IML: Towards an Instructional Modeling Language

Eric J. Rapos and Matthew Stephan
Department of Computer Science & Software Engineering,
Miami University, Oxford, OH, USA
{*rapose, stephamd*}*@miamioh.edu*

Keywords: Modeling Languages, Model-Driven Engineering, Model Education, Domain-Specific Modeling Language, Meta-Model, Model Transformation, Model-Based Testing

Abstract: Existing software modeling languages and tooling often contain features far beyond the comprehension of novice users. Additionally, tools focus on specific aspects of the model-driven software engineering (MDSE) process with little to no connection between various tools, phases, and applications. This paper describes our current work and project plans to develop a single modeling language that is aimed at teaching MDSE at the university level. The Instructional Modeling Language (IML), which implements both graphical and textual modeling paradigms, centres around a single tool that includes functionality for various modeling techniques and is not bloated with the full functionality that has been demonstrated to cause students to become overwhelmed when introduced to numerous tools at once. IML includes and describes crucial MDSE concepts and learning milestones including functionality for meta-modeling, instance creation, model transformations, real-time system modeling, code generation, model-based testing, and others. Ultimately, our goal is for IML to be used by instructors to introduce MDSE into their curriculum in a lightweight, easy-to-instruct manner. This includes industrial education to introduce employees with little or no modeling experience to MDSE concepts and applications. In this paper, we describe our plans for developing IML through four phases, our current progress including IML's scope and meta-model, our prototype, and future plans and anticipated challenges. Our hope is to continue engaging the MDSE community at the conference for feedback, suggestions, and for volunteers for case study and trial adoption.

1 INTRODUCTION

Model-Driven Software Engineering (MDSE) has been gaining traction in software development and engineering, particularly in embedded and real-time systems (Hutchinson et al., 2011). The uptake in usage in industry has led to a shortage of trained graduates with appropriate knowledge and skills in MDSE (Hutchinson et al., 2014). While many institutions have introduced courses in modeling (shown through increasing participation in the Educator's Symposium of the MoDELS conference), the tool support and languages that currently exist are cumbersome in scope and not explicitly designed for teaching. While many languages exist for teaching programming (Mannila and de Raadt, 2006), the same cannot be said of modeling languages. There is significant positive impact when using the model-driven approach in software engineering education (Hamou-Lhadj et al., 2009), however it is important that it be introduced as effectively as possible.

This position paper presents our work in progress towards the development of a modeling language intended to help teach modeling to university students. We plan for the Instructional Modeling Language (IML) to be a light-weight language that introduces multiple key concepts of MDSE, such as meta-modeling, model transformation, simulation, and code generation. It will do so in a single tool environment that neither has a cumbersome learning curve nor does it feature overwhelmingly complex structures and overhead. IML is, to the best our knowledge, the first modeling language targeted specifically at teaching MDSE skills and features. Thus, IML may be considered a domain specific modeling language (DMSL), where the domain is MDSE education.

To fully realize IML's implementation, we will develop a set of tools will to showcase its features. Thus, we use the term IML to refer to both the modeling language and the tool framework we propose to implement it. We based IML upon UML class diagrams for its core functionality, and implementing several of its key features while removing some of the more cumbersome aspects of UML in favour of teaching concepts simply. This differs from lightweight languages, such as lightweight UML, because it facilitates the complete MDSE process and is intended primarily for education.

We begin by presenting out motivations and planned contributions, followed by relevant work in Section 2. Section 4 presents the overall details of the proposed IML, detailing the 4 phases of development. Section 5 presents our current progress to date and Section 6 presents the future work. Finally in Section 7 we conclude with a summary of the proposed and completed work.

1.1 Motivation

The main motivation for our work comes from our previous experience teaching a course in MDSE to students with minimal to no exposure to any of the main modeling concepts. Their difficulties stemmed from the unavailability of a single tool capable of demonstrating the many aspects of MDSE, and the technical challenges associated with students installing and maintaining several different tools. This ranged from open source, to academic, and proprietary systems. The experience is shared by others in the community struggling with similar issues (Rapos, 2018). This is similar to the course offered by Poruban et al. whereby they had to use "several different practical tools" instead of a single tool in order to teach MDSE to their graduate students (Poruban et al., 2014).

The driving force behind the IML is rooted in two main areas: i) the need for a single tool capable of demonstrating the full spectrum of MDSE topics, and ii) the desire for a light-weight tool that allows users to learn concepts easier than full heavy-weight tools, and transfer them to those full tools after mastering the basics.

1.2 Contributions

In order for the IML to become successful, we plan on making four contributions which we describe in detail in this paper.

- a light-weight, yet fully functional, modeling language geared towards teaching MDSE;
- a set of example models and material to accompany the language;
- a facility for customization of the IML DSML for individual use/instruction; and
- a demonstration of the effectiveness of the DSML through case studies, examples and existing evaluations of teaching languages (Mannila and de Raadt, 2006).

The goal of our IML project is to develop a fully modularized and simple, yet comprehensive, DSML specifically tailored to teaching MDSE to college/university level students with some prior programming and system design experience. Beyond being a language to teach MDSE, a secondary goal for our project is to have the IML robust enough for use in very small scale modeling projects. That is, it will be a fully functional language that is useful for rudimentary real-world systems. Ideally it would find a place in the world of Agile modeling, where it could be used as a means of lightweight rapid prototyping systems to produce early results. Further, we intend for the models created and managed by the IML to be transferable to existing, more robust tools, in order for the lightweight rapid prototypes to be portable to full-scale development, if desired.

2 BACKGROUND AND RELATED WORK

In this section, we briefly overview DSMLs as the IML is a DSML intended for teaching. We also summarize existing work on teaching-specific modeling languages to both position our work and to utilize and learn from their experiences.

2.1 Domain-Specific Modeling Languages

DSMLs are a tried and tested way of facilitating MDSE by allowing engineers to create and manage modeling artifacts using domain-specific abstractions, concepts, and terms with which they are familiar. The DSML meta-models and their environments and tools are designed in such a way that facilitate full code generation (Amyot et al., 2006; Kelly and Tolvanen, 2008). They have been shown to be very effective in practice (Kärnä et al., 2009), and exist for many different domains including adaptive systems (Fleurey and Solberg, 2009), business models (Sonnenberg et al., 2011), computer games (Furtado and Santos, 2006), multiagent systems (Hahn, 2008), and others. In our case, we are developing a DSML for MDSE education.

2.2 Education-Specific Languages

In a comparison of languages for teaching introductory programming, Mannila and Raadt summarize seventeen criteria for evaluating introductory programming languages based on a survey of the field and languages considered 'teaching languages' (Mannila and de Raadt, 2006). Some examples include a language that is suitable for teaching, has the ability to apply physical analogies, features a general framework, is interactive and features rapid code development and more. They organize their criteria into four categories: learning, design and environment, support and availability, and its ability to go beyond introductory programming. We plan to use these seventeen criteria and four categories to guide not only the design of the IML but also our planned evaluations and case studies.

Silva-Maceda et al. performed an experiment demonstrated that allocating more time for students to absorb concepts, rather than better programming languages (C versus Raptor), was more of a factor in

student success in learning traditional code programming (Silva-Maceda et al., 2016). With the IML, our goal is to allow for students to spend more time focusing on the concepts rather than the individual tools and their intricacies.

Van Roy et al. demonstrated that the specific programming paradigm made no difference in student learning, but that the language they used was the key factor (Van Roy et al., 2003). This supports our mission in creating the IML, a language focused primarily on modeling education.

3 IML OVERVIEW

In this section we present an inital overview of the Instructional Modeling Language, specifically its intended features and our initial meta-model.

3.1 Feature Selection

Based on our personal academic and industrial experiences in modeling, and our research on MDSE education including the recommendations provided in the ongoing work towards an MDSE body of knowledge (Ciccozzi et al., 2018), we chose the following modeling concepts for inclusion in the IML,

1. conceptual system modeling
2. meta-modeling and instance creation
3. model-to-model and model-to-text model transformations
4. behavioral modeling
5. model execution and simulation
6. code generation
7. model-based testing and model validation

This list is by no means an exhaustive and complete list of MDSE topics. However, we contend it forms the basis of a core understanding and is the minimal set of concepts for us to consider IML to be completed. Additional techniques and concepts can, and likely will, be added in successive releases based on practitioner and academic feedback.

3.2 IML Meta-Model

In order for us to demonstrate the light-weight nature of the IML and the limited subset of UML model features it will implement, it was imperative that we first develop a meta-model for the language. Figure 1 presents the IML meta-model, which we developed as an Ecore model within the EMF. When compared to the Ecore meta-model, the meta-model for IML contains a constrained set of data types, and is simplified greatly in how relations are expressed

IML describes a simplified version of UML class diagrams, containing classes with a name and attributes, and a limited set of relations between classes. While the feature set is limited, it still allows users to create conceptual models with a variety of data types and relations, containing enough complexity to hone and grasp concepts related to meta-modeling, model creation, and the general abstraction elements associated with MDSE.

As the project progresses, it is possible that the meta-model will evolve to adapt to necessity, but our goal is to maintain its light-weight simplicity throughout development.

4 IMPLEMENTING IML

In order to accomplish the goals of IML, we have outlined a 4 phase project, which we describe in the following subsections.

4.1 Phase 1: Conceptualization

The first phase of the project is one of the more important steps in creating an effective framework for teaching MDSE. Our original intent was to conduct a review of courses being taught by colleagues and other institutions to examine the overlapping and key topics. However a recent initiative lead by many well established MDSE researchers and industry representatives works toward creating a *Body of Knowledge* specifically for MDSE (Ciccozzi et al., 2018). This work in progress will provide an already cultivated list of topics that will make an excellent feature set for the IML. We present our results in analyzing this work and how we will incorporate it into the IML in Section 5.

4.2 Phase 2: Implementation

The bulk of this project will focus on the implementation of the IML DSML and its supporting tools. Based on the functionality we determine in Phase 1, we will order concepts in sequence based on when they are used by modelers in the typical MSDE process. For example, conceptual system design should be early in the process, while model-based testing and code generation will ultimately be later in the list. The reason for this ordering is that our goal is to implement the set of IML tools in an iterative fashion, producing a working tool over many iterations, and adding new functionality during each iteration.

4.2.1 Implementation Alternatives

There are currently two possible options we are considering for the development of the IML: an EMF/GMF based plugin to emulate similar tools, and a

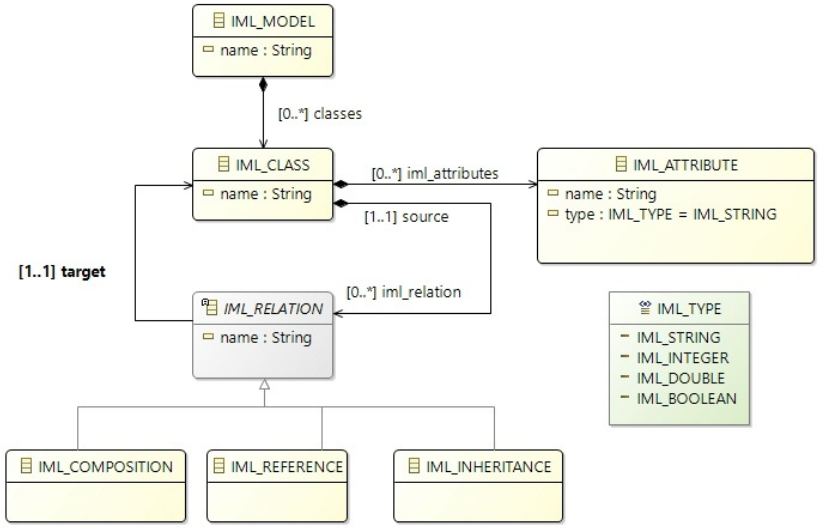

Figure 1: The IML Meta-Model implemented as an Ecore Model

standalone Java based tool implemented in JavaFX. Both of these options have pros and cons, which we discuss below. It is our hope that the modelsward community will provide further feedback and insights into these options.

Eclipse Based Implementation Option Eclipse is a well-known, heavily supported, platform that is used in numerous tools developed in both academia and industry. In addition to its prevalence, Eclipse is also an open source option meaning that there are no additional costs to the development, and students and users of the resulting DSML will not require software licenses in order to use the tools. This is not the case for many of the current modeling tools. By creating an Eclipse based tool, we can leverage existing open source plugins to Eclipse that will help implement parts of what we hope to accomplish. As an example, the Eclipse Modeling Framework (EMF) is an eclipse plugin itself, and could be leveraged easily for us to incorporate in the DSML we develop. However, it would be a pared down version to remove any high-complexity features that may bog down novice users, leaving only the required and key educational features.

Using this existing framework may have the pitfall of its associated complexities and bloat may make its way into the IML, which directly contradicts the goals and our vision of a light-weight framework designed for education. To make use of EMF we would have to be very careful to avoid using large numbers of data types, obfuscate complex aspects, and provide detailed support and documentation.

JavaFX Based Implementation Option JavaFX is the successor to Java Swing, and is used to create graphical interfaces to Java based programs. By building a framework from the ground up with JavaFX we are capable of avoiding any the complexity issues that could arise from the use of EMF. This approach allows for full control over the elements of the modeling language that the users will be interacting with. Another added benefit of a standalone Java based application is the fact that installation will not be an issue. This is in contrast to the Eclipse option since Eclipse plugins can often cause issues for those unfamiliar with the framework. Additionally, most potential users will already have a JRE installed on their machine[1], leaving practically no installation requirements.

The downside of pursuing this implementation option is that there is significantly less community and framework support for JavaFX than there is for EMF and GMF based solutions.

4.2.2 Model Representation Format

Another major milestone in the development of our DSML is determining the appropriate format for encoding IML models textually. In keeping with the open source mentality, as well as being able to reuse as much as possible, we have decided to represent the models for this DSML using an XML representation. This type of representation is widely accepted and used in many current modeling tools, and allows us to use existing naming conventions, representations and more to ensure smooth transitions from our modeling language to full-fledged modeling tools.

[1] https://web.archive.org/web/20100925204716/https://java.com/en/download/faq/whatis_java.xml

4.3 Phase 3: Validation, Verification, and Quality Assurance

As each feature is implemented in the previous phase, they will be tested independently for correctness, completeness, and quality. This interleaving of phases 2 and 3 will allow concurrent work to be done by all those involved in the project and will also ensure an iterative development of the modeling language. This will allow for individual aspects of the IML to be disseminated earlier on, rather than waiting for the entire framework. This will allow for additional community feedback and cyclical improvements throughout development.

Beyond modularity, this iterative approach will allow for dedicated testing of smaller units of implementation, which is a key factor in producing quality software tools (Ammann and Offutt, 2016). After we have implemented all phases and tested them individually, we will take part in several levels of integration testing to ensure the entire product suite works harmoniously as a complete end-to-end modeling tool (Ammann and Offutt, 2016). This will consist of several case studies, aimed at reproducing results from other mainstream modeling tools. When all aspects of the system work together, we will consider the system complete and we will proceed to the fourth and final phase.

4.4 Phase 4: Documentation and Support

Since one of the main purposes of the IML is its use in teaching, a large component of its success is dependent on proper documentation and support.

During the completion of the IML, we will focus efforts on providing supporting materials to accompany delivery of the tool, including documentation for installation, troubleshooting, the creation and packaging of several example models, and more. We will validate and verify this documentation during this phase by soliciting feedback from students in our modeling course, asking for peer reviews from both modeling practitioners and educators, and other techniques. We will also produce demonstration videos for instructors to use in course offerings and/or novice modelers to use in self-learning modeling principles.

In addition to the educational support, this phase will also include us developing support for the tool by means of marketing through websites, published works, forums, and others. We will note its availability to instructors, primarily those involved in MDE education and those responsible for the push towards the MBE body of knowledge (Ciccozzi et al., 2018), and more broadly via the modeling community.

5 CURRENT PROGRESS

This section presents the current progress in our realization of the IML. It is rooted in three main areas: selection of IML's features, initial development of the IML meta-model, and early prototype development.

5.1 Current Prototype

In an attempt to begin working on the IML, an early prototype has been developed using JavaFX. This section discusses the various aspects that we have currently implemented.

5.1.1 Overall Tool Layout

The first necessary step was the creation of the tools overall layout for its user interface. Since one of our other implementation options was an Eclipse based approach, along with our desire for the IML to allow users to easily transition to existing tools in the modeling domain, we opted for a GUI that was visually similar to Eclipse and its derivative plugins. Thus, we opted for a tool with a canvas in the center capable of displaying models or a text editor; a file/tree browser to the left; a tool box on the right, which currently contains placeholders for tools to be added later;and a view for displaying properties and other details on the bottom. JavaFX uses a layout manager that made maintaining these layouts rather simple, and easy to maintain independently in a *plug and play* manner. We present the main IML window in Figure 2.

5.1.2 Model Canvas

One of the main features of our early prototype is the model drawing canvas. As a modeling tool, it is imperative to be able to implement the creation of IML models accurately. Our current prototype allows the creation of various IML-named classes on the canvas, which can then have various typed and named attributes added to them. Each of the classes are capable of having various relations between them, for example, composition, reference, and inheritance, using the standard accepted notations.

Figure 2 shows an example model we have drawn on the canvas. It illustrates a university made up of departments, each of which reference certain faculty members that inherit features from the general Faculty class. Our current implementation does not present the cleanest lines, but its main purpose is for proof-of-concept of function, and should development continue using JavaFX, we will give further attention to UI aspects.

5.1.3 Model Format - Read and Write

The last major aspect of our current work relates to the model formatting. In addition to being able to

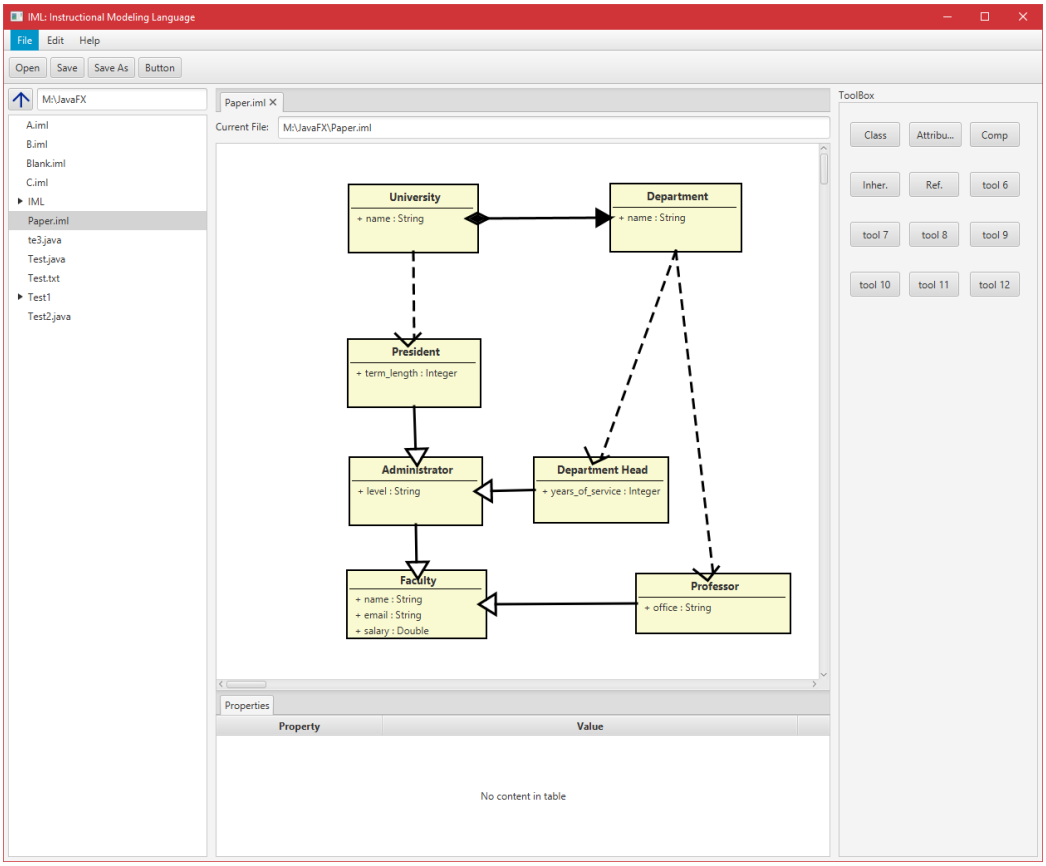

Figure 2: An example IML Model in the tool's model drawing canvas

draw models on the canvas, the IML is capable of saving the models to an XML based format, as well as reading in an XML input file and rendering the corresponding model on screen for further editing. We present an example XML representation for the model shown in Figure 2 in the code listing in Figure 3.

Each class is its own object, with attributes for its name, and position. Position includes four numerical values: X position, Y position, height, width, as shown on line 3. Every class then has each of its attributes embedded. Each attribute is stored with a name and type, for example, line 4. After each of the classes are defined in the XML file either by the IML tool or an external tool, the connections between them are interpreted by the IML tool as individual objects, containing a source, destination, and type such as, inheritance, reference, or composition, as we illustrate on line 26.

6 FUTURE WORK

Our immediate next step is to determine whether to continue development with the JavaFX method, or to switch to the Eclipse based toolset of EMF, GMF, et cetera. While the standalone nature of our current tool provides more flexibility in what is ultimately developed, the steep learning curve and lack of community support for JavaFX may make it a less viable option. However, a tool that is significantly more lightweight than existing modeling tools will help set the IML apart from cumbersome tools, which is one of the IML's main goals.

Regardless of which option we choose, the remaining future work involves us extending our current progress to implement a fully functional first iteration of IML. Recall, our intent is to iteratively add functionality from the list of features. Our first priority is conceptual system modeling, which is the ability to draw and manipulate models similar to existing UML tools; a feature nearly completely implemented in the current prototype. The remaining features missing from the first iteration are the ability to manipulate properties via the properties tab, and model checking for conformance to the a specified meta-model. Specifically, the IML meta-model in the first iteration, and custom meta-models in later iterations.

```
<?iml version="0.1" encdoing="UTF-8"?>
<iml:Model name="Paper.iml">
    <iml:Class name="University" position="160.0 50.0 80.0 154.0">
        <iml:Attribute identifier="name" Type="String">
    </iml:Class>
    <iml:Class name="Department" position="475.0 49.0 80.0 156.0">
        <iml:Attribute identifier="name" Type="String">
    </iml:Class>
    <iml:Class name="Professor" position="506.0 518.0 70.0 184.0">
        <iml:Attribute identifier="office" Type="String">
    </iml:Class>
    <iml:Class name="Faculty" position="158.0 514.0 80.0 166.0">
        <iml:Attribute identifier="name" Type="String">
        <iml:Attribute identifier="email" Type="String">
        <iml:Attribute identifier="salary" Type="Double">
    </iml:Class>
    <iml:Class name="Administrator" position="161.0 378.0 80.0 158.0">
        <iml:Attribute identifier="level" Type="String">
    </iml:Class>
    <iml:Class name="President" position="159.0 236.0 80.0 158.0">
        <iml:Attribute identifier="term_length" Type="Integer">
    </iml:Class>
    <iml:Class name="Department Head" position="383.0 378.0 77.0 160.0">
        <iml:Attribute identifier="years_of_service" Type="Integer">
    </iml:Class>
    <iml:Connection source="Administrator" dest="Faculty" type="inheritence">
    <iml:Connection source="Professor" dest="Faculty" type="inheritence">
    <iml:Connection source="Department Head" dest="Administrator" type="inheritence">
    <iml:Connection source="President" dest="Administrator" type="inheritence">
    <iml:Connection source="University" dest="Department" type="composition">
    <iml:Connection source="University" dest="President" type="reference">
    <iml:Connection source="Department" dest="Department Head" type="reference">
    <iml:Connection source="Department" dest="Professor" type="reference">
</iml:Model>
```

Figure 3: Sample XML representation of IML models

Following the completion of the the first iteration, the next logical step is the implementation of model transformations within IML. This involves the ability to provide two custom meta-models and a set of transformation rules to be able apply automatically the transformation from any instance models that conform to the source meta-model, thus creating corresponding instances that conform to the source meta-model.

Our minimum viable product (MVP) for the IML and tool suite entails the implementation of conceptual modeling including conformance checking to a specified meta-model, the ability to create custom meta-models and instance models, and model transformations. This MVP milestone will be useful in teaching several of the important high-level concepts in MDSE. Beyond this MVP, the remaining features will be implemented one by one until we have introduced the full feature set. We now describe each feature in detail.

To incorporate behavioral modeling into IML, the goal is to provide two alternative methods that emulate leading tools. The first method will be the use of state machines to implement behavior, much like existing UML-RT tool implementations, such as Papyrus-RT[2] which is the current focus of ongoing research (Hili et al., 2017; Kahani et al., 2017). Keeping with the simplified nature of the IML, this will be a significantly reduced subset of the UML-RT profile to demonstrate functionality of existing tools, such as Papyrus-RT, without the need to cover some of UML-RT's more complex implementation issues. The second method is to implement block based data-driven behavior in the style of Simulink models, again using a significantly reduced block library to demonstrate functionality. For each of these behavioral modeling methods, IML will include some mechanism of execution and/or simulation. The intent is to be able to walk though execution of the models, and provide simulated inputs to observe the system's responses.

For code generation, our intent will be to generate functional code, in Java for example, for the behavioral models that can be run independently from the IML framework. The focus of this aspect of IML is to demonstrate the power of MDSE to produce source code from graphical models.

The final area of work for us in development of the IML is related to model-based testing and model-checking techniques. We have yet to determine the specifics, but we will leverage existing testing technologies, such as test case generation through symbolic execution for the UML-RT based models (Zurowska and Dingel, 2012; Rapos and Dingel, 2012), and some form of Simulink model style of test-

[2] https://www.eclipse.org/papyrus-rt/

ing capable of generating full test suites that apply to both simulation and code generation, while considering the evolution of systems and their tests (Matinnejad et al., 2016).

7 CONCLUSIONS

The goal of the IML and its framework is to provide computer science and software engineering educators adequate tool support to effectively incorporate MDSE into their curriculum, either as modules in existing courses or, ideally, as full courses on the topic. We envision IML as a modeling language that is capable of producing small-scale and simpler working software systems, but one that is not burdened by the cumbersome nature of many existing languages and tools available currently. By stripping away many low-level details and leaving only what is necessary for a student to master modeling techniques such as meta-modeling, model transformations, model-based testing, and others, IML aims to fill a sufficient void in educational focused software.

REFERENCES

Ammann, P. and Offutt, J. (2016). *Introduction to software testing*. Cambridge University Press.

Amyot, D., Farah, H., and Roy, J.-F. (2006). Evaluation of development tools for domain-specific modeling languages. In *International Workshop on System Analysis and Modeling*, pages 183–197. Springer.

Ciccozzi, F., Famelis, M., Kappel, G., Lambers, L., Mosser, S., Paige, R. F., Pierantonio, A., Rensink, A., Salay, R., Taentzer, G., Vallecillo, A., and Wimmer, M. (2018). Towards a body of knowledge for model-based software engineering. In *MODELS*, pages 82–89, New York, NY, USA. ACM.

Fleurey, F. and Solberg, A. (2009). A domain specific modeling language supporting specification, simulation and execution of dynamic adaptive systems. In *MODELS*, pages 606–621. Springer.

Furtado, A. W. and Santos, A. L. (2006). Using domain-specific modeling towards computer games development industrialization. In *OOPSLA workshop on domain-specific modeling (DSM06)*.

Hahn, C. (2008). A domain specific modeling language for multiagent systems. In *Joint conference on Autonomous agents and multiagent systems*, pages 233–240. International Foundation for Autonomous Agents and Multiagent Systems.

Hamou-Lhadj, A., Gherbi, A., and Nandigam, J. (2009). The impact of the model-driven approach to software engineering on software engineering education. In *International Conference on Information Technology: New Generations*, pages 719–724.

Hili, N., Dingel, J., and Beaulieu, A. (2017). Modelling and code generation for real-time embedded systems with uml-rt and papyrus-rt. In *International Conference on Software Engineering Companion*, pages 509–510.

Hutchinson, J., Rouncefield, M., and Whittle, J. (2011). Model-driven engineering practices in industry. In *International Conference on Software Engineering*, pages 633–642. ACM.

Hutchinson, J., Whittle, J., and Rouncefield, M. (2014). Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure. *SCP*, 89:144–161.

Kahani, N., Hili, N., Cordy, J. R., and Dingel, J. (2017). Evaluation of uml-rt and papyrus-rt for modelling self-adaptive systems. In *International Workshop on Modelling in Software Engineering*, pages 12–18.

Kärnä, J., Tolvanen, J.-P., and Kelly, S. (2009). Evaluating the use of domain-specific modeling in practice. In *OOPSLA workshop on Domain-Specific Modeling*.

Kelly, S. and Tolvanen, J.-P. (2008). *Domain-specific modeling: enabling full code generation*. John Wiley & Sons.

Mannila, L. and de Raadt, M. (2006). An objective comparison of languages for teaching introductory programming. In *Baltic Sea conference on Computing education research*, pages 32–37. ACM.

Matinnejad, R., Nejati, S., Briand, L. C., and Bruckmann, T. (2016). Automated test suite generation for time-continuous simulink models. In *Proceedings of the 38th International Conference on Software Engineering*, pages 595–606, New York, NY, USA. ACM.

Poruban, J., Bacikova, M., Chodarev, S., and Nosal, M. (2014). Pragmatic model-driven software development from the viewpoint of a programmer: Teaching experience. In *FedCSIS*, pages 1647–1656. IEEE.

Rapos, E. J. (2018). We'll make modelers out of 'em yet: Introducing modeling into a curriculum. In *EduSymp 18, Educators Symposium at MODELS*. ACM/IEEE.

Rapos, E. J. and Dingel, J. (2012). Incremental test case generation for uml-rt models using symbolic execution. In *International Conference on Software Testing, Verification and Validation*, pages 962–963.

Silva-Maceda, G., Arjona-Villicana, P. D., and Castillo-Barrera, F. E. (2016). More time or better tools? a large-scale retrospective comparison of pedagogical approaches to teach programming. *IEEE Transactions on Education*, 59(4):274–281.

Sonnenberg, C., Huemer, C., Hofreiter, B., Mayrhofer, D., and Braccini, A. (2011). The rea-dsl: A domain specific modeling language for business models. In *International Conference on Advanced Information Systems Engineering*, pages 252–266. Springer.

Van Roy, P., Armstrong, J., Flatt, M., and Magnusson, B. (2003). The role of language paradigms in teaching programming. In *ACM SIGCSE Bulletin*, volume 35, pages 269–270. ACM.

Zurowska, K. and Dingel, J. (2012). Symbolic execution of uml-rt state machines. In *ACM Symposium on Applied Computing*, pages 1292–1299, New York, NY, USA. ACM.