# AutoModel: A Domain-specific Language for Automatic Modeling of Real-time Embedded Systems

Nafiseh Kahani
Queen's University
Kingston, Canada
kahani@cs.queensu.ca

## EXTENDED ABSTRACT

This paper introduces a new approach to the automation of real-time embedded systems modeling. Our approach is based on a new domain-specific language called AutoModel to specify the requirements of a system in terms of its components, goals and constraints. Our automated approach accepts the specified requirements and infers both structural and behavioral models to implement the requirements in the UML-RT modelling language. Existing modeling tools can then be used to generate an implementation from the inferred models without extra work.

## CCS CONCEPTS

• **Software and its engineering → Domain specific languages**; **Model-driven software engineering**;

## KEYWORDS

Model-Driven Development (MDD), Model Synthesis, UML-RT

**ACM Reference Format:**
Nafiseh Kahani. 2018. AutoModel: A Domain-specific Language for Automatic Modeling of Real-time Embedded Systems. In *ICSE '18 Companion: 40th International Conference on Software Engineering Companion, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3183440.3190333

## 1 RESEARCH PROBLEM AND MOTIVATION

Model-driven development (MDD) is a growing field that advocates the use of models during the entire software development process. By leveraging abstraction and automation, MDD techniques can simplify communication and design activities, increase productivity and compatibility between systems, and boost development efficiency [10, 12]. MDD has been considered a promising software development approach to deal with the complexity and safety requirements of real-time embedded (RTE) systems. An impressive number of MDD-based tools and standards for RTE systems have been introduced (e.g., [15, 16]), some of which have been successfully used in industry (e.g., IBM Rhapsody [16]).

Despite the promised benefits, application of MDD in practice has not yet reached its full potential [18]. One of the important reasons is related to the complex and time-consuming process of designing and creating the models, which often requires special expertise and experience [8]. To deal with this problem, several approaches have been proposed to automate various parts of the modelling process by synthesizing behavioural models from requirements specifications, often expressed as either positive or negative scenarios of the systems' expected behaviours [1, 3, 4, 19, 20].

Existing studies provide a good starting point to tackle this problem. However, they suffer from several issues, mainly: (1) They require scenarios provided by a modeller, and outcomes are sensitive to the quality and quantity of the scenarios. Defining a complete set of scenarios to include a full description of the system requirements is almost impossible [4]. Even if it were possible, defining a full set of scenarios would be a challenge much like defining a complete state-machine. (2) They normally synthesize behavioural models in the form of state-machines, while paying less attention to structural models, which are as necessary as the behavioural ones. To use them, the modeller must still design structural models compatible with the inferred behavioural models, and manually integrate the two.

In our proposed approach, we aim to advance these existing solutions to overcome these issues. We introduce a domain-specific language (DSL) to specify the high-level requirements of a system in terms of its components and their actions, goals and safety constraints. Taking these specified requirements as inputs, we apply design-space exploration techniques [9] to generate the scenarios required for inferring the models, and use existing techniques to infer the system's behavioural models from these generated scenarios. Thus manual creation of scenarios is no longer required. Using model transformation techniques, we then automatically generate structural models of the system consistent with the generated behavioural models. The generated structural and behavioural models are compatible with each other and ready to use, for example, to generate executable code using existing MDD tools. To maximize the impact of our work, our implementation generates models in the UML-RT (UML for Real-Time) language [13, 17], and uses open source tools and frameworks including Papyrus-RT [11, 15] for modelling and code generation, Epsilon [7] for model transformation, Xtext [23] for DSL implementation, and the Eclipse Modeling Framework (EMF) [6] for design-space exploration.

## 2 AN ILLUSTRATIVE EXAMPLE

As an illustrative example, we adopt the simplified train controller of [5]. We assume that the train has a door, and an engine. Controller software for the train must guarantee that the train's door is always closed when the train is moving. To develop the controller using existing modelling tools, the modeller typically manually

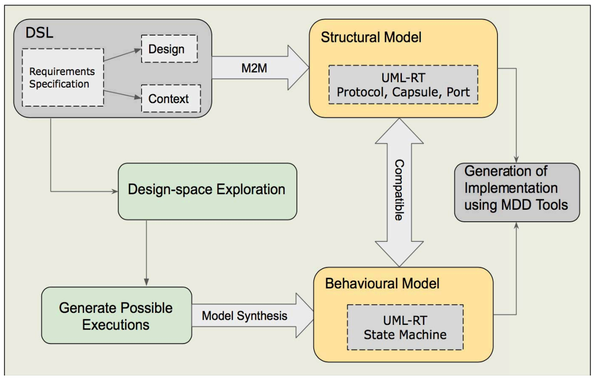

DSL
Requirements Specification
Design
Context
M2M
Structural Model
UML-RT Protocol, Capsule, Port
Compatible
Generation of Implementation using MDD Tools
Design-space Exploration
Generate Possible Executions
Model Synthesis
Behavioural Model
UML-RT State Machine

**Figure 1: High-level architecture of our proposed approach.**

defines the system's structural models, and their interface with each other. They then define a detailed behavioural model for each component, which is a time consuming and challenging task. In our work, users need only specify the high-level requirements. Our proposed solution automatically synthesizes structural and behavioural models of the system based on the requirements, from which an implementation can be automatically generated using existing MDD tools.

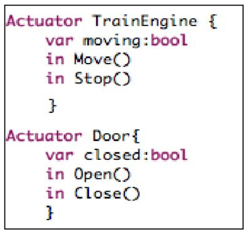
## 3 THE PROPOSED APPROACH

Figure 1 shows an overview of our approach for automation of modeling process that consists of three parts. Each part of the approach is briefly discussed in the following.

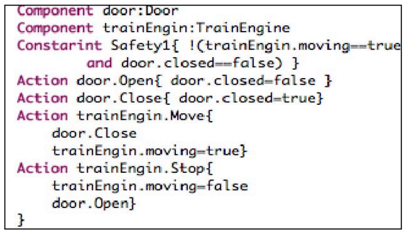
***Requirements Specification.*** Figure 1 shows how the requirements of the train system are specified in AutoModel. AutoModel provides a declarative textual language that allows users to specify the high-level requirements of a system by defining its components, their properties and their actions. An example is shown in Fig. 2a. After defining the components, a configuration of the system is defined to specify which components are included in the system, the system's goals and safety constraints, and how the component actions change the component properties (Fig. 2b).

***Synthesis of Structural Models.*** We analyze the system configuration and contained components, and use model transformation techniques to generate a structural model of the system in the UML-RT language. Compared to existing techniques, in which users are required to create each component and define communication protocols in detail, our approach requires only a simple and minimal specification.

***Synthesis of Behavioural Models.*** We have developed an interpreter for our DSL, which then simulates the execution of the defined configuration. Using the interpreter, we apply design-space exploration techniques to explore all possible executions and record the execution traces (i.e., execution scenarios). While exploring the design space, our exploration algorithm checks for constraints and stops an execution if any constraint is violated, marking the related traces as negative scenarios. After generating enough traces, we apply and existing behavioural model synthesis technique (e.g., [2]) to generate a state-machine corresponding to the traces.


```
Actuator TrainEngine {
    var moving:bool
    in Move()
    in Stop()
    }
Actuator Door{
    var closed:bool
    in Open()
    in Close()
    }
```

(a) System components


```
Component door:Door
Component trainEngin:TrainEngine
Constarint Safety1{ !(trainEngin.moving==true
        and door.closed==false) }
Action door.Open{ door.closed=false }
Action door.Close{ door.closed=true}
Action trainEngin.Move{
    door.Close
    trainEngin.moving=true}
Action trainEngin.Stop{
    trainEngin.moving=false
    door.Open}
}
```

(b) System configuration.

**Figure 2: Train system requirements in AutoModel.**

To address the scalability of the design-space exploration algorithm, we use a bounded breadth-first search (BFS) algorithm to bound the exploration of possible executions. The modeller can set the bound threshold to make sure that exploration will terminate. To improve algorithm performance, we use a compositional approach based on exploration and synthesis of each component model separately.

## 4 RELATED WORK

Synthesis of system models has long been an active research area [1, 3, 4, 19–21]. Existing work can be categorized into two groups:

(1) *Synthesis of state-machines based on requirements defined using message sequence charts (MSC) [4] or live sequence charts (LSD) [8].* Scenarios capture typical examples or counterexamples of system behaviour through sequences of interactions among agents. They support an informal, narrative, and concrete style of description. Scenarios are therefore easily accessible to end users involved in the requirements engineering process. The main drawback of these methods is that defining a full set of scenarios to include a full description of the system requirements is almost impossible [4].

(2) *Synthesis of behavioural or architectural models from system execution traces [1, 3].* These techniques are not applicable to designing the system, as they are based on existing system execution. Furthermore, their correctness depends on the quality of the logs (i.e., they require enough meaningful traces to be generated).

There is other work (e.g., [14, 22]) on model completion methods, to assist users in modeling activities by providing suggestions or applying possible operations to the defined models. However, model completion is limited to only the next few steps, and is not aimed at creating a complete model.

## 5 CONTRIBUTIONS AND CONCLUSIONS

Automating the modeling process can help to address issues related to the complex and time-consuming modeling process, and thus assist users in taking full advantage of MDD approaches. Our work aims to increase automation with the following contributions:

(1) We introduce a domain-specific language (DSL) to specify the high-level requirements of a system. (2) We advance existing solutions in theory and practice by leveraging design-space exploration and model transformation techniques. (3) We automatically synthesize both behavioural and structural models of the system from requirements. (4) By integrating with open source modeling tools (i.e., Papyrus-RT and EMF), and making AutoModel itself publicly

available, we enable researchers to extend and use our approach in their future research.

Thus far we have finished the design of our DSL, and completed the structural model synthesis, the AutoModel interpreter, and a framework for design-space exploration. Currently, we are working on the integration of our design-space exploration framework with existing behavioural model synthesis tools, to support the automatic generation of behavioural models.

## REFERENCES

[1] I. Beschastnikh, Y. Brun, M. D. Ernst, and A. Krishnamurthy. 2014. Inferring models of concurrent systems from logs of their behavior with CSight. In *Proceedings of the 36th International Conference on Software Engineering*. 468–479.

[2] A. W. Biermann and J. A. Feldman. 1972. On the synthesis of finite-state machines from samples of their behaviour. *IEEE Trans. Comput.* 100, 6 (1972), 592–597.

[3] I. Buzhinsky and V. Vyatkin. 2017. Automatic inference of finite-state plant models from traces and temporal properties. *IEEE Transactions on Industrial Informatics* 13, 4 (2017), 1521–1530.

[4] C. Damas, B. Lambeau, P. Dupont, and A. Van Lamsweerde. 2005. Generating annotated behavior models from end-user scenarios. *IEEE Transactions on Software Engineering* 31, 12 (2005), 1056–1073.

[5] Ch. Damas, B. Lambeau, and A. V. Lamsweerde. 2006. Scenarios, goals, and state machines: A win-win partnership for model synthesis. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 197–207.

[6] EMF. 2017. Eclipse Modeling Framework (EMF). (2017). https://www.eclipse.org/modeling/emf.

[7] Epsilon. 2017. Epsilon. (2017). https://www.eclipse.org/epsilon.

[8] D. Harel, H. Kugler, and A. Pnueli. 2005. Synthesis revisited: Generating statechart models from scenario-based requirements. In *Formal Methods in Software and Systems Modeling*. 309–324.

[9] Á. Hegedüs, Á. Horváth, and D. Varró. 2015. A model-driven framework for guided design space exploration. *Automated Software Engineering* 22, 3 (2015), 399–436.

[10] N. Kahani, M. Bagherzadeh, J.R. Cordy, J. Dingel, and D. Varró. 2018. Survey and classification of model transformation tools. *Software and Systems Modeling (SoSyM)* (2018).

[11] N. Kahani, M. Bagherzadeh, J. Dingel, and J.R. Cordy. 2016. The problems with Eclipse modeling tools: A topic analysis of Eclipse forums. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*. 227–237.

[12] N. Kahani and J.R. Cordy. 2015. Comparison and evaluation of model transformation tools. In *Technical Report 2015-627*. Queen's University, 1–42.

[13] N. Kahani, N. Hili, J.R. Cordy, and J. Dingel. 2017. Evaluation of UML-RT and Papyrus-RT for modelling self-adaptive systems. In *Proceedings of the 9th International Workshop on Modelling in Software Engineering*. 12–18.

[14] S. Mazanek, S. Maier, and Mark Minas. 2008. Auto-completion for diagram editors based on graph grammars. In *IEEE Symposium on Visual Languages and Human-Centric Computing*. 242–245.

[15] PapyrusRT. 2017. Papyrus for Real Time (Papyrus-RT). (2017). https://www.eclipse.org/papyrus-rt.

[16] Rhapsody. 2017. Rhapsody, IBM Rational. Systems-Engineering Tool Rhapsody. (2017). http://www-03.ibm.com/software/products/en/ratirhapfami.

[17] B. Selic. 1998. Using UML for modeling complex real-time systems. In *Languages, Compilers, and tools for Embedded Systems*. 250–260.

[18] B. Selic. 2012. What will it take? A view on adoption of model-based methods in practice. *Software and Systems Modelings* 11, 4 (2012), 513–526.

[19] S. Uchitel, G. Brunet, and M. Chechik. 2007. Behavior model synthesis from properties and scenarios. In *Proceedings of the 29th International conference on Software Engineering*. 34–43.

[20] S. Uchitel and J. Kramer. 2001. A workbench for synthesising behaviour models from scenarios. In *Proceedings of the 23rd International Conference on Software engineering*. 188–197.

[21] N. Walkinshaw, R. Taylor, and J Derrick. 2016. Inferring extended finite state machine models from software executions. *Empirical Software Engineering* 21, 3 (2016), 811–853.

[22] Y. Xiong, Z. Hu, H. Zhao, H. Song, M. Takeichi, and Hong Mei. 2009. Supporting automatic model inconsistency fixing. In *Proceedings of the the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*. 315–324.

[23] Xtext. 2017. Xtext. (2017). http://www.eclipse.org/Xtext.