การประยุกต์แบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่องกับกระบวนการดับเบิลยูเอส-บีเพล

นายทวี ไทยส่งสุวรรณ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2554

APPLYING PATTERNS FOR FAULT TOLERANT SOFTWARE TO WS-BPEL PROCESSES

Mr. Thawee Thaisongsuwan

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering Program in Computer Engineering
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2011

หัวข้อวิทยานิพนธ์	การประยุกต์แบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่องกับกระบวนการดับเบิลยูเอส-บีเพล
โดย	นายทวี ไทยส่งสุวรรณ
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	รองศาสตราจารย์ ดร.ทวิตีย์ เสนีวงศ์ ณ อยุธยา

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

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

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(อาจารย์ ดร.ยรรยง เต็งอำนวย)

..... อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก
(รองศาสตราจารย์ ดร.ทวิตีย์ เสนีวงศ์ ณ อยุธยา)

..... กรรมการภายนอกมหาวิทยาลัย
(ผู้ช่วยศาสตราจารย์ ดร.เบญจพร ลิ้มธรรมาภรณ์)

ทวี ไทยส่งสุวรรณ : การประยุกต์แบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่องกับกระบวนการดับเบิลยูเอส-บีเพล. (APPLYING PATTERNS FOR FAULT TOLERANT SOFTWARE TO WS-BPEL PROCESSES) อ.ที่ปรึกษาวิทยานิพนธ์หลัก : รศ. ดร.ทวิตีย์ เสนีวงศ์ ณ อยุธยา, 77 หน้า

เทคโนโลยีเว็บเซอร์วิซสนับสนุนการประกอบกันของเว็บเซอร์วิซในลักษณะของกระแสงานซึ่งจะแสดงกระบวนการทางธุรกิจ ดับเบิลยูเอส-บีเพลเป็นภาษามาตรฐานเพื่อใช้ในการกำหนดกระแสงานของการเรียกใช้เว็บเซอร์วิซตามกระบวนการทางธุรกิจ และอาศัยเครื่องประมวลบีเพลในการควบคุมการทำงาน อย่างไรก็ตามในช่วงที่มีการเรียกใช้เว็บเซอร์วิซ อาจพบความผิดพร่องที่ซ่อนอยู่ในระบบและส่งผลให้เกิดความผิดพลาดได้ เช่น เว็บเซอร์วิซคู่ค้าไม่สามารถตอบกลับได้ หรือตอบกลับมาไม่ทันเวลาที่กำหนด ซึ่งจะส่งผลให้เกิดความเสียหายให้กับองค์กรธุรกิจหากกระบวนการนั้นมีความวิกฤติและจำเป็นต้องดำเนินต่อไปให้เสร็จสิ้น เพื่อให้ระบบยังสามารถดำเนินงานต่อไปได้จำเป็นต้องออกแบบให้กระบวนการบีเพลรองรับการทนต่อความผิดพร่องที่เกิดขึ้น โดยสามารถใช้แบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่องมาเป็นแนวทางในการออกแบบได้

งานวิจัยนี้เสนอกระบวนการในการนำแบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่องมาประยุกต์ใช้กับการออกแบบกระแสงานบีเพล ตัวอย่างเช่น แบบรูปเชิงสถาปัตยกรรม แบบรูปการตรวจหา และแบบรูปการกู้ระบบจากความผิดพลาด สามารถนำมาใช้จัดการกับความผิดพร่องจากเซอร์วิซคู่ค้าได้ จากแบบรูปในกลุ่มดังกล่าวนำมาวิเคราะห์และคัดเลือกเพื่อหาแบบรูปที่มีความเหมาะสมในการนำมาใช้ในระดับกระแสงานบีเพล กล่าวคือสามารถใช้เพียงโครงสร้างของบีเพลในการออกแบบตามแบบรูปได้ และแสดงเป็นแม่แบบของโครงสร้างบีเพล เพื่อเป็นแนวทางให้ผู้ออกแบบสามารถนำไปใช้กับชุดเครื่องมือพัฒนาและสภาพแวดล้อมของบีเพลทั่วไปได้ นอกจากนี้ยังได้เสนอตัวอย่างของการประยุกต์ใช้แบบรูป และทำการทดสอบความเชื่อถือได้และผลกระทบต่อสมรรถนะของกระแสงานบีเพลหลังจากที่มีการประยุกต์ใช้แบบรูปแล้ว

ภาควิชา .....วิศวกรรมคอมพิวเตอร์..... ลายมือชื่อนิสิต .....

สาขาวิชา.....วิศวกรรมคอมพิวเตอร์..... ลายมือชื่อ อ. ที่ปรึกษาวิทยานิพนธ์หลัก .....

ปีการศึกษา .....2554.....

5170312821 : MAJOR COMPUTER ENGINEERING

KEYWORD : WS-BPEL / FAULT TOLERANCE PATTERN / WEB SERVICES

THAWEE THAISONGSUWAN : APPLYING PATTERNS FOR FAULT TOLERANT SOFTWARE TO WS-BPEL PROCESSES. ADVISOR : ASSOC. PROF. TWITTIE SENIVONGSE, Ph.D., 77 pp.

Web services technology supports composition of services into workflow of business processes. WS-BPEL is a standard language for orchestrating workflow of Web services invocation for business processes. Those processes would be executed by a WS-BPEL execution engine. During service invocation, errors may occur from latent faults, e.g. partner services may not be available or not respond within time. These failures interrupt the workflow and may cause damages to an organization if the business process is critical and needs to proceed to completion. WS-BPEL processes should be designed to support fault tolerance by using patterns for fault tolerant software as a design guideline.

This research proposes an approach to applying patterns for fault tolerant software to WS-BPEL. Fault tolerance patterns, such as those for architectural design, error detection, and error recovery, can be used to handle partner service faults. Selected patterns from those groups can be applied to WS-BPEL in workflow level. Templates of WS-BPEL constructs for implementing fault handling logic according to the patterns will be presented. With this approach, business processes can be made to execute in a fault tolerant manner within standard WS-BPEL execution environment and development tools. Examples of how the selected fault tolerance patterns are applied to WS-BPEL workflow are given. In addition, tests are conducted to compare reliability and performance of normal workflows and that of their fault tolerant versions.

Department: .....Computer Engineering..... Student's Signature.....

Field of Study: ....Computer Engineering..... Advisor's Signature.....

Academic Year: .....2011.....

กิตติกรรมประกาศ

ขอขอบพระคุณรองศาสตราจารย์ ดร.ทวิตีย์ เสนีวงศ์ ณ อยุธยา อาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่สละเวลาคอยให้คำปรึกษา คำแนะนำ ข้อคิด และความช่วยเหลือต่างๆ อันมีค่าอย่างยิ่งตลอดระยะเวลาการศึกษาและการวิจัย ทำให้วิทยานิพนธ์นี้สำเร็จลุล่วงได้ด้วยดี

ขอขอบพระคุณอาจารย์ ดร.ยรรยง เต็งอำนวย ประธานคณะกรมการสอบวิทยานิพนธ์ และผู้ช่วยศาสตราจารย์ ดร.เบญจพร ลิ้มธรรมาภรณ์ กรรมการสอบวิทยานิพนธ์ที่ให้ข้อชี้แนะในการปรับปรุงงานวิทยานิพนธ์ให้มีคุณภาพยิ่งขึ้น

ขอขอบพระคุณคณาจารย์ทุกท่านในภาควิชาวิศวกรรมคอมพิวเตอร์ที่ได้อบรม สั่งสอน ให้ความรู้ต่างๆ ซึ่งเป็นประโยชน์ต่อการทำวิจัยและการทำงานในอนาคต

ขอขอบคุณนายเอกวิชญ์ กุลธวัชวิชัย นายวีรภัทร พรหมชนะ นางสาวรัศมีทิพย์ วิตาและเพื่อนๆ พี่ๆ น้องๆ สมาชิกห้องปฏิบัติการวิศวกรรมระบบสารสนเทศ (ISEL) ที่คอยให้ความช่วยเหลือแบ่งปันข้อคิดในการทำวิจัย รวมทั้งแลกเปลี่ยนประสบการณ์ต่างๆ ทั้งสาระและความบันเทิงตลอดช่วงที่ทำวิจัยนี้

ท้ายที่สุดนี้ขอขอบพระคุณ คุณแม่ และครอบครัวที่เป็นกำลังใจและแรงสนับสนุนสำคัญในด้านการศึกษาจนทำให้ประสบความสำเร็จมาได้ถึงทุกวันนี้

สารบัญ

หน้า

บทคัดย่อภาษาไทย ..... ง

บทคัดย่อภาษาอังกฤษ ..... จ

กิตติกรรมประกาศ ..... ฉ

สารบัญ ..... ช

สารบัญตาราง ..... ฎ

สารบัญภาพ ..... ฐ

บทที่

1 บทนำ 1

1.1 ความเป็นมาและความสำคัญของปัญหา 1

1.2 วัตถุประสงค์ของการวิจัย 2

1.3 ขอบเขตของการวิจัย 2

1.4 ประโยชน์ที่คาดว่าจะได้รับ 3

1.5 วิธีดำเนินการวิจัย 3

1.6 ผลงานตีพิมพ์ 3

2 ทฤษฎีและงานวิจัยที่เกี่ยวข้อง 5

2.1 ดับเบิลยูเอส-บีเพล (WS-BPEL) 5

2.2 แบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่อง (Patterns for Fault Tolerant Software) 7

2.3 งานวิจัยที่เกี่ยวข้อง 9

2.3.1 งานวิจัยที่เกี่ยวกับการแบ่งประเภทความผิดพร่องของเว็บเซอร์วิซประกอบ 9

บทที่ หน้า

2.3.2 งานวิจัยที่เกี่ยวกับการทำให้ทนต่อความผิดพร่องของดับเบิลยูเอส-บีเพล 11

2.3.3 งานวิจัยที่เกี่ยวกับแบบรูปสำหรับการจัดการกับความผิดพร่อง 13

3 การวิเคราะห์แบบรูป 14

3.1 แบบรูปเชิงสถาปัตยกรรม (Architectural Patterns) 17

3.1.1 Units of Mitigation 17

3.1.2 Correcting Audits 18

3.1.3 Redundancy 18

3.1.4 Recovery Blocks 19

3.1.5 Minimize Human Intervention 20

3.1.6 Maximize Human Participation 20

3.1.7 Maintenance Interface 20

3.1.8 Someone in Charge 21

3.1.9 Escalation 21

3.1.10 Fault Observer 22

3.1.11 Software Update 22

3.2 แบบรูปการตรวจหา (Detection Patterns) 24

3.2.1 Fault Correlation 24

3.2.2 Error Containment Barrier 25

3.2.3 Complete Parameter Checking 25

3.2.4 System Monitor 26

3.2.5 Heartbeat 26

3.2.6 Acknowledgement 27

3.2.7 Watchdog 27

บทที่ หน้า

3.2.8 Realistic Threshold....28
3.2.9 Existing Metrics....29
3.2.10 Voting....29
3.2.11 Routine Maintenance....29
3.2.12 Routine Exercises....30
3.2.13 Routine Audits....30
3.2.14 Checksum....31
3.2.15 Riding Over Transients....31
3.2.16 Leaky Bucket Counter....32
3.3 แบบรูปการกู้ระบบจากความผิดพลาด (Error Recovery Patterns)....34
3.3.1 Quarantine....34
3.3.2 Concentrated Recovery....35
3.3.3 Error Handler....35
3.3.4 Restart....35
3.3.5 Rollback....36
3.3.6 Roll-Forward....36
3.3.7 Return to Reference Point....37
3.3.8 Limit Retries....38
3.3.9 Failover....38
3.3.10 Checkpoint....39
3.3.11 What to Save....40
3.3.12 Remote Storage....40
3.3.13 Individuals Decide Timing....40
3.3.14 Data Reset....41

บทที่ หน้า

3.4 แบบรูปที่ประยุกต์ใช้ในระดับกระแสงานบีเพล 43
4 การประยุกต์ใช้แบบรูป 44
4.1 แบบรูป Units of Mitigation และ แบบรูป Error Handler 44
4.2 แบบรูป Escalation 44
4.3 แบบรูป Recovery Block และ แบบรูป Limit Retries 45
4.4 แบบรูป Voting 47
4.5 แบบรูป Acknowledgement 48
4.6 แบบรูป Heartbeat 50
4.7 แบบรูป Rollback 51
4.8 แบบรูป Roll-Forward 53
4.9 แบบรูป Return to Reference Point 53
4.10 แบบรูป Correcting Audits และ แบบรูป Complete Parameter Checking 54
5 การทดลองกับกระแสงานบีเพล 55
5.1 การทดลอง 55
5.2 การทดสอบ 60
5.2.1 การทดสอบความเชื่อถือได้ 60
5.2.2 การทดสอบผลกระทบต่อสมรรถนะ 64
6 บทสรุป 70
6.1 สรุปผลการวิจัย 70
6.2 ปัญหาและข้อจำกัดที่พบจากการวิจัย 71
6.2.1 ข้อจำกัดของเครื่องมือพัฒนา 71
6.2.2 ปัญหาจากความแตกต่างกันของชุดเครื่องมือพัฒนา 71
6.2.3 ข้อจำกัดของภาษาบีเพล 72

บทที่ หน้า

6.3 ข้อเสนอแนะ 72

รายการอ้างอิง..... 73

ประวัติผู้เขียนวิทยานิพนธ์ 77

สารบัญตาราง

ตารางที่ หน้า

3.1 ระดับการประยุกต์ใช้กับบีเพลของกลุ่มแบบรูปเชิงสถาปัตยกรรม..... 23

3.2 ระดับการประยุกต์ใช้กับบีเพลของกลุ่มแบบรูปการตรวจหา 32

3.3 ระดับการประยุกต์ใช้กับบีเพลของกลุ่มแบบรูปการกู้ระบบจากความผิดพลาด..... 42

5.1 การคำนวณหาอัตราความล้มเหลวที่จะนำมาใช้ในกรณีของเว็บเซอร์วิซสภาพอากาศ..... 63

5.2 ค่าเฉลี่ยของเวลาที่ใช้ตอบกลับของกระแสงานตามแบบรูปและความผิดพร่องที่ใช้ทดสอบ 68

สารบัญภาพ

ภาพที่ หน้า

2.1 กระบวนการบีเพลการอนุมัติเงินกู้ 6
2.2 ขั้นตอนการทนต่อความผิดพร่อง [10] 7
2.3 ตัวอย่างความสัมพันธ์ของแบบรูป 8
3.1 แผนที่ภาษาแบบรูปรวมทั้งหมด [10] 15
3.2 แผนที่ภาษาแบบรูปเฉพาะ 3 กลุ่มแรก 16
3.3 แผนที่ภาษาแบบรูปเชิงสถาปัตยกรรม [10] 17
3.4 แผนที่ภาษาแบบรูปการตรวจหา [10] 24
3.5 แผนที่ภาษาแบบรูปการกู้ระบบจากความผิดพลาด [10] 34
3.6 แผนที่ภาษาแบบรูปที่นำมาใช้ในระดับกระแสงานบีเพลได้ 43
4.1 แม่แบบของบีเพลที่ใช้ <scope> และ <faultHandlers> 45
4.2 แม่แบบของบีเพลตามแบบรูป *Escalation* 45
4.3 แม่แบบของบีเพลตามแบบรูป *Recovery Block* 46
4.4 แม่แบบของบีเพลในการทำงานซ้ำ 47
4.5 แม่แบบของบีเพลตามแบบรูป *Voting* 48
4.6 แม่แบบของบีเพลฝั่งผู้รับคำร้องตามแบบรูป *Acknowledgement* 49
4.7 แม่แบบของบีเพลฝั่งผู้ส่งคำร้องตามแบบรูป *Acknowledgement* 50
4.8 แม่แบบของบีเพลของฝั่งที่ถูกเฝ้าดูตามแบบรูป *Heartbeat* 50
4.9 แม่แบบของบีเพลของฝั่งที่เฝ้าดูตามแบบรูป *Heartbeat* 51
4.10 ตัวอย่างการใช้ <compensationHandler> 52
4.11 แม่แบบของบีเพลตามแบบรูป *Roll-Forward* 53
5.1 กระแสงานบีเพล CityInfo 55
5.2 พาร์ตเนอร์ลิงก์ไทป์ในวิสเดิลของกระแสงานบีเพล CityInfo 56

ภาพที่ หน้า

5.3 เอกซ์เอ็มแอลสกีมาของกระแสงานบีเพล CityInfo 56
5.4 แผนที่ภาษาแบบรูปที่นำมาใช้กับกระแสงานบีเพล CityInfo 57
5.5 กระแสงานบีเพล CityInfo ที่ประยุกต์ตามแบบรูปแล้ว..... 58
5.6 โค้ดเทียมตามอัลกอริทึม “Exact plurality voting” 59
5.7 กระแสงานบีเพล CityInfo แบบปกติที่ใช้ทดสอบ 61
5.8 การทดสอบด้วยโปรแกรม soapUI ของกระแสงานบีเพล CityInfo แบบปกติ 61
5.9 กระแสงานบีเพล CityInfo แบบประยุกต์ใช้แบบรูปที่ใช้ทดสอบ 62
5.10 การทดสอบด้วยโปรแกรม soapUI ของกระแสงานบีเพล CityInfo ที่ปรับใช้แบบรูปแล้ว..... 64
5.11 อัตราความสำเร็จของกระแสงานบีเพล CityInfo 64
5.12 การจำลองเว็บเซอร์วิซสภาพอากาศในการทดสอบผลกระทบต่อสมรรถนะ..... 65
5.13 กระแสงานในการทดสอบที่ไม่ใช้แบบรูป 66
5.14 กระแสงานในการทดสอบตามแบบรูป *Error Handler*..... 66
5.15 กระแสงานในการทดสอบตามแบบรูป *Recovery Block* 67
5.16 กระแสงานในการทดสอบตามแบบรูป *Recovery Block* แบบเรียกซ้ำ..... 67
5.17 กระแสงานในการทดสอบตามแบบรูป *Voting* 68
5.18 ค่าเฉลี่ยของเวลาที่ใช้ตอบกลับของกระแสงานตามแบบรูปและความผิดพร่องที่ใช้ทดสอบ 69

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

การพัฒนาซอฟต์แวร์ในปัจจุบันมีแนวโน้มเข้าสู่การใช้หลักการของสถาปัตยกรรมเชิงบริการหรือเอสโอเอ (Service-Oriented Architecture: SOA) ซึ่งพัฒนาซอฟต์แวร์ให้เป็นบริการ (Service) โดยแต่ละบริการมีหน้าที่เฉพาะตัว และใช้การนำบริการต่างๆ มาประกอบกันเพื่อสร้างเป็นแอปพลิเคชันของทั้งระบบ การพัฒนาแอปพลิเคชันใหม่ๆ จึงสามารถทำได้รวดเร็ว และการปรับเปลี่ยนแก้ไขก็สามารถทำได้สะดวกและมีความยืดหยุ่น โดยสามารถทำการถอดประกอบบริการใหม่ๆ เข้าไปได้

เทคโนโลยีเว็บเซอร์วิซ (Web Services) เป็นเทคโนโลยีหนึ่งที่สามารถตอบสนองการพัฒนาซอฟต์แวร์ตามแนวคิดของเอสโอเอ โดยบริการซึ่งเรียกว่า เว็บเซอร์วิซ จะให้บริการกับผู้เรียกใช้ผ่านส่วนต่อประสาน (Interface) ที่เป็นมาตรฐานที่เรียกว่า วิสเดิล (Web Service Definition Language: WSDL) เทคโนโลยีเว็บเซอร์วิซสนับสนุนการประกอบกันของเว็บเซอร์วิซในลักษณะของกระแสงาน (Workflow) ซึ่งจะแสดงกระบวนการทางธุรกิจ (Business Process) ตัวอย่างเช่น เว็บเซอร์วิซการท่องเที่ยวสามารถสร้างได้จากการกำหนดกระแสงานของการเรียกใช้เว็บเซอร์วิซต่างๆ เช่น เว็บเซอร์วิซจองห้องพัก เว็บเซอร์วิซจองตั๋วเครื่องบิน และเว็บเซอร์วิซจองรถยนต์เช่า เป็นต้น

ภาษามาตรฐานที่ใช้ในการกำหนดกระแสงานของการเรียกใช้เว็บเซอร์วิซต่างๆ ตามกระบวนการทางธุรกิจคือ ภาษาดับเบิลยูเอส-บีเพล (Web Services Business Process Execution Language : WS-BPEL) [1] หรือเรียกสั้นๆ ว่าบีเพล อาศัยการประมวลผลของเครื่องประมวลบีเพล (BPEL Engine) ในการควบคุมการทำงานและการเรียกใช้เว็บเซอร์วิซต่างๆ ตามที่กำหนดไว้ในกระแสงานเพื่อสร้างเป็นเว็บเซอร์วิซประกอบ (Composite Web Service) จากการที่สามารถประกอบเว็บเซอร์วิซให้ทำงานร่วมกันได้อย่างสะดวก องค์กรธุรกิจเป็นจำนวนมากจึงเริ่มให้ความสนใจในการสร้างแอปพลิเคชันตามกระบวนการทางธุรกิจในลักษณะนี้ อย่างไรก็ตามในระหว่างการประมวลผลบีเพลอาจพบความขัดข้องได้ เช่น ไม่สามารถติดต่อเว็บเซอร์วิซที่กำหนดไว้ในกระแสงาน หรือข้อมูลที่รับส่งระหว่างกันเกิดข้อผิดพลาด ทำให้กระแสงานไม่สามารถดำเนินต่อไปได้ สภาพเช่นนี้จะก่อให้เกิดความเสียหายให้กับองค์กรธุรกิจหากกระแสงานนั้นมีความวิกฤตและจำเป็นต้องดำเนินต่อไปให้เสร็จสิ้น

งานวิจัยหลายงานได้นำเสนอวิธีการทำให้เว็บเซอร์วิซประกอบทนต่อความผิดพร่อง ซึ่งใช้วิธีการต่างๆ ตั้งแต่การใช้ความสามารถของบีเพลเพียงอย่างเดียวในการรองรับการทนต่อความผิด

พร่อง [2, 3, 4] แต่เนื่องจากการใช้โครงสร้างของบีเพลเพียงอย่างเดียวอาจยังมีข้อจำกัดอยู่ จึงมีงานวิจัยที่สร้างส่วนต่อขยายให้กับเครื่องประมวลบีเพล [5] และสร้างระบบพื้นฐานขึ้นมาเพื่อรองรับการเฝ้าสังเกตความผิดพร่อง (Fault Monitoring) และการกู้คืนระบบจากความผิดพร่อง (Fault Recovery) [6] อีกแนวทางหนึ่งคือการใช้พร็อกซี (Proxy) เป็นตัวช่วยในการเลือกเว็บเซอร์วิซที่สามารถทำงานแทนกันหรือหาผลลัพธ์ของการทำงานที่ถูกต้องให้จากกลุ่มของเว็บเซอร์วิซที่เหมือนกัน [7, 8, 9] แต่ละวิธีการนั้นมีทั้งข้อดีข้อด้อยขึ้นกับความเหมาะสมในการนำไปใช้งานและประเภทของความผิดพร่องที่จะรองรับได้ จากความหลากหลายของวิธีการเหล่านี้ทำให้ผู้ออกแบบกระบวนการทางธุรกิจพิจารณาได้ยากว่าควรจะเลือกใช้วิธีการใดมาปรับใช้กับกระบวนการทางธุรกิจของตน

Hanmer ได้เสนอแบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่อง (Patterns for Fault Tolerant Software) [10] เป็นการรวบรวมแบบรูปต่างๆ เพื่อเป็นแนวทางในการออกแบบซอฟต์แวร์ให้สามารถทนต่อความผิดพร่องได้ แบบรูปต่างๆ จะระบุถึงปัญหา บริบท และแนวทางการแก้ไข ซึ่งสามารถนำแบบรูปไปใช้ซ้ำกับการออกแบบซอฟต์แวร์ใดๆ ได้ ผู้วิจัยจึงมีแนวคิดที่จะทำการประยุกต์แบบรูปเหล่านั้นมาปรับใช้กับการออกแบบบีเพลให้ทนต่อความผิดพร่อง โดยในขั้นต้นจะทำการพิจารณาว่าแบบรูปใดบ้างที่สามารถนำมาประยุกต์ในระดับของกระแสงานบีเพล และจะเสนอเป็นแม่แบบของกระแสงานบีเพลที่ออกแบบตามแนวคิดของแต่ละแบบรูปนั้นและใช้เพียงโครงสร้างมาตรฐานของบีเพล เพื่อเป็นแนวทางให้กับผู้ออกแบบกระบวนการทางธุรกิจสามารถนำไปใช้ในการออกแบบกระแสงานบีเพลที่ทนต่อความผิดพร่องในสภาพแวดล้อมของบีเพลทั่วไปได้

1.2 วัตถุประสงค์ของการวิจัย

1.2.1 เพื่อทำการศึกษาและประเมินความสามารถของบีเพลตามมาตรฐานดับเบิลยูเอส-บีเพล 2.0 และเครื่องประมวลบีเพลแบบโอเพนซอร์ซ ในการทำให้ทนต่อความผิดพร่องตามแบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่อง

1.2.2 เพื่อนำเสนอแบบรูปในการออกแบบบีเพลที่ทนต่อความผิดพร่อง

1.3 ขอบเขตของการวิจัย

1.3.1 วิเคราะห์ความสามารถในการทนต่อความผิดพร่องโดยใช้แบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่องในกลุ่มต่อไปนี้เป็นอย่างน้อย

1.3.1.1 แบบรูปเชิงสถาปัตยกรรม (Architectural Patterns)

1.3.1.2 แบบรูปการตรวจหา (Detection Patterns)

1.3.1.3 แบบรูปการกู้ระบบจากความผิดพลาด (Error Recovery Patterns)

1.3.2 พิจารณาความผิดพร่องที่เกิดจากเซอร์วิซคู่ค้า (Partner Service Fault) เป็นอย่างน้อย

1.3.3 พิจารณาบีเพลตามมาตรฐานดับเบิลยูเอส-บีเพล 2.0

1.3.4 เครื่องประมวลบีเพลแบบโอเพนซอร์ซที่นำมาพิจารณาได้แก่ Apache ODE และ Glassfish ESB

1.4 ประโยชน์ที่คาดว่าจะได้รับ

ได้แบบรูปสำหรับบีเพลที่ทนต่อความผิดพร่อง ซึ่งสามารถนำไปช่วยในการออกแบบกระแสงานบีเพลให้มีความสามารถในการทนต่อความผิดพร่อง

1.5 วิธีดำเนินการวิจัย

1.5.1 ศึกษาโครงสร้างของดับเบิลยูเอส-บีเพลและข้อมูลการใช้งานเครื่องประมวลบีเพลแบบโอเพนซอร์ซ

1.5.2 ศึกษาแบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่อง

1.5.3 วิเคราะห์ดับเบิลยูเอส-บีเพลและเครื่องประมวลบีเพลแบบโอเพนซอร์ซกับแบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่อง

1.5.4 วิเคราะห์งานวิจัยที่เกี่ยวข้องกับการทำให้บีเพลทนต่อความผิดพร่องกับแบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่อง

1.5.5 กำหนดแบบรูปสำหรับบีเพลที่ทนต่อความผิดพร่อง

1.5.6 ทดสอบการใช้งานแบบรูปสำหรับบีเพลที่ทนต่อความผิดพร่อง

1.5.7 จัดทำวิทยานิพนธ์

1.6 ผลงานตีพิมพ์

ส่วนหนึ่งของวิทยานิพนธ์นี้ได้ตีพิมพ์และนำเสนอในการประชุมวิชาการดังนี้

1.6.1 บทความชื่อ *“Fault Tolerant WS-BPEL Business Processes”* [11]

1.6.1.1 ชื่อผู้แต่ง Thawee Thaisongsuwan และ Twittie Senivongse

1.6.1.2 ตีพิมพ์และนำเสนอในงานประชุมวิชาการชื่อ The 12th National Computer Science and Engineering Conference (NCSEC2008) ซึ่งจัดขึ้นในวันที่ 20-21 พฤศจิกายน 2551 ณ จ.ชลบุรี ประเทศไทย

1.6.2 บทความชื่อ *"Applying Software Fault Tolerance Patterns to WS-BPEL Processes"* [12]

1.6.2.1 ชื่อผู้แต่ง Thawee Thaisongsuwan และ Twittie Senivongse

1.6.2.2 ตีพิมพ์และนำเสนอในงานประชุมวิชาการชื่อ 2011 Eighth International Joint Conference on Computer Science and Software Engineering (JCSSE) ซึ่งจัดขึ้นในวันที่ 11-13 พฤษภาคม 2554 ณ จ.นครปฐม ประเทศไทย

บทที่ 2

ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 ดับเบิลยูเอส-บีเพล (WS-BPEL)

ดับเบิลยูเอส-บีเพล (Web Services Business Process Execution Language: WS-BPEL) [1] หรือ บีเพล เป็นภาษามาตรฐานที่อยู่ภายใต้การดูแลของโอเอซิส (Organization for the Advancement of Structured Information Standards: OASIS) มีรูปแบบเป็นภาษาเอกซ์เอ็มแอล (XML) ใช้อธิบายความหมายและการดำเนินงานของกระบวนการทางธุรกิจที่มีการเรียกใช้เว็บเซอร์วิซ โดยอยู่บนพื้นฐานของการประกอบกันของเว็บเซอร์วิซต่างๆ ที่บรรยายไว้ด้วยภาษาวิสเดิล (Web Service Definition Language: WSDL) [13] เมื่อนำมาประกอบกันแล้วสามารถบรรยายกระบวนการทางธุรกิจนั้นด้วยภาษาวิสเดิลเพื่อกำหนดให้เป็นเว็บเซอร์วิซประกอบ (Composite Web Service) ขึ้นมาได้

บีเพลสามารถบรรยายลักษณะของกระบวนการทางธุรกิจได้ 2 แนวทางคือ

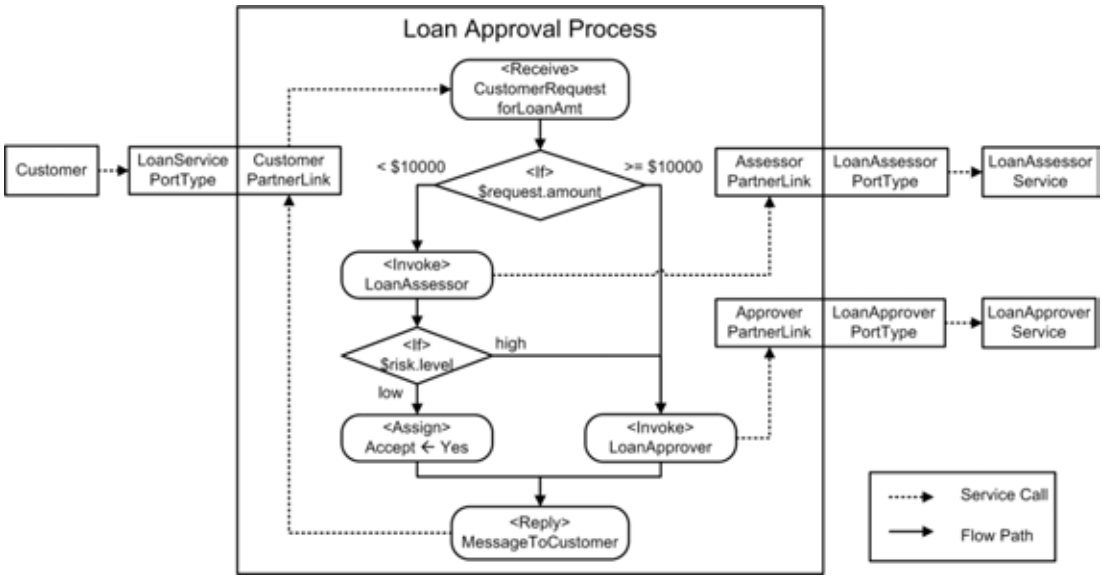
- แบบนามธรรม (Abstract) จะบอกเพียงสิ่งที่กระบวนการนั้นสามารถทำได้ ข้อมูลขาเข้าและข้อมูลขาออก แต่จะไม่บรรยายถึงขั้นตอนต่างๆ ในกระบวนการนั้น
- แบบดำเนินการได้ (Executable) จะบอกข้อมูลเช่นเดียวกับแบบนามธรรมรวมถึงวิธีการทำงานจริงๆ ของกระบวนการนั้นด้วย

ภายในกระบวนการทางธุรกิจของบีเพลจะใช้พาร์ตเนอร์ลิงก์ (<partnerLink>) เพื่อบรรยายถึงบทบาทของเว็บเซอร์วิซต่างๆ ที่เกี่ยวข้อง โดยพาร์ตเนอร์ลิงก์จะอ้างอิงตามพาร์ตเนอร์ลิงก์ไทป์ (<partnerLinkType>) ที่กำหนดไว้ในวิสเดิลของแต่ละเว็บเซอร์วิซ รวมทั้งวิสเดิลของกระบวนการบีเพลเองด้วย

บีเพลประกอบไปด้วยกิจกรรม (Activity) ต่างๆ เชื่อมต่อกัน อยู่ในรูปแบบของกระแสงาน โดยแบ่งกิจกรรมออกเป็น 2 ประเภท ได้แก่

- กิจกรรมพื้นฐาน (Basic Activity) หมายถึงการทำงานพื้นฐานของกระบวนการ เช่น <invoke>, <receive>, <reply>, <assign> เป็นต้น
- กิจกรรมตามโครงสร้าง (Structured Activity) หมายถึงการนำกิจกรรมพื้นฐานต่างๆ มารวมเป็นกิจกรรมใหม่ให้เป็นการทำงานที่ซับซ้อนยิ่งขึ้น เช่น <sequence>, <flow>, <if>, <switch>, <while>, <pick> เป็นต้น

ตัวอย่างของกระแสงานบีเพลแสดงไว้ดังภาพที่ 2.1 เป็นกระแสงานเพื่อขออนุมัติเงินกู้ [1] (เนื่องจากบีเพลไม่มีรูปสัญลักษณ์จึงใช้รูปของกระแสงานแทน) การทำงานเริ่มต้นจะรับคำร้องการขอเงินกู้มาจากลูกค้าผ่านทางพอร์ตไทป์ (<portType>) ที่กำหนดไว้ในวิสเดิลของกระบวนการนี้ และจะพิจารณาจำนวนเงินจากคำร้องนั้นถ้ามีจำนวนเงินน้อยกว่า 10,000 เหรียญจะทำการเรียกเว็บเซอร์วิซที่ทำหน้าที่ประเมินความเสี่ยง (InvokeLoanAssessor) ผ่านทางพอร์ตไทป์ในวิสเดิลของเว็บเซอร์วิซผู้ประเมิน เมื่อได้รับคำตอบแล้วจะตรวจสอบผลลัพธ์ที่ได้ หากได้ผลว่ามีความเสี่ยงต่ำจะอนุมัติคำร้องนี้และส่งข้อมูลกลับไปยังลูกค้า แต่ถ้าผลลัพธ์ที่ได้บอกว่ามีความเสี่ยงสูงจะทำการเรียกเว็บเซอร์วิซผู้อนุมัติ (InvokeLoanApprover) ต่อไป รวมทั้งถ้าจำนวนเงินที่ขอนั้นมากกว่าหรือเท่ากับ 10,000 เหรียญจะทำการเรียกเว็บเซอร์วิซผู้อนุมัติด้วย ซึ่งจะได้ผลลัพธ์ว่าจะอนุมัติคำขอหรือไม่แล้วจึงส่งข้อมูลกลับไปยังลูกค้าเช่นกัน



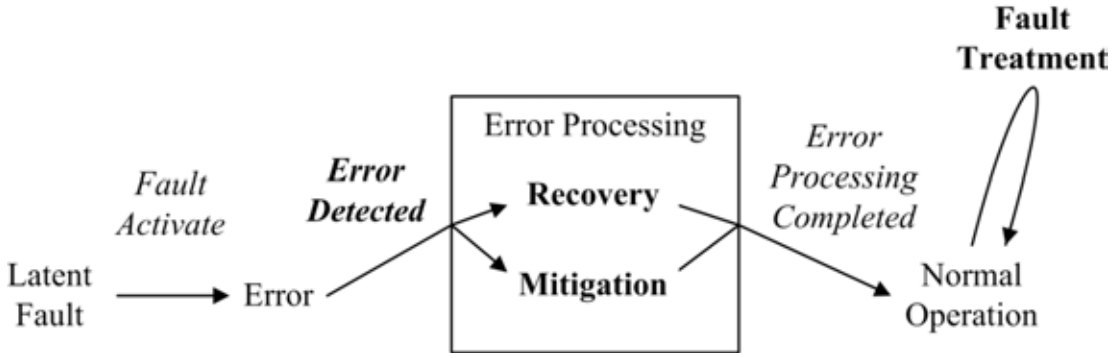
ภาพที่ 2.1 กระบวนการบีเพลการอนุมัติเงินกู้

จากตัวอย่างนี้หากระหว่างการเรียกเว็บเซอร์วิซเกิดความผิดปรกติขึ้น กระบวนการนี้จะไม่สามารถดำเนินต่อไปได้ ดังนั้นบีเพลจึงมีกลไกในการจัดการกับความผิดปรกติ (Exception) ต่างๆ ที่อาจเกิดขึ้นโดยใช้ <faultHandlers>, <compensationHandler> หรือ <eventHandler> เป็นต้น

ในการดำเนินงานตามกระแสงานบีเพลจะต้องใช้เครื่องประมวลบีเพล (BPEL Engine) เพื่อทำการอ่านกระแสงานบีเพลเข้ามา แล้วจึงจะประมวลผลเป็นการดำเนินงานต่างๆ ตามกระแสงานนั้น กระแสงานบีเพลที่ดำเนินการอยู่จะติดต่อได้ผ่านทางวิสเดิลที่ได้ประกาศไว้

2.2 แบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่อง (Patterns for Fault Tolerant Software)

การทนต่อความผิดพร่อง (Fault Tolerance) คือ คุณสมบัติที่ยอมให้ระบบสามารถทำงานต่อไปได้แม้ว่าจะเกิดความผิดพร่องขึ้นในระบบ [10] การทนต่อความผิดพร่องประกอบไปด้วย 4 ขั้น คือ การตรวจหาความผิดพลาด (Error Detection) การกู้ระบบจากความผิดพลาด (Error Recovery) การบรรเทาความผิดพลาด (Error Mitigation) และ การรักษาความผิดพร่อง (Fault Treatment) ดังภาพที่ 2.2



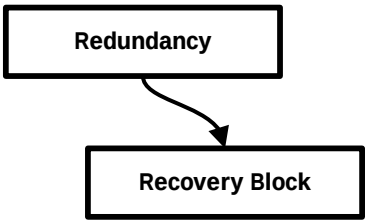
ภาพที่ 2.2 ขั้นตอนการทนต่อความผิดพร่อง [10]

ความผิดพร่องที่ซ่อนอยู่ในระบบ (Latent Fault) เช่น การเขียนโค้ดที่ผิด จะถูกกระตุ้นให้กลายเป็นความผิดพลาด (Error) เมื่อระบบมีการทำงานในโค้ดส่วนที่ผิดนั้น ในการทำให้ระบบทนต่อความผิดพร่องได้จะต้องเริ่มจากการตรวจจับความผิดพลาดที่เกิดขึ้นมา จากนั้นจึงเข้าสู่กระบวนการจัดการกับความผิดพลาด ซึ่งแบ่งได้เป็น 2 ทางคือการกู้ระบบจากความผิดพลาดซึ่งเป็นการเปลี่ยนสถานะของระบบให้กลับไปอยู่ในสถานะที่ไม่มีความผิดพลาด เช่นการเปลี่ยนไปทำงานที่ระบบสำรอง การจัดการอีกทางหนึ่งคือการบรรเทาความผิดพลาดซึ่งเป็นการจัดการโดยไม่จำเป็นต้องเปลี่ยนสถานะของระบบ เช่นสามารถแก้ไขข้อมูลที่ผิดพลาดได้ หากระบบสามารถจัดการกับความผิดพลาดได้แล้ว ขั้นต่อไปคือการรักษาความผิดพร่องซึ่งอาจยังคงอยู่ในระบบโดยเป็นการกำจัดความผิดพร่องให้ออกไปจากระบบ เช่น การปรับซอฟต์แวร์ให้เป็นปัจจุบัน สามารถทำหลังจากระบบกลับคืนสู่การทำงานปกติแล้ว

จากขั้นตอนการทำให้ทนต่อความผิดพร่อง Hanmer ได้รวบรวมแบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่องจำนวนทั้งหมด 63 แบบรูป แบ่งออกเป็น 5 กลุ่ม ประกอบไปด้วยแบบรูปตามขั้นตอนที่กล่าวมาได้แก่ แบบรูปการตรวจหาความผิดพลาด (Detection Patterns) แบบรูปการกู้ระบบจากความผิดพลาด (Error Recovery Patterns) แบบรูปการบรรเทาความผิดพลาด (Error Mitigation Patterns) และ แบบรูปการรักษาความผิดพร่อง (Fault Treatment Patterns) รวมทั้ง

ได้เพิ่มแบบรูปที่เกี่ยวข้องกับการออกแบบสถาปัตยกรรมของระบบออกมาเป็นอีกกลุ่มหนึ่งคือ แบบรูปเชิงสถาปัตยกรรม (Architectural Patterns) เพื่อเป็นการมองภาพรวมของทั้งระบบ และเป็นแบบรูปเบื้องต้นให้กับขั้นตอนที่เหลือของการทำให้ทนต่อความผิดพร่องต่อไป

แบบรูปเหล่านี้สามารถนำมาเขียนเป็นแผนที่ของภาษาแบบรูป (Pattern Language Map) เพื่อแสดงความสัมพันธ์ระหว่างแบบรูปได้ โดยแสดงไว้ในบทที่ 3 แผนที่ของภาษาแบบรูปมีลักษณะเป็นกราฟไม่มีวงแบบระบุทิศทาง (Directed Acyclic Graph) เช่นในภาพที่ 2.3 ลูกศรที่ชี้จากแบบรูป *Redundancy* ไปยังแบบรูป *Recovery Block* แสดงว่าแบบรูป *Recovery Block* สามารถแก้ปัญหาบางอย่างที่แบบรูป *Redundancy* นำเข้ามาได้



ภาพที่ 2.3 ตัวอย่างความสัมพันธ์ของแบบรูป

รายละเอียดของแต่ละกลุ่มมีดังต่อไปนี้

- แบบรูปเชิงสถาปัตยกรรม (Architectural Patterns) เป็นกลุ่มของแบบรูปที่พิจารณาถึงภาพรวมสำหรับการออกแบบระบบให้รองรับการทนต่อความผิดพร่อง และเป็นแบบรูปเบื้องต้นให้กับกลุ่มแบบรูปอื่นต่อไป แบบรูปในกลุ่มนี้มีจำนวน 11 แบบรูป
- แบบรูปการตรวจหา (Detection Patterns) เป็นกลุ่มของแบบรูปที่เกี่ยวกับการตรวจหาความผิดพร่องซึ่งเป็นขั้นแรกของการทำให้ทนต่อความผิดพร่อง แนวทางในการตรวจหาความผิดพร่องอาจใช้ความรู้ก่อนหน้ามาเป็นเกณฑ์ในการตรวจสอบการทำงาน หรืออีกแนวทางหนึ่งคือการนำผลลัพธ์จากการทำงานของตัวสำรองมาเปรียบเทียบกัน เพื่อหาว่ามีส่วนที่ทำงานผิดหรือไม่ แบบรูปในกลุ่มนี้มีจำนวน 16 แบบรูป
- แบบรูปการกู้ระบบจากความผิดพลาด (Error Recovery Patterns) เป็นกลุ่มของแบบรูปที่เกี่ยวกับการกู้ระบบจากความผิดพลาดซึ่งเป็นแนวทางหนึ่งในการจัดการกับความผิดพลาดที่เกิดขึ้น มีจุดประสงค์ให้ระบบยังคงดำเนินงานต่อไปได้แม้จะพบความผิดพลาดแล้ว โดยจะเปลี่ยนสถานะของระบบให้ไปอยู่ในสถานะที่ไม่มีความผิดพลาด แบบรูปในกลุ่มนี้มีจำนวน 14 แบบรูป

- แบบรูปการบรรเทาความผิดพลาด (Error Mitigation Patterns) เป็นกลุ่มของแบบรูปที่เกี่ยวกับการบรรเทาความผิดพลาดซึ่งเป็นอีกแนวทางหนึ่งในการจัดการกับความผิดพลาดที่ส่วนใหญ่เกี่ยวข้องกับเวลาหรือภาวะโหลดเกิน (Overload) แบบรูปกลุ่มนี้แตกต่างจากการกู้คืนระบบจากความผิดพลาดคือจะพยายามลดผลกระทบที่เกิดจากความผิดพลาดโดยไม่เปลี่ยนสถานะของระบบ อาจใช้การปิดบังความผิดพลาดหรือชดเชยผลกระทบที่เกิดจากความผิดพลาดนั้น แบบรูปในกลุ่มนี้มีจำนวน 16 แบบรูป
- แบบรูปการรักษาความผิดพร่อง (Fault Treatment Patterns) เป็นกลุ่มของแบบรูปที่เกี่ยวกับการรักษาความผิดพร่อง ซึ่งเป็นการป้องกันไม่ให้ความผิดพลาดเกิดขึ้นมาอีก โดยทำการแก้ไขความผิดพร่องที่เป็นสาเหตุทำให้เกิดความผิดพลาด เช่น ทำการติดตั้งซอฟต์แวร์หรือข้อมูลเวอร์ชันใหม่ที่ปราศจากความผิดพร่อง (Small Patch) หรือทำการปรับปรุงกระบวนการของระบบใหม่เพื่อไม่ให้เกิดความผิดพลาดอีก (Revise Procedure) แบบรูปในกลุ่มนี้มีจำนวน 6 แบบรูป

2.3 งานวิจัยที่เกี่ยวข้อง

สามารถแบ่งกลุ่มของงานวิจัยที่เกี่ยวข้องได้ออกเป็น 3 กลุ่ม ได้แก่

- การแบ่งประเภทความผิดพร่องของเว็บเซอร์วิซประกอบ
- การทำให้ทนต่อความผิดพร่องของดับเบิลยูเอส-บีเพล
- แบบรูปสำหรับการจัดการกับความผิดพร่อง

2.3.1 งานวิจัยที่เกี่ยวกับการแบ่งประเภทความผิดพร่องของเว็บเซอร์วิซประกอบ

กระแสงานบีเพลส่วนใหญ่จะมีการติดต่อกับเว็บเซอร์วิซต่างๆ ดังนั้นความผิดพร่องที่เกิดกับกระแสงานบีเพลสามารถพิจารณาได้จากความผิดพร่องของเว็บเซอร์วิซประกอบด้วยเช่นกัน งานวิจัยของ Chan และคณะ [14] ได้แบ่งความผิดพร่องที่เกิดขึ้นได้ออกได้เป็น 3 กลุ่ม ต่อไปนี้

- ความผิดพร่องทางกายภาพ (Physical faults) เกี่ยวข้องกับความผิดพลาดที่เกิดขึ้นกับโครงสร้างพื้นฐานของฝั่งผู้ให้บริการ เช่น ระบบฐานข้อมูลหรือเครื่องเซิร์ฟเวอร์ทำงานผิดพลาด นอกจากนี้ยังเกี่ยวข้องกับความผิดพลาดที่เกิดขึ้นในระดับเครือข่าย เช่น เครือข่ายระหว่างเครื่องประมวลกับผู้ให้บริการเซอร์วิซมีปัญหาทำให้เกิดภาวะที่เซอร์วิซไม่สามารถให้บริการได้ (Service unavailability)
- ความผิดพร่องในช่วงการพัฒนา (Development faults) อาจเกิดขึ้นจากความผิดพลาดของผู้พัฒนา เช่น ไม่มีการกำหนดค่าเริ่มต้นให้กับตัวแปร หรือ กิจกรรม

บางอย่างขาดหายไปจากกระแสงาน ความผิดพร่องที่กล่าวมานี้สามารถตรวจพบได้จากการให้เครื่องมือพัฒนาช่วยตรวจสอบในระหว่างการออกแบบ ความผิดพร่องอีกชนิดที่อาจเกิดขึ้นได้เรียกว่า ความไม่เข้ากันของพารามิเตอร์ (Parameter incompatibility) เกิดขึ้นจากผู้ให้บริการเว็บเซอร์วิซได้เปลี่ยนแปลงรายละเอียดของการเรียกใช้เว็บเซอร์วิซ และผู้ให้บริการไม่ได้แจ้งให้ทราบล่วงหน้า จะส่งผลให้เกิดความผิดพลาดจากการเรียกใช้เซอร์วิซได้

- ความผิดพร่องจากการโต้ตอบกัน (Interaction faults) สามารถแบ่งได้เป็น 2 ประเภท ได้แก่ ความผิดพร่องทางเนื้อหา (Content faults) และ ความผิดพร่องทางเวลา (Timing faults) ความผิดพร่องทางเนื้อหาเกิดขึ้นเมื่อเนื้อหาของผลลัพธ์ที่ได้รับมาจากเว็บเซอร์วิซแตกต่างไปจากผลลัพธ์ที่คาดว่าจะได้ซึ่งอิงจากรายละเอียดของเว็บเซอร์วิซนั้น ความผิดพร่องทางเวลาเกี่ยวข้องกับความผิดพร่องที่เกิดกับลำดับของผลลัพธ์ หรือผลลัพธ์ที่ไม่ทันเวลา

งานวิจัยของ Liu และคณะ [15] ได้แบ่งประเภทของความผิดพร่องที่เกิดกับเซอร์วิซคู่ค้า (Partner service faults) ออกเป็น 4 ประเภท ต่อไปนี้

- ความผิดพร่องเชิงตรรกะ (Logical faults) เป็นความผิดพร่องที่ส่งมาจากเว็บเซอร์วิซคู่ค้าเนื่องจากไม่สามารถทำตามคำสั่งที่เรียกไปให้สำเร็จได้ โดยอาจเกิดจากหลายสาเหตุ เช่น เว็บเซอร์วิซจองตั๋วเครื่องบินไม่สามารถหาที่นั่งได้ครบตามจำนวนที่สั่งไป หรือเว็บเซอร์วิซตรวจสอบสภาพอากาศไม่สามารถระบุสภาพอากาศของเมืองที่ส่งไปได้
- ความผิดพร่องทางระบบ (System faults) เป็นความผิดพร่องที่ถูกส่งขึ้นมาจากระบบแวดล้อมที่รองรับการทำงานขณะนั้น ตัวอย่างเช่น เว็บเซอร์วิซที่เรียกใช้นั้นถูกถอดออกจากระบบ (undeploy) เมื่อทำการเรียกไปจะได้รับการแจ้งกลับมาว่าไม่มีเซอร์วิซที่ต้องการอยู่ในระบบ
- ความผิดพร่องทางเนื้อหา (Content faults) เกี่ยวข้องกับผลลัพธ์ของเว็บเซอร์วิซที่ส่งมานั้นผิดพลาด ตัวอย่างเช่นเว็บเซอร์วิซส่งข้อมูลมาเกินขอบเขตที่เป็นไปได้
- ความผิดพร่องกับเอสแอลเอ (SLA faults) เกิดขึ้นเมื่อเว็บเซอร์วิซสามารถทำงานได้ตามความต้องการเชิงหน้าที่แต่ได้ละเมิดข้อตกลง (Service Level Agreement: SLA) ที่กำหนดไว้ เช่น ข้อตกลงกำหนดไว้ว่าระยะเวลาของการทำงานจะน้อยกว่า 5 วินาที แต่เวลาที่ใช้จริงคือ 8 วินาที

ประเภทของความผิดพร่องในงานวิจัยที่กล่าวมานั้นมีลักษณะที่คาบเกี่ยวกันอยู่ ซึ่งแตกต่างกันที่มุมมองในการแบ่งประเภท ในงานวิจัยนี้มุ่งเน้นไปที่ความผิดพร่องที่เกิดกับเซอร์วิซคู่ค้าที่มีลักษณะเป็นความผิดพร่องทางกายภาพและความผิดพร่องจากการโต้ตอบกัน เพราะความผิดพร่องที่อยู่ในช่วงการพัฒนาสามารถตรวจสอบและแก้ไขได้ก่อนการใช้งานจริง

2.3.2 งานวิจัยที่เกี่ยวกับการทำให้ทนต่อความผิดพร่องของดับเบิลยูเอส-บีเพล

งานวิจัยที่เกี่ยวข้องกับการทำให้ทนต่อความผิดพร่องของบีเพลหรือเว็บเซอร์วิซประกอบสามารถแบ่งได้เป็น 3 แนวทาง ได้แก่ (1) ใช้ความสามารถในการทนต่อความผิดพร่องของบีเพล (2) เพิ่มเติมส่วนต่อขยายของเครื่องประมวลบีเพลและสร้างระบบพื้นฐานขึ้นมารองรับ และ (3) ใช้พร็อกซีเซอร์วิซ

2.3.2.1 ใช้ความสามารถในการทนต่อความผิดพร่องของบีเพล

Dobson [2] เสนอการใช้ภาษาบีเพลเพื่อทำให้เกิดความสามารถในการทนต่อความผิดพร่องให้กับการเรียกเว็บเซอร์วิซหนึ่งๆ โดยใช้คุณสมบัติในการจัดการกับความผิดพร่องของบีเพลที่มีอยู่แล้ว วิธีการทำให้ทนต่อความผิดพร่องที่รองรับมี 2 กลุ่มคือ (1) การเรียกซ้ำ หมายถึงการเรียกเซอร์วิซเดิมหรือการเรียกเซอร์วิซสำรองที่กำหนดไว้ และ (2) การดำเนินงานแบบขนาน หมายถึงการหาผลโหวตจากกลุ่มของเซอร์วิซที่กำหนดไว้ งานวิจัยที่มีแนวทางใกล้เคียงกันคือ งานวิจัยของ Modafferi และ Conforti [3] ได้เสนอการทำให้บีเพลมีความสามารถในการกู้ระบบ (Recovery Action) จากความผิดพร่อง แต่ผู้ออกแบบกระแสงานต้องกำหนดข้อมูลที่เกี่ยวข้องกับการกู้ระบบลงไปในไฟล์บีเพลก่อน เรียกว่า “Annoted WS-BPEL” แล้วจึงนำไปประมวลผลให้เป็นบีเพลที่มีโครงสร้างตามบีเพลเวอร์ชัน 1.1 นอกจากนี้ยังรองรับวิธีการกู้ระบบที่ซับซ้อนกว่างานวิจัยของ Dobson เช่น การกำหนดตัวแปรจากภายนอก การทำโรลแบค (Rollback) เป็นต้น

งานวิจัยของ Liu และคณะ [4, 15] ซึ่งใช้แนวทางนี้เช่นกัน ได้เสนอการเพิ่มความสามารถในการทนต่อความผิดพร่องให้กระแสงานบีเพลโดยการกำหนดกฎอีซีเอ (Event-Condition-Action Rules) ให้กับแต่ละเซอร์วิซ เพื่อเป็นแนวทางหากเกิดความผิดพร่องขึ้นจะต้องทำงานอย่างไรภายใต้เงื่อนไขที่กำหนดไว้ กฎอีซีเอจะถูกกำหนดไว้ในอีกไฟล์หนึ่งแยกจากบีเพล แล้วนำมาประมวลผลร่วมกันให้ได้ไฟล์บีเพลใหม่ซึ่งเพิ่มเติมโครงสร้างที่ช่วยทำให้ทนต่อความผิดพร่อง วิธีการที่รองรับได้แก่ การเพิกเฉยต่อความผิดพร่องที่เกิดขึ้น (Ignore) การข้ามการเรียกใช้เซอร์วิซ (Skip) การเรียกซ้ำเซอร์วิซเดิม (Retry) และ การเรียกใช้เซอร์วิซสำรอง (Alternate) กฎอีซีเอสามารถกำหนดเงื่อนไขให้ใช้วิธีการที่ต่างกันได้ถึงแม้จะเป็นความผิดพร่องแบบเดียวกัน

จะเห็นได้ว่าข้อดีของแนวทางนี้คือบีเพลที่ได้หลังจากผ่านกระบวนการแปลงแล้วสามารถนำไปใช้กับเครื่องประมวลบีเพลทั่วไปได้ เพราะยังคงใช้โครงสร้างตามมาตรฐานของบีเพล แต่ข้อจำกัดของแนวทางนี้คือจำเป็นจะต้องเพิ่มโครงสร้างที่ทำให้ทนต่อความผิดพร่องลงไปในไฟล์ของบีเพล ทำให้การเปลี่ยนแปลงเว็บเซอร์วิซตัวแทนหรือวิธีการทำให้ทนต่อความผิดพร่องจะไม่สามารถทำได้ขณะที่กำลังดำเนินงานตามกระแสงานอยู่

2.3.2.2 เพิ่มเติมส่วนต่อขยายของเครื่องประมวลบีเพลและสร้างระบบพื้นฐานขึ้นมารองรับ

Moser และคณะ [6] กล่าวว่าบีเพลยังมีข้อด้อยอยู่ 2 ประการคือ ไม่สามารถเปลี่ยนแปลงอย่างพลวัตขณะดำเนินงานอยู่ได้ และ บีเพลไม่มีส่วนที่ทำหน้าที่ตรวจสอบการทำงานของระบบ จึงได้นำเสนอระบบ VieDAME (Vienna Dynamic Adaptation and Monitoring Environment for WS-BPEL) ที่เพิ่มเติมการทำงานต่อมาจากเครื่องประมวลบีเพลด้วยการโปรแกรมเชิงแง่มุม (Aspect-Oriented Programming) เพื่อรองรับการเฝ้าสังเกต (Monitoring) กระแสงานบีเพลตามคุณภาพของเซอร์วิซ (Quality of Service) ระบบสามารถปรับเปลี่ยนเซอร์วิซขณะดำเนินงานอยู่ได้ โดยเลือกเซอร์วิซตามเกณฑ์ที่กำหนดไว้ นอกจากนี้สามารถปรับเปลี่ยนข้อมูลที่รับส่งกันในกรณีที่ส่วนต่อประสานของเซอร์วิซตัวแทนนั้นไม่ตรงกับของเดิม

Subramanian และคณะ [16] ได้เสนอแนวทางในการเพิ่มความสามารถให้กับเครื่องประมวลบีเพลเพื่อให้ได้เว็บเซอร์วิซประกอบที่สามารถรักษาตัวเองได้ (Self-Healing) โดยทำการแก้ไขเครื่องประมวลบีเพล เพื่อเพิ่มส่วนที่เรียกว่า “SelfHeal-Engine” เข้าไป ประกอบไปด้วย 4 ส่วนคือ การวางแผน (Planning) การเฝ้าสังเกต (Monitoring) การวินิจฉัย (Diagnosing) และ การกู้ระบบ (Recovering) โดยหากตรวจสอบพบความผิดพร่องจะทำการวินิจฉัยและค้นหาวิธีการแก้ไขจากฐานข้อมูล ซึ่งอาจมีได้หลายวิธีขึ้นกับชนิดของความผิดพร่อง วิธีการแก้ไขที่เสนอได้แก่ การเรียกซ้ำ การแก้ไขข้อมูล การเปลี่ยนตัวแทน และ การแก้ไขกระบวนการใหม่

จะเห็นได้ว่าข้อดีของแนวทางนี้คือรองรับวิธีการทนต่อความผิดพร่องที่มีความซับซ้อนมากขึ้นได้ รวมทั้งมีความยืดหยุ่นในการปรับเปลี่ยนวิธีการทนต่อความผิดพร่องหรือการเลือกเซอร์วิซที่ใช้เป็นตัวแทนสามารถทำได้ขณะดำเนินงานอยู่ แต่ข้อจำกัดของแนวทางนี้คือไม่สามารถใช้กับเครื่องประมวลบีเพลทั่วไปเพื่อให้ได้ความสามารถในการทนต่อความผิดพร่องดังกล่าว

2.3.2.3 ใช้พร็อกซีเซอร์วิซ

แนวทางนี้จะยังคงใช้โครงสร้างตามมาตรฐานของบีเพล และสามารถใช้กับเครื่องประมวลบีเพลทั่วไปได้ แต่จะมีการแทรกแซงการเรียกเซอร์วิซในกระแสงาน โดยจะเปลี่ยนไปเรียกผ่านเว็บเซอร์วิซที่เป็นพร็อกซีแทน โดยพร็อกซีส่วนใหญ่นั้นจะมีความสามารถในการเรียกเซอร์วิซที่เกี่ยวข้องในกระแสงานแทนการเรียกจากเครื่องประมวลบีเพลโดยตรง หรือในกรณีที่ไม่สามารถเรียกเซอร์วิซนั้นได้จะหาเซอร์วิซอื่นที่ทำงานได้เหมือนกันมาใช้แทน โดยในงานวิจัยต่างๆ นั้นได้นำเสนอพร็อกซีที่มีความสามารถแตกต่างกัน เช่น งานวิจัยของ Ezenwoye และ Sadjadi [7] ได้เสนอพร็อกซีทั่วไป (Generic Proxy) ที่รองรับการทำงานกับหลายบีเพลและหลายเซอร์วิซพร้อมกันได้ งานวิจัยของ Laranjeiro และ Vieira [8] เสนอพร็อกซีที่สามารถหาผลโหวตจากกลุ่มของเซอร์วิซได้ และสามารถประเมินคุณสมบัติของแต่ละเซอร์วิซเพื่อเป็นตัวช่วยในการหาผลลัพธ์ในกรณีที่ได้ผลโหวตออกมาเท่ากัน งานวิจัยของ Christos และคณะ [9] เสนอพร็อกซีที่สามารถจัดอันดับเซอร์วิซตัวแทนจากคุณภาพของเซอร์วิซได้

ถึงแม้จะใช้งานได้กับเครื่องประมวลบีเพลทั่วไป แต่ข้อจำกัดของแนวทางนี้คือจำเป็นต้องมีระบบพื้นฐานขึ้นมานอกเหนือจากเครื่องประมวลบีเพลเพื่อรองรับการทำงานของพร็อกซี

2.3.3 งานวิจัยที่เกี่ยวกับแบบรูปสำหรับการจัดการกับความผิดพร่อง

ผู้วิจัยได้แนวคิดในการประเมินความสามารถของเครื่องประมวลบีเพลในการจัดการกับความผิดพร่องจากงานวิจัยของ Russell และคณะ [17, 18] ซึ่งได้เสนอเฟรมเวิร์กสำหรับการแบ่งกลุ่มการจัดการกับความผิดปรกติ (Exception Handling) ในกระแสงานโดยอิงตามแบบรูป โดยทำการพิจารณาถึงสถานะต่างๆ ของกระแสงานในการจัดการกับความผิดปรกติ และกำหนดเป็นแบบรูปที่เป็นไปได้แบ่งตามประเภทของความผิดปรกติ แล้วนำแบบรูปที่ได้นั้นมาประเมินความสามารถในการจัดการกับความผิดปรกติของระบบกระแสงานและภาษาสำหรับกระแสงานต่างๆ ที่มีการใช้งานอยู่ในขณะนั้นซึ่งรวมไปถึงบีเพลด้วย

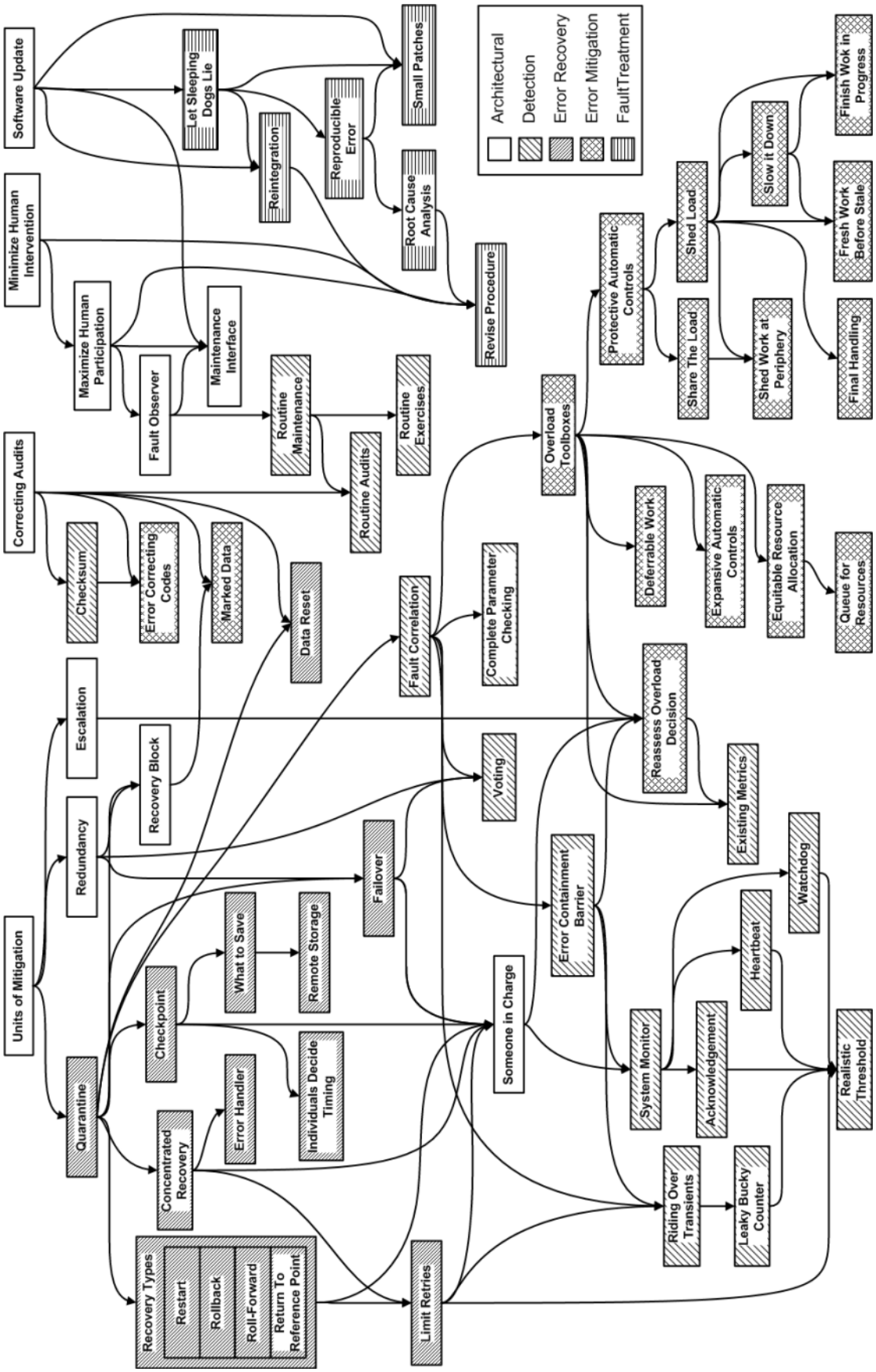
บทที่ 3

การวิเคราะห์แบบรูป

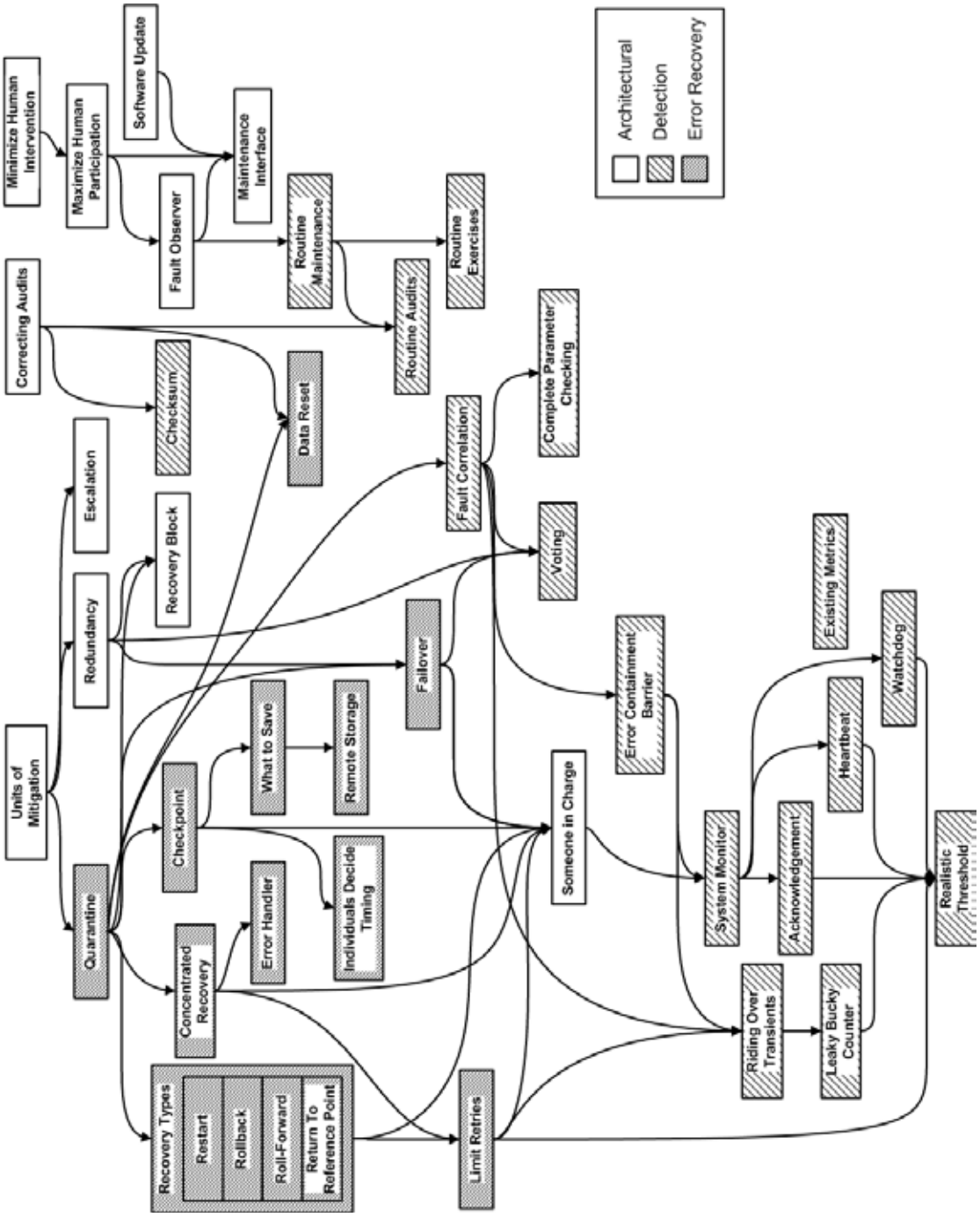
ในการวิเคราะห์แบบรูปเพื่อนำไปประยุกต์ใช้กับบีเพล จะทำการพิจารณาแบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่อง [10] ที่มีทั้งหมด 63 แบบรูป แบ่งออกเป็น 5 กลุ่ม ได้แก่ แบบรูปเชิงสถาปัตยกรรม แบบรูปการตรวจหา แบบรูปการกู้ระบบจากความผิดพลาด แบบรูปการบรรเทาความผิดพลาด และแบบรูปการรักษาความผิดพลาด แบบรูปทั้งหมดสามารถนำมาแสดงความสัมพันธ์ด้วยแผนที่ของภาษาแบบรูป (Pattern language map) แผนที่ของภาษาแบบรูปมีลักษณะเป็นกราฟไม่มีวงแบบระบุทิศทาง (Directed Acyclic Graph) โดยแผนที่แบบหนึ่งที่เป็นไปได้เป็นดังภาพที่ 3.1 ลูกศรที่ชี้จากแบบรูปหนึ่งไปสู่อีกแบบรูปหนึ่ง แสดงว่าแบบรูปที่ถูกชี้สามารถแก้ปัญหาบางอย่างที่แบบรูปก่อนหน้านำเข้ามาได้

งานวิจัยนี้จะพิจารณาแบบรูปใน 3 กลุ่มแรกเป็นหลัก เนื่องจากแบบรูปบรรเทาความผิดพลาดส่วนใหญ่มีไว้เพื่อรองรับความผิดพร่องที่เกี่ยวข้องกับสภาวะโหลดเกินของระบบ แต่งานวิจัยนี้จะพิจารณาถึงความผิดพร่องที่เกี่ยวข้องกับเว็บเซอร์วิซคู่ค้าเป็นหลัก ซึ่งสามารถใช้เพียงแบบรูปใน 3 กลุ่มแรกมาจัดการได้ อีกประการหนึ่งคือแบบรูปการรักษาความผิดพลาดเป็นแบบรูปที่เกี่ยวข้องกับการวิเคราะห์หาสาเหตุของความผิดพร่องต่างๆ ที่เกิดขึ้น แล้วทำการแก้ไขด้วยการปรับปรุงซอฟต์แวร์ใหม่เพื่อกำจัดความผิดพร่องที่ยังคงมีอยู่ในระบบให้หมดไป แบบรูปในกลุ่มนี้จึงควรปฏิบัติหลังจากจัดการกับความผิดพร่องไปแล้วและไม่ได้อยู่ในช่วงดำเนินการของระบบ ดังนั้นแบบรูปการบรรเทาความผิดพลาดและแบบรูปการรักษาความผิดพลาดจึงอยู่นอกเหนือขอบเขตของงานวิจัยนี้ ทำให้แบบรูปที่จะนำมาพิจารณามีจำนวน 41 แบบรูป หากนำมาจัดเป็นแผนที่ของภาษาแบบรูปใหม่ที่เหลือเฉพาะ 3 กลุ่มแรกจะได้ดังภาพที่ 3.2

ในการวิเคราะห์แบบรูปทั้ง 3 กลุ่มนี้ จะพิจารณาถึงจุดประสงค์ของแบบรูป และความเหมาะสมในการนำไปประยุกต์ใช้กับบีเพล เพื่อแยกเป็นระดับของการประยุกต์ใช้ ได้แก่ ระดับกระแสงาน ระดับเครื่องประมวลบีเพลและระบบพื้นฐาน หรือระดับขั้นตอนออกแบบ ซึ่งในงานวิจัยนี้มุ่งเน้นไปที่การประยุกต์ใช้แบบรูปกับบีเพลในระดับกระแสงาน เพื่อช่วยให้การออกแบบกระแสงานบีเพลรองรับการทนต่อความผิดพร่องได้ โดยเสนอเป็นโครงสร้างแม่แบบของบีเพลที่ประยุกต์ตามแบบรูปต่างๆ ไว้ดังจะแสดง ในบทที่ 4 ต่อไป



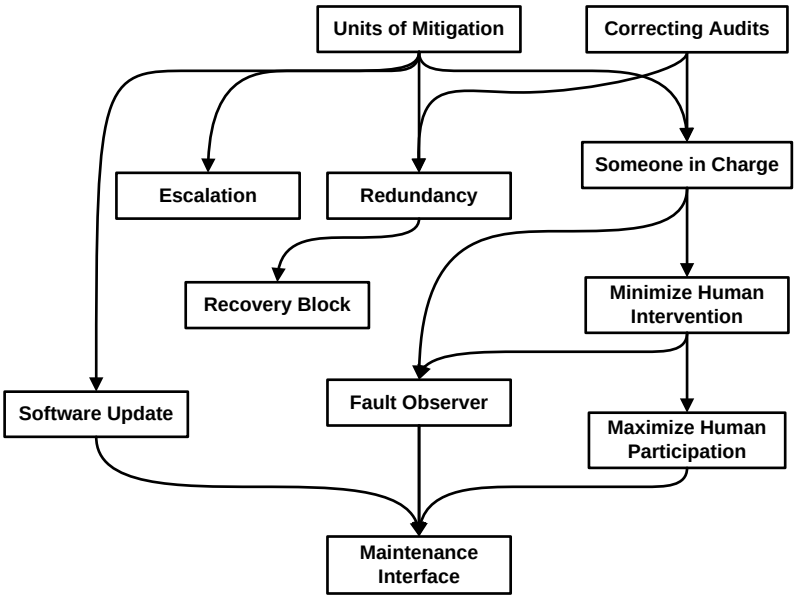
ภาพที่ 3.1 แผนที่ภาษาแบบรูปรวมทั้งหมด [10]



ภาพที่ 3.2 แผนที่ภาษาแบบรูปเฉพาะ 3 กลุ่มแรก

3.1 แบบรูปเชิงสถาปัตยกรรม (Architectural Patterns)

แบบรูปเชิงสถาปัตยกรรมเป็นการมองภาพรวมของระบบว่าควรจะเริ่มต้นออกแบบการทำให้ทนต่อความผิดพร่องอย่างไรบ้าง จะเห็นได้จากแผนที่ของภาษาแบบรูปในภาพที่ 3.2 ว่าแบบรูปในกลุ่มนี้จะอยู่ด้านบนของแผนที่ซึ่งจะเป็นพื้นฐานให้กับแบบรูปในกลุ่มอื่นๆ ต่อไป หากพิจารณาเฉพาะความสัมพันธ์ภายในกลุ่มแบบรูปเชิงสถาปัตยกรรมจะแสดงด้วยแผนที่ภาษาแบบรูปที่เป็นไปได้แบบหนึ่งดังภาพที่ 3.3



ภาพที่ 3.3 แผนที่ภาษาแบบรูปเชิงสถาปัตยกรรม [10]

การนำแบบรูปในกลุ่มนี้มาปรับใช้กับการออกแบบบีเพลให้ทนต่อความผิดพร่องสามารถพิจารณาทีละแบบรูป โดยจะกล่าวถึงความหมายและลักษณะการใช้งานของแต่ละแบบรูปก่อน [10] และกล่าวถึงการประยุกต์ใช้แบบรูปกับบีเพลในส่วนท้ายของแต่ละหัวข้อดังต่อไปนี้

3.1.1 Units of Mitigation

แบบรูปนี้มีจุดประสงค์ให้ระบบใหญ่สามารถดำเนินงานต่อไปได้แม้ว่าจะเกิดความผิดพร่อง จึงควรออกแบบระบบให้สามารถแบ่งเป็นหน่วยย่อยแทนที่จะให้ทั้งระบบเป็นหน่วยเดียว เมื่อเกิดความผิดพร่องกับหน่วยย่อยหนึ่ง หน่วยย่อยอื่นที่ไม่เกี่ยวข้องสามารถทำงานต่อไปได้โดยไม่ต้องรอให้การจัดการกับความผิดพร่องนั้นเรียบร้อยก่อน หน่วยย่อยที่แบ่งได้ควรมีคุณสมบัติดังนี้

- ควรประกอบไปด้วยการทำงานแบบอะตอมมิก (Atomic action)
- ควรรองรับการทำซ้ำเพื่อใช้เป็นตัวสำรองได้
- แต่ละหน่วยไม่จำเป็นต้องเหมือนหรือมีขนาดเท่ากัน

- ควรมีความสามารถในการตรวจสอบการทำงานว่ามีความผิดพลาดหรือไม่
- หากเกิดความผิดพลาดขึ้นแล้วสามารถกันไม่ให้กระทบกับการทำงานส่วนอื่นได้

เมื่อนำแบบรูปนี้มาประยุกต์ใช้กับบีเพล สามารถใช้โครงสร้างของแท็ก <scope> ในการแบ่งกระแสงานบีเพลให้เป็นส่วนย่อยๆ ได้ เนื่องจากรองรับการแบ่งหน่วยย่อยให้เป็นไปตามคุณสมบัติของแบบรูปที่กล่าวมา โดยภายในแท็ก <scope> จะต้องประกอบไปด้วยกิจกรรมบีเพลอย่างน้อยหนึ่งกิจกรรมจึงรองรับการทำงานแบบอะตอมมิก และแต่ละแท็ก <scope> ถือเป็นหน่วยย่อยที่รองรับการทำซ้ำได้ รวมทั้งยังรองรับการตรวจสอบความผิดพร่องด้วยการใช้แท็ก <faultHandlers> ซึ่งเป็นอิสระกับแท็ก <scope> อื่นๆ

3.1.2 Correcting Audits

ความผิดพร่องที่เกิดขึ้นกับข้อมูลหนึ่งอาจส่งผลให้การทำงานเกิดข้อผิดพลาดได้ หรืออาจส่งผลให้ข้อมูลอื่นเกิดความผิดพลาดตามไปด้วย ระบบจึงควรมีการตรวจหาและแก้ไขข้อมูลที่เกิดความผิดพร่องอย่างเร็วที่สุด และควรตรวจสอบข้อมูลที่เกี่ยวข้องกันเพิ่มเติมด้วย นอกจากนี้อาจทำการบันทึกความผิดพลาดที่เกิดขึ้นไว้ จากนั้นจึงกลับไปดำเนินงานตามปกติ

วิธีการตรวจสอบข้อมูลสามารถนำมาประยุกต์ใช้กับบีเพลได้ทั้งในระดับกระแสงานและในระดับเครื่องประมวลบีเพล การตรวจสอบข้อมูลในระดับกระแสงานควรดูข้อมูลที่รับส่งกันว่ามีโครงสร้างตรงตามที่ประกาศไว้ในเอกซ์เอ็มแอลสกีมาหรือไม่ หรือใช้คำสั่ง <validate> มาช่วยตรวจสอบ การตรวจสอบข้อมูลในระดับเครื่องประมวลบีเพลจำเป็นต้องมีหน่วยเฉพาะหรือส่วนเสริมมาช่วยทำหน้าที่ตรวจสอบ และสามารถใช้แบบรูป *Checksum* มาเป็นวิธีหนึ่งที่ช่วยในการตรวจสอบความถูกต้องของข้อมูลได้อีกทางหนึ่ง

3.1.3 Redundancy

เนื่องจากการจัดการกับความผิดพลาดอาจทำให้ระบบต้องเสียเวลาไปมากและทำให้อยู่ในสภาพไม่พร้อมใช้งาน แบบรูปนี้จึงมีจุดประสงค์เพื่อเพิ่มสภาพพร้อมใช้งานให้กับระบบด้วยการใช้ตัวสำรองที่สามารถปฏิบัติงานแทนในช่วงที่ระบบกำลังจัดการกับความผิดพลาดอยู่

การนำตัวสำรองมาประยุกต์ใช้มี 3 ประเภท ได้แก่

- ตัวสำรองเชิงพื้นที่ (Spatial redundancy) เป็นการใช้ตัวสำรองที่สำเนาไว้อยู่ในสถานที่ต่างกัน ช่วยทำให้ลดช่วงเวลาที่ระบบไม่พร้อมใช้งาน สามารถนำการโปรแกรมแบบหลายเวอร์ชัน (N-Version Programming) มาใช้ร่วมกับแบบรูป *Voting* ได้

- ตัวสำรองเชิงเวลา (Temporal redundancy) เป็นการใช้ตัวสำรองที่เกิดขึ้นในช่วงเวลาหนึ่ง สามารถใช้แบบรูป *Recovery Blocks* ในการทำตัวสำรองทางซอฟต์แวร์เชิงเวลาได้
- ตัวสำรองเชิงสารสนเทศ (Informational redundancy) เป็นการเก็บข้อมูลเดียวกันไว้หลายเวอร์ชันเพื่อเป็นประโยชน์ในการตรวจสอบความถูกต้อง เช่นการใช้แบบรูป *Correcting Audits*

บีเพลมิได้มีการรองรับการใช้ตัวสำรองโดยตรง แต่สามารถออกแบบกระแสงานให้รองรับการใช้ตัวสำรองตามแบบรูป *Recovery Blocks* ได้ หรือใช้ส่วนเสริมที่ทำงานร่วมกับเครื่องประมวลบีเพลในการสับเปลี่ยนตัวสำรอง

3.1.4 Recovery Blocks

เมื่อระบบรองรับการใช้งานตัวสำรอง (Redundancy) อยู่แล้ว วิธีการหนึ่งที่นำมาประยุกต์ใช้ได้คือแบบรูป *Recovery Blocks* แบบรูปนี้จะประกอบไปด้วยส่วนปฐมภูมิ (Primary block) และส่วนทุติยภูมิ (Secondary blocks) โดยจะทำงานเป็นลำดับต่อเนื่องกัน เริ่มจากส่วนปฐมภูมิก่อน ถ้าการทำงานในส่วนนี้ไม่ผ่านการทดสอบเพื่อยอมรับ (Acceptance Test) ซึ่งในที่นี้หมายถึงมีความผิดพร่องเกิดขึ้น ระบบจะต้องเปลี่ยนไปทำงานในส่วนทุติยภูมิแทน และหากยังไม่ผ่านอีกจะเปลี่ยนไปทำงานในส่วนทุติยภูมิลำดับต่อไปเรื่อยๆ จนกว่าจะผ่านได้ แต่ถ้าการทำงานในส่วนทุติยภูมิทั้งหมดนั้นไม่สามารถจัดการได้ จำเป็นต้องส่งต่อให้ส่วนอื่นมาจัดการกับความผิดพลาดนี้ต่อไปตามแบบรูป *Escalation*

การนำแบบรูปนี้มาประยุกต์ใช้กับบีเพลเริ่มจาก ส่วนปฐมภูมิคือลำดับของงานที่เกี่ยวข้องกับการเรียกใช้เซอร์วิซและคำสั่งอื่นที่ต้องใช้ร่วมกัน ต่อมาให้ส่วนทุติยภูมิคือลำดับของวิธีการทำงานที่สามารถใช้แทนส่วนแรกได้ โดยอาจเป็นการเรียกเซอร์วิซตัวแทนหรือเป็นลำดับของกระแสงานย่อยที่แทนกันได้ หากการทำงานของส่วนปฐมภูมิมีความผิดพร่องเกิดขึ้นจะถูกดักจับด้วย <faultHandlers> แล้วทำการกำหนดค่าตัวแปรเพื่อบอกว่าส่วนปฐมภูมินั้นทำงานไม่ผ่าน และส่งไปทำงานต่อในส่วนทุติยภูมิต่อไป ถ้าสุดท้ายแล้วยังไม่สามารถทำงานให้ผ่านได้ จะส่งสัญญาณออกไปว่ามีความผิดพร่องเกิดขึ้น

ข้อจำกัดของการใช้แบบรูปนี้คือหากมีการใช้งานส่วนทุติยภูมิหลายขั้นแล้วยังไม่สามารถทำงานให้ผ่านได้ จะทำให้ระบบใช้เวลาในส่วนนี้มากเกินไป จึงควรออกแบบให้ใช้ส่วนทุติยภูมิในจำนวนที่เหมาะสมตามแบบรูป *Limit Retries*

3.1.5 Minimize Human Intervention

มนุษย์ซึ่งเป็นผู้ปฏิบัติงานอาจเป็นส่วนหนึ่งที่ทำให้เกิดความผิดพลาดขึ้นในระบบ เช่น ปฏิบัติงานผิดขั้นตอน หรือละเว้นขั้นตอนที่จำเป็นไป ดังนั้นในการจัดการกับความผิดพลาดควรปรับลดขั้นตอนที่เกี่ยวข้องกับมนุษย์ให้น้อยลง และปรับปรุงกระบวนการให้เป็นอัตโนมัติมากยิ่งขึ้น เพื่อเพิ่มความรวดเร็วในการแก้ไขปัญหาและเพิ่มสภาพพร้อมใช้งานของระบบ อีกทั้งควรมีการบอกสถานะของการดำเนินงานให้ผู้ปฏิบัติงานทราบว่าอยู่ในขั้นตอนใดแล้ว

การนำแบบรูปนี้มาประยุกต์ใช้กับบีเพลหมายถึงการออกแบบกระแสงานบีเพลตามแบบรูปเพื่อให้สามารถจัดการกับความผิดพร่องได้โดยอัตโนมัติ และทางเครื่องประมวลบีเพลควรรองรับการทำงานต่างๆ ตามที่ได้ออกแบบไว้ และสามารถรายงานสถานะของแต่ละกระบวนการให้ผู้ปฏิบัติงานได้ทราบด้วย

3.1.6 Maximize Human Participation

ถึงแม้ระบบจะมีการออกแบบตามแบบรูป *Minimize Human Intervention* แล้วก็ตาม แต่ยังมีโอกาสที่จะยังมีความผิดพลาดเกิดขึ้นได้อีก ผู้เชี่ยวชาญที่เป็นผู้ออกแบบระบบหรือมีทักษะในการบำรุงรักษาอาจช่วยแก้ไขปัญหาที่เกิดขึ้น ระบบจึงควรออกแบบให้มีช่องทางในการเฝ้าดูความผิดพร่อง (ตามแบบรูป *Fault Observer*) และมีช่องทางที่รองรับการบำรุงรักษา (ตามแบบรูป *Maintenance Interface*) จากผู้เชี่ยวชาญในการออกคำสั่งเพื่อจัดการกับความผิดพลาดที่เกิดขึ้น

การนำแบบรูปนี้มาปรับใช้กับบีเพล ควรออกแบบให้ตัวเครื่องประมวลบีเพลและระบบพื้นฐานมีความสามารถในการรองรับการปฏิบัติงานจากผู้เชี่ยวชาญในการจัดการกับความผิดพร่องได้

3.1.7 Maintenance Interface

คำสั่งหรือการทำงานที่เกี่ยวข้องกับการบำรุงรักษาหากใช้ส่วนต่อประสานเดียวกันกับการทำงานหลักของระบบ อาจส่งผลให้เกิดปัญหาต่อการส่งคำสั่งในภาวะโหลดเกิน หรือเกิดปัญหาด้านความมั่นคงจากการส่งคำสั่งโดยผู้ไม่หวังดี ดังนั้นควรออกแบบให้ระบบมีส่วนต่อประสานที่เกี่ยวกับงานด้านบำรุงรักษาแยกออกจากส่วนต่อประสานหลัก ตัวอย่างการใช้งานของแบบรูปนี้คือ หน้าเว็บสำหรับปรับแต่งระบบของเราท์เตอร์ซึ่งแยกส่วนออกมาจากการทำงานหลักผ่านทางพอร์ตอีเธอร์เน็ต

เครื่องประมวลบีเพลควรรองรับการบำรุงรักษาผ่านส่วนต่อประสานอื่นที่แยกออกมาจากการทำงานหลัก เช่นมีหน้าเว็บสำหรับควบคุมและตรวจสอบการทำงาน หรือมีส่วนต่อประสานที่สามารถสั่งงานได้นอกเหนือจากการทำงานผ่านบีเพลเอง

3.1.8 Someone in Charge

เมื่อมีการแบ่งระบบออกเป็นหน่วยย่อยๆ ตามแบบรูป *Units of Mitigation* ไม่ว่าจะทำตามแบบรูปใดก็ตามควรจะมีหน่วยใดหน่วยหนึ่งมีหน้าที่รับผิดชอบหากเกิดความผิดพร่องขึ้น หน่วยดังกล่าวควรมีความสามารถในการคอยตรวจสอบว่าระบบได้ปฏิบัติงานในส่วนหนึ่งเสร็จสิ้นแล้วหรือไม่ หากมีการทำงานที่ไม่เสร็จสิ้น หน่วยที่รับผิดชอบอยู่มีหน้าที่ดำเนินการในการจัดการกับความผิดพลาด แต่ถ้าการจัดการกับความผิดพลาดนั้นยังไม่สำเร็จต้องส่งต่อหน้าที่ไปยังหน่วยอื่นที่จะรับหน้าที่ต่อไปตามแบบรูป *Escalation*

การประยุกต์ใช้กับบีเพลควรทำในระดับเครื่องประมวลบีเพล โดยควรมีความสามารถในการติดตามการทำงานของแต่ละหน่วยย่อยในกระแสงานบีเพล และสามารถดำเนินการตามแบบรูปเพื่อจัดการกับความผิดพลาดตามที่ออกแบบไว้สำหรับหน่วยย่อยนั้นๆ ได้

3.1.9 Escalation

ในสถานการณ์ที่ระบบพยายามจัดการกับความผิดพร่อง แต่วิธีการที่ใช้อยู่นั้นยังไม่สำเร็จ และการพยายามทำซ้ำอาจจะไม่ได้ผลเช่นกัน เพื่อหลีกเลี่ยงไม่ให้ระบบติดอยู่กับวิธีการเดิมนานเกินไป ระบบควรปรับไปใช้วิธีการที่รุนแรงขึ้น ซึ่งในที่นี้หมายถึงวิธีการที่มีผลกระทบกับหน่วยย่อยอื่นๆ หรือส่งผลให้ต้องใช้ทรัพยากรในการทำงานมากยิ่งขึ้น ตัวอย่างเช่น ขั้นตอนหนึ่งพยายามจะแก้ไขข้อมูลให้ถูกต้องแต่ไม่สำเร็จ และลองใช้แบบรูป *Rollback* กลับไปยังสถานะที่เคยบันทึกไว้แล้วก็ยังไม่สำเร็จ ขั้นต่อไปอาจจะต้องเริ่มกระบวนการทำงานในส่วนนั้นใหม่ตามแบบรูป *Restart* สุดท้ายแล้วถ้ายังไม่สามารถจัดการได้อาจจะกลับไปใช้ข้อมูลเริ่มต้นใหม่ตามแบบรูป *Data Reset* หรือส่งงานที่ต้องทำไปยังหน่วยประมวลผลอื่นแทนตามแบบรูป *Failover* การใช้แบบรูปนี้ควรรายงานสถานะของการทำงานให้กับหน่วยที่รับผิดชอบ (ตามแบบรูป *Someone in Charge*) และมีช่องทางให้ผู้ปฏิบัติงานสามารถเลือกวิธีการในการจัดการได้เมื่อจำเป็น

การนำแบบรูปนี้มาปรับใช้กับการออกแบบบีเพลสามารถช่วยได้ทั้งในระดับกระแสงานและระดับเครื่องประมวลบีเพล ในระดับกระแสงานควรออกแบบให้มีระดับชั้นของงานย่อยด้วยแท็ก <scope> โดยแต่ละชั้นจะมีการนำแบบรูปมาใช้ที่แตกต่างกัน และสามารถใช้แท็ก <rethrow> เพื่อส่งสัญญาณว่ามีความผิดพลาดเกิดขึ้นให้กับ <scope> ชั้นนอก แล้วจัดการกับความผิดพลาดด้วยวิธีการอื่นต่อไปได้ ส่วนในระดับเครื่องประมวลบีเพลและระบบพื้นฐานควรมีหน่วยสนับสนุนที่ทำหน้าเฝ้าดูการทำงานและสามารถช่วยตัดสินใจว่าควรจะแก้ปัญหาต่อไปด้วยวิธีใดต่อไป

3.1.10 Fault Observer

ระบบควรจะมีหน่วยที่ทำหน้าที่สังเกตการณ์ ในกรณีที่เกิดความผิดพร่องขึ้นหน่วยนี้จะรายงานข้อมูลของความผิดพร่องไปยังหน่วยอื่นที่มีหน้าที่รับผิดชอบหรือสนใจในข้อมูลนั้น เช่น รายงานข้อมูลตามแบบรูป *Maintenance Interface* เพื่อให้ผู้ปฏิบัติงานทราบ หน่วยสังเกตการณ์นี้สามารถมีได้หลายตัวเพื่อช่วยกันเก็บข้อมูลความผิดพร่อง และเป็นได้ทั้งหน่วยภายในหรือภายนอกระบบก็ได้

แบบรูปนี้นำมาใช้กับบีเพลได้ในระดับเครื่องประมวลบีเพลซึ่งควรมีความสามารถในการสังเกตการณ์อยู่ด้วย หรือสามารถหาส่วนต่อขยายเพื่อทำหน้าที่เฝ้าดูเพิ่มได้อีก

3.1.11 Software Update

ระบบที่นำมาใช้งานจริงแล้วมีโอกาสที่จะได้รับการปรับปรุงให้เป็นเวอร์ชันใหม่เพื่อทำการเพิ่มความสามารถหรือแก้ไขข้อผิดพลาดที่เกิดขึ้น แต่ช่วงที่ทำการปรับปรุงซอฟต์แวร์จะทำให้ระบบอาจต้องหยุดการทำงานและอยู่ในสภาพไม่พร้อมใช้งาน ระบบจึงควรมีกลไกที่รองรับการปรับปรุงซอฟต์แวร์โดยให้มีผลในช่วงเวลาที่ระบบทำงานน้อยที่สุด ระบบอาจใช้แบบรูป *Failover* เพื่อให้ระบบเก่าสามารถหยุดการทำงานและส่งต่อไปให้ระบบใหม่ได้ หรือแบบรูป *Recovery Block* เพื่อให้สามารถใช้ระบบเก่ามารองรับหากระบบใหม่มีปัญหา

ในการนำแบบรูปนี้มาประยุกต์ใช้กับบีเพลควรพิจารณาจากเครื่องประมวลบีเพลว่ารองรับการปรับปรุงกระแสงานใหม่โดยส่งผลต่อการทำงานหรือไม่ เช่น ใน Glassfish ESB การติดตั้งกระแสงานใหม่ที่ใช้ชื่อเดียวกันจำเป็นจะต้องหยุดงานและถอดกระแสงานเดิมออกจากระบบก่อนจึงจะสามารถติดตั้งได้ ซึ่งทำให้เกิดช่วงเวลาที่ระบบไม่สามารถตอบสนองต่อคำร้องที่มาถึงกระแสงานนี้ได้ ดังนั้นการปรับปรุงกระแสงานบีเพลใหม่ควรมีการแจ้งให้ผู้ใช้ได้ทราบล่วงหน้าเพื่อลดความผิดพลาดที่จะเกิดขึ้น

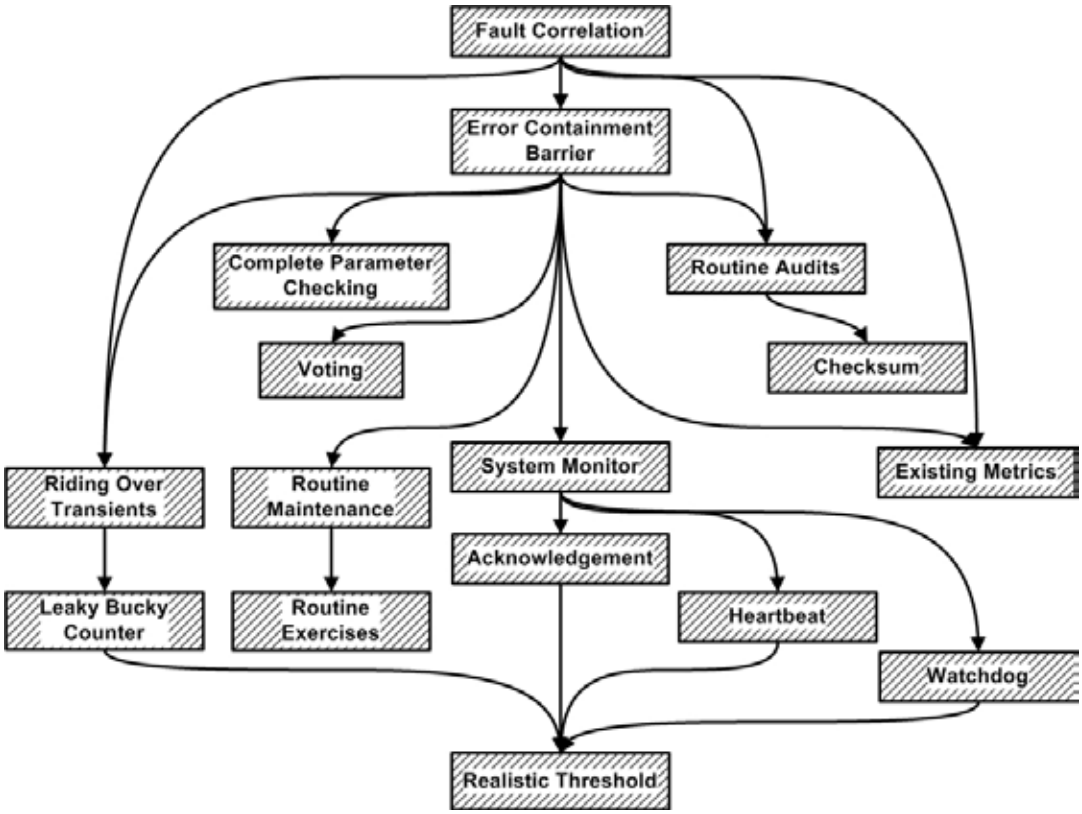
จากรายละเอียดของแบบรูปต่างๆ ในกลุ่มแบบรูปเชิงสถาปัตยกรรมที่กล่าวมานั้นสามารถสรุปได้ดังตารางที่ 3.1 โดยแสดงถึงจุดประสงค์ของแบบรูป (Pattern Intent) [10] และการนำแบบรูปมาประยุกต์กับการออกแบบกระแสงานบีเพลให้ทนต่อความผิดพร่อง ว่าแบบรูปใดสามารถใช้ในระดับกระแสงานได้ หรือแบบรูปใดจำเป็นต้องพึ่งพาความสามารถของเครื่องประมวลบีเพลและระบบพื้นฐานที่รองรับอยู่

ตารางที่ 3.1 ระดับการประยุกต์ใช้กับบีเพลของกลุ่มแบบรูปเชิงสถาปัตยกรรม

แบบรูป	จุดประสงค์ของแบบรูป	การประยุกต์ใช้กับบีเพล
Units of Mitigation	ในช่วงแรกของการออกแบบ พิจารณาว่าจะให้หน่วยย่อยใดของระบบทนต่อความผิดพร่อง	ระดับกระแสงาน
Correcting Audits	ออกแบบให้สามารถทำการตรวจสอบข้อมูลได้ ถ้าพบความผิดพลาดกับข้อมูล ต้องทำการแก้ไข และตรวจสอบข้อมูลที่เกี่ยวข้องด้วย	ระดับกระแสงาน
Redundancy	มีฮาร์ดแวร์หรือซอฟต์แวร์สำรองที่สามารถทำงานแทนได้ เพื่อเพิ่มสภาพพร้อมใช้งาน (Availability)	ระดับกระแสงาน
Recovery Blocks	มีวิธีการคำนวณหรือการทำงานแบบอื่นที่สามารถแทนได้ โดยนำมาใช้ในลำดับต่อจากวิธีการเดิม เพื่อให้การทำงานสำเร็จได้มากยิ่งขึ้น	ระดับกระแสงาน
Minimize Human Intervention	เนื่องจากมนุษย์มักจะทำผิดและมีความเชื่องช้ากว่าคอมพิวเตอร์ ระบบควรสามารถดูแลตัวเองได้โดยปราศจากมนุษย์ เพื่อลดระยะเวลาที่ระบบไม่สามารถทำงานได้ (Downtime)	ระดับเครื่องประมวลบีเพล
Maximize Human Participation	รองรับให้ผู้เชี่ยวชาญสามารถมาช่วยแก้ไขปัญหาได้	ระดับเครื่องประมวลบีเพล
Maintenance Interface	สร้างส่วนต่อประสานให้สามารถรองรับการบำรุงรักษาได้	ระดับเครื่องประมวลบีเพล
Someone in Charge	ทุกวิธีการที่ทำให้ทนต่อความผิดพร่องต้องมีหน่วยที่คอยควบคุมและตรวจสอบการทำงาน	ระดับเครื่องประมวลบีเพล
Escalation	เมื่อวิธีการจัดการกับความผิดพร่องที่ใช้นั้น ไม่ได้ผลตามที่ต้องการ ลองเปลี่ยนไปใช้วิธีที่รุนแรงขึ้น	ระดับกระแสงาน
Fault Observer	เมื่อมีความผิดพร่องเกิดขึ้น ควรประสานงานกับหน่วยที่คอยสังเกตการณ์ และรายงานไปยังส่วนที่เกี่ยวข้อง	ระดับเครื่องประมวลบีเพล
Software Update	ออกแบบระบบให้รองรับการปรับซอฟต์แวร์ให้เป็นปัจจุบัน	ระดับเครื่องประมวลบีเพล

3.2 แบบรูปการตรวจหา (Detection Patterns)

แบบรูปในกลุ่มนี้เป็นการรวบรวมวิธีที่ช่วยในการตรวจหาความผิดพลาดซึ่งเป็นขั้นแรกของการทำให้ทนต่อความผิดพร่อง ความสัมพันธ์ภายในกลุ่มแบบรูปการตรวจหาสามารถแสดงด้วยแผนที่ภาษาแบบรูปที่เป็นไปได้แบบหนึ่งดังภาพที่ 3.4



ภาพที่ 3.4 แผนที่ภาษาแบบรูปการตรวจหา [10]

การนำแบบรูปในกลุ่มนี้มาปรับใช้กับการออกแบบบีเพลให้ทนต่อความผิดพร่องสามารถพิจารณาทีละแบบรูป โดยจะกล่าวถึงความหมายและลักษณะการใช้งานของแต่ละแบบรูปก่อน [10] และกล่าวถึงการประยุกต์ใช้แบบรูปกับบีเพลในส่วนท้ายของแต่ละหัวข้อดังต่อไปนี้

3.2.1 Fault Correlation

เมื่อตรวจพบความผิดพลาดในระบบ สิ่งที่ควรจะพิจารณาคือหาสาเหตุของความผิดพลาดนั้นว่าเกิดจากความผิดพร่องประเภทใด ถ้าพบความผิดพร่องที่เป็นต้นเหตุจะทำให้การแก้ไขทำได้เหมาะสมมากยิ่งขึ้น ความผิดพลาดบางอย่างที่เกิดขึ้นแล้วในช่วงออกแบบและทดสอบระบบย่อมแสดงลักษณะเฉพาะตัวซึ่งจะช่วยในการระบุประเภทของความผิดพร่องเมื่อเกิดซ้ำอีกในช่วง

ดำเนินงานจริง ดังนั้นจึงควรดูลักษณะเฉพาะของความผิดพลาดเพื่อแบ่งแยกว่าเกิดจากความผิดพร่องประเภทใด และจะสามารถเลือกวิธีการจัดการที่เหมาะสมกับความผิดพร่องนั้นต่อไปได้

การประยุกต์ใช้แบบรูปนี้กับบีเพลจำเป็นต้องพิจารณาก่อนการออกแบบกระแสงาน เพื่อเป็นข้อมูลในการตัดสินใจว่าจะนำแบบรูปใดบ้างมาประยุกต์ใช้ด้วย ประเภทของความผิดพร่องที่อาจเกิดขึ้นได้กับบีเพลได้กล่าวไว้แล้วในหัวข้อที่ 2.3.1 ซึ่งในงานวิจัยนี้จะสนใจความผิดพร่องที่เกิดขึ้นกับเว็บเซอร์วิซคู่ค้า

3.2.2 Error Containment Barrier

เมื่อตรวจพบความผิดพลาดแล้ว ระบบควรจำกัดบริเวณที่จะได้รับผลกระทบจากความผิดพลาดนั้น การใช้แบบรูป *Units of Mitigation* ได้แบ่งระบบออกเป็นส่วนย่อยๆ ไว้แล้ว รวมทั้งการใช้แบบรูป *Quarantine* จะช่วยกันไม่ให้ผลของความผิดพลาดนั้นกระจายไปยังส่วนอื่นๆ เมื่อสามารถจำกัดบริเวณได้แล้วจึงจัดการกับความผิดพลาดต่อไป

การประยุกต์ใช้แบบรูปนี้กับบีเพลสามารถทำได้ในระดับกระแสงาน ด้วยการนำแบบรูป *Units of Mitigation* และแบบรูป *Quarantine* มาใช้ร่วมกัน โดยแบ่งกระแสงานออกเป็นส่วนย่อยด้วยแท็ก <scope> และติดตั้งแท็ก <faultHandlers> เพื่อดักจับความผิดพร่องให้กับทุกหน่วยย่อย

3.2.3 Complete Parameter Checking

วิธีการหนึ่งในการตรวจหาความผิดพลาดคือการเปรียบเทียบข้อมูล ตัวอย่างเช่น การเปรียบเทียบข้อมูลกับตัวสำรอง (แบบรูป *Redundancy*) ในทุกขั้นตอนของการดำเนินงาน หรือ การเปรียบเทียบผลลัพธ์ด้วยการหาผลโหวต (แบบรูป *Voting*) หรือ จะเป็นการเปรียบเทียบกับขอบเขตของข้อมูลที่ทราบอยู่ก่อนแล้ว ยิ่งทำการตรวจสอบถี่มากขึ้นยิ่งเพิ่มความเชื่อถือได้ให้กับระบบ แต่อาจต้องแลกกับสมรรถนะที่จะลดลงได้ กระบวนการตรวจสอบสามารถเพิ่มเข้าไปในกระบวนการทำงานหลัก แต่อาจมีผลตามมาคือทำให้เกิดความสับสนกับกระบวนการทำงานปกติ ดังนั้นจึงควรแยกส่วนตรวจสอบออกมาเป็นหน่วยใหม่ซึ่งจะช่วยให้การบำรุงรักษาทำได้สะดวกขึ้นและทำให้กระบวนการอื่นสามารถเรียกใช้ได้อีกด้วย

“Programming by contract” เป็นวิธีการหนึ่งในการตรวจสอบข้อมูลว่าเป็นไปตามที่ประกาศเอาไว้ในรายละเอียดของการปฏิบัติงานหรือไม่ เพื่อเป็นการรับรองว่าข้อมูลที่ได้รับและส่งออกไปนั้นถูกต้องและไม่มีปัญหา ซึ่งเครื่องมือออกแบบกระแสงานบีเพลได้นำแนวคิดนี้มาประยุกต์ใช้ในส่วนของการตรวจสอบข้อความที่รับส่งกันว่าตรงตามวิสเดิลและเอกซ์เอ็มแอลสกีมาที่ได้ประกาศไว้หรือไม่ โดยทำการตรวจสอบในช่วงของการออกแบบกระแสงาน ส่วนในช่วงดำเนินงาน

สามารถใช้แท็ก <validate> เพื่อตรวจสอบข้อมูลได้ แต่เครื่องประมวลบีเพลที่ใช้ต้องรองรับด้วย เพราะทั้งใน Glassfish ESB และ Apache ODE ต่างไม่รองรับการใช้งานของแท็กดังกล่าว

3.2.4 System Monitor

วิธีการตรวจหาความผิดพลาดอย่างหนึ่งคือการตรวจสอบว่าส่วนที่เกี่ยวข้องของระบบยังทำงานอยู่หรือไม่ หากไม่ทำงานจะต้องรีบแจ้งไปยังหน่วยที่รับผิดชอบเพื่อจะดำเนินการแก้ไขต่อไป ดังนั้นควรออกแบบให้มีหน่วยย่อยที่ทำหน้าที่เฝ้าดูการทำงานของส่วนต่างๆ ของระบบ วิธีการเฝ้าดูมีหลายแนวทางได้แก่ แบบรูป *Acknowledgement* แบบรูป *Heartbeat* และ แบบรูป *Watchdog* สิ่งที่ต้องพิจารณาต่อมาคือระยะเวลาที่หน่วยที่เฝ้าดูนี้จะสามารถรอได้ก่อนที่จะแจ้งว่าเกิดความผิดพลาดหากพบว่าส่วนที่ถูกเฝ้าดูอยู่ขาดการติดต่อไป ซึ่งระยะเวลาที่กำหนดควรจะเป็นไปตามการใช้งานจริงตามแบบรูป *Realistic Threshold*

3.2.5 Heartbeat

การส่งสัญญาณในแบบรูปนี้เป็นวิธีการหนึ่งที่จะช่วยให้หน่วยที่ทำหน้าที่เฝ้าดู (*System Monitor*) สามารถรับรู้ได้ว่าหน่วยที่กำลังถูกเฝ้าดูนั้นยังทำงานอยู่หรือไม่ การส่งสัญญาณสามารถทำได้ 2 แบบคือ หน่วยที่ถูกเฝ้าดูส่งข้อความออกมาเองเพื่อบอกว่าตนนั้นยังทำงานอยู่ หรือ หน่วยที่เฝ้าดูทำการร้องขอไปเพื่อให้ตอบข้อความกลับมา (*Acknowledgement*) การจะเลือกแบบใดมาใช้ขึ้นอยู่กับความซับซ้อนในการออกแบบทั้งฝั่งที่เป็นหน่วยเฝ้าดูและถูกเฝ้าดู การส่งสัญญาณควรทำเป็นระยะๆ โดยความถี่ในการส่งควรคำนึงจากระยะเวลาที่ข้อความถูกส่งไปและตอบกลับมา อาจใช้แบบรูป *Realistic Threshold* มาช่วยในการกำหนดได้ รวมทั้งปริมาณข้อความที่จะต้องเพิ่มเข้ามาไม่ควรไปกระทบกับการทำงานหลักด้วย

การประยุกต์ใช้แบบรูปนี้ในกระแสงานบีเพลที่ถูกเฝ้าดูและจำเป็นต้องส่งสัญญาณออกไปเพื่อแจ้งสถานะนั้น สามารถทำได้โดยใช้แท็ก <onAlarm> ภายใน <eventHandler> เพื่อคอยนับเวลาถอยหลังให้ส่งข้อความออกไปตามระยะเวลาที่กำหนดไว้และวนซ้ำไปเรื่อยๆ ได้ โดยการทำงานใน <eventHandler> จะทำขนานไปกับ <scope> ที่ติดอยู่ และจะหยุดเมื่อ <scope> นั้นเสร็จสิ้นการทำงานแล้ว ในทางกลับกันฝั่งที่ต้องการทราบสถานะสามารถประยุกต์ใช้ได้เช่นกัน โดยใช้แท็ก <onAlarm> เพื่อคอยเตือนให้ทำการสอบถามสถานะไปยังกระบวนการอื่นโดยใช้ช่องทางที่มีไว้ตามแบบรูป *Acknowledgement*

3.2.6 Acknowledgement

ในระบบที่มีการสื่อสารระหว่างกัน การส่งคำร้องไปหาอีกฝ่ายหนึ่งซึ่งอาจจะเป็นคำร้องที่ต้องการผลลัพธ์กลับมาหรือไม่ก็ได้ ฝ่ายที่ได้รับคำร้องควรมีการตอบรับกลับไปทุกครั้งเพื่อเป็นการแสดงว่าตนนั้นยังสามารถทำงานได้อยู่ ถ้าเป็นการทำงานที่ต้องใช้เวลานานอาจส่งข้อความชั่วคราวกลับไปก่อนเพื่อเป็นการยืนยันว่าได้รับคำร้องแล้ว ถ้าผู้ร้องขอไม่ได้รับข้อความตอบกลับมาในช่วงเวลาที่กำหนดแสดงว่ามีความผิดพลาดเกิดขึ้น ต้องแจ้งไปยังหน่วยที่รับผิดชอบเพื่อดำเนินการแก้ไขต่อไป

กระบวนการบีเพลที่มีการติดต่อกับเว็บเซอร์วิซแบบไปและกลับ (Request-response) เป็นแนวคิดเดียวกับแบบรูปนี้ โดยเมื่อมีการเรียกเว็บเซอร์วิซไปต้องได้รับการตอบรับกลับมา หากไม่มีข้อความตอบกลับมาในช่วงเวลาหนึ่ง เครื่องประมวลบีเพลจะแจ้งว่าไม่มีการตอบสนอง ซึ่งแสดงว่าเกิดความผิดพร่องขึ้น แต่ช่วงเวลาที่เครื่องประมวลบีเพลจะรอนั้นอาจน้อยเกินไป สามารถประยุกต์แบบรูปนี้ในระดับกระแสงานด้วยการใช้แท็ก <eventHandler> และกำหนดเวลาที่จะรอไว้ในแท็ก <onAlarm>

การติดต่อกับเว็บเซอร์วิซอีกประเภทหนึ่งของบีเพลคือแบบทางเดียว คือไม่ต้องรอการตอบกลับในทันทีจากเว็บเซอร์วิซที่เรียกไปแล้ว แต่เว็บเซอร์วิซจะทำการเรียกกลับมาผ่านช่องทางที่จัดไว้ กระบวนการบีเพลแบบนี้สามารถประยุกต์ใช้แบบรูปได้เช่นกัน โดยการใช้คำสั่ง <pick> ซึ่งจะมีแท็ก <onMessage> อยู่ภายใน เพื่อรอการเรียกกลับมาจากเว็บเซอร์วิซอื่น และยังสามารถกำหนดเวลาที่ใช้รอข้อความตอบกลับในแท็ก <onAlarm> เพิ่มเข้าไปได้เช่นกัน

กระบวนการบีเพลที่ต้องใช้เวลานานควรเพิ่มคำสั่งหรือพอร์ตไทป์ใหม่ในวิสเดิลเพื่อเป็นช่องทางในการติดต่อเพื่อสอบถามสถานะแยกออกจากการทำงานหลัก และต้องกำหนดค่าในแท็ก <correlationSet> ในกระแสงานเพื่อช่วยในการอ้างอิงว่ากำลังติดต่อกับกระบวนการที่ถูกต้อง

3.2.7 Watchdog

ในการออกแบบหน่วยของระบบเพื่อทำหน้าที่เฝ้าดูตามแบบรูป *Heartbeat* และแบบรูป *Acknowledgement* จำเป็นจะต้องเพิ่มการรับส่งข้อความขึ้นมาในกระบวนการทำงานต่างๆ แต่ในบางระบบที่มีการออกแบบการรับส่งข้อความไว้อย่างจำกัดจะไม่สามารถเพิ่มข้อความใหม่เข้าไปได้ หรือการเพิ่มข้อความใหม่อาจเป็นการเพิ่มความซับซ้อนเข้าไปในระบบเดิม ส่งผลให้มีโอกาสเกิดความผิดพร่องขึ้นได้ ดังนั้นแบบรูปนี้จึงแนะนำให้เพิ่มหน่วยที่เฝ้าดูซึ่งมีความสามารถในการสังเกตการทำงานของหน่วยที่ถูกเฝ้าดูได้โดยไม่จำเป็นต้องแก้ไขหน่วยที่ถูกเฝ้าดู และไม่ต้องเพิ่มการส่งข้อความเข้าไปใหม่เพราะสิ่งที่เฝ้าดูอาจเป็นผลลัพธ์ที่เห็นได้ชัดหรืออาจใช้มาตรวัดที่มีอยู่แล้วตามแบบรูป

Existing Metrics ก็ได้ แบบรูป *Watchdog* นี้จะเฝ้าดูเพียงการทำงานเดียวซึ่งแตกต่างจากแบบรูป *System Monitor* ที่จะเฝ้าดูการทำงานหลายๆ งานหรือทั้งระบบ

การจะใช้แบบรูปนี้กับบีเพลไม่สามารถทำได้ในระดับกระแสงานเพราะถ้าเพิ่มการทำงานในกระแสงานจำเป็นจะต้องมีการรับส่งข้อความระหว่างกันเพิ่มขึ้นด้วย จึงต้องพึ่งพาความสามารถของเครื่องประมวลบีเพลหรือเพิ่มส่วนต่อขยายเข้าไปเพื่อให้สามารถเฝ้าดูการทำงานของแต่ละกระบวนการย่อยของบีเพลได้

3.2.8 Realistic Threshold

ในการเฝ้าดูการทำงานหน่วยหนึ่งว่ายังคงทำงานอยู่หรือไม่ตามแบบรูป *System Monitor* ควรกำหนดระยะเวลาเพื่อเป็นการชี้ว่าหน่วยนั้นไม่ได้ทำงานแล้ว แบบรูปนี้แนะนำให้ใช้การคำนวณจากค่าทางเวลา 2 ค่า คือ เวลาแฝงการส่งข้อความ (Messaging latency) และ เวลาแฝงการตรวจหา (Detection latency)

- เวลาแฝงการส่งข้อความ (Messaging latency) หมายถึงระยะเวลาในการส่งข้อความถามสถานะจากหน่วยที่เฝ้าดูไปยังหน่วยที่ถูกเฝ้าดู ตามแบบรูป *Heartbeat* โดยระยะเวลาขั้นต่ำของค่านี้ควรมาจากผลรวมของ ระยะเวลาที่มากที่สุดในการส่งข้อความไปกลับ (round trip time) รวมกับ ระยะเวลาในการประมวลผลข้อความ (processing time) ทั้ง 2 ฝั่ง
- เวลาแฝงการตรวจหา (Detection latency) หมายถึงระยะเวลาที่หน่วยเฝ้าดูจะสามารถรอการตอบกลับจากหน่วยที่ถูกเฝ้าดูได้ ก่อนที่จะทำการแจ้งว่าเกิดความผิดพลาดขึ้น ซึ่งมักจะกำหนดเป็นจำนวนเท่าของเวลาแฝงการส่งข้อความ กล่าวคือเป็นจำนวนครั้งของการส่งข้อความไปแล้วไม่มีการตอบกลับมา

ถ้ากำหนดค่าทางเวลาทั้ง 2 ค่าสั้นเกินไป จะมีผลให้พบความผิดพลาดได้บ่อยกว่าความเป็นจริงเพราะอาจมีข้อความที่ส่งมาช้ากว่าที่กำหนดไว้ หรือถ้ากำหนดค่าไว้นานเกินไปอาจจะทำให้เกิดความผิดพลาดมาแล้วหลายครั้งก่อนที่จะตรวจพบจริงๆ

อีกปัจจัยหนึ่งในการกำหนดค่ามาจากความต้องการของระบบที่จะยอมให้อยู่ในสภาพไม่พร้อมใช้งานได้นานที่สุดเท่าใด ซึ่งผลรวมของเวลาแฝงการส่งข้อความ เวลาแฝงการตรวจหา และเวลาที่ใช้ในการกู้ระบบ ควรน้อยกว่าช่วงเวลาที่ระบบยอมให้อยู่ในสภาพไม่พร้อมใช้งานได้ (สภาพไม่พร้อมใช้งานนานที่สุด > เวลาแฝงการส่งข้อความ + เวลาแฝงการตรวจหา + เวลาที่ใช้ในการกู้ระบบ)

การประยุกต์ใช้แบบรูปนี้กับบีเพลควรพิจารณาจากช่วงการออกแบบและทดสอบระบบว่าการเรียกใช้เว็บเซอร์วิซแต่ละตัวนั้นใช้ระยะเวลาในการรับส่งข้อมูลเป็นเท่าใด เพื่อนำมากำหนดช่วงเวลาที่จะสามารถรอการตอบกลับจากเว็บเซอร์วิซในแต่ละครั้งได้

3.2.9 Existing Metrics

แบบรูปนี้เป็นการช่วยให้การตรวจหาความผิดพลาดในสภาวะโหลดเกินทำได้โดยไม่ต้องใช้การทำงานที่เป็นการเพิ่มภาระเข้าไปในระบบอีก ด้วยการใช้ตัวบ่งชี้ที่มีอยู่แล้วในระบบปฏิบัติการหรือระบบพื้นฐานที่รองรับ เช่น ปริมาณการใช้งานของซีพียู ซึ่งจะบ่งบอกถึงปริมาณงานที่เข้ามาในระบบโดยอ้อมได้ แบบรูปนี้จะช่วยอำนวยความสะดวกในการทำงานตามกลุ่มแบบรูปการบรรเทาความผิดพลาด (ซึ่งอยู่นอกเหนือจากงานวิจัยนี้)

หากจะนำแบบรูปนี้มาประยุกต์ใช้กับบีเพล เครื่องประมวลบีเพลควรมีความสามารถในการเรียกดูตัวบ่งชี้ต่างๆ ในระบบพื้นฐานได้

3.2.10 Voting

เมื่อในระบบมีตัวสำรองตามแบบรูป *Redundancy* อยู่หลายตัว แต่ได้ผลลัพธ์ออกมาหลายค่า วิธีการหาผลโหวตจะช่วยให้สามารถหาคำตอบที่เหมาะสมได้ และอาจช่วยบ่งชี้ได้ว่าเกิดความผิดพร่องขึ้นกับตัวสำรองที่ให้คำตอบแตกต่างจากตัวอื่น

ถ้าขนาดของหน่วยย่อยตามแบบรูป *Units of Mitigation* มีขนาดเล็กลง และผลจากการทำงานนั้นมีความซับซ้อนน้อยลงแล้ว จะช่วยให้การคำนวณหาผลโหวตจากหน่วยย่อยเหล่านั้นใช้ทรัพยากรที่น้อยลงตามไปด้วย

การนำแบบรูปนี้มาประยุกต์กับกระแสงานบีเพลทำได้ โดยการออกแบบการหาผลโหวตไว้ในกระแสงานหลัก หรือสร้างเป็นเว็บเซอร์วิซหาผลโหวตแยกออกมาให้กระแสงานหลักเรียกใช้ วิธีการหาผลโหวตต่างๆ สามารถศึกษาได้จากงานวิจัยของ Parhami [19]

3.2.11 Routine Maintenance

การบำรุงรักษาระบบอย่างเป็นประจำจะช่วยให้ตรวจพบความผิดพร่องได้ก่อน และเป็นการป้องกันไม่ให้เกิดความผิดพลาดขึ้นมาได้ด้วย ข้อมูลที่ถูกเก็บไว้หรือหน่วยความจำสามารถนำมาตรวจสอบและบำรุงรักษาได้เป็นระยะๆ สามารถนำแบบรูป *Routine Audits* และแบบรูป *Routine Exercises* มาช่วยได้ การบำรุงรักษาสามารถเริ่มโดยผู้ปฏิบัติงานสั่งการผ่านช่องทางที่กำหนดไว้ตามแบบรูป *Maintenance Interface* หรือตั้งการทำงานไว้อย่างอัตโนมัติ

การประยุกต์ใช้แบบรูปนี้กับบีเพลควรพิจารณาที่ความสามารถของเครื่องประมวลบีเพลว่ารองรับการบำรุงรักษากระบวนการบีเพล หรืออาจเพิ่มเติมส่วนต่อขยายให้กับเครื่องประมวลบีเพลเพื่อเพิ่มเติมความสามารถเหล่านี้ได้ นอกจากนี้ระบบพื้นฐานที่รองรับการทำงานอยู่ควรรองรับการบำรุงรักษาด้วยเช่นกัน

3.2.12 Routine Exercises

เมื่อระบบมีตัวสำรองตามแบบรูป *Redundancy* และจะถูกใช้งานเมื่อเกิดความผิดพลาดกับกระบวนการหลักเท่านั้น แต่ตัวสำรองที่ไม่เคยถูกเรียกใช้ย่อมมีโอกาสที่จะเกิดความผิดพร่องขึ้นมาได้ ส่งผลให้เมื่อต้องการใช้งานจริงก็ไม่สามารถทำงานให้สำเร็จได้ ดังนั้นการตรวจสอบอยู่เป็นประจำว่าตัวสำรองเหล่านั้นยังทำงานได้เป็นปกติจะช่วยให้ตรวจพบและสามารถจัดการกับความผิดพร่องที่อาจเกิดขึ้นได้ก่อนการใช้งานจริง นอกจากนี้การนำแบบรูปนี้มาใช้ควรคำนึงถึงภาระงานที่จะเพิ่มขึ้นมาด้วย เพื่อไม่ให้ส่งผลกระทบกับการทำงานหลักจึงควรทำในช่วงที่ระบบยังมีภาระงานน้อยอยู่

การประยุกต์ใช้แบบรูปนี้กับบีเพลควรทำในระดับระบบพื้นฐาน โดยเพิ่มเติมส่วนต่อขยายให้กับเครื่องประมวลบีเพลและระบบพื้นฐานที่จำเป็นเพื่อช่วยในการตรวจสอบสถานะของตัวสำรองได้อย่างเป็นประจำ รวมทั้งสามารถรายงานผลถ้าเกิดความผิดพร่องขึ้น หรือสามารถจัดการกับความผิดพร่องนั้นได้ก่อนการใช้งานจริง

3.2.13 Routine Audits

ข้อมูลที่มีความผิดพร่องอยู่ย่อมมีโอกาสส่งต่อความผิดพลาดกระจายไปยังส่วนต่างๆ ของระบบได้ การตรวจพบความผิดพร่องที่ซ่อนอยู่นั้นได้ก่อนถูกนำไปใช้งานจะช่วยให้โอกาสเกิดความผิดพลาดน้อยลงได้ ระบบควรมีการตรวจหาความผิดพลาดในข้อมูลที่เก็บไว้ในช่วงที่ระบบมีภาระงานน้อย ซึ่งอาจเป็นช่วงเวลาเดียวกับการบำรุงรักษาตามแบบรูป *Routine Maintenance* และเมื่อตรวจพบความผิดพร่องให้ทำการแก้ไขตามแบบรูป *Correcting Audits* และควรแจ้งผลที่เกิดขึ้นไปยังส่วนที่เกี่ยวข้องกับข้อมูลนั้นด้วย การตรวจสอบข้อมูลตามแบบรูปนี้เป็นประจำจะช่วยเพิ่มความเชื่อถือได้ให้กับระบบได้เพราะข้อมูลที่อาจก่อให้เกิดความผิดพลาดนั้นลดลง

การประยุกต์ใช้แบบรูปนี้กับบีเพลควรทำในระดับระบบพื้นฐาน โดยเพิ่มเติมส่วนต่อขยายให้กับเครื่องประมวลบีเพลและระบบพื้นฐานที่จำเป็นเพื่อช่วยในการตรวจสอบความถูกต้องของข้อมูลที่เก็บไว้ รวมทั้งสามารถรายงานผลถ้าเกิดความผิดพร่องขึ้น หรือสามารถจัดการแก้ไขได้ก่อนการใช้งานจริง

3.2.14 Checksum

แบบรูปนี้ช่วยในการตรวจสอบความถูกต้องของข้อมูล ซึ่งวิธีการหนึ่งในการตรวจสอบความถูกต้องคือการเปรียบเทียบกับข้อมูลที่ได้ทำซ้ำเก็บไว้อีกที่หนึ่ง แต่เพื่อให้ใช้เนื้อที่ในการเก็บข้อมูลที่เล็กลงจึงนำการคำนวณเฉพาะทางมาใช้กับข้อมูลหรือกลุ่มของข้อมูลจำนวนหนึ่ง แล้วเก็บผลลัพธ์ที่ได้เอาไว้ ซึ่งสามารถนำข้อมูลเหล่านั้นกลับมาคำนวณใหม่ด้วยวิธีการเดิม แล้วนำผลลัพธ์ที่ได้มาเปรียบเทียบกับค่าที่เคยเก็บไว้ว่าตรงกันหรือไม่ ตัวอย่างของการคำนวณที่นำมาใช้ ได้แก่ อัลกอริทึมแบบแฮช (Hashing algorithm) ซึ่งเป็นการคำนวณทางเดียวมีหลายวิธี เช่น SHA หรือ MD5

การประยุกต์ใช้แบบรูปนี้กับบีเพลควรมีการออกแบบให้สามารถเรียกใช้เว็บเซอร์วิซที่ช่วยในการตรวจสอบข้อมูลได้ รวมทั้งควรมีระบบพื้นฐานที่รองรับการจัดเก็บข้อมูลเหล่านั้นด้วย

3.2.15 Riding Over Transients

ความผิดพร่องบางอย่างอาจเกิดขึ้นเพียงชั่วคราวและอาจหายไปเองเมื่อเวลาผ่านไป ระบบไม่จำเป็นที่จะต้องเสียเวลาเพื่อจัดการกับความผิดพร่องประเภทนี้ ตัวอย่างของความผิดพร่องที่เกิดขึ้นชั่วคราวเช่น การเปิดเว็บไซต์ ผู้ใช้อาจพบว่าบางครั้งไม่สามารถเปิดเว็บไซต์ที่ต้องการได้ ซึ่งหากผู้ใช้รอชั่วครู่หนึ่งและพยายามเปิดใหม่อีกครั้งก็จะพบว่าสามารถเปิดได้ตามปกติ การเพิกเฉยต่อความผิดพลาดที่เกิดขึ้นเป็นแนวคิดเดียวกับแบบรูปนี้ แต่ในกรณีที่ผู้ใช้ไม่สามารถเปิดเว็บไซต์ใดๆ ได้เลย แสดงว่าเกิดความผิดพร่องแบบถาวรขึ้นซึ่งอาจเป็นความผิดพลาดที่ระบบเครือข่ายหรือผู้ให้บริการเว็บไซต์ กรณีนี้จึงควรเริ่มจัดการกับความผิดพลาดต่อไป

ในช่วงการออกแบบควรพิจารณาว่ามีโอกาสที่ความผิดพร่องชั่วคราวจะเกิดขึ้นหรือไม่ และจะมีลักษณะเฉพาะแบบใดได้บ้างตามแบบรูป *Fault Correlation* เมื่อเกิดความผิดพลาดขึ้นในช่วงการทำงาน ถ้าพิจารณาว่าเป็นความผิดพลาดแบบถาวรหรือไม่ได้ระบุเอาไว้จะต้องเริ่มการจัดการทันที แต่ถ้าพิจารณาว่าเป็นความผิดพร่องชั่วคราวควรจะเพิกเฉยไปก่อน แล้วคอยเฝ้าดูว่าความถี่ในการเกิดนั้นเกินจากที่คาดไว้หรือไม่ หากเกิดบ่อยเกินไปแสดงว่าเริ่มเป็นอย่างถาวรแล้ว วิธีการนี้สามารถประยุกต์ใช้แบบรูป *Leaky Bucket Counter* เพราะมีกลไกในการเฝ้าดูและช่วยพิจารณาว่าความผิดพลาดนั้นเป็นแบบชั่วคราวหรือถาวรได้

การประยุกต์แบบรูปนี้กับบีเพลควรมีระบบพื้นฐานที่รองรับการพิจารณาว่าความผิดพลาดที่เกิดขึ้นนั้นมีลักษณะแบบชั่วคราวหรือถาวรซึ่งจะเป็นตัวช่วยในการตัดสินใจให้กับเครื่องประมวลบีเพลว่าควรจะจัดการกับความผิดพลาดนั้นในทันทีหรือไม่

3.2.16 Leaky Bucket Counter

แบบรูปนี้ช่วยพิจารณาว่าความผิดพลาดที่เกิดขึ้นนั้นมีลักษณะเป็นความผิดพลาดแบบชั่วคราวหรือถาวร โดยให้ในแต่ละหน่วยย่อยตามแบบรูป *Units of Mitigation* มีตัวนับจำนวนของความผิดพลาด เมื่อมีความผิดพลาดเกิดขึ้นจึงเพิ่มจำนวนให้กับตัวนับนี้ แต่จะทำการลดค่าของตัวนับนี้ลงเป็นระยะๆ ตามเวลาที่กำหนดไว้ ถ้าเป็นความผิดพลาดแบบชั่วคราวซึ่งเกิดขึ้นไม่บ่อยนัก ค่าของตัวนับจะมีการเพิ่มลดอยู่ภายในเกณฑ์ที่กำหนดไว้ แต่ถ้ามีความผิดพลาดเกิดขึ้นบ่อยจนค่าตัวนับนั้นเกินเกณฑ์ที่กำหนดไว้แสดงว่าเป็นความผิดพลาดแบบถาวร ตัวแปรสำคัญที่ต้องกำหนดคือ จำนวนครั้งที่ยอมให้เกิดความผิดพลาด (เกณฑ์ที่ยอมรับได้) และความถี่ในการลดค่าตัวนับลง

แนวคิดนี้เปรียบได้กับถังน้ำที่มีรูรั่วและมีน้ำหยดลงไปเรื่อยๆ ถ้าปริมาณน้ำที่หยดลงมามีอัตราน้อยกว่าน้ำที่รั่วออกไป น้ำในถังก็จะมีไม่มากเกินไป แต่ถ้าน้ำที่หยดลงมาเร็วเกินกว่าน้ำที่ไหลออกมาทางรูรั่ว ถังก็จะรับน้ำไว้ไม่พอและล้นออกมาได้

การนำแบบรูปนี้มาประยุกต์ใช้กับบีเพลควรมีระบบพื้นฐานที่รองรับการนับจำนวนครั้งของความผิดพลาดที่เกิดขึ้นและมีการบันทึกไว้ ซึ่งอาจเพิ่มเป็นส่วนต่อขยายให้กับเครื่องประมวลบีเพลในการพิจารณาว่าความผิดพลาดเป็นแบบชั่วคราวหรือถาวร

จากรายละเอียดของแบบรูปต่างๆ ในกลุ่มแบบรูปการตรวจหาที่กล่าวมานั้นสามารถสรุปได้ดังตารางที่ 3.2 โดยแสดงถึงจุดประสงค์ของแบบรูป (Pattern Intent) [10] และการนำแบบรูปมาประยุกต์กับการออกแบบกระแสงานบีเพลให้ทนต่อความผิดพร่อง ว่าแบบรูปใดสามารถใช้ในระดับกระแสงานได้ หรือแบบรูปใดจำเป็นต้องพึ่งพาความสามารถของเครื่องประมวลบีเพลและระบบพื้นฐานที่รองรับอยู่

ตารางที่ 3.2 ระดับการประยุกต์ใช้กับบีเพลของกลุ่มแบบรูปการตรวจหา

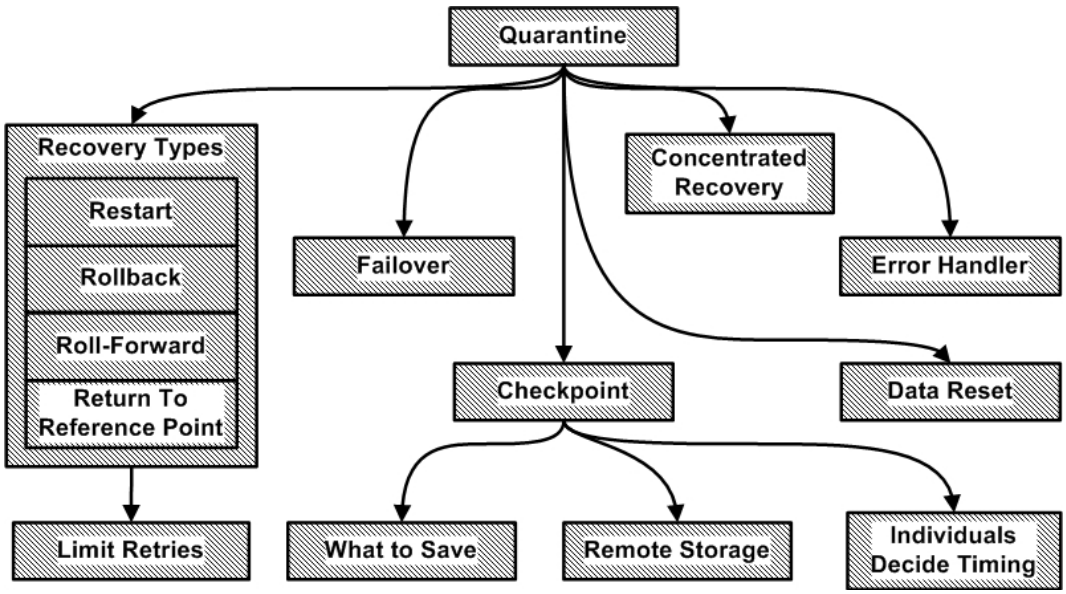
แบบรูป	**จุดประสงค์ของแบบรูป**	**การประยุกต์ใช้กับบีเพล**
Fault Correlation	วิเคราะห์สิ่งที่บ่งชี้ถึงความผิดพลาดต่างๆ เพื่อระบุว่าเป็นความผิดพร่องประเภทใดที่เกิดขึ้น	ระดับการออกแบบ
Error Containment Barrier	แยกความผิดพลาดออกมา เพื่อไม่ให้ผลกระทบกระจายไปยังส่วนอื่น	ระดับกระแสงานบีเพล
Complete Parameter Checking	ตรวจสอบข้อมูลขาเข้าและค่าพารามิเตอร์อย่างเข้มงวด เพื่อป้องกันไม่ให้เป็นสาเหตุของความผิดพลาดในขณะดำเนินงาน	ระดับกระแสงานบีเพล

ตารางที่ 3.2 ระดับการประยุกต์ใช้กับบีเพลของกลุ่มแบบรูปการตรวจหา (ต่อ)

แบบรูป	จุดประสงค์ของแบบรูป	การประยุกต์ใช้กับบีเพล
System Monitor	ความผิดพลาดบางประเภทจะแสดงออกมาในระดับของระบบ จึงต้องตรวจสอบการทำงานในระดับนี้ด้วย	ระดับเครื่องประมวลบีเพล
Heartbeat	รายงานสถานะของตนเองออกไปอย่างสม่ำเสมอ เพื่อให้ส่วนอื่นๆ ของระบบรับทราบ	ระดับกระแสงานบีเพล
Acknowledgement	ส่งข้อความตอบกลับไป เพื่อให้อีกฝ่ายที่สื่อสารด้วยทราบว่าผู้ส่งยังปฏิบัติงานได้อยู่	ระดับกระแสงานบีเพล
Watchdog	สร้างหน่วยพิเศษมีหน้าที่เฝ้าดูหน่วยอื่น เพื่อสร้างความมั่นใจว่าหน่วยที่ถูกเฝ้าดูนั้นยังปฏิบัติงานได้อย่างปกติ	ระดับระบบพื้นฐานที่รองรับ
Realistic Threshold	ควรกำหนดค่าที่เป็นไปได้ให้กับขีดเริ่มเปลี่ยนสำหรับการตรวจหาความผิดพลาด	ระดับการออกแบบ
Existing Metrics	เฝ้าสังเกตตัววัดที่มีอยู่แล้วในระบบพื้นฐาน และควรเป็นตัววัดที่ไม่เสียเวลาในการคำนวณเพิ่ม	ระดับระบบพื้นฐานที่รองรับ
Voting	เมื่อผลลัพธ์ที่ได้จากการคำนวณหรืองานที่ทำสามารถมีได้หลายค่า ควรโหวตเพื่อเลือกคำตอบที่ถูกต้อง	ระดับกระแสงานบีเพล
Routine Maintenance	ทำการบำรุงรักษาเชิงป้องกันอย่างอัตโนมัติเป็นระยะๆ เพื่อป้องกันไม่ให้ความผิดพร่องสะสมขึ้นมาอย่างเงียบๆ	ระดับระบบพื้นฐานที่รองรับ
Routine Exercises	ดำเนินการตามแบบรูปนี้เพื่อให้ทราบว่าตัวสำรองยังพร้อมใช้งานอยู่	ระดับระบบพื้นฐานที่รองรับ
Routine Audits	ตรวจสอบข้อมูลในช่วงเวลาเดินเครื่องเปล่า ของระบบ (Idle time) เพื่อให้มั่นใจได้ว่าข้อมูลนั้นยังถูกต้อง	ระดับระบบพื้นฐานที่รองรับ
Checksum	เพิ่มเติมข้อมูลบางอย่างลงไปในข้อความ เพื่อให้สามารถตรวจสอบได้ว่าข้อความถูกต้อง	ระดับการออกแบบ
Riding Over Transients	เพิกเฉยต่อความผิดพลาดที่อาจเกิดขึ้นแบบชั่วคราว	ระดับระบบพื้นฐานที่รองรับ
Leaky Bucket Counter	ทำตามแบบรูป *Riding Over Transients* โดยเพิ่มตัวนับจำนวนครั้งที่เกิดความผิดพลาด และจะลดจำนวนลงเมื่อเวลาผ่านไประยะหนึ่ง	ระดับระบบพื้นฐานที่รองรับ

3.3 แบบรูปการกู้ระบบจากความผิดพลาด (Error Recovery Patterns)

แบบรูปในกลุ่มนี้เป็นการจัดการกับความผิดพลาดอย่างหนึ่ง ซึ่งพยายามจะกู้ระบบให้กลับมาสู่สภาวะที่ไม่มีความผิดพลาด และสามารถกลับมาดำเนินงานต่อไปได้ ความสัมพันธ์ภายในกลุ่มแบบรูปการกู้ระบบจากความผิดพลาดสามารถแสดงด้วยแผนที่ภาษาแบบรูปที่เป็นไปได้แบบหนึ่งดังภาพที่ 3.5



ภาพที่ 3.5 แผนที่ภาษาแบบรูปการกู้ระบบจากความผิดพลาด [10]

การนำแบบรูปในกลุ่มนี้มาปรับใช้กับการออกแบบบีเพลให้ทนต่อความผิดพร่องสามารถพิจารณาทีละแบบรูป โดยจะกล่าวถึงความหมายและลักษณะการใช้งานของแต่ละแบบรูปก่อน [10] และกล่าวถึงการประยุกต์ใช้แบบรูปกับบีเพลในส่วนท้ายของแต่ละหัวข้อดังต่อไปนี้

3.3.1 Quarantine

เมื่อหน่วยย่อยของระบบตามแบบรูป *Units of Mitigation* มีความผิดพร่องเกิดขึ้นมาแล้ว ส่วนย่อยอื่นที่เกี่ยวข้องในระบบอาจได้รับผลกระทบจากการทำงานที่ผิดพลาดนั้นได้ ดังนั้นระบบจึงควรล้อมกรอบการทำงานของหน่วยย่อยที่เกิดปัญหานั้นและป้องกันไม่ให้เกิดผลกระทบกระจายไปยังส่วนอื่น

การประยุกต์ใช้แบบรูปนี้สามารถทำได้ในระดับกระแสงานบีเพล ถ้าระบบมีการออกแบบตามแบบรูป *Units of Mitigation* ซึ่งจะใช้แท็ก <scope> ในการแบ่งกระแสงานออกเป็นหน่วยย่อยแล้ว สามารถเพิ่มแท็ก <faultHandlers> ติดตั้งเข้าไปเพื่อเป็นการวางกรอบไว้ หากสามารถดักจับความผิดพร่องที่เกิดขึ้นมาได้ จะต้องหยุดการทำงานในแท็ก <scope> ที่เกิดความผิดพร่อง

แล้วเปลี่ยนการทำงานไปอยู่ภายในแท็ก <faultHandlers> แทนเพื่อจัดการกับความผิดพร่องนั้น วิธีการนี้จะช่วยให้แท็ก <scope> อื่นที่ทำงานขนานกันหรือต่อท้าย <scope> เดิม ยังสามารถทำงานต่อได้ โดยไม่ต้องหยุดการทำงานตาม <scope> ที่เกิดความผิดพร่อง

3.3.2 Concentrated Recovery

เมื่อตรวจพบความผิดพลาดแล้ว ระบบควรมุ่งจัดการกับความผิดพลาดนั้นให้เร็วที่สุด โดยดึงทรัพยากรของระบบเท่าที่จะนำมาใช้ในการกู้ระบบได้มาใช้ให้ได้ประโยชน์สูงสุด เพื่อลดช่วงเวลาที่ระบบอยู่ในสภาพไม่พร้อมใช้งานให้เหลือน้อยลง

การนำแบบรูปนี้มาประยุกต์ใช้กับบีเพล เครื่องประมวลบีเพลควรมีความสามารถในการบริหารจัดการทรัพยากรของระบบ เพื่อนำมาใช้ได้มากที่สุดในช่วงที่ต้องจัดการกับความผิดพร่องที่เกิดขึ้นในกระบวนการบีเพล

3.3.3 Error Handler

การทำงานในส่วนที่เกี่ยวข้องกับการจัดการความผิดพลาดควรแยกส่วนออกมาจากการทำงานหลักของโปรแกรมเพื่อความสะดวกในการบำรุงรักษาหรือแก้ไขในอนาคต เช่นในภาษาจาวาที่ใช้ โครงสร้างแบบ “try-catch” เพื่อแยกส่วนการทำงานหลักกับส่วนจัดการความผิดพลาดออกจากกัน และเมื่อการทำงานในส่วนจัดการความผิดพลาดเสร็จสิ้นแล้ว จะต้องกลับไปดำเนินงานในส่วนการทำงานหลักที่กำหนดไว้ได้ ซึ่งอาจจะเป็นจุดที่ก่อนจะเกิดความผิดพลาด (*Rollback*) จุดที่ตามหลังมา (*Roll-Forward*) หรือจุดที่กำหนดไว้ก่อนแล้ว (*Return to Reference Point*)

แบบรูปนี้มีการประยุกต์ใช้ในระดับกระแสงานบีเพลมาก่อนแล้วเพราะในภาษาบีเพลมีโครงสร้างที่ใช้จัดการกับความผิดพร่องคือแท็ก <faultHandlers> ซึ่งสามารถติดไว้กับคำสั่ง <invoke> หรือติดตั้งกับส่วนของแท็ก <scope> ก็ได้ คำสั่งนี้มีหน้าที่ดักจับสัญญาณความผิดพร่องที่เกิดขึ้นมาจากส่วนของกระแสงานที่ติดตั้งไว้ ภายในจะเป็นส่วนของกิจกรรมบีเพลที่จะนำมาใช้ในการจัดการกับความผิดพร่อง สามารถกำหนดแยกตามชื่อของความผิดพร่องได้ด้วยแท็ก <catch> หรือใช้กับความผิดพร่องชื่อใดก็ได้ด้วยแท็ก <catchAll>

3.3.4 Restart

เมื่อกลไกในการกู้ระบบจากความผิดพลาดตามแบบรูปต่างๆ ยังไม่สามารถนำระบบกลับสู่ภาวะปกติได้แล้ว จากแบบรูป *Escalation* ที่ระบุไว้ว่าควรใช้วิธีการที่รุนแรงขึ้นในการจัดการ ซึ่งแบบรูปนี้แนะนำให้เริ่มการทำงานใหม่ เพื่อเป็นการเปลี่ยนสถานะของระบบให้กลับไปสู่สภาวะเริ่มต้น

ซึ่งน่าจะมีความปลอดภัยในระดับหนึ่ง แต่เนื่องจากการเริ่มต้นใหม่มักจะต้องเสียเวลาค่อนข้างมาก หากสามารถแบ่งระดับของการเริ่มทำงานใหม่เป็นขั้นย่อยๆ ได้ จะช่วยให้ลดเวลาลงได้ส่วนหนึ่งเพราะไม่จำเป็นจะต้องเริ่มใหม่ทั้งระบบ และยังช่วยให้การทำงานส่วนอื่นยังคงดำเนินต่อไปได้

การประยุกต์ใช้แบบรูปนี้จำเป็นต้องพึ่งพาความสามารถของเครื่องประมวลบีเพลในการจัดการกระบวนการบีเพลที่อยู่ในระบบ และอาจต้องมีระบบพื้นฐานเพื่อใช้ในการจัดเก็บสถานะของแต่ละกระบวนการตามแบบรูป *Checkpoint* ด้วย

3.3.5 Rollback

หลังจากจัดการกับความผิดพลาดแล้ว ระบบต้องกลับไปดำเนินงานในกระแสงานปกติ ณ ตำแหน่งหนึ่งในกระแสงานที่สามารถพิจารณาได้ว่ามีความปลอดภัย คือตำแหน่งก่อนที่ความผิดพลาดจะเกิดขึ้น ถ้ามีการใช้แบบรูป *Checkpoint* สามารถพิจารณาตำแหน่งนี้ได้จากสถานะที่ถูกบันทึกไว้ก่อนเกิดความผิดพลาด แต่ถ้าไม่มีการบันทึกไว้ตำแหน่งที่จะกลับไปคือตำแหน่งที่รับคำสั่งล่าสุดก่อนเกิดความผิดพลาด

การย้อนกลับไปยังตำแหน่งก่อนหน้าอาจส่งผลให้มีการทำงานบางอย่างเกิดขึ้นซ้ำได้ จึงควรพิจารณาถึงความเหมาะสมว่าการทำงานซ้ำนั้นจะส่งผลเสียหรือไม่ และต้องใช้เวลามากเกินไปหรือไม่ ถ้าจำเป็นต้องใช้เวลามากขึ้นอาจไม่เหมาะสมกับงานที่ต้องการความรวดเร็ว นอกจากนี้การย้อนกลับไปตำแหน่งก่อนหน้ามีโอกาสที่ความผิดพร่องจะเกิดซ้ำขึ้นอีกได้ จึงควรจำกัดจำนวนครั้งในกรณีที่ต้องกลับไปทำงานซ้ำตามแบบรูป *Limit Retries*

การประยุกต์แบบรูปนี้มาใช้กับบีเพลสามารถทำได้ในระดับกระแสงาน แต่ยังมีข้อจำกัดที่ไม่สามารถกำหนดให้ทำตามแบบรูปอย่างพลวัตได้ เพราะไม่มีการบันทึก Checkpoint อย่างอัตโนมัติ จำเป็นต้องกำหนดจุดที่จะย้อนกลับไปภายในกรอบของหน่วยย่อยตามแบบรูป *Units of Mitigation* ไว้ในช่วงออกแบบก่อน และเมื่อเกิดความผิดพร่องขึ้นจึงย้อนกลับไปในจุดที่กำหนด โดยนำแบบรูป *Recovery Block* แบบเรียกซ้ำมาใช้ร่วมกันได้ และก่อนที่จะย้อนกลับไปตำแหน่งก่อนหน้าสามารถใช้แท็ก <compensate> ภายใน <faultHandlers> เพื่อสั่งให้ทำการชดเชยการทำงานที่เสร็จสิ้นไปแล้วของ <scope> ให้สามารถคืนค่ากลับไปยังสถานะก่อนหน้าได้ โดยคำสั่งนี้จะไปเรียกการทำงานของแท็ก <compensationHandler> ที่ติดตั้งอยู่กับ <scope> ซึ่งอยู่ชั้นในอีกทีหนึ่ง

3.3.6 Roll-Forward

ในบางกรณีการย้อนกลับไปดำเนินงานในจุดที่เกิดความผิดพลาดตามแบบรูป *Rollback* อาจไม่เหมาะสมเพราะจำเป็นต้องทำงานซ้ำเดิมซึ่งอาจใช้เวลามากขึ้นและมีโอกาสที่จะพบความผิดพลาด

ซ้ำได้อีก ถ้าระบบสามารถละเว้นส่วนของงานที่เกิดความผิดพร่องได้ การข้ามไปยังการดำเนินงานต่อไปแทนย่อมเป็นวิธีที่เหมาะสมกว่า เพราะจะทำให้ระบบยังสามารถทำงานส่วนอื่นต่อได้และไม่เสียเวลาในการจัดการกับความผิดพร่องนั้น

การประยุกต์ใช้แบบรูปนี้กับกระแสงานบีเพลสามารถใช้แท็ก <faultHandlers> มาจัดการได้ ซึ่งจำเป็นจะต้องแบ่งกระแสงานออกเป็น <scope> ย่อยๆ และติดแท็ก <faultHandlers> ไว้ เมื่อเกิดความผิดพร่องขึ้นใน <scope> จะถูกดักจับไว้ด้วย <faultHandlers> และสามารถปล่อยผ่านการทำงานของ <scope> นั้น และดำเนินงานตามกระแสงานต่อไปได้ นอกจากนี้ควรเพิ่มการแจ้งรายงานไปยังหน่วยอื่นๆ ที่เกี่ยวข้องถึงความผิดพร่องที่เกิดขึ้นก่อนจะส่งต่อการดำเนินงานไปยัง <scope> อื่นต่อไป

3.3.7 Return to Reference Point

ในกรณีที่ความผิดพลาดเกิดขึ้นในส่วนงานสนับสนุนซึ่งไม่ได้อยู่ในกระแสงานหลัก เช่น อยู่ในส่วนการจัดการความผิดพลาด หรือส่วนการบำรุงรักษา การใช้แบบรูป *Rollback* อาจไม่เหมาะสม เพราะว่าเป็นการย้อนกลับไปทำในส่วนงานสนับสนุนไม่ใช่กระแสงานหลัก รวมทั้งความผิดพลาดที่เกิดยังไม่จำเป็นต้องใช้แบบรูป *Restart* เพื่อเริ่มงานใหม่ทั้งหมด ดังนั้นเพื่อให้สามารถกลับไปดำเนินงานในกระแสงานหลักได้ จำเป็นต้องกำหนดจุดที่คาดว่าปลอดภัยไว้ก่อนในช่วงออกแบบ และให้ระบบกลับไปดำเนินงานในจุดนั้น

แบบรูปนี้แตกต่างกับแบบรูป *Rollback* ตรงที่ตำแหน่งในการกลับไปดำเนินงานตามแบบรูป *Rollback* นั้นเป็นแบบพลวัตซึ่งเปลี่ยนแปลงไปตามตำแหน่งและเวลาที่จัดเก็บตามแบบรูป *Checkpoint*

แนวคิดตามแบบรูปนี้มีการประยุกต์ใช้กับกระแสงานบีเพลแล้วในระดับหนึ่ง กล่าวคือมีการกำหนดตำแหน่งที่จะดำเนินงานต่อจากหน่วยสนับสุนนไว้แล้วตามข้อกำหนดของดับเบิลยูเอส-บีเพล 2.0 ใน <scope> หนึ่งๆ จะมีส่วนสนับสนุนที่ติดตั้งอยู่ได้หลายชนิด ได้แก่ <faultHandlers> <compensationHandler> <terminationHandler> (เรียกรวมกันว่าเป็น FCT-handler) และ <eventHandler> ซึ่งเมื่อจบการดำเนินงานในส่วนสนับสนุนแล้ว จะกลับไปสู่กระแสงานหลักในตำแหน่งที่แตกต่างกันดังนี้

- เมื่อกลุ่ม FCT-handler เริ่มทำงานจะถือได้ว่า <scope> ที่ติดตั้งอยู่ต้องจบการทำงานไปแล้ว ตำแหน่งที่จะกลับไปดำเนินงานจึงไม่อยู่ใน <scope> เดิม แต่เป็นลำดับที่ต่อจาก <scope> นั้นแทน

- แท็ก <eventHandler> จะทำงานขนานไปกับ <scope> ที่ติดตั้งอยู่และมีอายุตาม <scope> นั้น เมื่อจบการทำงานในแต่ละเหตุการณ์ หาก <scope> ที่ติดตั้งยังคงทำงานอยู่และเหตุการณ์ที่สนใจสามารถเกิดขึ้นอีกได้ ตำแหน่งต่อไปยังคงเป็นจุดที่รอรับเหตุการณ์ใหม่ของ <eventHandler> อยู่ แต่ถ้าไม่มีเหตุการณ์ที่จะเกิดได้อีก หรือ <scope> ที่ติดตั้งจบการทำงานแล้ว ตำแหน่งที่จะกลับไปดำเนินงานต่อคือจุดต่อไปที่ต่อจาก <scope> นั้น

เพื่อให้การทำงานภายในส่วนสนับสนุนดำเนินจนเสร็จสิ้นได้ควรกำหนดให้มีการดักจับความผิดพร่องด้วย <faultHandlers> เพิ่มเติมเข้าไปใน <scope> ที่อยู่ด้านในของส่วนสนับสนุน เพื่อช่วยให้สามารถจัดการกับความผิดพร่องที่เกิดขึ้นภายในส่วนสนับสนุนเหล่านี้ได้

3.3.8 Limit Retries

ในระหว่างการจัดการกับความผิดพลาด ถ้าเลือกที่จะกลับไปดำเนินงานในตำแหน่งก่อนที่จะพบความผิดพลาด ระบบย่อมมีโอกาสที่จะพบความผิดพลาดซ้ำอีกได้ เพราะมีลักษณะการทำงานหรือข้อมูลแบบเดิมที่อาจกระตุ้นให้เกิดความผิดพลาดได้ เช่น ข้อความที่รับมาส่งผลให้เกิดความผิดพลาด ถ้ากลับไปใช้ข้อความเดิมนั้นย่อมเกิดความผิดพลาดขึ้นได้อีก ดังนั้นการกลับไปทำงานซ้ำควรถูกจำกัดไว้ในจำนวนที่เหมาะสม เพื่อไม่ให้ระบบต้องติดอยู่กับการจัดการความผิดพลาดที่เกิดซ้ำเรื่อยๆ และเมื่อถึงเกณฑ์ที่กำหนดแล้วจำเป็นต้องเปลี่ยนไปใช้วิธีการอื่นมาจัดการแทน

การประยุกต์ใช้แบบรูปนี้ในระดับกระแสงานบีเพลสามารถใช้ประโยชน์จากโครงสร้างที่ใช้สำหรับการวนซ้ำ ได้แก่ แท็ก <while> หรือแท็ก <repeatUntil> มาช่วยในการควบคุมการทำงานให้จำกัดจำนวนรอบที่ทำงานได้ แต่วิธีการนี้ต้องกำหนดจำนวนรอบไว้ในช่วงการออกแบบ หากต้องการใช้จำนวนรอบที่ปรับเปลี่ยนได้ตามการทำงานจำเป็นต้องมีระบบพื้นฐานในการช่วยกำหนดอีกขั้นหนึ่ง

3.3.9 Failover

ในกรณีที่การจัดการกับความผิดพลาดที่นำมาใช้ไม่ได้ผล เช่น ปฏิบัติตามแบบรูป *Rollback* หรือแบบรูป *Roll-Forward* แล้วแต่ยังไม่สามารถนำระบบกลับสู่ภาวะที่ไม่มีความผิดพลาดได้ เหตุการณ์นี้ทำให้ระบบจำเป็นจะต้องใช้วิธีที่รุนแรงขึ้นตามแบบรูป *Escalation* แบบรูปนี้แนะนำให้เปลี่ยนการทำงานไปใช้ตัวสำรองแทนตามแบบรูป *Redundancy* โดยในระหว่างการเปลี่ยนจะมีการถ่ายโอนสถานะการทำงานของหน่วยที่มีปัญหาไปยังหน่วยสำรอง โดยหากเป็นตัวสำรองที่พร้อมใช้

งานอยู่แล้วจะช่วยให้การถ่ายโอนทำได้รวดเร็วขึ้น แต่ถ้าตัวสำรองนั้นอยู่ในสถานะที่ยังไม่เปิดใช้งาน การเรียกสถานะที่เก็บไว้ตามแบบรูป *Checkpoint* สามารถช่วยให้กลับมาทำงานได้เร็วขึ้น

ปัญหาที่อาจพบได้ในกรณีใช้ตัวสำรองแบบแอคทีฟ-แอคทีฟ คือมีการแจกจ่ายงานไปให้ทั้ง 2 ส่วนอยู่แล้วทำให้ไม่สามารถเปลี่ยนไปใช้ตัวสำรองได้ในทันทีเพราะยังคงมีงานที่ค้างอยู่ที่ตัวสำรอง ระบบต้องย้ายภาระงานจากตัวที่มีปัญหาไปยังตัวสำรองอย่างค่อยเป็นค่อยไป อีกปัญหาที่อาจพบคือ เมื่อตัวสำรองมาทำหน้าที่แทนแล้ว แต่ยังมีทรัพยากรที่หน่วยที่มีปัญหายังคงถือครองอยู่ จึงต้องมีหน่วยที่รับผิดชอบตามแบบรูป *Someone in Charge* มาจัดการกับปัญหาที่เกิดขึ้น

การประยุกต์ใช้แบบรูปนี้กับบีเพลจำเป็นต้องทำในระดับของเครื่องประมวลบีเพลและระบบพื้นฐาน เพื่อรองรับการเปลี่ยนการทำงานของกระบวนการบีเพลที่มีปัญหาไปยังกระบวนการสำรองที่เตรียมไว้ และสามารถจัดการถ่ายโอนสถานะการทำงานจากกระบวนการบีเพลเดิมไปยังตัวสำรองได้ด้วย

3.3.10 Checkpoint

เมื่อตรวจพบความผิดพลาดในระบบและเลือกทำการกู้ระบบโดยใช้วิธีย้อนกลับไปทำงานในสถานะก่อนหน้าตามแบบรูป *Rollback* หรือเริ่มต้นใหม่ตามแบบรูป *Restart* งานที่ระบบได้ดำเนินการมาแล้วอาจหายไปได้ระหว่างที่เกิดความผิดพลาด ทำให้เมื่อย้อนกลับมาในตำแหน่งก่อนหน้าระบบจำเป็นต้องเริ่มดำเนินงานใหม่ ซึ่งถ้ามีปริมาณมากด้วยแล้วย่อมส่งผลให้ต้องใช้เวลามากขึ้นไปอีก หากระบบมีการเก็บสถานะการทำงานเอาไว้ก่อนจะช่วยให้ไม่ต้องเสียเวลาในการทำงานใหม่ตั้งแต่ต้นได้ แบบรูป *Checkpoint* นี้จึงควรใช้ควบคู่ไปกับแบบรูป *Rollback* เพื่อให้สามารถเรียกสถานะการทำงานล่าสุดมาดำเนินการต่อได้

สิ่งที่ควรพิจารณาเมื่อเลือกใช้แบบรูปนี้แล้วคือ สถานะการทำงานที่ควรบันทึกไว้มีอะไรได้บ้าง เช่น ค่าของตัวแปร ตำแหน่งของการดำเนินงาน หรือปริมาณงานที่ต้องทำ ซึ่งพิจารณาต่อได้จากแบบรูป *What to Save* สิ่งที่จะพิจารณาต่อมาคือสถานที่จัดเก็บสถานะเหล่านี้ควรอยู่ที่ใดในกรณีที่ระบบรองรับตัวสำรอง ซึ่งพิจารณาต่อได้จากแบบรูป *Remote Storage* ลำดับสุดท้ายที่จะพิจารณาคือความถี่ในการจัดเก็บซึ่งพิจารณาต่อได้จากแบบรูป *Individuals Decide Timing*

การประยุกต์ใช้แบบรูปนี้กับบีเพลไม่สามารถจัดเก็บสถานะไว้ในกระแสงานได้โดยตรง ต้องพึ่งพาความสามารถของเครื่องประมวลบีเพลและระบบพื้นฐานให้รองรับการจัดเก็บสถานะของกระบวนการบีเพล และสามารถนำข้อมูลเหล่านั้นกลับมาดำเนินการใหม่ได้ในกรณีที่ต้องการกู้ระบบด้วยแบบรูป *Rollback*

3.3.11 What to Save

เมื่อระบบมีการประยุกต์ใช้แบบรูป *Checkpoint* เพื่อจัดเก็บสถานะของระบบไว้แล้ว สิ่งที่ต้องพิจารณาต่อมาคือควรจัดเก็บข้อมูลใดบ้าง คำว่าสถานะของระบบอาจประกอบไปด้วย ตัวแปรท้องถิ่น ตำแหน่งของการดำเนินงาน และสถานะของภาระงาน ในระบบที่ต้องใช้เวลานานหากสามารถเก็บข้อมูลเหล่านี้ไว้ได้ จะช่วยให้การกู้ระบบกลับมายังสถานะเดิมได้เร็วขึ้น ข้อมูลอีกประเภทหนึ่งที่ควรจัดเก็บไว้ด้วยคือ ข้อมูลส่วนรวม (Global information) ที่แต่ละหน่วยในระบบมีความจำเป็นต้องใช้ข้อมูลนี้ร่วมกัน

การประยุกต์ใช้กับบีเพลขึ้นอยู่กับการออกแบบกระบวนการบีเพลว่ามีความจำเป็นต้องจัดเก็บข้อมูลที่จำเป็นใดไว้บ้างขณะดำเนินงาน

3.3.12 Remote Storage

ในกรณีระบบที่มีตัวสำรองตามแบบรูป *Redundancy* เมื่อเกิดความผิดพลาดขึ้นกับระบบหลัก การเปลี่ยนไปทำงานในระบบสำรองโดยคืนค่าสถานะด้วยข้อมูลที่จัดเก็บไว้ตามแบบรูป *Checkpoint* จะช่วยให้ระบบกลับมาพร้อมใช้งานได้เร็วกว่าการเริ่มดำเนินงานใหม่หมด ถ้าข้อมูลที่จัดเก็บนั้นอยู่ในระบบหลักอาจทำให้เสียเวลาในการรอให้ระบบกลับมาดำเนินงานใหม่จึงจะสามารถเข้าถึงข้อมูลที่จัดเก็บไว้ได้ ดังนั้นจึงควรแยกการเก็บข้อมูลออกมาเป็นส่วนใหม่เป็นอิสระจากทั้งระบบหลักและระบบสำรองเพื่อช่วยให้การกู้ระบบสามารถทำได้เร็วขึ้น

การประยุกต์ใช้แบบรูปนี้กับบีเพลจำเป็นต้องพึ่งพาระบบพื้นฐานในการจัดเก็บข้อมูลที่เป็นอิสระแยกออกจากระบบหลักและระบบสำรอง และเครื่องประมวลบีเพลสามารถเข้าถึงข้อมูลเหล่านั้นเพื่อใช้ในการคืนค่าสถานะของกระบวนการบีเพลได้

3.3.13 Individuals Decide Timing

ระบบมีการประยุกต์ใช้แบบรูป *Checkpoint* สิ่งที่ต้องพิจารณานอกจากข้อมูลที่จะเก็บแล้วคือ เวลาใดที่ควรจัดเก็บข้อมูล แนวทางหนึ่งคือการจัดเก็บร่วมกัน ซึ่งเหมาะกับระบบที่แต่ละกระบวนการมีการติดต่อระหว่างกันน้อย เนื่องจากแต่ละกระบวนการที่มีความเกี่ยวข้องกัน เมื่อทำการจัดเก็บข้อมูลจำเป็นจะต้องทำไปพร้อมกันด้วยเพื่อให้ข้อมูลนั้นตรงกัน ถ้ามีกระบวนการที่เกี่ยวข้องกันมากการจัดเก็บจะกินเวลามากขึ้น

อีกแนวทางหนึ่งคือการจัดเก็บที่เป็นอิสระต่อกัน แต่ละกระบวนการทำการจัดเก็บตามช่วงเวลาของตัวเอง โดยจะมีเหตุการณ์ที่เป็นตัวกระตุ้นเพื่อบอกถึงช่วงเวลาที่เหมาะสม เช่น เมื่อ

กระบวนการได้รับข้อความที่บอกสถานะบางอย่างแล้ว แนวทางนี้เหมาะสมกับกระบวนการที่ต้องการความแม่นยำของเวลาเป็นสำคัญเนื่องจากข้อมูลที่จัดเก็บนั้นเกิดขึ้นตามเหตุการณ์ย่อยๆ ที่เป็นอิสระต่อกัน แต่แนวทางนี้จำเป็นจะต้องพิจารณาถึงความสอดคล้องกัน (Consistency) ของข้อมูลที่จัดเก็บในแต่ละกระบวนการด้วย

การนำมาประยุกต์ใช้กับบีเพลควรพิจารณาในระดับเครื่องประมวลบีเพลและระบบพื้นฐานว่าสามารถรองรับให้มีการจัดเก็บข้อมูลตามแต่ละกระบวนการบีเพลได้หรือไม่ แต่ละกระบวนการสามารถจัดเก็บสถานะแยกจากกันตามการทำงาน เช่น เมื่อได้รับคำร้องจากผู้ใช้ หรือ เมื่อทำการเรียกเว็บเซอร์วิช

3.3.14 Data Reset

ในกรณีที่เกิดความผิดพร่องกับข้อมูล แต่ไม่สามารถใช้วิธีการตามแบบรูป *Correcting Audit* มาแก้ไขให้ถูกต้องได้ และไม่สามารถกู้คืนกลับไปยังค่าก่อนหน้าได้เพราะไม่ได้ทำการจัดเก็บสถานะไว้ตามแบบรูป *Checkpoint* ส่งผลให้ต้องเลือกใช้วิธีการขั้นต่อไปคือการกลับไปใช้ค่าเริ่มต้นของข้อมูลนั้น วิธีการนี้ใกล้เคียงกับการใช้แบบรูป *Return to Reference Point* และแบบรูป *Restart* แต่กรณีนี้เป็นมุมมองของข้อมูลที่ใช้ และเมื่อมีการกำหนดค่าใหม่แล้ว ข้อมูลที่เกี่ยวข้องและสถานะของระบบอาจต้องเปลี่ยนตามไปด้วยเช่นกัน ในกรณีที่ไม่มีค่าเริ่มต้นของข้อมูลอาจใช้แบบรูป *Rollback* เพื่อย้อนกลับไปสถานะก่อนหน้าและทำการประมาณค่าจากข้อมูลอื่นๆ ที่เกี่ยวข้องแทน

ในการประยุกต์ใช้กับบีเพลในระดับกระแสงาน ถ้าข้อมูลที่ต้องเริ่มต้นใหม่นั้นเป็นเพียงตัวแปรที่เกิดขึ้นภายในกระบวนการบีเพลเท่านั้น จำเป็นต้องกำหนดค่าเริ่มต้นเก็บไว้ในกระแสงาน และนำมาใช้กำหนดค่าใหม่เมื่อมีการทำงานตามแบบรูป *Return to Reference Point* แต่ในกรณีทั่วไปที่ข้อมูลจัดเก็บไว้ในฐานข้อมูล หรือแหล่งข้อมูลอื่น การกำหนดค่าเริ่มตั้นใหม่จำเป็นต้องพึ่งพาระบบพื้นฐานที่รองรับการเก็บค่าเริ่มต้นให้กับข้อมูลเหล่านั้นและสามารถนำมาใช้ใหม่ได้

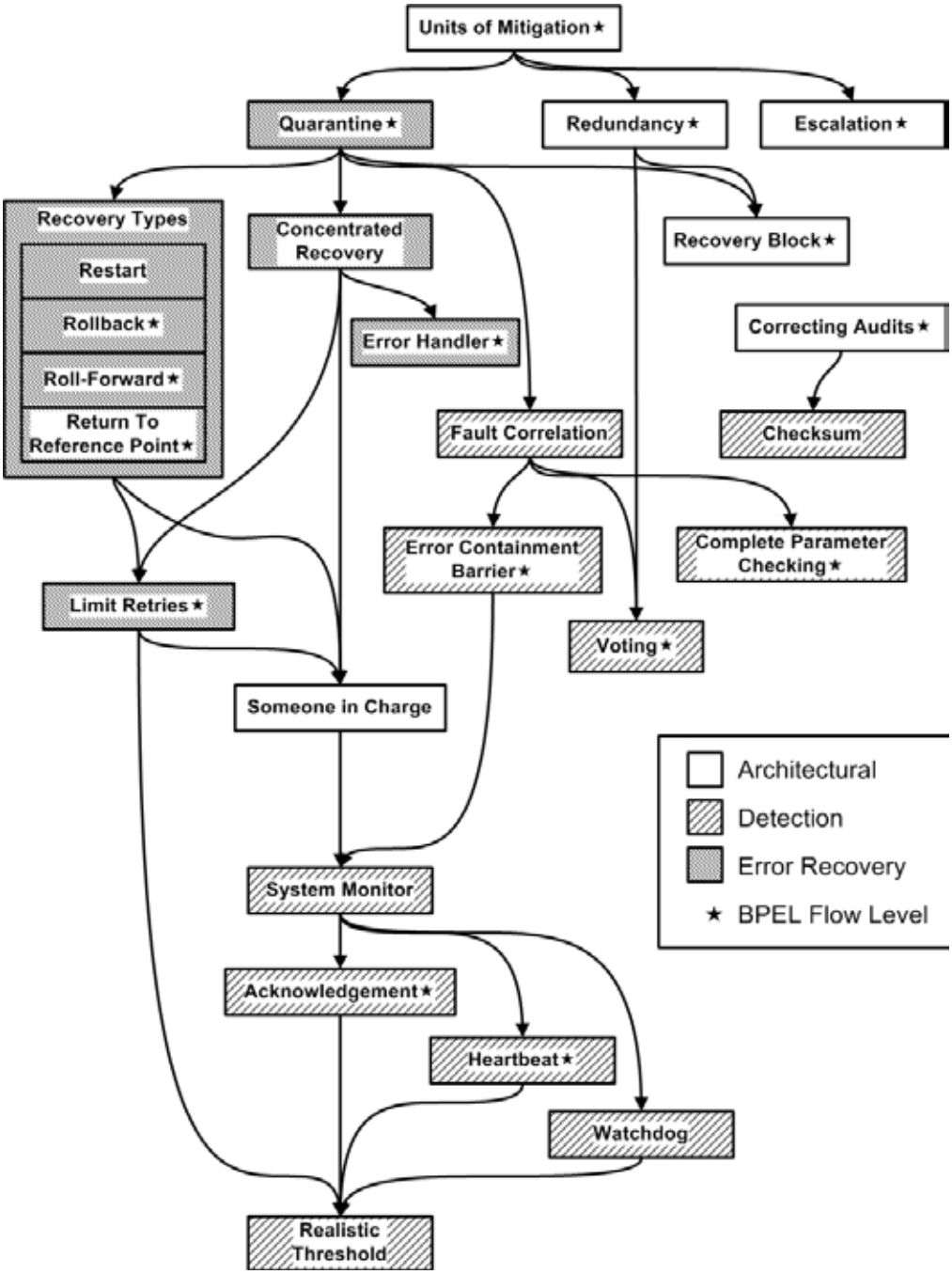
จากรายละเอียดของแบบรูปต่างๆ ในกลุ่มแบบรูปการกู้ระบบจากความผิดพลาดที่กล่าวมานั้นสามารถสรุปได้ดังตารางที่ 3.3 โดยแสดงถึงจุดประสงค์ของแบบรูป (Pattern Intent) [10] และการนำแบบรูปมาประยุกต์กับการออกแบบกระแสงานบีเพลให้ทนต่อความผิดพร่อง ว่าแบบรูปใดสามารถใช้ในระดับกระแสงานได้ หรือแบบรูปใดจำเป็นต้องพึ่งพาความสามารถของเครื่องประมวลบีเพลและระบบพื้นฐานที่รองรับอยู่

ตารางที่ 3.3 ระดับการประยุกต์ใช้กับบีเพลของกลุ่มแบบรูปการกู้ระบบจากความผิดพลาด

แบบรูป	จุดประสงค์ของแบบรูป	การประยุกต์ใช้กับบีเพล
Quarantine	แยกหน่วยของระบบที่มีปัญหาออกมาและจำกัดเอาไว้ เพื่อไม่ให้สร้างความเสียหายต่อส่วนอื่นของระบบ	ระดับกระแสงาน
Concentrated Recovery	ให้ความสำคัญกับการกู้ระบบจากความผิดพลาดมากที่สุดเท่าที่จะทำได้ เพื่อลดเวลาที่จะต้องใช้ในการกู้ระบบให้น้อยลง	ระดับเครื่องประมวลบีเพล
Error Handler	จัดหาวิธีที่ควบคุมได้เพื่อจัดการกับความผิดพลาด	ระดับกระแสงาน
Restart	กลับไปดำเนินงานโดยเริ่มโปรแกรมใหม่ตั้งแต่ต้น	ระดับเครื่องประมวลบีเพล
Rollback	กลับไปดำเนินงานโดยย้ายไปยังสถานะก่อนที่จะเกิดความผิดพลาดในเส้นทางการดำเนินงาน	ระดับกระแสงาน
Roll-Forward	กลับไปดำเนินงานโดยข้ามไปยังสถานะต่อไปที่ควรจะเป็นหากไม่เกิดความผิดพลาดขึ้น	ระดับกระแสงาน
Return to Reference Point	กลับไปดำเนินงานโดยย้อนกลับไปยังสถานะที่เจาะจงไว้ ซึ่งไม่จำเป็นต้องอยู่ในเส้นทางการดำเนินงานที่นำไปสู่ความผิดพลาด แต่ต้องเป็นจุดที่ทราบว่าปลอดภัย	ระดับกระแสงาน
Limit Retries	อย่ากลับไปทำงานยังจุดที่เกิดความผิดพลาดโดยไม่ได้ทำการเปลี่ยนแปลงใดๆ เพราะว่าความผิดพลาดนั้นอาจจะเกิดซ้ำอีก	ระดับกระแสงาน
Failover	กู้คืนระบบโดยเปลี่ยนไปใช้ตัวสำรอง	ระดับเครื่องประมวลบีเพล
Checkpoint	บันทึกสถานะของระบบเอาไว้เป็นระยะๆ เพราะการกู้ระบบไม่จำเป็นต้องเริ่มใหม่ตั้งแต่ต้น	ระดับเครื่องประมวลบีเพล
What to Save	การบันทึกสถานะของระบบ (Checkpoint) ควรบันทึกข้อมูลที่ต้องใช้ร่วมกันให้กับกระบวนการที่ใช้เวลานาน	ระดับการออกแบบ
Remote Storage	พิจารณาถึงตัวสำรองและปัจจัยอื่นๆ ในการตัดสินใจว่าจะบันทึกสถานะของระบบไว้ที่ใด	ระดับระบบพื้นฐานที่รองรับ
Individuals Decide Timing	ให้แต่ละกระบวนการตัดสินใจว่าเมื่อใดที่ควรจะใช้แบบรูป *Checkpoint* โดยอิงตามความต้องการของแต่ละกระบวนการ	ระดับเครื่องประมวลบีเพล
Data Reset	กู้คืนข้อมูลกลับไปยังค่าเริ่มต้นเมื่อพบว่ามันมีค่าไม่ถูกต้อง	ระดับระบบพื้นฐานที่รองรับ

3.4 แบบรูปที่ประยุกต์ใช้ในระดับกระแสงานบีเพล

เมื่อทำการวิเคราะห์ถึงความเหมาะสมในการนำแบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่องทั้ง 3 กลุ่มมาประยุกต์ใช้กับการออกแบบในระดับกระแสงานบีเพล สามารถนำแบบรูปที่เหลือทั้ง 16 แบบรูปมาจัดเป็นแผนที่ของภาษาแบบรูปที่ประยุกต์ได้ในระดับกระแสงานดังภาพที่ 3.6 โดยจะแสดงแบบรูปที่ใช้ได้ในระดับอื่นแต่มีความเกี่ยวข้องกันไว้ด้วย และสามารถสร้างเป็นแม่แบบโครงสร้างของกระแสงานบีเพลที่ประยุกต์ตามแบบรูปเหล่านี้ดังที่ได้แสดงไว้ในบทที่ 4



ภาพที่ 3.6 แผนที่ภาษาแบบรูปที่นำมาใช้ในระดับกระแสงานบีเพลได้

บทที่ 4

การประยุกต์ใช้แบบรูป

จากการวิเคราะห์ถึงความเหมาะสมในการนำแบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่องมาประยุกต์ใช้ในระดับการออกแบบกระแสงานบีเพลที่กล่าวมาแล้วในบทที่ 3 ในบทนี้เป็นการเสนอแม่แบบของโครงสร้างกระแสงานบีเพลที่ได้นำแบบรูปต่างๆ มาประยุกต์ใช้แล้ว เพื่อเป็นแนวทางในการออกแบบกระแสงานบีเพลต่อไป แม่แบบที่นำเสนอในงานวิจัยนี้ใช้โปรแกรม Netbeans เวอร์ชัน 6.7.1 ในชุด Glassfish ESB เป็นเครื่องมือพัฒนาช่วยในการออกแบบโครงสร้างของบีเพล และใช้โปรแกรม Eclipse ที่ติดตั้งส่วนเสริม Bpel Designer เพื่อให้ทำงานร่วมกับเครื่องประมวลบีเพล Apache ODE แล้วนำมาเป็นชุดพัฒนาอีกชุดหนึ่งที่ช่วยตรวจสอบว่าโครงสร้างที่ออกแบบสามารถใช้งานร่วมกันได้ในชุดพัฒนาที่ต่างกัน

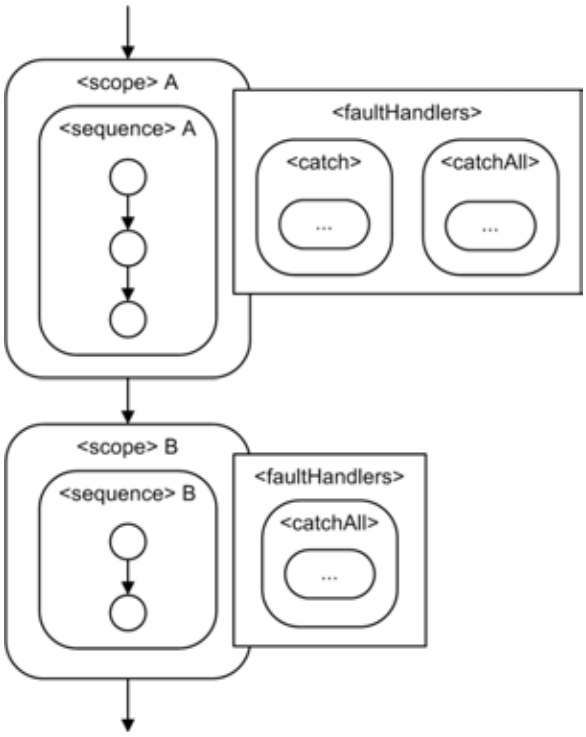
4.1 แบบรูป Units of Mitigation และ แบบรูป Error Handler

การนำแบบรูป *Units of Mitigation* และแบบรูป *Error Handler* มาประยุกต์ใช้ร่วมกันสามารถสร้างเป็นแม่แบบให้กับการออกแบบบีเพลได้ดังภาพที่ 4.1 โดยให้แต่ละส่วนของกระแสงานจัดกลุ่มด้วยแท็ก <scope> ซึ่งจะเป็นการแบ่งกระแสงานออกเป็นหน่วยย่อยตามแบบรูป *Units of Mitigation* และในแต่ละหน่วยย่อยจะต้องเพิ่มแท็ก <faultHandlers> เพื่อทำหน้าที่จัดการกับความผิดพร่องที่จะเกิดขึ้น สามารถแยกวิธีจัดการตามชื่อของความผิดพร่องด้วยแท็ก <catch> ซึ่งกำหนดได้หลายตัว แต่ถ้าเป็นความผิดพร่องอื่นที่ไม่สามารถระบุชื่อได้หรือความผิดพร่องใดๆ สามารถใช้แท็ก <catchAll> แทนได้

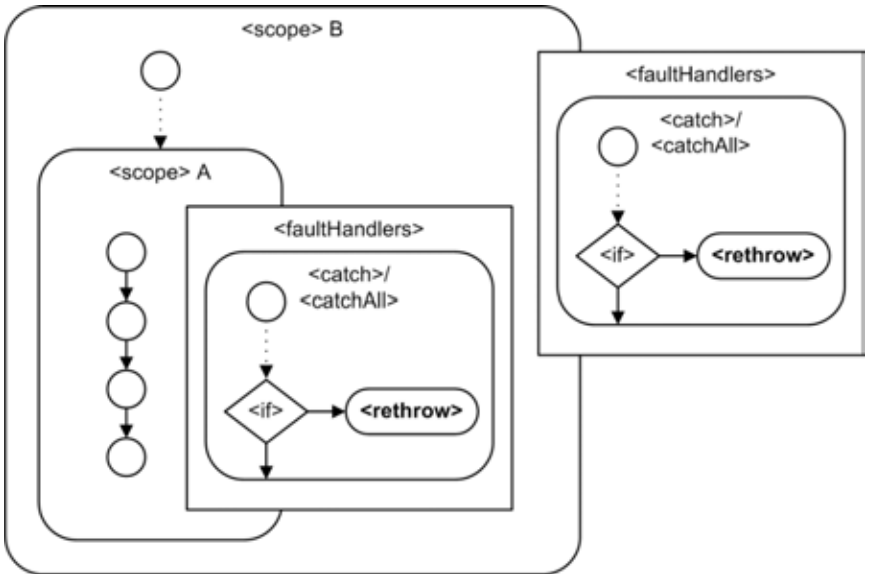
การใช้แท็ก <scope> และ <faultHandlers> จะเป็นพื้นฐานในการออกแบบให้กับแบบรูปอื่นๆ ต่อไป และช่วยป้องกันไม่ให้ความผิดพร่องที่เกิดขึ้นส่งผลกระจายไปยังหน่วยอื่นในกระแสงานได้ ทำให้มีคุณสมบัติเป็นไปตามแบบรูป *Error Containment Barrier* และแบบรูป *Quarantine* ด้วย

4.2 แบบรูป Escalation

การประยุกต์ใช้แบบรูป *Escalation* กับบีเพลสามารถแสดงเป็นแม่แบบของบีเพลได้ดังภาพที่ 4.2 โดยในกระแสงานจะมีการแบ่งระดับชั้นของ <scope> และติดตั้ง <faultHandlers> ไว้ด้วย เมื่อ <scope> ชั้นในไม่สามารถจัดการกับความผิดพร่องที่เกิดขึ้นได้สามารถใช้แท็ก <rethrow> เพื่อส่งสัญญาณว่าเกิดความผิดพร่องออกไปยัง <scope> ชั้นนอก และใช้วิธีการจัดการอื่นเป็นลำดับขั้นไปเรื่อยๆ



ภาพที่ 4.1 แม่แบบของบีเพลที่ใช้ <scope> และ <faultHandlers>

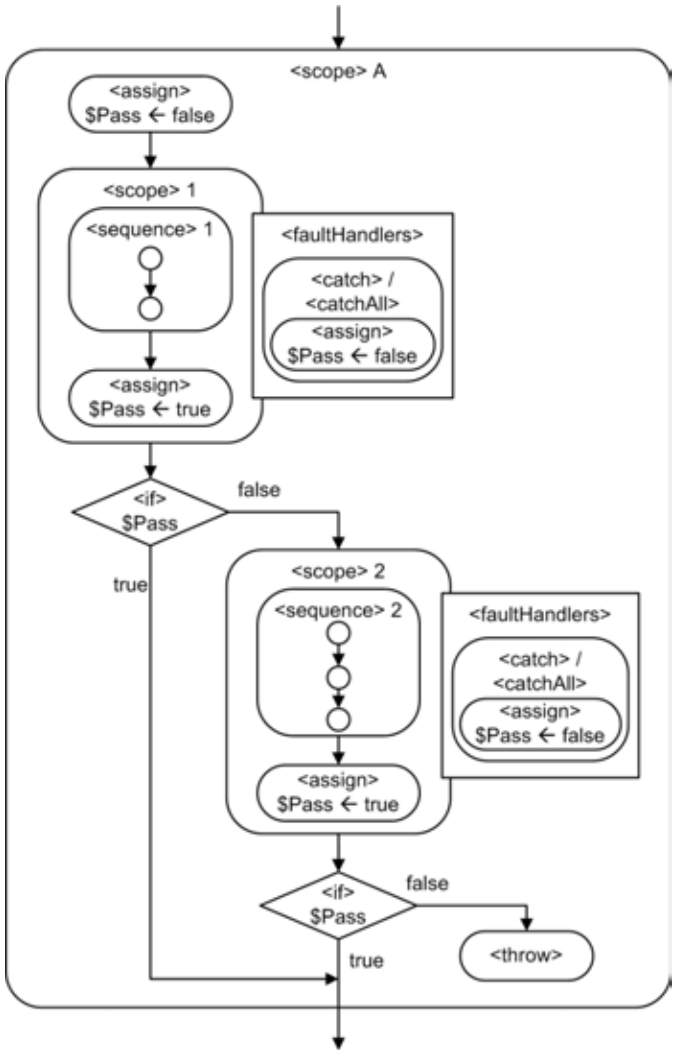


ภาพที่ 4.2 แม่แบบของบีเพลตามแบบรูป *Escalation*

4.3 แบบรูป Recovery Block และ แบบรูป Limit Retries

การประยุกต์ใช้แบบรูป *Recovery Block* กับบีเพลสามารถสร้างเป็นแม่แบบของบีเพลได้ดังภาพที่ 4.3 เริ่มจากบรรจุการทำงานทั้งส่วนปฐมภูมิและส่วนทุติยภูมิไว้ในแท็ก <scope> แยกกันเป็น “<scope> 1” และ “<scope> 2” ตามลำดับ และติดแท็ก <faultHandlers> ไว้ด้วยทั้ง 2 ส่วน

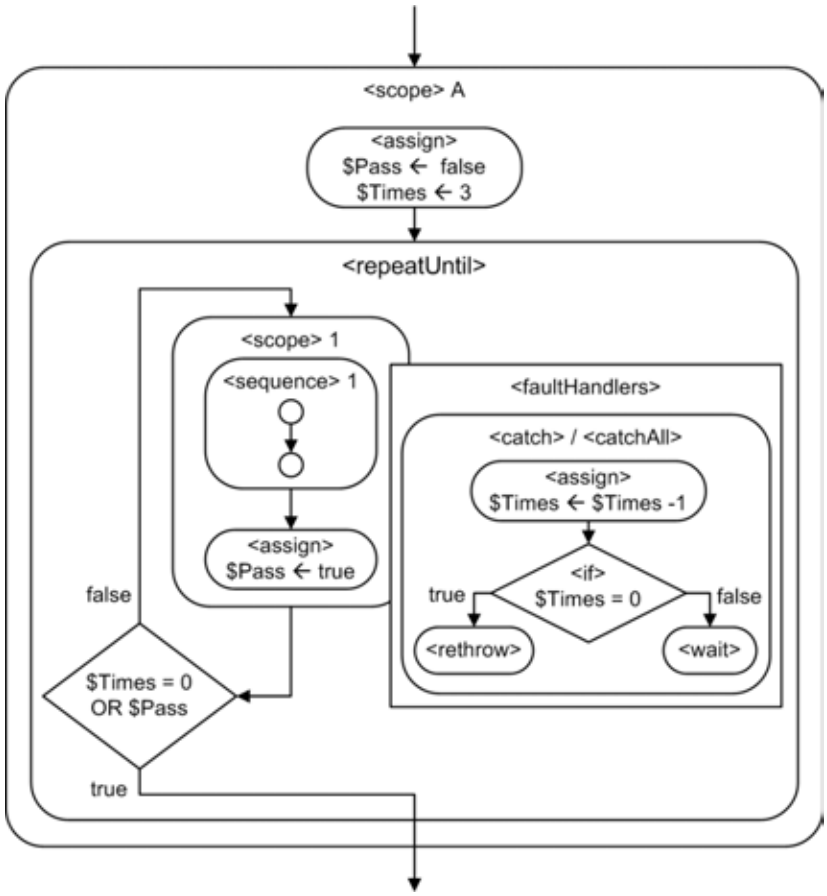
โดยในที่นี้จะกำหนดให้มีส่วนทุตยิภูมิเพียงขั้นเดียว ต่อมากำหนดตัวแปรชนิดบูลีน (boolean) ชื่อ “$Pass” เพื่อเป็นตัวชี้ว่าการทำงานในแต่ละขั้นเกิดความผิดพร่องขึ้นมาหรือไม่ กล่าวคือถ้าการทำงานในส่วนปฐมภูมิสามารถทำได้สำเร็จ จะกำหนดให้ $Pass มีค่าเป็นจริง แต่ถ้ามีความผิดพลาดเกิดขึ้นซึ่งถูกดักจับด้วย <faultHandlers> จะมีการกำหนดค่าให้เป็นเท็จแทน ส่งผลให้เข้าสู่การทำงานในส่วนทุติยภูมิต่อไป และเช่นเดียวกันหากการทำงานในส่วนทุติยภูมิสามารถทำได้สำเร็จก็จะกำหนดค่า $Pass เป็นจริงและสิ้นสุดการทำงานในส่วนนี้ได้ แต่หากเกิดความผิดพลาดขึ้น จะทำการแจ้งความผิดพร่องส่งออกไปยัง <scope> ชั้นนอกด้วยคำสั่ง <throw> และใช้วิธีการอื่นต่อไปตามแบบรูป *Escalation*



ภาพที่ 4.3 แม่แบบของบีเพลตามแบบรูป *Recovery Block*

แบบรูป *Recovery Block* สามารถกำหนดให้ส่วนทุติยภูมิเป็นการทำงานแบบเดียวกับในส่วนปฐมภูมิได้ และเมื่อประยุกต์ร่วมกับแบบรูป *Limit Retries* จะทำให้ได้แม่แบบของบีเพลในการทำงานซ้ำที่จำกัดจำนวนครั้งได้ดังภาพที่ 4.4 แม่แบบนี้จะใช้ในกรณีที่เว็บเซอร์วิซที่ใช้เกิดปัญหาเพียง

ชั่วขณะ เมื่อเวลาผ่านไประยะหนึ่งแล้วกลับมาใช้งานได้ปกติ โดยกำหนดตัวแปรชนิดจำนวนเต็ม (integer) ชื่อว่า “$Times” เพื่อใช้เป็นตัวกำหนดรอบของการทำงาน แล้วนำโครงสร้าง <repeatUntil> มาครอบส่วนของ <scope> ที่ต้องการให้วนซ้ำได้ โดยกำหนดเงื่อนไขไว้ว่าจะวนซ้ำไปจนกว่าการทำงานภายใน <scope> นั้นสำเร็จ กล่าวคือถ้าทำผ่านจะให้ค่าของตัวแปร $Pass เป็นจริง แต่ถ้าเกิดความผิดพร่องขึ้นใน <scope> จะถูกดักจับด้วยแท็ก <faultHandlers> และทำการลดค่าของ $Times ลง 1 ครั้ง หากค่าของ $Time ยังไม่หมด จะทำการรอตามเวลาที่กำหนดไว้ และกลับมาทำงานซ้ำใหม่ แต่ถ้าเป็นรอบสุดท้ายแล้วยังไม่สามารถทำงานให้สำเร็จได้ซึ่ง $Time มีค่าเป็นศูนย์ จะใช้แท็ก <rethrow> เพื่อแจ้งออกไปยัง <scope> ชั้นนอกว่าเกิดความผิดพร่องขึ้นแทน

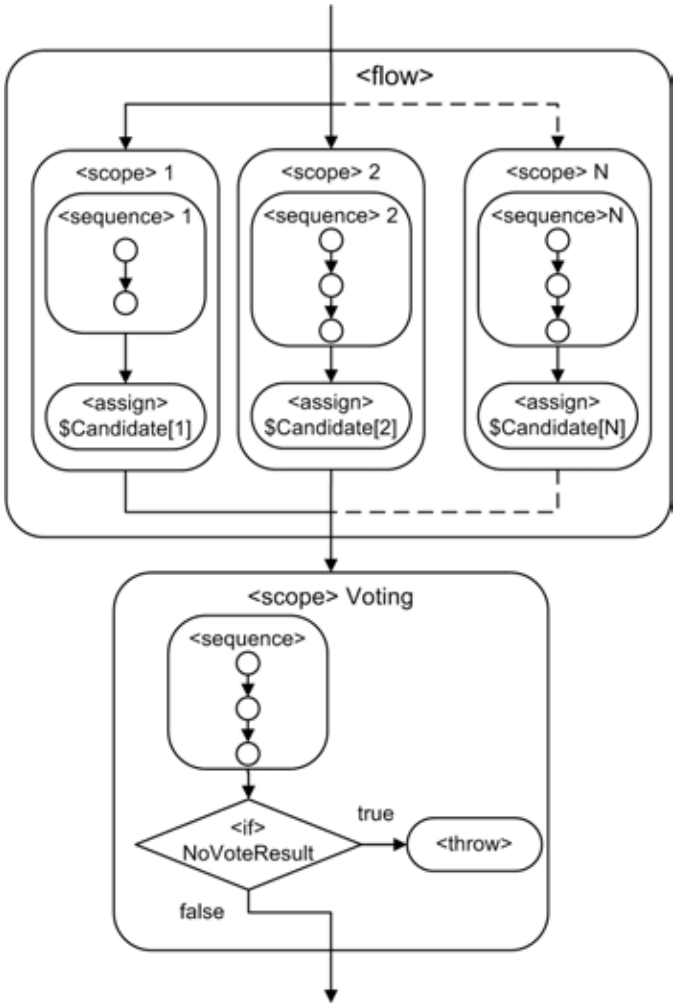


ภาพที่ 4.4 แม่แบบของบีเพลในการทำงานซ้ำ

4.4 แบบรูป Voting

การนำแบบรูป *Voting* มาประยุกต์ใช้กับบีเพลสามารถแสดงเป็นแม่แบบของบีเพลได้ดังภาพที่ 4.5 โดยการทำงานหรือการเรียกใช้เว็บเซอร์วิซที่ให้ผลลัพธ์แทนกันได้จะถูกเรียกใช้งานไปพร้อมๆ กันภายในโครงสร้างของแท็ก <flow> และเมื่อได้รับผลลัพธ์ครบทุกส่วนแล้ว จะส่งผลลัพธ์ที่ได้ต่อไปยังส่วนหาผลโหวต ในขั้นตอนการหาผลโหวตสามารถเพิ่มการทำงานลงไปในกระแสงานเดียวกันหรือ

สร้างเป็นกระบวนการใหม่และทำการเรียกใช้เหมือนการเรียกเว็บเซอร์วิซอื่นๆ ซึ่งจะช่วยให้การแก้ไขขั้นตอนทำได้เป็นอิสระต่อกัน ในกรณีที่หาผลโหวตไม่สำเร็จจะต้องแจ้งออกไปยัง <scope> ชั้นนอกว่าเกิดความผิดพร่องขึ้นด้วยแท็ก <throw>

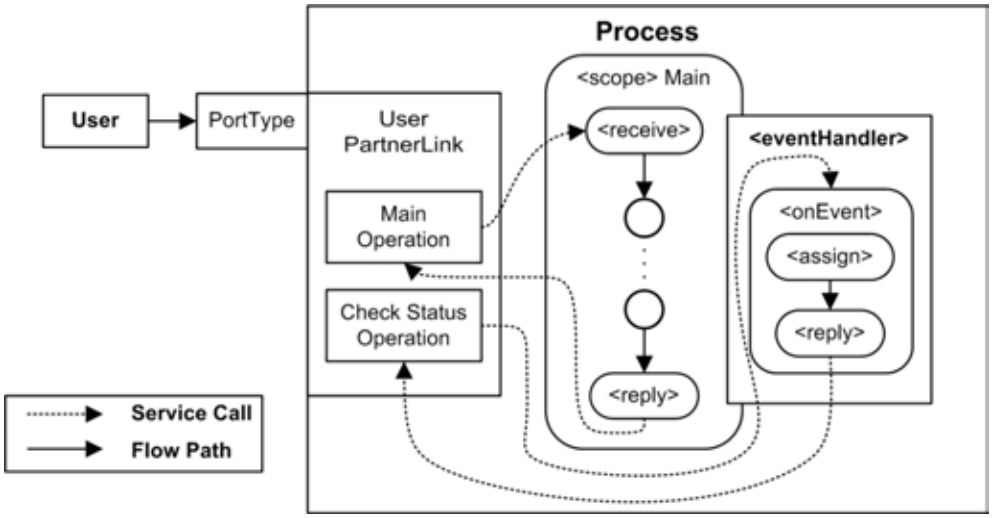


ภาพที่ 4.5 แม่แบบของบีเพลตามแบบรูป *Voting*

4.5 แบบรูป Acknowledgement

ในกระบวนการบีเพลที่ต้องใช้เวลานานควรมีช่องทางในการสอบถามสถานะการทำงานเพื่อให้ผู้ใช้งานสามารถทราบได้ว่ากระบวนการยังคงทำงานอยู่ โดยในเบื้องต้นควรมีการตอบกลับสถานะตามแบบรูป *Acknowledgement* ทุกครั้งที่ผู้ใช้มีการเรียกใช้งาน ฝั่งผู้ถูกเรียกควรเพิ่มโอเปอเรชัน (Operation) ไว้ในวิสเดิลเพื่อให้ผู้ใช้เรียกสอบถามสถานะ โดยแยกออกจากโอเปอเรชันที่ใช้ติดต่อกันตามปกติ กระแสงานบีเพลที่เป็นฝั่งผู้ถูกเรียกสามารถใช้แม่แบบดังภาพที่ 4.6 เป็นแนวทางในการประยุกต์ใช้ได้ โดยหลังจากเพิ่มโอเปอเรชันไว้ในวิสเดิลแล้ว ขั้นต่อมาคือเพิ่มแท็ก <eventHandler> ติดเข้าไปกับแท็ก <scope> ที่ต้องการ กระบวนการทำงานที่อยู่ใน

<eventHandler> นี้จะทำขนานไปกับการทำงานของ <scope> ที่ติดอยู่ และภายในจะใช้แท็ก <onEvent> มารอรับคำร้องจากผู้ใช้ซึ่งมีการทำงานเช่นเดียวกับแท็ก <receive> เมื่อได้รับคำร้องแล้วจึงทำการตอบกลับค่าสถานะกลับไปด้วยแท็ก <reply>



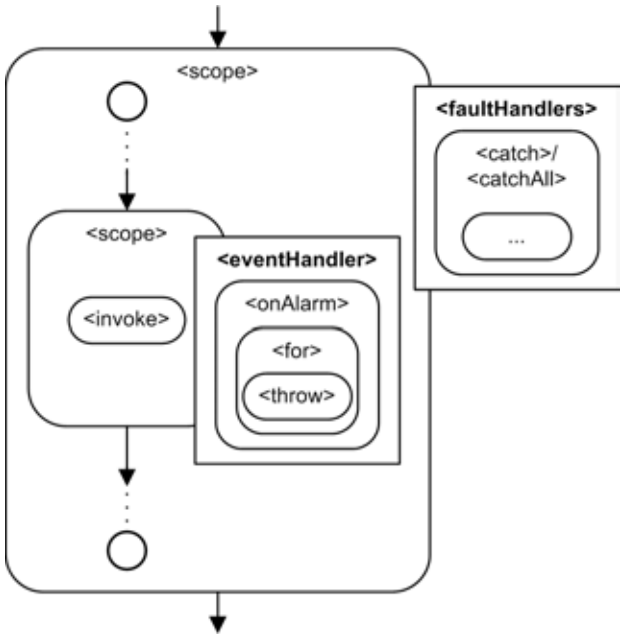
ภาพที่ 4.6 แม่แบบของบีเพลฝั่งผู้รับคำร้องตามแบบรูป *Acknowledgement*

เนื่องจากเป็นการเรียกกระแสงานเดิมด้วยโอเปอเรชันที่เพิ่มเข้ามาใหม่จึงต้องกำหนด <correlationSet> ไว้ในไฟล์บีเพลเพื่อช่วยในการอ้างอิงว่าต้องการติดต่อกับอินแสตนซ์ใดของกระแสงานบีเพล โดยค่าที่ใช้จะมาจากแท็ก <property> ที่กำหนดไว้ในไฟล์วิสเดิล ตัวอย่างของค่าอ้างอิง เช่น หมายเลขอินแสตนซ์ หมายเลขคำสั่งซื้อ หมายเลขผู้ใช้ หรือข้อความอื่นที่สามารถใช้อ้างอิงได้ และทำให้แต่ละอินแสตนซ์เป็นอิสระต่อกัน นอกจากนี้ในวิสเดิลต้องกำหนด <propertyAlias> เพื่อระบุตำแหน่งของค่าอ้างอิงในแต่ละข้อความที่ใช้รับส่งกัน

กิจกรรมบีเพลที่เกี่ยวข้องกับการรับส่งข้อความได้แก่แท็ก <receive> <reply> <invoke> <onEvent> (ในแท็ก <eventHandler>) และ <onMessage> (ในแท็ก <pick>) สามารถกำหนดค่าอ้างอิงไว้ได้ จากแม่แบบในภาพที่ 4.6 มีการรับข้อความได้ 2 ทางคือแท็ก <receive> และ <onEvent> ทั้ง 2 แท็กนี้จึงต้องกำหนดค่า <correlationSet> เดียวกันไว้เพื่ออ้างอิง และให้กำหนดค่าแอททริบิวต์ชื่อ "initiate" มีค่าเป็น "yes" ไว้ที่แท็ก <receive> เพื่อบอกว่าค่าอ้างอิงเริ่มใช้ที่กิจกรรมบีเพลนี้

ฝั่งผู้เรียกสามารถกำหนดช่วงเวลาที่สามารถรอการตอบกลับมาได้ด้วยแท็ก <onAlarm> ซึ่งเป็นส่วนหนึ่งของแท็ก <eventHandler> หรือแท็ก <pick> โดยจะทำการนับเวลาตามที่กำหนดไว้ในแท็ก <for> เมื่อหมดเวลาแล้วจึงทำการแจ้งออกไปด้วยแท็ก <throw> เพื่อให้ <faultHandlers> ดักสัญญาณนี้และบังคับให้หยุดการทำงานของ <scope> ที่ทำการเรียกใช้เซอร์วิซนั้น และให้

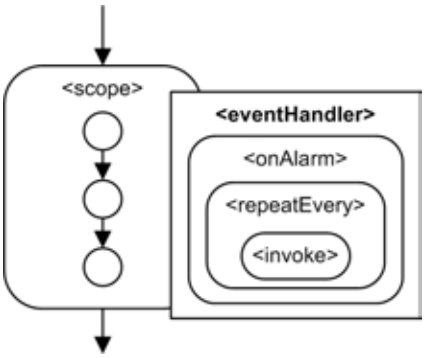
ดำเนินการขั้นตอนต่อไปใน <scope> ชั้นนอกแทน สามารถแสดงเป็นแม่แบบของบีเพลได้ดังภาพที่ 4.7



ภาพที่ 4.7 แม่แบบของบีเพลฝั่งผู้ส่งคำร้องตามแบบรูป *Acknowledgement*

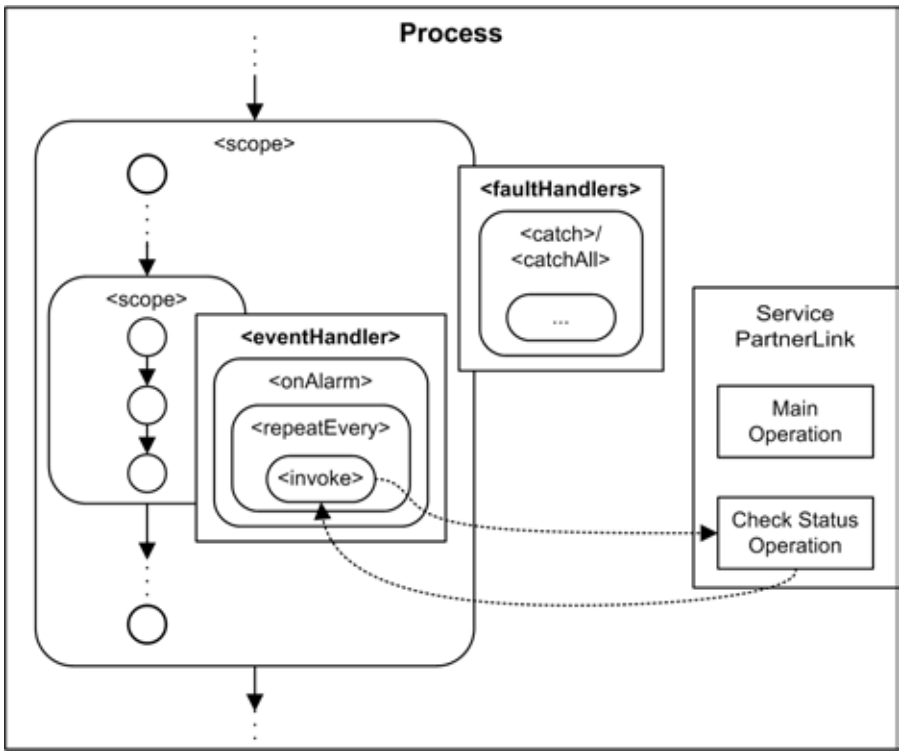
4.6 แบบรูป Heartbeat

แบบรูป *Heartbeat* สามารถนำมาประยุกต์ใช้กับบีเพลเพื่อช่วยให้การตรวจหาความผิดพลาดทำได้ดีขึ้น กระแสงานบีเพลฝั่งที่ถูกเฝ้าดูสามารถส่งสัญญาณเพื่อรายงานสถานะของตนออกไปเป็นระยะๆ โดยใช้แท็ก <eventHandler> เพิ่มเติมเข้าไปให้กับ <scope> ที่ต้องการ และกำหนดเวลาที่จะเตือนให้ส่งสถานะด้วยแท็ก <onAlarm> ซึ่งสามารถกำหนดให้วนซ้ำการเตือนเป็นรอบๆ ได้ โดยกำหนดเวลาของแต่ละรอบลงในแท็ก <repeatEvery> และใช้คำสั่ง <invoke>เพื่อส่งข้อความไปหาหน่วยที่เฝ้าดูหรือเรียกเซอร์วิซที่เกี่ยวข้องกับการรายงานสถานะต่อไป การใช้งานนี้แสดงเป็นแม่แบบของบีเพลไว้ดังภาพที่ 4.8



ภาพที่ 4.8 แม่แบบของบีเพลของฝั่งที่ถูกเฝ้าดูตามแบบรูป *Heartbeat*

ในทางกลับกันฝั่งที่ต้องการทราบสถานะสามารถใช้แนวทางนี้ได้เช่นกัน โดยทำการเรียกไปยังกระแสงานอื่นที่รองรับการสอบถามสถานะตามแบบรูป *Acknowledgement* และใช้การวนซ้ำด้วย <eventHandler> และ <onAlarm> และถ้าไม่มีคำตอบส่งกลับมา ระบบของเครื่องประมวลบีเพลจะแจ้งว่ามีความผิดพร่องเกิดขึ้น จึงต้องมีการดักจับความผิดพร่องด้วย <faultHandlers> ใน <scope> ชั้นนอก และดำเนินการในการจัดการกับความผิดพร่องนี้ต่อไป การใช้งานนี้แสดงเป็นแม่แบบของบีเพลไว้ดังภาพที่ 4.9

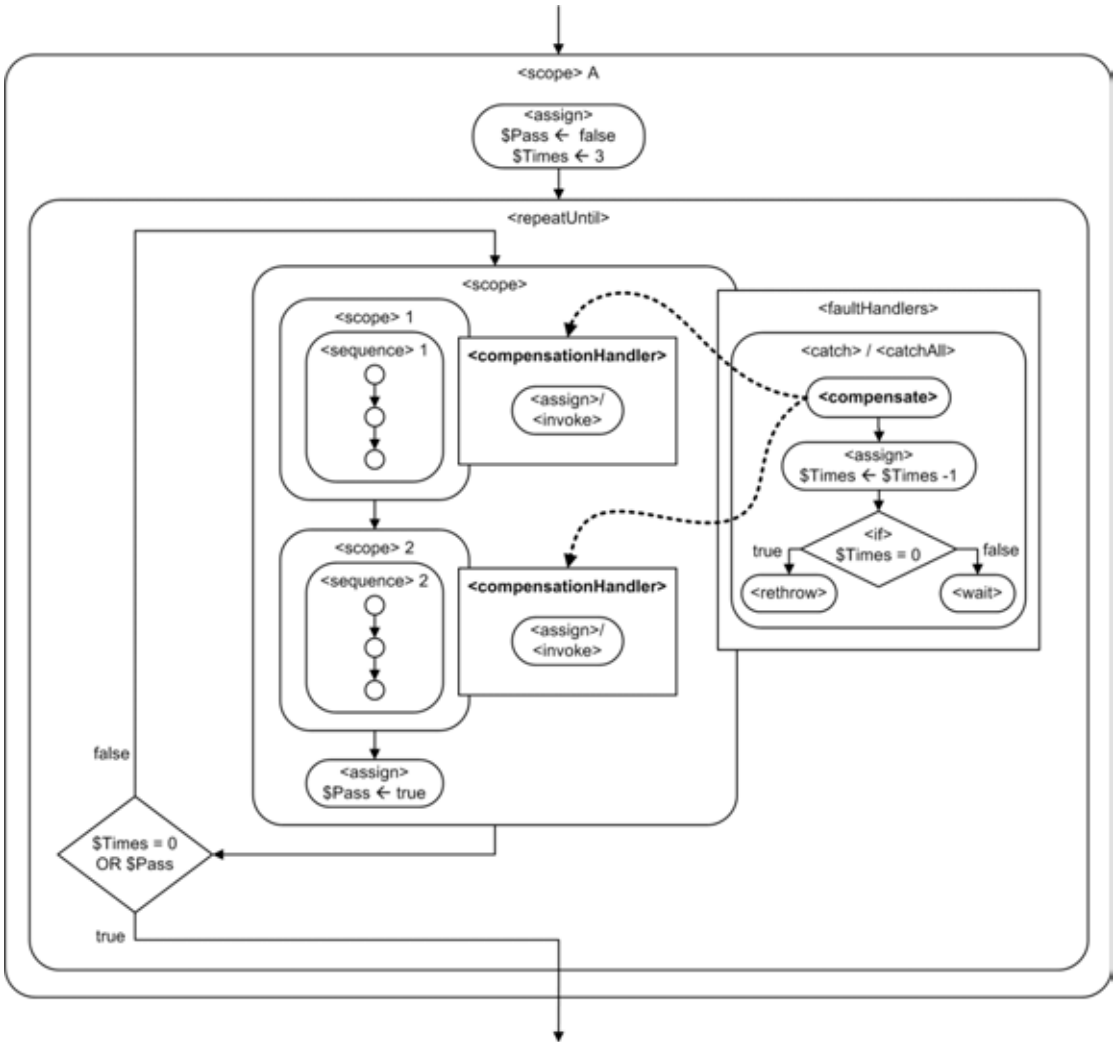


ภาพที่ 4.9 แม่แบบของบีเพลของฝั่งที่เฝ้าดูตามแบบรูป *Heartbeat*

4.7 แบบรูป Rollback

การประยุกต์ใช้แบบรูป *Rollback* กับบีเพลสามารถทำได้ในระดับกระแสงานบีเพล แต่มีข้อจำกัดที่ไม่สามารถย้อนกลับการทำงานอย่างพลวัตได้ เนื่องจากไม่มีการบันทึก Checkpoint เก็บไว้ในกระแสงานอย่างอัตโนมัติ ผู้ออกแบบจำเป็นต้องกำหนดตำแหน่งในการย้อนกลับไว้ก่อนโดยอาศัยโครงสร้างของ <scope> เพื่อแบ่งการทำงานตามกรอบของแบบรูป *Units of Mitigation* และสามารถกำหนดส่วนของกระแสงานที่ต้องการให้สามารถย้อนกลับไปยังจุดก่อนหน้าได้ โดยใช้โครงสร้างในการวนซ้ำเช่น <while> หรือ <repeatUntil> ในการย้อนการทำงาน หรือปรับใช้แม่แบบของบีเพลตามแบบรูป *Recovery Block* ในการทำงานซ้ำได้

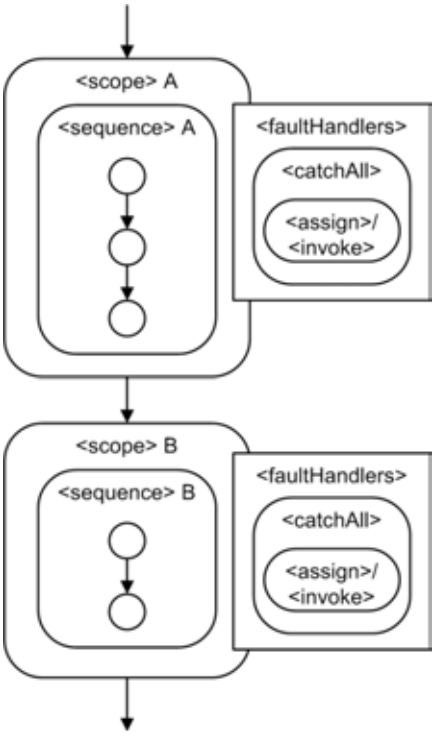
ก่อนที่จะย้อนกลับไปทำงานในตำแหน่งก่อนหน้า สามารถใช้โครงสร้างของแท็ก <compensationHandler> เพื่อชดเชยการทำงานที่ทำสำเร็จไปแล้ว เช่น กำหนดค่าให้กับข้อมูลใหม่ หรือสั่งให้ยกเลิกการทำงานนั้น โดยแท็ก <compensationHandler> จะติดตั้งไว้กับ <scope> และจะเริ่มการทำงานได้ก็ต่อเมื่อ <scope> ที่ติดอยู่นั้นเสร็จสิ้นการทำงานไปแล้ว และมีการใช้คำสั่ง <compensate> จากการทำงานชั้นนอกลงมา ซึ่งคำสั่ง <compensate> นี้สามารถใช้ได้กับ <catch> และ <catchAll> ภายใน <faultHandlers> หรือ <compensationHandler> หรือ <terminationHandler> เท่านั้น ตัวอย่างของการใช้ <compensationHandler> แสดงไว้ดังภาพที่ 4.10 โดยนำแม่แบบของบีเพลตามแบบรูป *Recovery Block* แบบเรียกซ้ำมาใช้ โดย <scope> หลักจะมีการทำงานย่อยอยู่ภายใน และแต่ละ <scope> ย่อยจะมี <compensationHandler> ติดอยู่ เมื่อเกิดความผิดพร่องขึ้นภายใน จะเข้าสู่การทำงานใน <faultHandlers> ที่อยู่ใน <scope> หลัก และใช้คำสั่ง <compensate> เพื่อสั่งให้มีการชดเชยให้กับ <scope> ที่อยู่ภายใน โดยจะเรียงตามลำดับของ <scope> ที่ทำงานเสร็จสิ้นแล้วย้อนกลับไปเรื่อยๆ



ภาพที่ 4.10 ตัวอย่างการใช้ <compensationHandler> กับแม่แบบของบีเพลในการทำงานซ้ำ

4.8 แบบรูป Roll-Forward

การประยุกต์ใช้แบบรูป *Roll-Forward* สามารถแสดงเป็นแม่แบบของบีเพลได้ดังภาพที่ 4.11 ซึ่งใช้ในกรณีที่กระแสงานสามารถละเว้นส่วนของ <scope> ที่เกิดความผิดพร่องได้ และข้ามไปยังการดำเนินงานที่ <scope> ต่อไป โดยเมื่อดักจับความผิดพร่องได้ด้วย <faultHandlers> แล้วควรมีการแจ้งว่าเกิดความผิดพร่อง เช่น บันทึกว่ามีความผิดพร่องเกิดขึ้นด้วย <assign> หรือรายงานไปยังส่วนที่มีหน้าที่เฝ้าดูด้วย <invoke> เป็นต้น



ภาพที่ 4.11 แม่แบบของบีเพลตามแบบรูป *Roll-Forward*

4.9 แบบรูป Return to Reference Point

การประยุกต์ใช้แบบรูป *Return to Reference Point* ในกระแสงานบีเพลเป็นการระบุว่าตำแหน่งใดในกระแสงานที่สามารถกลับมาดำเนินงานต่อได้ ในกรณีที่เกิดความผิดพร่องในส่วนสนับสนุนของแต่ละ <scope> ส่วนสนับสนุนในที่นี้คือ FCT-Handler (ประกอบไปด้วย <faultHandlers> <compensationHandler> และ <terminationHandler>) และส่วนสนับสนุนอีกประเภทคือ <eventHandler> ตามข้อกำหนดของดับเบิลยูเอส-บีเพล 2.0 หลังจากจบการทำงานในส่วนสนับสนุนแล้ว ตำแหน่งต่อไปที่จะดำเนินงานต่อจะเป็นลำดับที่ต่อจาก <scope> ที่ติดตั้งส่วนสนับสนุนนั้น

ในกลุ่ม FCT-Handler เมื่อเริ่มทำงานแสดงว่า <scope> ที่ติดตั้งนั้นหยุดการทำงานไปแล้ว (ทั้งกรณีที่ดำเนินงานจนจบหรือถูกบังคับให้หยุด) ทำให้เมื่อจบการทำงานใน FCT-Handler แล้วจะไม่สามารถกลับไปดำเนินงานใน <scope> เดิมได้ ต้องไปดำเนินงานในตำแหน่งที่ต่อจาก <scope> นั้น ถ้าเป็น <faultHandler> หรือ <terminationHandler> ตำแหน่งต่อไปจะอยู่ต่อจาก <scope> เดิม แต่ถ้าเป็น <compensationHandler> ตำแหน่งต่อไปจะนับตามลำดับของการชดเชย (Compensation Order) ซึ่งเรียงตามลำดับของ <scope> ย่อยที่ทำงานเสร็จสิ้นแล้ว และเริ่มจาก <scope> ที่ทำงานเสร็จหลังสุด

ในกรณีของ <eventHandler> จะสามารถดำเนินงานไปได้เรื่อยๆ ตามอายุของ <scope> ที่ติดตั้งอยู่ หากเหตุการณ์ที่สนใจสามารถเกิดซ้ำได้ ได้แก่ การรอรับข้อความจากภายนอกด้วย <onEvent> หรือ การใช้ <repeatEvery> ในแท็ก <onAlarm> เพื่อให้เหตุการณ์เกิดซ้ำตามรอบเวลาได้ เมื่อจบแต่ละเหตุการณ์แล้วตำแหน่งต่อไปจะยังอยู่ใน <eventHandler> เพื่อรอรับเหตุการณ์ใหม่ แต่ถ้าไม่มีเหตุการณ์เกิดซ้ำได้อีกแล้วหรือ <scope> ที่ติดตั้งหยุดการทำงาน จะถือว่าต้องหยุดการทำงานในส่วน <eventHandler> เช่นกัน และรอไปดำเนินงานต่อในตำแหน่งที่ต่อจาก <scope> ที่ติดตั้งนั้นต่อไป

ส่วนสนับสนุนสามารถเกิดความผิดพร่องได้เช่นกัน ควรกำหนดให้มีการดักจับความผิดพร่องด้วย <faultHandlers> เพิ่มเติมเข้าไปใน <scope> ที่อยู่ด้านในของส่วนสนับสนุน เพื่อช่วยให้สามารถจัดการกับความผิดพร่องที่เกิดขึ้นภายในส่วนสนับสนุนเหล่านี้และสามารถดำเนินงานจนเสร็จสิ้นได้ เมื่อจบการดำเนินงานในส่วนสนับสนุนแล้ว หากต้องการกลับไปดำเนินงานใน <scope> เดิมอีกครั้งจำเป็นต้องใช้โครงสร้างในการวนซ้ำได้แก่ <while> และ <repeatUntil> หรือนำแม่แบบของบีเพลในการทำงานซ้ำมาประยุกต์ใช้ร่วมกัน

4.10 แบบรูป Correcting Audits และ แบบรูป Complete Parameter Checking

การประยุกต์ใช้แบบรูป *Correcting Audits* และ แบบรูป *Complete Parameter Checking* สามารถใช้คำสั่ง <validate> เพื่อช่วยในการตรวจสอบข้อมูลของตัวแปรต่างๆ ในกระแสงานบีเพลได้ แต่เครื่องประมวลบีเพลและเครื่องมือพัฒนาที่ใช้ในงานวิจัยนี้ทั้ง 2 ชุดต่างไม่รองรับการใช้งานคำสั่ง <validate> ดังนั้นในการออกแบบกระแสงานบีเพลจำเป็นต้องมีกระแสงานย่อยหรือเว็บเซอร์วิซที่สามารถตรวจสอบโครงสร้างและความถูกต้องของข้อมูลได้แทน

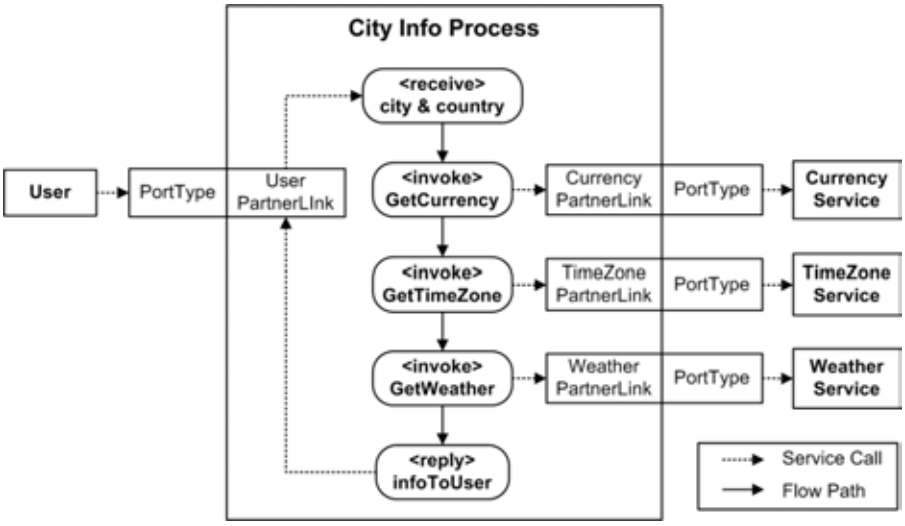
บทที่ 5

การทดลองกับกระแสงานบีเพล

ในบทนี้เป็นการทดลองนำแบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่องมาประยุกต์ใช้กับการออกแบบกระแสงานบีเพลตามตัวอย่างที่กำหนดไว้และทำการทดสอบถึงผลกระทบที่เกิดขึ้นทั้งความเชื่อถือได้และเวลาที่ใช้ว่าเปลี่ยนแปลงไปเช่นใด

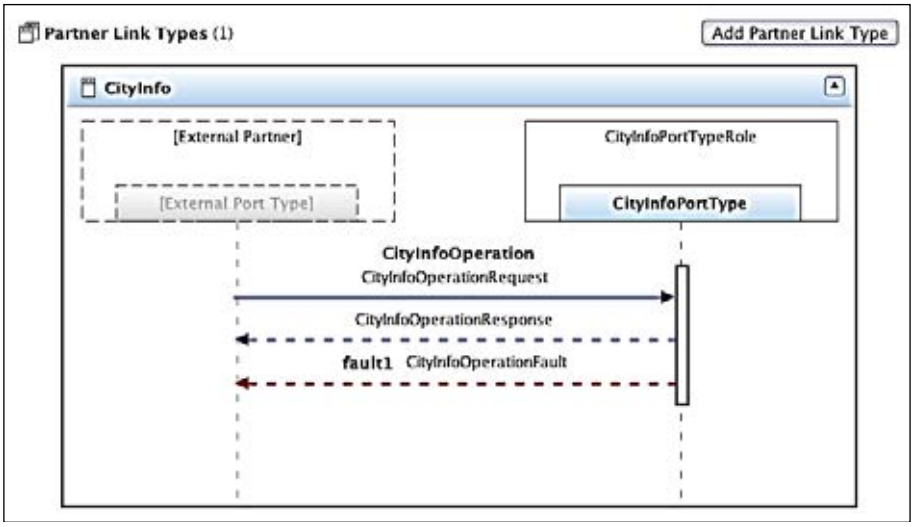
5.1 การทดลอง

ในการทดลองจะใช้ตัวอย่างของกระแสงานบีเพลที่ทำหน้าที่แสดงข้อมูลของเมืองตามที่ผู้ใช้ร้องขอ โดยรับข้อมูลขาเข้าซึ่งประกอบไปด้วยชื่อเมือง ชื่อรัฐ (ถ้ามี) และชื่อประเทศ จากนั้นจะทำการเรียกใช้เว็บเซอร์วิซที่ให้บริการข้อมูลใน 3 ส่วน ได้แก่ ข้อมูลสกุลเงิน ข้อมูลเขตเวลา และข้อมูลสภาพอากาศ เมื่อได้ผลลัพธ์ทั้งหมดแล้วจะส่งข้อมูลที่ได้กลับไปยังผู้ใช้ในขั้นตอนสุดท้าย ลักษณะของกระแสงานบีเพลตัวอย่างแสดงไว้ดังภาพที่ 5.1

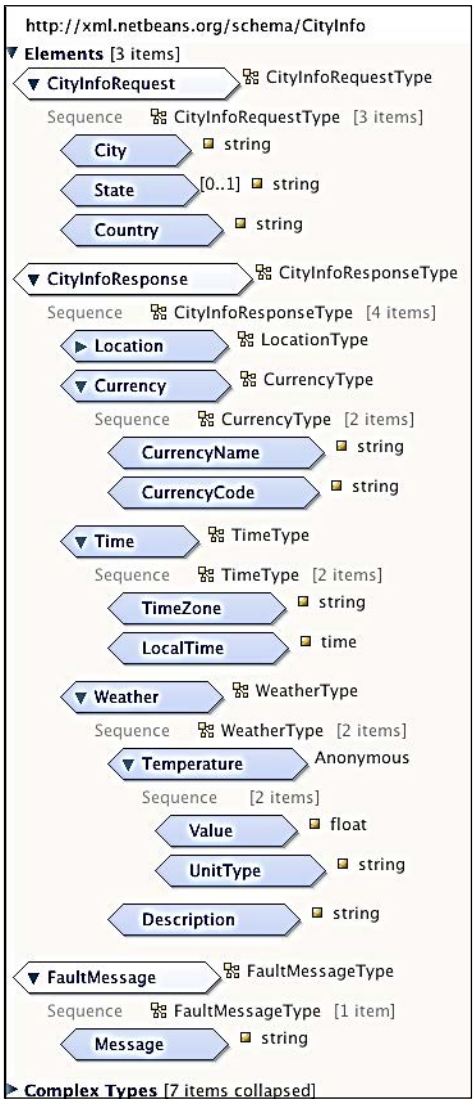


ภาพที่ 5.1 กระแสงานบีเพล CityInfo

ในการออกแบบกระบวนการบีเพลที่ใช้ในงานวิจัยนี้ใช้โปรแกรม Netbeans เวอร์ชัน 6.7.1 ซึ่งอยู่ในชุดของ Glassfish ESB เป็นเครื่องมือในการพัฒนา เริ่มต้นจากการกำหนดรายละเอียดของเซอร์วิซไว้ที่วิสเดิลซึ่งจะระบุโอเปอเรชันที่ใช้ได้ ข้อความที่เกี่ยวข้อง และที่อยู่ปลายทางของเซอร์วิซนั้น นอกจากนี้ต้องกำหนดพาร์ตเนอร์ลิงก์ไทป์เพื่อกำหนดบทบาทระหว่างบีเพลกับผู้อื่น จากตัวอย่างกระแสงาน CityInfo จะกำหนดพาร์ตเนอร์ลิงก์ไทป์มีโครงสร้างตามโปรแกรม Netbeans ดังภาพที่ 5.2 โดยกำหนดให้มีเพียงพอร์ตไทป์และโอเปอเรชันเดียวที่จะรับข้อมูลมา และตอบกลับไปได้ 2 อย่าง คือ ผลลัพธ์ที่ค้นหาได้หรือข้อความที่แจ้งว่าเกิดความผิดพร่อง



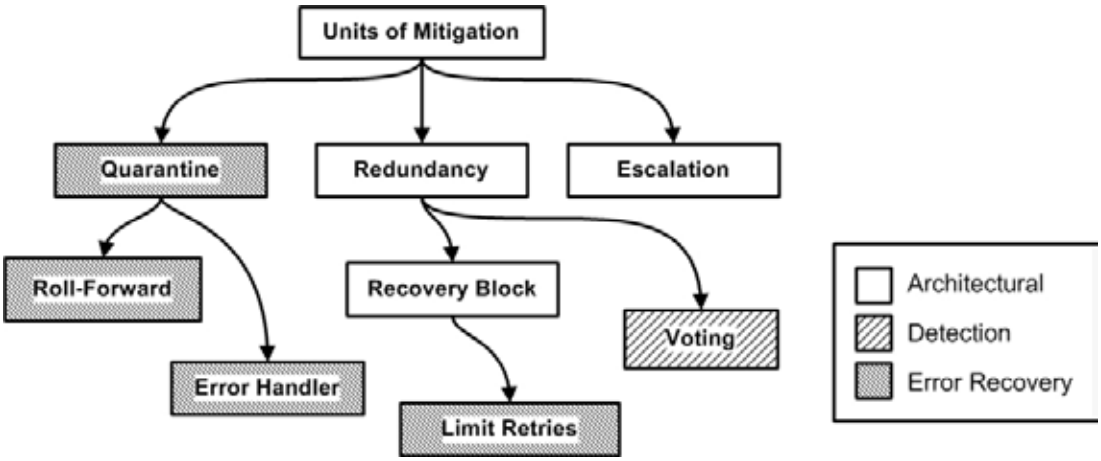
ภาพที่ 5.2 พาร์ตเนอร์ลิงก์ไทป์ในวิสเดิลของกระแสงานบีเพล CityInfo



ภาพที่ 5.3 เอกซ์เอ็มแอลสกีมาของกระแสงานบีเพล CityInfo

ข้อมูลที่ใช้รับส่งกันมีการกำหนดโครงสร้างตามเอกซ์เอ็มแอลสกีมา ดังภาพที่ 5.3 ด้วยโปรแกรม Netbeans ประกอบไปด้วยเอเลเมนต์แรกคือ CityInfoRequest แทนข้อมูลขาเข้าของเซอร์วิซซึ่งประกอบไปด้วยข้อมูลประเภทสตริง 3 ตัวคือชื่อเมือง (City) ชื่อรัฐ (State) และชื่อประเทศ (Country) เอเลเมนต์ที่ 2 คือ CityInfoResponse แทนข้อมูลขาออกของเซอร์วิซซึ่งประกอบไปด้วยข้อมูล 3 ชุดคือ ข้อมูลสกุลเงิน (Currency) เพื่อบอกชื่อและรหัสของสกุลเงิน ข้อมูลเขตเวลา (Time) เพื่อบอกเขตเวลาและเวลาท้องถิ่นในขณะนั้น และข้อมูลสภาพอากาศ (Weather) เพื่อบอกอุณหภูมิและสภาพอากาศขณะนั้น เอเลเมนต์อันดับสุดท้ายคือ FaultMessage ซึ่งใช้ในกรณีที่เกิดความผิดพร่องขึ้น

การเลือกว่าจะนำแบบรูปใดมาใช้จะพิจารณาตามความต้องการของระบบ เมื่อดูจากตัวอย่างของกระแสงานบีเพล CityInfo จะเห็นได้ว่าสามารถแบ่งการทำงานได้ออกเป็น 3 ส่วนและสามารถนำแบบรูปมาประยุกต์ใช้แยกจากกันได้ โดยใช้แท็ก <scope> มาแบ่งกลุ่มตามแบบรูป *Units of Mitigation* จากนั้นจึงกำหนดว่าจะให้แต่ละ <scope> ใช้แบบรูปใดบ้าง โดยใน <scope> แรกซึ่งเป็นการเรียกเว็บเซอร์วิซสกุลเงินจะใช้แบบรูป *Recovery Block* เนื่องจากสามารถเรียกเว็บเซอร์วิซสำรองได้ ใน <scope> ที่ 2 ซึ่งเป็นการเรียกเว็บเซอร์วิซเขตเวลาจะนำแบบรูป *Recovery Block* มาใช้เช่นกันแต่เป็นการเรียกเว็บเซอร์วิซเดิมซ้ำ และ <scope> ที่ 3 ที่เป็นการเรียกเว็บเซอร์วิซสภาพอากาศจะใช้การทำงานแบบขนานโดยเรียกเว็บเซอร์วิซไปพร้อมๆ กันแล้วใช้แบบรูป *Voting* เพื่อนำค่าอุณหภูมิมาเปรียบเทียบ ภาพรวมของแบบรูปที่นำมาใช้กับกระแสงานบีเพล CityInfo สามารถแสดงด้วยแผนที่ภาษาแบบรูปดังภาพที่ 5.4



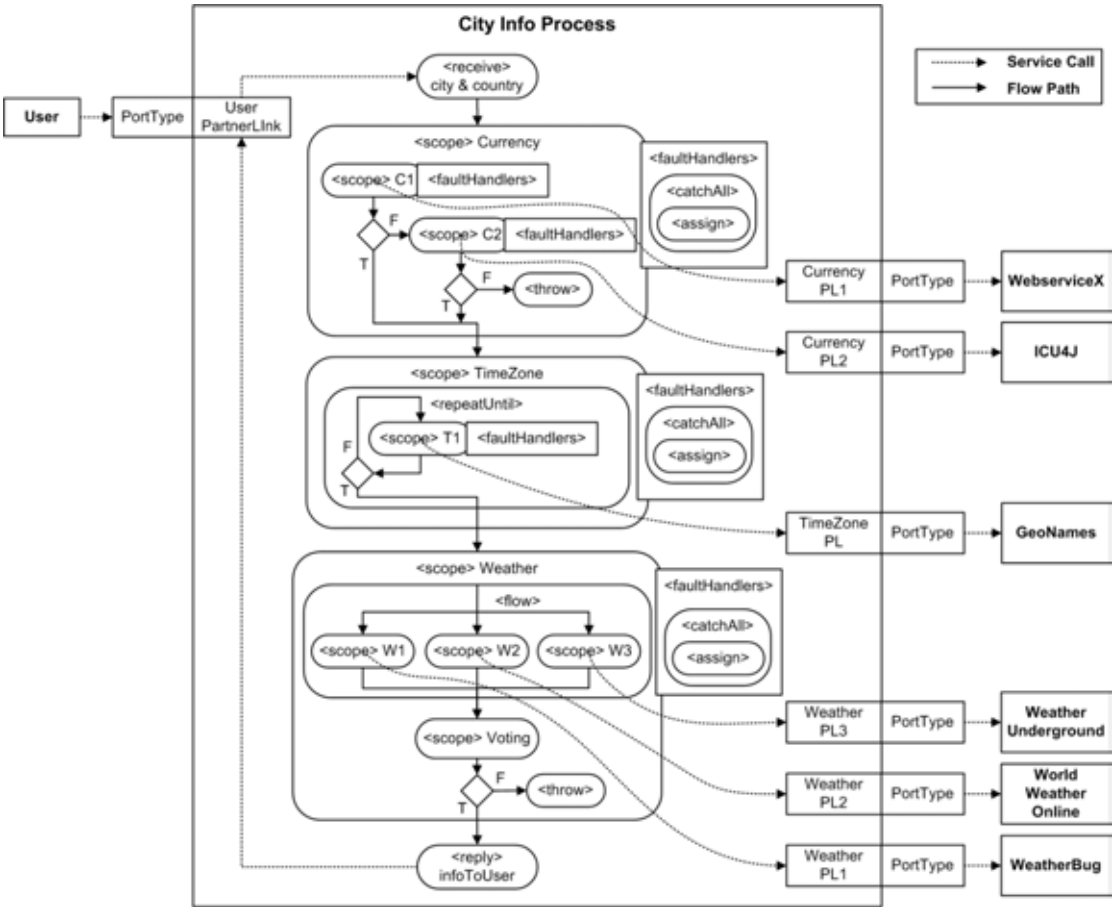
ภาพที่ 5.4 แผนที่ภาษาแบบรูปที่นำมาใช้กับกระแสงานบีเพล CityInfo

จากรายละเอียดของโครงสร้างแม่แบบของบีเพลที่กล่าวมาแล้วนั้นเมื่อนำมาประยุกต์ใช้กับกระแสงานบีเพล CityInfo จะได้โครงสร้างของบีเพลออกมาเป็นดังภาพที่ 5.5 โดยแบ่งการเรียกใช้เว็บเซอร์วิซออกเป็น 3 กลุ่มด้วยแท็ก <scope> ในกรณีที่ <scope> ย่อยภายในของแต่ละกลุ่มไม่

สามารถจัดการกับความผิดพร่องที่เกิดขึ้นได้จะแจ้งว่าเกิดความผิดพร่องออกไปยัง <scope> ชั้นนอกตามแบบรูป *Escalation* และใช้แบบรูป *Roll-Forward* เพื่อข้ามการทำงานของ <scope> ที่เกิดปัญหาและระบุไว้ในตัวแปรด้วยว่าคำตอบของเซอร์วิซนี้เกิดความผิดพลาด

ในส่วนแรกที่เป็นการเรียกใช้เว็บเซอร์วิซค้นหาสกุลเงิน ในที่นี้ได้นำเว็บเซอร์วิซชื่อ “Currency Convertor” จาก WebserviceX [20] มาใช้เป็นส่วนปฐมภูมิตามแบบรูป *Recovery Block* และในส่วนทุติยภูมิได้สร้างเว็บเซอร์วิซขึ้นมาเพื่อเป็นตัวสำรองโดยดึงข้อมูลของสกุลเงินต่างๆ มาจากไลบรารีภาษาจาวาชื่อ ICU4J [21] แต่หากยังมีความผิดพร่องเกิดขึ้นอีกจะแจ้งต่อไปยัง <scope> ชั้นนอกว่ามีความผิดพร่องเกิดขึ้น

ส่วนต่อมาซึ่งเป็นการเรียกใช้เว็บเซอร์วิซค้นหาเขตเวลาได้นำแบบรูป *Recovery Block* มาปรับใช้ในการเรียกซ้ำ ในที่นี้ใช้บริการเว็บเซอร์วิซจาก GeoNames [22] แต่เนื่องจากเป็นเว็บเซอร์วิซแบบ REST ซึ่งไม่สามารถนำมาใช้กับบีเพลได้โดยตรงจึงต้องสร้างเว็บเซอร์วิซแบบ SOAP มาครอบเพื่อทำหน้าที่เป็นตัวแทนในการเรียกใช้อีกต่อหนึ่ง



ภาพที่ 5.5 กระแสงานบีเพล CityInfo ที่ประยุกต์ตามแบบรูปแล้ว

ส่วนสุดท้ายคือการสอบถามสภาพอากาศได้ทำการเรียกใช้เว็บเซอร์วิซที่รายงานสภาพอากาศจาก 3 ผู้ให้บริการคือ WeatherBug [23] และ World Weather Online [24] และ Weather Underground [25] โดยทำการสร้าง <scope> ย่อยขึ้นมาสำหรับการเรียกใช้แต่ละเว็บเซอร์วิซและเรียกใช้ไปพร้อมกันภายใต้โครงสร้างของแท็ก <flow> เมื่อได้ผลลัพธ์ครบทุกเว็บเซอร์วิซแล้วจึงนำค่าอุณหภูมิที่ได้มาทำการหาผลโหวตซึ่งในที่นี้ได้สร้างเป็นกระแสงานบีเพลแยกออกมาอีกตัวหนึ่งและทำการเรียกใช้เช่นเดียวกับเว็บเซอร์วิซอื่นๆ อยู่ภายใน <scope> ต่อจาก <flow> แต่ถ้าการหาผลโหวตไม่สำเร็จจะแจ้งออกไปว่ามีความผิดพร่องเกิดขึ้น

ในการหาผลโหวตได้นำอัลกอริทึมชื่อ “Exact plurality voting” [19] มาปรับใช้โดยแบ่งการทำงานออกเป็น 2 ส่วน โดยในส่วนแรกจะทำการวนซ้ำเพื่อหาค่าผลลัพธ์ที่แตกต่างกันทั้งหมดจากการเรียกใช้แต่ละเว็บเซอร์วิซ รวมทั้งนับจำนวนครั้งที่เกิดขึ้นของแต่ละค่าไว้ด้วย ส่วนที่ 2 จะทำการเปรียบเทียบว่าค่าใดมีจำนวนครั้งที่เกิดขึ้นมากที่สุด ค่านั้นจะกลายเป็นผลโหวตที่จะตอบกลับไปยังผู้ใช้ กระบวนการทำงานของอัลกอริทึมนี้แสดงเป็นโค้ดเทียมได้ดังภาพที่ 5.6

```
-   Add variables
        • $Value as Candidate list type
        • $ValueCounter as integer list type
        • Use <forEach> to create blank items equal to size of candidate list in
          $Value and $ValueCounter
        • $LastIndex and $Threshold as integer
-   Initialize variables
        • $LastIndex ← 1, $Threshold ← 1
        • $ValueCounter[1] ← 1
        • $Value[1] ← $Candidate[1]
-   <forEach> $I from 2 to Candidate size
        • Add $Found ← false, $J ← 1
        • <while> $J <= $LastIndex AND not($Found)
                ▪ <if> value of $Value[$J] and Candidate[$I] are matched
                    • $Found ← true,
                    • Increment $ValueCounter[$J]
                    • Increment $J
            • <if> not($Found)
                ▪ Increment $LastIndex
                ▪ $ValueCounter[$LastIndex] ← 1
                ▪ $Value[$LastIndex] ← $Candidate[$I]
-   <forEach> $K from 1 to $LastIndex
        • <if> $ValueCounter[$K] > $Threshold
                ▪ $Threshold ← $ValueCounter[$K]
                ▪ $Output ← $Value[$K]
```

ภาพที่ 5.6 โค้ดเทียมตามอัลกอริทึม “Exact plurality voting”

5.2 การทดสอบ

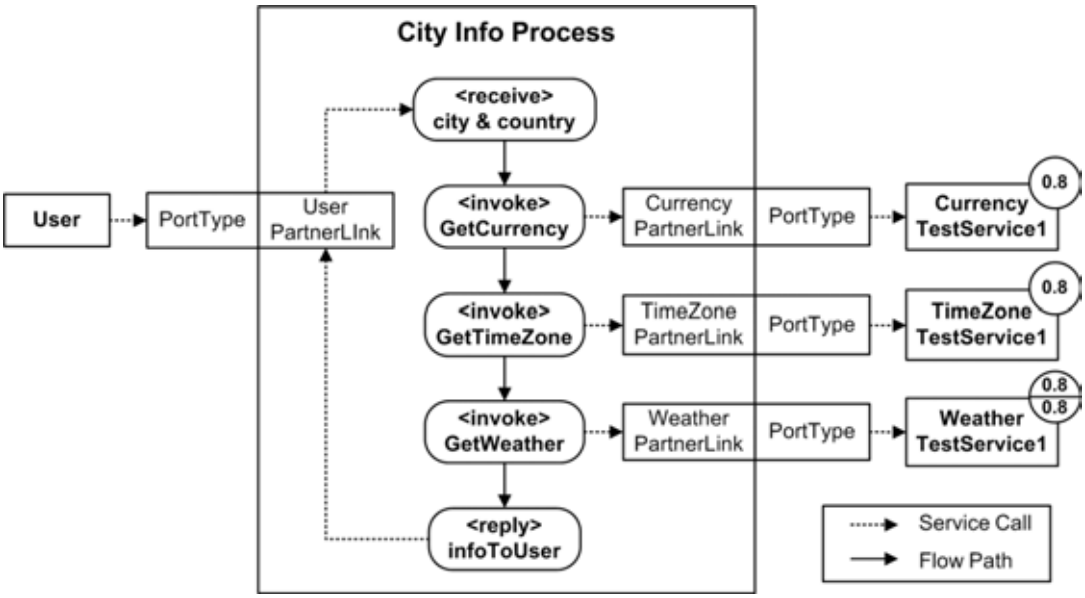
ในการทดสอบแบ่งออกเป็น 2 กลุ่มคือการทดสอบด้านความเชื่อถือได้ และด้านสมรรถนะ เพื่อเปรียบเทียบระหว่างกระแสงานบีเพลแบบปกติและแบบที่ประยุกต์ใช้แบบรูปแล้ว โดยใช้แนวทางในการทดสอบเช่นเดียวกับงานวิจัยของ Liu และคณะ [15]

5.2.1 การทดสอบความเชื่อถือได้

การทดสอบความเชื่อถือได้จะใช้กระแสงานบีเพลตัวอย่าง CityInfo มาทำการเรียกใช้เป็นจำนวน 1000 ครั้ง และหาอัตราส่วนของความสำเร็จในการเรียกใช้ว่ามีค่าเท่าใด โดยจะเปรียบเทียบระหว่างกระแสงานบีเพลปกติและกระแสงานบีเพลที่มีการออกแบบตามแบบรูปแล้ว กระแสงานบีเพลที่ใช้ทดสอบจะทำงานอยู่บนเครื่องประมวลบีเพลของ Glassfish ESB ผ่านทางโปรแกรม Netbeans 6.7.1 และทำการเรียกใช้ด้วยโปรแกรม soapUI 4.0 [26] ด้วยฟังก์ชัน “Load Test” และกำหนดให้มีการเรียกกระแสงานพร้อมกันจาก 5 หน่วยการทำงาน แต่ละหน่วยเรียก 200 ครั้ง รวมทั้งหมด 1000 ครั้ง ทั้งนี้ในการเรียกแต่ละครั้งมีการกำหนด Assertion ชื่อ “Not SOAP Fault” ให้กับผลลัพธ์ที่ได้ไว้ด้วยเพื่อตรวจสอบว่าต้องไม่มีการแจ้งว่าเกิดความผิดพร่องกลับมา จึงจะนับได้ว่าการเรียกครั้งนั้นสำเร็จ

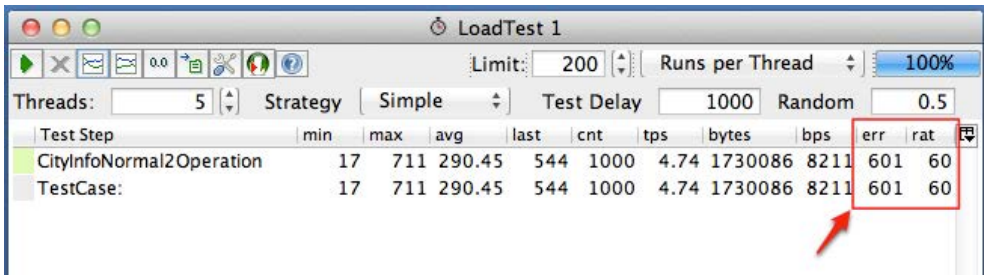
ในการทดสอบนี้จะทำการสร้างเว็บเซอร์วิซที่ถูกเรียกใช้ในกระแสงานขึ้นมา เพื่อจำลองเว็บเซอร์วิซสกุลเงิน เว็บเซอร์วิซเขตเวลา และเว็บเซอร์วิซสภาพอากาศ ที่อาจเกิดความผิดพร่อง โดยกำหนดให้แต่ละเซอร์วิซที่สร้างขึ้น ก่อนจะส่งคำตอบกลับมาต้องทำการสุ่มตัวเลขจาก 1 ถึง 100 ก่อน ถ้าได้ค่าไม่เกิน 80 จึงจะทำการตอบกลับ แต่ถ้าได้เกิน 80 จะแจ้งว่ามีความผิดพร่องกลับไปแทน ซึ่งกล่าวได้ว่าแต่ละเว็บเซอร์วิซจะมีโอกาสที่จะส่งคำตอบได้สำเร็จด้วยความน่าจะเป็น 0.8 นอกจากนี้ได้เพิ่มข้อกำหนดสำหรับเว็บเซอร์วิซสภาพอากาศอีกว่าจะมีโอกาสที่จะรายงานผลลัพธ์อุณหภูมิที่ถูกต้องด้วยความน่าจะเป็น 0.8 เช่นกัน ซึ่งถ้ากระแสงานตรวจพบว่าเป็นค่าอุณหภูมิที่ผิดก็จะทำการแจ้งขึ้นมาว่ามีความผิดพร่องเกิดขึ้นด้วย

ในกระแสงานบีเพล CityInfo แบบปกติจะไม่มีการใช้โครงสร้างในการจัดการกับความผิดพร่องกับเซอร์วิซทั้ง 3 กลุ่ม ส่งผลให้ถ้าการเรียกเว็บเซอร์วิซใดเกิดปัญหาขึ้นจะทำให้ทั้งกระแสงานต้องหยุดทันที ดังนั้นกระแสงานจะทำงานได้สำเร็จก็ต่อเมื่อทั้ง 3 เซอร์วิซไม่เกิดความผิดพร่อง และเว็บเซอร์วิซสภาพอากาศให้คำตอบที่ถูกต้องด้วย กระแสงานบีเพลที่ใช้ทดสอบมีลักษณะดังภาพที่ 5.7



ภาพที่ 5.7 กระแสงานบีเพล CityInfo แบบปกติที่ใช้ทดสอบ

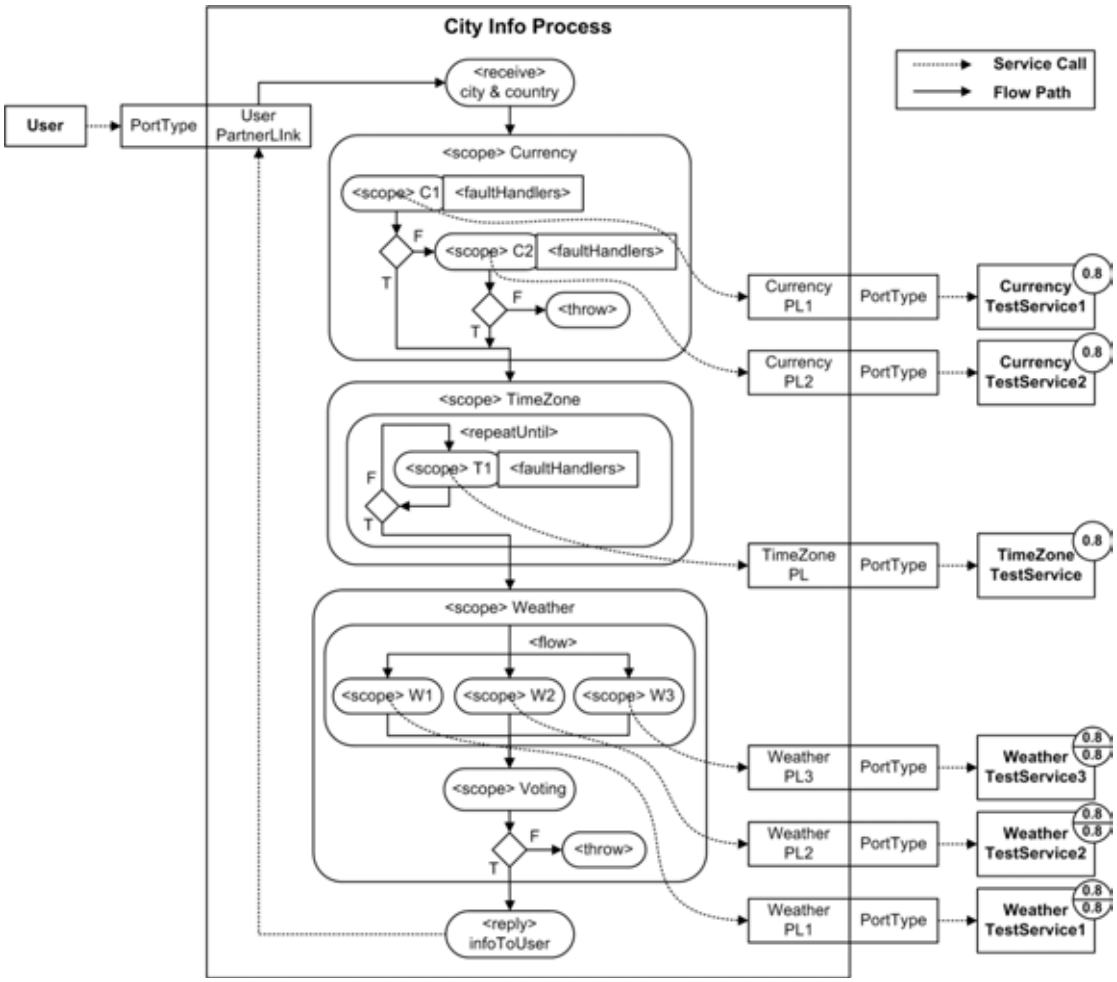
เมื่อนำข้อมูลที่กล่าวข้างต้นมาคำนวณเป็นอัตราความสำเร็จของกระแสงานแบบปกติซึ่งมาจากผลคูณของอัตราความสำเร็จจากทุกเซอร์วิซ มีค่าเป็น 0.8 x 0.8 x (0.8 x 0.8) ซึ่งเท่ากับ 0.4096 เมื่อทำการทดสอบด้วยโปรแกรม soapUI ดังภาพที่ 5.8 แล้วปรากฏว่า จากการเรียกใช้ 1000 ครั้ง เกิดความผิดพร่องขึ้น 601 ครั้ง จึงมีอัตราความสำเร็จเท่ากับ 0.399 ใกล้เคียงกับที่คำนวณไว้



ภาพที่ 5.8 การทดสอบด้วยโปรแกรม soapUI ของกระแสงานบีเพล CityInfo แบบปกติ

ลำดับต่อไปเป็นการทดสอบกระแสงานบีเพล CityInfo ที่ปรับใช้แบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่องตามภาพที่ 5.5 โดยกำหนดให้ในส่วนของเว็บเซอร์วิซสกุลเงินใช้แบบรูป *Recovery Block* จึงสร้างเว็บเซอร์วิซสกุลเงินเพิ่มขึ้นมาอีกตัวหนึ่งเพื่อมาเป็นส่วนทุติยภูมิ ต่อมาคือในส่วนของเว็บเซอร์วิซเขตเวลาใช้แบบรูป *Recovery Block* เช่นกันแต่เรียกเว็บเซอร์วิซเดิมซ้ำแทน โดยมีการหน่วงเวลาไว้ 500 มิลลิวินาทีก่อนเรียกซ้ำ และกำหนดให้เรียกได้ไม่เกิน 5 ครั้ง ส่วนสุดท้ายคือเว็บเซอร์วิซสภาพอากาศใช้แบบรูป *Voting* โดยสร้างเว็บเซอร์วิซสภาพอากาศเพิ่มขึ้นมาอีก 2 ตัว และนำมาเรียกใช้พร้อมกันจากนั้นจึงเรียกใช้เว็บเซอร์วิซในการหาผลโหวตต่อไป ในการทดสอบนี้จะไม่ใช้แบบรูป *Roll-Forward* โดยตัด <faultHandlers> ที่ติดอยู่กับ <scope> ชั้นนอกทั้ง 3 กลุ่ม

ออกไป เพราะต้องการให้มีการแจ้งความผิดพร่องส่งออกไปยังผู้เรียกกระแสงาน เพื่อให้สามารถเปรียบเทียบอัตราความสำเร็จกับกระแสงานปกติได้ กระแสงานบีเพลแบบประยุกต์ใช้แบบรูปที่ใช้ทดสอบมีลักษณะดังภาพที่ 5.9



ภาพที่ 5.9 กระแสงานบีเพล CityInfo แบบประยุกต์ใช้แบบรูปที่ใช้ทดสอบ

การคำนวณอัตราความสำเร็จของกระแสงานที่ปรับใช้แบบรูปแล้วจะเริ่มจากการหาผลรวมของอัตราความล้มเหลวของแต่ละกลุ่มก่อน ซึ่งหมายถึงความน่าจะเป็นที่จะไม่สามารถได้ผลลัพธ์จากการเรียกใช้เว็บเซอร์วิซในแต่ละกลุ่มแม้ว่าจะมีการประยุกต์ใช้แบบรูปแล้วก็ตาม การคำนวณอัตราความล้มเหลวของแต่ละกลุ่มทำโดยอาศัยวิธีการคำนวณอัตราความล้มเหลวของเว็บเซอร์วิซประกอบจากงานวิจัยของ Zheng และ Lyu [27] โดยแบ่งการคำนวณได้เป็นดังนี้

- เว็บเซอร์วิซสกุลเงิน ใช้แบบรูป *Recovery Block* โดยมีตัวสำรอง 1 ตัว อัตราความล้มเหลวจึงเท่ากับผลคูณของอัตราความล้มเหลวทั้ง 2 ตัว คือ 0.2 x 0.2 เท่ากับ 0.04

- เว็บเซอร์วิซเขตเวลา ใช้แบบรูป *Recovery Block* เรียกเซอร์วิซเดิมไม่เกิน 5 ครั้ง อัตราความล้มเหลวจึงเท่ากับผลคูณของอัตราความล้มเหลวทั้ง 5 ครั้ง คือ $(0.2)^5$ เท่ากับ 0.00032
- เว็บเซอร์วิซสภาพอากาศ ใช้แบบรูป *Voting* ต้องแบ่งการคำนวณออกเป็น 2 ส่วน คือ ส่วนการตอบกลับมาของเว็บเซอร์วิซ และส่วนการตอบกลับค่าที่ถูกต้องของเว็บเซอร์วิซ โดยใช้การแจกแจงแบบทวินาม และได้ผลการคำนวณออกมาเท่ากับ 0.218688 ตามตารางที่ 5.1

ตารางที่ 5.1 การคำนวณหาอัตราความล้มเหลวที่จะนำมาใช้ในกรณีของเว็บเซอร์วิซสภาพอากาศ

ความน่าจะเป็นเกี่ยวกับการตอบกลับมาของเว็บเซอร์วิซ	**ความน่าจะเป็นเกี่ยวกับการตอบกลับค่าที่ถูกต้องของเว็บเซอร์วิซ**	**อัตราความล้มเหลวที่นำมาคิด**
ตอบ 3 ตัว: $(0.8)^3 = 0.512$	ถูก 3 ตัว: $(0.8)^3$	นับเฉพาะกรณีที่ตอบผิด 2 ตัวและผิด 3 ตัวได้เป็น 0.512(0.096+0.008) =0.053248
	ผิด 1 ตัว: $3[(0.8)^2(0.2)]$	
	ผิด 2 ตัว: $3[(0.8)(0.2)^2] = 0.096$	
	ผิด 3 ตัว: $(0.2)^3 = 0.008$	
ตอบ 2 ตัว: $3[(0.8)^2(0.2)] = 0.384$	ถูก 2 ตัว: $(0.8)^2$	นับเฉพาะกรณีที่ตอบผิด 1 ตัวและผิด 2 ตัวได้เป็น 0.384(0.32+0.04) = 0.13824
	ผิด 1 ตัว: $2(0.8(0.2)) = 0.32$	
	ผิด 2 ตัว: $(0.2)^2 = 0.04$	
ตอบ 1 ตัว: $3[(0.8)(0.2)^2] = 0.096$	ถูก 1 ตัว: 0.8	นับเฉพาะกรณีที่ตอบผิด 1 ตัวเป็น (0.096)(0.2)= 0.0192
	ผิด 1 ตัว: 0.2	
ไม่ตอบมาเลย: $(0.2)^3 = 0.008$	-	0.008
รวม		**0.218688**

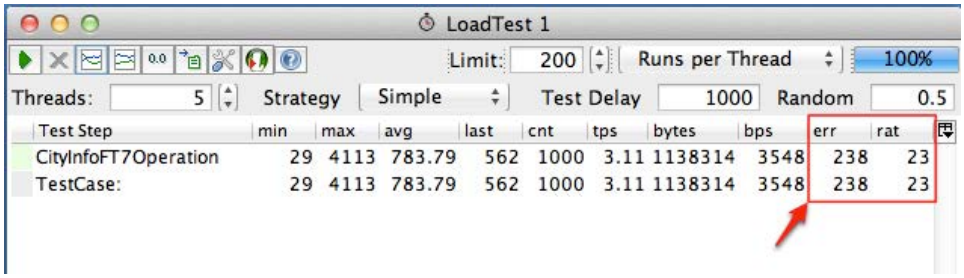
ดังนั้นอัตราความสำเร็จของกระแสงานที่ออกแบบตามแบบรูปคือ

1 – ผลรวมของอัตราความล้มเหลวในการเรียกใช้เว็บเซอร์วิซทั้ง 3 กลุ่ม

= 1 – (0.04 + 0.00032 + 0.21688)

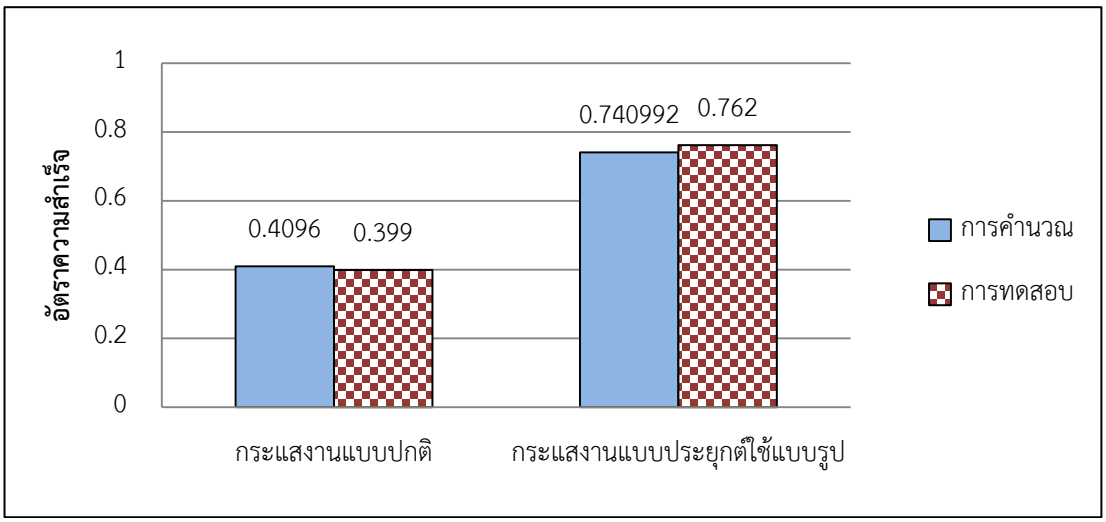
= 0.740992

เมื่อทำการทดสอบด้วยโปรแกรม soapUI ดังภาพที่ 5.10 แล้วปรากฏว่า จากการเรียกใช้ 1000 ครั้ง เกิดความผิดพร่องขึ้น 238 ครั้ง จึงมีอัตราความสำเร็จเท่ากับ 0.762 ใกล้เคียงกับที่คำนวณไว้



ภาพที่ 5.10 การทดสอบด้วยโปรแกรม soapUI ของกระแสงานบีเพล CityInfo ที่ปรับใช้แบบรูปแล้ว

เมื่อนำอัตราความสำเร็จของกระแสงานทั้ง 2 แบบมาเปรียบเทียบกันดังภาพที่ 5.11 พบว่ากระแสงานที่ประยุกต์ใช้แบบรูปมีอัตราความสำเร็จมากกว่ากระแสงานแบบปกติ แสดงว่าการประยุกต์ใช้แบบรูปช่วยเพิ่มความเชื่อถือได้ให้กับกระแสงานบีเพล



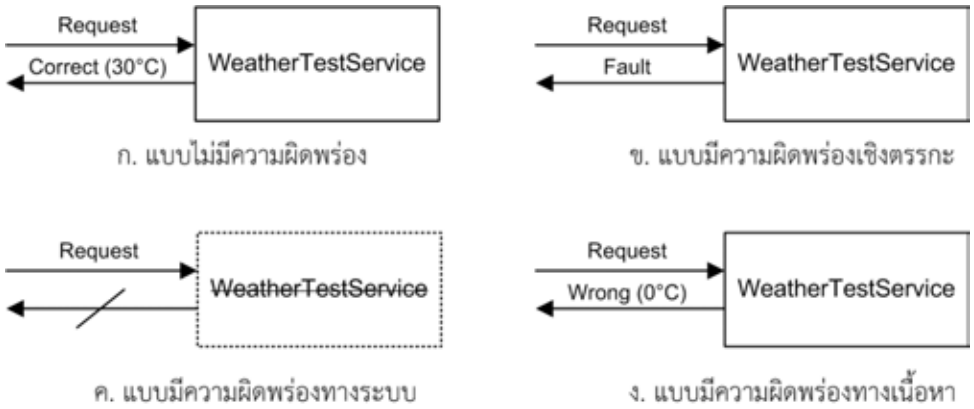
ภาพที่ 5.11 อัตราความสำเร็จของกระแสงานบีเพล CityInfo

5.2.2 การทดสอบผลกระทบต่อสมรรถนะ

จากการทดสอบที่ผ่านมาจะเห็นได้ว่าการนำแบบรูปมาประยุกต์กับการออกแบบกระแสงานบีเพลจะช่วยเพิ่มความเชื่อถือได้ให้กับกระแสงาน แต่เนื่องจากมีความซับซ้อนเพิ่มขึ้น ย่อมส่งผลต่อความเร็วในการประมวลผล ซึ่งวัดได้จากเวลาที่ใช้ในการตอบกลับ (Response time) เมื่อมีการเรียกใช้กระแสงานนั้น เพื่อเป็นการศึกษาผลกระทบที่เกิดขึ้นจึงทำการทดสอบเปรียบเทียบเวลาที่ใช้ของกระแสงานปกติกับกระแสงานที่มีการปรับใช้แบบรูปต่างๆ

การทดสอบนี้เลือกใช้เว็บเซอร์วิซสภาพอากาศมาเป็นตัวแทนของเว็บเซอร์วิซคู่ค้าที่จะทดสอบสมรรถนะ และทำการสร้างกระแสงานที่เรียกใช้เว็บเซอร์วิซสภาพอากาศอีกต่อหนึ่งเพื่อจำลองสภาพการเกิดความผิดพร่องจากการเรียกใช้เซอร์วิซ ซึ่งแบ่งออกเป็น ความผิดพร่องเชิงตรรกะ

ความผิดพร่องทางระบบ และ ความผิดพร่องทางเนื้อหา ดังนั้นจึงทำการจำลองเว็บเซอร์วิซสภาพอากาศที่แตกต่างกัน 4 ชุดสำหรับการทดสอบมีลักษณะดังภาพที่ 5.12



ภาพที่ 5.12 การจำลองเว็บเซอร์วิซสภาพอากาศในการทดสอบผลกระทบต่อสมรรถนะ

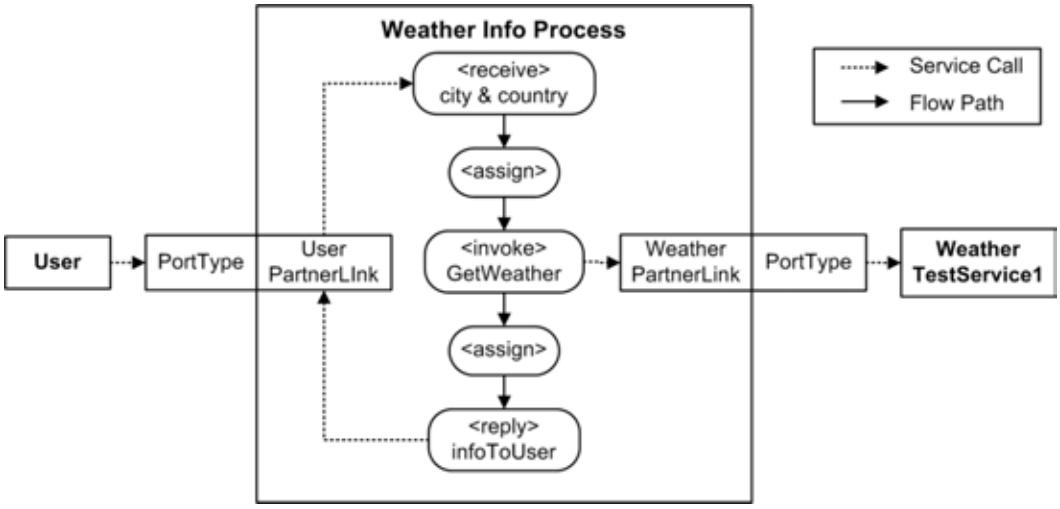
รายละเอียดของแต่ละชุดเป็นดังต่อไปนี้

- แบบไม่มีความผิดพร่อง คือให้เว็บเซอร์วิซรับคำร้องแล้วให้คำตอบมาตามปกติ (ในที่นี้ให้ตอบว่า 30°C)
- แบบมีความผิดพร่องเชิงตรรกะ คือให้เว็บเซอร์วิซรับคำร้องแล้วแจ้งกลับมาว่ามีความผิดพร่องเกิดขึ้นทุกครั้งที่เรียกใช้
- แบบมีความผิดพร่องทางระบบ คือเรียกใช้เว็บเซอร์วิซตัวเดียวกับแบบไม่มีความผิดพร่อง แต่มีการถอดเว็บเซอร์วิซนั้นออกจากระบบไปก่อนที่จะทำการเรียกใช้ เพื่อให้เครื่องประมวลบีเพลไม่พบเว็บเซอร์วิซที่ต้องการแล้วแจ้งว่าเกิดความผิดพร่อง
- แบบมีความผิดพร่องทางเนื้อหา คือให้เว็บเซอร์วิซรับคำร้องแล้วตอบอุณหภูมิที่ผิดกลับมาทุกครั้ง (ในที่นี้ให้ตอบว่า 0°C) ซึ่งเมื่อกระแสงานได้รับมาแล้วสามารถตรวจสอบได้ว่าเป็นคำตอบที่ผิด

เว็บเซอร์วิซแต่ละชุดจะมีตัวสำรองที่เหมือนกันอีก ชุดละ 3 ตัว เพื่อใช้ในการทดสอบกับแบบรูป *Recovery Block* และแบบรูป *Voting* นอกจากนี้แต่ละเว็บเซอร์วิซจะมีการหน่วงเวลาเอาไว้ด้วยกิจกรรมบีเพล <wait> เป็นเวลา 500 มิลลิวินาที ก่อนที่จะตอบกลับเพื่อช่วยให้เปรียบเทียบได้ชัดเจนยิ่งขึ้น

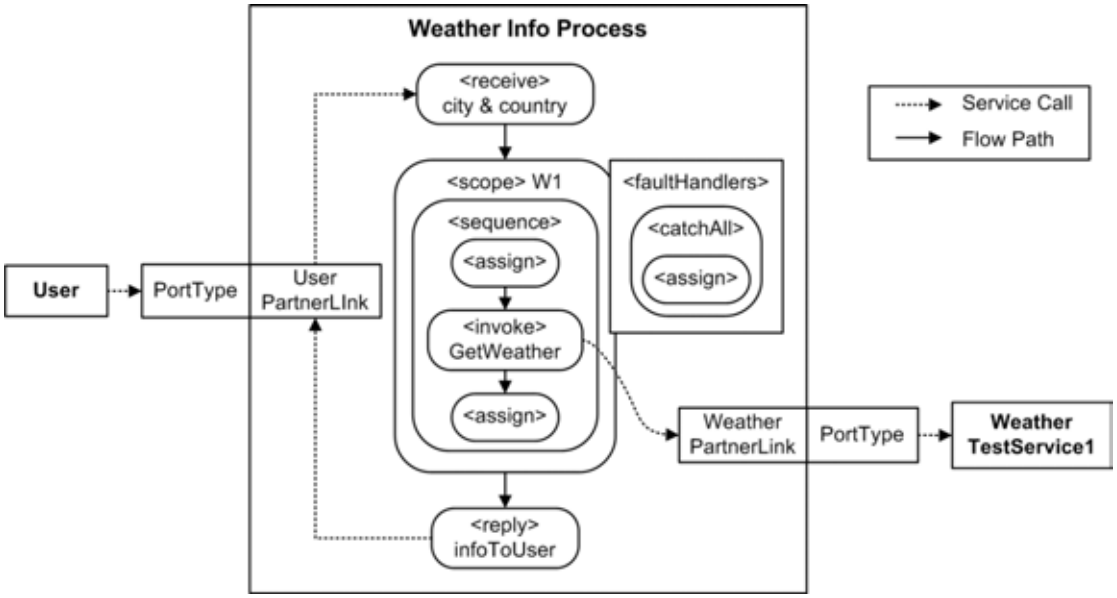
ต่อมาเป็นการสร้างกระแสงานเพื่อทำการเรียกใช้เว็บเซอร์วิซสภาพอากาศทั้ง 4 ชุดที่สร้างไว้ โดยสร้างออกมา 5 ชุด ตามแบบรูปที่นำมาประยุกต์ใช้ รายละเอียดของกระแสงานแต่ละชุดเป็นดังนี้

1) กระแสงานปกติที่ไม่ใช้แบบรูป มีเพียงกิจกรรมบีเพล <assign> และ <invoke> ที่ใช้เรียกเว็บเซอร์วิซเท่านั้น และให้เครื่องประมวลบีเพลดักจับความผิดพร่องเองดังภาพที่ 5.13



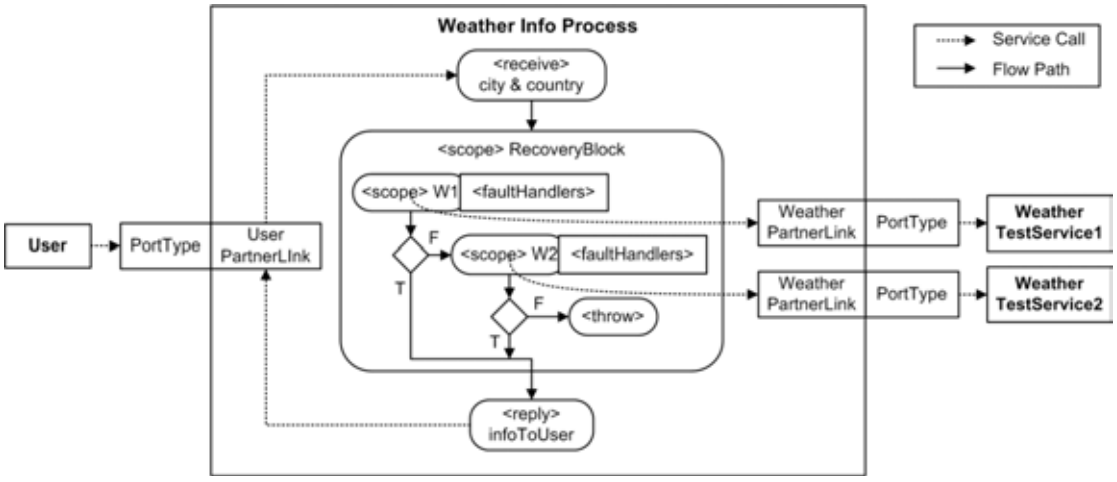
ภาพที่ 5.13 กระแสงานในการทดสอบที่ไม่ใช้แบบรูป

2) กระแสงานตามแบบรูป *Error Handler* เพิ่มการใช้งานโครงสร้างของ <scope> และ <faultHandlers> หากดักจับความผิดพร่องได้จะทำการกำหนดค่าตัวแปรและตอบกลับไปดังภาพที่ 5.14



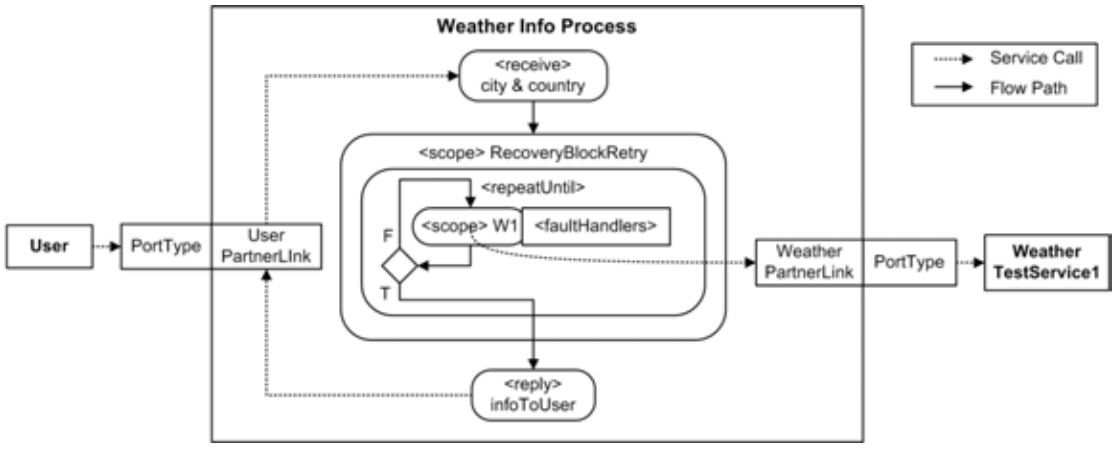
ภาพที่ 5.14 กระแสงานในการทดสอบตามแบบรูป *Error Handler*

3) กระแสงานตามแบบรูป *Recovery Block* ที่เพิ่มส่วนทุติยภูมิเข้ามาด้วยการเรียกเว็บเซอร์วิซสำรองอีก 1 ตัว และหากเรียกไม่ผ่านทั้งหมดจะแจ้งต่อไปด้วยแท็ก <throw> แล้วให้เครื่องประมวลบีเพลดักจับความผิดพร่องเองดังภาพที่ 5.15



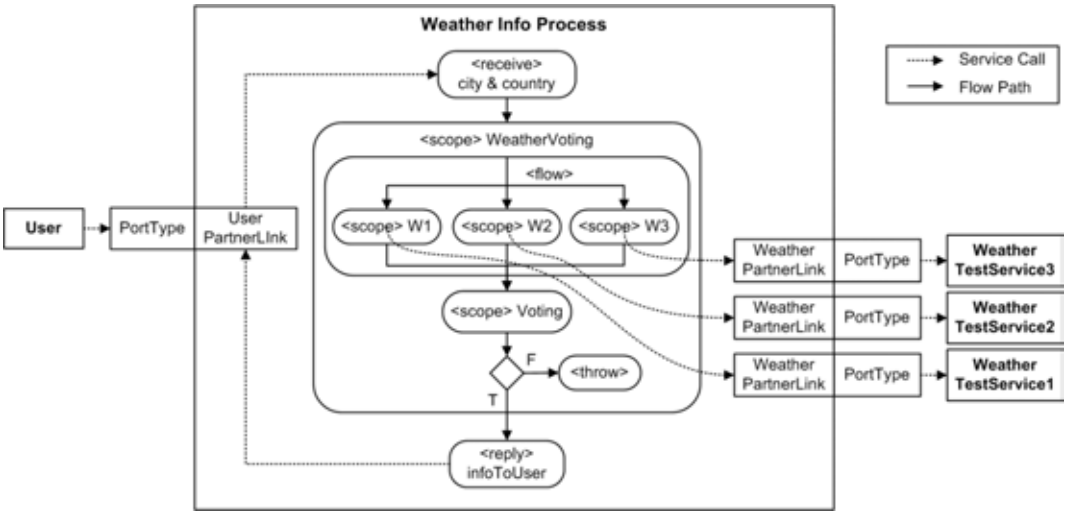
ภาพที่ 5.15 กระแสงานในการทดสอบตามแบบรูป *Recovery Block*

4) กระแสงานตามแบบรูป *Recovery Block* ที่เรียกซ้ำเว็บเซอร์วิซเดิมไม่เกิน 5 ครั้ง แต่ละรอบจะหน่วงเวลาไว้ 500 มิลลิวินาที และหากเรียกไม่ผ่านทั้งหมดจะแจ้งต่อไปด้วยแท็ก <rethrow> แล้วให้เครื่องประมวลบีเพลดักจับความผิดพร่องเองดังภาพที่ 5.16



ภาพที่ 5.16 กระแสงานในการทดสอบตามแบบรูป *Recovery Block* แบบเรียกซ้ำ

5) กระแสงานตามแบบรูป *Voting* ที่เรียกเว็บเซอร์วิซพร้อมกัน 3 ตัวและนำมาหาผลโหวต และหากหาผลโหวตไม่สำเร็จจะแจ้งต่อไปด้วยแท็ก <throw> แล้วให้เครื่องประมวลบีเพลดักจับความผิดพร่องเองดังภาพที่ 5.17



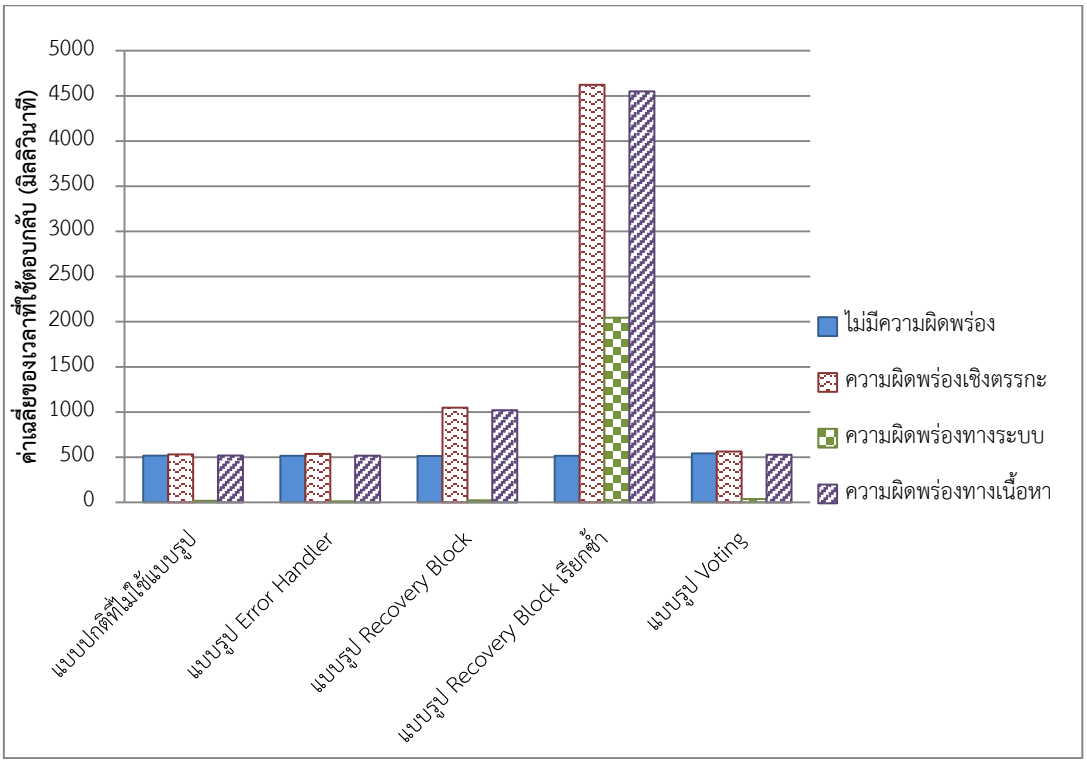
ภาพที่ 5.17 กระแสงานในการทดสอบตามแบบรูป *Voting*

หลังจากนั้นทำการทดสอบด้วยโปรแกรม soapUI โดยทำการเรียกกระแสงานแต่ละชุดเป็นจำนวน 1000 ครั้งแล้วหาค่าเฉลี่ยของเวลาที่ใช้ตอบกลับ จากชุดทดสอบทั้งหมด 20 ชุดได้ผลเป็นดังตารางที่ 5.2

ตารางที่ 5.2 ค่าเฉลี่ยของเวลาที่ใช้ตอบกลับของกระแสงานตามแบบรูปและความผิดพร่องที่ใช้ทดสอบ

ชนิดความผิดพร่อง	ค่าเฉลี่ยของเวลาที่ใช้ตอบกลับของกระแสงานตามแบบรูป (มิลลิวินาที)				
	แบบปกติที่ไม่ใช้แบบรูป	แบบรูป Error Handler	แบบรูป Recovery Block	แบบรูป Recovery Block เรียกซ้ำ	แบบรูป Voting
ไม่มีความผิดพร่อง	517.86	515.56	514.15	515.39	542.66
ความผิดพร่องเชิงตรรกะ	532.67	536.77	1049.64	4623.09	564.57
ความผิดพร่องทางระบบ	17.70	11.87	23.12	2044.53	37.01
ความผิดพร่องทางเนื้อหา	520.15	517.01	1021.35	4550.72	528.10

จากข้อมูลในตารางที่ 5.2 นำมาแสดงเป็นแผนภูมิเพื่อเปรียบเทียบค่าเฉลี่ยของเวลาที่ใช้ตอบกลับของกระแสงานตามแบบรูปและชนิดของความผิดพร่องที่ใช้ทดสอบ ดังภาพที่ 5.18



ภาพที่ 5.18 ค่าเฉลี่ยของเวลาที่ใช้ตอบกลับของกระแสงานตามแบบรูปและความผิดพร่องที่ใช้ทดสอบ

จะเห็นได้ว่าเวลาที่ใช้ในการเรียกกระแสงานในกรณีที่ไม่มีความผิดพร่องนั้นจะใกล้เคียงกันทั้งแบบปกติและแบบที่ปรับใช้แบบรูปแล้ว โดยแบบรูป *Voting* จะใช้เวลามากที่สุดเนื่องจากต้องรอการตอบกลับจากทั้ง 3 เซอร์วิซและนำไปหาผลโหวตต่อ ซึ่งใช้เวลามากกว่าแบบรูปอื่นๆ เพียงเล็กน้อย แสดงว่าการปรับใช้แบบรูปเข้ากับกระแสงานมีผลต่อเวลาที่ใช้เพียงเล็กน้อยในภาวะปกติ

ส่วนในกรณีที่มีความผิดพร่องเกิดขึ้นเวลาที่ใช้จะแตกต่างกันไปขึ้นอยู่กับแบบรูปที่นำมาปรับใช้ เนื่องจากเวลาที่ใช้ตอบกลับมานี้เกิดจากกรณีเลวร้ายสุดเพราะเกิดความผิดพร่องทุกครั้งที่เรียกเว็บเซอร์วิซ ส่งผลให้แบบรูป *Recovery Block* และแบบรูป *Recovery Block* เรียกซ้ำ จะมีการเรียกเว็บเซอร์วิซทั้งหมด 2 และ 5 ครั้งตามลำดับ ทำให้เสียเวลามากกว่าแบบรูปอื่น เพราะในแต่ละครั้งของการเรียกเว็บเซอร์วิซมีการหน่วงเวลาไว้ 500 มิลลิวินาที ส่วนแบบรูป *Error Handler* และแบบรูป *Voting* มีการเรียกเว็บเซอร์วิซเพียงครั้งเดียว จึงใช้เวลาใกล้เคียงกับกรณีที่ไม่ใช้แบบรูป

ข้อสังเกตอีกประการหนึ่งคือในกรณีของความผิดพร่องทางระบบจะใช้เวลาน้อยมากเพราะเครื่องประมวลบีเพลจะพบความผิดพร่องทันทีในขณะที่เรียกเว็บเซอร์วิซเนื่องจากไม่พบเว็บเซอร์วิซที่ต้องการ แต่ในแบบรูป *Recovery Block* เรียกซ้ำ จะมีการหน่วงเวลาก่อนเริ่มรอบใหม่เป็นเวลา 500 มิลลิวินาที ในการทดสอบนี้กำหนดให้มีการเรียกทั้งหมด 5 ครั้ง จึงมีการหน่วงเวลาก่อนเริ่มรอบใหม่ทั้งหมด 4 ครั้ง รวมเป็นเวลาที่ต้องใช้เพิ่มอีก 2000 มิลลิวินาที

บทที่ 6

บทสรุป

6.1 สรุปผลการวิจัย

งานวิจัยนี้ได้นำเสนอแนวทางในการนำแบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่องมาประยุกต์ใช้กับกระบวนการบีเพล เริ่มจากแบบรูปที่มีการรวบรวมและจัดหมวดหมู่ไว้ตามลำดับขั้นในการทำให้ซอฟต์แวร์ทนต่อความผิดพร่องทั้งหมด 63 แบบรูป และแบ่งได้เป็น 5 กลุ่ม ผู้วิจัยทำการพิจารณาในเบื้องต้นเฉพาะ 3 กลุ่มแรกซึ่งเพียงพอต่อการทำให้กระบวนการบีเพลทนต่อความผิดพร่องที่เกิดจากเว็บเซอร์วิซคู่ค้า จากนั้นวิเคราะห์แต่ละแบบรูปถึงความเหมาะสมในการใช้งานในระดับการออกแบบกระแสงานบีเพล เพื่อคัดเลือกแบบรูปที่สามารถใช้เพียงโครงสร้างตามมาตรฐานของดับเบิลยูเอส-บีเพลเวอร์ชัน 2.0 ในการออกแบบตามแบบรูปเหล่านั้น และสามารถนำไปใช้กับการออกแบบในสภาพแวดล้อมและเครื่องมือที่แตกต่างกันได้ ผลที่ได้จากการคัดเลือกแบบรูปตามที่ต้องการมีจำนวน 16 แบบรูป

ผู้วิจัยได้นำเสนอแม่แบบของโครงสร้างบีเพลที่ออกแบบตามแบบรูปที่ได้คัดเลือกไว้ ซึ่งสามารถนำไปใช้กับชุดเครื่องมือพัฒนาทั่วไปได้ โดยในงานวิจัยนี้ได้ใช้โปรแกรม Netbeans เวอร์ชัน 6.7.1 ในชุดของเครื่องประมวลบีเพล Glassfish ESB และโปรแกรม Eclipse ที่ทำงานร่วมกับเครื่องประมวลบีเพล Apache ODE มาใช้ในการออกแบบและสามารถใช้งานแม่แบบได้ในทั้ง 2 ชุดพัฒนา อีกทั้งได้แสดงตัวอย่างในการประยุกต์ใช้แบบรูปกับกระแสงานบีเพล เพื่อเป็นแนวทางให้ผู้ออกแบบกระแสงานบีเพลสามารถนำไปใช้ในการออกแบบต่อไปได้ โดยสร้างกระแสงานบีเพลแบบปกติขึ้นมาและพิจารณาถึงความผิดพร่องที่อาจเกิดขึ้นได้จากเว็บเซอร์วิซคู่ค้า และคัดเลือกแบบรูปที่เหมาะสมออกมาส่วนหนึ่งเพื่อมาปรับโครงสร้างของกระแสงานเดิมให้รองรับการทนต่อความผิดพร่องได้

จากการทดสอบความเชื่อถือได้ระหว่างกระแสงานแบบปกติและกระแสงานที่ปรับใช้แบบรูปแล้วตามตัวอย่างที่แสดงไว้ พบว่ากระแสงานที่ปรับใช้แบบรูปมีโอกาสที่จะทำงานได้สำเร็จเพิ่มขึ้น แต่เมื่อทดสอบผลกระทบต่อสมรรถนะของกระแสงานบีเพลพบว่าเมื่อกระแสงานที่มีการใช้แบบรูปจะใช้เวลาในการทำงานมากกว่ากระแสงานปกติเล็กน้อย ยกเว้นแบบรูปที่มีการวนซ้ำด้วยจะใช้เวลามากขึ้นตามจำนวนรอบที่กำหนดไว้

6.2 ปัญหาและข้อจำกัดที่พบจากการวิจัย

6.2.1 ข้อจำกัดของเครื่องมือพัฒนา

ในงานวิจัยนี้ได้นำเสนอแม่แบบของโครงสร้างบีเพลที่ประยุกต์ตามแบบรูปต่างๆ ซึ่งใช้โครงสร้างตามดับเบิลยูเอส-บีเพล 2.0 และสามารถนำไปใช้กับเครื่องมือพัฒนาทั่วไปได้ โดยใช้โปรแกรม Netbeans เวอร์ชัน 6.7.1 ซึ่งอยู่ในชุดของ Glassfish ESB เป็นชุดพัฒนาหลัก และใช้โปรแกรม Eclipse ที่ใช้งานร่วมกับ Apache ODE มาช่วยตรวจสอบความเข้ากันได้เมื่อนำโครงสร้างของบีเพลที่ออกแบบไว้ไปใช้ในเครื่องมือพัฒนาที่ต่างกัน แต่เครื่องมือพัฒนาทั้ง 2 ชุดต่างมีข้อจำกัดในการออกแบบโครงสร้างของบีเพลตามแบบรูปต่างๆ ดังนี้

- ทั้ง Glassfish ESB 2.2 และ Apache ODE 1.3.5 ไม่รองรับการใช้กิจกรรมบีเพล <validate> ทำให้ไม่สามารถสั่งให้ตรวจสอบข้อมูลโดยใช้คำสั่งนี้ได้ หากผู้ออกแบบต้องการประยุกต์ใช้แบบรูป *Complete Parameter Checking* จำเป็นต้องตรวจสอบในช่วงออกแบบด้วย Netbeans หรือ Eclipse หรือเพิ่มเติมส่วนตรวจสอบข้อมูลไว้ในกระแสงานเอง
- ทั้ง Netbeans และ Eclipse ไม่รองรับการออกแบบกระแสงานบีเพลด้วยโครงสร้างของแท็ก <link> ซึ่งสามารถเชื่อมต่อกิจกรรมบีเพลให้ทำงานตามเส้นทางที่ต้องการภายใต้โครงสร้างของแท็ก <flow> โดยสามารถกำหนดต้นทาง ปลายทาง และเงื่อนไขกำกับไว้ในแต่ละเส้นได้ รวมทั้งสามารถกำหนดปลายทางให้เป็นตำแหน่งเดียวกันได้ด้วย ซึ่งทำให้ใช้โครงสร้างที่ซับซ้อนน้อยกว่าการใช้โครงสร้างแบบ <if> และ <else> ซ้อนกันไปเรื่อยๆ

6.2.2 ปัญหาจากความแตกต่างกันของชุดเครื่องมือพัฒนา

การพัฒนากระแสงานบีเพลด้วยเครื่องมือพัฒนาที่แตกต่างกันจะมีขั้นตอนในการสร้างกระแสงานบีเพลที่ต่างกัน และเมื่อสร้างเสร็จแล้วจะได้ไฟล์บีเพลที่มีรายละเอียดแตกต่างกันด้วย ทำให้ไม่สามารถใช้ไฟล์ที่สร้างขึ้นมาแลกเปลี่ยนกันระหว่างเครื่องมือพัฒนาที่แตกต่างกันได้ ในงานวิจัยนี้ใช้โปรแกรม Netbeans เวอร์ชัน 6.7.1 ซึ่งอยู่ในชุดของ Glassfish ESB เป็นชุดพัฒนาหลัก และใช้โปรแกรม Eclipse ที่ใช้งานกับ Apache ODE เป็นตัวทดสอบความเข้ากันได้ของโครงสร้างบีเพลที่ออกแบบไป ปัญหาที่เกิดขึ้นเมื่อใช้งานชุดพัฒนาทั้ง 2 นี้ร่วมกันมีดังนี้

- ใน Eclipse เมื่อใช้ URL ในแอททริบิวต์ชื่อ location ของแท็ก <import> เพื่อการอ้างอิงตำแหน่งของไฟล์ จะเกิดปัญหาเมื่อสั่ง deploy เนื่องจากไม่สามารถค้นหาไฟล์ดังกล่าวได้ จำเป็นต้องเก็บไฟล์ที่ต้องการนั้นไว้ภายในโฟลเดอร์ของโปรเจค

เดียวกันกับไฟล์ที่อ้างถึง และระบุที่อยู่ของไฟล์แบบสัมพัทธ์ (relative path) จึงจะสามารถ deploy ให้สำเร็จได้

- ใน Netbeans การกำหนดค่าให้กับตัวแปรด้วยคำสั่ง <assign> สามารถอ้างอิงตำแหน่งของข้อมูลภายในตัวแปรนั้นตามรูปแบบของภาษา XPath ได้เลยโดยไม่ต้องใส่ค่าเริ่มต้นไว้ก่อน แต่ใน Eclipse จำเป็นต้องกำหนดค่าเริ่มต้นตามโครงสร้างของเอกซ์เอ็มแอลสกีมาให้กับตัวแปรนั้นก่อนถึงจะสามารถอ้างอิงข้อมูลภายในได้

6.2.3 ข้อจำกัดของภาษาบีเพล

ภาษาดับเบิลยูเอส-บีเพล 2.0 รองรับการเรียกเว็บเซอร์วิซผ่านวิสเดิล และส่งข้อความกันผ่านโพรโทคอลแบบ SOAP แต่ในปัจจุบันมีการใช้งานเว็บเซอร์วิซแบบ REST กันมากขึ้น เช่น World Weather Online [24] หรือ Weather Underground [25] ซึ่งเป็นเว็บเซอร์วิซที่ใช้ทดลองในงานวิจัยนี้ หากผู้ออกแบบต้องการให้บีเพลเรียกใช้เว็บเซอร์วิซแบบ REST ได้ จำเป็นต้องพึ่งพาความสามารถของชุดเครื่องมือพัฒนาและเครื่องประมวลบีเพลที่ใช้ว่ามีความสามารถรองรับหรือไม่ หรือจำเป็นต้องสร้างเว็บเซอร์วิซพร็อกซีขึ้นมาเพื่อรับส่งข้อมูลจากบีเพลด้วยข้อความแบบ SOAP และทำการส่งต่อไปยังเซอร์วิซจริงด้วยวิธีการแบบ REST อีกทีหนึ่ง ด้วยเหตุนี้ทำให้การสร้างกระแสงานบีเพลเพื่อเป็นเว็บเซอร์วิซประกอบยังมีข้อจำกัดอยู่เนื่องจากขาดความยืดหยุ่นในการเรียกใช้เว็บเซอร์วิซภายนอกที่ไม่ได้ออกแบบเป็น SOAP ไว้

6.3 ข้อเสนอแนะ

งานวิจัยนี้สามารถพัฒนาเพิ่มเติมได้โดยออกแบบเครื่องมือสนับสุนนในการเลือกแบบรูปที่เหมาะสมและสร้างเป็นส่วนของโครงสร้างเข้าไปในกระแสงานตามแม่แบบโครงสร้างบีเพลของแบบรูปที่ได้เลือกไว้ ซึ่งจะเพิ่มความสะดวกให้แก่ผู้ออกแบบกระแสงานบีเพลสามารถเลือกใช้และปรับเปลี่ยนโครงสร้างของกระแสงานตามแบบรูปได้รวดเร็วยิ่งขึ้น นอกจากนั้นยังพัฒนาต่อไปให้สามารถแปลงกระแสงานบีเพลปกติให้เป็นกระแสงานบีเพลที่รองรับการทนต่อความผิดพร่องได้ โดยเลือกเพียงแบบรูปที่ต้องการ และปรับแต่งค่าพารามิเตอร์ต่างๆ ตามความเหมาะสม

รายการอ้างอิง

[1] OASIS. Web Services Business Process Execution Language Version 2.0 [Online]. 2007. Available from: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html [2011, September].

[2] Dobson, G. Using WS-BPEL to Implement Software Fault Tolerance for Web Services. Proceedings of Software Engineering and Advanced Applications, 2006. SEAA '06. 32nd EUROMICRO Conference on, pp. 126-133. 2006.

[3] Modafferi, S. and Conforti, E., Methods for Enabling Recovery Actions in Ws-BPEL, in On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE. pp. 219-236, 2006.

[4] Liu, A., Li, Q., Huang, L., and Xiao, M. A Declarative Approach to Enhancing the Reliability of BPEL Processes. Proceedings of Web Services, 2007. ICWS 2007. IEEE International Conference on, pp. 272-279. 2007.

[5] Strunk, A., Braun, I., Reichert, S., and Schill, A. Supporting Rebinding in BPEL. Proceedings of Web Services, 2009. ICWS 2009. IEEE International Conference on, pp. 864-871. Los Angeles, CA 6-10 July 2009, 2009.

[6] Moser, O., Rosenberg, F., and Dustdar, S. Non-Intrusive Monitoring and Service Adaptation for WS-BPEL. Proceedings of the 17th international conference on World Wide Web, Beijing, China, 2008. ACM.

[7] Ezenwoye, O. and Sadjadi, S. M. TRAP/BPEL: A Framework for Dynamic Adaptation of Composite Services. Proceedings of the International Conference on Web Information Systems and Technologies (WEBIST 2007), 2007.

[8] Laranjeiro, N. and Vieira, M. Towards Fault Tolerance in Web Services Compositions. Proceedings of the 2007 workshop on Engineering fault tolerant systems, Dubrovnik, Croatia, 2007. ACM.

[9] Christos, K., Vassilakis, C., Rouvas, E., and Georgiadis, P. Exception Resolution for BPEL Processes: a Middleware-based Framework and Performance Evaluation. Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services, Linz, Austria, 2008. ACM.

[10] Hanmer, R., Patterns for Fault Tolerant Software. Chichester: Wiley Publishing, 2007.

[11] Thaisongsuwan, T. and Senivongse, T. Fault Tolerant WS-BPEL Business Processes. Proceedings of The 12th National Computer Science and Engineering Conference (NCSEC 2008), pp. 339-346. Pattaya, Thailand, November 19-21, 2008.

[12] Thaisongsuwan, T. and Senivongse, T. Applying Software Fault Tolerance Patterns to WS-BPEL Processes. Proceedings of 2011 Eighth International Joint Conference on Computer Science and Software Engineering (JCSSE), pp. 269-274. Nakhon Pathom, Thailand, May 11-13, 2011.

[13] W3C. Web Services Description Language (WSDL) 1.1 [Online]. 2001. Available from: http://www.w3.org/TR/wsdl [2011, September].

[14] Chan, K. S. M., Bishop, J., Steyn, J., Baresi, L., and Guinea, S., A Fault Taxonomy for Web Service Composition, in Service-Oriented Computing - ICSOC 2007 Workshops: ICSOC 2007, International Workshops, Vienna, Austria, September 17, 2007, Revised Selected Papers. pp. 363-375: Springer-Verlag, 2009.

[15] Liu, A., Li, Q., Huang, L., and Xiao, M. FACTS: A Framework for Fault-Tolerant Composition of Transactional Web Services. Services Computing, IEEE Transactions on 3 (2010): 46-59.

[16] Subramanian, S., Thiran, P., Narendra, N. C., Mostefaoui, G. K., and Maamar, Z. On the Enhancement of BPEL Engines for Self-Healing Composite Web Services. Proceedings of Applications and the Internet, 2008. SAINT 2008. International Symposium on, pp. 33-39. 2008.

[17] Russell, N., van der Aalst, W., and ter Hofstede, A., Workflow Exception Patterns, in Advanced Information Systems Engineering. pp. 288-302, 2006.

[18] Russell, N., Aalst, W. M. P. v. d., and Hofstede, A. H. M. t. Exception Handling Patterns in Process-Aware Information Systems. BPM Center Report (2006).

[19] Parhami, B. Voting algorithms. Reliability, IEEE Transactions on 43 (1994): 617-629.

[20] WebserviceX.NET. Currency Convertor [Online]. Available from: http://www.webservicex.net/ws/WSDetails.aspx?CATID=2&WSID=10 [2011, September].

[21] ICU. ICU - International Components for Unicode [Online]. Available from: http://site.icu-project.org/ [2011, September].

[22] GeoNames. GeoNames Web Services Documentation [Online]. Available from: http://www.geonames.org/export/web-services.html [2011, September].

[23] WeatherBug. WeatherBug API [Online]. Available from: http://apireg.weatherbug.com/defaultAPI.aspx [2011, September].

[24] World Weather Online. World Weather Online - Free Local Weather API Overview [Online]. Available from: http://www.worldweatheronline.com/free-weather-feed.aspx [2011, September].

[25] WeatherUnderground. WeatherUnderground API - XML Feed [Online]. Available from: http://wiki.wunderground.com/index.php/API_-_XML [2011, September].

[26] soapUI [Online]. Available from: http://www.soapui.org/ [2011, September].

[27] Zheng, Z. and Lyu, M. R. Optimal Fault Tolerance Strategy Selection for Web Services. International Journal of Web Services Research (IJWSR) 7 (2010): 21-40.

ประวัติผู้เขียนวิทยานิพนธ์

นายทวี ไทยส่งสุวรรณ เกิดเมื่อวันที่ 14 เมษายน พ.ศ. 2529 ที่จังหวัดกรุงเทพมหานคร สำเร็จการศึกษาระดับปริญญาวิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ เกียรตินิยมอันดับ 2 จากคณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2550 และเข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ ณ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2551 งานวิจัยที่สนใจได้แก่ เว็บเซอร์วิซ กระแสงานบีเพล และการทนต่อความผิดพร่อง