On the Applicability of ALF Language in Real Software Projects

Radek Kočí and Lukáš Osadský

Brno University of Technology, Faculty of Information Technology
Bozetechova 2, 612 66 Brno, Czech Republic
emails: koci@fit.vut.cz, xosads00@stud.fit.vutbr.cz

***Abstract*—Modeling is one of the critical activities in specifying requirements and designing a software system. In the design and development of software systems, there has been a long-term trend of shifting from static software models to feasible models. These models include, for example, state diagrams or techniques using automated model transformations, which are based on a subset of Unified Modeling Language (UML) models and supplement them with special languages, such as Action Language for Foundational UML (ALF). A common feature is to move part of the verification and testing from the implementation stage to the design stage and eliminate the implementation process. In this paper, we will focus on the possibilities of using one direction of application of models in software development, namely the Foundational Subset of Executable UML (fUML), in conjunction with the specification language ALF. The paper provides a literature search on the fundamental essence of the Model-Driven Engineering approaches, namely fUML and ALF language. Then, we tried to apply it to the case study of a conference system and captured all the problems. Due to the nature of the issues, we will not present this case study, as the substance is not essential for the paper.**

***Keywords*—modeling; software systems; model-driven engineering; ALF language.**

I. Introduction

Modeling is one of the critical activities in specifying requirements and designing a software system. The model can be understood as an abstraction of the system, over which the simulation can be performed, as well as part of the design process in the development of new systems. In the design and development of software systems, there has been a long-term trend of shifting from static software models to feasible models, which allow analyzing and verifying design properties in a simulation way, i.e., without the need to implement models. This category of use includes, for example, state diagrams or techniques using automated model transformations, which are based on a subset of UML models and supplement them with special languages, such as ALF. A common feature is to move part of the verification and testing from the implementation stage to the design stage and eliminate the implementation process.

Model-driven approaches assume that it would be more advantageous for application development if the requirements modeling and prototyping steps were combined. The resulting model could, with minor modifications, directly serve as a functional prototype of the application. The basis is the transformation of models according to their purpose. The process begins with the creation of specification models, which are further transformed into executable models that can be verified in a simulation manner. Models can be transformed into different forms, according to their purpose.

In this paper, we will focus on the possibilities of using one direction of application of models in software development, namely the Foundational Subset of Executable UML (fUML), in conjunction with the specification language ALF. The paper describes the fundamental essence of these approaches and summarizes the problems associated with fUML and ALF's case study of a conference system. Due to the nature of the issues, we will not present this case study, as the substance is not essential for the paper.

The paper is structured as follows. First, we briefly summarize concepts od modeling (Section II) and Model-Driven Development (MDD, Section III) of software systems. The current development environments for MDD with ALF language are presented in Section IV. A literature search of essential papers is presented in Section V. Section VI evaluates an applicability of MDD and ALF language in software projects. Finally, the possible development of these techniques is discussed in Section VII.

II. Modeling

The next section is devoted to modeling, various approaches, and resources related to this activity. Models are the basis of modeling and are used throughout all parts of the software development cycle. They are used for the needs of specification, documentation, design, and the like. The German philosopher Herbert Stachowiak described them on the basis of three characteristics.

- *Mapping*. The model is always a model of the original. The original can be something real or imaginary.
- *Reduction*. Models generally do not capture all the original attributes, but only those that are relevant for modeling purposes.
- *Pragmatism*. Models are not assigned to their originals in a one-to-one relationship; they always perform the function of replacement. To fulfill their purpose, they must be used instead of the original.

It is a form of abstraction that describes the state, properties, dynamics, or structure of the modeled object, system, or part of them. Several models can describe the entity we are modeling. One model can describe its structure, other properties, or behavior in different degrees of abstraction. In software engineering, models are used, for example, in the specifications of the customers' requirements. Their purpose is to capture the required system properties, which are used as

 ISBN: 978-1-61208-997-3

a model for the creation of the system. In other cases, models may serve as major artifacts in the implementation of systems.

When modeling, we can encounter several terms related to it. Model-Based Engineering (MBE) or Model-Driven Engineering (MDE) is a software paradigm that applies the principles of visual modeling during the software development lifecycle. It is an umbrella term for disciplines such as Model-Driven Development (MDD), Model-Based System Engineering (MBSE), Business Process Modeling (BPM), and OntologyEngineering. MDD is a sub-discipline dealing with the application of model-driven technologies to software development activities. Another discipline dealing with software development is the Model-Driven Architecture (MDA). It is a specific implementation of MDD. MBE is a discipline of systems engineering, whose primary goal is to use the model of information exchange between engineers. BPM is used to describe business processes using diagrams. Ontology Engineering is a discipline specialized in creating ontologies. This work will deal in more detail with the discipline of MDD and MDA.

III. Model-Driven Development

MDD is a sub-discipline of Model-Driven Engineering dealing with applying model-driven techniques to software development activities. It is an approach that uses models as first-class entities to create products. The motivation for using model-driven approaches is that standard software is continually evolving, and the platforms on which this software works are also changing. Thus, it is necessary to continuously modify the products for the platforms they are to operate, which brings with it the need to invest more time and resources in their modifications. One of the goals of model-driven approaches is to prevent this and save resources. MDE takes software development to a higher level of abstraction, so there is no need to adapt models to every platform change. One abstract model can be re-used for multiple platforms without intervention. Besides, they are easier to maintain.

Models as development artifacts are easier to understand even for non-interested parties. One of the problems with using models for development is capturing all implementation details [1].

A. Model-Driven Architecture

MDA is a specific implementation of MDD from Object Management Group (OMG). This implementation is based on several OMG standards, such as the Meta-Object Facility (MOF), XML Metadata Interchange (XMI), Object Constraint Language (OCL), UML, and others. MDA works with basic UML models, which are described using Domain Specific Language (DSL). DSL is a language that specializes in a particular application domain. An example might be SQL or ALF languages. MDA is based on three principles [1].

- *Direct representation*. It allows you to combine problems with solutions using DSL.
- *Automation*. Elements introduced by DSL are to be processed by tools that connect the gap of domain concepts with implementation technologies.
- *Standard*. It allows us to connect technical solutions.

The main idea of MDA is that each software can be used on different platforms. For this reason, MDA uses models that are transformed into other models using transformation rules. The individual transformations are based on the Query/View/Transformation (QVT) standard [2].

B. Executable MDA

Executable UML (xUML) is a software development concept and a highly abstract language that aims to compile and run UML models. xUML is a modification of UML, in which it is possible to describe the details of the model to such a level that it is possible to run it or generate functional code from it. xUML combines a subset of UML models with executable semantics. Models in this environment can be run, debugged, tested, and compiled by the domain of abstract languages. xUML supports MDA, for example allowing the translation of Platform Independent Models (PIMs) into Platform Specific Models (PSMs) [3].

C. Foundational Subset for Executable UML

fUML is an OMG standard. It is a platform-independent executable subset of standard UML that provides modeling concepts for defining UML classes and the behavior of these classes. This subset includes typical UML modeling constructs such as classes, data types, associations, and enumerations. It also provides the means to define model behavior using UML activities. fUML uses different types of diagrams from UML, i.e., it uses class diagrams to define a static structure and activity diagrams to define the behavior [4].

D. Foundational Subset for Executable UML

Modeling systems using graphical representation may not always be sufficient. It is not possible to capture all the details of the system, so text representation is also used to describe the models. There are several options for a textual description of the model. One of them is ALF language from OMG. In addition to the specific textual syntax for describing fUML models, ALF also provides execution semantics by mapping ALF syntax to the abstract syntax of fUML models. The specific syntax of the ALF language is based on a context-free grammar written in Enhanced-Backus-Naur-Form (EBNF) form, and its abstract syntax is represented by the UML model [11]. The primary goal of ALF is to use text syntax to specify executable behavior within a model, i.e., specify the bodies of the class methods needed for the operations contained in the class diagrams. The syntax of the ALF language is similar to the Java language's syntax, uses the default type system, and has static type checking. According to the specification, the ALF code can be executed in three ways [5].

- *Interpretive Execution*. The ALF model can be directly interpreted and triggered.

 ISBN: 978-1-61208-997-3

- *Compilated execution*. The ALF model is translated into a UML model to match fUML and is run according to the semantics specified by fUML.
- *Translation execution*. The ALF model is translated into code that is executable on the selected platform.

IV. Environments

While working on this paper two environments, which can be used for ALF development, were tested.

A. Eclipse IDE

Eclipse IDE is primarily used for developing Java applications, but with the integration of several plugins, it is possible to use ALF language. To use ALF with Eclipse IDE, it is needed to add plugins like Papyrus Modeling Environment for modeling, Moka for model execution/debugging, and Nebula to add custom widgets. We followed the manual [6] to set up this environment and run and debug ALF models and Papyrus Software designer. After installing all necessary plugins, it is needed to display widgets to work with ALF like debugging widgets, model properties widgets, and others. After all this, the environment is ready to use. This environment is suited to use ALF for describing the behavior and fUML models for describing the structure, but it is also possible to use ALF for both. For complete-textual usage, it is better to use the ALF reference implementation [7]. An example model mentioned in the manual [6] was used to test the environment. The given model can be modified, run, and debugged. The first complications occurred with the generation of code from this model. Papyrus Software designer allows us to generate code from UML models when trying to generate code. Nevertheless, only the class skeleton was created; the behavior implemented using the ALF language was not generated. It was not generated because Papyrus Software Designer does not support ALF code generation. For additional code generation options, it is needed to add the Acceleo extension to the Eclipse environment, which serves as a code generator or allows us to define our own code generation rules. Like Papyrus Software Designer, Acceleo does not support ALF, so it is needed to implement its own code generation template.

B. MagicDraw

Furthermore, the commercial tool MagicDraw from No-Magic is used as the primary tool for ALF. Because MagicDraw does not have a free license, its demo version was used. Installation and deployment of the environment are much more comfortable than in the case of Eclipse. It only needs to install MagicDraw and add the ALF integrated editor extension and Cameo Simulation Toolkit extension used to run fUML models. Like Papyrus, MagicDraw allows us to describe the structure using diagrams and model behavior with ALF. But, it is not possible to specify the system structure in ALF, the only thing which can be described is the behavior of parts of the model. As part of product testing, simple class diagrams were created with methods whose behavior was described in the ALF language. Subsequently, it is possible to run the whole model or part of it directly in MagicDraw. Once launched, a simulation window opens with a console. It is possible to start the simulation of behavior described with ALF, set breakpoints, and change the animation's speed. It is also possible to write simple tests with the help of activities. To create a simple model in this environment, these instructions were followed [8]. To generate code, it is necessary to create a "Code Engineering Set", in which the target language and the set of models from which the code is to be generated are selected. There is a manual related to code generation [9]. The problem with code generation is that only the UML model's skeleton is generated during code generation, and the behavior written in ALF is not. MagicDraw does not support code generation from ALF to target language.

Originally, ALF's focus was to ease the writing of complete executable UML models. Due to low market demand, this idea was abandoned, and today ALF serves primarily as an action language in the context of SysML model simulations. So, this is also the primary intention of its use in the MagicDraw environment.

V. Studies

Several studies show the possibilities of the ALF language and the technologies associated with it. Several sources have been studied for this work, and several of them are worth mentioning.

A. Combining ALF and UML in Modeling Tools

One of these works is [10], which deals with the use of the ALF language in the Papyrus environment. It briefly describes the ALF language, the Papyrus environment, its limits, and demonstrates its use on the example of a product order model. At the end of the work, it is mentioned that the Papyrus environment provides only essential functions for describing models using the ALF language. It will still take some time if new generation tools are available that would match the common use of programming languages.

B. Executable Modeling with fUML and ALF in Papyrus: Tooling and Experiments

[11] describes the use of ALF and fUML in Papyrus. This work has a similar topic to [10], but it focuses on experimentation and describes ALF's limits.

C. On Open Source Tools for Behavioral Modeling and Analysis with fUML and ALF

Another interesting work is [12], which describes available tools, that can be used for the ALF language. As mentioned in the work of [10], the conclusion is that although the instruments exist, their possibilities are limited, and they are still in the incubation phase.

Of other studies and articles studied for this work, the most relevant sources of information were [5], [13], and [14]. In these three works, it was possible to generate code from ALF, using different approaches.

 ISBN: 978-1-61208-997-3

D. On the Generation of Full-fledged Code from UML Profiles and ALF for Complex Systems

[14] deals with the generation of C++ code for more complex systems. It uses a combination of ALF language, CHESS tool, and the CHESS-ML modeling language. The CHESS-ML model is a model defined by a UML profile that describes the structure. Behavior is defined using ALF. The work describes the overall transformation, which includes the generation of structural and behavioral aspects of the modeled system. A simplified Asynchronous Transfer Mode (ATM) Adaptation Layer 2 (AAL2) subsystem used in telecommunications for voice transmission was adopted to demonstrate code generation. The code generation consists of a set of transformations such as the conversion of a CHESS-ML model into an unspecified Intermediate Model (InterM) using a model-to-model transformation (M2M) using QVT. Further transformation of the behavior model is defined in the ALF language into InterM using QVT to extend the existing model. The code generation is handled by a model-to-text (M2T) transformation using Xpand [15]. For the needs of transformation between individual models, transformation rules were created within the work, which consisted of approximately 6000 lines of code.

E. Unifying Modeling and Programming with ALF

Standard tools for working with the ALF language are not used in [13]. Instead, they implemented their tools. As described in work, for development in ALF language, it is possible to create own text editor using Eclipse framework Xtext [16], which is primarily used to develop programming and domain-specific languages. It allows to build a language infrastructure directly above the Eclipse environment, including linking, parsing and code validation, syntax highlighting, and more. It also allows to modify standard environment components using Xtend [17]. For the ALF language needs, it is necessary to implement own validation rules for displaying meaningful error messages and modify grammar rules in Xtext, given that ALF uses left-recursive rules, and the Xtext parsing generator does not support them by default. It is also necessary to modify the rules for determining the scope for the needs of importing ALF elements and implement their own type system [13]. Regarding code generation, ALF code is not generated directly, but using model transformations. The idea of transformations is that the ALF model is transformed with the help of transformation rules into another model, specifically the MoDisco Java model. The code is generated with the help of existing generators. The transformation between the ALF model and the MoDisco Java model is performed using the Atlas Transformation Language (ATL) [18], in which it is necessary to define own rules. The paper states that the creation of transformation rules was problematic, mainly because the abstraction of the ALF language is significantly higher than in Java. There were problems with access modifiers, code navigation, and others. In total, the implementation of the ATL rules consisted of more than 9000 lines of code [13].

F. On the automated transnational execution of the action language for foundational UML

In [5], the authors chose a different approach to this problem. No rules have been created to transform an ALF model into another model, but a tool has been created that generates C++ code from an ALF model. This is the first solution that automatically generates code directly from the ALF language. Their solution allows the translation of ALF behavior concepts within the minimum syntactic match. It provides a subset of the ALF language that can be used to describe behavior within the UML model. This includes only the options available in the procedural programming [19]. It also allows the translation of ALF units used to describe the structure of the model (namespaces, packages, classes, operations, and properties). It also provides a memory management system based on the smart pointers principle, a type deduction mechanism, and a scope mechanism. The transformation process can take place in two scenarios.

- **Scenario 1** - Structure and behavior are written using the ALF language. In terms of structure, code is directly generated from ALF units, and types are derived using a deduction mechanism. Three actions can be performed to generate behavior. In the case of the behavior described in ALF, the generator starts transforming the model into text, which results in the C++ code. If the behavior is defined in the C++ language, this block is only copied to the resulting file. In the case the behavior is described in another language, the generator will not take any action.
- **Scenario 2** - The structure is defined by UML elements, the behavior is defined using the ALF language. The translation of the structure defined in UML takes place with the help of a transformation, which is not specified in work. To translate the behavior, the generator starts the transformation of the model into text, as in Scenario 1.

ALF syntax transformation is performed by mapping ALF concepts to C++. In the discussed work, the mapping tables were created according to the place where the translation is performed. These mappings are in the form `<ALF code, C++ code>` and include pairs of qualified names, various expressions, name declarations, conditions, loops, class definitions, operations, indexing, and more. The functionality of this generator was tested on a robot system consisting of two classes written in ALF [5].

VI. Evaluation

Regarding the implementation of the conference review system, it is not possible to capture the application's complexity using the ALF language. No work has been found to create web systems or tools to make this possible. What is possible is to create a skeleton in the Papyrus environment, without considerable functionality. Most applications need to manipulate the data stored in the database. For these tasks, it is necessary to have libraries that allow such operations, but none have been found. Following an unsuccessful search, an inquiry was raised with OMG, which confirmed that there were no libraries to connect to the database or other subsystems.

 ISBN: 978-1-61208-997-3

At the beginning of ALF's development, there were plans to design a Ruby on Rails-style version of ALF that would allow the development of Service-Oriented Architectures (SOAs). Still, due to other business priorities, it was abandoned over time. The only option would be to implement the libraries manually. Another problem is that there are no freely available tools to generate code from the ALF language. So even if the whole system could be described in the ALF, it would be necessary to implement either your own transformation rules to convert the ALF model to model X or to design your code generator.

Communication with OMG revealed an Interaction Flow Modeling Language (IFML) for modeling web-based user interfaces. Furthermore, the communication showed that OMG managed to demonstrate the IFML frontend connection to the back-end business logic implemented using unspecified xUML in a simulated environment. So far, the possibilities of these technologies are the subject of discussion and active research. There is currently no solution on how to generate web systems directly from ALF code or UML models.

Continuous testing of technologies, a study of materials, communication in various discussion groups, communication directly with OMG, and technical support of MagicDraw, it was found that it is impossible to create and generate complex systems using the ALF language. Although there are studies where it was possible to generate functional code from the ALF language, for this to be possible, it is necessary to implement your tools or transformation rules that would allow this.

VII. Future Work

One of the fundamental questions for the successful adaptation of MDE techniques for real software projects is the connection of standard and already used modeling techniques with tools to support simulation verification, preferably in real conditions, and generate complex pieces of code in the implementation language.

Adapting standard techniques, such as UML models or simulation of state diagrams, will facilitate the transition from conventional practices to MDE. Designers will not be faced with the question of entirely new techniques. At the same time, tool support must be available to support the above concepts, ideally combined with existing development environments. As mentioned in the article, the basics of these tools often exist as a plugin to existing tools. Still, they are not very connected to the original environment, and it is always necessary to learn an entirely new concept or formalism for modeling and design.

Another problem is the different variants of the new specification languages, which should allow universal use for different target environments. This approach seems difficult to implement. An important aspect is a possibility of incorporating a programming language into specification models, which will facilitate their comprehensibility and subsequent transformation. Thus, it is not necessary to create new specification languages but to allow existing ones to describe the system behavior, which could, if necessary, be replaced by another one. Although the design system conceived in this way lacks universality, on the other hand, it increases its practical applicability and applicability to a broader part of the community of software system developers.

VIII. Conclusion

The ALF language can be used to simulate models in SysML, which is its primary use now. It is also possible to implement simple programs using two approaches, either alone based on the reference implementation of ALF or in combination with ALF and UML. However, this language is not suitable for implementing complex systems due to limited capabilities and missing tools.

The ALF language is relatively clumsy, necessary libraries offer limited capabilities, and there are no third-party libraries to facilitate development. Due to the fact that ALF is not very widespread, the only reliable guide is the official standard of ALF and academic studies that have dealt with this language. There are also only a limited number of environments in which ALF can be developed. These environments alone are insufficient, so various extensions need to be installed. Combining these environments is much more complicated than programming. Due to limitations, the ALF language cannot describe the whole system, but only part of it. There are no libraries that allow, for example, authorization, database connection, or communication via the REST API. If we look at the development in terms of time, the case study's preparation and implementation using a standard approach took about a month. In the case of the ALF approach, exploring its possibilities, studying the issue, finding relevant resources, reading various studies, and testing multiple technologies and environments took over three months.

Regarding the implementation of the conference review system, it is impossible to capture the application's complexity using the ALF language. No work has been found to make this possible. What is possible is to create a system skeleton, without considerable functionality. The ALF language can be used to simulate models in SysML or to implement simple programs. However, this language is not suitable for implementing complex systems due to limited capabilities and missing tools. Adapting standard techniques, such as UML models or simulation of state diagrams, will facilitate the transition from conventional practices to MDE. At the same time, tool support must be available. Another problem is the different variants of the new specification languages, which should allow universal use for different target environments. It is not necessary to create new specification languages but to let existing ones to describe the system behavior.

Acknowledgment

This work has been supported by the internal BUT project FIT-S-20-6427.

References

[1] J. Bezivin, "Model Driven Engineering: An Emerging Technical Space," Generative and Transformational Techniques in Software Engineering, GTTSE. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, vol. 4143, 2005, pp. 36–64.

 ISBN: 978-1-61208-997-3

[2] Object Management Group, "Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification," https://www.omg.org/spec/QVT/1.0/PDF, [online; retrieved: September, 2022].
[3] S. J. Mellor and M. J. Balcer, Executable UML: A Foundation for Model-Driven Architecture, 2002.
[4] T. Mayerhofer, P. Langer, and M. Wimmer, "xMOF: Executable DSMLs Based on fUML," Software Language Engineering, SLE. Lecture Notes in Computer Science. Springer., vol. 8225, 2013, pp. 56–75.
[5] F. Ciccozzi, "On the automated translational execution of the action language for foundational uml," Software and Systems Modeling, vol. 17, no. 4, 2018, doi: 10.1007/s10270-016-0556-7.
[6] S. S. Nejati and M. Maleki, "Report on How to Use ALF Action Language and fUML execution/debugging with Moka," https://wiki.eclipse.org/images/5/5a/ALF_fUML_Moka.pdf, [online; retrieved: September, 2022].
[7] "ALF Reference Implementation," http://modeldriven.github.io/Alf-Reference-Implementation/, [online; retrieved: September, 2022].
[8] "ALF Language - Getting started," https://docs.nomagic.com/display/ALFP185/Getting+Started, [online; retrieved: September, 2022].
[9] "Code Engineering," https://docs.nomagic.com/display/MD190/Code+Engineering, [online; retrieved: September, 2022].
[10] E. Seidewitz and J. Tatibouet, "Tool paper: Combining alf and uml in modeling tools an example with papyrus," in 15th Internation Workshop on OCL and Textual Modeling, MODELS 2015, pp. 105–119, [online; retrieved: September, 2022]. [Online]. Available: http://ceur-ws.org/Vol-1512/paper09.pdf
[11] S. Guermazi, J. Tatibouet, A. Cuccuru, S. Dhouib, S. Grard, and E. Seidewitz, "Executable modeling with fuml and alf in papyrus: Tooling and experiments," in 1st Internation Workshop on Executable Modeling, MODELS 2015, pp. 3–8, [online; retrieved: September, 2022]. [Online]. Available: http://ceur-ws.org/Vol-1560/paper1.pdf
[12] Z. Micskei, R.-A. Konnerth, B. Horvth, O. Semerth, A. Vrs, and D. Varr, "On open source tools for behavioral modeling and analysis with fuml and alf," in 1st Workshop on Open Source Software for Model Driven Engineering, MODELS 2014, pp. 31–41, [online; retrieved: September, 2022]. [Online]. Available: http://ceur-ws.org/Vol-1290/paper3.pdf
[13] T. Buchmann and A. Rimer, "Unifying modeling and programming with alf," in SOFTENG 2016: The Second International Conference on Advances and Trends in Software Engineering, 2016, pp. 10–15.
[14] F. Ciccozzi, A. Cicchetti, and M. Sjdin, "On the generation of full-fledged code from uml profiles and alf for complex systems," in 12th International Conference on Information Technology - New Generations, 2015, pp. 81–88.
[15] "Eclipse Model To Text Project," https://www.eclipse.org/modeling/m2t/?project=xpand, [online; retrieved: September, 2022].
[16] "Eclipse Xtext Project – A Framework for Development of Programming Languages," https://www.eclipse.org/Xtext/, [online; retrieved: September, 2022].
[17] "Eclipse Xtend – A Flexible and Expressive Dialect of Java," https://www.eclipse.org/xtend/, [online; retrieved: September, 2022].
[18] "Eclipse ATL – A Model Transformation Technology," https://www.eclipse.org/atl, [online; retrieved: September, 2022].
[19] OMG, Action Language for Foundational UML, Version 1.1, 2017, [online; retrieved: September, 2022]. [Online]. Available: https://www.omg.org/spec/ALF/1.1/PDF

 ISBN: 978-1-61208-997-3