Model-driven Development of Environmental Modeling Languages: Language and Model Coupling

Falko Theisselmann, Doris Dransch, Joachim Fischer

1Humboldt-Universität zu Berlin, Department of Computer Science, Systems Analysis
Rudower Chaussee 25, 12489 Berlin
2Helmholtz Centre Potsdam GFZ German Research Centre for Geosciences
theissel@informatik.hu-berlin.de

Abstract
Common characteristics of environmental modeling and simulation (M&S) are multi-disciplinary modeling and the need to reuse models in different contexts. There is a plethora of M&S frameworks available, but still challenges remain, when models are developed across different communities and organizations with specific modeling paradigms, languages, and tools. Issues arise due to technical complexity, model reuse, and model integration. We target these issues with the provision of declarative domain-specific modeling languages (DSLs) that allow for problem-oriented and technology-independent modeling. With this approach, DSLs are used to define technology independent environmental simulation models. Executable code that conforms to a simulation technology of choice is automatically generated from these models, thus the models can be reused on different platforms.

The implementation of DSLs causes effort. However, the model-driven approach for the definition of DSLs, as opposed to grammar-based approaches, allows the efficient definition of DSLs and respective tools, in particular with respect to language coupling. Language coupling is necessary for multi-disciplinary modeling, where different parts of a model may be defined using different DSLs. In this paper, we present a model-driven language engineering approach and show how object-oriented language modeling can be the base for coupling DSLs. The semantics of coupled models are based on established concepts of model decomposition and event-driven simulation. Since many generic simulation technologies implement these concepts, this facilitates model reuse on many platforms, although programming languages and interfaces might differ. We applied this approach to a DSL for Cellular Automata modeling and a simple DSL for describing computational agent models and defined code generation for one exemplary simulation framework combined with Geographic Information System technology (GIS). So far, the DSL has been used to reimplement published models of fire spread and seismicity. However, the approach is not limited to the presented DSLs, framework technologies, and application areas.

Keywords: Domain-specific modeling language, meta-model, Model-driven Architecture, language coupling

1. Introduction
Common issues in environmental modeling and simulation arise due to multi-disciplinary modeling and the aim to reuse models in different contexts (e.g. science, and application). The transfer of tools and technologies from computer science to environmental modeling already resulted in a number of technologies that face these issues and support efficient modeling and model application. While early technologies were mainly concerned with coupling simulation with Geographic Information System (GIS) technology and (legacy) model integration, object-oriented modeling and component-based implementation allow the incorporation of knowledge about the structure of modeled systems in the simulation software design (e.g. component-based design) and the implementation structure (e.g. distributed computing).

But, there is still the issue of the limited reusability of models, since models are usually bound to a particular modeling tool and a programming language. Argent and Rizzoli (2004) discusses several methods of integration of model components across tool boundaries. The authors state that a declarative modeling approach is the most generic, where models are defined by means of declarative statements that can be in-
terpreted or transformed to executable models. Declarative model descriptions, in contrast to procedural model descriptions, focus on the modeled domain and do not contain the definitions of algorithms that execute the model. Moreover, declarative models should not have syntactic dependencies to particular modeling languages. For example, a declarative specification of differential equations is formulated without the numeric approximation algorithm and without using the language of a specific tool.

In this paper, we present an approach to declarative modeling with domain-specific modeling languages (DSLs). The aim of our approach is to efficiently produce declarative DSLs for environmental modeling and simulation (EMS) through the application of meta-modeling and Model-driven Architecture (MDA). Our presentation focuses on a method to define and couple several DSLs into a new coupled DSL. This facilitates multi-paradigm modeling, as conceptually described in Vangheluwe et al. (2002), where models are defined using several modeling formalisms with respective DSLs. Traditionally DSLs are defined on the base of grammars, but the assistance for the coupling of grammar-based DSLs is poor and tool specific (Krahn et al. 2008), thus, implementation is costly. Instead of syntax-centered grammars, we use object-oriented meta-models for the definition of DSLs. This approach naturally provides the means for language coupling (Holz, 2003) and allows for intuitive and efficient implementation of coupled DSLs. In chapter 2, we sketch the main conceptual foundations of this model-driven DSL development approach (chapter 2.1), with focus on language and model coupling (chapters 2.2 and 2.3). Chapter 3 presents the application of this approach to a DSL for Environmental Cellular Automata modeling (ECA) and a DSL for describing simple computational agent models. Chapter 4 provides conclusion and outlook.

2. Model-driven meta-model-based development of DSLs

In our approach, environmental systems are modeled using declarative DSLs. Different submodels of a model may be described using different DSLs. For the definition of DSLs, we combine object-oriented meta-modeling, which is used to model DSLs, with Model-driven Architecture (MDA). MDA ensures the independence of simulation models from specific simulation frameworks.

2.1 MDA for modeling on different levels of abstraction

One key idea of the Model-driven Architecture (MDA), as defined by the Object Management Group (OMG), is that any software is described by models on different levels of abstraction (Miller and Mukerji, 2003). Figure 1a illustrates MDA for two levels of abstraction. At the upper level, the Platform Independent Model specifies a model without using features of specific implementation platforms. The lower-level, Platform Specific Model (PSM) is a model of the same system, but it contains platform specific details for execution. In the context of MDA, a platform is any technology that provides functionality through interfaces and usage patterns. A higher level model is automatically transformed into a model on the lower level by means of an automated tool-based transformation. Such model transformations can be of type model-to-model transformation, where a model is transformed into another model, or of type model-to-text transformation (a.k.a code generation), where a model is transformed into a textual representation (e.g. source code).

Figure 1b presents our application of the generic two-level approach to EMS. At the higher level (formalism level), a simulation model is described by means of modeling-formalism-specific DSLs (i.e. Cellular Automata and Agent DSL). This specification does neither contain dependencies to particular simula-

1 Please note that in this paper we distinguish between declarative modeling of environmental systems with the means of DSLs and the modeling of DSLs (abstract syntax and static semantics) with the means of meta-models.
tion frameworks nor details of how the model is computed at runtime. This declarative technology-independent specification is automatically transformed into executable source code. The source code is specific to a target technology, i.e. it is relates to a particular simulation framework (i.e. jDisco, see chapter 2.4.). Each level, and the transformation between them, is related to specific tools. For using DSLs, tools, such as editors and model checkers are needed to specify correct models at the formalism level. For model execution, the generated code must be processed with tools of the target technology (i.e. compiler, runtime environment, editor). Typically, the generated code must conform to the grammar of the target technology’s language and a programming interface (API). In contrast, the DSLs on the formalism level

On the framework level, our approach aims at exploiting existing framework-level technologies, thus, there exists tool support (editor, compiler etc.). But the tool support for new DSLs has to be implemented for each DSL. However, there exists off-the-shelf technology for storing and processing of models that are based on meta-models. By using this, the implementation of necessary tools, such as editors and transformers, can be highly automated due to the use of meta-models.

2.2 Language coupling on the formalism level

The definition and coupling of DSLs is based on meta-models. Meta-models define the language elements that are available in a DSL (e.g. data structures and functions). In our approach, there exists a specific meta-model for each of the coupled DSLs. Figure 2 (left) presents a well established three level meta-modeling approach. At level 1, a model, which is defined with a DSL, conforms to a meta-model. The meta-model (level 2), which defines the DSL, conforms to its meta-meta-model at level 3. The meta-meta model defines a language for defining meta-models. In our approach, all meta-models are defined by means of the same meta-meta-model, thus, each meta-model conforms to the same meta-meta-model (figure 2, right). With an object-oriented meta-meta-model, meta-models can be structured and related in an

Figure 1: (a) The general MDA pattern (source: Miller and Mukerji 2003, modified) applied to (b) the meta-model-based approach for modeling domain-specific modeling languages.
object-oriented way. By this, different DSLs can be coupled by simply referencing common elements of the meta-models (see chapter 3.1). Figure 2 (right) illustrates this, where the meta-models of the languages $L_1$ and $L_2$ ($M_{M1}$ and $M_{M2}$) conform to the same meta-meta-model and reference common meta-model elements. By this, a new meta-model ($M_{ML}$) is defined that incorporates the coupled meta-models ($MM_{ML1}$, $MM_{ML2}$). The resulting DSL allows for the definition of complex models, where the coupled models ($ML_{1}$, $ML_{2}$) are defined with specific DSLs.

2.3 Model coupling on the framework level

While language coupling refers to the relation of DSLs on the syntactic level, model coupling refers to the semantics of such couplings, or in other words, the realization of models and couplings on the framework level.

On our approach, the semantic foundation of modeling is a collection of well known established generic modeling and simulation concepts. These concepts can be seen as a common denominator of existing simulation frameworks, which conform to these concepts (see Evert et al. 2005). Such common characteristics are common data types, a common generic decompositional modeling structure, or standard algorithms, such as numerical solvers for differential equations. For modeling the dynamics and the synchronization of model coupling, we adopt the idea of event-driven simulation, which is applicable to models with discrete state changes with a continuous time base. With event-driven simulation, it is possible to model systems with discrete, as well as continuous behavior (Zeigler et al. 2000). Model coupling is realized by relating state variables of models directly or implicitly with the use of ports.

Environmental models and the exchanged data are usually related to space. For the relation of models to space and the geospatial referencing of data, we adopt generic concepts of geospatial modeling that are formalized by the standardizations of ISO (ISO/TC 211 Geographic information/Geomatics, URL: http://www.isotc211.org) and Open Geospatial Consortium (OGC, URL: http://www.opengeospatial.org).

2.4 Simulation technology on the framework level

On the framework level, we propose to use existing modeling and simulation technology in combination with GIS technology, if necessary. Possible target technologies are characterized by the functionality they offer (see chapter 2.3) and the way this functionality is accessible. There exist a number of simulation
and GIS technologies that provide the necessary functionality. However, programming languages, programming interfaces, and modeling paradigms are diverse.

It is important, that the required 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 the provision of 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, for example, process-based simulation libraries, e.g. jDisco (Helsgaun 2001) and ODEMx (Fischer and Ahrens 1996) or event-based simulation libraries that implement DEV-based formalisms (Zeigler et al. 2000). Also, there are generic GIS technologies like Geotools (URL: http://geotools.codehaus.org) or ArcObjects (URL: http://edndoc.esri.com/arcobjects/8.3) that fulfill the requirements for GIS-technologies and that can be integrated with simulation frameworks to support environmental modeling.

3. Application to a Cellular Automata and Agent modeling language

We applied the presented concepts to an implementation of a declarative DSL for Environmental Cellular Automata (ECA) modeling. The ECA DSL is designed for modeling complex ECA models. It has so far been used to reimplement published Cellular Automata models i.e. urban fire spread (Ohgai et al. 2007) and seismic wave propagation (Rothman 1987, Leamy 2008, Georgoudas et al. 2007).

For demonstration purpose, we defined a simple DSL for modeling simple agent models. An agent can perceive data from other models (ECA or agents) and may react on this perception as defined by a transition function.

Figure 3: The package structure of the modeling languages’ metamodel (UML 2.0 notation).

3. Application to a Cellular Automata and Agent modeling language

We applied the presented concepts to an implementation of a declarative DSL for Environmental Cellular Automata (ECA) modeling. The ECA DSL is designed for modeling complex ECA models. It has so far been used to reimplement published Cellular Automata models i.e. urban fire spread (Ohgai et al. 2007) and seismic wave propagation (Rothman 1987, Leamy 2008, Georgoudas et al. 2007).

For demonstration purpose, we defined a simple DSL for modeling simple agent models. An agent can perceive data from other models (ECA or agents) and may react on this perception as defined by a transition function.

433
3.1 Meta-models and language coupling

The coupled meta-models of the DSLs are defined by means of Eclipse Modeling Framework’s object-oriented Ecore meta-meta-model. Figure 3 shows the package structure of the coupled meta-model.

To enable the rapid implementation of new DSLs, basic language elements are modeled within the simCore-package. In this package there are, amongst others, common datatypes and mathematical expressions and the definition of the general structure of models. Abstract classes can be used as superclasses for specialization, non-abstract classes can be reused directly in the definition of DSLs.

The specific language concepts for ECA-modeling are modeled within the eca-package as specializations of simCore concepts. Analogously, the mobileAgent-package contains the meta-model of the agent-DSL. For data persistence and visualization the modeler specifies storage and visualization models that are coupled to the data producing models. Visualization and storage models are defined with respective DSLs (dataSinkModeling-package and emsDataSink-package2).

![Figure 4: Classes in the simCore-package (UML 2.0 notation).](image)

Figure 4 shows the main elements as classes within the simCore-Package. The rootModel-element represents the (coupled) model itself. It contains at least one SubModel. Each submodel contains variable definitions (not visible in Fig.4) and may contain the definition of InputPorts and OutputPorts. The definition of model couplings (Coupling) is contained in the rootModel. A Coupling definition contains a query which queries the value of a coupled model’s OutputPort and references an InputPort of another model to which the OutputPort is connected. Within a query it is possible to refine the coupling based on the state of the coupled models, e.g. to make a spatial selection based on a position (see example in chapter 3.2).

The couplingType attribute defines the synchronization of submodels in the case the event times of coupled models are equal. There is the possibility to specify PARALLEL and SERIAL (CouplingType). PARALLEL coupling means that the exchange of data takes place before the two models transit to a new state, so that the recipient model operates on the values prior to the transition. SERIAL coupling means

---

2 For further information on data persistence and experiment management within this approach see Kühnlenz et al. (2009).
that the model providing output executes the transition before the data is exchanged and the recipient model operates on values after the transition. With \textit{ONEVENT\_PARALLEL} and \textit{ONEVENT\_SERIAL} couplings, the modeler can specify that the time of the next event of a model (input) is dependent on the next event time of the coupled model (output), not on the model itself.

In our approach every DSL must contain a specialization of the meta-class \textit{SubModel} in the \textit{simCore} package. This convention ensures the coupling capability of every simulation model defined by the DSLs, since there are common elements in the meta-models, i.e. ports, variables, and types. Figure 5 shows a part of the meta-model of the agent DSL. The \textit{MobileAgentModel}-element represents the model itself and is a subclass of \textit{SubModel}. In addition to the inherited attributes, an agent model contains a definition of a \textit{SpatialReferenceSystem} and a transition function (\textit{AgentTransitionFunction}). This transition function defines the state change and it is executed at times of events. A detailed presentation of the meta-models is beyond the scope of this paper, but in the following chapter an example illustrates the coupling aspect of the DSL by means of an example model.

![Figure 5: Part of the meta-model of the agent language (UML 2.0 notation).](image)

### 3.2 Modeling coupled models with DSLs

This section presents the usage of the DSLs by means of screenshots of the definition of an example model. It is a simple agent model which is coupled to an ECA model. The ECA model models fire spread in an urban area (Ohgai et al. 2007). The agent model is purely fictive simple model for demonstration purpose only. The agent is virtually located in the space that is modeled by the ECA. The agent perceives and reacts on fire within its visible surroundings.

```plaintext
@mobileAgent(name = Agent, worldwidth: 254, worldheight: 173)
SpatialReference : Masterfile Directory = "./cainputdata/" Filename = "ohgai_s_small2" FileExt = "tif";
StateVariables
9 { State Variable Int PosX = 195;
10 State Variable Int PosY = 55;
11 State Parameter Int ViewWidth = 2;
12 State Variable List <CellData> View ;
13 }
InputPorts{
14 Input Name = viewInput, DataType = List <CellData>, Variable = View ;
15 }
OutputPorts{
16 Output Name = positionOut, DataType = List <CellData>,
17 Value = [CellDataList[CellData[X:PosX,Y:PosY,Value:1]]];
18 }
```

*Figure 6: Screenshot of the text editor for the mobileAgent-language.*

The definition of the model starts in line 6 (figure 6) with the name of the model (\textit{Agent}). The spatial reference is taken from a geodata file (l. 7). Typed variables and parameters model the position (\textit{PosX,PosY}) and the width of view (\textit{ViewWidth}) of the agent, and the perceived fire information from the ECA.
Figure 7 shows the definition of a coupling, where the transmitted values depend on the state of the coupled models. The coupling (ECAToAgentCoupling) links an output of the ECA model (Ohgai2007Eca) to the input port viewInput of the agent model (viewInput@Agent, l. 322). The synchronization is PARALLEL (l. 322). The value clause in lines 323-325 defines the query of values. The port burnOut is selected (SELECT [Port(burnout)]). If the position of any cell of the ECA model (X(cell), Y(cell)) is within the width of view of the agent (Agent>>ViewWidth, Agent>>PosX, Agent>>PosY), its value it is transmitted to the agent.

On the framework level, we use a combination of jDisco and Geotools as target technology for model execution. jDisco is a Java-based process-oriented simulation library. Geotools is a standard conform Java-library for geospatial data processing. The code generator is specified using off-the-shelf technology (OpenArchitectureWare, URL: http://www.openarchitectureware.org) that works on the basis of metamodels. The used text editor has been automatically generated using the meta-model technology TEF (Scheidgen, 2008). All DSL-technologies (meta-model processing, editor, transformer specification and execution) are off-the-shelf technologies that are embedded in the Eclipse Platform with its Eclipse Modeling Framework (EMF, URL: http://www.eclipse.org/modeling/emf/).

4. Conclusions and outlook

We presented a meta-model-based approach for implementing DSLs for environmental modeling and detailed how language and model coupling can be realized efficiently for two exemplary DSLs. With the application of MDA, declarative DSLs are independent of specific simulation and GIS frameworks. Thus, models can be reused with different simulation frameworks, if automated transformations exist. For this, possible framework-level technologies must follow well established generic concepts of decomposition, event-driven simulation, and standard GIS concepts, as for example the frameworks jDisco and Geotools.

The realization of coupled DSLs requires the implementation tool support for model processing by means of editors, model checkers and transformers. The use of object-oriented meta-models and existing meta-model-based technologies allows the implementation of coupled DSLs, where many tasks can be automated efficiently.

The direct reuse of legacy models is not possible in general with this approach. But, heritage models could be manually reimplemented. If the original implementation language of the model is meta-modeled too, the reimplementations of all models of this DSL could be automated with meta-model-based model transformations or the DSL could be directly coupled to other DSLs, as shown in this paper.

Common simulation technologies usually provide means for hybrid modeling incorporating continuous and discrete state changes. The definition of a hybrid cellular automaton formalism has been presented in Theisselmann and Dransch (2008), which is to be elaborated with respect to language and model coupling in the future.

In conclusion, the meta-model based model-driven approach to environmental modeling supports usage of declarative and framework independent DSLs by making the implementation of DSLs efficient. Envi-
Environmental modeling may profit from easy-to-use DSLs, transparent model descriptions, and reusable simulation models.

Acknowledgement

The presented work is supported by Deutsche Forschungsgemeinschaft, Graduiertenkolleg METRIK (GRK 1324/1).

References


Helsgaun, K. (2001): jDisco - a java package for combined discrete event and continuous simulation, Department of Computer Science, Roskilde University


