบการใช้งานอ็อบเจกต์ด้วยการใช้หลักการสามแบบข้างต้นมาผสมกัน

2.2.6.6. CalculateHeuristicValue Class ทำหน้าที่คำนวณค่าวิทยาการศึกษาสำนึกทั้งสามแบบ ตามบทที่ 3 หัวข้อที่ 4

2.2.7. Pluginmanagement Package

คลาสที่อยู่ใน Pluginmanagement Package จะทำหน้าที่ติดต่อส่วนประสานงานผู้ใช้ในรูปแบบของปลั๊กอินอีคลิปและการทำงานเบื้องหลังของโปรแกรม ใน Pluginmanagement Package มีทั้งสิ้น 5 คลาส โดยมีรายละเอียดดังนี้

2.2.7.1. IEditorAction Class คือคลาสที่ extends มาจากคลาส org.eclipse.ui.IEditorActionDelegate ทำหน้าที่เชื่อมต่อกับปลั๊กอินอีคลิปและการทำงานเบื้องหลังของโปรแกรม

2.2.7.2. GeneralUPAction Class คือคลาสที่ extends มาจาก IEditorAction จะทำงานเมื่อผู้ใช้เลือกปุ่ม “GeneralUPQuery”

2.2.7.3. UpperUPAction Class คือคลาสที่ extends มาจาก IEditorAction จะทำงานเมื่อผู้ใช้เลือกปุ่ม “UpperUPQuery”

2.2.7.4. LowerUPAction Class คือคลาสที่ extends มาจาก IEditorAction จะทำงานเมื่อผู้ใช้เลือกปุ่ม “LowerUPQuery”

2.2.7.5. OverallUPAction Class คือคลาสที่ extends มาจาก IEditorAction จะทำงานเมื่อผู้ใช้เลือกปุ่ม “OverallUPQuery”

2.3. ซีเควนไดอะแกรม (Sequence Diagram)

2.3.1. Query Usage Pattern Sequence Diagram

ซีเควนไดอะแกรมนี้ ดังรูปที่ 4.5 อธิบายขั้นตอนการเรียกดูรูปแบบการใช้งานของอ็อบเจกต์ โดยมีขั้นตอนดังนี้

1) ผู้ใช้เรียกดูรูปแบบการใช้งานอ็อบเจกต์ผ่านอีคลิปปลั๊กอิน (Eclipse Plugin)

2). อีคลิปปลั๊กอินเรียกฟังก์ชัน query() ของคลาส IEditorAction ให้เริ่มต้นกระบวนการเรียกดูการใช้งานอ็อบเจกต์

3) คลาส IEditorAction เรียกฟังก์ชัน extractContext() ของคลาส ContextVisitor เพื่อสกัดบริบทโค้ดรอบข้างของโปรแกรมที่ผู้ใช้กำลังพัฒนา

4) คลาส ContextVisitor เรียกฟังก์ชัน visit() ของคลาส ClassContext เพื่อเข้าไปอ่านรายละเอียดของบริบทคลาส

5) คลาส ClassContext ให้ผลลัพธ์เป็นอ็อบเจกต์ที่มีข้อมูลบริบทคลาสอยู่ภายใน

6) คลาส ContextVisitor รีเทิร์นอ็อบเจกต์ ClassContext ให้คลาส IEditorAction

7) คลาส IEditorAction เรียกฟังก์ชัน execute(ClassContext) ของคลาส MainQuery เพื่อเริ่มกระบวนการสกัดรูปแบบการใช้งานอ็อบเจกต์ โดยส่งอ็อบเจกต์ ClassContext ที่ได้จากข้อ 6) ไปเป็นพารามิเตอร์ เพื่อใช้ในกระบวนการต่อไป

8) คลาส MainQuery เรียกใช้ฟังก์ชัน retrieveCodeModel() ของคลาส CodeModelRetriever เพื่อเลือกไฟล์เอ็กซ์เอ็มแอลที่มีการอ้างอิงถึงอ็อบเจกต์ที่ผู้ใช้ต้องการจากคลังข้อมูล

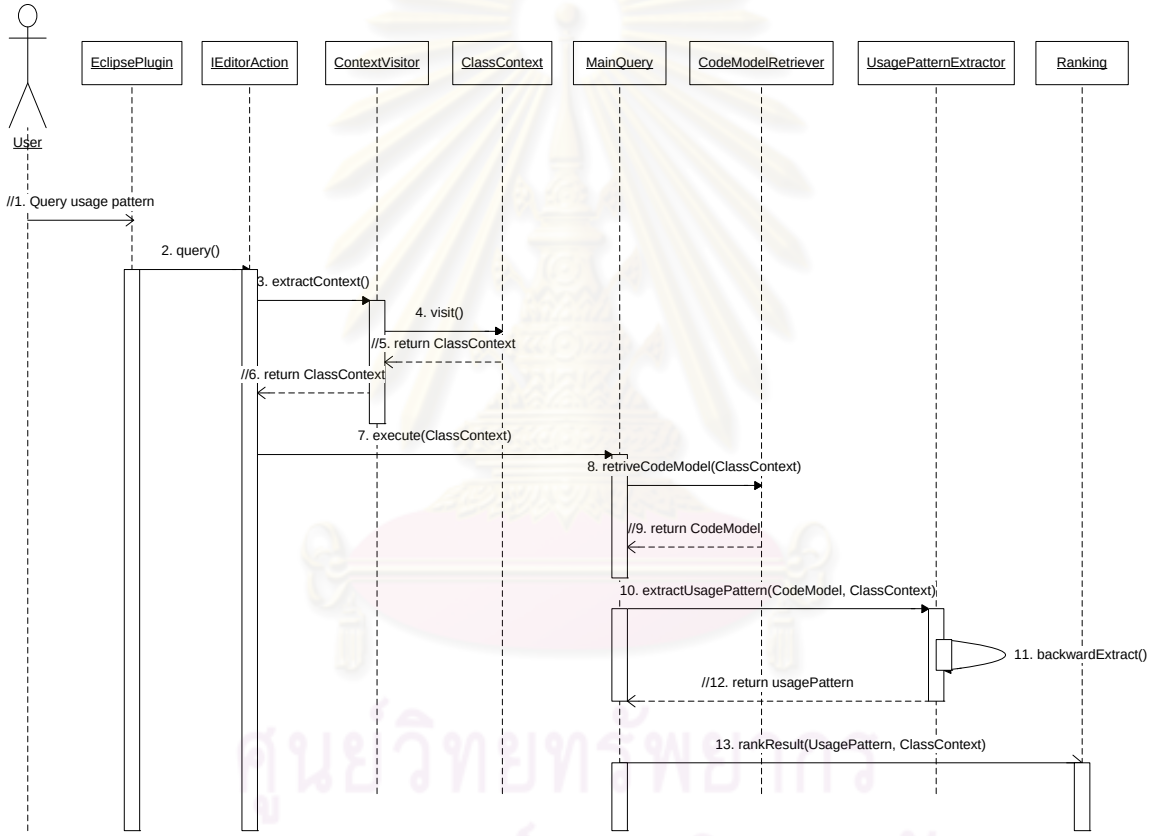
9) คลาส CodeModelRetriever รีเทิร์นผลลัพธ์เป็นแบบจำลองโค้ดให้แก่คลาส MainQuery

10) คลาส MainQuery เรียกใช้ฟังก์ชัน extractUsagePattern(CodeModel, ClassContext) ของคลาส UsagePatternExtractor(CodeModel, ClassContext) โดยส่งพารามิเตอร์ CodeModel และ ClassContext มาเพื่อใช้ในกระบวนการสกัดรูปแบบการใช้งานอ็อบเจกต์

11) ในกรณีที่พบมีพารามิเตอร์ภายในเมท็อดจะมีการเรียกใช้ฟังก์ชัน backwardExtract() ของคลาส UsagePatternExtractor เพื่อท่องไปตามเส้นทางของการสร้างพารามิเตอร์

12) คลาส UsagePatternExtractor ให้ผลลัพธ์เป็นรูปแบบการใช้งานอ็อบเจกต์แก่คลาส MainQuery

13) คลาส MainQuery เรียกใช้ฟังก์ชัน rankResult(UsagePattern, ClassContext) ของคลาส Ranking โดยส่งพารามิเตอร์ UsagePattern ที่ได้จากข้อ 12 และ ClassContext เพื่อจัดอันดับรูปแบบการใช้งานอ็อบเจกต์ต่อไป

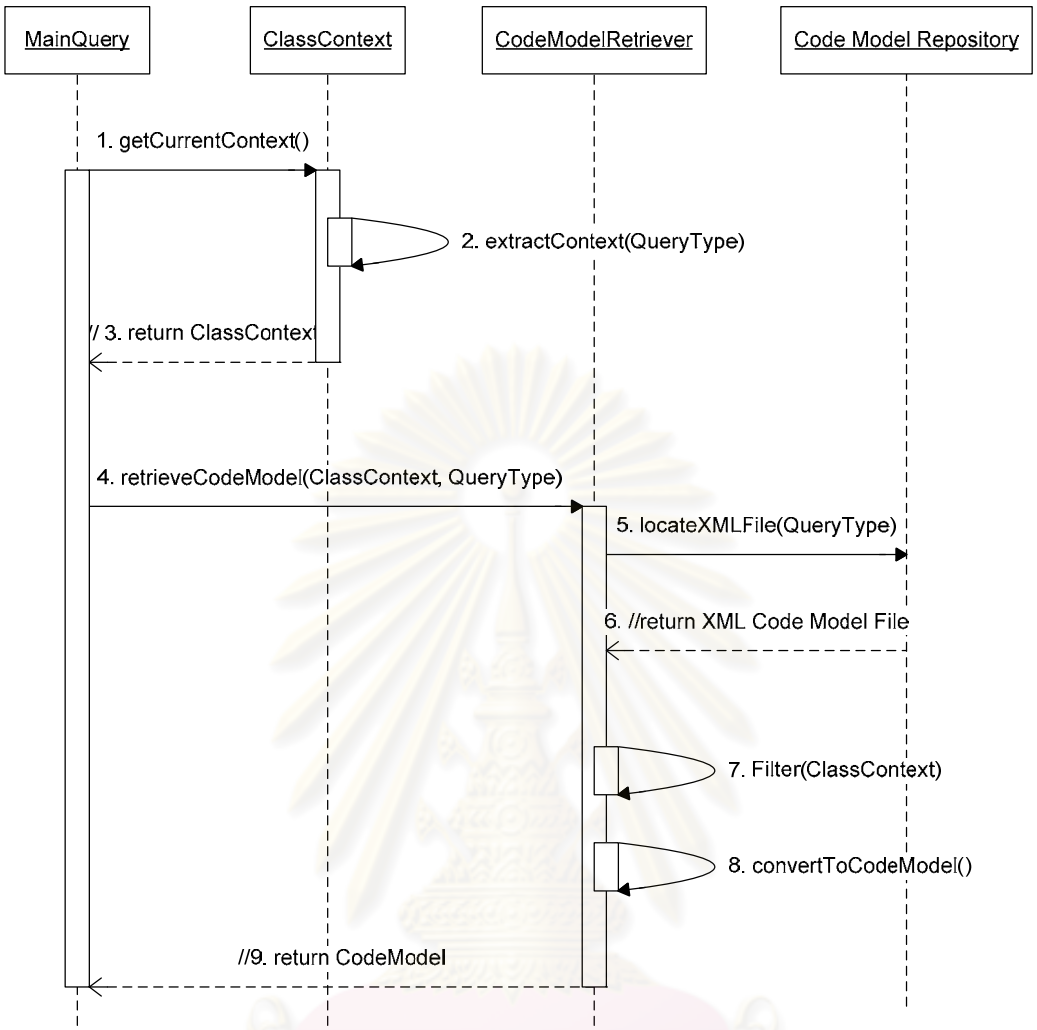


รูปที่ 4.5 Query Usage Pattern Sequence Diagram

2.3.2. Retrieve Code Model Sequence Diagram

ซีเควนไดอะแกรมนี้แยกย่อยมาจากซีเควนไดอะแกรมในรูปที่ 4.6 โดยจะอธิบายการดึงข้อมูลแบบจำลองโค้ดจากคลังข้อมูลโดยละเอียดดังนี้

1) คลาส MainQuery เรียกใช้ฟังก์ชัน getCurrentContext() จากคลาส ClassContext เพื่ออ่านข้อมูลคลาสที่ผู้ใช้กำลังพัฒนา



รูปที่ 4.6 Retrieve Code Model Sequence Diagram

2) คลาส ClassContext เรียกใช้ฟังก์ชัน extractContext() ของตัวเองสกัดและเพื่อแบ่งประเภทบริบทโค้ดจากคลาสที่ผู้ใช้กำลังพัฒนา

3) คลาส ClassContext ให้ผลลัพธ์เป็นอ็อบเจกต์ที่มีข้อมูลบริบทโค้ดอยู่ภายใน

4) คลาส MainQuery เรียกใช้ฟังก์ชัน retrieveCodeModel(ClassModel, QueryType) จากคลาส CodeModelRetriever และส่ง ClassModel และ QueryType ไปเป็นพารามิเตอร์ โดยที่ QueryType คือประเภทของอ็อบเจกต์ที่ผู้ใช้ต้องการทราบรูปแบบการใช้งาน

5) คลาส CodeModelRetriever เรียกใช้ฟังก์ชัน locateXMLFile(QueryType) โดยส่ง QueryType ไปเป็นพารามิเตอร์เพื่อใช้ QueryType เป็นดัชนีในการเลือกไฟล์เอกซ์เอ็มแอลที่มีการอ้างอิงถึง QueryType ออกมา

6) คลังข้อมูลให้ผลลัพธ์เป็นไฟล์เอกซ์เอ็มแอลที่มีรูปแบบตามเอกซ์เอ็มแอลสกีมาของแบบจำลองโค้ด

7) คลาส CodeModelRetriever เรียกใช้ฟังก์ชัน Filter(ClassContext) ของตัวเองโดยมี ClassContext เป็นพารามิเตอร์เพื่อคัดกรองไฟล์ที่มีการอ้างอิงถึงบริบทที่ต้องการก่อนจะส่งไปสกัดการใช้งานอ็อบเจกต์

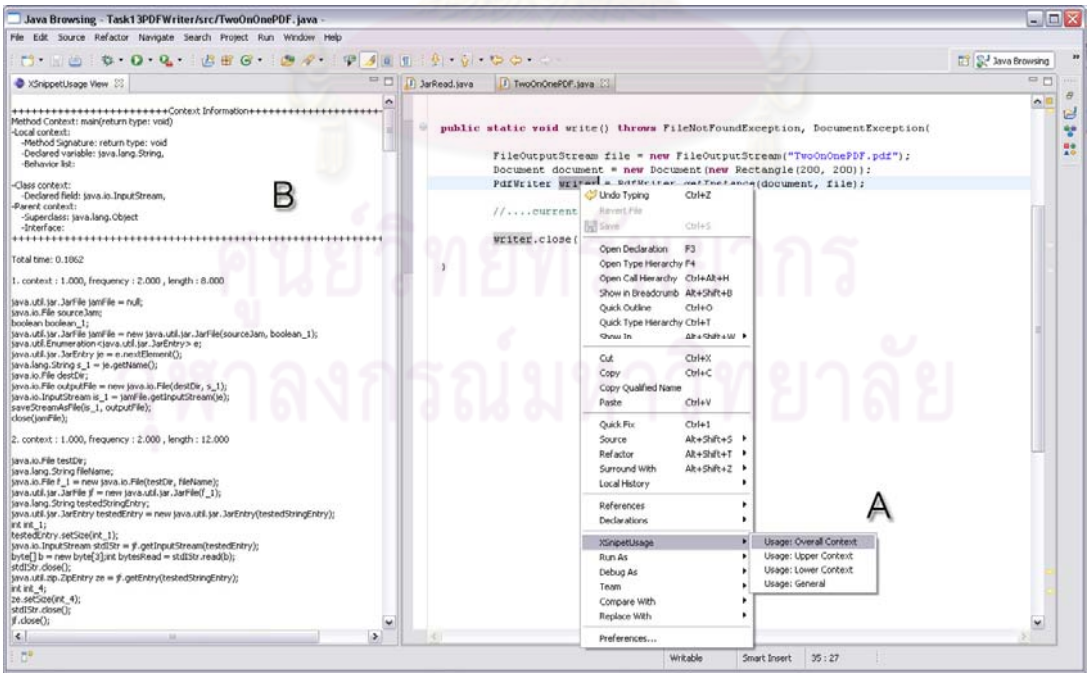
8) คลาส CodeModelRetriever เรียกใช้ฟังก์ชัน convertToCodeModel() เพื่อเปลี่ยนไฟล์ เอกซ์เอ็มแอลให้กลายเป็นอ็อบเจกต์ของคลาสในแพ็กเกจ CodeModel

9) คลาส CodeModelRetriever รีเทิร์นผลลัพธ์เป็นอ็อบเจกต์ของคลาสในแพ็คเกจ CodeModel ให้กับคลาส MainQuery

3. การออกแบบหน้าจอเครื่องมือ

วิทยานิพนธ์นี้ได้ออกแบบเครื่องมือสำหรับการเรียกดูรูปแบบการใช้งานอ็อบเจกต์ ซึ่งพัฒนาขึ้นเป็นปลั๊กอินโดยเชื่อมต่อกับโปรแกรมอีคลิปเพื่อความสะดวกในการเขียนโปรแกรม

รูปที่ 4.7 แสดงหน้าจอเครื่องมือขณะที่นักพัฒนาซอฟต์แวร์กำลังเขียนโปรแกรม โดยจำลองสถานการณ์ว่านักพัฒนาซอฟต์แวร์ต้องการสร้างไฟล์ PDF โดยใช้คลาส PDFWriter แต่คลาส PDFWriter อยู่ในคลาสไลบรารีที่นักพัฒนาซอฟต์แวร์ไม่คุ้นเคย จึงไม่ทราบว่าจะต้องใช้งานคลาส PDFWriter อย่างไร



รูปที่ 4.7 หน้าจอแสดงเครื่องมือเรียกดูรูปแบบการใช้งานอ็อบเจกต์

รูปที่ 4.7 (A) นักพัฒนาซอฟต์แวร์สามารถเลือกตัวแปรที่ต้องการดูรูปแบบการใช้งาน จากนั้นคลิกขวา โปรแกรมจะแสดงเมนู XSnippetUsage และฟังก์ชันการเรียกดูทั้งสี่แบบขึ้นบนหน้าจอ ได้แก่

1) Usage: Overall Context

2) Usage: Upper Context

3) Usage: Lower Context และ

4) Usage: General Context

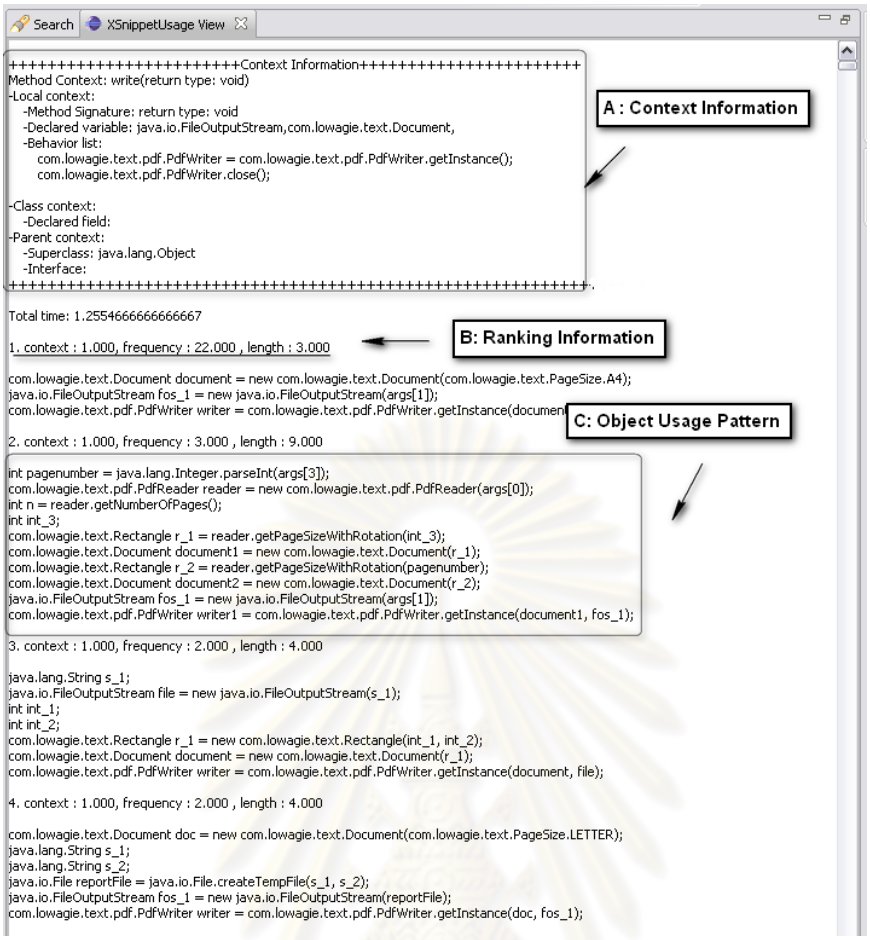
เมื่อนักพัฒนาซอฟต์แวร์เลือกรูปแบบที่ต้องการเรียกดูการใช้งานอ็อบเจกต์แล้ว โปรแกรมจะแสดงผลลัพธ์ในหน้าต่าง XSnippetUsage View ดังรูปที่ 4.7 (B) และรูปที่ 4.8 จะแสดงหน้าจอผลลัพธ์โดยละเอียดที่ได้จากการทำงานของโปรแกรม

รูปที่ 4.8 โปรแกรมจะแสดงรายละเอียดที่สำคัญ 3 ส่วน ประกอบด้วย

(A) คือ ข้อมูลบริบทโค้ดรอบข้างของโปรแกรมที่ผู้ใช้กำลังพัฒนา โดยจะแสดงข้อมูลของคลาส ซุปเปอร์คลาสและอินเตอร์เฟส แอททริบิวท์ของคลาส เมท็อดซิกเนเจอร์ ตัวแปรที่ประกาศใช้งานภายในเมท็อด และเมท็อดที่เรียกใช้งานภายใน

(B) คือ ค่าวิทยาการสำนึกของแต่ละประเภท ได้แก่ ค่าความเหมือนกันระหว่างบริบท ค่าความถี่หรือจำนวนของรูปแบบการใช้งานที่พบในฐานข้อมูล และค่าความยาวของรูปแบบการใช้งาน หรือจำนวนคำสั่งของรูปแบบการใช้งาน

(C) คือรูปแบบการใช้งานอ็อบเจกต์ที่ได้ โดยจะจัดเรียงอันดับตามค่าความเหมือนกันของบริบทเป็นอันดับแรก หากค่าความเหมือนกันเท่ากัน จึงจะเรียงอันดับด้วยความถี่ หากค่าความถี่เท่ากัน ก็จะเรียงลำดับด้วยความยาว



รูปที่ 4.8 หน้าจอแสดงผลลัพธ์โดยละเอียดจากการทำงานของโปรแกรม

บทที่ 5
การทดสอบเครื่องมือ

ในบทนี้จะกล่าวถึงการทดสอบเครื่องมือที่พัฒนาขึ้นเพื่อตรวจสอบว่าเครื่องมือดังกล่าวสามารถทำงานได้ถูกต้องตามขั้นตอนที่ได้ออกแบบไว้ในบทที่ 3 โดยจะอธิบายตัวอย่างโจทย์โปรแกรมที่ใช้ทดสอบ และอธิบายการเปรียบเทียบประสิทธิภาพของการสกัดรูปแบบการใช้งาน และการจัดอันดับในการสืบค้นแต่ละประเภท

1. ตัวอย่างโจทย์โปรแกรมที่ใช้ทดสอบ

การทดสอบเครื่องมือจะทดสอบโดยใช้โจทย์โปรแกรม โดยที่โจทย์โปรแกรมแต่ละข้อจะมีทรัพยากรต่างๆ ที่จำเป็นสำหรับการทำงานติดตั้งไว้เรียบร้อย แต่จะยกเว้นรูปแบบการใช้งานของอ็อบเจกต์ที่ต้องการไว้ เพื่อทดสอบว่าเครื่องมือสามารถให้คำตอบในส่วนที่หายไปได้ครบถ้วนและถูกต้องหรือไม่ คลังข้อมูลที่ใช้ทดสอบประกอบไปด้วยไฟล์ จาวา 9, 000 ไฟล์ ซึ่งสกัดมาจากคลาสไลบรารีต่างๆ ได้แก่ คลาสไลบรารีของอีคลิป 8 โจทย์ คลาสไลบรารีของจาวา 7 โจทย์ และคลาสไลบรารีอื่นๆ 4 โจทย์

ตารางที่ 5.1 สรุปคุณลักษณะของโจทย์โปรแกรมทั้งหมด 19 โจทย์ โดยจะแสดงบริบทแบบเมท็อดทั้งแบบบนและแบบล่างของแต่ละโจทย์ ในบทนี้ได้ยกตัวอย่างโจทย์โปรแกรม 4 โจทย์เพื่อแสดงให้เห็นถึงผลลัพธ์ที่เป็นรูปแบบการใช้งาน อันดับผลลัพธ์ที่ได้ และการวิเคราะห์ผลลัพธ์ โดยที่โดยตาราง 5.2 สรุปคำอธิบายโจทย์โปรแกรมไว้ดังนี้

1.1 การทดสอบเครื่องมือด้วยตัวอย่างโจทย์การใช้งานอ็อบเจกต์ ASTRewrite

1.1.1 คำอธิบายจากเอกสารเอพีไอ

org.eclipse.jdt.core.dom.rewrite.ASTRewrite คือ คลาสที่ทำหน้าที่เปลี่ยนแปลง DOM/AST tree โดยแจ้งแก่โหนด AST โดย ASTRewriter จะรวบรวมการเปลี่ยนแปลงทั้งหมดแล้วแปลให้กลายเป็น text edits ก่อนที่จะเปลี่ยนแปลงโค้ดจริงๆ

1.1.2 โจทย์โปรแกรม

ต้องการสืบค้นรูปแบบการใช้งานอ็อบเจกต์ ASTRewrite เพื่อเปลี่ยนแปลงข้อความที่อยู่ในอีคลิปเอดิเตอร์

1.1.3 สภาพแวดล้อมในระหว่างการเขียนโปรแกรม:

แสดงดังรูปที่ 5.1

ตารางที่ 5.1 ตารางสรุปคุณลักษณะของโจทย์โปรแกรมที่ใช้ทดสอบทั้งหมด 19 โจทย์

	t_q	CT_{US}	CT_{LS}
1.	ASTRewrite	(IDocument, new, IDocument, {String}), (IBuffer, getContent(), String, {})	(IBuffer, void, replace(), {int, String, String})
2.	IWorkingCopyManager	(JavaUI, getWorkingCopyManager(), IWorkingCoyManager, {})	(IWorkingCopyManager, void, disconnect(), {IEditorInput})
3.	IWorkingCopyManager	-	-
4.	ITextEditor	(IEditorPart, getEditorInput(), IEditorInput, {})	(ITextEditor, void, selectAndReveal(), {int, int}))
5.	ListRewrite	(ASTRewrite, ASTRewrite, create, {AST}) (ASTRewrite, getListRewrite(), ListRewrite, {Block, int})	(IBuffer, int, replace(), {int, String, String})
6.	JavaTextTools	(JFaceResources, getFont(), Font, {String})	(JavaSourceViewer, getControl(), StyledText, {})
7.	IScanner	(ToolFactory, createScanner(), IScanner, boolean, boolean, String, String)	(ToolFactory, createScanner, IScanner, {boolean, Boolean, Boolean, String, String})
8.	JarFile	(File, new(), File, {})	(ZipFile, void, close(), {})
9.	JarFile	-	-
10.	ScriptEngine	(ScriptEngineManager, new(), ScriptEngineManager, {}) (ScriptEngineManager, getEngineByExtension(), {String})	(ScriptEngine, eval(), void, {Reader})
11.	ZipOutputStream	(FileOutputStream, new(), FileOutputStream, {String}) (BufferedOutputStream, new(), BufferedOutputStream, {FileOutputStream})	(ZipOutputStream, close(), void, {})
12.	MimeMessage	(Session, getInstance(), Session, {Properties, Authenticator}) (Mesasage, new(), MimeMessage, {Session})	(Message, void, saveChanges(), {})
13.	Process	(Runtime, getRuntime(), Runtime, {}) (Runtime, exec(), Process, {String})	(Process, getInputStream(), InputStream, {})
14.	JarOutputStream	(FileOutputStream, new(), FileOutputStream, {String})	(JarOutputStream, close(), void, {})
15.	SearchControl	(InitialDirContext, new(), DirContext, {Hashtable})	(DirContext, close(), void, {})
16.	PDFWriter	(FileOutputStream, new(), FileOutputStream, {String})	(PDFWriter, close(), void, {})
17.	PDFWriter	(FileOutputStream, new(), FileOutputStream, {String}) (Color, new(), Color, {int, int, int})	(Document, add(), boolean, {Element})
18.	PDFWriter	-	-
19.	PDFStamper	(PDFReader, new(), PDFReader, {String}) (FileOutputStream, new(), FileOutputStream, {String})	(PDFContentByte, endText(), void, {})

ตารางที่ 5.2 คำอธิบายโจทย์โปรแกรม

หัวข้อ	คำอธิบาย
คำอธิบายจากเอกสารเอพีไอ	คำอธิบายการใช้งานคลาสที่ปรากฏภายในโจทย์โดยคัดลอกมาจากเอกสารเอพีไอของไลบรารีนั้นๆ
โจทย์โปรแกรม	โจทย์โปรแกรมที่คิดขึ้นเพื่อทดสอบว่าเครื่องมือสามารถตอบโจทย์โปรแกรมเป็นรูปแบบการใช้อ็อบเจกต์ตามที่คาดหวังไว้ได้หรือไม่
สภาพแวดล้อมในการเขียนโปรแกรม	โค้ดรอบข้างที่ผู้ใช้กำลังเขียนในระหว่างที่ใช้งานเครื่องมือ
ผลลัพธ์ที่ต้องการ	รูปแบบการใช้งานที่ถูกต้องซึ่งคาดหวังไว้ว่าเครื่องมือจะให้ผลลัพธ์ที่ตรงกัน
การเปรียบเทียบผลลัพธ์ที่ได้	การเปรียบเทียบผลลัพธ์ของแต่ละการสืบค้นว่าแตกต่างกันอย่างไร และเปรียบเทียบลำดับของผลลัพธ์ว่าการสืบค้นแบบใดจะให้ผลลัพธ์ในอันดับที่ดีที่สุด

```
public class MakeIMemberPrivateAction
   extends AbstractChangeMemberModifierAction
{
   private IMember im;
   protected boolean performAction()
   {
      ICompilationUnit icu = im.getCompilationUnit();
      IBuffer bf = icu.getBuffer();
      String content = bf.getContents();
      int i = bf.getLength();
      IDocument id = new Document(content);
      String s = id.get();
      ASTRewrite astr = ASTRewrite.create(null);
                              ////Current Development////
      bf.replace(0, i, s);
      return true;
   }
}
```

รูปที่ 5.1 สภาพแวดล้อมสำหรับโจทย์การใช้งานอ็อบเจกต์ ASTRewrite

1.1.4 ผลลัพธ์ที่ต้องการ

astr.set(node, property, value, editGroup)

TextEdit textEdit = astr.rewriteAST(id, null);

textEdit.apply(id);

โดยมีรายละเอียดแสดงดังรูปที่ 5.2

1.1.5 การเปรียบเทียบผลลัพธ์ที่ได้

การสืบค้นแบบ UQ_G ให้ผลลัพธ์ 473 จำนวน ผลลัพธ์ที่ต้องการอยู่ในลำดับที่ 1 การสืบค้น UQ_U ให้ผลลัพธ์ 280 จำนวน ผลลัพธ์ที่ต้องการอยู่ในลำดับที่ 1 การสืบค้น UQ_L ให้ผลลัพธ์ 23 จำนวน ผลลัพธ์ที่ต้องการอยู่ในลำดับที่ 1 และการสืบค้น UQ_A ให้ผลลัพธ์ 270 จำนวน ผลลัพธ์ที่ต้องการอยู่ในลำดับที่ 1 โดยรายละเอียดของผลลัพธ์แสดงดังตารางที่ 5.3

```
public class MakeIMemberPrivateAction
   extends AbstractChangeMemberModifierAction
{
   private IMember im;
   protected boolean performAction()
   {
      ICompilationUnit icu = im.getCompilationUnit();
      IBuffer bf = icu.getBuffer();
      String content = bf.getContents();
      int i = bf.getLength();
      IDocument id = new Document(content);
      String s = id.get();
      ASTRewrite astr = ASTRewrite.create(null);

      astr.set(node, property, value, editGroup)
      TextEdit textEdit = astr.rewriteAST(id, null);
      textEdit.apply(id);

      bf.replace(0, i, s);
      return true;
   }
}
```

ผลลัพธ์ที่ต้องการ

รูปที่ 5.2 ผลลัพธ์ที่ต้องการสำหรับตัวอย่างโจทย์การใช้งานอ็อบเจกต์ ASTRewrite

ตารางที่ 5.3 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ ASTRewrite

ประเภทของการสืบค้น	ลำดับ	จำนวนผลลัพธ์ที่พบในคลังข้อมูล	รูปแบบการใช้งานอ็อบเจกต์ที่ได้

UQ_{G}	1	473	ASTParser parser = ASTParser.newParser(AST.JLS3);
			IMember member;
			ICompilationUnit icu_1 = member.getCompilationUnit();
			parser.setSource(icu_1);
			ASTNode astn_1 = parser.createAST(null_1);
			CompilationUnit cuNode = (CompilationUnit) astn_1;
			BodyDeclaration bd = cuNode.getBodyDeclaration();
			int modFlags = bd.getModifiers();
			int PROTECTED;
			int modFlags = PROTECTED; int PRIVATE;
			int modFlags = PRIVATE;
			ICompilationUnit cu = member.getCompilationUnit();
			IBuffer buffer = cu.getBuffer();
			String s_1 = buffer.getContents();
			IDocument doc = new Document(s_1);

ตารางที่ 5.3 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ ASTRewrite (ต่อ)

ประเภทของการสืบค้น	ลำดับ	จำนวนผลลัพธ์ที่พบในคลังข้อมูล	รูปแบบการใช้งานอ็อบเจกต์ที่ได้

			ASTRewrite rewrite = ASTRewrite.create(null_2);
			Integer i_1 = new Integer(modFlags);
			rewrite.set(bd, FieldDeclaration.MODIFIERS_PROPERTY, i_1, null_3);
			Integer i_2 = new Integer(modFlags);
			rewrite.set(bd, FieldDeclaration.MODIFIERS_PROPERTY, i_2, null_4);
			TextEdit textEdit = rewrite.rewriteAST(doc, null_5);
			UndoEdit ue_1 = textEdit.apply(doc);
			int int_2 = buffer.getLength();
			String s_2 = doc.get();
			int int_1;
			buffer.replace(int_1, int_2, s_2);

UQ_{U}	1	280	ASTParser parser = ASTParser.newParser(AST.JLS3); IMember member; ICompilationUnit icu_1 = member.getCompilationUnit(); parser.setSource(icu_1);

ตารางที่ 5.3 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ ASTRewrite (ต่อ)

ประเภทของการสืบค้น	ลำดับ	จำนวนผลลัพธ์ที่พบในคลังข้อมูล	รูปแบบการใช้งานอ็อบเจกต์ที่ได้

			ASTNode astn_1 = parser.createAST(null_1);
			CompilationUnit cuNode = (CompilationUnit) astn_1;
			BodyDeclaration bd = cuNode.getBodyDeclaration();
			int modFlags = bd.getModifiers();
			int PROTECTED; int modFlags = PROTECTED;
			int PRIVATE; int modFlags = PRIVATE;
			ICompilationUnit cu = member.getCompilationUnit();
			IBuffer buffer = cu.getBuffer();
			String s_1 = buffer.getContents();
			IDocument doc = new Document(s_1);
			ASTRewrite rewrite = ASTRewrite.create(null_2);
			Integer i_1 = new Integer(modFlags);
			rewrite.set(bd, FieldDeclaration.MODIFIERS_PROPERTY, i_1, null_3);
			Integer i_2 = new Integer(modFlags);
			rewrite.set(bd, FieldDeclaration.MODIFIERS_PROPERTY, i_2, null_4);

ตารางที่ 5.3 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ ASTRewrite (ต่อ)

ประเภทของการสืบค้น	ลำดับ	จำนวนผลลัพธ์ที่พบในคลังข้อมูล	รูปแบบการใช้งานอ็อบเจกต์ที่ได้

UQ_{L}	1	23	TextEdit textEdit = rewrite.rewriteAST(doc, null_5);
			UndoEdit ue_1 = textEdit.apply(doc);
			int int_2 = buffer.getLength();
			String s_2 = doc.get();
			int int_1;
			buffer.replace(int_1, int_2, s_2);
			ASTParser parser = ASTParser.newParser(AST.JLS3);
			IMember member;
			ICompilationUnit icu_1 = member.getCompilationUnit();
			parser.setSource(icu_1);
			ASTNode astn_1 = parser.createAST(null_1);
			CompilationUnit cuNode = (CompilationUnit) astn_1;
			BodyDeclaration bd = cuNode.getBodyDeclaration();
			int modFlags = bd.getModifiers();
			int PROTECTED; int modFlags = PROTECTED;

ตารางที่ 5.3 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ ASTRewrite (ต่อ)

ประเภทของการสืบค้น	ลำดับ	จำนวนผลลัพธ์ที่พบในคลังข้อมูล	รูปแบบการใช้งานอ็อบเจกต์ที่ได้

			int PRIVATE; int modFlags = PRIVATE;
			ICompilationUnit cu = member.getCompilationUnit();
			IBuffer buffer = cu.getBuffer();
			String s_1 = buffer.getContents();
			IDocument doc = new Document(s_1);
			ASTRewrite rewrite = ASTRewrite.create(null_2);
			Integer i_1 = new Integer(modFlags);
			rewrite.set(bd, FieldDeclaration.MODIFIERS_PROPERTY, i_1, null_3);
			Integer i_2 = new Integer(modFlags);
			rewrite.set(bd, FieldDeclaration.MODIFIERS_PROPERTY, i_2, null_4);
			TextEdit textEdit = rewrite.rewriteAST(doc, null_5);
			UndoEdit ue_1 = textEdit.apply(doc);
			int int_2 = buffer.getLength();
			String s_2 = doc.get();
			int int_1;

ตารางที่ 5.3 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ ASTRewrite (ต่อ)

ประเภทของการสืบค้น	ลำดับ	จำนวนผลลัพธ์ที่พบในคลังข้อมูล	รูปแบบการใช้งานอ็อบเจกต์ที่ได้
			buffer.replace(int_1, int_2, s_2);

UQ_{A}	1	270	ASTParser parser = ASTParser.newParser(AST.JLS3);
			IMember member;
			ICompilationUnit icu_1 = member.getCompilationUnit();
			parser.setSource(icu_1);
			ASTNode astn_1 = parser.createAST(null_1);
			CompilationUnit cuNode = (CompilationUnit) astn_1;
			BodyDeclaration bd = cuNode.getBodyDeclaration();
			int modFlags = bd.getModifiers();
			int PROTECTED; int modFlags = PROTECTED;
			int PRIVATE; int modFlags = PRIVATE;
			ICompilationUnit cu = member.getCompilationUnit();
			IBuffer buffer = cu.getBuffer();
			String s_1 = buffer.getContents();
			IDocument doc = new Document(s_1);

ตารางที่ 5.3 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ ASTRewrite (ต่อ)

ประเภทของการสืบค้น	ลำดับ	จำนวนผลลัพธ์ที่พบในคลังข้อมูล	รูปแบบการใช้งานอ็อบเจกต์ที่ได้

			ASTRewrite rewrite = ASTRewrite.create(null_2);
			Integer i_1 = new Integer(modFlags);
			rewrite.set(bd, FieldDeclaration.MODIFIERS_PROPERTY, i_1, null_3);
			Integer i_2 = new Integer(modFlags);
			rewrite.set(bd, FieldDeclaration.MODIFIERS_PROPERTY, i_2, null_4);
			TextEdit textEdit = rewrite.rewriteAST(doc, null_5);
			UndoEdit ue_1 = textEdit.apply(doc);
			int int_2 = buffer.getLength();
			java.lang.String s_2 = doc.get();
			int int_1;
			buffer.replace(int_1, int_2, s_2);

1.1.6 วิเคราะห์ผลลัพธ์ที่เกิดขึ้น

การสืบค้นทั้งสี่ประเภทให้รูปแบบการใช้งานอ็อบเจกต์ที่ไม่แตกต่างกัน ให้ผลลัพธ์ที่ครบถ้วน บางคำสั่งที่เกินมาจากที่ต้องการ แต่ส่วนที่เกินมาแม้จะตัดทิ้งออกก็สามารถใช้งานได้ และผลลัพธ์ที่ต้องการอยู่ในอันดับหนึ่งเท่ากัน แสดงว่าการสืบค้นทั้งสี่ประเภทไม่มีผลอะไรกับโจทย์โปรแกรมข้อนี้ แต่ถ้าดูผลลัพธ์ในลำดับถัดไปจะพบว่า ในขณะที่ได้อันดับผลลัพธ์เท่ากัน $\mathrm{UQ_U}$ $\mathrm{UQ_L}$ และ $\mathrm{UQ_A}$ สามารถให้ผลลัพธ์ที่มีจำนวนคำสั่งน้อยกว่าการสืบค้นแบบ $\mathrm{UQ_G}$

1.2 การทดสอบเครื่องมือด้วยตัวอย่างโจทย์การใช้งานอ็อบเจกต์ IWorkingCopyManager

1.2.1 คำอธิบายจากเอกสารเอพีไอ

org.eclipse.ui.preferences.IWorkingCopyManager คืออินเตอร์เฟสที่ใช้สำหรับการเข้าถึงอ็อบเจกต์จำลองที่กำลังทำงานอยู่ของ ICompilationUnit สำหรับอ็อบเจกต์ ICompilationUnit จริงจะต้องเรียกผ่านทางอ็อบเจกต์ IEditorInput เท่านั้น

1.2.2 โจทย์โปรแกรม: ต้องการสืบค้นการใช้งานของอ็อบเจกต์ IWorkingCopyManager

1.2.3 สภาพแวดล้อมในระหว่างการเขียนโปรแกรม

แสดงดังรูปที่ 5.3

```
class IWorkingCopyManagerTask
{
  public void connect(IEditorPart cuEditor)
  {
    IEditorInput editorInput = cuEditor.getEditorInput();
    IWorkingCopyManager manager = JavaUI.getWorkingCopyManager();
                    //////Current Development//////
    manager.disconnect(editorInput);
  }
}
```

รูปที่ 5.3 สภาพแวดล้อมสำหรับตัวอย่างโจทย์การใช้งานอ็อบเจกต์ IWorkingCopyManager

1.2.4 ผลลัพธ์ที่ต้องการ

manager.connect(editorInput);

ICompilationUnit cu = manager.getWorkingCopy(cuEditor.getEditorInput());

List methodDeclarations =

new JavaMethodCollector(cu).getMethodDeclarations();

IBuffer buffer = cu.getBuffer();

MethodDeclaration methodDeclaration = (MethodDeclaration) iterator.next();

addTraceStatement(methodDeclaration, buffer);

cu.reconcile(ICompilationUnit.NO_AST, false, null, null);

โดยรายละเอียดแสดงดังรูปที่ 5.4

```
class IWorkingCopyManagerTask
{
  public void connect(IEditorPart cuEditor)
  {
    IEditorInput editorInput = cuEditor.getEditorInput();
    IWorkingCopyManager manager = JavaUI.getWorkingCopyM
                                                ผลลัพธ์ที่ต้องการ
    manager.connect(editorInput);
    ICompilationUnit cu = manager.getWorkingCopy(cuEditor.getEditorInput());
    List methodDeclarations =
      new JavaMethodCollector(cu).getMethodDeclarations();
    IBuffer buffer = cu.getBuffer();
    MethodDeclaration methodDeclaration = (MethodDeclaration) iterator.next();
    addTraceStatement(methodDeclaration, buffer);
    cu.reconcile(ICompilationUnit.NO_AST, false, null, null);

    manager.disconnect(editorInput);
  }
}
```

รูปที่ 5.4 ผลลัพธ์ที่ต้องการของโจทย์การใช้งานอ็อบเจกต์ IWorkingCopyManager

1.2.5 การเปรียบเทียบผลลัพธ์ที่ได้

การสืบค้นแบบ UQ_G ให้ผลลัพธ์ 17 จำนวน ผลลัพธ์ที่ต้องการอยู่ในลำดับที่ 16 การสืบค้น UQ_U ให้ผลลัพธ์ 16 จำนวน ไม่พบผลลัพธ์ที่ต้องการ การสืบค้น UQ_L ให้ผลลัพธ์ 4 จำนวน ผลลัพธ์ที่ต้องการอยู่ในลำดับที่ 4 และการสืบค้น UQ_A ให้ผลลัพธ์ 16 จำนวน ผลลัพธ์ที่ต้องการอยู่ในลำดับที่ 13 โดยรายละเอียดของผลลัพธ์แสดงดังตารางที่ 5.4

1.2.6 วิเคราะห์ผลลัพธ์ที่เกิดขึ้น

UQ_A สามารถให้ผลลัพธ์ในอันดับที่ดีกว่า UQ_G เนื่องจากการสืบค้นแบบ UQ_A ใช้บริบททั้งด้านบนและด้านล่างเข้าไปช่วยคัดกรองโค้ดก่อนที่จะเข้าสู่การสกัดรูปแบบการใช้งาน ทำให้ผลลัพธ์ ที่ไม่เกี่ยวข้องคัดกรองออกไป อันดับผลลัพธ์จึงดีขึ้น

ตารางที่ 5.4 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ IWorkingCopyManager

ประเภทของการสืบค้น	ลำดับ	จำนวนผลลัพธ์ในคลังข้อมูล	รูปแบบการใช้งานอ็อบเจกต์ที่ได้

UQ_G	16	17	IWorkingCopyManager manager = JavaUI.getWorkingCopyManager();
			IEditorPart cuEditor;
			IEditorInput editorInput = cuEditor.getEditorInput();
			manager.connect(editorInput);
			IEditorInput iei_1 = cuEditor.getEditorInput();
			ICompilationUnit cu = manager.getWorkingCopy(iei_1);
			List methodDeclarations = cu.getMethodDeclarations();
			IBuffer buffer = cu.getBuffer();
			Iterator iterator;
			Object o_1 = iterator.next();
			MethodDeclaration methodDeclaration = (MethodDeclaration) o_1;
			extends com.ibm.jdg2e.jdt.AddTraceStatementsAction;
			addTraceStatement(methodDeclaration, buffer);
			boolean boolean_1;
			CompilationUnit cu_1 = cu.reconcile(ICompilationUnit.NO_AST, boolean_1, null_1, null_2);

ตารางที่ 5.4 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ IWorkingCopyManager (ต่อ)

ประเภทของการสืบค้น	ลำดับ	จำนวนผลลัพธ์ในคลังข้อมูล	รูปแบบการใช้งานอ็อบเจกต์ที่ได้
			manager.disconnect(editorInput);
UQ_U	-	16	-

UQ_L	4	4	IWorkingCopyManager manager = JavaUI.getWorkingCopyManager();
			IEditorPart cuEditor;
			IEditorInput editorInput = cuEditor.getEditorInput();
			manager.connect(editorInput);
			IEditorInput iei_1 = cuEditor.getEditorInput();
			ICompilationUnit cu = manager.getWorkingCopy(iei_1);
			java.util.List methodDeclarations = cu.getMethodDeclarations();
			IBuffer buffer = cu.getBuffer();
			Iterator iterator;
			Object o_1 = iterator.next();
			MethodDeclaration methodDeclaration = (MethodDeclaration) o_1;
			extends com.ibm.jdg2e.jdt.AddTraceStatementsAction;
			addTraceStatement(methodDeclaration, buffer);

ตารางที่ 5.4 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ IWorkingCopyManager (ต่อ)

ประเภทของการสืบค้น	ลำดับ	จำนวนผลลัพธ์ในคลังข้อมูล	รูปแบบการใช้งานอ็อบเจกต์ที่ได้
			boolean boolean_1; CompilationUnit cu_1 = cu.reconcile(org.eclipse.jdt.core.ICompilationUnit.NO_AST, boolean_1, null_1, null_2); manager.disconnect(editorInput);
UQ_A	13	16	IWorkingCopyManager manager = JavaUI.getWorkingCopyManager(); IEditorPart cuEditor; IEditorInput editorInput = cuEditor.getEditorInput(); manager.connect(editorInput); IEditorInput iei_1 = cuEditor.getEditorInput(); ICompilationUnit cu = manager.getWorkingCopy(iei_1); List methodDeclarations = cu.getMethodDeclarations(); IBuffer buffer = cu.getBuffer(); Iterator iterator; Object o_1 = iterator.next(); MethodDeclaration methodDeclaration = (MethodDeclaration) o_1;

ตารางที่ 5.4 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ IWorkingCopyManager (ต่อ)

ประเภทของการสืบค้น	ลำดับ	จำนวนผลลัพธ์ในคลังข้อมูล	รูปแบบการใช้งานอ็อบเจกต์ที่ได้

			extends com.ibm.jdg2e.jdt.AddTraceStatementsAction;
			addTraceStatement(methodDeclaration, buffer);
			boolean boolean_1;
			CompilationUnit cu_1 = cu.reconcile(ICompilationUnit.NO_AST, boolean_1, null_1, null_2);
			manager.disconnect(editorInput);

1.3. การทดสอบเครื่องมือด้วยตัวอย่างโจทย์การใช้งานอ็อบเจกต์ IWorkingCopyManager

1.3.1 คำอธิบายจากเอกสารเอพีไอ: org.eclipse.ui.preferences.IWorkingCopyManager ทำหน้าที่เข้าถึงอ็อบเจกต์จำลองที่กำลังทำงานของ ICompilationUnit สำหรับอ็อบเจกต์ ICompilationUnit จริงจะต้องเรียกผ่านทางอ็อบเจกต์ IEditorInput เท่านั้น

1.3.2 โจทย์โปรแกรม

หารูปแบบการใช้งานอ็อบเจกต์ IWorkingCopyManager

1.3.3 สภาพแวดล้อมในระหว่างการเขียนโปรแกรม

แสดงดังรูปที่ 5.5

```
public ISourceViewer getISourceViewer (JavaEditor editor)
{
   ISelectionProvider selectionProvider = editor.getSelectionProvider();
   IEditorInput iei_1 = editor.getEditorInput();
   IWorkingCopyManager iwm = JavaUI.getWorkingCopyManager();
                        ///////Current Development////////
   ISourceViewer isv_1 = editor.getViewer();
   return isv_1;
   }
}
```

รูปที่ 5.5 สภาพแวดล้อมสำหรับตัวอย่างโจทย์การใช้งานอ็อบเจกต์ IWorkingCopyManager

1.3.4 ผลลัพธ์ที่ต้องการ

```
ISelection selection = selectionProvider.getSelection();
ITextSelection textSelection = (org.eclipse.jface.text.ITextSelection) selection;
ICompilationUnit cu = iwcm_1.getWorkingCopy(iei_1);
QuickTemplateProcessor quickTemp = new QuickTemplateProcessor();
int int_1 = textSelection.getOffset();
int int_2 = textSelection.getLength();
InvocationContext context = new AssistContext(cu, int_1, int_2);
IJavaCompletionProposal[] p = quickTemp.getAssists(context, null);
```

int int_3 = context.getSelectionOffset();

แสดงดังรูปที่ 5.6

```
public ISourceViewer getISourceViewer (JavaEditor editor)
{
  ISelectionProvider selectionProvider = editor.getSelectionProvider();
  IEditorInput iei_1 = editor.getEditorInput();
  IWorkingCopyManager iwm =
    org.eclipse.jdt.ui.JavaUI.getWorkingCopyManager();
                                                    ผลลัพธ์ที่ต้องการ
  ISelection selection = selectionProvider.getSelection();
  ITextSelection textSelection = (org.eclipse.jface.text.ITextSelection) selection;
  ICompilationUnit cu = iwcm_1.getWorkingCopy(iei_1);
  QuickTemplateProcessor quickTemp = new QuickTemplateProcessor();
  int int_1 = textSelection.getOffset();
  int int_2 = textSelection.getLength();
  InvocationContext context = new AssistContext(cu, int_1, int_2);
  IJavaCompletionProposal[] p = quickTemp.getAssists(context, null);
  int int_3 = context.getSelectionOffset();

  ISourceViewer isv_1 = editor.getViewer();
  return isv_1;
}
```

รูปที่ 5.6 ผลลัพธ์ที่ต้องการสำหรับโจทย์การใช้งานอ็อบเจกต์ IWorkingCopyManager

1.3.5 การเปรียบเทียบผลลัพธ์ที่ได้

การสืบค้นแบบ $\mathrm{UQ_G}$ ให้ผลลัพธ์ 17 จำนวน ผลลัพธ์ที่ต้องการอยู่ในลำดับที่ 8 การสืบค้น UQ_{U} ให้ผลลัพธ์ 15 จำนวน ไม่พบผลลัพธ์ที่ต้องการ การสืบค้น $\mathrm{UQ_L}$ ให้ผลลัพธ์ 4 จำนวน ผลลัพธ์ที่ต้องการอยู่ในลำดับที่ 1 และการสืบค้น $\mathrm{UQ_A}$ ให้ผลลัพธ์ 16 จำนวน ผลลัพธ์ที่ต้องการอยู่ในลำดับที่ 3 โดยรายละเอียดของผลลัพธ์แสดงดังตารางที่ 5.5

1.3.6 วิเคราะห์ผลลัพธ์ที่เกิดขึ้น

UQ_A และ UQ_L สามารถให้ผลลัพธ์ในอันดับที่ดีกว่า UQ_G เนื่องจากการสืบค้นแบบ UQ_A และ UQ_L จะใช้บริบททั้งด้านบนและด้านล่างเข้าไปช่วยคัดกรองโค้ดก่อนที่จะเข้าสู่การสกัดรูปแบบการใช้งาน ทำให้มีการคัดกรองผลลัพธ์ที่ไม่เกี่ยวข้องออกไป อันดับผลลัพธ์จึงดีขึ้น

ตารางที่ 5.5 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ IWorkingCopyManager 2

ประเภทของการสืบค้น	ลำดับ	จำนวนผลลัพธ์ในคลังข้อมูล	รูปแบบการใช้งานอ็อบเจกต์ที่ได้

UQ_G	8	17	JavaEditor editor;
			ISelectionProvider selectionProvider = editor.getSelectionProvider();
			ISelection selection = selectionProvider.getSelection();
			ITextSelection textSelection = (ITextSelection) selection;
			IWorkingCopyManager iwcm_1 = JavaUI.getWorkingCopyManager();
			IEditorInput iei_1 = editor.getEditorInput();
			ICompilationUnit cu = iwcm_1.getWorkingCopy(iei_1);
			QuickTemplateProcessor quickTemplateProcessor = new QuickTemplateProcessor();
			int int_1 = textSelection.getOffset();
			int int_2 = textSelection.getLength();
			InvocationContext context = new AssistContext(cu, int_1, int_2);
			IJavaCompletionProposal[] proposals = quickTemplateProcessor.getAssists(context, null_5);
			int int_3 = context.getSelectionOffset();
			ISourceViewer isv_1 = editor.getViewer();
			IAction[] ia_1 = getActionsFromProposals(proposals, int_3, isv_1);

ตารางที่ 5.5 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ IWorkingCopyManager 2 (ต่อ)

ประเภทของการสืบค้น	ลำดับ	จำนวนผลลัพธ์ในคลังข้อมูล	รูปแบบการใช้งานอ็อบเจกต์ที่ได้
UQ_U	-	15	-

UQ_L	1	4	JavaEditor editor;
			ISelectionProvider selectionProvider = editor.getSelectionProvider();
			ISelection selection = selectionProvider.getSelection();
			ITextSelection textSelection = (ITextSelection) selection;
			IWorkingCopyManager iwcm_1 = JavaUI.getWorkingCopyManager();
			IEditorInput iei_1 = editor.getEditorInput();
			ICompilationUnit cu = iwcm_1.getWorkingCopy(iei_1);
			QuickTemplateProcessor quickTemplateProcessor = new QuickTemplateProcessor();
			int int_1 = textSelection.getOffset();
			int int_2 = textSelection.getLength();
			InvocationContext context = new AssistContext(cu, int_1, int_2);
			IJavaCompletionProposal[] proposals = quickTemplateProcessor.getAssists(context, null_5);
			int int_3 = context.getSelectionOffset();
			ISourceViewer isv_1 = editor.getViewer();

ตารางที่ 5.5 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ IWorkingCopyManager 2 (ต่อ)

ประเภทของการสืบค้น	ลำดับ	จำนวนผลลัพธ์ในคลังข้อมูล	รูปแบบการใช้งานอ็อบเจกต์ที่ได้
			IAction[] ia_1 = getActionsFromProposals(proposals, int_3, isv_1);

UQ_A	3	16	JavaEditor editor;
			ISelectionProvider selectionProvider = editor.getSelectionProvider();
			ISelection selection = selectionProvider.getSelection();
			ITextSelection textSelection = (ITextSelection) selection;
			IWorkingCopyManager iwcm_1 = JavaUI.getWorkingCopyManager();
			IEditorInput iei_1 = editor.getEditorInput();
			ICompilationUnit cu = iwcm_1.getWorkingCopy(iei_1);
			QuickTemplateProcessor quickTemplateProcessor = new QuickTemplateProcessor();
			int int_1 = textSelection.getOffset();
			int int_2 = textSelection.getLength();
			InvocationContext context = new AssistContext(cu, int_1, int_2);
			IJavaCompletionProposal[] proposals = quickTemplateProcessor.getAssists(context, null_5);
			int int_3 = context.getSelectionOffset();
			ISourceViewer isv_1 = editor.getViewer();

1.4 การทดสอบเครื่องมือด้วยตัวอย่างโจทย์การใช้งานอ็อบเจกต์ ITextEditor

1.4.1 คำอธิบายจากเอกสารเอพีไอ

คลาส org.eclipse.ui.texteditor.ITextEditor ทำหน้าที่ส่งคำสั่งที่เกิดขึ้นบนส่วนการแก้ไขตัวอักษร (Text Editor) ให้แก่ส่วนให้บริการเอกสาร (Document Provider) เช่น คำสั่งบันทึก คำสั่งลบ เป็นต้น

1.4.2 โจทย์โปรแกรม

หารูปแบบการใช้งานของอ็อบเจกต์ ITextEditor เพื่อสั่งส่วนบริการเอกสารให้ทำงานตามต้องการ

1.4.3 สภาพแวดล้อมในระหว่างการเขียนโปรแกรม

แสดงดังรูปที่ 5.7

1.4.4 ผลลัพธ์ที่ต้องการ

```
IDocumentProvider provider = textEditor.getDocumentProvider();
provider.connect(input);
IDocument document = provider.getDocument(input);
int fFileLineNumber;
IRegion region= document.getLineInformation(fFileLineNumber - 1);
int fFileOffset = region.getOffset();
int fFileLength = region.getLength();
provider.disconnect(input);
```

โดยรายละเอียดแสดงดังรูปที่ 5.8

1.4.5 การเปรียบเทียบผลลัพธ์ที่ได้

การสืบค้นแบบ UQ_{G} ให้ผลลัพธ์ 130 จำนวน ผลลัพธ์ที่ต้องการอยู่ในลำดับที่ 24 การสืบค้น UQ_{U} ให้ผลลัพธ์ 48 จำนวน ผลลัพธ์ที่ต้องการอยู่ในลำดับที่ 48 การสืบค้น UQ_{L} ให้ผลลัพธ์ 45 จำนวน ผลลัพธ์ที่ต้องการอยู่ในลำดับที่ 14 และการสืบค้น UQ_{A} ให้ผลลัพธ์ 35 จำนวน ผลลัพธ์ที่ต้องการอยู่ในลำดับที่ 8 โดยรายละเอียดของผลลัพธ์แสดงดังตารางที่ 5.6

1.4.6 วิเคราะห์ผลลัพธ์ที่เกิดขึ้น

การสืบค้นแบบ $\mathrm{UQ_G}$ จะให้ผลลัพธ์ที่ยาวกว่าการสืบค้นรูปแบบอื่นๆ เนื่องจากไม่ได้มีการนำบริบทเข้าไปช่วยตัดทอนโค้ด และการสืบค้นแบบ $\mathrm{UQ_A}$ จะสามารถจัดอันดับผลลัพธ์ได้ดีกว่าการสืบค้นรูปแบบอื่นๆ เนื่องจากมีการใช้บริบทโค้ดทั้งแบบบนและแบบล่างเข้าไปคัดกรองโค้ดก่อนเข้าการสกัดรูปแบบการใช้งาน ในขณะที่ $\mathrm{UQ_U}$ และ $\mathrm{UQ_L}$ จะนำบริบทเฉพาะแบบบนและแบบล่างตามลำดับ เข้าไปช่วยคัดกรองเท่านั้น

```
public void itextEditorSelect (IEditorPart editorPart)
{
  int fFileOffset,fFileLength;
  IEditorInput input = editorPart.getEditorInput();
  ITextEditor textEditor;
                    /////////Current Development/////////////
  textEditor.selectAndReveal(fFileOffset, fFileLength);
}
```

รูปที่ 5.7 สภาพแวดล้อมสำหรับตัวอย่างโจทย์การใช้งานอ็อบเจกต์ ITextEditor

```
public void itextEditorSelect (IEditorPart editorPart)
{
   int fFileOffset,fFileLength;
   IEditorInput input = editorPart.getEditorInput();
   ITextEditor textEditor = null;

   IDocumentProvider provider = textEditor.getDocumentProvider
   provider.connect(input);
   IDocument document = provider.getDocument(input);
   int fFileLineNumber;
   IRegion region= document.getLineInformation(fFileLineNumber - 1);
   int fFileOffset = region.getOffset();
   int fFileLength = region.getLength();
   provider.disconnect(input);

   textEditor.selectAndReveal(fFileOffset, fFileLength);
}
```

ผลลัพธ์ที่ต้องการ

รูปที่ 5.8 ผลลัพธ์ที่ต้องการสำหรับตัวอย่างโจทย์การใช้งานอ็อบเจกต์ ITextEditor

ตารางที่ 5.6 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ ITextEditor

ประเภทของการสืบค้น	ลำดับ	จำนวนผลลัพธ์ในคลังข้อมูล	รูปแบบการใช้งานอ็อบเจกต์ที่ได้
UQ_G	24	130	IWorkbenchWindow window = DebugUIPlugin.getActiveWorkbenchWindow(); IWorkbenchPage page = window.getActivePage(); IFile fFile; FileEditorInput fei_1 = new org.eclipse.ui.part.FileEditorInput(fFile); String s_1 = getEditorId(); boolean boolean_1; IEditorPart editorPart = page.openEditor(fei_1, s_1, boolean_1); ITextEditor textEditor = null; ITextEditor textEditor = (ITextEditor) editorPart; Object o_1 = editorPart.getAdapter(ITextEditor.class); ITextEditor textEditor = (ITextEditor) o_1; IEditorInput input = editorPart.getEditorInput(); IDocumentProvider provider = textEditor.getDocumentProvider(); provider.connect(input); IDocument document = provider.getDocument(input);

ตารางที่ 5.6 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ ITextEditor (ต่อ)

ประเภทของการสืบค้น	ลำดับ	จำนวนผลลัพธ์ในคลังข้อมูล	รูปแบบการใช้งานอ็อบเจกต์ที่ได้

			int int_1;
			IRegion region = document.getLineInformation(int_1);
			int fFileOffset = region.getOffset();
			int fFileLength = region.getLength();
			provider.disconnect(input);
			textEditor.selectAndReveal(fFileOffset, fFileLength);
			FileEditorInput fei_1;
			String s_1;
			boolean boolean_1;

UQ_U	10	48	IWorkbenchWindow window = DebugUIPlugin.getActiveWorkbenchWindow(); IWorkbenchPage page = window.getActivePage(); IFile fFile; FileEditorInput fei_1 = new org.eclipse.ui.part.FileEditorInput(fFile); String s_1 = getEditorId();

ตารางที่ 5.6 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ ITextEditor (ต่อ)

ประเภทของการสืบค้น	ลำดับ	จำนวนผลลัพธ์ในคลังข้อมูล	รูปแบบการใช้งานอ็อบเจกต์ที่ได้

			boolean boolean_1;
			IEditorPart editorPart = page.openEditor(fei_1, s_1, boolean_1);
			ITextEditor textEditor = null;
			ITextEditor textEditor = (org.eclipse.ui.texteditor.ITextEditor) editorPart;
			Object o_1 = editorPart.getAdapter(ITextEditor.class);
			ITextEditor textEditor = (ITextEditor) o_1;
			IEditorInput input = editorPart.getEditorInput();
			IDocumentProvider provider = textEditor.getDocumentProvider();
			provider.connect(input);
			IDocument document = provider.getDocument(input);
			int int_1;
			IRegion region = document.getLineInformation(int_1);
			int fFileOffset = region.getOffset();
			int fFileLength = region.getLength();
			provider.disconnect(input);

ตารางที่ 5.6 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ ITextEditor (ต่อ)

ประเภทของการสืบค้น	ลำดับ	จำนวนผลลัพธ์ในคลังข้อมูล	รูปแบบการใช้งานอ็อบเจกต์ที่ได้
			textEditor.selectAndReveal(fFileOffset, fFileLength);

UQ_L	14	45	IWorkbenchWindow window = DebugUIPlugin.getActiveWorkbenchWindow();
			IWorkbenchPage page = window.getActivePage();
			IFile fFile;
			FileEditorInput fei_1 = new org.eclipse.ui.part.FileEditorInput(fFile);
			String s_1 = getEditorId();
			boolean boolean_1;
			IEditorPart editorPart = page.openEditor(fei_1, s_1, boolean_1);
			ITextEditor textEditor = (org.eclipse.ui.texteditor.ITextEditor) editorPart;
			java.lang.Object o_1 = editorPart.getAdapter(ITextEditor.class);
			ITextEditor textEditor = (ITextEditor) o_1;
			IEditorInput input = editorPart.getEditorInput();
			IDocumentProvider provider = textEditor.getDocumentProvider();
			provider.connect(input);
			IDocument document = provider.getDocument(input);

ตารางที่ 5.6 การเปรียบเทียบผลลัพธ์ที่ได้ของอ็อบเจกต์ ITextEditor (ต่อ)

ประเภทของการสืบค้น	ลำดับ	จำนวนผลลัพธ์ในคลังข้อมูล	รูปแบบการใช้งานอ็อบเจกต์ที่ได้

			int int_1;
			IRegion region = document.getLineInformation(int_1);
			int fFileOffset = region.getOffset();
			int fFileLength = region.getLength();
			provider.disconnect(input);
			textEditor.selectAndReveal(fFileOffset, fFileLength);

UQ_A	8	35	IWorkbenchWindow window = DebugUIPlugin.getActiveWorkbenchWindow(); IWorkbenchPage page = window.getActivePage(); IFile fFile; FileEditorInput fei_1 = new FileEditorInput(fFile); java.lang.String s_1 = getEditorId(); boolean boolean_1; IEditorPart editorPart = page.openEditor(fei_1, s_1, boolean_1); ITextEditor textEditor = null; ITextEditor textEditor = (ITextEditor) editorPart;

			java.lang.Object o_1 = editorPart.getAdapter(ITextEditor.class);
			ITextEditor textEditor = (ITextEditor) o_1;
			IEditorInput input = editorPart.getEditorInput();
			IDocumentProvider provider = textEditor.getDocumentProvider();
			provider.connect(input);
			IDocument document = provider.getDocument(input);int int_1;
			IRegion region = document.getLineInformation(int_1);
			int fFileOffset = region.getOffset();
			int fFileLength = region.getLength(); provider.disconnect(input);
			textEditor.selectAndReveal(fFileOffset, fFileLength);

2. การทดสอบเครื่องมือ

สำหรับการทดสอบเครื่องมือ จัดแบ่งออกเป็น 2 ส่วนด้วยกัน ได้แก่ การเปรียบเทียบประสิทธิภาพของการสกัดรูปแบบการใช้งานในการสืบค้นแต่ละประเภท และการเปรียบเทียบการจัดอันดับผลลัพธ์ของการสืบค้นแต่ละประเภท

2.1 การเปรียบเทียบประสิทธิภาพของการสกัดรูปแบบการใช้งานในการสืบค้นแต่ละประเภท

การทดลองนี้ใช้โจทย์โปรแกรมทั้งหมด 19 โจทย์ โดยประสิทธิภาพของการสกัดรูปแบบการใช้งานจะวัดโดยการนำผลลัพธ์ที่ได้จาก XSNIPPETUSAGE มาคอมไพล์และรันโปรแกรมในแต่ละโจทย์ และโปรแกรมนั้นจะต้องสามารถทำงานได้ตามจุดประสงค์ที่วางไว้ได้

รูปที่ 5.9 แสดงร้อยละของจำนวนโจทย์ที่การสืบค้นแต่ละประเภทสามารถหาคำตอบได้ โดยที่

- แกน x แสดงถึงคลาสไลบรารีต่างๆ ที่โจทย์ใช้ในการทดสอบ
- แกน y แสดงถึงร้อยละของจำนวนโจทย์ที่การสืบค้นสามารถหาคำตอบได้

จากกราฟจะเห็นว่าการสืบค้นข้อมูลประเภท $\mathrm{UQ_G}$ และ $\mathrm{UQ_A}$ ได้ผลร้อยละ 100 นั่นหมายความว่า $\mathrm{UQ_G}$ สามารถตอบโจทย์ทั้ง 19 โจทย์ได้ครบถ้วนสมบูรณ์ ในขณะที่ $\mathrm{UQ_L}$ และ $\mathrm{UQ_U}$ สามารถตอบโจทย์ได้เพียง 17 โจทย์ (89%) เนื่องจากไม่มีซอร์สโค้ดใดที่บริบทโค้ดแบบ $\mathrm{C_{ML}}$ ของโค้ดที่กำลังพัฒนาตรงกับบริบทโค้ดในคลังข้อมูล

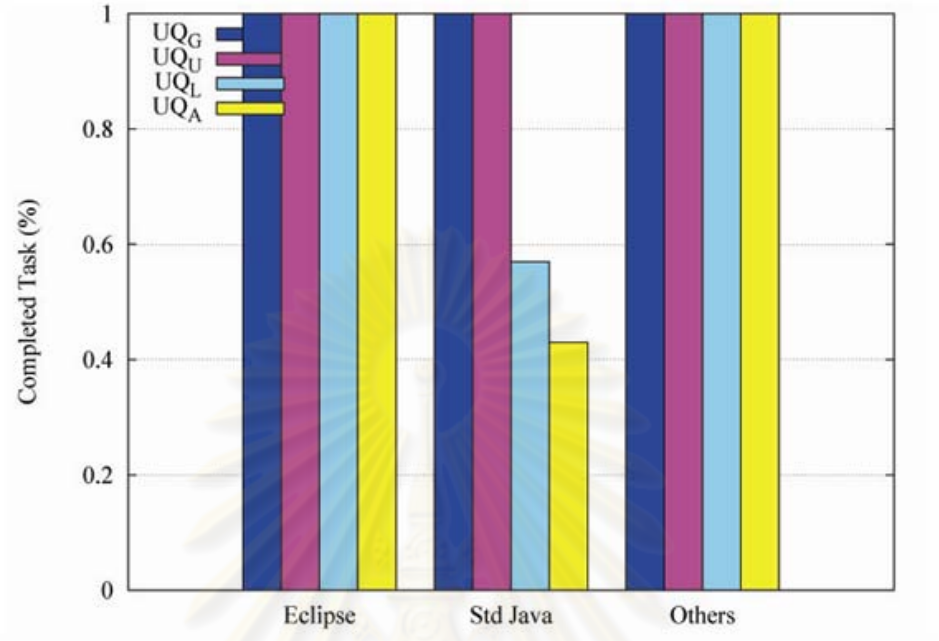
อย่างไรก็ตาม ซึ่งพบว่าปัญหานี้หากพิจารณาความสัมพันธ์สืบทอด (Inheritance Relationship) ของบริบทโค้ดจะสามารถเรียกข้อมูลโค้ดขึ้นมาจากคลังได้ ซึ่งจะพัฒนาต่อไปในอนาคต

2.2 การเปรียบเทียบการจัดอันดับผลลัพธ์ของการสืบค้นแต่ละประเภท

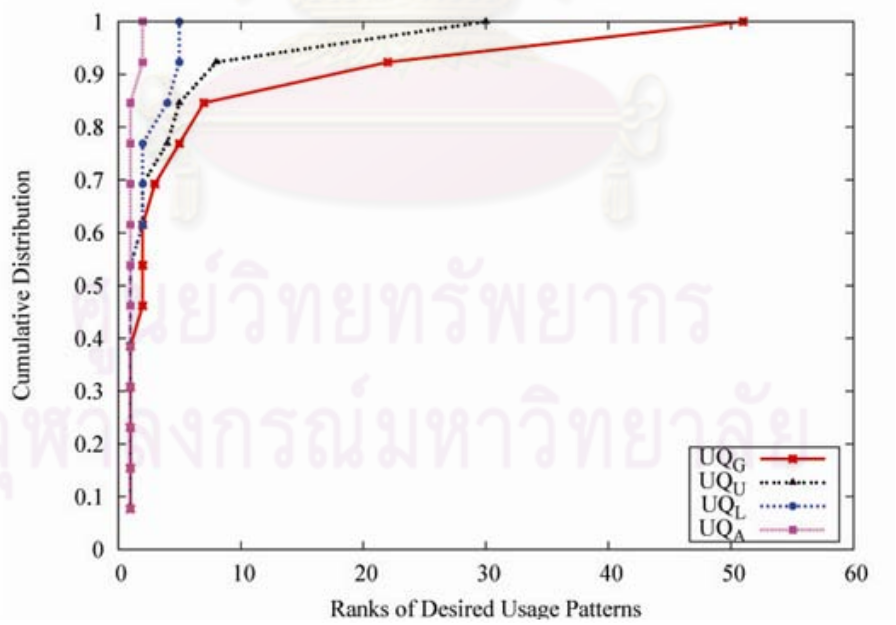
การทดลองที่สองจะวัดค่าการจัดอันดับผลลัพธ์ที่ได้จาก XSNIPPETUSAGE โดยนับจากอันดับแรกสุดที่ผลลัพธ์สามารถรันโปรแกรมได้ตรงตามวัตถุประสงค์

รูปที่ 5.10 แสดงการแจกแจงความถี่สะสมของอันดับของผลลัพธ์ที่เป็นคำตอบที่ถูกต้องและครบถ้วนที่สุดสำหรับการสืบค้นแต่ละประเภท จะเห็นได้ว่า $\mathrm{UQ_A}$ สามารถให้คำตอบใน 3 อันดับแรกทั้งหมด ในขณะที่ $\mathrm{UQ_L}$ ให้คำตอบใน 5 อันดับแรกและ $\mathrm{UQ_G}$ มีประสิทธิภาพน้อยที่สุด โดยมีเพียงร้อยละ 79 ที่ให้คำตอบที่ถูกต้องและครบภายใน 10 อันดับแรก ที่เป็นเช่นนี้เพราะในขั้นตอนการคัดเลือกซอร์สโค้ด การสืบค้นแบบ $\mathrm{UQ_G}$, $\mathrm{UQ_L}$ และ $\mathrm{UQ_A}$ ได้กรองเฉพาะรูปแบบการใช้งานอ็อบเจกต์ในคลังข้อมูลที่มีบริบทโค้ดที่ตรงกับบริบทโค้ดในโค้ดที่กำลังพัฒนา ในขณะที่ $\mathrm{UQ_G}$

ไม่ได้กรองผลลัพธ์ที่บริบทโค้ดไม่ตรงกันออก เพียงแต่เลือกเฉพาะผลลัพธ์ที่มีการใช้งานอ็อบเจกต์ที่ต้องการเป็นส่วนประกอบเท่านั้น



รูปที่ 5.9 ร้อยละของจำนวนโจทย์ที่การสืบค้นแต่ละประเภทสามารถหาคำตอบได้



รูปที่ 5.10 การแจกแจงแบบสะสมของอันดับผลลัพธ์ที่เหมาะสมที่สุด

2.3 การเปรียบเทียบกับ MAPO

การทดลองสุดท้ายเป็นการเปรียบเทียบเครื่องมือ XSnippetUsage กับ MAPO เนื่องจากงานวิจัยอื่นไม่ได้เปิดให้ดาวน์โหลดเครื่องมือ โดยคลังข้อมูลโค้ดและโจทย์โปรแกรมที่ใช้ทดสอบ MAPO เป็นชุดเดียวกับที่ใช้ทดสอบ XSnippetUsage

เมื่อทดสอบด้วยค่ารีเลทีฟซับพอร์ต 0.5 ซึ่งในงานวิจัยบอกว่าเป็นค่าที่เหมาะสมที่สุด พบว่า MAPO สามารถให้ผลลัพธ์หลายผลลัพธ์ แต่ไม่มีผลลัพธ์ใดที่สามารถนำไปตอบโจทย์ได้เลย เมื่อใช้รีเลชันซัพพอร์ตเท่ากับ 0.1 ก็พบว่ายังไม่มีผลลัพธ์ใดที่สามารถตอบโจทย์ได้ เมื่อลองลดค่ารีเลชันซัพพอร์ตลงอีกก็พบว่าใช้เวลาทำเหมืองข้อมูลโค้ดนานมาก จนไม่สามารถทำการทดลองต่อได้ รายละเอียดของการทดลองแสดงดังรูปที่ 5.11

Relative Supports	Results	Process Times
0.0002	unknown	gave up after 3 days
0.001	unknown	gave up after 1 day
0.1	still not useful	3 hours
0.5	not useful	5 mins

รูปที่ 5.11 ผลลัพธ์จากการทดลองเครื่องมือ MAPO

บทที่ 6
สรุปผลการวิจัย

จากการศึกษาการสกัดรูปแบบการใช้งานของอ็อบเจกต์โดยใช้การสืบค้นที่ขึ้นกับบริบทหลากหลายรูปแบบ สามารถสรุปผลการวิจัย ข้อจำกัดของเครื่องมือ และแนวทางในการพัฒนาได้ต่อได้ดังนี้

1. สรุปผล

วิทยานิพนธ์นี้นำเสนอกระบวนการสกัดรูปแบบการใช้งานอ็อบเจกต์โดยใช้การสืบค้นที่ขึ้นกับบริบทหลากหลายรูปแบบ ซึ่งจะช่วยให้นักพัฒนาซอฟต์แวร์สามารถนำรูปแบบการใช้งานที่ได้ไปพัฒนาซอฟต์แวร์ให้มีประสิทธิภาพและรวดเร็วยิ่งขึ้น โดยขั้นตอนการสกัดรูปแบบการใช้งานอ็อบเจกต์จะเริ่มจากการคัดเลือกแบบจำลองโค้ดที่เกี่ยวข้องกับรูปแบบการใช้งานของอ็อบเจกต์ที่ต้อง การจากคลังข้อมูล แบบจำลองโค้ดที่ได้จะนำไปสกัดรูปแบบการใช้งานอ็อบเจกต์ คัดแยกรูปแบบการใช้งานที่ซ้ำซ้อนออก นำไปจัดอันดับผลลัพธ์ ก่อนที่จะแปลงให้อยู่ในรูปแบบโค้ดที่สามารถนำไปใช้งานได้ทันที

ประโยชน์ที่ได้รับจากวิทยานิพนธ์นี้คือ วิธีการในการสกัดรูปแบบการใช้งานอ็อบเจกต์ เพื่อให้ผู้ใช้ลดภาระในการค้นหาตัวอย่างการใช้งานของอ็อบเจกต์ด้วยตนเอง ซึ่งจะใช้เวลามากและอาจจะไม่ได้ผลลัพธ์ตามต้องการ และช่วยสนับสนุนการนำโค้ดกลับมาใช้ใหม่ เนื่องจากผู้ใช้สามารถใช้งานไลบรารีหรือกรอบงานจากโอเพนซอร์สภายนอกได้สะดวกมากขึ้น นอกจากนี้ยังได้เครื่องมือสนับสนุนการสกัดรูปแบบการใช้งานของอ็อบเจกต์ และสามารถจัดลำดับความสำคัญของผลลัพธ์ที่ต้องการได้

จากผลการทดลองพบว่า การสืบค้นแบบ $\mathrm{UQ_G}$ สามารถให้ผลลัพธ์ที่มีความสมบูรณ์ครบถ้วนมากกว่า ในขณะที่การสืบค้นแบบ $\mathrm{UQ_U}$ $\mathrm{UQ_L}$ และ $\mathrm{UQ_A}$ สามารถจัดอันดับผลลัพธ์ได้ดีกว่าการสืบค้นแบบ $\mathrm{UQ_G}$

2. ข้อจำกัดของเครื่องมือและแนวทางการพัฒนาต่อ

2.1. สำหรับขั้นตอนการสกัดรูปแบบการใช้งานโค้ดจะวิเคราะห์โค้ดแบบสถิต (Static Analysis) เท่านั้น ไม่สามารถวิเคราะห์โค้ดแบบไดนามิก (Dynamic Analysis) ได้ ซึ่งวิทยานิพนธ์นี้สามารถปรับเปลี่ยนไปใช้วิธีวิเคราะห์โค้ดแบบไดนามิกแทนได้

2.2. สำหรับขั้นตอนการสกัดรูปแบบการใช้งานโค้ดไม่ได้คำนึงถึงโครงสร้างแบบกระแสควบคุม (Control Flow Structure e.g., if-else) และแบบวนซ้ำ (Iteration e.g.; while, for) ซึ่งทำให้รูปแบบการใช้งานอ็อบเจกต์ไม่ครบถ้วนสมบูรณ์ ดังนั้นจึงควรพัฒนาส่วนนี้เพิ่มเติม

2.3. สำหรับขั้นตอนการเปรียบเทียบบริบทโค้ดไม่ได้คำนึงถึงความสัมพันธ์แบบสืบทอด (Inheritance) ซึ่งทำให้การสกัดรูปแบบการใช้งานผิดพลาด หรือไม่ครบถ้วนสมบูรณ์ในบางกรณี ดังนั้นจึงควรคำนึงถึงความสัมพันธ์แบบสืบทอดลงในขั้นตอนการเปรียบเทียบบริบทโค้ดด้วย

3. ผลงานที่เกี่ยวข้องกับวิทยานิพนธ์

วิทยานิพนธ์นี้ได้รับคัดเลือกตีพิมพ์ในการประชุมวิชาการนานาชาติร่วมสาขาวิทยาการคอมพิวเตอร์และวิศวกรรมซอฟต์แวร์ ครั้งที่ 7 (JCSSE: Internation Joint Conference on Computer Science and Software Engineering 2010) ระหว่างวันที่ 12-14 พฤษภาคม 2553 ณ ห้องประชุม 321 ชั้น 3 อาคารสุโขทัย มหาวิทยาลัยรามคำแหง

รายการอ้างอิง

[1] Reid, H. Approximate Structural Context Matching: An Approach to Recommend Relevant Examples. IEEE Trans. Softw. Eng., 2006.

[2] Suresh, T., and Tao, X. Parseweb: a programmer assistant for reusing open source code on the web. Proc. Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, Atlanta, Georgia, USA 2007.

[3] David, M., Lin, X., Rastislav, B., and Doug, K.: Jungloid mining: helping to navigate the API jungle. Proc. Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation, Chicago, IL, USA2005.

[4] Naiyana, S., and Kajal, C. XSnippet: mining for sample code. Proc. Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications, Portland, Oregon, USA 2006.

[5] Tao, X., and Jian, P. MAPO: mining API usages from open source repositories. Proc. Proceedings of the 2006 international workshop on Mining software repositories, Shanghai, China 2006.

[6] Mithun, A., Tao, X., Jian, P., and Jun, X. Mining API patterns as partial orders from source code: from usage scenarios to specifications. Proc. Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, Dubrovnik, Croatia 2007.

[7] Dapeng, L., Andrian, M., Denys, P., and Vaclav, R. Feature location via information retrieval based filtering of a single scenario execution trace. Proc. Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, Atlanta, Georgia, USA 2007.

[8] Davide, L., Leonardo, M., and Mauro, P. Inferring state-based behavior models. Proc. Proceedings of the 2006 international workshop on Dynamic systems analysis, Shanghai, China 2006.

[9] Jochen, Q., and Rainer, K. Dynamic Object Process Graphs. Proc. Proceedings of the Conference on Software Maintenance and Reengineering 2006.

[10] Tao, X., Evan, M., and Hai, Y. Automatic extraction of abstract-object-state machines from unit-test executions. Proc. Proceedings of the 28th international conference on Software engineering, Shanghai, China 2006.

[11] Valentin, D., Christian, L., Andrzej, W., and Andreas, Z. Mining object behavior with ADABU. Proc. Proceedings of the 2006 international workshop on Dynamic systems analysis, Shanghai, China 2006.

[12] Annie, T.T.Y., Raymond, N., and Mark, C.C.-C. Predicting Source Code Changes by Mining Change History, IEEE Trans. Softw. Eng., 2004.

[13] Huzefa, K. Improving change prediction with fine-grained source code mining. Proc. Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, Atlanta, Georgia, USA2007.

[14] Mina, A., and Ric, H. Information theoretic evaluation of change prediction models for large-scale software. Proc. Proceedings of the 2006 international workshop on Mining software repositories, Shanghai, China2006.

[15] Peter, W., and Stephan, D. Mining Version Histories to Guide Software Changes, IEEE Trans. Softw. Eng., 2005.

[16] Sunghun, K., Kai, P., and E. E. James Whitehead, Jr. Memories of bug fixes. Proc. Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering, Portland, Oregon, USA 2006.

[17] Andrzej, W., Andreas, Z., and Christian, L. Detecting object usage anomalies. Proc. Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, Dubrovnik, Croatia 2007.

[18] Jinlin, Y., David, E., Deepali, B., Thirumalesh, B., and Manuvir, D. Perracotta: mining temporal API rules from imperfect traces. Proc. Proceedings of the 28th international conference on Software engineering, Shanghai, China 2006.

[19] Murali Krishna, R., Ananth, G., and Suresh, J. Path-Sensitive Inference of Function Precedence Protocols. Proc. Proceedings of the 29th international conference on Software Engineering 2007.

[20] Stephen, F., Eran, Y., Nurit, D., Ramalingam, G., and Emmanuel, G. Effective typestate verification in the presence of aliasing. Proc. Proceedings of the 2006 international symposium on Software testing and analysis, Portland, Maine, USA 2006.

[21] William, D., David, L., and Andy, P. Finding failures by cluster analysis of execution profiles. Proc. Proceedings of the 23rd International Conference on Software Engineering, Toronto, Ontario, Canada 2001.

[22] Livshits, B., and Zimmermann, T. DynaMine: finding common error patterns by mining software revision histories, SIGSOFT Softw. Eng. Notes, 2005.

[23] Renuka, S. Using an information retrieval system to retrieve source code samples. Proc. Proceedings of the 28th international conference on Software engineering, Shanghai, China 2006.

[24] Wei, Z., Lu, Z., Yin, L., Jiasu, S., and Fuqing, Y. SNIAFL: Towards a static noninteractive approach to feature location, ACM Trans. Softw. Eng. Methodol., 2006.

[25] Huzefa, K., Michael, L.C., and Jonathan, I.M. A survey and taxonomy of approaches for mining software repositories in the context of software evolution, J. Softw. Maint. Evol., 2007.

[26] Andrian, M., Vaclav, R., Joseph, B., Maksym, P., and Andrey, S. Static Techniques for Concept Location in Object-Oriented Code. Proc. Proceedings of the 13th International Workshop on Program Comprehension 2005.

[27] Claypool, N.T.a.K.T. Finding a Needle in the Haystack: A Technique for Ranking Matches between Components. Proc. Proceedings of the 8th International SIGSOFT Symposium on Component-based Software Engineering (CBSE 2005): Software Components at WorkMay 2005.

ภาคผนวก

ภาคผนวก ก
แบบจำลองโค้ดที่ใช้ในวิทยานิพนธ์

ภาคผนวก ก
แบบจำลองโค้ดที่ใช้ในวิทยานิพนธ์

แบบจำลองที่อธิบายในหัวข้อนี้และใช้ตลอดวิทยานิพนธ์นำมาจาก [4] เนื่องจากโค้ดในรูปแบบตัวอักษรมีข้อจำกัดในการวิเคราะห์และการสกัดรูปแบบการใช้งานของอ็อบเจกต์ จึงจำเป็นต้องแปลงโค้ดในรูปแบบตัวอักษรให้เป็นแบบจำลองโค้ด ยกตัวอย่างเช่น

รูปที่ ก.1 ลำดับเมท็อดอินโวเคชันของเมท็อด showPerspective() ในคลาส ReviewPerspectiveFactory ของรูปที่ ก.2 ซึ่งอ็อบเจกต์ประเภท IWorkbenceWindow จะสร้างขึ้นเมื่อเรียกใช้งานเมท็อด getWorkbenchWindow() อย่างไรก็ตามจะเห็นว่าอ็อบเจกต์ประเภท IWorkbenceWindow ไม่ปรากฏอยู่ในลำดับเมท็อดอินโวเคชันในรูปที่ ก.2 จะเห็นว่าการวิเคราะห์โค้ดจากตัวอักษรจึงไม่เพียงพอที่จะทราบข้อมูลที่ซ่อนไว้ได้ แบบจำลองโค้ดที่จะอธิบายดังต่อไปนี้จะช่วยให้มองเห็นลำดับของโค้ดที่ซ่อนไว้ได้ชัดเจนมากขึ้น

```
getViewSite().getWorkbenchWindow().getSelectionService()
```

รูปที่ ก.1 ลำดับเมท็อดอินโวเคชันของเมท็อด showPerspective()
ในคลาส ReviewPerspectiveFactory

แบบจำลองโค้ด (Source Code Model: CM) นิยามในรูปแบบของกราฟไดเร็กอไซคลิก (Direct Acyclic Graph) โดยที่ $CM = (N_{CM}, E_{CM})$ โหนด N_{CM} และเอดจ์ E_{CM} ที่นิยามในแบบจำลองโค้ดเลียนแบบโครงสร้างและพฤติกรรมมาจากไวยกรณ์ของคลาส และใช้แบบจำลองโค้ดที่นิยามขึ้นเพื่ออธิบายการสกัดรูปแบบการใช้งานของอ็อบเจกต์อย่างเป็นทางการ แต่ละอินสแตนท์ (Instance) ของแบบจำลองโค้ด cm_i ($cm_i \in CM$) เปรียบเทียบได้กับแต่ละไฟล์ของคลาส

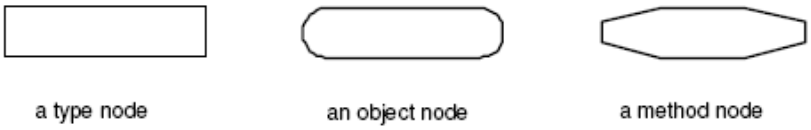
ในหัวข้อนี้จะอธิบายประเภทของโหนด N_{CM} และเอดจ์ E_{CM} ในแบบจำลองโค้ด CM ควบคู่ไปกับการแปลงคลาสของจาวา C_s ให้กลายเป็นอินสแตนท์ C_{ms} และแสดงวิธีการสกัดรูปแบบการใช้งานของอ็อบเจกต์ที่อยู่ในแบบจำลองโค้ดให้กลายเป็นโค้ด

```
public class JavaMetricsView extends ViewPart
  implements ISelectionListener, IJavaMetricsListener
{
  Text message;
  JavaMetrics jm;
  public void createPartControl(Composite parent)
  {
    parent.setLayout(new FillLayout());
    message = new Text(parent, SWT.MULTI);
    message.setText(NO SELECTION MESSAGE);
    getViewSite().getWorkbenchWindow().getSelectionService().
      addSelectionListener(this);
    jm = new JavaMetrics();
    jm.addListener(this);
  }

  public void setFocus() {..}

  public void selectionChanged (IWorkbenchPart part, ISelection selection)
  {
    ICompilationUnit cu;
  }
}
```

รูปที่ ก.2 คลาส JavaMetricView แสดงการทำงานของเมท็อด createPartControl()

1. แบบจำลองโหนด (Node Model)

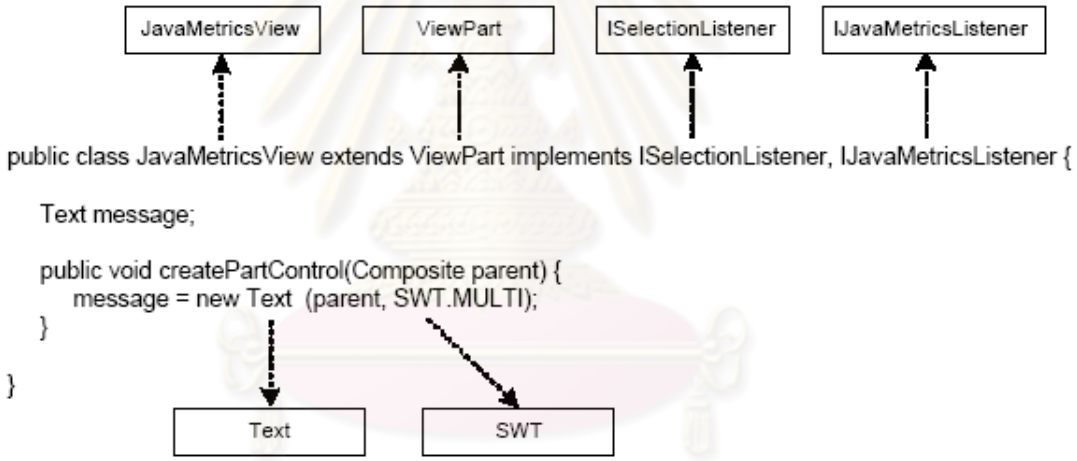
ประเภทของโหนดมีอยู่สามประเภทได้แก่ โหนดชนิดข้อมูล (Type Node) โหนดอ็อบเจกต์ (Object Node) โหนดเมท็อด (Method Node) สัญลักษณ์ที่ใช้แทนโหนดแสดงดังรูปที่ ก.3



รูปที่ ก.3 ประเภทของแบบจำลองโหนด

1.1. โหนดชนิดข้อมูล เช่น คลาส ซุปเปอร์คลาส อินเตอร์เฟส

รูปที่ ก.4 คลาส JavaMetricsView ซุปเปอร์คลาส ViewPart อินเตอร์เฟส ISelectionListener และ IJavaMetricsListener แปลงให้กลายเป็นโหนดชนิดข้อมูลชื่อว่า ‘JavaMetricsView’ ‘ViewPart’ ‘ISelectionListener’ และ ‘IJavaMetricsListener’ ตามลำดับ นอกเหนือจากนี้ ชนิดข้อมูล Text ที่เรียกโดยคอนสตักเตอร์ของตัวเอง และชนิดข้อมูล SWT ที่เป็น static field ของ MULTI และเป็นพารามิเตอร์ของคอนสตักเตอร์ Text ก็คือโหนดชนิดข้อมูลเช่นกัน



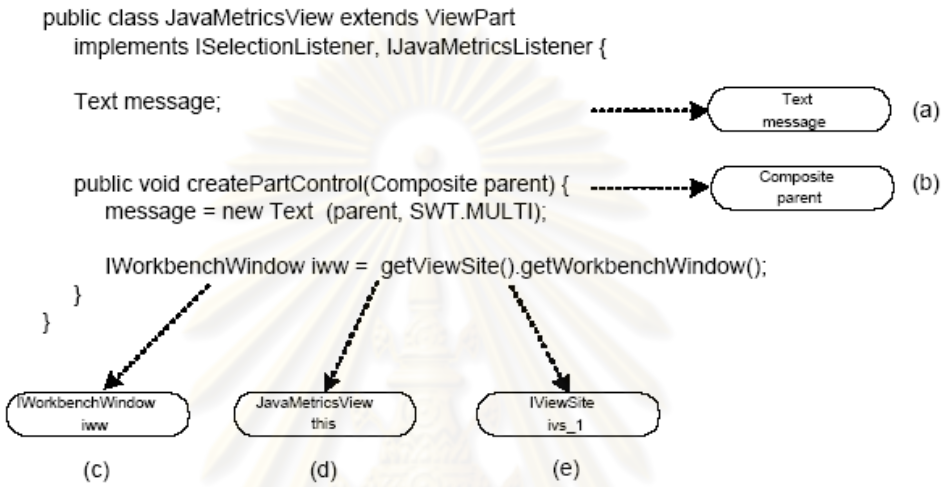
รูปที่ ก.4 แสดงการแปลงจากโค้ดไปเป็นแบบโหนดชนิดข้อมูล

1.2. โหนดอ็อบเจกต์ ได้แก่ แอททริบิวต์ของคลาส ตัวแปรที่ประกาศไว้ภายในเมท็อด

รูปที่ ก.5 แอททริบิวต์ Text message จะถูกแปลงให้เป็นโหนดอ็อบเจกต์ โดยมีชนิดข้อมูลชื่อว่า Text และชื่ออ็อบเจกต์ว่า message ดังรูปที่ ก.5 (a) เช่นเดียวกับพารามิเตอร์ของเมท็อด ‘Composite parent’ ดังรูปที่ ก.5 (b) และตัวแปรที่ประกาศไว้ภายในเมท็อด ‘IWorkbenchWindow iww’ ดังรูปที่ ก.5 (c) ตามลำดับ

นอกจากนี้ เมท็อด getViewSite() ในโค้ดแสดงในรูปที่ ก.5 สืบทอดมาจากซุปเปอร์คลาส ViewPart จึงเสมือนมีการประกาศใช้งานเมท็อดดังกล่าวซ่อนไว้ในคลาส JavaMetricsView เมท็อด getViewSite() ถูกเรียกด้วยอ็อบเจกต์ this ดังนั้นจึงสามารถแปลงให้กลายเป็นอ็อบเจกต์

โหนด ‘JavaMetricsView this’ ได้ ดังรูปที่ ก.5(d) เช่นเดียวกับอ็อบเจกต์ของชนิดข้อมูล IViewSite ที่ถูกสร้างขึ้นเพื่อเป็นตัวแทนของอ็อบเจกต์ที่รีเทิร์นโดยเมท็อด getViewSite() และเป็นผู้เรียกเมท็อด getWorkbenchWindow() ในเมื่ออ็อบเจกต์ดังกล่าวไม่มีชื่อที่ตั้งไว้โดยเฉพาะ ดังนั้นเราจึงสร้างชื่อให้โดยให้ชื่อว่า IViewSite ivs_1 ดังรูปที่ ก.5 (e) โดยการตั้งชื่อจะมาจากการนำตัวอักษรใหญ่ของชื่อคลาส ตามด้วยหมายเลขที่จะเพิ่มค่าไปเรื่อยๆ ในกรณีที่มีอ็อบเจกต์ที่ชื่อซ้ำกัน



รูปที่ ก.5 ประเภทของแบบจำลองเอดจ์

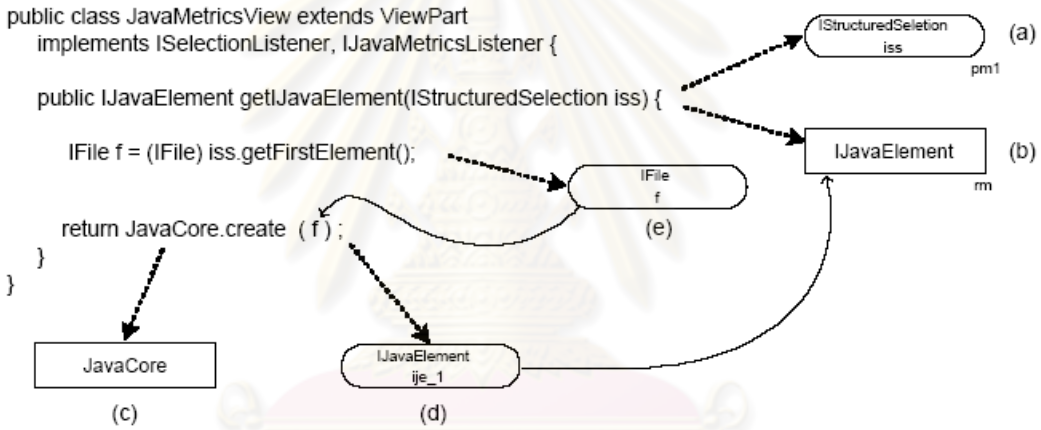
โหนดชนิดข้อมูล และโหนดอ็อบเจกต์สามารถใช้เป็นแบบจำลองของอินพุท เอาท์พุท พารา- มิเตอร์ รีเทิร์นชนิดข้อมูล และอ็อบเจกต์ โดยสามารถแบ่งหน้าที่ตามประเภทได้ดังนี้

1) โหนดอินพุท หมายถึงโหนดชนิดข้อมูลหรือโหนดอ็อบเจกต์ที่มีการเรียกใช้เมท็อดหรือเรียกค่าแอททริบิวต์ของตน เช่น โหนดชนิดข้อมูล ‘JavaCore’ ในรูปที่ ก.6 (c) แสดงตัวอย่างของโหนดอินพุทที่เรียกใช้เมท็อด create()
2) โหนดเอาท์พุท หมายถึงโหนดอ็อบเจกต์ที่สร้างขึ้นจากการเรียกใช้เมท็อด การสร้างอ็อบเจกต์ หรือการให้ค่า (Assignment) เช่นในรูปที่ ก.6 (c) โหนดอ็อบเจกต์ ‘IJaveElement ije_1’ ซึ่งเป็นโหนดเอาท์พุทที่ถูกสร้างขึ้นโดยเมท็อด create()
3) โหนดพารามิเตอร์ หมายถึงโหนดชนิดข้อมูลหรือโหนดอ็อบเจกต์ที่เป็นพารามิเตอร์ของเมท็อด m_i และถูกเรียกใช้ภายในเมท็อด m_s เช่น โหนดอ็อบเจกต์ ‘IFile f’ ซึ่งเป็นพารามิเตอร์ของเมท็อด create() ดังแสดงในรูปที่ ก.6 (e)
4) โหนดพารามิเตอร์ของเมท็อด หมายถึงโหนดชนิดข้อมูลหรือโหนดอ็อบเจกต์ที่ใช้เป็นพารามิเตอร์ภายในเมท็อดซิกเนเจอร์ เช่น โหนดอ็อบเจกต์ ‘IStructuredSelection

iss' ในรูปที่ ก.5 (a) ซึ่งเป็นพารามิเตอร์ในเมท็อดซิกเนเจอร์ของเมท็อด getIJaveElement()

5) โหนดรีเทิร์น หมายถึงโหนดชนิดข้อมูลหรือโหนดอ็อบเจกต์ที่รีเทิร์นมาจากเมท็อด $\mathbf{m_s}$ เช่น อ็อบเจกต์ 'IJavaElement ije_1' ในรูปที่ ก.6 (d) ที่รีเทิร์นมาจากการเรียกเมท็อด JaveCore.create(f)

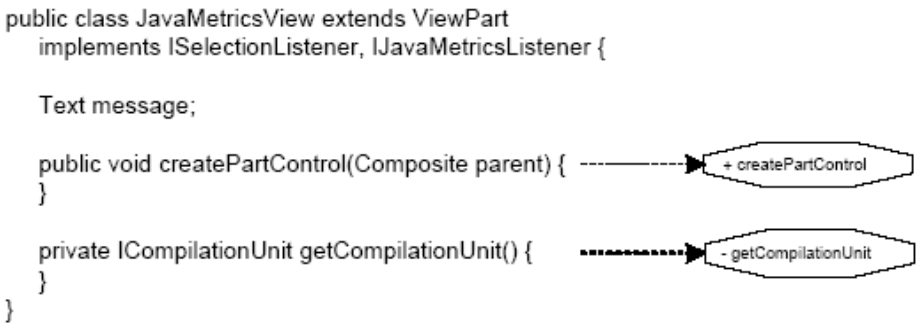
6) โหนดรีเทิร์นเมท็อด หมายถึงโหนดชนิดข้อมูลที่ประกาศไว้เป็นส่วนหนึ่งของรีเทิร์นชนิดข้อมูลของเมท็อดซิกเนเจอร์ เช่น โหนดชนิดข้อมูล 'IJaveElement' ในรูปที่ ก.6 (b) แสดงโหนดรีเทิร์นเมท็อดที่ประกาศไว้เป็นส่วนหนึ่งของซิกเนเจอร์ของเมท็อด getIJaveElement()



รูปที่ ก.6 ประเภทของแบบจำลองเอดจ์

1.3. โหนดเมท็อด ได้แต่ เมท็อดของคลาสรวมไปถึงเมท็อดซิกเนเจอร์

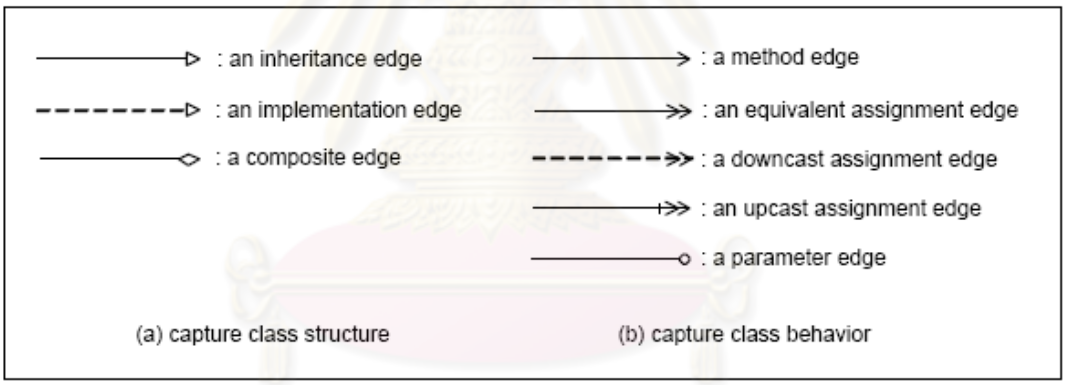
ยกตัวอย่างเช่น ในรูปที่ ก.7 เมท็อด createPartControl() และ getCompilationUnit() ถูกแปลงให้เป็นโหนดเมท็อด '+createPartControl' และ '-getCompilationUnit' โดยที่เครื่องหมาย + และ - หมายถึงโมดิฟายเออร์แบบ public และ private ตามลำดับ



รูปที่ ก.7 การแปลงจาวาโค้ดให้กลายเป็นโหนดเมท็อด

2. **แบบจำลองเอดจ์** (Edge Model)

เอดจ์สามารถแบ่งออกเป็นสองประเภทคือ เอดจ์ที่แสดงความสัมพันธ์ระหว่างคลาส และเอดจ์ที่แสดงพฤติกรรมของคลาส สัญลักษณ์ที่ใช้แทนเอดจ์แสดงดังรูปที่ ก.8



รูปที่ ก.8 ประเภทของแบบจำลองเอดจ์

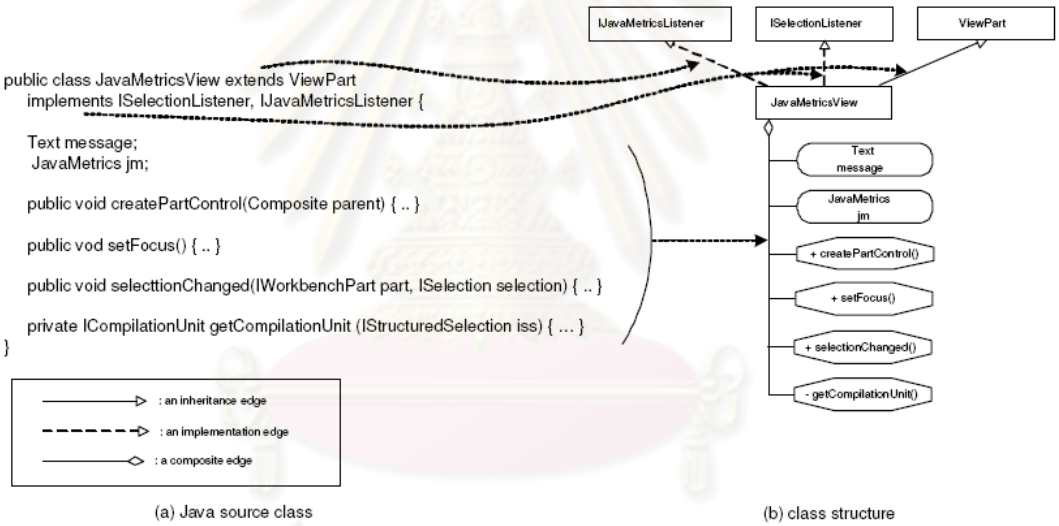
2.1. เอดจ์ที่แสดงความสัมพันธ์ระหว่างคลาส ได้แก่ เอดจ์อินเฮอริแทนซ์ (Inheritance Edige) เอดจ์อิมพลิเมนเทชัน (Implementation Edge) และเอดจ์คอมโพสิท (Composite Edge) ยกตัวอย่างเช่น ในรูปที่ ก.9 (a)

- ความสัมพันธ์ระหว่าง JavaMetricsView / ViewPart เชื่อมโยงได้ด้วยเอดจ์อินเฮอริแทนซ์
- ความสัมพันธ์ระหว่างJavaMetricsView/ISelectionListenerและ JavaMetricsView/IJavaMetricsListener เชื่อมโยงได้ด้วยเอดจ์อิมพลีเมนท์

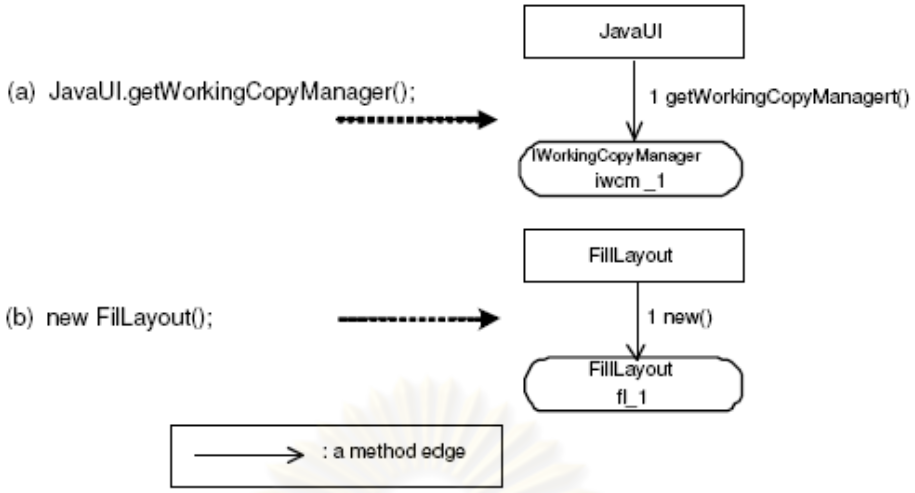
- ความสัมพันธ์ระหว่างJavaMetricsView/Text และ JavaMetricsView/JavaMetircs เชื่อมโยงได้ด้วยเอดจ์คอมโพสิท

2.2. เอดจ์ที่แสดงพฤติกรรมของคลาส ได้แก่ เอดจ์เมท็อด (Method Edge) เอดจ์การให้ค่า (Assignment Edge) และเอดจ์พารามิเตอร์ (Parameter Edge) โดยมีรายละเอียดดังนี้

- เอดจ์เมท็อด จะแสดงการเรียกใช้เมท็อด โดยเอดจ์จะออกจากโหนดชนิดข้อมูลหรือโหนดอ็อบเจกต์ n_i ที่เรียกใช้เมท็อด m_i และจะชี้ไปยังโหนดอ็อบเจกต์ n_o ซึ่งรีเทิร์นมาจากเมท็อด m_i รูปที่ก.10 (a) จะเห็นว่าเมท็อดสแตติก getWorkingCopyManager() ที่ประกาศไว้ในคลาส JavaUI มีการเรียกใช้และรีเทิร์นอ็อบเจ็กต์ของชนิดข้อมูล IWorkingCopyManager โดยสามารถแปลงเป็นเอดจ์เมท็อด getWorkingCopyManager() ซึ่งเชื่อมต่อโหนดชนิดข้อมูลอินพุท ‘JavaUI’ และโหนดอ็อบเจกต์เอาท์พุท ‘IWorkingCopyManager iwcm_1’



รูปที่ ก.9 ตัวอย่างการแปลงจาวาโค้ดไปเป็นโหนดที่เชื่อมต่อกันด้วยความสัมพันธ์แบบต่างๆ



รูปที่ ก.10 ตัวอย่างการแปลงจากจาวาโค้ดไปเป็นเอดจ์เมท็อด

- เอดจ์การให้ค่า สามารถแบ่งออกเป็น 3 ประเภท ได้แก่

  1) การให้ค่าแบบสมมูล (Equivalent Assignment) เกิดขึ้นเมื่อชนิดข้อมูลของตัวแปรเอาท์พุท v_o เป็นชนิดข้อมูลเดียวกับตัวแปรอินพุท v_i ยกตัวอย่างเช่นในรูปที่

  ก.11 ตัวแปร targetEditor และ cuEditor คือชนิดข้อมูล IEditorPart และสามารถแสดงในรูปแบบอ็อบเจกต์โหนด 'IEditorInput targetEditor' และ 'IEditor cuEditor150' ตามลำดับ และสามารถให้ค่าแบบสมมูล โดยมีลูกศรโยงไปจากโหนดอ็อบเจกต์อินพุท 'IEditorInput targetEditor' ไปยังโหนดอ็อบเจกต์เอาท์พุท 'IEditorPart cuEditor'

  2) การให้ค่าแบบดาวน์แคส (Downcast Assignment) เกิดขึ้นเมื่อชนิดข้อมูลของตัวแปรเอาท์พุท v_o มีความสัมพันธ์แบบสืบทอดในลำดับที่ต่ำกว่า (หรือมีความสัมพันธ์เป็นลูกของ) ตัวแปรอินพุท v_i ในรูปที่ ก.11 (b) ตัวแปร obj เป็นชนิดข้อมูล Object ตัวแปร icu_1 สร้างขึ้นจากการดาวน์แคสชนิดข้อมูล Object เป็นชนิดข้อมูล ICompilationUnit ตัวแปร obj และ icu_1 อยู่ในรูปของโหนดอ็อบเจกต์ โดยมีลูกศรแสดงการดาวน์แคสจากโหนดอ็อบเจกต์อินพุท 'Object obj' ไปยังโหนดเอาท์พุทอ็อบเจกต์ 'ICompilationUnit icu_1'

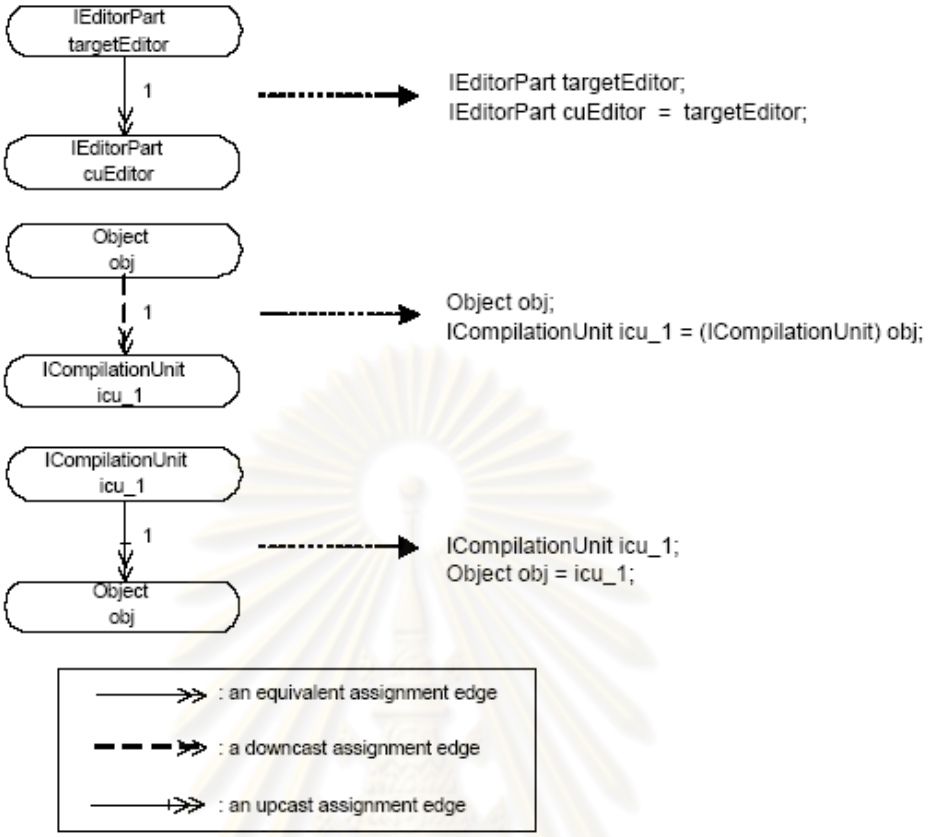
  3) การให้ค่าแบบอัพแคส (Upcast Assignment) เกิดขึ้นเมื่อชนิดข้อมูลของตัวแปรเอาท์พุท v_o มีความสัมพันธ์แบบสืบทอดในลำดับที่สูงกว่า (หรือมีความสัมพันธ์เป็นแม่ของ) ตัวแปรอินพุท v_i ในรูปที่ ก.11 (b) ตัวแปร obj และ

icu_1 เป็นชนิดข้อมูล Object และมีลำดับความสืบทอดที่สูงกว่า ICompilationUnit ตัวแปร obj และ icu_1 แสดงในรูปแบบของโหนดอ็อบเจกต์ และการให้ค่าแบบอัพแคสแสดงในรูปแบบของลูกศรที่ชี้จากโหนดอ็อบเจกต์อินพุท 'ICompilationUnit icu_1' ไปยังโหนดอ็อบเจกต์เอาท์พุท 'Object obj'

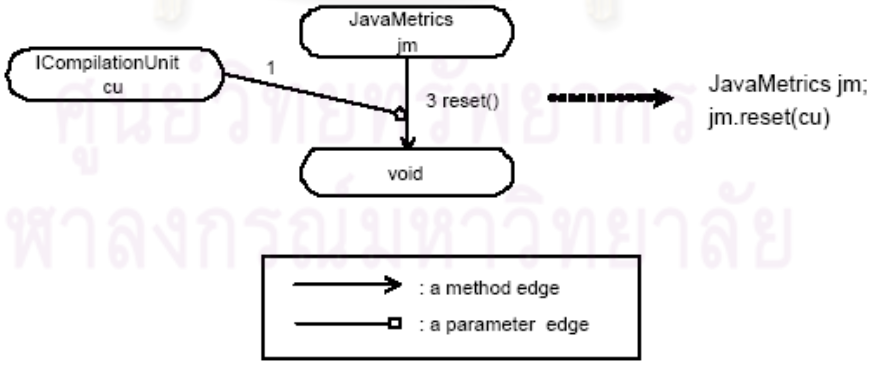
- เอดจ์พารามิเตอร์ แสดงความสัมพันธ์ระหว่างโหนดพารามิเตอร์ n_p ของเมท็อด m_i โดยที่เมท็อด m_i มีการเรียกใช้งานภายในเมท็อด m_s เอดจ์พารามิเตอร์จะแตกต่างจากเอดจ์ประเภทอื่นเพราะเกิดขึ้นบนเอดจ์เมท็อด ซึ่งแสดงให้เห็นว่าการเรียกใช้เมท็อด m_i จะต้องมีโหนดพารามิเตอร์ n_p ในรูปที่ ก.10 คำสั่ง 'jm.reset(cu)' จะเห็นว่า jm cu และ void แสดงอยู่ในรูปแบบโหนดอ็อบเจกต์ เช่นเดียวกับเมท็อด reset ที่แสดงอยู่ในรูปแบบเอดจ์เมท็อดโดยมีโหนดอินพุทอ็อบเจกต์ 'JavaMetrics jm' ผลลัพธ์คือโหนดเอาท์พุทอ็อบเจกต์ 'void' โดยมีเอดจ์พารามิเตอร์เชื่อมต่อระหว่างโหนดพารามิเตอร์ 'ICompilationUnit cu' และเอดจ์เมท็อด 'reset()'

เอดจ์เมท็อด เอดจ์การให้ค่า และเอดจ์พารามิเตอร์จะมีหมายเลขกำกับระบุลำดับก่อนหลังการเรียกใช้ภายในเมท็อด m_s สำหรับเอดจ์พารามิเตอร์ หมายเลขกำกับจะนับเฉพาะภายในขอบเขตของการเรียกใช้เมท็อดของพารามิเตอร์เท่านั้น

ยกตัวอย่างเช่น สมมติให้คำสั่ง 'jm.reset(cu)' มีการเรียกใช้งานเป็นลำดับสามภายในเมท็อด m_s เอดจ์เมท็อด 'reset' จึงได้รับหมายเลข 3 กำกับดังแสดงในรูปที่ ก.10 นอกจากนั้นเนื่องจากเมท็อด reset() มี ICompilationUnit เป็นพารามิเตอร์ ซึ่งสามารถแปลงเป็นโหนดอ็อบเจกต์ 'ICompilationUnit cu' ได้ รูปที่ ก.10 จะเห็นว่าเอดจ์พารามิเตอร์เป็นตัวเชื่อมต่อโหนด อ็อบเจกต์ 'ICompilationUnit cu' ให้เข้ากับเอดจ์เมท็อด 'reset()' โดยจะกำกับหมายเลขของเอดจ์เป็น 1 ใหม่ เพราะถือว่าหลุดจากขอบเขตของเมท็อด m_s แล้ว



รูปที่ ก.11 การแปลงจากจาวาโค้ดเป็นเอดจ์การให้ค่า



รูปที่ ก.12 การแปลงจาวาโค้ดให้กลายเป็นเอดจ์พารามิเตอร์

ภาคผนวก ข
คู่มือการใช้งาน

ภาคผนวก ข
คู่มือการใช้งาน

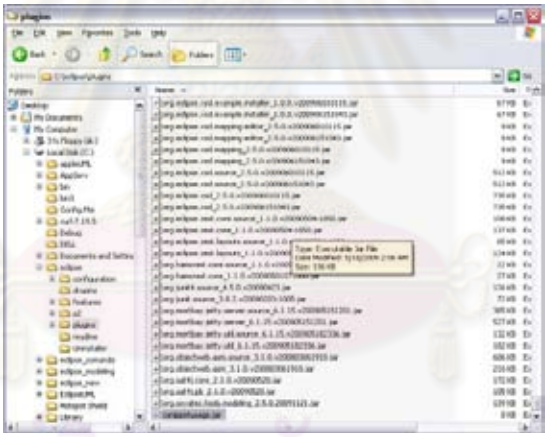
1. สภาพแวดล้อมที่จำเป็นในการติดตั้งเครื่องมือ XSNIPPETUSAGE

1.1. ความต้องการสำหรับการติดตั้ง XSNIPPETUSAGE มีดังต่อไปนี้

- Java Runtime Environment (JRE) เวอร์ชัน 5 เป็นต้นไป
- โปรแกรม Eclipse

2. การติดตั้งเครื่องมือ

1) เมื่อติดตั้งโปรแกรม Eclipse แล้ว นำไฟล์ XSNIPPETUSAGE.jar ไปวางไว้ที่ไดเร็กทอรีที่ลงโปรแกรม Eclipse ดังรูปที่ ข.1



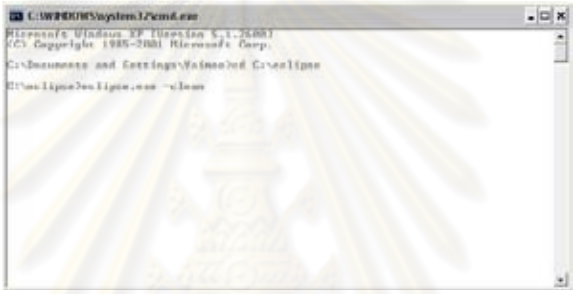
รูปที่ ข.1 ไฟล์ XSNIPPETUSAGE.jar ในไดเร็กทอรีที่ลงโปรแกรม Eclipse

2) เลือก Window>Run as>cmd เพื่อเรียก Command Prompt

3) พิมพ์คำสั่ง cd C:\eclipse แล้วกด Enter เพื่อเข้าไปในไดเร็กทอรีที่ลงโปรแกรม Eclipse ดังรูปที่ ข.2

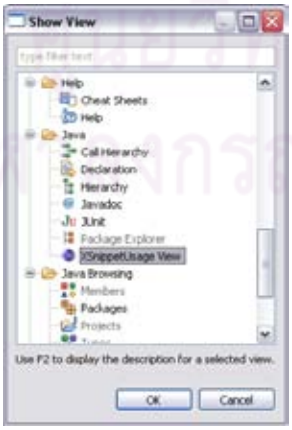
รูปที่ ข.2 หน้าจอ Command Prompt

4) พิมพ์คำสั่ง eclipse.exe -clean แล้วกด Enter เพื่อเคลียร์แคชของ Eclipse และเพื่อเรียกโปรแกรมให้ทำงาน ดังรูปที่ ข.3

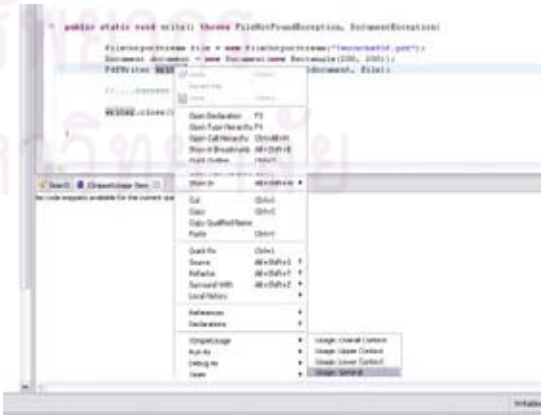


รูปที่ ข.3 หน้าจอ Command Prompt (2)

5) ในโปรแกรม Eclipse เลือก Window>Show View>Other ในไดเร็กทอรี Java เลือก XSnippetUsage View ดังรูปที่ ข.4 ทดสอบด้วยการคลิกขวาที่ชื่ออ็อบเจกต์ ถ้าพบฟังก์ชัน XSnippetUsage จะแสดงว่าโปรแกรมติดตั้งโดยสมบูรณ์แล้ว ดังรูปที่ ข.5



รูปที่ ข.4 แสดงหน้าจอการเลือก XSnippetUsage View



รูปที่ ข.5 หน้าจอแสดงการเรียกใช้งาน XSnippetUsage

ประวัติผู้เขียนวิทยานิพนธ์

นางสาวปารัช สุพงษ์พันธุ์ เกิดวันที่ 25 กรกฎาคม พ.ศ. 2528 สำเร็จการศึกษาระดับมัธยมศึกษาตอนปลายจากโรงเรียนสายน้ำผึ้ง จังหวัดกรุงเทพฯ เมื่อปีการศึกษา 2545 และสำเร็จการศึกษาในหลักสูตรวิทยาศาสตรบัณฑิต สาขาวิชาเอกวิทยาศาสตร์คอมพิวเตอร์ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยธรรมศาสตร์ จังหวัดปทุมธานี เมื่อปีการศึกษา 2549 และเข้าศึกษาต่อในหลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิทยาศาสตร์ซอฟต์แวร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2550 ที่อยู่ปัจจุบันที่สามารถติดต่อได้คือ 378 แยก 16 ซอยอ่อนนุช 17 เขตสวนหลวง แขวงสวนหลวง กรุงเทพมหานคร 10250 หมายเลขโทรศัพท์ 086-662-1250 อีเมล์ parat.s@gmail.com