Jul 1st, 12:00 AM

# A Generic Framework for Multi-Disciplinary Environmental Modelling

Rolf Hennicker

Sebastian S. Bauer

Stephan Janisch

Matthias Ludwig

*International Environmental Modelling and Software Society (iEMSs)*
*2010 International Congress on Environmental Modelling and Software*
*Modelling for Environment's Sake, Fifth Biennial Meeting, Ottawa, Canada*
*David A. Swayne, Wanhong Yang, A. A. Voinov, A. Rizzoli, T. Filatova (Eds.)*
*http://www.iemss.org/iemss2010/index.php?n=Main.Proceedings*

# A Generic Framework for Multi-Disciplinary Environmental Modelling *

**Rolf Hennicker**[a], **Sebastian S. Bauer**[a], **Stephan Janisch**[a], **and Matthias Ludwig**[a]

[a]*Institut für Informatik, Ludwig-Maximilians-Universität München, Germany*
{hennicker,bauerse,janisch,mludwig}@pst.ifi.lmu.de

**Abstract:** We present a generic framework for computer-based environmental modelling which supports the coupling of simulation models from various sciences to perform integrative simulations. The framework is, in principle, applicable to any kind of model which simulates spatially distributed environmental processes with an arbitrary, but discrete, time scale. During an integrative simulation the framework coordinates the coupled models which run in parallel exchanging iteratively data via their interfaces. For proving the correctness of the coordination, formal methods of software development have been applied. The framework provides a developer interface for the implementation of natural science and socio-economic simulation models. For the latter, a framework specialisation has been developed which supports the modelling of agent-based social simulation models. The framework design was driven by the idea of enforcing common rules for integrative simulations, which must be respected by all simulation models, while leaving as much freedom as possible for discipline-specific implementations. Within the GLOWA-Danube project, the framework has been successfully applied to construct the distributed simulation system DANUBIA which integrates up to 15 simulation models from various disciplines, like meteorology, hydrology, plant physiology, glaciology, economy, agriculture, tourism, and environmental psychology. Actually, DANUBIA is already in use as a tool for decision makers to support the sustainable planning of the future of water resources in the Upper Danube basin.

***Keywords:*** modelling framework, coupled simulations, agent-based social simulations, software development.

## 1 INTRODUCTION

Climate change has an increasing impact on our natural and social environment which reveals more and more the need of interdisciplinary research to better understand the complex, mutually dependent processes occurring in nature and in socio-economic systems. In this article, we report on a generic framework for computer-based environmental modelling which supports the runtime coupling of various simulation models from natural science and socio-economic disciplines. The framework is generic in the sense, that it is, in principle, applicable to any kind of model which simulates spatially distributed environmental processes on an arbitrary, but discrete time scale. During an integrative simulation for some simulation period, the framework coordinates the coupled models which run in parallel exchanging iteratively data via their interfaces. For the development of the modelling framework best practices of software engineering have been applied like abstraction, separation of concerns and formal methods based on precise mathematical

*This research has been partially supported by the GLOWA-Danube project 01LW0602A2 sponsored by the German Federal Ministry of Education and Research.

notations. For instance, the framework provides abstractions of simulation models and thus facilitates the development and integration of concrete simulation models of particular disciplines. Separation of concerns involves the aspects of information exchange between simulation models, consistent modelling of simulation space, coordination of concurrently running simulation models, and agent-based social simulations.

Technically, for the development of the simulation framework the Unified Modeling Language UML (cf., e.g., Rumbaugh et al. [2005]) has been used in the requirements and in the design phase, while the framework implementation is programmed in Java making use of Java's Remote Method Invocation interface (RMI) for communication between distributed components. Formal methods have been applied to specify the requirements for the general life cycle each simulation model must obey and for specifying and proving the correctness of the coordination of distributed simulation models, for which the framework is responsible. For this purpose, the process algebra Finite State Processes FSP (cf. Magee and Kramer [2006]) has been used. For the specification of requirements concerning correct simulation configurations, invariants have been stated and formalised in terms of the Object Constraint Language OCL; cf. Warmer and Kleppe [2003]. The framework design follows a component-oriented approach which allows to plug in arbitrary simulation models as long as the common requirements concerning, e.g., the simulation space and the life cycle of a model are satisfied. Thus the integration of different simulation models into a coupled simulation is considerably facilitated and (most) errors occurring in simulation models caused by not meeting common requirements can be detected already at compilation time.

The framework has been developed and successfully applied to construct the integrative simulation system DANUBIA within the interdisciplinary research project GLOWA-Danube[1] (Ludwig et al. [2003]), which is part of the German national initiative GLOWA (Global Change in the Hydrological Cycle) running from 2001 to 2010. Within GLOWA-Danube, a group of researchers from various natural science and socio-economic disciplines have teamed up to investigate the impact of climate change on the water cycle within the Upper Danube watershed and to support the development and evaluation of regional adaptation strategies. Actually 15 simulation models have been developed by the research groups of GLOWA-Danube, such that various simulation configurations can be built and run by our modelling framework within the DANUBIA system.

A number of other frameworks supporting integrated environmental modelling emerged since the GLOWA-Danube project started in 2001. In the field of integrated water resource management there are, e.g., the Object Modelling System OMS (cf. Kralisch et al. [2005]), ModCom (cf. Hillyer et al. [2003]), The Invisible Modelling Environment TIME (cf. Rahman et al. [2003]), and the Open Modelling Interface OpenMI (cf. Gregersen et al. [2007]). While TIME is a platform for the development of standalone modelling tools, OMS, ModCom, and OpenMI are frameworks which support the independent development of models and allow for execution of coupled simulations. In particular, OpenMI is designed to extend existing standalone models by standard interfaces for coupling. However, to our knowledge, none of these frameworks is tailored towards distributed and parallel execution of coupled simulation models which is a main characteristics of our approach.

The outline of this paper is as follows. In Sect. 2, we identify common requirements for integrative environmental simulations. Then, in Sect. 3, we summarise the design of our framework in accordance with the given requirements. Sect. 4 introduces briefly the integrative simulation system DANUBIA obtained by framework instantiation within the context of the GLOWA-Danube project. Finally, in Sect. 6, we summarise our main results.

[1] www.glowa-danube.de

## 2 REQUIREMENTS FOR INTEGRATIVE ENVIRONMENTAL SIMULATIONS

We consider in the following as a *simulation model* a computer program that simulates an environmental process over a certain time span, called the *simulation time*, with regard to a certain geographical area of the environment, called the *simulation space*. In an *integrative simulation system* several simulation models are coupled in order to analyse dependencies and feedbacks of the simulated processes. It is obvious that, besides the technique of coupling itself, the consistent treatment of simulation time and simulation space is crucial for the integration of different simulation models. Moreover, for a comprehensive environmental simulation not only processes occurring in nature but also processes reflecting human behaviour must be taken into account. Therefore we have identified four major requirements. The framework should support

1. data exchange between simulation models at runtime,
2. coordination of simulation models according to simulation time,
3. consistent treatment of simulation space, and
4. agent-based social simulations.

In the following we elaborate more on each of the four requirements.

### 2.1 Data Exchange between Simulation Models

The coupling of simulation models is based on *interfaces*. Interfaces for data exchange specify data queries. We distinguish between *provided* interfaces specifying queries for data that is provided by a simulation model, and *required* interfaces specifying queries for data that is needed by a simulation model for its own computation. The general requirements concerning data exchange are modelled in the UML class diagram in Fig. 1. It says that a simulation may involve arbitrarily many models, which play the role of the `participatingModels` for the simulation, and that a model may have arbitrarily many interfaces, playing the role of `provided` or `required` interfaces. For the reader, who is not familiar with the UML notation, a brief introduction of the modelling elements used hereafter is given in Sect. 7.

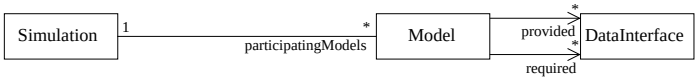


Figure 1: Requirements model for data exchange

A concrete example of a provided and required interface is given later on when we illustrate the framework instantiation in Fig. 9. The following invariant expresses a consistency requirement for data exchange which must be satisfied for any integrative simulation.

*Invariant for data exchange*

- In an integrative simulation, for each required interface of each participating model there exists exactly one participating model which provides that interface.

This invariant can be formalised in terms of the following OCL-expression:

```
context Simulation inv:
    self.participatingModels.forAll(m |
      m.required->forAll(r |
        self.participatingModels->one(n |
          n.provided->includes(r))))
```

All invariants described in the sequel of this paper can be formalised in a similar way in terms of OCL invariants (cf. Warmer and Kleppe [2003]) which will, however, be omitted here.

### 2.2 Coordination of Simulation Time

An important characteristics of our problem domain is the concurrent execution of different simulation models which iteratively exchange information at runtime via their interfaces. In order to guarantee the consistency of data exchange during a simulation run, the single simulation models must be appropriately coordinated with respect to the progressing simulation time. The *correct* coordination is a non-trivial task since, in general, simulation models have different, individual time steps determining the model time between two consecutive computations. Model time steps depend, of course, on the simulated processes which typically range from minutes or hours, like in natural sciences, to months, like in social sciences. Hence a precise, unambiguous specification of the coordination problem is mandatory. We first describe the general life cycle which a simulation model must follow:

- *provide* initial data at the model's provided interfaces
- while not at simulation end
  - *get data* from the model's required interfaces
  - *compute* new data for the next time step
  - *provide* newly computed data at the model's provided interfaces

For the formalisation of a model's life cycle we use the process algebra Finite State Processes FSP; cf. Magee and Kramer [2006]. The following FSP process `MODEL` specifies the general behaviour of a simulation model. In order to be generally applicable the process is parameterised with respect to the model's time step. The sequence of actions in line 5, `getData[t] -> compute[t+Step] -> provide[t+Step]`, is iteratively performed with increasing time `t` and thus formalises the iteration in the informal description of a model's life cycle given above. Note that the computation of new data for time `t+Step` relies on data obtained for time `t`. This time difference avoids deadlocks of concurrently running models (in the case of feedback loops) but it may also lead to imprecisions whose relevance must be analysed in concrete cases and, if necessary, can be resolved by using smaller time steps.

```
1 range SimTime = SimStart..SimEnd
2 MODEL(Step) = (start -> provide[SimStart] -> M[SimStart]),
3 M[t:SimTime] =
4     if (t+Step <= SimEnd)
5     then (getData[t] -> compute[t+Step] -> provide[t+Step] -> M[t+Step])
6     else (finish -> STOP).
```

When several simulation models are executed in parallel, it is essential that only valid data is exchanged, i.e. data that fits to the local model time of the participating models. To specify this

requirement we consider only two simulation models at a time, one, say $U$, acting as a user of data, and the other one, say $P$, acting as a data provider. From the user's point of view we obtain the coordination condition (U), from the provider's point of view the coordination condition (P).

(U) $U$ gets data expected to be valid at time $t_U$ only if the following holds:
The next data that $P$ provides is valid at time $t_P$ with $t_U < t_P$.

(P) $P$ provides data valid at time $t_P$ only if the following holds:
The next data that $U$ gets is expected to be valid at time $t_U$ with $t_U \geq t_P$.

Condition (U) ensures that the user does not get obsolete data while condition (P) guarantees that data, available at the provider's interface, will not be overwritten if it is not yet considered by the user model. If one can show that all (pairwise) combinations of all models participating in an integrative simulation considered in both roles, as user and as provider of data, satisfy the two coordination requirements, then the whole integrative simulation is coordinated correctly. We have again used FSP to formalise the coordination conditions in terms of so-called property processes. Then, in a next step, we have constructed an explicit coordination process with FSP and we have verified by model checking techniques that the coordination process provides a solution for the coordination problem of integrative simulations (which later on can be implemented in Java). For more details see Hennicker and Ludwig [2005].

### 2.3 Modelling of Simulation Space

In an integrative environmental simulation the consistent treatment of the underlying simulation space is crucial. It is obvious that in spatially distributed simulations one needs geographical units, which in the following will be called *proxels*. The term proxel (cf. Tenhunen and Kabat [1999]) stems from *process pixel* and suggests that a proxel does not only model a structural element of the simulation space, but it shows also dynamic behaviour by simulating the environmental processes on this particular geographical unit. The entire simulation area is then modelled by a set of (non-overlapping) proxels.

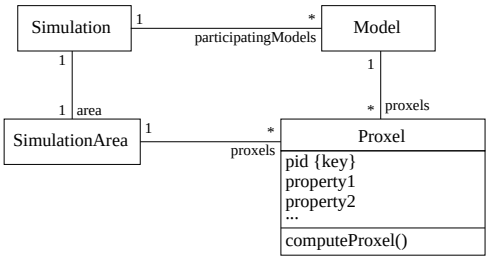


Figure 2: Requirements model for simulation space

The spatial requirements of an integrative simulation are described by the UML class diagram in Fig. 2. It says that a simulation concerns always exactly one simulation area which, in turn, consists of a set of proxels. The class `Proxel` requires that each proxel has a unique identifier `pid` and an operation `computeProxel()` to compute the next state of a proxel in each time step. Moreover, each proxel can have a number of properties which must be common to all simulation models (like, e.g., geographical coordinates, elevation, etc.). On the other hand, each simulation model has a set of proxels, on which it operates. The following invariant requires that the models

participating in an integrative simulation agree on the set of proxels determined by the area of the simulation.

*Invariant for the simulation space*

- In an integrative simulation, all participating models operate on proxels which belong to the simulation area of the simulation.

Besides the basic properties, a proxel may store domain-specific properties as illustrated for groundwater proxels in Fig. 9 later on.

### 2.4 Support for Agent-Based Social Simulation

Another fundamental aspect of integrated environmental simulation is the explicit support of societal issues concerning human behaviour. A common modelling approach within socio-economic disciplines is *agent-based social simulation* which uses concepts from the field of multiagent systems; cf. Weiss [1999]. While models for social simulation integrate into coupled environmental simulations like any other simulation model by interface-based data exchange, their local computations are conceptually different from natural science models. At the core of such simulation models are *actors* which model deciding entities such as individuals (e.g. farmers, tourists, households, etc.) or organisations (e.g. water suppliers, companies). The class diagram in Fig. 3 shows the fundamental concepts and relationships for agent-based social simulations.

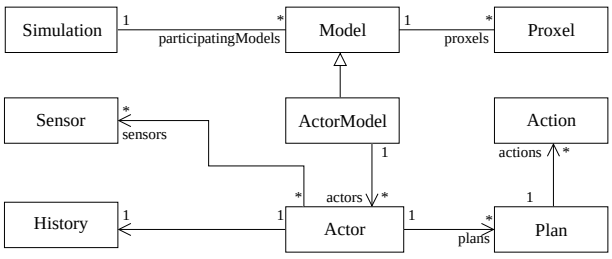


Figure 3: Requirements model for agent-based social simulation

An actor model is a specialisation of a simulation model that refers to an arbitrary number of actors. An actor administrates a set of plans, each consisting of a number of actions. Moreover, an actor can be linked to a number of sensors to observe the environment and it may use its history to retrieve information on the success of previously executed plans. Concerning the dynamic behaviour, an actor model is supposed to follow the same life cycle as known from ordinary simulation models; cf. Sect. 2.2. However, the computation step of an actor model is specialised in the sense, that each actor referred by the model must perform (possibly concurrently) the following steps:

- query sensors for the state of the environment
- decide which plan should be executed by performing the following steps:
  - select relevant plan options

  - filter the options to decide which plan should be implemented
- implement the selected plan by executing its associated actions
- inform the actor model on the effects of the plan implementation

Finally, the actor model must update its provided interfaces accordingly.

Our conception to support agent-based social simulation follows the principles of reactive agent architectures (cf., e.g., Wooldridge [1999]) characterised by simple behaviours, which directly map perceived states to actions to be executed, together with the parallel execution of these behaviours. How the agent-based simulation principles of our modelling framework work in a concrete example of (inter)acting water suppliers and households is described in Barthel et al. [2010].

## 3 SIMULATION FRAMEWORK

The requirements and concepts described in the previous section are realised in a component-based simulation framework. The framework defines a general data interface and *generic components* that implement common structure and behaviour, thus imposing general rules which must be respected by concrete simulation models when it comes to framework instantiation. D'Souza and Wills [1999] use the notion of *plug-point*, provided by a generic component, and *plug-in*, provided by some extension which completes the generic component to an executable implementation. In this section we focus on the framework and plug-points while, in Sect. 4, we show how the framework can be instantiated by simulation models with plug-ins. The framework itself is split into two layers, the framework core and the developer interface. To explain the principle ideas behind these layers we consider the framework excerpt shown in Fig. 4 (without taking into account components yet).

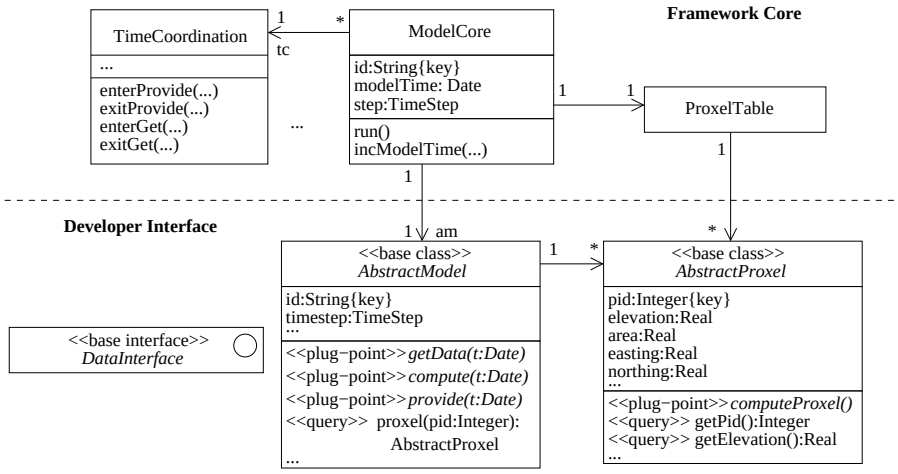


Figure 4: Two-layered framework architecture

The framework core implements all features that can be handled by the framework itself like, e.g., the time coordination, the common properties of a simulation model (`ModelCore`), and the management of the spatial distribution (`ProxelTable`). Concerning time coordination, there exists at runtime exactly one instance of the class `TimeCoordination` which is a monitor object that is called by each model core instance before data is fetched from or provided to a data exchange

interface. Each model core instance itself will be linked at runtime to exactly one concrete simulation model which must implement the abstract operations, given by the plug-points `getData`, `compute` and `provide`, of the class `AbstractModel` of the developer interface. The UML sequence diagram in Fig. 5 illustrates the sequence of interactions implemented in the `run` method by taking into account the life cycle of simulation models as described in Sect. 2.2.

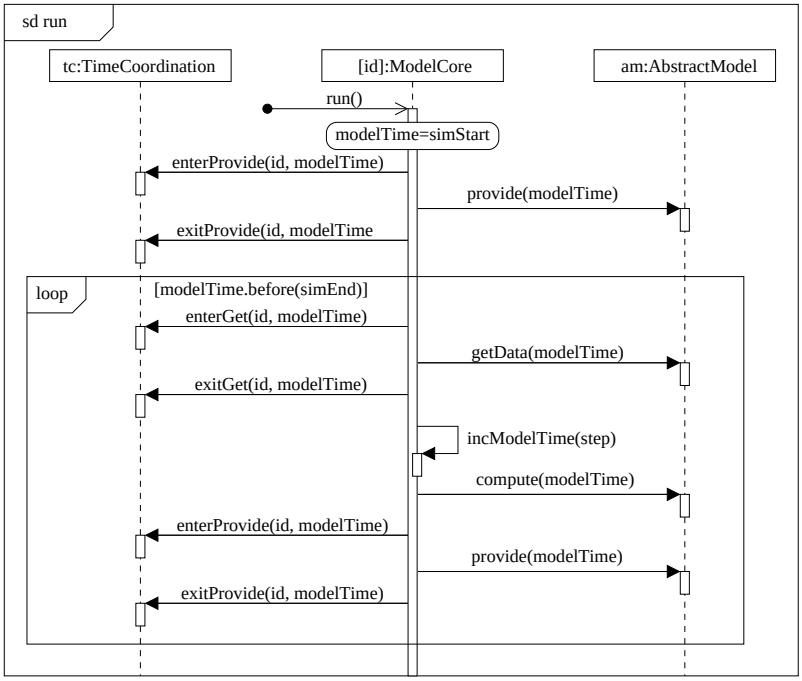


Figure 5: Interactions of the `run` method

Let us now consider how the two layers explained above fit into the component architecture of the simulation framework shown in Fig. 6.

The component `SimulationAdmin` is the central component for managing integrative simulations and hence provides an interface for a (graphical) user interface. This interface offers operations for starting, observing and aborting a simulation. Before starting a simulation the component `SimulationAdmin` checks the simulation configuration for consistency, i.e. if the invariant concerning interfaces as stated in Sec. 2.1 is fulfilled. At the beginning of a simulation the component `ModelLinker` establishes the links between simulation models over their interfaces for data exchange, including distribution over a network. The component `TimeCoordination` is responsible for the correct time coordination of the single models during a simulation run. The component `BaseData` reads and stores initialisation data for the basic properties of the simulation area for all simulation models. `Model` is a generic component which contains framework core classes as well as the classes constituting the programming interface for model developers as discussed above.

Concerning agent-based social simulations a particular so called DEEPACTOR framework has been developed providing a common architecture for socio-economic simulation models. The design of the DEEPACTOR framework follows again our general pattern which distinguishes core classes and (abstract) base classes for model developers, exemplified by the classes `ActorModelCore` and `AbstractActorModel` in Fig. 7. The remaining components of the DEEPACTOR framework are implemented similarly along the requirements described in Sect. 2.4 and we omit further details here.

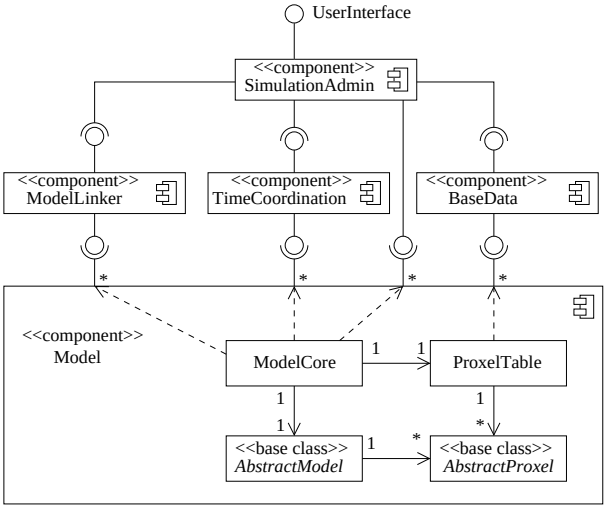


Figure 6: Component-based design of the simulation framework

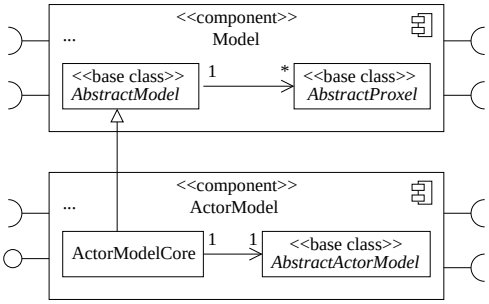


Figure 7: Integration of the DEEPACTOR framework

The integration of the DEEPACTOR framework into the general simulation framework is achieved by extension of the developer interface of the overall framework. As illustrated by the excerpt in Fig. 7, the core part of the DEEPACTOR framework specialises (solely) the developer interface of the general framework. Concrete actor models then use the developer interface of the DEEPACTOR framework and, for basic functionality, the developer interface of the general framework such as `AbstractProxel`. The manual Janisch [2007] provides detailed descriptions of the modelling features available in the DEEPACTOR framework and Barthel et al. [2010] shows its application for the development of DEEPACTOR models in the context of GLOWA-Danube.

We have used UML 2.0 for the complete framework design, in particular for the detailed documentation of the developer interfaces. The framework is implemented in Java SE 6 and contains approximately 25.000 lines of code.

## 4 APPLICATION OF THE FRAMEWORK: THE DANUBIA SYSTEM

Within the GLOWA-Danube project (Ludwig et al. [2003]) our simulation framework has been applied to construct the integrative simulation system DANUBIA which integrates up to 15 sim-

ulation models for natural processes (like hydrology, plant physiology, groundwater, glaciology etc.) as well as socio-economic models. The latter have been developed to model the behaviour of the involved actors in the areas of agriculture, economy, water supply, private households, and tourism based on the structure of societies and their interests. The purpose of DANUBIA is to serve as a tool for decision makers from policy, economy, and administration for the sustainable planning of water resources in the Upper Danube basin under global change conditions. DANUBIA is a distributed simulation system – the simulation models are executed in parallel on a computer cluster periodically exchanging data during a simulation run. DANUBIA was validated with comprehensive data sets of the years 1970 to 2005. It is actually in use to run and evaluate coupled simulations which are driven by climatic as well as societal scenarios for the next 50 years. Concerning performance, integrative simulation runs with DANUBIA for typical configurations over a 50 year period actually take (at least) 36 hours computing time. An overview of the system architecture of DANUBIA is given in Fig. 8.

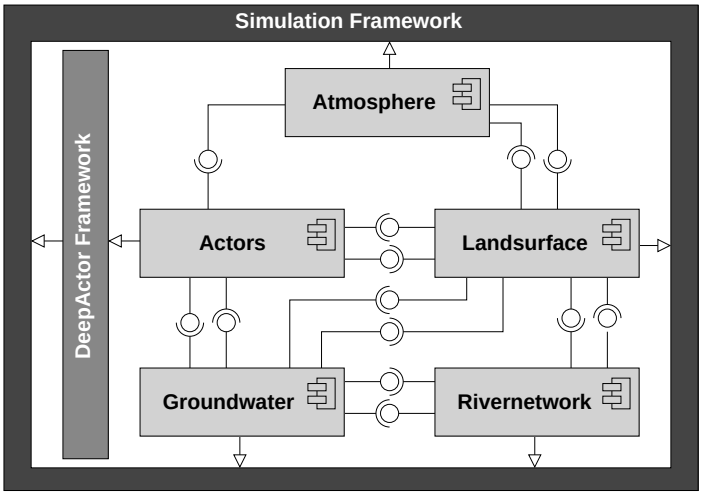


Figure 8: System architecture of DANUBIA

How a concrete simulation model is integrated in the framework is shown in Fig. 9. The upper layer of Fig. 9 depicts the part of the developer interface known from Fig. 4, and the lower layer shows parts of a sample `Groundwater` model. One can see that all model classes (and interfaces) of the groundwater model extend the base classes (the base interface `DataInterface` resp.) of the developer interface by certain domain-specific properties, like the proxel attributes `gwWithdrawal, gwLevel` etc., and by providing implementations for the plug-in operations like, e.g., `compute` and `computeProxel`. Thereby the framework's core functionality concerning runtime coordination, management tasks and the like is completely hidden for model developers.

Fig. 9 shows that the `Groundwater` model uses the interface `WatersupplyToGroundwater` for importing (getting) required data and it realises the interface `GroundwaterToWatersupply` by implementing the interface operations, e.g., `getGroundwaterLevel`. In each time step, when the (plug-in) operation `getData` is executed, the model imports a spatially resolved table of required data and distributes it on the corresponding attributes, e.g., `gwWithdrawal`, of its local proxels. Then a computation step is performed proxelwise and finally, when the (plug-in) operation `provide` is executed, the newly computed data on the corresponding attributes of the Groundwater proxels, e.g., `gwLevel`, are collected from the single proxels (which can be accessed using the

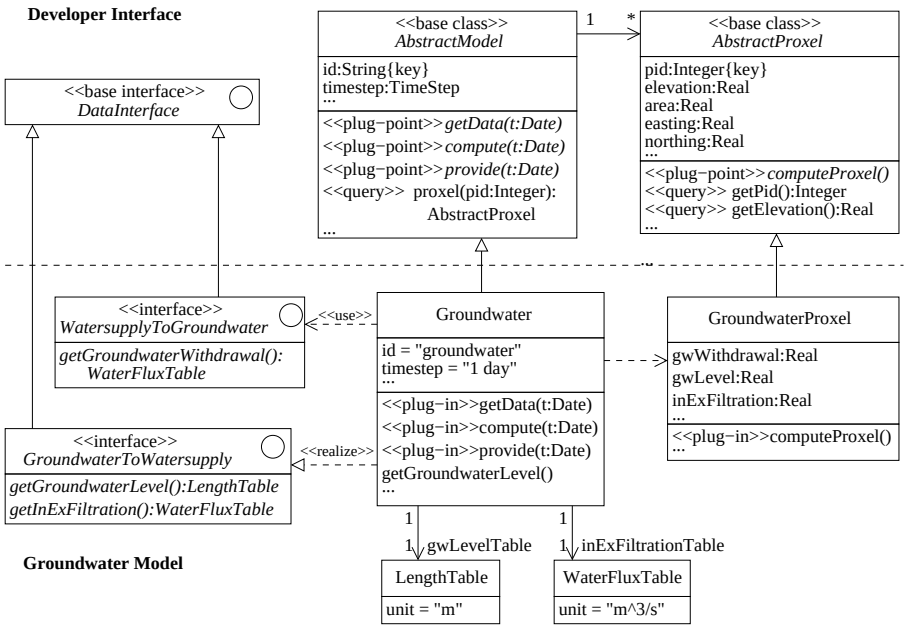


Figure 9: Framework instantiation

query `proxel` inherited from the model's superclass `AbstractModel`) and stored, again spatially resolved, in a corresponding *export table*. For example, the attribute values of `gwLevel` of each proxel are stored in the export table `gwLevelTable` of type `LengthTable`. When the operation `getGroundwaterLevel` is called by another model, the corresponding table is returned.

To ensure that models agree on the units of exchanged data each predefined data table type specifies a common unit which applies to all data in the table. Of course, a model can internally operate with different units, but then the values must be appropriately converted before data exchange. A further provision for data consistency is achieved by the possibility to enhance data interfaces by specifications which define the range of expected and guaranteed values when data is imported and provided via an interface. The model developer is then responsible to comply with the specifications during runtime.

While the framework is primarily intended for the development of new simulation models, legacy models can yet be integrated into the framework as long as their computation steps are controllable from the outside. In this case the legacy model is surrounded by a wrapper which must implement the (plug-in) operations like any other model. The concrete computation steps of the legacy model are then initiated within the `compute` operation of the wrapper, e.g. by using the Java Native Interface.

A concrete application of the DEEPACTOR framework for the simulation of water-related issues in the Upper Danube basin is reported in Ernst et al. [2007], describing a simulation model for the water consumption of households, and in Barthel et al. [2008b] who developed a model for water supply companies as an important link between socio-economic and natural science simulation models. An agent-based simulation model for the diffusion of environmental innovations is described in Schwarz and Ernst [2009]. From the natural science point of view, Barthel et al. [2008a] focus on the groundwater model within DANUBIA, which is based on MODFLOW and hence an example for the integration of legacy models, and, in Lenz-Wiedemann et al. [2010] the crop growth model within the landsurface component of DANUBIA is described.

## 5 PERFORMING AN INTEGRATIVE SIMULATION

Fig. 10 summarises the essential steps which are necessary to perform an integrative simulation with the DANUBIA system. We assume that the available simulation models – in terms of source code and executables, as well as metadata, initialisation data and other data which is required to run the model – are already checked-in at the project repository.

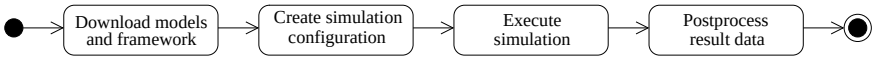


Figure 10: Workflow of an integrative simulation

At first the available models and the framework must be downloaded and installed on either a computer cluster, a local area network, or on a single computer, depending on the simulation configuration to be executed. In the next step a simulation configuration is created which defines the simulation time, the simulation space, the data resources for a consistent initialisation, some project specific parameter settings and, most importantly, the set of models to participate in the simulation. To assist the user in defining a simulation configuration the framework provides a graphical user interface which allows for setting the necessary global parameters and for selecting the participating models from a list of available models. In particular, each participating model must be allocated to a computer resource on which it will be executed.

Before starting a simulation, its configuration is checked for reasonableness of parameter settings and compatibility of interfaces for data exchange. When the simulation is started the simulation environment is initialised on each computer and the models are distributed and concurrently started on their allocated computer resources. Data interfaces are linked automatically by matching the names of corresponding import and export interfaces. During the simulation run, the progress of the simulation and of each participating model can be observed via the graphical user interface. After the simulation has finished, result data is available in a binary format which is optimised with respect to the required hard disc space. From this point, the data can be visualised and post-processed with appropriate tools which are also available in the project repository.

## 6 CONCLUSIONS

In this paper we have described requirements, design and implementation of a generic framework for environmental modelling. The framework supports the development and the coupling of simulation models from various disciplines to perform integrative simulations.

Applying the framework paradigm to integrative environmental modelling provides several advantages: common routines and services like network support, time coordination or space initialisation can be separated from the scientific code of the simulation models. The model developer only has to implement distinguished extension points of the framework, thus one can be sure that generally valid rules are respected by each model. The simulation framework is generic in the sense that it is independent from actual simulation models. In fact, it scales up to an arbitrary number of simulation models and can be applied to any simulation area as long as the requirements of the framework are satisfied.

The framework has been successfully instantiated in the interdisciplinary research project GLOWA-Danube by the implementation of the distributed simulation and decision support system DANUBIA. With DANUBIA a number of scenarios concerning changes of climate and society

have been simulated which shall show their impacts on the future of water resources thus giving hints for sustainable planning.

## REFERENCES

Barthel, R., J. Braun, and W. Mauser. Integrated modelling of global change effects on the water cycle in the upper Danube catchment (Germany) - the groundwater management perspective. In Carillo Rivera, J. J. and M. A. Ortega Guerrero, editors, *Groundwater flow understanding: from local to regional scale. Selected Papers on Hydrogeology SP12*, pages 47–72. Taylor & Francis, 2008a.

Barthel, R., S. Janisch, D. Nickel, A. Trifkovic, and T. Hörhan. Using the multiactor-approach in GLOWA-DANUBE to simulate decisions for the water supply sector under conditions of global climate change. *Water Resour Manag*, 24(2):239–275, Jan 2010.

Barthel, R., S. Janisch, N. Schwarz, A. Trifkovic, D. Nickel, C. Schulz, and W. Mauser. An integrated modelling framework for simulating regional-scale actor responses to global change in the water domain. *Environ Model Software*, 23(9):1095–1121, September 2008b.

D'Souza, D. and A. Wills. *Objects, Components and Frameworks with UML – The Catalysis Approach*. Addison-Wesley, Reading, Massachusetts, 1999.

Ernst, A., C. Schulz, N. Schwarz, and S. Janisch. Modeling of water use decisions in a large, spatially explicit coupled simulation system. In *Social Simulation: Technologies, Advances and New Discoveries*. Idea Group Inc., 2007.

Gregersen, J. B., P. J. A. Gijsbers, and S. J. P. Westen. OpenMI: Open modelling interface. *Journal of Hydroinformatics*, 9(3):175–191, 2007.

Hennicker, R. and M. Ludwig. Property-driven development of a coordination model for distributed simulations. In *Int. Conf. Formal Methods for Open Object-Based Distributed Systems*, volume 3535 of *LNCS*, pages 290–305. Springer, 2005.

Hillyer, C., J. Bolte, F. van Evert, and A. Lamaker. The ModCom modular simulation system. *European Journal of Agronomy*, 18(3):333–343(11), January 2003.

Janisch, S. DeepActor Framework Reference Manual. Draft available at `http://www.pst.ifi.lmu.de/~janisch/pub/refmanual.pdf`, 2007.

Kralisch, S., P. Krause, and O.David. Using the Object Modeling System for hydrological model development and application. *Advances in Geosciences*, 4:75–81, 2005.

Lenz-Wiedemann, V., C. Klar, and K. Schneider. Development and test of a crop growth model for application within a global change decision support system. *Ecol Model*, 221(2):314–329, 2010.

Ludwig, R., W. Mauser, S. Niemeyer, A. Colgan, R. Stolz, H. Escher-Vetter, M. Kuhn, M. Reichstein, J. Tenhunen, A. Kraus, M. Ludwig, M. Barth, and R. Hennicker. Web-based modeling of water, energy and matter fluxes to support decision making in mesoscale catchments – the integrative perspective of glowa-danube. *Physics and Chemistry of the Earth*, 28:621–634, 2003.

Magee, J. and J. Kramer. *Concurrency : state models & Java programming*. Wiley, Chichester, 2006.

Rahman, J. M., S. P. Seaton, J.-M. Perraud, H. Hotham, D. I. Verrelli, and J. R. Coleman. It's TIME for a New Environmental Modelling Framework. In *Proceedings of MODSIM 2003 International Congress on Modelling and Simulation*, volume 4, Townsville, Australia, July 2003. Modelling and Simulation Society of Australia and New Zealand Inc.

Rumbaugh, J., I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual, 2nd edition*. Pearson Education, Inc, 2005.

Schwarz, N. and A. Ernst. Agent-based modeling of the diffusion of environmental innovations – an empirical approach. *Technol Forecast Soc*, 76(4):497–511, 2009.

Tenhunen, J. D. and P. Kabat, editors. *Integrating Hydrology, Ecosystem Dynamics, and Biogeochemistry in Complex Landscapes*. Wiley, Chichester, 1999.

Warmer, J. and A. Kleppe. *The Object Constraint Language – Second Edition*. Addison-Wesley, 2003.

Weiss, G., editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, 1999.

Wooldridge, M. *Intelligent Agents*, chapter 1, pages 27–79. In Weiss [1999], 1999.

## 7 APPENDIX: UML NOTATIONS

We give a short summary of the graphical notations of the unified modelling language UML used in this paper. For more details see Rumbaugh et al. [2005].

A *class diagram* is a graphical description of system entities, in UML called classifier, and the relationships among them. Classifier and relationships are further distinguished and provide a number of concrete modelling features. Fig. 11 shows an overview of those modelling elements which are used in this paper.

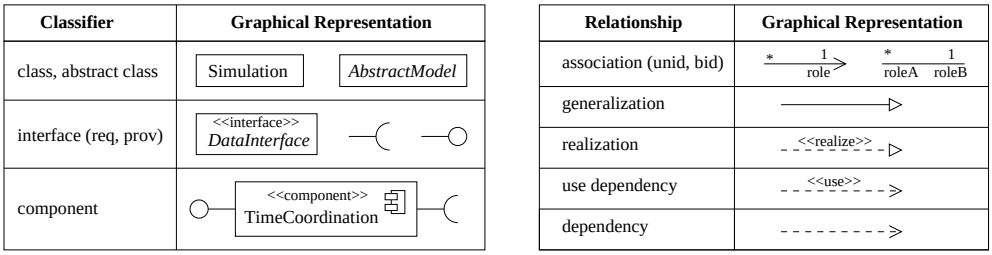


Figure 11: Relevant UML2 classifier and relationships

A *class* models a concept of some application domain. Objects are instances of classes. *Abstract classes* represent (abstract) concepts which can not be instantiated. They are used to describe base classes which are generalisations of more concrete classes. In our framework description we use abstract classes to model generic concepts that are reified within concrete simulation models. *Interfaces* gather operation signatures required (req) or provided (prov) by a class or component. *Providing* an interface means to provide an implementation of the particular operations, *requiring* an interface means to use the operations as described in the interface and implemented by a third party unknown (decoupled) from the user. This kind of decoupling is crucial for component-based system design. A *component* is a classifier which can encapsulate (sub-)classifier and which, usually, has required and/or provided interfaces. Therefore, a component defines a part of a system which may be implemented and used only in accordance with its interfaces. Implementation proceeds without knowledge of concrete usage, and usage proceeds without knowledge of the concrete implementation.

The most fundamental kind of relationship in UML is *association* used to specify possible connections between classifier instances. To emphasise the role one classifier plays for an associated classifier various features may be used. Among them is the distinction between unidirectional (unid) and bidirectional associations (bid), role names and multiplicities. *Directionality* expresses the possibility to navigate along the association in order to retrieve information about the connected entity. *Role names* provide an identifier to refer to that entity, and *multiplicities* are used to describe constraints on the number of associated classifier instances. A *generalization* relates a general concept, represented by a "parent" classifier, to a more specific "child" element which adds (reifies) information given by the parent classifier. It is graphically represented by an arrow from the child to the parent classifier. The relationship *realization* expresses that a classifier, usually a class or a component, implements a (provided) interface. A *dependency* relationship expresses that one model element makes use of some feature of another model element and hence is dependent on that.