ยูเอ็มแอลโปรไฟล์สำหรับแบบรูปการทนต่อความผิดพร่องสำหรับระบบอิงบริการ

นางสาวอรอนงค์ องค์ศิริพร

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2556

UML Profile for Fault Tolerance Patterns for Service-Based Systems

Ms. Ornanong Ongsiriporn

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science Program in Computer Science
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2013

หัวข้อวิทยานิพนธ์	ยูเอ็มแอลโปรไฟล์สำหรับแบบรูปการทนต่อความผิดพร่องสำหรับระบบอิงบริการ
โดย	นางสาวอรอนงค์ องค์ศิริพร
สาขาวิชา	วิทยาศาสตร์คอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	รองศาสตราจารย์ ดร.ทวิตีย์ เสนีวงศ์ ณ อยุธยา

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

..... คณบดีคณะวิศวกรรมศาสตร์
(ศาสตราจารย์ ดร.บัณฑิต เอื้ออาภรณ์)

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(รองศาสตราจารย์ ดร.พรศิริ หมื่นไชยศรี)

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

..... กรรมการภายนอกมหาวิทยาลัย
(ผู้ช่วยศาสตราจารย์ ดร.มฑุปายาส ทองมาก)

อรอนงค์ องค์ศิริพร : ยูเอ็มแอลโปรไฟล์สำหรับแบบรูปการทนต่อความผิดพร่องสำหรับระบบอิงบริการ. (UML PROFILE FOR FAULT TOLERANCE PATTERNS FOR SERVICE-BASED SYSTEMS) อ.ที่ปรึกษาวิทยานิพนธ์หลัก : รศ. ดร.ทวิตีย์ เสนีวงศ์ ณ อยุธยา, 139 หน้า

ระบบอิงบริการประกอบด้วยหน่วยที่เรียกว่า เซอร์วิซ ซึ่งมีหน้าที่ให้บริการงานต่างๆผ่านอินเทอร์เน็ต โดยเป็นส่วนหนึ่งของการทำงานในระบบของผู้ใช้บริการ ระบบมีโอกาสเกิดความล้มเหลวเนื่องจากปัญหาที่เกิดขึ้นกับเซอร์วิซ เช่น ปัญหาที่เกี่ยวกับการติดต่อสื่อสาร หรือปัญหาที่เกิดจากความผิดพร่องในตัวเซอร์วิซเอง งานวิจัยนี้มุ่งเน้นไปที่การให้ความสำคัญกับมุมมองด้านการทนต่อความผิดพร่องในขั้นตอนการออกแบบระบบอิงบริการ และนำเสนอยูเอ็มแอลโปรไฟล์สำหรับแบบรูปการทนต่อความผิดพร่อง ซึ่งสามารถนำไปใช้ในการออกแบบระบบอิงบริการใดๆให้มีการทนต่อความผิดพร่อง ยูเอ็มแอลโปรไฟล์ในงานวิจัยนี้ครอบคลุมแบบรูปการทนต่อความผิดพร่องในระดับการออกแบบเชิงสถาปัตยกรรมและแบบรูปสำหรับการตรวจหาและกู้ระบบจากข้อผิดพลาด จากนั้นนำเสนอการประยุกต์ใช้งานยูเอ็มแอลโปรไฟล์เข้ากับการออกแบบกรณีศึกษา ได้แก่ แอพพลิเคชันการจัดการห่วงโซ่อุปทาน ซึ่งเป็นตัวอย่างแอพพลิเคชันสำหรับระบบอิงบริการซึ่งเสนอโดยองค์กรด้านการทำงานร่วมกันระหว่างเว็บเซอร์วิซ งานวิจัยนี้ได้ทำการตรวจสอบความถูกต้องสอดคล้องของการออกแบบยูเอ็มแอลโปรไฟล์ตามคำนิยาม อีกทั้งในการประเมินผลยังพบว่าการออกแบบระบบให้ทนต่อความผิดพร่องยังส่งผลในเชิงบวกต่อคุณลักษณะเชิงคุณภาพหลายลักษณะ ยกเว้นคุณสมบัติด้านความสามารถในการทำความเข้าใจ อันเป็นผลเนื่องมาจากระบบมีความซับซ้อนเพิ่มขึ้น แม้กระนั้นก็ตามการออกแบบได้ช่วยเพิ่มความสามารถในการทนต่อความผิดพร่องให้กับระบบกรณีศึกษาที่นำไปพัฒนาจริง

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา วิทยาศาสตร์คอมพิวเตอร์ ลายมือชื่อ อ. ที่ปรึกษาวิทยานิพนธ์หลัก.....

ปีการศึกษา 2556

5471033121 : MAJOR COMPUTER ENGINEERING

KEYWORD : UMLPROFILE / FAULT TOLERANCE / PATTERNS / SERVICES

ORNANONG ONGSIRIPORN : UML PROFILE FOR FAULT TOLERANCE PATTERNS FOR SERVICE-BASED SYSTEMS PROCESSES. ADVISOR : ASSOC. PROF. TWITTIE SENIVONGSE, Ph.D., 139 pp.

Service-based systems consist of software units called services which provide software functionalities over the Internet to other parts of the systems. The systems may experience failure due to problems associated with the services such as communication problems and faults within the services themselves. In this research, we emphasize the importance of fault tolerance mindset during the design of service-based systems and propose a UML profile for fault tolerance patterns which can be used to build a design model for any fault tolerant service-based systems. The profile covers fault tolerance patterns at the architecture level and the patterns for error detection and recovery. We present how to use the UML profile to design a fault tolerant version of the supply chain management application, a sample application of the Web Services Interoperability Organization. The research reports the consistency checking between the design of UML profile and the corresponding fault tolerance patterns definition. Also an evaluation shows that fault tolerance design has a positive impact on a number of service quality attributes of the design model, except for understandability as a result of higher complexity. Nevertheless, it gives the ability to tolerate faults to the implementation of the case study application.

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

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

Academic Year: 2013

กิตติกรรมประกาศ

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

ขอขอบพระคุณรองศาสตราจารย์ ดร.พรศิริ หมื่นไชยศรี ประธานคณะกรมการสอบวิทยานิพนธ์ และ ผู้ช่วยศาสตราจารย์ ดร.มฑุปายาส ทองมาก กรรมการสอบวิทยานิพนธ์ที่ให้ข้อชี้แนะในการปรับปรุงงานวิทยานิพนธ์ให้มีคุณภาพยิ่งขึ้น

ขอขอบพระคุณคณาจารย์ทุกท่านในภาควิชาวิศวกรรมคอมพิวเตอร์ที่ได้อบรม สั่งสอน ให้ความรู้ต่างๆ ซึ่งเป็นประโยชน์ต่อการทำวิจัยและการทำงานในอนาคต

ขอขอบพระคุณนายศิริ จงเกษมถาวร และพี่ๆ ที่บริษัท โน เมจิค เอเชีย ที่คอยให้ความช่วยเหลือ ให้คำปรึกษา และให้คำแนะนำ ตลอดระยะเวลาการทำวิจัยนี้

ท้ายที่สุดนี้ขอขอบพระคุณพ่อ คุณแม่ และครอบครัวที่เป็นกำลังใจ ความรัก ความห่วงใยและแรงสนับสนุนสำคัญในด้านการศึกษาจนทำให้ประสบความสำเร็จมาได้ถึงทุกวันนี้

สารบัญ

หน้า

หน้า

หน้า

หน้า

หน้า

สารบัญตาราง

สารบัญภาพ

ภาพที่ หน้า

ภาพที่ หน้า

ภาพที่ หน้า

ภาพที่ หน้า

บทที่ 1

บทนำ

บทที่ 1 จะมีการนำเสนอความเป็นมาและความสำคัญของปัญหา วัตถุประสงค์ของการวิจัย ขอบเขตของการวิจัย ประโยชน์ที่คาดว่าจะได้รับ วิธีดำเนินการวิจัย และผลงานตีพิมพ์ แสดงดังหัวข้อต่อไปนี้

1.1 ความเป็นมาและความสำคัญของปัญหา

รูปแบบการใช้งานอินเทอร์เน็ตมีการเปลี่ยนแปลงจากรูปแบบที่มีการทำงานเพียงลำพังไปเป็นการทำงานแบบสนับสนุนการแลกเปลี่ยนข้อมูล หรือให้บริการระหว่างกันมากขึ้น จึงทำให้เกิดสถาปัตยกรรมของซอฟต์แวร์แบบเอสโอเอ (Service-Oriented Architecture: SOA) ที่นิยามถึงวิธีการใช้บริการที่เป็นอิสระต่อกัน เพื่อรองรับการทำงานร่วมกันของระบบต่างๆ ซึ่งเทคโนโลยีที่สนับสนุนการพัฒนาตามสถาปัตยกรรมเอสโอเอ คือ เว็บเซอร์วิซ (Web Services) และถูกนำมาใช้อย่างแพร่หลายในการทำธุรกรรมโดยอาศัยระบบอิเล็กทรอนิกส์ หรือ การบริหารจัดการภาครัฐผ่านสื่ออิเล็กทรอนิกส์ การนำเทคโนโลยีในเรื่องของเว็บเซอร์วิซมาใช้งานจำเป็นจะต้องคำนึงถึงความเสถียรของการทำงาน ปัจจัยหนึ่งที่สำคัญในการพิจารณาเซอร์วิซ คือ คุณภาพของเซอร์วิซ (Quality of Service) [1] ซึ่งประกอบด้วย ความมั่นคงของเซอร์วิซ (Security) ความเชื่อถือได้ของเซอร์วิซ (Reliability) และประสิทธิภาพของเซอร์วิซ (Efficiency) องค์ประกอบหนึ่งที่ผู้วิจัยได้นำมาพิจารณาเป็นหลักคือ ความเชื่อถือได้ของเซอร์วิซในหัวข้อคุณสมบัติของเซอร์วิซที่ทำให้ระบบสามารถปฏิบัติงานต่อไปได้แม้ว่าจะมีองค์ประกอบภายในบางอย่างเสียหาย ซึ่งเรียกคุณสมบัตินี้ว่า การทนต่อความผิดพร่อง (Fault Tolerance) รูปแบบของการเกิดความผิดพร่องสามารถแบ่งได้หลายลักษณะ เช่น ความผิดพร่องซึ่งเกิดจากการพัฒนา (Development Fault) หรือความผิดพร่องซึ่งเกิดจากการทำงาน (Operational Fault)

โดยทั่วไปกระบวนการการพัฒนาระบบหนึ่งๆจะประกอบด้วยขั้นตอนหลักๆ ได้แก่ ส่วนของการออกแบบ การพัฒนาโค้ด และการติดตั้งระบบ ส่วนมากในขั้นตอนของการพัฒนาโค้ด มักมีการนำเฟรมเวิร์คหรือไลบรารีต่างๆมาช่วยในการแก้ปัญหาบางอย่างของระบบ เช่น ปัญหาทางความผิดพร่องของระบบ แต่ในมุมมองของผู้วิจัย การแก้ปัญหาตั้งแต่ขั้นตอนการออกแบบยังไม่ถูกกล่าวถึงมากนัก ซึ่งผู้วิจัยคิดว่าการแก้ปัญหาตั้งแต่ขั้นตอนการออกแบบสามารถลดปัญหาที่เกิดขึ้นในช่วงของการพัฒนาโค้ดและลดปัญหาที่จะเกิดขึ้นกับผู้ใช้งาน จากข้อกำหนด “UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms” [2] ของโอเอ็มจี (OMG) ได้กล่าวไว้ว่า โดยส่วนใหญ่ในขั้นตอนการออกแบบมักจะมุ่งเน้นไปในส่วนของการออกแบบ

ตามความต้องการที่เป็นฟังก์ชันการทำงานหลักของระบบ (Functional Requirement) เท่านั้น โดยยังไม่เน้นการออกแบบส่วนที่ไม่ใช่ฟังก์ชันหลัก (Non-functional Requirement) เช่น ความมั่นคงของระบบ หรือ ความเชื่อถือได้ของระบบเท่าที่ควร ถึงแม้ว่าในขั้นตอนการออกแบบจะมีการกล่าวถึงความสามารถในด้านที่ไม่ใช่ฟังก์ชันหลักของระบบ ก็จะเป็นเพียงคำอธิบายเท่านั้น โดยไม่มีการออกแบบเป็นแผนภาพอย่างชัดเจน เช่นเดียวกับการพัฒนาระบบอิงบริการซึ่งการออกแบบยังมุ่งเน้นไปในส่วนของการออกแบบตามความต้องการที่เป็นฟังก์ชันการทำงานหลักของระบบเพียงเท่านั้น ดังตัวอย่างเช่น การออกแบบส่วนต่อประสาน (Interface) ของเซอร์วิซจะเป็นการออกแบบตามการทำงานของฟังก์ชันหลักของระบบอิงบริการ แต่ไม่มีการระบุถึงคุณสมบัติของส่วนที่ไม่ใช่ฟังก์ชันหลัก ดังนั้นเมื่อผู้ใช้บริการเรียกดูความสัมพันธ์ของส่วนต่อประสานของเซอร์วิซเหล่านี้ก็จะไม่มีการระบุถึงส่วนที่ไม่ใช่ฟังก์ชันหลักด้วย

ภาษาแบบจำลองที่ใช้ในการวิเคราะห์และออกแบบระบบที่ใช้กันอย่างแพร่หลายคือยูเอ็มแอล (Unified Modeling Language: UML) ซึ่งเป็นภาษาที่ใช้ในการแสดงแบบจำลองการทำงานของระบบหรือโปรแกรม ในงานวิจัยนี้จะมีการนำเสนอยูเอ็มแอลโปรไฟล์ (UML Profile) [3] สำหรับการออกแบบระบบที่ทนต่อความผิดพร่อง เนื่องจากสามารถนำยูเอ็มแอลโปรไฟล์ไปประยุกต์ใช้เข้ากับการออกแบบของระบบที่มีอยู่แล้ว ให้มีความสามารถในการทนต่อความผิดพร่องโดยไม่ต้องทำการออกแบบระบบใหม่ทั้งหมด

งานวิจัยหลายงานได้มีการนำเสนอการออกแบบเพื่อรองรับการทนต่อความผิดพร่อง แต่งานวิจัยเหล่านี้ยังออกแบบไม่ครอบคลุมทุกขั้นตอนของการทนต่อความผิดพร่อง การออกแบบยังมุ่งเน้นไปยังแบบรูปแบบใดแบบหนึ่ง หรือออกแบบเพื่อมุ่งเน้นไปยังโดเมนใดโดเมนหนึ่งเท่านั้น ดังตัวอย่างของ Shim, Baek, Kim และ Park [4] จะมุ่งเน้นไปเพียงการออกแบบในกระบวนการของการตรวจหาข้อผิดพลาดและการกู้ระบบจากข้อผิดพลาดสำหรับระบบหุ่นยนต์เท่านั้น

ดังนั้น งานวิจัยนี้จึงมีแนวคิดในการนำเสนอยูเอ็มแอลโปรไฟล์ที่ใช้ในการออกแบบระบบอิงบริการเพื่อให้มีความสามารถด้านการทนต่อความผิดพร่อง โดยมีการนำแนวคิดแบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่อง (Patterns for Fault Tolerant Software) ของ Hanmer [5] มาเป็นแนวทางในการออกแบบซอฟต์แวร์ให้ทนต่อความผิดพร่อง ซึ่งแบบรูปของ Hanmer จะมีการระบุถึงปัญหา บริบท และ แนวทางแก้ไข มาประยุกต์ใช้กับการออกแบบ โดยจะมุ่งเน้นในแบบรูปที่แก้ปัญหาในขั้นตอนของการออกแบบสถาปัตยกรรม (Architectural Pattern) การตรวจหาข้อผิดพลาด (Error Detection Pattern) และการกู้ระบบจากข้อผิดพลาด (Error Recovery Pattern) และมีการนำมาใช้งานกับกรณีศึกษา: ตัวอย่างสถาปัตยกรรมโปรแกรมประยุกต์สำหรับ

ระบบจัดการห่วงโซ่อุปทาน (Supply Chain Management Sample Application Architecture) [6, 7] ซึ่งเป็นตัวอย่างกรณีศึกษาสำหรับระบบอิงบริการ

1.2 วัตถุประสงค์ของการวิจัย

1.2.1 เพื่อนำเสนอยูเอ็มแอลโปรไฟล์สำหรับแบบรูปการทนต่อความผิดพร่องสำหรับระบบอิงบริการ

1.3 ขอบเขตของการวิจัย

1.3.1 วิเคราะห์แบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่องของ Hanmer [5] สำหรับกลุ่มต่อไปนี้เป็นอย่างน้อย

1.3.1.1 แบบรูปเชิงสถาปัตยกรรม (Architectural Patterns)

1.3.1.2 แบบรูปการตรวจหา (Detection Patterns)

1.3.1.3 แบบรูปการกู้ระบบจากข้อผิดพลาด (Error Recovery Patterns)

เพื่อนำมาใช้ในการออกแบบยูเอ็มแอลโปรไฟล์สำหรับแบบรูปการทนต่อความผิดพร่องทั้งหมด 3 กลุ่ม รวมอย่างน้อย 15 แบบรูป

1.3.2 กำหนดคุณสมบัติสำคัญของแบบรูปเพื่อเป็นคุณสมบัติของการนำโปรไฟล์ไปใช้งาน

1.3.3 พิจารณาการออกแบบยูเอ็มแอลโปรไฟล์ตามยูเอ็มแอลรุ่น 2.4

1.3.4 พิจารณาการประเมินผลยูเอ็มแอลโปรไฟล์โดยวัดค่าคุณลักษณะเชิงคุณภาพของการออกแบบระบบโดยประเมินจากการออกแบบที่มีการประยุกต์ใช้ยูเอ็มแอลโปรไฟล์สำหรับแบบรูปการทนต่อความผิดพร่องและประเมินผลความสามารถในการทนต่อความผิดพร่องของระบบที่พัฒนาขึ้นมาตามการออกแบบด้วยยูเอ็มแอลโปรไฟล์

1.3.5 ประยุกต์ใช้แบบรูปการทนต่อความผิดพร่องเข้ากับกรณีศึกษาที่ประกอบด้วยอย่างน้อย 5 เซอร์วิซ

1.4 ประโยชน์ที่คาดว่าจะได้รับ

ได้ยูเอ็มแอลโปรไฟล์สำหรับแบบรูปการทนต่อความผิดพร่องสำหรับระบบอิงบริการ ซึ่งนำไปเป็นแนวทางในการออกแบบระบบอิงบริการเพื่อให้สามารถทนต่อความผิดพร่อง ซึ่งจะนำไปสู่การลดปัญหาในขั้นตอนของการพัฒนาโปรแกรม

1.5 วิธีดำเนินการวิจัย

1.5.1 ศึกษาการออกแบบยูเอ็มแอลโปรไฟล์

1.5.2 ศึกษาและวิเคราะห์แบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่อง

1.5.3 วิเคราะห์งานวิจัยที่เกี่ยวข้องกับการออกแบบยูเอ็มแอลโปรไฟล์สำหรับการทนต่อความผิดพร่องของระบบ

1.5.4 วิเคราะห์ข้อกำหนดของการสร้างโปรไฟล์สำหรับคุณภาพของเซอร์วิซและการทนต่อความผิดพร่องที่กำหนดโดยโอเอ็มจี

1.5.5 กำหนดยูเอ็มแอลโปรไฟล์สำหรับแบบรูปการทนต่อความผิดพร่องสำหรับระบบอิงบริการ

1.5.6 ประยุกต์ใช้ยูเอ็มแอลโปรไฟล์กับกรณีศึกษา

1.5.7 ทดสอบการใช้งานยูเอ็มแอลโปรไฟล์สำหรับแบบรูปการทนต่อความผิดพร่อง

1.5.8 จัดทำบทความวิชาการและวิทยานิพนธ์

1.6 ผลงานตีพิมพ์

ส่วนหนึ่งของวิทยานิพนธ์นี้ได้ตีพิมพ์และนำเสนอในการประชุมวิชาการดังนี้

1.6.1 บทความชื่อ *“UML Profile for Fault Tolerance Patterns for Service Based System”* [8]

1.6.1.1 ชื่อผู้แต่ง Ornanong Ongsiriporn และ Twittie Senivongse

1.6.1.2 ตีพิมพ์และนำเสนอในงานประชุมวิชาการชื่อ The 10th International Joint Conference on Computer Science and Software Engineering (JCSSE2013) ซึ่งจัดขึ้นในวันที่ 29-31 พฤษภาคม 2556 ณ จ.ขอนแก่น ประเทศไทย

บทที่ 2

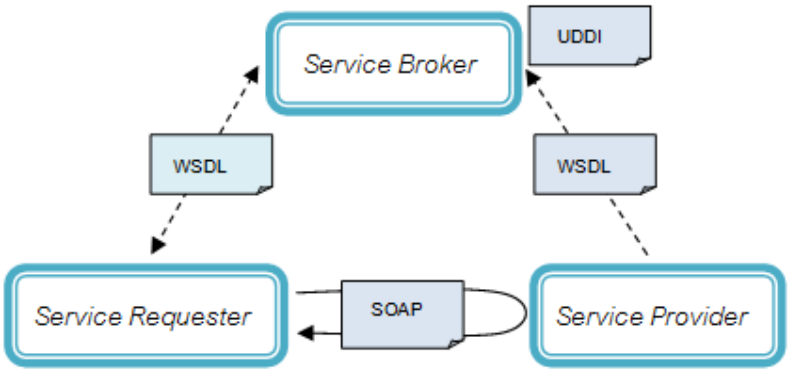
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

บทที่ 2 นำเสนอทฤษฎีและงานวิจัยที่เกี่ยวข้องสำหรับงานวิจัยนี้ แสดงดังหัวข้อต่อไปนี้

2.1 เว็บเซอร์วิซ

เว็บเซอร์วิซ (Web service) [9] เป็นเทคโนโลยีรูปแบบหนึ่งที่ถูกพัฒนามาตามสถาปัตยกรรมของซอฟต์แวร์แบบเอสโอเอ ซึ่งออกแบบมาเพื่อสนับสนุนการแลกเปลี่ยนข้อมูลกันระหว่างเครื่องคอมพิวเตอร์ผ่านระบบเครือข่าย โดยไม่ต้องคำนึงถึงแฟลตฟอร์มที่ใช้ในการพัฒนา ภาษาที่ใช้ในการพัฒนา และระบบปฏิบัติการ ภาษาที่ใช้ในการติดต่อสื่อสารระหว่างเครื่องคอมพิวเตอร์ คือเอกซ์เอ็มแอล โดยที่เว็บเซอร์วิซจะมีส่วนต่อประสาน ได้แก่ วิสเดิล (WSDL) ที่ใช้อธิบายรูปแบบการให้บริการและที่อยู่ของเซอร์วิซ ระบบคอมพิวเตอร์ใช้งานเว็บเซอร์วิซโดยการสื่อสารโต้ตอบกับเว็บเซอร์วิซตามรูปแบบที่ได้กำหนดไว้แล้ว โดยการส่งข้อมูลตามส่วนต่อประสานของเว็บเซอร์วิซนั้น อาจส่งไปกับโพรโทคอลโซป (SOAP) หรือส่งตามส่วนต่อประสานที่เซอร์วิซกำหนดไว้ ข้อมูลจะถูกส่งโดยใช้อินเทอร์เน็ตโพรโทคอล เช่น เอชทีทีพี (HTTP) เอสเอ็มทีพี (SMTP) และเอฟทีพี (FTP) เป็นต้น และมียูดีดีไอ (UDDI: Universal Description, Discovery, and Integration) ทำหน้าที่เป็นไดเรกทอรีสำหรับผู้ให้บริการมาลงประกาศไว้ เพื่อให้ผู้ใช้บริการมาค้นหาหรือสอบถามบริการ โดยที่ในการพัฒนาโปรแกรมตามรูปแบบของเว็บเซอร์วิซ โปรแกรมที่ถูกพัฒนาบนแพลตฟอร์มต่างๆกันสามารถใช้เว็บเซอร์วิซเพื่อแลกเปลี่ยนข้อมูลผ่านทางเครือข่ายคอมพิวเตอร์ เช่น อินเทอร์เน็ต ในลักษณะเดียวกับการสื่อสารระหว่างโปรเซส (Inter-process communication) บนเครื่องเดียวกัน ความสามารถของการแลกเปลี่ยนข้อมูลระหว่างระบบที่ต่างกันมีดังเช่น การแลกเปลี่ยนข้อมูลระหว่างโปรแกรมที่เขียนด้วยภาษาจาวา และโปรแกรมที่เขียนด้วยภาษาไพทอน หรือการแลกเปลี่ยนข้อมูลระหว่างโปรแกรมประยุกต์ที่ทำงานบนระบบปฏิบัติการวินโดวส์และโปรแกรมประยุกต์ที่ทำงานบนระบบปฏิบัติการลีนุกซ์

การทำงานของเว็บเซอร์วิซเริ่มต้นโดยผู้ให้บริการทำการสร้างเอกสารวิสเดิลเพื่ออธิบายรายละเอียดของการให้บริการและประกาศข้อมูลเกี่ยวกับบริการไว้ที่ยูดีดีไอ เมื่อผู้ใช้บริการต้องการเรียกใช้บริการอย่างใดอย่างหนึ่งจะทำการค้นหาผ่านทางยูดีดีไอ และผู้ใช้บริการจะได้รับเอกสารวิสเดิลผ่านทางยูดีดีไอ ซึ่งเป็นเอกสารที่อธิบายรายละเอียดเกี่ยวกับรูปแบบการเรียกใช้บริการ ผลลัพธ์ วิธีการติดต่อ และตำแหน่งที่อยู่ของผู้ให้บริการ เพื่อให้ผู้ใช้บริการทำการติดต่อไปยังผู้ให้บริการได้โดยตรง เมื่อผู้ใช้บริการติดต่อไปยังเว็บเซอร์วิซของผู้ให้บริการ เว็บเซอร์วิซจะปฏิบัติตามคำร้องขอและส่งผลลัพธ์กลับมายังผู้ให้บริการตามรูปแบบที่ระบุไว้ในเอกสารวิสเดิล แสดงดังภาพที่ 2.1



ภาพที่ 2.1 การทำงานของเว็บเซอร์วิซ [9]

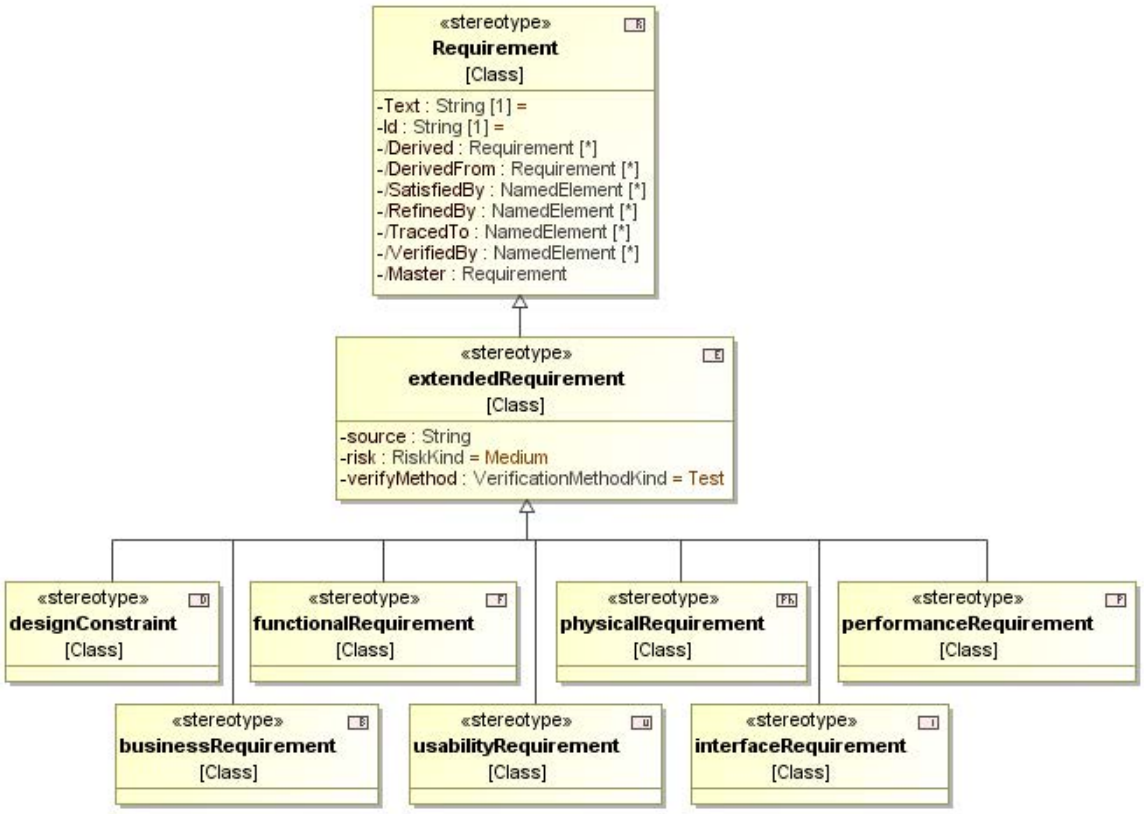
2.2 ยูเอ็มแอลโปรไฟล์

ยูเอ็มแอล (Unified Modeling Language: UML) [3, 10, 11] ถูกพัฒนาโดยโอเอ็มจี (Object Management Group: OMG) คือภาษาที่ใช้ในการสร้างแบบจำลองของการวิเคราะห์และออกแบบระบบ โดยปัจจุบันยูเอ็มแอลรุ่น 2.4 เป็นรุ่นที่ใหม่ล่าสุด ยูเอ็มแอลจำแนกได้เป็น 3 ส่วนหลักๆ คือ (1) ส่วนของโครงสร้าง (Structure) เป็นส่วนที่ทำการอธิบายถึงโครงสร้างคงที่ของระบบ เช่น แบบจำลองคลาสของยูเอ็มแอล (UML Class Diagram) (2) ส่วนของพฤติกรรม (Behavior) เป็นส่วนที่อธิบายโครงสร้างพฤติกรรมของระบบ เช่น แบบจำลองแอคทิวิตีของยูเอ็มแอล (UML Activity Diagram) (3) ส่วนเพิ่มเติม (Supplement) อธิบายการสร้าง Auxiliary และ โปรไฟล์ที่ปรับแต่งเพื่อให้เหมาะสมกับโดเมน หรือแพลตฟอร์มต่างๆ โดยผู้ใช้งานไม่จำเป็นต้องเข้าใจการทำงานของยูเอ็มแอลทั้งหมด ผู้ใช้งานสามารถเลือกที่จะเข้าใจเฉพาะส่วนที่อธิบายตามความต้องการของผู้ใช้งานได้

โปรไฟล์ (Profile) คือ แพคเกจที่ใช้เก็บรวบรวมสเตอรีโอไทพ์ (Stereotype) และเมตาคลาส (Metaclass) ซึ่งสร้างจากเมตาโมเดล (Metamodel) ซึ่งโปรไฟล์ที่ใช้สำหรับการออกแบบบนมาตรฐานของยูเอ็มแอล เรียกว่า ยูเอ็มแอลโปรไฟล์

ยูเอ็มแอลโปรไฟล์ (UML Profile) คือ กระบวนการของยูเอ็มแอลที่ใช้ในการออกแบบระบบโดยเก็บรวบรวมเมตาคลาสไว้ในโปรไฟล์ เพื่อให้ผู้สร้างแบบจำลองสามารถนำเมตาคลาสไปใช้งานหรือสร้างโปรไฟล์ใหม่ ซึ่งประกอบด้วยสเตอรีโอไทพ์ที่ขยายจากยูเอ็มแอลโปรไฟล์อีกต่อหนึ่ง ตัวอย่างของโปรไฟล์ที่สร้างขึ้นมาจากเมตาโมเดลของยูเอ็มแอลโปรไฟล์ เช่น ซิสเอ็มแอลโปรไฟล์ (SysML Profile) คือโปรไฟล์สำหรับภาษาที่ใช้ในการสร้างแบบจำลองวิศวกรรมระบบ (System Engineering) หรือบีพีเอ็มเอ็นโปรไฟล์ (BPMN Profile) คือโปรไฟล์สำหรับภาษาที่ใช้ในการสร้างแบบจำลองกระบวนการทางธุรกิจ (Business Process Modeling) ตัวอย่างการนำยูเอ็มแอลโปรไฟล์ไปใช้ในโดเมนที่แตกต่าง

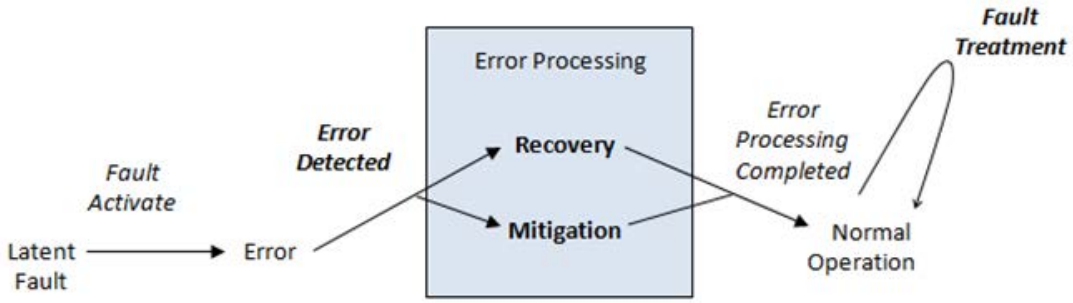
กัน เช่น มาร์เต (MARTE) ซึ่งนำไปใช้งานในระบบการทำงานแบบทันทีและระบบการทำงานแบบฝังตัว (Real Time and Embedded Systems:RTES) หรือบีพีเอ็มเอ็นโปรไฟล์ซึ่งนำไปใช้ในแบบจำลองกระบวนการทางธุรกิจ ตัวอย่างสเตอรีโอไทพ์ Requirement ของซิสเอ็มแอลโปรไฟล์แสดงดังภาพที่ 2.2



ภาพที่ 2.2 ตัวอย่างสเตอรีโอไทพ์ของซิสเอ็มแอลโปรไฟล์ [12]

2.3 แบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่อง (Patterns for Fault Tolerant Software)

การทนต่อความผิดพร่อง คือ ความสามารถที่ระบบสามารถปฏิบัติงานต่อไปได้แม้ว่าจะมีข้อผิดพร่องเกิดขึ้นในระบบ ความผิดพร่องสามารถเกิดได้จากหลายสาเหตุดังเช่น การที่ระบบทำงานไม่ตรงตามความต้องการของระบบ การออกแบบระบบไม่ถูกต้อง หรือข้อผิดพลาดในขั้นตอนการพัฒนาโค้ด ขั้นตอนของการทนต่อความผิดพร่องประกอบด้วย 4 ขั้นตอน ได้แก่ ขั้นตอนการตรวจหาข้อผิดพลาด (Error Detection) การกู้ระบบจากข้อผิดพลาด (Error Recovery) การบรรเทาข้อผิดพลาด (Error Mitigation) และ การรักษาความผิดพร่อง (Fault Treatment) แสดงดังภาพที่ 2.3



ภาพที่ 2.3 ขั้นตอนการทนต่อความผิดพร่อง [5]

จากภาพที่ 2.3 เมื่อความผิดพร่องที่แฝงอยู่ในระบบ (Latent Fault) ถูกกระตุ้นให้เกิดเป็นข้อผิดพลาด จะมีขั้นตอนการทนต่อความผิดพร่อง ดังนี้

1. การตรวจหาข้อผิดพลาด ซึ่งสามารถทำการตรวจหาได้อย่างเป็นประจำ เช่น การตรวจสอบโดยใช้ค่าการตรวจสอบข้อผิดพลาด (Checksum) หรือมีส่วนหนึ่งของระบบทำหน้าที่ตรวจหาข้อผิดพลาดโดยเฉพาะ
2. การกู้ระบบจากข้อผิดพลาดหรือการบรรเทาข้อผิดพลาด ซึ่งขั้นตอนของการกู้ระบบจากข้อผิดพลาด คือการแทนที่ระบบที่เกิดข้อผิดพร่องด้วยระบบที่ไม่มีข้อผิดพลาด และขั้นตอนที่สามารถกำจัดข้อผิดพลาด หรือบรรเทาข้อผิดพลาดลง โดยที่การทำงานของระบบยังเป็นเหมือนเดิม เช่นเมื่อข้อผิดพลาดของค่าข้อมูลถูกแก้ไข ระบบก็จะสามารถทำงานต่อได้ตามปกติ
3. การรักษาความผิดพร่อง เป็นขั้นตอนที่สามารถรักษาความผิดพร่องภายในระบบโดยการทำการอัพเดทระบบหรือการติดตั้งส่วนเสริมไปยังระบบ

จากขั้นตอนการทำให้ทนต่อความผิดพร่อง Hanmer [5] นำเสนอแบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่อง จำนวน 63 แบบรูป โดยแบ่งออกเป็น 5 กลุ่ม ได้แก่ แบบรูปเชิงสถาปัตยกรรม (Architectural Patterns) แบบรูปการตรวจหาข้อผิดพลาด (Detection Patterns) แบบรูปการกู้ระบบจากข้อผิดพลาด (Error Recovery Patterns) แบบรูปการบรรเทาข้อผิดพลาด (Error Mitigation Patterns) และ แบบรูปการรักษาความผิดพร่อง (Fault Treatment Patterns)

โดยที่รายละเอียดของแบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่อง มีดังต่อไปนี้

- แบบรูปเชิงสถาปัตยกรรม (Architectural Patterns) เป็นกลุ่มของแบบรูปอย่างแรกที่ใช้ในการออกแบบระบบให้รองรับการทนต่อความผิดพร่อง และเป็นพื้นฐานสำหรับแบบรูปอื่น ๆ ต่อไป แบบรูปในกลุ่มนี้ มีจำนวน 11 แบบรูป

- แบบรูปการตรวจหาข้อผิดพลาด (Detection Patterns) เป็นกลุ่มของแบบรูปที่ทำให้ระบบมีความทนต่อความผิดพร่องในขั้นแรก โดยมีการตรวจหาสัญญาณของข้อผิดพลาดที่จะเกิดขึ้น ซึ่งอาจจะใช้ความรู้ก่อนหน้ามาเป็นเกณฑ์ในการตรวจสอบ หรือนำผลลัพธ์จากการทำงานของตัวสำรองมาเปรียบเทียบกัน เพื่อหาว่าระบบมีส่วนที่ทำงานผิดพลาดหรือไม่ แบบรูปในกลุ่มนี้มีจำนวน 16 แบบรูป
- แบบรูปการกู้ระบบจากข้อผิดพลาด (Error Recovery Patterns) เป็นกลุ่มของแบบรูปที่ทำการจัดการกับข้อผิดพลาดที่เกิดขึ้น มีจุดประสงค์ให้ระบบสามารถทำงานต่อไปได้ถึงแม้ระบบจะเกิดข้อผิดพลาด โดยสถานะการทำงานของระบบจะถูกเปลี่ยนแปลงไปยังสถานะที่ไม่มีข้อผิดพลาด แบบรูปในกลุ่มนี้มีจำนวน 14 แบบรูป
- แบบรูปการบรรเทาข้อผิดพลาด (Error Mitigation Patterns) เป็นกลุ่มของแบบรูปของการทำงานที่จัดการกับข้อผิดพลาดที่เกิดขึ้นให้บรรเทาความรุนแรงลง แบบรูปในกลุ่มนี้แตกต่างจากแบบรูปการกู้ระบบจากข้อผิดพลาดคือพยายามลดผลกระทบที่เกิดขึ้น โดยที่ระบบจะยังคงทำงานตามสถานะเดิม แบบรูปในกลุ่มนี้มีจำนวน 16 แบบรูป
- แบบรูปการรักษาข้อผิดพร่อง (Fault Treatment Patterns) เป็นกลุ่มของแบบรูปที่เมื่อตรวจพบข้อผิดพลาด จะทำการหาความผิดพร่อง ซึ่งเป็นสาเหตุของข้อผิดพลาด และทำการแก้ไขไม่ให้เกิดข้อผิดพลาดนั้นขึ้นอีก เพื่อให้ระบบทำงานต่อไปได้ เช่น การติดตั้งส่วนเสริมไปยังระบบเพื่อเป็นการแก้ไขข้อผิดพลาดที่เกิดขึ้น แบบรูปในกลุ่มนี้มีจำนวน 6 แบบรูป

2.4 งานวิจัยที่เกี่ยวข้อง

สามารถแบ่งกลุ่มของงานวิจัยที่เกี่ยวข้องได้เป็น 4 กลุ่ม ได้แก่

- การออกแบบยูเอ็มแอลโปรไฟล์ของระบบให้มีการทนต่อความผิดพร่องและประยุกต์ใช้กับโดเมนต่างๆ
- ข้อกำหนดของแบบรูปการทนต่อความผิดพร่อง
- การแนะนำแบบรูปการทนต่อความผิดพร่องสำหรับระบบอิงบริการ
- การประยุกต์ใช้งานแบบรูปการทนต่อความผิดพร่อง

2.4.1 งานวิจัยที่เกี่ยวข้องในการออกแบบยูเอ็มแอลโปรไฟล์ของระบบให้มีการทนต่อความผิดพร่องและประยุกต์ใช้กับโดเมนต่างๆ

งานวิจัยของ Bernedi และ Merseguer [13] ได้นำเสนอยูเอ็มแอลโปรไฟล์ของการทนต่อความผิดพร่องสำหรับการสร้างแบบจำลองและการวิเคราะห์ของระบบการทำงานแบบทันทีและระบบ

ฝังตัว ซึ่งเรียกว่า “Modeling and Analysis of Real-Time and Embedded Systems” หรือ “MARTE” การทนต่อความผิดพร่องของระบบ MARTE จะดูได้จากการวิเคราะห์ความสามารถในการพึ่งพาได้ (Dependability Analysis: DA) ปัจจัยที่มีผลต่อ DA ได้แก่ สภาพพร้อมใช้งาน (Availability) ความเชื่อถือได้ (Reliability) ความปลอดภัย (Safety) บูรณภาพของข้อมูล (Integrity) และ ความสามารถในการบำรุงรักษา (Maintainability) Bernedi และ Merseguer ได้นิยามปัจจัยทั้งหมดนี้ว่าเป็น การทนต่อความผิดพร่อง และ การพยากรณ์ความผิดพร่อง (Fault Forecasting) และทำการออกแบบยูเอ็มแอลโปรไฟล์โดยอ้างอิงจากงานวิจัยอื่นๆ โดยที่นำมาประยุกต์ใช้กับกรณีศึกษาระบบควบคุมกังหันลม

งานวิจัยของ Shim, Baek, Kim และ Park [4] ได้นำเสนอการออกแบบระบบให้มีการทนต่อความผิดพร่องสำหรับโดเมนที่เกี่ยวกับหุ่นยนต์ และทำการวิเคราะห์ประเภทของการทนต่อความผิดพร่องสำหรับหุ่นยนต์ โดยพิจารณาในกระบวนการของการตรวจหาข้อผิดพลาดและการกู้ระบบจากข้อผิดพลาด เพื่อหาความสัมพันธ์กับแบบรูปการทนต่อความผิดพร่องของ Hanmer และนำการออกแบบระบบที่ได้นำเสนอมาประยุกต์ใช้กับกรณีศึกษา OPRoS (Open Platform for Robotic Service) โดยที่การออกแบบเพื่อให้การทำงานของหุ่นยนต์มีการทนต่อความผิดพร่อง จะทำให้ระบบมีความเชื่อถือได้มากขึ้น

งานวิจัยของ Tucci-Piergiovanni, Mraidha, Wozniak, Lanusse, และ Gerard [14] นำเสนอยูเอ็มแอลสำหรับระบบ Automotive Open System Architecture (AUTOSAR) ซึ่งเป็นระบบสำหรับควบคุมยานพาหนะ ให้ทนต่อความผิดพร่อง โดยพิจารณาเฉพาะแบบรูปการทำ Redundancy หรือที่เรียกว่า Software Replication โดยมี 3 รูปแบบหลักๆ ได้แก่ (1) Active Replication (2) Passive Replication (3) Semi-active Replication และนำมาใช้ในกรณีศึกษา Cruise Control System (CCS) ซึ่งเป็นระบบควบคุมความเร็วของรถยนต์ ซึ่งทำการออกแบบทั้งในส่วนของฮาร์ดแวร์ ซอฟต์แวร์ และการส่งข้อมูลในระบบ

จากงานวิจัยดังกล่าวทำให้ได้แนวคิดเกี่ยวกับการสร้างยูเอ็มแอลโปรไฟล์สำหรับแบบรูปการทนต่อความผิดพร่อง โดยแต่ละงานวิจัยจะนำเสนอเฉพาะโดเมนที่เกี่ยวข้องกับยูเอ็มแอลโปรไฟล์ที่นำเสนอในแต่ละงานวิจัยเท่านั้น ซึ่งผู้วิจัยต้องการนำเสนอยูเอ็มแอลโปรไฟล์สำหรับแบบรูปการทนต่อความผิดพร่องสำหรับระบบอิงบริการ โดยที่ยูเอ็มแอลโปรไฟล์สำหรับแบบรูปทนต่อความผิดพร่องสำหรับระบบอิงบริการนั้นสามารถที่จะนำไปใช้กับระบบอิงบริการใดๆก็ได้ เช่น ระบบอิงบริการสำหรับการชำระเงินผ่านธนาคาร หรือระบบอิงบริการสำหรับจองโรงแรม

2.4.2 ข้อกำหนดของแบบรูปการทนต่อความผิดพร่อง

โอเอ็มจี (OMG) ได้นำเสนอข้อกำหนดที่ชื่อว่า “UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms” [2] โดยที่ส่วนของการทนต่อความผิดพร่องได้นำเสนอในหัวข้อที่ชื่อว่า “FT Mitigation Solutions” ซึ่งเป็นการนำเสนอยูเอ็มแอลโปรไฟล์สำหรับการทนต่อความผิดพร่อง ในข้อกำหนดนี้กล่าวถึงการตรวจหาความผิดพร่อง และการบรรเทาข้อผิดพลาด โปรไฟล์หลักๆมีดังต่อไปนี้

- ServerObjectGroup คือส่วนของระบบที่ต้องการทำให้ทนต่อความผิดพร่อง พร้อมทั้งคุณสมบัติต่างๆที่จำเป็นถ้าต้องการทำให้ระบบมีการทนต่อความผิดพร่อง
- FaultDetectionDeploymentPolicy เป็นการระบุนโยบายของการตรวจหาความผิดพร่องในระบบ
- ReplicationStyle เป็นส่วนที่สนับสนุนการทำซ้ำเมื่อมีความผิดพร่องเกิดขึ้น และกล่าวถึงคุณสมบัติที่จำเป็นในการทำสำเนา และประเภทของการทำสำเนาแบบต่างๆ

จากข้อกำหนดดังกล่าวจะถูกนำมาใช้เป็นแนวทางในการออกแบบยูเอ็มแอลโปรไฟล์สำหรับแบบรูปการทนต่อความผิดพร่องสำหรับระบบอิงบริการของผู้วิจัย โดยที่ยูเอ็มแอลโปรไฟล์ที่นำเสนอในงานวิจัยนี้จะมีโครงสร้างไปในแนวทางเดียวกับข้อกำหนดดังที่กล่าวมา แต่ผู้วิจัยจะนำเสนอยูเอ็มแอลโปรไฟล์ที่ครอบคลุมแบบรูปของความผิดพร่องมากขึ้น

จากงานวิจัยที่เกี่ยวข้องในข้อที่ 2.4.1 และ 2.4.2 ซึ่งนำเสนอยูเอ็มแอลโปรไฟล์สำหรับแบบรูปการทนต่อความผิดพร่อง สามารถเปรียบเทียบระหว่างแบบรูปที่มีการนำเสนอในแต่ละงานวิจัย รวมทั้งแบบรูปที่จะนำเสนอต่อไปในวิทยานิพนธ์นี้ โดยจัดกลุ่มตามแบบรูปการทนต่อความผิดพร่องของ Hanmer [5] และแสดงจำนวนของแบบรูปที่นำเสนอในแต่ละกลุ่ม ดังตารางที่ 2.1

ตารางที่ 2.1 การเปรียบเทียบแบบรูปการทนต่อความผิดพร่องสำหรับแต่ละงานวิจัย

งานวิจัยที่เกี่ยวข้อง	รายละเอียดแบบรูปการทนต่อความผิดพร่อง
Bernedi และ Merseguer [13]	**แบบรูปการตรวจหา 2 แบบรูป:** Error Detection, Watchdog **แบบรูปการกู้ระบบจากข้อผิดพลาด 2 แบบรูป:** Error Recovery, Repair
Shim และคณะ [4]	**แบบรูปการตรวจหา 9 แบบรูป:** Fault Correlation, System Monitor, Ping/Echo, Heartbeat, Watchdog, Routine Maintenance, Realistic Threshold, Riding Over Transients, Leaky bucket counter **แบบรูปการกู้ระบบจากข้อผิดพลาด 8 แบบรูป:** Rollback, Roll-Forward, Restart, Failover, Data Reset, N-Version, Limit Retries, Checkpoint
Tucci-Piergiovanni และ คณะ [14]	**แบบรูปเชิงสถาปัตยกรรม 3 แบบรูป:** ActiveReplica, PassiveReplica, SemiActiveReplica
OMG [2]	**แบบรูปเชิงสถาปัตยกรรม 5 แบบรูป:** StatelessReplicationStyle, WarmPassiveReplicationStyle, ColdPassiveReplicationStyle, ActiveReplicationStyle, ActiveWithVotingReplicationStyle **แบบรูปการตรวจหา 6 แบบรูป:** PullMonitoringStyle, PushMonitoringStyle, ApplicationMonitoringStyle, IndividualMonitoringStyle, LocationMonitoring, LocationAndTypeMonitoring
งานวิจัยนี้	**แบบรูปเชิงสถาปัตยกรรม 6 แบบรูป:** Units of Mitigation, Redundancy, Recovery Blocks, Someone in Charge, Escalation, Fault Observer **แบบรูปการตรวจหา 5 แบบรูป:** System Monitor, Heartbeat, Acknowledgement, Watchdog, Voting **แบบรูปการกู้ระบบจากข้อผิดพลาด 11 แบบรูป:** Error Handler, Restart, Rollback, Roll-Forward, Return to Reference Point, Limit Retries, Failover, Checkpoint, What to Save, Remote Storage, Individuals Decide Timing

2.4.3 งานวิจัยที่เกี่ยวกับการแนะนำแบบรูปการทนต่อความผิดพร่องสำหรับระบบอิงบริการ

Zheng และ Lyu [15] นำเสนอประเภทของแบบรูปการทำซ้ำ (Redundancy) โดยมี 2 ประเภท ได้แก่ แบบรูปการทำซ้ำด้านเวลา (Time Redundancy) และแบบรูปการทำซ้ำด้านพื้นที่ (Space Redundancy) แบบรูปการทำซ้ำด้านเวลาเป็นแบบรูปที่จะใช้เวลาเพิ่มขึ้นในการทำซ้ำเพื่อให้ทนต่อความผิดพร่อง เช่น เวลาในการคำนวณหรือเวลาในการติดต่อสื่อสาร และแบบรูปการทำซ้ำด้านพื้นที่เป็นแบบรูปที่จะใช้พื้นที่เพิ่มขึ้นเพื่อเป็นสำเนาเพิ่มในการทำซ้ำเพื่อให้ทนต่อความผิดพร่อง ซึ่งพื้นที่จะขึ้นอยู่กับทรัพยากรที่มีไม่ว่าจะเป็น ฮาร์ดแวร์ หรือซอฟต์แวร์ โดยที่การทำซ้ำด้านพื้นที่ประกอบด้วย แบบ Active และแบบ Passive ซึ่งแบบ Active สำเนาจะถูกเรียกใช้งานพร้อมกันแบบขนาน และแบบ Passive สำเนาจะถูกเรียกใช้งานทีละตัวตามลำดับ

จากงานวิจัยข้างต้นและงานวิจัยอื่นๆที่เกี่ยวกับการแนะนำแบบรูปการทนต่อความผิดพร่องสำหรับระบบอิงบริการ ผู้วิจัยได้นำแบบรูปจากงานวิจัยเหล่านั้นมาเพิ่มเติมกับแบบรูปการทนต่อความผิดพร่องของ Hanmer [5] เพื่อนำมาสนับสนุนการออกแบบยูเอ็มแอลโปรไฟล์สำหรับการทนต่อความผิดพร่องสำหรับระบบอิงบริการ

2.4.4 งานวิจัยที่ประยุกต์ใช้งานแบบรูปการทนต่อความผิดพร่อง

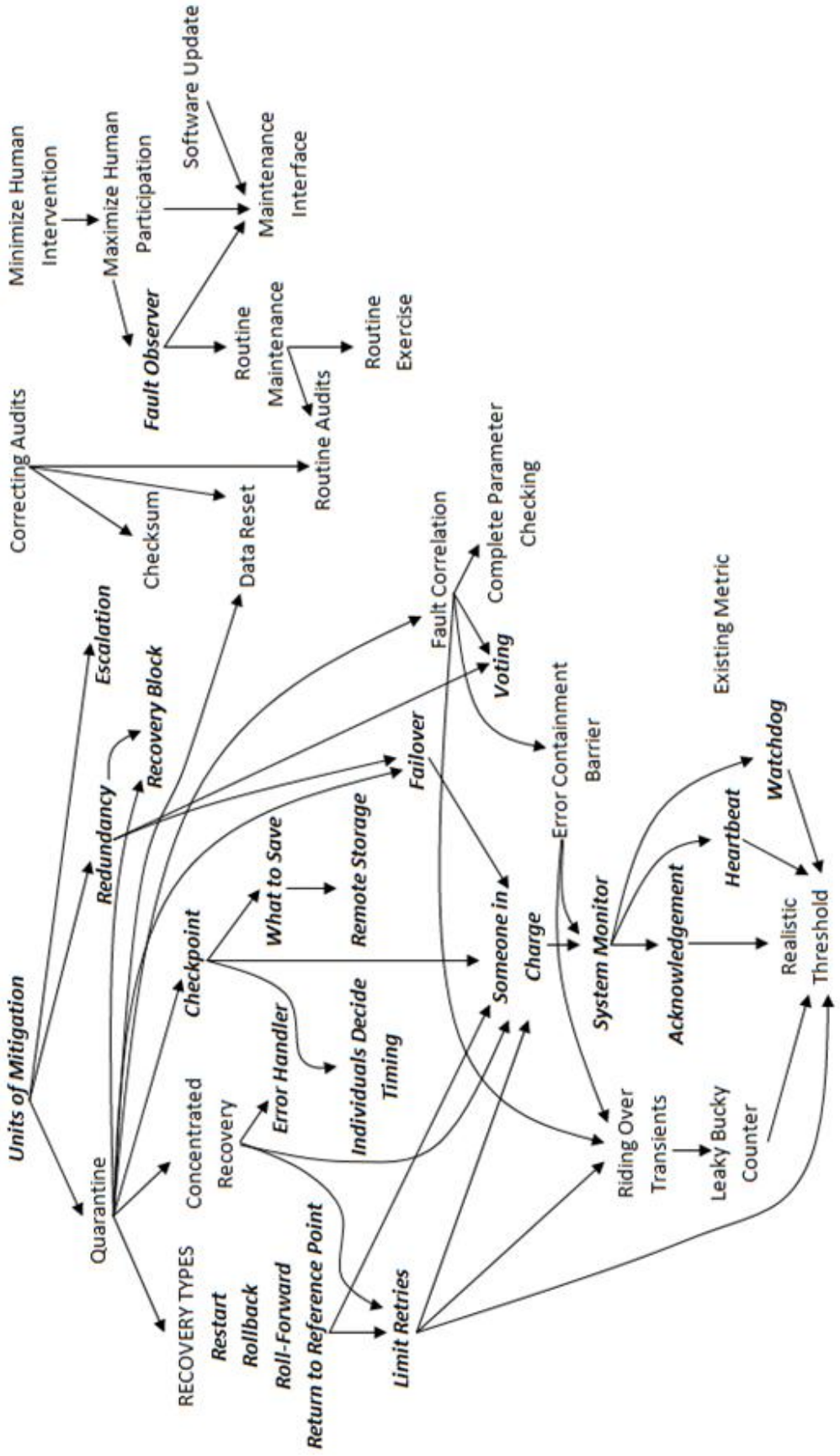
Thawee Thaisongsuwan และ Twittie Senivongse [16] นำเสนอการประยุกต์ใช้แบบรูปการทนต่อความผิดพร่องของ Hanmer มาปรับใช้กับการออกแบบกระบวนการทำงานด้วยภาษาดับเบิลยูเอส-บีเพล (WS-BPEL) ซึ่งเป็นภาษามาตรฐานที่ใช้ในการกำหนดกระแสงานของการเรียกใช้เว็บเซอร์วิซต่างๆ ตามกระบวนการทางธุรกิจ ให้ทนต่อความผิดพร่อง โดยจะบอกถึงโครงสร้างบีเพลที่ใช้หรือแนวทางการพัฒนาตามแบบรูปต่าง ๆ

บทที่ 3

การศึกษาและวิเคราะห์แบบรูป

ในการศึกษาและวิเคราะห์แบบรูปเพื่อนำไปออกแบบยูเอ็มแอลโปรไฟล์ โดยพิจารณาแบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่อง [5] ทั้งหมด 63 แบบรูป ซึ่งแบ่งออกเป็น 5 กลุ่ม ได้แก่ แบบรูปเชิงสถาปัตยกรรม แบบรูปการตรวจหา แบบรูปการกู้ระบบจากข้อผิดพลาด แบบรูปการบรรเทาข้อผิดพลาด และแบบรูปการรักษาความผิดพร่อง

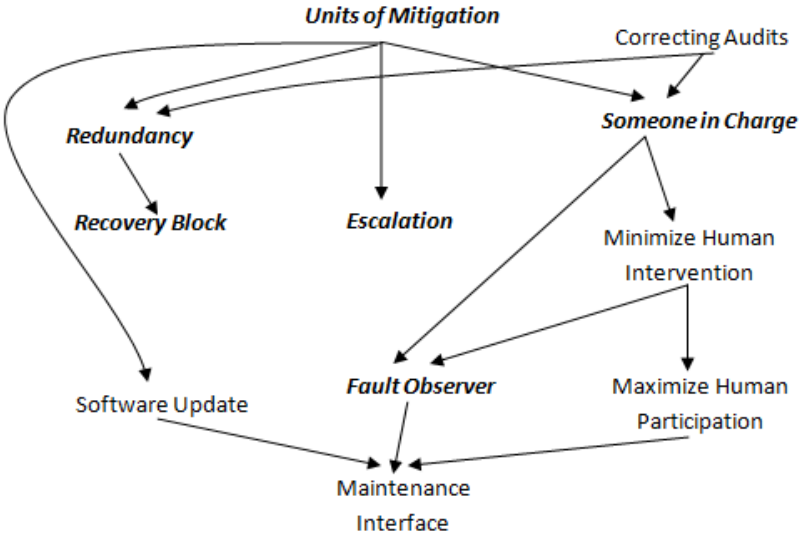
งานวิจัยนี้จะพิจารณาแบบรูปใน 3 กลุ่มแรกเท่านั้น ในการศึกษาจะทำการศึกษารายละเอียดแบบรูปในส่วนของ การทำงาน คุณสมบัติ และพิจารณาถึงความสัมพันธ์ของแบบรูป ซึ่งสามารถพิจารณาได้จากแผนภาพความสัมพันธ์ของแบบรูปใน 3 กลุ่มแรกดังภาพที่ 3.1 จากนั้นทำการวิเคราะห์ว่าแบบรูปนั้นสามารถนำมาสร้างโปรไฟล์ได้หรือไม่ โดยจะมีการนำเสนอยูเอ็มแอลโปรไฟล์สำหรับแบบรูปการทนต่อความผิดพร่องต่างๆ ไว้ดังแสดงในบทที่ 4 ต่อไป



ภาพที่ 3.1 แผนภาพแสดงความสัมพันธ์ของแบบรูปใน 3 กลุ่มแรก [5]

3.1 แบบรูปเชิงสถาปัตยกรรม (Architectural Patterns)

เป็นกลุ่มของแบบรูปอย่างแรกที่ใช้ในการออกแบบระบบให้รองรับการทนต่อความผิดพร่อง ซึ่งเป็นการมองภาพรวมของระบบเพื่อจะนำไปออกแบบ และเป็นพื้นฐานสำหรับแบบรูปอื่น ๆ ต่อไป โดยจะแสดงความหมาย การใช้งานของแบบรูป ความสัมพันธ์กับแบบรูปอื่น โดยพิจารณาเฉพาะความสัมพันธ์ภายในแบบรูปเชิงสถาปัตยกรรมดังแสดงในภาพที่ 3.2 และอธิบายว่าสามารถนำแบบรูปใดไปออกแบบยูเอ็มแอลโปรไฟล์ได้บ้าง (แสดงด้วยตัวหนาในภาพ) รายละเอียดการพิจารณามีดังนี้



ภาพที่ 3.2 แผนภาพแสดงความสัมพันธ์ภายในแบบรูปเชิงสถาปัตยกรรม [5]

3.1.1 Units of Mitigation

แบบรูปนี้มีจุดประสงค์เพื่อให้ระบบสามารถทำงานต่อไปได้เมื่อมีความผิดพร่องเกิดขึ้น จึงควรออกแบบระบบให้แบ่งเป็นหน่วยย่อย เมื่อมีความผิดพร่องเกิดขึ้นจะสามารถจัดการกับความผิดพร่องนั้นได้ภายในหน่วยย่อยหน่วยเดียวเท่านั้น และส่วนอื่นๆยังสามารถทำงานได้ตามปกติ

แบบรูปนี้สามารถนำไปใช้ออกแบบเนื่องจากสามารถกำหนดส่วนต่างๆในระบบให้เป็นหน่วยย่อยได้

3.1.2 Correcting Audits

ข้อมูลที่มีข้อผิดพลาดจะนำไปสู่การทำงานที่ผิดพลาดในอนาคต และบริบทของข้อมูลต้องมีการพิจารณา เช่น 1974 อาจจะถูกต้องเมื่ออยู่ในบริบทของปี แต่ไม่ถูกต้องเมื่ออยู่ในบริบทของอายุ ดังนั้นจึงต้องมีการตรวจสอบว่าข้อมูลมีข้อผิดพลาดเกิดขึ้นหรือไม่ หรืออาจมีการบันทึก

ข้อผิดพลาดที่เกิดขึ้นไว้เพื่อนำไปตรวจสอบในอนาคต การตรวจสอบข้อผิดพลาดอาจจะใช้ข้อมูลอื่นเป็นตัวช่วยในการตรวจสอบ ตัวอย่างการตรวจสอบข้อผิดพลาดของข้อมูล ได้แก่

- ตรวจสอบคุณสมบัติโครงสร้าง (Check Structural Properties) – เป็นการตรวจสอบความถูกต้องของข้อมูลในลิงค์ สแต็ก หรือ คิว ว่าข้อมูลเหล่านั้นถูกต้องตามโครงสร้างของข้อมูล เช่น ค่าจำนวนข้อมูลในลิงค์จะต้องเป็นจำนวนของข้อมูลที่มีอยู่จริงในลิงค์
- ทราบถึงความสัมพันธ์ของข้อมูล (Known Correlation) – ข้อมูลที่เหมือนกันหรือมีความเกี่ยวข้องกันถูกเก็บอยู่ในที่ที่แตกต่างกัน ดังนั้นจึงต้องทราบถึงการแปลงข้อมูลระหว่างค่าข้อมูลต่างๆ เช่น การแปลงอุณหภูมิจากหน่วยองศาเซลเซียสไปเป็นหน่วยฟาเรนไฮต์
- ตรวจสอบขอบเขตของข้อมูล (Sanity Checks) – ค่าข้อมูลจะต้องอยู่ในขอบเขตที่เป็นไปได้
- การเปรียบเทียบข้อมูลโดยตรง (Direct Comparison) – มีการเก็บข้อมูลเดียวกันไว้ในหลายที่ เพื่อใช้สำหรับนำมาตรวจสอบซึ่งกันและกัน

แบบรูปนี้กล่าวถึงการตรวจสอบความถูกต้องของข้อมูลในรูปแบบต่างๆ เนื่องจากการตรวจสอบความถูกต้องโดยทั่วไปมีการทำการตรวจสอบในกระบวนการทำงานของเซอร์วิซอยู่แล้วจึงไม่นำแบบรูปนี้มาออกแบบเป็นยูเอ็มแอลโปรไฟล์

3.1.3 Redundancy

การทำซ้ำคือการที่ระบบมีการจัดเตรียมตัวสำรองเพื่อทำงานแทนเมื่อระบบเกิดข้อผิดพลาดเพื่อเพิ่มสภาพพร้อมใช้งานให้แก่ระบบ

การนำตัวสำรองมาประยุกต์ใช้มี 3 ประเภท ได้แก่

- ตัวสำรองเชิงพื้นที่ (Spatial Redundancy) ระบบมีการทำตัวสำรองหลายตัว ซึ่งสำเนาไว้คนละสถานที่ เพื่อให้ตัวสำรองทำงานแทนเมื่ออีกตัวสำรองหนึ่งเกิดข้อผิดพลาด ตัวสำรองเหล่านี้จะมีการทำงานเหมือนกัน

  แบบรูป *Active Replication* จะเป็นชนิดหนึ่งของตัวสำรองเชิงพื้นที่ โดยที่ทุกๆตัวสำรองทำงานพร้อมกัน และมีการใช้แบบรูป *Voting* มาช่วยในการเลือกคำตอบที่เหมาะสม และแบบรูปอีกประเภทหนึ่งของตัวสำรองเชิงพื้นที่ คือ แบบรูป *Passive Replication* หรือ *Recovery Blocks* ซึ่งแบบรูปนี้จะมีทั้งคุณสมบัติของตัวสำรองเชิง

พื้นที่และตัวสำรองเชิงเวลา (Temporal Redundancy) ด้วย เนื่องจากตัวสำรองจะทำงานหลังจากที่ตัวสำรองอื่นทำงานแล้วเกิดข้อผิดพลาด

- ตัวสำรองเชิงเวลา (Temporal Redundancy) ระบบจะมีการทำงานของตัวสำรองที่เป็นลำดับต่อเนื่องกัน แบบรูปที่มีลักษณะของตัวสำรองเชิงเวลา คือ แบบรูป *Limit Retries*
- ตัวสำรองเชิงสารสนเทศ (Informational Redundancy) ระบบจะมีการทำสำเนาของข้อมูลหลายเวอร์ชัน โดยที่สำเนาข้อมูลจะเก็บในที่เก็บที่แตกต่างกันเพื่อใช้สำหรับการตรวจสอบซึ่งกันและกัน ตัวอย่างของแบบรูปคือ แบบรูป *Correcting audits*

จากแบบรูป *Redundancy* จะนำมาพิจารณาเฉพาะตัวสำรองเชิงพื้นที่และตัวสำรองเชิงเวลาเท่านั้น เนื่องจากตัวสำรองใน 2 รูปแบบนั้นสามารถนำมาออกแบบเป็นยูเอ็มแอลโปรไฟล์และนำไปประยุกต์ใช้ได้อย่างชัดเจน

3.1.4 Recovery Blocks

เมื่อระบบมีการใช้งานตัวสำรองแล้ว แบบรูปที่สามารถนำมาประยุกต์ใช้ได้คือแบบรูป *Recovery Blocks* โดยแบบรูปนี้จะประกอบด้วยส่วนปฐมภูมิ (Primary Block) และส่วนทุติยภูมิ (Secondary Block) ซึ่งส่วนปฐมภูมิจะเริ่มทำงานก่อน และเมื่อมีข้อผิดพลาดเกิดขึ้นในระบบจะต้องเปลี่ยนไปทำงานยังส่วนทุติยภูมิ ถ้าหากการทำงานในส่วนทุติยภูมิยังไม่สำเร็จจะไปทำยังส่วนทุติยภูมิส่วนอื่นถัดไป โดยจะมีการใช้แบบรูป *Checkpoint* มาช่วยในการเก็บข้อมูลเพื่อให้สามารถไปทำงานยังส่วนทุติยภูมิได้ในสถานะเดิม

จากแบบรูป *Recovery Blocks* สามารถนำไปออกแบบยูเอ็มแอลโปรไฟล์ได้ เพื่อสามารถนำไปประยุกต์ใช้กับการออกแบบว่าให้ส่วนใดทำงานเป็นส่วนปฐมภูมิหรือทุติยภูมิตามลำดับ

3.1.5 Minimize Human Intervention

ข้อผิดพลาดนั้นอาจจะเกิดขึ้นจาก ฮาร์ดแวร์ ซอฟต์แวร์หรือ กระบวนการทำงานของมนุษย์ การทำงานของคอมพิวเตอร์จะทำงานได้ดีกว่ามนุษย์เมื่อเป็นการทำงานที่เป็นแบบประจำ ดังนั้นควรออกแบบให้ระบบมีการประมวลผลและแก้ปัญหาแบบอัตโนมัติ เนื่องจากมนุษย์จะมีการตอบสนองได้ช้ากว่าคอมพิวเตอร์และเกิดข้อผิดพลาดได้ง่าย

จากแบบรูป *Minimize Human Intervention* เป็นการนำเสนอคำแนะนำของการออกแบบระบบ ดังนั้นไม่สามารถนำไปออกแบบเป็นยูเอ็มแอลโปรไฟล์ได้

3.1.6 Maximize Human Participation

ถึงแม้ว่าระบบจะออกแบบตามแบบรูป *Minimize Human Intervention* แต่ระบบสามารถเกิดความผิดพร่องขึ้นได้ จึงต้องมีการออกแบบระบบให้ผู้เชี่ยวชาญสามารถเข้าไปแก้ปัญหาระบบได้ เนื่องจากผู้เชี่ยวชาญมีความสามารถในการแก้ปัญหาได้ดีกว่า ระบบจึงควรมีการออกแบบให้มีการเฝ้าสังเกตความผิดพร่องที่เกิดได้ตามแบบรูป *Fault Observer* หรือออกแบบให้มีช่องทางสำหรับให้ผู้เชี่ยวชาญสามารถเข้าไปแก้ไขข้อผิดพลาดได้ตามแบบรูป *Maintenance Interface*

จากแบบรูป *Maximize Human Participation* เป็นการนำเสนอคำแนะนำของการออกแบบระบบ เช่นเดียวกับแบบรูป *Minimize Human Intervention* ดังนั้นไม่สามารถนำไปออกแบบเป็นยูเอ็มแอลโปรไฟล์ได้

3.1.7 Maintenance Interface

ระบบต้องออกแบบให้มีส่วนที่ทำการบำรุงรักษาระบบแยกออกจากส่วนการทำงานหลักของระบบ เพื่อเป็นการลดปัญหาที่เกิดระหว่างการบำรุงรักษาระบบ เช่น ปัญหาของภาระงานเกินและช่วยลดระยะเวลาที่ระบบใช้งานไม่ได้ ตัวอย่างของการใช้งานตามแบบรูปนี้คือ แยกหน้าเว็บสำหรับผู้ใช้งานออกจากหน้าเว็บสำหรับผู้ดูแลรักษาระบบ

จากแบบรูป *Maintenance Interface* ได้มีการแนะนำเพื่อการออกแบบส่วนบำรุงรักษาแยกออกจากส่วนหลักของระบบ ซึ่งเป็นการออกแบบที่แยกต่างหากออกมา ไม่ใช่การทำงานหลักของระบบ จึงไม่นำไปออกแบบเป็นยูเอ็มแอลโปรไฟล์

3.1.8 Someone in Charge

เมื่อระบบออกแบบตามแบบรูป *Units of Mitigation* โดยที่ระบบจะมีการแบ่งออกเป็นหน่วยย่อย ต้องกำหนดหนึ่งหน่วยย่อยเพื่อเป็นผู้รับผิดชอบเกี่ยวกับข้อผิดพลาดที่เกิดขึ้นในระบบ โดยที่หน่วยย่อยนั้นจะต้องมีหน้าที่ในการเฝ้าสังเกตหรือควบคุมหน่วยย่อยอื่นๆในระบบว่ามีข้อผิดพลาดเกิดขึ้นหรือไม่ หากมีข้อผิดพลาดเกิดขึ้นหน่วยย่อยนั้นจะต้องทำการจัดการกับข้อผิดพลาดหรือทำการรายงานไปยังหน่วยย่อยอื่นที่ต้องการรับรู้ ตามแบบรูป *Fault Observer*

จากแบบรูป *Someone in Charge* สามารถนำไปสร้างเป็นยูเอ็มแอลโปรไฟล์ เพื่อนำไปประยุกต์เข้ากับส่วนของระบบที่ทำหน้าที่เป็นผู้รับผิดชอบข้อผิดพลาดที่เกิดขึ้นในระบบ

3.1.9 Escalation

เมื่อระบบมีข้อผิดพลาดเกิดขึ้น ระบบต้องทำหน้าที่จัดการกับข้อผิดพลาดนั้นๆ แต่ถ้าวิธีการจัดการกับข้อผิดพลาดยังไม่ประสบความสำเร็จและถ้าหากมีการพยายามทำซ้ำอาจจะไม่ได้ผลเช่นเดิม ดังนั้นต้องมีการจัดการกับข้อผิดพลาดที่เข้มขึ้นเรื่อยๆ ตัวอย่างเช่น การจัดการกับข้อผิดพลาดในลำดับแรกได้มีการทำตามแบบรูป *Limit Retries* คือเป็นการทำงานซ้ำที่หน่วยย่อยนั้น โดยจะทำตามจำนวนครั้งที่ได้กำหนดไว้ตามแบบรูป ถ้าหากยังมีข้อผิดพลาดเกิดขึ้นอีกจะต้องเปลี่ยนไปจัดการกับข้อผิดพลาดนั้นในระดับที่เข้มขึ้นที่ได้ทำการออกแบบไว้ ซึ่งทำตามแบบรูป *Rollback* คือกลับไปทำงานยังจุดก่อนหน้าที่จะเกิดข้อผิดพลาด หากยังไม่สามารถจัดการกับข้อผิดพลาดได้อีก ก็จะเปลี่ยนแปลงไปยังการจัดการที่เข้มงวดขึ้นไปเรื่อยๆ การจัดการกับข้อผิดพลาดควรให้หน่วยที่มีการใช้แบบรูป *Someone in Charge* เป็นผู้จัดการ

จากแบบรูป *Escalation* สามารถนำไปสร้างยูเอ็มแอลโปรไฟล์ที่มีการระบุขั้นตอนการจัดการกับข้อผิดพลาด และสามารถนำไปประประยุกต์ใช้กับหน่วยที่ต้องการให้มีขั้นตอนการจัดการกับข้อผิดพลาด

3.1.10 Fault Observer

ระบบควรมีหน่วยที่ทำหน้าที่สังเกตความผิดพร่องต่างๆที่เกิดขึ้นในระบบ ซึ่งเรียกว่า *Fault Observer* หรือหน่วยสังเกตความผิดพร่อง เมื่อหน่วยสังเกตความผิดพร่องรับรู้ว่ามีความผิดพร่องเกิดขึ้นในระบบ จะทำการรายงานข้อมูลของความผิดพร่องไปยังส่วนที่สนใจหรือส่วนที่รับผิดชอบ โดยที่หน่วยสังเกตความผิดพร่องสามารถเป็นได้ทั้งหน่วยภายในและภายนอกระบบ

จากแบบรูป *Fault Observer* สามารถนำไปสร้างเป็นยูเอ็มแอลโปรไฟล์เพื่อนำไปประยุกต์ใช้เข้ากับหน่วยทั้งที่ทำหน้าที่เป็นหน่วยที่ถูกสังเกต และหน่วยสังเกตความผิดพร่อง และสามารถกำหนดส่วนที่สนใจหรือส่วนที่รับผิดชอบต่อความผิดพร่องนั้น

3.1.11 Software Update

ระบบที่ได้นำมาติดตั้งและใช้งานจริง ย่อมมีโอกาสได้รับการทำการแก้ไขข้อผิดพลาดหรือมีการเพิ่มความสามารถของระบบ ซึ่งในช่วงของการปรับปรุงระบบจำเป็นต้องทำให้ช่วงเวลาที่ระบบไม่สามารถใช้งานได้เป็นช่วงสั้นที่สุด โดยอาจจะดำเนินการตามแบบรูป *Failover* หรือแบบรูป *Recovery Blocks*

จากแบบรูป *Software Update* ไม่สามารถนำมาสร้างเป็นยูเอ็มแอลโปรไฟล์เพื่อใช้ในการออกแบบระบบได้ เนื่องจากเป็นงานที่จะต้องทำโดยมนุษย์หรือมีการทำงานอัตโนมัติอยู่แล้ว

จากรายละเอียดของแบบรูปในกลุ่มของแบบรูปเชิงสถาปัตยกรรม สามารถนำมาสรุปแบบรูปที่จะนำไปพัฒนาเป็นยูเอ็มแอลโปรไฟล์ และระบุถึงจุดประสงค์ของแบบรูป ดังตารางที่ 3.1

ตารางที่ 3.1 การนำแบบรูปเชิงสถาปัตยกรรมไปออกแบบยูเอ็มแอลโปรไฟล์

แบบรูป	**จุดประสงค์ของแบบรูป**	**นำไปสร้างยูเอ็มแอลโปรไฟล์หรือไม่**
Units of Mitigation	แบ่งระบบออกเป็นหน่วยย่อยๆ เพื่อทำให้หน่วยย่อยๆนั้นมีการทนต่อความผิดพร่อง	√
Correcting Audits	ต้องออกแบบให้มีการตรวจสอบความถูกต้องของข้อมูล ถ้าหากพบข้อผิดพลาดต้องทำการตรวจสอบข้อมูลที่เกี่ยวข้องด้วย	× (ตรวจสอบข้อมูล)
Redundancy	ออกแบบระบบให้มีตัวสำรองที่สามารถทำงานแทนเมื่อเกิดข้อผิดพลาด	√
Recovery Blocks	มีการกำหนดตัวสำรองที่จะทำงานแทนเมื่อตัวหลักเกิดข้อผิดพลาด โดยจะกำหนดเป็นลำดับของการทำงานแทนไว้	√
Minimize Human Intervention	ออกแบบระบบให้มีการทำงานแบบอัตโนมัติมากที่สุด เพื่อป้องกันความล่าช้าและข้อผิดพลาดในการทำงานของมนุษย์	× (คำแนะนำ)
Maximize Human Participation	ออกแบบระบบให้รองรับการแก้ปัญหาจากผู้เชี่ยวชาญ เนื่องจากในบางเหตุการณ์มนุษย์สามารถแก้ปัญหาได้ดีกว่าการแก้ปัญหาของเครื่อง	× (คำแนะนำ)
Maintenance Interface	ออกแบบระบบให้มีส่วนต่อประสานสำหรับการบำรุงรักษาระบบ	× (ไม่ใช่ส่วนหลัก)
Someone in Charge	ระบบจะต้องมีหน่วยที่ทำหน้าที่ควบคุมและจัดการระบบเมื่อมีข้อผิดพลาดเกิดขึ้น	√
Escalation	เมื่อการจัดการกับความผิดพร่องไม่สำเร็จ ควรจะเปลี่ยนไปทำวิธีที่เข้มขึ้น	√

ตารางที่ 3.1 การนำแบบรูปเชิงสถาปัตยกรรมไปออกแบบยูเอ็มแอลโปรไฟล์ (ต่อ)

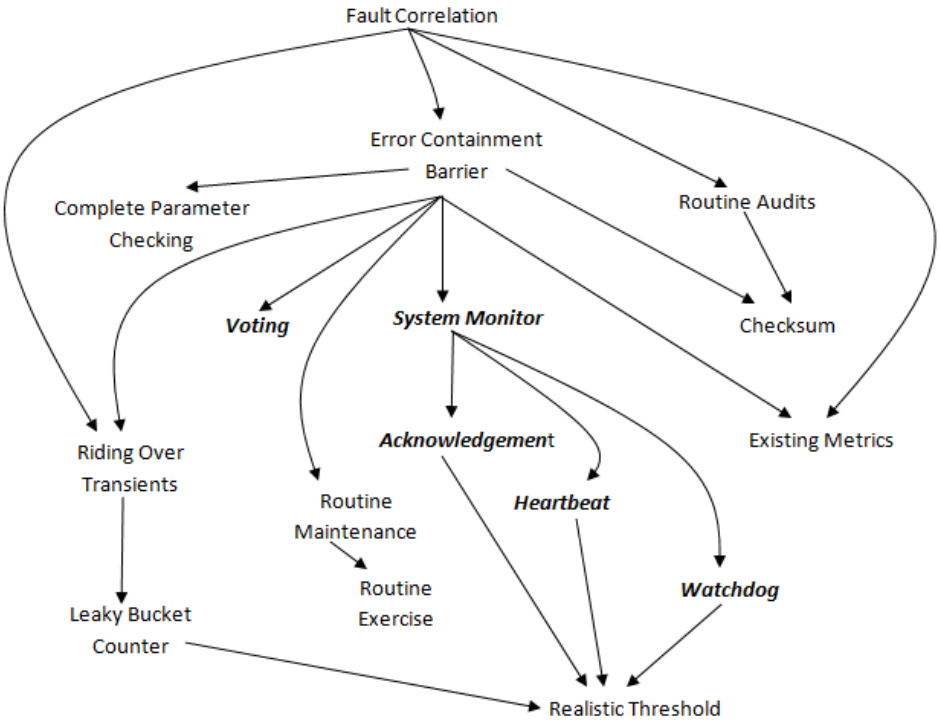
แบบรูป	**จุดประสงค์ของแบบรูป**	**นำไปสร้างยูเอ็มแอลโปรไฟล์หรือไม่**
Fault Observer	เมื่อมีความผิดพร่องเกิดขึ้น จะต้องมีหน่วยที่เฝ้าสังเกตและรายงานไปยังผู้ที่สนใจความผิดพร่องนั้น	√
Software Update	ออกแบบให้ระบบมีการปรับปรุงและแก้ไขระบบให้เป็นปัจจุบันเสมอ	× (ทำโดยมนุษย์)

จากตารางที่ 3.1 รายละเอียดเหตุผลของการไม่นำแบบรูปนั้นๆ ไปสร้างเป็นยูเอ็มแอลโปรไฟล์ มีดังนี้

- ตรวจสอบข้อมูล – แบบรูปนี้จะไม่ถูกนำไปออกแบบเป็นยูเอ็มแอลโปรไฟล์ เนื่องจากมีการกล่าวถึงการตรวจความถูกต้องของข้อมูล ซึ่งควรทำในกระบวนการทำงานมากกว่าขั้นตอนการออกแบบ
- คำแนะนำ – แบบรูปนี้จะไม่ถูกนำไปออกแบบเป็นยูเอ็มแอลโปรไฟล์ เนื่องจากเป็นเพียงการนำเสนอคำแนะนำในการออกแบบระบบ ซึ่งผู้ออกแบบจะนำไปใช้ในการออกแบบหรือไม่ก็ได้
- ไม่ใช่ส่วนหลัก – แบบรูปนี้จะไม่ถูกนำไปออกแบบเป็นยูเอ็มแอลโปรไฟล์ เนื่องจากเป็นการนำเสนอการออกแบบที่ไม่ใช่การทำงานหลักของระบบ
- ทำโดยมนุษย์ – แบบรูปนี้จะไม่ถูกนำไปออกแบบเป็นยูเอ็มแอลโปรไฟล์ เนื่องจากการนำเสนอแบบรูปดังกล่าวเป็นกระบวนการทำงานที่ต้องทำโดยมนุษย์หรือไม่มีการทำงานแบบอัตโนมัติ

3.2 แบบรูปการตรวจหา (Detection Patterns)

เป็นแบบรูปในกลุ่มแรกของขั้นตอนการทนต่อความผิดพร่อง ซึ่งเป็นกลุ่มของแบบรูปที่ทำให้ระบบมีความทนต่อความผิดพร่อง โดยมีการตรวจหาสัญญาณของข้อผิดพลาดที่จะเกิดขึ้น ซึ่งจะแสดงความหมาย การใช้งานของแบบรูป ความสัมพันธ์กับแบบรูปอื่น โดยพิจารณาเฉพาะความสัมพันธ์ภายในแบบรูปการตรวจหาดังแสดงในภาพที่ 3.3 และอธิบายว่าสามารถนำแบบรูปใดไปออกแบบยูเอ็มแอลโปรไฟล์ได้บ้าง (แสดงด้วยตัวหนาในภาพ) รายละเอียดการพิจารณามีดังนี้



ภาพที่ 3.3 แผนภาพแสดงความสัมพันธ์ภายในแบบรูปการตรวจหา [5]

3.2.1 Fault Correlation

เมื่อตรวจพบว่ามีข้อผิดพลาดเกิดขึ้นในระบบ ควรทำการพิจารณาลักษณะของข้อผิดพลาดและสาเหตุที่ทำให้เกิดข้อผิดพลาดนั้นว่าเกิดจากความผิดพร่องประเภทใด และทำการจำแนกประเภทของความผิดพร่อง เพื่อนำไปสู่การเลือกวิธีการจัดการที่เหมาะสมกับความผิดพร่องนั้น

จากแบบรูป *Fault Correlation* ไม่สามารถนำมาออกเป็นยูเอ็มแอลโปรไฟล์ได้ เนื่องจากต้องทำการพิจารณาถึงข้อผิดพลาดที่เกิดขึ้นก่อนเพื่อจะนำไปสู่การเลือกแบบรูปที่เหมาะสมมาใช้ในขั้นตอนการออกแบบ

3.2.2 Error Containment Barrier

เมื่อตรวจพบว่ามีข้อผิดพลาดเกิดขึ้นภายในหน่วยย่อยของระบบ ข้อผิดพลาดที่เกิดขึ้นจะต้องถูกจำกัดไม่ให้ส่งผลกระทบไปยังหน่วยย่อยอื่น ตามการใช้งานของแบบรูป *Units of Mitigation* ซึ่งได้แบ่งระบบออกเป็นหน่วยย่อยๆ รวมทั้งแบบรูป *Quarantine* ซึ่งจะกั้นไม่ให้ข้อผิดพลาดกระจายไปยังส่วนย่อยอื่นของระบบ เพื่อการจัดการกับข้อผิดพลาดต่อไป

จากแบบรูป *Error Containment Barrier* กล่าวถึงพฤติกรรมการจำกัดขอบเขตของข้อผิดพลาด ซึ่งถือว่าหน่วยย่อยซึ่งได้ออกแบบตามแบบรูป *Units of Mitigation* และแบบรูป

Quarantine จะเป็นการจำกัดขอบเขตของข้อผิดพลาดให้อยู่ภายในหน่วยย่อยนั้น จึงไม่นำแบบรูปนี้ไปออกแบบเป็นยูเอ็มแอลโปรไฟล์

3.2.3 Complete Parameter Checking

แบบรูปนี้เป็นแบบรูปหนึ่งที่ใช้ในการตรวจหาข้อผิดพลาด โดยตรวจสอบจากข้อมูลในทุกการดำเนินงานของระบบ ตัวอย่างเช่น แบบรูป *Redundancy* ชนิด *active* หรือตัวสำรองมีการทำงานพร้อมกัน มีการตรวจสอบข้อมูลในขั้นตอนการดำเนินการทุกขั้นตอน เช่น ข้อมูลที่ผ่านเข้าไปยังการดำเนินงานหรือ ตรวจสอบผลลัพธ์ที่ได้จากการดำเนินงาน และเปรียบเทียบกันระหว่างตัวสำรอง ซึ่งการตรวจสอบข้อมูลตลอดขั้นตอนการดำเนินงานด้วยความถี่มาก จะทำให้การดำเนินการมีความเชื่อถือได้มากขึ้น แต่จำเป็นต้องแลกกับประสิทธิภาพที่ลดลง เนื่องจากจะมีการใช้เวลาไปในส่วนของการตรวจสอบ การเพิ่มการตรวจสอบข้อมูลไปยังกระบวนการดำเนินงานแบบปกติ อาจจะทำให้เกิดความสับสน ดังนั้นควรมีการแยกส่วนที่ทำการตรวจสอบออกจากการดำเนินงานแบบปกติ

จากแบบรูป *Complete Parameter Checking* จะไม่นำมาออกแบบยูเอ็มแอลโปรไฟล์ เนื่องจากเป็นแบบรูปที่ใช้ในการตรวจสอบข้อมูลในระบบ ควรทำในส่วนของกระบวนการทำงานมากกว่าการออกแบบ

3.2.4 System Monitor

แบบรูปการตรวจหาข้อผิดพลาดนี้ จะมีหน่วยที่ทำหน้าที่ตรวจสอบการทำงานของหน่วยภายในระบบว่ายังทำงานอยู่หรือไม่ โดยที่หน่วยนี้จะอยู่ภายในฮาร์ดแวร์เดียวกับระบบหรือแยกออกจากระบบก็ได้ แต่ถ้าต้องการตรวจสอบว่าฮาร์ดแวร์ของระบบยังทำงานอยู่หรือไม่ ก็ควรจะติดตั้งหน่วยที่ทำการตรวจสอบไว้แยกออกจากระบบ ซึ่งหน่วยที่ทำหน้าที่ตรวจสอบสามารถทำการตรวจสอบเพียงหน่วยเดียวหรือหลายหน่วยภายในระบบก็ได้ เมื่อตรวจพบว่าเกิดข้อผิดพลาดขึ้นจะทำการรายงานไปยังหน่วยที่มีหน้าที่รับผิดชอบ การนำแบบรูป *System Monitor* มาใช้ต้องมีการกำหนดเวลาที่เหมาะสม เมื่อไม่มีการตอบรับจากหน่วยที่ถูกตรวจสอบ จะกล่าวได้ว่ามีข้อผิดพลาดเกิดขึ้นและเวลาที่กำหนดจะต้องกำหนดตามเวลาที่เป็นจริงในระบบ ตามแบบรูป *Realistic Threshold*

จากแบบรูป *System Monitor* สามารถนำมาออกแบบยูเอ็มแอลโปรไฟล์ได้โดย กำหนดยูเอ็มแอลโปรไฟล์สำหรับตัวที่ทำหน้าที่เป็นหน่วยตรวจสอบ

3.2.5 Heartbeat

การตรวจหาข้อผิดพลาดตามแบบรูปนี้ทำได้โดยหน่วยที่ถูกเฝ้าดูจะทำการส่งสัญญาณชีพของตนเอง มาเพื่อแสดงว่าตนเองยังทำงานอยู่ และทำให้หน่วยที่เฝ้าดูทราบว่าหน่วยที่กำลังถูกเฝ้าดูยังทำงานอยู่ ซึ่งการส่งสัญญาณจะส่งตามระยะเวลาที่กำหนดไว้ โดยที่การกำหนดระยะเวลาที่ทำการส่งสัญญาณชีพต้องทำให้สอดคล้องกับการทำงานของระบบ อาจจะอาศัยแบบรูป *Realistic Threshold* ช่วยในการกำหนดค่า เช่น ถ้าระบบมีการส่งข้อความไปมาระหว่างกันบ่อยอยู่แล้ว ก็ควรใช้แบบรูป *Acknowledgement* เพราะเนื่องจากแบบรูป *Heartbeat* ต้องมีการเพิ่มการส่งข้อความพิเศษ ซึ่งอาจจะส่งผลกระทบต่อการทำงานหลักของระบบ

จากแบบรูป *Heartbeat* สามารถนำไปออกแบบยูเอ็มแอลโปรไฟล์เพื่อประยุกต์ใช้เข้ากับหน่วยที่จะต้องมีการส่งสัญญาณชีพ

3.2.6 Acknowledgement

การทำงานของระบบปกติจะมีการส่งการร้องขอจากหน่วยหนึ่งไปยังหน่วยหนึ่ง ซึ่งอาจจะมีการตอบกลับหรือไม่มีก็ได้ อีกวิธีหนึ่งของการตรวจสอบว่าหน่วยที่ถูกตรวจสอบยังทำงานอยู่หรือไม่ คือฝ่ายที่ถูกร้องขอจะต้องมีการตอบรับไปยังผู้ร้องขอทุกครั้ง เพื่อให้ผู้ร้องขอทราบถึงสถานะของผู้ที่ถูกร้องขอ และทั้งฝ่ายผู้ร้องขอและผู้ถูกร้องขอจะต้องเพิ่มกระบวนการที่ใช้ในการพิจารณาข้อความตอบรับที่ได้มาและกระบวนการที่ใช้ในการส่งข้อมูลตอบรับกลับมาตามลำดับ ในกรณีที่ไม่มีข้อความตอบรับมาภายในเวลาที่กำหนด จะถือว่าผู้ถูกร้องขอมีข้อผิดพลาดเกิดขึ้น และจะต้องแจ้งไปยังหน่วยที่มีหน้าที่รับผิดชอบต่อข้อผิดพลาดนั้นต่อไป

จากแบบรูป *Acknowledgement* สามารถนำไปออกแบบยูเอ็มแอลโปรไฟล์สำหรับระบุหน่วยที่ต้องมีการดำเนินงานตามแบบรูปนี้

3.2.7 Watchdog

การเฝ้ามองในแบบรูปนี้มีหน่วยที่ทำหน้าที่เป็นหน่วยเฝ้ามอง หรือที่เรียกว่า *WatchDog* ซึ่งเสมือนกับสุนัขเฝ้าฝูงสัตว์ หน่วยที่เฝ้ามองจะเป็นได้ทั้งฮาร์ดแวร์หรือซอฟต์แวร์ตามความต้องการของระบบ โดยจะเฝ้ามองที่การทำงานปกติของระบบ ทำให้หน่วยที่ถูกเฝ้ามองไม่จำเป็นต้องแก้ไขใดๆเลย และไม่จำเป็นต้องเพิ่มการร้องขอหรือส่งข้อความใดๆ เมื่อตรวจพบว่ามีข้อผิดพลาดเกิดขึ้นในระบบ หน่วยที่เฝ้ามองจะทำการรายงานไปยังผู้รับผิดชอบเพื่อให้จัดการกับข้อผิดพลาดที่เกิดขึ้น

จากแบบรูป *Watchdog* สามารถนำไปสร้างเป็นยูเอ็มแอลโปรไฟล์ เพื่อนำไปประยุกต์ใช้เข้ากับตัวที่ทำหน้าที่เป็นหน่วยเฝ้ามอง

3.2.8 Realistic Threshold

ระยะเวลาในการเฝ้าดูการทำงานของระบบ ควรกำหนดจากเวลาที่ใช้จริงในระบบ ซึ่งคำนวณจากเวลา 2 ค่า คือ เวลาแฝงของการส่งข้อความ (Message Latency) และ เวลาแฝงของการตรวจหา (Detection Latency)

- เวลาแฝงของการส่งข้อความ (Message Latency) คือระยะเวลาในการส่งข้อความเพื่อถามสถานะของระบบตามแบบรูป *Heartbeat* โดยคำนวณจากเวลาของกรณีแย่ที่สุดในการส่งข้อความไปกลับ (Communication Time) รวมกับเวลาที่ใช้ในการประมวลผลข้อความ (Processing Time) ของทั้ง 2 ฝ่าย
- เวลาแฝงของการตรวจหา (Detection Latency) คือระยะเวลาที่หน่วยเฝ้าดูจะรอการตอบรับจากหน่วยที่ถูกเฝ้าดู ก่อนที่จะแจ้งว่ามีข้อผิดพลาดเกิดขึ้นในระบบ ซึ่งมักจะกำหนดเป็นจำนวนเท่าของเวลาแฝงของการส่งข้อความ

อีกหนึ่งปัจจัยที่ใช้ในการกำหนดระยะเวลาที่จะยอมให้ระบบอยู่ในสภาพไม่พร้อมใช้งานได้นานที่สุด ได้แก่ผลรวมของเวลาแฝงของการส่งข้อความ เวลาของแฝงการตรวจหา และเวลาที่ใช้ในการกู้ระบบ ควรน้อยกว่าเวลาที่ยอมให้ระบบอยู่ในสภาพไม่พร้อมใช้งาน (สภาพไม่พร้อมใช้งานนานที่สุด > เวลาแฝงของการส่งข้อความ + เวลาแฝงของการตรวจหา + เวลาที่ใช้ในการกู้ระบบ)

จากแบบรูป *Realistic Threshold* เป็นการแนะนำการกำหนดค่าเวลาสำหรับนำมาใช้ในแบบรูปที่มีการควบคุมหน่วยการทำงาน ซึ่งมีการกำหนดระยะเวลา เช่น แบบรูป *System Monitor* แบบรูป *Heartbeat* หรือ แบบรูป *Watchdog* จึงไม่นำแบบรูปนี้ไปสร้างเป็นยูเอ็มแอลโปรไฟล์

3.2.9 Existing Metrics

แบบรูปนี้ช่วยในการตรวจหาข้อผิดพลาดของระบบโดยที่ไม่จำเป็นต้องเพิ่มภาระเข้าไปในระบบ โดยการใช้มาตรวัดที่มีอยู่ในระบบอยู่แล้ว จึงเหมาะสมกับระบบที่มีสภาวะโหลดเกิน เช่น การใช้มาตรวัดปริมาณการใช้งานซีพียู ซึ่งจะบ่งบอกถึงปริมาณงานที่เข้ามาระบบ สำหรับใช้จัดการบรรเทาข้อผิดพลาด เมื่อพบว่าระบบมีสภาวะโหลดเกิน

ตามแบบรูป *Existing Metrics* นำเสนอการนำมาตรวัดไปใช้สำหรับขั้นตอนของการบรรเทาข้อผิดพลาด ซึ่งอยู่นอกขอบเขตของงานวิจัยนี้ จึงไม่นำไปออกแบบเป็นยูเอ็มแอลโปรไฟล์

3.2.10 Voting

การออกแบบโดยใช้แบบรูป *Redundancy* ในบางรูปแบบ ตัวสำรองจะทำงานพร้อมกันแบบขนาน จึงทำให้มีผลลัพธ์มากกว่า 1 ตัว และต้องมีการทำการโหวตเพื่อหาผลลัพธ์ที่เหมาะสม นอกจากนี้การโหวตยังช่วยบ่งบอกถึงความผิดพร่องที่เกิดขึ้นกับตัวสำรองที่ให้คำตอบแตกต่างจากตัวสำรองอื่นได้

ถ้ามีการทำการโหวตจากหน่วยย่อยหรือตัวสำรองที่มีจำนวนหน่วยย่อยที่ถูกออกแบบตามแบบรูป *Units of Mitigation* เป็นจำนวนมาก การหาคำตอบอาจจะต้องมีความซับซ้อนมากขึ้นและมีการใช้ทรัพยากรมากขึ้นตามไปด้วย

จากแบบรูป *Voting* สามารถนำมาออกแบบยูเอ็มแอลโปรไฟล์ เพื่อกำหนดหน่วยที่ทำหน้าที่รับผิดชอบในการโหวตผลลัพธ์ และกำหนดวิธีการหาผลลัพธ์ที่เหมาะสม

3.2.11 Routine Maintenance

การบำรุงรักษาระบบอย่างเป็นประจำ สามารถช่วยในการตรวจสอบข้อผิดพลาดได้ก่อน หรือป้องกันไม่ให้ข้อผิดพลาดเกิดขึ้นมาได้อีก แบบรูป *Routine Maintenance* สามารถทำได้โดยดำเนินการตามแบบรูป *Hardware Exercise* แบบรูป *Routines Audits* และแบบรูป *Routine Exercises* การบำรุงรักษาระบบสามารถทำผ่านแบบรูป *Maintenance Interface* หรือสั่งการบำรุงรักษาไว้ให้ทำงานอย่างอัตโนมัติ

จากแบบรูป *Routine Maintenance* จะไม่นำไปออกแบบยูเอ็มแอลโปรไฟล์ เนื่องจากการบำรุงรักษาระบบเป็นหน้าที่ของมนุษย์ ไม่เหมาะสมที่จะนำไปทำการออกแบบยูเอ็มแอลโปรไฟล์

3.2.12 Routine Exercises

เมื่อระบบมีการออกแบบตามแบบรูป *Redundancy* และตัวสำรองถูกเรียกใช้เฉพาะเมื่อมีข้อผิดพลาดเกิดขึ้น ถ้าตัวสำรองนั้นยังไม่ถูกเรียกขึ้นมาใช้งานจะไม่มีโอกาสทราบเลยว่าตัวสำรองจะมีข้อผิดพลาดเกิดขึ้น ดังนั้นจึงต้องมีการตรวจสอบว่าตัวสำรองเหล่านั้นสามารถใช้งานหรือจัดการกับข้อผิดพลาดที่จะเกิดขึ้นได้ก่อน นอกจากนี้การนำแบบรูปนี้มาใช้ควรคำนึงถึงภาระงานที่จะเพิ่มขึ้นในระบบด้วย เพื่อไม่ให้ส่งผลต่อการทำงานหลักของระบบ

จากแบบรูป *Routine Exercises* เป็นแบบรูปคำแนะนำให้มีการตรวจสอบระบบเช่นเดียวกับแบบรูป *Routine Maintenance* จึงไม่นำไปออกแบบเป็นยูเอ็มแอลโปรไฟล์

3.2.13 Routine Audits

ข้อมูลที่มีความผิดพร่องที่อยู่ในระบบ มีโอกาสกระจายความผิดพร่องไปยังส่วนต่างๆของระบบ ดังนั้นควรมีการตรวจหาข้อผิดพลาดในข้อมูลในช่วงเวลาที่ระบบมีภาระงานน้อย ซึ่งอาจจะทำในช่วงเวลาเดียวกันกับการบำรุงรักษาตามแบบรูป *Routine Maintenance* เมื่อตรวจพบข้อผิดพลาดจะต้องทำการแก้ไขตามแบบรูป *Correcting Audits* และจะต้องรายงานไปยังส่วนที่เกี่ยวข้อง

จากแบบรูป *Routine Audits* เป็นคำแนะนำให้ทำการตรวจสอบระบบ เช่นเดียวกับแบบรูป *Routine Maintenance* จึงไม่นำมาออกแบบเป็นยูเอ็มแอลโปรไฟล์

3.2.14 Checksum

เมื่อต้องการจะตรวจสอบความถูกต้องของข้อมูล วิธีหนึ่งของการตรวจสอบ คือทำการตรวจสอบกับข้อมูลชุดเดียวกันที่มีการเก็บไว้ในอีกที่หนึ่ง แต่วิธีการเก็บข้อมูลไว้ในหลายที่จะทำให้เสียเนื้อที่ในการเก็บข้อมูลจำนวนมาก จึงควรทำการคำนวณค่าบางค่าจากข้อมูลบางส่วนหรือกลุ่มข้อมูลโดยใช้อัลกอรึทึมแบบแฮช (Hash Algorithm) เช่น SHA, MD5 หรืออื่นๆ และเมื่อต้องการตรวจสอบก็จะทำการคำนวณตามวิธีเดิมและนำมาเปรียบเทียบกับค่าที่เก็บไว้ ซึ่งเป็นตัวบ่งชี้การเกิดข้อผิดพลาด

จากแบบรูป *Checksum* ไม่นำมาออกแบบเป็นยูเอ็มแอลโปรไฟล์ เนื่องจากเป็นแบบรูปที่ทำงานเกี่ยวกับการตรวจสอบข้อมูลในระบบ ควรมีการดำเนินการในกระบวนการทำงานไม่ใช่การออกแบบ อีกทั้งในโพรโทคอลของการสื่อสารจะใช้ *Checksum* ตรวจสอบความถูกต้องของข้อมูลในระดับหนึ่งอยู่แล้ว

3.2.15 Riding Over Transients

ข้อผิดพลาดบางอย่างที่เกิดขึ้นในระบบอาจจะเกิดขึ้นแบบชั่วคราว กล่าวคือข้อผิดพลาดนั้นเกิดขึ้นแล้วอาจจะหายไปเอง หรือข้อผิดพลาดนั้นไม่ได้เกิดขึ้นซ้ำๆ ระบบควรจะทำการเพิกเฉยกับข้อผิดพลาดดังกล่าว เพื่อไม่ให้ระบบสูญเสียเวลาและทรัพยากรไปกับการประมวลผลข้อผิดพลาดนั้นๆ โดยแบบรูป *Riding Over Transients* จะมีกลไกในการพิจารณาว่าข้อผิดพลาดที่เกิดขึ้นเป็นข้อผิดพลาดแบบชั่วคราวหรือถาวร

จากแบบรูป *Riding Over Transients* ต้องทำการวิเคราะห์ข้อผิดพลาดที่เกิดขึ้นก่อนที่จะทำการออกแบบการจัดการกับข้อผิดพลาด จึงไม่นำแบบรูปนี้มาออกแบบเป็นยูเอ็มแอลโปรไฟล์

3.2.16 Leaky Bucket Counter

แบบรูปนี้ช่วยในการตรวจสอบว่าข้อผิดพลาดที่เกิดขึ้นในระบบเป็นข้อผิดพลาดแบบชั่วคราวหรือถาวร โดยที่การตรวจสอบข้อผิดพลาดที่เกิดขึ้นแบบซ้ำๆ สามารถทำได้โดยนับจำนวนของข้อผิดพลาดที่เกิดขึ้นในช่วงระยะเวลาหนึ่ง ถ้าข้อผิดพลาดนั้นไม่เกิดขึ้นซ้ำอีกในช่วงเวลาที่กำหนด จะทำการลดจำนวนของตัวนับนี้ลงไปเรื่อยๆ ถ้าจำนวนข้อผิดพลาดที่เกิดขึ้นมากกว่าเกณฑ์ที่กำหนด จะถือว่าข้อผิดพลาดนั้นเป็นข้อผิดพลาดแบบถาวร

จากแบบรูป *Leaky Bucket Counter* เป็นการวิเคราะห์ข้อผิดพลาดที่เกิดขึ้นในระบบเช่นเดียวกับแบบรูป *Riding Over Transient* จึงไม่นำแบบรูปนี้มาออกแบบเป็นยูเอ็มแอลโปรไฟล์

จากรายละเอียดของแบบรูปในกลุ่มของแบบรูปการตรวจหา สามารถนำมาสรุปแบบรูปที่จะนำไปพัฒนาเป็นยูเอ็มแอลโปรไฟล์ และระบุถึงจุดประสงค์ของแบบรูป ดังตารางที่ 3.2

ตารางที่ 3.2 การนำแบบรูปการตรวจหาไปออกแบบยูเอ็มแอลโปรไฟล์

แบบรูป	**จุดประสงค์ของแบบรูป**	**นำไปสร้างยูเอ็มแอลโปรไฟล์หรือไม่**
Fault Correlation	วิเคราะห์สัญญาณของข้อผิดพลาด เพื่อระบุประเภทของความผิดพร่องที่จะเกิดขึ้น	× (ต้องวิเคราะห์ก่อน)
Error Containment Barrier	ข้อผิดพลาดที่เกิดขึ้นในหน่วยหนึ่งๆของระบบ จะไม่ส่งผลกระทบต่อหน่วยอื่น	× (ออกแบบแล้ว)
Complete Parameter Checking	ตรวจสอบข้อมูลในกระบวนการทำงาน เพื่อเป็นการตรวจสอบและป้องกันข้อผิดพลาดที่จะเกิดขึ้นในระบบ	× (ตรวจสอบข้อมูล)
System Monitor	ออกแบบระบบให้มีตัวควบคุมเพื่อตรวจสอบพฤติกรรมของระบบ	√
Heartbeat	ออกแบบระบบให้มีการส่งสถานะของตัวเองไป เพื่อแจ้งให้ทราบว่าตัวเองยังทำงานอยู่	√
Acknowledgement	ออกแบบให้มีการส่งข้อความตอบกลับ เพื่อแจ้งให้อีกฝ่ายทราบว่ายังทำงานอยู่	√

ตารางที่ 3.2 การนำแบบรูปการตรวจหาไปออกแบบยูเอ็มแอลโปรไฟล์ (ต่อ)

แบบรูป	**จุดประสงค์ของแบบรูป**	**นำไปสร้างยูเอ็มแอลโปรไฟล์หรือไม่**
Watchdog	สร้างหน่วยหนึ่งขึ้นมาเพื่อคอยเฝ้ามองการทำงานของระบบว่าทำได้อย่างปกติ	√
Realistic Threshold	การตั้งค่าต่างๆในระบบ จะต้องยึดจากค่าที่เป็นไปได้ของการปฏิบัติจริงในระบบ เพื่อให้มีความแม่นยำ	× (คำแนะนำ)
Existing Metrics	ใช้ตัววัดที่มีอยู่แล้วในระบบมาใช้งาน เพื่อไม่ให้เสียเวลาในการคำนวณค่าใหม่	× (นอกขอบเขต)
Voting	ออกแบบให้มีกระบวนการในการโหวตคำตอบที่เหมาะสม เมื่อมีผลลัพธ์มากกว่า 1 ค่า	√
Routine Maintenance	ต้องมีการบำรุงรักษาระบบอย่างเป็นประจำ เพื่อไม่ให้มีข้อผิดพลาดสะสมในระบบ	× (ทำโดยมนุษย์)
Routine Exercises	ต้องมีการตรวจสอบตัวสำรองอยู่เสมอ ว่าพร้อมใช้งานเมื่อมีข้อผิดพลาดเกิดขึ้น	× (ทำโดยมนุษย์)
Routine Audits	ต้องมีการตรวจสอบความถูกต้องของข้อมูลในระบบ เพื่อมั่นใจว่าข้อมูลนั้นยังถูกต้อง	× (ทำโดยมนุษย์)
Checksum	เก็บข้อมูลบางอย่างไว้ เพื่อไว้ใช้ตรวจสอบความถูกต้องของข้อมูล	× (ตรวจสอบข้อมูล)
Riding Over Transients	เพิกเฉยข้อผิดพลาดในระบบที่เป็นข้อผิดพลาดแบบชั่วคราว	× (ต้องวิเคราะห์ก่อน)
Leaky Bucket Counter	เพิ่มตัวนับในระบบ เพื่อใช้พิจารณาว่าข้อผิดพลาดที่เกิดขึ้นเป็นข้อผิดพลาดแบบถาวรหรือชั่วคราว	× (ต้องวิเคราะห์ก่อน)

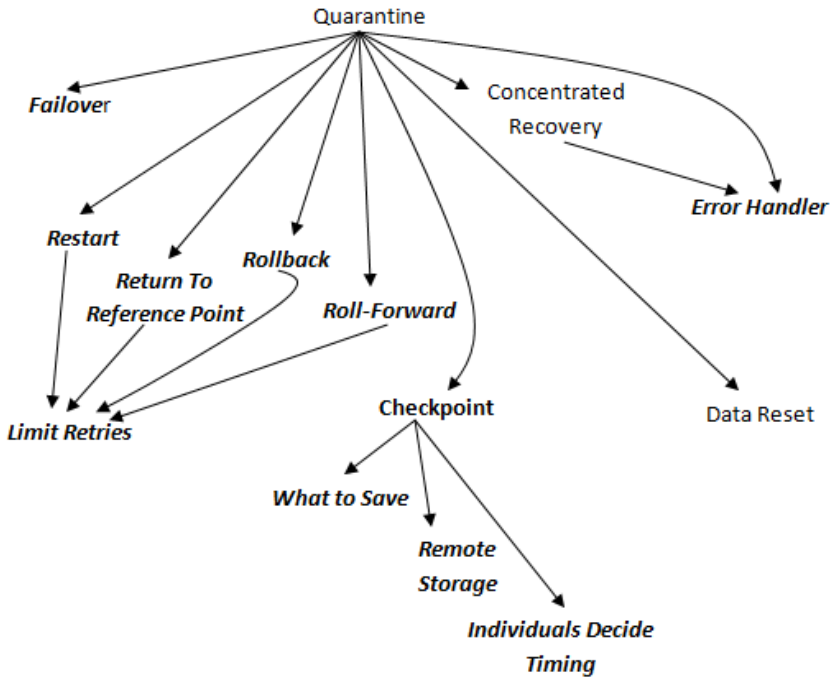
จากตารางที่ 3.2 รายละเอียดเหตุผลของการไม่นำแบบรูปนั้นๆ ไปสร้างเป็นยูเอ็มแอลโปรไฟล์ มีดังนี้

- ต้องวิเคราะห์ก่อน - แบบรูปนี้จะไม่ถูกนำไปออกแบบเป็นยูเอ็มแอลโปรไฟล์ เนื่องจากต้องมีการวิเคราะห์ข้อผิดพลาดที่เกิดขึ้นก่อนที่จะนำมาออกแบบ

- ออกแบบแล้ว – แบบรูปนี้จะไม่ถูกนำไปออกแบบเป็นยูเอ็มแอลโปรไฟล์ เนื่องจากรายละเอียดของแบบรูปมีความใกล้เคียงกับแบบรูปอื่นที่มีการออกแบบเป็นยูเอ็มแอลโปรไฟล์แล้ว
- ตรวจสอบข้อมูล – แบบรูปนี้จะไม่ถูกนำไปออกแบบเป็นยูเอ็มแอลโปรไฟล์ เนื่องจากมีการกล่าวถึงการตรวจความถูกต้องของข้อมูล ซึ่งควรทำในกระบวนการทำงานมากกว่าขั้นตอนการออกแบบ
- คำแนะนำ – แบบรูปนี้จะไม่ถูกนำไปออกแบบเป็นยูเอ็มแอลโปรไฟล์ เนื่องจากเป็นเพียงการนำเสนอคำแนะนำในการออกแบบระบบ ซึ่งผู้ออกแบบจะนำไปใช้ในการออกแบบหรือไม่ก็ได้
- นอกขอบเขต – แบบรูปนี้จะไม่ถูกนำไปออกแบบเป็นยูเอ็มแอลโปรไฟล์ เนื่องจากรายละเอียดของแบบรูปนี้อยู่นอกขอบเขตของงานวิจัย
- ทำโดยมนุษย์ – แบบรูปนี้จะไม่ถูกนำไปออกแบบเป็นยูเอ็มแอลโปรไฟล์ เนื่องจากการนำเสนอแบบรูปดังกล่าวเป็นกระบวนการทำงานที่ต้องทำโดยมนุษย์หรือมีการทำงานแบบอัตโนมัติ

3.3 แบบรูปการกู้ระบบจากข้อผิดพลาด (Error Recovery Patterns)

เป็นกลุ่มของแบบรูปที่ทำการจัดการกับข้อผิดพลาดที่เกิดขึ้น ซึ่งพยายามกู้ระบบจากข้อผิดพลาด ให้ระบบสามารถทำงานต่อไปได้ ถึงแม้ว่าสถานะของระบบจะถูกเปลี่ยนแปลงไป ซึ่งจะแสดงความหมาย การใช้งานของแบบรูป ความสัมพันธ์กับแบบรูปอื่น โดยพิจารณาเฉพาะความสัมพันธ์ภายในแบบรูปการกู้ระบบจากข้อผิดพลาดดังแสดงในภาพที่ 3.4 และอธิบายว่าสามารถนำแบบรูปใดไปออกแบบยูเอ็มแอลโปรไฟล์ได้บ้าง (แสดงด้วยตัวหนาในภาพ) รายละเอียดการพิจารณามีดังนี้



ภาพที่ 3.4 แผนภาพแสดงความสัมพันธ์ภายในแบบรูปการกู้ระบบจากข้อผิดพลาด [5]

3.3.1 Quarantine

เมื่อหน่วยย่อยที่ถูกออกแบบตามแบบรูป *Units of Mitigation* มีข้อผิดพลาดเกิดขึ้น อาจจะเกิดผลกระทบต่อส่วนอื่นที่เกี่ยวข้อง เช่น ผลกระทบเกี่ยวกับการส่งข้อความไปยังระบบอื่น การใช้งานหน่วยความจำ การเปลี่ยนสถานะของระบบ การเปลี่ยนการทำงานของระบบ ซึ่งสิ่งต่างๆเหล่านี้ทำการยกเลิกยาก ดังนั้นควรทำการล้อมกรอบการทำงานของหน่วยย่อยเพื่อเป็นการป้องกันไม่ให้ข้อผิดพลาดกระจายไปยังหน่วยอื่นของระบบ

จากแบบรูป *Quarantine* การล้อมกรอบข้อผิดพลาดที่เกิดขึ้นในหน่วยหนึ่ง สามารถทำได้ในขั้นตอนการออกแบบซึ่งออกแบบตามแบบรูป *Units of Mitigation* จึงไม่นำแบบรูปนี้ไปออกแบบยูเอ็มแอลโปรไฟล์อีก

3.3.2 Concentrated Recovery

เมื่อมีข้อผิดพลาดเกิดขึ้นระบบต้องใช้เวลากู้ระบบจากข้อผิดพลาดให้น้อยที่สุด ด้วยการออกแบบโดยใช้แบบรูป *Units of Mitigation* ทำให้ระบบสามารถที่จะทำการกู้ระบบเพียงแค่หน่วยย่อยที่เกิดข้อผิดพลาดเท่านั้น โดยหน่วยย่อยอื่นยังสามารถทำงานได้อย่างปกติ แต่การทำงานของระบบโดยที่มีส่วนใดส่วนหนึ่งมีข้อผิดพลาดทำให้ระบบมีความเสี่ยง ดังนั้นจึงต้องทำการกู้ระบบให้เร็ว

ที่สุด โดยจะต้องทำการดึงทรัพยากรที่จำเป็นต่อการกู้ระบบมาใช้ประโยชน์ให้มากที่สุด เพื่อเป็นการลดช่วงเวลาที่ระบบไม่สามารถใช้งานได้

จากแบบรูป *Concentrated Recovery* ไม่นำมาออกแบบเป็นยูเอ็มแอลโปรไฟล์ เนื่องจากเป็นแบบรูปที่เป็นข้อแนะนำให้ดำเนินการอย่างเต็มที่ในการกู้ระบบ

3.3.3 Error Handler

การจัดการกับข้อผิดพลาดที่เกิดขึ้นในระบบไม่ทำให้เกิดผลิตภาพในระบบ ดังนั้นควรแยกหน่วยของการจัดการข้อผิดพลาดออกจากการทำงานหลักของระบบ เพื่อนำกลับมาใช้ใหม่ได้เมื่อมีข้อผิดพลาดเดิมเกิดขึ้นในระบบอีกครั้ง และเป็นการลดขนาดของโค้ด และเพื่อความสะดวกในการบำรุงรักษาหรือแก้ไขระบบในอนาคต เมื่อมีการจัดการข้อผิดพลาด ควรจะหยุดการทำงานอื่นของระบบก่อน เพื่อทำการแก้ไขข้อผิดพลาด โดยการกู้ระบบจากข้อผิดพลาดไม่จำเป็นต้องทำที่หน่วยที่ออกแบบตามแบบรูป *Error Handler* เสมอ ซึ่งหลังจากทำการจัดการข้อผิดพลาดเสร็จสิ้นแล้ว ต้องกลับมาทำงานในการทำงานปกติของระบบ บางภาษา เช่น C++ หรือ JAVA มีวิธีเพื่อรองรับการจัดการข้อผิดพลาด

จากแบบรูป *Error Handler* สามารถออกแบบเป็นยูเอ็มแอลโปรไฟล์สำหรับประยุกต์เข้าไปยังหน่วยที่ทำหน้าที่จัดการกับข้อผิดพลาด

3.3.4 Restart

เมื่อระบบทำตามแบบรูป *Escalation* ซึ่งจะทำการจัดการกับข้อผิดพลาดด้วยวิธีที่เข้มขึ้นเรื่อยๆ แต่ถ้ายังไม่สามารถจัดการกับข้อผิดพลาดนั้นได้ ดังนั้นจึงควรกลับไปทำใหม่ตั้งแต่จุดเริ่มต้นตามแบบรูป *Restart* แต่การกลับไปเริ่มทำในสถานะเริ่มต้นจะเสียเวลา ถ้ามีการแบ่งระบบออกเป็นหน่วยย่อยๆ และเริ่มทำใหม่แค่แต่ละหน่วยย่อยๆจะทำให้ไม่จำเป็นต้องกลับสู่สถานะเริ่มต้นทั้งระบบ และหน่วยอื่นของระบบยังสามารถทำงานต่อไปได้

จากแบบรูป *Restart* สามารถนำไปออกแบบเป็นยูเอ็มแอลโปรไฟล์เพื่อประยุกต์เข้ากับการจัดการกับข้อผิดพลาดที่เกิดขึ้นในหน่วยย่อยของระบบ

3.3.5 Rollback

เมื่อระบบจัดการกับข้อผิดพลาดเรียบร้อยแล้ว จะต้องกลับไปทำงานยังการทำงานปกติของระบบ ถ้าระบบมีการประยุกต์ใช้แบบรูป *Checkpoint* จะสามารถกลับไปทำงานยังจุดที่บันทึกไว้ได้ แต่ถ้าไม่มีการใช้แบบรูป *Checkpoint* จะกลับไปยังจุดที่ทำคำสั่งล่าสุดก่อนการเกิดข้อผิดพลาด โดยที่

การย้อนกลับไปยังจุดที่บันทึกไว้ หากยังทำงานไม่สำเร็จ จะทำให้เสียเวลาในการทำงานมากขึ้น ดังนั้นจึงควรคำนึงถึงจำนวนครั้งของการทำซ้ำ ตามแบบรูป *Limit Retries*

จากแบบรูป *Rollback* สามารถนำไปออกแบบเป็นยูเอ็มแอลโปรไฟล์เพื่อประยุกต์เข้ากับการจัดการกับข้อผิดพลาดที่เกิดขึ้นในหน่วยย่อยของระบบ เช่นเดียวกับแบบรูป *Restart*

3.3.6 Roll-Forward

เมื่อระบบมีการจัดการกับข้อผิดพลาดเรียบร้อยแล้ว จะต้องย้อนกลับมาทำยังจุดที่มีการเก็บข้อมูลตามแบบรูป *Checkpoint* ไว้ แต่การย้อนกลับมาทำอาจจะมีความเสี่ยง เนื่องจากการกลับมาทำยังจุดเดิมโดยที่ยังไม่มีการเปลี่ยนแปลงแก้ไขใดๆ อาจจะทำให้เกิดข้อผิดพลาดขึ้นอีก หรือทำให้ใช้เวลาในการทำงานมากขึ้น ซึ่งถ้าสามารถละเว้นข้อผิดพลาดนั้นได้และไปทำงานยังจุดถัดไปเลยจะมีความเหมาะสมมากกว่า และเหมาะสมกับระบบที่เป็นการทำงานแบบทันทีทันใดมากกว่า

จากแบบรูป *Roll-Forward* สามารถนำไปออกแบบเป็นยูเอ็มแอลโปรไฟล์เพื่อประยุกต์เข้ากับการจัดการกับข้อผิดพลาดที่เกิดขึ้นในหน่วยย่อยของระบบ เช่นเดียวกับแบบรูป *Restart* และ *Rollback*

3.3.7 Return to Reference Point

เมื่อตรวจพบว่ามีข้อผิดพลาดเกิดขึ้นในส่วนที่ไม่ใช่การทำงานหลัก เช่น ในขั้นตอนการจัดการข้อผิดพลาด หรือการบำรุงรักษาระบบ การทำตามแบบรูป *Rolback* เป็นการย้อนกลับไปยังส่วนของการทำงานหลักของระบบ จึงไม่มีส่วนช่วยในการกู้ระบบในส่วนที่ไม่ใช่การทำงานหลัก ทำให้ต้องมีการกำหนดจุดอ้างอิง (Reference Point) ซึ่งเป็นจุดที่จะถูกกำหนดไว้แบบสถิต (Static) ในช่วงของการออกแบบ สำหรับใช้เป็นจุดที่จะกลับมาทำงานต่อในส่วนของการทำงานหลัก หากเกิดข้อผิดพลาดในส่วนที่ไม่ใช่การทำงานหลัก จุดอ้างอิงจะแตกต่างจากจุดตรวจสอบ (Checkpoint) โดยที่จุดตรวจสอบจะถูกกำหนดเป็นระยะในระหว่างการทำงาน การกลับไปทำงานที่จุดตรวจสอบจึงมีลักษณะพลวัต (Dynamic) ขึ้นอยู่กับว่าข้อผิดพลาด ในระหว่างการทำงานเกิดขึ้นใกล้กับจุดตรวจสอบใด

จากแบบรูป *Return to Reference Point* สามารถนำไปออกแบบเป็นยูเอ็มแอลโปรไฟล์เพื่อประยุกต์เข้ากับการจัดการกับข้อผิดพลาดที่เกิดขึ้นในหน่วยย่อยของระบบ เช่นเดียวกับแบบรูป *Restart* แบบรูป *Rollback* และแบบรูป *Roll-Forward*

3.3.8 Limit Retries

ในการจัดการข้อผิดพลาด ถ้ามีการกลับไปทำงานยังจุดเดิม ซึ่งมีสถานะเดิม และข้อมูลเหมือนเดิม ทำให้มีโอกาสเกิดข้อผิดพลาดขึ้นอีก ดังนั้นจึงต้องมีการกำหนดจำนวนครั้งของการทำซ้ำให้เหมาะสม เพื่อไม่ให้ระบบติดอยู่ในการจัดการข้อผิดพลาดนั้นซ้ำๆ ในกรณีที่ไม่สามารถจัดการกับข้อผิดพลาดได้ควรมีการจัดการกับข้อผิดพลาดในรูปแบบอื่น

จากแบบรูป *Limit Retries* สามารถนำไปออกแบบเป็นยูเอ็มแอลโปรไฟล์เพื่อกำหนดหน่วยที่จะทำตามแบบรูป *Limite Retries* และกำหนดจำนวนครั้งของการทำซ้ำ

3.3.9 Failover

เมื่อการจัดการกับข้อผิดพลาดนำมาใช้ไม่ได้ผล เช่น การทำตามแบบรูป *Rollback* แบบรูป *Roll-Forward* หรือแบบรูป *Return to Reference Point* จึงต้องมีการจัดการโดยใช้แบบรูป *Escalation* คือทำการจัดการที่เข้มขึ้น เช่นการเปลี่ยนไปใช้งานยังตัวสำรองตามแบบรูป *Redundancy* ถ้าตัวสำรองมีสถานะพร้อมใช้งานอยู่แล้วจะทำให้การถ่ายโอนทำได้เร็วขึ้น แต่ถ้าตัวสำรองยังไม่พร้อมใช้งาน การเรียกใช้งานแบบรูป *Checkpoint* จะคืนสถานะให้กลับมาใช้งานได้

จากแบบรูป *Failover* สามารถนำไปออกแบบเป็นยูเอ็มแอลโปรไฟล์เพื่อประยุกต์ใช้กับหน่วยที่ทำหน้าที่ในการถ่ายโอนงานไปยังหน่วยอื่น

3.3.10 Checkpoint

เมื่อมีข้อผิดพลาดเกิดขึ้นในระบบจะต้องทำการกู้ระบบตามแบบรูป *Rollback* คือย้อนกลับไปทำงานในสถานะก่อนหน้า หรือตามแบบรูป *Restart* คือกลับไปเริ่มทำงานใหม่ตั้งแต่จุดเริ่มต้น การทำงานตามแบบรูป *Restart* ถ้าระบบประกอบด้วยหน่วยย่อยจำนวนมากจะทำให้ใช้เวลามาก การใช้แบบรูป *Checkpoint* ทำให้มีการเก็บสถานะของระบบเป็นระยะในช่วงการทำงานเพื่อไม่ให้ระบบต้องกลับไปทำงานยังจุดเริ่มต้น จึงควรใช้แบบรูป *Checkpoint* ควบคู่กับแบบรูป *Rollback* เพื่อให้เรียกสถานะการทำงานล่าสุดกลับมาทำงานต่อได้

สิ่งที่ควรพิจารณาในการทำตามแบบรูป *Checkpoint* คือ ข้อมูลที่จะต้องทำการบันทึก สถานที่ในการจัดเก็บข้อมูล และความถี่ในการเก็บข้อมูล ตามแบบรูป *What to Save* แบบรูป *Remote Storage* และแบบรูป *Individuals Decide Timing* ตามลำดับ

จากแบบรูป *Checkpoint* สามารถนำมาสร้างเป็นยูเอ็มแอลโปรไฟล์เพื่อประยุกต์เข้ากับหน่วยที่ต้องการบันทึกข้อมูล เพื่อนำไปใช้ประโยชน์ในขั้นตอนของการจัดการกับข้อผิดพลาดที่เกิดขึ้นต่อไป

3.3.11 What to Save

เมื่อระบบมีการออกแบบตามแบบรูป *Checkpoint* เพื่อจัดเก็บสถานะของระบบ จึงต้องมีการกำหนดว่าจะต้องจัดเก็บข้อมูลใดบ้าง สถานะของข้อมูล คือ ตัวแปรท้องถิ่น ตำแหน่งการทำงานของโปรแกรม และสถานะของภาระงาน ซึ่งถ้ามีการเก็บข้อมูลเหล่านี้แล้วจะทำให้กู้ระบบกลับมายังสถานะเดิมได้เร็วขึ้น และจำเป็นต้องมีการเก็บข้อมูลส่วนรวม (Global Variable) ซึ่งเป็นตัวแปรที่แต่ละหน่วยต้องใช้ร่วมกัน

จากแบบรูป *What to Save* สามารถนำไปออกแบบเป็นยูเอ็มแอลโปรไฟล์เพื่อกำหนดข้อมูลที่ทำการบันทึก

3.3.12 Remote Storage

เมื่อระบบมีการออกแบบตามแบบรูป *Redundancy* เมื่อระบบเกิดข้อผิดพลาดจะประมวลผลที่ตัวสำรองของระบบโดยจะใช้สถานะที่เก็บไว้ตามแบบรูป *Checkpoint* ดังนั้นการทำ *Checkpoint* ควรเก็บไว้ในส่วนที่แยกจากส่วนการทำงานหลักและส่วนสำรอง เพื่อลดโอกาสการเกิดข้อผิดพลาดพร้อมกันและเพื่อให้แต่ละหน่วยสามารถมาใช้ข้อมูล *Checkpoint* ได้

จากแบบรูป *Remote Storage* สามารถนำไปออกแบบเป็นยูเอ็มแอลโปรไฟล์เพื่อประยุกต์เข้ากับหน่วยที่มีหน้าที่ในการเก็บข้อมูลที่ทำการบันทึก

3.3.13 Individuals Decide Timing

เมื่อระบบมีการออกแบบตามแบบรูป *Checkpoint* ต้องมีการกำหนดช่วงเวลาที่เหมาะสมในการเก็บข้อมูล รูปแบบหนึ่ง คือทุกกระบวนการในระบบจัดเก็บข้อมูลร่วมกัน ซึ่งรูปแบบการจัดเก็บข้อมูลแบบนี้เหมาะสมกับระบบที่มีกระบวนการเกี่ยวข้องกันน้อย แต่ถ้าแต่ละกระบวนการมีความเกี่ยวข้องกัน เมื่อมีการจัดเก็บข้อมูลไปพร้อมกันเพื่อให้ข้อมูลตรงกัน การจัดเก็บข้อมูลจะใช้เวลามากตามไปด้วย อีกรูปแบบหนึ่งคือแต่ละกระบวนการจัดเก็บข้อมูลตามช่วงเวลาของตัวเอง โดยจะมีช่วงเวลาที่เหมาะสม เช่น เมื่อกระบวนการได้รับสถานะบางอย่าง รูปแบบนี้เหมาะกับกระบวนการที่ต้องการเก็บข้อมูล ณ เวลาใดเวลาหนึ่ง

จากแบบรูป *Individuals Decide Timing* สามารถนำมาออกแบบเป็นยูเอ็มแอลโปรไฟล์เพื่อนำไปประยุกต์ใช้กับแบบ *Checkpoint* เพื่อกำหนดเวลาในการเก็บข้อมูล

3.3.14 Data Reset

เมื่อมีความผิดพร่องของข้อมูลเกิดขึ้นในระบบ แต่ไม่สามารถทำการแก้ไขข้อมูลตามแบบรูป *Correcting Audits* และไม่สามารถทำการกู้ข้อมูลก่อนหน้ากลับมาได้ เนื่องจากไม่ได้ทำการจัดเก็บสถานะของข้อมูลตามแบบรูป *Checkpoint* ดังนั้นจึงต้องเลือกวิธีในการกลับไปใช้ค่าเริ่มต้นของข้อมูล และเมื่อมีการกำหนดค่าของข้อมูลไปยังค่าเริ่มต้น ข้อมูลที่เกี่ยวข้องและสถานะของระบบอาจจะต้องเปลี่ยนตามไปด้วยเช่นกัน

จากแบบรูป *Data Reset* ไม่นำมาออกแบบยูเอ็มแอลโปรไฟล์ เนื่องจากสามารถใช้แบบรูป *Checkpoint* แทนได้ โดยทำการบันทึกข้อมูลในช่วงการเริ่มต้นของระบบ และสามารถทำการกู้คืนข้อมูลจากแบบรูป *Checkpoint* ที่บันทึกไว้

จากรายละเอียดของแบบรูปในกลุ่มของแบบรูปการกู้ระบบจากข้อผิดพลาด สามารถนำมาสรุปแบบรูปที่จะนำไปพัฒนาเป็นยูเอ็มแอลโปรไฟล์ และระบุถึงจุดประสงค์ของแบบรูป ดังตารางที่ 3.3

ตารางที่ 3.3 การนำแบบรูปการกู้ระบบจากข้อผิดพลาดไปออกแบบยูเอ็มแอลโปรไฟล์

แบบรูป	**จุดประสงค์ของแบบรูป**	**นำไปสร้างยูเอ็มแอลโปรไฟล์หรือไม่**
Quarantine	แยกระบบออกเป็นหน่วยย่อย เพื่อเวลามีข้อผิดพลาดเกิดขึ้นในระบบ จะไม่ส่งผลกระทบไปยังส่วนอื่นของระบบ	× (ออกแบบแล้ว)
Concentrated Recovery	เมื่อมีข้อผิดพลาดเกิดขึ้นในระบบ ควรให้ความสำคัญกับการกู้ระบบจากการจัดการกับข้อผิดพลาด เพื่อใช้เวลาในการกู้ระบบน้อยที่สุด	× (คำแนะนำ)
Error Handler	ออกแบบระบบให้มีส่วนที่ทำการจัดการกับข้อผิดพลาด	√
Restart	เมื่อมีข้อผิดพลาดเกิดขึ้นในระบบ ให้เริ่มกลับไปทำงานยังจุดเริ่มต้นของระบบ	√

ตารางที่ 3.3 การนำแบบรูปการกู้ระบบจากข้อผิดพลาดไปออกแบบยูเอ็มแอลโปรไฟล์ (ต่อ)

แบบรูป	**จุดประสงค์ของแบบรูป**	**นำไปสร้างยูเอ็มแอลโปรไฟล์หรือไม่**
Rollback	เมื่อมีข้อผิดพลาดเกิดขึ้นในระบบ ให้กลับไปทำงานยังจุดก่อนที่จะเกิดข้อผิดพลาด	√
Roll-Forward	เมื่อมีข้อผิดพลาดเกิดขึ้นในระบบ ให้ข้ามส่วนที่มีข้อผิดพลาดไปทำงานยังส่วนถัดไปที่ไม่มีข้อผิดพลาด	√
Return to Reference Point	เมื่อมีข้อผิดพลาดเกิดขึ้นในส่วนที่ไม่ใช่การทำงานหลักในระบบ ให้กลับไปทำงานยังจุดอ้างอิงที่ได้มีการบันทึกข้อมูลไว้ และต้องเป็นจุดที่แน่ใจว่าไม่มีข้อผิดพลาด	√
Limit Retries	การกลับไปทำงาน ณ จุดเดิมที่มีข้อผิดพลาดซ้ำๆ โดยไม่เปลี่ยนแปลงแก้ไขใดๆเลย ข้อผิดพลาดนั้นย่อมมีโอกาสเกิดขึ้นซ้ำอีก จึงควรจำกัดจำนวนการกลับไปทำซ้ำ	√
Failover	เมื่อมีความผิดพร่องเกิดขึ้น จะทำการกู้ระบบโดยเปลี่ยนไปทำงานยังตัวสำรองแทน	√
Checkpoint	บันทึกข้อมูลของระบบไว้เป็นระยะ เพื่อนำไปใช้เมื่อต้องทำการกู้ระบบจากข้อผิดพลาด	√
What to Save	มีการกำหนดว่าข้อมูลที่จะต้องทำการบันทึก เป็นข้อมูลใดบ้าง	√
Remote Storage	พิจารณาตัวสำรองในระบบและปัจจัยอื่นๆ เพื่อพิจารณาว่าจะทำการบันทึกข้อมูลไว้ที่ใด	√
Individuals Decide Timing	ออกแบบให้แต่ละกระบวนการทำการกำหนดว่าเมื่อใดที่จะต้องทำการบันทึกข้อมูล	√
Data Reset	ทำการตั้งค่าข้อมูลไปยังค่าเริ่มต้น เมื่อพบว่าข้อมูลนั้นไม่ถูกต้อง	× (ออกแบบแล้ว)

จากตารางที่ 3.3 รายละเอียดเหตุผลของการไม่นำแบบรูปนั้นๆ ไปสร้างเป็นยูเอ็มแอลโปรไฟล์ มีดังนี้

- ออกแบบแล้ว – แบบรูปนี้จะไม่ถูกนำไปออกแบบเป็นยูเอ็มแอลโปรไฟล์ เนื่องจากรายละเอียดของแบบรูปมีความใกล้เคียงกับแบบรูปอื่นที่มีการออกแบบเป็นยูเอ็มแอลโปรไฟล์แล้ว
- คำแนะนำ – แบบรูปนี้จะไม่ถูกนำไปออกแบบเป็นยูเอ็มแอลโปรไฟล์ เนื่องจากเป็นเพียงการนำเสนอคำแนะนำในการออกแบบระบบ ซึ่งผู้ออกแบบจะนำไปใช้ในการออกแบบหรือไม่ก็ได้

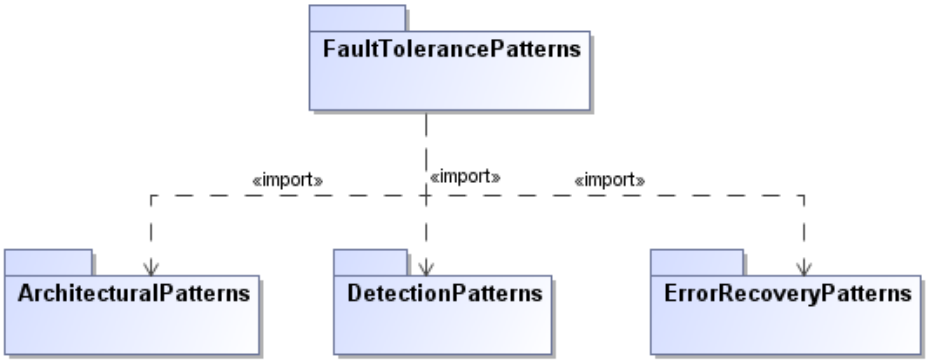
บทที่ 4

การออกแบบและการประยุกต์ใช้ยูเอ็มแอลโปรไฟล์

จากการวิเคราะห์ถึงลักษณะของแบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่อง และพิจารณาถึงความสัมพันธ์ของแบบรูป และความเหมาะสมที่จะนำมาออกแบบเป็นยูเอ็มแอลโปรไฟล์หรือไม่ ดังที่กล่าวมาแล้วในบทที่ 3 ในบทนี้เป็นการนำเสนอยูเอ็มแอลโปรไฟล์สำหรับการทนต่อความผิดพร่อง ตัวอย่างการใช้งานของแต่ละยูเอ็มแอลโปรไฟล์สำหรับแต่ละแบบรูป และนำเสนอการประยุกต์ใช้งานยูเอ็มแอลโปรไฟล์สำหรับแบบรูปการทนต่อความผิดพร่อง

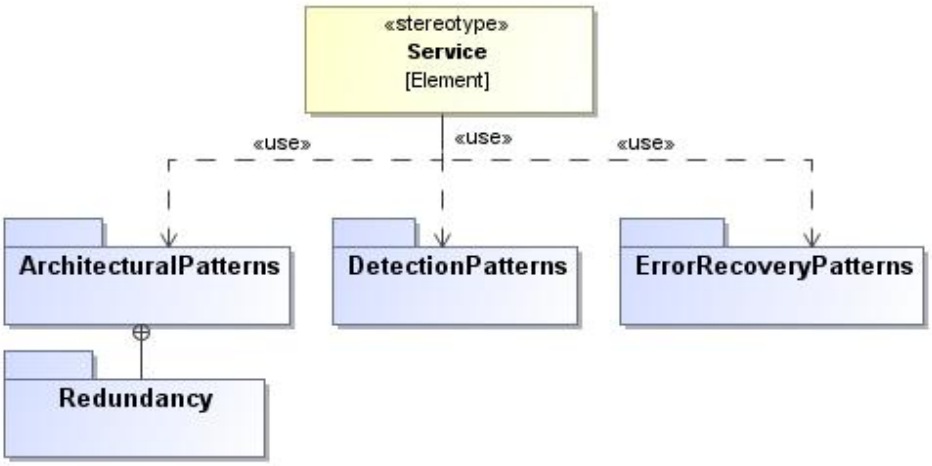
4.1 ยูเอ็มแอลโปรไฟล์สำหรับแบบรูปการทนต่อความผิดพร่องสำหรับระบบอิงบริการ

จากการศึกษากลุ่มการทำงานของแบบรูป คือ แบบรูปเชิงสถาปัตยกรรม แบบรูปการตรวจหา และแบบรูปการกู้ระบบจากข้อผิดพลาด สามารถแสดงได้ดังภาพที่ 4.1



ภาพที่ 4.1 กลุ่มของแบบรูปการทนต่อความผิดพร่อง

จากภาพที่ 4.1 แสดงกลุ่มของแบบรูปต่าง ๆ ซึ่งสามารถนำแบบรูปการทนต่อความผิดพร่องในแต่ละกลุ่มไปประยุกต์ใช้กับเซอร์วิซของระบบอิงบริการ โดยที่สเตอรีโอไทพ์ Service คือสเตอรีโอไทพ์ที่แสดงถึงเซอร์วิซที่สามารถนำแบบรูปไปประยุกต์ใช้ ซึ่งสามารถประยุกต์ใช้แบบรูปได้ทั้ง 3 กลุ่ม คือ แบบรูปเชิงสถาปัตยกรรม แบบรูปการตรวจหาข้อผิดพลาด และแบบรูปการกู้ระบบจากข้อผิดพลาด ดังแสดงในภาพที่ 4.2



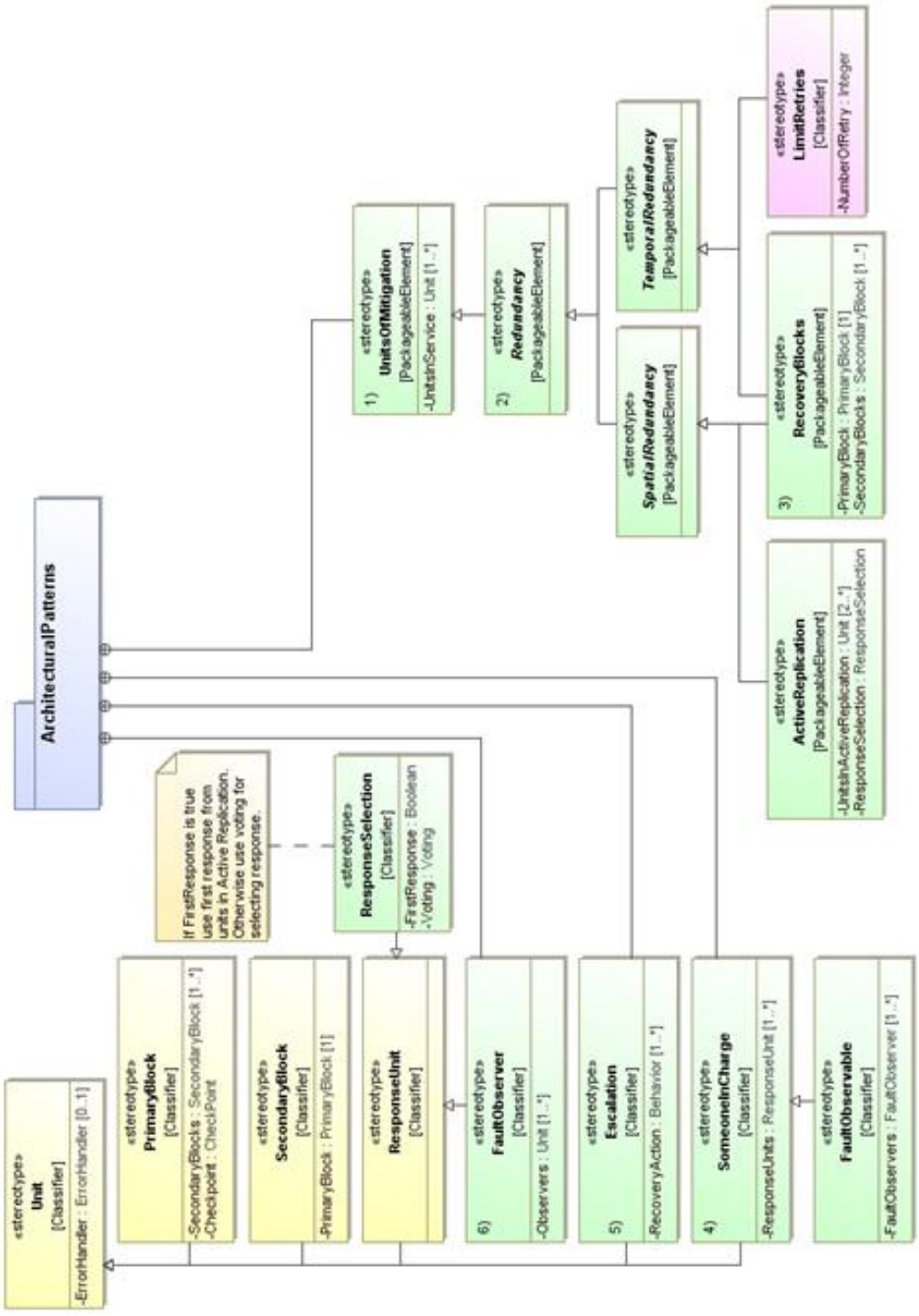
ภาพที่ 4.2 การประยุกต์ใช้แบบรูปการทนต่อความผิดพร่องกับระบบอิงบริการ

จากการวิเคราะห์แบบรูปการทนต่อความผิดพร่อง นำมาสร้างเป็นยูเอ็มแอลโปรไฟล์สำหรับแบบรูปการทนต่อความผิดพร่อง โดยออกแบบในรูปแบบของสเตอรีโอไทพ์และสเตอรีโอไทพ์เชิงนามธรรม ซึ่งสเตอรีโอไทพ์เป็นส่วนที่สามารถนำไปประยุกต์ใช้เข้ากับระบบอิงบริการ แต่สเตอรีโอไทพ์เชิงนามธรรมเป็นเพียงส่วนที่ช่วยในการจัดกลุ่มของสเตอรีโอไทพ์

การนำเสนอรายละเอียดของแบบรูปจะนำแนวทางการอธิบาย Design Patterns ที่นำเสนอโดย Erich Gamma และคณะ [17] มาปรับใช้ เพื่ออธิบายรายละเอียดที่จำเป็นต่อการนำเสนอยูเอ็มแอลโปรไฟล์สำหรับการทนต่อความผิดพร่องสำหรับระบบอิงบริการ โดยจะแสดงรายละเอียดยูเอ็มแอลโปรไฟล์ในกลุ่มดังต่อไปนี้

4.1.1 ยูเอ็มแอลโปรไฟล์ในกลุ่มของแบบรูปเชิงสถาปัตยกรรม

จากการวิเคราะห์แบบรูปเชิงสถาปัตยกรรม ซึ่งจำนวนแบบรูปที่สามารถนำมาออกแบบเป็นยูเอ็มแอลโปรไฟล์ คือ 6 แบบรูป โดยแบบรูปทั้งหมดแสดงดังภาพที่ 4.3



ภาพที่ 4.3 ยูเอ็มแอลโปรไฟล์สำหรับแบบรูปในกลุ่มของแบบรูปเชิงสถาปัตยกรรม

4.1.1.1 แบบรูป Units of Mitigation

Intent

แบ่งระบบออกเป็นหน่วยย่อยๆ เพื่อทำให้หน่วยย่อยๆนั้นมีการทนต่อความผิดพร่อง

Motivation

ในการออกแบบระบบให้มีการทนต่อความผิดพร่อง ผู้ออกแบบมักจะออกแบบให้มีการรองรับความผิดพร่องอย่างมีคุณภาพและเป็นมาตรฐาน เมื่อระบบมีข้อผิดพลาดเกิดขึ้น ระบบอาจจะหยุดทำงาน (Fail-Stop) หรือระบบอาจจะยังทำงานได้ (Fail-Safe) แต่อย่างไรก็ตามก็มีความเสี่ยงเนื่องจากการเกิดความผิดพร่องในส่วนใดส่วนหนึ่งของระบบอาจจะทำให้ระบบทั้งหมดไม่สามารถใช้งานได้

แบบรูป Units of Mitigation จึงมีจุดประสงค์เพื่อให้เซอร์วิซมีการออกแบบโดยการแบ่งเป็นหน่วยย่อยหรือเรียกว่า Unit ซึ่งแต่ละหน่วยย่อยสามารถเป็นหน่วยย่อยที่ทำงานแบบเดียวกันหรือทำงานแตกต่างกัน เพื่อทำการรองรับการทนต่อความผิดพร่องของเซอร์วิซ และเมื่อเกิดข้อผิดพลาดจะไม่กระทบต่อทั้งระบบ และไม่จำเป็นต้องรอให้หน่วยย่อยที่เกิดข้อผิดพลาดใช้งานได้ก่อน แต่จะสามารถไปทำส่วนอื่นได้ทันที

Applicability

แบบรูป Units of Mitigation จะถูกใช้เมื่อต้องการระบุหน่วยย่อยของการจัดการกับข้อผิดพลาด

Structure

โครงสร้างของแบบรูปนี้นำเสนอในเลขที่ 1) จากภาพที่ 4.3

Participants

- Sterotype UnitsOfMitigation สามารถประยุกต์เข้ากับเมตาโมเดล PackageableElement
    - *UnitsInService* กำหนดรายการหน่วยย่อยทั้งหมดของเซอร์วิซ โดยจะต้องมีจำนวนหน่วยย่อยอย่างน้อย 1 หน่วย

Collaborations

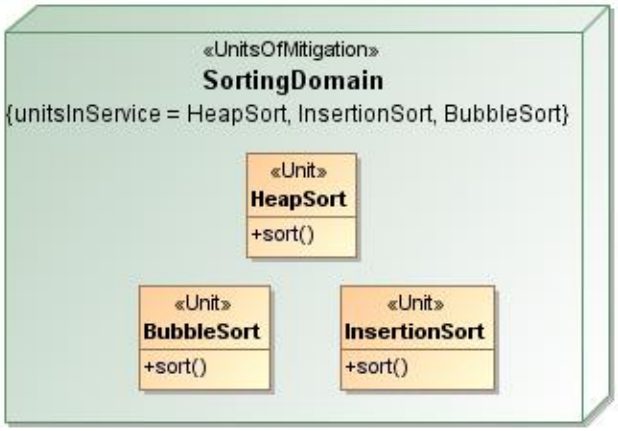
แบบรูป Units of Mitigation ไม่มีการนำเสนอพฤติกรรมการทำงานร่วมกัน เนื่องจากเป็นแบบรูปพื้นฐานทางสถาปัตยกรรม

Consequences

การใช้งานแบบรูป Units of Mitigation มีข้อดีคือทำให้เมื่อมีข้อผิดพลาดเกิดขึ้นในหน่วยย่อยหนึ่ง การจัดการจะอยู่ภายในหน่วยย่อยนั้นและจะไม่ส่งผลกระทบต่อการทำงานของระบบทั้งหมด แต่อาจส่งผลกระทบต่อสมรรถนะของระบบในการจัดการข้อผิดพลาดภายในหน่วยย่อยต่างๆ

Implementation

การประยุกต์ใช้แบบรูป Units of Mitigation เข้ากับตัวอย่าง SortingDomain โดยมีการแบ่งระบบออกเป็นส่วนย่อยๆ คือ HeapSort, BubbleSort และ InsertionSort แสดงดังภาพที่ 4.4



ภาพที่ 4.4 ตัวอย่างการใช้งานแบบรูป Units of Mitigation

Related Patterns

Redundancy, Escalation

4.1.1.2 แบบรูป Redundancy

Intent

ออกแบบระบบให้มีตัวสำรองที่สามารถทำงานแทนเมื่อเกิดข้อผิดพลาด

Motivation

การออกแบบระบบจะต้องทำให้ระบบมีการทนต่อความผิดพร่องและมีช่วงเวลาที่ระบบสามารถใช้งานได้มากที่สุด โดยช่วงเวลาที่ระบบไม่สามารถใช้งานได้เนื่องจากมีข้อผิดพลาดเกิดขึ้นและช่วงเวลาในการกู้ระบบให้กลับมาใช้งานได้เหมือนเดิมจะต้องสั้นที่สุด

แบบรูป Redundancy จึงมีจุดประสงค์เพื่อให้มีการจัดเตรียมตัวสำรอง เพื่อมาทำงานแทนเมื่อระบบเกิดข้อผิดพลาด ซึ่งแบบรูป Redundancy จะประกอบด้วย 2 รูปแบบ คือ การทำซ้ำในเชิงพื้นที่ (Spatial Redundancy) และ การทำซ้ำในเชิงเวลา (Temporal Redundancy)

- การทำซ้ำในเชิงพื้นที่ คือ การมีตัวสำรองที่ทำงานเหมือนกันอยู่ในหลายสถานที่ เมื่อเกิดข้อผิดพลาดจะสามารถนำตัวสำรองมาทำงานแทนได้

- การทำซ้ำในเชิงเวลา คือ การทำงานซ้ำของตัวสำรอง ในเวลาที่เป็นลำดับต่อเนื่องกัน

Applicability

แบบรูป Redundancy จะถูกใช้เมื่อระบบมีหน่วยย่อยเพื่อทำงานแทนในกรณีที่ระบบมีข้อผิดพลาดเกิดขึ้น

Structure

โครงสร้างของแบบรูปนี้นำเสนอในเลขที่ 2) จากภาพที่ 4.3

Participants

- Abstract Stereotype Redundancy สามารถประยุกต์เข้ากับเมตาโมเดล PackageableElement

Collaborations

แบบรูป Redundancy ไม่มีการนำเสนอพฤติกรรมการทำงานร่วมกัน เนื่องจากเป็นแบบรูปที่ใช้ในการออกแบบลักษณะของหน่วยหนึ่งในระบบ

Consequences

การใช้งานแบบรูป Redundancy มีข้อดีคือมีตัวสำรองเพื่อมารองรับการทำงานของระบบเมื่อมีข้อผิดพลาดเกิดขึ้น แต่อาจส่งผลกระทบต่อสมรรถนะของระบบในการจัดการกับตัวสำรองนั้นๆ

Implementation

แบบรูป Redundancy ไม่ถูกนำไปประยุกต์ใช้กับระบบได้โดยตรงเนื่องจากเป็นแบบรูปเชิงนามธรรมที่ช่วยในการจัดกลุ่มแบบรูปเท่านั้น

Related Patterns

Recovery Blocks, Failover, Voting

จากแบบรูป Redundancy สามารถแยกเป็นแบบรูปย่อยที่ไม่ได้มีการนำเสนอในแบบรูปสำหรับซอฟต์แวร์การทนต่อความผิดพร่องของ Hanmer [5] คือแบบรูป Active Replication ซึ่งมีรายละเอียดดังต่อไปนี้

แบบรูป Active Replication

Intent

ออกแบบระบบให้มีตัวสำรองในระบบ โดยที่ตัวสำรองทุกตัวจะทำงานพร้อมกัน

Motivation

การออกแบบระบบให้มีการทนต่อความผิดพร่อง โดยการออกแบบระบบให้มีตัวสำรองเพื่อให้ตัวสำรองแต่ละตัวสามารถทำงานแทนกันได้และสามารถแน่ใจได้ว่าผลลัพธ์ที่ได้จากตัวสำรองเหล่านั้นมีโอกาสถูกต้องมากกว่าการทำงานโดยไม่มีตัวสำรอง

แบบรูป Active Replication ซึ่งเป็นรูปแบบหนึ่งในการทำซ้ำในเชิงพื้นที่ มีจุดประสงค์เพื่อให้มีการจัดเตรียมตัวสำรอง และให้ตัวสำรองเหล่านั้นทำงานพร้อมกัน และมีการเลือกคำตอบที่เหมาะสมจากตัวสำรองเหล่านั้น โดยใช้วิธีการเลือกตามที่ผู้ออกแบบระบบต้องการ

Applicability

แบบรูป Active Replication จะถูกใช้เมื่อให้ระบบมีหน่วยย่อยเพื่อทำงานแทนเมื่อระบบมีข้อผิดพลาดเกิดขึ้นและต้องการให้ทุกหน่วยมีการทำงานแบบขนานกัน

Structure

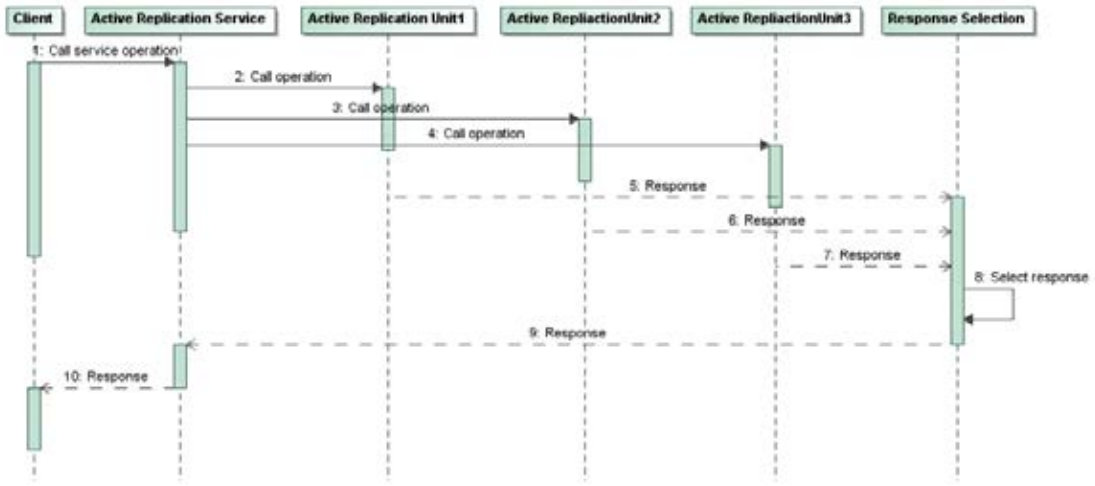
โครงสร้างของแบบรูปนี้นำเสนอในสเตอรีโอไทพ์ ActiveReplication จากภาพที่ 4.3

Participants

- Stereotype ActiveReplication สามารถประยุกต์เข้ากับเมตาโมเดล PackageableElement
  - *UnitsInActiveReplication* กำหนดหน่วยย่อยทั้งหมดที่ต้องการให้ทำงานพร้อมกัน
  - *ResponseSelection* กำหนดหน่วยที่ทำหน้าที่เป็นตัวเลือกคำตอบที่เหมาะสมจากคำตอบที่ได้มาจากหน่วยย่อยทั้งหมด โดยผู้เลือกคำตอบจะต้องมีชนิดเป็น ResponseSelection
- Stereotype ResponseSelection สามารถประยุกต์เข้ากับเมตาโมเดล Classifier
  - *FirstResponse* กำหนดว่าจะเลือกคำตอบที่ตอบมาเป็นตัวแรกเป็นคำตอบที่เหมาะสม
  - *Voting* กำหนดวิธีการเลือกคำตอบที่เหมาะสม โดยมีชนิดเป็นคลาส Voting

Collaborations

การทำงานของแบบรูป Active Replication เริ่มจากหน่วยที่ทำหน้าที่จัดการการทำ Active Replication หรือ Active Replication Service จะเรียกหน่วยย่อยทั้งหมดที่กำหนดไว้ใน UnitsInActiveReplication เช่น มี 3 หน่วยย่อย คือ Active Replication Unit (1), Active Replication Unit (2) และ Active Replication Unit (3) ให้มาทำงานพร้อมกัน เมื่อได้รับคำตอบจากหน่วยย่อยทั้งหมด จากนั้น Response Selection จะทำหน้าที่เป็นผู้เลือกคำตอบที่เหมาะสมและตอบกลับไปยังผู้ร้องขอ แสดงดังภาพที่ 4.5



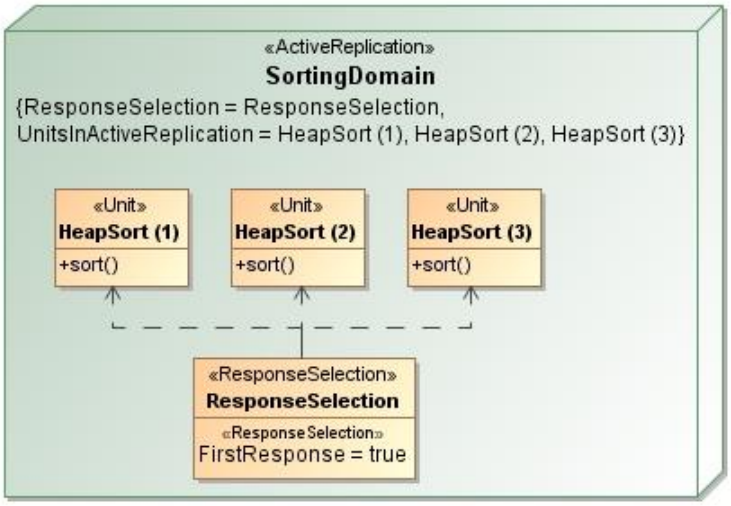
ภาพที่ 4.5 แผนภาพซีเควนซ์การทำงานตามแบบรูป Active Replication

Consequences

การใช้งานแบบรูป Active Replication มีข้อดีคือระบบจะยังสามารถหาผลลัพธ์ของการร้องขอได้ ในกรณีที่มีการเลือกคำตอบที่เหมาะสมจากคำตอบแรกที่ตอบมา แม้ว่ามีข้อผิดพลาดเกิดขึ้นที่หน่วยใดหน่วยหนึ่งในจำนวนหน่วยที่ทำหน้าที่ประมวลผลพร้อมกัน นอกจากนี้หากมีการพิจารณาจากคำตอบที่ได้มาจากหลายๆหน่วยในกรณีที่มีการใช้วิธีในการเลือกคำตอบที่เหมาะสมจากคำตอบทั้งหมด จะทำให้มั่นใจได้ว่าคำตอบที่ได้มาจะมีความถูกต้องแม่นยำสูง อย่างไรก็ตามการทำงานตามแบบรูปนี้อาจส่งผลกระทบต่อสมรรถนะของระบบในการจัดการกับหน่วยย่อยต่างๆและเสียเวลาในการคัดเลือกคำตอบที่เหมาะสม ถ้าหากมีจำนวนหน่วยที่ต้องเลือกคำตอบจำนวนมาก

Implementation

การประยุกต์ใช้แบบรูป Active Replication เข้ากับตัวอย่าง SortingDomain โดยระบบประกอบด้วยหน่วยย่อย HeapSort (1), HeapSort (2) และ HeapSort (3) ทำงานพร้อมกัน และมี ResponseSelection เป็นหน่วยในการเลือกคำตอบที่เหมาะสม แสดงดังภาพที่ 4.6



ภาพที่ 4.6 ตัวอย่างการใช้งานแบบรูป Active Replication

Related Patterns

Voting

4.1.1.3 แบบรูป Recovery Blocks

Intent

มีการกำหนดตัวสำรองที่จะทำงานแทนเมื่อตัวหลักเกิดข้อผิดพลาด โดยจะกำหนดเป็นลำดับของการทำงานแทนไว้

Motivation

หากมีความผิดพร่องที่ซ่อนอยู่ในระบบ เมื่อระบบมีการทำงานด้วยสถานะเดิมและข้อมูลชุดเดิมๆจะมีโอกาสกระตุ้นให้มีข้อผิดพลาดเดิมๆเกิดขึ้นได้

แบบรูป Recovery Blocks จึงมีการกำหนดตัวสำรองมาทำงานแทนเมื่อมีข้อผิดพลาดเกิดขึ้น เพื่อให้ระบบไม่ติดกับสถานะเดิม โดยที่แบบรูป Recovery Blocks จะมีลักษณะเป็นการทำซ้ำในเชิงพื้นที่ (Spatial Redundancy) และมีลักษณะการทำซ้ำในเชิงเวลา (Temporal Redundancy) ซึ่งแบบรูปนี้ประกอบไปด้วยส่วนปฐมภูมิ (Primary Block) และส่วนทุติยภูมิ (Secondary Blocks) เมื่อเริ่มทำงานจะเริ่มทำในส่วนปฐมภูมิก่อน และเมื่อมีข้อผิดพลาดเกิดขึ้นจะเปลี่ยนไปทำงานในส่วนของทุติยภูมิตามลำดับ ยูเอ็มแอลโปรไฟล์นี้จะต้องระบุหน่วยย่อยที่ทำหน้าที่เป็นส่วนทุติยภูมิ

Applicability

แบบรูป Recovery Blocks จะถูกใช้เมื่อต้องการกำหนดหน่วยย่อยในระบบเพื่อทำงานแทนหน่วยย่อยหนึ่งในระบบซึ่งมีข้อผิดพลาดเกิดขึ้น และต้องกำหนดลำดับการทำงานของแต่ละหน่วยย่อยอย่างชัดเจน

Structure

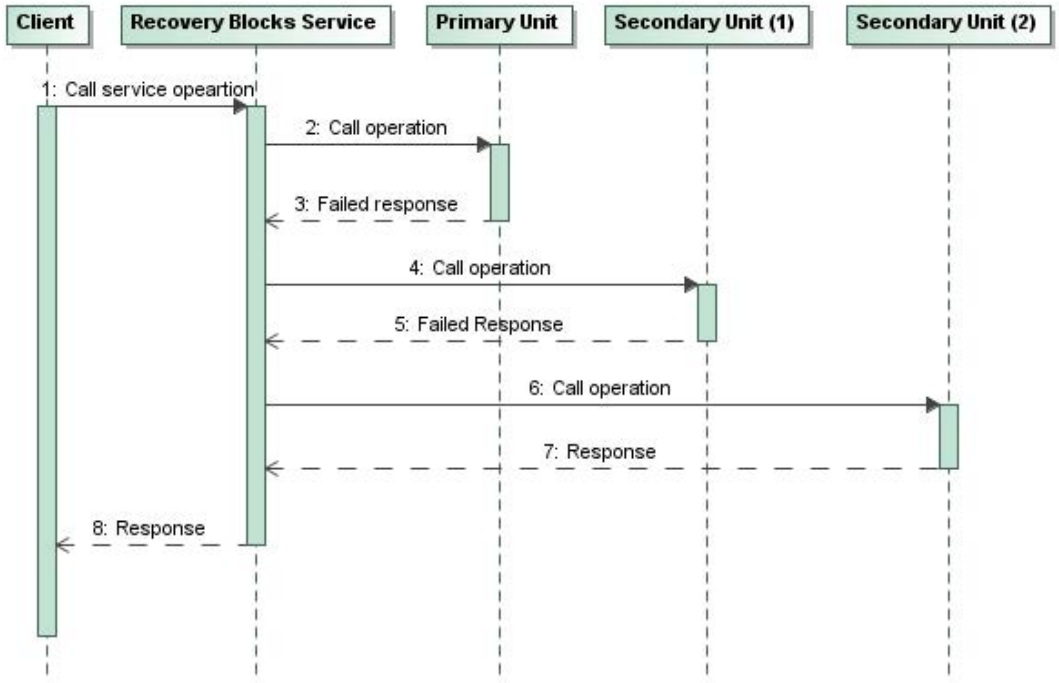
โครงสร้างของแบบรูปนี้นำเสนอในเลขที่ 3) จากภาพที่ 4.3

Participants

- Stereotype RecoveryBlocks สามารถประยุกต์เข้ากับเมตาโมเดล PackageableElement
    - *PrimaryBlock* กำหนดหน่วยย่อยที่ทำหน้าที่เป็นส่วนปฐมภูมิ ซึ่งมีจำนวน 1 หน่วย
    - *SecondaryBlocks* กำหนดรายการหน่วยย่อยที่ทำหน้าที่เป็นส่วนทุติยภูมิ ซึ่งมีจำนวนอย่างน้อย 1 หน่วย

Collaborations

การทำงานของแบบรูป Recovery Blocks เริ่มจากหน่วยที่ทำหน้าที่จัดการการทำ Recovery Blocks หรือ Recovery Blocks Service จะเรียกหน่วยที่ทำหน้าที่เป็น Primary Unit ให้มาทำงานก่อน ถ้า Primary Unit มีคข้อผิดพลาดเกิดขึ้น จะเรียก Secondary Unit ขึ้นมาทำงานตามลำดับ แสดงดังภาพที่ 4.7



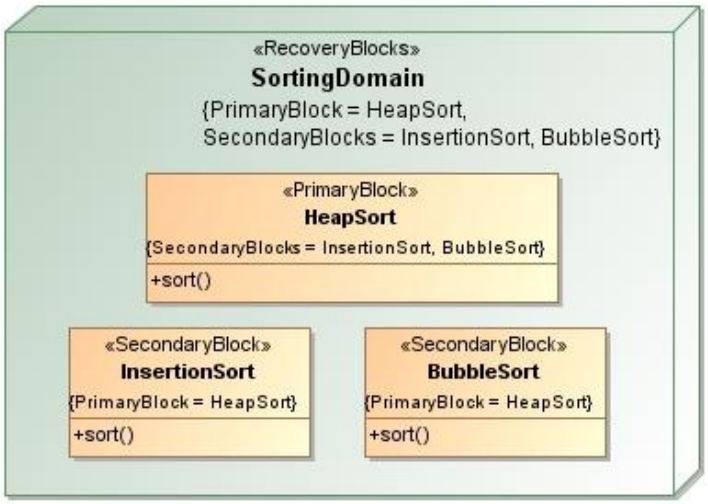
ภาพที่ 4.7 แผนภาพซีเควนซ์การทำงานตามแบบรูป Recovery Blocks

Consequences

การใช้งานแบบรูป Recovery Blocks มีข้อดีคือระบบจะมีตัวสำรองมาเพื่อรองรับเมื่อตัวหลักหรือตัวสำรองตัวใดตัวหนึ่งมีข้อผิดพลาดเกิดขึ้นและสามารถกำหนดได้อย่างชัดเจนว่าจะให้ตัวสำรองใดทำงานก่อนหลัง แต่อาจส่งผลกระทบต่อสมรรถนะของระบบในการจัดการกับหน่วยย่อยต่างๆและใช้เวลาในการทำงานเพิ่มเนื่องจากหน่วยย่อยแต่ละตัวจะทำงานต่อกันไปเรื่อยๆ

Implementation

การประยุกต์ใช้แบบรูป Recovery Blocks เข้ากับตัวอย่าง SortingDomain ประกอบด้วยส่วนที่เป็น PrimaryUnit คือ HeapSort และส่วนที่เป็น SecondaryUnit คือ InsertionSort และ BubbleSort ตามลำดับ แสดงดังภาพที่ 4.8



ภาพที่ 4.8 ตัวอย่างการใช้งานแบบรูป Recovery Blocks

Related Patterns

ไม่มี

4.1.1.4 แบบรูป Someone in Charge

Intent

ระบบจะต้องมีหน่วยที่ทำหน้าที่ควบคุมและจัดการระบบเมื่อมีข้อผิดพลาดเกิดขึ้น

Motivation

เมื่อมีการออกแบบระบบให้มีการทนต่อความผิดพร่อง ในระบบจะประกอบไปด้วยหลายๆ ส่วนตามแบบรูป Units of Mitigation และ Redundancy และต้องมีการออกแบบให้ระบบมีการจัดการกับข้อผิดพลาดแบบอัตโนมัติ ซึ่งกระบวนการจัดการกับข้อผิดพลาดถือเป็นกระบวนการสำคัญและต้องใช้เวลาในการจัดการ

แบบรูป Someone in Charge จึงมีการกำหนดหน่วยย่อยเพื่อเป็นหน่วยรับผิดชอบเกี่ยวกับการจัดการข้อผิดพลาดที่เกิดขึ้นในระบบ โดยที่หน่วยรับผิดชอบนั้นจะต้องมีหน้าที่ในการเฝ้าสังเกตหรือควบคุมหน่วยย่อยอื่นๆในระบบว่ามีความผิดพร่องเกิดขึ้นหรือไม่ หน่วยรับผิดชอบในระบบสามารถมีได้หลายหน่วย และต้องมีการกำหนดว่าแต่ละหน่วยรับผิดชอบนั้นคอยเฝ้าสังเกตหน่วยย่อยใดบ้าง

Applicability

แบบรูป Someone in Charge จะถูกใช้เมื่อต้องการกำหนดหน่วยย่อยเพื่อรับผิดชอบงานใดงานหนึ่งภายในระบบ และต้องการระบุว่าหน่วยใดมีหน้าที่รับผิดชอบเกี่ยวกับการจัดการข้อผิดพลาดของหน่วยย่อยตัวนั้นๆ

Structure

โครงสร้างของแบบรูปนี้นำเสนอในเลขที่ 4) จากภาพที่ 4.3

Participants

- Stereotype SomeoneInCharge สามารถประยุกต์เข้ากับเมตาโมเดล Classifier
    - *ResponseUnits* กำหนดหน่วยรับผิดชอบที่ทำหน้าที่เฝ้าสังเกตต่อหน่วยย่อยนี้ ซึ่งหน่วยรับผิดชอบต้องมีจำนวนอย่างน้อย 1 หน่วยย่อย โดยที่หน่วยรับผิดชอบจะต้องมีชนิดเป็น ResponseUnit

Collaborations

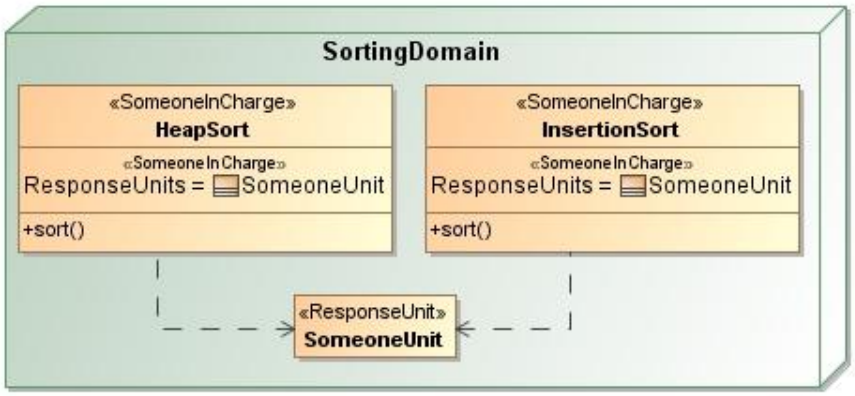
แบบรูป Someone in Charge ไม่มีการนำเสนอพฤติกรรมการทำงานร่วมกัน เนื่องจากเป็นแบบรูปที่ใช้ในการออกแบบลักษณะของหน่วยหนึ่งในระบบ

Consequences

การใช้งานแบบรูป Someone in Charge มีข้อดีคือระบบจะมีการกำหนดผู้รับผิดชอบอย่างชัดเจน เมื่อมีข้อผิดพลาดเกิดขึ้นในหน่วยย่อยนั้นจะได้มั่นใจว่ามีผู้รับผิดชอบ แต่อาจส่งผลกระทบต่อสมรรถนะของระบบเนื่องจากหน่วยที่รับผิดชอบอาจจะต้องมีการตรวจสอบหน่วยที่ถูกรับผิดชอบอย่างสม่ำเสมอ

Implementation

การประยุกต์ใช้แบบรูป Someone in Charge เข้ากับตัวอย่าง SortingDomain ประกอบด้วย SomeoneUnit ที่ทำหน้าที่เป็นหน่วยรับผิดชอบเกี่ยวกับการจัดการข้อผิดพลาด และ HeapSort และ InsertionSort มีการประยุกต์ใช้สเตอรีโอไทพ์ SomeoneInCharge เพื่อกำหนดหน่วยย่อยที่ทำงานในระบบโดยมีหน่วยรับผิดชอบสำหรับหน่วยย่อยนั้นคอยจัดการเกี่ยวกับข้อผิดพลาด แสดงดังภาพที่ 4.9



ภาพที่ 4.9 ตัวอย่างการใช้งานแบบรูป Someone in Charge

Related Patterns

System Monitor

4.1.1.5 แบบรูป Escalation

Intent

เมื่อการจัดการกับข้อผิดพลาดไม่สำเร็จ ควรจะเปลี่ยนไปทำวิธีที่เข้มขึ้น

Motivation

ในระบบที่มีการจัดการกับข้อผิดพลาดในหน่วยย่อยของระบบ กระบวนการจัดการกับข้อผิดพลาดนั้นจะต้องทำงานสำเร็จและให้ระบบสามารถกลับมาทำงานได้ตามปกติ ตัวอย่างเช่นถ้าการทำงานตามแบบรูป Correcting Audits ไม่สำเร็จ จากนั้นพยายามไปทำตามแบบรูป Rollback หรือ Roll-Forward และยังคงทำไม่สำเร็จ ระบบก็จะยังคงติดอยู่ในสถานการณ์ที่มีข้อผิดพลาด ดังนั้น ถ้าระบบมีการออกแบบให้จัดการกับข้อผิดพลาดแบบอัตโนมัติ ควรจะมีการกำหนดให้หน่วยใดหน่วยหนึ่งเป็นผู้รับผิดชอบต่อไปตามแบบรูป Someone in Charge

แบบรูป Escalation จึงมีการกำหนดวิธีการจัดการกับปัญหาที่เกิดขึ้น โดยจะต้องกำหนดวิธีการจัดการกับปัญหาที่เข้มขึ้นตามลำดับ เพื่อป้องกันปัญหาที่จะเกิดขึ้นซ้ำอีก

Applicability

แบบรูป Escalation จะถูกใช้เมื่อต้องการกำหนดวิธีการจัดการกับข้อผิดพลาดให้กับหน่วยใดหน่วยหนึ่ง เพื่อใช้ในขณะที่มีข้อผิดพลาดเกิดขึ้นในหน่วยนั้น

Structure

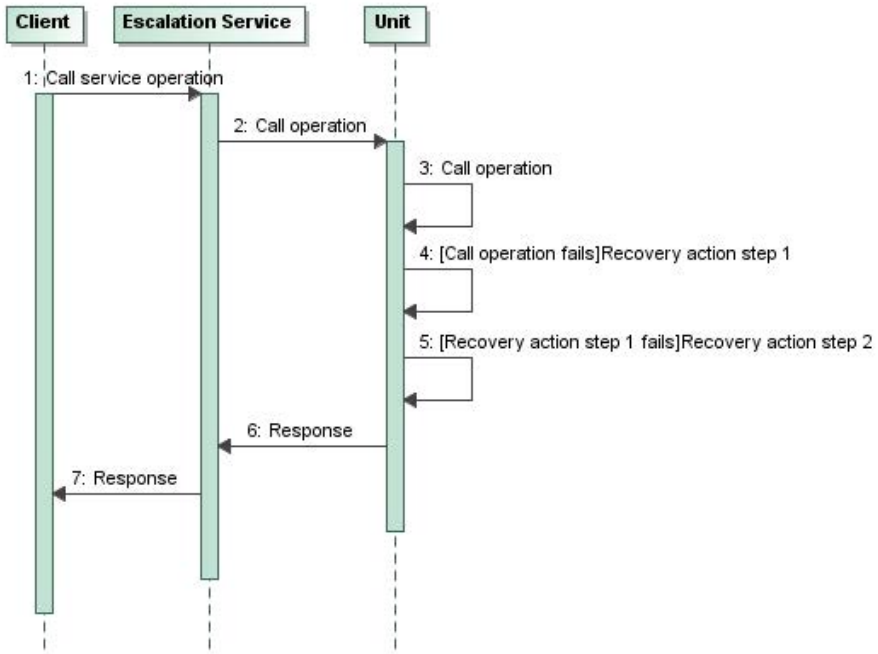
โครงสร้างของแบบรูปนี้นำเสนอในเลขที่ 5) จากภาพที่ 4.3

Participants

- Stereotype Escalation สามารถประยุกต์เข้ากับเมตาโมเดล Classifier
  - *RecoveryAction* กำหนดรายการวิธีการจัดการปัญหาจากเข้มน้อยไปยังเข้มมาก โดยที่จะต้องมีการกำหนดวิธีการจัดการปัญหาอย่างน้อย 1 วิธี

Collaborations

การทำงานของแบบรูป Escalation เมื่อมีข้อผิดพลาดเกิดขึ้นในส่วนของระบบ จะเริ่มจัดการกับข้อผิดพลาดด้วยวิธีที่ระบุไว้ในคุณสมบัติ RecoveryAction ตามลำดับ แสดงดังภาพที่ 4.10



ภาพที่ 4.10 แผนภาพซีเควนซ์การทำงานตามแบบรูป Escalation

Consequences

การใช้งานแบบรูป Escalation มีข้อดีคือระบบจะกำหนดการจัดการกับข้อผิดพลาดหลายระดับ เพื่อรองรับกับข้อผิดพลาดในระดับความรุนแรงที่แตกต่างกัน แต่อาจส่งผลกระทบต่อสมรรถนะของระบบเนื่องจากหากเกิดข้อผิดพลาดในระดับที่มีความรุนแรงมาก ระบบจะเริ่มจากการจัดการกับข้อผิดพลาดด้วยวิธีเบาที่สุดก่อนจะยกระดับการจัดการให้เข้มขึ้นตามลำดับ

Implementation

การประยุกต์ใช้แบบรูป Escalation เข้ากับตัวอย่าง SortingDomain มีการประยุกต์ใช้สเตอรีโอไทพ์ Escalation เข้ากับหน่วยย่อยของระบบคือ HeapSort และมีการกำหนดวิธีการยกระดับการจัดการกับข้อผิดพลาดตาม Recovery Plan A แสดงดังภาพที่ 4.11 และ ภาพที่ 4.12 ซึ่งหมายถึงหากยังไม่สามารถจัดการข้อผิดพลาดได้ ให้ยกระดับการจัดการเป็นการ Restart การทำงาน

ภาพที่ 4.11 ตัวอย่างการใช้งานแบบรูป Escalation

ภาพที่ 4.12 แผนภาพแอคทิวิตีสำหรับ Recovery Plan A

Related Patterns

ไม่มี

4.1.1.6 แบบรูป Fault Observer

Intent

เมื่อมีข้อผิดพลาดเกิดขึ้น จะต้องมีหน่วยที่เฝ้าสังเกตและรายงานไปยังผู้ที่สนใจความผิดพร่องนั้น

Motivation

ในระบบที่มีการออกแบบให้ทนต่อความผิดพร่อง ผู้ออกแบบจะต้องทราบว่าข้อผิดพลาดมีโอกาสที่จะเกิดขึ้นและต้องมีการเตรียมการสำหรับข้อผิดพลาดที่จะเกิดขึ้น โดยจะต้องมีผู้หนึ่งทำหน้าที่เฝ้าดูความผิดพร่องที่เกิดขึ้นในระบบ ตัวอย่างเช่น ให้มนุษย์เป็นผู้เฝ้าดูการทำงานของไมโครโพรเซสเซอร์ของตู้เย็น มนุษย์คงจะไม่ต้องการรับรายงานของการทำงานอย่างปกติ แต่อยากจะทราบว่าสาเหตุอะไรที่ทำให้การทำงานล้มเหลวมากกว่า ระบบที่ถูกออกแบบให้มีการตรวจจับและจัดการกับข้อผิดพลาดแบบอัตโนมัติก็เช่นเดียวกัน ต้องมีส่วนที่สนใจเกี่ยวกับความผิดพร่องที่เกิดขึ้นและความผิดพร่องอะไรบ้างที่ต้องแก้ไข

แบบรูป Fault Observer จึงมีการกำหนดตัวที่ทำหน้าที่สังเกตสถานะของหน่วยที่ถูกสังเกตและรายงานไปยังหน่วยที่มีหน้าที่จัดการ เมื่อหน่วยสังเกตได้รับการรายงานว่าหน่วยที่ถูกสังเกตในระบบมีสถานะที่เปลี่ยนไป จะทำการรายงานไปยังผู้สนใจต่อการเปลี่ยนแปลงนั้นๆ ที่ได้ทำการลงทะเบียนไว้

Applicability

แบบรูป Fault Observer จะถูกใช้เมื่อต้องการกำหนดหน่วยสังเกตความผิดพร่องที่เกิดขึ้นในระบบ และรายงานไปยังผู้เกี่ยวข้อง เพื่อการแก้ไขความผิดพร่องนั้นต่อไป

Structure

โครงสร้างของแบบรูปนี้นำเสนอในเลขที่ 6) จากภาพที่ 4.3

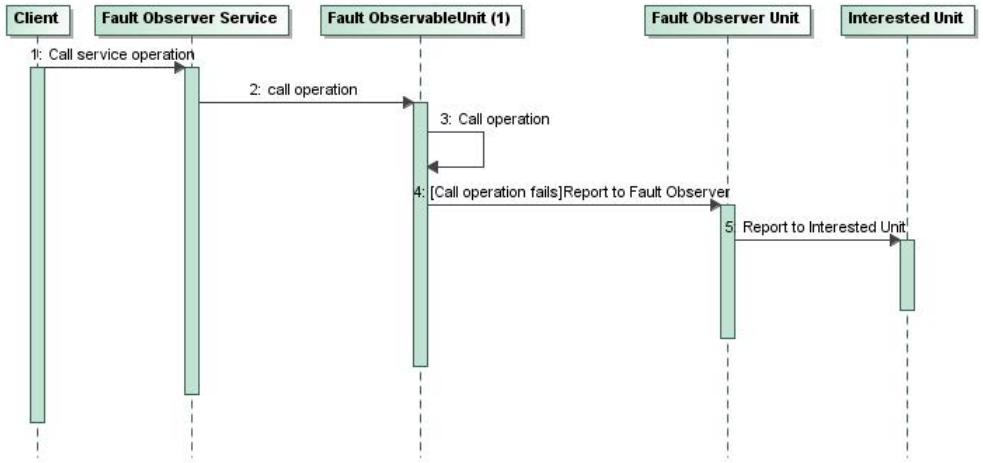
Participants

- Stereotype FaultObserver สามารถประยุกต์เข้ากับเมตาโมเดล Classifier
  - *Observers* กำหนดรายการหน่วยที่สนใจต่อสถานะที่เปลี่ยนแปลงในระบบ เมื่อระบบมีสถานะที่เปลี่ยนแปลง ผู้สังเกต หรือ FaultObserver จะรายงานไปยังรายการหน่วยที่สนใจทั้งหมด

- Stereotype FaultObservable สามารถประยุกต์เข้ากับเมตาโมเดล Classifier
    - *FaultObservers* กำหนดรายการหน่วยสังเกต ที่เป็นผู้สังเกตต่อหน่วยนี้ เมื่อมีการเปลี่ยนแปลงสถานะจะทำการรายงานไปยังรายการหน่วยสังเกตทั้งหมด โดยที่จะต้องมีการกำหนดหน่วยสังเกตอย่างน้อย 1 หน่วย

Collaborations

การทำงานของแบบรูป Fault Observer เมื่อมีข้อผิดพลาดเกิดขึ้นในส่วนของระบบ จะทำการรายงานไปยังหน่วยที่ทำหน้าที่เป็นหน่วยสังเกตหรือ Fault Observer Unit จากนั้นหน่วยสังเกตจะรายงานต่อไปยังหน่วยที่สนใจต่อความผิดพร่องนั้นๆหรือ Interested Unit เพื่อจัดการกับความผิดพร่อง แสดงดังภาพที่ 4.13



ภาพที่ 4.13 แผนภาพซีเควนซ์การทำงานตามแบบรูป Fault Observer

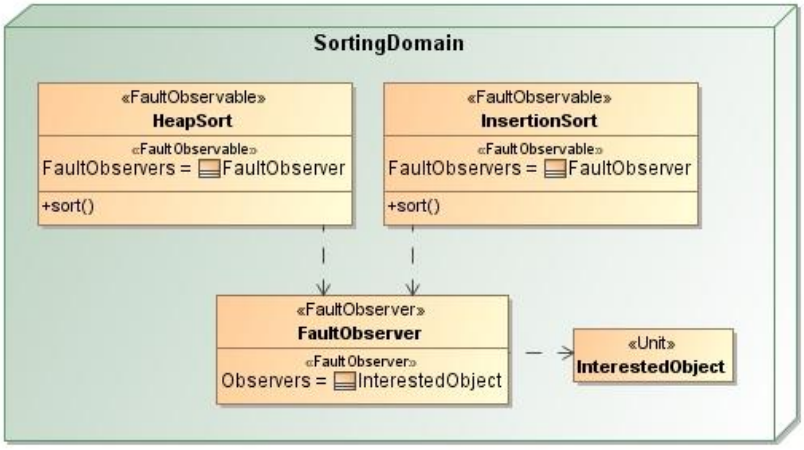
Consequences

การใช้งานแบบรูป Fault Observer มีข้อดีคือระบบจะทราบว่ามีข้อผิดพลาดเกิดขึ้นในระบบ เพื่อนำไปสู่การจัดการกับข้อผิดพลาดนั้นต่อไป แต่อาจส่งผลกระทบทำให้สมรรถนะของระบบแย่ลงเนื่องจากระบบจะมีความซับซ้อนเพิ่มมากขึ้นในการทำงาน

Implementation

การประยุกต์ใช้แบบรูป Fault Observer เข้ากับตัวอย่าง Sorting Domain มีการประยุกต์ใช้สเตอรีโอไทพ์ FaultObserver เข้ากับหน่วยย่อย FaultObserver เพื่อทำหน้าที่เป็นผู้สังเกต และระบุ InterestedObject เพื่อทำหน้าที่เป็นหน่วยสนใจ เมื่อ FaultObserver ได้รับทราบการเปลี่ยนแปลงที่เกิดขึ้นในระบบจากหน่วยที่ถูกสังเกต จะทำการรายงานไปยังหน่วยสนใจหรือ

InterestedObject ซึ่งจากตัวอย่างนี้หน่วยที่ทำหน้าที่เป็นหน่วยที่ถูกสังเกต หรือหน่วยที่ประยุกต์ใช้สเตอรีโอไทพ์ FaultObservable คือ HeapSort และ InsertionSort ดังภาพที่ 4.14



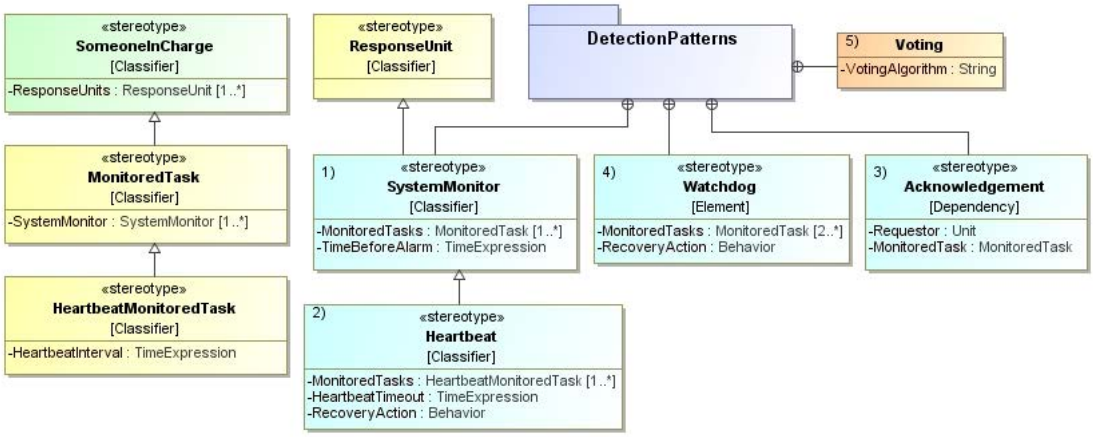
ภาพที่ 4.14 ตัวอย่างการใช้งานแบบรูป Fault Observer

Related Patterns

ไม่มี

4.1.2 ยูเอ็มแอลโปรไฟล์ในกลุ่มของแบบรูปการตรวจหาข้อผิดพลาด

จากการวิเคราะห์แบบรูปการตรวจหาข้อผิดพลาด ซึ่งจำนวนแบบรูปที่สามารถนำมาออกแบบเป็นยูเอ็มแอลโปรไฟล์ คือ 5 แบบรูป โดยแบบรูปทั้งหมดแสดงดังภาพที่ 4.15



ภาพที่ 4.15 ยูเอ็มแอลโปรไฟล์สำหรับแบบรูปในกลุ่มของแบบรูปการตรวจหาข้อผิดพลาด

4.1.2.1 แบบรูป System Monitor

Intent

ออกแบบระบบให้มีตัวควบคุมเพื่อตรวจสอบพฤติกรรมของระบบ

Motivation

ในการออกแบบระบบให้สามารถใช้งานได้อย่างต่อเนื่อง ต้องคำนึงถึงความล้มเหลวในระบบ ซึ่งอาจจะเกิดขึ้นแล้วทำให้ส่วนใดส่วนหนึ่งของระบบหยุดทำงานไปโดยไม่มีสัญญาณใดบ่งบอก จึงจำเป็นต้องมีการค้นหาและจัดการกู้ส่วนนั้นอย่างเร่งด่วน อาจจะมีการทำงานตามแบบรูป Fault Observer หรืออื่นๆ ดังนั้นจึงควรออกแบบให้มีส่วนหนึ่งเป็นตัวควบคุมเหตุการณ์เหล่านี้ที่เกิดขึ้น

แบบรูป System Monitor จึงมีการกำหนดหน่วยตรวจสอบ เพื่อทำการตรวจหาข้อผิดพลาดที่เกิดขึ้นภายในหน่วยต่างๆของระบบ โดยถ้าหน่วยต่างๆไม่มีการตอบรับมาในเวลาที่กำหนดจะถือว่าหน่วยนั้นมีข้อผิดพลาดเกิดขึ้น

Applicability

แบบรูป System Monitor จะถูกใช้เมื่อต้องการกำหนดหน่วยตรวจสอบ เพื่อตรวจหาข้อผิดพลาดที่เกิดขึ้นในหน่วยที่ต้องการตรวจสอบ

Structure

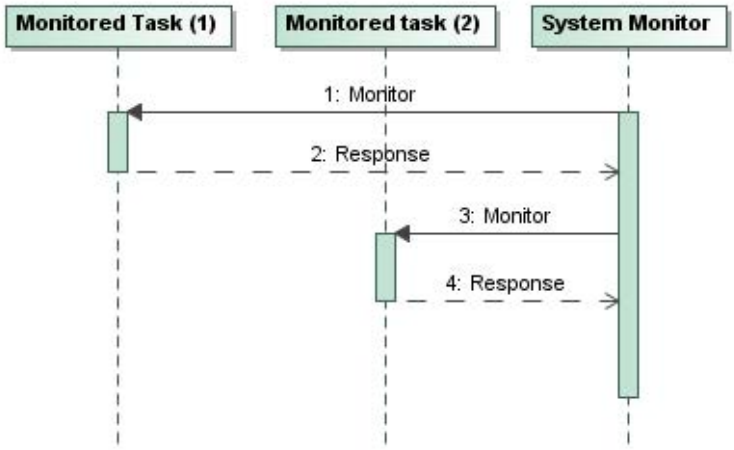
โครงสร้างของแบบรูปนี้นำเสนอในเลขที่ 1) จากภาพที่ 4.15

Participants

- Stereotype SystemMonitor สามารถประยุกต์เข้ากับเมตาโมเดล Classifier
  - *MonitoredTasks* กำหนดรายการหน่วยที่ถูกตรวจสอบโดยหน่วยตรวจสอบ ซึ่งจะมีชนิดเป็น MonitoredTask
  - *TimeBeforeAlarm* กำหนดระยะเวลาสำหรับให้ผู้ตรวจสอบทราบว่าหน่วยในระบบมีข้อผิดพลาดเกิดขึ้น ถ้าหน่วยที่ถูกตรวจสอบไม่มีการตอบสนองภายในเวลาที่กำหนด
- Stereotype MonitoredTask
  - *SystemMonitor* กำหนดรายการหน่วยตรวจสอบ ที่เป็นผู้ตรวจสอบสำหรับหน่วยนี้ โดยที่ต้องการกำหนดหน่วยตรวจสอบอย่างน้อย 1 หน่วย

Collaborations

การทำงานของแบบรูป System Monitor กำหนดหน่วยที่ทำหน้าที่ควบคุมคือ System Monitor ซึ่งจะคอยตรวจสอบข้อผิดพลาดที่เกิดขึ้นในหน่วยที่ถูกควบคุมหรือ Monitored Task แสดงดังภาพที่ 4.16



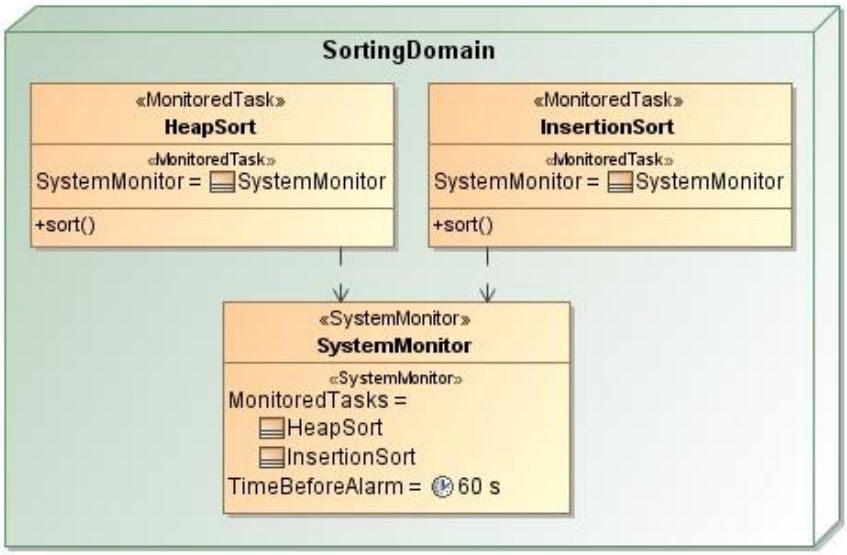
ภาพที่ 4.16 แผนภาพซีเควนซ์การทำงานตามแบบรูป System Monitor

Consequences

การใช้งานแบบรูป System Monitor มีข้อดีคือสามารถตรวจสอบและทราบได้ทันทีว่ามีข้อผิดพลาดเกิดขึ้นที่ส่วนใดของระบบ โดยที่ไม่ต้องคอยจนกว่าจะมีข้อผิดพลาดเกิดขึ้นแล้วจึงตรวจสอบ แต่อาจส่งผลกระทบทำให้สมรรถนะของระบบแย่ลงเนื่องจากจะต้องทำการตรวจสอบตลอดเวลา

Implementation

การประยุกต์ใช้แบบรูป System Monitor เข้ากับตัวอย่าง SortingDomain โดยมีการประยุกต์ใช้สเตอรีโอไทพ์ SystemMonitor เข้ากับหน่วยย่อยเพื่อทำหน้าที่เป็นตัวควบคุมในระบบ และมีการกำหนดหน่วยที่ถูกควบคุมซึ่งประยุกต์ใช้สเตอรีโอไทพ์ MonitoredTask คือ HeapSort และ InsertionSort และ SystemMonitor มีการกำหนดเวลาที่จะบอกแจ้งว่ามีข้อผิดพลาดเกิดขึ้นเมื่อไม่สามารถติดต่อหน่วยที่ถูกควบคุมได้ในเวลาที่กำหนด ดังภาพที่ 4.17



ภาพที่ 4.17 ตัวอย่างการใช้งานแบบรูป System Monitor

Related Patterns

Acknowledgement, Heartbeat, Watchdog

4.1.2.2 แบบรูป Heartbeat

Intent

ออกแบบระบบให้มีการส่งสถานะของตัวเองไป เพื่อแจ้งให้ทราบว่าตัวเองยังทำงานอยู่

Motivation

ทั้งความล้มเหลวที่เกิดขึ้นแล้วไม่มีสัญญาณใดๆหรือความล้มเหลวที่เกิดขึ้นและมีสัญญาณบ่งบอก อาจใช้เวลามากในการตรวจหา จึงควรย่นระยะเวลาในการตรวจหาข้อผิดพลาด

แบบรูป Heartbeat จึงมีการกำหนดหน่วยตรวจสอบ โดยที่หน่วยที่ถูกตรวจสอบจะมีการส่งสัญญาณชีพของตัวเองมา เพื่อแสดงว่าตัวเองยังทำงานอยู่ และเมื่อตรวจพบว่าหน่วยภายในระบบมีข้อผิดพลาดเกิดขึ้น หน่วยตรวจสอบจะต้องจัดการกับข้อผิดพลาดนั้น

Applicability

แบบรูป Heartbeat จะถูกใช้เมื่อต้องการให้หน่วยย่อยส่งสถานะของตัวเองไปยังหน่วยตรวจสอบที่ทำหน้าที่ควบคุมอยู่ โดยที่หน่วยที่ทำหน้าที่ควบคุมนี้ไม่ส่งข้อความมาถามสถานะ

Structure

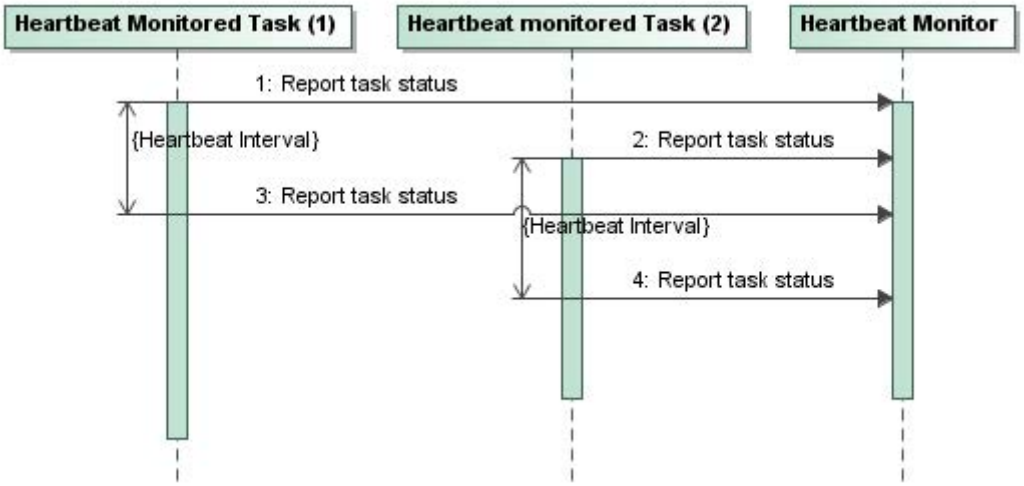
โครงสร้างของแบบรูปนี้นำเสนอในเลขที่ 2) จากภาพที่ 4.15

Participants

- Stereotype Heartbeat สามารถประยุกต์เข้ากับเมตาโมเดล Classifier
  - *MonitoredTasks* กำหนดรายการหน่วยที่ถูกตรวจสอบโดยหน่วยตรวจสอบ ซึ่งจะมีชนิดเป็น HeartbeatMonitoredTask
  - *HeartbeatTimeout* กำหนดช่วงเวลาว่ามีข้อผิดพลาดเกิดขึ้น ถ้าหน่วยที่ถูกตรวจสอบไม่ส่งสัญญาณชีพมาภายในช่วงเวลานี้
  - *RecoveryAction* กำหนดวิธีการจัดการกับข้อผิดพลาด เมื่อตรวจพบว่าระบบมีข้อผิดพลาดเกิดขึ้น
- Stereotype HeartbeatMonitoredTask สามารถประยุกต์เข้ากับเมตาโมเดล Classifier
  - *SystemMonitor* กำหนดรายการหน่วยตรวจสอบ ที่เป็นผู้ตรวจสอบสำหรับหน่วยนี้ โดยที่ต้องกำหนดหน่วยตรวจสอบอย่างน้อย 1 หน่วย คุณสมบัตินี้สืบทอดมาจากสเตอรีโอไทพ์ MonitoredTask
  - *HeartbeatInterval* กำหนดช่วงเวลาที่หน่วยนี้จะมีการส่งสัญญาณชีพของตัวเองไปยังรายการหน่วยตรวจสอบ

Collaborations

การทำงานของแบบรูป Heartbeat กำหนดหน่วยที่ทำหน้าที่ควบคุมการส่งสถานะคือ Heartbeat Monitor ซึ่งจะคอยตรวจสอบการส่งสถานะของหน่วยในระบบ แสดงดังภาพที่ 4.18



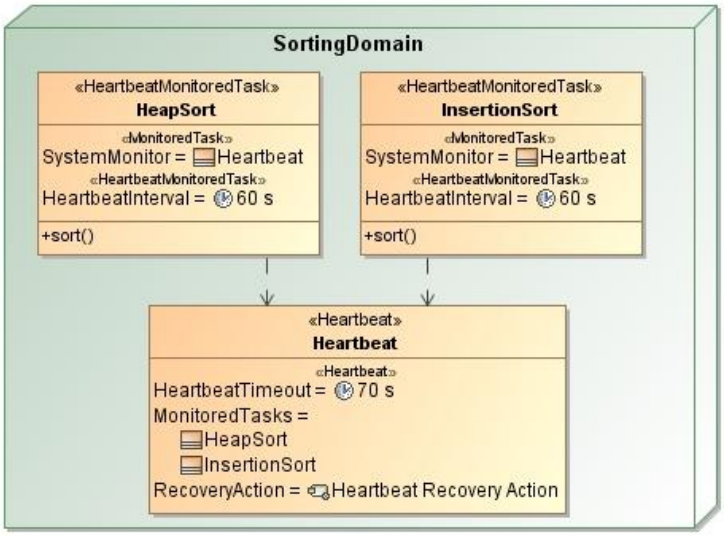
ภาพที่ 4.18 แผนภาพซีเควนซ์การทำงานตามแบบรูป Heartbeat

Consequences

การใช้งานแบบรูป Heartbeat มีข้อดีคือสามารถตรวจสอบและทราบได้ทันทีว่ามีข้อผิดพลาดเกิดขึ้นที่ส่วนใดของระบบ โดยใช้เวลาในการส่งข้อความน้อยกว่าแบบรูป System Monitor เนื่องจากไม่ต้องส่งข้อความไปถามสถานะก่อน แต่อาจส่งผลกระทบทำให้สมรรถนะของระบบแย่ลงเนื่องจากจะต้องทำการตรวจสอบสถานะที่ส่งมาตลอดเวลา

Implementation

การประยุกต์ใช้แบบรูป Heartbeat เข้ากับตัวอย่าง SortingDomain โดยมีการประยุกต์ใช้สเตอรีโอไทพ์ Heartbeat เข้ากับหน่วยที่ทำหน้าที่จัดการกับสัญญาณชีพ และมีการกำหนดหน่วยที่ถูกจัดการ ซึ่งหน่วยนั้นจะต้องมีการประยุกต์ใช้สเตอรีโอไทพ์ HeartbeatMonitoredTask โดยที่หน่วยที่ถูกจัดการนั้นจะต้องกำหนดช่วงเวลาที่ทำการส่งสัญญาณชีพไปยังหน่วยจัดการ ดังภาพที่ 4.19 และภาพที่ 4.20



ภาพที่ 4.19 ตัวอย่างการใช้งานแบบรูป Heartbeat

ภาพที่ 4.20 แผนภาพแอคทิวิตีสำหรับ Heartbeat Recovery Action

Related Patterns

ไม่มี

4.1.2.3 แบบรูป Acknowledgement

Intent

ออกแบบให้มีการส่งข้อความตอบรับข้อความที่ได้มา เพื่อแจ้งให้อีกฝ่ายทราบว่ายังทำงานอยู่

Motivation

ระบบที่ประกอบด้วยหน่วยถูกควบคุมโดยปกติแล้วจะมีการสื่อสารกับหน่วยควบคุมหรือหน่วยภายนอกที่เชื่อถือได้ ตัวอย่างเช่นแบบรูป Fault Observer ในการสื่อสารหากมีการส่งข้อความเพิ่มเติมจากข้อความที่ถูกส่งในการทำงานปกติของหน่วยถูกควบคุมจะเป็นการเพิ่มแบนด์วิดท์ในระบบและยังอาจจะทำให้การส่งข้อความสำคัญในระบบมีความล่าช้า แต่เนื่องจากการทำงานปกติจะต้องมีการส่งข้อความระหว่างผู้ร้องขอและถูกร้องขออยู่แล้ว และการส่งข้อความเหล่านั้นอาจมีการ

ตอบกลับหรือไม่มีก็ได้ ดังนั้นหากกำหนดให้การส่งข้อความ มีการตอบรับเสมอ การตอบรับจะเป็นสัญญาณที่บ่งบอกได้ว่าผู้ถูกร้องขอยังทำงานอยู่

แบบรูป Acknowledgement จึงมีการกำหนดให้ต้องมีการส่งข้อความตอบรับข้อความร้องขอกลับไปยังผู้ร้องขอทุกครั้ง แม้ว่าการส่งข้อความร้องขอนั้นจะต้องการคำตอบกลับหรือไม่ก็ตาม เพื่อเป็นการบ่งบอกว่าหน่วยนั้นยังคงทำงานอยู่ โดยที่ต้องมีการเพิ่มกระบวนการในการประมวลผลข้อความ ถ้าไม่มีข้อความตอบรับมาในเวลาที่กำหนดจะถือว่ามีข้อผิดพลาดเกิดขึ้น

Applicability

แบบรูป Acknowledgement จะถูกใช้เมื่อต้องการตรวจสอบสถานะของหน่วยย่อยโดยไม่ต้องการเพิ่มเติมส่วนอื่นเข้าไปในระบบแต่จะใช้เพียงการเพิ่มการตอบรับให้กับการส่งข้อความที่ไม่มีการตอบกลับเท่านั้น

Structure

โครงสร้างของแบบรูปนี้นำเสนอในเลขที่ 3) จากภาพที่ 4.15

Participants

- Stereotype Acknowledgement สามารถประยุกต์เข้ากับเมตาโมเดล Dependency
  - *Requestor* กำหนดหน่วยที่ทำหน้าที่เป็นผู้ร้องขอ ซึ่งจะมีกระบวนการทำการประมวลผลข้อความที่ได้รับมาจากผู้ถูกร้องขอ
  - *MonitoredTask* กำหนดหน่วยที่ทำหน้าที่เป็นผู้ถูกร้องขอ ซึ่งจะต้องมีกระบวนการในการส่งข้อความตอบรับไปยังผู้ร้องขอ

Collaborations

การทำงานของแบบรูป Acknowledgement กำหนดให้ผู้ถูกร้องขอส่งข้อความ acknowledge กลับมายังผู้ร้องขอ ถึงแม้ว่าการร้องขอนั้นตามปกติจะต้องการการตอบกลับหรือไม่ก็ตาม แสดงดังภาพที่ 4.21

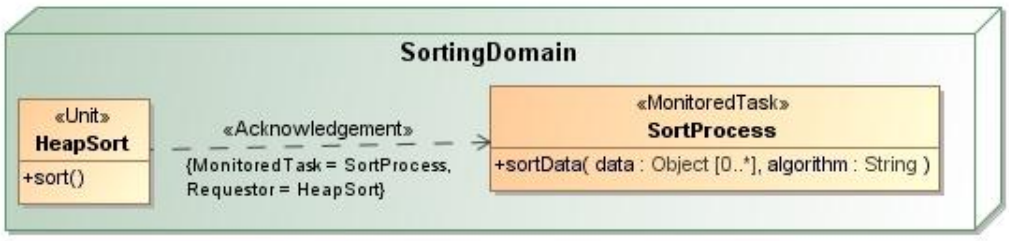
ภาพที่ 4.21 แผนภาพซีเควนซ์การทำงานตามแบบรูป Acknowledgement

Consequences

การใช้งานแบบรูป Acknowledgement มีข้อดีคือการตรวจสอบข้อผิดพลาดในระบบไม่จำเป็นต้องเพิ่มเติมส่วนพิเศษเข้าไปยังระบบ เพียงแต่เพิ่มข้อความ acknowledge เข้าไป แต่อาจส่งผลกระทบต่อสมรรถนะเนื่องจากต้องเพิ่มกระบวนการในการประมวลข้อความ acknowledge

Implementation

การประยุกต์ใช้แบบรูป Acknowledgement เข้ากับตัวอย่าง SortingDomain มีการประยุกต์ใช้สเตอรีโอไทพ์ Acknowledgement เข้ากับ Dependency ระหว่าง HeapSort และ SortProcess เพื่อระบุว่าการส่งข้อความระหว่างกันต้องมีการส่งข้อความตอบรับทุกครั้ง ดังภาพที่ 4.22



ภาพที่ 4.22 ตัวอย่างการใช้งานแบบรูป Acnowledgement

Related Patterns

ไม่มี

4.1.2.4 แบบรูป Watchdog

Intent

สร้างหน่วยหนึ่งขึ้นมาเพื่อ คอยเฝ้ามองการทำงานของระบบว่าทำได้อย่างปกติ

Motivation

แต่ละโดเมนของระบบย่อยต้องการการออกแบบการตรวจหาข้อผิดพลาดโดยใช้แบบรูปที่แตกต่างกันออกไป เช่นสามารถเพิ่มการส่งข้อความพิเศษเข้าไป ส่งข้อความเพิ่มเติมไปกับการร้องขอที่ต้องทำอยู่แล้ว หรือเพิ่มหน่วยพิเศษเข้าไปเพื่อทำหน้าที่ตรวจสอบ แต่การตรวจหาในลักษณะดังกล่าวอาจทำให้ผู้ออกแบบกังวลเกี่ยวกับความซับซ้อนที่เกิดขึ้นในซอฟต์แวร์เพราะอาจจะเป็นการลดความน่าเชื่อถือของระบบได้

แบบรูป Watchdog จึงมีการกำหนดฮาร์ดแวร์หรือซอฟต์แวร์ เพื่อทำหน้าที่เฝ้ามองการทำงานปกติในระบบ โดยหน่วยที่ถูกเฝ้ามองไม่จำเป็นต้องแก้ไขใดๆเลย หรือไม่ต้องเพิ่มการส่งข้อความพิเศษใดๆ

Applicability

แบบรูป Watchdog จะตรวจสอบข้อผิดพลาดในระบบโดยไม่ต้องการเพิ่มการส่งข้อความพิเศษในระบบ เพียงแต่เพิ่มหน่วยเฝ้ามองเข้าไป เพื่อเฝ้ามองการสื่อสารระหว่างหน่วยต่างๆในระบบ

Structure

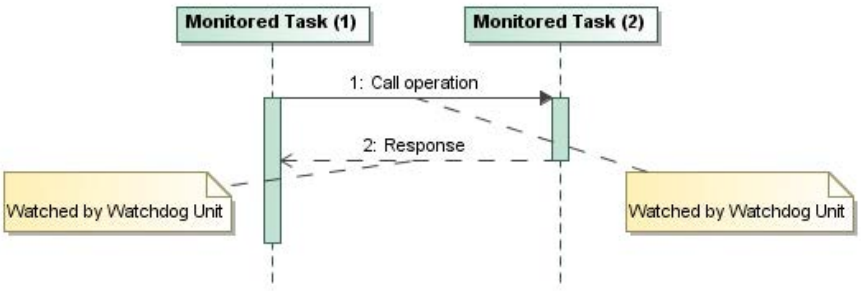
โครงสร้างของแบบรูปนี้นำเสนอในเลขที่ 4) จากภาพที่ 4.15

Participants

- Stereotype Watchdog สามารถประยุกต์เข้ากับเมตาโมเดล Element
  - *MonitoredTask* กำหนดรายการหน่วยที่ทำหน้าที่ถูกตรวจสอบโดย Watchdog โดยที่ต้องมีการกำหนดหน่วยที่ถูกตรวจสอบอย่างน้อย 2 หน่วยขึ้นไป
  - *RecoveryAction* กำหนดวิธีการจัดการกับข้อผิดพลาด เมื่อตรวจพบว่าระบบมีข้อผิดพลาดเกิดขึ้น

Collaborations

การทำงานของแบบรูป Watchdog กำหนดให้มีหน่วยเฝ้ามองหรือ Watchdog Unit เฝ้ามองการทำงานแบบปกติของระบบ แสดงดังภาพที่ 4.23



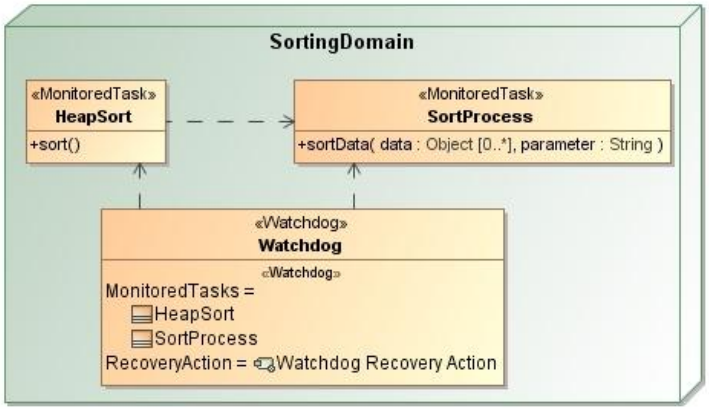
ภาพที่ 4.23 แผนภาพซีเควนซ์การทำงานตามแบบรูป Watchdog

Consequences

การใช้งานแบบรูป Watchdog มีข้อดีคือการตรวจสอบข้อผิดพลาดของระบบไม่จำเป็นต้องเปลี่ยนแปลงการทำงานใดๆในระบบ เพียงแต่เพิ่มส่วนเฝ้ามองไปในระบบ แต่อาจส่งผลกระทบต่อสมรรถนะเนื่องจากต้องเพิ่มส่วนคอยเฝ้ามอง อาจจะทำให้ระบบมีความซับซ้อนมากขึ้น

Implementation

การประยุกต์ใช้แบบรูป WatchDog เข้ากับตัวอย่าง SortingDomain โดยมีการประยุกต์ใช้สเตอรีโอไทพ์ Watchdog เข้ากับหน่วยที่ทำหน้าที่เฝ้ามอง เพื่อเฝ้ามองการทำงานของ HeapSort และ SortProcess ซึ่งมีการประยุกต์ใช้สเตอรีโอไทพ์ MonitoredTask และจะทำการจัดการกับระบบตาม Watchdog Recovery Action ที่กำหนดไว้ใน Watchdog เมื่อมีข้อผิดพลาดเกิดขึ้น ดังภาพที่ 4.24



ภาพที่ 4.24 ตัวอย่างการใช้งานแบบรูป Watchdog

Related Patterns

ไม่มี

4.1.2.5 แบบรูป Voting

Intent

ออกแบบให้มีกระบวนการในการโหวตคำตอบที่เหมาะสม เมื่อมีผลลัพธ์มากกว่า 1 คำตอบ

Motivation

เมื่อระบบมีการประยุกต์ใช้แบบรูป Redundancy บางรูปแบบของการทำงานจะมีตัวสำรองที่ทำงานพร้อมกันแบบขนาน ทำให้มีคำตอบได้หลายคำตอบ จึงต้องมีการเลือกคำตอบที่เหมาะสมจากคำตอบที่ได้มาทั้งหมด

แบบรูป Voting เป็นการกำหนดวิธีที่ใช้ในการเลือกผลลัพธ์ โดยที่ผู้ที่รับผิดชอบในการเลือกผลลัพธ์ถูกกำหนดโดยสเตอรีโอไทพ์ ResponseSelection และใช้วิธีเลือกผลลัพธ์ตามที่กำหนดใน class Voting

Applicability

แบบรูป Voting ควรนำไปใช้กับระบบที่มีการประยุกต์ใช้แบบรูป Redundancy และมีคำตอบที่ได้รับมามากกว่า 1 คำตอบ

Structure

โครงสร้างของแบบรูปนี้นำเสนอในเลขที่ 5) จากภาพที่ 4.15

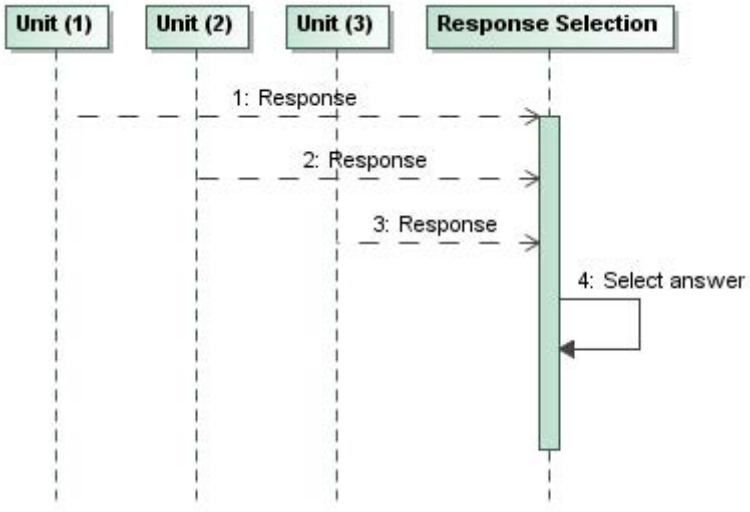
Participants

- Class Voting
  - *VotingAlgorithm* กำหนดวิธีที่ใช้ในการเลือกผลลัพธ์ที่เหมาะสม
- Stereotype ResponseSelection สามารถประยุกต์เข้ากับเมตาโมเดล Classifier
  - *FirstResponse* กำหนดวิธีการเลือกผลลัพธ์ โดยใช้วิธีเลือกผลลัพธ์แรกที่ได้ ซึ่งแสดงว่าต้องการคำตอบโดยเร็วที่สุดจากการทำงาน

- *Voting* กำหนดวิธีในการเลือกผลลัพธ์ โดยมีชนิดเป็นคลาส Voting ซึ่งแสดงว่าต้องการตรวจสอบผลลัพธ์ที่ได้ เพื่อเลือกคำตอบที่ถูกต้องจากการทำงาน

Collaborations

การทำงานของแบบรูป Voting มี Response Selection ทำหน้าที่เป็นตัวเลือกคำตอบที่เหมาะสมที่ได้จาก Unit (1), Unit (2) และ Unit (3) แสดงดังภาพที่ 4.23



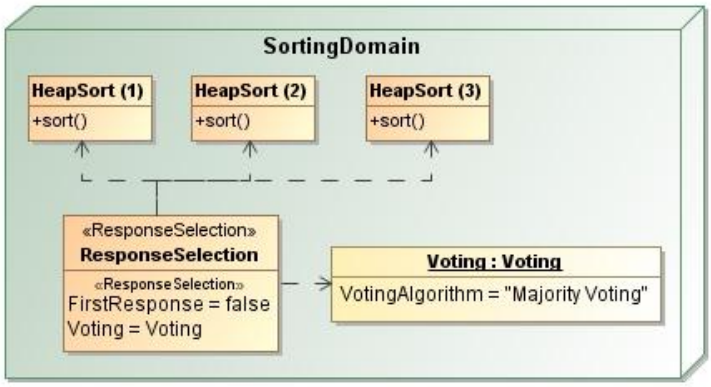
ภาพที่ 4.25 แผนภาพซีเควนซ์การทำงานตามแบบรูป Voting

Consequences

การใช้งานแบบรูป Voting มีข้อดีคือมีหน่วยที่ทำหน้าที่เป็นผู้เลือกคำตอบโดยเฉพาะและมีการกำหนดวิธีการเลือกคำตอบไว้อย่างชัดเจน นอกจากนี้ยังสามารถบ่งบอกถึงความผิดพร่องที่เกิดขึ้นกับตัวสำรองที่ไม่ได้ให้คำตอบหรือให้คำตอบที่แตกต่างจากตัวสำรองอื่นได้

Implementation

การประยุกต์ใช้แบบรูป Voting เข้ากับตัวอย่าง SortingDomain จะมีการกำหนดสเตอรีโอไทพ์ ResponseSelection เข้ากับหน่วยที่ทำหน้าที่ในการเลือกคำตอบจากการทำงานของ HeapSort (1), HeapSort (2) และ HeapSort (3) โดยมีการกำหนดว่าจะไม่เลือกคำตอบที่ตอบมาคำตอบแรกเป็นคำตอบที่เหมาะสม แต่จะทำการเลือกคำตอบตามวิธีการที่กำหนดไว้ในคลาส Voting ดังภาพที่ 4.26 ซึ่งใช้วิธี Majority Voting ซึ่งแปลว่าคำตอบที่เป็นเสียงส่วนใหญ่จะถูกเลือกเป็นคำตอบของการทำงาน



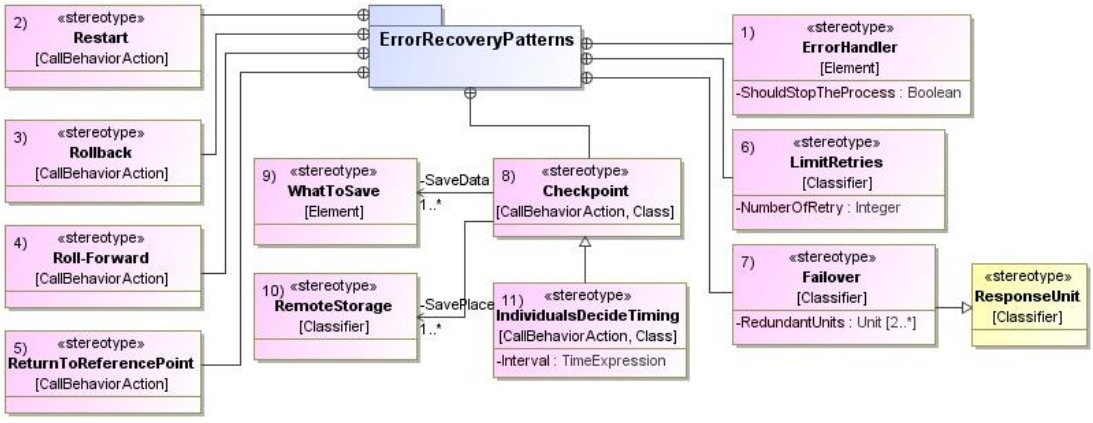
ภาพที่ 4.26 ตัวอย่างการใช้งานแบบรูป Voting

Related Patterns

ไม่มี

4.1.3 ยูเอ็มแอลโปรไฟล์ในกลุ่มของแบบรูปการกู้ระบบจากข้อผิดพลาด

จากการวิเคราะห์แบบรูปการกู้ระบบจากข้อผิดพลาด ซึ่งจำนวนแบบรูปที่สามารถนำมาออกแบบเป็นยูเอ็มแอลโปรไฟล์ คือ 11 แบบรูป โดยแบบรูปทั้งหมดแสดงดังภาพที่ 4.27



ภาพที่ 4.27 ยูเอ็มแอลโปรไฟล์สำหรับแบบรูปในกลุ่มของแบบรูปการกู้ระบบจากข้อผิดพลาด

4.1.3.1 แบบรูป Error Handler

Intent

ออกแบบระบบให้มีส่วนที่ทำการจัดการกับข้อผิดพลาด

Motivation

ระบบจะมีการตรวจหาข้อผิดพลาดในขณะที่มีการทำงาน และข้อผิดพลาดที่เกิดขึ้นจะทำให้ระบบไม่สามารถทำงานต่อไปได้ ถ้าไม่มีการจัดการกับข้อผิดพลาดเหล่านั้น

แบบรูป Error Handler จึงเป็นการกำหนดส่วนที่ทำหน้าที่จัดการกับข้อผิดพลาดโดยแยกออกจากการทำงานหลักของระบบ เพื่อความสะดวกในการบำรุงรักษาหรือแก้ไขระบบในอนาคต

Applicability

แบบรูป Error Handler ควรนำไปใช้กับระบบที่ต้องการให้มีส่วนที่จัดการกับข้อผิดพลาดที่เกิดขึ้นในระบบ

Structure

โครงสร้างของแบบรูปนี้นำเสนอในเลขที่ 1) จากภาพที่ 4.27

Participants

- Stereotype ErrorHandler สามารถประยุกต์เข้ากับเมตาโมเดล Element
    - *ShouldStopTheProcess* กำหนดให้มีการหยุดทำงานหน่วยอื่นหรือไม่เพื่อมาจัดการกับข้อผิดพลาดที่เกิดขึ้น โดยมีจุดประสงค์เพื่อให้ระบบมีช่วงเวลาที่ไม่สามารถทำงานได้สั้นที่สุด

Collaborations

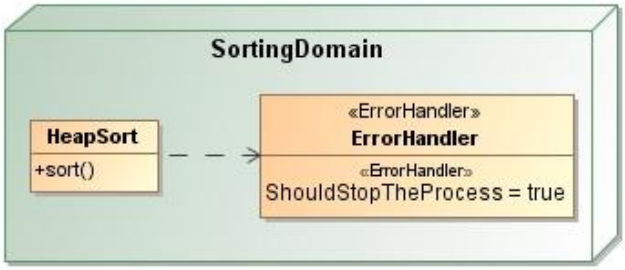
แบบรูป Error Handler ไม่มีการนำเสนอพฤติกรรมการทำงานร่วมกัน เนื่องจากเป็นแบบรูปที่ใช้กำหนดหน่วยที่ทำหน้าที่เป็นหน่วยจัดการกับข้อผิดพลาดในระบบ

Consequences

การใช้งานแบบรูป Error Handler มีข้อดีคือมีการกำหนดหน่วยที่ทำหน้าที่จัดการกับข้อผิดพลาดไว้อย่างชัดเจน แต่อาจส่งผลกระทบต่อสมรรถนะเนื่องจากต้องเพิ่มส่วนจัดการกับข้อผิดพลาดแยกออกมาจากการทำงานหลักของระบบ อาจจะทำให้ระบบมีความซับซ้อนมากขึ้น

Implementation

การประยุกต์ใช้แบบรูป Error Handler เข้ากับตัวอย่าง SortingDomain เพื่อระบุว่าหน่วยที่ประยุกต์ใช้สเตอรีโอไทพ์ ErrorHandler ทำหน้าที่เป็นหน่วยที่คอยจัดการข้อผิดพลาดที่เกิดขึ้น ดังภาพที่ 4.28



ภาพที่ 4.28 ตัวอย่างการใช้งานแบบรูป Error Handler

Related Patterns

ไม่มี

4.1.3.2 แบบรูป Restart

Intent

เมื่อมีข้อผิดพลาดเกิดขึ้นในระบบ ให้เริ่มกลับไปทำงานยังจุดเริ่มต้นของระบบ

Motivation

ถ้าข้อผิดพลาดที่เกิดขึ้นในระบบเป็นข้อผิดพลาดที่เลวร้าย แม้ว่าจะมีความพยายามกู้ระบบจากข้อผิดพลาด แต่การกู้ระบบไม่สามารถทำได้สำเร็จ

แบบรูป Restart จึงมีการกำหนดว่าเมื่อมีข้อผิดพลาดเกิดขึ้น ให้กลับไปทำงานยังจุดเริ่มต้นของระบบ

Applicability

แบบรูป Restart สามารถนำไปใช้กับหน่วยที่ต้องการกลับไปทำงานยังจุดเริ่มต้น เมื่อมีข้อผิดพลาดเกิดขึ้น

Structure

โครงสร้างของแบบรูปนี้นำเสนอในเลขที่ 2) จากภาพที่ 4.27

Participants

- Stereotype Restart สามารถประยุกต์เข้ากับเมตาโมเดล CallBehaviorAction

Collaborations

แบบรูป Restart ไม่มีการนำเสนอพฤติกรรมการทำงานร่วมกัน เนื่องจากเป็นแบบรูปที่ใช้ในกำหนดหน่วยที่ต้องการย้อนกลับไปทำงานยังจุดเริ่มต้น เมื่อมีข้อผิดพลาดเกิดขึ้น

Consequences

การใช้งานแบบรูป Restart มีข้อดีคือมีการกำหนดหน่วยที่ต้องการย้อนกลับไปทำงานใหม่อย่างชัดเจน แต่การย้อนกลับไปทำงานยังจุดเริ่มต้นย่อมทำให้เสียเวลาในการประมวลผลมาก

Implementation

การประยุกต์ใช้แบบรูป Restart เข้ากับเซอร์วิซ Heap Sort เพื่อกำหนดให้เซอร์วิซ Heap Sort กลับไปเริ่มทำงานยังจุดเริ่มต้น ดังภาพที่ 4.29

ภาพที่ 4.29 ตัวอย่างการใช้งานแบบรูป Restart

Related Patterns

Limit Retries, Someone in Charge

4.1.3.3 แบบรูป Rollback

Intent

เมื่อมีข้อผิดพลาดเกิดขึ้นในระบบ ให้กลับไปทำงานยังจุดก่อนที่จะเกิดข้อผิดพลาด

Motivation

เมื่อมีข้อผิดพลาดเกิดขึ้นในระบบและมีการกู้ระบบจากข้อผิดพลาดสำเร็จ แต่คำร้องขอที่เกิดขึ้นระหว่างการกู้ระบบจะไม่ถูกประมวลผล จึงต้องมีการย้อนกลับมาทำงานยังจุดก่อนที่จะเกิดข้อผิดพลาด เนื่องจากการร้องขอที่ทำให้ระบบเกิดข้อผิดพลาดและการร้องขอที่เกิดขึ้นระหว่างการกู้ระบบจากข้อผิดพลาดเป็นการร้องขอที่มีความสำคัญต่อระบบ

แบบรูป Rollback จึงมีการกำหนดว่าเมื่อมีข้อผิดพลาดเกิดขึ้น ให้กลับไปทำงานยังจุดก่อนที่จะเกิดข้อผิดพลาดหรือจุดที่บันทึกไว้

Applicability

แบบรูป Rollback สามารถนำไปใช้กับหน่วยที่ต้องการกลับไปทำงานยังจุดก่อนหน้า เมื่อมีข้อผิดพลาดเกิดขึ้น

Structure

โครงสร้างของแบบรูปนี้นำเสนอในเลขที่ 3) จากภาพที่ 4.27

Participants

- Stereotype Rollback สามารถประยุกต์เข้ากับเมตาโมเดล CallBehaviorAction

Collaborations

แบบรูป Rollback ไม่มีการนำเสนอพฤติกรรมการทำงานร่วมกัน เนื่องจากเป็นแบบรูปที่ใช้ในการกำหนดหน่วยที่ต้องการย้อนกลับไปทำงานยังจุดก่อนหน้าที่ข้อผิดพลาดเกิดขึ้น

Consequences

การใช้งานแบบรูป Rollback มีข้อดีคือมีการกำหนดจุดที่ต้องการย้อนกลับไปทำงานใหม่อย่างชัดเจนและไม่จำเป็นต้องย้อนกลับไปทำงานยังจุดเริ่มต้น แต่การย้อนกลับไปทำงานยังจุดก่อนหน้าย่อมทำให้เสียเวลาในการประมวลผลเพิ่มขึ้น

Implementation

การประยุกต์ใช้แบบรูป Rollback เข้ากับเซอร์วิซ Heap Sort เพื่อกำหนดให้เซอร์วิซ Heap Sort กลับไปทำงานยังจุดก่อนที่จะมีข้อผิดพลาดเกิดขึ้น ดังภาพที่ 4.30

ภาพที่ 4.30 ตัวอย่างการใช้งานแบบรูป Rollback

Related Patterns

Limit Retries, Someone in Charge

4.1.3.4 แบบรูป Roll-Forward

Intent

เมื่อมีข้อผิดพลาดเกิดขึ้นในระบบ ให้ข้ามส่วนที่มีข้อผิดพลาดไปทำงานยังส่วนถัดไปที่ไม่มีข้อผิดพลาด

Motivation

เมื่อมีข้อผิดพลาดเกิดขึ้นในระบบและมีการกู้ระบบจากข้อผิดพลาดสำเร็จ หลังจากนั้นสามารถไปทำงานยังจุดถัดไปได้เลย โดยทำการเพิกเฉยกับการร้องขอที่ทำให้เกิดข้อผิดพลาด เนื่องจากการร้องขอนั้นเป็นการร้องขอที่ไม่มีความสำคัญต่อระบบ และเพื่อทำให้ข้อผิดพลาดนั้นไม่เกิดขึ้นซ้ำในระบบ

แบบรูป Roll-Forward จึงมีการกำหนดว่าเมื่อมีข้อผิดพลาดเกิดขึ้น ให้ละเว้นการร้องขอที่ทำให้เกิดข้อผิดพลาดและข้ามไปทำงานยังจุดถัดไปในระบบ

Applicability

แบบรูป Roll-Forward สามารถนำไปใช้กับหน่วยที่ต้องการข้ามไปทำงานยังจุดถัดไปหลังจากมีข้อผิดพลาดเกิดขึ้น

Structure

โครงสร้างของแบบรูปนี้นำเสนอในเลขที่ 4) จากภาพที่ 4.27

Participants

- Stereotype Roll-Forward สามารถประยุกต์เข้ากับเมตาโมเดล CallBehaviorAction

Collaborations

แบบรูป Roll-Forward ไม่มีการนำเสนอพฤติกรรมการทำงานร่วมกัน เนื่องจากเป็นแบบรูปที่ใช้ในการกำหนดหน่วยที่ต้องการข้ามไปทำงานยังจุดถัดไปหลังจากมีข้อผิดพลาดเกิดขึ้น

Consequences

การใช้งานแบบรูป Roll-Forward มีข้อดีคือมีการกำหนดจุดที่ต้องการข้ามไปทำงานอย่างชัดเจนและสามารถเพิกเฉยการร้องขอที่ทำให้เกิดข้อผิดพลาดและการร้องขอในช่วงเวลาที่ทำการกู้ระบบ แต่การข้ามไปทำงานยังจุดถัดไปเลยอาจจะทำให้พลาดต่อการร้องขอที่มีความสำคัญที่เกิดขึ้นในระบบ

Implementation

การประยุกต์ใช้แบบรูป Roll-Forward เข้ากับเซอร์วิซ Heap Sort เพื่อกำหนดให้เซอร์วิซ Heap Sort ข้ามไปทำงานยังจุดถัดไปเลย โดยไม่ต้องสนใจกับการร้องขอที่ทำให้เกิดข้อผิดพลาด ดังภาพที่ 4.31

ภาพที่ 4.31 ตัวอย่างการใช้งานแบบรูป Roll-Forward

Related Patterns

Limit Retries, Someone in Charge

4.1.3.5 แบบรูป Return to Reference Point

Intent

เมื่อมีข้อผิดพลาดเกิดขึ้นในระบบ ให้กลับไปทำงานยังจุดที่ได้มีการบันทึกข้อมูลไว้ และต้องเป็นจุดที่แน่ใจว่าไม่มีข้อผิดพลาด

Motivation

เมื่อมีข้อผิดพลาดเกิดขึ้นในระบบ และมีการกู้ระบบจากข้อผิดพลาด แต่อาจเกิดข้อผิดพลาดเกิดขึ้นอีกในช่วงที่กู้ระบบ ซึ่งไม่ใช่การทำงานหลักของระบบ

แบบรูป Return to Reference Point จึงมีการกำหนดว่าเมื่อมีข้อผิดพลาดเกิดขึ้นในส่วนที่ไม่ใช่การทำงานหลักของระบบ ให้กลับไปทำงานยังจุดอ้างอิงที่มีการบันทึกไว้ในส่วนการทำงานหลัก

Applicability

แบบรูป Return to Reference Point สามารถนำไปใช้กับหน่วยที่ต้องการกลับไปทำงานต่อยังจุดอ้างอิงที่มีการบันทึกไว้

Structure

โครงสร้างของแบบรูปนี้นำเสนอในเลขที่ 5) จากภาพที่ 4.27

Participants

- Stereotype ReturnToReferencePoint สามารถประยุกต์เข้ากับเมตาโมเดล CallBehaviorAction

Collaborations

แบบรูป Return to Reference Point ไม่มีการนำเสนอพฤติกรรมการทำงานร่วมกัน เนื่องจากเป็นแบบรูปที่ใช้ในการกำหนดหน่วยที่ต้องการกลับไปทำงานยังจุดอ้างอิงที่บันทึกไว้หลังจากมีข้อผิดพลาดเกิดขึ้น

Consequences

การใช้งานแบบรูป Return to Reference Point มีข้อดีคือมีการกำหนดจุดที่ต้องการทำงานต่ออย่างชัดเจน สำหรับข้อผิดพลาดที่เกิดขึ้นระหว่างการทำงานที่ไม่ใช่งานหลักของระบบ แต่การกลับไปทำงานยังจุดอ้างอิงที่บันทึกไว้อาจจะทำให้เสียเวลาในการทำงานซ้ำบางส่วนที่ได้ทำไปแล้ว

Implementation

การประยุกต์ใช้แบบรูป Return to Reference Point เข้ากับเซอร์วิซ Heap Sort เพื่อกำหนดให้เซอร์วิซ Heap Sort กลับไปทำงานยังจุดที่มีการบันทึกข้อมูลไว้ ดังภาพที่ 4.32

ภาพที่ 4.32 ตัวอย่างการใช้งานแบบรูป Return to Reference Point

Related Patterns

Limit Retries, Someone in Charge

4.1.3.6 แบบรูป Limit Retries

Intent

การกลับไปทำงาน ณ จุดเดิมที่มีข้อผิดพลาดซ้ำๆ โดยไม่เปลี่ยนแปลงใดๆเลย ข้อผิดพลาดนั้นย่อมมีโอกาสเกิดขึ้นซ้ำอีก จึงควรกำหนดจำนวนครั้งของการทำซ้ำ

Motivation

ระบบจะทำงานและได้ผลลัพธ์ที่เหมือนกันเมื่อมีการทำงานในกิจกรรม ข้อมูลและสถานะเดียวกัน ถ้าการทำงานเหล่านั้นมีข้อผิดพลาดเกิดขึ้นย่อมทำให้ข้อผิดพลาดเกิดขึ้นอีก

แบบรูป Limit Retries จึงมีการจำกัดจำนวนของการทำซ้ำให้เหมาะสม เพื่อไม่ให้ระบบติดอยู่กับการจัดการข้อผิดพลาดนั้นซ้ำๆ

Applicability

แบบรูป Limit Retries สามารถนำไปใช้กับหน่วยที่ต้องการจำกัดจำนวนครั้งของการทำซ้ำเมื่อหน่วยนั้นมีข้อผิดพลาดเกิดขึ้น

Structure

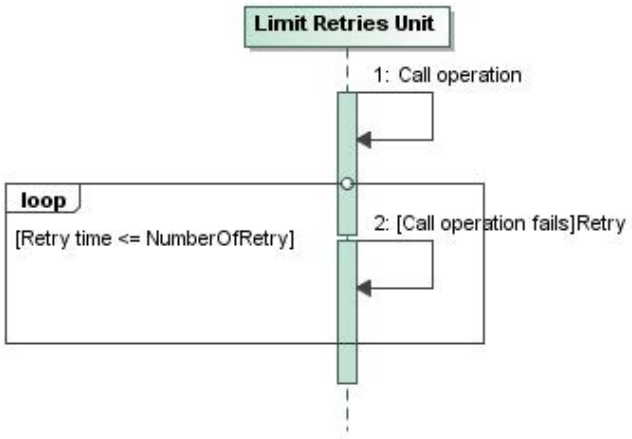
โครงสร้างของแบบรูปนี้นำเสนอในเลขที่ 6) จากภาพที่ 4.27

Participants

- Stereotype LimitRetries สามารถประยุกต์เข้ากับเมตาโมเดล Classifier
  - *NumberOfRetry* กำหนดจำนวนครั้งของการทำซ้ำของหน่วยย่อย เมื่อมีข้อผิดพลาดเกิดขึ้น

Collaborations

สำหรับการทำงานของแบบรูป Limit Retries เมื่อมีข้อผิดพลาดเกิดขึ้นในหน่วย จะมีการทำซ้ำตามจำนวนที่กำหนดในคุณสมบัติ NumberOfRetry แสดงดังภาพที่ 4.33



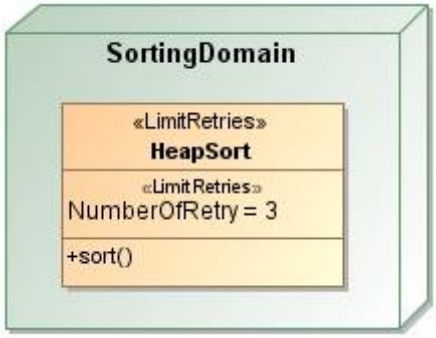
ภาพที่ 4.33 แผนภาพซีเควนซ์การทำงานตามแบบรูป Limit Retries

Consequences

การใช้งานแบบรูป Limit Retries มีข้อดีคือมีการกำหนดจำนวนการทำซ้ำที่ชัดเจน เพื่อที่ระบบจะไม่ติดอยู่กับข้อผิดพลาดที่เกิดขึ้น แต่การกำหนดจำนวนครั้งต้องกำหนดอย่างถูกต้อง มิฉะนั้นอาจจะมีการทำซ้ำที่มากเกินไป

Implementation

การประยุกต์ใช้แบบรูป LimitRetries เข้ากับตัวอย่าง SortingDomain โดยมีการประยุกต์ใช้สเตอริโอไทพ์ LimitRetries เข้ากับ HeapSort และระบุจำนวนครั้งในการทำงานซ้ำเมื่อมีข้อผิดพลาดเกิดขึ้น ดังภาพที่ 4.43



ภาพที่ 4.34 ตัวอย่างการใช้งานแบบรูป Limit Retries

Related Patterns

Someone in Charge

4.1.3.7 แบบรูป Failover

Intent

เมื่อมีความผิดพร่องเกิดขึ้น จะทำการกู้ระบบโดยเปลี่ยนไปทำงานยังตัวสำรองแทน

Motivation

ระบบที่ออกแบบตามแบบรูป Redundancy ระบบจะต้องมีการตรวจจับข้อผิดพลาดจากหน่วยสำรองหนึ่ง และต้องมีหน่วยที่ทำหน้าที่จัดการย้ายงานไปทำงานยังหน่วยสำรองอื่น

แบบรูป Failover จึงมีการกำหนดตัวจัดการในการถ่ายโอนงานไปยังหน่วยอื่น

Applicability

แบบรูป Failover สามารถนำไปใช้กับหน่วยที่ทำหน้าที่เป็นหน่วยจัดการการถ่ายโอนงานไปยังหน่วยอื่น

Structure

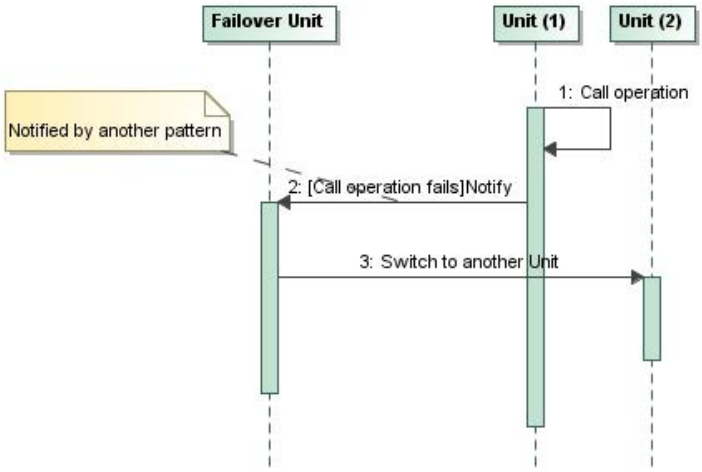
โครงสร้างของแบบรูปนี้นำเสนอในเลขที่ 7) จากภาพที่ 4.27

Participants

- Stereotype Failover สามารถประยุกต์เข้ากับเมตาโมเดล Classifier
  - *RedundantUnits* กำหนดรายการหน่วยที่เป็นตัวสำรองซึ่งกันและกัน โดยที่ต้องมีการกำหนดหน่วยสำรองอย่างน้อย 2 หน่วยขึ้นไป โดยที่ต้องมีชนิดเป็น Unit

Collaborations

การทำงานของแบบรูป Failover จะทำหน้าที่สลับการทำงานไปยังอีกหน่วยหนึ่งที่กำหนดไว้ใน RedundantUnits เมื่อทราบว่าหน่วยนั้นมีข้อผิดพลาดเกิดขึ้น โดยที่การแจ้งว่าหน่วยใดมีความผิดพร่องเกิดขึ้นเป็นหน้าที่ของแบบรูปอื่นๆ เช่น แบบรูป Fault Observer แสดงดังภาพที่ 4.35



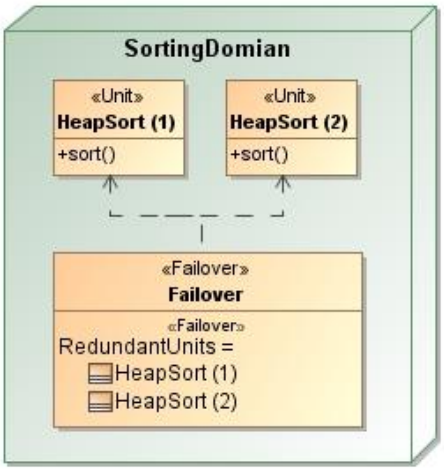
ภาพที่ 4.35 แผนภาพซีเควนซ์การทำงานตามแบบรูป Failover

Consequences

การใช้งานแบบรูป Failover มีหน่วยที่ทำหน้าที่เป็นตัวสลับงานไปยังหน่วยอื่นอย่างชัดเจน แต่การแจ้งให้หน่วยที่ทำหน้าที่สลับงานต้องอาศัยแบบรูปอื่นเป็นตัวช่วย ดังนั้นการประยุกต์ใช้แบบรูปที่มากเกินไปอาจจะทำให้ระบบมีความซับซ้อน

Implementation

การประยุกต์ใช้แบบรูป Failover เข้ากับตัวอย่าง SortingDomain มีการประยุกต์ใช้สเตอรีโอไทพ์ Failover เข้ากับหน่วยที่ทำหน้าที่จัดการถ่ายโอนงานระหว่าง HeapSort (1) และ HeapSort (2) ดังภาพที่ 4.36



ภาพที่ 4.36 ตัวอย่างการใช้งานแบบรูป Failover

Related Patterns

Someone in Charge, Voting

4.1.3.8 แบบรูป Checkpoint

Intent

บันทึกข้อมูลของระบบไว้เป็นระยะๆ เพื่อนำไปใช้เมื่อต้องทำการกู้ระบบจากข้อผิดพลาด

Motivation

เมื่อต้องการสร้างระบบให้มีการทนต่อความผิดพร่อง และมีสภาพพร้อมใช้งานในระดับสูง ระบบจำเป็นจะต้องมีการแก้ไขข้อผิดพลาดที่เกิดขึ้น

แบบรูป Checkpoint จึงมีการกำหนดให้มีการจัดเก็บสถานะของระบบ เพื่อนำมาใช้ในการจัดการกับข้อผิดพลาด โดยที่ไม่ต้องกลับไปทำงานยังจุดเริ่มต้นของระบบ

Applicability

แบบรูป Checkpoint สามารถนำไปใช้ในหน่วยที่มีการจัดเก็บข้อมูลในระบบ

Structure

โครงสร้างของแบบรูปนี้นำเสนอในเลขที่ 8) จากภาพที่ 4.27

Participants

- Stereotype Checkpoint สามารถประยุกต์เข้ากับเมตาโมเดล CallBehaviorAction, Class
    - *SaveData* กำหนดข้อมูลที่ต้องทำการบันทึก โดยข้อมูลที่ต้องการทำการบันทึกต้องเป็นไปตามแบบรูป WhatToSave (มีการประยุกต์ใช้สเตอรีโอไทพ์ WhatToSave) โดยต้องมีข้อมูลที่ทำการบันทึกอย่างน้อย 1 ข้อมูล
    - *SavePlace* กำหนดสถานที่สำหรับทำการบันทึก โดยสถานที่ที่จะทำการบันทึกต้องเป็นไปตามแบบรูป RemoteStorage (มีการประยุกต์ใช้สเตอรีโอไทพ์ RemoteStorage) โดยต้องมีการกำหนดสถานที่ที่ต้องทำการบันทึกข้อมูลอย่างน้อย 1 สถานที่

Collaborations

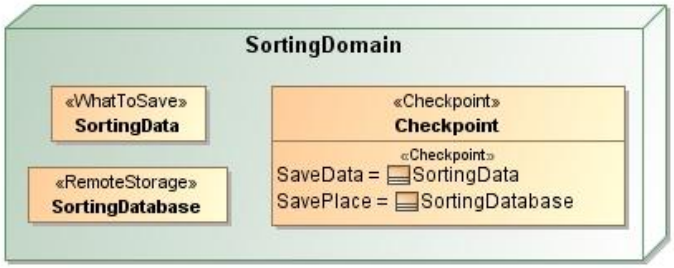
แบบรูป Checkpoint ไม่มีการนำเสนอพฤติกรรมการทำงานร่วมกัน เนื่องจากเป็นแบบรูปที่ใช้ในการกำหนดหน่วยที่ทำหน้าที่บันทึกข้อมูลของระบบ เพื่อนำมาใช้ในการกู้ระบบจากข้อผิดพลาด

Consequences

การใช้งานแบบรูป Checkpoint มีหน่วยที่ทำหน้าที่บันทึก เพื่อระบบจะได้กลับมาทำงานยังจุดที่ได้บันทึกข้อมูลไว้ ไม่ต้องกลับไปเริ่มต้นทำงานใหม่ทั้งหมด แต่การบันทึกข้อมูลจำเป็นจะต้องกำหนดข้อมูลและสถานที่บันทึกข้อมูลให้เหมาะสม มิเช่นนั้นจะทำให้ระบบมีความซับซ้อนมากเกินไป และการใช้งานข้อมูลในขณะที่กู้ระบบจากข้อผิดพลาดทำงานได้ยาก

Implementation

การประยุกต์ใช้แบบรูป Checkpoint เข้ากับตัวอย่าง SortingDomain มีการประยุกต์ใช้สเตอรีโอไทพ์ Checkpoint เข้ากับหน่วยที่จัดการเกี่ยวกับการบันทึกข้อมูล โดยมีการระบุข้อมูลที่จะบันทึกและสถานที่ที่ทำการบันทึก ดังภาพที่ 4.37



ภาพที่ 4.37 ตัวอย่างการใช้งานแบบรูป Checkpoint

Related Patterns

What to Save

4.1.3.9 แบบรูป What to Save

Intent

มีการกำหนดว่าข้อมูลที่จะต้องทำการบันทึก เป็นข้อมูลใดบ้าง

Motivation

การออกแบบระบบตามแบบรูป Checkpoint จำเป็นจะต้องมีการส่งข้อมูลให้กับระบบในกรณีที่มีข้อผิดพลาดเกิดขึ้น เพื่อนำไปใช้ในการกู้ระบบ

แบบรูป What to Save จึงมีการกำหนดข้อมูลที่จะทำการบันทึก ตามแบบรูป Checkpoint

Applicability

แบบรูป What to Save สามารถนำไปใช้ในหน่วยที่เป็นข้อมูลที่ต้องการบันทึกในระบบ

Structure

โครงสร้างของแบบรูปนี้นำเสนอในเลขที่ 9) จากภาพที่ 4.27

Participants

- Stereotype WhatToSave สามารถประยุกต์เข้ากับเมตาโมเดล Element

Collaborations

แบบรูป What to Save ไม่มีการนำเสนอพฤติกรรมการทำงานร่วมกัน เนื่องจากเป็นแบบรูปที่ใช้ในการกำหนดหน่วยที่เป็นข้อมูลที่ต้องการบันทึกตามแบบรูป Checkpoint เพื่อนำมาใช้ในการกู้ระบบจากข้อผิดพลาด

Consequences

การใช้งานแบบรูป What to Save มีข้อดีคือมีการกำหนดข้อมูลที่ทำการบันทึกอย่างชัดเจน เพื่อระบบจะได้นำข้อมูลที่บันทึกไปใช้ในการกู้ระบบจากข้อผิดพลาดอย่างถูกต้อง

Implementation

การประยุกต์ใช้แบบรูป What to Save เข้ากับ SortingDomain มีการประยุกต์ใช้สเตอรีโอไทพ์ WhatToSave เข้ากับข้อมูลที่ต้องการบันทึก ดังภาพที่ 4.38

ภาพที่ 4.38 ตัวอย่างการใช้งานแบบรูป What to Save

Related Patterns

Remote Storage

4.1.3.10 แบบรูป Remote Storage

Intent

พิจารณาตัวสำรองในระบบและปัจจัยอื่นๆ เพื่อพิจารณาว่าจะทำการบันทึกข้อมูลไว้ที่ใด

Motivation

การออกแบบระบบให้มีการบันทึกข้อมูลตามแบบรูป What to Save จำเป็นต้องทำการบันทึกข้อมูลในที่ที่ปลอดภัย และมีความเชื่อถือได้มากที่สุด เนื่องจากเป้าหมายสำคัญของแบบรูป Checkpoint คือให้ระบบมีสภาพพร้อมใช้งานและสามารถกู้ระบบจากข้อผิดพลาดได้อย่างรวดเร็ว

แบบรูป Remote Storage จึงมีการกำหนดสถานที่ที่ทำการบันทึกข้อมูล ตามแบบรูป Checkpoint

Applicability

แบบรูป Remote Storage สามารถนำไปใช้ในหน่วยที่เป็นที่บันทึกข้อมูลตามแบบรูป Checkpoint

Structure

โครงสร้างของแบบรูปนี้นำเสนอในเลขที่ 10) จากภาพที่ 4.27

Participants

- Stereotype RemoteStorage สามารถประยุกต์เข้ากับเมตาโมเดล Classifier

Collaborations

แบบรูป Remote Storage ไม่มีการนำเสนอพฤติกรรมการทำงานร่วมกัน เนื่องจากเป็นแบบรูปที่ใช้ในการกำหนดสถานที่ในการบันทึกข้อมูลตามแบบรูป Checkpoint เพื่อนำมาใช้ในการกู้ระบบจากข้อผิดพลาด

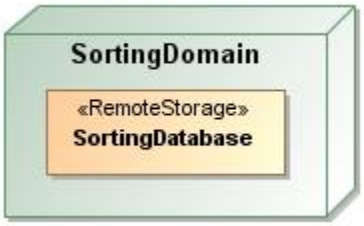
Consequences

การใช้งานแบบรูป Remote Storage มีข้อดีคือมีการกำหนดสถานที่ในการบันทึกข้อมูลอย่างชัดเจน เพื่อระบบจะได้นำข้อมูลที่บันทึกไปใช้ในการกู้ระบบจากข้อผิดพลาดอย่างถูกต้อง แต่ถ้า

บันทึกข้อมูลไว้ในที่ไม่เหมาะสม เช่น บันทึกไว้ในส่วนเดียวกับส่วนที่มีข้อผิดพลาดเกิดขึ้น จึงอาจจะทำให้ไม่สามารถกู้ระบบจากข้อผิดพลาด

Implementation

การประยุกต์ใช้แบบรูป Remote Storage เข้ากับตัวอย่าง SortingDomain โดยมีการประยุกต์ใช้สเตอริโอไทพ์ RemoteStorage เข้ากับสถานที่ที่ทำการบันทึกข้อมูล ดังภาพที่ 4.39



ภาพที่ 4.39 ตัวอย่างการใช้งานแบบรูป Remote Storage

Related Patterns

ไม่มี

4.1.3.11 แบบรูป Individuals Decide Timing

Intent

ออกแบบให้แต่ละกระบวนการทำการกำหนดว่าเมื่อใดที่จะต้องทำการบันทึกข้อมูล

Motivation

การออกแบบระบบตามแบบรูป Checkpoint จะมีการบันทึกข้อมูลบางอย่าง เช่น ข้อมูลเกี่ยวกับโครงสร้าง ข้อมูลเกี่ยวกับระบบ ซึ่งข้อมูลเหล่านี้จะเปลี่ยนแปลงไปตามสถานะในระบบ จึงต้องมีการบันทึกเป็นระยะ

แบบรูป Individuals Decide Timing จึงมีการกำหนดเวลาที่เหมาะสมในการทำการบันทึกข้อมูล ซึ่งออกแบบตามแบบรูป Checkpoint

Applicability

แบบรูป Individuals Decide Timing สามารถนำไปใช้กับการทำงานตามแบบรูป Checkpoint ที่สามารถกำหนดเวลาในการบันทึกข้อมูล

Structure

โครงสร้างของแบบรูปนี้นำเสนอในเลขที่ 11) จากภาพที่ 4.27

Participants

- Stereotype IndividualsDecideTiming สามารถประยุกต์เข้ากับเมตาโมเดล CallBehaviorAction, Class
    - *SaveData* กำหนดข้อมูลที่ต้องทำการบันทึก โดยข้อมูลที่ต้องทำการบันทึกต้องเป็นไปตามแบบรูป What to Save (มีการประยุกต์ใช้สเตอรีโอไทพ์ WhatToSave) โดยต้องมีข้อมูลที่ทำการบันทึกอย่างน้อย 1 ข้อมูล ตามแบบรูป Checkpoint
    - *SavePlace* กำหนดสถานที่สำหรับทำการบันทึก โดยสถานที่ที่จะทำการบันทึกต้องเป็นไปตามแบบรูป Remote Storage (มีการประยุกต์ใช้สเตอรีโอไทพ์ RemoteStorage) โดยต้องมีการกำหนดสถานที่ที่ต้องทำการบันทึกข้อมูลอย่างน้อย 1 สถานที่ ตามแบบรูป Checkpoint
    - *Interval* กำหนดช่วงเวลาที่เหมาะสมสำหรับการบันทึกข้อมูล

Collaborations

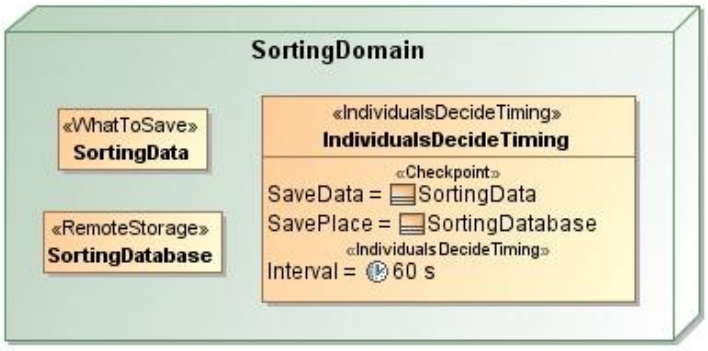
แบบรูป Individuals Decide Timing ไม่มีการนำเสนอพฤติกรรมการทำงานร่วมกัน เนื่องจากเป็นแบบรูปที่ใช้ในกำหนดเวลาในการบันทึกข้อมูล เพื่อนำมาใช้ในการกู้ระบบจากข้อผิดพลาด

Consequences

การใช้งานแบบรูป Individuals Decide Timing มีข้อดีคือมีการกำหนดเวลาที่ชัดเจนในการบันทึกข้อมูลตามแบบรูป Checkpoint เพื่อระบบจะได้นำข้อมูลที่บันทึกไปใช้ในการกู้ระบบจากข้อผิดพลาดอย่างถูกต้อง แต่ถ้าบันทึกข้อมูลในเวลาที่ไม่เหมาะสมจะไม่ได้ข้อมูลที่เป็นประโยชน์สำหรับนำไปใช้ในการกู้ระบบได้

Implementation

การประยุกต์ใช้แบบรูป Individuals Decide Timing เข้ากับตัวอย่าง SortingDomain โดยการประยุกต์ใช้สเตอรีโอไทพ์ IndividualsDecideTiming เข้ากับหน่วยที่จัดการเกี่ยวกับการบันทึกข้อมูลที่มีการทำงานตามรอบเวลาที่กำหนดไว้ และมีการกำหนดข้อมูลที่ทำการบันทึกและสถานที่ที่บันทึกข้อมูล ดังภาพที่ 4.40



ภาพที่ 4.40 ตัวอย่างการใช้งานแบบรูป Individuals Decide Timing

Related Patterns

ไม่มี

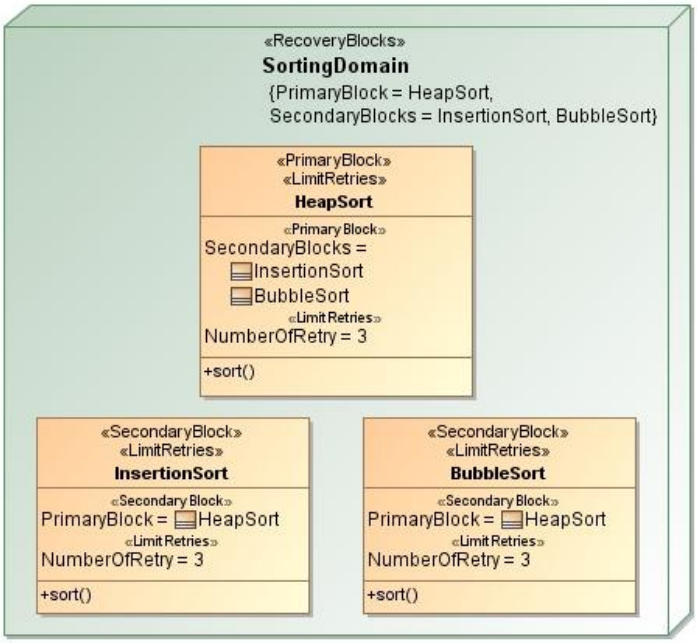
4.2 การประยุกต์ใช้งานยูเอ็มแอลโปรไฟล์สำหรับแบบรูปการทนต่อความผิดพร่อง

หัวข้อนี้จะทำการจัดกลุ่มแบบรูปที่มีความสัมพันธ์ในการใช้งานร่วมกันได้ โดยนำเสนอเป็นตัวอย่างเพื่อเป็นแนวทางในการประยุกต์ใช้งานในระดับการออกแบบของระบบอิงบริการต่อไป ตัวอย่างการประยุกต์ใช้งานร่วมกันของแบบรูปในยูเอ็มแอลโปรไฟล์ มีดังต่อไปนี้

4.2.1 การประยุกต์ใช้งานยูเอ็มแอลโปรไฟล์สำหรับแบบรูป Recovery Blocks และ แบบรูป Limit Retries

การประยุกต์ใช้แบบรูป Recovery Blocks เข้ากับระบบอิงบริการ โดยทำการประยุกต์ใช้สเตอรีโอไทพ์ Recovery Blocks เข้ากับระบบ SortingDomain และทำการประยุกต์ใช้สเตอรีโอไทพ์ PrimaryUnit เข้ากับ HeapSort เพื่อกำหนดหน่วยสำรองที่เริ่มทำงานเป็นหน่วยแรก และสเตอรีโอไทพ์ SecondaryUnit เข้ากับ InsertionSort และ BubbleSort ตามลำดับ เพื่อกำหนดรายการของ

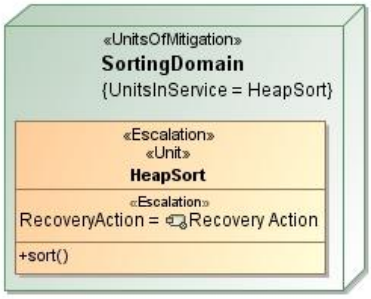
หน่วยสำรองที่จะทำงานเมื่อ PrimaryUnit มีข้อผิดพลาด และทำการประยุกต์ใช้แบบรูป Limit Retries เข้ากับ HeapSort, InsertionSort และ BubbleSort ซึ่งเป็นหน่วยสำรองของ Recovery Blocks พร้อมทั้งกำหนดจำนวนของการทำงานซ้ำของหน่วยย่อย ดังภาพที่ 4.41 ซึ่งหมายถึงเมื่อเกิดข้อผิดพลาดจะมีการทำซ้ำภายในหน่วยย่อยก่อนตามจำนวนครั้งที่กำหนด ก่อนจะเปลี่ยนไปทำงานด้วยหน่วยย่อยสำรองอื่นหากยังไม่สามารถจัดการข้อผิดพลาดได้



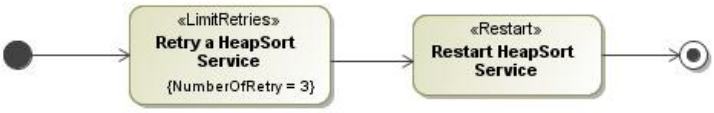
ภาพที่ 4.41 การประยุกต์ใช้งานยูเอ็มแอลโปรไฟล์สำหรับแบบรูป Recovery Blocks และ แบบรูป Limit Retries

4.2.2 การประยุกต์ใช้งานยูเอ็มแอลโปรไฟล์สำหรับแบบรูป Units of Mitigation แบบรูป Escalation แบบรูป Limit Retries และ แบบรูป Restart

การประยุกต์ใช้แบบรูป Units of Mitigation ทำการระบุหน่วยย่อยในระบบคือ HeapSort ซึ่งหน่วยย่อยนั้นมีการประยุกต์ใช้แบบรูป Escalation เพื่อกำหนดขั้นตอนการจัดการกับข้อผิดพลาดตามแผนภาพแอคทิวิตี Recovery Action ในขั้นตอนการจัดการกับข้อผิดพลาดในขั้นแรกจะเป็นการทำงานซ้ำที่เซอร์วิซตามแบบรูป Limit Retries และหากไม่สำเร็จ การจัดการกับข้อผิดพลาดในขั้นถัดไปซึ่งเข้มขึ้นคือการเริ่มทำงานเซอร์วิซใหม่ตามแบบรูป Restart ดังภาพที่ 4.42 และ ภาพที่ 4.43



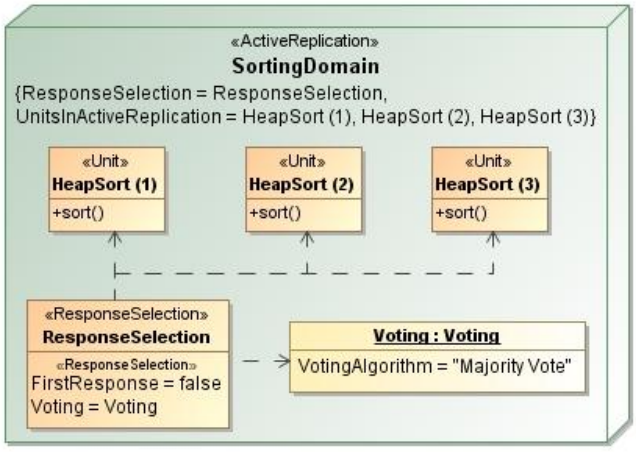
ภาพที่ 4.42 การประยุกต์ใช้งานยูเอ็มแอลโปรไฟล์สำหรับแบบรูป Units of Mitigation แบบรูป Escalation แบบรูป Limit Retries และ แบบรูป Restart



ภาพที่ 4.43 แผนภาพแอคทิวิตีสำหรับ Recovery Action

4.2.3 การประยุกต์ใช้งานยูเอ็มแอลโปรไฟล์สำหรับแบบรูป Active Replication และแบบรูป Voting

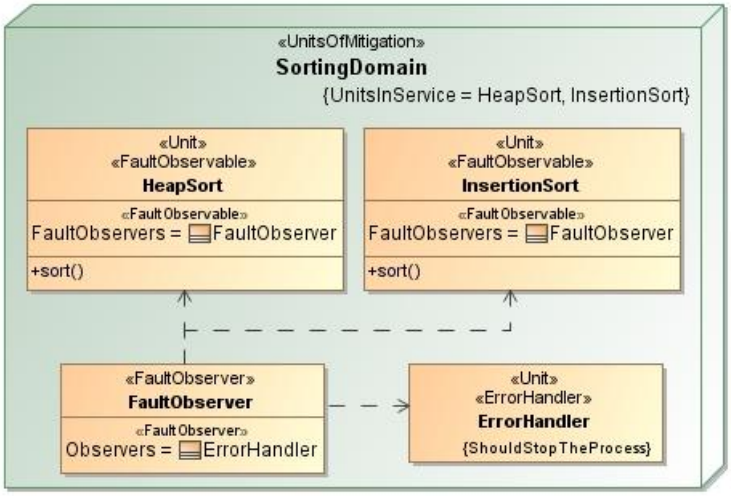
การประยุกต์ใช้แบบรูป Active Replication มีการประยุกต์ใช้สเตอรีโอไทพ์ ActiveReplication เข้ากับตัวอย่าง SortingDomain ซึ่งมีการกำหนดหน่วยที่ทำงานพร้อมกันและกำหนดหน่วยที่ทำหน้าที่เลือกผลลัพธ์ที่เหมาะสม โดยประยุกต์ใช้สเตอรีโอไทพ์ ResponseSelection พร้อมกำหนดว่าจะเลือกคำตอบแรกเป็นตอบที่เหมาะสมหรือไม่ ซึ่งในตัวอย่างนี้ไม่ได้ใช้คำตอบแรกเป็นคำตอบที่เหมาะสม แต่จะกำหนดวิธีการในการเลือกคำตอบที่เหมาะสม เป็นชนิดของ Voting ดังภาพที่ 4.44



ภาพที่ 4.44 การประยุกต์ใช้งานยูเอ็มแอลโปรไฟล์สำหรับแบบรูป Active Replication และ แบบรูป Voting

4.2.4 การประยุกต์ใช้งานยูเอ็มแอลโปรไฟล์สำหรับแบบรูป Units of Mitigation แบบรูป Fault Observer และ แบบรูป Error Handler

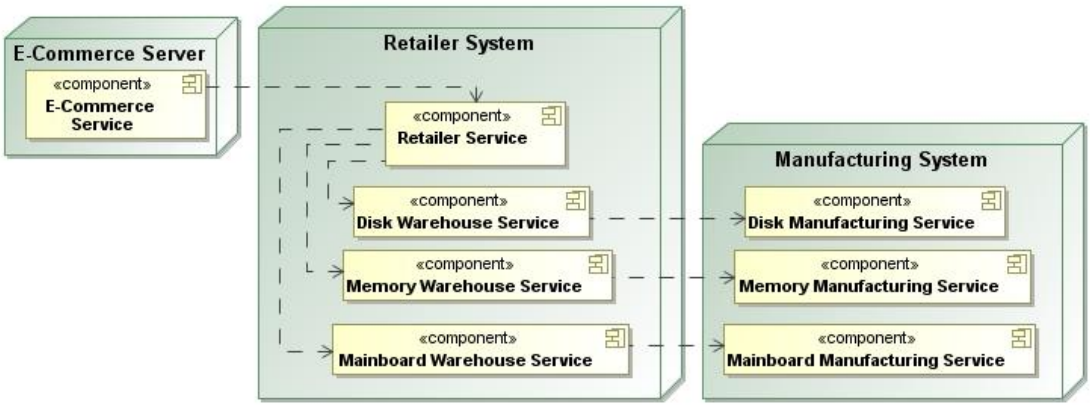
การประยุกต์ใช้แบบรูป Units of Mitigation มีการประยุกต์ใช้สเตอรีโอไทพ์เข้ากับ SortingDomain และกำหนดหน่วยย่อยในระบบ สำหรับการใช้แบบรูป Fault Observer มีการประยุกต์ใช้สเตอรีโอไทพ์ FaultObserver เพื่อทำหน้าที่เป็นผู้สังเกต หน่วยที่มีการประยุกต์ใช้สเตอรีโอไทพ์ FaultObservable คือ HeapSort และ InsertionSort เมื่อมีข้อผิดพลาดเกิดขึ้น ทั้งสองหน่วยนั้นจะทำการรายงานไปยัง FaultObserver ซึ่งจะรายงานไปยังหน่วยที่สนใจคือ ErrorHandler เพื่อจัดการกับข้อผิดพลาดนั้น ดังภาพที่ 4.45



ภาพที่ 4.45 การประยุกต์ใช้งานยูเอ็มแอลโปรไฟล์สำหรับแบบรูป Units of Mitigation แบบรูป Fault Observer และ แบบรูป Error Handler

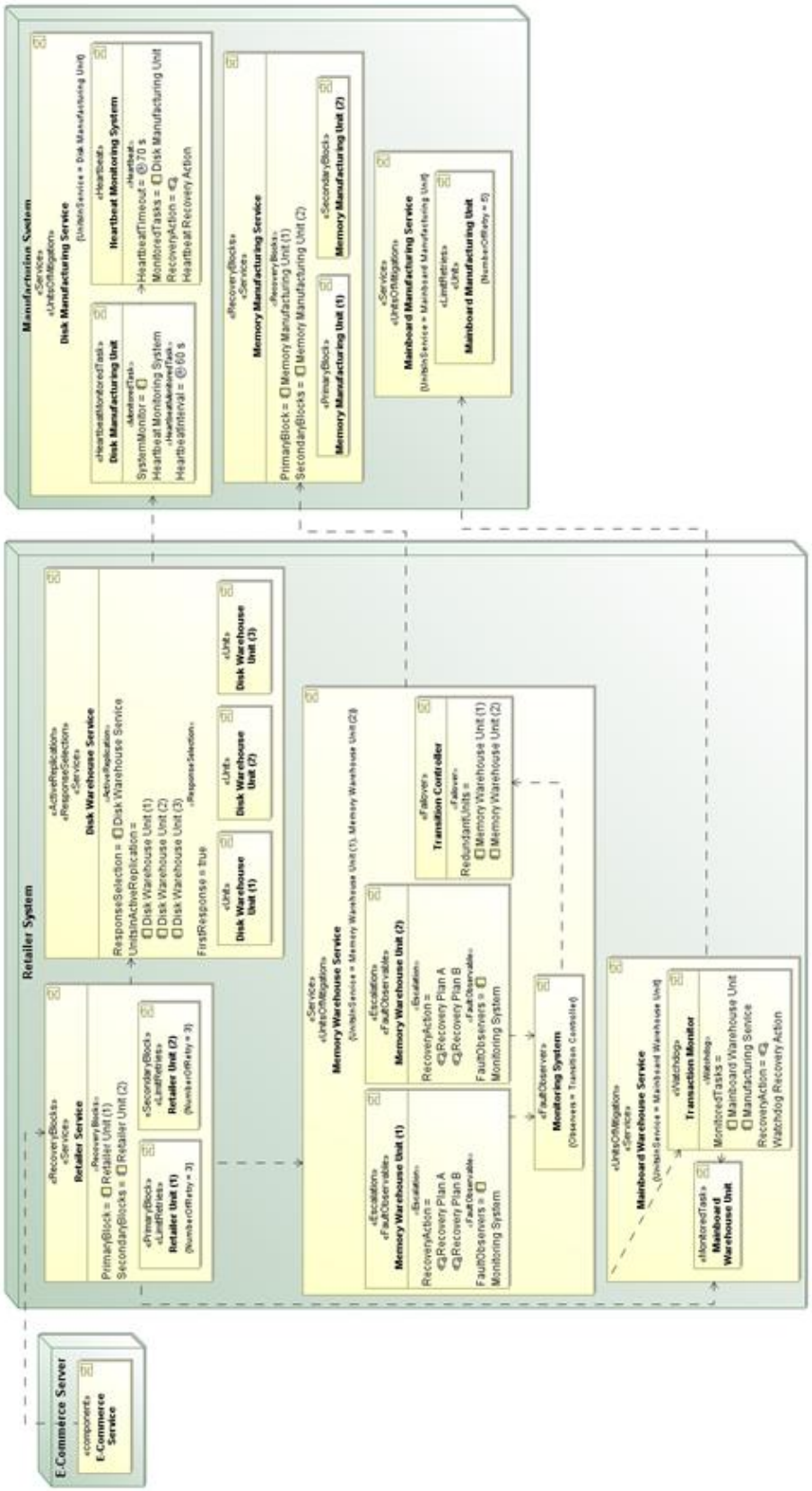
4.3 การประยุกต์ใช้ยูเอ็มแอลโปรไฟล์เข้ากับกรณีศึกษา

เมื่อทำการออกแบบยูเอ็มแอลโปรไฟล์ตามข้อที่ 4.1 แล้วจะนำบางส่วนของยูเอ็มแอลโปรไฟล์มาประยุกต์ใช้เข้ากับกรณีศึกษาโดยเลือกเฉพาะยูเอ็มแอลโปรไฟล์ที่ต้องการเพื่อให้เหมาะสมกับระบบ โดยในงานวิจัยนี้จะเลือกกรณีศึกษา: ตัวอย่างสถาปัตยกรรมโปรแกรมประยุกต์สำหรับระบบจัดการห่วงโซ่อุปทาน ซึ่งนำเสนอโดย ดับเบิลยูเอสไอ (WS-I: Web Services Interoperability Organization) โดยที่ในระบบจะประกอบด้วยองค์ประกอบหลัก ๆ คือ (1) ส่วนของผู้ใช้งานระบบ (Demo Customer) (2) ระบบของผู้ขายปลีก (Retailer System) (3) ระบบของการผลิต (Manufacturing System) ดังแสดงในภาพที่ 4.46

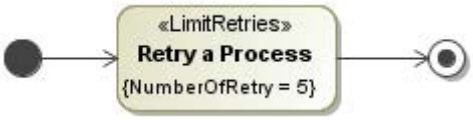


ภาพที่ 4.46 แผนภาพดีพลอยเมนต์ของตัวอย่างสถาปัตยกรรมโปรแกรมประยุกต์สำหรับระบบจัดการห่วงโซ่อุปทาน

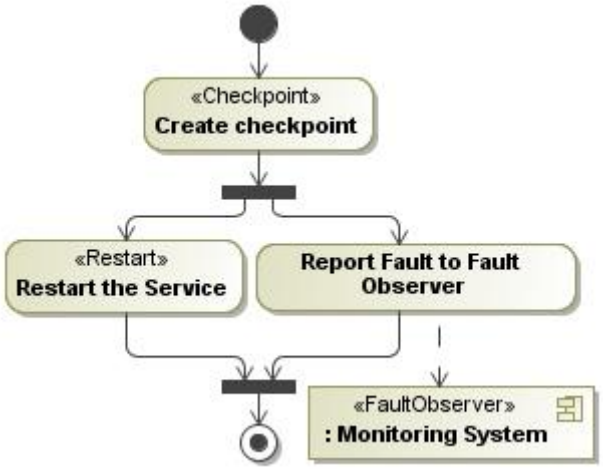
จากภาพที่ 4.46 ซึ่งแสดงแผนภาพดีพลอยเมนต์ของตัวอย่างสถาปัตยกรรมโปรแกรมประยุกต์สำหรับระบบจัดการห่วงโซ่อุปทานก่อนที่จะทำการประยุกต์ใช้แบบรูป จากนั้นจึงทำการประยุกต์ใช้ยูเอ็มแอลโปรไฟล์ของแบบรูปการทนต่อความผิดพร่องเข้ากับระบบจัดการห่วงโซ่อุปทาน ดังภาพที่ 4.47 ภาพที่ 4.48 ภาพที่ 4.49 ภาพที่ 4.50 และภาพที่ 4.51



ภาพที่ 4.47 แผนภาพดีพลอยเมนต์การประยุกต์ใช้ยูเอ็มแอลโปรไฟล์สำหรับแบบรูปการทนต่อความผิดพร่องสำหรับตัวอย่างสถาปัตยกรรมโปรแกรมประยุกต์สำหรับระบบจัดการห่วงโซ่อุปทาน



ภาพที่ 4.48 แผนภาพแอคทิวิตีสำหรับ Recovery Plan A



ภาพที่ 4.49 แผนภาพแอคทิวิตีสำหรับ Recovery Plan B

ภาพที่ 4.50 แผนภาพแอคทิวิตีสำหรับ Watchdog Recovery Action

ภาพที่ 4.51 แผนภาพแอคทิวิตีสำหรับ Heartbeat Recovery Action

จากภาพที่ 4.47 แสดงแผนภาพดีพลอยเมนต์การประยุกต์ใช้ยูเอ็มแอลโปรไฟล์สำหรับแบบรูปการทนต่อความผิดพร่องสำหรับตัวอย่างสถาปัตยกรรมโปรแกรมประยุกต์สำหรับระบบจัดการห่วงโซ่อุปทาน ซึ่งมีการประยุกต์ใช้แบบรูปการทนต่อความผิดพร่องโดยมีรายละเอียดดังนี้

4.3.1 เซอร์วิซสำหรับผู้ขายปลีก

เซอร์วิซสำหรับผู้ขายปลีกหรือ Retailer Service มีการประยุกต์ใช้แบบรูป Recovery Blocks ซึ่งจะประกอบด้วยหน่วยสำรองจำนวน 2 หน่วย หน่วยแรก คือ Retailer Unit (1) ซึ่งทำหน้าที่เป็นหน่วยปฐมภูมิ (PrimaryBlock) และมีการประยุกต์ใช้แบบรูป Limit Retries โดยกำหนด

จำนวนของการทำซ้ำ (NumberOfRetry) เป็น 3 เมื่อมีการทำซ้ำ Retailer Unit (1) ครบตามจำนวนของการทำซ้ำตามที่กำหนดไว้ จะเปลี่ยนไปทำงานยัง Retailer Unit (2) ซึ่งทำหน้าที่เป็นหน่วยทุติยภูมิ (SecondaryBlock) โดยที่มีการประยุกต์ใช้แบบรูป Limit Retries และมีการกำหนดจำนวนการทำซ้ำ (NumberOfRetry) เป็น 3 เช่นกัน ในกรณีที่ทำตามขั้นตอนของการจัดการกับข้อผิดพลาดครบทุกขั้นตอนแล้วแต่ยังทำงานไม่สำเร็จ จะถือว่าเซอร์วิซไม่สามารถทำงานสำเร็จ

4.3.2 เซอร์วิซสำหรับระบบโกดังสินค้าประเภทดิสก์

เซอร์วิซสำหรับระบบโกดังสินค้าประเภทดิสก์หรือ Disk Warehouse Service มีการประยุกต์ใช้แบบรูป Active Replication และประยุกต์ใช้สเตอรีโอไทพ์ ResponseSelection สำหรับแบบรูป Active Replication จะประกอบด้วยตัวสำรองจำนวน 3 ตัว คือ Disk Warehouse Unit (1) ถึง (3) โดยที่ตัวสำรองทั้งหมดจะทำงานพร้อมกัน และสเตอรีโอไทพ์ ResponseSelection กำหนดหน่วยที่ทำหน้าที่เลือกคำตอบที่เหมาะสม จากตัวอย่างมีการกำหนดให้เลือกคำตอบแรกที่ได้รับเป็นคำตอบที่เหมาะสม (FirstResponse) ดังนั้นเมื่อ Disk Warehouse Service ได้รับคำตอบใดเป็นคำตอบแรกจาก Disk Warehouse Unit ทั้ง 3 หน่วย จะนำคำตอบนั้นตอบไปยังผู้ร้องขอใช้บริการ ในกรณีที่ไม่ได้รับคำตอบจากหน่วยได้เลย จะถือว่าเซอร์วิซไม่สามารถทำงานสำเร็จ

4.3.3 เซอร์วิซสำหรับระบบโกดังสินค้าประเภทหน่วยความจำ

เซอร์วิซสำหรับระบบโกดังสินค้าประเภทหน่วยความจำหรือ Memory Warehouse Service มีการประยุกต์ใช้แบบรูป Units of Mitigation ซึ่งประกอบด้วยหน่วยสำรองจำนวน 2 หน่วย คือ Memory Warehouse Unit (1) และ Memory Warehouse Unit (2) หน่วยสำรองทั้ง 2 หน่วยได้มีการประยุกต์ใช้แบบรูป Escalation และสเตอรีโอไทพ์ FaultObservable ตามแบบรูป Fault Observer เมื่อมีการร้องขอมายัง Memory Warehouse Service การร้องขอจะถูกส่งไปยังหน่วยสำรองหน่วยใดหน่วยหนึ่ง ในกรณีที่มีข้อผิดพลาดเกิดขึ้นในหน่วยสำรอง RecoveryAction ที่กำหนดตามแบบรูป Escalation จะถูกเรียกมาทำงาน โดยมีการกระทำคือ การจัดการกับข้อผิดพลาดตามที่กำหนดใน Recovery Plan A ตามภาพที่ 4.48 จะถูกเรียกขึ้นมาทำงานก่อน ได้แก่ การจัดการตามแบบรูป Limit Retries และถ้าการจัดการไม่สำเร็จ การจัดการกับข้อผิดพลาดตามที่กำหนดใน Recovery Plan B ตามภาพที่ 4.49 จะถูกเรียกขึ้นมาทำงาน ได้แก่ การจัดการตามแบบรูป Checkpoint โดยเก็บสถานะของระบบเพื่อนำไปใช้ในหน่วยสำรองและ Restart หน่วยสำรองนั้น หากยังไม่สำเร็จจะทำการรายงานไปยัง Monitoring System ซึ่งทำหน้าที่เป็นหน่วยสังเกต โดยที่หน่วยสังเกตมีการประยุกต์ใช้แบบรูป Fault Observer จากนั้น Monitoring System จะรายงานข้อผิดพลาดนั้นไปยัง Observers ซึ่งก็คือ Transition Controller และ Transition Controller มี

การประยุกต์ใช้แบบรูป Failover เพื่อทำหน้าที่สลับการทำงานไปยังหน่วยสำรองอื่นที่ไม่มีข้อผิดพลาด ในการจัดการกับข้อผิดพลาดของเซอร์วิซ ถ้ายังมีข้อผิดพลาดเกิดขึ้นจะจัดการกับข้อผิดพลาดตามที่กำหนดใน Recovery Plan A และ Recovery Plan B ตามลำดับ ซึ่งภายในการจัดการกับข้อผิดพลาดตาม Recovery Plan B หน่วยสำรองจะทำการ Restart จนกว่าเซอร์วิซจะทำงานสำเร็จ

4.3.4 เซอร์วิซสำหรับระบบโกดังสินค้าประเภทเมนบอร์ด

เซอร์วิซสำหรับระบบโกดังสินค้าประเภทเมนบอร์ดหรือ Mainboard Warehouse Service มีการประยุกต์ใช้แบบรูป Units of Mitigation ซึ่งแบ่งออกเป็นหน่วยย่อย คือ Mainboard Warehouse Unit โดยมี Transaction Monitor ที่ประยุกต์ใช้แบบรูป Watchdog เพื่อทำหน้าที่เป็นหน่วยสังเกต เพื่อสังเกตการทำงานระหว่าง Mainboard Warehouse Unit และ Mainboard Manufacturer Service เมื่อมีข้อผิดพลาดจะทำการจัดการกับข้อผิดพลาดตามแบบรูป Restart ตามภาพที่ 4.50 หากยังมีข้อผิดพลาดเกิดขึ้นในหน่วยสำรอง จะทำการ Restart จนกว่าเซอร์วิซจะทำงานสำเร็จ

4.3.5 เซอร์วิซสำหรับระบบการผลิตสินค้าประเภทดิสก์

เซอร์วิซสำหรับระบบการผลิตสินค้าประเภทดิสก์หรือ Disk Manufacturing Service มีการประยุกต์ใช้แบบรูป Units of Mitigation ซึ่งแบ่งออกเป็นหน่วยย่อย คือ Disk Manufacturing Unit โดยมี Heartbeat Monitoring System ที่ประยุกต์ใช้แบบรูป Heartbeat เพื่อทำหน้าที่เป็นหน่วยสังเกตการส่งสัญญาณชีพของ Disk Manufacturing Unit ที่ประยุกต์ใช้สเตอรีโอไทพ์ HeartbeatMonitoredTask และมีการกำหนดรอบของการส่งสัญญาณชีพ (HeartbeatInterval) ตามคุณสมบัติ เป็น 60 วินาที (60 s) เมื่อ Heartbeat Monitoring System ไม่ได้รับสัญญาณชีพจาก Disk Manufacturing Unit ภายในเวลาที่กำหนด จะถือว่ามีข้อผิดพลาดเกิดขึ้นและจะทำการจัดการกับข้อผิดพลาดนั้นตามแบบรูป Restart ตามภาพที่ 4.51 หากยังมีข้อผิดพลาดเกิดขึ้นในหน่วยสำรอง จะทำการ Restart จนกว่าเซอร์วิซจะทำงานสำเร็จ

4.3.6 เซอร์วิซสำหรับระบบการผลิตสินค้าประเภทหน่วยความจำ

เซอร์วิซสำหรับระบบการผลิตสินค้าประเภทหน่วยความจำหรือ Memory Manufacturing Service มีการประยุกต์ใช้แบบรูป Recovery Blocks ซึ่งประกอบด้วยหน่วยสำรองจำนวน 2 หน่วย คือ Memory Manufacturing Unit (1) ทำหน้าที่เป็นหน่วยปฐมภูมิ (PrimaryBlock) และ Memory Manufacturing Unit (2) ซึ่งทำหน้าที่เป็นหน่วยทุติยภูมิ (SecondaryBlock) โดยที่

Memory Manufacturing Unit (1) จะถูกทำงานก่อน ถ้ามีข้อผิดพลาดเกิดขึ้นจะเปลี่ยนไปทำงานที่ Memory Manufacturing Unit (2) ตามลำดับ ถ้ายังทำงานไม่สำเร็จจะถือว่าเซอร์วิซทำงานไม่สำเร็จ

4.3.7 เซอร์วิซสำหรับระบบการผลิตสินค้าประเภทเมนบอร์ด

เซอร์วิซสำหรับระบบการผลิตสินค้าประเภทเมนบอร์ดหรือ Mainboard Manufacturing Service มีการประยุกต์ใช้แบบรูป Units of Mitigation ซึ่งประกอบด้วยหน่วยสำรอง คือ Mainboard Manufacturing Unit โดยหน่วยสำรองนี้มีการประยุกต์ใช้แบบรูป Limit Retries และกำหนดจำนวนครั้งของการทำซ้ำ (NumberOfRetry) เป็น 5 ครั้ง เมื่อหน่วยสำรองมีข้อผิดพลาดเกิดขึ้นจะทำการประมวลผลซ้ำตามจำนวนการทำซ้ำที่มีการกำหนดไว้ ในกรณีที่ทำครบจำนวนครั้งแล้ว หน่วยนั้นยังมีข้อผิดพลาดเกิดขึ้นจะถือว่าเซอร์วิซนั้นทำงานไม่สำเร็จ

บทที่ 5

การประเมินผล

ในบทนี้เป็นการประเมินผลการใช้งานยูเอ็มแอลโปรไฟล์สำหรับแบบรูปการทนต่อความผิดพร่องสำหรับระบบอิงบริการ ในการประเมินผลแบ่งออกเป็น 3 กลุ่มคือการประเมินความถูกต้องของการออกแบบยูเอ็มแอลโปรไฟล์ตามคำนิยาม การประเมินค่าคุณลักษณะเชิงคุณภาพของการออกแบบ และการประเมินผลความสามารถในการทนต่อความผิดพร่องของระบบที่พัฒนาจริง

5.1 การประเมินความถูกต้องของการออกแบบยูเอ็มแอลโปรไฟล์ตามคำนิยาม

ในการประเมินผลความถูกต้องของการออกแบบยูเอ็มแอลโปรไฟล์ตามนิยามของแบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่อง ที่นำเสนอโดย Hanmer [5] ทำการตรวจสอบโดยวิธีการตามรอยความต้องการของซอฟต์แวร์ (Requirement Traceability) [18]

การตามรอยความต้องการของซอฟต์แวร์ เป็นรูปแบบหนึ่งของการจัดการความต้องการซอฟต์แวร์ เพื่อให้แน่ใจว่าระบบพัฒนาขึ้นถูกต้องตามความต้องการของซอฟต์แวร์ การตามรอยสามารถทำได้โดยเชื่อมโยงความสัมพันธ์ระหว่างแต่ละระดับของความต้องการของซอฟต์แวร์และส่วนต่างๆที่ถูกพัฒนาขึ้นมาในขั้นตอนการพัฒนาซอฟต์แวร์ เช่น ในการพัฒนาซอฟต์แวร์หนึ่งควรจะมีการตามรอยระหว่าง การออกแบบเชิงสถาปัตยกรรม การออกแบบอย่างละเอียด คลาส โค้ด หรือเอกสารต่างๆ ซึ่งสิ่งต่างๆเหล่านี้ได้พัฒนามาจากความต้องการของซอฟต์แวร์ การตามรอยมี 2 รูปแบบคือ การตามรอยแบบไปข้างหน้า (Forward Traceability) ซึ่งมีการตามรอยจากความต้องการของระบบไปยังส่วนอื่นๆที่ถูกพัฒนาจากความต้องการของระบบ และการตามรอยแบบย้อนกลับ (Backward Traceability) ซึ่งเป็นการตามรอยย้อนกลับจากส่วนที่ถูกพัฒนาขึ้นมาในระบบไปยังความต้องการของระบบ

ในงานวิจัยนี้การประเมินความถูกต้องของการออกแบบยูเอ็มแอลโปรไฟล์ตามคำนิยาม เลือกการตามรอยแบบไปข้างหน้า โดยทำการเชื่อมโยงความสามารถของซอฟต์แวร์ตาม Hanmer [5] เข้ากับส่วนต่างๆของยูเอ็มแอลโปรไฟล์ที่ทำการออกแบบ แสดงดังตารางที่ 5.1 และแสดงเป็นแผนภาพการเชื่อมโยงได้ดังภาพที่ 5.1-ภาพที่ 5.3

ตารางที่ 5.1 การเชื่อมโยงความสามารถของซอฟต์แวร์เข้ากับส่วนต่างๆของยูเอ็มแอลโปรไฟล์ที่ทำการออกแบบ

กลุ่มของแบบรูป	**แบบรูป**	**ความสามารถของซอฟต์แวร์**	**การออกแบบยูเอ็มแอลโปรไฟล์**	**ภาพที่ 5.1-5.3**
Common		กำหนดหน่วยย่อยในระบบ เพื่อนำไปใช้กับแบบรูป Units of Mitigation	<<Unit>>	c1.1
		กำหนดหน่วยย่อยที่ทำหน้าที่รับผิดชอบหน้าที่บางอย่างในระบบ	<<ResponseUnit>>	c1.2
Architectural Pattern	Units of Mitigation	แบบรูปที่ออกแบบระบบให้เป็นหน่วยย่อย และระบุกลุ่มของหน่วยย่อยในระบบ	<<UnitsOfMitigation>> และคุณสมบัติ UnitsInService ของ <<UnitsOfMitigation>>	a1.1 และ a1.2
	Redundancy	เพิ่มความสามารถของหน่วยย่อยตามแบบรูป Units of Mitigation เพื่อทำหน้าที่เป็นตัวสำรองในระบบ	<<Redundancy>> สืบทอดมาจาก <<UnitsOfMitigation>>	a2.1
		การนำตัวสำรองมาประยุกต์ใช้มี 3 ประเภท คือ ตัวสำรองเชิงพื้นที่ ตัวสำรองเชิงเวลาและตัวสำรองเชิงสารสนเทศ (ตัวสำรองเชิงสารสนเทศจะไม่นำไปออกแบบยูเอ็มแอลโปรไฟล์)	<<SpatialRedundancy>> และ <<TemporalRedundancy>>	a2.2 และ a2.3
		แบบรูป Active Replication จะเป็นชนิดหนึ่งของตัวสำรองเชิงพื้นที่	<<ActiveReplication>> สืบทอดมาจาก <<SpatialRedundancy>>	a2.4

ตารางที่ 5.1 การเชื่อมโยงความสามารถของซอฟต์แวร์เข้ากับส่วนต่างๆของยูเอ็มแอลโปรไฟล์ที่ทำการออกแบบ (ต่อ)

กลุ่มของแบบรูป	แบบรูป	ความสามารถของซอฟต์แวร์	การออกแบบยูเอ็มแอลโปรไฟล์	ภาพที่ 5.1-5.3
Architectural Pattern	Redundancy	แบบรูป Active Replication ประกอบด้วยหน่วยสำรองที่ทำงานพร้อมกัน	คุณสมบัติ UnitsInActiveReplication ของ <<ActiveReplication>>	a2.5
		แบบรูป Active Replication ต้องประกอบด้วยหน่วยที่ทำหน้าที่รับผิดชอบในการเลือกผลลัพธ์ที่เหมาะสม	คุณสมบัติ ResponseSelection ของ <<ActiveReplication>>	a2.6
		ผลลัพธ์ที่ได้จากแต่ละหน่วยย่อย ต้องทำการเลือกมาหนึ่งค่าที่เหมาะสม โดยทำการเลือกจากผลลัพธ์ของหน่วยย่อยแรกที่ตอบมาหรือจากวิธีการในการคัดเลือก	<<ResponseSelection>>	a2.7
	Recovery Blocks	แบบรูปที่มีการใช้งานตัวสำรองแล้ว ประกอบด้วยส่วนปฐมภูมิ (Primary Block) และส่วนทุติยภูมิ (Secondary Block) ตามแบบรูป Recovery Blocks	<<RecoveryBlocks>>	a3.1
		ประกอบด้วยหน่วยย่อยที่ทำหน้าที่เป็นหน่วยปฐมภูมิ (Primary Block) ซึ่งจะเริ่มทำงานก่อน	คุณสมบัติ PrimaryBlock ของ <<RecoveryBlocks>>	a3.2
		ประกอบด้วยรายการหน่วยย่อยที่ทำหน้าที่เป็นหน่วยทุติยภูมิ (Secondary Block) ซึ่งจะเริ่มทำงานเมื่อหน่วยปฐมภูมิทำงานไม่สำเร็จ	คุณสมบัติ SecondaryBlocks ของ <<RecoveryBlocks>>	a3.3

ตารางที่ 5.1 การเชื่อมโยงความสามารถของซอฟต์แวร์เข้ากับส่วนต่างๆของยูเอ็มแอลโปรไฟล์ที่ทำการออกแบบ (ต่อ)

กลุ่มของแบบรูป	แบบรูป	ความสามารถของซอฟต์แวร์	การออกแบบยูเอ็มแอลโปรไฟล์	ภาพที่ 5.1-5.3
Architectural Pattern	Recovery Blocks	มีการใช้แบบรูป Checkpoint มาช่วยในการเก็บข้อมูลจากส่วนปฐมภูมิเพื่อให้สามารถไปทำงานยังส่วนทุติยภูมิได้ในสถานะเดิม	คุณสมบัติ Checkpoint ของ <<PrimaryBlock>>	a3.4
	Someone in Charge	มีการกำหนดหนึ่งหน่วยย่อยเพื่อเป็นผู้รับผิดชอบเกี่ยวกับข้อผิดพลาดที่เกิดขึ้นในระบบ	<<ResponseUnit>>	a4.1
		เมื่อมีข้อผิดพลาดเกิดขึ้นในหน่วยย่อยใดหน่วยย่อยหนึ่ง ต้องมีผู้รับผิดชอบกับข้อผิดพลาดนั้น ตามแบบรูป Someone in Charge	<<SomeoneInCharge>>	a4.2
		หน่วยรับผิดชอบมีหน้าที่ในการเฝ้าสังเกตหรือควบคุมหน่วยย่อยอื่นๆ	คุณสมบัติ ResponseUnits ของ <<SomeoneInCharge>>	a4.3
	Escalation	หน่วยย่อยต้องมีการยกระดับการจัดการกับข้อผิดพลาดที่เกิดขึ้นให้เข้มขึ้นเรื่อยๆ	<<Escalation>> และ คุณสมบัติ RecoveryAction ของ <<Escalation>>	a5.1 และ a5.2
	Fault Observer	มีหน่วยสังเกตที่ทำหน้าที่สังเกตความผิดพร่องในระบบ ซึ่งเรียกว่า Fault Observer	<<FaultObserver>> สืบทอดมาจาก <<ResponseUnit>>	a6.1
		เมื่อมีความผิดพร่องเกิดขึ้นและหน่วยสังเกตได้รับการรายงานจากหน่วยย่อยนั้น จะรายงานการเกิดความผิดพร่องให้หน่วยที่สนใจทุกๆหน่วย	คุณสมบัติ Observers ของ <<FaultObserver>>	a6.2

ตารางที่ 5.1 การเชื่อมโยงความสามารถของซอฟต์แวร์เข้ากับส่วนต่างๆของยูเอ็มแอลโปรไฟล์ที่ทำการออกแบบ (ต่อ)

กลุ่มของแบบรูป	แบบรูป	ความสามารถของซอฟต์แวร์	การออกแบบยูเอ็มแอลโปรไฟล์	ภาพที่ 5.1-5.3
Architectural Pattern	Fault Observer	มีการกำหนดหน่วยสังเกตให้กับหน่วยย่อยในระบบ เมื่อเกิดความผิดพร่อง จะรายงานไปยังหน่วยสังเกตนั้น	<<FaultObservable>> และคุณสมบัติ FaultObserver ของ FaultObservable	a6.3 และ a6.4
Detection Pattern	System Monitor	กำหนดหน่วยที่ทำหน้าที่ตรวจสอบการทำงานของหน่วยย่อยอื่นของระบบว่ายังทำงานอยู่หรือไม่	<<SystemMonitor>>	d1.1
		มีการกำหนดหน่วยย่อยที่ถูกตรวจสอบการทำงานโดยหน่วยตรวจสอบว่ายังทำงานอยู่หรือไม่	คุณสมบัติ MonitoredTask ของ <<SystemMonitor>>	d1.2
		มีการกำหนดเวลาที่เหมาะสม เมื่อไม่มีการตอบรับจากหน่วยที่ถูกตรวจสอบ จะถือว่ามีข้อผิดพลาดเกิดขึ้นในระบบ	คุณสมบัติ TimeBeforeAlarm ของ <<SystemMonitor>>	d1.3
	Heartbeat	กำหนดหน่วยที่ทำหน้าที่ตรวจสอบการส่งสัญญาณชีพของหน่วยย่อยอื่นในระบบว่ายังทำงานอยู่หรือไม่	<<Heartbeat>>	d2.1
		กำหนดหน่วยถูกเฝ้ามองการส่งสัญญาณชีพ	คุณสมบัติ MonitoredTask ของ <<Heartbeat>>	d2.2
		กำหนดระยะเวลาที่หน่วยย่อยที่ถูกเฝ้ามองไม่ตอบสนอง จะถือว่ามีข้อผิดพลาดเกิดขึ้น	คุณสมบัติ HeartbeatTimeout ของ <<Heartbeat>>	d2.3

ตารางที่ 5.1 การเชื่อมโยงความสามารถของซอฟต์แวร์เข้ากับส่วนต่างๆของยูเอ็มแอลโปรไฟล์ที่ทำการออกแบบ (ต่อ)

กลุ่มของแบบรูป	**แบบรูป**	**ความสามารถของซอฟต์แวร์**	**การออกแบบยูเอ็มแอลโปรไฟล์**	**ภาพที่ 5.1-5.3**
Detection Pattern	Heartbeat	ถ้าหน่วยย่อยที่ถูกเฝ้ามองไม่ตอบสนองในเวลาที่กำหนด จะต้องทำการจัดการกับข้อผิดพลาดตามกระบวนการที่กำหนด	คุณสมบัติ RecoveryAction ของ <<Heartbeat>>	d2.4
		หน่วยที่ถูกเฝ้ามองจะต้องส่งสัญญาณชีพไปยังหน่วยเฝ้ามองของตนเอง	คุณสมบัติ SystemMonitor ของ <<MonitoredTask>>	d2.5
		หน่วยที่ถูกเฝ้ามองจะต้องส่งสัญญาณชีพไปยังหน่วยเฝ้ามอง ตามช่วงเวลาที่กำหนด	คุณสมบัติ HeartbeatInterval ของ <<HeartbeatMonitoredTask>>	d2.6
	Acknowledgement	กำหนดให้การร้องขอต้องมีการตอบรับทุกการร้องขอ	<<Acknowledgement>>	d3.1
		กำหนดให้ฝ่ายที่ถูกร้องขอจะต้องมีการตอบรับไปยังผู้ร้องขอทุกครั้งเพื่อให้ผู้ร้องขอทราบถึงสถานะของผู้ที่ถูกร้องขอ	คุณสมบัติ Requestor และ MonitoredTask ของ <<Acknowledgement>>	d3.2
	Watchdog	กำหนดฮาร์ดแวร์หรือซอฟต์แวร์ที่ทำหน้าที่เฝ้ามองการทำงานปกติของระบบ	<<Watchdog>>	d4.1
		กำหนดหน่วยที่ถูกเฝ้ามองการทำงานปกติ	คุณสมบัติ MonitoredTask ของ <<Watchdog>>	d4.2

ตารางที่ 5.1 การเชื่อมโยงความสามารถของซอฟต์แวร์เข้ากับส่วนต่างๆของยูเอ็มแอลโปรไฟล์ที่ทำการออกแบบ (ต่อ)

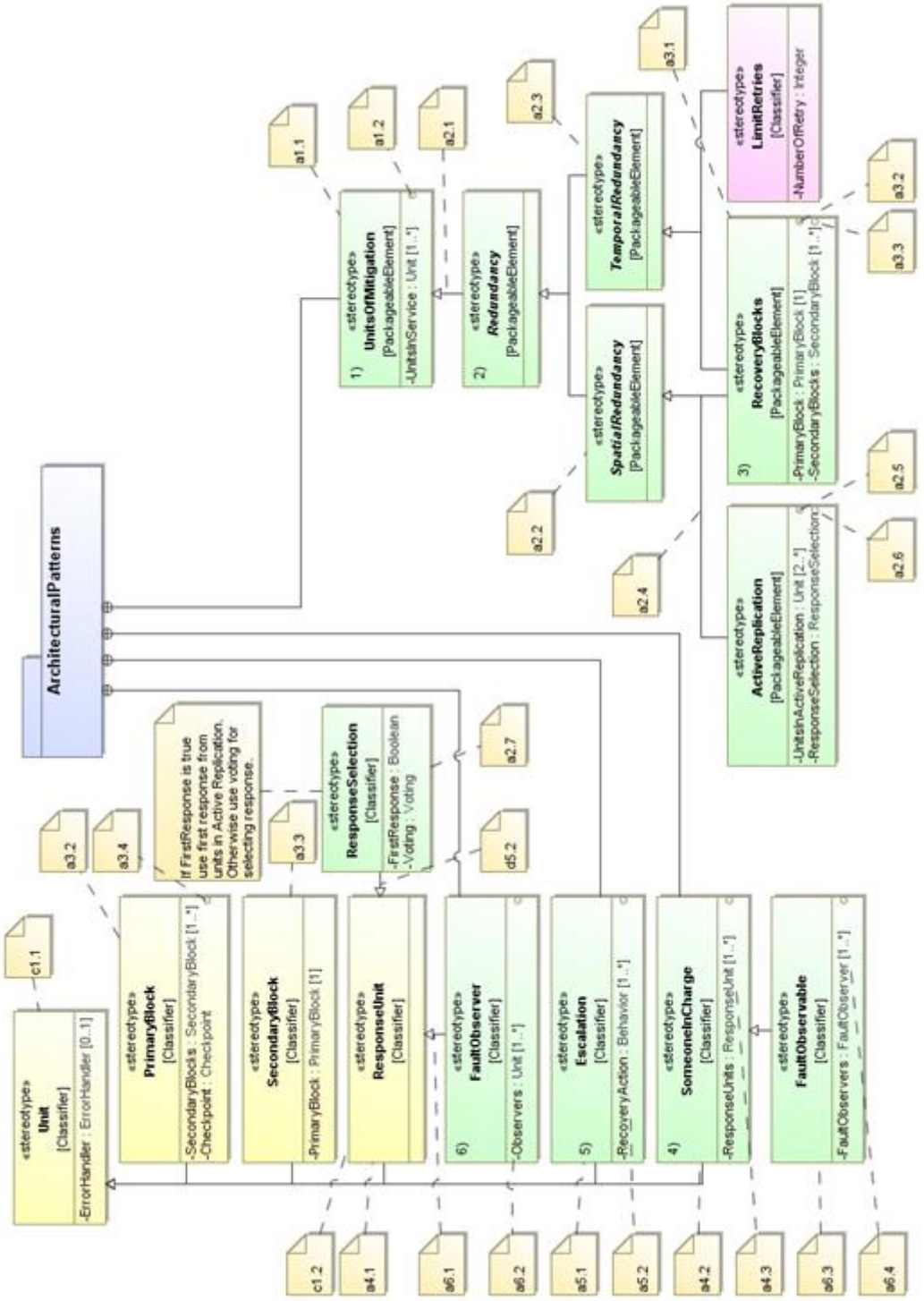
กลุ่มของแบบรูป	แบบรูป	ความสามารถของซอฟต์แวร์	การออกแบบยูเอ็มแอลโปรไฟล์	ภาพที่ 5.1-5.3
Detection Pattern	Watchdog	ถ้าหน่วยเฝ้ามองตรวจพบว่ามีข้อผิดพลาดเกิดขึ้นจะจัดการกับข้อผิดพลาดนั้น ตามที่ได้กำหนดไว้	คุณสมบัติ RecoveryAction ของ <<Watchdog>>	d4.3
	Voting	เมื่อมีคำตอบมามากกว่าหนึ่งคำตอบจะต้องมีวิธีในการเลือกคำตอบที่เหมาะสม	คุณสมบัติ VotingAlgorithm ของ class Voting	d5.1
		การทำการเลือกคำตอบจะต้องถูกจัดการโดยหน่วยที่ทำหน้าที่เป็น Someone in Charge	ResponseSelection สืบทอดมาจาก ResponseUnit	d5.2
Error Recovery Pattern	Error Handler	กำหนดหน่วยของการจัดการข้อผิดพลาดซึ่งแยกออกจากการทำงานหลักของระบบ	<<ErrorHandler>>	e1.1
		เมื่อมีการจัดการข้อผิดพลาด อาจจำเป็นต้องหยุดการทำงานของระบบก่อน เพื่อทำการแก้ไขข้อผิดพลาด	คุณสมบัติ ShouldStopTheProcess ของ <<ErrorHandler>>	e1.2
	Restart	จัดการกับข้อผิดพลาดในระบบ โดยการกลับไปทำใหม่ตั้งแต่จุดเริ่มต้น	<<Restart>>	e2.1
	Rollback	จัดการกับข้อผิดพลาดในระบบ โดยการกลับไปยังจุดที่ทำคำสั่งล่าสุดก่อนการเกิดข้อผิดพลาด	<<Rollback>>	e3.1
	Roll-Forward	จัดการกับข้อผิดพลาดในระบบ โดยการละเว้นข้อผิดพลาดนั้นและข้ามไปทำงานยังจุดถัดไป	<<Roll-Forward>>	e4.1

ตารางที่ 5.1 การเชื่อมโยงความสามารถของซอฟต์แวร์เข้ากับส่วนต่างๆของยูเอ็มแอลโปรไฟล์ที่ทำการออกแบบ (ต่อ)

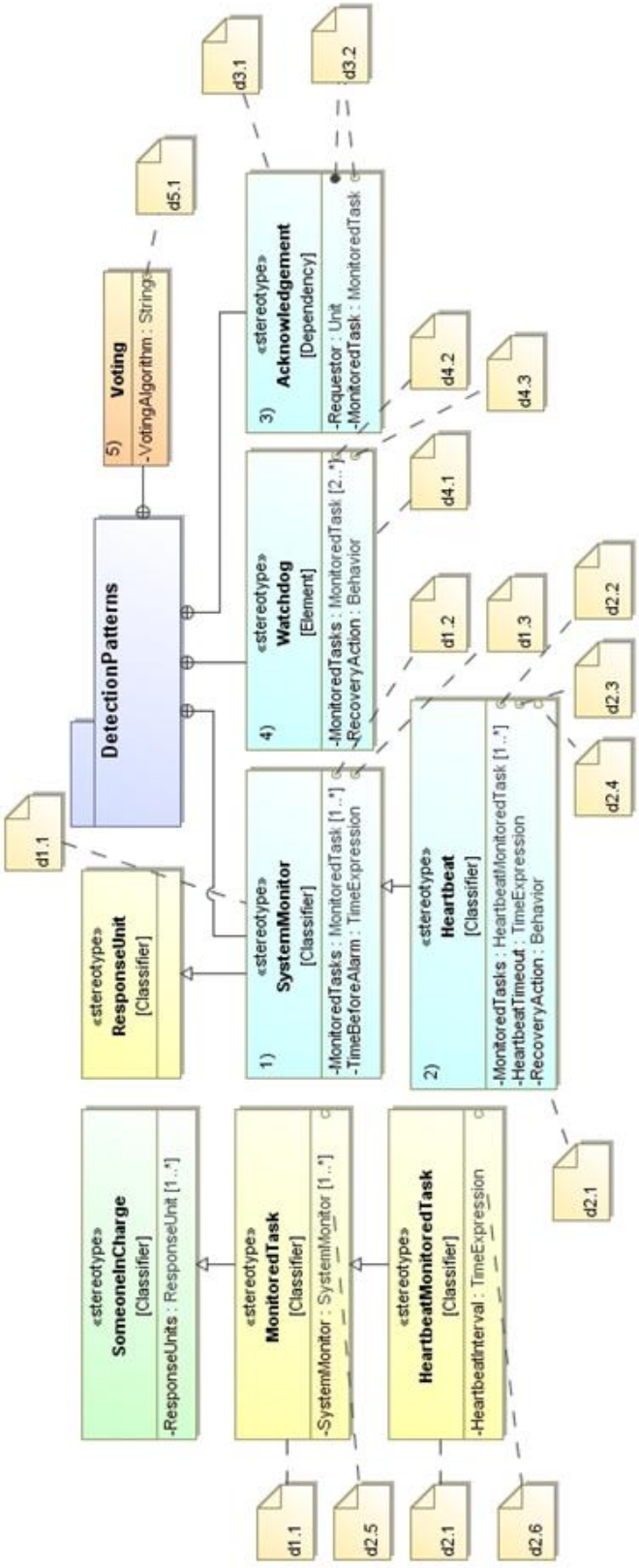
กลุ่มของแบบรูป	แบบรูป	ความสามารถของซอฟต์แวร์	การออกแบบยูเอ็มแอลโปรไฟล์	ภาพที่ 5.1-5.3
Error Recovery Pattern	Return to Reference Point	จัดการกับข้อผิดพลาดในระบบ โดยการกลับไปทำงานยังจุดอ้างอิงที่มีการบันทึกไว้	<<ReturnToReferencePoint>>	e5.1
	LimitRetries	จำกัดการทำซ้ำ หากไม่มีการแก้ไขเปลี่ยนแปลงระบบส่วนที่เกิดข้อผิดพลาด	<<LimitRetries>>	e6.1
		กำหนดจำนวนครั้งของการทำซ้ำ เพื่อไม่ให้ระบบติดอยู่ในการจัดการข้อผิดพลาดนั้นซ้ำๆ	คุณสมบัติ NumberOfRetry ของ <<LimitRetries>>	e6.2
	Failover	จัดการการสลับไปใช้งานยังหน่วยสำรอง	<<Failover>>	e7.1
		เมื่อมีข้อผิดพลาดเกิดขึ้นภายในหน่วยหนึ่ง หน่วยที่เพิ่มเข้ามาในระบบจะทำการกู้คืนระบบโดยการสลับไปยังหน่วยสำรองที่กำหนดไว้	คุณสมบัติ RedundantUnit ของ <<Failover>>	e7.2
	Checkpoint	ทำการเก็บสถานะของระบบเพื่อให้ระบบสามารถเริ่มต้นทำงานใหม่ตามสถานะนั้น	<<Checkpoint>>	e8.1
	What to Save	กำหนดข้อมูลที่ต้องการบันทึก ตามแบบรูป Checkpoint	<<WhatToSave>>	e9.1
	Remote Storage	กำหนดสถานที่ที่ต้องการบันทึกข้อมูล ตามแบบรูป Checkpoint	<<RemoteStorage>>	e10.1
	Individuals Decide Timing	ทำการเก็บสถานะของระบบอย่างเป็นอัตโนมัติ ตามช่วงเวลาที่กำหนด	<<IndividualsDecideTiming>>	e11.1

ตารางที่ 5.1 การเชื่อมโยงความสามารถของซอฟต์แวร์เข้ากับส่วนต่างๆของยูเอ็มแอลโปรไฟล์ที่ทำการออกแบบ (ต่อ)

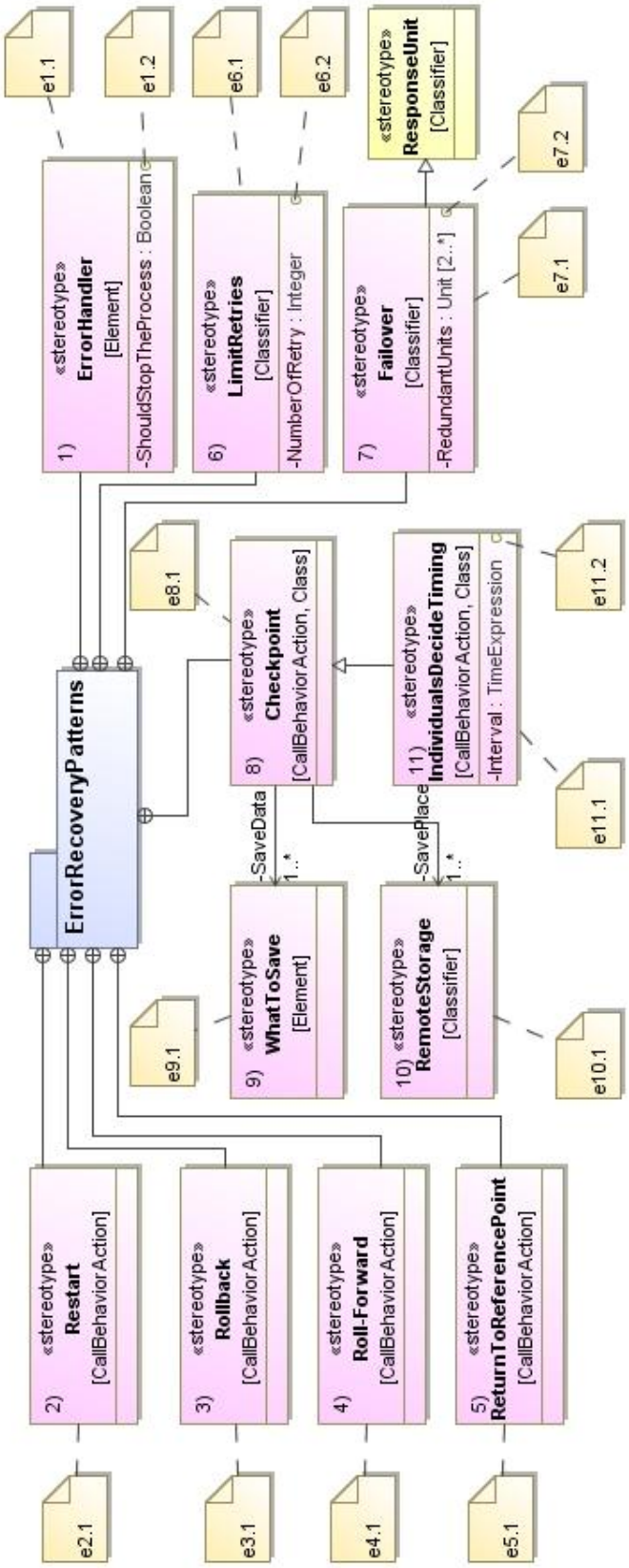
กลุ่มของแบบรูป	**แบบรูป**	**ความสามารถของซอฟต์แวร์**	**การออกแบบยูเอ็มแอลโปรไฟล์**	**ภาพที่ 5.1-5.3**
Error Recovery Pattern	Individuals Decide Timing	การกำหนดช่วงเวลาที่เหมาะสมในการเก็บข้อมูล	คุณสมบัติ Interval ของ <<IndividualsDecideTiming>>	e11.2



ภาพที่ 5.1 ยูเอ็มแอลโปรไฟล์สำหรับแบบรูปในกลุ่มของแบบรูปเชิงสถาปัตยกรรมพร้อมการเชื่อมโยงกับความสามารถของซอฟต์แวร์



ภาพที่ 5.2 ยูเอ็มแอลโปรไฟล์สำหรับแบบรูปในกลุ่มของแบบรูปการตรวจหาข้อผิดพลาดพร้อมการเชื่อมโยงกับความสามารถของซอฟต์แวร์



ภาพที่ 5.3 ยูเอ็มแอลโปรไฟล์สำหรับแบบรูปในกลุ่มของแบบรูปการกู้ระบบจากข้อผิดพลาดพร้อมการเชื่อมโยงกับความสามารถของซอฟต์แวร์

5.2 การประเมินค่าคุณสมบัติทางการออกแบบและคุณลักษณะเชิงคุณภาพ

การประเมินค่าคุณสมบัติทางการออกแบบและคุณลักษณะเชิงคุณภาพ เพื่อดูผลกระทบที่เกิดขึ้นจากการประยุกต์ใช้ยูเอ็มแอลโปรไฟล์สำหรับแบบรูปการทนต่อความผิดพร่องเข้ากับกรณีศึกษาในข้อที่ 4.3 เมื่อเปรียบเทียบกับระบบกรณีศึกษาที่ไม่ได้ประยุกต์ใช้ยูเอ็มแอลโปรไฟล์ โดยพิจารณาคุณลักษณะเชิงคุณภาพของทั้งระบบ ดังนี้

- ประสิทธิผล (Effectiveness) คือระดับความต้องการทางธุรกิจซึ่งสะท้อนอยู่ในการออกแบบ
- ความสามารถในการทำความเข้าใจ (Understandability) คือระดับความสามารถในการเรียนรู้หรือทำความเข้าใจการออกแบบ
- ความยืดหยุ่น (Flexibility) คือระดับความสามารถของระบบในการเปลี่ยนแปลงการออกแบบเพื่อรองรับการทำงานใหม่ที่จะเพิ่มเข้ามา
- ความสามารถในการนำกลับมาใช้ใหม่ (Reusability) คือระดับความสามารถในการออกแบบระบบเพื่อนำกลับมาใช้ใหม่ได้ในอนาคต

คุณลักษณะเชิงคุณภาพดังกล่าวข้างต้นเป็นผลสืบเนื่องมาจากค่ามาตรวัดคุณสมบัติทางการออกแบบสำหรับเอสโอเอซึ่งนำเสนอโดย Shim และคณะ [19] ดังนี้

- การต่อประกบ (Coupling) คือระดับการขึ้นต่อกันระหว่างเซอร์วิซเฉลี่ยทั้งระบบ สามารถคำนวณได้จากสูตร ดังต่อไปนี้

$$Coupling = \frac{Number\ of\ Directly\ Connected\ Producer\ Services\ +\ Number\ of\ Directly\ Connected\ Consumer\ Services}{System\ Size\ in\ Number\ of\ Services}$$

- การเชื่อมแน่น (Cohesion) คือระดับความสัมพันธ์ระหว่างโอเปอเรชันภายในเซอร์วิซเฉลี่ยทั้งระบบ สามารถคำนวณได้จากสูตร ดังต่อไปนี้

$$Cohesion = \frac{System\ Size\ in\ Number\ of\ Service}{Total\ Number\ of\ Message\ Used}$$

- ความซับซ้อน (Complexity) คือระดับความยากในการทำความเข้าใจความสัมพันธ์ระหว่างเซอร์วิซในระบบ สามารถคำนวณได้จากสูตร ดังต่อไปนี้

$$Complexity = Number\ of\ Synchronous\ Operations\ +\ Number\ of\ Asynchronous\ Operation\ \times\ 1.5$$

- ขนาดของการออกแบบ (Design Size) คือ ขนาดของการออกแบบระบบ สามารถคำนวณได้จากสูตร ดังต่อไปนี้

$$Design\ Size = System\ Size\ in\ Number\ of\ Services$$

- ขนาดของเซอร์วิซ (Service Granularity) คือระดับความเหมาะสมของขนาดเซอร์วิซเฉลี่ยทั้งระบบ โดยพิจารณาจากความสมบูรณ์และปริมาณของงานที่เซอร์วิซทำ ซึ่งขึ้นอยู่กับจำนวนโอเปอเรชันในเซอร์วิซและจำนวนข้อความที่ใช้งานร่วมกัน สามารถคำนวณได้จากสูตร ดังต่อไปนี้

$$Service\ Granularity = \frac{\left(\frac{Number\ of\ Synchronous\ Operations\ +\ Number\ of\ Asynchronous\ Operation}{System\ Size\ in\ Number\ of\ Services}\right)^2}{\left(\frac{Total\ Number\ of\ Message\ Used}{System\ Size\ in\ Number\ of\ Services}\right)^2}$$

- ขนาดของพารามิเตอร์ (Parameter Granularity) คือระดับความเหมาะสมของขนาดพารามิเตอร์ของโอเปอเรชันของเซอร์วิซเฉลี่ยทั้งระบบ โดยพิจารณาจากปริมาณข้อมูลที่ครอบคลุมโดยพารามิเตอร์ สามารถคำนวณได้จากสูตร ดังต่อไปนี้

$$Parameter\ Granularity = \frac{Number\ of\ Asynchronous\ Operation\ +\ Number\ of\ Synchronous\ Operations\ -\ Number\ of\ Fine-Grained\ Parameter\ Operations}{Number\ of\ Asynchronous\ Operation + Number\ of\ Synchronous\ Operations}$$

- ความสามารถในการถูกค้นพบและใช้งาน (Consumability) คือความสามารถของเซอร์วิซในการถูกค้นพบและเรียกใช้งานโดยผู้อื่นเฉลี่ยทั้งระบบ โดยพิจารณาจากจำนวนชื่อของเซอร์วิซและโอเปอเรชันที่สื่อความหมาย สามารถคำนวณได้จากสูตร ดังต่อไปนี้

$$Consumability = \frac{System\ Size\ in\ Number\ of\ Service\ -\ Number\ of\ Inadequately\ Named\ Services}{System\ Size\ in\ Number\ of\ Service\ \times\ 2} + \frac{Number\ of\ Synchronous\ Operations\ +\ Number\ of\ Asynchronous\ Operation\ -\ Number\ of\ Inadequately\ Named\ Operations}{(Number\ of\ Synchronous\ Operations\ +\ Number\ of\ Asynchronous\ Operation)\ \times\ 2}$$

ในการประยุกต์ใช้ยูเอ็มแอลโปรไฟล์เข้ากับกรณีศึกษาในข้อที่ 4.3 หน่วยสำรองและหน่วยควบคุมหรือตรวจสอบ ซึ่งเพิ่มเติมเข้ามาถือเป็นเซอร์วิซที่เพิ่มขึ้นในระบบที่ถูกออกแบบให้ทนต่อความผิดพร่อง ผลการเปรียบเทียบค่าคุณสมบัติทางการออกแบบของระบบกรณีศึกษาก่อนและหลังการประยุกต์ใช้ยูเอ็มแอลโปรไฟล์ ดังภาพที่ 4.46 และภาพที่ 4.47 ตามลำดับ แสดงในตารางที่ 5.2

ตารางที่ 5.2 การเปรียบเทียบค่าคุณสมบัติทางการออกแบบของระบบกรณีศึกษาก่อนและหลังการประยุกต์ใช้แบบรูปการทนต่อความผิดพร่อง

Design Property	Value		Normalized Value	
	Before	After	Before	After
Coupling	1.75	2.17	1	1.11
Cohesion	0.75	2	1	2
Complexity	6.5	27.5	1	4.23
Design Size	8	24	1	3
Service Granularity	0.56	4.68	1	8.36
Parameter Granularity	1	1	1	1
Consumability	1	1	1	1

จากตารางที่ 5.2 มีการแสดงผลการคำนวณค่าคุณสมบัติทางการออกแบบ ซึ่งคุณสมบัติเหล่านั้นจะมีผลกระทบต่อคุณลักษณะเชิงคุณภาพของระบบอิงบริการในแนวทางที่แตกต่างกัน แสดงดังตารางที่ 5.3

ตารางที่ 5.3 ทิศทางของผลกระทบที่คุณสมบัติทางการออกแบบมีต่อคุณลักษณะเชิงคุณภาพ [19]

Design Property	Quality Attribute			
	Effectiveness	Understandability	Flexibility	Reusability
Coupling		↓	↓	↓
Cohesion	↑	↑		↑
Complexity		↓		
Design Size		↓		
Service Granularity	↑	↑	↑	↑
Parameter Granularity	↑	↑	↑	
Consumability		↑		↑

จากตารางที่ 5.3 แสดงทิศของผลกระทบที่คุณสมบัติทางการออกแบบมีต่อคุณลักษณะเชิงคุณภาพ และสามารถนำคุณสมบัติทางการออกแบบไปคำนวณค่าคุณลักษณะเชิงคุณภาพได้ตามสูตรที่ซึ่งนำเสนอโดย Shim และคณะ [19] โดยมีรายละเอียดแสดงดังตารางที่ 5.4

ตารางที่ 5.4 สูตรการคำนวณคุณลักษณะเชิงคุณภาพ [19]

Quality Attribute	Formula
Effectiveness	0.33 x Cohesion + 0.33 x Service Granularity + 0.33 x Parameter Granularity
Understandability	-0.66 x Coupling + 0.25 x Cohesion -0.66 x Complexity - 0.66 x Design Size + 0.25 x Service Granularity + 0.25 x Parameter Granularity + 0.25 x Consumability
Flexibility	-0.22 x Coupling + 0.61 x Service Granularity + 0.61 x Parameter Granularity
Reusability	-0.5 x Coupling + 0.5 x Cohesion + 0.5 x Service Granularity + 0.5 x Consumability

จากสูตรการคำนวณคุณลักษณะเชิงคุณภาพจาก

ตารางที่ 5.4 สามารถนำมาประเมินผลคุณลักษณะเชิงคุณภาพจากคุณสมบัติทางการออกแบบ ของระบบกรณีศึกษาก่อนและหลังการประยุกต์ใช้แบบรูปการทนต่อความผิดพร่องโดยคำนวณจากค่าคุณสมบัติทางการออกแบบซึ่งถูกนอร์มัลไลซ์แล้ว ซึ่งการประเมินผลแสดงดังตารางที่ 5.5

ตารางที่ 5.5 การเปรียบเทียบค่าคุณลักษณะเชิงคุณภาพของระบบกรณีศึกษาก่อนและหลังการประยุกต์ใช้แบบรูปการทนต่อความผิดพร่อง

Quality Attribute	Value		Normalized Value	
	Before	After	Before	After
Effectiveness	0.99	3.97	1	4.01
Understandability	-0.98	-2.25	-1	-2.30
Flexibility	1	5.47	1	5.47
Reusability	1	5.46	1	5.46

จากตารางที่ 5.5 จะเห็นได้ว่าค่าคุณลักษณะเชิงคุณภาพซึ่งมีผลมาจากคุณสมบัติทางการออกแบบสามารถเป็นได้ในทางบวกและในทางลบ ซึ่งจะเห็นได้ว่าคุณสมบัติทางการออกแบบทุกตัวมีผลกระทบในทางบวกต่อคุณลักษณะเชิงคุณภาพ ยกเว้นความสามารถในการทำความเข้าใจ (Understandability) เนื่องจากค่าการต่อประกบและค่าความซับซ้อนที่เพิ่มขึ้นจากการประยุกต์ใช้แบบรูปส่งผลมากในเชิงลบต่อการทำความเข้าใจ

5.3 การประเมินผลความสามารถในการทนต่อความผิดพร่องของระบบที่พัฒนาจริง

การประเมินผลความสามารถในการทนต่อความผิดพร่องของระบบที่พัฒนาจริงแบ่งออกเป็น 2 กลุ่มคือการประเมินผลด้านความเชื่อถือได้ และด้านสมรรถนะ เพื่อเปรียบเทียบระหว่างระบบกรณีศึกษาที่ไม่ได้ประยุกต์ใช้ยูเอ็มแอลโปรไฟล์และระบบกรณีศึกษาที่ประยุกต์ใช้ยูเอ็มแอลโปรไฟล์ โดยพิจารณาการประเมินผลของทั้งระบบ โดยใช้แนวทางในการทดสอบเช่นเดียวกับงานวิจัยของ Zheng และ Lyu [15]

5.3.1 การประเมินผลด้านความเชื่อถือได้

การประเมินผลด้านความเชื่อถือได้ ทำการทดสอบจากการพัฒนาระบบจริง ตามการออกแบบตามภาพที่ 4.46 และภาพที่ 4.47 โดยใช้ภาษาจาวาผ่านทางโปรแกรม Eclipse 4.2.0 ในการพัฒนา ซึ่งการพัฒนาในรูปแบบของเซอร์วิซอาศัยไลบรารี JAX-WS ของจาวา ใช้ฐานข้อมูล H2 1.2.167 การติดต่อฐานข้อมูลทำผ่านไลบรารี Hibernate 4.2.0 และใช้ Apache Tomcat 7.0.39 เป็นเว็บเซิร์ฟเวอร์ โดยที่ระบบจะมีรายละเอียดโอเปอเรชันและการทำงาน ดังนี้

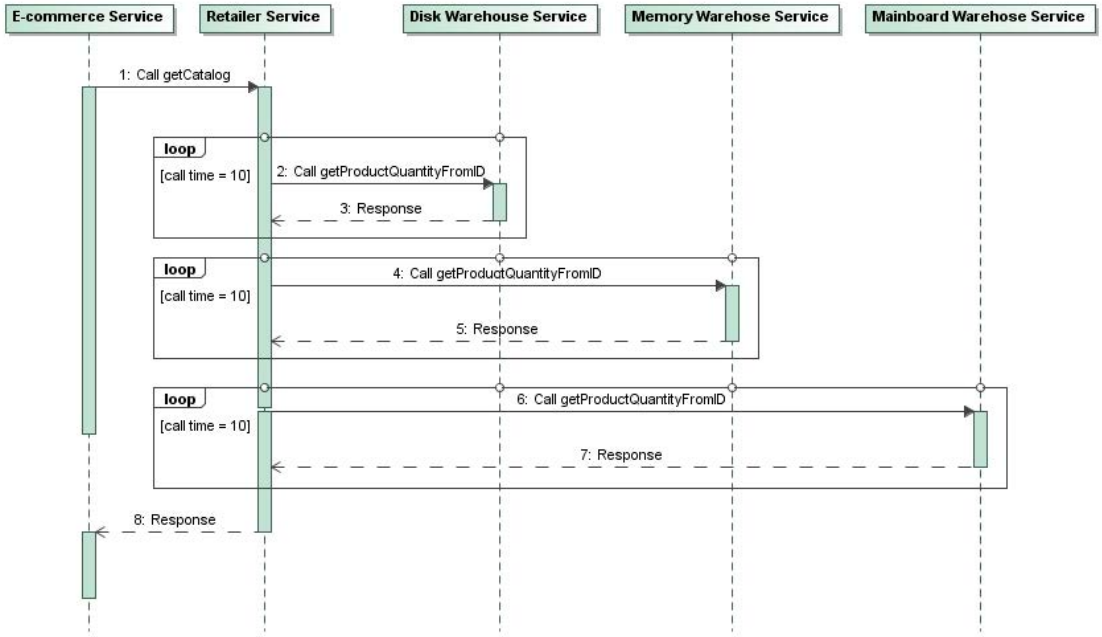
- เซอร์วิซอีคอมเมิร์ซ (E-Commerce Service) ทำหน้าที่เป็นผู้สั่งซื้อสินค้า ประกอบด้วยการทำงาน 2 ขั้นตอน ดังนี้
  - เรียกดูรายการสินค้าทั้งหมดจากเซอร์วิซผู้ขายปลีกผ่านทางโอเปอเรชัน getCatalog ของเซอร์วิซผู้ค้าปลีก รายการสั่งซื้อจะประกอบด้วยสินค้า 3 ประเภท คือ ดิสก์หน่วยความจำ และเมนบอร์ด แต่ละประเภทจะประกอบด้วยสินค้า 10 รุ่น รวมสินค้าทุกประเภทมี 30 รุ่น
  - สั่งซื้อสินค้าผ่านทางโอเปอเรชัน submitOrder ของเซอร์วิซผู้ค้าปลีก โดยจะทำการสุ่มเลือกสินค้า 10 รุ่นจากสินค้าทั้งหมด 30 รุ่น ซึ่งจำนวนการสั่งซื้อสำหรับสินค้าแต่ละรุ่นจะทำการสุ่มระหว่าง 0 ถึง สองเท่าของจำนวนชิ้นคงเหลือในคลังสินค้า
- เซอร์วิซผู้ค้าปลีก (Retailer Service) ทำหน้าที่รับการเรียกดูรายการสินค้าทั้งหมดและรับการสั่งซื้อสินค้าจากผู้สั่งซื้อสินค้า โดยมีรายละเอียดโอเปอเรชัน ดังนี้
  - โอเปอเรชัน getCatalog ทำการอ่านข้อมูลสินค้าทั้งหมดในฐานข้อมูล จากนั้นทำการติดต่อเซอร์วิซคลังข้อมูลเพื่อทำการสอบถามจำนวนสินค้าที่มีอยู่ในคลังสินค้าตามประเภทของสินค้า โดยการสอบถามจะทำผ่านโอเปอเรชัน getProductQuantityFromID ซึ่งทำการติดต่อเป็นจำนวนครั้งตามจำนวนรุ่นของสินค้าทั้งหมด จากการพัฒนากรณีศึกษานี้จะติดต่อเซอร์วิซคลังสินค้าทุกประเภทเป็นจำนวน 30 ครั้ง

- โอเปอเรชัน submitOrder ทำหน้าที่รับการสั่งซื้อสินค้าจากผู้สั่งซื้อ และแยกประเภทของสินค้าเพื่อส่งต่อไปยังเซอร์วิซคลังสินค้า

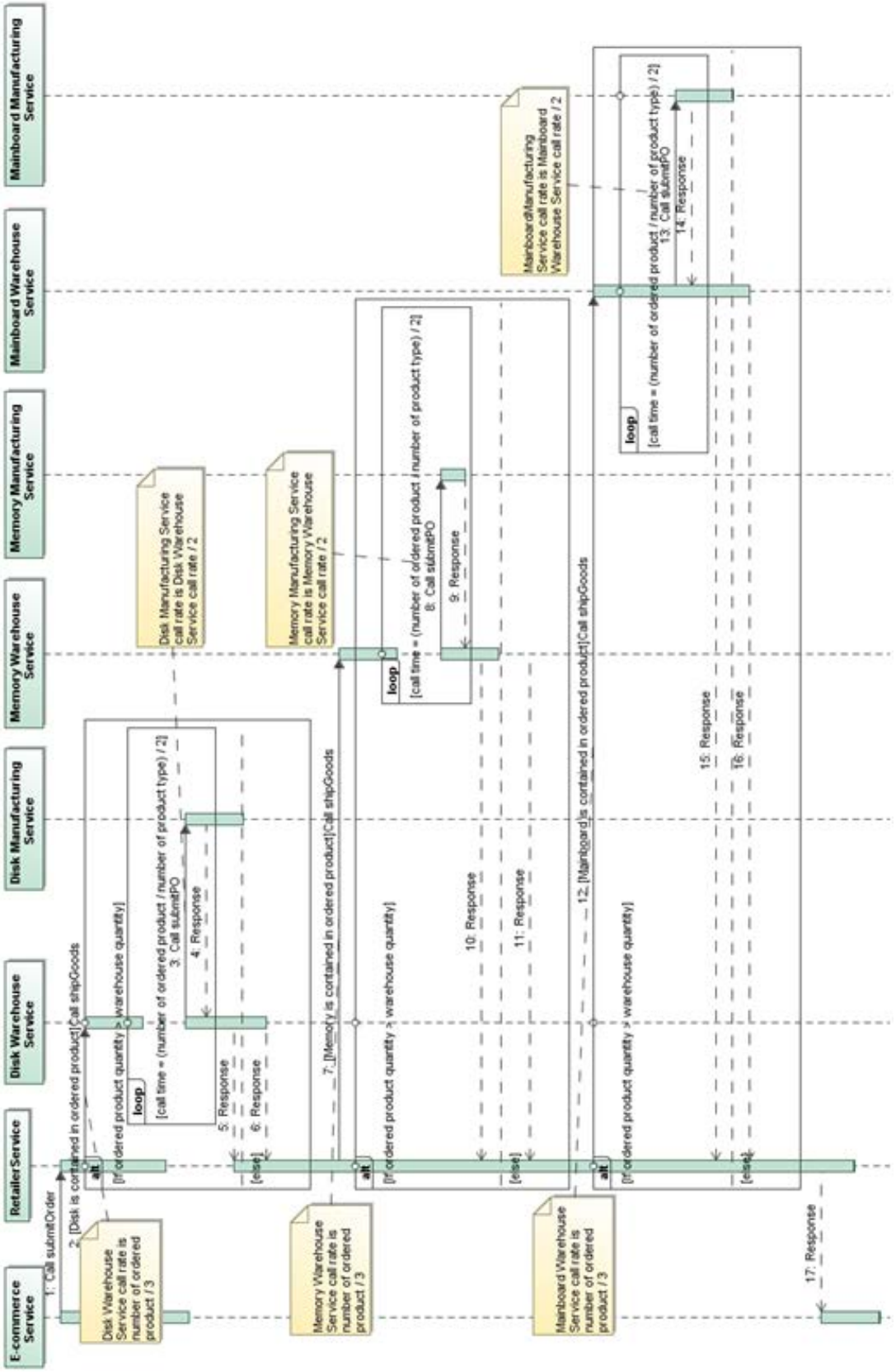
- เซอร์วิซคลังสินค้า (Warehouse Service) ทำหน้าที่ให้บริการสอบถามจำนวนสินค้าในคลังสินค้าและรับการสั่งซื้อสินค้าจากเซอร์วิซผู้ค้าปลีก โดยมีรายละเอียดโอเปอเรชัน ดังนี้
  - โอเปอเรชัน getProductQuantityFromID ให้บริการสอบถามจำนวนสินค้าที่มีอยู่ในคลังสินค้า จากรหัสสินค้า จากการพัฒนากรณีศึกษานี้โอเปอเรชันนี้จะถูกเรียกใช้งานรวมกันเป็นจำนวน 30 ครั้ง สำหรับสินค้าทั้งหมด 3 ประเภท
  - โอเปอเรชัน shipGoods ให้บริการรับการสั่งสินค้าจากเซอร์วิซผู้ค้าปลีก โดยทำการตรวจสอบจำนวนสินค้าที่สั่งซื้อ ถ้าจำนวนการสั่งซื้อมีน้อยกว่าหรือเท่ากับจำนวนสินค้าในคลังสินค้า จะทำการส่งสินค้าให้ผู้สั่งซื้อ และทำการเปลี่ยนสถานะของการสั่งซื้อสินค้ารุ่นนั้น ให้ว่ามีการสั่งซื้อสำเร็จ ในกรณีที่จำนวนการสั่งซื้อสินค้ามีมากกว่าจำนวนสินค้าในคลังสินค้า เซอร์วิซคลังสินค้าจะต้องติดต่อไปยังเซอร์วิซผู้ผลิตสินค้า ผ่านทางโอเปอเรชัน submitPO เพื่อทำการสั่งการผลิตสินค้า โดยที่ในการสั่งสินค้าแต่ละรุ่นจะสั่งครั้งละ 144 ชิ้น และทำการเปลี่ยนสถานะของการสั่งซื้อสินค้ารุ่นนั้น ให้ว่ามีการสั่งซื้อไม่สำเร็จ จากการพัฒนากรณีศึกษานี้อัตราการเรียกใช้โอเปอเรชัน shipGoods โดยโอเปอเรชัน submitOrder ของเซอร์วิซผู้ค้าปลีกจะเป็นหนึ่งในสามของจำนวนรุ่นของสินค้าที่สั่งซื้อทั้งหมดจำนวน 10 รุ่น ดังนั้นอัตราการเรียกใช้งานโอเปอเรชัน shipGoods ของคลังสินค้าแต่ละประเภทเท่ากับ 10/3

- เซอร์วิซผู้ผลิตสินค้า (Manufacturing Service) ทำหน้าที่รับคำสั่งการผลิตสินค้าจากเซอร์วิซคลังสินค้า โดยมีรายละเอียดโอเปอเรชัน ดังนี้
  - โอเปอเรชัน submitPO เมื่อได้รับคำสั่งการผลิตสินค้า จะทำการผลิตสินค้าหรือทำการแสดงผลข้อความออกทางหน้าจอ ในกรณีไม่มีข้อผิดพลาดเกิดขึ้นจะทำการส่งสถานะกลับไปยังผู้ร้องขอเพื่อบอกว่าการผลิตสินค้าสำเร็จ แต่ในกรณีที่มีข้อผิดพลาดเกิดขึ้นหรือจำนวนการสั่งผลิตสินค้าเท่ากับ 0 จะทำการส่งสถานะกลับไปผู้ร้องขอว่าการผลิตสินค้าไม่สำเร็จ จากการพัฒนากรณีศึกษานี้อัตราการเรียกใช้โอเปอเรชัน submitPO โดยโอเปอเรชัน shipGoods ของเซอร์วิซคลังสินค้าจะเป็นครึ่งหนึ่งของการเรียกโอเปอเรชัน shipGoods ไปยังคลังสินค้าแต่ละประเภท นั่นคือโอเปอเรชัน submitPO ของผู้ผลิตสินค้าแต่ละประเภทจะมีอัตราการเรียกใช้เท่ากับ (10/3)/2

จากรายละเอียดของโอเปอเรชันสำหรับแต่ละเซอร์วิซข้างต้น สามารถนำมาแสดงแผนภาพซีเควนซ์สำหรับการเรียกดูสินค้าทั้งหมดและการสั่งซื้อสินค้าได้ดังภาพที่ 5.4 และภาพที่ 5.5



ภาพที่ 5.4 แผนภาพซีเควนซ์สำหรับการเรียกดูสินค้าทั้งหมด



ภาพที่ 5.5 แผนภาพซีเควนซ์สำหรับการสั่งซื้อสินค้า

การประเมินผลด้านความเชื่อถือได้จะทำการทดสอบตามการทำงานทั้งหมดของระบบตามที่ได้กล่าวไปแล้ว โดยที่จะทำการประเมินผลโดยการคำนวณค่าทางทฤษฎีและทดสอบการทำงานของระบบจริง สำหรับระบบกรณีศึกษาทั้งก่อนและหลังการประยุกต์ใช้แบบรูปการทนต่อความผิดพร่อง โดยที่การประเมินค่าแต่ละรูปแบบจะมีการกำหนดอัตราการทำงานสำเร็จสำหรับแต่ละโอเปอเรชัน (Success rate of each service) ในอัตราต่างๆเป็นจำนวน 10 ค่าที่แตกต่างกัน ได้แก่ {0.9, 0.91, 0.92,..., 0.99} ซึ่งรายละเอียดเป็นดังนี้

- การประเมินค่าอัตราการทำงานสำเร็จทางทฤษฎีของระบบกรณีศึกษาก่อนการประยุกต์ใช้แบบรูปการทนต่อความผิดพร่อง โดยแสดงตัวอย่างการคำนวณอัตราการทำงานสำเร็จเมื่อมีอัตราการทำงานสำเร็จสำหรับแต่ละโอเปอเรชันเป็น 0.9 การคำนวณเป็นดังนี้
  - ขั้นตอนการเรียกดูรายการสินค้าทั้งหมด: อัตราการทำงานสำเร็จสำหรับโอเปอเรชัน getCatalog x (อัตราการทำงานสำเร็จสำหรับโอเปอเรชัน getProductQuantityFromID)$^{\text{จำนวนการเรียกใช้งานโอเปอเรชันของสินค้าแต่ละประเภท x จำนวนประเภทสินค้า}}$ = 0.9 x $(0.9)^{10x3}$ = 0.0382
  - ขั้นตอนการสั่งซื้อสินค้า: อัตราการทำงานสำเร็จสำหรับโอเปอเรชัน submitOrder x (อัตราการทำงานสำเร็จสำหรับโอเปอเรชัน shipGoods)$^{\text{อัตราการเรียกใช้งานโอเปอเรชันของสินค้าแต่ละประเภท x จำนวนประเภทสินค้า}}$ x (อัตราการทำงานสำเร็จสำหรับโอเปอเรชัน submitPO)$^{\text{อัตราการเรียกใช้งานโอเปอเรชันของสินค้าแต่ละประเภท x จำนวนประเภทสินค้า}}$ = 0.9 x $(0.9)^{(10/3)x3}$ x $(0.9)^{((10/3)/2)x3}$ = 0.1853

  ดังนั้นอัตราการทำงานสำเร็จตามทฤษฎีของระบบกรณีศึกษาก่อนการประยุกต์ใช้แบบรูปการทนต่อความผิดพร่องเท่ากับ 0.0382 x 0.1853 = 0.7078%

- การประเมินค่าอัตราการทำงานสำเร็จทางทฤษฎีของระบบกรณีศึกษาหลังการประยุกต์ใช้แบบรูปการทนต่อความผิดพร่อง โดยแสดงตัวอย่างการคำนวณอัตราการทำงานสำเร็จเมื่อมีอัตราการทำงานสำเร็จสำหรับแต่ละโอเปอเรชันเป็น 0.9 การคำนวณจะต้องแยกตามแต่ละเซอร์วิซเนื่องจากแต่ละเซอร์วิซมีการประยุกต์ใช้แบบรูปที่แตกต่างกัน โดยมีรายละเอียดการคำนวณเป็นดังนี้
  - เซอร์วิซผู้ขายปลีกใช้แบบรูป Limit Retries และแบบรูป Recovery Blocks: 1 – (อัตราการทำงานล้มเหลวสำหรับแต่ละโอเปอเรชันของหน่วยย่อยที่หนึ่ง$^{\text{จำนวนครั้งของการทำซ้ำ}}$ x อัตราการทำงานล้มเหลวสำหรับแต่ละโอเปอเรชันของหน่วยย่อยที่สอง$^{\text{จำนวนครั้งของการทำซ้ำ}}$) = 1 - ($0.1^3$ x 0.1^3) = 0.999999

- เซอร์วิซคลังสินค้าสำหรับสินค้าประเภทดิสก์ ซึ่งมีการประยุกต์ใช้แบบรูป Active Replication โดยมีหน่วยสำรอง 3 หน่วยและมีการกำหนดให้เลือกคำตอบที่ตอบมาคำตอบแรกเป็นคำตอบที่เหมาะสม โดยที่การคำนวณค่าอัตราการทำงานสำเร็จสำหรับเซอร์วิซคลังสินค้าสำหรับสินค้าประเภทดิสก์ แสดงดังตารางที่ 5.6

ตารางที่ 5.6 การคำนวณค่าอัตราการทำงานสำเร็จสำหรับเซอร์วิซคลังสินค้าสำหรับสินค้าประเภทดิสก์

ความน่าจะเป็นของการตอบกลับมาของเซอร์วิซ	**อัตราการทำงานสำเร็จที่นำมาคิด**
ตอบกลับ 1 ตัว	$3(0.9 * 0.1^2) = 0.027$
ตอบกลับ 2 ตัว	$3(0.9^2 * 0.1) = 0.243$
ตอบกลับ 3 ตัว	$0.9^3 = 0.729$

จากตารางที่ 5.6 อัตราการทำงานสำเร็จสำหรับเซอร์วิซคลังสินค้าสำหรับสินค้าประเภทดิสก์เท่ากับ 0.027 + 0.243 + 0.729 = 0.999

- เซอร์วิซคลังสินค้าสำหรับสินค้าประเภทหน่วยความจำและเมนบอร์ด และเซอร์วิซผู้ผลิตสินค้าสำหรับสินค้าประเภทดิสก์ มีการประยุกต์ใช้แบบรูปการทนต่อความผิดพร่องตามแบบรูป Restart เมื่อมีข้อผิดพลาดเกิดขึ้นในระบบ และมีการจำลองให้การทำตามแบบรูป Restart สามารถทำได้ประสบความสำเร็จทุกครั้ง เพราะฉะนั้นระบบจะมีการทำงานจนสำเร็จจึงจะตอบกลับมายังผู้ร้องขอ ดังนั้นอัตราการทำงานสำเร็จของเซอร์วิซทั้ง 3 เซอร์วิซ จะมีค่าเท่ากับ 1 เสมอ
- เซอร์วิซผู้ผลิตสินค้าสำหรับสินค้าประเภทหน่วยความจำใช้แบบรูป Recovery Blocks: 1 – (อัตราการทำงานล้มเหลวสำหรับแต่ละโอเปอเรชันของหน่วยย่อยที่หนึ่ง x อัตราการทำงานล้มเหลวสำหรับแต่ละโอเปอเรชันของหน่วยย่อยที่สอง) = 1 – (0.1 x 0.1) = 0.99
- เซอร์วิซผู้ผลิตสินค้าสำหรับสินค้าประเภทเมนบอร์ดใช้แบบรูป Limit Retries: 1 – อัตราการทำงานล้มเหลวสำหรับแต่ละโอเปอเรชันของหน่วยย่อย$^{\text{จำนวนครั้งของการทำซ้ำ}}$ = $1 - 0.1^5 = 0.99999$

จากการคำนวณอัตราการทำงานสำเร็จของแต่ละเซอร์วิซ เมื่อนำมาคำนวณอัตราการทำงานสำเร็จของขั้นตอนต่างๆ จะได้ผลดังนี้

- อัตราการทำงานสำเร็จของขั้นตอนการเรียกดูรายการสินค้าทั้งหมด: อัตราการทำสำเร็จสำหรับโอเปอเรชัน getCatalog$^{\text{จำนวนการเรียกใช้งานโอเปอเรชัน}}$ x อัตราการทำสำเร็จสำหรับโอเปอเรชัน getProductQuantityFromID สำหรับเซอร์วิซคลังสินค้าประเภทดิสก์$^{\text{จำนวนการเรียกใช้งานโอเปอเรชัน}}$ x อัตราการทำสำเร็จสำหรับโอเปอเรชัน getProductQuantityFromID สำหรับเซอร์วิซคลังสินค้าประเภทหน่วยความจำ$^{\text{จำนวนการเรียกใช้งานโอเปอเรชัน}}$ x อัตราการทำสำเร็จสำหรับโอเปอเรชัน getProductQuantityFromID สำหรับเซอร์วิซคลังสินค้าประเภทเมนบอร์ด$^{\text{จำนวนการเรียกใช้งานโอเปอเรชัน}}$ จะได้เท่ากับ $1 \times 0.999^{10} \times 1^{10} \times 1^{10} = 0.99$
- อัตราการทำงานสำเร็จของขั้นตอนการสั่งซื้อสินค้า: อัตราการทำสำเร็จสำหรับโอเปอเรชัน getProductQuantityFromID x อัตราการทำสำเร็จสำหรับโอเปอเรชัน shipGoods สำหรับเซอร์วิซคลังสินค้าประเภทดิสก์$^{\text{จำนวนการเรียกใช้งานโอเปอเรชัน}}$ x อัตราการทำสำเร็จสำหรับโอเปอเรชัน shipGoods สำหรับเซอร์วิซคลังสินค้าประเภทหน่วยความจำ$^{\text{จำนวนการเรียกใช้งานโอเปอเรชัน}}$ x อัตราการทำสำเร็จสำหรับโอเปอเรชัน shipGoods สำหรับเซอร์วิซคลังสินค้าประเภทเมนบอร์ด$^{\text{จำนวนการเรียกใช้งานโอเปอเรชัน}}$ x อัตราการทำสำเร็จสำหรับโอเปอเรชัน submitPO สำหรับเซอร์วิซผู้ผลิตสินค้าประเภทดิสก์$^{\text{จำนวนการเรียกใช้งานโอเปอเรชัน}}$ x อัตราการทำสำเร็จสำหรับโอเปอเรชัน submitPO สำหรับเซอร์วิซผู้ผลิตสินค้าประเภทหน่วยความจำ$^{\text{จำนวนการเรียกใช้งานโอเปอเรชัน}}$ x อัตราการทำสำเร็จสำหรับโอเปอเรชัน submitPO สำหรับเซอร์วิซผู้ผลิตสินค้าประเภทเมนบอร์ด$^{\text{จำนวนการเรียกใช้งานโอเปอเรชัน}}$ จะได้เท่ากับ $1 \times 0.999^{10/3} \times 1^{10/3} \times 1^{10/3} \times 1^{(10/3)/2} \times 0.99^{(10/3)/2} \times 1^{(10/3)/2} = 0.9801$

ดังนั้นอัตราการทำงานสำเร็จของระบบกรณีศึกษาที่มีการประยุกต์ใช้แบบรูปการทนต่อความผิดพร่องตามทฤษฎีเท่ากับ $0.99 \times 0.9801 = 97.03\%$

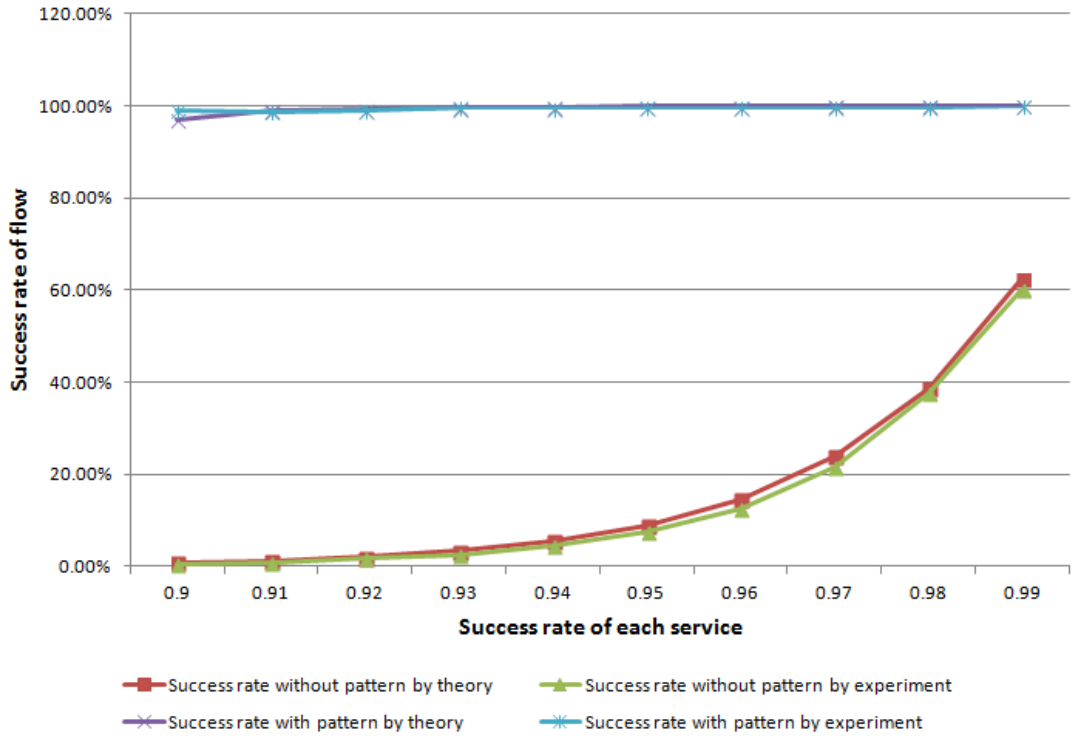
- การประเมินค่าอัตราการทำงานสำเร็จของระบบกรณีศึกษาก่อนและหลังการประยุกต์ใช้แบบรูปการทนต่อความผิดพร่อง โดยวัดผลจากการทำงานของระบบจริงจะทำการทดสอบเป็นจำนวน 1000 ครั้ง

ผลการประเมินค่าอัตราการทำงานสำเร็จของระบบกรณีศึกษาก่อนและหลังการประยุกต์ใช้แบบรูปการทนต่อความผิดพร่องซึ่งได้จากการคำนวณค่าจากทฤษฎีและการทดสอบการทำงานของระบบจริง แสดงดังตารางที่ 5.7

ตารางที่ 5.7 ผลการประเมินค่าอัตราการทำงานสำเร็จของกรณีต่างๆ

อัตราการทำสำเร็จสำหรับแต่ละโอเปอเรชัน	อัตราการทำสำเร็จตามทฤษฎีก่อนใช้แบบรูป	อัตราการทำสำเร็จจากการทดลองก่อนใช้แบบรูป	อัตราการทำสำเร็จตามทฤษฎีหลังใช้แบบรูป	อัตราการทำสำเร็จจากการทดลองหลังใช้แบบรูป
0.9	0.71%	0.50%	97.03%	99%
0.91	1.19%	0.80%	98.95%	98.80%
0.92	1.99%	1.80%	99.22%	99.10%
0.93	3.30%	2.40%	99.55%	99.60%
0.94	5.46%	4.40%	99.70%	99.50%
0.95	8.97%	7.40%	99.85%	99.70%
0.96	14.68%	12.50%	99.85%	99.70%
0.97	23.90%	21.70%	100%	99.80%
0.98	38.69%	37.60%	100%	99.80%
0.99	62.36%	60%	100%	99.90%

จากตารางที่ 5.7 นำข้อมูลมาแสดงเป็นกราฟได้ดังภาพที่ 5.6



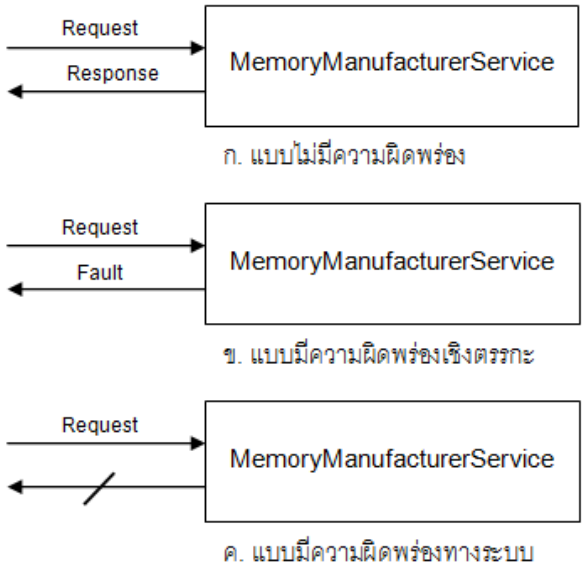
ภาพที่ 5.6 แผนภาพเปรียบเทียบค่าอัตราความสำเร็จจากการคำนวณทางทฤษฎีและจากการทดสอบการทำงานของระบบจริงสำหรับระบบกรณีศึกษาก่อนและหลังการประยุกต์ใช้แบบรูปการทนต่อความผิดพร่อง

จากภาพที่ 5.6 จะเห็นได้ว่าค่าอัตราความสำเร็จของระบบกรณีศึกษาที่ได้จากการทดลองมีค่าใกล้เคียงกับค่าที่คำนวณได้ตามทฤษฎี จึงนับว่าผลการทดลองสามารถยอมรับได้ อีกทั้งจะเห็นได้ว่าอัตราความสำเร็จของระบบกรณีศึกษาก่อนการประยุกต์ใช้แบบรูปการทนต่อความผิดพร่องจะมีค่าเพิ่มขึ้นเรื่อยๆอย่างต่อเนื่อง เมื่อมีการเปลี่ยนแปลงค่าอัตราความสำเร็จของแต่ละโอเปอเรชันในเซอร์วิซ เนื่องจากการเพิ่มขึ้นของอัตราความสำเร็จของแต่ละโอเปอเรชันจะมีผลต่อเซอร์วิซต่างๆในระบบเท่าๆกัน ทำให้อัตราความสำเร็จของระบบทั้งหมดค่อยๆเพิ่มขึ้นเรื่อยๆ สำหรับระบบกรณีศึกษาหลังการประยุกต์ใช้แบบรูปการทนต่อความผิดพร่องจะมีอัตราความสำเร็จที่ได้จากการคำนวณค่าทางทฤษฎีและอัตราความสำเร็จที่ได้จากการทดสอบการทำงานของระบบจริงที่มีค่าใกล้เคียงกันเช่นกัน แต่อัตราความสำเร็จของแต่ละโอเปอเรชันในเซอร์วิซ จะมีผลกระทบต่ออัตราความสำเร็จของทั้งระบบเพียงเล็กน้อยเท่านั้น เนื่องจากอัตราความสำเร็จของแต่ละโอเปอเรชันในเซอร์วิซมีผลกระทบแต่เพียงบางเซอร์วิซในระบบเท่านั้น เช่น เซอร์วิซผู้ค้าปลีก แต่สำหรับเซอร์วิซคลังสินค้าสำหรับสินค้าประเภทหน่วยความจำและเมนบอร์ด และเซอร์วิซผู้ผลิตสินค้าสำหรับสินค้าประเภทดิสก์ จะไม่ได้รับผลกระทบจากการเปลี่ยนแปลงของอัตราความสำเร็จของแต่ละโอเปอเรชัน เพราะเมื่อเกิดข้อผิดพลาดทั้ง 3 เซอร์วิซนี้จะถูกเริ่มทำงานใหม่จนสำเร็จ อีกทั้งแบบรูปต่างๆจะช่วยให้การทำงานของแต่ละเซอร์วิซมีโอกาสสำเร็จมากขึ้นกว่าเมื่อไม่ใช้แบบรูปอยู่แล้ว

5.3.2 การประเมินผลด้านสมรรถนะ

การนำแบบรูปการทนต่อความผิดพร่องมาประยุกต์ใช้กับระบบกรณีศึกษา ทำให้การออกแบบระบบมีความซับซ้อนมากขึ้นและย่อมส่งผลกระทบต่อความเร็วในการประมวลผล ซึ่งวัดได้จากเวลาที่ใช้ในการตอบกลับ (Response time) เมื่อมีการเรียกใช้งานเซอร์วิซ เพื่อเป็นการศึกษาผลกระทบที่เกิดขึ้นจึงทำการทดสอบโดยการเปรียบเทียบเวลาที่ใช้ในการเรียกใช้งานเซอร์วิซก่อนหรือหลังการประยุกต์ใช้แบบรูปการทนต่อความผิดพร่อง

การประเมินผลด้านสมรรถนะจะเลือกใช้เซอร์วิซผู้ผลิตสินค้าสำหรับสินค้าประเภทหน่วยความจำมาเป็นตัวแทนในประเมินค่าสมรรถนะของการทำงาน ซึ่งในการทดสอบจะทำการเรียกใช้งานเซอร์วิซนี้โดยตรงและจำลองสภาพการเกิดข้อผิดพร่องจากการเรียกใช้งาน ซึ่งแบ่งออกเป็น ความผิดพร่องเชิงตรรกะ และความผิดพร่องทางระบบ ดังนั้นจึงทำการจำลองเซอร์วิซผู้ผลิตสินค้าสำหรับสินค้าประเภทหน่วยความจำแตกต่างกัน 3 ชุด สำหรับการทดสอบมีลักษณะดังภาพที่ 5.7



ภาพที่ 5.7 การจำลองเซอร์วิซผู้ผลิตสินค้าสำหรับสินค้าประเภทหน่วยความจำในการประเมินผลด้านสมรรถนะ

รายละเอียดของแต่ละชุดเป็นดังต่อไปนี้

- แบบไม่มีความผิดพร่อง คือเซอร์วิซจะรับคำร้องขอและตอบกลับมาตามปกติ (ตอบกลับมาเป็นชนิด boolean)

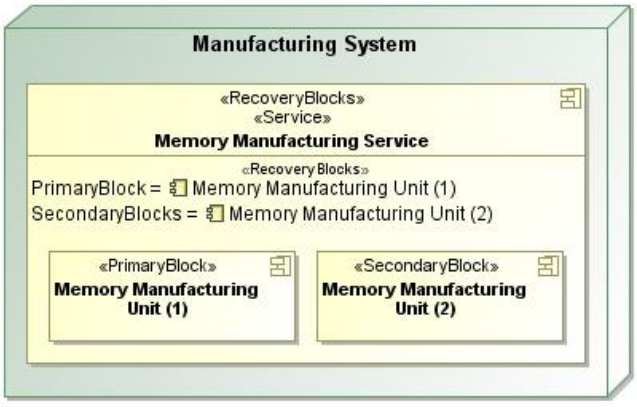
- แบบมีความผิดพร่องเชิงตรรกะ คือเซอร์วิซรับคำร้องขอและแจ้งกลับมาว่ามีความผิดพร่องทุกครั้งที่เรียกใช้งาน
- แบบมีความผิดพร่องทางระบบ ในกรณีก่อนการประยุกต์ใช้แบบรูปการทนต่อความผิดพร่อง จะเรียกใช้งานเซอร์วิซตัวเดียวกับแบบไม่มีความผิดพร่องแต่จะมีการถอดเซอร์วิซนั้นออกจากระบบ แต่สำหรับกรณีหลังการประยุกต์ใช้แบบรูปการทนต่อความผิดพร่อง เซอร์วิซจะไม่ถูกถอดออกจากระบบแต่จะถอดหน่วยย่อยที่ทำหน้าที่เป็นหน่วยสำรองออกจากระบบแทน

การประเมินผลด้านสมรรถนะนี้ จะทำการเรียกใช้งานเซอร์วิซผู้ผลิตสินค้าสำหรับสินค้าประเภทหน่วยความจำทั้ง 3 ชุด โดยมีการออกแบบด้วยแบบรูปการทนต่อความผิดพร่องที่แตกต่างกันทั้งหมด 8 รูปแบบ คือ

1) การออกแบบที่ไม่ใช้แบบรูป มีการเรียกใช้เซอร์วิซโดยตรง ดังภาพที่ 5.8

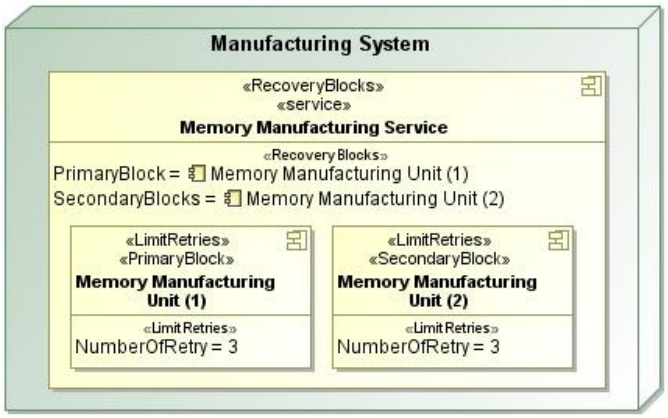
ภาพที่ 5.8 การออกแบบที่ไม่ใช้แบบรูป

2) การออกแบบที่มีการประยุกต์ใช้แบบรูป Recovery Blocks โดยมีหน่วยสำรองเพื่อทำการรองรับข้อผิดพลาด 2 หน่วย ถ้าหน่วยย่อยที่หนึ่งทำงานไม่สำเร็จ จะส่งไปทำงานยังหน่วยย่อยที่สอง ดังภาพที่ 5.9



ภาพที่ 5.9 การออกแบบที่มีการประยุกต์ใช้แบบรูป Recovery Blocks

3) การออกแบบที่มีการประยุกต์ใช้แบบรูป Recovery Blocks และแบบรูป Limit Retries โดยมีหน่วยสำรองเพื่อทำการรองรับข้อผิดพลาด 2 หน่วย และแต่ละหน่วยย่อยมีการทำซ้ำตามจำนวนครั้งที่กำหนดไว้ คือ 3 โดยก่อนการทำซ้ำแต่ละรอบจะมีเวลาหน่วงเท่ากับ 200 มิลลิวินาที ดังภาพที่ 5.10

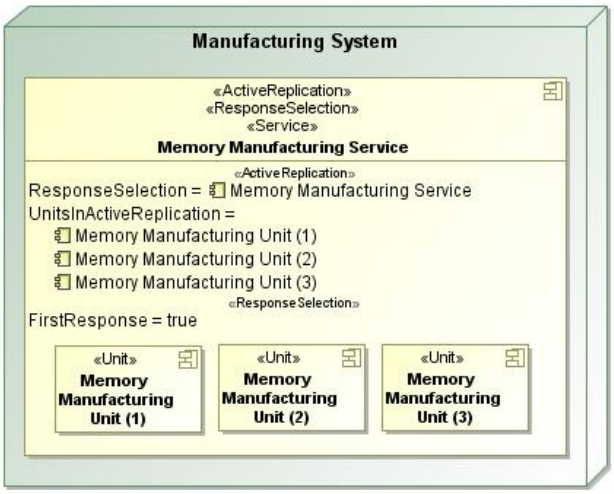


ภาพที่ 5.10 การออกแบบที่มีการประยุกต์ใช้แบบรูป Recovery Blocks และแบบรูป Limit Retries

4) การออกแบบที่มีการประยุกต์ใช้แบบรูป Limit Retries โดยมีหน่วยย่อยที่จะทำงานซ้ำตามจำนวนครั้งที่กำหนดไว้ เมื่อมีข้อผิดพลาดเกิดขึ้น โดยก่อนการทำซ้ำแต่ละรอบจะมีเวลาหน่วงเท่ากับ 200 มิลลิวินาทีเช่นกัน ดังภาพที่ 5.11

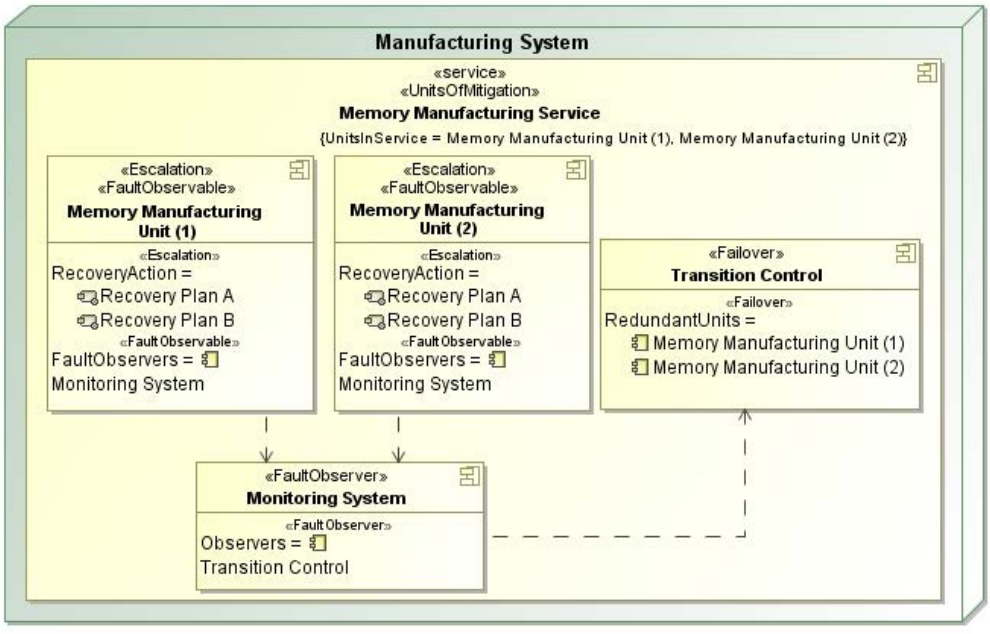
ภาพที่ 5.11 การออกแบบที่มีการประยุกต์ใช้แบบรูป Limit Retries

5) การออกแบบที่มีการประยุกต์ใช้แบบรูป Voting โดยมีหน่วยย่อยทั้งหมด 3 หน่วยย่อย และมีการเลือกคำตอบที่เหมาะสมตามแบบรูป Voting ซึ่งมีการกำหนดว่าคำตอบที่ตอบมาคำตอบแรกเป็นคำตอบที่เหมาะสม ดังภาพที่ 5.12



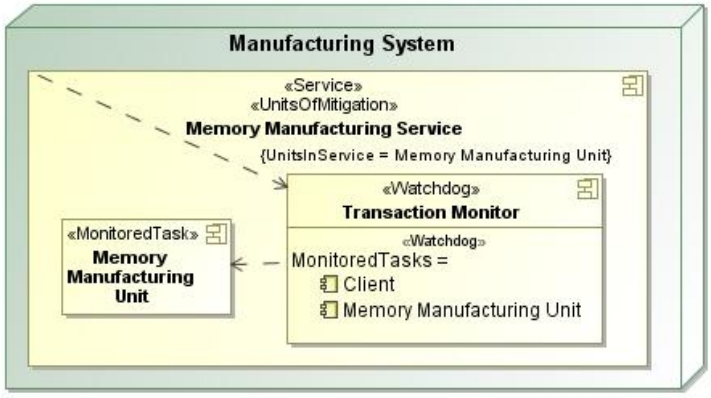
ภาพที่ 5.12 การออกแบบที่มีการประยุกต์ใช้แบบรูป Voting

6) การออกแบบที่มีการประยุกต์ใช้แบบรูป Fault Observer และแบบรูป Failover โดยที่มีหน่วยย่อยทั้งหมด 2 หน่วย และมีหน่วยสังเกตทำหน้าที่เฝ้ามองหน่วยย่อยทั้ง 2 หน่วย และเมื่อมีความผิดพร่องเกิดขึ้นจะทำการรายงานไปยังผู้สนใจ ซึ่งผู้สนใจในการออกแบบนี้จะทำหน้าที่จัดการกับความผิดพร่องที่เกิดขึ้นในระบบ ดังภาพที่ 5.13



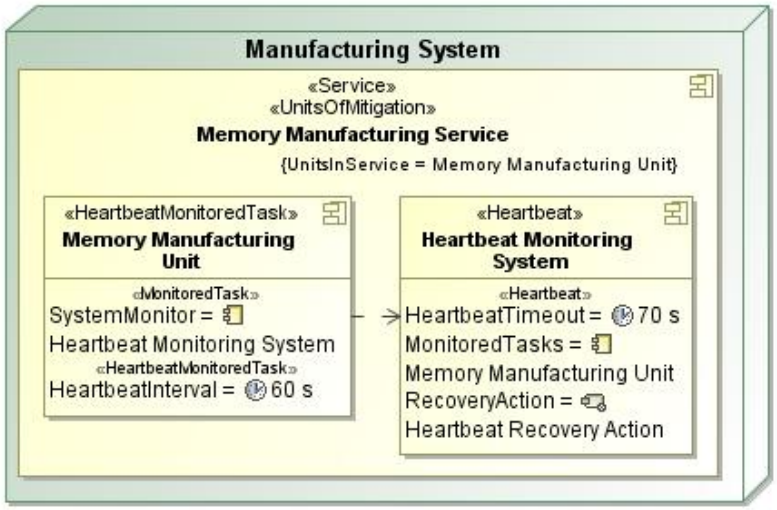
ภาพที่ 5.13 การออกแบบที่มีการประยุกต์ใช้แบบรูป Fault Observer และแบบรูป Failover

7) การออกแบบที่มีการประยุกต์ใช้แบบรูป Watchdog ประกอบด้วยหน่วยที่ทำหน้าที่เป็นหน่วยเฝ้าสังเกตการทำงานระหว่างหน่วยย่อยและผู้ร้องขอ ดังภาพที่ 5.14



ภาพที่ 5.14 การออกแบบที่มีการประยุกต์ใช้แบบรูป Watchdog

8) การออกแบบที่มีการประยุกต์ใช้แบบรูป Heartbeat ประกอบด้วยหน่วยที่ทำหน้าที่สังเกตการส่งสัญญาณชีพของหน่วยย่อย ดังภาพที่ 5.15



ภาพที่ 5.15 การออกแบบที่มีการประยุกต์ใช้แบบรูป Heartbeat

จากนั้นทำการทดสอบโดยการเรียกเซอร์วิซเป็นจำนวน 1000 ครั้งแล้วหาค่าเฉลี่ยเวลาที่ใช้ตอบกลับ จากชุดทดสอบทั้งหมด ผลที่ได้เป็นดังตารางที่ 5.8 และ ตารางที่ 5.9

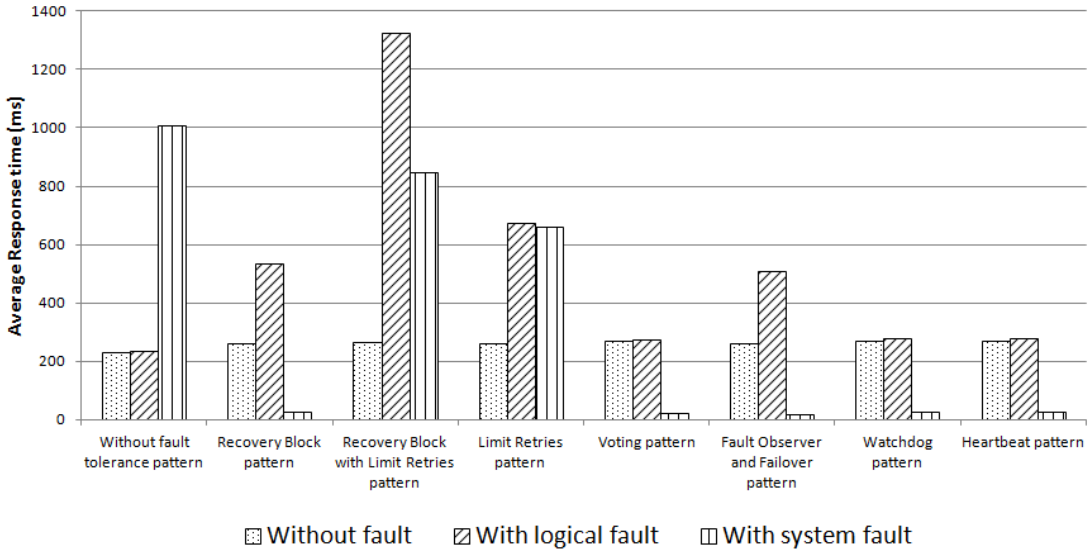
ตารางที่ 5.8 ตารางค่าเฉลี่ยของเวลาที่ใช้ตอบกลับสำหรับแต่ละแบบรูป (1)

ชนิดความผิดพร่อง	ค่าเฉลี่ยของเวลาที่ใช้ตอบกลับสำหรับแต่ละแบบรูป (มิลลิวินาที)			
	แบบปกติที่ไม่ใช้แบบรูป	แบบรูป Recovery Blocks	แบบรูป Recovery Blocks และ Limit Retries	แบบรูป Limit Retries
ไม่มีความผิดพร่อง	228.71	261.74	265.52	262.47
ความผิดพร่องเชิงตรรกะ	235.01	534.66	1323.91	673.40
ความผิดพร่องทางระบบ	1006.05	24.36	848.40	660.90

ตารางที่ 5.9 ตารางค่าเฉลี่ยของเวลาที่ใช้ตอบกลับสำหรับแต่ละแบบรูป (2)

ชนิดความผิดพร่อง	ค่าเฉลี่ยของเวลาที่ใช้ตอบกลับสำหรับแต่ละแบบรูป (มิลลิวินาที)			
	แบบรูป Voting	แบบรูป Fault Observer และ Failover	แบบรูป Watchdog	แบบรูป Heartbeat
ไม่มีความผิดพร่อง	268.05	261.74	270.14	270.05
ความผิดพร่องเชิงตรรกะ	275.54	506.92	278.50	275.70
ความผิดพร่องทางระบบ	21.37	17.02	25.17	25.06

จากตารางที่ 5.8 และ ตารางที่ 5.9 นำข้อมูลมาแสดงเพื่อเปรียบเทียบค่าเฉลี่ยของเวลาในการตอบกลับสำหรับแต่ละแบบรูป ดังภาพที่ 5.16



ภาพที่ 5.16 ค่าเฉลี่ยของเวลาในการตอบกลับสำหรับแต่ละแบบรูป

จากภาพที่ 5.16 ค่าเฉลี่ยของเวลาในการตอบกลับสำหรับแต่ละแบบรูป จะเห็นได้ว่าเวลาที่ใช้ในการเรียกใช้เซอร์วิซที่ไม่มีความผิดพร่องจะใกล้เคียงกันทั้งแบบไม่มีการประยุกต์ใช้แบบรูปและแบบที่มีการใช้แบบรูป แต่เซอร์วิซที่มีการประยุกต์ใช้แบบรูปการทนต่อความผิดพร่องจะใช้เวลามากกว่าเล็กน้อย เนื่องจากในขั้นตอนการทำงานจะต้องติดต่อไปยังเซอร์วิซแล้วเซอร์วิซจะมีการส่งต่องานไปยังหน่วยย่อยอีกครั้ง แสดงว่าการออกแบบโดยประยุกต์ใช้แบบรูปการทนต่อความผิดพร่องมีผลต่อเวลาที่ใช้เพียงเล็กน้อยเท่านั้นในสภาวะปกติ

ส่วนในกรณีที่มีความผิดพร่องเชิงตรรกะเกิดขึ้น เวลาที่ใช้จะแตกต่างกันไปขึ้นอยู่กับแบบรูปที่นำมาปรับใช้ การประยุกต์ใช้แบบรูป RecoveryBlocks และแบบรูป Limit Retries พร้อมกันจะใช้เวลาในการตอบกลับมากที่สุดเนื่องจากมีการเรียกใช้หน่วยย่อย 2 หน่วยต่อกันตามลำดับ และแต่ละหน่วยยังมีการทำซ้ำเป็นจำนวน 3 ครั้ง ซึ่งในแต่ละการเรียกซ้ำจะมีการหน่วงเป็นเวลา 200 มิลลิวินาที สำหรับแบบรูป Voting แบบรูป Watchdog และแบบรูป Heartbeat จะใช้เวลาไม่แตกต่างจากการไม่มีการประยุกต์ใช้แบบรูป เนื่องจากมีการเรียกใช้งานหน่วยย่อยเพียงครั้งเดียว

ในกรณีที่เกิดความผิดพร่องของระบบ สำหรับเซอร์วิซที่ไม่มีการประยุกต์ใช้แบบรูปจะใช้เวลามาก เนื่องจากข้อจำกัดของไลบรารีที่ใช้ในการพัฒนาระบบ โดยที่ในการเรียกใช้งานเซอร์วิซ ถ้าไม่สามารถค้นหาเซอร์วิซเจอ จะรอจนหมดเวลาที่ไลบรารีกำหนด (Timeout) ก่อนจึงจะถือว่าไม่สามารถติดต่อเซอร์วิซได้ เพราะฉะนั้นจึงมีการใช้เวลามาก แต่สำหรับในกรณีที่มีการประยุกต์แบบรูปหน่วยย่อยจะเป็นส่วนที่ถูกถอดออกจากระบบไม่ใช่เซอร์วิซ ดังนั้นความผิดพร่องทางระบบจะเกิดขึ้นระหว่างเซอร์วิซกับหน่วยย่อย ซึ่งถือเป็นเซอร์วิซเช่นกัน ตามข้อจำกัดของไลบรารีมีกระบวนการทำงานที่แตกต่างกัน จึงไม่ต้องรอจนหมดเวลาที่ไลบรารีกำหนด และสำหรับแบบรูป Recovery

Blocks และแบบรูป Recovery Blocks รวมกับแบบรูป Limit Retries จะใช้เวลามากกว่าแบบรูปอื่นเนื่องจากมีการทำซ้ำ ซึ่งแต่ละการทำซ้ำจะมีการหน่วงเวลา 200 มิลลิวินาที ตัวอย่างเช่น การออกแบบที่มีการใช้แบบรูป Limit Retries ต้องทำการทำซ้ำจำนวน 3 รอบ เพราะฉะนั้นต้องใช้เวลาในการหน่วงเพิ่มอีก 600 มิลลิวินาที

บทที่ 6

บทสรุป

6.1 สรุปผลการวิจัย

งานวิจัยนี้ได้นำเสนอแนวทางในการนำแบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่องมาออกแบบเป็นยูเอ็มแอลโปรไฟล์สำหรับการทนต่อความผิดพร่องสำหรับระบบอิงบริการ แบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่องแบ่งออกได้ 5 กลุ่ม หรือทั้งหมด 63 แบบรูป ซึ่งงานวิจัยนี้จะเลือกพิจารณาแบบรูปใน 3 กลุ่มแรกเท่านั้น ผู้วิจัยได้ทำการคัดเลือกแบบรูปที่สามารถนำมาออกแบบเป็นยูเอ็มแอลโปรไฟล์ตามยูเอ็มแอลรุ่น 2.4 จำนวนทั้งหมด 22 แบบรูป ซึ่งเป็นแบบรูปในกลุ่มแบบรูปเชิงสถาปัตยกรรมจำนวน 6 แบบรูป แบบรูปในกลุ่มแบบรูปการตรวจหาข้อผิดพลาดจำนวน 5 แบบรูป และแบบรูปในกลุ่มแบบรูปการกู้ระบบจากข้อผิดพลาดจำนวน 11 แบบรูป เพื่อนำไปใช้ในการออกแบบระบบอิงบริการเพื่อให้มีความสามารถด้านการทนต่อความผิดพร่อง และสามารถนำไปใช้ในการออกแบบสำหรับทุกโดเมนของระบบอิงบริการ

ผู้วิจัยได้นำเสนอตัวอย่างการใช้งานสำหรับแต่ละแบบรูปและนำบางส่วนของยูเอ็มแอลโปรไฟล์ไปประยุกต์ใช้เข้ากับระบบกรณีศึกษาตัวอย่างสถาปัตยกรรมโปรแกรมประยุกต์สำหรับระบบจัดการห่วงโซ่อุปทาน จากนั้นทำการทดสอบความถูกต้องของการออกแบบโดยวิธีตามรอยความสามารถของระบบ และนำระบบกรณีศึกษาไปพัฒนาจริงเพื่อประเมินค่าคุณสมบัติทางการออกแบบและคุณลักษณะเชิงคุณภาพ โดยในงานวิจัยนี้ได้ใช้โปรแกรม Eclipse เวอร์ชัน 4.2 ไลบรารี JAX-WS และเว็บเซิร์ฟเวอร์ Apache Tomcat เวอร์ชัน 7.0.39

จากการประเมินค่าคุณสมบัติทางการออกแบบ แสดงให้เห็นว่าการนำยูเอ็มแอลโปรไฟล์มาประยุกต์ใช้เข้ากับระบบ ทำให้ค่าคุณลักษณะเชิงคุณภาพทุกชนิดมีแนวโน้มไปในทางที่ดีขึ้น ยกเว้นความสามารถในการทำความเข้าใจมีแนวโน้มไปในทางที่ไม่ดี เนื่องจากการออกแบบที่มีความซับซ้อนมากขึ้นทำให้สามารถเข้าใจระบบได้ยากขึ้น สำหรับการประเมินความสามารถในการทนต่อความผิดพร่อง แสดงให้เห็นว่าหลังจากการประยุกต์ใช้ยูเอ็มแอลโปรไฟล์ ระบบสามารถทนต่อความผิดพร่องได้จริง และมีการใช้เวลาในการทำงานเพิ่มขึ้นในกรณีที่มีความผิดพร่องเกิดขึ้นในระบบ ดังนั้นจึงสรุปได้ว่าการประยุกต์ใช้ยูเอ็มแอลโปรไฟล์สำหรับการทนต่อความผิดพร่อง สามารถทำให้ระบบมีความสามารถในการทนต่อความผิดพร่องมากกว่าระบบที่ไม่มีการประยุกต์ใช้ยูเอ็มแอลโปรไฟล์ แต่จำเป็นจะต้องแลกกับเวลาในการทำงานและความซับซ้อนของระบบที่เพิ่มขึ้นด้วย

6.2 ปัญหาและข้อจำกัดที่พบจากการวิจัย

6.2.1 ข้อจำกัดของแบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่อง

แบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่องของ Hanmer [5] ได้นำเสนอไปในแนวทางของข้อปฏิบัติเพื่อให้ผู้ใช้นำไปใช้ในการพัฒนาระบบมากกว่า ซึ่งแบบรูปส่วนมากที่นำเสนอนำมาออกแบบเป็นยูเอ็มแอลโปรไฟล์ได้ยาก หรือแบบรูปบางแบบรูปเป็นการนำเสนอแนวทางปฏิบัติซึ่งไม่สามารถนำมาออกแบบเป็นยูเอ็มแอลโปรไฟล์ได้ และข้อจำกัดอีกอย่างหนึ่งคือ แบบรูปบางแบบรูปมีเนื้อหาที่คาบเกี่ยวกัน ไม่ว่าจะเป็นแบบรูปที่อยู่ในกลุ่มเดียวกันหรือแบบรูปที่อยู่คนละกลุ่ม จากข้อจำกัดเหล่านี้ในงานวิจัยนี้จึงจำเป็นต้องเลือกเฉพาะแบบรูปที่สามารถนำมาออกแบบเป็นยูเอ็มแอลโปรไฟล์และนำไปประยุกต์ใช้ได้จริงกับระบบงานที่มีอยู่ในปัจจุบัน

6.2.2 ข้อจำกัดของการคัดเลือกแบบรูปไปประยุกต์ใช้ในระบบ

ถึงแม้ว่าในงานวิจัยนี้ได้มีการนำแบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่องของ Hanmer [5] มาออกแบบเป็นยูเอ็มแอลโปรไฟล์พร้อมทั้งอธิบายรายละเอียดและนำเสนอตัวอย่างการประยุกต์ใช้ แต่อย่างไรก็ตามการที่ผู้ใช้งานจะนำยูเอ็มแอลโปรไฟล์ใดไปใช้งานยังต้องอาศัยความสามารถในการตัดสินใจของผู้ออกแบบระบบ เพื่อให้มีความเหมาะสมกับระบบของตนเอง และเมื่อนำยูเอ็มแอลโปรไฟล์ไปประยุกต์ใช้แล้วระบบจะมีความเชื่อถือได้ตามแบบรูปที่ใช้

6.3 ข้อเสนอแนะ

งานวิจัยนี้สามารถพัฒนาเพิ่มเติมได้โดยการเพิ่มการออกแบบสำหรับกลุ่มของแบบรูปสำหรับซอฟต์แวร์ที่ทนต่อความผิดพร่องกลุ่มอื่น คือแบบรูปการบรรเทาข้อผิดพลาด และแบบรูปการรักษาความผิดพร่อง นอกจากนั้นยังพัฒนาต่อไปให้ สามารถนำยูเอ็มแอลโปรไฟล์ไปประยุกต์ใช้เข้ากับหลักการของ Model-Driven Architecture เพื่อที่จะใช้ในการสร้างโค้ดโดยอัตโนมัติได้

รายการอ้างอิง

[1] Quality of Web services [Online]. 2006. Available from: http://www.di.unito.it/~baroglio/SummerSchool06/Slides/QoWS.pdf. [2012, September].

[2] OMG. UML™ Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms [Online]. 2006 Available from: http://www.inf.ufrgs.br/~juliog/profiles/06-05-02.pdf. [2012, September].

[3] uml-diagrams.org. UML Profile Diagrams [Online]. 2013 Available from: http://www.uml-diagrams.org/profile-diagrams.html. [2013, Febuary].

[4] Shim, B., Baek, B., Kim, S., and Park, S. A Robot Fault-Tolerance Approach Based on Fault Type. Proceedings of the 2009 Ninth International Conference on Quality Software (QSIC '09), pp. 296-304. 2009.

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

[6] WS-I. Supply Chain Management Sample Application Architecture [Online]. 2003 Available from: http://www.ws-i.org/sampleapplications/supplychainmanagement/2003-12/scmarchitecture1.01.pdf. [2012, September].

[7] WS-I. Supply Chain Management Use Case Model [Online]. 2003 Available from: http://www.ws-i.org/sampleapplications/supplychainmanagement/2003-12/scmusecases1.0.pdf. 2012, September].

[8] Ongsiriporn, O. and Senivongse, T. UML Profile for Fault Tolerance Patterns for Service-Based Systems. Proceedings of The 10th International Joint Conference on Computer Science and Software Engineering (JCSSE'13), pp. 240-245. Thailand, May 29-31, 2013.

[9] W3C Working Group Note. Web Services Architecture [Online]. 2004 Available from: http://www.w3.org/TR/ws-arch/#id2260892. [2012, December].

[10] OMG. UML Version 2.4, Infrastructure Specification [Online]. 2010 Available

from: http://www.omg.org/spec/UML/2.4/Infrastructure/PDF. [2012, September].

[11] OMG. UML Version 2.4, Superstructure Specification [Online]. 2010 Available from: http://www.omg.org/spec/UML/2.4/Superstructure/PDF. [2012, September].

[12] OMG. OMG Systems Modeling Language (OMG SysML™) version 1.3 [Online]. 2012 Available from: http://www.omg.org/spec/SysML/1.3/PDF/. [2012, December].

[13] Bernard, S. and Merseguer, J. A UML Profile for Dependability Analysis of Real-time Embedded Systems. Proceedings of the 6th International Workshop on Software and Performance (WOSP '07), pp. 115-124. 2007.

[14] Tucci-Piergiovanni, S. Mraidha, C. Wozniak, E. Lanusse, A. and Gerard, S. A UML Model-Based Approach for Replication Assessment of AUTOSAR Safety-Critical Applications. Proceedings of the 2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications (TRUSTCOM '11), pp. 1176-1187. 2011.

[15] 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.

[16] Thaisongsuwan, T. and Senivongse, T. Applying Software Fault Tolerance Patterns to WS-BPEL Processes. Proceedings of The Eighth International Joint Conference on Computer Science and Software Engineering (JCSSE 2011), pp. 269-274. 2011.

[17] Gamma,E. Helm,R. Johnson, R. and Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software [Online]. Available from: http://www.uml.org.cn/c++/pdf/DesignPatterns.pdf. [2013, July].

[18] Westfall, L. Bidirectional Requirements Traceability [Online]. 2006 Available from: http://www.compaid.com/caiinternet/ezine/westfall-bidirectional.pdf. [2013, June].

[19] Shim,B. Choue, S. Kim, S. and Park, S. A Design Quality Model for Service-Oriented Architecture. Proceedings of The 2008 15th Asia-Pacific Software Engineering Conference (APSEC '08), pp. 403-410. 2008.

ประวัติผู้เขียนวิทยานิพนธ์

นางสาวอรอนงค์ องค์ศิริพร เกิดเมื่อวันที่ 1 มิถุนายน พ.ศ. 2530 ที่จังหวัดกรุงเทพมหานคร สำเร็จการศึกษาระดับปริญญาวิทยาศาสตรบัณฑิต สาขาวิทยาการคอมพิวเตอร์ เกียรตินิยมอันดับ 1 จากคณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังในปีการศึกษา 2551 และได้เข้าศึกษาในหลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิทยาศาสตร์คอมพิวเตอร์ ณ ภาควิศวกรรมศาสตร์คอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2554 งานวิจัยที่สนใจ ได้แก่ ระบบอิงบริการ ยูเอ็มแอลโปรไฟล์ และการทนต่อความผิดพร่อง