Model-Driven Development of Simulation-Based System Design Tools

Sven Jäger, Ralph Maschotta, Tino Jungebloud, Alexander Wichmann, and Armin Zimmermann
Systems and Software Engineering Group
Computer Science and Automation Department
Technische Universität Ilmenau
Ilmenau, Germany
Contact: see http://www.tu-ilmenau.de/sse/

***Abstract*—Analysis and validation using simulation is a helpful tool in systems engineering, but requires in-depth knowledge of various aspects of the system itself, used model classes, and an appropriate software tool. Usually, this expertise is spread over a number of team members, thus making it a non-trivial task. A domain-specific simulation-based system design tool (termed here simulation-based application, SBA) could fill this gap. It hides the complexity of the model from the system designer and allows to configure parameters and analyze results in one single application. The necessary extra effort in software development compared to a bare modeling tool can be reduced with techniques for model-driven development.**

This paper presents an approach to use such methods to improve the development of SBA as a case of domain-specific software product lines. The workflow is described as well as the existing meta-model and applied techniques from model-driven development. The paper also shows the necessary elements of a meta model to describe SBAs. An example shows the complete workflow using an example of a wireless sensor networks for avionic applications.

***Index Terms*—Model-based design, simulation, model-driven development, meta-model**

I. Introduction

The importance of executable simulation models in the requirements and design process of systems is increasing. Especially for complex systems with many parameters and dependencies between components, a simulation of the end system can predict the final behavior and reveal design flaws in early stages of system development. The model in these stages can be seen as an executable specification.

In later stages of system development these models can be reused to find optimal operational parameters, to check customer configurations of the system, or for debugging purposes. In these stages, models and generic modeling/simulation tools are often used by system experts which are often no modeling experts. This gap can lead to configuration errors in the models and wrong result interpretation. Diverse models and simulation tools are used to describe different aspects of the system. Parameters are often spread over simulation models and model hierarchies.

A *simulation-based application* (SBA) could fill this gap [1]. It hides the complexity of the model and allows to configure parameters and analyze results in one single application. Therefore, it consists of components for setting parameters, control simulations, and to analyze results. Figure 1 depicts a simple possible structure of an SBA with one simulation model and a database for exchanging configuration and result data between application and simulation. A comparison between simulation and hardware results is also possible in an extended setup [2].

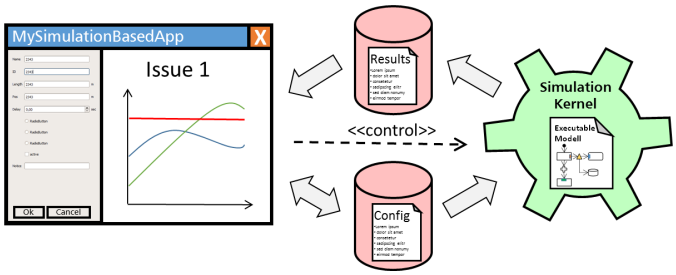

Fig. 1. Structure of a simple simulation based application.

The development of such simulation-based applications can be regarded as a realization of software product lines for simulation-based applications. Software product lines are used to implement flexible, reusable, expandable and complex software systems [3], [4].

Until now, the development of an SBA highly depends on the domain of the system models and adds a lot of extra effort to the creation of the simulation model. The use of a specialized framework for SBA creation [1] reduces the effort substantially. However, the system modeling and software development tasks, the definition of the content of the configuration and result database, the definition of conditions and dependencies between these database elements, and the design of the GUI are still major tasks for the development of a simulation-based applications. All these tasks are similar for each SBA, but cannot be summarized in a single framework.

To summarize the tasks and refocus the main effort on the system simulation model, it is necessary to raise the description of the SBA to a higher level of abstraction. This can be done by using techniques from model-driven development, which try to describe a software system using different kinds of models. A domain-specific language (DSL) has to be defined first. This can be done by deriving the DSL from a meta-meta-model such as the meta object facility

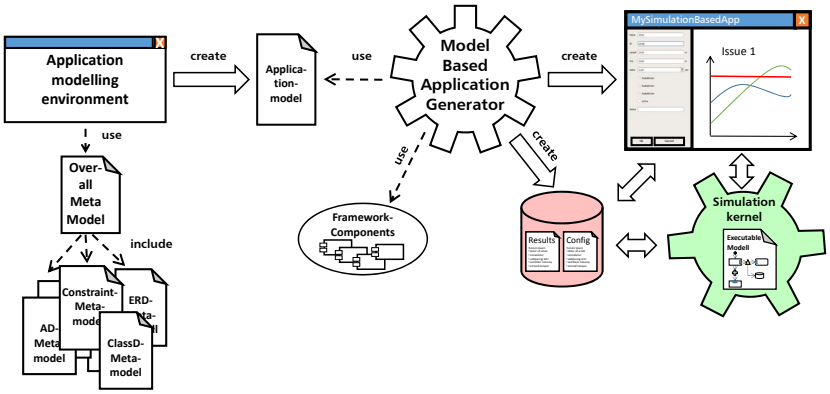

Fig. 2. Main components of the framework.

(MOF), or extending the universal modeling language (UML) with profiles [5].

With the aid of transformation and generation algorithms, a platform-independent model (PIM) can be transformed into platform-specific models (PSM) [6]. Moreover, automatic generation of source code as well as other artifacts is possible as well [7]. The process was described by the Object Management Group (OMG) within a number of standards [8], [9].

Over a number of years, a comprehensive collection of tools for model driven development was developed under the Eclipse project. The underlying meta-model can be defined by using the eclipse modeling framework [10]. It uses Ecore as a meta-meta-model which is similar to the essential meta object facility (EMOF) for describing a meta-model. The Ecore meta-meta-model is a well-developed, lightweight model with around 20 classes which is the base for all Eclipse modeling, transformation, and generation tools. An up-to-date implementation of the UML reference meta-model is provided as well, which supports profiles. The UML is an expressive specification for most aspects of the modeling of software systems. Unlike Ecore, it can describe not only the structure, but also the behavior of a software system.

Starting from this meta-model graphical editors can be created with the graphical modeling framework (GMF)[11]. It simplifies the process of creating editors by using mapping models and generators.

Sirius [12] and its predecessor Obeo Designer allows to specify different viewpoints of the model. They claim to rapidly increase the speed of development of graphical editors from 30 days for GMF to 5 days for Sirius [13].

For model-to-model transformations, the Eclipse modeling project implemented the OMG QVT standard [6] for both the declarative and the imperative language.

To generate artifacts or source code from the model, the Eclipse modeling projects provide the generator Acceleo [14] an implementation of the OMG's MOF Model-to-text transformation standard [7]. It uses templates and queries on model elements for the generation process. With these techniques it is possible to reduce the effort of developing a SBA.

This paper presents an approach to use model-driven development techniques to improve the development of simulation-based applications. A comprehensive UML meta-model is used to define the structure and behaviour of components of the SBA, the system simulation model, and the data-exchange components. For this purpose an EMF like UML Generator in C++ is used, wich is presented in [15].

Firstly, the description of workflow and needed models are described in Section II. An excerpt of our current state of the model-driven development method including data model and our developed C++ framework implementing the generic elements of SBAs are shown in Section III. The used simulation model of wireless sensor networks for avionic applications is the result of former projects which are already presented in [16], [2], [17].

II. Methodology

This section describes the overall structure and components of our proposed workflow for creating a model-driven development toolchain for SBAs.

Figure 2 sketches our suggestion of a model-based application generator for simulation-based applications with a simple architecture such as shown in Figure 1.

In order to describe a specific SBA, our proposal adds an application modeling environment (left side of Figure 2). The important aspect of the application modeling environment is the comprehensive meta-model, which is used to create application models of the structure and behavior of simulation-based applications. It includes sub meta-models for data model, simulation control, analysis, constraints, and dialogs. These models strongly depends on the system domain; in this paper, a system model for wireless sensor networks in avionics is selected.

The model-based application generator in the center of Figure 2 creates the simulation-based application based on the application model and predefined components of our SBA framework.

By using techniques such as model-to-text transformations, the generator transforms the model to source code, configurations for components of the framework, scripts for creating databases, and model components for simulation tools.

A. Model

The basis for model-driven techniques is a domain-specific language. Such a language consists of a meta/syntax-model, a semantic model, and a graphical notation of elements which can be used in actual models. The meta-model can be described with the help of a meta-meta-model like Eclipse Ecore or OMG MOF.

The OMG suggest three different kinds of extensions for a meta-model in its UML specification [5]: A first-class extension, an extension at the level of the UML meta-model, and a mixture between both.

A first-class extension is handled through the MOF and allows to define subclasses and associations. The resulting meta-model is independent from the UML meta-model. The expressiveness of the UML language is lost or has to be recreated in this case.

An extension at the level of the UML meta-model is done with *Profiles*. These profiles define *Stereotypes*, *TaggedValues*, and *Constraints* which extend the meta-model elements with new semantics, structure, or icons. An introduction of new meta-model elements is not possible. This kind of extension is applied to the model at the design time of the domain model.

For our approach we need elements which can express both syntactic and semantic aspects of the software system. The lightweight extension through *profiles* is sufficient in our case, because it suffices to extend the existing UML meta-model only. This is supported by the UML 2.5 reference implementation of the Eclipse framework, which can be used in nearly all Eclipse modeling tools.

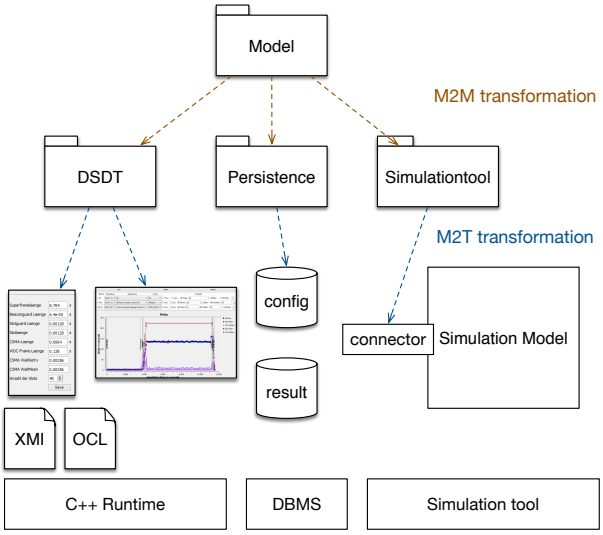

Fig. 3. MDA approach to generate major components of SBA.

An SBA consists of three major components: an application, a data-exchange component, and the simulation model. All of these artifacts are based on the same platform-independent model which describes the structure of data, constraints, and behavior. An overall view of a simple system is depicted in Figure 3.

The platform-specific models for an *SBA*, namely, *Persistence* and *Simulation Tool*, are derived from the platform-independent model by using a model-to-model transformation. As we are using the Eclipse modeling tools, this transformation is done by QVT. Each model is included in a separate *Package*. To distinguish between different scopes, *Stereotypes* are applied to the packages.

a) SBA: The *SBA* package is transformed from the platform-independent model and contains all elements which are required to build a simulation-based application. It has the Stereotype *SBAModel*.

The model has to be extended by platform-specific implementation details to target different simulation-based application platforms. It is possible to select only the needed data structures for the object layer.

b) Persistence: The *Persistence* package represents the data storage model, which is extended with the Stereotype *PersistentModel*. The package is derived from the platform-independent model and adds information to the platform-specific model. It is used to describe access to and deployment of application data.

The use of multiple *PersistentModel* packages in the model allows the modeling of a distributed data storage and as well as their interdependencies.

If a new specific storage should be added, an extension has to be defined. This extension includes a profile for the specific storage model which extends elements of the *PersistentModel*, a transformation described by a QVT-transformation and necessary generator files. The transformation could enhance the platform-specific model by adding creator and access functions to the classes.

c) Simulation Tool: The simulation tool's access to application data is specified via tool-specific model elements. This sub-model is represented by packages with the Stereotype *SimulationTool*. The transformation is the same as the former one described for sub-models. The possibility to target different simulation tools by adding multiple *SimulationTool* packages allows the communication between different simulations.

B. Views

There are five main views on the model, which are shown in Figure 4.

1) Data view: The data view is the central view of the model. It focuses on the structure and relation of available data in the software system. The modeled elements of this view are used to generate the object layer of the simulation-based application, the description and the access to the database, as well as data structures of the simulation tool and their mapping.

2) Constraints view: Based on the data view, the constraint view defines conditions and dependencies between configuration entities, simulation parameters, and system parameters.

A possibility to model constraints is the use of SysML constraint blocks [18]. With these blocks it is possible to define constraints using the object constraint language (OCL [19])

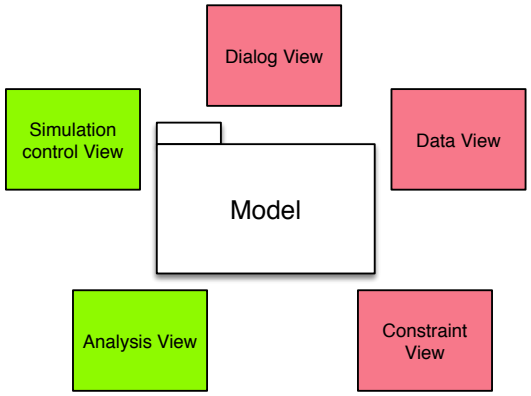

Fig. 4. Different views on a model for SBAs (red: structure views, green: behavior views)

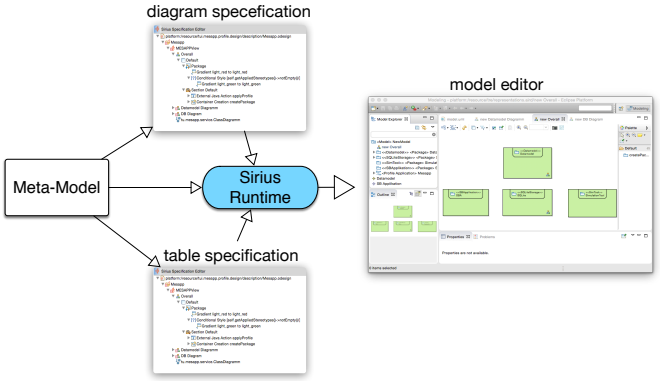

Fig. 5. Overview of the Sirius workflow.

and attach them to model elements. Validation of constraints can be done during run time using an OCL interpreter.

3) Dialog view: Dialog editors are well-known software tools to define graphical user interfaces. In combination with data and constraints, this view maps the data model entities to dialog elements for the defined constraints. Moreover, properties for GUI representation are added by using stereotypes.

4) Simulation control view: The simulation control view is a behavioral view of the simulation-based application, and focuses on activities for controlling simulations. This may include the definition of optimization loops [16] or a simulation deployment. Constraints, results of analysis and other data in the system can be used to control the flow.

5) Analysis view: Similar to the simulation control view, the analysis view is a behavioral view on the model focusing on activities to analyze results of a simulation. These activities can be described by signal-processing-like pipelines where the simulation results are processed.

C. Graphical editors

Besides the specification of the meta-models, the notation of the models in diagrams helps the user to rapidly understand and create models. Based on the meta-models described above, different model-based editors can be generated by using the graphical modeling framework (GMF), a specialized component of the Eclipse Modeling Project (EMP).

In contrast to GMF, Sirius [12] extends this approach by allowing to specify multiple views on the model in a single specification. This specification is interpreted by the Sirius runtime environment, and allows to view changes in the design nearly in real time. This rapidly speeds up and simplifies the creation of model editors; diagrams, tables, and tree editors are possible. Figure 5 depicts the workflow of Sirius.

Sirius also allows to write extensions for a diagram such that additional platform-specific extensions can be integrated easily into existing diagrams.

Our model describes structure as well as behavior, thus we need different kinds of corresponding diagrams.

Structural diagrams are needed for the top view model, the data view, and constraints view. For these purpose we use package diagrams, class diagrams, mapping diagrams, parametric diagrams especially for the constraints view. A sample structural editor is shown at the right side of Figure 5.

Behavior models such as the Simulation control view or the pipelines of the analysis view are modeled with behavior diagrams similar to activity diagrams.

The dialog model has a special status: For this kind of model, a common graphical user interface editor is useful.

D. Generator

The mentioned editors are used to create models of simulation-based applications. Based on these models, an application generator will generate source code, configurations, simulation model elements, and result queries for accessing data.

The specific generator output for a model depends on the contents of platform-specific extensions. The generation is done with a model-to-text transformation. For this paper we use the tool Acceleo, an implementation of the OMG standard, which is part of the Eclipse modeling project.

The Acceleo tool is used to generate the Ecore and UML meta-model packages, and object layer for C++ to allow the loading of xmi files as well as the OCL constraint validation during runtime. An Ecore-to-C++ and a UML-to-C++ generator were developed for this reason, which generate C++ classes and reflection packages of the model. These generators are described in detail in [15]. Besides the C++ code generator there are also generators for database schemata, configurations, and for components to access the data in the simulation tool available.

E. A Framework for Domain-Specific Simulation Design Tools

The presented workflow is now used to generate the components of a simulation-based application framework.

1) Framework Design: From an abstract point of view, the overall framework design could be described as a white-box, component-based framework. It is based on the plugin architectural pattern [20] and depicted in Figure 6 as a UML2 component diagram [5]. Different block colors visualize the various types of framework components. The software system

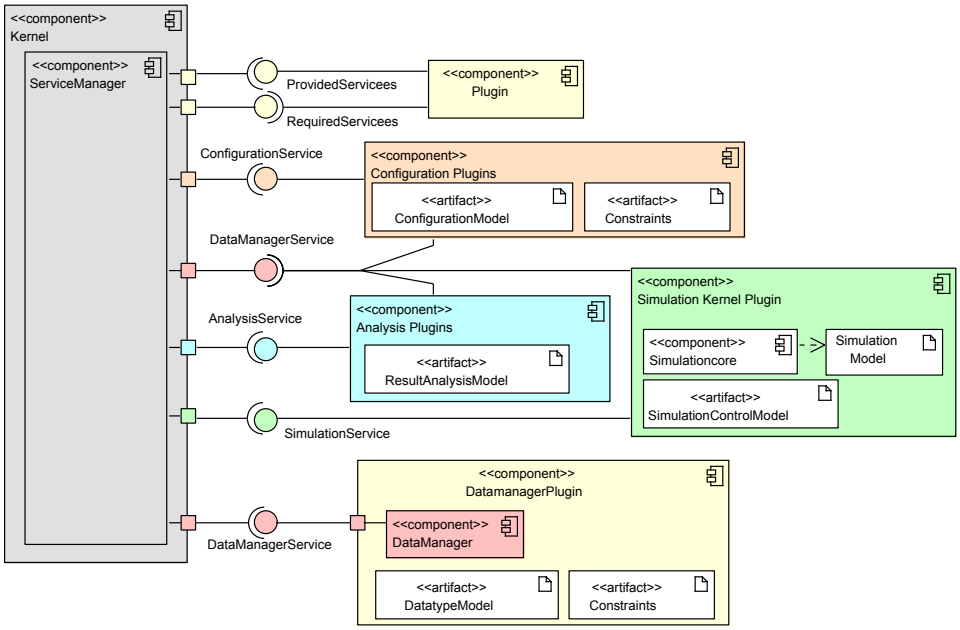

Fig. 6. Main components of the plugin framework.

consists of a small main component, including a service manager, which loads all plugins during application startup. Each individual functional component is encapsulated into one separate plugin, which provides or requires several services. These services are defined as interface classes. First of all, the plugins register their provided services to the service manager. If a plugin requires a service, the service manager will provide the service.

This framework design is implemented by using the CTK plugin framework [21], an OSGi [22]-like modularization framework for dynamic, modular C++ applications.

The realized simulation-based application includes the following types of plugins:

- Configuration plugins: Components with the purpose of defining or modifying simulation input data and model configurations for dynamic model elements.
- Analysis plugins: Components for static or interactive visualization and analysis of simulation results.
- Simulation control plugins: Components used to control the execution of the simulation kernel.
- Datamanager plugins: components which provide factories to create objects based on predefined data structures.

All these plugins are designed in a generic way to support Model-Driven Development for different domain-specific requirements. Therefore, the plugins are customizable by using the defined domain-specific models.

This allows to use model-to-text transformation capabilities of model-driven development frameworks such as Eclipse's Acceleo, which we use to generate a code representation of the domain-specific model/metamodel or parts of plugins.

By using a code representation of the metamodel it is possible to load an xmi representation of the actual model, which allows an easier change of parameters.

III. Example

This section demonstrates the proposed usage of the described meta-models, editors, and generators. It includes excerpts of the data model and the underlying C++ framework. These components are part of a project for simulating wireless sensor networks in an avionic setting as described earlier [2], [16].

Figure 7 presents the specific application of profiles for the data model.

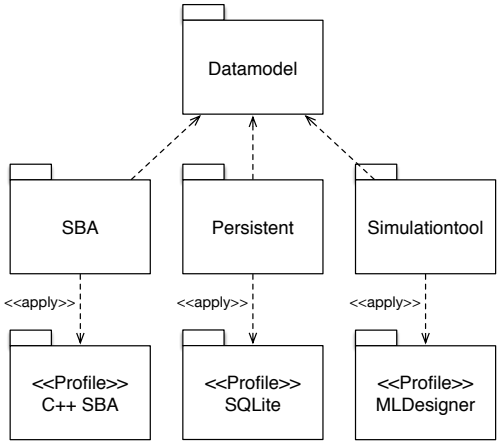

Fig. 7. Package structure of the model for an simple simulation-based application with application of platform specific profiles.

The basis is an abstract description of the data in the model, which is done by interface classes, shown at the top of the figure. Three sub-models are derived with the help of

a QVT transformation. To target a specific generator, these derived packages are annotated with platform-specific profiles for *C++ SBA*, *SQLite*, and the underlying simulation tool *MLDesigner* [23].

The *C++ SBA* profile summarizes all aspects of the simulation-based application. A subordinated aspect is the representation of data elements in the GUI. Properties of a data element, which can be manipulated throughout the simulation-based application, are enhanced by stereotypes and generated or interpreted during runtime.

Figure 8 shows an excerpt of the profile which creates the specific dialog for an editor plugin.

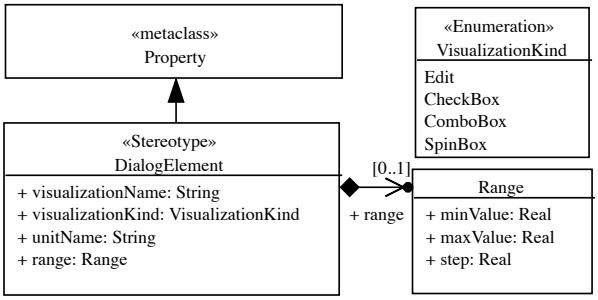

Fig. 8. Excerpt of a profile for GUI elements.

The profile extends the meta-class *Property* with a stereotype *DialogElement*, which adds the properties visualization name, visualization kind, unit name, and range.

The editor plugin queries the reflection package during runtime of the SBA to get access to the properties and applied stereotypes for a type. If a property of the type has an element with a stereotype application, the editor shows the property with the designated parameter in the GUI.

Figure 9 shows a part of a PIM model with applied stereotypes of an arbitrarily chosen example element *MAC_Hybrid*.

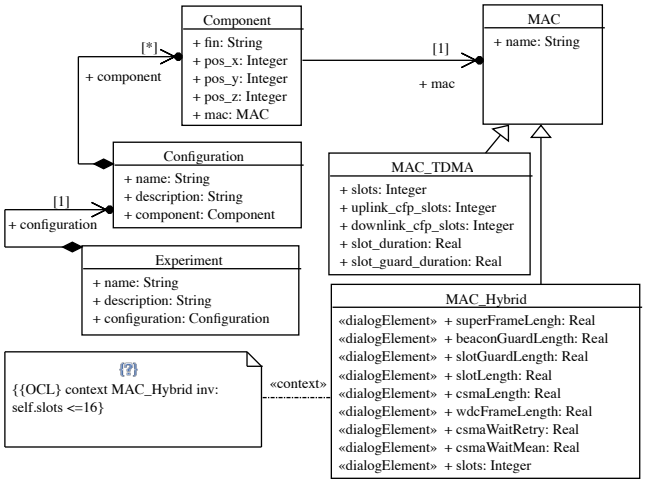

Fig. 9. Excerpt of the data model of a simulation-based application for simulation of wireless sensor networks.

The example shows a part of the model of a simulation-based application for planning and simulating wireless sensor networks in aircrafts. Different configurations of these networks have to be simulated, analyzed, and compared with each other, to find an optimal network configuration during system design. It shows the derived properties for the protocol of the network, which are part of the configuration of the components.

With the help of constraints it is possible to define property ranges or dependencies between properties. An OCL interpreter, which is also written in C++, is used to check these constraints in our SBA. It uses the model reflection capabilities of the generated object layer to gather information about the meta-classes and their constraints. It checks the objects against the constraints with the actual user-specified values, before a simulation is started.

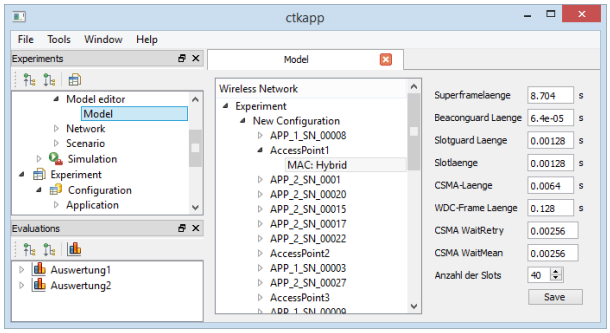

Fig. 10. The simulation-based application with a generated editor dialog.

Figure 10 depicts the resulting dialog which was generated from our model.

IV. Conclusion / Further Work

This paper presented an overview of our approach towards model-driven development of simulation-based applications by using tools of the Eclipse project for a C++ application. A small example, which creates GUI dialogs from UML models, shows the capabilities of our tool to generate components for our framework based on a domain-specific model. These techniques will significantly decrease the time and effort for developing such kind of applications.

This paper presents an excerpt of the current status of our project for simulating wireless sensor networks in an avionic domain. Further tasks include the generation of the components for the data abstraction layer, which connects the application with the simulation tools and allows access to data and model sources. The application behavior is planned to be modeled and executed using a C++ fUML implementation.

Acknowledgments

The work has been supported by the Federal Ministry of Economic Affairs and Energy of Germany (FKZ:20K1306D).

REFERENCES

[1] R. Maschotta, S. Jäger, T. Jungebloud, and A. Zimmermann, "A framework for agile development of simulation-based system design tools," in *Proc. IEEE International Systems Conference (SYSCON'13)*, 2013.

[2] S. Jäger, T. Jungebloud, R. Maschotta, and A. Zimmermann, "Model-based QoS evaluation and validation for embedded wireless sensor networks," *Systems Journal, IEEE*, vol. PP, no. 99, pp. 1–12, 2014.

[3] I. Philippow, K. Bllert, D. Streitferdt, and M. Riebisch, "Methodical aspects for the development of product lines," in *2nd WSEAS International Conference on Information Science and Applications*, 2002, pp. 30–45.

[4] D. Streitferdt and A. Mansoor, "Experiences of a product line migration project," in *Proceedings of the 2012 IEEE 36th Annual Computer Software and Applications Conference*, ser. COMPSAC '12, 2012, pp. 99–104.

[5] OMG, "Unified Modeling Language TM (OMG UML), Version 2.5," Object Management Group, Tech. Rep., June 2015. [Online]. Available: http://www.omg.org/spec/UML/2.5/PDF

[6] ——, "MOF 2.0 Query/View/ Transformation Specification," Object Management Group, Tech. Rep., January 2011. [Online]. Available: http://www.omg.org/spec/QVT/1.1/PDF/

[7] ——, "MOF Model to Text Transformation Language 1.0," Object Management Group, Tech. Rep., January 2008. [Online]. Available: http://www.omg.org/spec/MOFM2T/1.0/PDF

[8] ——, "Model driven architecture (mda) mda guide rev. 2.0," Object Management Group, Tech. Rep., 2014. [Online]. Available: http://www.omg.org/cgi-bin/doc?ormsc/14-06-01

[9] ——, "Meta object facility (mof) 2.0 core specification," Object Management Group, Tech. Rep., 2040. [Online]. Available: http://www.omg.org/cgi-bin/doc?ptc/03-10-04

[10] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework 2.0*, 2nd ed. Addison-Wesley Professional, 2009.

[11] R. C. Gronback, *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*, 1st ed. Addison-Wesley Professional, 2009.

[12] Eclipse, "Sirius," 2014. [Online]. Available: http://wiki.eclipse.org/Sirius

[13] E. Juliot and J. Benois, "Viewpoints creation using obeo designer or how to build eclipse dsm without being an expert developer?" Obeo, Tech. Rep., 2010, obeo designer whitepaper.

[14] Eclipse, "Acceleo," 2014. [Online]. Available: http://wiki.eclipse.org/Acceleo

[15] S. Jäger, R. Maschotta, T. Jungebloud, A. Wichmann, and A. Zimmermann, "An EMF-like UML generator for C++," in *4th Int. Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, 2016.

[16] A. Wichmann, S. Jäger, T. Jungebloud, R. Maschotta, and A. Zimmermann, "System architecture optimization with runtime reconfiguration of simulation models," in *IEEE Int. Systems Conference (SysCon 2015)*, Vancouver, Canada, April 2015, pp. 660–667.

[17] S. Jager, A. Zimmermann, and R. Maschotta, "A simulation-based system design tool for avionic fiber-optical networks," in *Systems Conference (SysCon), 2014 8th Annual IEEE*, March 2014, pp. 330–336.

[18] OMG, "Systems Modeling Language (OMG SysML), Version 1.3," Object Management Group, Tech. Rep., 2012. [Online]. Available: http://www.omg.org/spec/SysML/1.3/

[19] ——, "Object Constraint Language (OCL). Version 2.3.1," Object Management Group, Tech. Rep., 2012. [Online]. Available: http://www.omg.org/spec/OCL/2.3.1/

[20] M. Fowler, *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[21] (2015, 4) Ctk - the common toolkit. [Online]. Available: http://www.commontk.org

[22] O. Alliance, "OSGi Service Platform, Core Specification, Release 4, Version 4.2," OSGI Alliance, Tech. Rep., Sep. 2009.

[23] A. Agarwal, C.-D. Iskander, R. Shankar, and G. Hamza-Lup, "System-level modeling environment: MLDesigner," in *IEEE Int. Systems Conf. (SysCon 2008)*, Montreal, Canada, April 2008.