

Principles of Integrative Environmental Simulations *

Rolf Hennicker, Sebastian S. Bauer, Stephan Janisch,
and Matthias Ludwig

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

Abstract

We report on the role of informatics in the interdisciplinary research project GLOWA-Danube which focuses on the comprehensive analysis of the future of water resources in the Upper Danube. In this project the distributed integrative simulation system DANUBIA has been developed which couples simulation models for different natural sciences as well as socio-economic processes. During the development the role of the informatics group turned out to be of particular importance. The methods and techniques of software engineering for abstraction and separation of concerns were beneficial in both technical and scientific aspects: on the one hand, the implementation and integration of the simulation models of different research groups into the system was facilitated, and on the other hand their scientific development was supported by a common understanding of general concepts of integrated and coupled environmental simulation systems.

1 Introduction and Overview

The increasing impact of climate change on our natural and social environment 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. Of particular importance are water-related processes which have an eminent impact on the global change of the hydrological cycle with various consequences concerning water availability, water quality and water risks like water pollution, water deficiency, and floods. In the last decade several projects have been initiated dealing with methods, techniques and tools to support a sustainable water resource management.

In this article, we report on the role of informatics in the interdisciplinary project GLOWA-Danube [7] which is part of the German initiative GLOWA (Global Change in the Hydrological Cycle; cf. [6]) and sponsored by the German

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

Federal Ministry of Education and Research between 2001 and 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 which covers parts of Germany, Austria, Switzerland, Italy, and the Czech Republic.

The aim of the project is to identify and simulate strategies for adaptation to and mitigation of the consequences of climate change and to test the effectiveness of these strategies. Therefore, the decision support system DANUBIA has been developed in order to investigate the impact of climate change, population and land use on the water resources of the Upper Danube in different scenarios. It aims to support the development and evaluation of regional adaptation strategies. DANUBIA is a distributed simulation system which integrates simulation models for natural processes (hydrology, plant physiology, groundwater, and glaciology) as well as socio-economic models (agriculture, economy, water supply, private households, and tourism), simulating the behaviour of the involved actors based on the structure of societies and their interests. The simulation models of DANUBIA 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. After the current project phase it will be made available as open source software for decision makers from policy, economy, and administration as a tool for the sustainable planning of water resources based on the evaluation of coupled simulations which are driven by scenarios for the next 50 years.

The informatics group played a major role in this interdisciplinary project, being responsible for the development and realisation of the DANUBIA system. For this purpose, best practices of software engineering have been applied to deal with the system complexity which involves (among others) the aspects of information exchange, simulation space, and coordination of concurrently running simulation models. For guaranteeing the correctness of the coordination, formal methods have been applied. For supporting the reusability of the system a component-based framework has been implemented which allows to plug in arbitrary simulation models as long as the requirements concerning the simulation space and the general life cycle of a model are satisfied. For this purpose, the framework provides abstractions for the implementation of simulation models and thus facilitates the development of the models by software engineers of the different research groups. Furthermore, the integration of various simulation models into a coupled simulation (which is coordinated by the informatics group) is considerably simplified since most errors in simulation models caused by not meeting the common requirements for simulation models are detected during compilation and thus early before actually running a simulation.

The contributions of the informatics group were, however, also crucial for the scientific development of the simulation models of the different research groups. Indeed for interdisciplinary projects the risk to fail is particularly high due to problems arising from the diversity of the various disciplines ranging from different notions, physical units, and terminologies up to completely different methodologies and modelling approaches. In GLOWA-Danube it emerged that the rigorous rules for abstraction and separation of concerns introduced by the informatics group helped to avoid misunderstandings and inconsistent assumptions. In particular, it was mandatory for all research groups of the project to model the integrative aspects of their simulation models in terms of (a simple

subset of) the UML notation which turned out to be very useful to achieve a common understanding of the system.

The outline of this paper is as follows. In Sect. 2 we identify common requirements for integrative environmental simulations and elaborate in detail on the different aspects of the design of such systems. Moreover, we comment on the impact of applying software engineering techniques during development of DANUBIA on the development of simulation models of the different research groups. Then, in Sect. 3 our framework approach is presented followed by a description of its implementation. Finally, in Sect. 4 we shortly discuss related work and summarise the main results and conclusions that we draw from this project.

2 Aspects of Integrative Environmental Simulations

As our field of interest is computer-based environmental simulation, 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*, regarding 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 handling of simulation time and simulation space in the integrated model is crucial.

The major goal of the GLOWA-Danube project was to develop the simulation and decision support system DANUBIA. With DANUBIA the simulation models of the single participating disciplines implementing the dynamics of natural or socio-economic processes should be coupled. Within the project the informatics group was responsible to implement the DANUBIA framework (described in Sect. 3.2), whereas the single simulation models should be developed by the respective disciplines.

The development of a system like DANUBIA is a great challenge due to its great complexity imposed by the amount of different models coordinated. Fortunately, by applying well-proven informatics techniques the complexity of the problem could be reduced significantly. By abstraction from the concrete requirements of integrative environmental simulations, four major system aspects could be identified:

1. data exchange between simulation models,
2. coordination of simulation time,
3. modelling of simulation space, and
4. support for agent-based social simulation.

After dividing the requirements into several aspects, by applying the principle of separation of concerns each of these functional requirements was implemented separately and the solutions have then been integrated. In the following we explain the four major and fundamental aspects of the integrative environmental simulation system under consideration with an emphasis on how principles and techniques of informatics have influenced system development.

2.1 Data Exchange between Simulation Models

In our approach the coupling of simulation models is based on interfaces. In general, interfaces are abstractions of software entities that specify provided methods to be used for communication. However, in DANUBIA, an interface always plays two roles: for a simulation model implementing the methods of that interface it is a provided interface; for a simulation model using the methods of that interface (or more precisely, using a model for which that interface is a provided interface) it is a required interface. Simulation models may always be equipped with several provided and required interfaces specifying which methods are required and which are provided. In DANUBIA, methods specified in interfaces are used for data exchange, so provided interfaces specify data that is offered to other models, and required interfaces specify data that is needed from other models. The usage of interfaces has several advantages. First of all, low coupling between the simulation models is achieved, i.e., one model does not need to know about implementation details of the other, but only knows the public interface. This leads directly to the advantage that simulation models can easily be exchanged as long as they provide and require the same interfaces.

Furthermore, an interface always specifies a contract. The model which implements an interface is obliged to comply with the contract (i.e. implement all methods of the interface), and the model that uses an interface can rely on the contract (i.e. only use methods that are specified in the interface). Thus, for the model developers and scientific researchers the interface is the point for discussions about the data to be exchanged.

For the notation of interfaces the Unified Modelling Language (UML, [14]) turned out to be an appropriate means. Fig. 1 shows an example of three interconnected simulation models (*Groundwater*, *Watersupply*, and *Economy*) in UML notation. As pointed out above, an interface always plays two roles: it acts as an export interface for the provider model and as an import interface for the user model. For example, the interface `GroundwaterToWatersupply` is an export interface of `Groundwater` and provides an operation `getGroundwaterLevel` to be used by `Watersupply`.

The operations for data exchange are required to comply to a predefined signature pattern of the form `getX():T`, where X denotes a parameter name, like e.g. `GroundwaterWithdrawal` and T a predefined datatype, like e.g. `WaterFlux`. The use of project-wide common datatypes is crucial to avoid errors caused by the usage of different physical units in the various disciplines. For example, water flux is typically measured by hectolitres per year in social sciences and by cubic metre per second in natural sciences. So under the guidance of the informatics group an agreement had to be achieved which consists in the usage of standard (SI) units. Possibly in some models data have to be converted in order to match the standard units, but this effort is by far compensated by avoiding mistakes and misunderstandings.

Furthermore, the interfaces can be enhanced by specifying the expected and assured ranges of exchanged data such that interface compatibility can be checked upon setup of a simulation configuration.

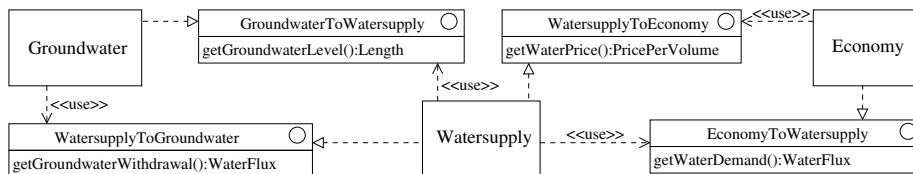


Figure 1: Interfaces between simulation models

2.2 Coordination of Simulation Time

An important characteristics of DANUBIA is the possibility to perform integrative simulations where the single simulation models run concurrently and exchange information at run time over their interfaces. In order to obtain reliable simulation results it is necessary that the exchanged data is suitable for the current time of the importing model, and hence an appropriate coordination of the single simulation models is crucial. As each model usually has an individual time step (i.e. the model time between two consecutive computations) depending on the simulated process and ranging from one hour, like e.g. in meteorology, to one month, like in social sciences, the correct coordination of the simulation models is a non-trivial task.

To cope with problems like this the application of formal software engineering methods dealing with concurrent programming like petri nets or process algebras is useful. Here we use the process algebra Finite State Processes (FSP) introduced by Magee and Kramer ([13]). The particular advantage of FSP is that it allows for specifying requirements in a property-based approach as well as for developing a design model that can be systematically transformed into an implementation. Moreover, by applying model checking techniques it could be verified that the design model meets the specified requirements.

In [9] the coordination problem has been specified in terms of FSP property processes. The idea to simplify the coordination problem is to consider only two simulation models at a time, one acting only as provider, the other one as user of data. The resulting property process specifying the requirement of getting valid (i.e. data that matches the local model time) then reads as follows:

```

property VALIDDATA(User=1,StepUsr=2,Prov=2,StepPrv=3)=
  VD[SimStart][SimStart],
  VD[nextGet:Time][nextProv:Time] =
    // no obsolete data
    (when (nextGet<nextProv)
     [User].get[nextGet]->VD[nextGet+StepUsr][nextProv]
    // no overwritten data
    |when (nextProv<=nextGet)
     [Prov].prov[nextProv]->VD[nextGet][nextProv+StepPrv])

```

It says that the user model is only allowed to get data (from its import interface) if the most recent data is available (i.e. the next data will be provided “in the future”) and, conversely, that the provider model is only allowed to provide data (on its export interface) if its last recently provided data is not to be read anymore by any other simulation model.

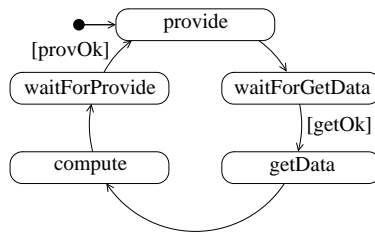


Figure 2: Life cycle of a coordinated simulation model

A design model for a global coordination component (timecontroller) has been derived from the specification, and the correctness of the design model could be verified by model checking techniques. An implementation of a time coordination component derived from the given design model is presented in [10]. Fig. 2 shows the resulting life cycle of a coordinated simulation model with the states *getData*, *compute* and *provide* as well as intermediate waiting states caused by synchronisation. In the single states the model has to execute the following actions:

- *get* required import *data* from other models;
- *compute* new data for the next time step;
- *provide* the newly computed data for other models (i.e. change the state visible over data interfaces).

To ensure that the behaviour of each simulation model obeys the rules for coordination the life cycle shown in Fig. 2 is implemented in the DANUBIA framework (cf. Sect. 3.2).

2.3 Modelling of Simulation Space

In DANUBIA, the simulation space (i.e. the Upper Danube Basin) is divided by a two dimensional grid of 430 rows and 425 columns into quadratic cells of one square kilometre. These atomic, geographic cells are modelled by so-called *proxels* (from process pixel, [15]). A proxel is not only modelling a structural element of the simulation space describing the (current) environmental state of the respective area, but also characterises the environmental processes that are to be simulated at this point in the simulation space. The abstract class `Proxel` specifies basic properties (e.g. elevation, cf. Fig. 3) that are common for all simulation models and that remain constant during simulations. By using object-oriented inheritance concrete proxels can be modelled by adding in a subclass of `Proxel` domain-specific properties and by implementing the abstract operation `computeProxel` which determines the local environmental process. In Fig. 3, an excerpt of the resulting proxel hierarchy is shown. The concept of proxels is implemented in the DANUBIA framework (cf. Sect. 3.2).

2.4 Support for Agent-Based Social Simulation

A further fundamental aspect of DANUBIA is the explicit support of social simulation using concepts from the field of multiagent systems.

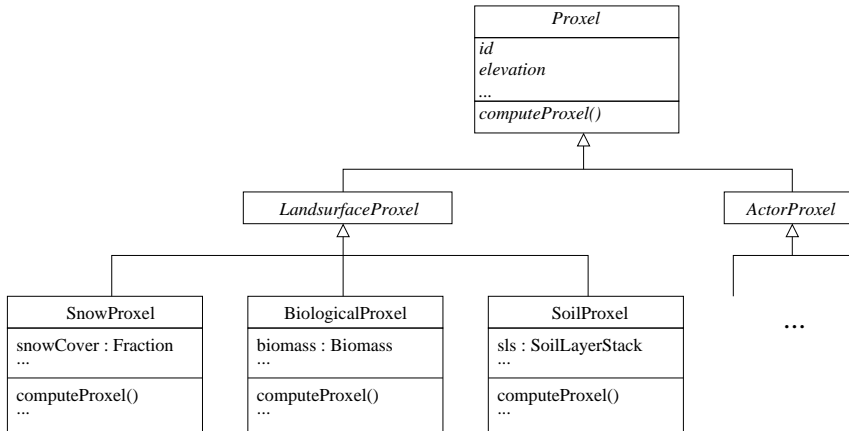


Figure 3: Proxel hierarchy

While the social simulation models of DANUBIA integrate into coupled environmental simulations like any other simulation model by interface-based data exchange, their local computation differs from that in the natural science models. Here the key elements are *actors* which model deciding entities such as individuals (e.g. farmers), organisations (e.g. water suppliers), companies, and so forth. To decide means to react to a certain environmental state by choosing a certain action among a set of predefined options. The reasoning which precedes the selection process usually consists of simple if-then rules. The actors may interact with each other by data exchange, or may modify properties of the part of the simulation area they are situated in. Therefore interaction is either direct, or indirect by perceiving modifications of the simulation area. Social science simulation models then often are not interested in the local modification of a single spot of the simulation area, but aim to study so-called emergent phenomena. Emergence can be understood as a kind of collective result, which is not existing or visible on the level of its causing entities. For example temperature is an emergent phenomena of moving atoms.

Modelling decision-making entities is a challenge which has been studied in artificial intelligence for quite some time. The field of multiagent system then came up as an approach to distributed artificial intelligence [16]. Multiagent systems focus on the interaction between agents instead of properties or abilities of single intelligent entities. This focus makes them an excellent tool for the modelling and simulation of societal issues, in particular for the development of social simulation models with a conception as described above. Even though the social simulation models of GLOWA-Danube made use of only a few concepts known from multiagent systems, knowledge on artificial intelligence and multiagent systems helped to clarify their common requirements and to settle down a common understanding of basic notions such as “actor”, “plan” or “action” and, in particular, what it means for an actor to “decide”. We implemented the common conceptualisation within the so-called DEEPACTOR framework, an object-oriented extension of DANUBIA. The framework and its integration is described in more detail in Sect. 3. In the following we provide a short discussion on the conceptual relationship between the DEEPACTOR framework and

multiagent systems.

The basic entities of multiagent systems are intelligent agents. Wooldridge discusses their abstract properties in [18, 17] which can be considered independent from an application domain. Agents are situated within some environment which evolves in a discrete way by passing through a sequence of states. A standard agent is equipped with a set of actions and its behaviour might then be characterised by a mapping from environmental states to actions, i.e. an interrelation between the abilities of a certain agent within a certain environmental state. The behaviour of the environment is defined by a mapping of environmental states and agent actions to new environmental states. By this means, interaction of agents and their environment can be represented by a history, an alternating sequence of environmental states and agent actions.

However, this simple form of interaction is useful only for purely reactive agents or simple history-equipped agents, whose decision is based solely on the current environmental state or a history thereof. Often it is required to record information which is local to the particular agent. Such a requirement naturally gives rise to the concept of agents with state. The decision now incorporates also the local state of an agent and agent behaviour now essentially is described by the following scheme: after started within an initial state, agents perceive environmental states and update local states where necessary. After that an action is chosen which may modify the environment and the agent is ready to start another round of environmental perception, decision and action. The cyclic nature of such an agent behaviour provided a perfect match with the requirements of continued cyclic computation during a simulation with DANUBIA. Therefore, the basic computation cycle of an actor within the socio-economic simulation models of DANUBIA follows exactly this pattern of data import, internal state update, decision and action.

Depending on the concrete decision process, various fundamental types of agents can be distinguished, giving rise to different kinds of “agent architectures”. Wooldridge [17] considers four concrete agent architectures: logic based architectures, reactive architectures, belief-desire-intention architectures and layered architectures. For the conception of the DEEPACTOR framework only reactive architectures played a role, since this kind of architecture turned out to be an appropriate common basis for all of the socio-economic simulation models in DANUBIA. The subsumption architecture of Rodney Brooks is one of the best-known examples of this kind of agent architecture. It is characterised by simple behaviours which directly map perceived states to actions to be executed, together with the parallel execution of these simple behaviours. Exactly these two characteristics were taken up for the design of the DEEPACTOR framework as an abstract basis for the development of socio-economic simulation models in GLOWA-Danube.

3 Framework and Implementation

The concepts described in the previous section are implemented in the distributed simulation system DANUBIA which is a component-based object-oriented system consisting of the simulation frameworks on the one hand, and the simulation models encapsulated by components with import and export interfaces on the other hand. Figure 4 provides an overview of the system’s architecture

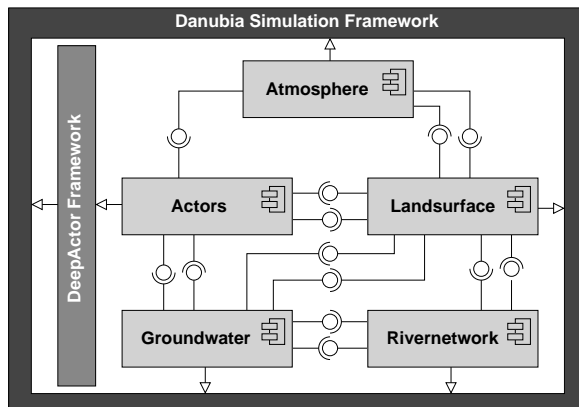


Figure 4: System architecture

showing the main components together with their relation to the DANUBIA and the DEEPACTOR framework.

The implementation followed a framework-based approach similar to the Catalysis approach described in [3]. Therefore, we give an overview of our approach in the succeeding section and, after that, detail on the realisation of the aspects discussed above.

3.1 Frameworks and Components

The development of DANUBIA within GLOWA-Danube was driven, on the one hand by the interdisciplinary project structure and, on the other hand, by the application domain of natural-science and socio-economic simulation models. The project structure called for support of a decentralised development of the single simulation models. To obtain an integrated, coupled simulation system, obviously these single components must be composed at some later point in time. Therefore, we chose a component-based approach to system development. Components are encapsulated units which require clearly specified interfaces to support late composition. Moreover component-based software development explicitly aims to support independent implementability and substitution of components by relying only on interface specifications. For DANUBIA this yields a flexible way to compose simulation models essentially relying only on the interfaces specifying their required import and their provided export data.

Even though component concepts seem to provide an appropriate design paradigm, it is still not clear how to realise an abstract implementation for the remaining aspects simulation space, time and actors. What is needed is a notion of generic components that implement structure and mechanisms which, on the one hand is common for all simulation models and, on the other hand, is easily extended to fill in the missing bits or add further required concepts. Catalysis [3] uses the notion of plug-points, provided by a generic component, and plugins provided at some later point in time to complete the generic component into an executable implementation.

Generic components can be realised using an object-oriented approach to framework development. At the core of such an approach are design patterns,

in particular the template pattern [5]. The idea of this pattern is to implement the skeleton of an algorithm using calls of abstract methods whose implementation is deferred to subclasses. If such a subclass is provided, the execution of the algorithm automatically integrates the missing bits by calls to the subclass object. An important characteristic of the framework approach in general and the template pattern in particular is the inversion of control principle¹: calls go from framework to concrete level but not the other way around. Allowing up-calls in an uncontrolled way usually leads to unmaintainable control flow structures. Committing to the principle of inversion of control was therefore an important principle for the development of the DANUBIA system.

In a nutshell, our approach for the development of the DANUBIA and DEEP-ACTOR framework refines, reifies and extends the framework approach to code reuse of Catalysis [3]. The frameworks provide abstract base classes which define plug-points to be implemented by particular concrete subclasses. To establish basic structural relationships interfaces are applied to define basic roles one modelling element plays for another modelling element. The main extension with respect to [3] is the strict separation of the framework from the modelling level, which is illustrated in Fig. 5. The uppermost level decouples the framework's core implementation from the abstract programming interface in the middle layer (developer interface) used by simulation model developers as shown in the third layer. The abstract base classes of the middle layer contain those methods only which are in fact relevant for their concrete subclasses. This approach completely hides the framework usage of the plug-point implementations, reducing the complexity of the resulting programming interface to the absolutely necessary minimum. We have used UML 2.0 for the framework design, specifically for the documentation of the developer framework, and Java SE 6 for the implementation.

3.2 Danubia Framework

The DANUBIA framework is a component-based object-oriented framework (implemented in Java) which comprises a runtime environment for the execution of integrative simulations on the one hand and a developer interface for the implementation of DANUBIA-conform simulation models on the other hand. The class diagram in Fig. 5 shows a simplified excerpt of DANUBIA (without taking into account components) which illustrates the framework approach of DANUBIA.

Fig. 5 is separated into three layers by dashed lines. The uppermost layer represents the framework core layer. In this layer all features that can be handled by the framework are implemented like, e.g., the time coordination (represented by the class `Timecontroller`) (cf. Sec. 2.2), the initialisation of simulation space (`ProxelTable`) and the common properties of a simulation model (`ModelCore`). The life cycle of a simulation model (cf. Fig. 2) together with calls to the global timecontroller instance is implemented in the `run` method of the class `ModelCore`. From here the abstract operations of the class `AbstractModel` are called according to the coordinated life cycle. At runtime a concrete simulation model instance is substituted for the type `AbstractModel` and the concrete operations of this class are called and executed. The core layer of the framework is transparent to the model developer.

¹also known as the Hollywood principle: "don't call us, we'll call you."

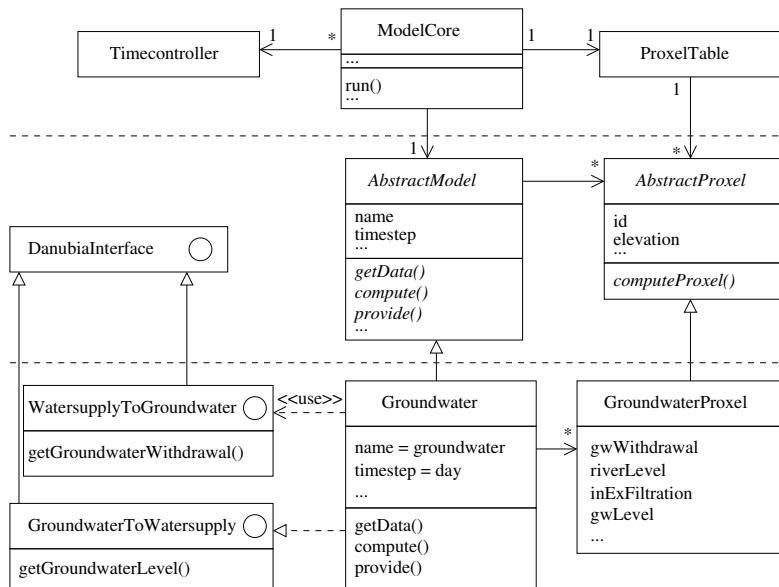


Figure 5: The three levels: framework core, developer interface and concrete simulation model

The middle layer in Fig. 5 constitutes the developer interface of the DANUBIA framework. Here one can find abstract base classes like `AbstractModel` or `AbstractProxel` which contain plug points in terms of abstract operations. The plug points have in turn to be implemented in concrete subclasses which represent concrete simulation models. For example, the base class `AbstractModel` defines the plug points `getData`, `compute` and `provide` corresponding to the actions of the model within its life cycle. The interface `DanubiaInterface` is a marker interface for all data interfaces in DANUBIA. The developer interface contains also datatypes that can be used in model interfaces and tools for pre- and post-processing of input and output data.

The lowermost layer represents a concrete simulation model, in Fig. 5 the representation of the `Groundwater` model is depicted. One can see that all model classes are derived from base classes of the developer interface. From this we can elaborate the steps that are necessary to implement a DANUBIA simulation model (in Java).

1. Implement a concrete subclass of `AbstractModel`. This class is called the model main class. In the model main class implement all plug points according to the simulated processes. Within `getData` the required import data from other models are requested. In `compute` the computation of new data for the next time step is performed, probably by calling `computeProxel` on each relevant proxel object. In `provide` the newly computed data is provided to other models in form of export tables which can be retrieved by other models in their `getData` operation.
2. Implement a concrete subclass of `AbstractProxel` which contains attributes for all properties of the simulation space the respective model is interested in. The plug point `computeProxel` represents the computation of one time

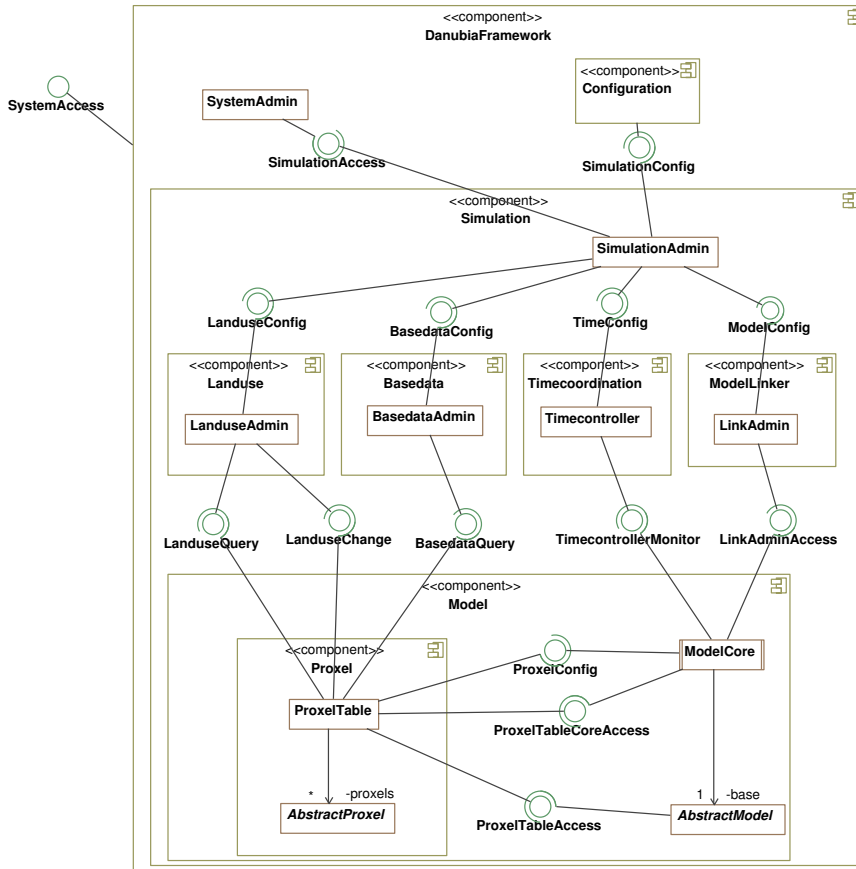


Figure 6: Overview of the DANUBIA framework

step at the respective point in simulation space represented by the proxel object.

3. Make sure that all obligations caused by provided interfaces are fulfilled.

Of course a model implementation can consist of more than the classes depicted in Fig. 5. The use of the model main class is yet mandatory. In the case of a legacy model, the model main class may act as a wrapper class for the model, where the plug point implementations possibly delegate to the Java Native Interface (JNI).

An overview of the DANUBIA framework is given by the component diagram in Fig. 6. The framework includes components for coordination, model linking (including network support), initialisation of common spatial data, and a component for the management of simulation configurations. Moreover, it comprises a graphical user interface where simulation configurations can be flexibly built (in accordance with the interface conformance rules) and integrative simulations can be started and observed during runtime.

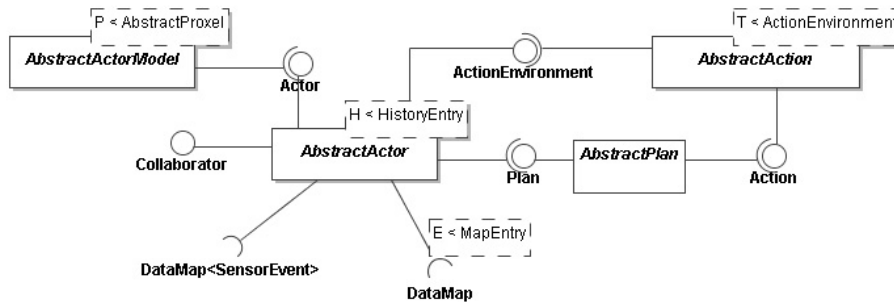


Figure 7: Basic structural relationships for DEEPACTOR models

3.3 DeepActor Framework

The DEEPACTOR framework is a generic Java implementation providing a common architecture for the modelling and implementation of agent-based social simulation models in GLOWA-Danube, called DEEPACTOR models hereafter. This section first explains the implementation of the actor concepts within the DEEPACTOR framework based on Sect. 2.4, then briefly addresses the integration with the DANUBIA framework and after that provides an example to illustrate further details of our framework approach.

For the concrete framework design and implementation we have used abstract base classes and interfaces as depicted in the structural overview of Fig. 7. The base classes are directly related to the concepts described in Sect. 2.4 above. DEEPACTOR models (`AbstractActorModel`) refer to an arbitrary number of actors (`AbstractActor`). Actors choose among a set of plans (`AbstractPlan`), each comprising a number of actions (`AbstractAction`) whose execution may modify the actor’s environment. Plan selection is usually triggered by certain sensor events (`DataMap<SensorEvent>`). The interface `Collaborator` can be used to model interaction between different actors. Applying interfaces for the specification of structural relationships leads to a looser coupling between the abstract base classes. This allows for a more modular implementation and reuse of the concrete single elements of a socio-economic simulation model. The integration into the DANUBIA framework is by extension of the developer framework of DANUBIA. The base class `AbstractActorModel` is realised as a subclass of `AbstractModel`, thereby introducing the complete set of base classes and interfaces for DEEPACTOR models solely by an extension of the middle layer of Fig. 5, transparent for the core layer of DANUBIA.

Finally, by focusing on the base class `AbstractAction` and its associated interfaces `Action` and `ActionEnvironment`, we provide a further illustration of our framework approach from a model developers point of view. An action is an explicit representation of an environmental state modification and, as already depicted in Fig. 7, is always part of a plan. Figure 8 shows the detailed specification of the particular base class. The specification makes use of UML stereotypes `query` and `plug-point` which were introduced in our framework approach to indicate the kind of a method available in concrete subclasses. Queries are allowed to be used in subclasses (safe up-calls) while plug-points are methods which are required to be implemented. Such a distinction directly relates to the

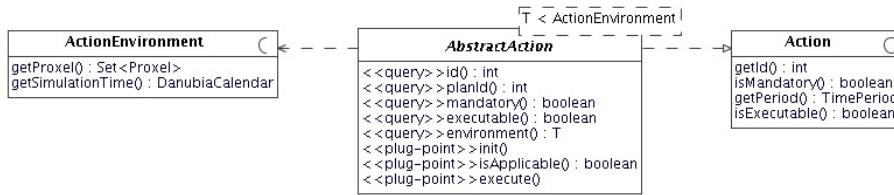


Figure 8: Base class `AbstractAction` and its associated interfaces

three layers of our framework approach (cf. Fig. 5). Queries are implemented by the framework in the middle layer for a controlled delegation of up-calls and plug-points are implemented by the model developer in the lower layer to provide the missing bits, specific to the particular simulation model.

The remaining concepts of socio-economic simulation models in GLOWA-Danube have been specified and implemented analogously. By this means, a concise and precisely defined developer framework was developed and successfully applied by the different project partners for the realisation of their simulation models [4, 2, 1].

4 Conclusions and Related Work

In this paper we reported on our experiences to develop an integrative environmental simulation system in an interdisciplinary project. On the one hand, the informatics group supported the conceptual development of the system, for example by introducing the UML as a project-wide graphical notation for interfaces and the documentation of the single models; by creating a common base for the integration of simulation models from various disciplines with regard to common concepts for simulation time and space; by supporting agent-based social simulation with concepts from the field of multiagent systems. On the other hand, based on these concepts, a framework-based integrative environmental simulation and decision support system has been developed and successfully applied in the project. With the system a number of water-related scenarios have been simulated which shall show the impacts of human behaviour on the future of the water resources under different future assumptions. Concerning performance, integrative simulation runs for typical configurations over a 50 year period actually take (at least) 36 hours.

By applying the framework paradigm common routines and services like network support, time coordination or space initialisation could 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.

In the end let us say some words about related approaches. In the literature one can find numerous systems deling with integrative environmental simulations. Examples of such systems very similiar to the DANUBIA approach are the

ModCom (cf. [11]) or the Object Modeling System (OMS, cf. [12]). Both systems provide a framework which allow for implementation of simulation models and execution of integrative simulations.

A quite different approach to that of DANUBIA is that of the Open Modelling Interface and Environment (OpenMI, cf. [8]). The OpenMI standard is not based on a framework, but on a set of interfaces a model has to implement in order to be OpenMI compliant. Like in DANUBIA, simulation models are accessed directly for data exchange during a simulation run. One of the main differences between OpenMI and DANUBIA is the fact, that OpenMI does not incorporate a global time coordination instance although allowing for different time steps of the single models. Instead, OpenMI is based on a request and reply mechanism, where one model at the beginning of a simulation is triggered and then requests data from other models which it needs for its computation. The requested model computes the requested data for which it in turn requests data from the next model and so on. OpenMI therefore does not allow for parallel execution of the models, which is one of the main benefits of DANUBIA.

Acknowledgement. We would like to thank Ludwig Adam and Philip Mayer for careful reading of a draft of this paper and giving many valuable hints and suggestions.

References

- [1] R. Barthel, 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, 2009. in press; available online as DOI: <http://dx.doi.org/10.1007/s11269-009-9445-y>.
- [2] R. Barthel, 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. *Env. Modelling & Software*, 23(9):1095 – 1121, 2008.
- [3] D. D’Souza and A. Wills. *Objects, Components and Frameworks with UML – The Catalysis Approach*. Addison-Wesley, Reading, Massachusetts, 1999.
- [4] A. Ernst, C. Schulz, N. Schwarz, and S. Janisch. Modeling of water use decisions in a large, spatially explicit coupled simulation system. In B. Edmonds, C.H. Iglesias, and K.G. Troitzsch, editors, *Social Simulation: Technologies, Advances and New Discoveries*. Idea Group Inc., 2007.
- [5] E. Gamma, R. Helm R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995.
- [6] GLOWA. Project website. <http://www.glowa.org/> (last accessed on 2009-09-26), 2009.
- [7] GLOWA-Danube. Project website. <http://www.glowa-danube.de/> (last accessed on 2009-09-26), 2009.

- [8] J. B. Gregersen, P. J. A. Gijsbers, and S. J. P. Westen. OpenMI: Open modelling interface. *Journal of Hydroinformatics*, 9(3):175–191, 2007.
- [9] R. Hennicker 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.
- [10] R. Hennicker and M. Ludwig. Design and implementation of a coordination model for distributed simulations. In *Proc. Modellierung 2006*, volume P-82 of *Lect. Notes Informatics*, pages 83–97. Gesellschaft für Informatik, 2006.
- [11] C. Hillyer, J. Bolte, F. van Evert, and A. Lamaker. The ModCom modular simulation system. *European Journal of Agronomy*, 18(3):333–343(11), January 2003.
- [12] S. Kralisch, P. Krause, and O. David. Using the Object Modeling System for hydrological model development and application. *Advances in Geosciences*, 4:75–81, 2005.
- [13] J. Magee and J. Kramer. *Concurrency : state models & Java programs*. Wiley, Chichester, 1999.
- [14] Object Management Group. *OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2*, 2007.
- [15] J. D. Tenhunen and P. Kabat, editors. *Integrating Hydrology, Ecosystem Dynamics, and Biogeochemistry in Complex Landscapes*. Wiley, Chichester, 1999.
- [16] G. Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, 1999.
- [17] M. Wooldridge. *Intelligent Agents*, chapter 1, pages 27–79. In Weiss [16], 1999.
- [18] M. Wooldridge. *An Introduction to Multiagent Systems*. Wiley and Sons, 2002.