การสกัดยูเอ็มแอลซีเควนซ์ไดอะแกรมจากโมเดิร์นฟอร์แทรน
Towards Extraction of UML Sequence Diagrams from Modern Fortran

อนวัช เล่ห์ทองคำ (Anawat Leatongkam)* และ อซีส นันทอมรพงศ์ (Aziz Nanthaamornphong)*

Received : October 19, 2017
Revised : May 23, 2018
Accepted : July 18, 2018

บทคัดย่อ

ปัจจุบันวิศวกรรมย้อนกลับได้ถูกนำไปใช้อย่างแพร่หลาย โดยเป็นกระบวนการสกัดแนวคิดของระบบ และข้อมูลการออกแบบออกจากระบบ งานวิจัยนี้เกี่ยวข้องกับเครื่องมือวิศวกรรมย้อนกลับที่มีชื่อว่า ForUML ซึ่งเป็นเครื่องมือสำหรับสร้างยูเอ็มแอลไดอะแกรมขึ้นมาจากซอร์สโค้ดโมเดิร์นฟอร์แทรน ซึ่งยังคงเป็นภาษาที่มีนักพัฒนาซอฟต์แวร์ทางด้านวิทยาศาสตร์ และวิศวกรรมศาสตร์ใช้อยู่ในปัจจุบัน ในเวอร์ชันแรกของ ForUML สามารถแสดงได้เฉพาะคลาสไดอะแกรมเท่านั้น ซึ่งคลาสไดอะแกรมนั้นจะแสดงให้เห็นถึงโครงสร้าง และความสัมพันธ์ระหว่างคลาสในระบบ อย่างไรก็ตามในทางวิศวกรรมซอฟต์แวร์คลาสไดอะแกรมเพียงอย่างเดียวยังไม่เพียงพอต่อการวิเคราะห์และทำความเข้าใจระบบ โดยเฉพาะพฤติกรรม และการปฏิสัมพันธ์ระหว่างคลาสในระบบ ซึ่งข้อมูลเหล่านี้สามารถถูกแสดงได้โดยยูเอ็มแอลซีเควนซ์ไดอะแกรม ดังนั้นในงานวิจัยนี้ผู้วิจัยได้นำเสนอแนวคิดการออกแบบกฎการแปลงซอร์สโค้ดภาษาฟอร์แทรนเป็นยูเอ็มแอลซีเควนซ์ไดอะแกรม โดยกฎที่ถูกพัฒนาขึ้นนี้สามารถนำไปพัฒนาต่อยอดความสามารถของเครื่องมือ ForUML ซึ่งจะช่วยให้นักพัฒนาสามารถทำความเข้าใจระบบที่พัฒนาด้วยฟอร์แทรน อีกทั้งยังช่วยส่งเสริมการพัฒนา การบำรุงรักษา กระบวนการตัดสินใจ และการสื่อสารของชุมชนนักพัฒนาทางด้านวิทยาศาสตร์ และวิศวกรรมศาสตร์ทั่วโลก

คำสำคัญ: วิศวกรรมย้อนกลับ ฟอร์แทรน แผนภาพยูเอ็มแอลซีเควนซ์ วิศวกรรมซอฟต์แวร์

Abstract

Recently, reverse engineering has been widely adopted as a valuable process for extracting system abstractions and design information from existing software systems. This research will focus on ForUML, a reverse engineering tool developed to extract UML diagrams from modern, object-oriented Fortran code, which are still used by scientist and engineering application developers. The first version of ForUML produces only UML class diagrams, which provide a useful window into the static structure of a program, including the make-up of each class and the relationships between classes. Rather than visualizing class diagrams, the developers need to understand class behavior and interactions between classes. UML sequence diagrams provide such important algorithmic information. Therefore, we proposed to design rules for transforming object-oriented Fortran into UML sequence diagrams with the goal to extend the ability of ForUML. The proposed capability will enable the visualization of modern Fortran software behavior and algorithmic structures. This ability would enhance the development, maintenance practices, decision processes, and communications in the scientific software community worldwide.

Keywords: Reverse Engineering, Fortran, UML Sequence Diagram, Software Engineering.

* *วิทยาลัยการคอมพิวเตอร์ มหาวิทยาลัยสงขลานครินทร์ วิทยาเขตภูเก็ต*
* *College of Computing, Prince of Songkla University, Phuket Campus.*

** ส่วนหนึ่งของบทความวิจัยนี้ได้นำเสนอในงานประชุมทางวิชาการระดับชาติด้านคอมพิวเตอร์และเทคโนโลยีสารสนเทศ ครั้งที่ 13 และได้รับรางวัล Best Paper Award

1. บทนำ

ในปัจจุบันกระบวนการวิศวกรรมย้อนกลับ (Reverse Engineering) เป็นที่รู้จักกันอย่างแพร่หลายโดยเฉพาะอย่างยิ่งในกลุ่มของนักพัฒนาซอฟต์แวร์ การทำวิศวกรรมย้อนกลับในด้านวิศวกรรมซอฟต์แวร์นั้นจะเกี่ยวข้องกับการตรวจสอบและทำความเข้าใจซอร์สโค้ด ซึ่งเป็นส่วนหนึ่งของกระบวนการในการทำความเข้าใจระบบซอฟต์แวร์ (Program Comprehension) ซึ่งในกระบวนการพัฒนาซอฟต์แวร์ถ้าหากระบบซอฟต์แวร์มีขนาดใหญ่ หรือมีจำนวนบรรทัดของโค้ดมาก อาจจะก่อให้เกิดปัญหาเรื่องความซับซ้อนของระบบ จึงทำให้ยากต่อการตรวจสอบและทำความเข้าใจซอร์สโค้ด ดังนั้นกระบวนการวิศวกรรมย้อนกลับจึงได้มีส่วนช่วยให้นักพัฒนาสามารถมองเห็นถึงภาพรวมของระบบได้ง่าย เพื่อที่จะได้ทำการบำรุงรักษาและปรับปรุงซอฟต์แวร์ อย่างไรก็ตามกระบวนการวิศวกรรมย้อนกลับของระบบซอฟต์แวร์ที่มีขนาดใหญ่ หรือซับซ้อนนั้นเป็นเรื่องที่ยากและมีความท้าทายเป็นอย่างยิ่ง [1] หนึ่งในความท้าทายของกระบวนการวิศวกรรมย้อนกลับคือ การสร้างมุมมองที่สื่อความหมายของสิ่งที่เป็นนามธรรมหรือสิ่งที่ไม่มีรูปร่าง เช่น ซอร์สโค้ด เพื่อให้อยู่ในรูปแบบที่สามารถตรวจสอบและทำความเข้าใจความซับซ้อนได้ [2] เช่น ยูเอ็มแอล (Unified Modeling Language: UML) เป็นภาษาที่ใช้สำหรับอธิบายแบบจำลองต่างๆ หรือเป็นภาษาสัญลักษณ์รูปภาพมาตรฐาน ที่ช่วยในการสร้างแบบจำลองเชิงวัตถุ โดยมีแบบจำลองที่มีมุมมองหลากหลาย เช่น ยูเอ็มแอลคลาสไดอะแกรม (UML Class Diagram) ยูเอ็มแอลซีเควนซ์ไดอะแกรม (UML Sequence Diagram)

จากงานวิจัยก่อนหน้านี้ได้มีการพัฒนาเครื่องมือที่มีชื่อว่า ForUML [3] ซึ่งมีคุณสมบัติในการแปลงโค้ดภาษาฟอร์แทรนไปเป็นยูเอ็มแอลคลาสไดอะแกรม โดยยูเอ็มแอลคลาสไดอะแกรม จะเป็นแผนภาพที่แสดงให้เห็นถึงโครงสร้างการทำงานของคลาส ซึ่งจะบอกว่าแต่ละคลาสมีหน้าที่การทำงานที่สัมพันธ์กับคลาสอื่นอย่างไรในภาษาโปรแกรมเชิงวัตถุ สำหรับภาษาฟอร์แทรนนั้น ปัจจุบันได้ถูกพัฒนาเป็นภาษาโปรแกรมเชิงวัตถุคล้ายกับภาษาจาวา (Java) หรือภาษาซีพลัสพลัส (C++) ซึ่งเป็นที่นิยมใช้สำหรับการพัฒนาซอฟต์แวร์ในด้านที่เกี่ยวข้องกับวิทยาศาสตร์ และวิศวกรรมศาสตร์ (Scientific Software) เช่น การพยากรณ์อากาศ ดาราศาสตร์ และการแพทย์ ซึ่งการพัฒนาซอฟต์แวร์ทางด้านนี้ยังขาดเครื่องมือที่ดีในการพัฒนาซอฟต์แวร์ควบคู่กันไป [4] อีกทั้งการพัฒนาซอฟต์แวร์ในด้านนี้ส่วนใหญ่จะพัฒนาขึ้นด้วยวิธีการการลองผิดลองถูก และการศึกษาค้นคว้าด้วยตัวเอง เนื่องจากนักพัฒนาซอฟต์แวร์ในด้านนี้เป็นนักวิทยาศาสตร์ และวิศวกร ที่มีความรู้เพียงแค่พื้นฐานทั่วไปเกี่ยวกับการเขียนโค้ด จึงทำให้การเขียนโค้ดที่มีความซับซ้อนนั้นจำเป็นต้องใช้เวลา และมีแนวโน้มที่อาจจะก่อให้เกิดปัญหาตามมา [5]

อย่างไรก็ตามเครื่องมือ ForUML ที่ถูกพัฒนาขึ้นในเวอร์ชันแรกนั้นยังมีขีดความสามารถที่จำกัด เนื่องจากเครื่องมือนี้สามารถแสดงได้เฉพาะคลาสไดอะแกรมเพียงเท่านั้น โดยแผนภาพที่ได้จะเป็นแบบจำลองในมุมมองเชิงโครงสร้างซึ่งจะแสดงโครงสร้างของระบบที่ไม่บ่งบอกถึงพฤติกรรมการทำงาน และไม่มีการระบุขั้นตอนการดำเนินงานหรือลำดับการทำงานก่อนหลัง ดังนั้นจึงไม่เพียงพอต่อการวิเคราะห์ และทำความเข้าใจระบบ นอกจากนี้ผู้ใช้งานเครื่องมือได้ให้ข้อเสนอแนะสำหรับการเพิ่มคุณสมบัติหรือความสามารถใหม่ๆ เช่น การสร้างยูเอ็มแอลซีเควนซ์ไดอะแกรม ซึ่งเป็นแบบจำลองในมุมมองเชิงพฤติกรรมที่แสดงถึงลำดับการทำงานภายในระบบ เพื่อให้สามารถมองเห็นถึงภาพรวมของระบบที่พัฒนาขึ้นมาจากภาษาฟอร์แทรน รวมถึงสามารถช่วยในการตัดสินใจเกี่ยวกับกระบวนการพัฒนาซอฟต์แวร์และการบำรุงรักษาซอฟต์แวร์ได้ดียิ่งขึ้น

งานวิจัยนี้มีวัตถุประสงค์ที่จะนำเสนอกฎการแปลงซอร์สโค้ดภาษาฟอร์แทรนเป็นยูเอ็มแอลซีเควนซ์ไดอะแกรม ซึ่งในปัจจุบันยังไม่มีผู้คิดค้นกฎการแปลงดังกล่าว ผู้วิจัยเชื่อว่างานวิจัยนี้จะเป็นประโยชน์ในการพัฒนาต่อยอดความสามารถของเครื่องมือ ForUML ให้สามารถสร้างยูเอ็มแอลซีเควนซ์ไดอะแกรมได้ นอกจากนี้การมีแผนภาพแสดงแบบจำลองของโปรแกรมในหลายมุมมอง จะช่วยให้นักพัฒนาสามารถวิเคราะห์ และทำความเข้าใจในระบบซอฟต์แวร์ได้ดียิ่งขึ้น อีกทั้งยังมีส่วนช่วยในกระบวนการพัฒนาซอฟต์แวร์และบำรุงรักษาซอฟต์แวร์ [6]

2. ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

ในบทนี้จะเป็นการอธิบายทฤษฎีและวรรณกรรมที่เกี่ยวข้อง ได้แก่ ภาษาฟอร์แทรน วิศวกรรมย้อนกลับ ยูเอ็มแอล เมต้าโมเดล และเครื่องมือ ForUML โดยมีรายละเอียดดังต่อไปนี้

2.1 ภาษาฟอร์แทรน (Fortran)

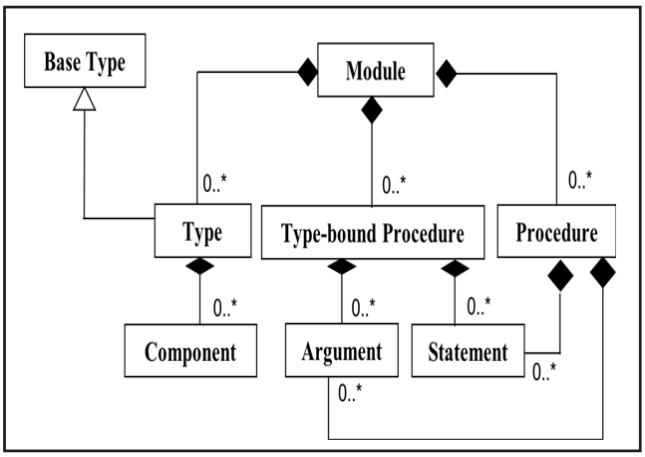
ปัจจุบันภาษาฟอร์แทรนได้ถูกพัฒนาให้มีลักษณะเชิงวัตถุ (Object-Oriented) หรือเรียกอีกอย่างว่า โมเดิร์นฟอร์แทรน (Modern Fortran) [7] เพื่อให้รองรับการพัฒนาซอฟต์แวร์ที่มีความซับซ้อน และเป็นการนำหลักการทางด้านวิศวกรรมซอฟต์แวร์มาประยุกต์ใช้ให้ซอฟต์แวร์มีประสิทธิภาพมากยิ่งขึ้น ดังจะเห็นได้ว่ามีนักวิทยาศาสตร์ และวิศวกรจำนวนมากให้ความสนใจและมีการนำโมเดิร์นฟอร์แทรนมาใช้ในการพัฒนาซอฟต์แวร์ [8], [9] นอกจากนี้ในปัจจุบันผู้ผลิตคอมไพเลอร์สำหรับภาษาฟอร์แทรนจำนวนมากได้พัฒนาให้คอมไพเลอร์มีความสามารถรองรับโมเดิร์นฟอร์แทรน เช่น Numerical Algorithm Group (NAG) และ Intel Fortran

ตารางที่ 1 *การเปรียบเทียบโครงสร้างของภาษาเชิงวัตถุระหว่างภาษาฟอร์แทรน และภาษาจาวา*

Object-Oriented Equivalent	Fortran	Java
Abstract data type (ADT)	Derived type	Class
Attribute	Component	Property
Method	Type-bound procedure	Method
Parent class	Parent type	Base Class
Child class	Extend type	Subclass
Package	Module	Package
Static polymorphism	Generic interface	Overloading
Abstract method	Deferred procedure binding	Abstract
Primitive type	Intrinsic type	Primitive type

เพื่อความเข้าใจโครงโมเดิร์นฟอร์แทรนผู้วิจัยได้แสดงตัวอย่างการเปรียบเทียบโครงสร้างของภาษาเชิงวัตถุระหว่างภาษาฟอร์แทรน และภาษาจาวา ซึ่งเป็นภาษาที่นิยมใช้ในการเขียนโปรแกรมเชิงวัตถุในปัจจุบัน ดังแสดงในตารางที่ 1 ขณะที่ภาษาฟอร์แทรนเองก็มีคุณสมบัติที่สำคัญของภาษาเชิงวัตถุ เช่น การสืบทอดคุณสมบัติ (Inheritance) การพ้องรูป (Polymorphism) การจัดสรรแบบพลวัต (Dynamic type allocation) และการกำหนดขอบเขตการทำงาน (Type-bound procedures)

จากภาพที่ 1 แสดงถึงโมเดลของภาษาฟอร์แทรน โดยจะมี Module เป็นตัวจัดเก็บในส่วนของ Type และ Procedure ซึ่งจะเทียบเท่าคลาสและเมธอดในภาษาจาวาตามลำดับ สำหรับ Type-bound Procedure นั้นจะเป็น Instances ของ Type โดยที่ Type-bound Procedure และ Procedure จะประกอบไปด้วย คำสั่งโค้ด (Statement) และ ตัวแปรของ



ภาพที่ 1 *โมเดลของภาษาฟอร์แทรน*

เมธอด (Argument หรือ Parameter) สำหรับกฎการแปลงซอร์สโค้ดภาษาฟอร์แทรนเป็นยูเอ็มแอลซีเควนซ์ไดอะแกรม ผู้วิจัยได้ทำการพิจารณาในส่วนของ Type, Type-bound Procedure, Procedure, Statement และ Argument เพื่อหาความสัมพันธ์กันระหว่างซอร์สโค้ดภาษาฟอร์แทรน และสัญลักษณ์ของยูเอ็มแอลซีเควนซ์ไดอะแกรม

เนื่องจากโมเดิร์นฟอร์แทรนยังค่อนข้างเป็นเรื่องใหม่ในโลกของการพัฒนาซอฟต์แวร์เชิงวัตถุ (Object-Oriented Programming) จึงทำให้เครื่องมือที่เกี่ยวข้องยังมีน้อย และยังไม่ได้นำหลักการทางด้านวิศวกรรมซอฟต์แวร์เข้ามาช่วยมากนัก หากเปรียบเทียบกับภาษาเชิงวัตถุอื่นๆ เช่น ภาษาจาวา หรือภาษาซีพลัสพลัส โดยเฉพาะเครื่องมือในกลุ่มของการทำความเข้าใจโปรแกรม (Program Comprehension Tool) ซึ่งเป็นเครื่องมือที่จะช่วยให้นักพัฒนา และนักออกแบบซอฟต์แวร์สามารถทำความเข้าใจโค้ด หรือระบบ

2.2 วิศวกรรมย้อนกลับ (Reverse Engineering)

วิศวกรรมย้อนกลับจะช่วยให้นักพัฒนาเข้าใจ โครงสร้างของระบบซอฟต์แวร์ที่มีขนาดใหญ่ โดยเฉพาะอย่างยิ่งกับระบบดั้งเดิมที่มีการพัฒนาต่อเนื่องมาหลายเวอร์ชัน โดยที่ระบบเหล่านี้จะมีความซับซ้อน และอาจจะไม่มีเอกสารเอกสารการออกแบบ หรือเอกสารการออกแบบเกิดสูญหาย หรือเอกสารการออกแบบไม่ได้รับการปรับปรุงให้ตรงตามความเป็นจริง ดังนั้นวิศวกรรมย้อนกลับจึงสามารถช่วยกู้คืนเอกสารการออกแบบ ทำให้นักพัฒนาสามารถนำไปเปรียบเทียบกับโครงสร้างการทำงานของระบบ เพื่อใช้ในการวิเคราะห์ ทำความเข้าใจ และปรับปรุงกระบวนการพัฒนาซอฟต์แวร์ได้ดีขึ้น

การทำวิศวกรรมย้อนกลับสำหรับซอฟต์แวร์โดยส่วนใหญ่จะเกี่ยวข้องกับซอร์สโค้ด ซึ่งจะเป็นกระบวนการวิเคราะห์ส่วนต่างๆ ของซอร์สโค้ด เพื่อทำความเข้าใจซอร์สโค้ดโดยทั่วไปมักจะใช้สำหรับการวิเคราะห์รหัสไบนารี่ (Binary Code) สำหรับตัวอย่างของซอฟต์แวร์ที่สามารถทำการวิศวกรรมย้อนกลับจากรหัสไบนารี่กลับมาอยู่ในรูปของซอร์สโค้ด (Decompile) นั้นได้แก่ Jad [10] ซึ่งเป็นซอฟต์แวร์ที่สามารถแปลงรหัสไบนารี่ของภาษาจาวา เช่น ไฟล์ที่มีนามสกุล .class เพื่อให้กลับมาอยู่ในรูปของซอร์สโค้ดที่นักพัฒนาซอฟต์แวร์สามารถอ่านและทำความเข้าใจได้

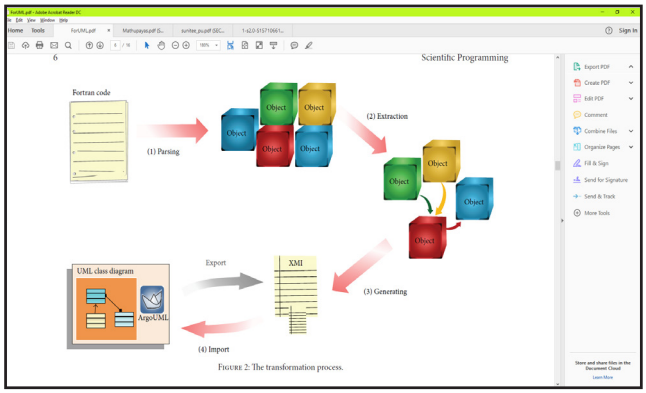
Andritsos และ Miller [11] ได้กล่าวถึงระบบซอฟต์แวร์โดยส่วนใหญ่เมื่อเริ่มมีอายุมากขึ้น การที่จะทำความเข้าใจและบำรุงรักษานั้นจึงเป็นเรื่องยาก บางครั้งอาจทำให้ระบบไม่มีประสิทธิภาพ และมีค่าใช้จ่ายเพิ่มสูงขึ้น ดังนั้นชุมชนวิศวกรซอฟต์แวร์จึงให้ความสำคัญกับการสร้างเครื่องมือที่ช่วยให้นักวิเคราะห์ระบบเพื่อที่ช่วยในการมองเห็นถึงโครงสร้างของระบบดังกล่าว

2.3 ยูเอ็มแอลเมต้าโมเดล (UML Meta Model)

ยูเอ็มแอลเป็นภาษาที่ใช้สำหรับอธิบายโมเดลต่างๆ ซึ่งเป็นมาตรฐานในการสร้างโมเดลเชิงวัตถุ หรือเป็นมาตรฐานสำหรับสร้างแบบพิมพ์เขียวให้กับระบบซอฟต์แวร์โดยสามารถใช้ยูเอ็มแอลในการสร้างมุมมองของระบบ กำหนดรายละเอียดของระบบ และพัฒนาระบบขึ้นมาได้ สำหรับเมต้าโมเดลจะเป็นโมเดลที่ใช้สำหรับกำหนดรายละเอียดของโมเดล ซึ่งจะเป็นตัวแทนแสดงถึงรายละเอียดโมเดลนั้นๆ สำหรับเครื่องมือ ForUML ได้ใช้เอกสารเอ็กซ์เอ็มไอ (XML Metadata Interchange: XMI) มาแสดงเป็นตัวแทนของยูเอ็มแอลซีเควนซ์ไดอะแกรม ยูเอ็มแอลเมต้าโมเดลมีวัตถุประสงค์เพื่อกำหนดคำจำกัดความของวากยสัมพันธ์ (Syntax) และความหมายสำหรับโครงสร้าง หรือองค์ประกอบในยูเอ็มแอลโมเดล ซึ่งเมต้าโมเดลจะช่วยให้นักพัฒนาเข้าใจถึงความหมายของโมเดลที่ต้องการจะสื่อได้ถูกต้องตรงกัน และสามารถสร้างแบบจำลองต่างๆ ได้ตรงตามมาตรฐานของยูเอ็มแอล

2.4 เครื่องมือ ForUML

ForUML [3] เป็นเครื่องมือที่สามารถช่วยเรื่องวิศวกรรมย้อนกลับจากซอร์สโค้ดที่พัฒนาขึ้นด้วยโมเดิร์นฟอร์แทรนเป็นยูเอ็มแอลไดอะแกรม ซึ่งเครื่องมือนี้ได้ถูกพัฒนาขึ้นมาโดยหนึ่งในคณะผู้วิจัยของงานวิจัยนี้ [12] และได้ถูกนำไปใช้งานจริง สำหรับขั้นตอนการแปลงซอร์สโค้ดไปเป็นยูเอ็มแอลไดอะแกรมนั้นได้มีการคิดค้นโมเดลขึ้นมาเอง ซึ่งโมเดลที่คิดค้นนี้มีรากฐานแนวคิดมาจากโครงสร้างจำลอง (Meta-Model) ของภาษาเชิงวัตถุที่ถูกพัฒนาขึ้นมาโดย Lethbridge และคณะ [13] ซึ่งทีมนักวิจัยของ Lethbridge ได้ทำการออกแบบโครงสร้างที่เรียกว่า The Dagstuhl Middle Metamodel (DMM) ซึ่งเป็นโครงสร้างจำลองที่ได้รับการพิสูจน์ในทางปฏิบัติแล้วว่าเป็นประโยชน์ต่อกระบวนการวิศวกรรมย้อนกลับซอฟต์แวร์ โดย ForUML จะประยุกต์ใช้โครงสร้างดังกล่าวกับซอร์สโค้ดของภาษาฟอร์แทรน สำหรับกระบวนการแปลงนั้นมีทั้งหมด 4 ขั้นตอนดังแสดงในภาพที่ 2



ภาพที่ 2 *กระบวนการแปลงซอร์สโค้ดภาษาฟอร์แทรนไปเป็นยูเอ็มแอลไดอะแกรมของเครื่องมือ ForUML*

จากภาพที่ 2 แสดงกระบวนการแปลงซอร์สโค้ดภาษาฟอร์แทรนไปเป็นยูเอ็มแอลไดอะแกรมซึ่งมีรายละเอียดในแต่ละขั้นตอนดังนี้

1) Parsing เป็นกระบวนการวจีวิภาคโค้ดออกเป็นส่วนย่อยๆ (Element) โดยใช้ไลบรารี่ Open Fortran Parser (OFP) ซึ่งในกระบวนการตัดคำนี้จะอาศัยไฟล์ไวย์กรณ์ (Grammar) และวากยสัมพันธ์ ของภาษาฟอร์แทรนที่ได้ถูกพัฒนาไว้แล้วในไลบรารี่ OFP โดยกระบวนการนี้จะเป็นการตรวจสอบความถูกต้องของโค้ดที่ผู้ใช้ป้อนให้กับระบบ เพื่อป้องกันไม่ให้เกิดข้อผิดพลาดในขั้นตอนถัดไป

2) Extraction เป็นการค้นหาความสัมพันธ์ระหว่างส่วนย่อยๆ ที่ได้มาจากขั้นตอนที่ 1 เช่น การสืบทอดคุณสมบัติ ความสัมพันธ์ระหว่างคลาสแบบ Aggregation หรือ Composition

3) Generating เป็นการรวบรวมเอาส่วนย่อยๆ และความสัมพันธ์ที่ค้นหาได้จากขั้นตอนที่ 1 และ 2 มาสร้างเป็น

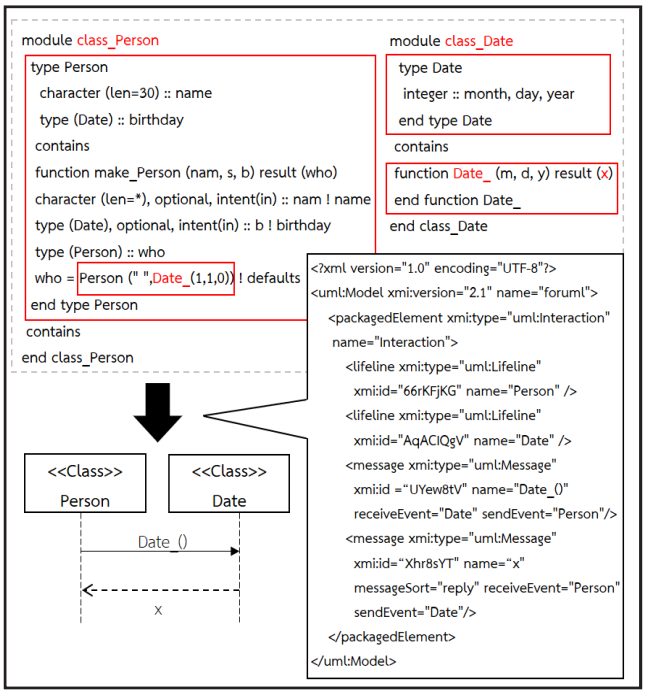
เอกสารที่อยู่ในรูปแบบเอ็กซ์เอ็มไอ ซึ่งเอกสารเอ็กซ์เอ็มไอนี้จะเก็บข้อมูลที่จำเป็นเพื่อใช้ในการแสดงยูเอ็มแอลคลาสไดอะแกรม

4) Importing ในขั้นตอนนี้ ForUML จะนำเอาเอกสารเอ็กซ์เอ็มไอ ที่ได้ในขั้นตอนที่ 3 มาแปลงเข้าไปใน ArgoUML เพื่อแสดงยูเอ็มแอลคลาสไดอะแกรม

3. วิธีดำเนินการวิจัย

การศึกษาวิจัยนี้มีวัตถุประสงค์ที่จะทำการออกแบบกฎการแปลงซอร์สโค้ดภาษาฟอร์แทรนเป็นยูเอ็มแอลซีเควนซ์ไดอะแกรม ซึ่งเป็นแนวคิดที่ได้มาจากการนำมาตรฐานของยูเอ็มแอลซีเควนซ์ไดอะแกรมตามเอกสารรายละเอียดของยูเอ็มแอล (UML Specification) [14] และกฎการแปลงยูเอ็มแอลซีเควนซ์ไดอะแกรมที่มาจากวรรณกรรมที่เกี่ยวข้อง [15], [16], [17] มาประยุกต์ใช้เพื่อสร้างกฎการแปลงซอร์สโค้ดภาษาฟอร์แทรนเป็นยูเอ็มแอลซีเควนซ์ไดอะแกรมขึ้นมาใหม่

การออกแบบกฎการแปลงซอร์สโค้ดภาษาฟอร์แทรนเป็นยูเอ็มแอลซีเควนซ์ไดอะแกรมจะเริ่มต้นจากการศึกษารายละเอียดของเอกสารเอ็กซ์เอ็มไอ ซึ่งเป็นรูปแบบมาตรฐานของ Object Management Group (OMG) ที่แสดงให้เห็นถึงรายละเอียดข้อมูลของยูเอ็มแอลโมเดล โดยจะเป็นรูปแบบเพื่อใช้สำหรับอธิบายความหมายของกฎต่างๆ ในการแปลงเป็นยูเอ็มแอลซีเควนซ์ไดอะแกรม และเพื่อใช้เป็นมาตรฐานกลางในการแลกเปลี่ยนข้อมูลระหว่างกัน สำหรับตัวอย่างเอกสารเอ็กซ์เอ็มไอที่มีข้อมูลของยูเอ็มแอลซีเควนซ์ไดอะแกรมจะแสดงในภาพที่ 3 ซึ่งจะประกอบด้วย

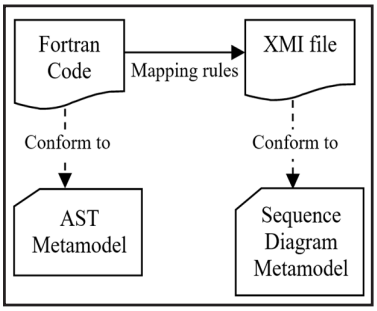


ภาพที่ 3 *ตัวอย่างของเอกสารเอ็กซ์เอ็มไอสำหรับ ยูเอ็มแอลซีเควนซ์ไดอะแกรมของโปรแกรมในภาษาฟอร์แทรน*

1) xmi:type="uml:Lifeline" จะเป็นรูปแบบที่ไว้กำหนดรายละเอียดของไลฟ์ไลน์ โดยมี xmi:id="66rKFjKG" ที่เป็นหมายเลขกำกับของไลฟ์ไลน์ และ name=Person" ที่แสดงชื่อของไลฟ์ไลน์

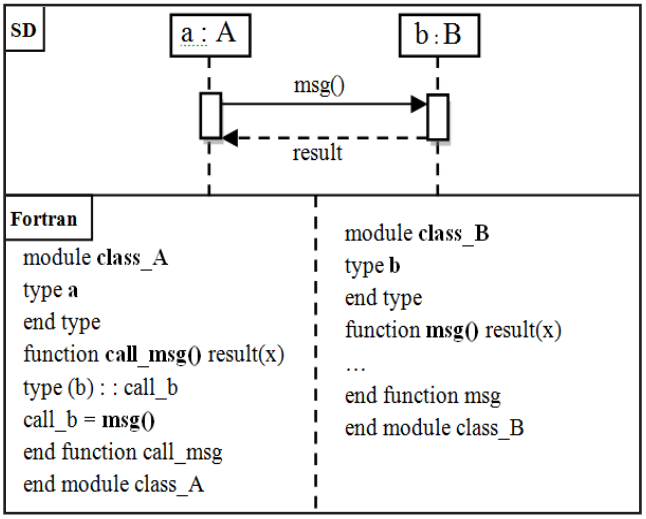
2) xmi:type="uml:Message" จะเป็นรูปแบบที่ไว้กำหนดรายละเอียดของสาร โดยมี xmi:id="Xhr8sYT" เป็นหมายเลขกำกับของสาร messageSort="reply" แสดงประเภทของสาร name="Person" แสดงชื่อของสาร receiveEvent="Person" แสดงไลฟ์ไลน์ที่รับสาร และ sendEvent="Date" แสดงไลฟ์ไลน์ที่ส่งสาร

สำหรับการออกแบบกฎการแปลงโมเดิร์นฟอร์แทรนเป็นยูเอ็มแอลซีเควนซ์ไดอะแกรมนั้นจะมีขั้นตอนหลักคือการหาความสัมพันธ์ระหว่าง Abstract Syntax Tree (AST) Metamodel ของภาษาฟอร์แทรนและเอกสารเอ็กซ์เอ็มไอ โดยเมต้าโมเดลของทั้งสองจะเป็นโมเดลที่ใช้เป็นตัวแทนของโมเดลหลักดังแสดงในภาพที่ 4



ภาพที่ 4 *กระบวนการแปลงโมเดิร์นฟอร์แทรนเป็นยูเอ็มแอลซีเควนซ์ไดอะแกรม*

ในส่วนของกฎการแปลงซอร์สโค้ดภาษาฟอร์แทรนเป็นยูเอ็มแอลซีเควนซ์ไดอะแกรม เพื่อให้ง่ายต่อการทำความเข้าใจ ผู้วิจัยจะใช้ ภาพที่ 5 สำหรับอธิบายกฎการแปลงที่สร้างขึ้นมา โดยการสร้างกฎการแปลงนั้นผู้วิจัยได้เลือกใช้ภาษา Atlas Transformation Language (ATL) ซึ่งเป็นภาษาที่นิยมใช้สำหรับการแปลงแบบจำลอง โดยผู้วิจัยได้นำส่วนที่สำคัญของกฎการแปลงที่ได้ทำการออกแบบมาแสดงดังนี้



ภาพที่ 5 *ตัวอย่างใช้อธิบายกฎการแปลงซอร์สโค้ดภาษาฟอร์แทรนเป็นยูเอ็มแอลซีเควนซ์ไดอะแกรม*

1) กฎสำหรับการสร้างไลฟ์ไลน์ เป็นกฎที่ใช้ในการเชื่อมโยงระหว่างไลฟ์ไลน์ของยูเอ็มแอลซีเควนซ์ไดอะแกรมและชื่อคลาสของซอร์สโค้ดภาษาฟอร์แทรน แสดงในภาพที่ 6

```
(SD) Lifeline name = (Fortran) Type name
Ex: Lifeline a = Type a
-----------------------------------------
rule Type2Lifeline {
from
    f: Fortran!Type
 to
    l: SD!Lifeline (name <- f.name)
}
```

ภาพที่ 6 *กฎสำหรับการสร้างไลฟ์ไลน์*

จากภาพที่ 6 เป็นการแปลงคลาส Type ของเมต้าโมเดล AST จากซอร์สโค้ดภาษาฟอร์แทรน ไปสู่ Lifeline Node ของเมต้าโมเดลซีเควนซ์ไดอะแกรม โดยมีแอตทริบิวท์ Name เป็นตัวกำหนดชื่อของไลฟ์ไลน์ให้ตรงกับชื่อของคลาสในซอร์สโค้ดภาษาฟอร์แทรน

2) กฎสำหรับการสร้างสารระหว่างไลฟ์ไลน์ เป็นกฎที่ใช้ในการเชื่อมโยงระหว่างสารของยูเอ็มแอลซีเควนซ์ไดอะแกรมและชื่อเมธอดของภาษาฟอร์แทรน แสดงในภาพที่ 7

```
(SD) Lifeline name = (Fortran) Function/ Subroutine name
Ex: Message msg = Function msg
-----------------------------------------
rule Procedures2Message {
from
    f: Fortran!Procedures
to
    m1: SD!SynchronousMessage(name <- f.FunctionName,
        messageSort  <- f. SynchronousMsg),
    m2: SD!SynchronousMessage(name <- f.SubroutineName,
        messageSort  <- f. SynchronousMsg),
    m3: SD!AsynchronousMessage(name <- f.SubroutineName,
        messageSort  <- f. AsynchronousMsg),
    m4: SD!CreateMessage(name <- f.ObjectName,
        messageSort  <- f. CreateMsg),
    m5: SD!ReplyMessage(name <- f. FunctionReplyName,
        messageSort  <- f. ReplyMsg),
    m6: SD!ReplyMessage(name <- f. SubroutineReplyName,
        messageSort  <- f. ReplyMsg)
}
```

ภาพที่ 7 *กฎสำหรับการสร้างไลฟ์ไลน์ของสารแบบ Synchronous Asynchronous Create และ Reply*

จากภาพที่ 7 เป็นการแปลงคลาส Procedures ของเมต้าโมเดล AST จากซอร์สโค้ดภาษาฟอร์แทรน ไปสู่ Message Node ของเมต้าโมเดลซีเควนซ์ไดอะแกรม โดยมีการแบ่งสารออกเป็น 4 แบบด้วยกันคือ 1) Synchronous 2) Asynchronous 3) Create และ 4) Reply ซึ่งจะมีแอตทริบิวท์ Name เป็นตัวกำหนดชื่อของสารให้ตรงกับชื่อของเมธอดในซอร์สโค้ดภาษาฟอร์แทรน และจะมีแอตทริบิวท์ messageSort เป็นตัวกำหนดชนิดของสารแต่ละชนิด

3) กฎสำหรับกำหนดการรับและส่งของสารที่เกิดขึ้น เป็นกฎที่ใช้ในการกำหนดการรับสารของไลฟ์ไลน์ และการส่งสารของไลฟ์ไลน์ แสดงในภาพที่ 8

4) กฎสำหรับกำหนดการเริ่มต้นและสิ้นสุดการดำเนินงานของสารบนไลฟ์ไลน์ที่เกิดขึ้น เป็นกฎที่ใช้ในการกำหนดจุดเริ่มต้นของสารที่ส่งไปยังไลฟ์ไลน์ และจุดสิ้นสุดของสารบนไลฟ์ไลน์ แสดงในภาพที่ 8

5) กฎสำหรับกำหนดการดำเนินงานของสารบนไลฟ์ไลน์ เป็นกฎที่ใช้ในการกำหนดการดำเนินงานที่มีการเกิดขึ้นบนไลฟ์ไลน์นั้น แสดงในภาพที่ 8

```
Lifeline name {[Maping name] !/?(send/receive) [Message name];}
Ex: Lifeline a {ab_msg!msg; ab_result?result;}
    Lifeline b {ab_msg?msg; ab_result!result;}
-----------------------------------------------------------------
rule Interaction {
from
      f: Fortran!Operation
to
      s: SD!SendOperationEven( id<- f.MessageSendOrder+1,
         covered<- f.TypeName, message<-f.ProceduresName),
     r: SD!ReceiveOperationEven(id<- f.MessageReceiveOrder+1,
         covered <- f.TypeName, message<-f.ProceduresName),
   m: SD!MessageOccurrence (name <- f.ProceduresName,
        send <- s, receive <- r),
    e: SD!ExecuteOccurrence (id<- f.LifelineExecuteOrder+1,
        covered <- f.TypeName),
  se: SD!SendExecuteSpecification(id<- f.SendExecuteOrder+1,
        covered <- f.TypeName, start <- s, finish <- e),
  re: SD!ReceiveExecuteSpecification(
       id<- f. ReceiveExecuteOrder+1,
       covered <- f.TypeName, start <- r, finish <- e)
}
```

ภาพที่ 8 *กฎสำหรับการสื่อสารกันระหว่างสารและไลฟ์ไลน์*

จากภาพที่ 8 เป็นการแปลงคลาส Operation ของเมต้าโมเดล AST จากซอร์สโค้ดภาษาฟอร์แทรน ไปสู่ Message Occurrence Node, Execution Occurrence Node และ Execution Specification Node ของเมต้าโมเดลซีเควนซ์ไดอะแกรม โดยในส่วนของ Message Occurrence Node จะเป็นการกำหนดการรับและส่งของสารที่เกิดขึ้นบนไลฟ์ไลน์ โดยจะมีแอตทริบิวท์ Name เป็นตัวกำหนดชื่อของสาร แอตทริบิวท์ Send เป็นตัวอ้างอิงถึง SendOperationEven ที่เป็นการกำหนดการส่งของสารบนไลฟ์ไลน์ และแอตทริบิวท์ Receive เป็นตัวอ้างอิงถึง ReceiveOperationEven ที่เป็นการกำหนดการรับของสารบนไลฟ์ไลน์ สำหรับในส่วนของ Execution Occurrence Node จะเป็นการกำหนดการดำเนินงานของสารบนไลฟ์ไลน์ โดยจะมีแอตทริบิวท์ Covered เป็นตัวกำหนดชื่อของไลฟ์ไลน์ที่มีการดำเนินงาน สำหรับในส่วนของ Execution Specification Node จะเป็นการกำหนดการเริ่มต้นและสิ้นสุดของสารบนไลฟ์ไลน์นั้น โดยแบ่งออกเป็น 2 ส่วนคือ ส่วนที่กำหนดการรับของสาร และส่วนที่กำหนดการส่งของสาร โดยจะมีแอตทริบิวท์ Start เป็นตัวอ้างอิงถึง SendOperationEven, ReceiveOperationEven และแอตทริบิวท์ Finish เป็นตัวอ้างอิงถึง ExecuteOccurrence เพื่อกำหนดการดำเนินงานของสารบนไลฟ์ไลน์นั้น

6) กฎสำหรับการสร้างกรอบของแผนภาพ เพื่อแสดงเงื่อนไข ทางเลือกที่มีหลายเงื่อนไข และการทำซ้ำ เป็นกฎที่ใช้ในการกำหนดกรอบของแผนภาพที่เกิดขึ้น แสดงในภาพที่ 9

```
rule Alt {
from
     f: Fortran!Statement(f.Isif_else)
 to
     a: SD!CombinedFragment( interactionOperator <- 'Alt')
}
```

ภาพที่ 9 *กฎสำหรับการการสร้างกรอบของแผนภาพในส่วนที่แสดงเงื่อนไข*

จากภาพที่ 9 เป็นการแปลงคลาส Statement ของเมต้าโมเดล AST จากซอร์สโค้ดภาษาฟอร์แทรน ไปสู่ Combined Fragment Node ของเมต้าโมเดลซีเควนซ์ไดอะแกรม โดยในส่วนของ Statement นั้นจะมีการตรวจสอบว่าเป็น Combined Fragment ลักษณะใด เช่นเงื่อนไข หรือการทำซ้ำ จากนั้นจึงทำการกำหนดชื่อของ Combined Fragment นั้นๆ

4. ผลการดำเนินงานวิจัย

จากการออกแบบกฎการแปลงซอร์สโค้ดภาษาฟอร์แทรนเป็นยูเอ็มแอลซีเควนซ์ไดอะแกรมข้างต้นผู้วิจัยได้นำผลลัพธ์เสนอแก่ผู้เชี่ยวชาญด้านภาษาโมเดิร์นฟอร์แทรน ซึ่งเป็นผู้ก่อตั้งสถาบันที่เปิดให้การอบรม รวมถึงที่ปรึกษาทางด้านโมเดิร์นฟอร์แทรน และการพัฒนาซอฟต์แวร์ด้านวิทยาศาสตร์ [10] ซึ่งมีที่ตั้งอยู่ในประเทศสหรัฐอเมริกา ผลจากการตรวจสอบโดยผู้เชี่ยวชาญนั้นพบว่ากฎที่ได้การออกแบบนั้นมีความถูกต้อง และเพื่อตรวจสอบถึงความถูกต้องของกฎการแปลง นอกจากนี้ผู้วิจัยได้ให้ผู้เชี่ยวชาญพิจารณาข้อมูลเปรียบเทียบกับภาษาเชิงวัตถุภาษาอื่น ได้แก่ การเปรียบเทียบวากยสัมพันธ์ของภาษาเชิงวัตถุ เพื่อหาความสัมพันธ์ที่มีความคล้ายคลึงกันในการแสดงสัญลักษณ์แต่ละสัญลักษณ์ของยูเอ็มแอลซีเควนซ์ไดอะแกรม ซึ่งผู้วิจัยได้ทำการเปรียบเทียบกับภาษาจาวา ซึ่งเป็นภาษาโปรแกรม เชิงวัตถุที่นิยมในปัจจุบัน ดังแสดงในตารางที่ 2 โดยแบ่งออกเป็น 2 ส่วนตามลักษณะการทำงานของสัญลักษณ์ประกอบด้วย

1) Lifeline แสดงถึงตัวแทนของคลาส ประกอบด้วยชื่อของอินสแตนซ์ (Instance) และชื่อของคลาส

2) Messages แสดงถึงสารที่ส่งระหว่างไลฟ์ไลน์ประกอบด้วย

สารที่ใช้สร้างวัตถุ สารที่ใช้ตอบกลับ สารแบบซินโครนัส และสารแบบอะซิงโครนัส

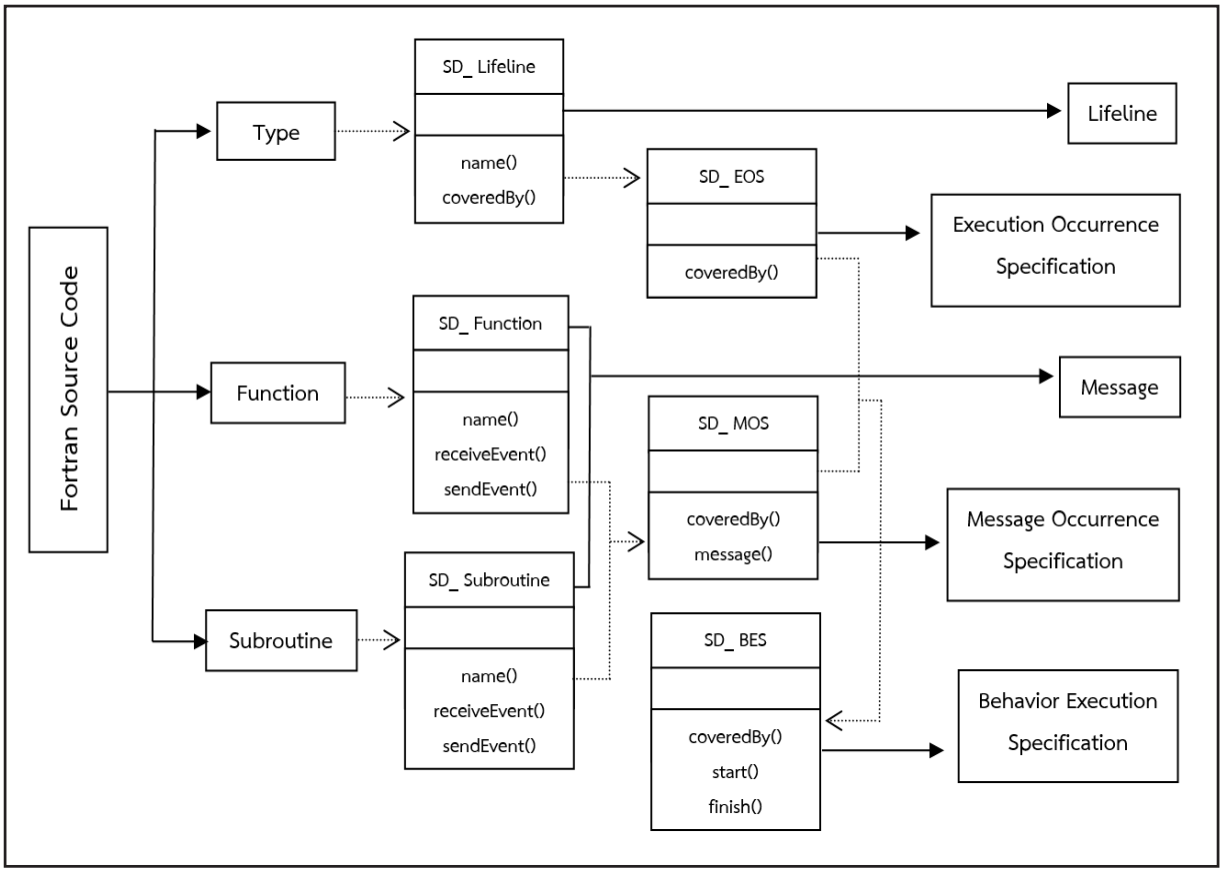
สำหรับในส่วน Interaction Fragment ซึ่งเป็นส่วนที่สื่อสารกันระหว่างสารและไลฟ์ไลน์ ประกอบด้วย การรับและส่งของสารที่เกิดขึ้น การเริ่มต้นและสิ้นสุดการดำเนินงานของสารบนไลฟ์ไลน์ที่เกิดขึ้น ข้อกำหนดการดำเนินงานของสารบนไลฟ์ไลน์ และกรอบของแผนภาพ สามารถแสดงสัญลักษณ์ที่เหมือนกับในภาษาจาวาได้ดังตารางที่ 3

เพื่อเป็นการตรวจสอบว่ากฎการแปลงที่ได้สร้างขึ้นสามารถนำไปประยุกต์ใช้กับภาษาฟอร์แทรนได้จริง ผู้วิจัยได้ทำการพัฒนาโปรแกรมขึ้นมาใช้ในการทดสอบ โดยประยุกต์ใช้กฎการแปลงที่ได้ทำการออกแบบไว้กับซอร์สโค้ดในภาษาฟอร์แทรน โดยโปรแกรมที่พัฒนาขึ้นมาเพื่อใช้สำหรับหาความสัมพันธ์ระหว่างซอร์สโค้ดภาษาฟอร์แทรนและเอกสารเอ็กซ์เอ็มไอนั้นจะมีการแยกส่วนประกอบต่างๆ ของซอร์สโค้ดอยู่ในรูปของโครงสร้าง AST โดยจะมีการใช้ไลบรารี่ OFP ในการแยกส่วนประกอบของซอร์สโค้ด จากนั้นจึงนำส่วนประกอบที่ได้มาหาความสัมพันธ์กันเพื่อสร้างเอกสารเอ็กซ์เอ็มไอ ซึ่งการสร้างเอกสารเอ็กซ์เอ็มไอจำเป็นต้องมีการกำหนดรายละเอียดที่สำคัญ 5 ส่วนด้วยกัน ตามมาตรฐานของ OMG ได้แก่ 1) Lifeline 2) Message 3) Message Occurrence Specification 4) Execution Occurrence Specification และ 5) Behavior Execution Specification โดยผู้วิจัยได้พัฒนาไลบรารีเพื่อหาความสัมพันธ์ระหว่างซอร์สโค้ดภาษาฟอร์แทรนและแผนภาพไดอะแกรม ซึ่งมีโครงสร้างและรายละเอียดของคลาสดังแสดงในภาพที่ 10 ซึ่งแต่ละคลาสจะมีหน้าที่การทำงานดังนี้

1) Type เป็นข้อมูลชื่อของคลาสที่ได้มาจากซอร์สโค้ดภาษาฟอร์แทรน ซึ่งเป็นข้อมูลที่ใช้สำหรับกำหนดรายละเอียดที่เกี่ยวข้องกับไลฟ์ไลน์

2) Function เป็นข้อมูลชื่อของเมธอดที่ได้มาจากซอร์สโค้ดภาษาฟอร์แทรน ซึ่งเป็นข้อมูลที่ใช้สำหรับกำหนดรายละเอียดที่เกี่ยวข้องกับสารที่เป็นฟังก์ชัน

3) Subroutine เป็นข้อมูลชื่อของเมธอดที่ได้มาจากซอร์สโค้ดภาษาฟอร์แทรน ซึ่งเป็นข้อมูลที่ใช้สำหรับกำหนดรายละเอียดที่เกี่ยวข้องกับสารที่เป็นซับรูทีน



ภาพที่ 10 *ตัวอย่างของเอกสารเอ็กซ์เอ็มไอสำหรับ ยูเอ็มแอลซีเควนซ์ไดอะแกรมของโปรแกรมในภาษาฟอร์แทรน*

4) SD_ Lifeline ทำหน้าที่ในการรับข้อมูลชื่อของ Type จากเอกสารซอร์สโค้ดภาษาฟอร์แทรนเพื่อใช้ในการกำหนดรายละเอียดของไลฟ์ไลน์ขึ้นมา มีการกำหนดการทำงานที่เกิดขึ้นครอบคลุมบนไลฟ์ไลน์นั้นๆ และมีการเชื่อมโยงกับคลาส SD_EOS เพื่อกำหนดการทำงานที่มีการสิ้นสุดบนไลฟ์ไลน์นั้น

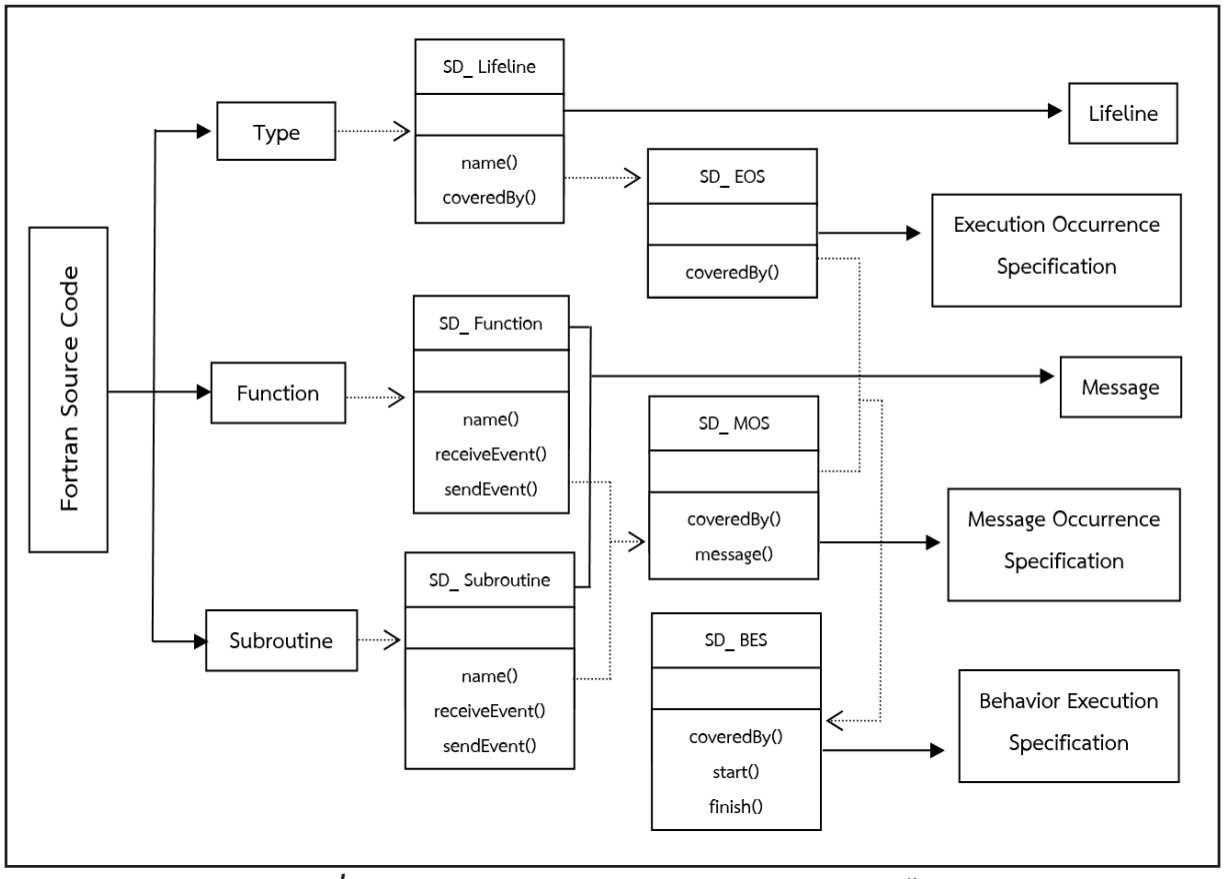
5) SD_Function ทำหน้าที่ในการรับชื่อของ Function จากเอกสารซอร์สโค้ดภาษาฟอร์แทรนเพื่อใช้ในการกำหนดรายละเอียดของสารขึ้นมา มีการกำหนดรายละเอียดการเรียกใช้งานของ Function เพื่อกำหนดการรับและส่งของสารที่เกิดขึ้น และมีการเชื่อมโยงกับคลาส SD_MOS เพื่อกำหนดว่าสารดังกล่าวอยู่บนไลฟ์ไลน์ใด

6) SD_Subroutine ทำหน้าที่ในการรับชื่อของ Subroutine จากเอกสารซอร์สโค้ดภาษาฟอร์แทรนเพื่อใช้ในการกำหนดรายละเอียดของสารขึ้นมา มีการกำหนดรายละเอียดการเรียกใช้งานของ Subroutine เพื่อกำหนดการรับและส่งของสารที่เกิดขึ้น และมีการเชื่อมโยงกับคลาส SD_MOS เพื่อกำหนดว่าสารดังกล่าวอยู่บนไลฟ์ไลน์ใด

7) SD_EOS ทำหน้าที่กำหนดจุดสิ้นสุดการทำงานเกิดขึ้นตรงไลฟ์ไลน์ใด กำหนดโดยการตรวจสอบจากการเรียกใช้งานของ Function หรือ Subroutine ถ้าหาก Function หรือ Subroutine ไม่มีการเรียกใช้งานไปยัง Type อื่นแสดงว่า Function หรือ Subroutine นั้นมีการสิ้นสุดการทำงาน และมีการเชื่อมโยงกับคลาส SD_BES เพื่อกำหนดพฤติกรรมที่เกิดขึ้นของสารบนไลฟ์ไลน์

8) SD_MOS ทำหน้าที่กำหนดการทำงานของสารว่ามีการเกิดขึ้นตรงไลฟ์ไลน์ใด กำหนดโดยการตรวจสอบ Function หรือ Subroutine มีการทำงานเกิดขึ้นตรง Type ใดในเอกสารเอกสารซอร์สโค้ดภาษาฟอร์แทรน และมี การเชื่อมโยงกับคลาส SD_BES เพื่อกำหนดพฤติกรรมที่เกิดขึ้นของสารบนไลฟ์ไลน์

9) SD_BES ทำหน้าที่กำหนดพฤติกรรมการทำงานระหว่างสารและไลฟ์ไลน์ กำหนดโดยการตรวจสอบการเรียกใช้งานของ Function หรือ Subroutine ระหว่าง Type ถ้าหากมีการเรียกใช้งานเกิดขึ้นจะทำการกำหนดจุดเริ่มต้นของไลฟ์ไลน์ที่มีการเรียกใช้งานและจุดสิ้นสุดของไลฟ์ไลน์ที่ถูกใช้งาน



ภาพที่ 11 *ความสัมพันธ์ส่วนของการสร้างเอกสารเอ็กซ์เอ็มไอ*

โดยรับข้อมูลมาจากคลาส SD_MOS และถ้าหาก Function หรือ Subroutine ไม่มีการเรียกใช้งานต่อจะทำการกำหนดจุดสิ้นสุดการทำงานโดยรับข้อมูลมาจากคลาส SD_EOS

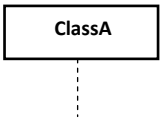
การสร้างเอกสารเอ็กซ์เอ็มไอเป็นขั้นตอนหลังจากการหาความสัมพันธ์ โดยหลังจากได้ความสัมพันธ์ที่เป็นไปตามข้อกำหนดในการสร้างเอกสารเอ็กซ์เอ็มไอแล้วจะนำความสัมพันธ์ที่ได้มาสร้างเอกสารเอ็กซ์เอ็มไอ ผู้วิจัยได้พัฒนาไลบรารีเพื่อสร้างเอกสารเอ็กซ์เอ็มไอที่มีโครงสร้างดังแสดงใน ภาพที่ 11 โดยมีรายละเอียดดังนี้

1) ParserProcessor ทำหน้าที่ในการประมวลผลการสร้างเอกสารเอ็กซ์เอ็มไอ โดยมีการตรวจสอบเอกสารซอร์สโค้ดภาษาฟอร์แทรนที่รับเข้ามาเป็นไฟล์ชนิด .F90 มีส่วนประกอบของ Module และไม่มีส่วนของซอร์สโค้ดที่ผิดพลาดจากนั้นจะส่งเอกสารที่ผ่านการตรวจสอบแล้วไปแยกส่วนประกอบต่างๆ ของซอร์สโค้ด โดยมีไลบรารี่ OFP ในการแยกส่วนประกอบ เมื่อแยกส่วนประกอบเสร็จแล้วจะทำการหาความสัมพันธ์ดังที่อธิบายไปในขั้นตอนการหาความสัมพันธ์ เมื่อได้ความสัมพันธ์แล้วจะทำการส่งรายละเอียดที่ได้กลับมาเพื่อทำการสร้างเอกสารเอ็กซ์เอ็มไอโดยมีการเชื่อมโยงกับคลาส Generator

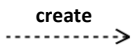
2) Generator ทำหน้าที่ในการสร้างเอกสารเอ็กซ์เอ็มไอโดยมีข้อกำหนดของเอกสารเอ็กซ์เอ็มไอตามมาตรฐานของ OMG เพื่อแสดงแผนภาพแผนภาพยูเอ็มแอลซีเควนซ์ไดอะแกรมเวอร์ชัน 2.0

ตารางที่ 2 *เปรียบเทียบการแปลงซอร์สโค้ดเป็นยูเอ็มแอลซีเควนซ์ไดอะแกรมระหว่างภาษาจาวาและฟอร์แทรน*

Rule	Java Syntax	Modern Fortran Syntax	Notations

Lifeline	package class_A;
	public class **ClassA** {
	…statement
	}
		module class_A
		type **ClassA**
		…
		end type ClassA
		… statement
		end module class_A
			

Messages			

Create Message	public class ClassB {
	ClassA a = new **ClassA**();
	}
		type ClassB
		type(**ClassA**) :: **a**
		end type ClassB
			

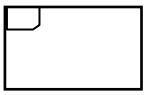
Reply Message	public class ClassC {
	public int getId(x) {
	…
	return id;
	}
	}
		type ClassC
		end type ClassC
		function getId(x) **result(id)**
		…
		end function getId	

Synchronous Message	public class ClassA {
	public void setA(..) {
	}
	}
	
	public class ClassB {
	public void callA(..) {
	a.setA(..);
	}
	}
		type ClassA
		end type ClassA
		subroutine setA (..)
		end subroutine setA
		
		type ClassB
		end type ClassB
		subroutine callA (..)
		call setA(..)
		end subroutine callA
			

Asynchronous Message	public class AsyncClassA {
	public void setA(..) {
	}
	}
	
	public class ClassB {
	public void callA(..) {
	a.setA(..);
	}
	}
		type AsyncClassA
		asynchronous :: a
		end type ClassA
		subroutine **setA** (..)
		end subroutine setA
		
		type ClassB
		end type ClassB
		subroutine callA (..)
		call setA(..)
		end subroutine callA
			

โดยในการทดสอบกับภาษาฟอร์แทรน ผู้วิจัยนำโค้ดมาจากงานของ Akin [18] เพื่อสร้างเอกสารเอ็กซ์เอ็มไอ โดยเอกสารเอ็กซ์เอ็มไอที่ได้จะสอดคล้องกับสัญลักษณ์ของภาษายูเอ็มแอลดังแสดงในตารางที่ 2 และตารางที่ 3 โดยซอร์สโค้ดภาษาฟอร์แทรนจะถูกแสดงดังภาพที่ 12, 13, 14 และ 15 และเอกสารเอ็กซ์เอ็มไอที่ได้จะถูกแสดงดังภาพที่ 16

ตารางที่ 3 *สัญลักษณ์ของยูเอ็มแอลในส่วน Interaction Fragment*

Rule	Fortran Semantics	Notations
Interaction Fragment		
Message Occurrence	Send and Receive Occurrence	
Execution Specification	Start and Finish Occurrence	
Execution Occurrence	Activation	
Combined Fragment	Loops, Branches, and Other Alternatives	

```
include 'class_Date.f90'
include 'class_Person.f90'
include 'class_Student.f90' ! see previous figure
program main ! create or correct a student
    use class_Student ! inherits class_Person, class_Date also
    type (Person) :: p ; type (Student) :: x
    ! Method 1
    p = make_Person ("Ann Jones","",0) ! optional person constructor
    x = Student_(p, "219360061", Date_(8,29,1955), 9, 3.1) ! public
    call set_DOB (p, 5, 13, 1977) ! add birth to person data
    call print_Name (p) ! list name
    print *, "Born :"; call print_DOB (p) ! list dob
    print *, "Sex :"; call print_Sex (p) ! list sex
    print *, "Matriculated:"; call print_DOM (x) ! list dom
    call print_GPA (x) ! list gpa
end program main
```

ภาพที่ 12 *กฎสำหรับการการสร้างกรอบของแผนภาพในส่วนที่แสดงเงื่อนไข*

จากภาพที่ 12 แสดงตัวอย่างของซอร์สโค้ดภาษาฟอร์แทรนในส่วนของโปรแกรมหลัก ซึ่งจะมีการเรียกใช้งานไฟล์ class_Date.f90 class_Person.f90 และ class_Student.f90 โดยทั้ง 3 ไฟล์นั้นจะเป็นโมดูลย่อยไว้สำหรับให้โปรแกรมหลักกำหนดการทำงาน ซึ่งโปรแกรมหลักจะประกอบด้วยส่วนที่เรียกใช้งานฟังก์ชันคือ p = make_Person และ x = Student_ ส่วนที่เรียกใช้งานซับรูทีนคือ call set_DOB call print_Name call print_DOB call print_Sex call print_DOM และ call print_GPA

```
module class_Student ! filename class_Student.f90
 use class_Person ! inherits class_Date
 public :: Student, set_DOM, print_DOM
 type Student
  private
  type (Person) :: who ; character (len=9) :: id
  type (Date) :: dom; integer :: credits;  real :: gpa
 end type Student
 contains ! coupled functionality
 function get_person (s) result (p)
  type (Student), intent(in) :: s ; type (Person) :: p ! name and sex
  p = s % who ; end function get_person
 ! Optional Constructor for a Student type
 function make_Student (w, n, d, c, g) result (x)
  type (Person), intent(in) :: w ! who
  character (len=*), optional, intent(in) :: n ! ssn
  type (Date), optional, intent(in) :: d ! matriculation
  integer, optional, intent(in) :: c ! credits
  real, optional, intent(in) :: g ! grade point ave
  type (Student) :: x ! new student
  x = Student_(w, " ", Date_(1,1,1), 0, 0.) ! defaults
  if ( present(n) ) x % id = n
  if ( present(d) ) x % dom = d ! optional values
  if ( present(c) ) x % credits = c
  if ( present(g) ) x % gpa = g; end function make_Student
subroutine print_DOM (who)
  type (Student), intent(in) :: who
  call print_Date(who%dom) ; end subroutine print_DOM
 subroutine print_GPA (x)
  type (Student), intent(in) :: x
  print *,"My name is "; call print_Name (x % who)
  print *,", and my G.P.A. is ", x % gpa, "."; end subroutine
 subroutine set_DOM (who, m, d, y)
  type (Student), intent(inout) :: who ; integer, intent(in) :: m, d, y
  who % dom = Date_( m, d, y) ; end subroutine set_DOM
! Public Constructor for a Student type
 function Student_ (w, n, d, c, g) result (x)
  type (Person), intent(in) :: w ;   character (len=*), intent(in) :: n
  type (Date), intent(in) :: d
  integer, intent(in) :: c; real, intent(in) :: g
  type (Student) :: x ! new student
  x = Student (w, n, d, c, g) ; end function Student_
 end module class_Student
```

ภาพที่ 13 *ตัวอย่างของซอร์สโค้ดภาษาฟอร์แทรนในส่วนของ class_ Student*

จากภาพที่ 13 แสดงตัวอย่างของซอร์สโค้ดภาษาฟอร์แทรนในส่วนของ class_Student ซึ่งจะมีการกำหนดชื่อของคลาสคือ Student และมีส่วนที่เป็นฟังก์ชันคือ get_person make_Student และ Student_ ส่วนที่เป็นซับรูทีนคือ print_DOM

```
module class_Person ! filename: class_Person.f90
use class_Date
public :: Person
type Person
private
 character (len=20) :: name; character (len=20) :: nationality
 integer :: sex
type (Date) :: dob, dod ! birth, death
end type Person
 contains
! Optional Constructor for a Person type
function make_Person (nam, nation, s, b, d) result (who)
 character (len=*), optional, intent(in) :: nam, nation
 integer, optional, intent(in) :: s ! sex
 type (Date), optional, intent(in) :: b, d ! birth, death
 type (Person) :: who
 who = Person (" ","USA",1,Date_(1,1,0),Date_(1,1,0))! defaults
 if ( present(nam) ) who % name = nam;
 if ( present(nation) ) who % nationality = nation
 if ( present(s) ) who % sex = s
 if ( present(b) ) who % dob = b
 if ( present(d) ) who % dod = d ; end function
! Public Constructor for a Person type
function Person_ (nam, nation, s, b, d) result (who)
 character (len=*), intent(in) :: nam, nation
 integer, intent(in) :: s ! sex
 type (Date), intent(in) :: b, d ! birth, death
 type (Person) :: who
 who = Person (nam, nation, s, b, d) ; end function Person_
subroutine print_DOB (who)
 type (Person), intent(out) :: who
 call print_Date (who % dob) ; end subroutine print_DOB
subroutine print_DOD (who)
 type (Person), intent(inout) :: who
 call print_Date (who % dod) ; end subroutine print_DOD
subroutine print_Name (who)
 type (Person), intent(in) :: who
 print *, who % name ; end subroutine print_Name
subroutine print_Nationality (who)
 type (Person), intent(in) :: who
 print *, who % nationality ; end subroutine print_Nationality
subroutine print_Sex (who)
 type (Person), intent(in) :: who
 if ( who % sex == 1 ) then ; print *, "male"
 else ; print *, "female" ; end if ; end subroutine print_Sex
subroutine set_DOB (who, m, d, y)
 type (Person), intent(inout) :: who
 integer, intent(in) :: m, d, y ! month, day, year
 who % dob = Date_ (m, d, y) ; end subroutine set_DOB
subroutine set_DOD(who, m, d, y)
 type (Person), intent(inout) :: who
 integer, intent(in) :: m, d, y ! month, day, year
 who % dod = Date_ (m, d, y) ; end subroutine set_DOD
end module class_Person
```

ภาพที่ 14 *ตัวอย่างของซอร์สโค้ดภาษาฟอร์แทรนในส่วนของ class_Person*

print_GPA และ set_DOM โดยในส่วนของซับรูทีน print_DOM จะมีการเรียกใช้งานซับรูทีน print_Date ในคลาส Date และในส่วนของซับรูทีน print_GPA จะมีการเรียกใช้งานซับรูทีน print_Name ในคลาส Person

จากภาพที่ 14 แสดงตัวอย่างของซอร์สโค้ดภาษาฟอร์แทรนในส่วนของ class_Person ซึ่งจะมีการกำหนดชื่อของคลาสคือ Person และมีส่วนที่เป็นฟังก์ชันคือ make_Person และ Person_ ส่วนที่เป็นซับรูทีนคือ print_DOB print_DOD print_Name print_Nationality print_Sex set_DOB และ set_DOD โดยในส่วนของฟังก์ชัน make_Person จะมีการเรียกใช้งานฟังก์ชัน Date_ ในคลาส Date และในส่วนของซับรูทีน print_DOB จะมีการเรียกใช้งานซับรูทีน print_Date ในคลาส Date

```
module class_Date ! filename: class_Date.f90
    public :: Date ! and everything not "private"
type Date
    private
    integer :: month, day, year
end type Date
contains ! encapsulated functionality
function Date_ (m, d, y) result (x) ! public constructor
    integer, intent(in) :: m, d, y ! month, day, year
    type (Date) :: x ! from intrinsic constructor
    if ( m < 1 .or. d < 1 ) stop 'Invalid components, Date_'
    x = Date (m, d, y) ; end function Date_
subroutine print_Date (x) ! check and pretty print a date
    type (Date), intent(out) :: x
    character (len=*),parameter :: month_Name(12) = &
    (/ "January ", "February ", "March ", "April ",&
    "May ", "June ", "July ", "August ",&
    "September", "October ", "November ", "December "/)
    if ( x%month < 1 .or. x%month > 12 ) print *, "Invalid month"
    if ( x%day < 1 .or. x%day > 31 ) print *, "Invalid day "
    print *, trim(month_Name(x%month)),' ', x%day, ", ", x%year;
end subroutine print_Date
subroutine read_Date (x) ! read month, day, and year
    type (Date), intent(out) :: x ! into intrinsic constructor
    read *, x ; end subroutine read_Date
function set_Date (m, d, y) result (x) ! manual constructor
    integer, optional, intent(in) :: m, d, y ! month, day, year
    type (Date) :: x
    x = Date (1,1,1997) ! default, (or use current date)
    if ( present(m) ) x%month = m ; if ( present(d) ) x%day = d
    if ( present(y) ) x%year = y ; end function set_Date
end module class_Date
```

ภาพที่ 15 *ตัวอย่างของซอร์สโค้ดภาษาฟอร์แทรนในส่วนของ class_Date*

```
<?xml version="1.0" standalone="yes"?>
<uml:Model xmlns:uml="http://schema.omg.org/spec/UML/2.1.1"
xmlns:xmi="http://schema.omg.org/spec/XMI/2.1" xmi:version="2.1"
xmi:id="Autogent1" name="Autogent2">
 <eAnnotations xmi:id="548945626" source="Objing">
  <contents xmi:type="uml:Property" xmi:id="1890496138"
name="exporterVersion">
   <defaultValue xmi:type="uml:LiteralString" xmi:id="1952735779"
value="3.0.0"></defaultValue>
  </contents>
 </eAnnotations>
 <packagedElement xmi:type="uml:Interaction" xmi:id="2136499180"
name="Interaction">
! Lifeline ----------------------------------------------------------------------------
  <lifeline xmi:id="LifelineDate" name="Date" coveredBy="MOS-Receive-
Dateprint_Date EOS-Dateprint_Date BES-Dateprint_Date MOS-ReceiveDate-
Date_ EOS-DateDate_ BES-DateDate_ …"></lifeline>
  <lifeline xmi:id="LifelineStudent" name="Student" coveredBy="MOS-
SendStudentprint_Date MOS-SendStudentDate_ MOS-SendStudentprint_
Name MOS-SendStudentDate…"></lifeline>
  <lifeline xmi:id="LifelinePerson" name="Person" coveredBy="MOS-
ReceivePersonprint_Name EOS-Personprint_Name BES-Personprint_Name
MOS-SendPersonprint_Date…"></lifeline>
  <lifeline xmi:id="Lifelinemain" name="main" coveredBy="MOS-
Sendmainmake_Person MOS-SendmainStudent_ MOS-Sendmainset_DOB
MOS-Sendmainprint_Name…"></lifeline>
! Fragment ----------------------------------------------------------------------------
  <fragment xmi:type="uml:MessageOccurrenceSpecification"
xmi:id="MOS-Sendmainmake_Person" covered="Lifelinemain"
message="Messagemake_Person"></fragment>
  <fragment xmi:type="uml:MessageOccurrenceSpecification"
xmi:id="MOS-ReceivePersonmake_Person" covered="LifelinePerson"
message="Messagemake_Person"></fragment>
  <fragment xmi:type="uml:BehaviorExecutionSpecification" xmi:id=
"BES-Personmake_Person" covered="LifelinePerson" start="MOS-Receive-
Personmake_Person" finish="EOS-Personmake_Person"></fragment>
  <fragment xmi:type="uml:ExecutionOccurrenceSpecification"
xmi:id="EOS-Personmake_Person" covered="LifelinePerson"></fragment>
  <fragment xmi:type="uml:MessageOccurrenceSpecification"
xmi:id="MOS-SendmainStudent_" covered="Lifelinemain"
message="MessageStudent_"></fragment>
  <fragment xmi:type="uml:MessageOccurrenceSpecification"
xmi:id="MOS-ReceiveStudentStudent_" covered="LifelineStudent"
message="MessageStudent_"></fragment>
  <fragment xmi:type="uml:BehaviorExecutionSpecification" xmi:id="BES-
StudentStudent_" covered="LifelineStudent" start="MOS-ReceiveStudentStudent_"
finish="EOS-StudentStudent_"></fragment>
  <fragment xmi:type="uml:ExecutionOccurrenceSpecification"
xmi:id="EOS-StudentStudent_" covered="LifelineStudent"></fragment>
! Message ----------------------------------------------------------------------------
  <message xmi:type="uml:Message" xmi:id="Messagemake_Person"
name="F:make_Person" receiveEvent="MOS-ReceivePersonmake_Person"
sendEvent="MOS-Sendmainmake_Person"></message>
  <message xmi:type="uml:Message" xmi:id="MessageStudent_"
name="F:Student_" receiveEvent="MOS-ReceiveStudentStudent_"
sendEvent="MOS-SendmainStudent_"></message>
 </packagedElement>
</uml:Model>
```

ภาพที่ 16 *ผลลัพธ์การแปลงของซอร์สโค้ดจากระบบที่ได้นำมาทดสอบ*

จากภาพที่ 15 แสดงตัวอย่างของซอร์สโค้ดภาษาฟอร์แทรนในส่วนของ class_Date ซึ่งจะมีการกำหนดชื่อของคลาสคือ Date และมีส่วนที่เป็นฟังก์ชันคือ Date_ และ set_Date ส่วนที่เป็นซับรูทีนคือ print_Date และ read_Date

ภาพที่ 16 แสดงเนื้อหาบางส่วนของเอกสารเอ็กซ์เอ็มไอที่ได้จากการประยุกต์กฎที่สร้างขึ้น โดยจะประกอบด้วย Lifeline ได้แก่ Program Main Person Date และ Student สำหรับในส่วนของ Fragment สามารถแบ่งออกได้ 2 ส่วนคือ ส่วนที่กำหนดการสื่อสารระหว่างสารและไลฟ์ไลน์ ได้แก่ Message Occurrence Specification Behavior Execution Specification และ Execution Occurrence Specification และส่วนสำหรับการสร้างกรอบของแผนภาพ ได้แก่ Alternatives Option และ Loop โดยซอร์สโค้ดที่นำมาทดสอบจะไม่มีรายละเอียดของส่วนการสร้างกรอบของแผนภาพ สำหรับส่วนสุดท้ายคือ Message ดังที่แสดงเช่น Messagemake_Person จะเป็นสารประเภทที่เรียกใช้งานทั่วไปหรือ Synchronous Message ซึ่งจะแสดงถึงการเรียกใช้งานของสารระหว่างไลฟ์ไลน์ โดยจะมีการกำหนดไลฟ์ไลน์ที่ทำการรับและส่งสาร เป็นต้น ในขั้นตอนสุดท้ายผู้วิจัยได้ตรวจสอบความถูกต้องด้วยการพิจารณาเปรียบเทียบระหว่างเอกสารเอ็กซ์เอ็มไอที่ได้มาจากโปรแกรม กับซอร์สโค้ดฟอร์แทรนแต่ละส่วนซึ่งการเปรียบเทียบนี้ให้ผลถูกต้องในทุกส่วน

5. อภิปรายและสรุปผล

งานวิจัยนี้นำเสนอ แนวคิดการออกแบบกฎการแปลงโมเดิร์นฟอร์แทรนเป็นยูเอ็มแอลซีเควนซ์ไดอะแกรม โดยมีจุดมุ่งหมายเพื่อนำกฎที่ได้ไปประยุกต์ใช้ในการพัฒนาเครื่องมือสำหรับแปลงโมเดิร์นฟอร์แทรนเป็นยูเอ็มแอลซีเควนซ์ไดอะแกรม จากการออกแบบกฎการแปลงข้างต้น ผู้วิจัยได้นำไปเปรียบเทียบกับภาษาจาวา ซึ่งเป็นภาษาเชิงวัตถุที่สมบูรณ์ ในขณะที่ภาษาฟอร์แทรนนั้นจะเป็นภาษาเชิงวัตถุที่แตกต่างจากภาษาเชิงวัตถุโดยทั่วไป เช่น การจัดการหน่วยความจำ (Garbage Collection) เพื่อจัดการกับอ็อบเจ็กต์ของภาษาฟอร์แทรนจะต้องมีการเขียนโค้ดขึ้นมาเพื่อจัดสรรการทำงานเอง แต่ในภาษาเชิงวัตถุภาษาอื่นนั้น เช่น ภาษาจาวาจะมีการจัดสรรโดยอัตโนมัติ ในเรื่องของเมธอดในภาษาฟอร์แทรนจะมีการแบ่งการทำงานโดยแยกเป็นฟังก์ชันและซับรูทีน แต่ในภาษาจาวาจะไม่มีการแบ่งการทำงานของเมธอด ซึ่งจากข้อแตกต่างในเรื่องของภาษาเชิงวัตถุนี้ผู้วิจัย

จึงทำการออกแบบกฎการแปลงให้เข้ากับภาษาฟอร์แทรน เช่น ในส่วนของการแสดงแผนภาพยูเอ็มแอลซีเควนซ์ไดอะแกรมผู้วิจัยได้มีการออกแบบชื่อของสารโดยถ้าสารมีการเรียกใช้งานแบบฟังก์ชันจะแสดงสัญลักษณ์ F ไว้หน้าชื่อ และถ้าหากสารมีการเรียกใช้งานแบบซับรูทีนจะแสดงสัญลักษณ์ S ไว้หน้าชื่อ

ในอนาคตผู้วิจัยจะนำกฎการแปลงที่นำเสนอในงานวิจัยนี้ไปใช้สำหรับพัฒนาความสามารถของเครื่องมือ ForUML เพื่อใช้ในการสร้างซีเควนซ์ไดอะแกรม

6. เอกสารอ้างอิง

[1] M. Lanza and S. Ducasse. “Polymetric views-a lightweight visual approach to reverse engineering.” *IEEE Transactions on Software Engineering*, Vol. 29, No. 9, pp. 782–795, September, 2003.

[2] T. Systa. “On the relationships between static and dynamic models in reverse engineering java software.” *In Proceedings of Reverse Engineering Sixth Working Conference*, 1999, pp. 304–313.

[3] A. Nanthaamornphong, J. Carver, K. Morris, and S. Filippone. “Extracting uml class diagrams from object-oriented fortran: Foruml.” *Scientific Programming, Article ID 421816*, Vol. 2015, pp. 1–15, January, 2015.

[4] J. C. Carver, R. P. Kendall, S. E. Squires, and D. E. Post, “Software development environments for scientific and engineering software: A series of case studies.” *In Proceedings of the 29th International Conference on Software Engineering (ICSE’07)*, pp. 550–559, 2007.

[5] J. C. Carver. “Report: the second international workshop on software engineering for CSE.” *Computing in Science & Engineering*, Vol. 11, No. 6, pp. 14–19, November, 2009.

[6] B. Dobing and J. Parsons, “How UML is used.” *Communications of the ACM*, Vol. 49, No. 5, pp. 109–113, May, 2006.

[7] N. S. Clerman and W. Spector. “Modern Fortran: style and usage.” *Cambridge University Press*, 2011.

[8] D. Barbieri, V. Cardellini, S. Filippone, and D. Rouson, “Design patterns for scientific computations on sparse matrices.” *In Proceedings of European Conference on Parallel*, 2011, pp. 367–376.

[9] K. Morris, D. W. Rouson, M. N. Lemaster, and S. Filippone. “Exploring capabilities within ForTrilinos by solving the 3D Burgers equation.” *Scientific Programming*, Vol. 20, No. 3, pp. 275–292, July, 2012.

[10] A. Rukin. *Java decompilers*. Available Online at http://www.javadecompilers.com, accessed on 25 November 2016.

[11] P. Andritsos and R. J. Miller. “Reverse engineering meets data analysis.” *In Proceedings of the 9th International Workshop on Program Comprehension*, pp. 157–166, 2001.

[12] D. Rouson. *Sourcery Institute*. Available Online at http://www.sourceryinstitute.org/software.html accessed on 25 November 2016.

[13] T. C. Lethbridge, S. Tichelaar, and E. Plödereder. “The dagstuhl middle metamodel: A schema for reverse engineering.” *Electronic Notes in Theoretical Computer Science*, Vol. 94, pp. 7–18, May, 2004.

[14] OMG. *UML specification v2.5*. Available Online at http://www.omg.org/spec/UML/2.5/PDF accessed on 2 January 2017

[15] P. Sawprakhon and Y. Limpiyakorn. “Sequence Diagram Generation with Model Transformation Technology.” *In Proceedings of the International MultiConference of Engineers and Computer Scientists*, pp. 12–14, 2014.

[16] C. Li, L. Dou, and Z. Yang. “A metamodeling level transformation from UML sequence diagrams to Coq.” *In Proceedings of International Conference on Information and Communication Technology for Competitive Strategies*, pp. 147–157, 2014.

[17] E. Merah. “Design of ATL Rules for Transforming UML 2 Sequence Diagrams into Petri Nets.” *International Journal of Computer Science and Business Informatics*, Vol. 8, No. 1, pp. 1–21, January, 2014.

[18] J. Akin. “Object oriented programming via Fortran 90.” *Engineering Computations*, Vol. 16, pp. 26–48, 1999.