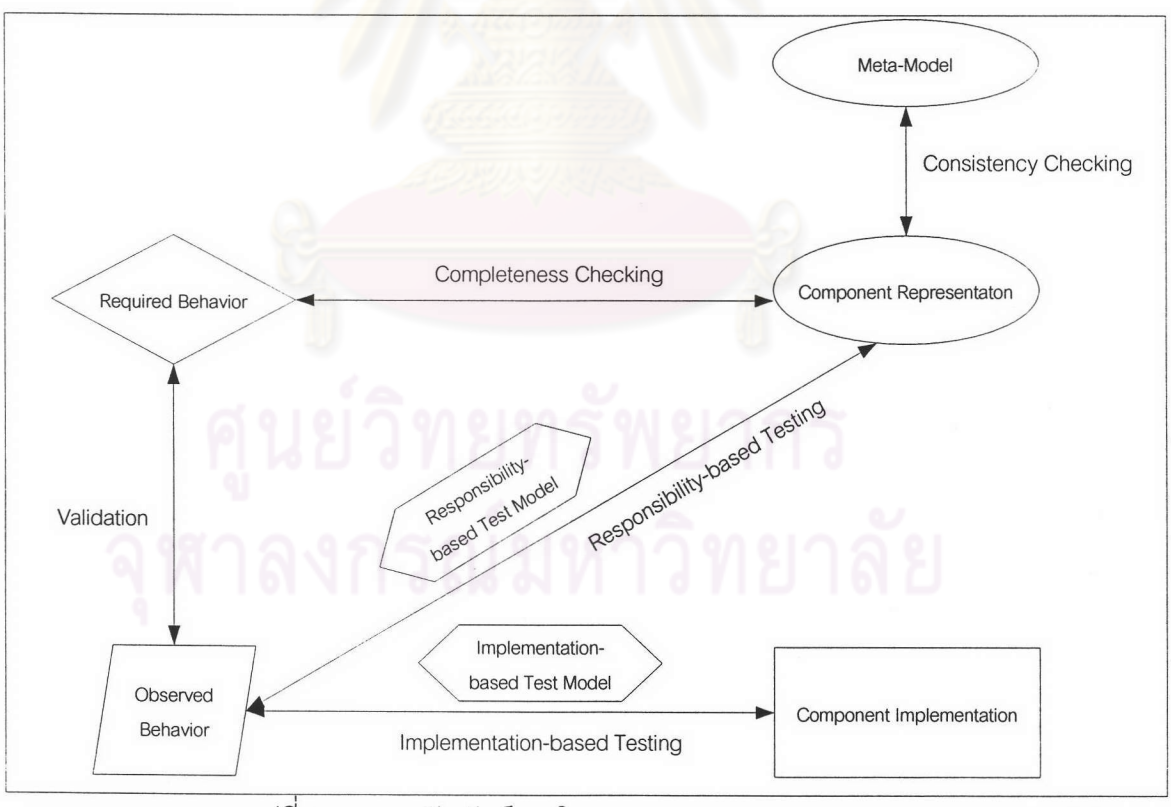
บทที่ 2

ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

ในบทนี้จะกล่าวถึงทฤษฎีที่เกี่ยวข้องกับการสร้างกรณีทดสอบจากแผนภาพสเตทชาร์ต ซึ่งประกอบไปด้วย โมเดลการทดสอบ การทดสอบซอฟต์แวร์ ส่วนประกอบของกรณีทดสอบ แผนภาพยูเอ็มแอล กฎการขยายเหตุการณ์ที่มีการกระตุ้น เอ็กซ์เอ็มไอ เอ็กซ์เอสแอลที นอกจากนี้จะกล่าวถึงงานวิจัยที่เกี่ยวข้อง ได้แก่ การสร้างกรณีทดสอบจากข้อกำหนดยูเอ็มแอล หลักการในการสร้างการทดสอบจากข้อกำหนดรายละเอียด และการเลือกการทดสอบจากแผนภาพสเตทชาร์ต ซึ่งแต่ละส่วนมีรายละเอียดดังนี้

2.1 ทฤษฎีที่เกี่ยวข้อง

2.1.1 โมเดลการทดสอบ (Testing model) [1]



รูปที่ 2.1 ความสัมพันธ์ของโมเดล การทดสอบ และการตรวจสอบระบบ [1]

จากรูปที่ 2.1 เป็นโมเดลการทดสอบซึ่งแสดงความสัมพันธ์ของโมเดลการทดสอบ และการตรวจสอบความถูกต้องของระบบ ซึ่งเมตาโมเดล (Meta model) ในที่นี้คือโครงสร้างของ

แผนภาพยูเอ็มแอล ซึ่งจะมีหลายแผนภาพเป็นส่วนประกอบ (Component representation) แผนภาพที่เป็นส่วนประกอบในแผนภาพยูเอ็มแอลเช่น แผนภาพคลาส แผนภาพสเตทชาร์ต ซึ่งแผนภาพที่ต้องการนำมาทดสอบนั้นต้องมีการตรวจสอบความสอดคล้อง (Consistency-checking) กับโครงสร้างของแผนภาพยูเอ็มแอล คือทำการตรวจสอบว่าแผนภาพที่ต้องการนำมาทดสอบมีการออกแบบตามสัญลักษณ์หรือโครงสร้างของแผนภาพยูเอ็มแอลมีความถูกต้องหรือไม่ จากนั้นนำมาตรวจสอบกับพฤติกรรมที่ต้องการ (Required behavior) ซึ่งอาจจะหมายถึงความต้องการของผู้ใช้ว่าครบสมบูรณ์หรือไม่ (Completeness checking) เมื่อต้องการทดสอบแผนภาพโดยขึ้นอยู่กับหน้าที่ความรับผิดชอบ (Responsibility-based testing) ต้องสร้างโมเดลการทดสอบที่อยู่บนพื้นฐานของหน้าที่ความรับผิดชอบ (Responsibility-based test model) เพื่อให้ได้พฤติกรรมจากการสังเกต (Observed behavior) ซึ่งอาจจะหมายถึงผลลัพธ์ที่ได้จากการทดสอบ จากนั้นนำมาตรวจสอบความถูกต้อง (Validation) กับความต้องการของผู้ใช้ว่าถูกต้องหรือไม่ ซึ่งในส่วนที่อธิบายมานี้จะเป็นส่วนของการวิเคราะห์และออกแบบในกระบวนการผลิตซอฟต์แวร์ ซึ่งในส่วนของการพัฒนาซอฟต์แวร์ จะทำการเขียนโปรแกรมและทำการสร้างโมเดลทดสอบที่อยู่บนพื้นฐานของการพัฒนา (Implementation-based test model) ซึ่งดูจากผลลัพธ์จากการทดสอบ

2.1.2 การทดสอบซอฟต์แวร์ [2, 4, 5, 6, 7]

Beizer [4] ได้นิยามระดับในการทดสอบซอฟต์แวร์อยู่ 3 ระดับ คือ

1) การทดสอบระดับหน่วย เป็นการทดสอบในระดับที่เล็กที่สุด คือเป็นการทดสอบการทำงานของแต่ละโมดูล (Module) ภายในโปรแกรม โดยแต่ละโมดูลถูกทดสอบแยกออกจากกันทำให้บางโมดูลจำเป็นต้องใช้โปรแกรมที่ถูกเขียนขึ้นมาเฉพาะ นั่นคือตัวขับ (Driver) และตัวดำเนินการ โดยตัวขับเป็นโปรแกรมที่ใช้จำลองการเรียกโมดูลที่ต้องการทดสอบพร้อมกับให้ข้อมูลเข้าและรับผลลัพธ์จากการทำงานของโมดูลนั้น และตัวดำเนินการเป็นโปรแกรมจำลองโมดูลที่จะทดสอบนั้นต้องการเรียกใช้ ซึ่งกรณีทดสอบของการทดสอบระดับหน่วยต้องระบุตัวขับ และตัวดำเนินการสำหรับแต่ละโมดูลไว้ โดยผู้วิจัยสนใจในการที่จะทำการทดสอบระดับหน่วยของแผนภาพสเตท-ชาร์ตของคลาสใดคลาสหนึ่ง
2) การทดสอบแบบรวม (Integration testing) เป็นการทดสอบข้อผิดพลาดที่เกิดจากการรวมโมดูลเข้าด้วยกัน และเป็นการทดสอบความเข้ากันได้ของโมดูล
3) การทดสอบระบบ (System testing) เป็นการตรวจสอบความถูกต้องในการทำงานของส่วนต่างๆ ในระบบทั้งหมด โดยจะครอบคลุมการทำงานของระบบทั้งซอฟต์แวร์และฮาร์ดแวร์ เพื่อให้ระบบที่พัฒนาไม่เกิดความผิดพลาดขึ้นในการทำงาน เช่น

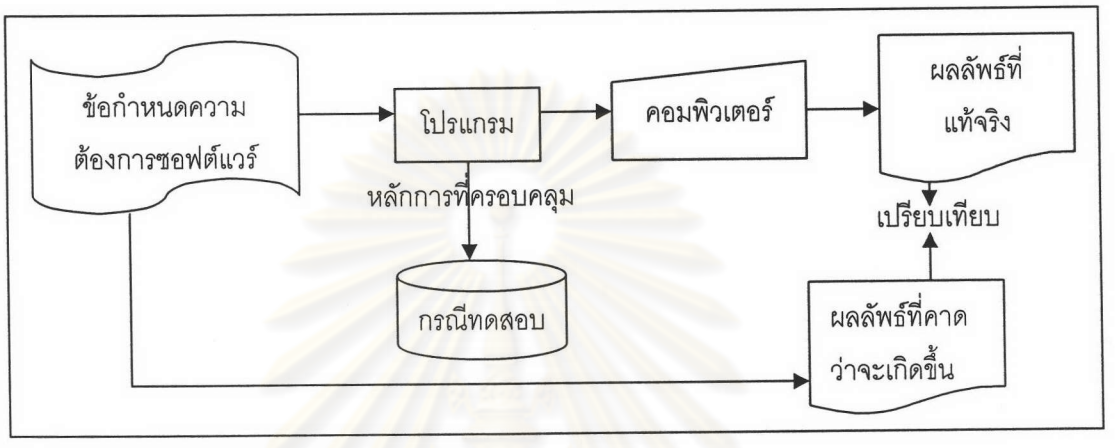
ทดสอบประสิทธิภาพ (Performance testing) ว่าเป็นไปตามที่กำหนดไว้หรือไม่ การทดสอบความปลอดภัย (Security testing) โดยตรวจสอบว่าข้อมูลมีความปลอดภัยจากผู้บุกรุก (Hacker) หรือไม่ เป็นต้น

ซึ่งมีกระบวนการในการทดสอบโดยทั่วไปมีดังนี้

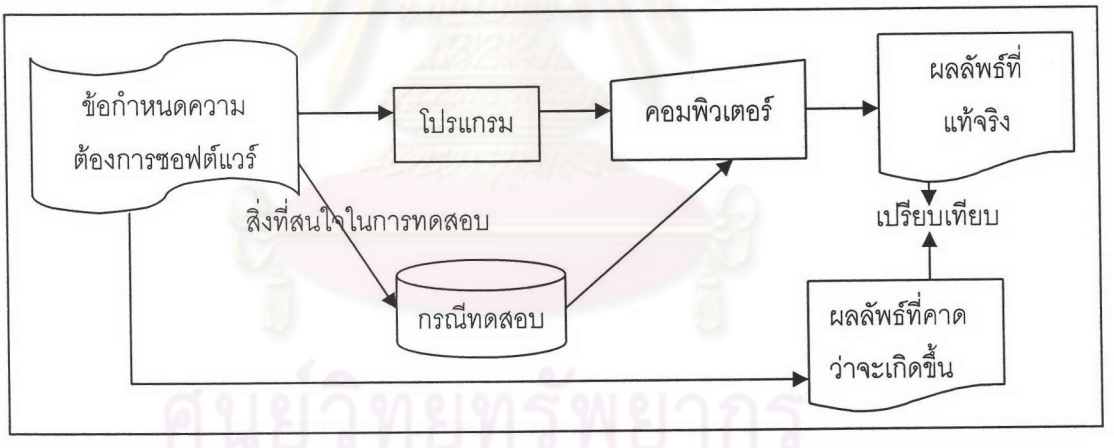
1) การทดสอบแบบแบล็กบอกซ์ (Black-box testing) เป็นกระบวนการสร้างข้อมูลการทดสอบ โดยไม่ต้องใช้ความรู้ในเรื่องโครงสร้างของซอฟต์แวร์ที่ทำการทดสอบ ซึ่งโดยทั่วไปจะขึ้นอยู่กับความต้องการผู้ใช้ (User requirements) และข้อกำหนดความต้องการของซอฟต์แวร์ (Software requirements specification) ซึ่งทำการทดสอบโดยสนใจเฉพาะข้อมูลเข้า (Inputs) และข้อมูลออก (Outputs) เท่านั้น โดยไม่สนใจกระบวนการทำงานภายใน (Process) ว่าจะทำงานอย่างไร ซึ่งผู้วิจัยสนใจที่จะสร้างกรณีทดสอบแบบแบล็กบอกซ์เนื่องจากว่าต้องการทดสอบว่าคลาสที่พิจารณาทำงานถูกต้องหรือไม่ รูปที่ 2.3 แสดงภาพรวมของการสร้างกรณีทดสอบโดยพิจารณาจากข้อกำหนดความต้องการของซอฟต์แวร์ ซึ่งจากรูป สิ่งที่สำคัญในการสร้างการทดสอบจากข้อกำหนดความต้องการคือ การทดสอบในช่วงต้นๆ ของกระบวนการพัฒนาและพร้อมสำหรับการดำเนินการก่อนโปรแกรมจะเสร็จ นอกจากนี้เมื่อการทดสอบถูกสร้างขึ้น ผู้ทดสอบสามารถหาความไม่สอดคล้องกัน (Inconsistency) และความกำกวม (Ambiguity) ในข้อกำหนดความต้องการได้ ซึ่งจะช่วยให้ข้อกำหนดถูกปรับปรุงก่อนที่จะเขียนโปรแกรม ซึ่งผู้วิจัยสนใจในการสร้างกรณีทดสอบจากข้อกำหนดความต้องการโดยดูจากข้อกำหนดของแผนภาพสเตทชาร์ท

2) การทดสอบแบบไวท์บอกซ์ (White-box testing) เป็นกระบวนการสร้างข้อมูลการทดสอบจากโครงสร้างของโปรแกรมซึ่งโดยทั่วไปจะขึ้นอยู่กับโปรแกรมต้นฉบับ (Source code) ซึ่งเป็นการทดสอบที่พิจารณาในส่วนของโครงสร้างหรือกระบวนการภายในของระบบหรือของตัวโปรแกรมว่าทำงานได้ถูกต้องหรือไม่ และมีข้อผิดพลาดตรงไหน รูปที่ 2.2 แสดงภาพรวมของการสร้างกรณีทดสอบโดยพิจารณาจากโปรแกรม ซึ่งจากรูป กรณีทดสอบที่ได้ จะได้มาจากการพิจารณาจากโปรแกรม ซึ่งเป็นโปรแกรมที่สอดคล้องกันข้อกำหนดความต้องการซอฟต์แวร์ตามหลักการที่ครอบคลุมบางหลักการของการสร้างกรณีทดสอบ เช่น การแตกสาขา (Branch) หรือการไหลของข้อมูล (Data flow) โดยกรณีทดสอบที่ได้จะสร้างผลลัพธ์ที่แท้จริง ซึ่งต้องเปรียบเทียบกับผลลัพธ์ที่คาดว่าจะเกิดขึ้นว่าสอดคล้องกันหรือไม่

การทดสอบระบบจะประกอบไปด้วยชุดของกรณีทดสอบ ซึ่งใช้ในการทดสอบการทำงานของซอฟต์แวร์ ซึ่งทำการเปรียบเทียบผลลัพธ์ที่แท้จริง (Actual outputs) กับผลลัพธ์ที่คาดว่าจะเกิดขึ้น (Expected outputs) ซึ่งการทดสอบและการออกแบบการทดสอบ เป็นส่วนหนึ่งของการประกันคุณภาพ (Quality assurance) โดยให้ความสนใจกับการป้องกันการเกิดข้อผิดพลาด (Fault prevention) ที่อาจจะเกิดขึ้น



รูปที่ 2.2 การสร้างกรณีทดสอบโดยพิจารณาจากโปรแกรม [7]



รูปที่ 2.3 การสร้างกรณีทดสอบโดยพิจารณาจากข้อกำหนดความต้องการของซอฟต์แวร์ [7]

2.1.3 ส่วนประกอบของกรณีทดสอบ [4, 8]

กรณีทดสอบคือ ข้อมูลที่ผู้ทดสอบใช้ทำการทดสอบ เพื่อให้ทราบว่าการทำงานที่ต้องการถูกต้องหรือไม่ ซึ่งจะประกอบไปด้วย [4]

2.1.3.1 ค่าของกรณีทดสอบ (Test case values) เป็นค่าที่มาจากความต้องการของการทดสอบ (Test requirement) อาจเป็นได้ทั้ง คำสั่ง

(Command) ข้อมูลเข้าของผู้ใช้ (User inputs) หรือค่าของพารามิเตอร์ในการสร้างการทดสอบจากการพิจารณาสถานะของระบบ (State-based testing) ค่าของกรณีทดสอบจะได้โดยตรงจากตัวกระตุ้นเหตุการณ์ และเงื่อนไขก่อนหน้าสำหรับทรานสิชัน

2.1.3.2 ค่าของพรีฟิก (Prefix values) จะประกอบไปด้วยข้อมูลนำเข้าทั้งหมดที่จำเป็นในการเข้าถึงสถานะก่อนหน้า (Pre-state) นั่นก็คือสิ่งแวดล้อมของการทดสอบ (Test environments) ซึ่งเป็นสิ่งที่กำหนดให้ถูกต้องก่อนที่จะมีการทดสอบกรณีทดสอบใดๆ ซึ่งแต่ละกรณีทดสอบก็จะมีสิ่งแวดล้อมของการทดสอบที่ไม่เหมือนกันขึ้นอยู่กับเหตุการณ์ที่เข้ามา

2.1.3.3 ค่าของเวอริไฟ (Verify values) เป็นข้อมูลนำเข้าใดๆ ที่จำเป็นในการแสดงผล

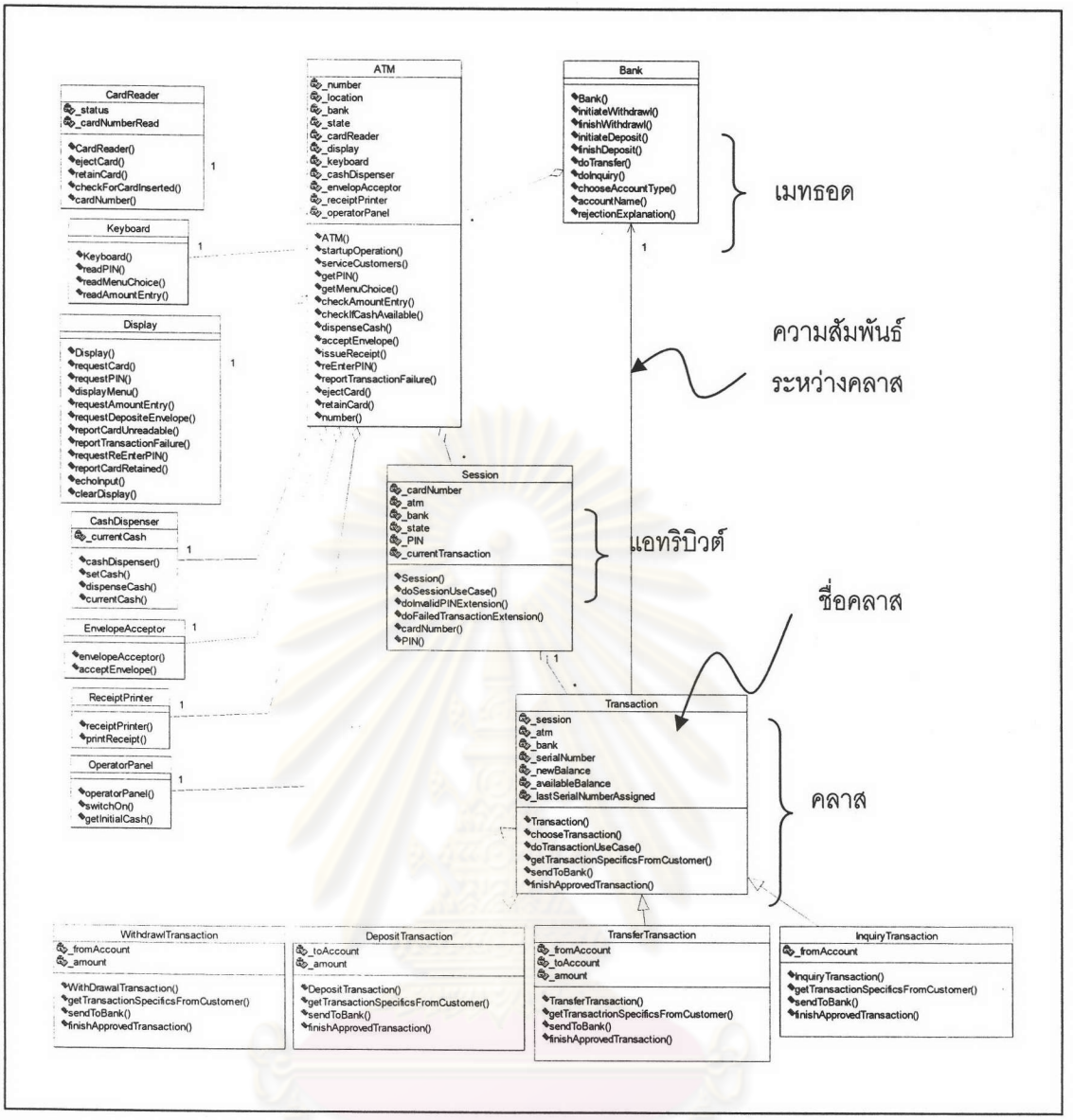
2.1.3.4 คำสั่งให้หยุดการทำงาน (Exit Commands) เป็นคำสั่งที่ทำให้หยุดการทำงานซึ่งจะขึ้นอยู่กับระบบที่ทำการทดสอบ

2.1.3.5 ผลลัพธ์ที่คาดว่าจะเกิดขึ้น สร้างมาจากค่าหลังจากดำเนินการ (After-value) ของตัวกระตุ้นเหตุการณ์และเงื่อนไขหลังการดำเนินการ (Post-condition) ที่เกี่ยวข้องกับทรานสิชัน

2.1.4 แผนภาพยูเอ็มแอล [2, 3, 9, 10, 11, 12]

แผนภาพยูเอ็มแอลที่เกี่ยวข้องกับงานวิจัยนี้ ได้แก่

2.1.4.1 แผนภาพคลาส เป็นแผนภาพที่แสดงความสัมพันธ์กันระหว่างคลาสว่ามีความสัมพันธ์กันแบบไหน มีคุณสมบัติอะไรบ้าง ในรูปที่ 2.4 แสดงส่วนประกอบของแผนภาพคลาส ซึ่งเป็นของระบบเอทีเอ็ม (ATM: Automatic Teller Machine) โดยรายละเอียดมีดังนี้ (ซึ่งวากยสัมพันธ์ของแผนภาพคลาสแสดงในภาคผนวก ก)



รูปที่ 2.4 ส่วนประกอบของแผนภาพคลาส

1) ชื่อคลาส (Class Name) และแอทริบิวต์ (Attribute) ของแต่ละคลาส จะบอกถึงคุณสมบัติของคลาสที่มี เช่น คลาสเอทีเอ็ม (ATM class) มีแอทริบิวต์ดังนี้คือ หมายเลขบัตร (number) ธนาคารที่ฝาก (bank) สถานที่ฝาก (location) เป็นต้น ซึ่งส่วนของแอทริบิวต์ต่างๆ ภายในคลาส อาจจะนำไปแทนเป็นสถานะต่างๆ ภายในแผนภาพสเตทชาร์ตได้

2) โอเปอร์เรชัน (Operation) หรือเมทธอด (Method) แทนการกระทำ (Action) หรือฟังก์ชันที่คลาสสามารถดำเนินการได้ เช่น เมทธอดการใส่รหัสผ่าน (getPIN) ของคลาสเอทีเอ็ม จะเป็นการ

ให้ใส่รหัสผ่านและตรวจสอบรหัสผ่านว่าถูกต้องหรือไม่ เป็นต้น ซึ่งโอเปอร์เรชันของแผนภาพคลาสที่เกิดขึ้นจะเทียบได้กับเหตุการณ์ (Event) ของแผนภาพสเตทชาร์ต

3) ความสัมพันธ์แบบแอสโซซิเอชัน (Association) แทนความสัมพันธ์ระหว่างหลายๆ คลาสหรือระหว่างคลาสและตัวเอง

4) คลาสที่มีลักษณะขึ้นอยู่กับสถานะ คือ ผลลัพธ์ที่ได้จะไม่ขึ้นอยู่กับข้อมูลที่เข้ามา แต่จะขึ้นอยู่กับว่าสถานะปัจจุบัน ณ ขณะนั้นเป็นอะไร เช่น ในระบบเอทีเอ็ม จะมีคลาสการอ่านบัตร (CardReader class) คลาสเอทีเอ็ม คลาสเซสชัน (Session-class) เป็นต้น ซึ่งจะมีคลาสเซสชัน เป็นคลาสที่ขึ้นกับสถานะคือมีข้อมูลเข้าเป็นใส่บัตรเอทีเอ็ม (CardInserted) แต่ผลลัพธ์ที่ได้ออกมาจะขึ้นอยู่กับว่า ณ ขณะนั้นทำงานอยู่ที่สถานะใด เช่น สถานะถอนเงิน สถานะฝากเงิน เป็นต้น ส่วนคลาสที่ไม่ขึ้นอยู่กับสถานะนั้น (State-independent) จะเป็นคลาสที่ผลลัพธ์จะเป็นไปตามข้อมูลที่เข้ามา เช่น คลาสของการพิมพ์รายงาน (ReceiptPrinter class) คือ เมื่อคลาส ก มีการร้องขอรายงานจากคลาส ข คลาส ข ก็จะมีการพิมพ์รายงานส่งไปให้คลาส ก เป็นต้น

2.1.4.2 แผนภาพสเตทชาร์ต เป็นแผนภาพที่แสดงถึงสถานะที่ต่างกันของอ็อบเจกต์ในระหว่างการทำงานว่ามีเหตุการณ์ใดบ้างที่ทำให้อ็อบ-เจกต์หนึ่งเปลี่ยนจากสถานะหนึ่งไปเป็นอีกสถานะหนึ่ง โดยในยูเอ็มแอลได้แบ่งประเภทของทรานสิชันและเหตุการณ์ ได้เป็นดังนี้ (ซึ่งวากยสัมพันธ์ของแผนภาพสเตทชาร์ตแสดงในภาพผนวก ก)

ประเภทของทรานสิชัน แบ่งออกเป็น 5 ประเภท [10] คือ

1) ทรานสิชันระดับสูง (High-level transitions) เป็นทรานสิชันที่อยู่ในขอบเขตของสถานะที่มีสถานะย่อย ซึ่งเป็นสถานะที่ประกอบด้วยสถานะย่อยที่ทำงานพร้อมกัน หรือเป็นสถานะย่อย

ธรรมดาคือไม่มีการทำงานพร้อมกัน เป็นการทำงานเหมือนสถานะปกติ

2) ทรานสิชันแบบสมบูรณ์ (Completions transitions) เป็นทรานสิชันที่ไม่มีเหตุการณ์ที่เป็นตัวกระตุ้น

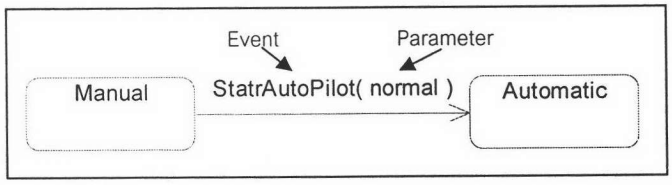
3) ทรานสิชันภายใน (Internal transitions) ดำเนินการโดยไม่มีสถานะเป้าหมายแต่จะมีสถานะเริ่มต้น คือ มีการกระทำเกิดขึ้นภายในสถานะโดยไม่ได้ออกจากสถานะ

4) เอนนาเบิลทรานสิชัน ทำงานได้ด้วยมีเหตุการณ์ และเริ่มจากสถานะที่พร้อมจะทำงาน (Active state) ซึ่งจะถูกกระตุ้นเมื่อมีทรานสิชันอย่างน้อย 1 ทรานสิชันจากสถานะเริ่มต้นไปยังสถานะเป้าหมาย

5) ทรานสิชันกับตัวเอง เป็นทรานสิชันที่มีสถานะเริ่มต้นและสถานะเป้าหมายเป็นจุดเดียวกัน ซึ่งจะต่างจากทรานสิชันภายใน คือ ทรานสิชันกับตัวเอง มีสถานะเป้าหมายโดยเมื่อมีเหตุการณ์มากระตุ้นแล้วเกิดการกระทำขึ้นภายนอกสถานะ เมื่อทำการกระทำเสร็จก็จะเข้ามายังสถานะเดิม

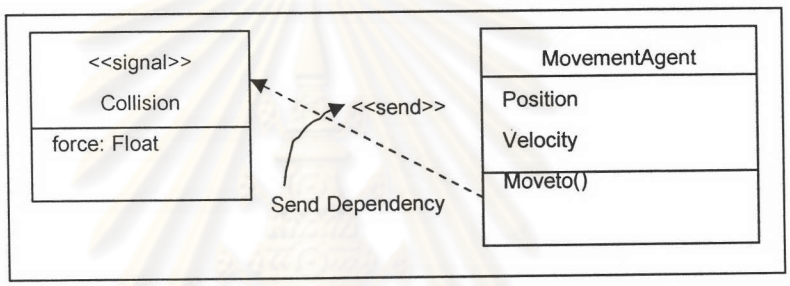
ประเภทของเหตุการณ์ แบ่งออกเป็น 4 ประเภท [2] คือ

1) คอลอีเวนท์ (Call event) แทนการที่อ็อบเจกต์หนึ่งร้องขอใช้เมทธอดหรือโอเปอร์เรชันของอีกอ็อบเจกต์หนึ่งที่มีสถานะ แสดงได้ดังรูปที่ 2.5 จากรูปเป็นการแสดงเหตุการณ์แบบคอลอีเวนท์ ซึ่งสถานะของ Manual จะเปลี่ยนไปเป็นสถานะ Automatic โดยจะมีเหตุการณ์ที่ชื่อว่า startAutopilot เข้ามากระตุ้นให้เกิดการเปลี่ยนสถานะโดยมีการส่งพารามิเตอร์เข้ามาด้วยคือ normal



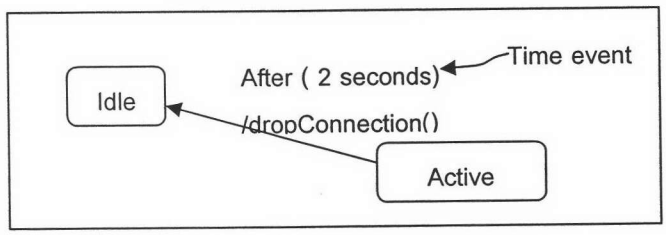
รูปที่ 2.5 คอลอีเวนท์

2) ซิกเนลอีเวนท์ (Signal event) แทนการรับสัญญาณจากอ็อบเจกต์อื่น แสดงได้ดังรูปที่ 2.6 ซึ่งเหตุการณ์ที่เป็นข้อยกเว้น (Exception) เป็นตัวอย่างหนึ่งของซิกเนล (Signal) จากรูปสามารถจำลองซิกเนล ได้ด้วยคลาส Collision ที่มี stereotype เป็น <<signal>> (ซึ่ง stereotype จะแทนด้วยเครื่องหมาย "<< >>") โดยที่เมทธอดที่ชื่อว่า Moveto() ในคลาส MovementAgent ทำการส่งสัญญาณ Collision ไปยังคลาสอื่น โดยมีพารามิเตอร์ที่ส่งไปด้วยคือ force ซึ่งมีชนิดเป็นเลขทศนิยม โดยที่ <<send>> จะเป็นตัวระบุว่าเมทธอด Moveto() ได้ทำการส่งสัญญาณนี้



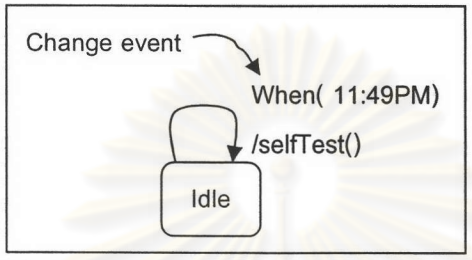
รูปที่ 2.6 ซิกเนลอีเวนท์

3) ไทม์อีเวนท์ (Time event) แทนช่วงของเวลาหลังจากเรียกเหตุการณ์ (โดยทั่วไปเป็นทางเข้าของสถานะปัจจุบัน) ในยูเอ็มแอล ไทม์อีเวนท์ถูกโมเดลด้วยการใช้คำหลัก (keyword) ว่า "After" ตามหลังนิพจน์บางอย่างที่กำหนดเป็นช่วงเวลา แสดงได้ดังรูปที่ 2.7 ซึ่งเริ่มแรกจะเป็นสถานะ Active เมื่อมีไทม์อีเวนท์เข้ามาคือ After(2 seconds) นั่นคือ หลังจาก 2 วินาทีไปก็จะทำการปิดการติดต่อ(dropConnection()) จากนั้นจึงเปลี่ยนสถานะจาก Active เป็นสถานะ Idle



รูปที่ 2.7 ไทม์อีเวนท์

4) เช้จอีเวนท์ เป็นเหตุการณ์ที่เกิดขึ้นเมื่อมีนิพจน์แบบบูลลีนมีค่าเป็นจริงในยูเอ็มแอล เช้จอีเวนท์ถูกโมเดลด้วยการใช้คำหลักว่า "When" ตามหลังนิพจน์แบบบูลลีน แสดงได้ดังรูปที่ 2.8 ซึ่งเมื่อมีเหตุการณ์เข้ามาคือ When(11:49PM) เมื่อมีค่าเป็นจริงแล้วก็จะทำการกระทำคือ การทดสอบตัวเอง (selfTest()) จากนั้นก็จะเปลี่ยนสถานะจาก Idle ไปเป็นสถานะเดิม เป็นต้น



รูปที่ 2.8 เช้จอีเวนท์

2.1.5 กฎการขยายเหตุการณ์ที่มีการกระตุ้น [13]

เหตุการณ์ที่มีการกระตุ้น จะทำให้ทรานสิชันเกิดการเปลี่ยนแปลงในแผนภาพสเตท-ชาร์ต จึงทำให้เกิดการเปลี่ยนสถานะจากสถานะหนึ่งไปเป็นอีกสถานะหนึ่ง โดยจะพิจารณากฎการขยายเหตุการณ์ที่มีการกระตุ้น ดังนี้

- $@T(X) = \neg X \wedge X'$ ---(1)
- $@T(X \wedge Y) = \neg(X \wedge Y) \wedge (X' \wedge Y')$
  $= (\neg X \vee \neg Y) \wedge X' \wedge Y'$ ---(2)
- $@T(X \vee Y) = \neg(X \vee Y) \wedge (X' \vee Y')$
  $= \neg X \wedge \neg Y \wedge (X' \vee Y')$ ---(3)
- $@F(X) = X \wedge \neg X'$ ---(4)
- $@F(X \wedge Y) = (X \wedge Y) \wedge \neg(X' \wedge Y')$
  $= (X \wedge Y) \wedge (\neg X' \vee \neg Y')$ ---(5)
- $@F(X \vee Y) = (X \vee Y) \wedge \neg(X' \vee Y')$
  $= (X \vee Y) \wedge (\neg X' \wedge \neg Y')$ ---(6)

จากกฎข้างต้นนิยามแต่ละค่าได้ดังนี้

(1) @T หรือ @F แทนเหตุการณ์ที่มีการกระตุ้น (Triggering event) ซึ่งหมายความว่าค่าของ X ต้องเปลี่ยนเมื่อมีการกระตุ้นให้เกิดการเปลี่ยนสถานะในทรานสิชัน โดยที่ X คือค่าในวงเล็บของ @T หรือ @F

(2) @T(X) หมายความว่า ค่า X ต้องเปลี่ยนจากเท็จ เป็น จริง จึงจะทำให้เกิดการเปลี่ยนสถานะ

(3) @F(X) หมายความว่า ค่า X ต้องเปลี่ยนจากจริง เป็น เท็จ จึงทำให้เกิดการเปลี่ยนสถานะ

(4) X และ X' แทนค่าของเหตุการณ์ก่อนการกระตุ้น และ ค่าของเหตุการณ์หลังการกระตุ้นซึ่งจะทำให้เกิดการเปลี่ยนสถานะ

จากกฎทั้ง 6 กฎที่ได้แสดงข้างต้น จะมีการแยกพิจารณาค่าของเหตุการณ์เป็น 2 ค่า คือ ค่าของเหตุการณ์ก่อนการกระตุ้น และค่าของเหตุการณ์หลังจากมีการกระตุ้นเกิดขึ้น ซึ่งจะทำให้เกิดการเปลี่ยนสถานะในแผนภาพสเตทชาร์ต ตัวอย่างเช่น จากกฎข้อที่ 1 เมื่อทำการขยายกฎจะได้ค่าของเหตุการณ์ก่อนและหลังการกระตุ้น โดยจะเกิดการเปลี่ยนสถานะได้ก็ต่อเมื่อค่าของเหตุการณ์ก่อนการกระตุ้นต้องเป็นเท็จ และค่าหลังการกระตุ้นต้องเป็นจริง โดยจะให้ค่า X และ X' คือ เท็จและจริง เมื่อเข้าสู่กฎที่ 1 จะได้ ¬(เท็จ) ∧ (จริง) ซึ่งจะให้ผลลัพธ์ออกมาเป็นจริง ดังนั้นจึงเกิดการเปลี่ยนสถานะขึ้น ส่วนกฎข้ออื่นๆ พิจารณาเหมือนกับกฎข้อที่ 1 เป็นต้น

จากทฤษฎีดังกล่าวผู้วิจัยได้เลือกใช้กฎทั้ง 6 ข้อข้างต้นในการพิจารณาสร้างกรณีทดสอบจากแผนภาพสเตทชาร์ต เนื่องจากมีการพิจารณาค่าของเหตุการณ์ก่อนและหลังการกระตุ้นซึ่งทำให้เห็นค่าของเหตุการณ์ก่อนและหลังการเปลี่ยนสถานะได้ชัดเจนขึ้น ซึ่งจะทำให้สะดวกในการทดสอบ

2.1.6 เอ็กซ์เอ็มไอ (XMI : XML Metadata Interchange) [14, 15]

จุดประสงค์หลักของเอ็กซ์เอ็มไอ คือเพื่ออำนวยความสะดวกในการแลกเปลี่ยนเมตา-เดตา ระหว่างเครื่องมือที่เกี่ยวข้องกับการทำโมเดล (บนพื้นฐานของยูเอ็มแอล) กับที่เก็บเมตา-เดตา (Metadata repositories) ที่มีพื้นฐานมาจากเอ็มโอเอฟ (MOF : Meta Object Facility) โดยเอ็กซ์เอ็มไอเป็นการรวมเอามาตรฐาน 3 มาตรฐานดังต่อไปนี้เข้าไว้ด้วยกัน

1. เอ็กซ์เอ็มแอล (XML : Extensible Markup Language) [16] เป็นมาตรฐานของดับเบิลยูทรีซี (W3C : World Wide Web Consortium)

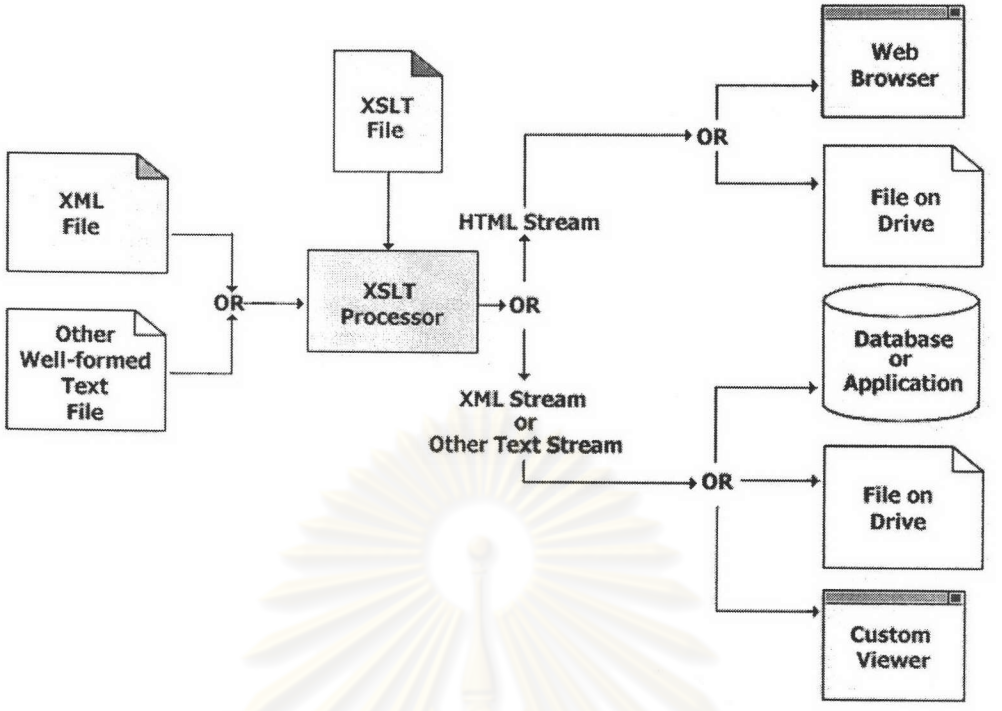
2. ยูเอ็มแอล เป็นมาตรฐานที่กำหนดโดยโอเอ็มจี (OMG : Object Management Group) เพื่อใช้ในการทำโมเดล
3. เอ็มโอเอฟ เป็นมาตรฐานที่กำหนดโดยโอเอ็มจี เพื่อใช้ในการทำเมตาโมเดลและที่เก็บเมตาเดตา

การรวมกันของมาตรฐานทั้ง 3 นี้ จึงถือเป็นการรวมเอาเทคโนโลยีเกี่ยวกับเมตาเดตาและการทำโมเดลของโอเอ็มจีและดับเบิลยูทรีซี (W3C: The World Wide Web Consortium) เข้าไว้ด้วยกัน ซึ่งทำให้นักพัฒนาซอฟต์แวร์สามารถทำการแลกเปลี่ยนโมเดลของอ็อบเจกต์หรือเมตาเดตาชนิดอื่นๆ ได้สะดวกโดยเฉพาะอย่างยิ่งในการแลกเปลี่ยนข้อมูลกันทางอินเทอร์เน็ต (Internet) (ซึ่งโครงสร้างของแฟ้มข้อมูลเอ็กซ์เอ็มไอที่นำมาทำการแปลงเป็นกรณีทดสอบแสดงได้ดังภาคผนวก ข)

ในวิทยานิพนธ์นี้ ได้ใช้แฟ้มข้อมูลตามมาตรฐานเอ็กซ์เอ็มไอ เพื่อแลกเปลี่ยนข้อมูลระหว่างเครื่องมือที่จะสร้างขึ้นกับเครื่องมือสร้างภาพยูเอ็มแอล

2.1.7 เอ็กซ์เอสแอลที (XSLT : eXtensible Stylesheet Language Transformations) [17, 18, 19, 20, 21, 22]

เอ็กซ์เอสแอลทีเป็นมาตรฐานของดับเบิลยูทรีซี เพื่อใช้ในการอธิบายการแปลงเอกสารเอ็กซ์เอ็มแอลแบบหนึ่งไปเป็นเอกสารเอ็กซ์เอ็มแอลอีกแบบหนึ่งที่มีโครงสร้างแตกต่างกัน หรืออาจใช้อธิบายการแปลงเอกสารเอ็กซ์เอ็มแอลไปเป็นเอกสารประเภทอื่นก็ได้เช่นเดียวกัน เช่นเอกสารเอชทีเอ็มแอล (HTML) เอกสารข้อความ (Text) เอกสารพีดีเอฟ (PDF) เป็นต้น โดยภาษาเอ็กซ์เอสแอลทีจะถูกระบุอยู่ในรูปแบบของเอกสารเอ็กซ์เอ็มแอล โดยจะมีเอ็กซ์เอสแอลทีโปรเซสเซอร์ (XSLT processor) ทำการอ่านเอกสารเอ็กซ์เอ็มแอล ซึ่งจะพิจารณาโหนดภายในเอกสารเทียบกับเทมเพลต (Template) ที่อยู่ในเอกสารเอ็กซ์เอสแอลที และให้ผลลัพธ์ออกมาเป็นเอกสารรูปแบบต่างๆ ที่ต้องการ ดังแสดงในรูปที่ 2.9 เป็นกระบวนการในการแปลงเอกสารด้วยภาษาเอ็กซ์เอสแอลที



รูปที่ 2.9 การแปลงเอกสารด้วยภาษาเอ็กซ์เอสแอลที [19]

โครงสร้างของภาษาเอ็กซ์เอสแอลทีจะประกอบไปด้วยเทมเพลต (Template) ต่าง ๆ โดยแต่ละเทมเพลตจะระบุกฎในการทำการแปลงเอกสารเอ็กซ์เอ็มแอล โดยวิธีที่จะให้เทมเพลตแต่ละตัวทำการแปลงสามารถแบ่งออกได้เป็น 2 วิธี ดังนี้

1. ระบุแอทริบิวต์ “match” ให้กับเทมเพลต ซึ่งเป็นการบ่งบอกว่าเทมเพลตนี้จะทำการแปลงส่วนย่อย (Element) ใดในเอกสารเอ็กซ์เอ็มแอลที่เป็นข้อมูลเข้า
2. ระบุแอทริบิวต์ “name” ให้กับเทมเพลต แล้วทำการเรียกเทมเพลตนี้โดยตรงด้วยส่วนย่อย “xsl:call-template”

รูปที่ 2.10 และ 2.11 เป็นตัวอย่างของภาษาเอ็กซ์เอสแอลที โดยในรูปที่ 2.10 เป็นตัวอย่างในการใช้ภาษาเอ็กซ์เอสแอลทีทำการแปลงเอกสารเอ็กซ์เอ็มแอลไปเป็นเอกสารเอชทีเอ็ม-แอล โดยมีการสร้างเทมเพลตต่าง ๆ ที่มีแอทริบิวต์ “match” สอดคล้องกับส่วนย่อยแต่ละชนิดในเอกสารเอ็กซ์เอ็มแอล และมีการระบุกฎการแปลงในแต่ละเทมเพลตเพื่อแปลงส่วนย่อยของเอกสารเอ็กซ์เอ็มแอลแต่ละชนิดไปเป็นภาษาเอชทีเอ็มแอลที่สอดคล้องกัน

XSLT Stylesheet

```
<xsl:stylesheet version = '1.0'
     xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>

<xsl:template match="bold">
   <p>
     <b>
       <xsl:value-of select="."/>
     </b>
   </p>
</xsl:template>

<xsl:template match="red">
   <p style="color:red">
     <xsl:value-of select="."/>
   </p>
</xsl:template>

<xsl:template match="italic">
   <p>
     <i>
       <xsl:value-of select="."/>
     </i>
   </p>
</xsl:template>


</xsl:stylesheet>
```

XML Source

```
<source>

     <bold>Hello, world.</bold>
     <red>I am </red>
     <italic>fine.</italic>

</source>
```

Output

```
<p>
  <b>Hello, world.</b>
</p>

<p style="color:red">I am </p>

<p>
  <i>fine.</i>
</p>
```

HTML View

Hello, world.

I am

Fine.

รูปที่ 2.10 ตัวอย่างการแปลงเอกสารเอ็กซ์เอ็มแอลไปเป็นเอกสารเอชทีเอ็มแอล ด้วยภาษาเอ็กซ์เอสแอลที [20]

รูปที่ 2.11 เป็นตัวอย่างการหาค่าแฟกทอเรียลด้วยภาษาเอ็กซ์เอสแอลที โดยเทมเพลตแรกมีแอทริบิวต์ “match” เป็น “Number” เพื่อทำการแปลงแต่ละส่วนย่อย “Number” โดยในเทมเพลตแรกนี้จะมีการเรียกใช้เทมเพลตชื่อ “findFactorial” เพื่อหาค่าแฟกทอเรียล ซึ่งในเทมเพลต “findFactorial” นี้ก็ยังมีการเรียกเทมเพลตตัวเองซ้ำเพื่อหาค่าแฟกทอเรียลของตัวเลขที่น้อยกว่าตัวเลขที่มันกำลังหาค่าแฟกทอเรียลอยู่ 1 อีกด้วย

การใช้ตัวแปรและพารามิเตอร์ในภาษาเอ็กซ์เอสแอลทีมีสิ่งที่ต้องพึงระวัง คือจะทำการกำหนดค่าได้เพียงครั้งเดียวเท่านั้น ดังนั้นในการหาค่าแฟกทอเรียลในรูปที่ 2.11 จะใช้วิธีวนลูป (Loop) เพื่อหาผลคูณตั้งแต่ 1 จนถึงเลขที่ต้องการหาค่าแฟกทอเรียลไม่ได้

ในงานวิจัยนี้จะได้นำภาษาเอ็กซ์เอสแอลทีมาใช้ในการอิมพลีเมนต์กระบวนการแปลงแผนภาพสเตทชาร์ตไปเป็นกรณีทดสอบที่ต้องการ

2.2 งานวิจัยที่เกี่ยวข้อง

งานวิจัยที่เกี่ยวข้อง จะมีด้วยกัน 3 งานวิจัยดังต่อไปนี้

2.2.1 การสร้างการทดสอบจากข้อกำหนดยูเอ็มแอล (Generating tests from UML Specifications) โดย Jeff Offutt and Aynur Abdurazik [7]

งานวิจัยนี้ได้เสนอหลักการใหม่ในการสร้างกรณีทดสอบจากข้อกำหนดความต้องการโดยใช้แผนภาพสเตทชาร์ต โดยมีหลักการ 4 ระดับคือ ระดับที่ครอบคลุมทุกทรานสิชัน (Transition coverage level) ระดับที่ครอบคลุมประพจน์ (Full-predicate coverage level) ซึ่งประพจน์คือเหตุการณ์ที่เปลี่ยนแปลง ระดับที่ครอบคลุมคู่ของเหตุการณ์ที่เปลี่ยนแปลง (Transition-pair coverage level) ทำการทดสอบลำดับของการเปลี่ยนแปลงสถานะ ระดับที่มีลำดับที่สมบูรณ์ (Complete-sequence level) พิจารณาลำดับสถานะการเปลี่ยนแปลง (State transition) ซึ่งก่อให้เกิดการใช้งานที่สมบูรณ์ของระบบ แต่ว่าจำนวนของลำดับที่เป็นไปได้มีจำนวนไม่จำกัด มีมากเกินที่จะเลือกมาทุกๆ ลำดับที่สมบูรณ์ นอกจากนี้ยังมีการสร้างเครื่องมือที่เรียกว่า ยูเอ็มแอลเทสทูล (UMLTest tools) ในการตรวจสอบความเป็นไปได้ของหลักการนี้ โดยในงานวิจัยนี้ได้ทำการตรวจสอบ ระดับที่ครอบคลุมประพจน์ และ ระดับที่ครอบคลุมคู่ของเหตุการณ์ที่เปลี่ยนแปลง โดยเปรียบเทียบกับแบบครอบคลุมทุกสถานะ (State coverage) ว่าผลลัพธ์ที่ได้ หลักการไหนจะครอบคลุมข้อผิดพลาด (Fault coverage) ได้มากกว่ากัน

ผลลัพธ์ที่ได้จากงานวิจัยนี้คือ หลักการในการสร้างการทดสอบจากแผนภาพสเตทชาร์ตที่ครอบคลุม 4 แบบ และสร้างเครื่องมือในการสร้างกรณีทดสอบจากแผนภาพสเตทชาร์ต และผลการทดสอบของงานวิจัยนี้สรุปได้ว่ากรณีทดสอบที่สร้างจากการพิจารณาระดับที่ครอบคลุมประพจน์สามารถค้นพบข้อผิดพลาดได้มากที่สุด จากเหตุผลดังกล่าวผู้วิจัยได้นำเทคนิคการทดสอบแบบครอบคลุมประพจน์และครอบคลุมทรานสิชันมาใช้ในการทดสอบ เพื่อใช้ในการพิจารณาในการสร้างกรณี

ทดสอบจากแผนภาพสเตทชาร์ต ซึ่งในงานวิจัยนี้พิจารณาเฉพาะทรานสิชันแบบเอน-นาเบิลทรานสิชัน โดยมีเหตุการณ์แบบเช้จอีเวนท์เข้ามากระตุ้น โดยผู้ทำวิทยานิพนธ์ได้สนใจในกระบวนการสร้างกรณีทดสอบจากงานวิจัยนี้จึงพิจารณาในส่วนทรานสิชันแบบทรานสิชันกับตัวเอง และสถานะที่มีสถานะย่อยอยู่ภายในเพิ่มต่อจากงานวิจัยนี้เพื่อให้กรณีทดสอบที่ได้ครอบคลุมเงื่อนไขที่เกิดขึ้นมากยิ่งขึ้น

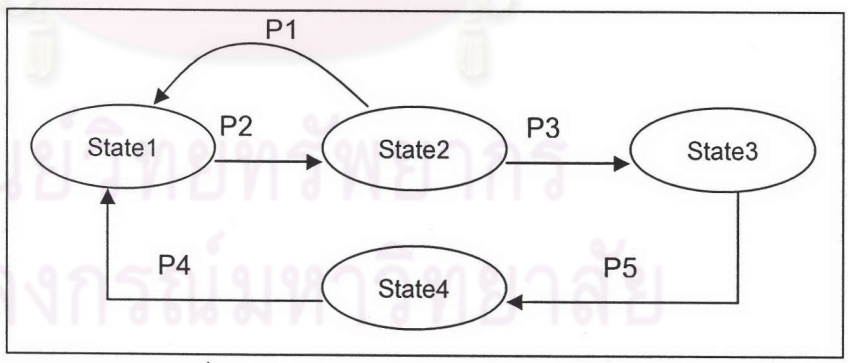
2.2.2 หลักการในการสร้างการทดสอบจากข้อกำหนดรายละเอียด (Criteria for Generating Specification-based Tests) โดย A. Jeff Offutt [23]

งานวิจัยชิ้นนี้ได้เสนอหลักการในการสร้างการทดสอบจากข้อกำหนดรายละเอียด โดยมีหลักการในการทดสอบที่แตกต่างกัน 4 หลักการ คือ

2.2.2.1 หลักการที่ครอบคลุมทรานสิชัน (Transition coverage criterion)

ผู้ทดสอบควรทดสอบทุกๆ เงื่อนไขก่อนหน้าในข้อกำหนดรายละเอียดอย่างน้อย 1 ครั้ง ซึ่งนิยามในรูปของกราฟข้อกำหนดรายละเอียด (Specification-Graph หรือ SG) แสดงดังรูปที่ 2.12 โดยทำการทดสอบทุกทรานสิชันในกราฟ

นิยาม ให้ T เป็นเซตของกรณีทดสอบ และ กราฟข้อกำหนดรายละเอียด เป็นกราฟข้อกำหนดรายละเอียด ดังนั้น Transition coverage: การทดสอบเซต T ต้องเป็นไปตามทุกทรานสิชันใน กราฟข้อกำหนดรายละเอียด



รูปที่ 2.12 กราฟข้อกำหนดรายละเอียด

จากรูปที่ 2.12 แสดงกราฟข้อกำหนดรายละเอียด โดยที่วงรีแทนชื่อสถานะ ลูกศรแทนทรานสิชันซึ่งทำการเชื่อมต่อกันระหว่างสถานะหนึ่งไปยังสถานะหนึ่ง โดยจะมีชื่อของประพจน์ประกอบอยู่บนเส้น

2.2.2.2 หลักการที่ครอบคลุมประพจน์

หลักการนี้จะทำการทดสอบแต่ละประโยค ในแต่ละประพจน์บนแต่ละทรานสิชัน จะถูกทดสอบอย่างเป็นอิสระ สมมติให้ตัวดำเนินการทางตรรกะ (Boolean operation) เป็น AND OR NOT ซึ่งนิพจน์แบบบูลลีน (Boolean expression)ประโยค และประพจน์ ถูกนิยามดังนี้

- นิพจน์แบบบูลลีน เป็นนิพจน์ (Expression) ที่มีค่าเป็นจริง (True) หรือเท็จ (False) เช่น (P∧Q) ให้มีค่าเป็นจริง เป็นต้น ซึ่งนิพจน์เป็นข้อคำสั่งที่ใช้กำหนดค่าของข้อมูล เช่น การบวกตัวเลข การเปรียบเทียบข้อมูล โดยการกำหนดชื่อของตัวแปรตามด้วยเครื่องหมายที่ต้องการกระทำต่อข้อมูลเป็นผลให้เกิดค่าข้อมูลใหม่ค่าหนึ่งให้กับตัวแปรเพื่อนำไปใช้งาน

- ประโยค เป็นนิพจน์บูลลีนที่ไม่มีตัวดำเนินการทางตรรกะ เช่น ตัวแปรของบูลลีน เช่น P Q เป็นต้น

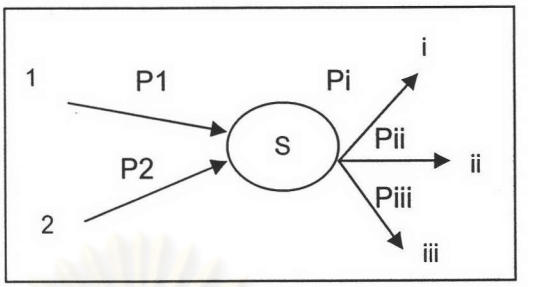
- ประพจน์ เป็นนิพจน์ทางตรรกะที่มีประโยค และมีตัวดำเนินการทางตรรกะที่มากกว่า 1 หรือไม่มีเลย ประพจน์ที่ไม่มีตัวดำเนินการทางตรรกะจะเป็นประโยค ตัวอย่างของประพจน์เช่น (P∧Q) V R เป็นต้น

นิยาม Full predicate coverage: แต่ละประพจน์ P บนแต่ละทรานสิชัน T ต้องรวมการทดสอบที่เป็นต้นเหตุให้แต่ละ ประโยค C ใน P เป็นผลลัพธ์ในคู่ของผลที่ตามมา ซึ่งค่าของ P มีความเกี่ยวข้องโดยตรง (Directly correlated) กับค่าของ C ถ้าได้ ครอบคลุมประพจน์แล้วก็จะได้ ครอบคลุมทรานสิชันด้วย

2.2.2.3 หลักการที่ครอบคลุมคู่ของทรานสิชัน (Transition-pair coverage criterion)

หลักการ 2 แบบที่ผ่านมาเป็นการทดสอบทรานสิชันแบบอิสระ แต่ไม่ได้ทดสอบลำดับของการเปลี่ยนสถานะ (Sequence of state transition) อาจมีข้อผิดพลาดบางอย่างเกิดขึ้น เช่น ข้อผิดพลาดอาจเกิดขึ้นเนื่องจากลำดับของการเปลี่ยนสถานะไม่ถูกต้องหรือลำดับที่ถูกต้องไม่ถูกทดสอบ เพื่อทำการตรวจสอบข้อผิดพลาดดังกล่าวใช้วิธีครอบคลุมคู่ของทรานสิชัน (Transition-pair-coverage) ในการพิจารณา

นิยาม Transition-pair coverage: สำหรับแต่ละคู่ของทรานสิชันที่อยู่ใกล้กัน S_i: S_j และ S_j: S_k ใน กราฟข้อกำหนดรายละเอียด ซึ่ง T ต้องมีการทดสอบที่ว่า ท่องไปยังคู่ของทรานสิชันในลำดับ พิจารณาสถานะดังรูปที่ 2.13



รูปที่ 2.13 กราฟข้อกำหนดรายละเอียดในการพิจารณาครอบคลุมคู่ของทรานสิชัน

เพื่อทำการทดสอบสถานะ S ด้วยหลักการ ครอบคลุมคู่ของทรานสิชัน จะมีการทดสอบเกิดขึ้น 6 การทดสอบที่ต้องการคือ

(1) จาก 1 → i

(2) จาก 2 → i

(3) จาก 1 → ii

(4) จาก 2 → ii

(5) จาก 1 → iii

(6) จาก 2 → iii

ซึ่งการทดสอบนี้ต้องการข้อมูลเข้าที่เป็นไปตามคู่ของประพจน์ดังนี้ (P1:Pi), (P1:Pii) (P1:Piii) (P2:Pi) (P2:Pii) และ (P2:Piii) โดย P1 และ P2 คือ ประพจน์ที่เข้าไปยังสถานะ S ส่วน Pi, Pii และ Piii คือ ประพจน์ที่เป็นไปได้ของสถานะ S ซึ่งคู่ของทรานสิชันที่ได้จะมีเท่ากับ จำนวนของทรานสิชันที่เข้าไปยังสถานะ (M) คูณด้วยจำนวนของทรานสิชันที่ออกจากสถานะ (N) ซึ่งก็คือ M * N

2.2.2.4 หลักการแบบลำดับเหตุการณ์ที่สมบูรณ์ (Complete sequence criterion)

ลำดับเหตุการณ์ที่สมบูรณ์ (Complete sequence) เป็นลำดับเหตุการณ์ที่ต่อเนื่องกันของการเปลี่ยนสถานะที่ทำให้มีการใช้งานของระบบได้สมบูรณ์ ซึ่งการที่จะเลือกลำดับเหตุการณ์ที่ต่อเนื่องกันในแผนภาพสเตทชาร์ตนั้น สามารถเลือกได้โดยจำเป็นที่จะต้องใช้ความรู้และประสบการณ์ของผู้ทดสอบเท่านั้น และ

ในความเป็นจริงแล้วจำนวนของลำดับที่เป็นไปได้จะมีจำนวนเยอะมากเกินที่จะเลือกลำดับเหตุการณ์ที่สมบูรณ์จริงๆ ซึ่งจำนวนของลำดับที่สมบูรณ์จะมีไม่จำกัด

นิยาม Complete sequence: T ต้องมีการทดสอบว่าการท่องไปยังลำดับเหตุการณ์ที่สำคัญของ ทรานสิชันบนกราฟข้อกำหนดรายละเอียด โดยการเลือกลำดับเหตุการณ์ของสถานะที่ควรจะมีการทำงานเพื่อที่จะสร้างการทดสอบตามกระบวนการที่กล่าวมาทั้ง 4 แบบ

ผลลัพธ์จากงานวิจัยนี้คือ หลักการในการทดสอบ 4 แบบ โดยจะทำการสร้างกรณีทดสอบตามหลักการแบบครอบคลุมประพจน์ หลักการแบบครอบคลุมคู่ของทรานสิชัน และการสร้างกรณีทดสอบแบบสุ่ม (Random) ผลการทดสอบสรุปได้ว่า กรณีทดสอบที่ได้จากการพิจารณาแบบหลักการครอบคลุมประพจน์สามารถหาข้อผิดพลาดที่เกิดขึ้นได้มากที่สุด นอกจากนี้กรณีทดสอบที่ได้จากการพิจารณาแบบครอบคลุมประพจน์ จะครอบคลุมไปถึงหลักการสร้างกรณีทดสอบแบบครอบคลุมทรานสิชันด้วย จากงานวิจัยนี้ผู้ทำวิทยานิพนธ์จึงได้เลือกใช้แนวความคิดในการสร้างกรณีทดสอบที่ครอบคลุมประพจน์เนื่องจากสามารถพบข้อผิดพลาดได้มากที่สุด

2.2.3 การเลือกการทดสอบจากแผนภาพสเตทชาร์ต (Test Selection from UML Statecharts) โดย Li Liuying and Qi Zhichang [24]

งานวิจัยนี้เสนอวิธีการสำหรับการทดสอบและการเลือกกรณีทดสอบจากแผนภาพยูเอ็มแอล โดยพิจารณาจากแผนภาพสเตทชาร์ต สร้างกรณีทดสอบซึ่งใช้คุณสมบัติบางส่วนในระบบแทนที่จะพิจารณาทุกคุณสมบัติที่มีอยู่ แต่สามารถรับรองได้ว่าครอบคลุมข้อผิดพลาดที่เกิดขึ้นได้อย่างสมบูรณ์ การเลือกการทดสอบจากแผนภาพสเตทชาร์ตมี 2 แบบคือ

- เลือกการทดสอบจากแผนภาพสเตทชาร์ตที่ง่ายๆ ไม่ซับซ้อน ซึ่งจะพิจารณาครอบคลุมทุกๆ สถานะและทุกๆ ทรานสิชันของสถานะ
- เลือกการทดสอบจากแผนภาพสเตทชาร์ตที่ซับซ้อนขึ้น โดยพิจารณาเรื่องของสถานะย่อยที่เกิดขึ้นและสถานะที่ทำงานพร้อมกัน

ซึ่งกรณีทดสอบที่ได้มีการพิจารณาสิ่งที่จะกำหนดให้เป็นข้อมูลเข้า ดังนี้คือ เหตุการณ์ที่เป็นตัวกระตุ้นของทุกๆ ทรานสิชัน ลำดับเหตุการณ์ของตัวกระตุ้นของแต่ละสถานะ ทรานสิชันทุกๆ ทรานสิชันทำการทดสอบอย่างน้อยหนึ่งครั้ง โดยจะทำการ

สร้างสูตรในการหากรณีทดสอบจากสิ่งที่พิจารณา หลังจากนั้นนำกรณีทดสอบที่ได้ไปเปรียบเทียบกับหลักการครอบคลุมทุกสาขา (Branch Coverage) โดยเปรียบเทียบดูว่าครอบคลุมข้อผิดพลาดที่เกิดขึ้นแค่ไหน

ผลลัพธ์ที่ได้จากงานวิจัยนี้คือ วิธีการในการเลือกการทดสอบและการสร้างกรณีทดสอบจากแผนภาพสเตทชาร์ต แต่งานวิจัยนี้ยังไม่มีการสร้างเครื่องมือสำหรับการสร้างกรณีทดสอบจากวิธีการที่คิดขึ้น