Towards automated support for blended modelling of UML-RT embedded software architectures

Malvina Latifaj[1], Federico Ciccozzi[1], Mattias Mohlin[2] and Ernesto Posse[3]

[1]Mälardalen University, Högskoleplan 1, 722 20 Västerås, Sweden
[2]HCL Technologies, Gustav Adolfs torg 12, Malmö, Sweden
[3]Zeligsoft, 259 St Joseph Blvd, Suite 208 Gatineau, Quebec J8Y 6T1 Canada

Abstract

The Unified Modeling Language for Real Time (UML-RT) is a UML-based domain-specific language for modelling real-time embedded systems. HCL RTist, a model-based development environment for creating complex, event-driven and real-time software with advanced automation features provided by HCL Technologies, provides advanced support for UML-RT. Historically, as for the majority of UML profiles, editing support for UML-RT has also mainly exploited graphical notations (e.g., composite component and state-machine diagrams). Nevertheless, our previous experiments with blended graphical and textual modelling showed that the seamless use of different notations (i.e., graphical and textual) can significantly boost the work of architects and modellers. The results of those experiments together with the exposed wish of RTist customers of being able to design software architectures and applications via multiple notations led us to initiate this work towards an automated support for blended modelling of UML-RT. In this paper we describe the first step of the work – the effort of designing, implementing and integrating a textual notation for UML-RT state-machines in RTist.

Keywords

UML-RT, HCL RTist, Xtext, blended modelling, textual modelling, graphical modelling

1. Introduction

Software steers our daily life and methods for architecting it in more effective and efficient ways have been the focal point of research in software architecture for decades now. Automation is an inescapable ingredient of efficient architecting and software is not an exception. In the architecting of software, automation boosts the throughput, as well as improves the quality of the results, of virtually any architecting task, from requirements specification to maintenance, through design, development and validation. While automation relieves the architect from tedious and time-consuming activities, abstraction in terms of modelling allows to focus on the problem at hand from a more human-oriented perspective than programming. Abstraction and automation are considered to be the core pillars of Model-Driven Engineering. Models are first-class entities of the architecting and engineering process that abstract from certain

ECSA'21: 15th European Conference on Software Architecture, September 13–17, 2021, Virtual
malvina.latifaj@mdh.se (M. Latifaj); federico.ciccozzi@mdh.se (F. Ciccozzi); mattias.mohlin@hcl.com (M. Mohlin); eposse@gmail.com (E. Posse)
http://www.es.mdh.se/staff/4313-Malvina_Latifaj (M. Latifaj);
http://www.es.mdh.se/staff/266-Federico_Ciccozzi (F. Ciccozzi)
0000-0002-2754-9568 (M. Latifaj); 0000-0002-0401-1036 (F. Ciccozzi); 0000-0003-1302-9497 (E. Posse)

aspects of the problem at hand; models are automatically manipulated via transformations for multiple purposes, from validation to code generation, but also communication among different stakeholders.

Commonly, domain-specific abstractions described in Domain Specific Modelling Languages (DSML) [1] are leveraged to allow domain experts, who may or may not be software experts, to express complex functions in a domain-focused and human-oriented way than if using traditional programming languages. DSMLs formalise (for computer-based analysis and synthesis purposes) the *communication* language of architects at the level of domain-specific concepts such as an engine and wheels for a car. UML is the most used architecture description language in industry [2], the de-facto modelling standard in industry [3], and an ISO/IEC (19505-1:2012) standard. It is general-purpose, but it provides powerful profiling mechanisms to constrain and extend the language to achieve UML-based DSMLS, called UML profiles; in this paper, we focus on the UML real-time profile (UML-RT)[4], and its implementation in an industrial tool, HCL RTist[1].

1.1. Problem, motivation, and the RTist case

Domain-specific modelling tools, like RTist, traditionally focus on one specific editing notation (such as text, diagrams, tables or forms). This limits human communication, especially across stakeholders with varying roles and expertise. Moreover, architects and engineers may have different notation preferences; not supporting multiple notations negatively affects their throughput. Besides the limits to communication, choosing one particular kind of notation has the drawback of limiting the pool of available tools to develop and manipulate models that may be needed. For example, choosing a graphical representation limits the usability of text manipulation tools such as text-based diff/merge, which is essential for team collaboration. When tools provide support for both graphical and textual modelling, it is mostly done in a mutual exclusive manner. Most off-the-shelf UML modelling tools, such as IBM Rational Software Architect[2] or Sparx Systems Enterprise Architect[3], focus on graphical editing features and do not allow seamless graphical–textual editing. This mutual exclusion suffices the needs of developing small scale applications with only very few stakeholder types. RTist is not an exception. It provides support for modelling UML-RT architectures and applications based on graphical *composite structure diagrams*, to model structure, and *state-machine diagrams*, to model behavior. In addition, the implementation of UML-RT in RTist provides support for leveraging C/C++ action code for the description of fine-grained, algorithmic, behaviors within graphical state-machines. That is needed to enable the definition of full-fledged UML-RT models from which executable code can be automatically generated. While providing means to model graphical entities and "program" algorithmic behaviours textually, the two are disjoint, since the modelling of UML-RT is graphical only and the textual C/C++ is injected in graphical models as "foreign" entity and with almost no overlapping with graphical model elements. The aim is instead to achieve a modelling tool that is able to make different stakeholders to work on overlapping parts of the models using different modelling notations (e.g., graphical and textual)

[1] https://www.hcltechsw.com/rtist

[2] http://www-03.ibm.com/software/products/en/ratsadesigner/

[3] https://sparxsystems.com/

in an automated manner.

1.2. Paper contribution

In this paper we describe the first step towards providing a fully blended graphical-textual modelling environment for UML-RT in RTist. Our experiments in a previous study with blended graphical-textual modelling showed that the seamless use of different notations can significantly boost the architecting of software using UML profiles [5]. The results of those experiments together with the exposed wish of RTist customers of being able to design software via multiple notations led us to initiate this work towards an automated support for blended modelling of UML-RT in RTist. In this paper we focus on the design, implementation and integration of a textual notation for UML-RT state-machines in RTist.

2. Blended modelling: what and why

We have previously defined the notion of *blended modelling* [6] as:

> *the activity of interacting seamlessly with a single model (i.e., abstract syntax) through multiple notations (i.e., concrete syntaxes), allowing a certain degree of temporary inconsistencies.*

A seamless blended modelling environment, which allows stakeholders to freely choose and switch between graphical and textual notations, can greatly contribute to increase productivity as well as decrease costs and time to market. Such an environment is expected to support at least graphical and textual modelling notations in parallel as well as properly manage synchronisation to ensure consistency among the two. The possibility to visualise and edit the same information through a set of diverse perspectives always in sync has the potential to greatly boost communication between stakeholders, who can freely select their preferred notation or switch from one to the other at any time. Besides obvious notation-specific benefits, such as for instance the possibility to edit textual models in any textual editor outside the modelling environment, a blended framework would disclose the following overall benefits.

Flexible separation of concerns and better communication. Providing graphical and textual modelling editors for different aspects and sub-parts (even overlapping) of a DSML like UML-RT enables the definition of concern-specific architectural views characterised by either graphical or textual modelling (or both). These views can interact with each other and are tailored to the needs of their intended stakeholders. Due to the multi-domain nature of modern software systems (e.g., cyber-physical systems, Internet-of-Things), this represents a necessary feature to allow different domain experts to describe specific parts of a system using their own domain-specific vocabulary and notation, in a so called *multi-view modelling* [7] fashion. The same information can then be rendered and visualised through other notations in other perspectives to maximise understanding and boost communication between experts from different domains as well as other stakeholders in the development process.

Faster modelling activities. We have experimented with blended modelling of UML profiles [5] and the seamless combination of graphical and textual modelling has shown a decreased modelling effort in terms of time thanks to the following two factors:

1. Any stakeholder can choose the notation that better fits her needs, personal preference, or the purpose of her current modelling task, at *any time*. For instance, while structural model details can be faster to describe by using diagrammatic notations, complex algorithmic model behaviours are usually easier and faster to describe using textual notations (e.g., Java-like action languages).
2. Text-based editing operations on graphical models[4], such as copy&paste and regex search&replace, syntax highlighting, code completion, quick fixes, cross referencing, recovery of corrupted artefacts, text-based diff and merge for versioning and configuration, are just few of the features offered by modern textual editors. These would correspond to very complex operations if performed through graphical editors; thereby, most of them are currently not available for diagrams. Seamless blended modelling would enable the use of these features on graphically-described models through their textual editing view. These would dramatically simplify complex model changes; an example could be restructuring of a hierarchical state-machine by moving the insides of a hierarchical state. This is a demanding re-modelling task in terms of time and effort if done at graphical level, but it becomes a matter of a few clicks (copy&paste) if done at textual level.

3. A textual notation for UML-RT state-machines

In this paper we introduce a textual notation for UML-RT state-machines that is intended to be part of future RTist release. UML-RT is a real-time profile that aims to simplify the ever-increasing complex software architecture specification for real-time embedded systems. The UML-RT concepts are inherited from the ones defined in the Real-time Object-Oriented modeling Language (ROOM) [4] and represented using UML extensibility mechanisms. UML-RT enables both *structure modelling* and *behavior modelling* of real-time systems [8]. The structural part is represented using composite structure diagrams, whereas the behavioral part is represented using state-machine diagrams. The fundamental concepts of UML-RT are capsules, which are encapsulated active entities that can execute in parallel.

UML-RT relies on state-machines for the modelling of capsules' behaviour. In the case of a missing state-machine, the capsule only operates as container for other sub-capsules. A behavioral state-machine in UML-RT is composed of states, pseudo states, and transitions. States can be simple or composite, and the presence of composite states results in a hierarchical state-machine. Pseudo states consist of the initial pseudo state that acts as the starting point of the state-machine, and choice and junction pseudo states where guards on outgoing transitions determine which one to execute next. The remaining pseudo states (i.e., entry, exit, and history) are only used in hierarchical state-machines. Entry and exit pseudo states are used to enter and exit composite states, while history pseudo state is used to invoke the last active state prior to the exit of the composite state. Transitions indicate a change of state and can contain triggers that initiate transitions in the form of events, guard conditions that must evaluate to true for the initiation of the transition, and effects.

[4]Please note that by *graphical/textual model*, we intend a model rendered using a graphical/textual notation.

3.1. Textual language workbench

To complement the existing graphical editor in RTist with a textual notation and editor, a language workbench for such purpose needed to be carefully selected. HCL RTist is an Eclipse-based environment that leverages the Eclipse Modeling Framework (EMF)[5] as a backbone. Thereby, by choosing an EMF-based language workbench, we could leverage EMF as a common data layer. For this reason, we chose Xtext[6], a framework for the development of textual DSMLs, based on EBNF grammars. The textual editor supports an outline view, syntax highlighting, error checking, quick-fix proposals, and many other features provided by Xtext. Furthermore, Xtext provides code completion for keywords and cross-references by increasing the usability of the language and decreasing the learning curve.

3.2. Definition of a textual notation

Our goal was to introduce a textual notation (and related editor) to the already existing UML-RT profile supported by RTist. A possible alternative was to use this underlying metamodel consumed by the RTist's graphical editor as an input for an Xtext plugin to automatically generate a textual editor. However easy to implement, this process generates erroneous and unintuitive grammar, too far from the expectations of RTist's users. Manually editing this generated grammar would have been a tedious and potentially error-prone process. Therefore, we decided to design a textual notation, in terms of an Xtext grammar, from scratch. Starting from a wish-list of RTist's customers and architects, and using the UML-RT metamodel portion describing state-machines as blueprint, we manually defined of our UML-RT textual notation for state-machines in Xtext. The steps needed for the definition of the grammar were the following.

Identify reserved keywords. When defining a DSML, it is crucial to identify the reserved keywords used to typify the core concepts of the language. The importance of these keywords lies in improved readability, higher language familiarity, and efficient parsing as they serve as directives for specific concepts. The chosen keywords for the textual syntax for UML-RT state-machines are the following: *capsule, statemachine, state, initial, junction, choice, entry, exit, entrypoint, exitpoint, history, transition, when, on* and *inherits*.

Elements' ordering strategy. Even though it is not mandatory for our language to have a fixed order of elements, this approach enhances readability and navigation of the textual syntax, as well as increased predictability on where the elements created in other notations will be placed in the textual syntax. Our grammar is based on the vertical distance approach where elements that affect each other's understandability and are closely related [9], are grouped together and have a low vertical distance. Furthermore, being that this grammar prohibits cross-references before element declaration, we take the aforementioned statement into consideration and make sure that elements that need to be cross-referenced will be declared before the cross-reference takes place.

A spoonful of syntactic sugar- The majority of programming languages, including C++, that is used as action code for behavioral state-machines, makes use of statement terminators in the form of semi-colons. Being that one of the main goals when introducing this textual syntax

[5] https://www.eclipse.org/modeling/emf/
[6] https://www.eclipse.org/Xtext/

is for developers to use it jointly with the C++ action code, we introduced consistent use of semi-colons for indication statement termination to make the grammar more conforming to C++ and to increase readability. For the same readability reasons and developers' preferences, we also introduce colons after transition names. Furthermore, to make the grammar more compact, we allow the declaration of multiple objects of the same type in one single line of code. Due to the combination of the textual syntax with action code, we need to handle C++ code blocks so we can "isolate" them and make them distinguishable from the rest of the grammar. For this reason, we include backticks in order to enclose code snippets and to make the lexer aware of where the code block begins and ends.

The overall goal during this process was to keep a fixed concrete syntax while simultaneously enhancing the abstract syntax, even though frequently we had to trade-off between ease of expression in the concrete syntax and extra complexity in the abstract syntax.

3.3. Scoping

Scoping in Xtext is concerned with the visibility of elements; therefore, the scope provider computation returns the target candidates that are visible in the current context and by a given reference. In order to enforce the UML-RT's modularity it is necessary to specify a custom scope provider. The default behavior of Xtext allows establishing a cross-reference to all the elements of a particular type that are located inside the same Eclipse resource (i.e., project). By customizing the scope provider, we restrict this behaviour, and only allow cross-references for elements declared in the same model file. The rationale behind this decision lies in the fact that multiple model files containing different capsules can be located inside the same resource, and a particular capsule should not be able to cross-reference elements of other capsules. However, a key concept in which UML-RT relies on to reuse and extend parts of existing state-machines is the inheritance mechanism. When capsule A inherits capsule B, the state-machine of capsule A implicitly inherits the state-machine of capsule B. Therefore, to support inheritance, we need to customize the scope provider so that it allows cross-references for elements not only from the capsule itself, but also from the inherited capsule, in case there is one.

Another default behavior of Xtext consists in allowing cross-references for all elements of a particular type declared in the same model file, regardless of their level of nesting. This contradicts an important UML-RT concept; compound transitions. Since transitions in UML-RT state-machines can not cross state boundaries, the concept of compound transitions is applied, consisting of multiple segments that are connected by means of pseudo-states. However, with the default behaviour of Xtext, a transition can cross state boundaries. Therefore, the scope provider is customized to restrict that, and provide the desired behavior in conformance with UML-RT concepts, by allowing transitions to only cross-reference pseudo states and states that are on the same level of nesting as the transition, or their immediate entry and exit points.

3.4. Integration in RTist

By customizing the Xtext ASTFactory and Linker it becomes possible to map the syntax to the existing environment for UML-RT modelling in RTist. This means that RTist treats a textual

state-machine in exactly the same way as a graphical state-machine. The state-machine AST created by the Xtext parser must be inserted into the proper model context, which is the capsule to which the state-machine belongs. The Xtext parser recreates the AST every time the state-machine textual model is modified (or to be more accurate, a short time after each consecutive sequence of text modifications). Each newly created AST will contain cross-references that target elements both within the AST itself as well as elements contained in the RTist model. As soon as the AST is inserted into a capsule, RTist will treat this state-machine in the same way as a state-machine that was modelled graphically (since the model representation is identical).

4. Challenges

There are some challenges involved in synchronizing changes across graphical and textual state-machine models. These challenges are identified during the process of integrating the textual notation in RTist and are described in the following sections.

4.1. Incoming cross-references

The user may create cross-references that target elements in the state-machine. For example, she may create a dependency from a capsule to one of the states in the state-machine. If attention is not paid when inserting the state-machine into the capsule, such cross-references could break. For instance, if the state-machine is updated by simply deleting the old version and then inserting the new updated version this will happen, since the deletion will trigger clean-up of cross-references (i.e., the tool "thinks" that the state was deleted and hence resets the dependency to avoid a broken reference).

4.2. Model information not represented in the textual notation

The textual state-machine notation does not cover the entire UML-RT metamodel. That is, there are certain pieces of information that cannot be specified using the textual syntax, but which other views in RTist may allow to view and edit. One example is UML stereotypes which may be applied to any model element, including elements in a state-machine. When the state-machine model gets updated due to a change in the textual notation it is important not to lose this additional information. Both these problems are related to how the state-machine graphical model is updated when the textual model changes. The solution is to "merge" the changed parsed AST into the RTist graphical model, rather than replacing it (i.e., updating it by a delete followed by an insert).

4.3. Model element unique identifiers

It is important to ensure that URIs of elements in the AST are stable. More specifically, the last part of an EMF URI, the so called fragment, which identifies the element within its file, shall remain the same. RTist uses by default random unique IDs as fragments, but this obviously does not work for AST elements, since it would mean that all AST elements get new URIs each time the textual state-machine is modified (and hence it becomes impossible to keep incoming

cross-references bound). To solve this, an Xtext fragment provider was implemented. Its job is to assign fragments using fully qualified names instead. The alternative would have been to make the fragment strings visible in the textual notation, which is obviously not an option from a user-friendliness point of view.

4.4. Code formatting and comments

Just like it is possible to have additional information in the UML-RT state-machine graphical model that is not carried by its textual counterpart, the opposite is also possible. A textual state-machine model may have lexical entities that the parser would not reflect in the AST. Typical examples include code formatting (i.e., indentations, use of newlines etc) and comments. If a state-machine is modified in another way than through the textual editor, it is necessary to serialize the updated model and then update the textual model in a way that preserves code formatting and comments. This problem is not fully solvable in an automated manner since there usually is no formalized means for how code formatting and comments are used. Xtext provides an approach for serialization that attempts the preservation of as much code formatting and comments as possible. If the user has experienced how this algorithm works in practice, she could probably adjust the code formatting and use of comments to avoid losing information when the model is serialized. Anyhow, we will look into potential semi-automated solutions for this.

5. Related Work

The Action Language for Foundational UML (Alf) [10] is a textual language standardized by the Object Management Group (OMG) for representing UML models. Since its underlying semantics are indicated by a limited subset of UML named Foundational UML (fUML), the Alf syntax is restricted within its bounds and does not support state-machines as they are not available in the fUML subset. tUML is a textual language for a limited subset of the standard UML metamodel targeted at real-time embedded systems that consists of class diagrams, composite structure diagrams, and state diagrams. The implementation of tUML has been carried out to have a very close proximity to the UML metamodel. Consideration has been given to propose tUML to OMG as an extension of Alf, being that the latter lacks support for state-machines [11]. There also exists a plethora of tools and modeling languages that support textual notations for UML models. Earl Grey [12] is a textual modeling language that supports the creation of UML class and state models. MetaUML [13] is a MetaPost library for creating UML diagrams using textual notations, and it supports class diagrams, package diagrams, component diagrams, use case diagrams, activity diagrams, and state diagrams. The textual notation is not only used to define the elements and their relationships but also their layout properties. PlantUML [14] is an open-source tool that supports the generation of both UML and non-UML diagrams from a textual language. Among the most important UML diagrams they support are sequence diagrams, class diagrams, activity diagrams, state diagrams, and more. Umple [15] is an open-source modeling tool that can be used to add UML abstractions to programming languages (i.e., Java, C++, PHP, and Ruby) and create UML class and state diagrams from a textual notation. The generated graphical view for class diagrams can be edited, while for state-machines, it is

read-only. Textual, executable, and translatable UML (txtUML) [16] is an open-source project that supports the creation of models from a textual notation and generates the corresponding graphical visualization. TextUML Toolkit [17] is an open-source IDE that allows the creation of UML2 models from a textual notation. This toolkit is available on Cloudfier, as a plug-in for Eclipse IDE and as a standalone command-line tool.

There have been a handful of attempts at providing textual syntax for UML-RT, and we have been involved with some of them. Calur [18] provides a textual syntax only for UML-RT's action language, not state-machines. Unlike our approach, both eTrice[7] and Papyrus-RT[8] provide a kind of all-or-nothing approach. They both provide syntax for both structure and behaviour, but the entire model is described as either textual or graphical, whereas in our approach the user can select only parts of the model to be represented textually. This allows the user to retain the ability to use existing RTist tooling for graphical modelling. Note also that our textual notation for UML-RT state-machines has been designed and implemented to maximise user experience of architects and engineers, as their throughput thanks to the possibility of blended modelling.

6. Outlook

The provision of textual modelling as complement to the existing graphical modelling for UML-RT state-machines is the first step towards a fully blended modelling environment for UML-RT in RTist. The next planned steps concern the extension of the textual notation to the rest of UML-RT concepts as well as the provision of more effective synchronisation mechanisms. Moreover, we plan to investigate the possibility to leverage the Language Server Protocol, and in particular the Graphical LSP in Eclipse[9] to provide a web-based modelling solution too. In addition, we intend to implement this approach using other graphical modelling tools (e.g., Sirius, GMF) and compare the results.

The involved architects at HCL have confirmed a satisfactory result achieved via the textual notation, and they were involved in the effort from start to end. Nevertheless, to provide a quantification of the improvements brought by blended modelling in RTist, we plan to run additional experiments with a larger number of stakeholders from multiple companies, including RTist's users. We have built an international consortium across 4 countries and running a project in the ITEA3 cluster programme on blended graphical–textual modelling called BUMBLE[10]. In that context, we will run more extensive controlled experiments and industrial case-studies too.

An important element of the dissemination plan consists in leveraging the different opportunities provided in the Eclipse community, including Eclipse conferences (e.g., EclipseCon Europe) and marketing. We will also collaborate with the Eclipse Working Groups, Papyrus and Capella Industry Consortia to reach out to industrial MDE tool users.

We plan to disseminate results via research forums (conferences, workshops), corporate presentations, participation to industrial events like expos, on-line community forums for Eclipse, social media, fact sheets and wikis.

[7] https://www.eclipse.org/etrice/
[8] https://www.eclipse.org/papyrus-rt/
[9] https://www.eclipse.org/glsp/
[10] https://itea3.org/project/bumble.html

Acknowledgments

This work was supported by Vinnova through the ITEA3 BUMBLE project (rn. 18006).

References

[1] G. Mussbacher, D. Amyot, R. Breu, J.-M. Bruel, B. H. Cheng, P. Collet, B. Combemale, R. B. France, R. Heldal, J. Hill, et al., The relevance of model-driven engineering thirty years from now, in: Procs of MoDELS, Springer, 2014, pp. 183–200.

[2] I. Malavolta, P. Lago, H. Muccini, P. Pelliccione, A. Tang, What industry needs from architectural languages: A survey, IEEE Trans. on Soft. Eng. 39 (2012) 869–891.

[3] J. Hutchinson, J. Whittle, M. Rouncefield, S. Kristoffersen, Empirical assessment of MDE in industry, in: Procs of ICSE, IEEE, 2011, pp. 471–480.

[4] B. Selic, Real-time object-oriented modeling, IFAC Proceedings Volumes 29 (1996) 1–6.

[5] L. Addazi, F. Ciccozzi, Blended graphical and textual modelling for uml profiles: A proof-of-concept implementation and experiment, Journal of Systems and Software 175 (2021) 110912.

[6] F. Ciccozzi, M. Tichy, H. Vangheluwe, D. Weyns, Blended modelling – what, why and how, in: MPM4CPS workshop, 2019. URL: http://www.es.mdh.se/publications/5642-.

[7] A. Cicchetti, F. Ciccozzi, A. Pierantonio, Multi-view approaches for software and system modelling: a systematic literature review, Software & Systems Modeling (2019).

[8] E. Posse, J. Dingel, An executable formal semantics for uml-rt, Software & Systems Modeling 15 (2016) 179–217.

[9] R. C. Martin, Clean code: a handbook of agile software craftsmanship, Pearson Education, 2009.

[10] Object Management Group (OMG), Action Language for Foundational UML (Alf), Version 1.1, OMG Document Number formal/2017-07-04 (http://www.omg.org/spec/ALF/1.1), 2017.

[11] F. Jouault, J. Delatour, Towards fixing sketchy uml models by leveraging textual notations: Application to real-time embedded systems., in: OCL@ MoDELS, 2014, pp. 73–82.

[12] M. Mazanec, O. Macek, On general-purpose textual modeling languages., in: Dateso, volume 12, Citeseer, 2012, pp. 1–12.

[13] O. Gheorghies, Metauml: Tutorial, reference and test suite, 2005.

[14] PlantUML Language Reference Guide, Version 1.2021.2, 2021. URL: http://plantuml.com/guide, Last accessed on May 2, 2021.

[15] T. C. Lethbridge, V. Abdelzad, M. H. Orabi, A. H. Orabi, O. Adesina, Merging modeling and programming using umple, in: International Symposium on Leveraging Applications of Formal Methods, Springer, 2016, pp. 187–197.

[16] G. Dévai, G. F. Kovács, Á. An, Textual, executable, translatable uml., in: OCL@ MoDELS, Citeseer, 2014, pp. 3–12.

[17] TextUML Toolkit, 2021. URL: http://abstratt.github.io/textuml/, Last accessed on May 2, 2021.

[18] N. Hili, E. Posse, J. Dingel, Calur: an action language for UML-RT, in: 9th European Congress on Embedded Real Time Software and Systems (ERTS 2018), 2018.