การติดตามการทำงานของโปรแกรมจาวา โดยใช้การเขียนโปรแกรมเชิงลักษณะและการออกแบบเชิงกิจกรรม

Java Program Monitoring using Aspect Oriented Programming and Sequence Diagram

นัทธี ศิริพรไพศาล (Natti Siripornpaisarn)* และ พรชัย มงคลนาม (Pornchai Mongkolnam)*

บทคัดย่อ

การวิจัยนี้มีวัตถุประสงค์เพื่อสร้างโปรแกรมสำหรับติดตามการทำงานของโปรแกรมจาวา ว่าในขณะที่โปรแกรมทำงานนั้นมีการทำงานเป็นอย่างไรโดยดูว่ามีการเรียกใช้คลาสอะไรบ้างมีเมท็อดไหนในคลาสถูกเรียกใช้บ้าง แล้วนำมาสรุปว่าแต่ละเมท็อดที่ทำงานใช้เวลามากน้อยขนาดไหน มีเมท็อดไหนไม่เคยถูกเรียกใช้เลยหรือมีเมท็อดไหนที่ใช้เวลามากๆ ซึ่งจะช่วยลดเวลาที่ต้องใช้ในการตรวจสอบเมท็อดในกรณีที่ซอฟต์แวร์ที่ใช้งานนั้นมีการทำงานที่ใช้เวลามากจนเกินไป ทำให้เข้าไปตรวจสอบเมท็อดได้ถูกต้องมากขึ้น ซึ่งในงานวิจัยนี้ได้นำวิธีการออกแบบเชิงกิจกรรม (Sequence Diagram) เข้ามาช่วยในการแสดงผลเพื่อทำให้ผู้ใช้มองเห็นภาพการทำงานและลำดับการทำงานของซอฟต์แวร์ได้ชัดเจน โดยได้นำหลักการการเขียนโปรแกรมเชิงลักษณะ (Aspect Oriented Programming) หรือ AOP มาใช้สร้าง Sequence Diagram ขณะที่โปรแกรมทำงานซึ่งการเขียนโปรแกรมเชิงลักษณะนี้จะไม่มีการแก้ไขโค้ดเดิมของซอฟต์แวร์ แต่โค้ดที่ถูกเขียนเพิ่มจะถูกแยกไว้ต่างหาก ซึ่งทำให้ผู้ใช้ไม่ต้องกังวลว่า โค้ดของซอฟต์แวร์ที่นำมาใช้จะถูกแก้ไขหรือถูกแทรกโค้ดเข้าไปในโค้ดของซอฟต์แวร์เดิม

คำสำคัญ: โปรแกรมเชิงลักษณะ การออกแบบเชิงกิจกรรม ลำดับการทำงานของซอฟต์แวร์ โปรแกรมติดตามการทำงาน

Abstract

This research aims to create the program which monitors the execution flows of a Java program by tracking both method calls and object interactions. Having done so would give us some insights of which methods and classes are used according to their time usage and numbers of calls. This would significantly reduce the time a programmer needs to check for any complexity, risk, and running time of a given method and class. We use the UML sequence diagram to display calling sequences of the methods and classes of the running program, with the help of an Aspect Oriented Programming (AOP) used in keeping tracks of all calls. The Java program is kept unmodified, whereas the AOP code is put in a different file. This effectively allows us to freely track the calling sequences without having to interfere with an existing code.

Keyword: Aspect oriented programming, AOP, Sequence Diagram, program flow, programming monitoring.

1. บทนำ

ในปัจจุบันมีการพัฒนาซอฟต์แวร์กันอย่างแพร่หลาย ซึ่งซอฟต์แวร์ที่พัฒนาขึ้นมานั้นนำไปใช้ประโยชน์ในหลายๆ ด้าน เช่น นำไปใช้เพื่อจัดการผลิต, นำไปใช้ในทางการแพทย์, นำไปใช้ในการจัดการข้อมูลภายในองค์กร เป็นต้น ซึ่งการพัฒนาซอฟต์แวร์ขึ้นมาใช้งานนั้นมีหลายประเภท ทำได้หลายแบบ เช่น การเริ่มพัฒนาซอฟต์แวร์ตั้งแต่ต้นใหม่หมดเลยโดยที่ไม่มีของเดิมอยู่เลย, การพัฒนาซอฟต์แวร์เป็นบางส่วนโดยการเพิ่มเติมจากของเดิมที่มีอยู่ หรือการนำโมดูลมาจากหลายๆ ที่เพื่อนำมาใช้รวมกันเพื่อสร้างซอฟต์แวร์ขึ้นมา เป็นต้น ซึ่งในงานวิจัยนี้จะเน้นไปถึงการพัฒนาซอฟต์แวร์แบบ

* คณะเทคโนโลยีสารสนเทศ มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี

ที่เป็นการเพิ่มเติมจากของเดิมที่มีอยู่แล้ว โดยการพัฒนาซอฟต์แวร์จะเริ่มขึ้นเมื่อเราต้องไปทำการดูการทำงานของซอฟต์แวร์เดิมที่มีอยู่แล้วว่าควรจะเพิ่มเติมที่ตรงจุดไหน สามารถนำตรงไหนมาต่อยอดได้บ้าง ซึ่งการตรวจสอบซอฟต์แวร์ในขั้นตอนนี้ทำให้ต้องเสียเวลามาทำการเรียกดูโค้ดการทำงานของซอฟต์แวร์ทีละบรรทัดว่าซอฟต์แวร์นี้มีการทำงานเป็นอย่างไร ทำอะไรบ้าง ทำอะไรก่อนอะไรหลัง ที่ตรงจุดไหน แล้วถึงจะมาทำการวิเคราะห์เพิ่มเติมอีกที

ดังนั้นในงานวิจัยนี้จึงเห็นว่าการนำ AspectJ [1], [2] ซึ่งเป็นเครื่องมือที่สามารถนำมาใช้กับแนวคิดแบบ Aspect Oriented Programming (AOP) [3] โดยใช้หลักการของการ Cross-cutting Concerns [4] เข้ามาประยุกต์เข้ากับการใช้ Sequence Diagram โดยการทำ Cross-cutting Concerns นี้จะมุ่งเน้นไปที่การสร้าง Sequence Diagram เพราะการใช้ Sequence Diagram นั้นจะทำให้เราเห็นลำดับการทำงานของโปรแกรมอย่างชัดเจนว่าลำดับไหนก่อนหลัง และสามารถเปรียบเทียบเวลาได้จาก Lifeline ด้วยว่าส่วนไหนใช้เวลามากน้อยแค่ไหน เมื่อเห็นว่าส่วนไหนที่ใช้เวลามากก็สามารถที่จะเข้าไปปรับปรุงโค้ดในส่วนนั้นเพื่อให้สามารถทำงานให้เร็วขึ้นได้ และยังนำมาใช้ช่วยลดเวลาในการวิเคราะห์ตรวจสอบการทำงานของโปรแกรมว่ามีการทำงานอย่างไร และยังช่วยเพิ่มความเข้าใจและความชัดเจน [5] ในการทำงานของโปรแกรมมากยิ่งขึ้น

2. ทฤษฎีและบทความที่เกี่ยวข้อง

2.1 AspectJ

AOP เป็นแนวความคิดที่ถูกคิดขึ้นมาโดย Kiczales [3] ซึ่งสามารถนำมาใช้แก้ปัญหาการทำ Cross-cutting ของ Object Oriented Programming (OOP) โดยใช้หลักการที่เรียกว่าการ Cross-cutting Concern คือการตัดหรือแยกเอาเฉพาะสิ่งที่เราสนใจ ออกมาพิจารณาหรือนำสิ่งที่ทำงานเหมือนๆ กันมาไว้รวมกันในที่ๆ เดียวซึ่งจะทำให้โค้ดดูเป็นระเบียบมากขึ้นไม่เกิดความซับซ้อนของตัวโค้ด ซึ่ง AspectJ นี้สนับสนุนการเขียนโปรแกรมตามแนวคิดนี้โดยมีส่วนประกอบต่างๆ ดังนี้

Join Point คือจุดต่างๆที่แสดงให้เห็นถึงการทำงานของโปรแกรม เช่น Method และ Constructor เป็นต้น

Pointcut คือส่วนที่เอาไว้ระบุถึงตัว Join Point ที่เราได้ทำการเลือกระหว่างที่โปรแกรมทำงานว่าต้องการเจาะจงไปที่ Join Point ไหน

Advice คือส่วนที่เอาไว้กำหนดการทำงานของตัว Pointcut ที่ถูกจับคู่เข้ากับตัวโค้ดเมื่อโปรแกรมเริ่มทำงานแล้ว และมีการเรียกใช้ Join Point ที่ถูกระบุอยู่ใน Pointcut นั้นๆ ก็จะทำให้ส่วนของ Advice นี้เริ่มทำงาน โดย Advice [6] มี 3 แบบคือ

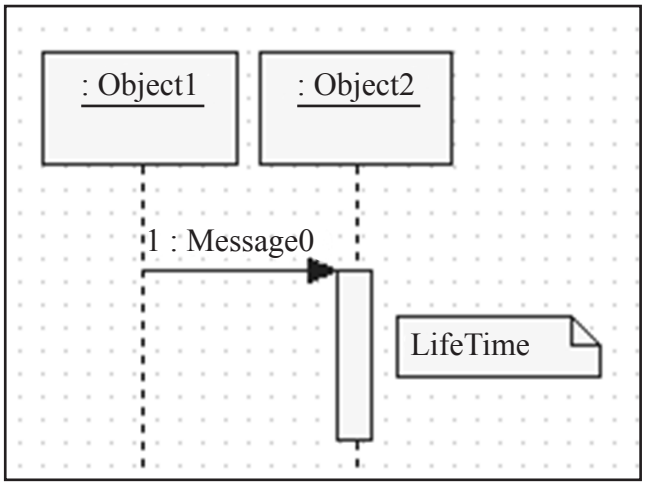
1) Before จะเริ่มทำงานก่อนที่ Join Point ที่ถูกระบุใน Pointcut นั้นๆ ทำงาน

2) After จะเริ่มทำงานหลังจากที่ Join Point ที่ถูกระบุใน Pointcut นั้นๆ ทั้งงานเสร็จแล้ว

3) Around จะเริ่มทำงานก่อนที่ Join Point ที่ถูกระบุใน Pointcut นั้นๆ เริ่มทำงานแต่จะต่างกับ Before ตรงที่สามารถระบุใน Around ได้ว่าให้ไปทำงานต่อที่ Join Point นั้นต่อหรือไม่

2.2 Sequence Diagram

Sequence Diagram [7], [8] เป็นหนึ่งใน Diagram ของ UML ซึ่งเป็น Diagram ที่ใช้แสดงให้เห็นถึงลำดับการทำงานของโปรแกรมโดยมี Object เป็น Instance ของคลาสนั้น โดย Object จะใช้ ":" นำหน้าชื่อคลาสนั้นและ Lifetime ใน Lifeline เป็นตัวแสดงถึงลำดับการทำงานก่อนหลัง โดยมี Message ที่ส่งหากันระหว่าง Object เป็นตัวบอกว่าทำอะไรบ้าง ดังภาพที่ 1



ภาพที่ 1 *แสดงสัญลักษณ์ของ Sequence Diagram*

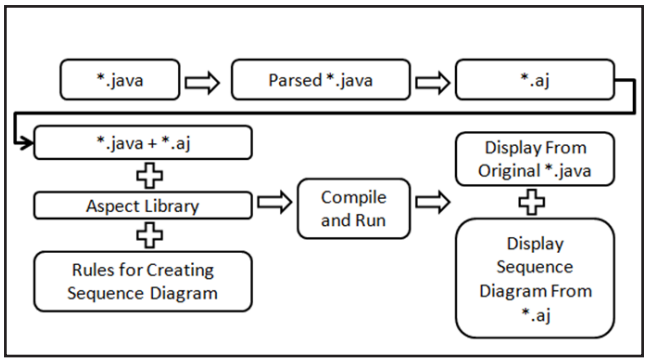
2.3 งานวิจัยอื่นที่เกี่ยวข้อง

ในงานวิจัยของ Remi Douence และ Mario Südholt [9], [10] ได้ใช้การแทรกโค้ดลงในโค้ดเดิมเพื่อให้รู้ว่าขณะนี้เกิดเหตุการณ์ใดขึ้นมาแล้วโต้ตอบออกไปและสามารถตรวจสอบผลลัพธ์ของเหตุการณ์นั้นๆ ได้

ในงานวิจัยของ Richters และ Gogolla [11] ได้นำ Use-Tool มาใช้ซึ่งเป็น Tool ที่อาศัยหลักการของ AOP มาใช้ในการสร้างล็อกไฟล์และไดอะแกรมต่างๆ โดยสร้างมาจากโค้ดของ OCL Constraint เป็นตัวกำหนดรูปแบบ Diagram ต่างๆ และจะแสดงให้เห็นข้อผิดพลาดในไดอะแกรมต่างๆ ได้

ในงานวิจัยของ Malloy และ Power [12] ได้นำโค้ด C++ มาทำการ Parse ผ่าน gcc เพื่อนำไปสร้างเป็น UML Diagram ต่างๆ ซึ่งนำหลักการของ AOP มาใช้โดยแบ่งงานออกเป็น 2 ส่วนคือส่วนที่เป็น Static กับส่วนที่เป็น Dynamic โดยส่วนที่เป็น Static จะสร้างมาจาก Select Tool จะทำการสร้าง Class Diagram และ Call Graph และใช้ Aspect Generator ทำการสร้าง Pointcut แล้วนำไปรวมกับ Advice Code กับ Python/C++ ที่เขียนขึ้นมาเพื่อสร้างเป็นโค้ดสำหรับใช้งานในส่วนของ Dynamic ซึ่งจะถูกนำไปใช้ใน Profiler Tool โดยจะนำไปใช้สร้าง Sequence Diagram และ Communication Diagram ในขณะที่รันโค้ด C++ นั้น

ในงานวิจัยของ Coelho และ คณะ [13] ได้นำหลักการของ AOP มาใช้กำหนดรูปแบบสำหรับนำไปใช้โดยจะกำหนดเป็นขั้นตอนต่างๆ เอาไว้ให้เพื่อดึงเอาเฉพาะสิ่งที่เราสนใจออกมาตรวจสอบได้



ภาพที่ 2 แสดงขั้นตอนการดำเนินการวิจัย

จากภาพที่ 2 จะแบ่งขั้นตอนการทำวิจัยออกเป็นขั้นๆ โดยจะเริ่มจาก " *.java " คือการนำไฟล์จาวาไปทำการ Parse เพื่อให้ได้ไฟล์ " *.aj " ขึ้นมา หลังจากนั้นจะนำเอาไฟล์ " *.aj " ไปใช้งานร่วมกับไฟล์ " *.java " เดิมและไฟล์เพื่อให้แสดง Sequence Diagram ตามลำดับการทำงานของไฟล์จาวานั้นขึ้นมา

3.1 การ Parse ไฟล์จาวา

โดยในขั้นตอนนี้จะนำเอาโค้ดจาวาที่ต้องการทดสอบมาทำการแยกเอาชื่อคลาสและชื่อเมท็อดที่ถูกสร้างในโค้ดจาวามาเก็บไว้โดยใช้วิธีการสแกนที่ละตัวอักษร โดยอันดับแรกจะทำการตัดคอมเม้นท์ต่างๆ ที่อยู่ในโค้ดจาวาออกก่อน แล้วจึงเริ่มทำการสแกนหาตัวอักษร "{" ก่อนเมื่อพบแล้วจะทำการเช็คว่าตัวอักษรที่อยู่ข้างหน้ามีคำว่า "class" หรือไม่ ถ้ามีจะเอาตัวอักษรที่อยู่หลัง "class" ไปเก็บไว้เป็นชื่อคลาส แล้วทำการสแกนหาตัวอักษร "{" ต่อไปและเมื่อพบแล้วปรากฏว่าตัวอักษรที่อยู่หน้า "{" ไม่มีคำว่า "class" เลยก็จะเข้าสู่ขั้นตอนถัดไปคือมาตรวจสอบดูว่าเป็นเมท็อดหรือคอนสตรักเตอร์ โดยตรวจสอบจากตัวอักษรที่อยู่ข้างหน้า "{" ว่าเหมือนชื่อคลาสที่พบก่อนหน้าหรือไม่ถ้าเหมือนจะจัดให้เป็นคอนสตรักเตอร์ ถ้าไม่เหมือนก็จะจัดให้เป็นเมท็อด หลังจากที่ได้ชื่อเมท็อดแล้วจะทำการดูตัวอักษรข้างหน้าชื่อเมท็อดต่อเพื่อทำการหาว่าเมท็อดนั้นมีการคืนค่าหรือไม่ เพื่อแบ่งประเภทของเมท็อด โดยชื่อคลาส ชื่อเมท็อด และประเภทของเมท็อด จะถูกนำไปใช้ในลำดับต่อไป

3.2 รูปแบบไฟล์ AspectJ

ในขั้นตอนนี้จะนำโดยชื่อคลาส ชื่อเมท็อด และประเภทของเมท็อดมากำหนดรูปแบบของไฟล์ AspectJ จะต้องมีการกำหนด Pointcut เพื่อเข้าถึงแต่ละ Joint Point ขึ้นมาก่อน โดยจะนำชื่อคลาสนำมารวมกับชื่อเมท็อดไปประกอบเป็นชื่อ Pointcut และนำชื่อเมท็อดกับประเภทของเมท็อดมากำหนดเป็น Join Point ที่ระบุใน Pointcut นั้นๆ ส่วน Advice ที่นำมาใช้คือ before และ after ซึ่งจะอ้างถึง Pointcut ที่กำหนดไว้ตอนแรกให้เริ่มทำการสร้าง Sequence Diagram ขึ้นมาเมื่อ Join Point ที่ถูกระบุใน Pointcut นั้นเริ่มทำงานโดยจะใช้ชื่อของ Class เป็นชื่อ Object ถ้ามีการสร้าง และใช้ชื่อเมท็อดเป็นชื่อของ Message ที่ส่งหากันระหว่าง Object

3.3 โครงสร้างของโปรแกรม

โดยตัวโปรแกรมนี้จะแบ่งโฟลเดอร์ที่ต้องใช้งานเป็น 3 โฟลเดอร์คือ Input, Output และ Pattern โดย Input จะเป็นโฟลเดอร์สำหรับนำโค้ดไฟล์จาวาต่างๆ ที่ต้องการใช้มาใส่ไว้ Output จะเป็นโฟลเดอร์สำหรับเก็บผลลัพธ์ที่ได้จากการทำการสแกนโค้ดในโฟลเดอร์ Input แล้วสร้างเป็นไฟล์ AspectJ ออก

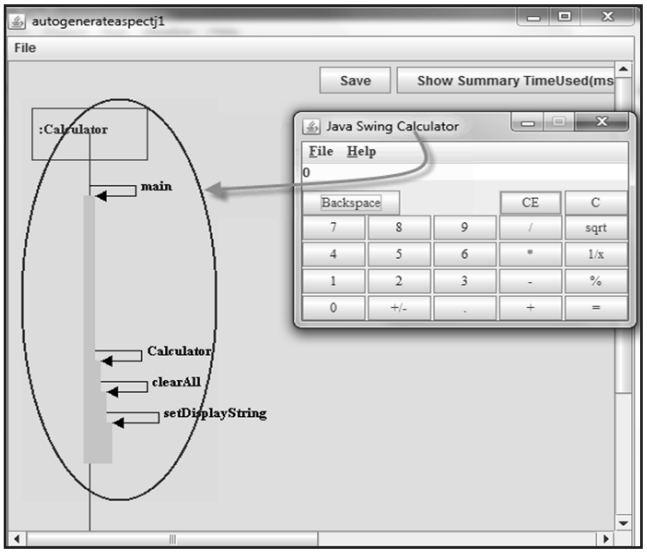
มาเก็บไว้ ส่วนโฟลเดอร์ Pattern จะเป็นโมเดล [14] สำหรับเก็บรูปแบบไฟล์ที่จะถูกใช้นำไปสร้างเป็นไฟล์ AspectJ

4. ผลการดำเนินงาน

หลังจากที่ได้นำโปรแกรมเครื่องคิดเลข [15] และโปรแกรม Visitor Pattern Example [16] มาทำการทดสอบจะได้ผลดังนี้ โดยจะแบ่งผลการทดลองออกเป็น 2 ส่วน

4.1 ส่วนแสดงผลการสร้าง Sequence Diagram ในขณะที่ซอฟต์แวร์ทำงาน

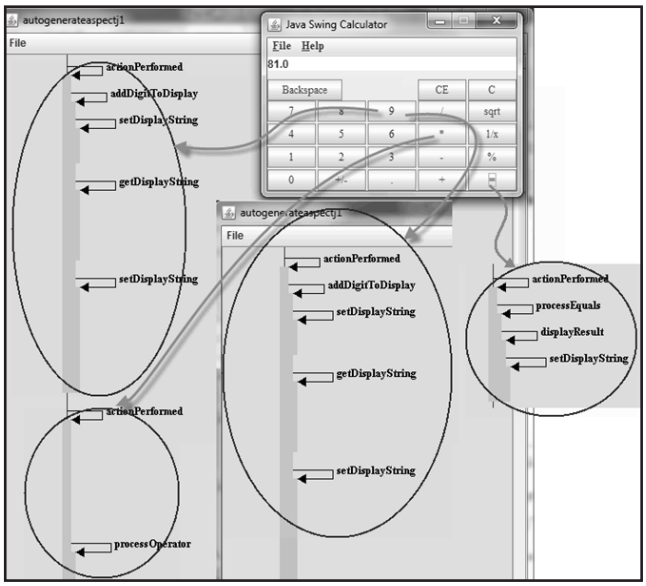
4.1.1 ผลจากโปรแกรมเครื่องคิดเลข



ภาพที่ 3 *แสดงสัญลักษณ์ของ Sequence Diagram*

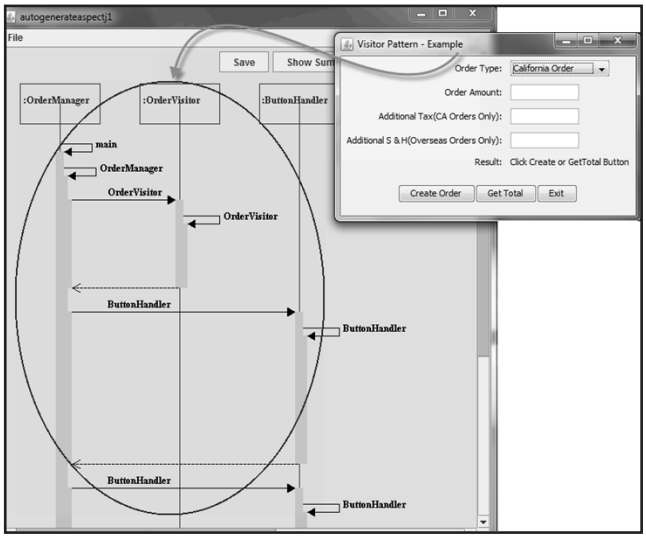
จากภาพที่ 3 เมื่อทำการรันโปรแกรมเครื่องคิดเลขขึ้นมาจะมี Sequence Diagram แสดงขึ้นมาเพราะว่าเมื่อโปรแกรมเครื่องคิดเลขเริ่มมีการทำงานโดยมีการเรียกใช้เมท็อดภายในคลาส Calculator คือ “main” โดยเมท็อด “main” จะไปเรียกใช้คอนสตรักเตอร์ “Calculator” และคอนสตรักเตอร์ “Calculator” ไปเรียกใช้เมท็อด “clearAll” และเมท็อด “clearAll” ไปเรียกใช้เมท็อด “setDisplayString” และจึงสิ้นสุดการทำงานในขั้นตอนการเรียกโปรแกรมเครื่องคิดเลขขึ้นมาแสดง ตามลำดับดังภาพที่ 4

จากภาพที่ 4 หลังจากได้ทำการทดลองป้อน Input ใส่เครื่องคิดเลขโดยใช้ Input เป็น “9*9 = ” จึงได้ผลลัพธ์ดังรูป ซึ่งจะแสดงเมท็อดต่างๆ ตามลำดับที่ถูกเรียกใช้ขึ้นมา ซึ่งจะเห็นได้ว่าเมท็อดทั้งหมดทำงานบนคลาส Calculator เพียงคลาสเดียว โดยมีการเรียกใช้เมท็อดอื่นๆ ผ่านเมท็อด “actionPerformed”



ภาพที่ 4 *แสดง Sequence Diagram หลังจากใส่ Input บนโปรแกรมเครื่องคิดเลข*

4.1.2 ผลจากโปรแกรม Visitor Pattern Example ดังภาพที่ 5



ภาพที่ 5 *แสดง Sequence Diagram หลังจากโปรแกรม Visitor Pattern Example เริ่มทำงาน*

จากภาพที่ 5 เมื่อทำการรันโปรแกรม Visitor Pattern Example ขึ้นมาจะเห็นได้ว่า Sequence Diagram ที่แสดงขึ้นมามีการเรียกใช้เมท็อดระหว่าง Object กันตามภาพโดยโปรแกรม Visitor Pattern Example จะเริ่มทำงานที่เมท็อด “main” และคอนสตรักเตอร์ “OrderManager” ซึ่งอยู่บน Object ที่ชื่อว่า “:OrderManager” โดยที่คอนสตรักเตอร์ “OrderManager” จะไปเรียกใช้คอนสตรักเตอร์ “OrderVisitor”

ที่อยู่บน Object ที่ชื่อว่า ":OrderVisitor" และเมื่อ คอนสตรักเตอร์ "OrderVisitor" ทำงานเสร็จก็จะกลับมาที่คอนสตรักเตอร์ "OrderManager" ก่อนแล้วจึงไปเรียกใช้คอนสตรักเตอร์ "ButtonHandler" ที่อยู่บน Object ที่ชื่อว่า ":ButtonHandler" อีกสองครั้งจึงสิ้นสุดการทำงานในขั้นตอนการเรียกโปรแกรม Visitor Pattern Example ขึ้นมาแสดง

4.2 ส่วนแสดงผลรวมของแต่ละเมท็อดในขณะที่ซอฟต์แวร์ทำงาน

ซึ่งผลรวมของเวลาที่ใช้ จะแสดงจะอยู่ในรูปของเปอร์เซนต์ซึ่งจะทำให้มองเป็นสัดส่วนชัดเจนขึ้นว่ามีเมท็อดไหนใช้เวลามากหรือน้อยเป็นสัดส่วนเท่าไรของเวลาที่ใช้ทั้งหมด

4.2.1 แสดงผลรวมจากโปรแกรมเครื่องคิดเลข

```
Summary By Method
Object #1 Calculator
        Method #1 Calculator : Time = 4 % ,No. of calls = 1
        Method #2 windowClosed : Time = 0 % ,No. of calls = 0
        Method #3 actionPerformed : Time = 32 % ,No. of calls = 4
        Method #4 setDisplayString : Time = 16 % ,No. of calls = 6
        Method #5 getDisplayString : Time = 6 % ,No. of calls = 2
        Method #6 addDigitToDisplay : Time = 23 % ,No. of calls = 2
        Method #7 addDecimalPoint : Time = 0 % ,No. of calls = 0
        Method #8 processSignChange : Time = 0 % ,No. of calls = 0
        Method #9 clearAll : Time = 2 % ,No. of calls = 1
        Method #10 clearExisting : Time = 0 % ,No. of calls = 0
        Method #11 processOperator : Time = 2 % ,No. of calls = 1
        Method #12 processEquals : Time = 2 % ,No. of calls = 1
        Method #13 displayResult : Time = 2 % ,No. of calls = 1
        Method #14 displayError : Time = 0 % ,No. of calls = 0
        Method #15 main : Time = 5 % ,No. of calls = 1
Object #2 DivideByZeroException
        Method #1 DivideByZeroException : Time = 0 % ,No. of calls = 0
Object #3 CustomABOUTDialog
        Method #1 windowClosing : Time = 0 % ,No. of calls = 0
        Method #2 actionPerformed : Time = 0 % ,No. of calls = 0
Summary By Object
        Object #1 Calculator : Time = 100 % ,No. of calls = 20
        Object #2 DivideByZeroException : Time = 0 % ,No. of calls = 0
        Object #3 CustomABOUTDialog : Time = 0 % ,No. of calls = 0
```

ภาพที่ 6 *แสดงผลรวมของเวลาและจำนวนครั้งที่ถูกเรียกใช้แต่ละเมท็อดจากโปรแกรมเครื่องคิดเลข*

จากภาพที่ 6 จะเห็นได้ว่าเมท็อด "actionPerformed" ใช้เวลาในการทำงานมากที่สุดคือ 32 % แต่เมท็อดที่ถูกเรียกใช้มากครั้งที่สุดคือ "setDisplayString" คือ 6 ครั้ง และมีเพียงคลาส "Calculator" เพียงคลาสเดียวที่ถูกเรียกใช้งาน

4.2.2 แสดงผลรวมของเวลาและจำนวนครั้งที่ถูกเรียกใช้แต่ละเมท็อดจากโปรแกรมโปรแกรม Visitor Pattern Example

จากภาพที่ 7 สามารถสรุปได้ว่าคลาส "OrderManager" จะใช้เวลาในการทำงานมากที่สุดคือ 73% และมีจำนวนครั้งในการเรียกใช้เมท็อดในคลาสมากที่สุดคือ 3 ครั้ง

```
Summary By Method
Object #1 OrderManager
        Method #1 OrderManager : Time = 45 % ,No. of calls = 1
        Method #2 main : Time = 46 % ,No. of calls = 1
        Method #3 windowClosing : Time = 0 % ,No. of calls = 0
        Method #4 setTotalValue : Time = 0 % ,No. of calls = 0
        Method #5 getOrderVisitor : Time = 8 % ,No. of calls = 1
        Method #6 getOrderType : Time = 0 % ,No. of calls = 0
        Method #7 getOrderAmount : Time = 0 % ,No. of calls = 0
        Method #8 getTax : Time = 0 % ,No. of calls = 0
        Method #9 getSH : Time = 0 % ,No. of calls = 0
Object #2 ButtonHandler
        Method #1 actionPerformed : Time = 0 % ,No. of calls = 0
        Method #2 createOrder : Time = 0 % ,No. of calls = 0
        Method #3 ButtonHandler : Time = 100 % ,No. of calls = 2
Object #3 NonCaliforniaOrder
        Method #1 NonCaliforniaOrder : Time = 0 % ,No. of calls = 0
        Method #2 getOrderAmount : Time = 0 % ,No. of calls = 0
        Method #3 accept : Time = 0 % ,No. of calls = 0
Object #4 CaliforniaOrder
        Method #1 CaliforniaOrder : Time = 0 % ,No. of calls = 0
        Method #2 getOrderAmount : Time = 0 % ,No. of calls = 0
        Method #3 getAdditionalTax : Time = 0 % ,No. of calls = 0
        Method #4 accept : Time = 0 % ,No. of calls = 0
Object #5 OverseasOrder
        Method #1 OverseasOrder : Time = 0 % ,No. of calls = 0
        Method #2 getOrderAmount : Time = 0 % ,No. of calls = 0
        Method #3 getAdditionalSH : Time = 0 % ,No. of calls = 0
        Method #4 accept : Time = 0 % ,No. of calls = 0
Object #6 OrderVisitor
        Method #1 OrderVisitor : Time = 100 % ,No. of calls = 1
        Method #2 visit : Time = 0 % ,No. of calls = 0
        Method #3 getOrderTotal : Time = 0 % ,No. of calls = 0
Summary By Object
        Object #1 OrderManager : Time = 73 % ,No. of calls = 3
        Object #2 ButtonHandler : Time = 21 % ,No. of calls = 2
        Object #3 NonCaliforniaOrder : Time = 0 % ,No. of calls = 0
        Object #4 CaliforniaOrder : Time = 0 % ,No. of calls = 0
        Object #5 OverseasOrder : Time = 0 % ,No. of calls = 0
        Object #6 OrderVisitor : Time = 5 % ,No. of calls = 1
```

ภาพที่ 7 *แสดงผลรวมหลังจากโปรแกรม Visitor Pattern Example เริ่มทำงาน*

5. สรุปและแนวทางการพัฒนาต่อไป

ในงานวิจัยนี้ได้นำเทคนิค AOP มาใช้ในการสร้าง Sequence Diagram ให้เกิดขึ้น ณ ขณะที่โปรแกรมทำงาน ทำให้เราสามารถมองเห็นภาพได้ชัดเจนมากขึ้นว่าโปรแกรมมีการทำงานอย่างไรเห็นลำดับการทำงานก่อนหลังว่ามีการเรียกใช้เมท็อดไหนก่อนและสามารถเปรียบเทียบการใช้เวลาได้จาก Lifetime ที่เกิดบน Lifeline ด้วยว่ามีเมท็อดไหนที่ใช้เวลามากน้อยเพียงใด และนำข้อมูลที่ได้มาทำการสรุปว่าคลาสแต่ละคลาสหรือเมท็อดแต่ละเมท็อดใช้เวลาทำงานไปเป็นกี่เปอร์เซนต์ของเวลาที่ทำงานทั้งหมด และมีจำนวนที่ถูกเรียกใช้ทั้งหมดเท่าไร ทำให้สามารถนำมาวิเคราะห์ต่อได้ว่าเมท็อดไหนที่ควรปรับปรุงหรือเมท็อดไหนที่ไม่ต้องการใช้ก็สามารถตัดทิ้งได้เพื่อเป็นการประหยัดเวลาในการพัฒนาโปรแกรม โดยที่การใช้เทคนิคการเขียนโปรแกรมเชิงลักษณะนี้จะไม่มีแก้ไขใดๆ กับโค้ดเดิมเมื่อเปรียบเทียบกับงานวิจัยของ Douence และคณะ [10] ที่ต้องทำการแทรกโค้ดลงในโค้ดเดิมซึ่งจะทำให้เกิดความซับซ้อนขึ้นในโค้ดเดิม แต่

เทคนิคการเขียนโปรแกรมเชิงลักษณะนี้จะไม่ทำให้โค้ดเดิมนั้นเกิดความซับซ้อน

แนวทางที่สามารถนำไปพัฒนาต่อได้คือรูปแบบของไฟล์ AspectJ ถ้านำไปพัฒนาต่อได้โดยที่ไม่ต้องใช้การสแกนหาชื่อคลาสและชื่อเมท็อดก่อน โดยใช้ไฟล์ AspectJ เป็นตัวหาชื่อคลาสและเมท็อดเลยทำให้สามารถลดขั้นตอนการใช้งานโปรแกรมลงได้และมีความแม่นยำในการสร้าง Sequence Diagram มากยิ่งขึ้น

6. เอกสารอ้างอิง

[1] Julie Waterhouse and Mik Kersten, "Aspect-Oriented Programming with AspectJ," *Cascon*, 2004.

[2] AspectJ. It is an aspect-oriented extension created at PARC for the Java programming language. Available at http://www.eclipse.org/aspectj/

[3] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier and John Irwin, "Aspect-Oriented Programming," *In proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, Finland, Springer-Verlag LNCS 1241, June 1997.

[4] Ali Mesbah and Arie van Deursen, "Crosscutting Concerns in J2EE Applications," http://java.sun.com/j2ee/

[5] Isaac Yuen, "Improving Software Modularity Through Crosscutting Concern Extraction," April, 2009.

[6] Erik Hilsdale and Jim Hugunin, "Advice Weaving in AspectJ," *In AOSD '04*, Lancaster, UK, March, 2004.

[7] Grady Booch, James Rumbaugh and Ivar Jacobson, "The Unified Modeling Language User Guide - Booch-Rumbaugh-Jacobson," *In Part III chapter*.14

[8] W. Scott Ambler, "The Elements of UML Style," *ch*.6

[9] Remi Douence and Mario Südholt, "A model and a tool for Event-based Aspect-Oriented Programming," *french version accepted at LMO'03, 2nd edition*, December, 2002.

[10] Remi Douence, Pascal Fradet and Mario Südholt, "Trace-based Aspects," *AOSD*, 2004.

[11] Mark Richters and Martin Gogolla, "AspectOriented Monitoring of UML and OCL Constraints", *In AOSD Modeling With UML Workshop, 6th International Conference on the Unified Modeling Language*, 2000.

[12] A Brian Malloy and F. James Power, "Exploiting UML dynamic object modeling for the visualization of C++ programs," *ACM*, 2005.

[13] Roberta Coelho, Ayla Dantas, Uirá Kulesza, Walfredo Cirne, Arndt von Staa and Carlos Lucena, "The Application Monitor Aspect Pattern," *PLoP '*06, Portland, OR, USA, October 21–23, 2006.

[14] Andy Kellens, Kim Mens, Johan Brichau and Kris Gybels, "Managing the Evolution of Aspect-Oriented Software with Model-based Pointcuts," *Ph.D. scholarship funded by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT Vlaanderen)*.

[15] Examples Java code Internet: http://richardbowles.tripod.com/java/menu.htm

[16] Examples Java code Internet: http://www.java2s.com/Code/Java