MODEL-DRIVEN ENGINEERING FOR TRANSPARENT ENVIRONMENTAL MODELING AND SIMULATION

F. Kühnlenz[1], F. Theisselmann[1,2], J. Fischer[1]
[1]Humboldt-Universität zu Berlin, Germany,
[2]Helmholtz Center Potsdam German Research Center for Geosciences, Germany

Corresponding author: Frank Kühnlenz Humboldt-Universität zu Berlin, Department of Computer Science
10099 Berlin, Unter den Linden 6, Germany, kuehnlenz@informatik.hu-berlin.de

Abstract. In this paper we address the issue of model reusability and transparency across different application contexts. It is particularly challenging to consider both, technical aspects and the process of how a model and the knowledge associated with it has been obtained, including, for example, the experimental setup. We show that technologies and concepts from Software Engineering, in particular meta-model-based model-driven engineering (MDE), offer means to meet the challenge of transparency and reuse in modeling and simulation (M&S). We provide the concept and an architecture for a model-based development of an integrated modeling system including meta-model-based DSL and experiment management. We apply this approach to environmental modeling with cellular automata, with a respective modeling language and experiment management system.

1 Introduction

One well-known issue in modeling and simulation (M&S) is to provide model reusability and transparency across different application contexts. It is particularly challenging to consider both, technical aspects and the process of how a model and the knowledge associated with it has been obtained, including, for example, the experimental setup [1, 2].

Models are an important instrument for gathering information about dynamic systems. In addition, models are a means to store and communicate the (assumed) knowledge about a system from different viewpoints. As such, it is vitally important that both, the model and the process of how a model has been created, modified and used, are transparent. Transparency means that model descriptions are comprehensible and that all artifacts associated with the modeling and simulation (M&S) process are consistent.

The process of simulation modeling usually follows an iterative, cyclic experimentation pattern in which a simulation model is being *formalized*, *executed* and *analyzed* in order to achieve a investigative goal. In order to provide transparency in this process, it is vitally important to document all relevant information and to provide consistency of model descriptions, model configurations, execution environment and model observations.

Existing M&S-frameworks may provide transparency to some degree, e.g. with the provision of domain-specific modeling languages (DSLs), but modelers are usually bound to their chosen framework [3]. Moreover, many M&S-technologies are designed for particular class of models and respective modeling formalisms, particularly domain-specific technologies, e.g. for environmental M&S. This is an issue when models are reused within different application contexts with different functional or non-functional requirements. Usually, the reuse of models requires the re-implementation or transformation of models in another implementation technology with other functional and non-functional properties and different modeling languages [4, 5]. Important contextual information is likely to be lost this way.

We focus on three aspects of M&S, *(improving) model description*, *model execution* and *model evaluation*. Accordingly we deal with the following issues:

- modeling transparency and consistency across framework boundaries
- model reuse across framework boundaries
- transparency of model and experimentation descriptions
- support model coupling

Technologies and concepts from Software Engineering, in particular meta-model-based model-driven engineering (MDE), offer means to meet the challenge of transparency and reuse in M&S. In meta-model-based MDE, software (e.g. a simulation model) is not formalized by code, but by means of models. These models conform to a meta-model, not to a specific programming language. The meta-model defines the modeling language, that is used to produce a model of a software (i.e. simulation model). Thus, a meta-model is comparable to the grammar of traditional computer languages.

With meta-modeling there are many possible ways of achieving reusability. A meta-model-based DSL can be used to describe a model without dependencies to particular implementation technologies [6]. MDE provides tool

support to efficiently define transformations from technology independent models different to target technologies (e.g. M&S technologies). This way, a model can be used different M&S frameworks, for example.

Moreover, MDE provides means to specify automated transformations between different meta-model-based modeling languages, that possibly are related to specific technologies. This way automatic transformation between respective technologies is possible. Moreover, the implementation of necessary, DSL-specific tools (e.g. editor) can be automated easily with meta-model based technologies.

Meta-model-based MDE can also be used to couple different DSLs and their respective models, if the DSLs' meta-models conform to the same meta-meta-model. The meta-meta-model defines a language that is used to define meta-models. This facilitates the coupling of models, that are defined in different DSLs.

We apply the concept of MDE with meta-modeling to environmental modeling and simulation in order to support transparent M&S.

In following we present, the architecture of a meta-model-based system for transparent M&S, and how we applied this concepts to simulation modeling and modeling with environmental cellular automata. This is complemented with the MDE-approach to experimentation.

2 Integration of DSL with experiment management

In order to provide transparency and reusability, we integrate modeling with DSL and experiment management. For this, we exploit model-driven engineering with meta-modeling.

The experiment management ensures the consistency of artifacts, that are necessary for a simulation (especially simulation observations), so that during analysis, the modeler is provided with all information that characterize experimentation: e.g. parameter settings, results, characteristics of the simulator (code generation process, software version, algorithms used etc.) and the machine (account, operating system, execution time, etc.).

DSL provide modeling concepts that are aligned with the concepts of the modeler, not with concepts of specific general-purpose programming languages. Thus, DSL support transparency and ease-of-use for domain experts.

We describe the architecture of our system, selected aspects of meta-model-based MDE, and how these relate. This is followed by a description of the application of this architecture to environmental modeling with cellular automata (ECA) within our prototypical modeling system ECA-EMS (Environmental Cellular Automata modeling and Experiment Management System).

2.1 Architecture of ECA-EMS

Figure 1 shows the architecture of the ECA-EMS and can be read according to the supported three workflow phases: experiment planning, experiment execution and evaluation of the experiment results. The *Experiment Management System* (EMS) ensures the consistency of all artifacts necessary for a simulation within the three phases, so that during analysis, the modeler is provided with all information needed: e.g. parameter settings, results, characteristics of the simulator (code generation process, software version, algorithms used etc.) and the machine (account, operating system, execution time, etc.). The Eclipse Rich Client Platform is a suitable platform for integrating our tools under one uniform GUI and additionally, it provides standard interfaces for user specific extensions.

Within the planning phase the *ECA Editor* is used for editing the ECA model, which is stored into the *Model Repository* (belongs to the *Experiment Repository*). The *Experiment Editor* provides a GUI including a configuration language to define certain experiments with their input artifacts on a technology independent level (like the ECA model itself).

In the *pre-execution phase* a *Builder*, which is in our concrete ECA-EMS a specific code generator, provides the technology dependent code accepted by the *Compiler*. The resulting binary object will be linked against the *GIS-* and *Simulator Libraries* to form the *Simulator Binary*. The *Simulator Binary* is executed during the execution phase that is followed by the evaluation phase.

The complex task of evaluating the experiment results is accomplished by the *Experiment Evaluator*, which basically provides statistical views on the data and mainly by the *Visualizer/GIS*[1]. In fact we rely on the GIS for visualization that offers powerful methods doing this task.

2.2 Model-driven engineering for simulation modeling

In model-driven engineering, software is described by models, not by code [7]. Our approach is based on the concepts of the Model-driven Architecture (MDA) as defined by the Object Management Group (OMG) [6]. One key idea is that a model (of a software) can be described on different levels of abstraction. Lower-level models

[1] A geographic information system (GIS), captures, stores, analyzes, manages, and presents data that refers to or is linked to location (source: http://en.wikipedia.org/w/index.php?title=Geographic_information_system\&oldid=254371420).

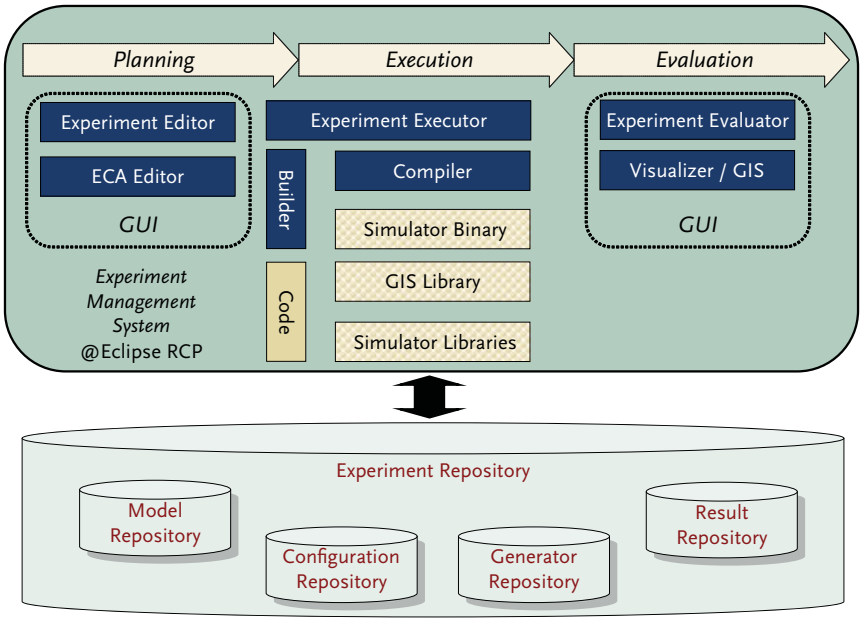

Figure 1: Architecture of ECA-EMS

contain more technical detail than models on higher levels. With this it is possible to specify models with no syntactic dependency to specific technical platforms. Technology specific models reside on lower levels. Higher-level models are automatically transformed to models on lower, technically more specific, levels by automated model transformation or code generation.

An important element of this approach is *meta-modeling*. Meta-models define the formalisms, that are used to specify level-specific models. These meta-models prescribe the modeling elements that are available and the relationships between them. Thus, a meta-model defines a level-specific modeling language, which can be complemented with rules of how the language is actually presented (textual, graphical symbols). Thus, a DSL can be defined by means of meta-models.

The relationship between the modeling levels is formalized by transformations. A transformation can be of type *model-to-model* transformation, where a model is transformed into a model, or of type *model-to-text* transformation (a.k.a code generation), where a model is transformed into a textual representation of a model. Usually, a textual model representation is source code in a programming language. In our model-driven simulation modeling approach, we use two levels of abstraction (see Figure 2). On the higher level, a simulation model is modeled using a modeling-formalism-specific DSL. On the lower level, the simulation model is represented as source code. This source code is specific to the respective target technology.

Possible target technologies are characterized by the functionality they offer and the way this functionality is accessible. We propose to use existing, generic simulation and GIS technology as target technology. It is important, that the functionality is accessible via source code, since source code is finally generated in our approach. A common pattern for the provision of simulation and GIS functionality is provide libraries that can be used with general purpose programming languages. These technologies offer simulation or GIS functionality combined with the expressive power of the host programming language. Generic simulation technologies that follow this pattern are process-based simulation libraries (e.g. jDisco [8] and ODEMx [9]) or event-based libraries that follow DEVS-based formalisms [2, 10]. Also, there are generic GIS technologies like Geotools[2] or ArcObjects[3] that follow this pattern.

Although, programming languages, programming interfaces and modeling paradigms of available generic simulation technologies are diverse, the offered functionality often follows common generic concepts (e.g. event-driven simulation, model-composition and coupling, standard algorithms for solving differential equations, standard statistical functions etc.). In the field of spatial data processing, there are extensive standardization efforts by ISO[4]

[2] http://geotools.codehaus.org
[3] http://edndoc.esri.com/arcobjects/8.3
[4] ISO/TC 211 Geographic information/Geomatics, http://www.isotc211.org

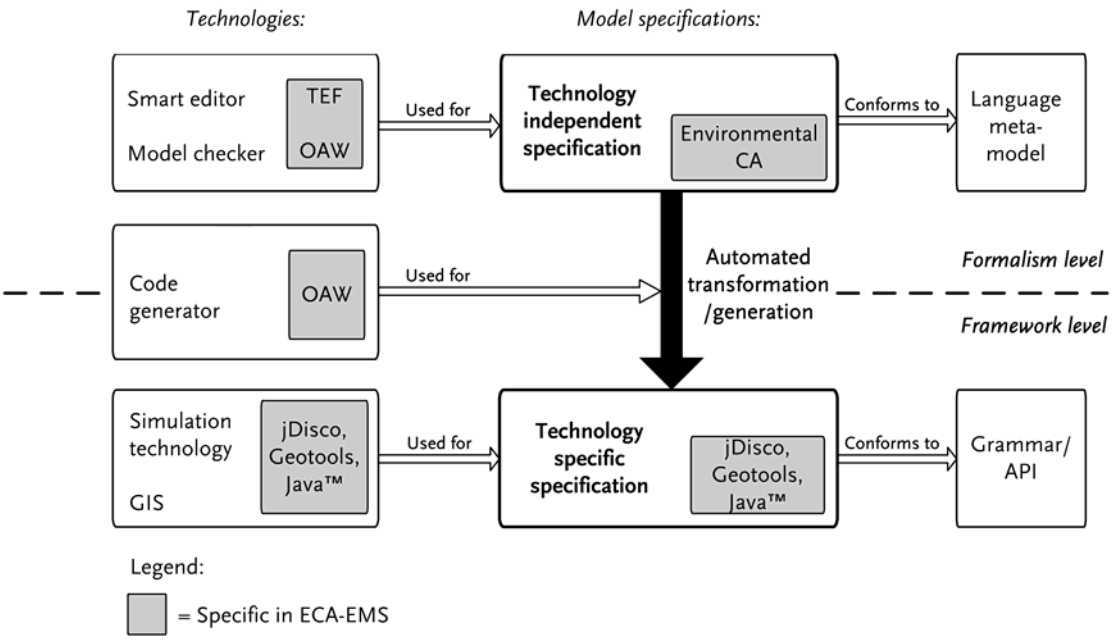

Figure 2: Two-level DSL approach for environmental cellular automata modeling

and OGC[5], which show and promote a functional convergence of GIS technologies. Thus, there is the possibility to exchange the simulation and GIS technology without changing the higher-level technology-independent model description. For model reuse with different technologies, a new code generator has to be specified.

In this approach, different DSLs may be used to describe different parts or aspects of the modeled system. Different DSLs may implement different modeling formalisms. With meta-modeling it is possible to integrate different DSLs for modeling. Integrated DSLs have in common that their respective meta-models are defined by means of the same meta-meta-model. The meta-meta-model defines the language, that is used to define a meta-model. By this, it is technically possible to integrate different meta-models (DSL respectively), e.g. for interdisciplinary modeling with different DSL for different disciplines. However, the different DSL must not provide means to express models that cannot be executed by the (possible) target technologies. For this, it is necessary to use powerful target technologies like generic event-driven simulation technologies in combination with generic general purpose programming languages. Table 1 gives an overview of meta-modeling, respective modeling levels and examples of supporting technologies.

Language modeling level	Purpose	Technology (examples)	Tools (examples)
M_3: Meta-meta-model	Language for language definition	Ecore, Meta-Object Facility	Eclipse Modeling Framework (EMF)
M_2: Meta-model	Language for model definition	ECA, Unified Modeling Language	EMF
M_1: Model	Model definition	XMI	EMF, Textual Editing Framework, OpenArchitectureWare
M_0: Executable	Execution	Binary	Compiler, runtime environment

Table 1: Overview of language modeling levels

Model-driven engineering is concerned with the *efficient* provision of tools that facilitate the definition of domain-specific modeling languages, the specification of model transformators and code generators. The fact that meta-models define modeling languages facilitates extensive automation in this field. Based on high-level specifications and meta-models, necessary tools (e.g. editor, generator) can be automatically be generated.

2.3 Meta-model based modeling of ECA

We applied this approach to environmental modeling with cellular automata (ECA). For examples of such models see [11, 12, 13]. A graphical overview is given in Figure 2. For this we defined a DSL, which is particularly tailored to ECA-modeling. The language allows the modeler to define time-driven ECA using high-level, specific

[5]Open Geospatial Consortium, http://www.opengeospatial.org

language concepts and avoids expressivity that is not needed. By this it supports transparency in the modeling process, since language concepts are aligned with the problem domain and model descriptions do not contain technology specific details. With this DSL it is possible to define ECA with discrete and continuous states, arbitrary dynamic neighborhoods, cell grouping, spatial reference and to define the coupling of different ECA models to more complex ECA models. Some code snippets of a model can be seen in Figure 4.

Figure 3 shows how the meta-model of the ECA-modeling language is structured with the use of packages. Within the *SimCore*-package, basic language elements (datatypes, expressions) and the structure of model descriptions is modeled. The language concepts for ECA-modeling are provided with the *ECA*-package. Within this package there are references to elements within the *SimCore*-package. The persistence of output data is described with special kind of models (data sink models) that are coupled to models. For the description of data sink models there is a special (sub-)language, which is modeled within the *DataSinkModeling*-package. This language is integrated with the ECA-modeling language by referencing common modeling elements from within the *SimCore*-package. The integration with the EMS, requires to produce output in a specific way. For the specification of this, we implemented a small language for the specification particular EMS-datasink-models. This language is modeled within the *EMSDataSink*-package. This language integration is possible, because all meta-models are formalized by means of the same meta-meta-model[6]. (see Table 1).

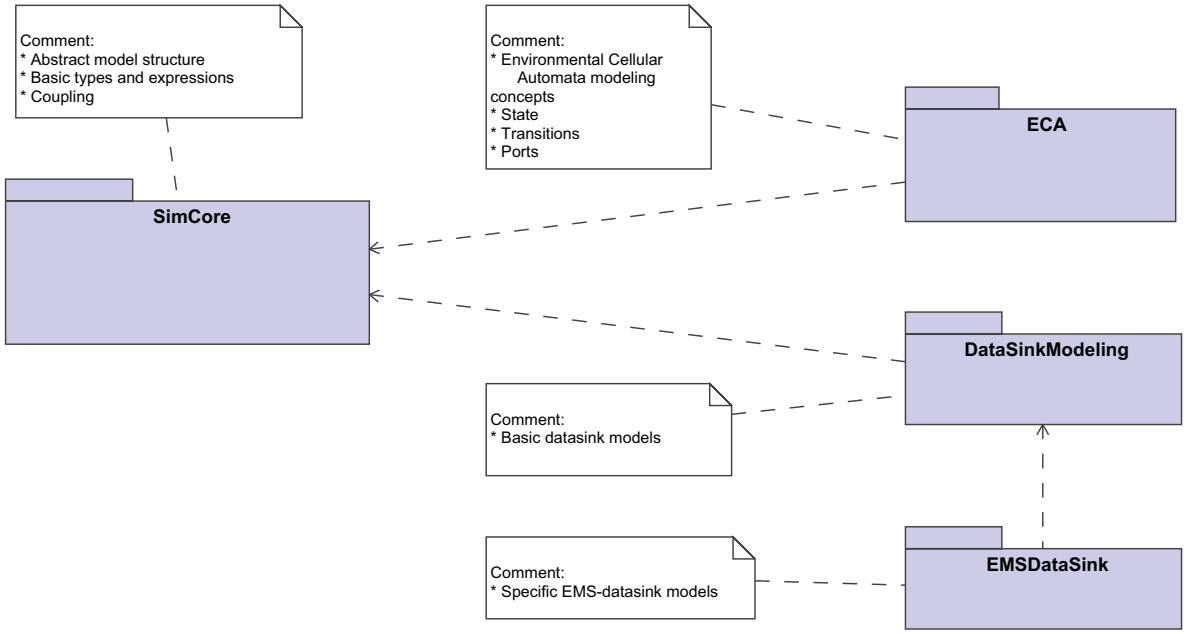

Figure 3: The structure of the composed meta-model for ECA modeling (UML notation)

For illustration of how the languages are used, Figure 4 shows a collage of code snippets taken from a reimplementation of an ECA model by means of our languages. The original model is described in [11]. The reimplementation is produced with our languages. The code snippets in the upper two boxes are part of the definition of the ECA model which is conform to the meta-model in the package *ECA* (see Figure 3). The third box defines a EMS-datasink-model which is used for the storage of output (meta-model package *EMSDataSink*). The coupling of the ECA-model and the EMS-datasink-model is specified in the code box at the bottom (meta-model package *SimCore*).

In our implementation we use a combination of *jDisco* and *Geotools* as our target technology for model execution. JDisco is a process-based simulation library and Geotools is a library that provides OGC/ISO-standard conform GIS functionality. Both libraries are written in Java, so that integration is straightforward. The code generator is specific to the ECA-language's meta-model, to the Java-grammar, and the target technologies' programming interfaces. For the description of code generators, the model-driven engineering community already provides ready-to use tools: we use *openArchitectureWare*[7] (OAW) for this (see Figure 2). The resulting Java-source code is compiled and executed using the standard tools and runtime environment.

For model-editing, we provide a smart textual editor (with syntax highlighting, code-completion etc.) that is automatically generated from the meta-model and a grammar using the Textual Editing Framework (TEF, [14]). The grammar describes the textual syntax of the language and the relationship of textual symbols with the meta-model. Code snippets from this editor can be seen in Figure 4.

[6]We use Ecore as our meta-meta-model. http://help.eclipse.org/ganymede/index.jsp?topic=/org.eclipse.emf.doc/references/overview/EMF.html

[7]http://www.openarchitectureware.org

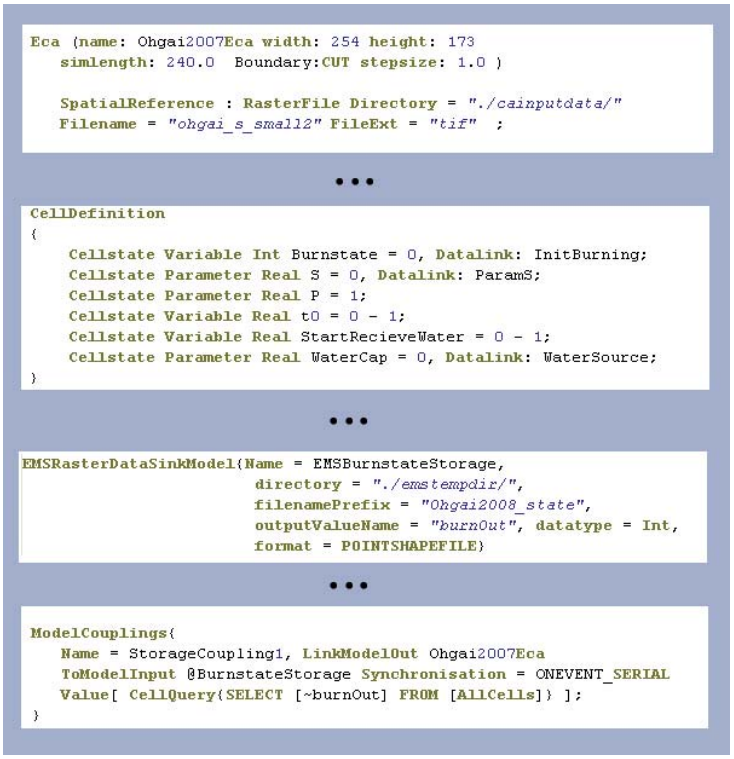

Figure 4: Collage of code snippets of an ECA model from TEF-textual editor

2.4 Model-driven experiment engineering

According to our approach using model-driven engineering to be independent from a specific technology in modeling, also simulations with that model are handled in a technology independent way. Considering the simulation models to be managed as black-boxes defining input and output parameters, this definition is important for the EMS infrastructure (mainly its datamodel). The runtime representations of these models have defined input parameters and a system state that can be usually described by values of their state variables. The values of the input parameters are part of the experiment configuration whereas the values of state variables are time dependent and kind of interest to be stored during the simulation execution within the EMS Repository. ECA models use spatial references, which can be stored also within the EMS Repository. Therefore the EMS Repository is realized in a relational database management system (RDBMS) with a spatial extension (PostgreSQL with Postgis).

Based on an *Experiment Management DSL* it will be possible to describe project-specific properties for experiment management and generate the database schema for the EMS Repository and corresponding access methods (API) to store the experiment configuration and model observations (e.g. state variables). Actually this is work in progress and performed manually at the moment.

For the planning phase, the EMS tool provides a second DSL, the *Experiment DSL* for describing experiment sets. Figure 5 shows the meta-model of this DSL as UML class diagram. Based on this meta-model a concrete syntax is specified using the same TEF tool as for ECA model editing [14]. The result is a smart textual editor (with syntax highlighting, code-completion etc.) for specifying experiment descriptions in a textual way.

Figure 6 shows a screenshot of this editor with the experiment description *simple repetition multi-run example* that simply executes one experiment in sequence with the desired 100 runs (without varying the input parameters). This is useful for non-deterministic simulation models to analyze the statistical qualities of these models, as it is the case in the models represented in Figure 4.

Nevertheless the fundamental idea of experiment sets is to vary input parameters in a certain way, e.g. within a defined range and step-width without the need to plan each single variation in a single experiment by hand. Figure 7 shows a screenshot of the textual experiment editor with a more complex example. In this example the experiment set *range multi-run example* defines a `RangeMultiRunSpecification`[8] for the two input parameters `P2` and `P3`. `P2` has to change from 1 to 5 in a step-width of one and for `P3` only the values 1 and 2 can be assigned. Therefore a total of 10 experiment runs are necessary to execute this experiment set.

It is also planned to provide other common multi-run descriptions, e.g. the Monte Carlo method. Of course it is also possible to combine both types of experiment description in the same experiment set.

In the evaluation phase EMS supports to order and to rate the experiments of such an experiment set, e.g. in

[8] For the `RangeMultiRunSpecification` type the execution support is not available yet.

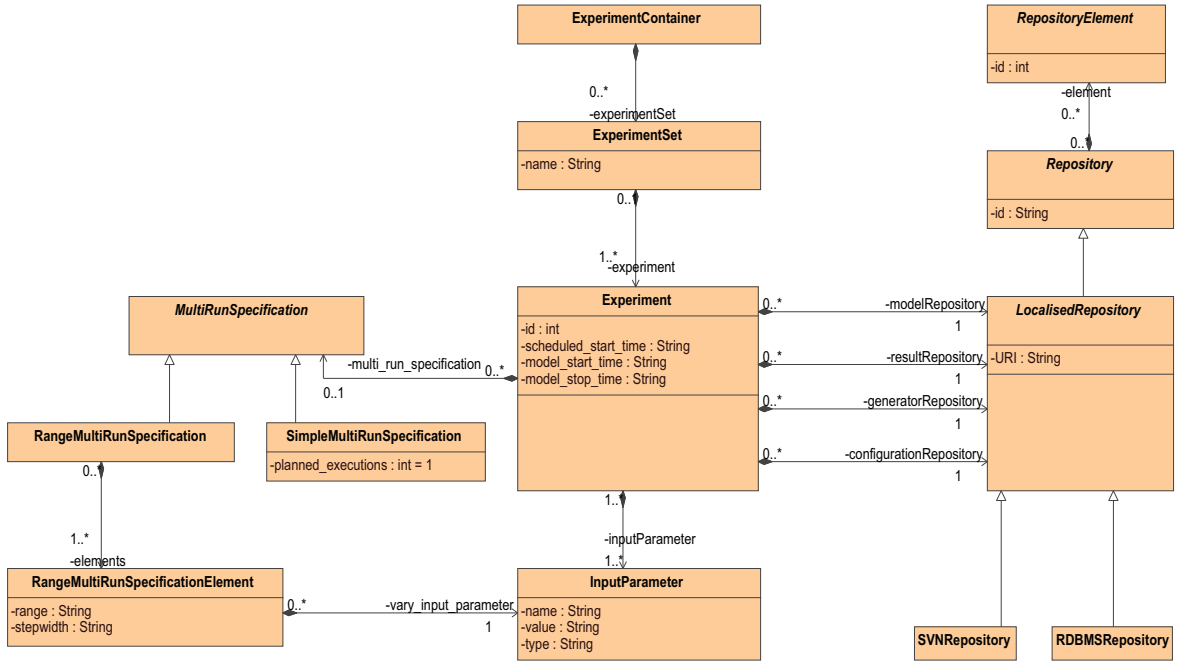

Figure 5: Meta-model of the Experiment DSL for describing experiments (UML class diagram)

```
Experiments{
  ExperimentSet{
    name ="simple repetition multi-run example";
    Experiment{
      id =2; scheduled_start_time ="2009-02-10 14:00:00"; model_start_time ="0"; model_stop_time ="5000";
    }
    modelRepository=SVNRepository("svn+ssh://repo.informatik.hu-berlin.de/modRepo");
    resultRepository=RDBMSRepository("repo.informatik.hu-berlin.de");
    generatorRepository=SVNRepository("svn+ssh://repo.informatik.hu-berlin.de/genRepo");
    configurationRepository=RDBMSRepository("svn+ssh://repo.informatik.hu-berlin.de");
    InputParameters{
      Parameter "P1", value="1", type="Integer";
    } MultiRunSpecification{
      SimpleMultiRunSpecification(100);
    }
  }
}
```

Figure 6: Screenshot of the textual editor showing the experiment description *simple repetition multi-run example* and a code completion box on cursor position

```
Experiments{
  ExperimentSet{
    name ="range multi-run example";
    Experiment{
      id =2; scheduled_start_time ="2009-02-10 14:00:00"; model_start_time ="0"; model_stop_time ="5000";
    }
    modelRepository=SVNRepository("svn+ssh://repo.informatik.hu-berlin.de/modRepo");
    resultRepository=RDBMSRepository("repo.informatik.hu-berlin.de");
    generatorRepository=SVNRepository("svn+ssh://repo.informatik.hu-berlin.de/genRepo");
    configurationRepository=RDBMSRepository("svn+ssh://repo.informatik.hu-berlin.de");
    InputParameters{
      Parameter "P1", value="1", type="Integer";
    }
    MultiRunSpecification{
      RangeMultiRunSpecification {
        RangeMultiRunSpecificationElement{
          Parameter "P2", value="2", type="Byte";
          range="1..5", stepwidth="1";
        }
        RangeMultiRunSpecificationElement{
          Parameter "P3", value="2", type="Byte";
          range="1..2", stepwidth="1";
        }
      }
    }
  }
```

Figure 7: Screenshot of the textual editor showing the experiment description *range multi-run example*

dependence of a specific state variable. So far, switching the framework for executing the simulation model means for EMS only changing the build-configuration to be able to generate and compile the new simulator binary.

3 Conclusion

Model-driven engineering facilitates support for transparent simulation modeling processes. Our contribution is the transfer of concepts from software engineering to the M&S domain. We provided the concept and architecture for a model-based development of an integrated modeling system including meta-model-based DSL and experiment management.

We applied this approach to environmental modeling with cellular automata, with a respective modeling language and experiment management system. The DSL provides the means to specify a model independently from implementation technology. The EMS ensures consistency of M&S artifacts.

The use of model-driven engineering facilitates the separation of model descriptions and implementation technology. With our prototype (ECA-EMS) we showed that ECA models can be transformed to the target technologies jDisco and Geotools. These technologies are exemplary for a number of similar technologies, providing common simulation modeling and GIS functionality. However, the transformation to other possible target technologies (e.g. ODEMx and ArcObjects) has not been implemented yet, but is planned for future work.

Meta-meta-modeling facilitates the integration of several DSLs for technology independent modeling. However, it must be clear that the possible concepts of DSLs are limited by the functionality of the used target technologies.

Meta-modeling and automatic tool generation (e.g. editor, generator) allow for efficient language development and implementation.

4 Acknowledgement

The presented work is supported by Deutsche Forschungsgemeinschaft, Graduiertenkolleg METRIK (GRK 1324/1) and the European Union within the Sixth Framework Program in the SAFER Project.

5 References

[1] Cellier F.E.: *Continuous System Modeling*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1991.

[2] Zeigler B.P., Praehofer H., and Kim T.G.: *Theory of Modeling and Simulation*, Academic Press, San Diego, 2nd edn., 2000.

[3] Theisselmann F., and Dransch D.: *Improving the reusability of spatiotemporal simulation models: using MDE to implement cellular automata*, in: *Headway in Spatial Data Handling: 13th International Symposium on Spatial Data Handling*, (eds. A. Ruas, and C. Gold), Lecture Notes in Geoinformation and Cartography, 177–195, 2008.

[4] Argent R.M.: *An overview of model integration for environmental applications–components, frameworks and semantics*, Environmental Modelling & Software, 19:219–234, 2004.

[5] Argent R.M., and Rizzoli A.: *Development of Multi-Framework Model Components*, 2004.

[6] Miller J., and Mukerji J.: *MDA Guide Version 1.0.1*, 2003.

[7] Schmidt D.C.: *Model-driven Engineering*, Computer, 25–31, 2006.

[8] Helsgaun K.: *jDisco - a java package for combined discrete event and continous simulation*, Tech. rep., Department of Computer Science, Roskilde University, 2001.

[9] Fischer J., Ahrens K., Witaszek D., and Gerstenberger R.: *ODEMx - Object oriented Discrete Event Modelling*, http://odemx.sourceforge.net, 2007.

[10] Praehofer H., Auering F., and Reisinger G.: *An Environment for DEVS-Based Multi-Formalism Simulation in Common Lisp/CLOS*, Discrete Event Dynamic Systems: Theory and Applications, 3(2/3):119–149, 1993.

[11] Ohgai A., Gohnai Y., and Watanabe K.: *Cellular automata modeling of fire spread in built-up areas–A tool to aid community-based planning for disaster mitigation*, Computers, Environment and Urban Systems, 31:441–460, 2007.

[12] Rothman D.H.: *Modeling seismic P-waves with cellular automata*, Geophysical Research Letters, 14:17–20, 1987.

[13] Georgoudas I.G., Sirakoulis G.C., Scordilis E.M., and Andreadis I.: *A cellular automaton simulation tool for modelling seismicity in the region of Xanthi*, Environ. Model. Softw., 22(10):1455–1464, 2007.

[14] Scheidgen M.: *Integrating Content Assist into Textual Model Editors*, in: *Modellierung 2008*, Gesellschaft für Informatik e.V., 2008, lNI.