MATERA2-AlfTester: An Exhaustive Simulation and Test Generation Tool for fUML Models

Junaid Iqbal, Adnan Ashraf, Dragos Truscan, Ivan Porres
Faculty of Science and Engineering
Åbo Akademi University, Turku, Finland
{jiqbal, aashraf, dtruscan, iporres}@abo.fi

***Abstract*—The Foundational Subset for Executable UML Models (fUML) and the Action language for fUML (Alf) can be used for creating executable models in the Eclipse-based UML editing tool called Papyrus. An fUML execution engine in Papyrus, such as Moka, allows to simulate or execute fUML models along with their associated Alf code. However, for exhaustive simulation of such models, one must provide input data required to reach and cover all important elements not only in the graphical fUML models, but also in the textual Alf code. In this paper, we present MATERA2-AlfTester, an Eclipse-plugin for exhaustive simulation and test generation for fUML models. MATERA2-AlfTester integrates with Papyrus and Moka tools and extends their functionally by allowing one to automatically generate test data, test suite with test oracle, and partial Java code at design time. We also present the simulation and testing process of MATERA2-AlfTester with the help of an example and demonstrate how exhaustive simulation and test generation with MATERA2-AlfTester can help designers in assessing and improving the quality of fUML models.**

***Keywords*-fUML, simulation, test generation, Papyrus, Moka, Alf**

I. Introduction

Papyrus [1] is a Unified Modeling Language (UML) [2] modeling tool for the Eclipse Modeling Framework [3] and provides tool support for executable UML modeling [4]. It includes technologies for the simulation and debugging of models, as well as editing facilities to produce executable models more efficiently. The simulation and debugging part is handled by a plug-in called Moka [5], which relies on an implementation of the Foundational Subset for Executable UML Models (fUML) standard [6] and the Precise Semantics of UML Composite Structures (PSCS) standard [7]. These specifications formalize the execution semantics of a UML subset. In addition to graphical editing, Papyrus also supports the Action language for fUML (Alf) standard [8], with an editor and compiler for Alf.

The execution and simulation features of Moka allow one to provide input data required to execute the graphical fUML and textual Alf models. However, manual generation of input data might be suitable for small and simple models, but it is often not the case for real-life complex models. Similarly, test generation for executable models is a difficult and tedious task when performed manually.

To address these issues, we present an Eclipse plugin, called the *MATERA2-AlfTester* (M2-AT), which implements an exhaustive simulation and test generation approach with the following objectives: (1) generate test data for simulation to achieve maximum model coverage, (2) run these inputs automatically in the Moka plugin to simulate the model (3) generate a test suite with test oracle (expected output) that can be used for testing the system implementation and 4) generates corresponding Java code skeleton for fUML model which can be used as a starting point for system development. To the best of our knowledge, our tool is the first one which generates test data from fUML activity diagrams (ADs) and associated Alf code for exhaustive simulation in Moka. The underlying approach of M2-AT can be found in [9].

II. Tool Overview

Papyrus provides the ability to simulate fUML models via the Moka plugin. During the simulation process, it allows to measure model coverage graphically and produces a model coverage report at the end of the simulation. However, one must provide input data required to reach and simulate all essential elements in the graphical fUML and textual Alf models. By using inputs during the simulation, one can observe the model coverage in the simulation framework to identify the potential uncovered parts of the model. The process can be continued until a satisfiable quality and coverage of the models is achieved. Moreover, the test inputs and test suite also allow one to conform the specification to model at an early stage in the system development. As presented in Figure 1, M2-AT provides an automated way to generate such test data, a JUnit test suite, and partial Java code with convenience which reduces the cognitive effort and time in system development and testing. It is important to note that our use of Moka is mostly simulation-oriented. Other uses (e.g., code generation and deployment) from Moka would probably require alternative solutions or implementation strategies as discussed in [10], [11].

III. Naming Scheme and Modeling Guidelines

By exploiting both graphical and textual notations [12], the dominating modeling feature of Papyrus allow designers to design models by using different architectural contexts including UML and executable UML. The executable UML models are based on graphical fUML models and textual Alf code which enable the designers to model precise semantics of the specifications. The architecture context of executable UML in Papyrus is based on structure and behavior viewpoints.

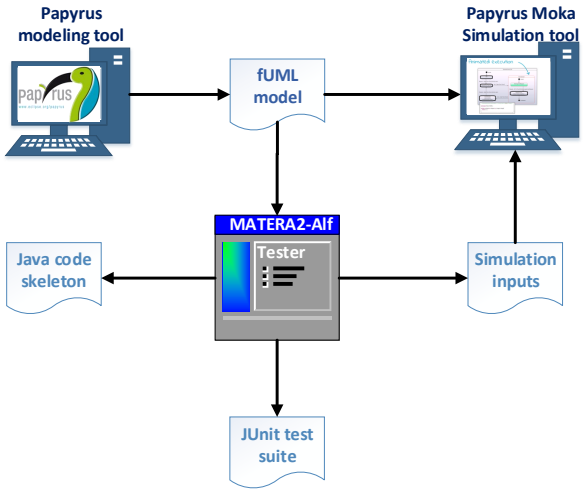

Fig. 1: M2-AT : Automated generation of (1) input data generation for simulation and its execution, (2) test suite generation and (3) Java code skeleton generation

The structural viewpoint is based on *class diagram* (CD) and component structure diagrams. The behavioral viewpoint is described via state machines and *Activity Diagrams* (ADs).

To develop an fUML model, designers need to identify the structural and behavioral aspects of the system to model the static and dynamic part of the system. The CDs describe the static part of the system via attributes and operations. Additionally, associations are used to model the relationships among them. In over approach, the behavioral model is mainly composed of activity diagrams which describe precise step-by-step actions to be performed to model the operation's functionality. In order to generate test inputs from syntactically correct and a minimal functional Java code skeleton, we propose some naming conventions and modeling guidelines to be adapted during the modeling phase for designing both structural and behavior models as follow:

A. Structural model

We propose the following naming conventions for static part of the CDs as follow:

- *Attributes* must have a type and a field modifier.
- *Operation* must have a *public* field modifier, and must be associated to a behavior model (*i.e., activity*) otherwise not considered in test suite generation.
- *Classes* must belong to a *package*. A class may refer objects from the same or different package which must be generated in order to have syntactically correct and compilable Java code. Therefore, we discourage standalone classes and they are not included in the test generation phase.

An example of the recommended structural hierarchy is shown in Figure 2.

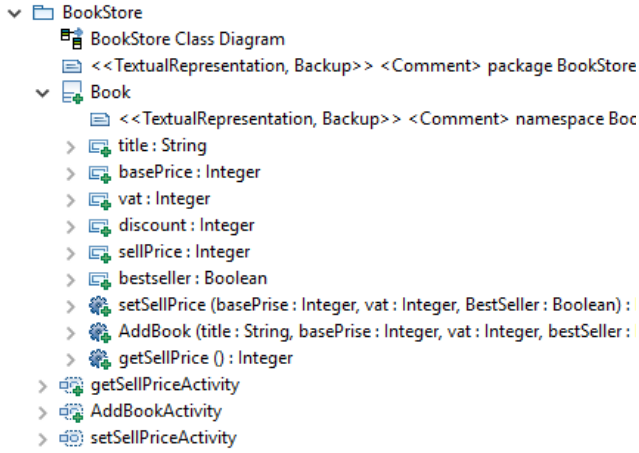

Fig. 2: Structural hierarchy of different elements in the model

B. Behavioral model

The behavior model in fUML is primarily based on *activities* which describe the functionality of pairing classifiers (operations). The activities can be modeled via either using the graphical notation *i.e.,* AD or Alf textual notation. In AD, nodes are connected via control-flow and object-flow edges. The control-flow edges denote the sequence of the nodes to be followed in the execution of the AD without carrying a data object. The object-flow edges are used to transfer the data objects between nodes and connected via input and output pins. The pins attached to the endpoints of the object-flow connect the source and target nodes except for control nodes, *i.e.,* decision, fork, join, merge, and activity parameter node. We propose the following guidelines while modeling AD:

- Both pins at the endpoints of an object-flow edge must use the same name as the data object carried by the edge to avoid syntactically incorrect code which may generate imprecise test inputs.
- A decision node must have a paired merge node.
- AD must have at most one initial and complementing final node.
- The input and output parameter nodes must be connected with at most one object-flow edge.

Figure 3 presents an example AD with recommended modeling guidelines and naming conventions.

IV. Example

We use the *Book* class in Figure 2 as an example. The functionally of the book class is to add new book title in a bookstore based on the base price, value added tax and popularity of the book. The class operations are modeled with fUML ADs and Alf code. The *AddBook* operation adds a book by using the *setSellPrice* operation. Similarly, the *getSellPrice* operation returns the selling price of the book. The *setSellPrice* operation is modeled with fUML AD shown in Figure 3.

The *setSellPriceActivity* AD takes three input parameters namely, *BasePrice*, *BestSeller* and *value-added tax (VAT)*. The discount is decided based on the BestSeller. The discount is half if the book is a best seller otherwise full. Based on base

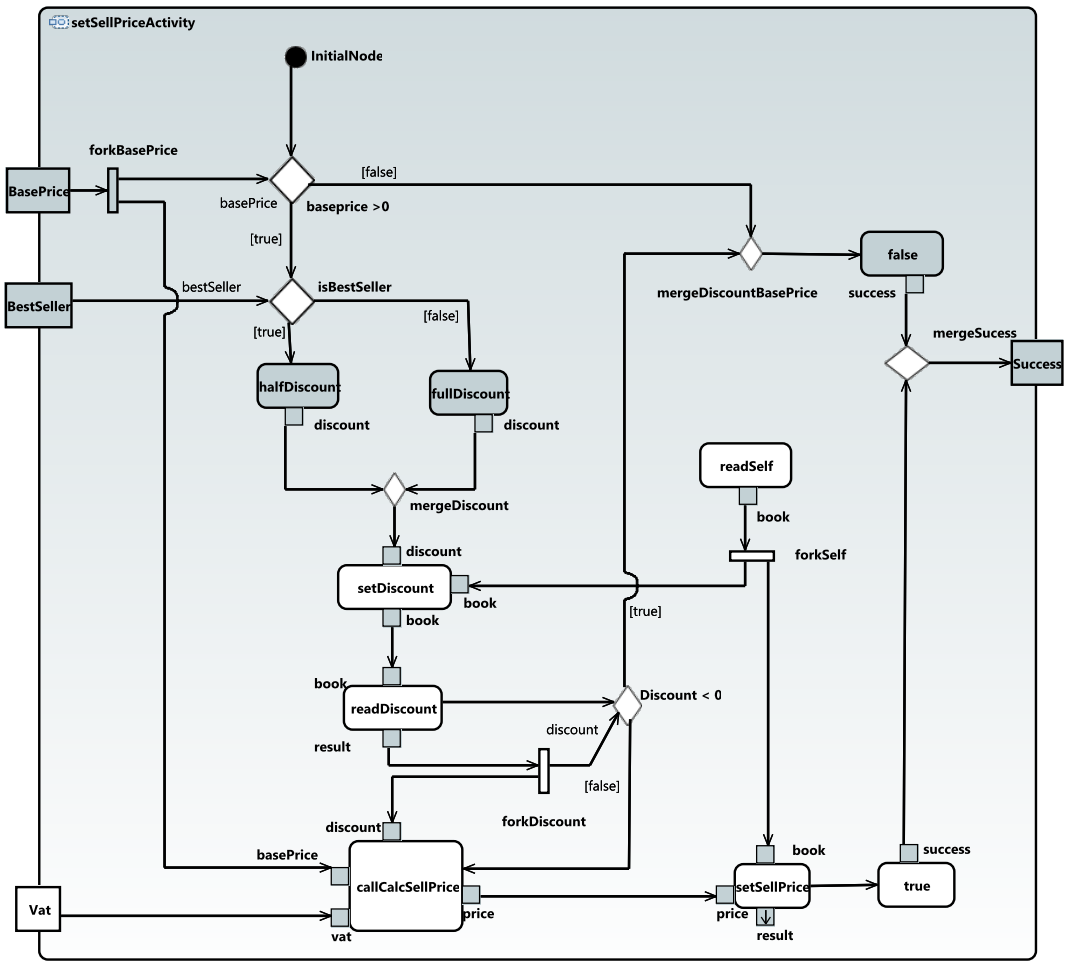

Fig. 3: fUML AD of *setSellPriceActivity*

price, VAT and discount, the selling price is calculated by *calcSellPrice* activity, written in Alf, and stored in the *sellPrice* attribute. The *setSellPriceActivity* AD returns *true* if the selling price is successfully stored. Otherwise, if the base price or discount is below zero the activity returns *false* via return parameter node *Success*.

V. Simulation and Test Generation

M2-AT tool is implemented as an Eclipse plugin, integrated with the Eclipse Modeling platform and it is based on the Papyrus and Moka tools. The tool generate simulation inputs that cover 100% of the specification and executes them automatically. This can help in detecting problematic parts of the model, such as non-reachable elements, uninitialized objects, or illegal input combinations. In addition, the tool can generate simulation inputs, execute them automatically and generate tests with test oracle and Java code skeleton. The tool takes advantage of Eclipse UML2 implementation along with the Eclipse Graphical Modeling Framework (GMF) to parse the fUML model. The fUML model is parsed to extract packages containing classes and activities later used to generate static and dynamic parts of the model. Hence, the static part of the fUML model is generated via attributes and operation signatures while the dynamic part of the fUML model is generated by processing activities.

By using the line and branch coverage criteria, M2-AT is expected to yields 100% coverage in the generated Java code ensuring 100% node and edge coverage in fUML ADs. However, the coverage less the 100% indicate a potential unreachable area in model as well in generated Java code. In the following subsections, we present the steps to use M2-AT by using the example presented in Section IV.

A. Model transformation and test suite generation

In order to generate test suite along with test data for simulation, the fUML model is loaded in Papyrus. The model transformation and test suite generation can be triggered via selecting *MATERA2-Package code* option from the context menu in Papyrus model explorer as shown in Figure 4. The generated Java code is stored in user's selected directory.

M2-AT invokes EvoSuite, must be downloaded separately and accessible to M2-AT, uses the generated Java code and generates a test suite. Figure 5 presents the EvoSuite console log during the test suite generation.

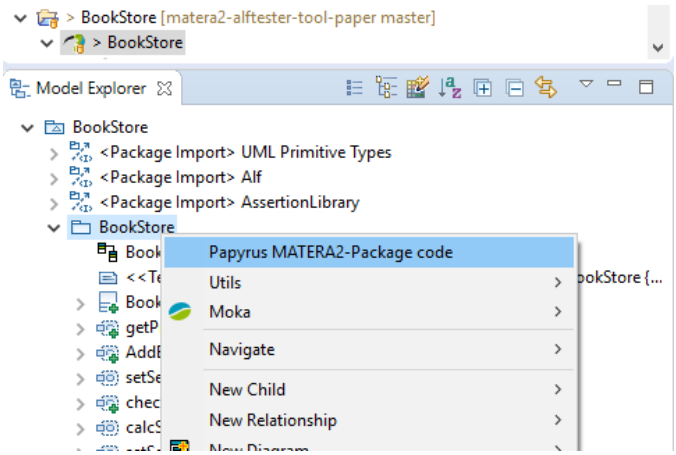

Fig. 4: Generating test suite from fUML package

```
* EvoSuite 1.0.6
* Found 1 matching classes in target BookStore
* Analyzing classpath (generating inheritance tree)
  - Z:\crap\BookStore
* Current class: BookStore.Book
* Going to generate test cases for class: BookStore.Book
* Starting client
* Connecting to master process on port 20369
* Analyzing classpath:
* Inheritance tree loaded from C:\Users\jiqbal\AppData\Local\Temp\ES_inheritancetree2379041136513175
* Finished analyzing classpath
* Generating tests for class BookStore.Book
* Test criteria:
  - Line Coverage
  - Branch Coverage
* Setting up search algorithm for whole suite generation
* Total number of test goals:
  - Line 23
[Progress:>                                0%] [Cov:>                                0%]  - Branch 7
* Using seed 1553875429567
* Starting evolution
[Progress:==============================100%] [Cov:==============================>   90%]
* Search finished after 61s and 3871 generations, 1296242 statements, best individual has fitness: 2.
* Minimizing test suite
* Going to analyze the coverage criteria
* Coverage analysis for criterion LINE
* Coverage of criterion LINE: 96%
* Total number of goals: 23
* Number of covered goals: 22
* Coverage analysis for criterion BRANCH
* Coverage of criterion BRANCH: 86%
* Total number of goals: 7
* Number of covered goals: 6
* Generated 3 tests with total length 6
* Resulting test suite's coverage: 91% (average coverage for all fitness functions)
* Generating assertions
* Resulting test suite's mutation score: 11%
* Compiling and checking tests
* Writing JUnit test case 'Book_ESTest' to evosuite-tests
* Done!
```

Fig. 5: EvoSuite console log during test suite generation

B. Alf test script generation and simulation

An Alf test script, which encodes the test inputs in ALF, can be generated by accessing the context menu of the AD in the model explorer view of Papyrus. The generated test script, as shown in Figure 6, is added into the original model as a testing activity and can be executed by using the context menu and selecting: $Moka- > Run$. As shown in Figure 7, one can observe the model coverage during the execution via green and black color edges. The green edges shows the covered areas of the model during the simulation, however, the black color edges may identify the potential uncovered areas in the model. In Figure 7, the edge between $Discount < 0$ and *MergeDiscountBasePrice* nodes is unreachable during the simulation, which identifies a modeling error in the AD. Consequently, the log presented in Figure 5 showing the overall coverage reported by EvoSuite is 91% including 96% of line and 86% of branch coverage.

Additionally, an illegal input combination can be detected

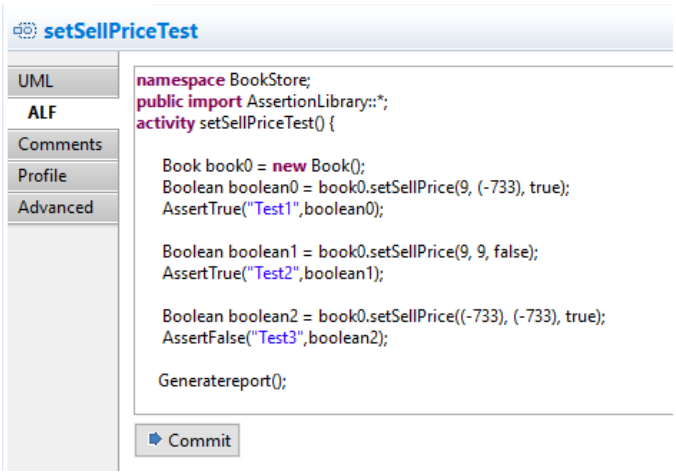

Fig. 6: Alf test script for *setSellPriceActivity*

during simulating via exceptions raised by the simulation engine. The designer can benefit form exhaustive simulation to identify potential unreachable areas in the model and rectify the modeling errors.

When the models are of good quality with desired coverage, Java code skeleton can be used as a starting point in system development. Furthermore, the generated test suite can be run in later stages to conform the specification to the developed system. The main benefits of M2-AT include the automation of the process which ensures good quality models, testing at an early stage during system development process, and providing a starting point for the actual development of the system.

VI. Conclusion

The Foundational Subset for Executable UML Models (fUML) and the Action language for fUML (Alf) allow to create executable models, which can be executed using an fUML execution engine. However, to execute such models exhaustively, one must provide input data required to reach and cover all essential elements not only in the graphical fUML models, but also in the textual Alf code associated with the graphical models. In this paper, we presented an Eclipse-based tool, called *MATERA2-Alf Tester* (M2-AT), which translates fUML ADs and associated Alf code into equivalent Java code and then automatically generates: (1) input data needed to cover or execute all paths in the executable fUML and Alf models and (2) a test suite comprising test cases with test oracle for testing the actual implementation of the system under development. The generated test cases in M2-AT satisfy 100% code coverage of the Java code. The generated input data is used for executing the original fUML and Alf models in the Moka simulation engine. The interactive execution in Moka allows to measure model coverage of the executable models. In addition, the generated Java code can be reused as a starting point for the actual implementation of the system. We also presented our tool and demonstrated our proposed approach with the help of an example. Our proposed tool integrates M2-AT code generation and Alf script generation components

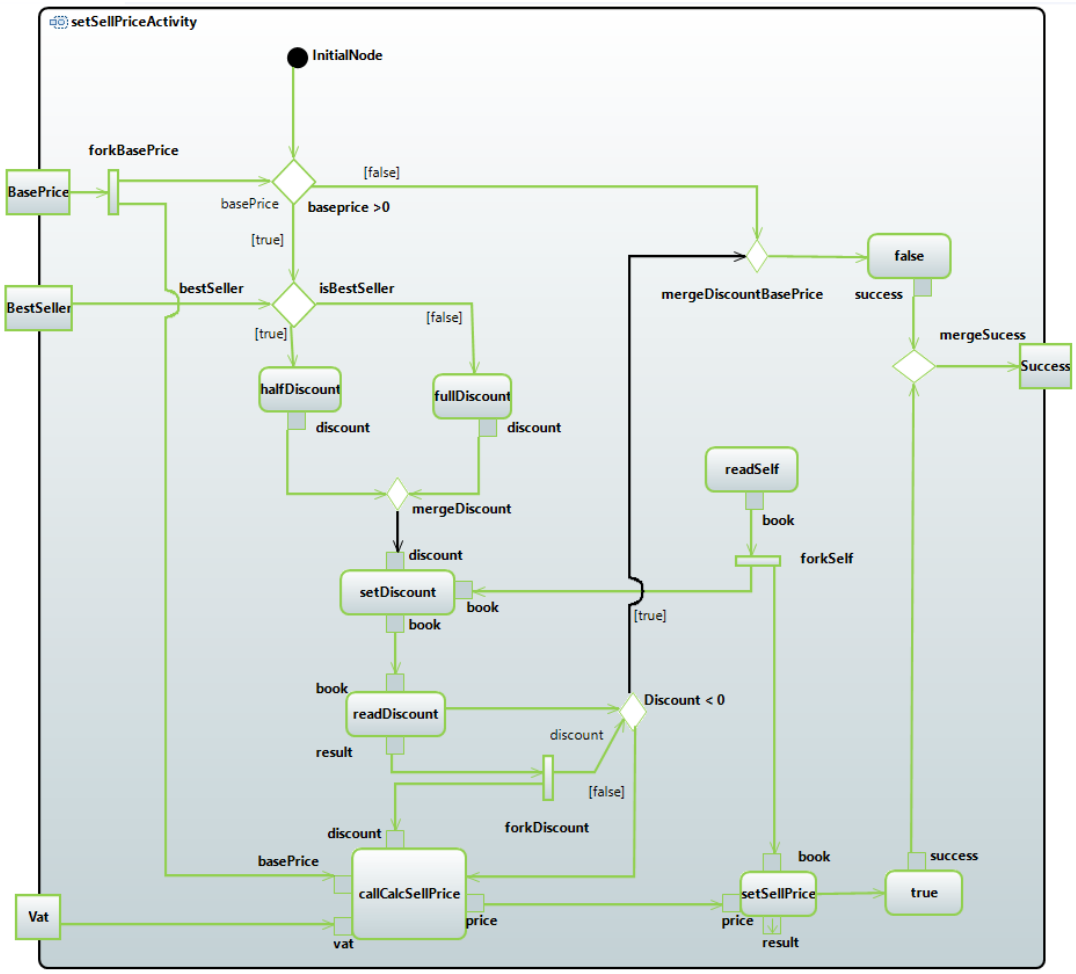

Fig. 7: Running Alf test script for *setSellPriceActivity* and observing model coverage

with the state-of-the-art model simulation and test generation tools allowing researchers and practitioners to generate test suites and input data for exhaustive model simulation at early stages of the software development life cycle.

ACKNOWLEDGMENTS

This work has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement number 737494. This Joint Undertaking receives support from the European Unions Horizon 2020 research and innovation programme and Sweden, France, Spain, Italy, Finland, the Czech Republic.

REFERENCES

[1] E. Foundation, "Eclipse papyrus modeling environment." [Online]. Available: http://www.eclipse.org/papyrus

[2] OMG, "About The Unified Modeling Language Specification," 2017. [Online]. Available: https://www.omg.org/spec/UML/

[3] Eclipse, "Eclipse Modeling Framework (EMF)," 2019. [Online]. Available: https://www.eclipse.org/modeling/emf/

[4] S. J. Mellor and M. Balcer, "Executable UML," *A Foundation for Model-Driven Architecture. Addison-Wesleyy*, 2002.

[5] E. Foundation, "Eclipse papyrus moka." [Online]. Available: https://git.eclipse.org/c/papyrus/org.eclipse.papyrus-moka.git/

[6] OMG, "About the semantics of a foundational subset for executable uml models specification." [Online]. Available: https://www.omg.org/spec/FUML

[7] OMG, "About the precise semantics of uml composite structures specification PSCS," 2019. [Online]. Available: https://www.omg.org/spec/PSCS/About-PSCS/

[8] OMG, "About the action language for foundational uml specification." [Online]. Available: https://www.omg.org/spec/ALF

[9] J. Iqbal, A. Ashraf, D. Truscan, and I. Porres, "Exhaustive simulation and test generation using fUML activity diagrams," in *International Conference on Advanced Information Systems Engineering (CAiSE)*, P. Giorgini and B. Weber, Eds. Cham: Springer International Publishing, 2019, pp. 96–110.

[10] G. Dévai, G. F. Kovács, and Á. An, "Textual, executable, translatable uml." in *OCL@ MoDELS*, 2014, pp. 3–12.

[11] Z. Micskei, R.-A. Konnerth, B. Horváth, O. Semeráth, A. Vörös, and D. Varró, "On open source tools for behavioral modeling and analysis with fuml and alf." in *OSS4MDE@ MoDELS*. Citeseer, 2014, pp. 31–41.

[12] S. Guermazi, J. Tatibouet, A. Cuccuru, S. Dhouib, S. Gérard, and E. Seidewitz, "Executable modeling with fUML and Alf in Papyrus: tooling and experiments," in *1st International Workshop on Executable Modeling MODELS,Ottawa, Canada*, P. L. Tanja Mayerhofer, Ed., 2015. [Online]. Available: http://ceur-ws.org/Vol-1560/