Improving the Reusability of Spatiotemporal Simulation Models: Using MDE to Implement Cellular Automata

Falko Theisselmann[1,2], Doris Dransch[2]

[1] Graduiertenkolleg METRIK, Department of Computer Science, Humboldt-Universität zu Berlin
[2] GeoForschungsZentrum Potsdam (GFZ), Telegrafenberg, Potsdam

Abstract

Numerous modeling and simulation tools, frameworks, and environments support domain experts with the implementation of spatiotemporal simulation models. The implemented models are usually bound to specific tools, because specific modeling languages, simulation engines, or processing platforms have to be used. To improve model reusability, we propose an implementation approach that applies Model Driven Engineering (MDE). In this approach, a simulation model is described on three different levels of abstraction. Starting from an abstract description of a simulation model by the modeler, this model is automatically transformed through all levels into executable code. In contrast to common implementation technologies, the intermediate steps of the transformation are clearly and formally defined by metamodels. For model execution, existing general purpose simulation and spatial data processing frameworks may be used. In this paper, the three-level approach and its application to the modeling of cellular automata are described. Partial metamodels and transformations are presented for two of the three levels. The MDE-approach provides means to enhance model reusability and promotes transparency in simulation modeling. Moreover, a tight integration of simulation and spatial data processing

can be realized by synthesizing executable software which is composed of generic spatial data processing and simulation functionality.

Keywords: Spatiotemporal modeling; Model Driven Engineering; Cellular automata

1 Introduction

Spatiotemporal simulation models are widely used to model the spatiotemporal behavior of environmental phenomena. An important characteristic of spatiotemporal simulation modeling is that models are executed in order to observe the spatiotemporal behavior of phenomena. Statements about the original system are derived from these observations. Usually, model execution is realized by the means of software, thus the simulation software is a key artifact in the simulation modeling approach.

The reuse of simulation models could reduce implementation effort and provide the means to share models. By this, the exchange of models between scientists and the transfer of scientific models to application domains, like disaster management or urban planning, can be supported.

To improve the reusability of spatiotemporal simulation models, we propose an implementation approach that applies Model Driven Engineering (MDE) methods and tools. Based on a high-level description of a model, an executable model is automatically generated. This model includes simulation and spatial data processing functionality.

In this paper, our approach is detailed for spatiotemporal modeling with cellular automata (CA). The next section provides an overview of common simulation model implementation approaches with the focus on simulation model reuse and software implementation. This is followed by the introduction of the concept of our three-level approach. After this, two levels of the three-level approach presented in more detail for modeling CA. For modeling CA on one presented level, we developed a generic formalism: the hybrid cellular automaton (HCA), which is completed by means to model access to spatial data. The second presented level is represented by a simulation framework (jDisco) and a library that provides spatial data processing functionality (Geotools). Geotools and jDisco are used to produce executable models. These technologies exemplify how executable models can be synthesized from a formal abstract description that is based on generic functionality. Moreover, simulation and spatial data processing can be integrated using this approach. The paper concludes with remarks and outlook.

2 Implementation Technologies and Approaches to Spatiotemporal Modeling

From an engineering point of view, the implementation of spatiotemporal simulation models requires the realization of simulation and spatial data processing functionality. Fig. 1 shows the tasks that we assume to be processed for a spatiotemporal simulation. Spatial data is used to provide the parameters for the simulation model. For this, it may be necessary to preprocess spatial data. Simulation processing is realized through the iterative calculation of the state of the model. These calculations may require simulation functionality (i.e. execution of transition, synchronization of submodels, data exchange) and spatial data processing. Depending on the modeler's needs, spatial data may need to be stored, processed and visualized before, during, and after a simulation run.

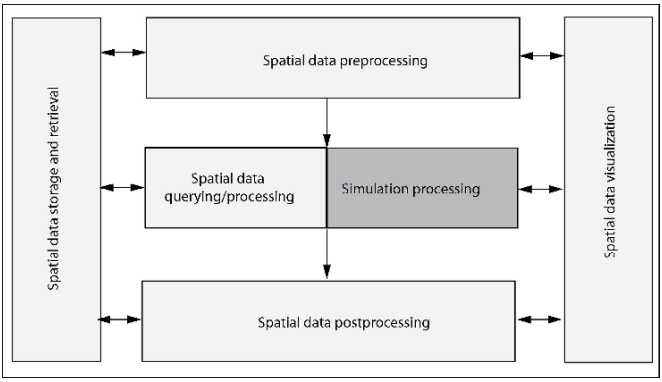

Fig. 1. Spatial data processing and spatiotemporal simulation. Spatial data handling (light grey) and simulation functionality (dark grey) has to be integrated

If spatial data processing functionality is needed before and after a simulation run, simulation and spatial data processing is executed serially. Functionalities are executed in parallel, if both functionalities are needed during simulation. Either way, the required functionality may be provided by an integrated modeling and simulation framework, or by distinct, frameworks (e.g. separate GIS and simulation frameworks)[1]. Serial execution and

[1] In the remainder of this paper the term *framework* will be used for any software that provides the functionality to model and simulate dynamic models. This subsumes software that is commonly named simulation library, tool, framework, language, or environment.

the coupling of respective frameworks, is mainly relevant for pre- and postprocessing of spatial data. This may be associated with tasks like georefencing, resampling, and filtering. In this case, data exchange between the frameworks may simply be realized by manual export and import of data. In the case of parallel execution of functionality, the integration of frameworks at runtime is required, where data exchange may be realized via shared memory or network based exchange mechanisms. Runtime integration can be used to read and write spatial data during simulation. This permits to persist intermediate results or to read subsets of spatial data selectively, depending on the state of a model at runtime. In this paper, the integration of simulation and spatial data processing functionality at runtime is relevant.

In the following, a brief overview of a selection of implementation approaches is given, focusing on implementation languages and model reuse. It is common practice to implement CA models using general purpose programming languages, but there are disadvantages, such as limited reusability and high implementation costs of the resulting simulation software. In the past, this led to the development of special purpose tools, frameworks, languages, and environments.

One approach to provide means of modeling and simulation is the development of high level *domain specific modeling languages* (DSL) and respective execution frameworks. As such, SELES (Fall and Fall 2001) and PCRaster (Karssenberg 2002) provide modeling languages to model spatiotemporal processes, based on a cellular discretization of space. These languages contain simulation functionality, including the calculation of state transitions, data input, output, and visualization, predefined functions (i.e. statistic, stochastic, algebraic) and additional simulation functionality, such as batch simulation (SELES 1999, Karssenberg 2002). Within these frameworks, modeling, simulation and spatial data processing functionality is highly integrated. Model reuse is possible within the respective frameworks, but the reuse of models in other frameworks is limited.

The idea of *component based modeling* is the core concept of numerous simulation frameworks which focus on model reuse and integration (Argent 2004). In component based modeling, the system under study is decomposed into connected interacting subsystems which are modeled as components. Although component based frameworks share this key idea, there are differences with respect to their architecture, the modeling concepts, interfaces, and the underlying technologies. Due to this heterogeneity, the reuse of components and models with different frameworks is difficult (Argent 2004).

One way to reuse models across framework boundaries is to use a declarative modeling language as an exchange format for models between

frameworks (Argent 2004). However, a generic formalism may be hard to use for domain experts, so that single models may be modeled with the use of a more adequate formalism and consequently be transformed to this generic formalism. With such approach named *multi-paradigm modeling*, Vangheluwe et al (2002) show that by using a common generic and expressive modeling formalism, also the integration of a variety of models is possible.

In an analysis of today's modeling and simulation frameworks for environmental modeling, Evert et al (2005) classify environmental modeling and simulation frameworks as *modeling-level* or *implementation-level* frameworks. Modeling-level frameworks provide domain specific abstractions to domain experts for the definition of models. With implementation-level frameworks, existing models are linked and executed. It is argued that the differences between implementation-level frameworks are relatively unimportant, so that code generators of modeling level frameworks could target a widely accepted implementation-level framework, based on a generic modeling formalism, such as DEVS (Zeigler et al 2000).

Our MDE-approach to spatiotemporal simulation model implementation is conceptually based on this suggestion of Evert et al (2005): Domain specific models, as specified by the modeler, are automatically transformed to executable code with well defined intermediate representations. But, instead of targeting a specific simulation level modeling formalism, we suggest to focus on generic simulation functionality that may be implemented by several frameworks, possibly conforming to different modeling formalisms. This facilitates the storage of models in a more independent way, allowing for more framework independence. Moreover, we integrate spatial data processing based on generic functionality that also may be provided by different frameworks in different application scenarios. Model Driven Engineering provides the tool support for the efficient realization of this approach.

3 A Three-level Model Driven Engineering Approach to Spatiotemporal Modeling

MDE is a software engineering approach to software development. The core idea of MDE is that software is mainly described by models, not by code. Executable code is automatically synthesized from these models (Schmidt 2006). Different models describe the same software on different levels of abstraction, which may be distinct in the amount of implementation detail they encompass. By the provision of domain specific modeling

concepts on the most abstract level, this approach promises to hide implementation detail from the domain expert, thus it helps to unburden the modeler from the need of detailed implementation (Muzy et al 2005). Moreover, an MDE-based approach facilitates platform independence, thus this approach enhances the possibility for the reuse of models on different platforms (Schmidt 2006).

The realization of MDE is based on a clear, formal definition of the levels of abstraction and the relationship between them. The different levels of abstraction are formalized by respective modeling languages. Transformations formalize the relationship between these languages. MDE technologies provide the necessary tool support to define modeling languages, model transformations, and code generators, which analyze and synthesize models and code (Schmidt 2006).

Key artifacts within this approach are metamodels, since they are used to define the modeling elements of the modeling languages and relationships between the modeling elements. All models must conform to metamodels, thus metamodels prescribe the set of possible models at each modeling level. For example, a metamodel may prescribe that a model may contain variables and that variables must have a name and a type. The transformations between models are defined on the basis of their metamodels.

The meaning of metamodels and the respective models is twofold. On the one hand the meaning of a model, or a model element, is given by the modeler, and the underlying understanding of the modeling elements and their relations, i.e. a variable may describe a model parameter or a state. On the other hand, as models are processed by the computer, meaning is given by model transformations that finally result in model execution, i.e. a variable is referenced memory that holds a value of a specific type.

We propose an application of MDE for spatiotemporal modeling, which is detailed for spatiotemporal modeling with CA. In this approach, a CA simulation software is explicitly modeled on three distinct levels of abstraction. At each level, the simulation model is represented by a level-specific model. The concepts that are available for modeling on the different levels are formally prescribed by level-specific metamodels.

Fig. 1 shows the concept of the three-level approach to implement of spatiotemporal simulation models. On the highest level, a *domain specific model* is defined by the domain expert using a domain specific modeling language (DSL). This DSL should be particularly tailored to the needs of the modeler and should provide modeling concepts with a clear relation to the problem domain (i.e. hydrological modeling, fire spread modeling). A domain specific model is automatically transformed to a *formalism specific model*.

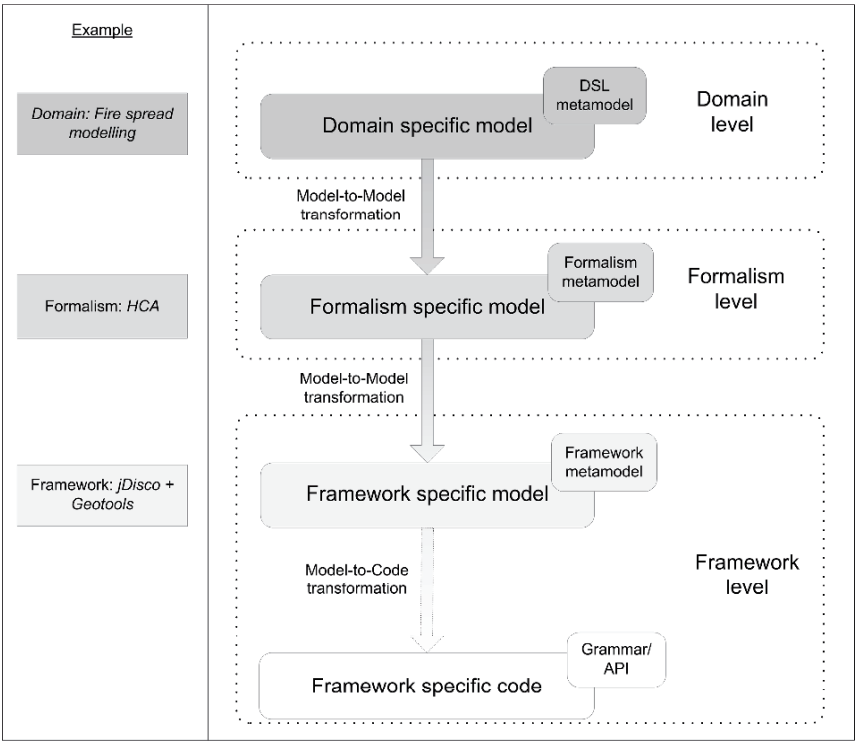

Fig. 2. A three-level, MDE-based implementation approach to simulation modeling

The formalism specific model is modeled using the concepts of a more generic modeling formalism in comparison to the DSL. For modeling on this level, we developed the HCA formalism as a generic modeling language for CA models (see Sec. 4). This level has two main characteristics. On the one hand, the formalism is generic, so that models of different domains may be expressed by the means of this language. On the other hand the formalism level is based on the functionality provided by the executing frameworks, since modeling concepts on the formalism level that can not be realized by frameworks do not make sense in simulation modeling.

A formalism specific model is automatically transformed to a *framework specific model* from which executable code is automatically generated by a code generator. A feature of our approach is to use an existing general purpose simulation framework to execute simulation models. Consequently, the concepts of the modeling language at the framework level

are provided by the respective simulation framework and are derived from the API and documentation. Since the approach is based on widely used simulation modeling concepts (see section 4.1), one may find a number of simulation frameworks that realize the required functionality.

The inclusion of spatial data processing functionality is based on the same principle. For this, it is assumed that there are widely accepted concepts related to the modeling of spatial data and spatial data processing. A common ground can be seen in widely used standards i.e. ISO and OGC standards. Software that provides implementation of these standardized concepts is to be combined with simulation frameworks.

In general, two application scenarios for this approach are possible. In the first scenario, several DSLs may be defined for different domains, but modeling uses the same modeling formalism on the formalism-level, i.e. HCA. For each DSL, a transformation to the formalism level is defined individually; an existing transformation from the formalism level to the framework level and code generation can be reused for all DSLs. In the second application scenario, existing models on the formalism level are executed by different frameworks, for example, if the model is reused in another application context, with a different simulation and spatial data processing infrastructure. For this, a new transformation from the formalism level to the framework level and code generation must be defined. The domain specific models, the formalism specific models, and the transformation from the domain level to the formalism level remain unchanged and can be reused in this scenario.

4 A Three-level MDE Approach to Model Cellular Automata

In this section, two levels of our approach, the formalism level and the framework level, are detailed for modeling cellular automata, as an example for the presented generic approach. The main modeling elements on the two levels are presented by the means of (partial) metamodels. An illustration of the relationship between the modeling levels concludes this section. A detailed presentation of the domain level is beyond the scope of this paper, but the two levels are sufficient to illustrate the main characteristics of the approach.

4.1 Common Modeling Concepts on the Formalism and the Framework Level

Basically, we adopt the notion of event-based modeling. A model has a state that changes at times of events. This discrete behavior is extended with the possibility to model continuously changing states. Moreover, a model can be composed of several coupled submodels. The state of a simulation model is modeled by state variables. For modeling state variables, standard data types (i.e. integer, float, string) and standard data structures (i.e. enumeration, array) are used.

Discrete state changes are modeled by discrete transition functions with the means of standard arithmetic expressions that calculate new values that are assigned to state variables. Continuous state changes are modeled by means of ordinary differential equations (ODEs). During a simulation run, ODEs are evaluated by the simulation framework to approximate state changes over time using an integration algorithm.

State changes depend on the value of state variables at certain points in simulation time. This dependency is modeled by conditions. A condition is an expression with which a model's state is evaluated: if the model is in a state that is specified by a condition, the respective condition is true; false otherwise. Within a condition, the model's state variables and input may be evaluated. Conditions are specified by means of standard relational, logical and arithmetic expressions.

Conditions are attached to discrete transition functions and ODEs in order to specify in which state which particular transition is to be executed. In addition, conditions are used to describe conditions on states, which trigger an event (state event). Events interrupt the continuous behavior of a model. Discrete transitions or communication with other models happen at times of events. Events may occur at arbitrary times, thus the time base of models is continuous.

Raster data provides the spatial parameters of a CA model and is used to store the spatial output of a model. Fig. 3 shows how the access to raster datasets residing in the file system can be metamodeled. In the example, a raster dataset is a *RasterDataFile* that can be of type *RasterDataASCIIFile*, *RasterDataWorldImageFile*, or *RasterDataGeoTiffFile*. To access the raster dataset, it is sufficient to provide the type and the location of the dataset (URL).

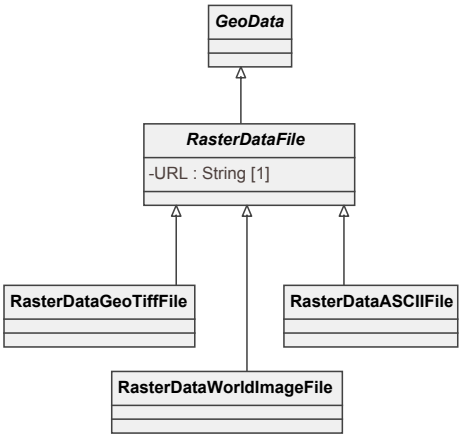

Fig. 3. A partial metamodel for modeling access to raster data (UML notation)

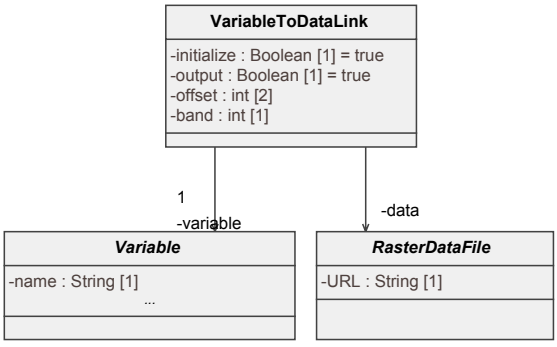

Fig. 4. A partial metamodel showing how the link between model variables and datasets is modeled (UML notation)

For the use of raster data in the simulation model, raster data is linked to a variable of a model. For this, a *VariableToDataLink* is specified, that holds a reference to a variable and the corresponding dataset (Fig. 4).

The *variable* in a *VariableToDataLink* references the *Variable* elements within the metamodels of the modeling formalism with which dynamic behavior is modeled (see Figs 6 and 8 below). The excerpt of the metamodel in Fig. 4 shows, that additional information can be given by means of attributes, e.g. if the data should be used to initialize the variable or if

the dataset is used for storing output[2]. This scheme is likewise applied on the formalism and the framework level, which is presented in the following.

4.2 Modeling Cellular Automata on the Formalism Level: The Hybrid Cellular Automaton Formalism (HCA)

For modeling on the formalism level, we extended the classical CA with the means to model continuous behavior of cells in order to be able to express greater variety of CA and to exploit the possibilities of modern simulation frameworks. This enhances expressiveness and may lead to a greater precision of models (for examples see Yacoubi et al 2003, Wainer and Giambiasi 2005).

A HCA consist of a set of structurally equivalent cells. Fig. 5 illustrates the possible behavior of single cells.

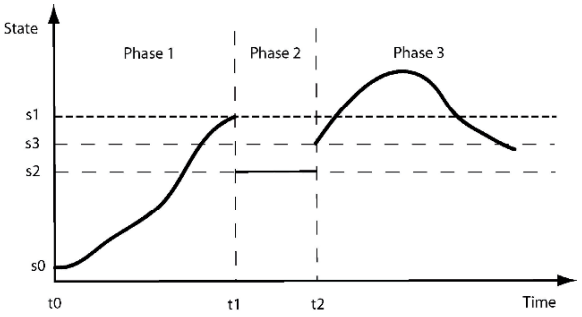

Fig. 5. The hybrid behavior of single cells of the HCA

Starting from state *s0* at time *t0*, the cell's state evolves continuously until time *t1*. At time *t1* a state event occurs as the state reaches the threshold *s1*. The state changes instantaneously to state *s2* and the cell transits into phase 2, where the state remains constant at *s2*. At time *t2* the state changes to state *s3* and evolves continuously in phase 3. The continuous behavior of cells is qualitatively different in the different phases, i.e. it is described by different ODEs. Note that the state change at time *t2* is triggered by an event outside the cell, for example a neighboring cell.

The metamodel in Fig. 6 shows the basic modeling elements of HCA. A cell's state is modeled by variables (*discreteStateVariable*, *continuousStateVariable*). All cells have an identical set of variables. For each phase, an

[2] For simplicity, it is assumed that a dataset has one band.

ODE (*rateOfChangeFunction*) can be specified. The selection of the appropriate ODE for a phase is modeled by a phase selection function (*phaseSelectionFunction*) which contains respective conditions (*phaseCondition*). Discrete state changes are specified with the *discreteTransitionFunction* that holds conditions (*condition*) and single expressions (*expression*) which define the state change. Discrete state changes depend on the state of the cell and the neighboring cells. State events (*stateEvent*) are modeled by means of conditions within *eventState*.

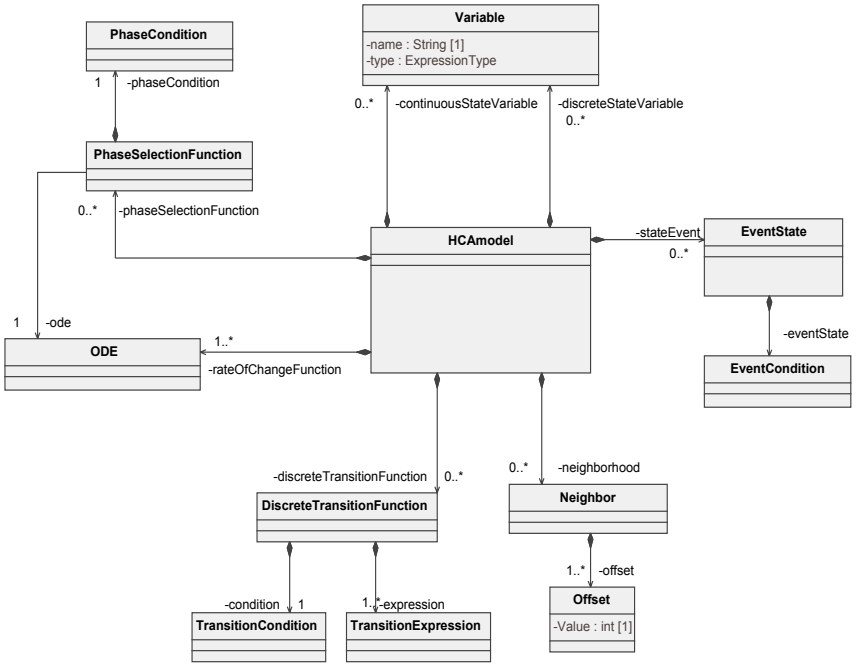

Fig. 6. A partial metamodel for the HCA (UML notation)

Fig. 6 informally illustrates the operation of an HCA model. Simulation time passes only during phases of continuous state change (*rateOfChangeFunction*). This evolution of state is eventually interrupted by an event. An event is followed by the application of the discrete transition function in each cell and the application of the *phaseSelectionFunction*. After this, the state of the cells evolves continuously until the next event or the end of the simulation.

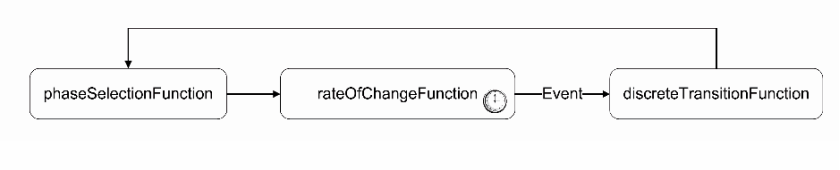

Fig. 7. The execution of HCA cell models

An event occurs when a cell's state variables have values as specified by a condition (*stateEvent*). The *neighborhood* of a cell is modeled by offsets from the origin of a cell. By definition, dependencies between the neighboring cells, as expressed in the *discreteTransitionFunction*, have to be reflected in the state events of the respective neighboring cells.

4.3 Modeling on the Framework Level: jDisco and Geotools

HCA models are transformed to framework specific models and finally to code (see Fig. 2). On the framework level a HCA is modeled by the means of the simulation framework jDisco and Geotools. Geotools is a library that provides spatial data processing functionality (Custer 2006).

JDisco is a framework that implements the process simulation world view (Helsgaun 2001). In process based modeling, a system is modeled as a collection of interacting processes that compete for common resources. In jDisco, a process can be a continuous (*ContinuousProcess*) or a discrete process (*DiscreteProcess*, see Fig. 8). Conceptually, a *ContinuousProcess* can change the state continuously between events according to ODE (*derivatives*).

Each process has a state which is defined by a set of state variables (*continuousState*, *discreteState*). Processes may hold references to other processes (*partnerProccess*) and other processes' variables (*referencedVariable*). Via references, values of variables can be read and set, which enables interaction and communication between processes. The behavior of a *DiscreteProcess* is modeled within a lifecycle function (*actions*) which is a sequence of actions (*ActionExpression*). A common action is a discrete change of the value of variables (*StateChangeExpression*).

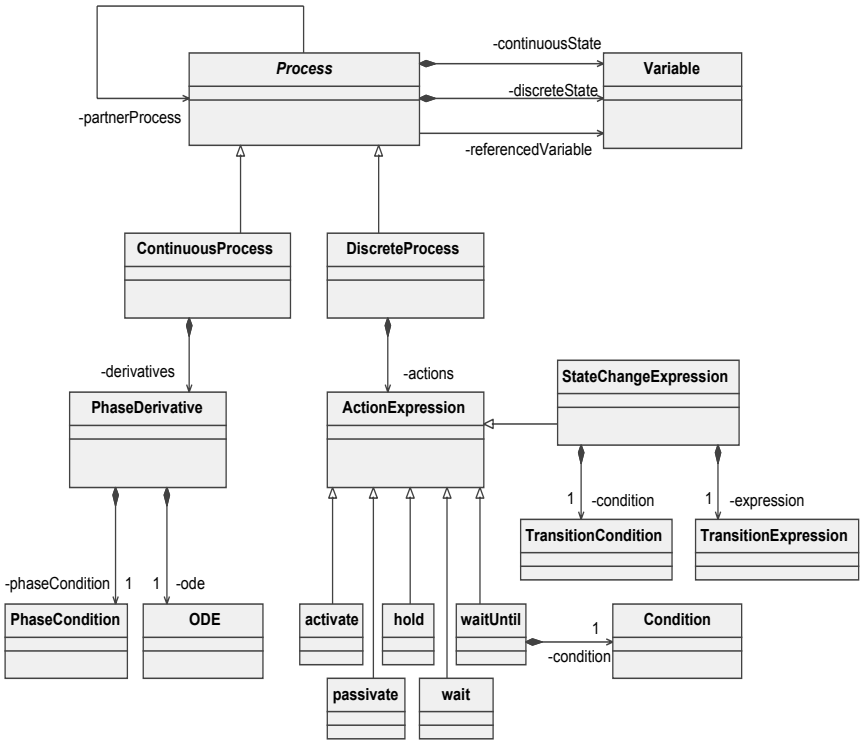

Fig. 8. A partial metamodel for jDisco (UML notation)

To enable interaction, processes must be synchronized. In jDisco, a *DiscreteProcess* can synchronize with a *ContinuousProcess* via the *waitUntil* – action. When performing *waitUntil*, the synchronizing *DiscreteProcess* suspends its lifecycle function until an event condition (*condition*) is true. Within condition, state events are specified. After an event, the lifecycle function (*actions*) continues performing actions.

Geotools is a Java language code library which provides the means to implement standards conformant geospatial applications. The library consists of various modules which can be combined according to the programmer's needs (Custer 2006). Geotools provides classes to access and process spatial data, which is needed during a simulation run. The code to realize raster access to a local raster data can be compiled from the information about the type and the location of the dataset (see the corresponding metamodels in Figs 3 and 4).

4.4 Transformation to the framework level

In the following, the main aspects of how HCA models can be expressed with the means of jDisco and Geotools are presented. Due to the common modeling elements, variables, discrete transition functions, ODE, conditions, and spatial data access can simply be mapped from HCA to the submodels in the framework's formalism (see Section 4.1). The conditions encoded in HCA's *phaseSelectionFunction* are simply mapped onto the respective *phaseCondition* within the continuous transition functions of the jDisco model. HCA-state-events are mapped onto conditions within the condition of a *waitUntil* action in jDisco.

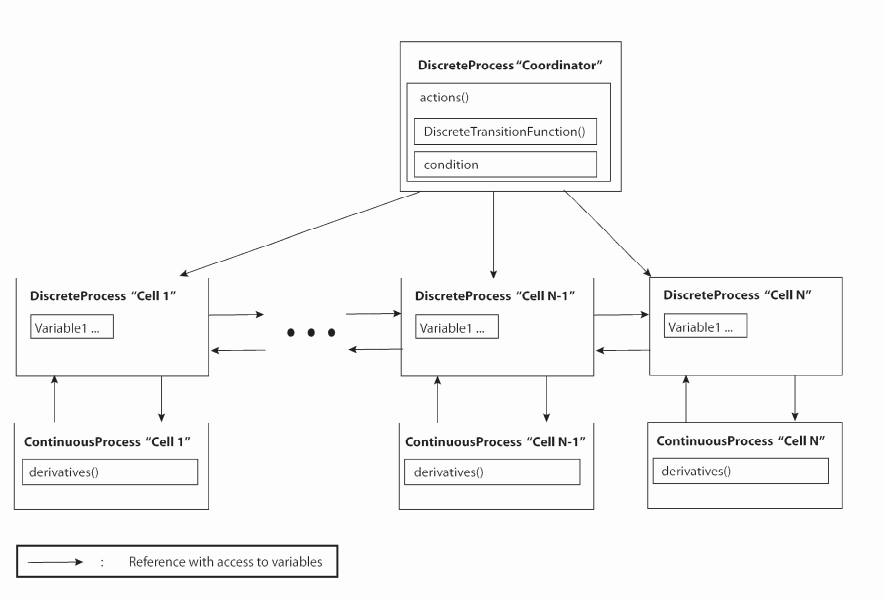

Fig. 9. Scheme of a jDisco model of a HCA

In a jDisco-model of HCA, one *DiscreteProcess* and one *ContinuousProcess* is created for each cell of the HCA. In addition, one *DiscreteProcess* is created that acts as a coordinator for all other cell processes. The "coordinator process" holds references to all discrete cell processes (Fig. 9). Each discrete cell process holds the continuous and discrete state variables of the respective cell and references to all neighboring discrete cell processes and their variables. Moreover, the transformation adds a variable to each discrete cell process that holds the position of the cell

within the model. The neighborhood is derived from the neighborhood definition of the HCA. Each continuous cell process holds the description of the cell's continuous behavior as ODE.

The discrete coordinator process "controls" the HCA. For this, a condition function holds the condition for the detection state events in all cell processes. Also the execution of the discrete transitions is realized by the coordinator process. This is possible because it holds references to all cells and their variables. The lifecycle function of the coordinator process is simple, as illustrated in Figure 10.

```
while simulation runs
   waitUntil(condition)
   for all cells
      DiscreteTransitionFunction()
   endfor
endwhile
```

Fig. 10. Pseudo-code of the action-function of the coordinator process

In jDisco the coupling of cells is realized through direct access to state variables. Each cell process holds references to the neighboring cell so that state values can be read directly. Since access to neighbors happens only at the execution of *discreteTransitionFunction* after synchronization (*waitUntil*), it is ensured that values of state variables have correct values at the time of access.

The transformation of the spatial data access description to the framework level and code is straightforward. A cell of a raster dataset represents the value of a cell process' variable. The reference to the corresponding raster dataset's cell is realized by mapping a cells' position to corresponding pixel's position, which is the same in the simplest case. Loading a raster file and initializing the value of variables is realized by the instantiation appropriate Geotools-classes, which are chosen by the code generator. The pseudo code in Fig. 11 illustrates this: appropriate classes are initialized at the beginning of the simulation holding a reference to the dataset (*file*) and classes that are provided by Geotools to process the raster dataset's data. Finally, with *DataBuffer*, the data can be directly accessed. The example shows how a variable representing a variable of an instance of a *DiscreteProcess* (*Cell5*) is initialized with a value from the dataset, by calling *db.getElemFloat(...)*.

```
…
simulationMainFunction {
                        …
      File file = getImageFile (rasterUrl);
      WorldImageReader reader = new WorldImageReader( file );
      RenderedImage image = coverage.getRenderedImage();
      Raster raster = image.getData();
      DataBuffer db = raster.getDataBuffer();
                        …
}

…

DiscreteProcess Cell5{
                        …
      float cellVariable = db.getElemFloat(toIndex (coordinate));
                        …
}
…
```

Fig. 11. Pseudo-code showing how access to a raster dataset can be read using Geotools. By using global variables, cells' variables can be initialized by direct access

The right pixel value from the dataset is chosen by using an index that is calculated from the position (*coordinate*) of the cell inside the model by a function that is generated for that (*toIndex()*). The position of the cell is assigned during the model generation process. In a similar way it is possible to integrate the creation and modification of raster datasets.

5 Concluding Remarks and Outlook

Today's simulation and spatial data processing technologies facilitate an MDE-based approach to spatiotemporal simulation modeling, as presented in this paper. A simulation model is initially defined on an abstract level by the modeler and consequently refined by automatic model transformation to finally obtain an executable model. This approach promises to overcome shortcomings of the architecture of today's tools that are related to the reusability of simulation models.

The implementation of models is not bound to a specific simulation framework, but it is required, that generic functionality is implemented by the executing framework. In particular, the framework must provide discrete-event simulation and algorithms to solve ODE. States and discrete state transitions are defined using the means of common general purpose

programming languages. The description of spatial data processing is based on concepts, as defined by widely accepted standards in the field.

In the presented example, the simulation and the spatial data processing framework use the same programming language, thus data exchange is straightforward. To integrate more different technologies (i.e. a C++-based simulation framework and a Java-based GIS library), it is necessary to generate "bridges" between technologies, for example wrappers.

The formal description of modeling levels by means of metamodels, not only provides the technical means for implementation, it also serves as the means to understand and communicate models from the viewpoint of the different users and developers of simulation software. Thus, the use of metamodeling supports transparency in model implementation. However, this requires a common understanding of the twofold meaning of metamodels by the users and developers on the different levels of abstraction.

It is an inherent characteristic of the presented approach, that through model transformation generic implementation patterns and recipes are applied to obtain executable software. On the one hand, this gives the possibility to apply best practices. On the other hand it is an obstacle for the inclusion of specific optimizations that may be possible for models with particular characteristics. However, prototypical implementations show that generic optimization is a key challenge, as model execution is demanding and efficient simulation is crucial to make this approach feasible for big cellular automata.

In further work the presented approach is to be elaborated, particularly focusing on the expressivity of HCA, optimization, and the role of spatial data processing.

Acknowledgements

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

References

Argent, R. (2004). "An overview of model integration for environmental applications--components, frameworks and semantics", Environmental Modelling & Software, Volume 19 (3), Pages 219-234.

Custer, A. (2006). "Geotools User's Manual, v.0.1", URL: http://www.geotools.fr/manual/geotools.xhtml

Evert, F. van, Holzworth, D., Muetzelfeldt, R., Rizzoli, A., and Villa, F. (2005). "Convergence in integrated modelling frameworks", in Zerger, A. and Argent, R.M. (eds) MODSIM 2005 International Congress on Modelling and Simulation. Modelling and Simulation Society of Australia and New Zealand, 745-750.

Fall, A. and Fall, J. (2001). "A domain-specific language for models of landscape dynamics", Ecological Modelling, 141(1-3):1–18.

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

Karssenberg, D. (2002). "Building dynamic spatial environmental models", PhD thesis, Utrecht University.

Muzy, A. Innocenti, E. Aiello, A. Santucci, J-F. Santoni P-A. and David R. C. Hill. (2005). "Modelling and simulation of ecological propagation processes: application to fire spread", Environmental Modelling & Software, 20 (7), 827-842.

Schmidt D. (2006). "Model-driven Engineering", Computer, 2/2006, 25-31.

SELES. (1999). "SELES v.1.0 Spatially Explicit Landscpe Event Simulator".

Vangheluwe, H. de Lara, J and Mosterman, P. (2002). "An introduction to multi-paradigm modeling and simulation", in Barros, F. and Giambiasi, N. (editors), AIS'2002 Conference, 9–20.

Wainer G. and Giambiasi N. (2005). "Cell-DEVS/GDEVS for Complex Continuous Systems", Simulation, 81, 137-151.

Yacoubi S. El, Jai A. El, Jacewicz P. and Pausas J. G. (2003). "LUCAS: an original tool for landscape modeling", Environmetal Modeling & Software, 18, 429-437.

Zeigler, B. Praehofer, H. and Kim T. (2000). Theory of Modeling and Simulation, Academic Press, San Diego, 2nd edition, 2000.