

## Chapter 2

# DANUBIA: A Web-Based Modelling and Decision Support System to Investigate Global Change and the Hydrological Cycle in the Upper Danube Basin

Rolf Hennicker, Stefan Janisch, Andreas Kraus, and Matthias Ludwig

**Abstract** We describe architecture and design principles of the integrative modelling and simulation system DANUBIA. The system integrates the distributed simulation models of the socio-economic and natural science disciplines of the GLOWA-Danube project. During an integrative simulation the simulation models run in parallel and exchange iteratively data. The validity of the exchanged data with respect to model time is guaranteed by a central time controller which coordinates the single simulation models. The consistency of spatial data is ensured through the common proxel concept used by all models. The development of DANUBIA is based on object-oriented software engineering methods. A generic framework architecture has been constructed which implements general rules for the behaviour of all simulation models. It can be specialised and thus instantiated by concrete simulation models. We also have developed a particular framework for the socio-economic deep actor models.

**Keywords** GLOWA-Danube • Distributed system • Coupled simulation • Time controller • Proxel concept • Framework architecture

## 2.1 Overview

DANUBIA (Barth et al. 2004) is an integrative simulation and decision support system that was developed in the context of GLOWA-Danube. DANUBIA facilitates the study of scenarios related to the hydrological cycle from ecological and economic points of view in order to support scientists and policymakers in designing strategies for sustainable environmental management.

---

R. Hennicker (✉) • S. Janisch • A. Kraus • M. Ludwig  
Institute for Informatics, Ludwig-Maximilians-Universität München (LMU Munich),  
Munich, Germany  
e-mail: [hennicker@ifi.lmu.de](mailto:hennicker@ifi.lmu.de); [stephanjanisch@gmail.com](mailto:stephanjanisch@gmail.com); [AndiKraus@web.de](mailto:AndiKraus@web.de);  
[Matthias.Ludwig@ifi.lmu.de](mailto:Matthias.Ludwig@ifi.lmu.de)

Sixteen simulation models from all of the participating research groups are integrated within DANUBIA. This permits the study of both sectoral and interdisciplinary issues while taking into account the interactions of interdependent processes.

The development of DANUBIA is based on methods from object-oriented software engineering and web engineering. At all stages of the development process, the *Unified Modeling Language (UML)*, Booch et al. 1999) played a key role as the common notation that was used by all project partners to describe the integrative aspects of the systems.

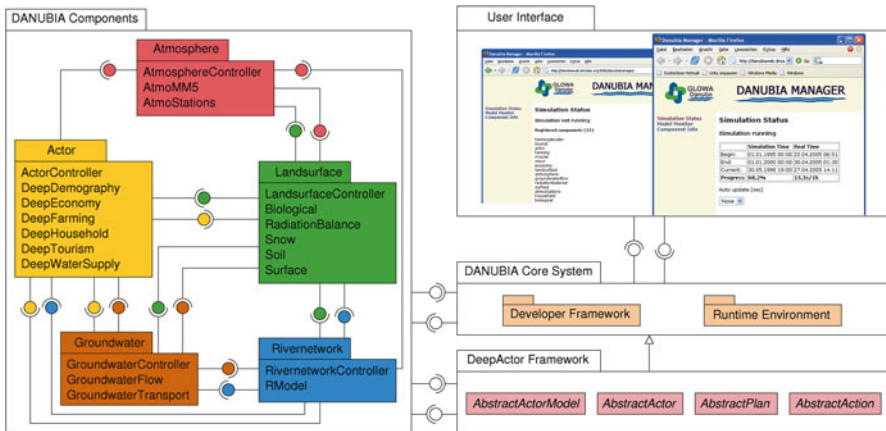
The architecture of the DANUBIA system is illustrated in Fig. 2.1. The simulation models from the different specialist groups of GLOWA-Danube are realised within the *Components* component of DANUBIA. The *Core System* component of DANUBIA consists of a developer framework and a runtime environment (see Sect. 2.3). DANUBIA contains the DeepActor framework for integration of actors for the implementation of the socio-economic models.

At present, DANUBIA runs on a computing cluster with 56 processors. However, the system can be installed on individual computers or small networks for testing purposes.

## 2.2 Concept

### 2.2.1 Main Components and Interfaces

The sixteen simulation models integrated within DANUBIA are grouped by theme into five main components: *Atmosphere*, *Actor*, *Landsurface*, *Groundwater* and *Rivernetwork* (see Fig. 2.1). The exchange of data both among the main



**Fig. 2.1** Architecture of the DANUBIA system

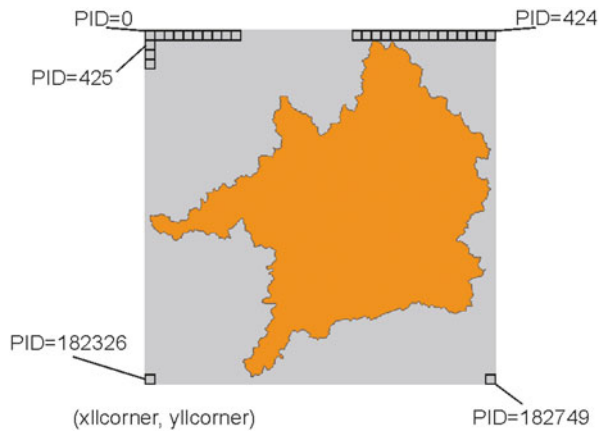
components and between the individual simulation models takes place via interfaces. The interfaces contain the identifier and the data type for each parameter that is to be exchanged. The validity period of the exchanged data is ensured separately by an internal temporal coordination concept (see Sect. 2.2.3).

### 2.2.2 Spatial Concept

One central element of simulation of the environment involves the treatment of the simulation area. In DANUBIA, the simulation area is represented using a two-dimensional grid (see Fig. 2.2). The rows and columns of this grid are arranged using Lambert conformal conic projections of geographical coordinates. The coordinate points correspond to those in the Hydrological Atlas of Germany.

The length and width of the grid cells are each 1,000 m, resulting in a grid area of 1 km<sup>2</sup>. The use of object-oriented paradigms gives the simulation area a structure that is both static and dynamic, the latter consisting of the processes that take place on the corresponding site in the grid. This led to the term proxel (an acronym for process pixel). A proxel object is identified by an identification number that is unique within the simulation area being considered; this identifier is known as the ProxelID (see PID in Fig. 2.2). All proxel objects are administered in a table object, called a ProxelTable. A universal proxel object stores the relevant characteristics of the specified grid point (coordinates, terrain elevation, land use, etc.) for all simulation models. Additionally, subject-specific characteristics of the proxels are added to the individual DANUBIA components via specialisation.

**Fig. 2.2** The Upper Danube drainage basin



### 2.2.3 Temporal Concept

Over the entire simulation period, a simulation model calculates data at discrete time points; these data describe the current valid state of the modelled system. The interval between two such points in time is constant for each model and is known as the model time interval. The lengths of the model time interval in DANUBIA range from 1 h (e.g. in the *Atmosphere* and *Landsurface* models) to 1 month (in most of the actor models). Several models with differing time intervals are coupled within an integrated simulation, and periodic calculations are carried out, and data is exchanged during the runtime. In order to obtain reliable results from these simulations, the exchange of data must comply with the following conditions:

- The exchanged data must be stable; i.e. there must not be simultaneous access to the data for writing and reading.
- Each model must receive data upon a data request; this data must be valid with respect to its own local model time.

In order to comply with these requirements, DANUBIA has a component for temporal coordination among the individual models involved, known as a *Timecontroller* (Hennicker and Ludwig 2005, 2006). For the models themselves, there is thus the following life cycle (see Fig. 2.3):

*waitForGetData*: wait for the release to read data from other models using the *Timecontroller*.

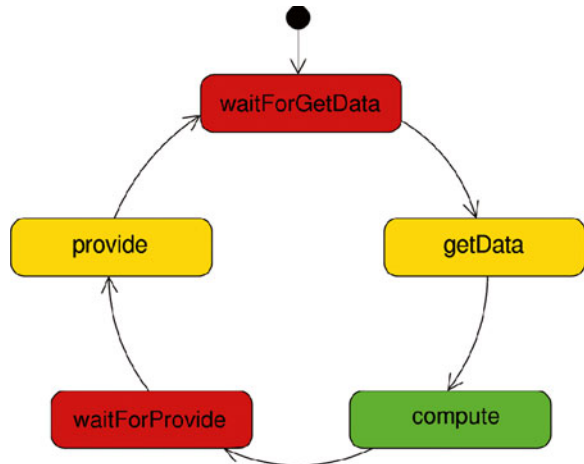
*getData*: read data from models.

*compute*: calculate the relevant data at the next simulation time point.

*waitForProvide*: wait for the release to provide the new calculated data for other models using the *Timecontroller*.

*provide*: provide the new calculated data for other models.

**Fig. 2.3** Life cycle of a coupled simulation model



### 2.3 DANUBIA Core System

The DANUBIA core system consists of a developer framework and a runtime environment (see Fig. 2.4).

#### 2.3.1 Developer Framework

The developer framework supports the model developer by providing the classes and interfaces that are used by the model developers to integrate their model implementations in DANUBIA.

The first aspect involves something known as base classes, which must be specialised according to the object-oriented inheritance principle. The most important base class is the *AbstractModel*, from which the specific model implementations are derived in DANUBIA. This base class already contains the implementation of the life cycle of a simulation model (see Sect. 2.2.3) and forms the basis for the connection of the model to the runtime environment (see Sect. 2.3.2). The *AbstractProxel* class is also in the base class category; this class realises the spatial concept described in Sect. 2.2.2.

Additional classes that are used by the model developers include the implementations of the jointly used data types in DANUBIA and tools for pre- and post-processing of input and output data. These tools allow the conversion of

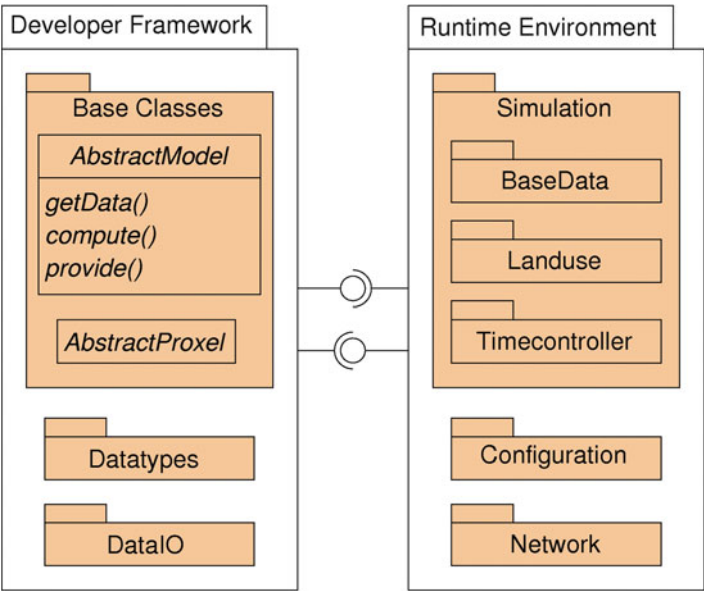


Fig. 2.4 DANUBIA core system

data between typical GIS format and the internal binary data format of DANUBIA, for example; they also allow temporal and spatial aggregation or averaging of data.

### 2.3.2 Runtime Environment

The runtime environment can handle either integrated simulation runs of the DANUBIA system on a single computing cluster or test runs of smaller model configurations on individual computers or smaller local networks. The runtime environment essentially consists of the components described below.

The main component of the runtime environment is the *Simulation* component with its subcomponents, *BaseData*, *Landuse* and *Timecontroller*. The *Simulation* component is responsible for the administration of the simulation models during a simulation, the subcomponents are assigned the following tasks: the *BaseData* component initialises the relevant universal characteristics of the proxel (see Sect. 2.2.2); the *Landuse* component administers the changing land use of the simulation regions during a simulation run and provides it to the simulation models that are involved. Finally, the *Timecontroller* component implements the temporal coordination of the individual simulation models described in Sect. 2.2.3 during an integrated simulation run.

The *Configuration* component generates the simulation configurations. In doing so, it interacts with the user interface (see Sect. 2.4). The distribution of the individual parts of the system over a network is carried out by the *Network* component. In particular, it is responsible for connecting the corresponding data import and data export interfaces of the models involved in an integrated simulation.

## 2.4 The User Interface

Distributed simulation runs can be configured, controlled and monitored with the DANUBIA web interface (see the *UserInterface* component in Fig. 2.1). The user interface offers different views depending on the status of the simulation system.

Prior to the start of a simulation run, the names of the simulation models are indicated. Once the simulation configuration is complete, the simulation can begin.

Once a simulation has commenced, the time progress of both the individual models and the integrated model group can be seen. The progress is indicated in real time and in simulation time. In addition, the user interface provides detailed information on the individual simulation models. For example, this information includes performance information such as processor and storage space usage, as well as metadata such as author and version of the simulation model.

## 2.5 The DeepActor Framework

The DeepActor framework enhances the developer framework provided by DANUBIA (see Sect. 2.3.1) with additional base classes and interfaces. It implements a joint conceptual basis of the DeepActor approach developed in the context of GLOWA-Danube and enables the socio-economic simulation models from GLOWA-Danube (the *Actor* main component in Fig. 2.1) to explicitly model the decision processes of the so-called deep actors. The DeepActor approach substantiates the method for agent-based simulation from the social sciences (Gilbert and Troitzsch 2005), which in turn is based on the agent concepts from the field of (distributed) artificial intelligence (Norvig and Russell 2003; Weiss 1999). The term actor is used to avoid confusion with the term software agent that is related only conceptually. A software agent is a programme that has the properties to operate autonomously, responsively, proactively and integratively. These properties are interpreted in technical terms for software agents; for example, the autonomous property is interpreted as the internal assignment of threads or processes to each agent. Here, in contrast, these kinds of properties are interpreted in conceptual terms for actors and hence always depend on the modelling within a specific simulation model. An actor represents an entity acting within the simulation area, which may be the private household, farmer or tourism infra- and suprastructure facility, for example. Simulation models that use the DeepActor framework are called DeepActor models.

The fundamental modelling elements of a DeepActor model are shown in Fig. 2.5. The *AbstractModel* base class pre-specified for DANUBIA is refined by a special base class, *AbstractActorModel*. In addition, there are base classes for actors, plans and actions. The static structure from Fig. 2.5, and hence the structural concept of the DeepActor approach in GLOWA-Danube, is explained in more detail below. The dynamic aspects of the method are explained in Sects. 2.5.1 and 2.5.2.

It should be noted that in general, the abstract elements of the framework are optionally specified in a DeepActor model. Depending on the type and complexity of the simulation model to be implemented, there are simple, reactive actors and complex actors that can learn. Both are implemented using the same basis of the DeepActor framework.

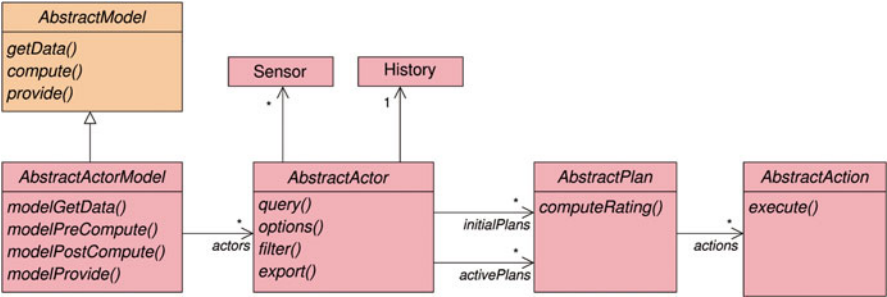


Fig. 2.5 Static structure of the DeepActor framework

In the simplest manifestation, the specified derivation of the *AbstractActorModel* base class is used for the administration and initialisation of the specific actors in a DeepActor model. The actors have sensors with which they perceive their surroundings, especially for the import of proxel data and for the import of data from other actors within the same model. Actors also have a history or memory, in which they store decision of previous simulation steps. Plans represent the behaviour options of an actor, and each plan contains a number of actions, which explicitly model the effects of a plan implementation. The number of initial plans (*initialPlans*) is the total number which is available to an actor over the simulation run. The number of active plans (*activePlans*) is the number of the initial plans that an actor has selected for implementation in a given time step.

### 2.5.1 Life Cycle of a DeepActor Model

The life cycle of a DeepActor model illustrated in Fig. 2.6 is a refined version of the cycle of a DANUBIA model (see Fig. 2.3).

The key difference is a distinction between the model and the actor calculations. The former represents the macro-level of the simulation, which is required at least for the coordinated exchange of data (*modelGetData*, *modelProvide*) with coupled simulation models, and the latter simulates the micro-level of individual decisions (*filter*, *options*), the effects of which are accounted for by the model calculations on the macro-level. The activities *getData*, *compute* and *provide* in Fig. 2.3 are implemented by the DeepActor framework using the abstract operations from Fig. 2.5. A specific DeepActor model supplies specific implementations for this as follows:

- *modelGetData* and *query*: the model imports data from coupled simulation models, and the actors sample the sensors.
- *modelPreCompute*: the model can initialise actor calculations.
- *options*: the actors activate the plans according to external conditions.
- *computeRating*: all active plans compute an evaluation attribute.
- *filter*: the number of active plans is reduced based on criteria specific to each actor.

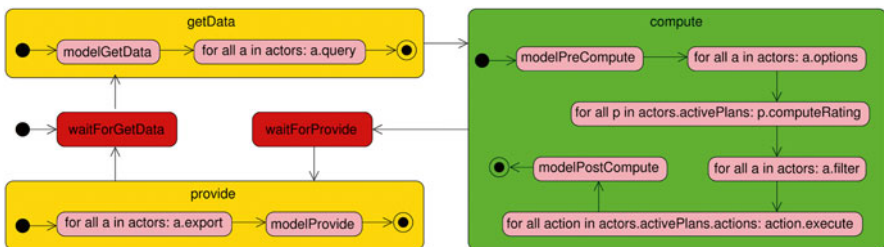


Fig. 2.6 Life cycle of a DeepActor model as refinement of Fig. 2.3



- *execute*: the actions of the remaining active plans are executed.
- *modelPostCompute*: the model can initialise actor calculations.
- *export* and *modelProvide*: the actors and the model export data for other actors and/or for coupled simulation models.

## 2.5.2 Decision Process of a Deep Actor

The decision by an actor comprises two steps: *options*, to determine the active number of plans in a given simulation step, and *filter*, to reduce the number of active plans to only those plans that actually should be executed. In the first step, plans are deactivated if, at the current simulation time point, they cannot be executed because of external conditions upon which the actor has no influence. In the second step, an actor considers his implicitly available goals and preferences and thereby determines the set of active plans to be executed in this time step. The calculation of an optional evaluation attribute (*computeRating*) happens in between. The resulting value can be used as the basis of the plan selection in the *filter* step. For example, in this way a selection algorithm based on a multiattribute utility theory (Norvig and Russell 2003) can be implemented.

Goals that motivate a decision by an actor are not an explicit element of the DeepActor method. Instead, the specific implementation of the *filter* step must account for the goals of each actor.

## References

- Barth M, Hennicker R, Kraus A, Ludwig M (2004) DANUBIA: an integrative simulation system for global change research in the Upper Danube Basin. *Cybernet Syst* 35:639–666
- Booch G, Rumbaugh J, Jacobson I (1999) The unified modeling language user guide. Object technology series. Addison Wesley, Reading
- Gilbert N, Troitzsch KG (2005) Simulation for the social scientist, 2nd edn. Open University Press, Berkshire
- Hennicker R, Ludwig M (2005) Property-driven development of a coordination model for distributed simulations. In: Proceedings of the 7th IFIP international conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS 2005), LNCS 3535. Springer, Berlin, pp 290–305
- Hennicker R, Ludwig M (2006) Design and implementation of a coordination model for distributed simulations. In: Mayr HC, Breu R (eds) Proc. Modellierung 2006 (MOD'06). Lecture notes in informatics, vol P-82. Gesellschaft für Informatik, Bonn
- Norvig P, Russell SJ (2003) Artificial intelligence: a modern approach. Prentice Hall, Englewood Cliffs
- Weiss G (ed) (1999) Multiagent systems: a modern approach to distributed artificial intelligence. The MIT Press, Cambridge

Regional Assessment of Global Change Impacts

The Project GLOWA-Danube

Mauser, W.; Prasch, M. (Eds.)

2016, XXI, 670 p., Hardcover

ISBN: 978-3-319-16750-3