

Agent-Based Social Simulation within a Generic Framework for Environmental Modelling*

Rolf Hennicker, Stephan Janisch, and Matthias Ludwig

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

Abstract. We present an approach for the support of agent-based social simulation (ABSS) models within a generic framework for computer-based environmental modelling. The framework allows for the coupling of simulation models from various sciences to perform integrative simulations. It is applicable to any kind of model which simulates spatially distributed environmental processes with an arbitrary, but discrete, time scale. Our support for ABSS is geared towards agents that decide over a given set of plans which represent their course of actions. However, since all of the framework elements are on the one hand optional, and on the other hand easily and arbitrarily extendable, there is in principle no restriction on its applicability. 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.

1 Introduction

Environmental simulation is concerned with the modelling of physical and societal aspects of our natural and social environment. During the last years it became apparent that it is important to take the interplay of different singular environmental processes within a multi-disciplinary approach to environmental modelling into account. By this means experts may focus on the modelling within their particular discipline and provide a simulation model that is dynamically integrated and coupled with the simulation models of other disciplines. In [1] we have described a generic framework that supports such a multi-disciplinary approach for both, natural and social science simulation models. While [1] focused on the common framework for any kind of simulation model, we aim here to provide a more detailed account on our support for *agent-based social simulation (ABSS)* models.

After a short summary of basic concepts for multi-disciplinary environmental modelling, we will discuss general precise requirements and the object-oriented design of our framework for ABSS models. We illustrate its applicability briefly. More detailed studies can be found in [2–5]. We use the Unified Modelling Language UML2 [6] for the description of static modelling aspects and assume that the reader is familiar with the notation and meaning of basic UML class diagram elements.

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

2 Multi-Disciplinary Environmental Modelling

Our framework for social simulation is embedded into a generic framework for multi-disciplinary environmental modelling [1] whose underlying concepts are depicted in Fig. 1 and briefly explained in the following.

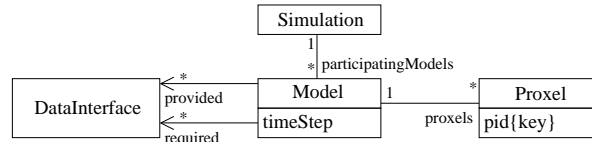


Fig. 1. Concepts of the simulation framework

In an *integrative simulation* (represented by the class *Simulation*) a number of *simulation models* (class *Model*) are participating which are coupled for continuous data exchange during a simulation run. Coupling of simulation models is facilitated in that each model defines appropriate interfaces (class *DataInterface*) either as provided or required interfaces for exported and imported data respectively. The simulation area is modelled by a set of geographical units, so called *proxels* (from *process pixel*, cf. [7]), represented by the class *Proxel*. Each simulation model may have its individual time step (attribute *timeStep*) depending on the simulated process. A coupled simulation model must obey the following general life cycle:

- *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

The mentioned concepts are realised in a component-based simulation framework. Due to a lack of space we do not elaborate on the component-based aspects of our implementation. The framework’s principles, however, may also be discussed within a pure object-oriented view to the framework design. A fundamental principle of the framework design is the separation of two layers, the *framework core* and the *developer interface*, as it is depicted by the framework excerpt in Fig. 2. While the framework core implements all features that can be handled by the framework itself like, e.g., the common properties of a simulation model (*ModelCore*), and the management of the spatial distribution (*ProxelTable*), the developer interface provides abstract base classes, e.g. *AbstractModel*, which have to be extended by model developers to implement a concrete simulation model.

A «plug-point» is a method that needs to be implemented within a concrete simulation model. The implementation of a plug-point is a «plug-in», illustrated later, in Sect. 5. A «query» is a method that can be used to access data that is managed by the core layer. For instance the query `proxel(pid:Integer)` of *AbstractModel* returns a *proxel* object that is managed by *ModelCore* using a *ProxelTable* in the core layer of the framework. Note that we do not show return types in our diagrams.

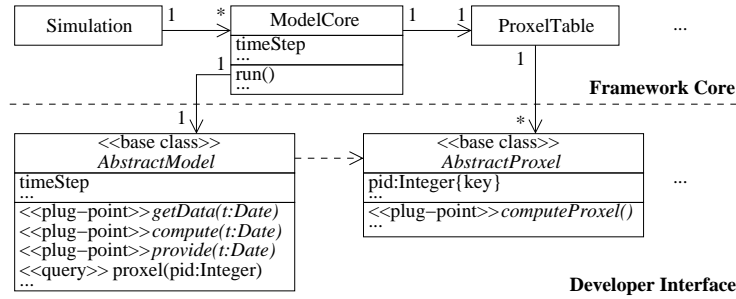


Fig. 2. Layered framework architecture

3 Requirements for Agent-Based Social Simulation Models

Within this section we identify requirements for a basic generic framework for ABSS models. At the core of any such model are agents, or *actors*, which represent deciding entities such as individuals (e.g. farmers, households, etc.) or organisations (e.g. water suppliers, companies). Actors are considered to be encapsulated entities that perceive their environment via sensors. The environment may define spatial aspects such as a geographic locations but also societal aspects such as social networks.

We take *deciding* to mean the selection of plans that represent courses of action of an actor. In general, the complexity of decision-making varies considerably for different simulation models. Some models use simple reactive agents which map their perceptions directly to certain plans or actions, while other models use actors with more elaborated features such as learning capabilities in order to adapt their decisions in a more flexible way to varying environmental conditions. In any case, however, plan selection results in the execution of associated actions, modelling the impact of the execution of the selected plan for the actor’s state but also for the environment of the actor. Finally, the environment of an actor should be managed by a “surrounding” *model* which coordinates the computations of a given set of actors.

Summarising the mentioned features, we obtain the following requirements for a basic common architecture of agent-based social simulation models:

1. Actors represent decision-making entities (situated, part of a social network)
2. Actors perceive their environment using sensors
3. Actors are equipped with a set of plans and corresponding actions
4. Actors are aware of a history to remember decisions
5. Actor decisions are embedded into the computation of a model

In the following we elaborate on each of the five requirements using, besides UML class diagrams for the description of concepts and their interrelation, informal invariants to specify constraints with regard to the respective concept.

Actor Model and Actors. An actor model is a simulation model that additionally refers to a set of actors; cf. Fig. 3. It is important to distinguish between *actor types* and actor instances, *actors* for short. The actor type determines the properties of the actor via the availability of certain attributes, while an actor instance provides certain values for these

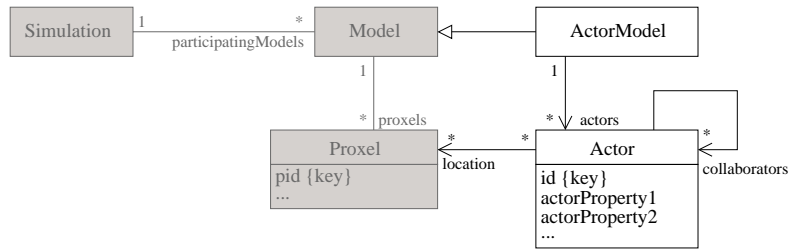


Fig. 3. Actors are managed by a model

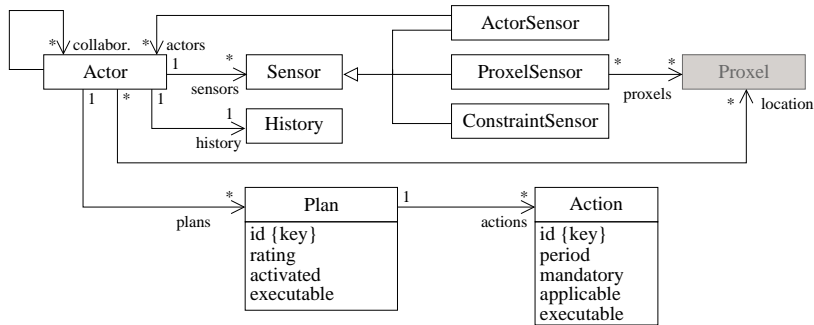


Fig. 4. Actors and their relation to sensors, plans, actions and history

properties. For example, an actor type Company defines a property name, to which in each actor instance (of this type) a certain company name (like, e.g., Siemens, BMW, etc.) is assigned. Similarly, different actor types may show different decision-making capabilities.

An actor is situated within the simulation area according to its location. We reuse the proxel-based modelling of the simulation area as given by the general framework (grey in succeeding figures); cf. Sect. 2. Thus the location of an actor is given by a set of (not necessarily connected) proxels. This allows, for instance, the modelling of a company whose sites are distributed within the given simulation area. An actor may be part of a social network which is defined by its collaborators. We do not (yet) support interaction between actors of different actor models and thus require the collaborators to be actors from the same actor model. However, by the coupling mechanism inherited from Model of the general framework (cf. Sect. 2), an (indirect) interaction between actors of different models is facilitated.

Actors have a unique ID and arbitrary type specific properties. The complete set of actors, their locations and their potential sets of collaborators are required to be known at the beginning of a simulation and to remain constant during the simulation run. Note, however, that the values of the actors' properties may of course change dynamically during a simulation run.

Invariants for location and collaborators

- The location of an actor is part of the simulation area of the particular model.
- The collaborators of an actor are a subset of the actors of the particular model.

Invariants can be described more formally and therefore less ambiguously using the Object Constraint Language OCL (cf. [8]). However, for a lack of space, we omit OCL constraints and continue with an informal description of invariants.

Sensors and History. Actors perceive their environment explicitly via their sensors (cf. Fig. 4). Sensors allow for the reception of data or events. The modelling with events provides a means to realise, for instance, thresholds that trigger a certain decision of an actor as a first-class citizen of the given model. We distinguish between three concrete kinds of sensors: A proxel sensor for the simulation area, an actor sensor for information exchange with other actors of the same model and a constraint sensor. Constraints are an explicit concept for the modelling of further influencing aspects such as legal constraints, for instance.

Invariants for sensors

- The proxel sensor refers only to proxels occurring in the location of the actor
- The actor sensor refers only to actors belonging to the collaborators of the actor

Besides sensors, actors dispose of a History that allows to “remember” the plan execution status of previous time steps. By this means a basic mechanism for actors with learning capabilities is provided.

Plans and Actions. Based on their local state, their history and their perceived environmental state, actors decide which plan is to be selected for execution within a given time step. A plan, in turn, is related to a set of actions which model the concrete impact of plan execution; cf. Fig. 4. In this vein, plans represent the possible courses of action of an actor. The conceptual difference between types and instances of plans and actions is analogous to actor types and actor instances.

The actions of a plan are to be executed if the respective actor decided for the particular plan (activated), and the plan is executable. The decision to activate a plan might be based on its rating which allows to realise multi-attribute utility based decision algorithms conveniently. A plan is executable only if all of its mandatory actions are executable in the current time step – an action is either mandatory or optional, which is an initially assigned property of an action. The executability of an action depends on two type-specific properties that are evaluated dynamically within each time step: first, the action needs to be applicable and, second, the period of the action needs to be consistent with the current simulation time. For instance, harvesting is an action of a farming actor that is relevant only during certain months of a year.

Both, the plans of an actor and the actions of a plan are required to be initially known and must not change during a simulation run. Additionally, and analogous to actors, plans and actions have a unique identifier.

Invariants for the executability of plans and actions

- A plan is executable if all of its mandatory actions are executable
- An action is executable if it is applicable and the current simulation time lies within the period of that action

Actor Life Cycle and Embedding. The computation of an actor arranges the dynamic aspects of the concepts introduced so far. After an initialisation which assigns basic

attributes such as an ID, a location, etc., an actor queries the environment using its sensors. The decision is then split into two steps; first available options, represented by plans, are evaluated and second, these options are filtered to those plans which indeed are currently executable. After that the history will be updated and the results of the decision are made available for different models. The actor model integrates these computations in a fixed order such that the complete life cycle is as follows:

- actors export initial data
- provide initial data at the model’s provided interfaces
- while not at simulation end
 - get data from the model’s required interfaces
 - actors query sensors
 - prepare actors’ computations (preCompute)
 - actors evaluate their options
 - new rating values for plans are computed
 - actors activate plans that should be executed (filter)
 - actions of activated and executable plans are executed
 - post-process actors’ computations (postCompute)
 - actors update history and export newly computed data
 - provide newly computed data at the model’s provided interface

The computation of actors is completely embedded into the computation of the model. Therefore the life cycle of an actor model can be considered to be a refinement of the life cycle of general simulation models as introduced in Sect. 2.

4 Design and Implementation

As mentioned in Sect. 2, the distinction of core and base elements is fundamental to our approach. Base elements are used to define a developer interface that shows only those elements that are indeed relevant for concrete model development. Core elements are completely hidden and thus do not burden the developer with irrelevant details of upper layers. The layered framework architecture in Fig. 5 shows an excerpt of the object-oriented design of our framework, again illustrating this aspect. The middle layer (*Actor Framework Core*) integrates an extended developer interface for actor models (*Actor Developer Interface*) with the developer interface for general simulation models. Note that the actor core layer does not refer to the core layer of the general framework in the upper part of Fig. 2. The actor framework core part is realised like a simulation model extending abstract base classes and implementing plug-points. The plug-ins implemented by *ActorModelCore* delegate to the corresponding operations of a new abstract base class *AbstractActorModel* for actor models. The compute step is split into a pre- and post-compute allowing for the execution of actors, plans and actions in between.

The developer interface of our framework for ABSS is obtained from Fig. 5 by hiding the core layer. By this means, model development is completely decoupled from framework functionality, leaving only those elements visible that are indeed relevant for model development.

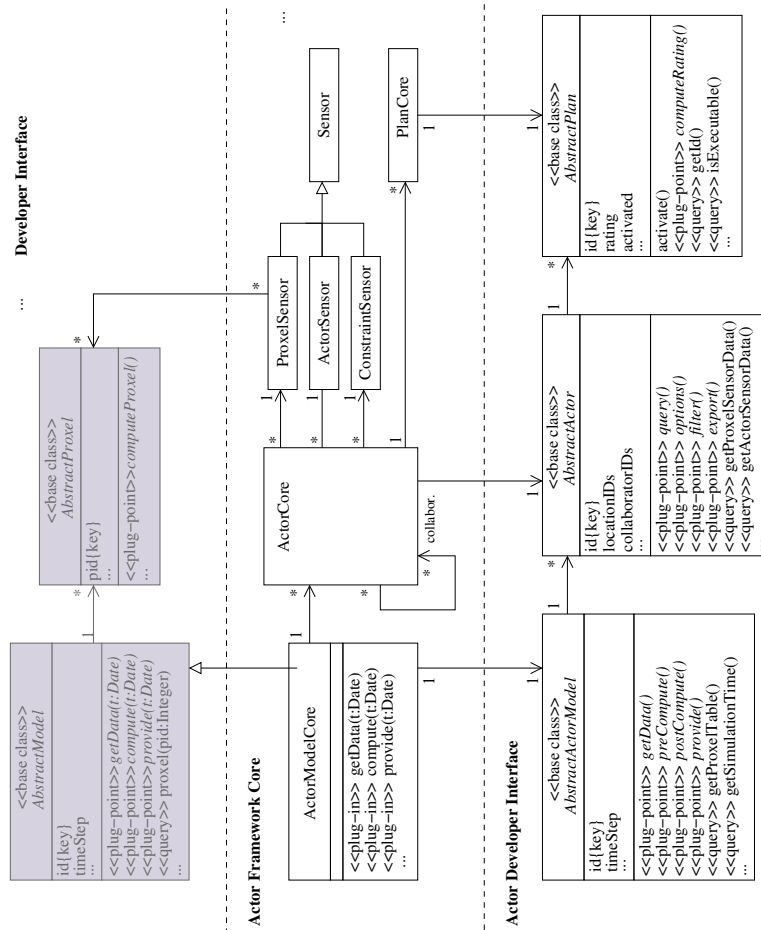


Fig. 5. Layered framework design

5 Application

The application of our framework is analogous to the application of the general framework as discussed in [1]. The difference lies in an extended set of base classes. The concrete model `WaterSupply` depicted in the example of Fig. 6 uses two actor types `CommunityWSC` and `RegionalWSC`. The class `ExpandCapacities` is an example for a concrete plan type. The classes implement plug-ins that may vary among different subtypes of the same base class such as `CommunityWSC` and `RegionalWSC`. Different actor types may use different sensors within query or different concrete decision procedures within options and filter, etc. Similarly, `computeRating` is usually implemented differently for different types of plans. Besides varying plug-ins, different types may also show different values of the inherited attributes. For instance, `timeStep` may have assigned a different value within different simulation models.

The mentioned principles apply to the remaining core and base classes analogously. We would like to stress that all of the provided modelling features are optional and all

Actor Developer Interface

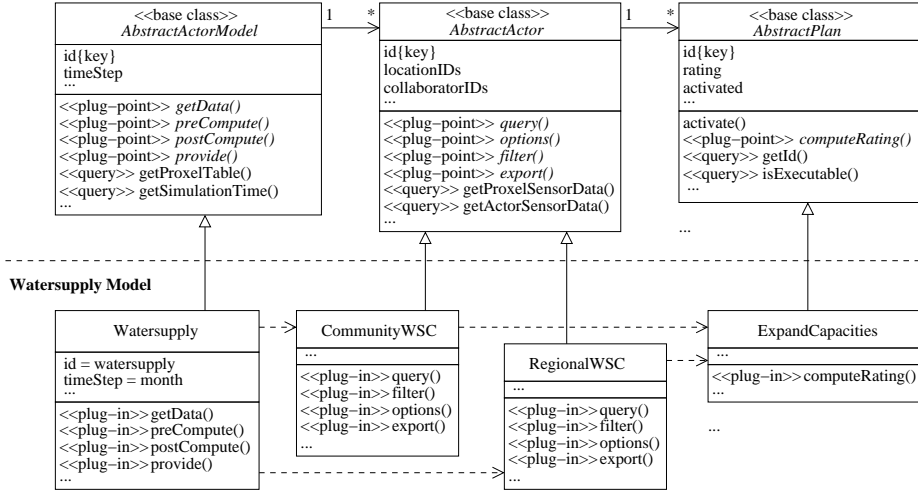


Fig. 6. Framework application

of them are arbitrary extendable. For instance, even though the location of an actor is initially assigned to be a fixed part of the simulation area it is possible to model actors that move dynamically within that given part. Such a feature is merely not available on the common framework level but there is nothing which disallows to implement additional structures and mechanisms on the concrete model level.

The concrete application of the framework is illustrated within the project GLOWA-Danube [9]. Here, five actor models have been realised with varying numbers of actor types and objects: Watersupply (2 actor types, 1717 actor objects), Household (5, 46050), Tourism (9, 5065), Farming (28, 58984), Economy (1, 1354); cf. [10]. Published examples of (integrated) models and their development are to be found in [2–5].

6 Conclusions

We have discussed the requirements, design and implementation of a framework for agent-based social simulation models. The framework is embedded into an existing generic framework for multi-disciplinary environmental simulation that provides an elaborated and established model of simulation time, space and data exchange in the context of dynamically coupled environmental simulation models. This integration is one of the important distinguishing factors which sets our framework apart from existing toolkits for multi-agent simulations such as Repast [11], MASON [12] or Swarm [13]. Even though these toolkits usually provide more comprehensive support for the development of single simulation models, none of them provides a framework for dynamically coupled simulation which is coordinated with regard to a common simulation area and a global simulation time.

An interesting issue for future work could be an extension to allow for more direct interaction between actors. Our current approach focuses on a strict protocol of getting

data, computation and exporting data, thus disallowing bidirectional actor communication within the same time step of a simulation run. Allowing to depart from such a protocol, however, paves the way for potentially diverging communications which then endangers the progress of a simulation run. A naive approach to integrate direct interaction would lose such important guarantees concerning the execution of simulation runs. For this reason, more involved mechanisms, for instance to guarantee the termination of distributed decision making, are required. In the same vein it might be advantageous to allow for direct actor interaction between different actor models.

Acknowledgements. We would like to thank our colleagues within GLOWA-Danube for their cooperation and in particular Andreas Ernst, Roland Barthel and Wolfram Mauser for discussions on concepts and application of the actor framework.

References

1. Bauer, S., Hennicker, R., Janisch, S., Ludwig, M.: A Generic Framework for Multi-Disciplinary Environmental Modelling. In: International Congress on Environmental Modelling and Software 2010. (2010) Accepted.
2. Barthel, R., Janisch, S., Schwarz, N., Trifkovic, A., Nickel, D., Schulz, C., Mauser, W.: An integrated modelling framework for simulating regional-scale actor responses to global change in the water domain. *Environmental Modelling & Software* **23**(9) (Sep 2008) 1095–1121
3. Barthel, R., Janisch, S., Nickel, D., Trifkovic, A., Hörhan, T.: Using the multiactor-approach in glowa-danube to simulate decisions for the water supply sector under conditions of global climate change. *Water Resources Management* **24**(2) (Jan 2010) 239–275
4. Ernst, A., Schulz, C., Schwarz, N., Janisch, S.: 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)
5. Schwarz, N., Ernst, A.: Agent-based modeling of the diffusion of environmental innovations – an empirical approach. *Technol Forecast Soc* **76**(4) (2009) 497–511
6. Rumbaugh, J., Jacobson, I., Booch, G.: *The Unified Modeling Language Reference Manual*, 2nd edition. Pearson Education, Inc (2005)
7. Tenhunen, J.D., Kabat, P., eds.: *Integrating Hydrology, Ecosystem Dynamics, and Biogeochemistry in Complex Landscapes*. Wiley, Chichester (1999)
8. Warmer, J., Kleppe, A.: *The Object Constraint Language – Second Edition*. Addison-Wesley (2003)
9. Ludwig, R., Mauser, W., Niemeyer, S., Colgan, A., Stolz, R., Escher-Vetter, H., Kuhn, M., Reichstein, M., Tenhunen, J., Kraus, A., Ludwig, M., Barth, M., Hennicker, R.: 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** (2003) 621–634
10. Janisch, S., Kuhn, S., Barthel, R., Hennicker, R., Mauser, W., Ernst, A.: DeepActor – Framework and Modelle in Danubia. Nationale GLOWA-Konferenz Potsdam (Okt 2009)
11. North, M.J., Collier, N.T., Vos, J.R.: Experiences creating three implementations of the repast agent modeling toolkit. *ACM Trans. Model. Comput. Simul.* **16**(1) (2006) 1–25
12. Luke, S., Cioffi-Revilla, C., Panait, L., K, K.S.: Mason: A new multi-agent simulation toolkit. In: *Proceedings of the 2004 SwarmFest Workshop*. (2004)
13. Minar, N., Burkhart, R., Langton, C., Askenazi, M.: *The swarm simulation system: A toolkit for building multi-agent simulations* (1996)