Towards an Extensible Architecture and Tool Support for Model-based Verification

David Delgado
Universidad de Málaga
Málaga, Spain
daviddelgado@uma.es

Lola Burgueño, Javier Cámara, Javier Troya
ITIS Software, Universidad de Málaga
Málaga, Spain
{lolaburgueno,jcamara,jtroya}@uma.es

***Abstract*—Model-based software engineering (MBSE) brings models to the center of software and system design. Models are powerful abstractions used to support all phases of the software development life cycle of complex software. As these models grow larger and their complexity increases, they need to be verified and validated to preserve their correctness. One possible way to do so is by means of the use of formal methods. However, the availability of MBSE tools with support for validation and verification is limited, and they usually require the cumbersome deployment of software burdened by dependencies, preventing the adoption of these tools. This paper presents a web-based architecture designed to support the definition of domain models and provide translation capabilities to different verification formalisms. As a proof of concept for our architecture, we have developed a tool prototype that is light-weight, runs in the browser and supports: (i) definition of domain models represented as class diagrams and (ii) partial translation of class diagrams into the Alloy specification language, enabling verification of structural domain properties. We show how we have used this tool to verify properties for the public bus management system in the city of Málaga, Spain.**

***Index Terms*—Model-based software engineering, verification, web-based modeling tool, structural analysis, domain models, Alloy**

I. Introduction

Model-based software engineering (MBSE) uses models as abstractions that facilitate the software development process. As these models grow larger and become more complex, ensuring their correctness becomes essential.

Formal verification tools for domain models, such as those available for UML, are of utmost importance, because they enable engineers to ensure the correctness and consistency of their models. Specifying and analyzing the satisfaction of structural and behavioral properties (e.g., constraints or invariants) that must hold within the model reduces the risk of design flaws or implementation errors.

Although there are tools that enable formal verification of various classes of properties in domain models [3], [7], [18], these tend to be focused on a specific class of property, such as structural constraints [3], or behavioral properties [7].

Moreover, these tools tend to require local installation of large pieces of software and cumbersome management of complex dependencies that bog down the specification and verification of domain models. We find a similar situation with UML profiles, such as the MARTE profile from the OMG [17], as they need additional expertise and to be integrated in specific tools. Hence, if designers want to verify different types of properties in a domain model, they need to create a complex local ecosystem of tools that enable various verification capabilities, often without any interoperability between them.

As a consequence of the aforementioned situation, we posit that engineers require tools that enable: (i) integrated comprehensive verification of domain models, (ii) interoperability among multiple verification capabilities, and (iii) systematic and transparent management of dependencies and other complex software administration issues.

Building such tools, however, is not straightforward and demands careful thinking about the type of underlying architecture required to enable all the desirable traits mentioned above. To advance in this direction, in this paper we propose a web-based architecture that enables online graphical definition of domain models, as well as the systematic development of translations into multiple formalisms and integration of formal analysis capabilities.

We establish the grounds to claim the feasibility of our architecture by presenting a prototype tool that enables the definition of domain models represented as class diagrams and their partial translation into Alloy, and illustrate its use in the verification of structural properties from the public bus management system in the city of Málaga, Spain.

This paper is organized as follows. In Section II, we present our architecture and describe our-proof-of-concept tool. In Section III we validate our proposal by applying it for the verification of structural properties in the public bus management system of Málaga. Section IV describes the related work. Finally, in Section V, we present our conclusions and future work.

II. Architecture and Tool Support

In this section, we present our architecture which is intended to be the first contribution towards a modeling playground that allows users to easily develop their software models and transform them to different formalisms to perform a variety of tasks, including structural and behavioral verification.

Figure 1 presents our web-based architecture designed to support the definition of domain models graphically and the translation of these models to different verification formalisms. Our approach has been designed to be light-weight and extensible.

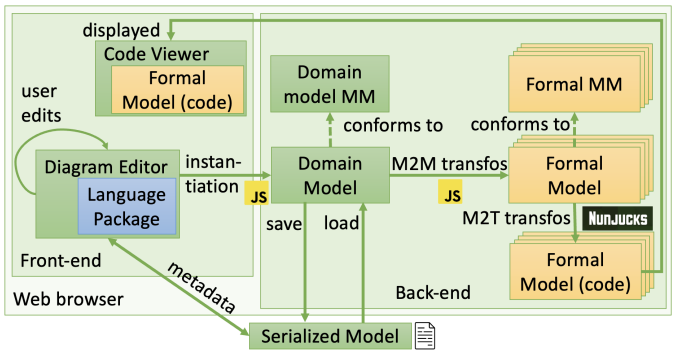

Fig. 1. Generic Architecture

Our architecture considers a solution that runs exclusively on the browser, so nothing needs to be installed in the system, fostering its portability. At a high-level, we can make the distinction between the front-end and the back-end. Using the front-end, the user can interact graphically with their domain models as well as access the code generated from them. Each diagram editor will have its own *language package*, which will integrate the necessary language tools (e.g., parser, syntax highlighter, auto-complete features, code linting).

The back-end contains the internal representation of the domain model metamodel and the domain models created by the user. It also offers an extension mechanism to introduce new metamodels and models that will be used for verification purposes, and supports model-to-model (M2M) and model-to-text (M2T) transformations. It also provides functionalities to serialize (save) and deserialize (load) domain models. As a proof of concept, we have implemented a first prototype of this tool, which supports domain models in form of class diagrams and Alloy [9] as a verification language. In the following we illustrate our architecture and explain the implementation of our proof of concept, which is available on our Github repository[1].

A. Front-end: Web-based Modeling Tool

In the front-end, for the diagram editor, we have used the GoJS library[2]. The basic constructs in GoJS are nodes and links. For the representation of domain models in GoJS, a class is a node, an attribute is a property of a node, and an association is a link. GoJS also has a data model, which is user-defined and allows mapping data to nodes and links. This data model also includes required metadata, e.g., position of the elements on the canvas, color, etc. For saving and loading we use JSON for both the models and their metadata.

The front-end of our tool is shown in Figure 3. It is composed of a modeling canvas on the left-hand side, a toolbar that enables editing the element properties (classes, attributes, associations, etc.) on the bottom of the right-hand side, and a window that displays the generated Alloy code on the upper right-hand side.

[1]https://github.com/atenearesearchgroup/web-model-based-verification-tool
[2]https://gojs.net/latest/index.html

Instead of creating a complete language package for the GoJS graphical editor, in our proof-of-concept prototype, we have created a simplified version of it that we call GoJS *language adapter*. This language adapter uses on-click event listeners that monitor the graphical diagram editor and toolbar. Every time the diagram is edited by the user, the corresponding event is captured by the event listener, and a transaction is created. Transactions contain information about the changes made. We use these transactions, in the back-end, to propagate the changes from the editor to the model. When a user modifies the properties of an element using the toolbar these modifications are applied directly to the domain model and propagated to the GoJS diagram through the adapter.

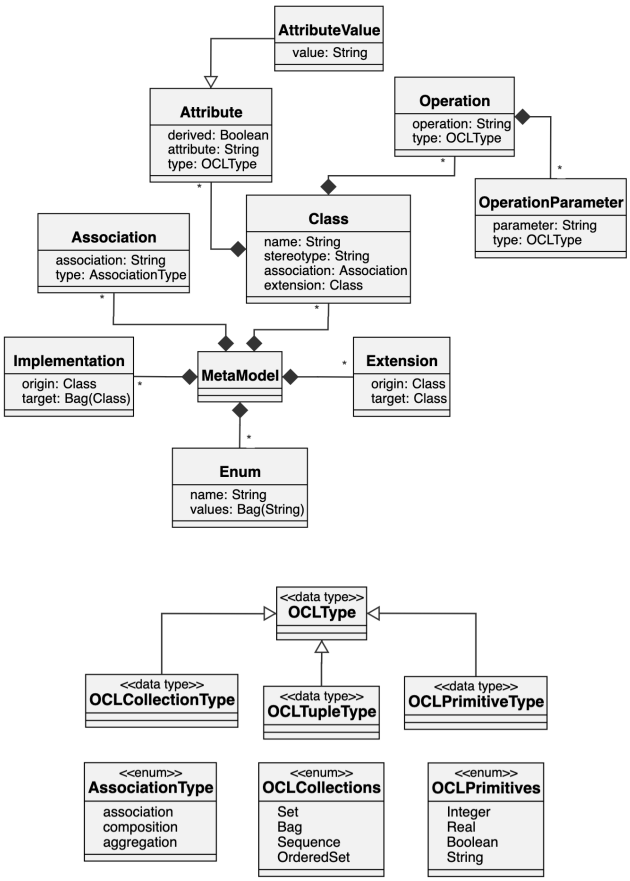

Fig. 2. Metamodel for domain models

B. Back-end: Bridging MBSE and Verification Models

As part of the back-end of our architecture, we have defined a domain model metamodel to represent domain models, which is shown in Figure 2.

While the user is editing a diagram using the graphical interface, the corresponding metamodel is instantiated and/or edited. We propose the use of event listeners and a transaction management system to constantly keep the model synchronized with the editor.

Our architecture supports the definition of user-defined metamodels and models in an extensible manner. When users

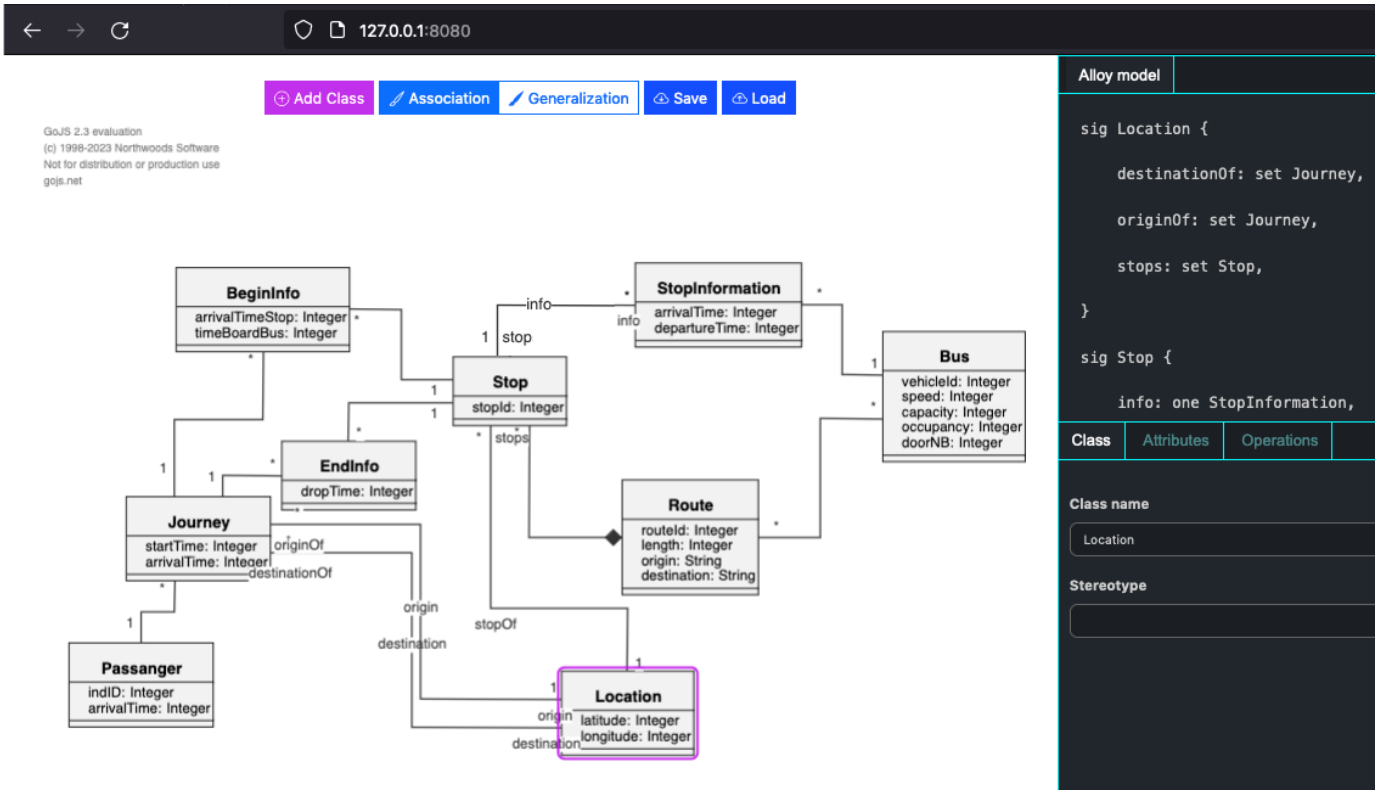

Fig. 3. Screenshot of our tool

need to introduce a new verification language, they only need to add to the tool the metamodel of the language (designated by *Formal Metamodel* in Figure 1), a M2M transformation to transform the domain model to a model that conforms to the formal metamodel, and a M2T transformation that generates code from the formal model.

In the current state of our proof-of-concept tool, the Alloy specification language is available. This is, we added to our tool a metamodel of a simplified version of Alloy, as well as a M2M transformation that maps domain models to Alloy models and a M2T transformation that generates Alloy code from the Alloy model.

Figure 4 shows our simplified Alloy metamodel. Note that the Alloy metamodel is not intended to be an accurate representation of Alloy, but to cover the minimal requirements to create an Alloy model of a system as well as to facilitate the creation of the corresponding M2M and M2T transformations.

Both the M2M and M2T transformations are triggered every time the domain model is updated. This way, both views of our tool (i.e., the graphical diagram and the corresponding formal model) are always synchronized.

C. Mapping domain models to Alloy

The mapping from domain models to Alloy consists in creating an Alloy signature for each class in the domain model. The associations between classes are mapped as Alloy attributes in the corresponding signature. The type of these attributes depends on the cardinality of the association end.

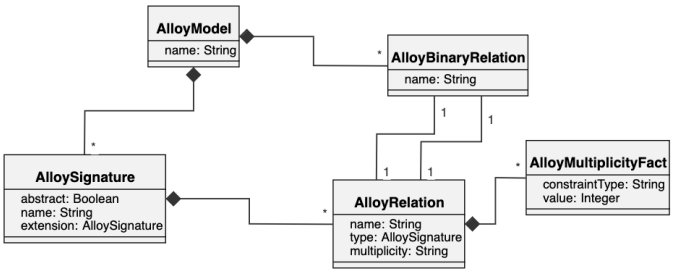

Fig. 4. Alloy metamodel

If the association end has a cardinality of 1, then the Alloy attribute is defined as *one*. If the cardinality of the association end is different than one, then the Alloy attribute is mapped as a set. Since associations are usually bidirectional in domain models, we have designed a mapping that encodes each bidirectional association as a symmetric relation in Alloy. Figure 5 shows a simple example of this mapping visually.

D. Tool Support

To be able to use our tool in the browser and without a server, it has been necessary to use the Javascript (JS) language. We have used JS version ES6 compiled to ES5 by means of RollUp[3].

[3]https://rollupjs.org/

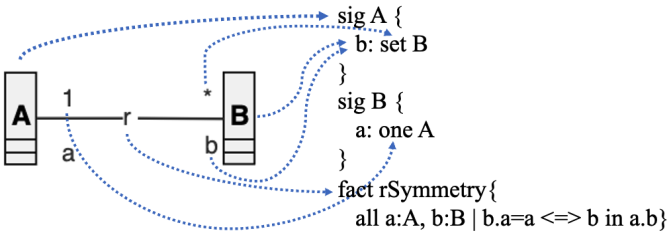

Fig. 5. Mapping from domain models to Alloy

The M2M transformation has been created in JS ES6, while the M2T transformation uses a template system in Javascript called Nunjucks [4]. We chose Nunjucks given its wide adoption, support by the community and the fact that its syntax is used in Jinja2 and Twig (widely used in Python, PHP, TextX, etc.).

III. Validation

We have been granted with a national research project related to urban-digital twins where one of the case studies is about the bus transportation system that runs in our city: Malaga. To validate our proposal, we have used our tool to develop a domain model of this system and to verify some properties.

The domain model of the system was introduced in [15], where we envision a deployment of a urban-digital twin along the cloud-to-thing continuum, where citizens are regarded as first-class entities. Here, we present a simplified version of the domain model, as displayed in Figure 3 using our tool. According to the figure, we can see that a *Passenger* can take any number of *Journey*s that always start and end at a specific time and from/at a specific *Location*. A journey likely involves getting on and off one or several buses at specific *Stop*s. When a passenger arrives at a stop, we store (*BeginInfo* class) the arrival time at the stop and the boarding time, so that we know how long they have been waiting at the stop. The model also registers the time someone gets off the bus (*EndInfo* class). In turn, *Bus*es follow specific *Route*s, where each route is composed of a sequence of stops. Regardless of the route a bus is following, it is always stored the arrival time at and departure time from stops (*StopInformation* class).

Once this model is created in our tool, which is shown in the domain model of Figure 3, we are able to generate the corresponding Alloy representation. Bellow we present an excerpt of it.

```
sig Journey {
    end: set EndInfo,
    begin: set BeginInfo, [...]
}
sig BeginInfo {
     beginOf: one Journey,
     beginInfoOf: one Stop
}
sig EndInfo {
     endOf: one Journey,
     endInfoOf: one Stop
}
sig StopInformation { [...] }
sig Stop {
  info: set StopInformation,
  beginInfo: set BeginInfo,
  endInfo: set EndInfo, [...]
}
fact journeyEndSymmetry{
  all a:Journey, b:EndInfo | b.endOf=a <=> b in
      a.end
}
fact journeyBeginSymmetry{
  all a:Journey, b:BeingInfo | b.beginOf=a <=> b in
      a.begin
}
[...]
```

Given this Alloy model, we are able to verify properties such as:

```
// Journey origins and ends have to be different
assert acyclic_journeys {all j:Journey | disj[j.
    begin. beginInfoOf, j.end. endInfoOf] }

//A bus only visits a stop once in a route
assert unique_stops { all b:Bus | all s, s': b.info
    | s.stop.route = s'.stop.route => s!=s' }
```

IV. Related Work

A. Web-based Modeling Tools

Several web-based modeling tools have been proposed. Picto web [10] enables complex model exploration by means of different views in a website. Epsilon playground [20] is a tool that runs in the browser and supports model verification. However, these two tools are tailored to the Epsilon family of languages. Our tool, however, is extensible and could support any modeling and verification language.

Umple [4], [13] is a tool composed by a compiler and a web-based front-end[5]. The front-end allows the user to edit Umple code either textually, or rendered as diagrams, and enables interaction with the compiler running as a server. Webgme [14] is a web tool that supports the design of Domain Specific Modeling Languages (DSML) and the creation of corresponding domain model. AToMPM [19] is an open-source framework for Multi-Paradigm Modeling that allows designing DSML environments and performing operations on them. While all these tools are similar to ours in some extent, the common difference between them and our approach is that our proposal is light-weigh and dependency-free as everything runs in the browser. More importantly, our approach is extensible to new analysis/verification models and languages.

Gentleman [12] is a web-based projectional editor generator that allows the user to define a model and projections for its concepts, and use the generated editor to create the model instances. Our tool does not consider a projectional approach.

Siriusweb[6] is a low-code platform to define web applications using visual languages. The goal of our tool is not to create applications but to model domains and perform verification tasks on them.

There are also web tools such as PlantUML[7] for the creation

[4]https://mozilla.github.io/nunjucks/

[5]https://cruise.umple.org/umpleonline/

[6]https://github.com/eclipse-sirius/sirius-web

[7]https://plantuml.com/

of diagrams with support for the UML notation. However, we do not consider those as part of this related work and we focus only on tools that, like our approach, allow the creation and verification of models.

B. Integration of Domain Specific Languages and Formal Methods

Since the appearance of early theoretical approaches that aimed at bridging the gap between CASE and formal verification tools, such as VeriAgent [16], there have been multiple efforts targeting tool support in this area [8]. Their number is too broad to be discussed in this paper, so we focus on a selected subset closely related to our approach.

UML-VT [7] is a graphical specification environment and translation tool that supports formal verification of UML activity diagrams using the model checkers UPPAAL, SPIN, NuSMV and PES. The tool is implemented as an Eclipse-plugin that automatically translates the UML activities and logical requirements into valid input notation for the model checkers. Silva et al. [18] propose a methodology for creating formal DSL tools using the Epsilon platform. While these approaches provide integration between graphical environments and formal verification, they require local deployments with complex dependencies and are not designed for extensibility to multiple formalisms and verification capabilities.

Other efforts have mapped UML to various formal methods languages, such as CSP [1], [5], Z [2], and Alloy [3]. Moreover, tools like HaiQ [6] combine different verification capabilities such as structural and behavioral (probabilistic) by projecting parts of its own specification language into Alloy and probabilistic model checkers like PRISM [11]. These approaches provide tool support that is mostly focused on the translation between domain models and formalisms amenable to automated analysis, but neglect the interactive specification and verification of models.

V. Conclusions and Future work

In this paper, we have presented a generic, extensible and light-weight architecture and proof-of-concept-tool for the verification of domain models expressed as class diagrams using the Alloy specification language.

There are different lines of work that we would like to explore in the future. First of all, we would like to extend our transformations from domain models to Alloy to provide support for class attributes and unidirectional associations. Second, we acknowledge that running tools in the browser has advantages but also limitations. We would like to be able to reuse existing modeling technology and popular notations such as EMF, hence we plan to extend our architecture in the future to consider a server integration where these technologies could run. We would also like to extend our approach and tool to support the verification of, not only structural properties, but behavioural properties. For this, we would like to introduce behavioural diagrams such as sequence diagrams that will be mapped to languages such as PRISM [11].

Acknowledments. This work was partially funded by Universidad de Málaga (Campus Internacional de Excelencia), and the Spanish Government under projects PID2021-125527NB-I00 and TED2021-130523B-I00.

References

[1] Islam Abdelhalim, Steve Schneider, and Helen Treharne. Towards a practical approach to check uml/fuml models consistency using csp. In *Proc. of ICFEM'11*, pages 33–48. Springer, 2011.

[2] Nuno Amálio, Susan Stepney, and Fiona Polack. Formal proof from uml models. In *Proc. of ICFEM'04*, pages 418–433, 2004.

[3] Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, and Indrakshi Ray. UML2Alloy: A challenging model transformation. In *Proc. of MODELS'07*, pages 436–450, 2007.

[4] Omar Badreddin. Umple: A model-oriented programming language. In *Proc. of ICSE'10*, page 337–338, New York, NY, USA, 2010.

[5] Jordi Cabot, Robert Clarisó, and Daniel Riera. Umltocsp: A tool for the formal verification of uml/ocl models using constraint programming. In *Proc. of ASE'07*, page 547–548, 2007.

[6] Javier Cámara. Haiq: Synthesis of software design spaces with structural and probabilistic guarantees. In Kyungmin Bae, Domenico Bianculli, Stefania Gnesi, and Nico Plat, editors, *Proc. of FormaliSE@ICSE'20*, pages 22–33. ACM, 2020.

[7] Zamira Daw, John Mangino, and Rance Cleaveland. UML-VT: A formal verification environment for uml activity diagrams. In *Proc. of P&D@MODELS'15*, pages 48–51, 2015.

[8] Carlos A González and Jordi Cabot. Formal verification of static software models in mde: A systematic review. *Information and Software Technology*, 56(8):821–838, 2014.

[9] Daniel Jackson. Alloy: A language and tool for exploring software designs. *Commun. ACM*, 62(9):66–76, aug 2019.

[10] Dimitris Kolovos and Antonio Garcia-Dominguez. The epsilon playground. In *Proc. of MODELS'22 (Companion Proceedings)*, page 131–137, 2022.

[11] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Proc. of CAV'11*, volume 6806, pages 585–591, 2011.

[12] Louis-Edouard Lafontant and Eugene Syriani. Gentleman: a light-weight web-based projectional editor generator. In Esther Guerra and Ludovico Iovino, editors, *Proc. of MODELS'20 (Companion Proceedings)*, pages 1:1–1:5. ACM, 2020.

[13] Timothy C. Lethbridge, Andrew Forward, Omar Badreddin, Dusan Brestovansky, Miguel A. Garzón, Hamoud Aljamaan, Sultan Eid, Ahmed Husseini Orabi, Mahmoud Husseini Orabi, Vahdat Abdelzad, Opeyemi Adesina, Aliaa Alghamdi, Abdulaziz Algablan, and Amid Zakariapour. Umple: Model-driven development for open source and education. *Sci. Comput. Program.*, 208:102665, 2021.

[14] Miklós Maróti, Tamás Kecskés, Róbert Kereskényi, Brian Broll, Péter Völgyesi, László Jurácz, Tihamer Levendovszky, and Ákos Lédeczi. Next generation (meta)modeling: Web- and cloud-based collaborative tool infrastructure. In *Proc. of MPM@MODELS'14*, volume 1237, pages 41–60, 2014.

[15] Nathalie Moreno, Lorenzo Toro-Gálvez, Javier Troya, and Carlos Canal. Modeling urban digital twins over the cloud-to-thing continuum. In *Proc. of MeSS@STAF'23*, 2023.

[16] E. Mota, E. Clarke, A. Groce, W. Oliveira, M. Falcão, and J. Kanda. Veriagent: an approach to integrating uml and formal verification tools. *Electronic Notes in Theoretical Computer Science*, 95:111–129, 2004.

[17] OMG. UML profile for MARTE: Modeling and analysis of real-time embedded systems, 2009.

[18] Robson Silva, Alexandre Mota, and Rodrigo Rizzi Starr. Creating gui-based dsl formal tools. In *Proc. of IRI'13*, pages 520–527, 2013.

[19] Eugene Syriani, Hans Vangheluwe, Raphael Mannadiar, Conner Hansen, Simon Van Mierlo, and Hüseyin Ergin. Atompm: A web-based modeling environment. In *Joint Proceedings of MODELS'13*, volume 1115, pages 21–25, 2013.

[20] Alfa Yohannis, Dimitris S. Kolovos, Antonio García-Domínguez, and Carlos Javier Fernández Candel. Picto web: a tool for complex model exploration. In *Proc. of MODELS'22 (Companion Proceedings)*, pages 56–60, 2022.