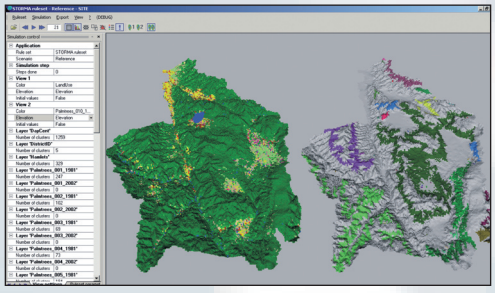
Matthias Mimler and Joerg A. Priess

Design and implementation of a generic modeling framework – a platform for integrated land use modeling

Center for Environmental Systems Research

CESR-PAPER 2



kassel university press

UNIKASSEL VERSITÄT

CESR – Paper 2

Center for Environmental
System Research

Matthias Mimler, Joerg A. Priess

Design and implementation of a generic modeling framework - a platform for integrated land use modeling

kassel university press

UNIKASSEL VERSITÄT

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar

ISBN 978-3-89958-467-7
URN: urn:nbn:de:0002-4675

Druck und Verarbeitung: Unidruckerei der Universität Kassel
Printed in Germany

Contents

1. Introduction

Land-use as "the total of arrangement, activities and inputs that people undertake in a certain land cover type", in contrast to land-cover being the "observed physical and biological cover of the earth's land, as vegetation or man-made features" (FAO, 1999), is a crucial link between human activities and the natural environment and one of the main driving forces of global environmental change (Lambin et al., 2000). Large parts of the terrestrial land surface are used for agriculture, forestry, settlements and infrastructure. Concerns about land-use and land-cover change first emerged on the agenda of global environmental change research several decades ago when the research community became aware that land-surface processes influence climate (Lambin et al., 2006). While the focus in the beginning lay on the surface-atmosphere energy exchanges determined by modified surface albedo (Ottermann, 1974; Charney and Stone, 1975; Sagan et al., 1979), the view later on shifted to terrestrial ecosystems acting as sources and sinks of carbon (Woodwell et al., 1983; Houghton et al., 1985). A broader range of impacts of land-use change on ecosystems was identified since then. Besides being a major influencing factor on climate (Brovkin et al., 1999), land-use meanwhile is regarded the most important factor influencing both biodiversity and biogeochemical cycles on the global scale (Sala et al., 2000). To close the circle, land-use itself is strongly influenced by environmental conditions like climate and soil quality, affecting e.g. suitability for certain crop types and thus affecting agricultural use or biomass production (Mendelsohn and Dinar, 1999; Wolf et al., 2003).

Given the importance of land-use, it is essential to understand the interactions between the multitude of influential factors and resulting land use patterns. An essential methodology to study and quantify such interactions is provided by the adoption of land-use models. With land-use models it is possible to analyze the complex structure of linkages and feedbacks and to determine the relevance of driving forces (Heistermann et al., 2006). Land-use models are used to project how much land is used where and for what purpose considering different boundary conditions. After several years of research, land-use modeling has become an important technique for the projection of alternative pathways into the future, for conducting experiments that test our understanding of key processes, and for describing these processes quantitatively (Lambin et al., 2000). Since land-use change models represent part of the complexity of land-use systems, they offer the possibility to test the sensitivity of land-use patterns to changes in selected variables. Through scenario building, they additionally allow testing of the stability of linked social and ecological systems (Veldkamp and Lambin, 2001).

In the past years a multitude of land-use models have been developed addressing different applications, regions and scales. According to Lambin et al. (2000) land-use change modeling has to address at least one of the following three problems. The first problem is to identify the environmental or socio-economic variables that actually cause land-use changes. The second problem is to find out at which location land-use change does occur. Besides using spatially explicit representation, this question can also be answered using economic location theory (Irwin and Geoghegan, 2001). The third target is to answer to what amount land-use changes do occur.

Implementing a model technically implies to formulate it in a set of computer instructions. Two complementary possibilities to achieve this are to either reuse and adapt an existing model that resembles the actual modeling task, or by programming a new model. It is obvious that both ways have several advantages and drawbacks. Reusing an existing model definitely saves cost and time. However, it is likely that a model created for a different though potentially similar task needs to be adapted to a certain degree. This might not always be possible to achieve in a satisfying manner as too many compromises might need to be accepted. Programming a new model by using a general-purpose programming language (e.g. C++, Fortran) will definitely result in a tailor-made application, but this approach is usually cost- and time-intensive depending on the complexity of the desired application. In addition one will usually be able to identify certain structure in the model, that can be equally used in other applications. For cellular automata (CA) models, e.g., the underlying cell lattice is such a generic structure, which only differs among models in its size and the spatial resolution it represents. The actual differences of CA models lie in their transition rules and constraints.

Another complication is caused by the fact that landscape ecologists, planners and modelers, i.e. the persons actually developing models, are not necessarily programmers. To provide efficient model development and model adoption conditions, model developers should be able to transform their conceptual models into computer simulations without having to implement them in a general-purpose computer language or at least without having to deal with specific details of software development, e.g. memory management. This conflict has been recognized by the modeling community in the past years and a number of possible solutions, in the following termed modeling environments, have been proposed to "untangle the beauty of a model from the beast of its implementation on a computer"(Fall and Fall, 2001).

Fig. 1.1 shows several examples of existing modeling environments along a spectrum of specificity. On the left lie general purpose programming languages, opposed to complete specific models on the right edge whose possibilities for adaption are restricted to a number of adjustable parameters. Along that gradient, a number of solutions is listed. Examples on the left side are program-level support tools. Simscript and DEVS (Clark, 1992; De Vasconcelos and Zeigler, 1993) are extensions to programming languages providing modeling functionality. EcoSim (Lorek and Sonnenschein, 1998) is a software library that reduces complexity by providing modeling functionality, thus enabling model developers to concentrate on writing high-level code, i.e. rule set and model parameterization. A similar concept is pursued for multi-agent simulation by Swarm (Minar et al., 1996), which additionally provides a set of graphical tools for display and analysis of modeling results. The use of the latter solutions still requires programming knowledge. They can reduce development time and cost, but result in specialized modeling applications.

In contrast to program-level support tools, model-level support tools can assist model construction without requiring the creation of source code (Fall and Fall, 2001). FORSUM (Frelich and Lorimer, 1991) and STORM (Krauchi, 1995) are applications that implement a template for a set of similar modeling applications. Portability to other applications is enabled by providing an extensive set of parameters for which values or definitions can be altered. Flexibility is increased with approaches like LANDIS (Mladenoff et al., 1996) and TELSA (Klenner et al., 1997) that provide extended features like user-definable spatial layers or variables.

Largest flexibility combined with minimum programming requirements is given by tools that provide model-level support for classes of models rather than individual model types. Examples can be found in the middle of the spectrum depicted by Fig. 1.1. Different approaches are used

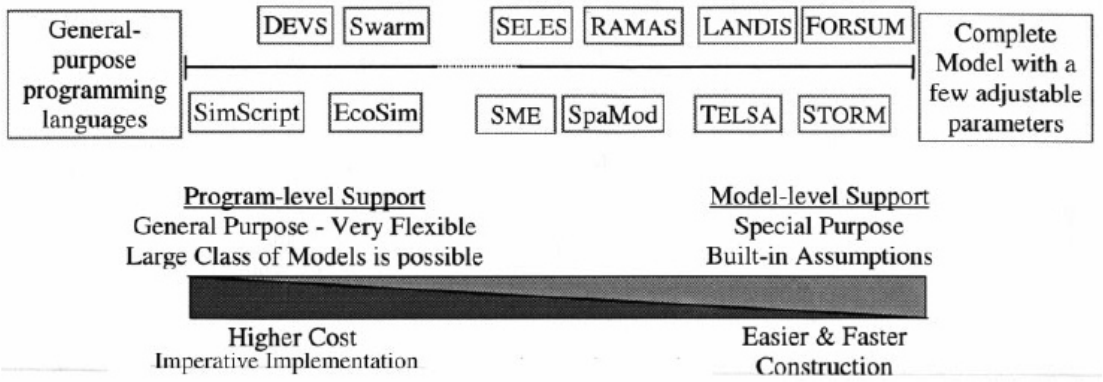


Figure 1.1.: Spectrum of approaches to implementing spatial landscape models (Fall and Fall, 2001).

to achieve this target. SpaMod (Gao, 1996), for example, enables high-level formulation of mathematical models through differential equations which are automatically translated into C code and subsequently linked to the system run-time environment, thus creating a simulation tool. SME (Maxwell and Costanza, 1997) uses a similar technique as it provides a high-level language for the specification of modular spatio-temporal models that is later translated into C++ code and linked to the SME run-time environment. STELLA (Costanza et al., 1998), suitable for non-spatial simulations, allows to freely design a model via building blocks on a graphical user interface. SELES (Fall and Fall, 2001) supports spatially explicit, CA-based land-use modeling applications by providing the necessary generic infrastructure, in particular the simulation grid. For the formulation of application rule sets, Fall and Fall (2001) developed a domain-specific language. This language enables a rule set developer to concentrate on the actual modeling task and fulfills a number of specific criteria like simplicity, flexibility capability modularity, transparency, efficiency, reusability, adaptability and communicability.

The increased use of models in ecological sciences, the advent of modeling environments and frameworks and the trend toward integrated systems of models from different disciplines imply increasing complexity regarding the implementation as software. A number of concepts for handling software complexity is provided from computer science. In particular, modularization, object-oriented design and encapsulation of models by means of components with clearly defined interfaces are of specific interest (Argent, 2004). Modularization and object-oriented design contribute to an improved quality of the implementation of single models. The creation of model components can help to facilitate model coupling and assembling of integrated systems. The advantages of a component-based design are numerous. For example, making changes in the implementation of one particular model will not affect other parts of an integrated system due to encapsulation. Models responsible for specific processes in the integrated system can simply be replaced by other models, provided they have the same interface.

In the past years, studies on synergies between environmental modeling and computer science have been conducted and object- and component-oriented tools and designs have entered the field of land-use modeling (Maxwell and Costanza, 1997; Fall and Fall, 2001). Villa and Costanza (2000) adopted a component-based software architecture for their Simulation Network Interface (SNI), emphasizing the advantages of encapsulated complexity, exchangeability and reusability of single models. In a similar manner, the DEVS framework was advanced

to enable the definition of components (Filippi and Bisgambiglia, 2004). In a study focusing on forest landscape modeling, He et al. (2002) experienced that the use of component-based model integration supports scientists with few expertise in computer science in building complex, but still manageable applications. In the future, integrated models could be constructed from models implemented as components in a so called "plug and play" approach, where a model component such as a water balance algorithm could simply be replaced by an alternative component (Argent, 2004). Modern tools that support the development of component-based architectures like Microsoft's .Net offer information about components using meta data and introspection. ICMS (Interactive Component Modeling System) (Rahman et al., 2004) takes advantage of these features to improve usability of modeling frameworks by generating self-documenting components and custom meta data. In the combination with other advantages of modern programming platforms (e.g. multi-language development, web enabled models), Rahman et al. (2004) see strong impulses for model developers and uses.

In this paper, we present the design, development and implementation of a new framework supporting spatially explicit land-use modeling, the SITE (Simulation of Terrestrial Environments) framework. It resembles the SELES framework (Fall and Fall, 2001) in that it provides basic data structures like the simulation grid and a framework managing cell attributes. However, we discovered, that proprietary domain-specific languages are indeed simple, but still do not provide enough flexibility to handle certain modeling tasks with complex rule sets. In such cases, the possibilities of a full-fledged programming language might be required. Modern scripting languages are able to bridge this gap of providing a maximum of flexibility and at the same time being simple enough to be handled by non-programmers. Scripting languages are typically run by interpreters and thus can be integrated in other applications such as modeling frameworks. In addition, it is possible to write language extensions with which it is possible to introduce data structures specific to land-use modeling (e.g. simulation grid, cells, attributes). Using this strategy, on rather extends a full-fledged programming language to a domain-specific language instead of taking the effort to create a new one. SITE defines an extended version of the widely used Python scripting language for the formulation of transition rules.

Compared to most existing land-use modeling environments, SITE functionality does not confine itself to managing the execution of land-use models and respective simulation runs. It furthermore integrates tools that are crucial for modeling into its generic framework. Among them are a component for model testing (via various map-comparison algorithms), a component for model calibration using optimization heuristics (e.g. genetic algorithms) and functionality enabling the interactive handling of scenarios. In the following paragraphs, the design and implementation of the SITE modeling framework will be presented.

2. Requirements

Land-use dynamics are driven by a variety of factors, biophysical as well as socio-economic. Therefore, comprehensive research on land-use dynamics needs to be organized in interdisciplinary projects. Figure 2.1 shows the structure of the Global Land Project (GLP, 2005), which is a long-term research framework for land systems and a structural template for regional projects. Figure 2.1 delineates the large variety of interactions between the earth system, terrestrial subsystems and land use. A more detailed view of interactions between the socio-economic and the biophysical subsystem and their influence on land-use is provided by Figure 2.2. It is the challenge of integrated land-use modeling to include the insights of different sectoral views and disciplines and their respective interactions for the simulation and assessment of land-use changes.

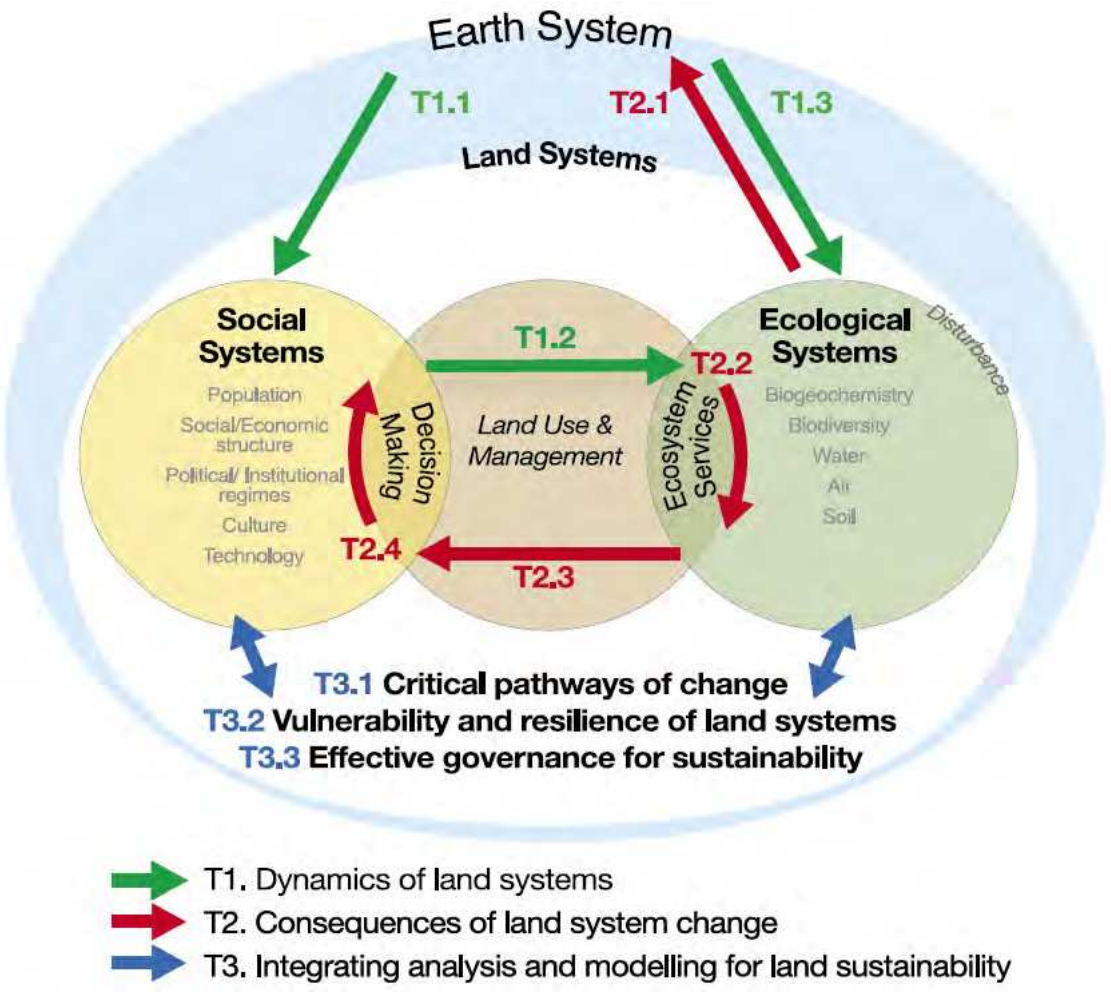


Figure 2.1.: Analytical structure of the interdisciplinary Global Land Project (GLP, 2005).

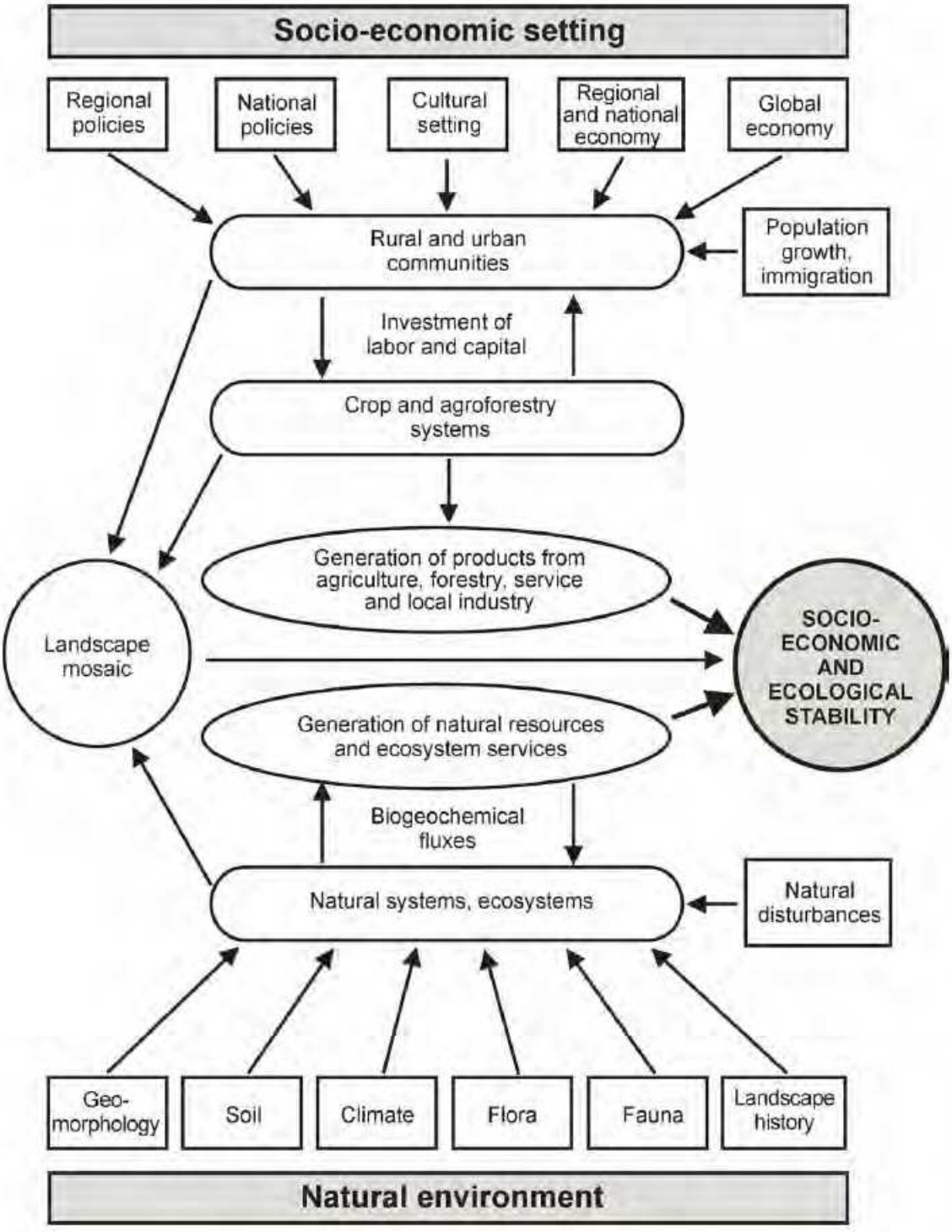


Figure 2.2.: Interactions between socio-economic (upper half) and natural (lower half) systems. The landscape mosaic is strongly determined by both domains and is a major influencing factor for socio-economic and ecological stability (STORMA, 2003).

According to that, the main target in the development of SITE was to create a land-use modeling framework capable of integrating scientific results of the STORMA[1] research project. STORMA is an interdisciplinary long-term project funded by the German Research Foundation. The target is to analyze the stability of rainforest margins in a research area in Central Sulawesi, Indonesia (STORMA, 2003). The proposed modeling outline for STORMA is guided by a similar scheme as the Global Land Project. The nature of SITE, being an integrative tool in the context of STORMA and potentially other similar research projects, directly implies a number of requirements with respect to the system design. In turn, certain technical demands result from the scientific requirements. In the following, these requirements and technical issues will be identified and defined.

Integrated modeling

The requirement of integrated modeling capability was implied directly by the definition of SITE, being a framework for modeling land-use changes in an interdisciplinary research context. Along with generic applicability, it can be considered the most important demand.

Integrated modeling is already established in the land use modeling community (Alcamo et al., 1998; Voinov et al., 1999; White and Engelen, 1997, 2000; Oxley et al., 2004; Van Delden et al., 2007). When reviewing literature on regional land-use modeling frameworks, however, it is noticeable that there are approaches that either support the implementation of models (Gao, 1996; Fall and Fall, 2001) or ones that enable integration or combination of existing models (He et al., 2002; Filippi and Bisgambiglia, 2004; Argent, 2004).

Following project requirements, the framework needed to enable both model development on the application side and interfacing capabilities to existing models. The only comprehensive solution for both providing a generic modeling platform and integrated modeling is GEONAMICA (Engelen et al., 1999; Oxley et al., 2004; Van Delden et al., 2007), which, however, is a commercial product. In addition, model integration in the context of the STORMA project additionally required high flexibility concerning feed back mechanisms from sub models to the actual land-use model as a basis for further decision making (Priess et al., 2007). Consequently, interfaces that allow model integration in SITE had to include functionality to feed back results to the calling instance. To handle possible performance problems regarding runtime that can arise from model coupling, the SITE interface was designed to enable parallel processing if allowed by the modeling methodology.

Generic platform

The definition of SITE as a framework for land-use modeling implies that it needs to provide a generic platform for operating land-use modeling applications. As a generic platform, the use of SITE is not restricted exclusively to STORMA. The framework is also suitable for other similar research projects with an interdisciplinary outline.

A generic modeling framework is mainly characterized by the separation of the actual modeling application (rule set specification) from the implementation of structures that are shared by all potential models that can be operated within the framework. For spatially explicit models, these structures basically are the simulation grid, the single grid cells and structures for the handling of cell attributes. Model-specific values, like the grid dimensions as well as concrete

[1] SFB 552 "Stability of rainforest margins in Indonesia" (STORMA, 2003)

attributes and attribute values characterizing grid cells are in turn specified by the modeling application. Connection to the data structures supplied by the framework is established via a specific interface. For the implementation of the model itself, a variety of solutions are imaginable. Costanza et al. (1998) used a graphical front end for the definition of model semantics. Another solution is the introduction of an application specific programming or description language (Fall and Fall, 2001). With GEONAMICA (Engelen et al., 1999), also a commercial product is available that supports both model definition and model integration. In the SITE framework, a scripting language, the functionality of which the functionality was specifically extended to match land-use modeling demands, was chosen for the task of implementing modeling applications. The main advantage of a framework is that model developers can concentrate entirely on their modeling task, ignoring implementation details. Thus, a faster and more efficient formulation of land-use models is possible. This is of specific interest for interdisciplinary working environments, in which model demands are likely to be altered frequently throughout the communication process among different parties. It also facilitates the execution of further case studies and the development of model prototypes since model code can be rapidly altered without having to consider any side effects on implementation details.

Integration of calibration methodology

Modeling practice reveals the necessity for model calibration. This field is widely discussed among the scientific community (Boumans et al., 2001; Oliva, 2003; Straatman et al., 2004). Despite its importance, there is no framework available yet, that implements calibration as an integral part. As a novelty, the goal of allowing rapid and effective development of land-use models and model integration was extended to also include model calibration. The design of the system even allows for a simultaneous calibration of different component models, which are both influencing land allocation (e.g. the land-use model and the biophysical sub model). Generic model calibration in SITE is based on parameter optimization with respect to an objective function. Consequently, beside optimization algorithms or heuristics, additional methods serving as objective functions needed to be integrated. The SITE calibration procedures use map comparison algorithms as objective function. The quality of simulation and calibration results is benchmarked based on reference maps.

Integration of scenario handling

As simulations of land-use dynamics are generally conducted under specific scenario assumptions, SITE was required to provide functionality to handle scenario information. Like calibration functionality, the handling of scenarios is not explicitly addressed by available modeling frameworks. SITE was designed to explicitly represent the handling of quantified representations of scenarios in its implementation. In addition, the scenario implementation supports interactive use of scenarios, during which simulations can be stopped to examine if simulation targets have been achieved. Based on this intermediate analysis, scientists can alter the underlying scenario (e.g. by adjusting management parameters), thus simulating the interaction of policy makers. This method was proposed by Alcamo et al. (2006) to overcome one of the major limitations of the current scenario methodology and is currently implemented in no other modeling framework.

Usability and cummunicability

As an integrative tool, SITE was required to be able to transport insights back to all other parties involved in the respective research project. To increase the acceptance among participating scientists, especially of those with backgrounds which are not easily related to computer science, it was crucial that the modeling system was designed for simple handling and transparent delineation of the modeling concept. Moreover, it needed to be capable of communicating its modeling and simulation results. In recent approaches, model developers became increasingly aware of this fact and introduced different solutions ranging from display of the simulation grid (Fall and Fall, 2001) to graphical tools for editing and viewing the model structure (Costanza et al., 1998; Filippi and Bisgambiglia, 2004). For SITE, a detailed graphical user interface (GUI) was implemented that facilitates model operation and understanding by e.g. providing 3-dimensional multiple views of the simulation grid.

Besides model operation, also model development, which usually involves programming, can be simplified, thus increasing usability by enlarging the potential user community. Graphical model builders (Costanza et al., 1998; Filippi and Bisgambiglia, 2004) are attractive but potentially inflexible approaches to this task, since they typically force specific modeling methodology. For SITE, we took advantage of the large potential that lies in modern scripting languages, which are powerful but nonetheless relatively simple and extendable programming tools.

As there is practically no restriction to the number of model parameters and settings for SITE applications, it is likely that single simulation runs cannot be reproduced, as the selected parameter values for an earlier simulation are not saved persistently. This situation conflicts with the requirement of communicability of modeling results. To overcome this limitation of existing frameworks, this feature was integrated in the SITE usability concept.

Expandability

Although expandability is a general requirement of state-of-the-art software development, it was of specific interest for the development of SITE, as it strongly supports long-term usability of the framework, which is of particular importance in projects with an envisaged duration of 12 to 15 years. One particular benefit from creating expandable software lies in facilitating the integration of new features, which keeps the system open for new developments and allows long-term use of the system. This way, a system is also prepared to effectively handle short-term demands. In the context of the SITE framework, two aspects were of special importance. The first was the established understanding of expandability by means of keeping the basic system open for the integration of further functionality. This goal could mainly be achieved by the adoption of modern software development concepts, such as component design, implementation by object oriented programming and the heavy use of design patterns. The second aspect addressed the interface between generic system implementation and application (i.e. simulation rule sets).

Summary

The listed requirements were derived from the demands implied by the application of the framework in interdisciplinary projects. They revealed, that such a land-use modeling framework had to be a generic platform for operating land-use models. The system had to be capable of

Table 2.1.: Comparison of SITE to selected spatial modeling frameworks with respect to the requirements identified for a generic framework for integrated land-use modeling.

	SELES (Fall and Fall, 2001)	**DEVS/JDEVS** (Filippi and Bisgambiglia, 2004)	**SimuMap** (Pullar, 2004)	**Eclpss** (Wenderholm, 2005)	**GEONAMICA** (Engelen et al., 1999)	**SITE** (this study)
Purpose	Land-use modeling	Environmental modeling	Spatial modeling of environmental processes	Grid-based ecosystem modeling	Dynamic land-use modeling and spatial decision support	Integrated land-use modeling
Integrated modeling	No	Yes	No	Yes, capable of parallel processing	Yes	Yes, capable of parallel processing
Generic land-use modeling platform	Yes (using a domain-specific language)	No	Yes (proprietary language MapScript, manipulates rasters based on map algebra)	Yes	Yes (CA-based)	Yes (grid-based, using an extended scripting language)
Integrated calibration	No	No	No	No	No	Yes
Scenario handling	Implicit[a]	Implicit (experimental frames)	No	No	Yes	Explicit representation, interactive handling
Usability/ Communicability	GUI available	Detailed GUI, graphical model builder	GUI including graphical model builder	Detailed GUI	Detailed GUI	Detailed GUI
Software availability	Upon request	Free for non-commercial use	Upon request	Free	Commercial	Free, upon request

[a]In SELES, a simulation scenario is defined as a complete set of initial state information and the definition of landscape event.

integrating sub models and of feeding back sub model results for further decision-making. In addition, we identified specific requirements concerning usability and communicability of modeling results. Further requirements were consequence of drawbacks we recognized in existing frameworks, such as the integration of calibration functionality and the explicit representation of scenarios. An analysis of published modeling environments revealed that existing generic approaches do satisfy single requirements, but not their entirety (see Table 2.1). With SITE, we present a holistic approach for a generic framework for spatially explicit land-use modeling. The system was designed to overcome some of the limitations of previous approaches. Major innovations in the field of land-use modeling are the the high degree of integration of components of the land system, the integration of calibration functionality allowing the simultaneous calibration of interacting component models.

3. System design features

3.1. Aspects of software quality

As mentioned above, with the SITE land-use modeling framework it is intended to provide a generic platform for performing regional land-use change modeling and simulation tasks. Especially the aspect of generic enforces a number of specific design features on which the implementation of the framework is based.

The task of creating a generic platform for running different land-use change models implies the development of complex software. Handling software complexity is a focus of research of its own (Gamma et al., 1995). One basic method to handle complexity is encapsulation, which means specific tasks are implemented in separate modules. To allow interaction of such modules, a way for communication by defining adequate interfaces has to be established. Thus, by altering functionality in one module, only the module itself is affected. Following this strategy in the process of software development leads to component-oriented design for the entire system and object-oriented design for the implementation of single components.

Modularizing software also is a way to create more robust software and thus contribute to the achievement of a certain level of quality. As a matter of fact, the SITE framework needs to meet a high level of software quality since it is intended to be used over a longer time period and for a larger number of applications. Software quality is defined by the non-functional requirements of the system and is not obvious from the catalog of functional requirements. Important non-functional requirements are, amongst others, changeability, interoperability, efficiency, reliability (error tolerance, robustness), testability and reusability (Buschmann et al., 1998). A comprehensive collection to gauge software quality that should not only be respected during software architecture development but also in all other phases of the development process, is given by the international standard ISO 9126. Table 3.1 gives a short description of the quality criteria defined by ISO 9126.

Apart from a few minor sub attributes listed in ISO 9126, this standard was crucial in the development of the SITE framework. For instance, SITE development is not critical concerning security issues. Maturity, to mention another aspect, must of course be achieved over time and adoption in a number of projects. Other features like usability or changeability can already be found in the list of system requirements. The SITE system is designed to show a high usability both in aspects of user-friendliness by providing a graphical user interface and being able to house a wide range of different modeling tasks due to the establishment of a generic structure.

A central focus lies on the maintainability of the SITE system. The capability to enable a large number of modeling applications implies that the software might be utilized over a longer period in time and thus has to be administered accordingly. It can also be expected that the persons being in charge of the maintenance change over time. Multiple applications will most probably raise the need to change and expand the system. Due to these reasons the design will be characterized by strong modularization of the software, resulting in a component-based implementation and the use of object-oriented analysis.

Table 3.1.: Software quality attributes defined by the international standard ISO/IEC 9126

Attribute	Sub-attributes
Functionality: A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs.	*Suitability, Accuracy, Interoperability, Compliance, Security*
Reliability: A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time.	*Maturity, Recoverability, Fault Tolerance*
Usability: A set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users.	*Learnability, Understandability, Operability*
Efficiency: A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions.	*Time Behavior, Resource Behavior*
Maintainability: A set of attributes that bear on the effort needed to make specified modifications.	*Stability, Analysability, Changeability, Testability*
Portability: A set of attributes that bear on the ability of software to be transferred from one environment to another.	*Installability, Replaceability, Adaptability*

3.2. Separation of implementation and application

One main target in the development of the SITE regional land-use modeling framework is to provide a generic platform for implementing regional land-use models of different character and running the respective simulations. This goal is achieved by strictly separating the units dealing with project specific aspects and the units providing functionality for all applications. In the following paragraphs, the unit housing all generic functionality will be referred to as the system domain, while the project-specific unit will be named the application domain (Fig. 3.1). Analogous to that, it will further be distinguished between system developer and application developer. Defining such a separation implies accepting a number of compromises, since modeling projects might be very specific in some details and thus it may not be possible to provide a generic platform for all eventualities that may arise. In consequence, the way of separation is strongly determined by what the parties involved in the design of the SITE system define as relevant to all modeling studies.

Utilizing a system design which strictly distinguishes between system and application domain has a number of advantages that compensate for extra efforts required by implementing the desired generic. Since the system provides elements needed by all applications, the application developer will not have to deal with them, save time and efforts and can focus on his or her specific application. Thus, generic functionality will not be implemented repeatedly and redundancy will be avoided. In addition, multiple applications using the same system will lead to a robust system implementation exhibiting a minimal number of errors and consequently improving its reliability.

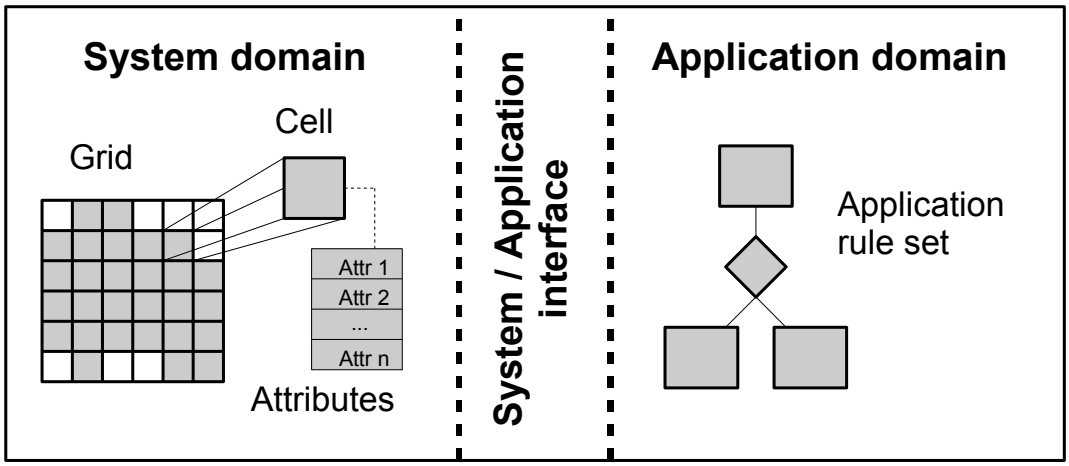


Figure 3.1.: Separation of the SITE framework into system domain and application domain. The system domain provides generic data structures that are used and initialized by applications.

A second advantage is the possibility to administrate the implementations of system and application separately. In case there is no separation, making changes to either a system- or application-specific feature means altering the whole implementation. System and application domain disjunction will avoid this and thus contribute to error prevention and improved quality of both system and application.

For a definition of the system domain of the SITE framework, it is necessary to recall the classification of land use models and modeling techniques. The SITE system is designed to support regional land-use models. Besides scale differences concerning the rules and processes underlying land-use models on the continental or global scale, regional land-use models usually also differ in aspects like data volume (which is usually smaller) or the representation of the spatial data. The SITE modeling framework is capable of supporting all regional land-use models following the criteria listed below. These features and respective data structures are an integral part of the SITE system domain and all functionality for their maintenance is implemented there.

- *Spatial explicitness based on cellular automata:* The way in which spatial data is represented in a model is a basic feature which determines the data structures of the modeling system. One could think of a solution which puts the representation of spatial information in the responsibility of the application domain, but that would result in increasing complexity when implementing the respective application code and hence collide which design features like usability or efficiency. Cellular automata based models are numerous and very prominent in the community, so this confinement proves to be a good compromise.

- *Georeference based on coordinate systems:* This means that each grid cell represents a piece of land of the same size or a size that can be derived by a functional dependency (based on cell position, e.g. dependent on geographic latitude). In its current state, however, the SITE model only supports rectangular coordinate systems (e.g. UTM), but it can be easily extended to support other coordinate systems as well.

There are no restrictions with respect to the spatial resolution of the model, i.e. the value

can be freely defined by the application. All other criteria used in the classification of land-use modeling systems can be addressed in the application domain.

Based on this definition of responsibilities the system domain can be configured. As a central data structure, it houses a class representing a two-dimensional grid of application-defined size and resolution plus respective iteration functionality and methods to access each grid position. Every single grid cell needs to be represented by an own instance of a data structure which specifically addresses the problem of handling cell attributes. Analogous to grid size and resolution, the number of attributes, their names and data types are only known at run-time as soon as an application is selected. Hence attribute handling must be implemented in a highly dynamic manner. The definition of attributes must be conducted via the system-application interface.

The grid-cell-attribute complex provides the main data structures needed for modeling and simulation functionality. Besides that, one could think of additional data structures and respective functionalities that are useful for modeling tasks and can be provided by the system domain for use in applications such as the aggregation of cells to clusters based on application-defined rules. Despite of the fact, that such functionalities can be implemented as part of an application, there are cases where an implementation as generic service by the system side proves as useful. The SITE system domain houses such functionality, which will be described and discussed below.

The data structures for the grid, cells and attributes represent the static aspect of the SITE system domain. For simulation dynamics, the system domain also needs to provide respective functionality. Since the SITE system integrates cellular automata (CA) as the fundament for land-use modeling and simulation, it must also provide the basic operations typically performed by CA (Weimar, 1997; White et al., 1997). Three basic CA operations were implemented:

- *Initialization:* This operation is performed upon connecting an application to the system domain. The basic procedures during initialization are (i) to create a grid with size and resolution defined by the application, (ii) create the required number of attributes and (iii) set their initial values. Depending on the application, additional procedures can be carried out during the initialization step.

- *Start simulation step:* This system domain operation signals the application to start executing the code containing the logic for performing a simulation step.

- *Allocation of new attribute values:* In CA, the actual assignment of new attribute values is done after finishing a simulation step, so this task is done automatically by the framework as soon as the respective application code has been executed. Nevertheless, an application might require that attribute value changes become effective immediately, thus the system/application interface comprises a respective method.

Until now, only the term simulation step was used, but no actual time was assigned to that. The definition of simulation time and the appropriate temporal resolution depend on the model in use and have to be defined in the application domain.

3.3. Technologies applied in SITE

Due to the strict separation of the SITE framework into system domain and application domain, different technologies can be used for the implementation of each module. However,

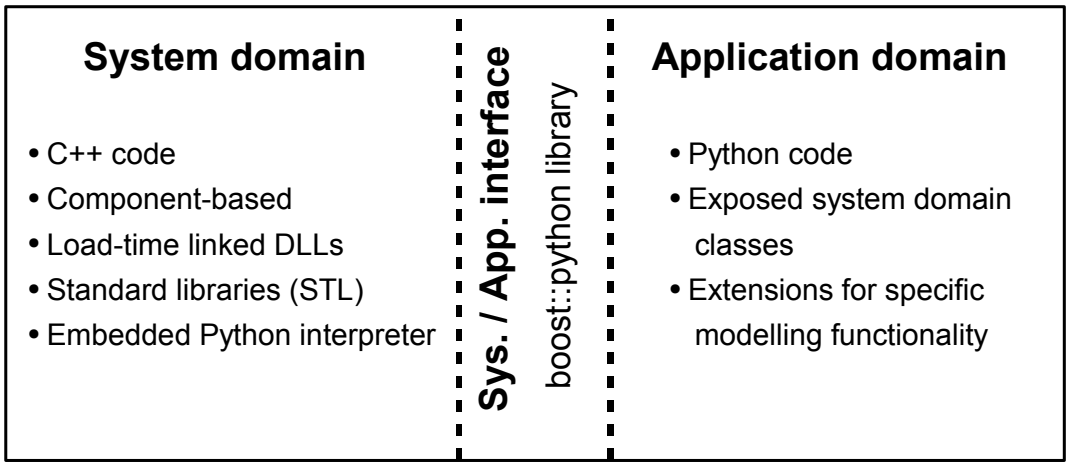


Figure 3.2.: Technology used for the implementation of the SITE framework's system and application domains. The system domain is C++ software capable to interpret application code programmed in Python. The interfacing between the two domains is implemented using the Boost Python library.

a minimum compliance between both domains is required. Following the strategy of providing generic functionality for a number of yet undefined modeling applications means that the system domain must provide a possibility to load application code and execute it. A number of technical options are available to achieve this aim. In the SITE framework the code of the system domain is written in the C++ language, embedding an interpreter for the Python scripting language. The Python language is used to code modeling applications (Fig. 3.2).

Due to the requirement that the SITE system has to run on standard PC hardware and shall be made available to different research groups, it was developed on the Win32 platform. The entire handling of the software is compliant with the Windows philosophy.

System domain

For the implementation of the system domain the C++ programming language was selected. As an object-oriented language it allows a structured implementation ensuring further expandability combined with high efficiency concerning run time and system resources. Although portability (e.g. to a Linux platform) was not a major design goal, the effort necessary for porting the system to another platform is reduced to a minimum by using libraries that are provided for both platforms respectively.

The system domain consists of a number of components with each component being responsible for specific tasks. However, due to portability reasons, no technology like COM was adopted to wrap the component binaries. Instead, each component is built as a Windows dynamic link library (DLL) with a clearly defined interface. This facilitates porting since a similar concept known as shared libraries is applied on Linux platforms. On startup of the system domain, the components are load-time linked and invoked by either a GUI or a command line application.

To enable the execution of Python application code, the system domain embeds an appropriate interpreter and provides all respective functionality like Python language extensions to expose system domain data structures (especially the grid-cell-attribute framework) to the application domain.

The system domain also includes a number of libraries providing basic data structures and functionality. One example is a wrapper library which provides access to the xerces DOM (document object model) implementation which in turn provides an interface to access XML files. The SITE system domain uses XML for configuration files and to save and restore system states.

Application domain

SITE modeling applications need to be coded in the Python scripting language. It is an excellent tool for the formulation of rules, processes and logic to describe the rules underlying a land-use model. Since it is a full object-oriented programming language, there are no limitations with respect to the complexity of an application rule set. In addition, Python has a very large user community and is also used as a scripting language for commercial software packages (e.g. ESRI's ArcView GIS software).

The Python language can be extended by using its C API. In the SITE framework such extensions have been created to make the data structures for the grid, cells and attributes defined in the system domain available on the application domain. In addition, the implementation of such extensions is especially useful when time critical operations have to be performed. Functionality used on the Python side is in fact carried out by a C or C++ module. The SITE framework provides such extensions for time-critical operations like the calculation of distance maps.

System/application domain interface

As described above, generic data structures and functionality are exposed to the application domain via the system/application interface. With its C API, Python already offers a solution to implement this interface. However, being a low-level interface, it is relatively complicated and error-prone to utilize this API. Due to this reason and the fact that Python is widely used, there are a number of libraries which support the exposure of functionality coded in C++ to the Python language. In the SITE framework, the Boost Python library was used. Boost Python is a subset of the Boost library, which itself is a collection of free libraries that extend the functionality of the C++ programming language. Boost Python offers a concise syntax for exposing whole C++ classes and the necessary subset of methods. Especially this property makes it favorable for use in the SITE framework as compared against tools like SWIG (Simplified Wrapper and Interface Generator) where exposing of classes can only be achieved indirectly by a workaround. A detailed description of the SITE system/application interface is provided in appendix A.

4. System architecture

The objective of software design is to develop an adequate software architecture that meets the predefined requirements. In its Unified Modeling Language Specification, the Object Management Group (OMG) has defined the term *Architecture* as an organized structure and associated behavior of a system which can be decomposed recursively into different parts. These parts interact and include classes, components and subsystems (OMG, 1999). This definition of an architecture considers aspects like the fragmentation of the entire system into multiple components, communication between single components and relations of components among each other. An appropriate definition of software architecture which matches the context of SITE framework development is provided by Endejan (2003), where software architecture is the basic structure of a software system that describes an assemblage of defined components interacting via interfaces. The architecture specifies the components' scope and their relationships among each other. A component is defined as an enclosed binary software module that implements application-oriented and semantically mated functionality that is provided to clients via interfaces (Balzert, 2000).

4.1. Existing standards and architectures for integrated modeling

As discussed in chapter 1, integrated land-use modeling is a tool of in increasing importance throughout the scientific community. A number of models already exists and consequently, there have already been efforts to bring the different modeling approaches together regarding both modeling concepts and underlying technology.

Beside the International Organization for Standardization (ISO) there are other organizations which have issued recommendations or standards for the technical realization of integrated modeling systems that are also relevant for land-use modeling. Among those, the Institute of Electrical and Electronic Engineers (IEEE), the World Wide Web Consortium (W3C) and the Open Geospatial Consortium (OGC, the former OpenGIS Consortium) are the most relevant ones.

High Level Architecture (HLA)

The High Level Architecture is an architecture to combine interacting sub models to aggregated models pursuing the target to significantly increase the interoperability of simulation models (Kuhl et al., 1999). It was originally developed for military applications but is increasingly adopted in the civil domain (Schulze et al., 1999; Lindenschmidt et al., 2005). In the year 2000 it became an IEEE standard. Since HLA is a generic architecture it only provides functionality to increase the interoperability of simulation models. This functionality is encapsulated in the HLA run-time infrastructure (RTI). There are several commercial implementations of the RTI available.

NIST/ECMA reference model

Having been developed as an architecture to integrate different applications in the context of computer aided software engineering (CASE), the NIST/ECMA reference model provides an extendable framework to establish communication among single applications and a consistent graphical user interface for data representation. The problem of the consistent integration of data and software that led to the development of this architecture is comparable to those arising when setting up an integrated modeling framework. Chen and Norman (1992) provide further information on the NIST/ECMA reference model. The NIST/ECMA reference model was one basis for the development of the reference architecture issued by the Open Geospatial Consortium.

Open distributed processing reference model

Since single components of an integrated modeling system do not necessarily have to run on the same machine, one has to consider the possible distributed character of the system. A distributed system makes special demand to the underlying software architecture, therefore ISO created a framework to facilitate and encourage the creation of standards for such distributed systems and published it as the standard ISO/IEC 10746-(1 to 4). Four basic elements are postulated for standardization: System description using object-oriented analysis, system description via five separate but related viewpoints (enterprise viewpoint, computational viewpoint, information viewpoint, engineering viewpoint and technology viewpoint), the definition of a system-infrastructure to ensure transparency regarding the distribution of applications and finally a framework to assert that the system is compliant to the respective ISO standard. An application of the open distributed processing reference model is the OpenGIS service architecture introduced below. Further information is provided by Farooqui et al. (1995) and Schürmann (1995).

OpenGIS service architecture

The OpenGIS service architecture (Percivall, 2002), issued by the Open Geospatial Consortium, is a technical reference model. It has been taken over by the International Organization for Standardization as standard ISO 19119 in April 2001. It assumes that underlying target systems are distributed and implemented using object-oriented analysis. It provides a taxonomy for geographic services and regulates how platform-independent specification for services have to be created and how to derive respective platform-dependent specifications. The goals pursued by the OpenGIS service architecture standard are to

- provide an abstract framework, that allows the development of specific services,
- enable interoperable services by standardization of interfaces,
- support the development of service catalogs through the definition of meta data of services,
- enable the separation of specific data and services,
- allow the use of services from one provider to work on data of another provider,

- define an abstract framework, that can be implemented in different manners.

The OpenGIS service architecture refers to the Open Distributed Processing Reference Model by adopting four of the five viewpoints defined there. Viewpoints considered are the computational, information, engineering and technological viewpoint. The enterprise viewpoint is described in other parts of the ISO 19100 series of standards (e.g. in the ISO 19101 reference model). For a detailed description and discussion of this and the above introduced architectures, see Endejan (2003).

SISA architecture

Based on a review of existing architectures for integrated modeling, Endejan (2003) developed an architecture for a system for integrated simulation-based assessment (SISA). He defines a system for integrated simulation-based assessment as a software system that combines both data and simulation models from different disciplines dealing with the "system earth" in a consistent frame and that computes and provides new data describing state and possible long-term changes of the "system earth". This is basically done to support policy-makers. Referring to the quality of assessment results the consistent frame is considered being of special importance since it contributes to the transparency and comprehensibility of results.

Figure 4.1 gives an overview over the SISA architecture. Not considering a component implementing the client side of whatever kind (e.g. command line or GUI), the architecture features twelve different components.

Simulation System The simulation system component is the central component. It is responsible for computation, storage and propagation of simulation results. For the propagation of results it features a specific interface which ensures interoperability and reusability of the simulation system.

Simulation-Run Manager The Simulation-Run Manager ensures comprehensibility and reproduceability of simulation results. Its responsibility lie in both managing the specification of simulation runs and propagating them to the actual simulation system.

Data Access This component contributes to data integration and the allocation of simulation results. Services to transform data to make them consistent with the data format required by the simulation system are also assigned to it. Thus, the data access component supports requirements such as interoperability and ex changeability.

Database System The actual data used during simulation runs are housed in the database system component which encapsulates database functionality. Beside having an interface providing common database access operations, it features a separate interface for performing queries. Nevertheless, data are not accessed directly by the simulation system, this is done via the data access component which serves as an integration layer and performs necessary data transformations.

Catalog Manager One function of a system for integrated simulation-based assessment is the provision of meta data describing data sets used for simulation. Since indexing of resource meta data can be seen as an independently functionality inside a SISA, this functionality is implemented as a separate component responsible for the management and delivery of meta data concerning SISA resources.

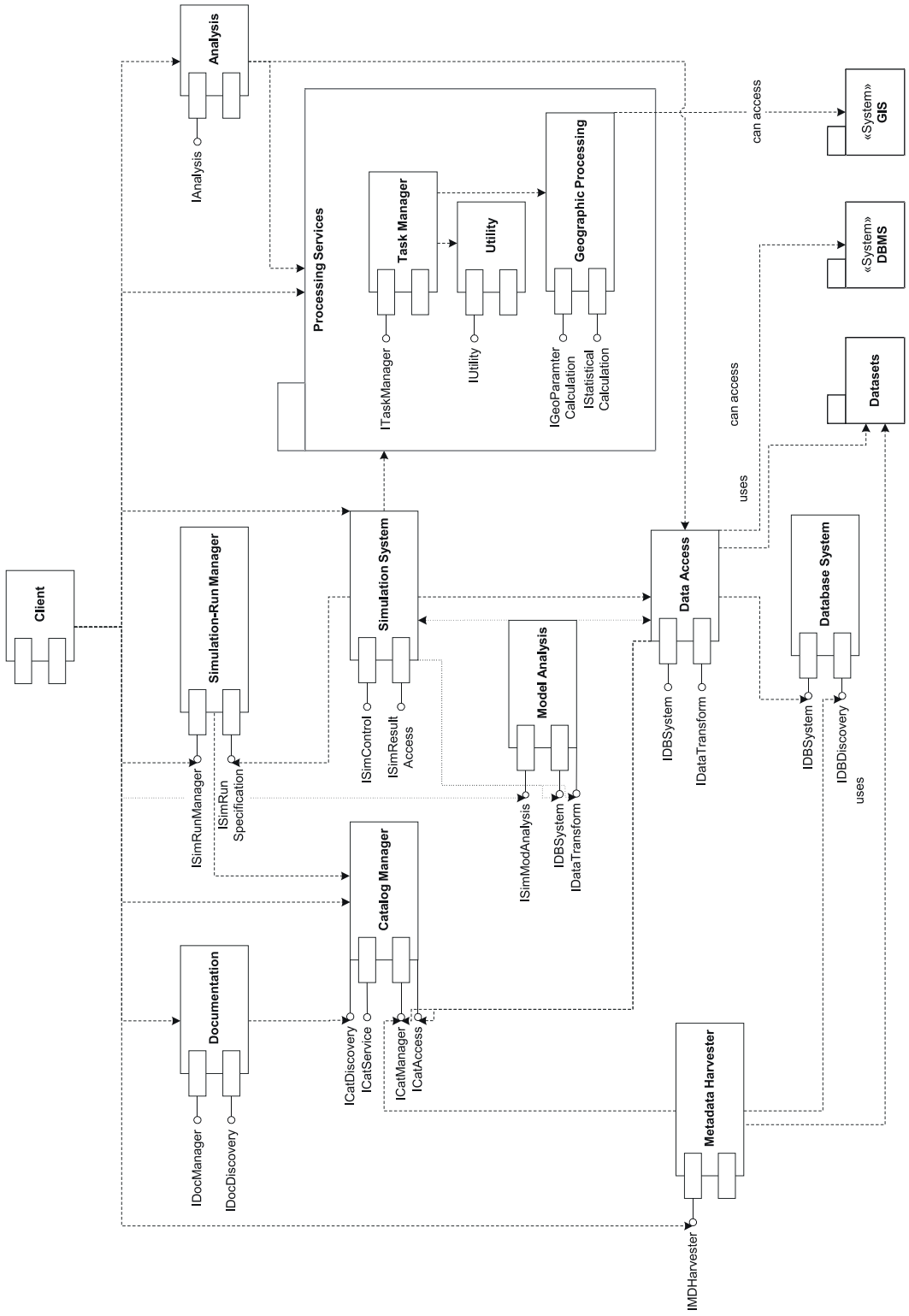


Figure 4.1.: Architecture of a system for integrated simulation-based assessment (SISA) as proposed by Endejan (2003). Arrows referring to complete components instead of single interfaces denote dependency to all interfaces of the respective component.

Metadata Harvester There is a conflict between centrally managing meta data and storing them locally. The first option is favorable for the integration inside a SISA while the latter one can be advantageous in other aspects since it is sensible to store meta data at the same location as the data they describe. The implementation of a meta data harvester can solve this conflict. The harvester is a program that automatically searches the a defined file system for the desired meta data and thus can simulate a central storage for the catalog manager component.

Documentation While management of meta data lies in the responsibility of the catalog manager, the documentation component administrates all information about executed simulation runs and underlying scenarios. In addition it gives model users information about the handling of the system.

Utility This is a component reserved for any kind of data processing that can be realized independent from the simulation system.

Task Manager The single components in the SISA provide reusable operations which naturally can be used by all other components inside the framework. To facilitate the use of these services, functionality for the claim and control of services should be provided. Thus, the task manager component is responsible for the program controlled invocation of other SISA services.

Geographic Processing This component provides services for geographic data processing and can encapsulate an existing GIS system or implement GIS functionality itself.

Model Analysis The responsibility of this component is to manage procedures to analyze model behavior like sensitivity or uncertainty analysis. Usually this is done by altering specific model parameters and evaluate their influence on the simulation result. Respective functionality is implemented here.

Analysis Set up upon data processing services and services provided by the data access component the analysis component is responsible for supporting the model user in the analysis of simulation results.

Realization of the described SISA architecture can be achieved using simple technical tools and free software as has been proven by Endejan (2003) who set up a SISA to run the GLASS (Global Assessment of Security) integrated model (Alcamo et al., 2001). Especially the meta data framework and the simulation run manager component contribute significantly to increased transparency and reproducibility of simulation runs. Applying the simulation system interfaces to sub models leads to improved reusability and interoperability among sub models. In addition, the information provided by the documentation component has proven useful for a transparent assessment.

4.2. SITE system domain architecture

The SISA architecture provides a well suited template for the development of the SITE architecture. Many features of SISA match well with the requirements listed for the SITE framework.

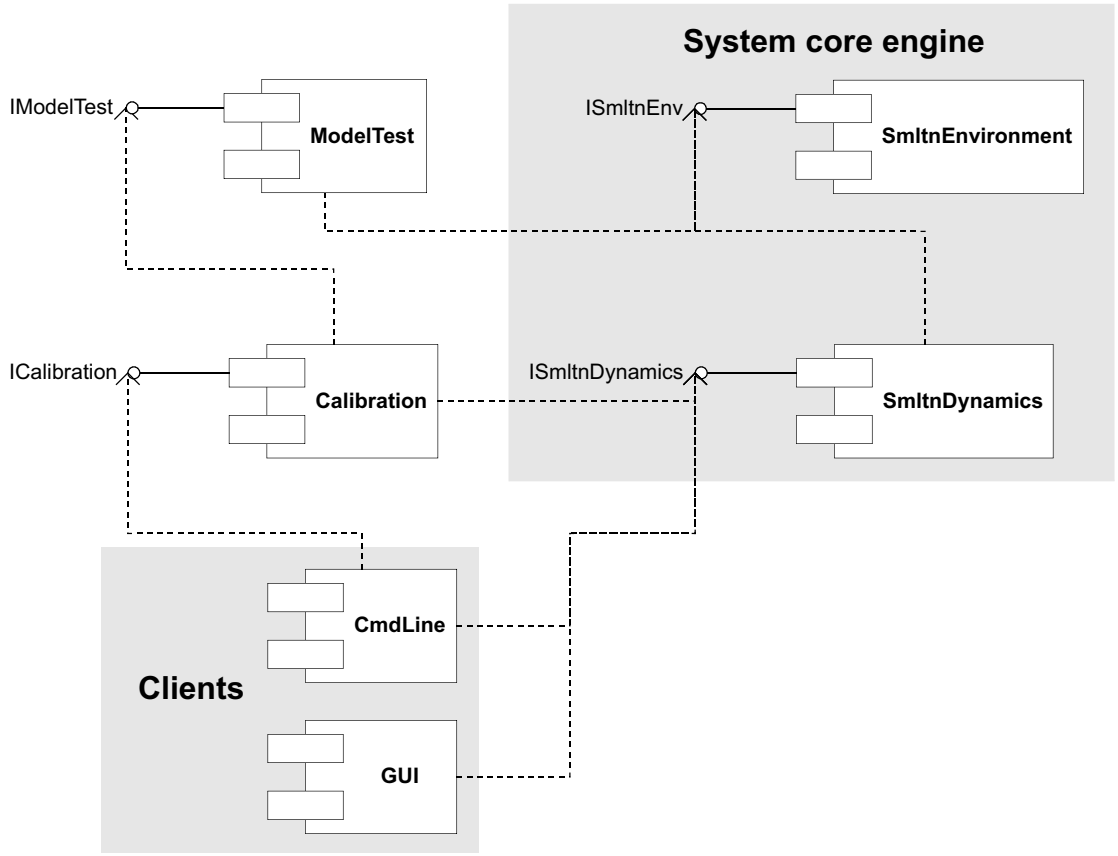


Figure 4.2.: Architecture of the SITE system domain. Each displayed component represents a binary entity. While the client components are executable files, the functional components are realized as DLLs and are linked to the GUI ore CmdLine component respectively.

However, the SITE architecture has to be a more compact solution due to requirements concerning ease of use or ease of distribution. In particular, this means that SITE components are more closely coupled than their SISA counterparts. Nevertheless, loose coupling of components like in a SISA is not required due to the basic design feature of the SITE system, which is its separation into a system and application domain. For applications, there is only a minimum of constraints and thus it is imaginable to implement a SISA-based architecture in the SITE application domain. In contrast to SITE, the simulation system component inside the SISA architecture already represents an application.

Figure 4.2 shows the architecture of the SITE system domain. Four components are defined to house all necessary functionality and to meet the defined requirements. In addition, two client components are specified. For model development, performing simulation runs and presentation tasks, a graphical user interface is provided that features a variety of possibilities to work with an application interactively. For elaborate tasks like performing multiple simulations or a calibration run, a command line client is provided which enables the use of SITE inside a batch processing framework. A detailed description of the client components is given in section 5.3.

The components *SmltnEnvironment* (simulation environment) and *SmltnDynamics* (simulation dynamics) are the central building blocks of the SITE system domain and together will

be referred to as the system core engine in the following. Component *SmltnEnvironment* is responsible for managing all static aspects of the simulation. It provides all data structures necessary for representing spatially explicit modeling data. In particular, these are classes that make grid, grid cell and cell attribute functionality available for use in the application domain. Other classes support the application-defined organization of grid cells into layers determined by attribute values. For the grid class adequate iterators for different purposes (e.g. entire grid iteration, attribute layer-specific iteration, moving window iteration) are supplied. The component exposes its functionality via the *ISmltnEnv* interface.

As can be assumed by its name, the *SmtlnDynamics* component, to be utilized via the *ISmltnDynamics* interface, implements all functionality dealing with dynamic aspects of a simulation. On one side, this is a framework for the basic cycle of cellular automata operations (initialization, simulation step, attribute allocation). On the other side, since change rules are project-specific and thus are integral part of an application, the *SmltnDynamics* component is the instance where the system/application interface (which is technically the connection of the SITE system C++ part to the Python scripts) is implemented. In addition to the interfacing technology the component features extensions to the Python language that have been found useful and can be expanded respectively. The *SmltnDynamics* component operates on the data structures provided by *SmltnEnvironment* using the *ISmltnEnv* interface.

In addition to the basic simulation functionality, the system domain provides other generic services supporting quality aspects of modeling applications. Functionality to assess the quality of simulation results through a number of map comparison algorithms that work on categorical data is implemented in the *ModelTest* component. It is designed for simple expandability and its functionality is exposed via the *IModelTest* interface. The featured map comparison algorithms work on grid cell attributes representing the classification and thus operate on the data structures implemented in the *SmltnEnvironment* component via its *ISmltnEnv* interface.

The *Calibration* component provides methodology for automated calibration of application rule sets. Basically, it is a collection of algorithms that are capable of finding optimal or adequate solutions for an application-defined parameter set with respect to an objective function (currently it only includes the implementation of genetic algorithms). The implementation is analogous to the implementation of the *ModelTest* component and allows simple integration of new algorithms. The evaluation of candidate solutions created by an optimization algorithm is based on map comparison algorithms provided by the *ModelTest* component. Therefore there is a dependency of component Calibration from component *ModelTest* which is accessed via its *IModelTest* interface. Due to the fact that performing model calibration using an optimization algorithm means running multiple simulations (one for each candidate parameter set), the *Calibration* component must be enabled to repeatedly start simulation runs with altered parameter sets until an application-defined termination criterion is met. To ensure this, a dependency of the *Calibration* component to the *SmltnDynamics* component via its *ISmltnDynamics* interface is defined. For details on the *Calibration* component see section 5.2.

As has already been noted, all SITE system domain components are either compiled into dynamic link libraries (DLLs) or executable files. No higher level technology like COM or .Net for the component definition has been used, since this is not imperative due to all components being implemented in the C++ language and their rather close coupling. Each component specifies exactly one interface. Exchangeability of components on the binary level is given provided that interfaces are identical.

5. Implementation of the SITE system domain components

In this section all components that are part of the SITE system domain architecture introduced in section 4.2 will be described in detail especially regarding their static structure, dynamic behavior and they way they are interfaced among each other. In class diagrams, not all methods and class members will be displayed. Instead, there will be a focus on methods contributing to component interfacing and to exposure of functionality to the application domain.

5.1. System core engine

5.1.1. Simulation environment

The *SmltnEnvironment* component contains and provides all data structures that are required to run cellular automata-based spatially explicit land use change simulations. It can be seen as the main component of the SITE framework since it implements the largest share of generic modeling functionality. As a consequence, it is also the most complex component of the SITE system, both regarding the number of implemented classes and the relations among classes. Most other components depend directly or indirectly from *SmltnEnvironment*.

The overall class layout is displayed in figure 5.1. Inside the *SmltnEnvironment* component four functional parts can be identified: The actual simulation grid, a framework for managing and handling of attribute data and data structures for the representation of information layers determined by cell attribute values. The fourth part contains different iterators that enable the operation on the main functional classes and provide adequate access to them.

Since the main characteristic of applications designed to be operated by the SITE framework is the spatial explicitness represented by a cellular automata approach, the central functional part is the cluster of classes referred to as the simulation grid, with the *Grid* class being the fundamental data structure. Figure 5.2 displays the static structure of this part in more detail together with class methods and class members involved. The *Grid* class acts as entry point to all *SmltnEnvironment* objects, implements a two-dimensional rectangular array of cells and defines a number of methods to enable access to grid information and single cells. The size of the simulation grid, its georeference and spatial resolution are dynamic and defined by an actual application. Georeferencing is handled in class *GeoRef* which is owned by *Grid*. Class *GeoRef* provides a small number of methods to specify georeference and to perform simple scaling calculations. In the current version, georeferencing must be based on UTM (Universal Transverse Mercator; see Snyder, 1987) coordinates. The single grid positions are instances of the class *Cell* aggregated in the *Grid* class. *Cell* instances basically contain a reference to an instance of class *AttrSet* (attribute set) describing the overall cell state and a number of methods to provide access to attribute values.

Regions examined in land use change simulations typically do not have the shape of a rectangle. The two-dimensional cell array implemented in the *Grid* class therefore represents the smallest possible bounding box for the application region. Consequently, not all grid cells are part of the project regions and have to be marked as invalid. In SITE framework, only valid

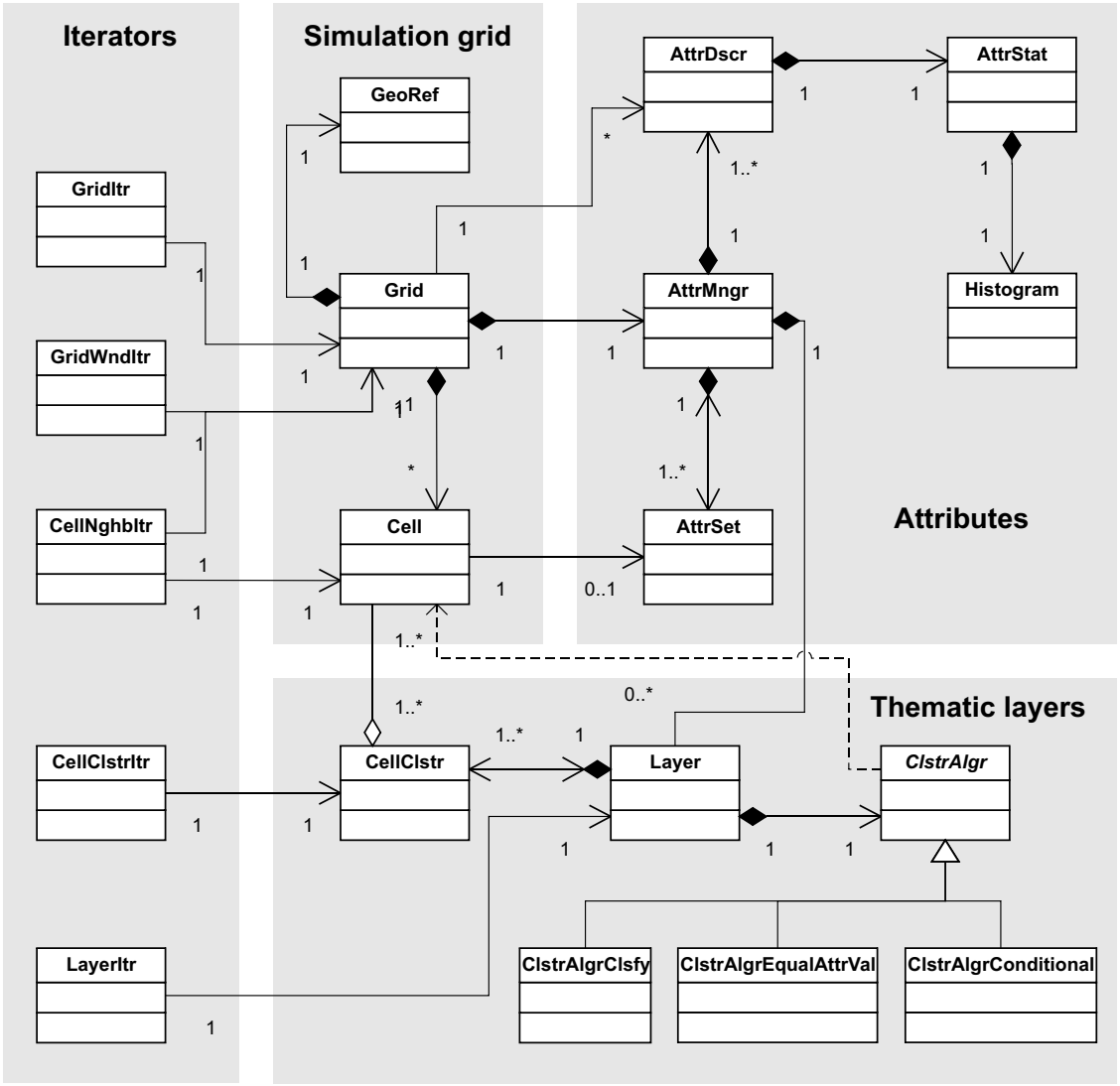


Figure 5.1.: Class diagram showing the overall static structure and class relations of the *SmltnEnvironment* component. For clarity reasons class methods and data members are not displayed.

cells have a reference (realized by a pointer to class *AttrSet*) to an attribute set. This is the criterion checked in the *Cell.IsValid()* method. There is a $1 \rightarrow 0..1$ relation between the classes *Cell* and *AttrSet* and no attribute values are managed for invalid cells. As pictured in figure 5.2, the *Cell* class defines methods to directly access a cell's x and y coordinate and its cell ID (*X()*, *Y()*, *GetId()*). These three criteria are internally handled as normal attributes and could also be accessed using the method *GetAttrVal(attrName)*, but since they represent integral information for every cellular automata-based simulation, these methods are hard coded and respective attributes in applications must be named accordingly (*OBJECTID*, *x*, *y*)). Attribute values are returned using a *Variant* data type.

Although class *Grid* provides direct access to single cells by specifying cell coordinates, this is not the usual procedure when analyzing the grid and applying land use change rules. Typically, only valid cells are of interest in such an analysis and each cell has to be accessed. Therefore a safe way to traverse all valid cells of the grid has to be established. This is done by

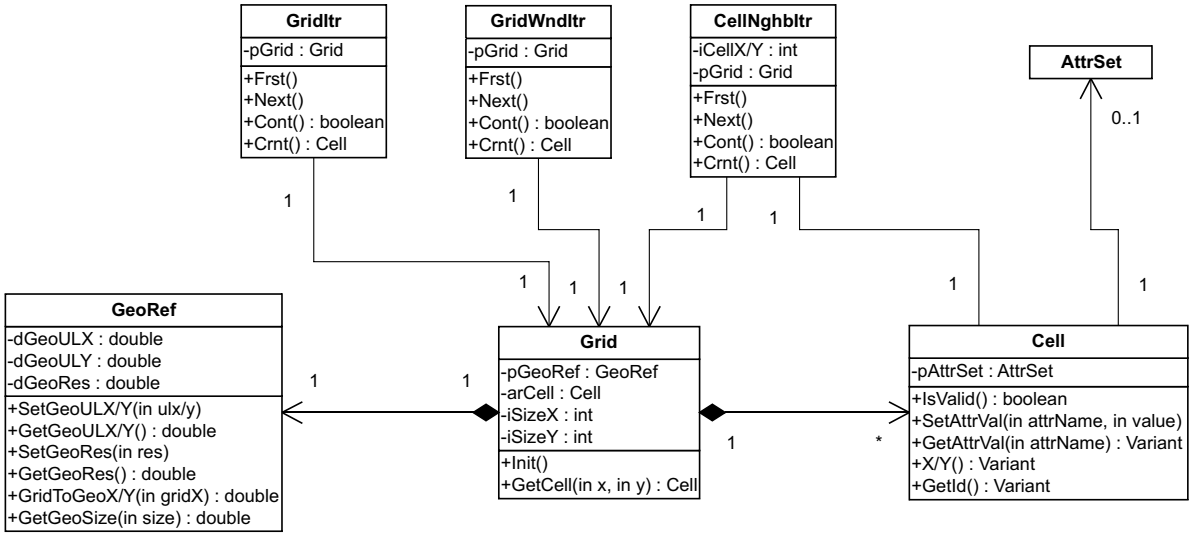


Figure 5.2.: Detailed class layout of the SITE simulation grid. Class methods and members involved in the grid representation are displayed.

implementing iterator classes (applying the iterator design pattern; see Gamma et al., 1995) that can be configured to traverse the grid in a way required by the application. Three different iterators are offered. The *GridItr* class encapsulates functionality to iterate the entire grid. By default, invalid cells are ignored. This iterator can be configured to only return cells that serve as start cells for iterations handled by the second iterator, *GridWndItr*. This iterator traverses the cells of a rectangular grid subset while the subset size is defined by the current application. Combining these first two iterators, a moving window iteration is possible. This service is used in the *Validation* component for map comparison algorithms that utilize such moving windows. The third iterator, *CellNghbItr*, traverses all neighbor cells of a given center cell (specified by its grid coordinates). It can be configured to apply Von-Neumann (4-pixel) or Moore (8-pixel) neighborhood. Extended Moore neighborhood is currently not implemented but the system can be easily extended.

All iterator classes implement the same interface which consists of the methods *Frst()* (set iterator to first element of underlying object collection), *Next()* (step iterator to next element), *Cont()* (check if current element is valid or if there is a current element at all) and *Crnt()* (return the current element of whatever type). Using this interface, iterators can easily be employed in C++ *for* and *while* loops. Depending on the purpose of the iterator and the complexity of the underlying object set, there are additional methods for configuration. All iterator classes are available globally throughout the SITE system and iterator objects are instantiated when needed. They require a reference to the underlying object set and establish a directed association of the multiplicity $1 \rightarrow 1$.

The attribute framework of the SITE *SmltnEnvironment* component is designed to enable generic handling of cell attributes while causing a minimum number of restrictions for applications. The only restriction so far is that an application must specify three attributes representing describing the cell ID and a cell's x and y-coordinate on the simulation grid. Figure 5.3 depicts the class layout of the attribute framework. It shows a straightforward implementation approach by associating an attribute set object (instance of class *AttrSet*) to each valid grid cell object to retain the attribute values for the respective cell, thus establish-

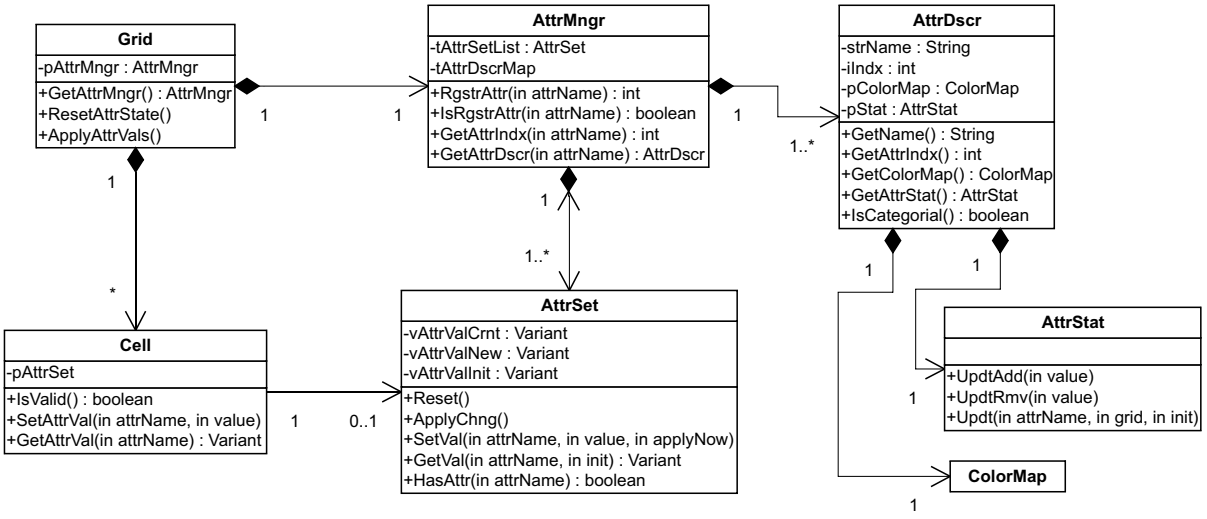


Figure 5.3.: Detailed class layout of the SITE attribute framework. Classes *Grid* and *Cell* are also shown since they serve as entry points to the attribute framework; for those classes only methods and members dealing with handling of attributes are displayed.

ing a $1 \rightarrow 0..1$ relationship between the classes *Cell* and *AttrSet*. To allow easy handling of attributes on the application side, it is reasonable to address specific attributes by their names, i.e. by using an ID of string type. However, using a map for associating attribute names to attribute values inside the multitude of *AttrSet* objects is not advisable as attribute names would have to be stored for each instance of *AttrSet* which results in high redundancy and an increased consumption of memory. This is even more problematic when meta information for each attribute has to be managed. A more efficient way to store attribute values, both regarding memory usage and access time, is to use a vector as container for attribute values. Attribute values thus have to be addressed by an index. In the SITE attribute framework implementation, management of attribute data lies in the responsibility of the three classes *AttrMngr* (attribute manager), *AttrSet* (attribute set, attribute values for one cell) and *AttrDscr* (attribute descriptor, attribute meta data).

The grid object delegates management of attributes and attribute data to an instance of class *AttrMngr*. The attribute manager is the central object of the attribute framework. It aggregates all *AttrSet* and *AttrDscr* instances, handles adding of new attributes (also during simulation runs) and mediates between *AttrSet* and *AttrDscr* objects. Each attribute has to be registered by the *AttrMngr* instance. On registration of an attribute, specified by the attribute name, an *AttrDscr* object is created to maintain meta information for the attribute. To store *AttrDscr* objects the attribute manager uses a map that associates the attribute name with the actual descriptor object. On creation of a new attribute, an index is assigned to it by which respective cell attribute values can be accessed from the data vectors encapsulated in *AttrSet* objects. This index is stored as part of the attribute meta data. Meta data maintained by the descriptor objects additionally includes the reference to an attribute-specific statistics object providing basic descriptive statistics and to a color map object for categorially-scaled attributes (see figure 5.4 for attribute framework dynamics).

To access an attribute value for a cell, the respective member function (*GetAttrVal()*) is called using the attribute name. This request is forwarded to the *AttrSet* object. The attribute set object retrieves the index of the specified attribute from the attribute manager

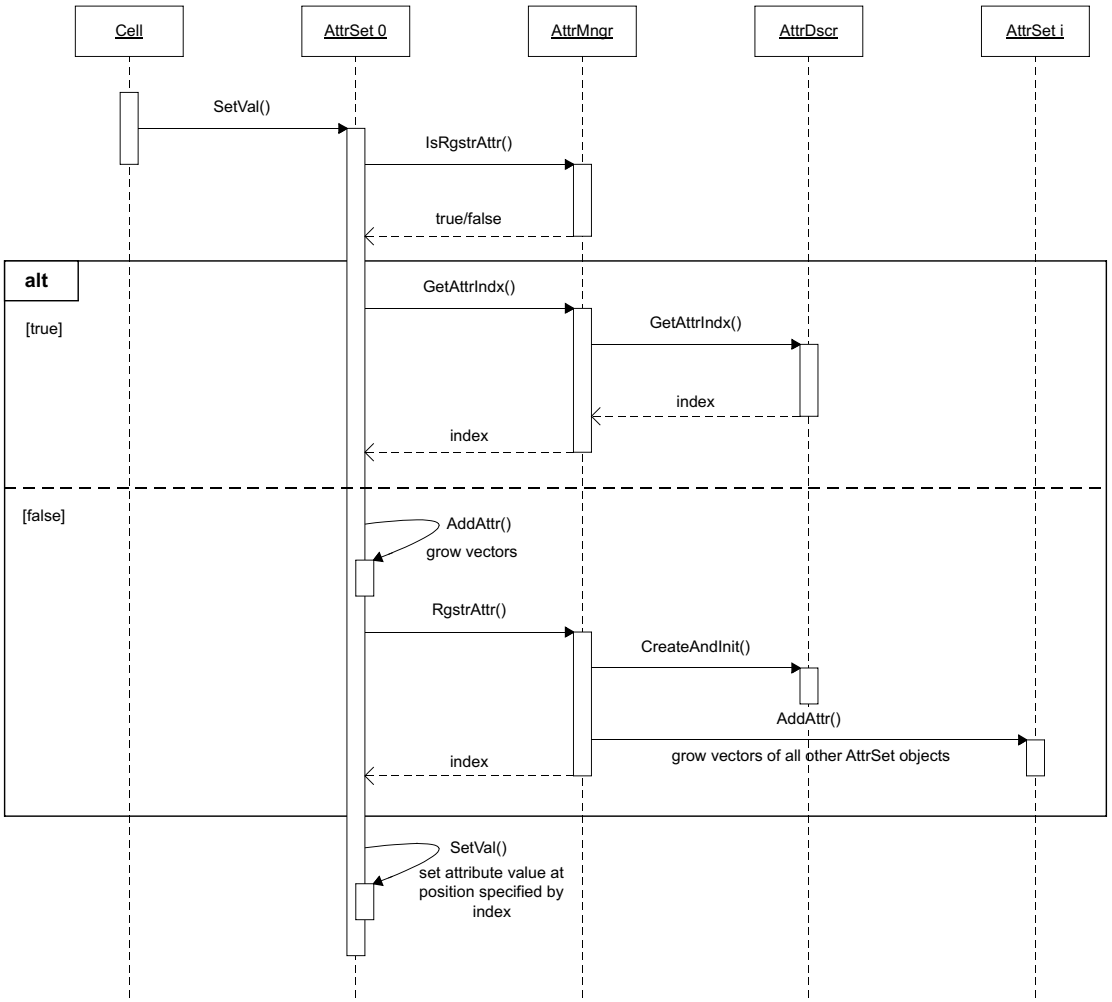


Figure 5.4.: Dynamics implemented in the SITE attribute framework. When accessing an attribute (in this case to set an attribute value), it is first checked, whether the attribute is registered or not. For registered attributes, the *AttrMngr* object derives the index of the attribute values by accessing the attribute meta data object of type *AttrDscr*. For yet unregistered attributes, a new meta data object is created and an index is assigned. In addition, the data vectors managed by *AttrSet* objects must be enlarged.

and accesses the desired attribute value. The *AttrSet* class encapsulates three different data vectors. The first vector (*vAttrValCrnt*) contains the attribute values at the current point in time of the simulation. In cellular automata, new attribute values are not necessarily applied directly; this is done after finishing the respective simulation step. Attribute values to be allocated after finishing a simulation step are stored in vector *vAttrValNew*. To replace the current values with the marked new values, the SITE framework calls the *ApplyChg()* (apply change) method for each *AttrSet* instance at the end of a simulation step. Delaying attribute value allocation to the end of a simulation step is not mandatory. To specify whether to directly allocate a value or not, the *AttrSet::SetVal()* method provides the *applyNow* flag. The third vector (*vAttrValInit*) is used to store initial attribute values (value of creation time). This functionality is desirable for use in application rule sets, for map comparisons and for the possibility to reset a simulation to its initial state. If an attribute is not registered but

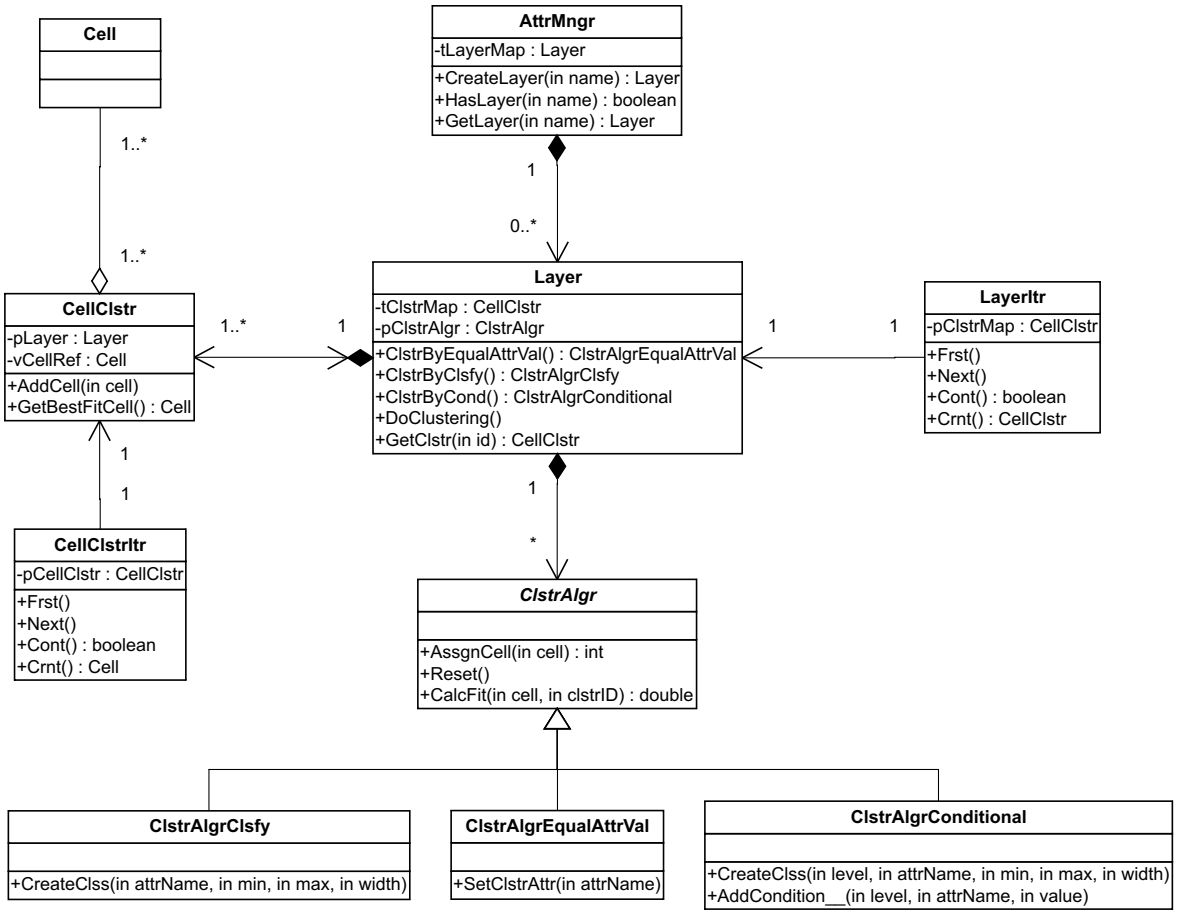


Figure 5.5.: Detailed class layout of the SITE thematic layers implementation together with the three available clustering algorithms.

requested by the application, the attribute manager creates a new attribute, respective meta information and access index, and enlarges the data vectors for all attribute sets.

Simulation grid and attribute framework represent the basic generic functionality that has to be provided for cellular automata-based simulation systems. Technically, it is possible to implement all other functionality in the application domain. However, there are other features that can be seen as fundamental to a great extend so that it is reasonable to provide respective structures by the system domain. Throughout the development of the STORMA application based on the SITE framework it became obvious that it is very useful to aggregate grid cells to cell clusters that are themselves organized in thematic layers. For the clustering cells, it is possible to specify similarity measures (cluster algorithms). Based on such a clustering it is possible to e.g. aggregate grid cells that belong to the same spatial unit (determined by an identical attribute value representing the ID of the respective spatial unit) enabling an application to specifically analyze single spatial units. Another important application is the aggregation of similar cells resulting in a significant reduction of the amount of data for time consuming processing by only performing computations for representative cells.

Figure 5.5 depicts the implementation of thematic layers. Since a thematic layer and the cell clusters it consists of is determined by a selection of attributes and respective attribute values, objects representing thematic layers are composed in the *AttrMngr* instance. The attribute

manager object establishes a $1 \rightarrow 0..n$ relationship resulting in an arbitrary number of layers that can be instantiated by an application. In addition, it implements an interface to create and access thematic layers; this interface is exposed to the application side (see appendix A for a complete description of the system/application domain interface). On creation of a thematic layer, a unique string ID (layer name) has to be specified by the application.

An instance of class *Layer* is the representative for an actual thematic layer. It consists of a collection of cell clusters (instances of class *CellClstr*) and a specification of how to assign single grid cells to cell clusters. This specification is provided by a clustering algorithm object (instance of class *ClstrAlgr*). The Layer interface is exposed to the application domain and consists of methods to select the clustering algorithm, to start the actual clustering and to access single cell clusters after their creation. The process of the actual assignment of grid cells to cell clusters (the actual clustering) is decoupled from the specification of the underlying clustering algorithm. This is necessary to enable the recalculation of a thematic layer for the case that attribute values for grid cells have changed after a simulation step.

The *SmltnEnvironment* component implements three different clustering algorithms each represented by a respective class. The three algorithm classes are derived from the base class ClstrAlgr which declares an interface for assigning cells to clusters, resetting the clustering process and calculating how good a given cell matches a cell cluster. Method *AssgnCell()* takes a cell as argument and return the ID of the cluster to which the cell has been added. Method *CalcFit()* takes a cell and a cluster ID and returns a value between 0 (not in cluster) and 1 (ideal fit) describing how good the passed cell represents the cluster. The public interfaces of the algorithm classes themselves consist of methods to parameterize the clustering process and are exposed to the application domain.

The simplest algorithm, *ClstrAlgrEqualAttrVal*, is configured by an attribute name; for this attribute it reads the value for all cells and assigns cells with equal values for the specified attribute to respective clusters. Using this algorithm, it is possible to e.g. create a thematic layer representing administrative units like districts or villages (provided that cells have attributes like district ID or village ID). Due to the nature of clustering, each cell assigned to a cluster is a perfect representative.

The second clustering algorithm, *ClstrAlgrClsfy*, executes cell assignment through classification. Using the *CreateClss()* method, an application can specify value intervals for an arbitrary attribute, thus defining classes. Other attributes value intervals can be added by repeatedly calling this method. Two cells are assigned to the same cluster if they fall in the same value interval for all specified attributes. How well a cell fits to a cluster depends on how its respective attribute values lie relative to their intervals. A value close to an interval border leads to a decreased fitness value.

The third clustering algorithm (*ClstrAlgrConditional*) is an extension of *ClstrAlgrClsfy*. Based on an initial classification an application can establish further evaluation for specific outcomes of the initial classification by defining conditions to be evaluated for those results. To formulate such conditions it provides a set of methods called *AddCondition()* (equality/inequality, larger than, less than) in addition to the *CreateClss()* method, each implementing a comparison to an application-defined value determining the decision tree path. For the ongoing tree paths below the conditional nodes, further classification of values can be defined. Decision trees can be of arbitrary depth. In contrast to class *ClstrAlgrClsfy*, methods require the current level in the decision tree as argument. This algorithm is especially useful if clustering has to be specific to categories (e.g. land use classes).

Each thematic layer manages at least one cell cluster (composition of multiplicity $1 \rightarrow 1..n$). Each cluster can be identified by a unique ID assigned by the clustering algorithm during its creation. The *CellClstr* class defines methods to add new cells and to access the cell which is the best representative of the respective cluster. Each *CellClstr* instance hold references to all grid cells assigned to it. Each single grid cell, in turn, can be assigned to only one cluster for each thematic layers but to more than one cluster in different layers. Cell clusters can be accessed directly by their ID or via an instance of *LayerItr*, an iterator that traverses all *CellClstr* object of a thematic layer. Access to cells aggregated by *CellClstr* instances is provided by a specific iterator (*CellClstrItr*).

5.1.2. Simulation dynamics

The handling of all dynamic aspects of a simulation lies exclusively in the responsibility of the *SmltnDynamics* (simulation dynamics) component. It operates on the data structures provided by component *SmltnEnvironment*, hence the dependency of *SmltnDynamics* from *SmltnEnvironment* defined in the overall layout of components. Dynamic aspects include performing actual simulation steps and providing scenario data as well as the storage and export of simulation data for single time steps. In the implementation of the *SmltnDynamics* component three major sections can be identified: a central instance representing the entire application rule set functionality inside the system domain, a framework for handling the different kinds of dynamic information and functionality for interfacing the application rule set implementation with the system domain, i.e. the integration of the Python interpreter.

Figure 5.6 depicts the class layout of the SITE *SmltnDynamics* component. An application inside the SITE framework is represented by a single instance of class *RuleSet*. This object provides access to the underlying data structures implemented in the *SmltnEnvironment* component, to all other objects managing dynamic information and to objects representing the integration of the Python scripting language. As the representative of an application's simulation logic, it also provides the necessary methods to control simulation runs. The *Load()* method is used to import a user-specified rule set script into the framework. The initialization of the rule set script is done explicitly using the *Init()* method. An imported rule set script can be executed repeatedly through the use of a reset mechanism invoked by method *Reset()*. An arbitrary number of simulation steps can be performed through the *DoSmltnStep()* method which accepts the number of simulation steps as an argument.

The integration of the Python scripting language together with the establishment of information exchange between the Python (application domain) and C++ (system domain) side is delegated to a set objects of classes encapsulating both the low level Python C API and the *boost::python* library. The central class in this context is *PythonEmbd* (Python embedding) which is instantiated once. The rule set object holds and manages a reference to this instance. The python embedding class contains the Python interpreter which is initialized and terminated by the class methods *StartUp()* and *ShutDown()*. A SITE rule set script must contain two functions called *Initialize()* and *SimulationStep()* featuring initialization logic and simulation rules for one time step respectively. To run a simulation, the SITE system domain calls these functions on the application side through the *PythonEmbd* methods *CallFnctInitialize()* and *CallFnctSmltnStep()*.

In addition to the invocation of these basic functions, the Python integration framework establishes the exchange of information between the SITE system and application domain.

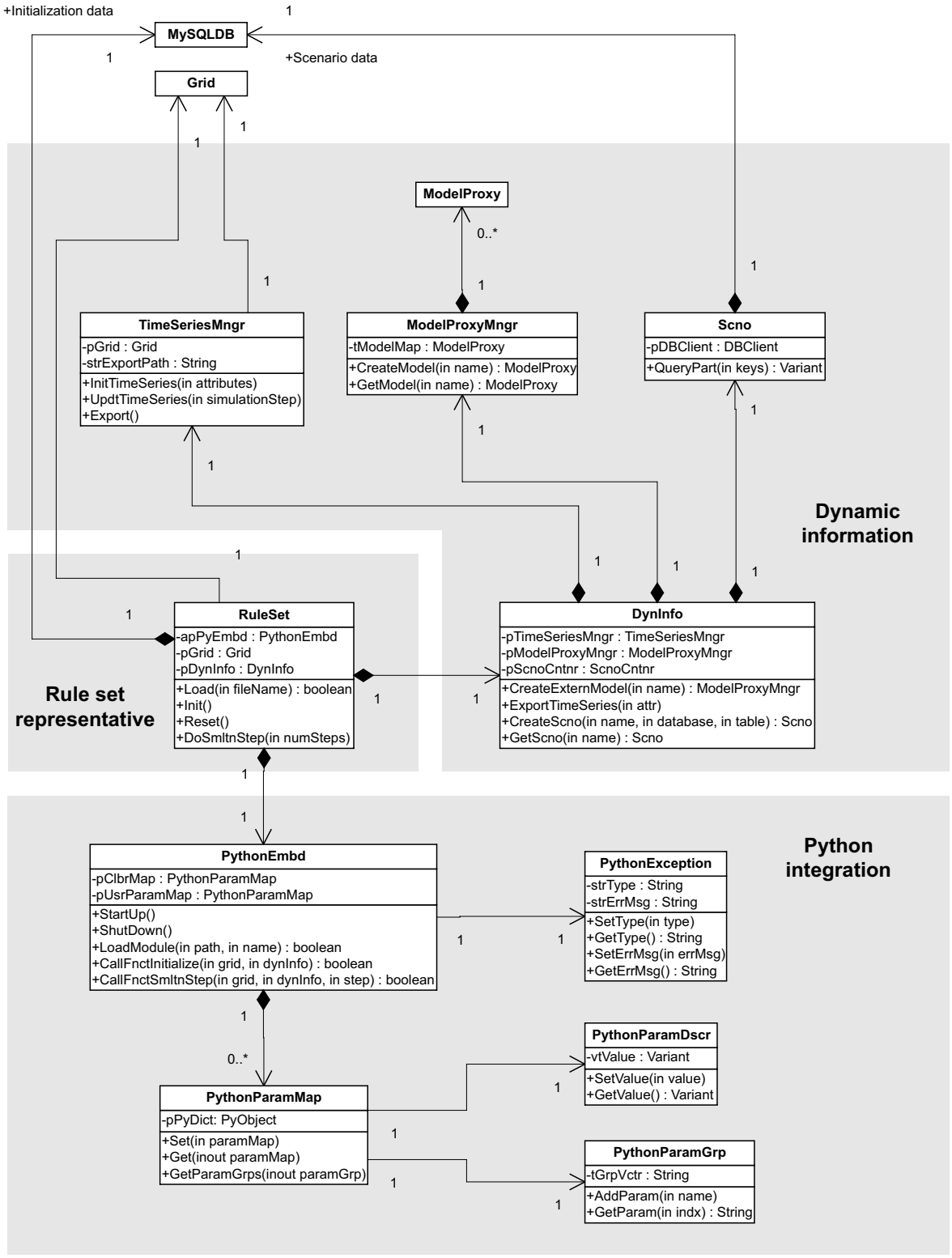


Figure 5.6.: Class layout of the SITE *SmltnDynamics* component. Dynamic information and Python scripting language integration are managed by a single object representing the entire application rule set.

In the subject of information exchange, one has to distinguish between two different types: The manipulation of the SITE system domain data structures (e.g. grid cell attributes) from the Python side and the manipulation of variables in the application rule set script by the SITE system domain (e.g. parameter sets for rule set calibration). To allow accessing of data structures defined in C++, the respective C++ classes need to be exposed to Python, i.e. the Python language has to be extended in a way that data structures like the SITE system domain classes *Grid*, *Cell* etc. plus a set their methods become part of it. To allow manipulation of these classes, an instance of the SITE simulation grid and the dynamic information object are passed to the required Python functions *Initialize()* and *SimulationStep()* (see their C++ counterparts *CallFnctInitialize()* and *CallFnctSmltnStep()* in Fig. 5.6).

While language extensions represent a way to access and manipulate system domain information from the application domain, the access and manipulation of Python variables from the system domain is realized differently. Variables to be accessed by the system domain need to be defined exclusively in separate Python modules. Information exchange is done through the *PythonParamMap* class which is an encapsulation of such a Python module (a *PyDict* object in the Python C API). The helper classes *PythonParamDscr* and *PythonParamGroup* provide meta information on these variables. In the SITE framework variables of the rule set scripts need to be accessible for two different aspects. First, during rule set calibration, the calibration algorithm (see section 5.2) will determine candidate solutions for the respective parameter set. Since the calibration methodology is implemented in the SITE system domain, these values need to be transferred to the application side. Second, to support the design goal of user-friendliness, a number of rule set parameters can be made editable via the GUI, which also is part of the SITE system domain (see Fig. 4.2).

Dynamic information in the SITE framework refers to all aspects of information that are dependent on the simulation time step. It includes several characteristics of information like output data (simulation time series), model driving forces and input data (scenarios) or the processing of necessary information through the use of sub-models. The management of dynamic information is delegated to a special object of class *DynInfo* which is passed to the application rule set script together with the *Grid* object and thus can be accessed from inside the application code. The *DynInfo* object itself holds references to management objects dealing with the specific types of dynamic information. Time series are created for selected attributes. The selection can be made using the GUI or by defining attributes in the application rule set. Time series tables (saving the change of attribute values for every single cell) are exported in form of a csv file for each selected attribute. Each column of a time series table represents the attribute state after completing a simulation time step.

Scenario data are managed by an instance of class *Scno* (scenario manager). In SITE, scenarios are technically represented by compilations of different input data sources in form of both database tables and specifically tagged rule set parameters. *Scno* objects are configured via an XML file which the user must select together with the rule set script prior to performing a simulation run. Based on this configuration, the *Scno* object provides rule set parameterization and establishes the connection to database tables that represent time series. Simulation runs are always determined by the combination of a rule set with one specific scenario. The SITE scenario functionality is designed to allow user interaction. This means that a simulation run can be stopped at on predefined step, The results produced so far can be analyzed to extrapolate whether certain targets will be achieved. Depending how trends to target achievement are, scenario parameters (e.g. a rule set parameter representing a management

parameter) can be edited, thus simulating policy interaction. After that, the simulation can be continued. When editing scenario parameters, SITE automatically tracks these changes to ensure reproducibility and filing of simulation runs.

Third-party models are represented by model proxy objects that are made accessible inside application code via the passed *DynInfo* object. Depending on the application, an arbitrary number of sub models which are contained by a model management object (*ModelProxyMngr*) can be integrated. For details on the integration of sub models and the respective interface provided by the SITE framework see section 5.5.

5.1.3. Summary

The implementation of the SITE core engine already provides a solution to most of the listed requirements (see section 2). It houses all functionality for generic CA-based land-use modeling. Through the use of an established scripting language extended by a concise interface to manipulate the modeling data structures, it makes a maximum of functionality available for different modeling applications. No directives for specific modeling methodologies are made. Thus, applicability to a multitude of modeling projects is established. Expandability and maintainability are ensured by the consequent use of object-oriented programming paradigms. Innovative functionality compared to existing approaches are automated documentation of simulation runs and rule set parameterization as well as the possibility to interactively handle scenario analysis. Integrated modeling is supported via model proxy objects through which an interface to third-party models is made available (see paragraph 5.5 for a detailed description).

5.2. Model calibration and model testing components

Calibrating complex application rule sets is crucial to achieve valuable simulation results. The SITE framework provides generic functionality to find an adequate solution for application-specific parameter sets. Figure 5.7 shows the class layout of the *Calibration* component in connection with component *ModelTest*. The calibration component basically consists of a management object that possesses a reference to an algorithm capable of optimizing the defined parameter set. For all optimization algorithms to be used in the context of the SITE framework, a base class (*OptmzAlgr*) declaring a generic interface is provided. In the current version, an optimization procedure utilizing genetic algorithms (*OptmzAlgrGntc*) is implemented. Other algorithms can be added and used polymorphically. Optimization algorithms have a reference to the rule set object representing the rule set to calibrate. From this object, also the parameter set to optimize is retrieved. Via the rule set object, the optimization algorithm triggers the repeated reset and restart of simulation runs (hence the dependency of component *Calibration* from *SmltnDynamics*). After each single simulation run based on a specific solution for the parameter set to calibrate, the optimization algorithm evaluates the respective simulation result via its assigned objective function. In the SITE framework, the quality of a simulation result is assessed by comparison of a result map with a reference map. Depending on the application, different map comparison algorithms might be favorable. Map comparison algorithms to be used as objective functions for rule set calibration are not implemented in the *Calibration* component, but in the *ModelTest* component. Optimization algorithms utilize functionality provided by *ModelTest*.

In the SITE framework, assessing the quality of simulation results and thus the quality of

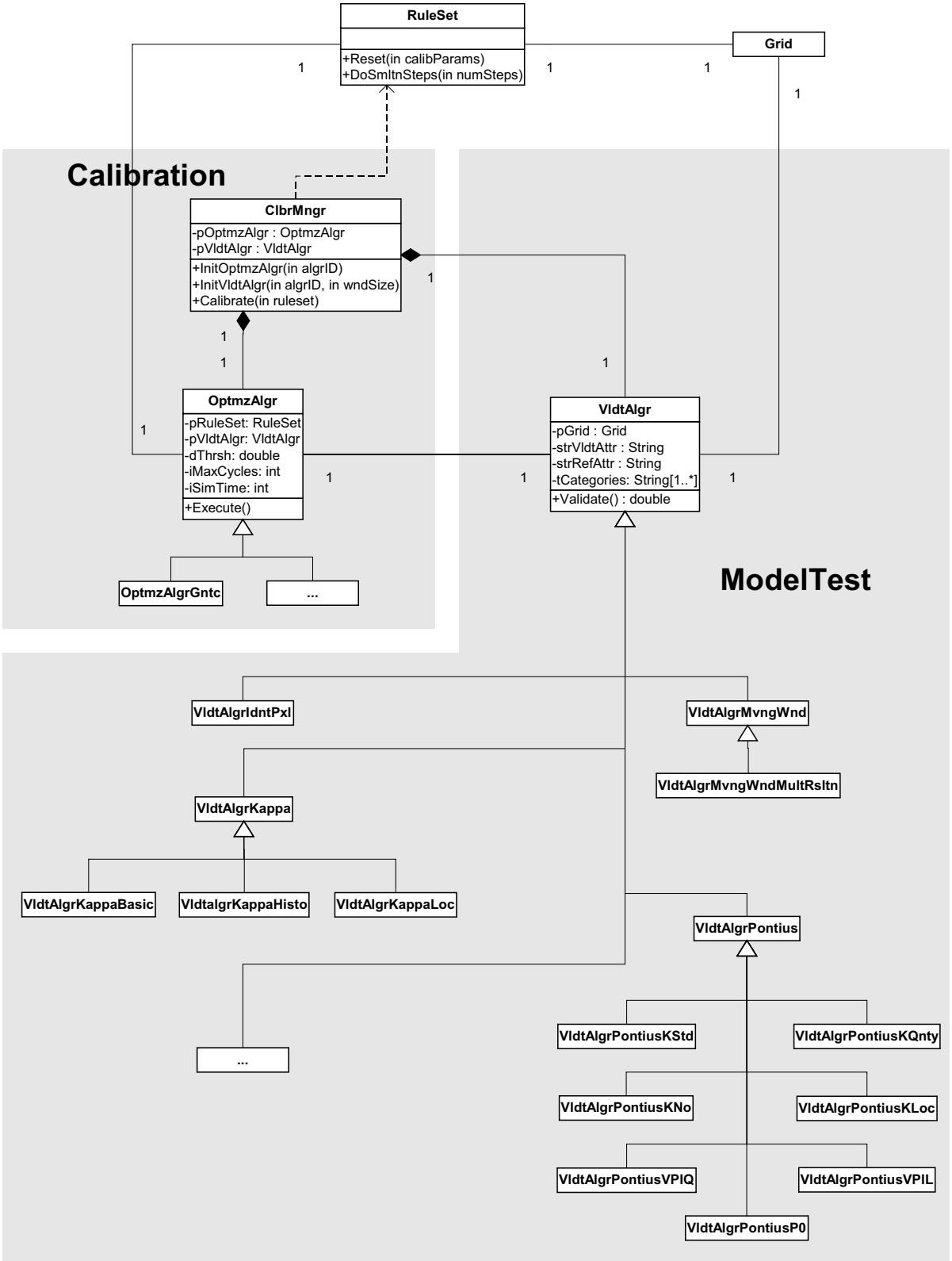


Figure 5.7.: Class layout of the SITE *SmltnDynamics* Class layout and connection between the SITE calibration and model testing components. In this constellation, the model testing component is used to provide an objective function for the calibration methodology.

Table 5.1.: List and description of map comparison algorithms implemented in the SITE *ModelTest* component.

Class name	Algorithm description
Simple pixel comparison	
VldtAlgrIdntPxl	Compares all corresponding grid cells and calculates the ratio of the number of all matching cells and the number of all cells.
Pixel-based Kappa	*Reference: Pontius (2000)*
VldtAlgrKappaBasic	Calculates Standard Kappa value based on contingency matrix
VldtAlgrKappaHisto	Calculates Kappa value based on contingency matrix specifically regarding quantification error
VldtAlgrKappaLoc	Calculates Kappa value based on contingency matrix specifically regarding location error
Moving window-based Kappa	*Reference: Pontius (2002)*
VldtAlgrPontiusKStd	Standard Kappa value
VldtAlgrPontiusKNo	Kappa value without considering quantification and location error
VldtAlgrPontiusKQnty	Kappa value specifically regarding quantification error
VldtAlgrPontiusKLoc	Kappa value specifically regarding location error
VldtAlgrPontiusVPIQ	Value of perfect information of quantity
VldtAlgrPontiusVPIL	Value of perfect information of location
VldtAlgrPontiusVP0	Observed proportion correct
Moving window-based ratio	*Reference: Kuhnert et al. (2005)*
VldtAlgrMvngWnd	Ratio of correctly classified cells for specified moving window size
VldtAlgrMvngWndMultRsltn	As above, but integrated over multiple resolutions up to moving window size
Figure of Merit	*Reference: Klug et al. (1992); Pontius et al. (2007)*
VldtAlgrFigMerit	Pixel-based figure of merit, assessing actual change

the underlying models is done by comparing the simulation result maps with a reference map (e.g. historical land use maps throughout rule set calibration). For this purpose, the *ModelTest* component provides a number of map comparison algorithms (Fig. 5.7). All algorithms share the same base class *VldtAlgr* which basically holds the main information required by all concrete map comparison algorithms. In addition, it defines an interface valid for all algorithms and allows their equal utilization by clients (e.g. an optimization algorithm from the *Calibration* component) through polymorphism. The *VldtAlgr* base class includes a reference to the *Grid* object (component *SmltnEnvironment*) which is the data structure on which all map comparison algorithms operate. The class hierarchy of map comparison algorithm can be arbitrarily extended. The *ModelTest* component features different families of algorithm like direct pixel-to-pixel comparisons, pixel-based comparisons using the Kappa with regard on quantification or location error (Pontius, 2000), moving window-based Kappa algorithms also considering quantification and location error (Pontius, 2002) and moving window-based algorithms based on the quantities category occurrences inside an actual window (Kuhnert et al., 2005). Table 5.1 lists all map comparison algorithms that are currently available in the SITE

framework. These map comparison algorithms typically deliver result values between 0 (no match) and 1 (perfect match, identical maps). Results delivered by Kappa-based algorithms need to be interpreted differently: A value of 1 means perfect classification while values out of the interval $]0;1[$ indicate, that the proportion of cells classified correctly is greater than the expected proportion classified correctly due to chance. A value of 0 or smaller means that there are no more correctly classified cells than there would be due chance. The categories to be regarded for map comparison can be selected freely. All categories that are not selected are considered belonging to a rest category.

So far, only algorithms comparing categorical maps are implemented. However, the *ModelTest* component data structures are technically not restricted to categorical maps. Integration of other types of validation algorithms like the ROC method (Pontius and Schneider, 2001; Pontius and Pacheco, 2004) which assesses the quality of suitability maps or algorithms applying an amount of fuzziness to both location and category of cells (Power et al., 2001) is possible.

The *ModelTest* component is designed to be adopted at all places where the quality of simulation results needs to be assessed. The SITE framework provides the functionality to assess the quality of simulation results directly via its GUI. In addition, the provided map comparison algorithms are used by the *Calibration* component as objective functions (as depicted in Fig. 5.7).

With the calibration and model test components, the respective methodology is consistently integrated into the SITE framework. All models operated within the framework can utilize the functionality. The implementation satisfies the respective requirements listed in section 2. An application of the calibration and model test components is provided by Mimler and Priess (2007). Calibration methodology is restricted to algorithms that find an optimal or adequate solution for a set of rule set parameters with respect to an objective function. In the particular case of the SITE framework, map comparison algorithms have to serve as objective functions.

5.3. Clients

All SITE components introduced so far physically are libraries that have to be linked dynamically. Consequently there is no fixed directive how to put the functionality they offer into operation. Typically, software is operated either using a graphical user interface or via the command line. Both ways have their specific advantages and drawbacks depending on the respective software is being used for. The SITE component architecture is suitable for both kinds of clients. While due to the requirement of user friendliness a detailed graphical user interface has been created, there is also a command-line client available which can be used for time-consuming calculations (typically rule set calibration runs) and for integration of the SITE framework into batch processes.

5.3.1. Graphical user interface

The GUI developed for SITE is designed to enable simple handling of the rather complex framework. Hereby, the requirement of simple applicability is addressed. It supports both the utilization by non-expert users and by rule-set developers. The SITE components are linked dynamically to the GUI. The dependencies between client and the components and the inter-component dependencies are displayed in Fig. 4.2. For inputs and for controlling

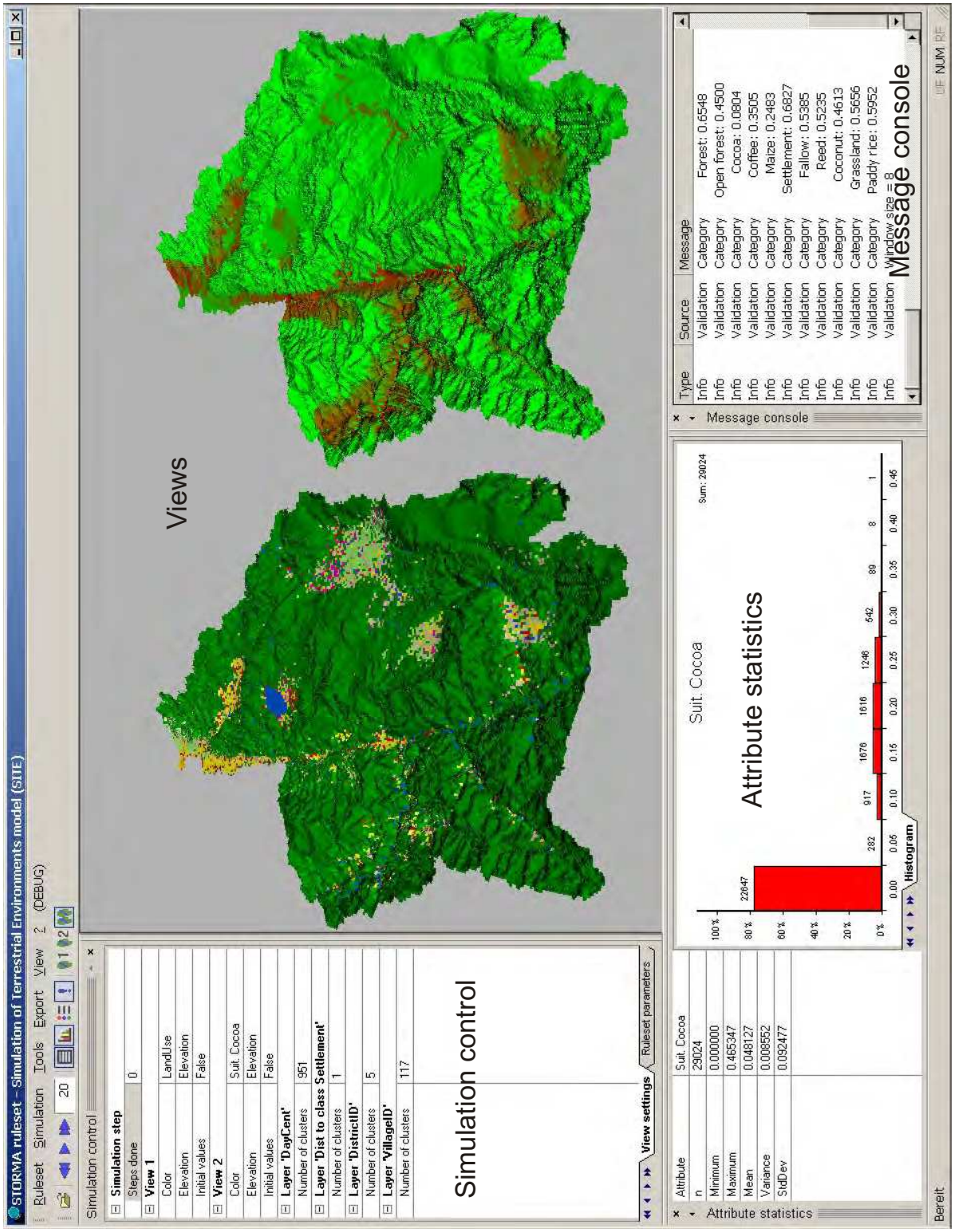


Figure 5.8.: Snapshot of the SITE graphical user interface in typical configuration. The left view shows current land use, the right view a suitability map. For both views, elevation is displayed on the z-axis.

simulation runs, the GUI directly calls the respective methods from the component interfaces. The visualization of simulation results is based on an observer pattern (Gamma et al., 1995) through which respective GUI elements are notified as soon as an update is necessary. The GUI is based on the Microsoft Foundation Classes (MFC). Additional functionality like more elaborated GUI elements (e.g. dockable control bars) and a flexible layout which is restored at each startup are provided by a commercial extension to the MFC (Business Components Gallery). Consequently, using SITE in combination with its GUI is only possible on Windows systems.

A snapshot of the SITE GUI is given by Fig. 5.8. As depicted, there are two different kinds of visualization of simulation data. The first one is the view on the simulation grid, where one attribute, which can be selected by the operator, is displayed for the entire area using a specific color code. For continuous data (e.g. elevation), attribute values are coded by a transition of colors from green (lowest value) to red (highest value). Categorical data (e.g. land use classes) use a color coding that has to be specified in the rule set script (see appendix A). Two instances of this view can be displayed simultaneously to facilitate the examination or comparison of two different attributes. The views are capable of rendering data in three dimensions and can be freely navigated and zoomed. Therefore, it is possible to use the z-axis to display a second attribute value inside each view. Typically, one would select the elevation attribute (if available) to get a true representation of the project area, but the 3D functionality might also prove useful for other data, e.g. to check whether distance maps are calculated correctly. For the rendering of the simulation grid, it is represented by a graphical model which is displayed using the OpenGL graphics hardware interface. The second aspect of data visualization is the display of basic attribute statistics. Statistics maintained for each attribute include simple descriptive statistics (mean, minimum, maximum, variance, standard deviation) and a distribution of attribute value frequencies.

The current status of a simulation run is displayed by the control bar labeled as "Simulation control" in Fig. 5.8. It consists of two tabbed pages. The first one gives information about the current simulation time step and about existing thematic layers and allows the selection of attributes for display. The second page allows editing the value of selected rule set parameters. Which rule set parameters appear is specified inside the application rule set script. It is this functionality which makes SITE accessible to a wider range of users. Two types of users are addressed: expert users capable of writing their own application code and non-expert users working on finished but parameterizable rule sets.

The basic control of simulation runs is done via the GUI tool bar where an operator can load and reset a simulation and start single or multiple simulation steps. Further control and configuration of input and output is provided by specific menu items. Via the menu bar, specific tools for managing interactive scenarios and assessing the quality of simulation runs by means of map comparison procedures, are provided.

By default, user system messages and warnings are directed into a message console. In addition, a file labeled "SITELog.txt" is written containing the same output as the message console. Analogous to the other GUI elements, an observer pattern is used for the output of system messages and warnings. Output devices are specified in the code of the respective client.

5.3.2. Command-line client

While the GUI is specifically suited for interactively performing simulation runs and assisting in the development of application rule sets, the use of a command-line based client addresses the application of the SITE framework with particular consideration of productivity and automation. The command-line client is intended to be used inside batch processing frameworks and for time-consuming jobs, especially for the calibration of application rule sets. Therefore, a special dependency of the command-line client from the *Calibration* component is established (see Fig. 4.2). Compared to the GUI, the SITE command-line client is a very lightweight application which can easily be ported to different platforms.

When using the command-line client, system messages and warnings are directed to the standard output and standard error channels respectively. The output can be redirected to other devices like e.g. files.

5.4. Import/export, database connectivity

A SITE application is initialized based on data stored in a database table. Database clients for Microsoft Access and MySQL are available. The database connection details (database name or file path and table name) need to be specified in the main module of the application script. Initialization data is read upon initialization of the simulation grid.

In its current version, the SITE framework writes output data to files as comma separated lists (using semicolons as separators) which can be easily imported into other software for evaluation. There are two types of output, output of values of a selected attribute at the current step in simulation time and output of time series for selected attributes. Attribute selection can be done via the respective menu items in the graphical user interface or alternatively by Python methods provided by the system/application interface in the application script. The latter alternative has to be used if the SITE is operated by its command-line client.

Configuration of components and functional parts is done based on XML files. Configuration files need to be edited whenever basic selection have to be made (e.g. selection of the database client via file *DBConfig.xml*, specification of working directories for DayCent integration via file *DayCentDrvrConfig.xml*, etc.).

System messages are directed to specific message devices depending on the SITE client used. If SITE is operated using the GUI all messages are displayed in their own message console GUI element. In addition, messages are written to a log file. The command-line client directs messages to the standard output and standard error channels. Specific messages are issued by the calibration component during model calibration runs. This output contains all data associated with a calibration run.

5.5. Integration of third-party models

One of the main requirements defined for the SITE framework is the integration of sub models. Sub models provide services for the superior land use model by performing specific calculations and making their results available for the land use model which in turn uses these results in its own process of decision making. This way a feedback loop between the land use model and its sub models is established.

The technical realization of integrating external sub models basically depends on complexity and character of the models to integrate. In case of very low complexity, e.g. if the sub model is none more than a regression function, the best way of integration is to simply code the model in the application script. However, for more complex models that already have been implemented as software this is not applicable. A mechanism is required by which it is possible to configure and invoke the sub model and to read in the respective modeling results. Since SITE is designed to be a generic framework for regional land use modeling applications, it is also desirable to establish a universal solution for the integration of existing sub models. Although this goal is rather impossible to achieve for all different kinds of models, by making a number of compromises an adequate solution can be achieved. Things get particularly complicated if it is desired to integrate a model that defines its own spatial grid, which, among other difficulties, results in the need of synchronization with the SITE grid.

SITE development occurred largely parallel to the development of the first major SITE application, a rule set to model the stability of tropical rain forest margins in the context of the STORMA project (STORMA, 2003). This application required the integration of the DAYCENT agro/ecosystem model (Parton et al., 1998) to calculate the productivity and soil parameters for single grid cells representing crop areas. The DAYCENT model represents a class of models that have no explicit spatial reference and are only valid for one single location, or, referring to the spatial explicit SITE framework, for one of its grid cells. In addition, it can be assumed that there is no influence by neighboring grid cells when performing DAYCENT

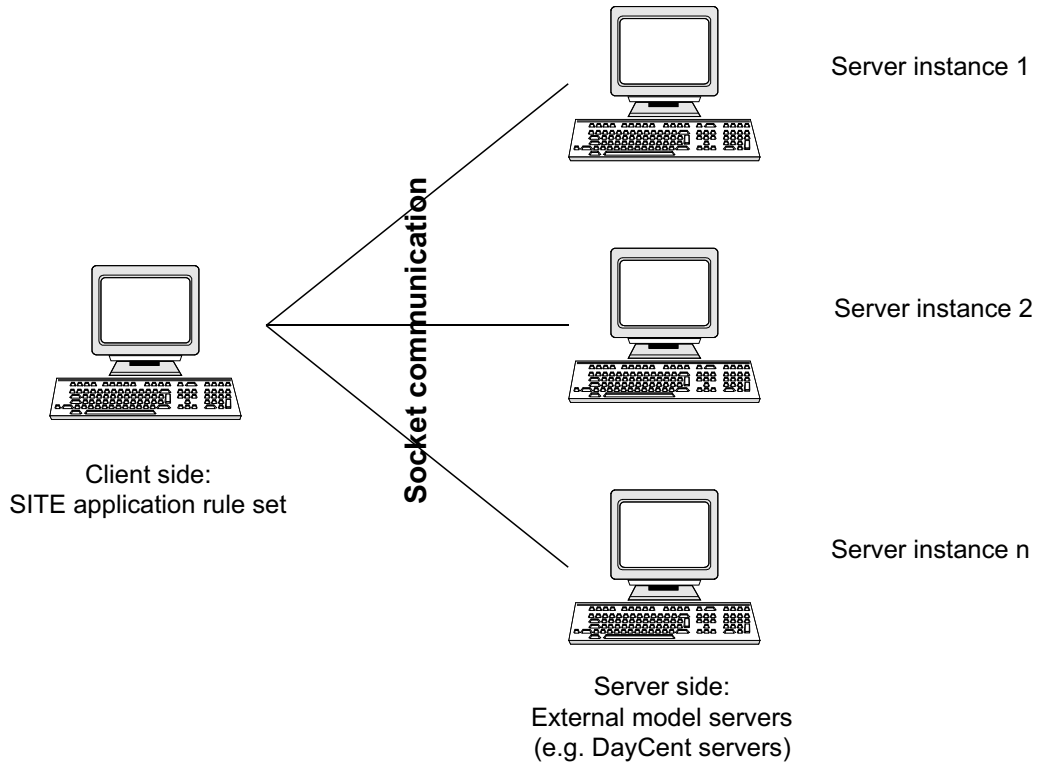


Figure 5.9.: A client/server approach is used inside the SITE framework to integrate external models. External models are wrapped and operated via server applications. The actual land use model has the client role and configures and send modeling jobs. The system is capable of parallel processing.

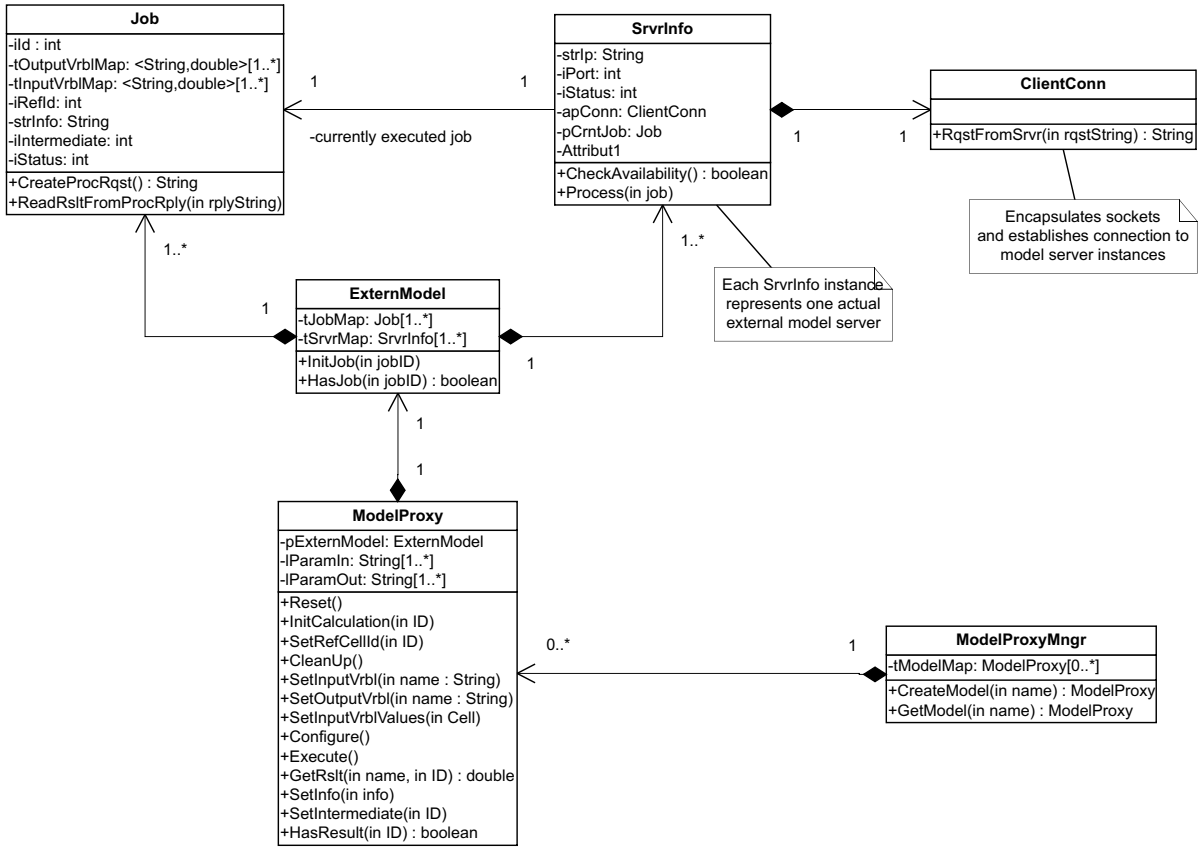


Figure 5.10.: Class diagram of the client side of the SITE sub model integration. The *ModelProxy* methods to configure and operate sub model can also be found in the classes ExternModel and *Job* and are exposed the application domain. Each instance of *SrvrInfo* is connected to one server application.

calculations for a particular cell. Accepting these criteria, no explicit spatial reference and no influence from neighboring cells, as compromises, it is possible to define an interface to integrate a variety of external models. Technically, the fact that sub model calculations for single grid cells can be seen as isolated processes, they have the property of being concurrent and thus can be executed in a parallelized framework with a significant gain in run time.

The technical framework to integrate third-party models established in SITE is depicted by Fig. 5.9. It is based on a client/server architecture with the SITE application (land use model) acting as client that requests modeling jobs from the actual third-party models on the server side. Communication between client and the model servers is established via sockets. Socket functionality is encapsulated in a library providing simple functionality to send and receive job requests and replies. Based on the integration of the DAYCENT model, an interface to configure and execute processing jobs and receive the respective modeling results for use in the application has been created. Additional effort was put into keeping this interface generic for use with other models.

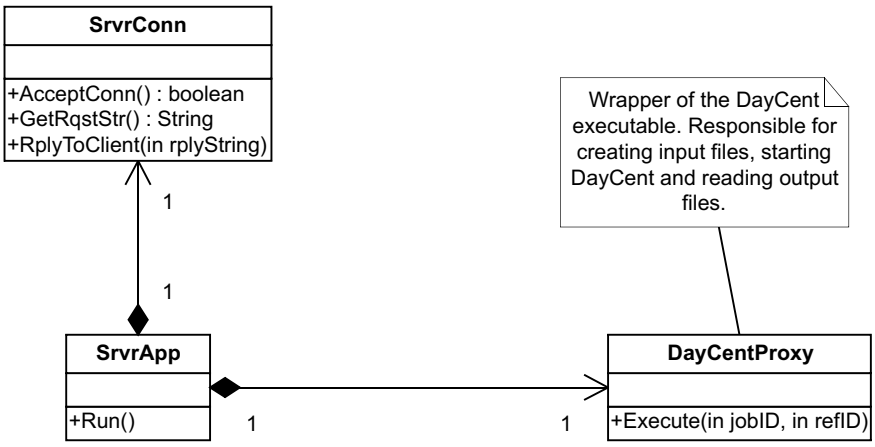


Figure 5.11.: Integration of the DayCent model as an example for integration of third-party models. A server instance receives calculation requests. Configured jobs are started by a wrapper instance including the actual third-party model.

5.5.1. Client side

The interface is implemented in the *ModelProxy* class. An instance of this class represents the use of a specific third-party model (e.g. DAYCENT). The methods forming the interface can be seen in class *ModelProxy* in Fig. 5.10 which shows the static structure of the sub model framework client side. All displayed classes are part of the *SmltnDynamics* component, the classes *ExternModel*, *Job* (processing job), *SrvrInfo* (server info) and *ClientConn* (client connection) being implemented in libraries other that the SmltnDynamics DLL, but statically linked to it. The sub model interface from class *ModelProxy* is exposed to the Python language to enable its use inside application rule set scripts. *ExternModel* is responsible for creating and configuring the single processing jobs (e.g. for single grid cells), for configuring the single server instances that do the actual processing and for distributing processing jobs to server instances. Each object representing a processing job holds the complete configuration data. Based on this, it provides two functions, *CreateProcRqst()*, which generates a request string transmitted to a server instance, and *ReadRsltFromProcRply()* (read result from processing reply), that is able to interpret the reply string received from a server instance after the job processing is finished. Actual server instances are represented and accessed by objects of type *SrvrInfo*. These objects provide information about the connection to the server (using IP address and port) and whether a server is available (function *CheckAvailability()*). To start a process, they define a function *Process()* which takes a reference to the job to be processed as argument. The job object, in turn, is capable of providing a request string to be sent to the server instance. Each *SrvrInfo* object starts a separate thread in which the request string is transmitted to the server. Consequently, it is not necessary to wait until the server instance replies after finishing the job processing. Instead, the *ExternModel* object can directly send the next job request to an idle serve, if available. Availability and status of model servers is indicated by a set of status flags (unknown, not available, ready, processing). In addition, each job object indicates its current status using a set of flags (initialized, configured, processing, finished). Based on these flags, the *ExternModel* object carries out the distribution of processing jobs. The communication between client and servers is done using functionality

provided by a *ClientConn* object that encapsulates socket functionality behind its function *RqstFromSrvr()* (request from server).

5.5.2. Server side

Which servers are potentially available has to be defined prior to a SITE simulation in an XML file, where the IP address and port number of each instance have to be specified. Currently, it is not possible to add new servers during a simulation run. However, out of this specified set, not all servers must necessarily be running. The system is robust concerning failure and temporary unavailability of servers. In this XML file, a directory for intermediate data used by a third-party model can be defined. The DayCent model requires this information to store intermediate files which it needs to resume its status based on former simulation steps.

The actual job processing takes place on the server side (Fig. 5.11). Job requests are received by an object of class *SrvrConn* (server connection), which is the counterpart of *ClientConn* and is implemented in the same messaging library. It defines the functions *AcceptConn()* (accept a connection from a client), GetRqstStr() (receive an incoming request string) and *RplyToClient()* (send a reply string containing processing results back to client). A *SrvrApp* (server application) instance is capable of interpreting request strings and creating reply strings. This *SrvrApp* object then configures the third-party model using a model proxy object. The proxy object is a wrapper of the actual third-party model. Due to the variety of models that can be used, this wrapper has to be a specific development. Figure 5.11 shows the wrapping of DAYCENT model. DAYCENT needs a set of specific input files and preprocessing steps typically performed manually or inside batch jobs. It delivers results by means of result files and files describing its intermediate status. The *DayCenProxy* wrapper is designed to carry out all these steps programmatically (e.g. starting the actual DAYCENT executable). An advantage of this approach using a wrapper is that it is not necessary to manipulate the code of the integrated model which avoids the introduction of additional complexity.

5.6. Extendibility and portability issues

The entire SITE system is designed to be extendable. The component-based architecture defines clear functional units and encapsulates the respective complexity. Implementation of components themselves has been done with strict use of the object oriented programming paradigm. Design patterns (especially factory, observer, iterator) have been used heavily. The system/application interface can be extended by simply adding new classes or class methods to expose to the Python language.

Another focus during the development of the SITE framework lay on the minimization of efforts necessary if components need to be ported to other platforms, in particular to Linux systems. With the exception of the graphical user interface, which uses the Microsoft MFC library, only libraries have been used that are available on both platforms. However, the technical solutions of how dynamic linking of components differs between Windows and Linux, which requires a certain amount of work when porting the system.

6. Discussion

Land use and its dynamics are determined by a wide variety of factors. Since research on land-use dynamics is mostly interdisciplinary research, a modeling framework used in this context consequently has to be able to reproduce and utilize interactions between the different factors determining land-use change. The increasing availability of sectoral models (e.g. for population dynamics, crop growth, ecosystem services) favors a modular assembly resulting in integrated modeling systems. Since SITE was developed to be used as such an integrative tool, its value for the modeling community has to be benchmarked largely based on its capability of model integration and the way this capability is combined with other innovative features.

SITE provides two different ways of model integration. On the one hand, integration can be achieved via a specifically designed interface. This interface facilitates the coupling of complex models that are available in the form of components. Component-based coupling of models has become a popular approach in ecological modeling, as it supports modularity, and interchangeability of integrated models (He et al., 2002; Argent, 2004). However, SITE advances this functionality by establishing a mechanism to feed back results from the coupled model to the calling instance. Beside the capability to establish feedback loops, the SITE model coupling interface supports parallel processing, provided that the modeling methodology allows concurrent processes (e.g. the calculation of yields for crop cells which does not interfere with any processes in the cell neighborhood). This results in a significant reduction of processing time for simulation runs. As a second method, model coupling can be achieved by creating extensions to the SITE scripting language. For the integration of relatively simple models (e.g. regression models, functional dependencies) this method is even superior to the component approach, since respective language extensions can be implemented quickly. The implementation of feedback loops is also possible for the latter case. The applicability of the integration and feedback functionality has been shown in case studies, where SITE was linked to an agro-ecosystem model and a model integrating ecosystem services (Priess et al., 2007).

The advanced possibilities for model integration are combined with a generic land-use modeling platform. As for model integration, a number of solutions for generic platforms are available (e.g. GEOMOD2, Pontius et al., 2001; SELES, Fall and Fall, 2001). However, these solutions gain simplicity at the cost of flexibility (e.g. the SELES domain-specific language requires the definition of so-called landscape events and thus does not allow the use of other modeling methodology). Generic applicability is ensured by SITE through its central design characteristic, which is the strict separation of implementation and application into system and application domains with the use of a modern high-level scripting language (Python) for the implementation of land-use modeling applications. The Python language was extended to match requirements specific to land-use modeling (e.g. by adding classes for the simulation grid, cells and attributes). Thus, a full-fledged programming language is available for model implementation; no restrictions remain regarding modeling methodology as opposed to existing solutions. In addition, Python is already being used as scripting language in a number of established software products with significance to the land-use modeling community (e.g. GIS-Software), which enables further possibilities with respect to synergies with these

products.

Model calibration, although indispensable (Boumans et al., 2001; Oliva, 2003; Straatman et al., 2004), is not integrated in most of the available modeling frameworks. In the SITE framework, calibration functionality is implemented in an integrated system component. In the current version, only genetic algorithms are available, but the component can be extended to house additional methodologies. Calibration algorithms used by SITE aim to find an optimal or adequate solution of an arbitrarily defined parameter set (defined in the application script) based on an objective function. The objective function, in turn, can be freely selected from another system component (*ModelTest*), which provides a selection of map comparison algorithms. This design enables model operators to freely combine optimization algorithms and algorithms for objective functions. Apart from the process of parameter selection, which requires expert knowledge of the underlying rule set, SITE is capable of automated rule set calibration. The component that implements the different map comparison algorithms, can also be used independently from the calibration as an integrated tool for model tests based on map comparison methodologies. The SITE calibration methodology seamlessly interacts with the generic modeling functionality and integrated models, thus it can be used for all applications that are operated within SITE. Moreover, the calibration methodology is not restricted to the land-use model, but also allows to simultaneously calibrate different integrated models.

The explicit representation of scenarios in SITE is a further innovation in the field of land-use modeling frameworks. Performing a simulation in SITE always implies to use the underlying model rule set in combination with a quantified scenario. Model rule set and quantified scenario are separate instances. This concept allows simulation runs under different scenarios without having to edit model code, thus improving system handling and facilitating maintenance. With the possibility to interactively handle and alter scenarios based on an analysis of interim simulation results, it was possible to overcome a major limitation of scenario analysis (Alcamo et al., 2006).

Although the SITE concept of integrating a scripting language significantly facilitates model implementation, programming knowledge is still requested. To enable scientists without programming knowledge to also work with the SITE framework, a detailed graphical user interface (GUI) has been created. In this GUI, arbitrary rule set parameters (i.e. variables in the Python application code) can be edited directly. Thus, one can distinguish between two different application levels for SITE: (i) model development, performed by users that are capable of writing application domain code, and (ii) application of complete parameterizable models via the GUI. The latter application level is open for non-expert users. With this compromise, the requirement of simple accessibility to researchers of different scientific background is satisfied. Furthermore, model handling and operation is strongly facilitated by the design of the SITE GUI, which provides two 3-dimensional views on the simulation grid. All attributes of a case study grid can be displayed. With these features, the GUI is also capable of supporting model development since it can give rapid feedback through its configurable views. A high communicability of simulation results is provided. Although the SITE GUI does not directly support definition of rules like other frameworks, that e.g. provide a graphical interface for the definition of rules (Costanza et al., 1998) or model component assembly (Filippi and Bisgambiglia, 2004), it is open for further development in that direction. Another innovative contribution to usability is the automated logging of simulation and model settings for every simulation run, which guarantees reproducible results combined with minimum administration efforts.

Much effort was laid on the architectural design of the SITE framework. The system archi-

tecture was developed with respect to the requirements posed for integrative tools. The SITE component design is based on a study by Endejan (2003), who developed a system architecture for integrated simulation-based assessment of global change, emphasizing the advantages of a component-based approach. In fact, recent developments in integrated modeling show that there is a trend toward model encapsulation into components (He et al., 2002; Argent, 2004). Compared to the architecture proposed by Endejan (2003), the SITE architecture represents an advancement with respect to land-sue modeling in the context of interdisciplinary projects. Due to the target of usability, the design is more compact, as several components have been merged (e.g. documentation and simulation-specific components), while on the other side additional components were introduced, based on the set of scientific and technical requirements (e.g. calibration, model test, simulation environment and dynamics components).

In contrast to other publications available on land-use modeling frameworks, this study stresses the importance of a well designed architecture, accurate implementation and the overall software development process for the final system. These technical aspects ensure that the framework can be successfully applied for modeling applications while at the same time being open for further developments in both information and land-use change research. A long term availability of the SITE framework for land-use modeling applications can be expected.

The development of SITE included a couple of innovations in the field of land-use modeling. In particular, however, it was the combination of these features that made SITE an innovative and valuable tool for land-use modeling. As it enables flexible integration of models, including the implementation of feedback loops, together with a generic platform for the formulation of land-use models it is a flexible integrative tool in interdisciplinary land-use modeling projects and an advancements to existing solutions. In addition, it is the only comprehensive approach so far available (see Table 2.1). The underlying architecture ensures expandability of the system and the integration of new functionality, thus enabling long-term usage.

7. Conclusions and outlook

In this study, the design and implementation details of SITE were introduced and discussed with respect to their contribution to research on land-use dynamics. SITE was planned as an integrative tool for interdisciplinary search projects. This application scenario implied a number of specific requirements, among which the capability of integration was the most important one. Besides that, additional requirements could be identified, e.g. generic applicability, integration of calibration and model test functionality or high usability of the system and communicability of simulation results. A review of existing modeling frameworks revealed that none of them could match all of our requirements. Particular emphasis during the implementation was laid on a component-based architecture and use of the object-oriented programming paradigm. The system was designed to be expandable, thus enabling long-term usability and the possibility to integrate further developments.

To allow model integration, a generic interface for the coupling of models was implemented. This interface enables the feedback of modeling results to the calling instance. In addition, it supports parallel processing, provided that this is allowed by the modeling methodology. In addition to the model coupling interface, integration of sub models is also possible at the level of the SITE application scripting language.

SITE provides a generic modeling platform by separating general modeling functionality from the specification of actual model semantics (modeling applications). For the implementation of modeling applications, SITE resembles and enhances the concept of domain-specific languages by using an established and widely used scripting language (Python) for the actual implementation of land-use models. The functionality of the scripting language was extended to match land-use modeling requirements. With this approach, no fixed guidelines for specific modeling methodology are made, thus providing a maximum of flexibility. Integration of models is possible in this generic context.

Unlike other land-use modeling frameworks, SITE integrates functionality to automatically calibrate models. Calibration in SITE is understood as finding an optimal or adequate solution for a freely definable parameter set based on an objective function that is provided via a SITE component housing a collection of map comparison algorithms. Typically, calibration is performed using historical land-use maps as reference. The SITE calibration component is designed to contain an arbitrary number of optimization algorithms that can be freely combined with map comparison algorithms acting as objective function. The system design also include all integrated models simultaneously in the calibration process.

SITE implements an explicit representation of quantified scenarios. Model semantics and scenario data are two separate instances, thus simulations can be performed based on an arbitrary combination of model and scenario. Scenarios in SITE are handled interactively. It is possible to stop a simulation run at a predefined step and to evaluate if simulation targets are likely to be reached. Depending on the outcome of that analysis, it is possible to edit scenario parameters (e.g. management parameters). Thus, interaction of policy makers can be simulated. Conceptually, interactive scenarios establish a feedback loop to the model driving forces. This way, a major limitation in the field of scenario analysis could be overcome.

Although model development requires programming knowledge, SITE is open to be used by scientists from a wide variety of disciplines as it can be operated via a graphical user interface. It is also possible to expose the parameterization of a model via the GUI, enabling users to simply change parameters without having to edit the model code. All changes made by users via the SITE GUI are automatically recorded, thus guaranteeing reproducibility of simulation results.

The SITE framework was designed to overcome limitations of previous approaches. The entirety of innovations make it a valuable tool in the interdisciplinary field of land-use modeling, especially due to its high degree of integration it provides for components of the land system. SITE has been applied in case studies in the context of the collaborative research center "Stability of Rainforest Margins in Indonesia" (STORMA, SFB 552). At present, applications for an Indian and a Mongolian region are being developed.

In its current state, the SITE model has proven to be a valuable tool in the field of land-use modeling. In addition to the case studies described in this thesis, SITE is already being applied in other research projects, e.g. for regional land-use modeling in Mongolia ("Integrated Water Resource Management for Central Asia: Model Region Mongolia", http://www.iwrm-momo.de) and India.

Nonetheless, there is large potential for further developments and improvements. Since SITE has been designed expandable, users can expect to be able to rapidly take advantage from new developments, while at the same time long-term usability is ensured. New features that are currently being implemented are a closer coupling to databases and the integration of tools supporting the analysis of simulation results. Mid- to long-term improvements could be e.g. the establishment of a graphical model builder on top of the SITE scripting language or the integration of parameterizable model building blocks.

Besides the large number of imaginable developments and improvements on the SITE system side, the framework will play an important role in the development of a generalized regional land-use model that is applicable and parameterizable for a variety of world regions. The development of a generalized regional land-use model is a long-term task due to a large number of remaining open research questions. SITE can support this process significantly by providing the ideal platform for development, analysis and test of model prototypes.

Bibliography

Alcamo, J., Leemans, R., Kreilemann, E., 1998. Global Change Scenarios of the 21st Century – Results from the IMAGE 2.1 model. Elsevier Science. Oxford.

Alcamo, J., Endejan, M., Kaspar, F., Rösch, T., 2001. The GLASS model: a strategy for quantifying global environmental security. Environmental Science and Policy, 4, 1–12.

Alcamo, J., Kok, K., Busch, G., Priess, J.A., 2006. Searching for the Future of Land: Scenarios from the Local to Global Scale. In: Lambin, E.F., Geist, H.J. (eds). Land-Use and Land-Cover Change. pp 137–155. Springer.

Argent, R.M., 2004. An overview of model integration for environmental applications – components, frameworks and semantics. Environmental Modelling & Software, 19, 219–234.

Balzert, H., 2000. Lehrbücher der Informatik. Bd.1. Software-Entwicklung: Lehrbuch der Software-Technik. Spektrum, Akad. Verlag. Heidelberg, Berlin.

Boumans, R.M., Villa, F., Costanza, A., Voinov, A., Voinov, H., 2001. Non-spatial calibrations of a general unit model for ecosystem simulations. Ecological Modelling, 146, 17–32.

Brovkin, V., Ganopolski, A., Claussen, M., Kubatzki, C., Pethoukov, V., 1999. Modelling climate response to historical land cover change. Global Ecology and Biogeography, 8, 509–517.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., 1998. Pattern-orientierte Software-Architektur: Ein Pattern-System. Addison-Wesley-Longman. Bonn, Paris.

Charney, J., Stone, P.H., 1975. Drought in the Sahara: A biogeophysical feedback mechanism. Science, 187, 434–435.

Chen, M., Norman, R.J., 1992. A framwork for integrated CASE. IEEE Software, 9, 18–22.

Clark, J.D., 1992. Modeling and simulating complex spatial dynamic systems: a framework for application in environmental analysis. Simulation Digest, 21, 9–19.

Costanza, R., Duplisea, D., Kautsky, U., 1998. Ecological modeling and economic systems with STELLA. Ecological Modelling, 110, 1–4.

De Vasconcelos, M., Zeigler, B., 1993. Discrete-event simulation of forest landscape response to fire disturbance. Ecological Modelling, 65, 177–198.

Endejan, M., 2003. Entwicklung einer Software-Architektur für Systeme zum integrierten simulationsbasierten Assessment des globalen Wandels. PhD thesis. University of Kassel. Kassel, Germany.

Engelen, G., Geertman, S., Smits, P., Wessels, C., 1999. Dynamic GIS and Strategic Physical Planning: A Practical Application. In: Stillwell, J., Geertman, S., Openshaw, S. (eds). Geographical Information and Planning. Advances in Spatial science. Springer.

Fall, A., Fall, J., 2001. A domain-specific language for models of landscape dynamics. Ecological Modelling, 141, 1–18.

FAO, UNEP (ed), 1999. Terminology for Integrated Resources Planning and Management. Food and Agriculture Organization / United Nations Environmental Programme. Rome, Italy / Nairobi, Kenia.

Farooqui, K., Logrippo, L., Meer, J.de, 1995. The ISO Reference Model for open distributed processing: an introduction. Computer Networks and ISDN Systems, 2, 1215–1229.

Filippi, J.-B., Bisgambiglia, P., 2004. JDEVS: an implementation of a DEVS based formal framework for environmental modelling. Environmental Modelling & Software, 19, 261–274.

Frelich, L.E., Lorimer, C.G., 1991. A simulation of landscape-level stand dynamics in the northern hardwood region. Journal of Ecology, 79, 223–234.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. Boston, San Francisco, USA.

Gao, Q., 1996. Dynamic modeling of ecosystems with spatial heterogeneity – A structured approach implemented in Windows environment. Ecological Modelling, 85, 241–252.

GLP, (ed), 2005. Science Plan and Implementation Strategy. IGBP Report No. 53 / IHDP Report No. 19. IGBP Secretariat, Stockholm.

He, H.S., Larsen, D.R., Mladenoff, D.J., 2002. Exploring component-based approaches in forest landscape modeling. Environmental Modelling & Software, 17, 519–529.

Heistermann, M., Müller, C., Ronneberger, K., 2006. Land in sight? Achievements, deficits and potentials of continental to global scale land-use modeling. Agriculture, Ecosystems & Environment, 114, 141–158.

Houghton, R.A., Boone, R.D., Melillo, J.M., Palm, C.A., Woodwell, G.M., Myers, N., Moore, B., Skole, D.L., 1985. Net flux of carbon dioxide from tropical forest in 1980. Nature, 316, 617–620.

Irwin, E.G., Geoghegan, J., 2001. Theory, data, methods: developing spatially explicit economic models of land use change. Agriculture, Ecosystems & Environment, 85, 7–23.

Klenner, W., Kurz, W.A., Webb, T.M., 1997. Projecting the spatial and temporal distribution of forest ecosystem characteristics. In: Proceedings GIS 97. pp 418–421. GIS World Inc., Ft. Collins, Colorado, USA.

Klug, W., Graziani, G., Grippa, G., Pierce, D., Tassone, D., 1992. Evaluation of long range atmospheric transport models using environmental radioactivity data from the Chernobyl accident: The ATMES report. Elsevier. London, UK.

Krauchi, N., 1995. Application of the model FORSUM to the Solling spruce site. Ecological Modelling, 83, 219–228.

Kuhl, F., Weatherly, R., Dahmann, J., 1999. Creating Computer Simulation Systems: An Introduction to the High Level Architecture. Prentice Hall PTR. Upper Saddle River, NJ, USA.

Kuhnert, M., Voinov, A., Seppelt, R., 2005. Comparing Raster Map Comparison Algorithms for Spatial Modeling and Analysis. Photogrammetric Engineering & Remote Sensing, 71, 975–984.

Lambin, E.F., Rounsevell, M.D.A., Geist, H.J., 2000. Are agricultural models able to predict changes in land-use intensity? Agriculture, Ecosystems & Environment, 82, 321–331.

Lambin, E.F., Geist, H.J., Rindfuss, R.R., 2006. Local processes with global impacts. In: Lambin, E.F., Geist, H.J. (eds). Land-Use and Land-Cover Change. pp 1–8. Springer.

Lindenschmidt, K.E., Rauberg, J., Hesser, F.B., 2005. Extending uncertainty analysis of a hydrodynamic-water quality modelling system using high-level architecture (HLA). Water Quality Research Journal of Canada, 40, 59–70.

Lorek, H., Sonnenschein, M., 1998. Object-oriented support for modelling and simulation of individual-oriented ecological models. Ecological Modelling, 108, 77–96.

Maxwell, T., Costanza, R., 1997. A language for modular spatio-temporal simulation. Ecological Modelling, 103, 105–113.

Mendelsohn, R., Dinar, A., 1999. Climate change, agriculture and developing countries: does adaptation matter? World Bank Research Observer, 14, 277–293.

Mimler, M., Priess, J.A., 2007. An automated rule set calibration procedure for spatially explicit land-use change modelling. Submitted to Environmental Modelling and Software.

Minar, N., Burkhart, R., Langton, C., Askenazi, M., 1996. The Swarm simulation system: A toolkit for for building multi-agent simulations. http://www.swarm.org.

Mladenoff, D.J., Host, G.E., Boeder, J., Crow, T.R., 1996. Landis: a spatial model of forest landscape disturbance, succession and management. In: Goodchild, M., Steyaert, L.T., Parks, B.O. (eds). GIS and Environmental Modelling. pp 175–179. GIS World Books.

Oliva, R., 2003. Model calibration as a testing strategy for system dynamics models. European Journal of Operations Research, 151, 552–568.

OMG, (ed), 1999. OMG Unified Modeling Language - Specification V.1.3 (June 1999). Object Mangement Group.

Ottermann, J., 1974. Baring high-albedo soils by overgrazing: A hypothesized desertification mechanism. Science, 86, 531–533.

Oxley, T., McIntosh, B.S., Winder, N., Mulligan, M., Engelen, G., 2004. Integrated modelling and decision-support tools: a Mediterranean example. Environmental Modelling & Software, 19, 999–1010.

Parton, W.J., Hartman, M., Ojima, D., Schimel, D., 1998. DAYCENT and its land surface submodel: description and testing. Global and Planetary Science, 19, 35–48.

Percivall, G. (ed), 2002. The OpenGIS Abstract Specification – Topic 12: OpenGIS Service Architecture. Open Geospatial Consortium. Wayland, MA, USA.

Pontius, R.G.Jr., 2000. Quantification Error Versus Location Error in Comparison of Categorial Maps. Photogrammetric Engineering & Remote Sensing, 66, 1011–1016.

Pontius, R.G.Jr., 2002. Statistical Methods to Partition Effects of Quantity and Location During Comparison of Categorial Maps at Multiple Resolutions. Photogrammetric Engineering & Remote Sensing, 68, 1041–1049.

Pontius, R.G.Jr., Pacheco, P., 2004. Calibration and validation of a model of forest disturbance in the Western Ghats, India 1920-1990. GeoJournal, 61, 325–334.

Pontius, R.G.Jr., Schneider, L.C., 2001. Land-cover change model validation by an roc method for the Ipswich watershed, Massachusetts, USA. Agriculture, Ecosystems & Environment, 85, 239–248.

Pontius, R.G.Jr, Cornell, J.D., Hall, C.A.S., 2001. Modeling the spatial pattern of land-use change with GEOMOD2: application and validation for Costa Rica. Agriculture, Ecosystems & Environment, 85, 191–203.

Pontius, R.G.Jr., Boersma, W., Castella, J.-C., Clarke, K., De Nijs, T., Dietzel, C., Zengquiang, D., Fotsing, E., Goldstein, N., Kok, K., Koomen, E., Lippitt, C.D., McConnell, W., Pijanowski, B., Pithadia, S., Sood, A.M., Sweeney, S., Trung, T.N., Veldkamp, A.T., Verburg, P., 2007. Comparing the input, output, and validation maps for several models of land change. Annals of Regional Science, in press.

Power, C., Simms, A., White, R., 2001. Hierarchical fuzzy pattern matching for the regional comparison of land use maps. International Journal of Geographical Information Science, 15, 77–100.

Priess, J.A., Mimler, M., Klein, A.-M., Schwarze, S., Tscharntke, T., Steffan-Dewenter, I., 2007. Linking deforestation scenarios to pollination services and economic returns in coffee agroforestry systems. Ecological Applications, 17, 407–417.

Pullar, D., 2004. SimuMap: a computational system for spatial modelling. Environmental Modelling & Software, 19, 235–243.

Rahman, J.M., Seaton, S.P., Cuddy, S.M., 2004. Making frameworks more usable: using model introspection and metadata to develop model processing tools. Environmental Modelling & Software, 19, 275–284.

Sagan, C., Toon, O.B., Pollack, J.B., 1979. Anthropogenic albedo changes and the Earth's climate. Science, 206, 1363–1368.

Sala, O., Chapin, F., Armesto, J., Berlow, E., Bloomfield, J., Dirzo, R., Huber-Sanwald, E., Huenneke, L., Jackson, R., Kinzig, A., Leemans, R., Lodge, D., Mooney, H., Oesterheld, M., Poff, N., Sykes, M., Walker, B., Walker, M., Wall, D., 2000. Biodiversity - global biodiversity scenarios for the year 2100. Science, 287, 1770–1774.

Schürmann, G., 1995. The evolution from open systems interconnection (OSI) to open distributed processing (ODP). Computer Standards & Interfaces, 17, 107–113.

Schulze, T., Strassburger, S., Klein, U., 1999. Migration of the HLA into civil domains: Solutions and prototypes for transportation applications. Simulation, 73, 296–303.

Snyder, J.P., 1987. Map Projections – A Working Manual. U.S. Geological Survey Professional Paper 1395. United States Government Printing Office. Washington, D.C., USA.

STORMA, (ed), 2003. Stability of Rainforest Margins. Application for continued funding of the SFB 552. University of Göttingen. Göttingen, Germany.

Straatman, B., White, R., Engelen, G., 2004. Towards an automatic calibration procedure for constrained cellular automata. Computer, Environment and Urban Systems, 28, 149–170.

Van Delden, H., Luja, P., Engelen, G., 2007. Integration of multi-scale dynamic spatial models of socio-economic and physical processes for river basin management. Environmental Modelling & Software, 22, 223–238.

Veldkamp, A., Lambin, E.F., 2001. Predicting land-use change. Agriculture, Ecosystems & Environment, 85, 1–6.

Villa, F., Costanza, R., 2000. Design of multi-paradigm integrating modelling tools for ecological research. Environmental Modelling & Software, 15, 169–177.

Voinov, A., Costanza, R., Wainger, L.A., Boumans, R., Villa, F., Maxwell, T., Voinov, H., 1999. Patuxent landscape model: integrated ecological economic modeling of a watershed. Environmental Modelling & Software, 14, 473–491.

Weimar, J.R., 1997. Simulation with cellular automata. Logos Verlag. Berlin, Germany.

Wenderholm, E., 2005. Eclpss: a Java-based framework for parallel ecosystem simulation and modeling. Environmental Modelling & Software, 20, 1081–1100.

White, R., Engelen, G., 1997. Cellular automata as the basis of integrated dynamic regional modelling. Environment and Planning B – Planning and Design, 24, 235–246.

White, R., Engelen, G., 2000. High-resolution integrated modelling of the spatial dynamics of urban and regional systems. Computer, Environment and Urban Systems, 24, 383–400.

White, R., Engelen, G., Uljee, I., 1997. The use of Constrained Cellular Automata for high-resolution modelling of urban land-use dynamics. Environment and Planning B – Planning and Design, 24, 323–343.

Wolf, J., Bindraban, P.S., Luijten, J.C., Vleeshouwers, L.M., 2003. Exploratory study on the land area required for global food supply and the potential global production of bioenergy. Agricultural Systems, 76, 841–861.

Woodwell, G.M., Hobbie, J.E., Houghton, R.A., Melillo, J.M., Moore, B., Peterson, B.J., Shaver, G.R., 1983. Global deforestation: Contribution to atmospheric carbon dioxide. Science, 222, 1081–1086.

A. System/application domain interface documentation

The SITE system/application domain interface is technically realized by extending the Python scripting language used to implement SITE application rule sets to also include the specific data structures used by SITE to represent the simulation environment. In the following a detailed reference on the language extensions established is provided. Three groups of objects can be discriminated: Object directly passed to the application domain by the application, objects that are managed by those passed objects and are made accessible, and object that can be instantiated independently in the application domain. To enhance readability of Python application rule sets, the Python classes and class methods do not necessarily have the same name as their C++ counterparts which in some cases might appear rather cryptic.

Base objects passed to the SITE application domain

These are the instances of the classes *Grid* and *DynInfo* that are the arguments of the basic Python functions *Initialize()* and *SimulationStep()* which are called by the SITE system domain. Both instances provide access to a number of other objects, representing simulation data structures (*Grid* object as representative of static aspects in a simulation) and dynamic components (*DynInfo* object).

Methods of class Grid

Method	Arguments	Return value
GetSizeX()		Number of grid columns
GetSizeY()		Number of grid rows
SetGeoreferene()	Upper left x and y coordinate Cell resolution	
GetCell()	x/y-coordinates	Cell object
GetAttr()	Attribute name	Attribute object
ApplyAttrChanges()		
SetInitState()		
CreateClusterLayer()	Layer name	Layer object
GetClusterLayer()	Layer name	Layer object

The Python *Grid* class is a representation of the underlying simulation grid one the application side. It provides access to all necessary data structures that can be manipulated by a rule set developer. Beside the simulation grid itself such data structures are single cells (accessible by their grid coordinates), attribute descriptor objects (accessible by attribute name) and thematic layer objects (also accessible by their name). The grid itself only offers limited possibilities of manipulation. Application can specify their location using the *SetReoreference()* method. With the methods *ApplyAttrChanges()* and *SetInitState()* attribute values

for all grid cells can be influenced directly. The first method forces all attribute values that are subject to change to actually change their values (the C++ counterpart of this method is called automatically after each simulation step), the latter one changes all attribute values to their initial value (i.e. their value at time of their creation) and thus can be regarded as a reset of the simulation grid.

Methods of class DynInfo

Method	Arguments	Return value
CreateExternModel()	Model name	Extern model object
SetScenario()		
GetScenario()		
SetTimeSeriesExportPath()	Path	
ExportTimeSeries()	Attribute name	

The *DynInfo* objects basically serves as carrier for objects dealing with dynamic aspects of a simulation run. It provides access to objects representing external models (method *CreateExternModel()*), model drivers (scenarios) and simulation time-dependent output (time series).

Objects accessible via base objects

Classes described in this paragraph provide the access to selected parts of the underlying modeling data structures. All instances of these classes are created by the SITE system domain and are accessed through the *Grid* and *DynInfo* objects passed to the application script entry functions. It is not possible to instantiate these classes on the application side.

Methods of class Cell

Method	Arguments	Return value
HasAttr()	Attribute name	TRUE or FALSE
SetAttr()	Attribute name	
	Attribute value	
	Immediate effect flag	
SetAttrNoData()	Attribute name	
	Immediate effect flag	
IsAttrNoData()	Attribute name	TRUE of FALSE
GetAttr()	Attribute name	Attribute value
X()		Cell x coordinate
Y()		Cell y coordinate

Cell objects are representatives of single grid cells. Methods of the cell class entirely deal with the access and the altering of cell attribute values, where the location access methods *X()* and *Y()* are wrappers of the *GetAttr()* function accessing the cell coordinates. Cell attributes can be of value NODATA (represented by -9999 to be compliant with standard GIS software). This value can be checked and set by specific methods (*IsAttrNoData()* and *SetAttrNoData()*).

All methods that set an attribute value require a so called immediate effect flag of boolean type as argument. By evaluating this flag, the method triggers whether a new attribute value is valid directly after its termination. In case of the immediate effect flag being of value FALSE, the new attribute value will be set after a call to the *Grid* method *ApplyAttrChanges()* or its C++ counterpart in the system domain.

All *Cell* objects are contained by the *Grid* object which provides a method to access them by specifying their coordinates. However, the usual way to access grid cell is to use one of the different iterator objects described below.

Methods of class Attr

Method	Arguments	Return value
SetColor()	Category number	
	8bit value for red portion	
	8bit value for green portion	
	8bit value for blue portion	
	Category name	
SetRandomColor()		
InitHistogram()	Value of lowest histogram class	
	Histogram class width	

Class *Attr* provides meta data on cell attributes. Its C++ counterpart is the class *AttrDscr* (component *SmltnEnvironment*). For each attribute available, the respective instance can be retrieved yb specifying the attribute's name. The interface of this class deals with attribute statistics and special configurations for category attributes (especially land use classes). These latter configurations mainly serve visualization purposes. By default, attribute values are assumed to be continuous and visualized using a color gradient from green (lowest value) to red (highest value). For category attributes it is advisable though not technically required to assign colors and category names to the different attribute values that can occur. This is done using the member function *SetColor()*.

Histograms are created and kept up to date for each attribute as part of the attribute statistics. Without explicit configuration using method *InitHistogram()*, the system assumes that histograms have a minimum class of value 0 and a class width of 1. Using these two parameter as method arguments, histograms for selected attributes can be customized.

Methods of class Layer

Method	Arguments	Return value
ClusterByEqualAttrValue()		Configurable cluster algorithm object
ClusterByClasses()		Configurable cluster algorithm object
ClusterConditional()		Configurable cluster algorithm object
DoClustering()		
GetClstr()	Cluster ID	Cell cluster object

Class *Layer*, instances of it accessible via the *Grid* object by a unique name, provides func-

tionality to aggregate similar grid cells to cell clusters (objects of class *Clstr*). The definition of similarity depends on the application and needs to be done by the rule set developer. The SITE framework provides three different algorithms to define similarity and assign grid cell to cell clusters. Objects representing these clustering algorithms can be accessed through a *Layer* object with the methods *ClusterByEqualAttrValue()*, *ClusterByClasses()* and *ClusterConditional()*. These objects provide specific interfaces to configure the represented algorithms. Calling the *DoClustering()* method starts the actual clustering process based on the underlying clustering rules and is used for both initial clustering and cluster update.

Existing cell clusters can be accessed with method *GetClstr()* by specifying the unique ID assigned during the clustering process. However, cell clusters of a thematic layer are typically traversed using a specific iterator (see below).

Methods of class Clstr

Method	Arguments	Return value
GetSignature()		Unique integer value assigned to the cell cluster
GetBestCell()		Cell that is representative for the cell cluster

The actual cell clusters are represented by class *Clstr*. The interface provides access to the unique cluster id and a grid cell that is representative for the cluster.

Methods of class ClstrAlgorithmEqualAttrVal

Method	Arguments	Return value
SetClusterAttr()	Attribute name	

This cluster algorithm aggregates all grid cells that have the same value for the attribute specified by method *SetClusterAttr()*. To ensure that the algorithm works properly, only categorial attributes should be used. In the resulting cell clusters, the category value is used as the unique cluster ID.

Methods of class ClstrAlgorithmClassify

Method	Arguments	Return value
CreateClasses()	Attribute name	
	Minimum class	
	Maximum class	
	Class width	
	Flag: TRUE for categorial values, otherwise FALSE	

This cluster algorithm performs the aggregation of grid cells based on classification rules specified by method *CreateClasses()*. An arbitrary number of attributes can be used. Therefore this method needs to be called repeatedly passing the respective attribute name and classification information. It is also possible to define classification schemes for one attribute that do not have equal class spacing. To do this, the *CreateClasses()* method can be called repeatedly

using the same attribute name, but different values for class minimum, class maximum and class width.

Methods of class ClstrAlgorithmConditional

Method	Arguments	Return value
CreateClasses()	Node ID	
	Attribute name	
	Minimum class	
	Maximum class	
	Class width	
	Flag: TRUE for categorial values, otherwise FALSE	
AddConditionXXX()	Node ID	
	Attribute name	
	Comparison value	
AddConditionElse()	Node ID	
GoToNode()	NodeID	

This classification algorithm is similar to the previous one (ClstrAlgorithmClassify) since it allows the specification of classification schemes. Additionally, it gives the possibility to define additional classification paths if a cells gets a specific classification. Therefore it allows to define condition´s using the family of *AddConditionXXX()* methods (conditions available are: equality, greater than, less than, inequality). Using the algorithm interface, the rule set developer creates a conditional tree structure consisting of classifications and subsequent conditional nodes. Each is assigned a unique node ID. The method GoToNode() is used to access a specific conditional node.

Methods of class Scno

Method	Arguments	Return value
QueryPart()	Time series name	Scenario value
	Key 1	
	...	
	Key n	
GetParameter()	Parameter name	Parameter value

An object of class *Scno* represents a scenario used for a SITE simulation run. To combine a rule set to be executed with a scenario is obligatory. Using method *QueryPart()*, the rule set developer can access scenario time series, where the part name is the name of database table holding the respective time series data. To get the correct value, a unique key must be specified (e.g. composed of simulation step, district ID, etc). The method returns a value of type double, which is the scenario value stored in the underlying database table for the given key. Independent rule set parameters that are part of a scenario are accessed using the *GetParameter()* method by specifying their name.

Methods of class ExternModel

Method	Arguments	Return value
SetInputVariable()	Variable name	
SetOutputVariable()	Variable name	
SetInputVrblValues()	Cell object	
InitCalculation()	Job ID	
SetRefCellId()	ID of cell for which model is invoked	
SetInfo()	Information contained in string	
SetIntermediate()	Flag defining intermediate model behavior	
Configure()		
Execute()		
HasResult()	Job ID	TRUE or FALSE
GetResult()	Parameter name	Result value
	Job ID	
CleanUp()		

The *ExternModel* class exposes the functionality to establish, configure and run calculation of simulation data using a third-party sub model. Calculations are based on single cells. The interface is identical to the external model interface introduced in section 5.5. Since th SITE sub model interface is designed to integrate point models and process modeling job parallelized if more than one server instance is available, it is necessary to first configure all job before starting the actual processing using the *Execute()* method. *Execute()* returns after all jobs have been processed. At this point, all results are stored in the *ExternModel* object and can be accessed via the *GetResult()* method by specifying the respective job IDs.

Iterators

The SITE Python language extensions also include a number of classes that can be instantiated directly inside an application rule set script. Such objects are basically helpers to access specific parts of the underlying simulation infrastructure like iterators or algorithms that could be as well implemented as part of the application script but are significantly more efficient that way. Those algorithms are referred to as update operations.

All iterator classes expose the same interface providing a method *Frst()* to set the iterator to the first element of the underlying set, a method *Next()* to step to the next (valid) element and a method *Cont()* which can be used to check whether all elements have been traversed or not. These methods can be included in the standard python *while* loop. The current element of the set being traversed can be accessed using the method *Crnt()*. However, instances of the exposed iterator classes cannot be used in Python *for* loops. In contrast to the C/C++ *for* statement, the Python counterpart requires a traversable collection of objects that fulfill the Python standard of iterators. Python has its own concept of iterators which uses a special exception thrown as soon as the traversal is complete. This concept is not compliant with iterators in C++. Therefore it is recommendable to wrap the exposed classes by iterator classes programmed in Python to match the Python requirements and thus enable the use of iterators in Python *for* loops.

Methods of class GridItr

Method	Arguments	Return value
Constructor	Grid object to traverse	
Frst()		
Next()		
Cont()		TRUE or FALSE
Crnt()		Cell object

One basic procedure in application rule sets is to traverse all cells of the underlying simulation grid and perform operations on them. The *GridItr* (grid iterator) class provides the functionality to traverse the simulation grid. To create an iterator object, the *Grid* object as representative for the simulation grid needs to be passed to the constructor.

Methods of class CellNghbItr

Method	Arguments	Return value
Constructor	Center cell object Grid object	
Frst()		
Next()		
Cont()		TRUE or FALSE
Crnt()		Cell object

The *CellNghbItr* (cell neighbor iterator) class implements functionality to iterate over all direct neighbor cells of a specified center cell. Since Moore neighborhood is assumed, the collection to be traversed consists of at eight grid cells at most depending on the location of the center cell in the simulation grid.

Methods of class LayerItr

Method	Arguments	Return value
Constructor	Layer object	
Frst()		
Next()		
Cont()		TRUE or FALSE
Crnt()		Clstr object

This iterator traverses all *Clstr* (cell aggregate) objects of a previously defined thematic layer. To create an object of this class, the Layer object housing the requested cell clusters has to be passed to the constructor.

Methods of class ClstrItr

Method	Arguments	Return value
Constructor	Clstr object	
Frst()		
Next()		
Cont()		TRUE or FALSE
Crnt()		Cell object

Cell clusters as collections of similar cells are traversed using the iterator class *ClstrItr*. The cluster object of interest is the argument passed to the constructor upon iterator object creation.

Update operations

Update operations are operations applied to all cell of the simulation grid of a simulation. Although they could as well be implemented on the application side using the Python language, it is favorable to create respective language extensions due to performance reasons. This way, high performance C++ code is invoked from Python. The SITE system provides two such operations.

Methods of class UpdtOprtnNghbCount

Method	Arguments	Return value
Constructor	Target attribute name	
	Source attribute name	
	Condition value	
SetTargetAttr()	Attribute name	
SetSourceAttr()	Source attribute name	
SetConditionVal()	Condition value	
Execute()	Grid object	

This update operation calculates the number of neighbor objects fulfilling a specific criterion for each object on the simulation grid. The criterion is specified by defining a target attribute and an appropriate attribute value. The number of neighbors that meet this criterion is stored in the source attribute. An example would to calculate the number of neighbor cells of land use class water for each cell. Configuration can either be done by passing these three arguments to the constructor or by calling the single configuration methods provided by the class interface. The actual calculation is started with the *Execute()* method.

Methods of class UpdtOprtnDistCalc

Method	Arguments	Return value
Constructor	Target attribute name	
	(continued on next page)	

Methods of class UpdtOprtnDistCalc – continued

Method	Arguments	Return value
	Reference attribute name	
	Source attribute name	
	Condition value	
SetTargetAttr()	Attribute name	
SetSourceAttr()	Source attribute name	
SetReferenceAttr()	Reference attribute name	
SetConditionVal()	Condition value	
Execute()	Grid object	

Using this update operation it is possible to calculate for each grid cell the distance to the closest grid cell fulfilling a specific criterion (e.g. the distance to the closest cell of land use class water). I addition the algorithm sets the cell ID of the closest cell as second attribute value. In the nomenclature of this update operation, the calculated distance is referred to as the target attribute while the ID of the closest cell is the reference attribute. The criterion after which it is decided to which cells the distance is calculated is defined by specifying a source attribute together with a condition value. The underlying algorithm is of complexity $O(n)$ and thus also works efficiently on large simulation grids.

University of Kassel . Center for Environmental Systems Research
Kurt-Wolters-Straße 3 . 34125 Kassel . Germany
Phone +49.561.804.3266 . Fax +49.561.804.3176
cesr@usf.uni-kassel.de . http://www.usf.uni-kassel.de