Using the object modeling system for hydrological model development and application

S. Kralisch, P. Krause, O. David

Advances in Geosciences, 4, 75–81, 2005
SRef-ID: 1680-7359/adgeo/2005-4-75
European Geosciences Union

Advances in Geosciences

Using the object modeling system for hydrological model development and application

S. Kralisch[1], **P. Krause**[1], **and O. David**[2]

[1]Friedrich-Schiller-University Jena, Institute for Geography, Jena, Germany
[2]Colorado State University, Fort Collins, CO, USA

Received: 1 August 2004 – Revised: 1 November 2004 – Accepted: 15 November 2004 – Published: 9 August 2005

Abstract. State of the art challenges in sustainable management of water resources have created demand for integrated, flexible and easy to use hydrological models which are able to simulate the quantitative and qualitative aspects of the hydrological cycle with a sufficient degree of certainty. Existing models which have been de-veloped to fit these needs are often constrained to specific scales or purposes and thus can not be easily adapted to meet different challenges. As a solution for flexible and modularised model development and application, the Object Modeling System (OMS) has been developed in a joint approach by the USDA-ARS, GPSRU (Fort Collins, CO, USA), USGS (Denver, CO, USA), and the FSU (Jena, Germany). The OMS provides a modern modelling framework which allows the implementation of single process components to be compiled and applied as custom tailored model assemblies. This paper describes basic principles of the OMS and its main components and explains in more detail how the problems during coupling of models or model components are solved inside the system. It highlights the integration of different spatial and temporal scales by their representation as spatial modelling entities embedded into time compound components. As an exam-ple the implementation of the hydrological model J2000 is discussed.

1 Introduction

With the implementation of the European Water Framework Directive (WFD – European Union, 2000) prognostic modelling for sustainable management of water resources has become even more important than it was before. The goals set up by the WFD require a stronger integrative and multidisciplinary approach than usually practiced in the last decades. In addition to quantitative and qualitative hydrological issues, socio-economic and legislative objectives must be considered to find the best solutions for maintenance or improvement of water quality in European water bodies.

Although such an interdisciplinary and holistic approach is the most promising way to reach the goals set up by the WFD, it also introduces new problems, which have to be solved in advance. The most obvious problem is that each discipline involved in the development of strategies for sustainable management of water resources uses their own methods and tools for prognostic simulation and modelling of the single processes of the water cycle throughout Europe. This is true not only from the multidisciplinary standpoint but also from a regional point of view. Scientists from one discipline in one part of Europe often use different models for the same purpose than other scientists in other parts of the continent because the constraints and environmental circumstances are different.

The most significant differences between the single models or modelling systems applied in Europe and worldwide are the specific model cores which simulate the single processes. On the other hand, all models or modelling systems have systematic functionalities (e.g. data in- and output) which are essentially common for all models even if they had been implemented in different ways.

For future proof model development and application, a modular approach which divides the systematic routines from the scientific parts is the most promising approach. Such an approach should provide the basic functionality for data in- and output, application and communication of the single components as well as an application programming interface (API) for the implementation of the scientific methods in the form of encapsulated programme modules. The most relevant benefit of such a framework for model developers would be to enable them to concentrate on only the implementation of most suitable methods and always be confronted with a familiar interface and modelling environment.

The Object Modeling System (OMS) (David et al., 2002), developed at the Friedrich-Schiller-University in Jena, Germany and at the USDA-ARS, Great Plains Systems Research Unit in Fort Collins, CO, USA is such a modelling

Correspondence to: S. Kralisch
(sven.kralisch@uni-jena.de)

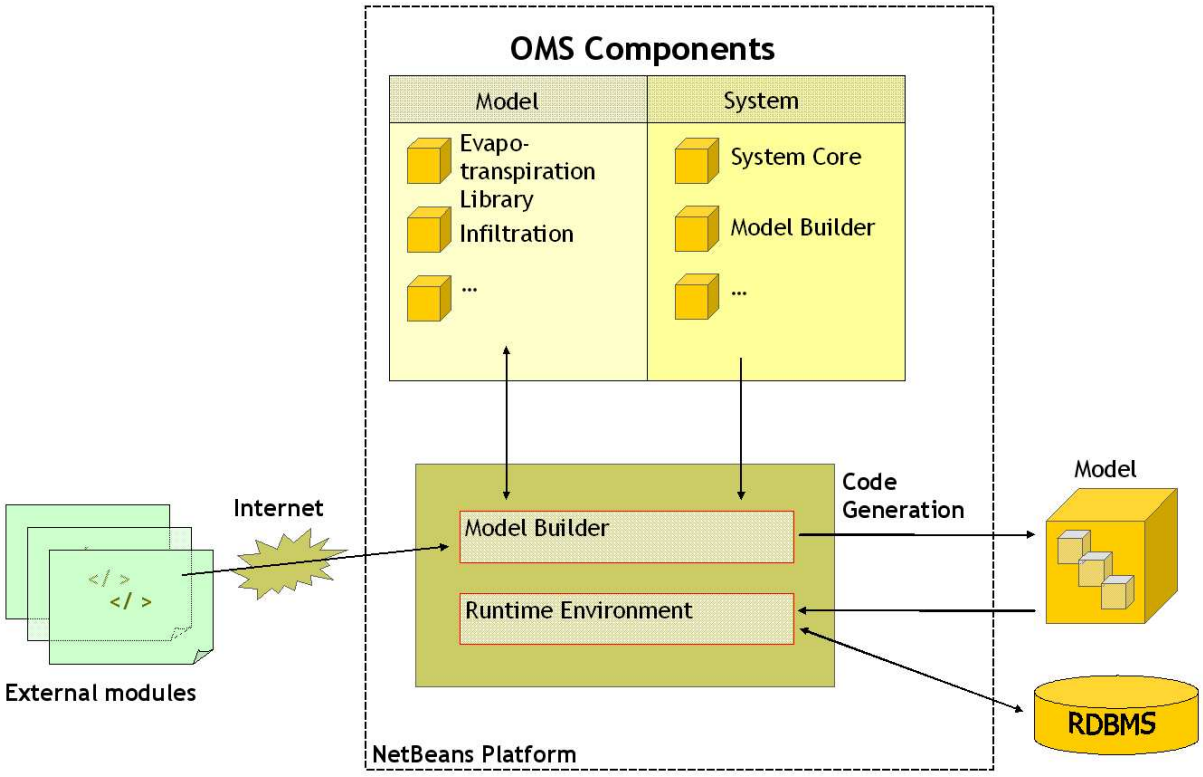

Fig. 1. Principal layout of the Object Modeling System OMS.

framework. This paper deals with its introduction, the current development and implementation of suitable programme modules.

2 The Object Modeling Systems OMS

The basic OMS concept is the representation of all system and model components as independent modules. These modules are coupled by standardised software interfaces. In order to achieve maximum platform independence, OMS was implemented in JAVA on top on the NetBeans platform (http://www.netbeans.org). NetBeans is an open source software which provides common desktop applications requirements like menus, document management, and settings for the user or developer. Extension of the OMS by new system components is guarantied through integration into the NetBeans context because of the flexible and generic interface of this platform. The principal layout of the Object Modelling System is shown in Fig. 1. The modules can be divided into system and scientific components as shown. The system components provide a Model Builder with single scientific modules that can be assembled to provide a running model. The model itself can then be executed inside the Runtime Environment. The result is a modular system whose components can be divided into system components and model components.

2.1 OMS system components

All functions which are needed for the modelling system are implemented by the system components. System components handle all coupling and execution of model components. The following main system components can be identified.

– **System core**: The system core provides basic functionality for all other components and forms the runtime environment for model as well as component development and execution. The system core implements all the data types that can be used by model components. Besides simple data objects a number of more complex objects and components are implemented. As an example, objects covering time issues (e.g., the OMSCalendar) implement specific time steps for model application and methods for time step management. Other objects, like the OMSComponents are used as basic classes for the implementation of model components. They can be understood as containers, which have to be filled by the model or module developers by the implementation of own methods or processes. The interaction and communication of such modules with others is done by standardised functions which hook the modules into their specific context. Such basic functionalities provided by these containers offers a solid basis for consistent and sound module development and allow the module

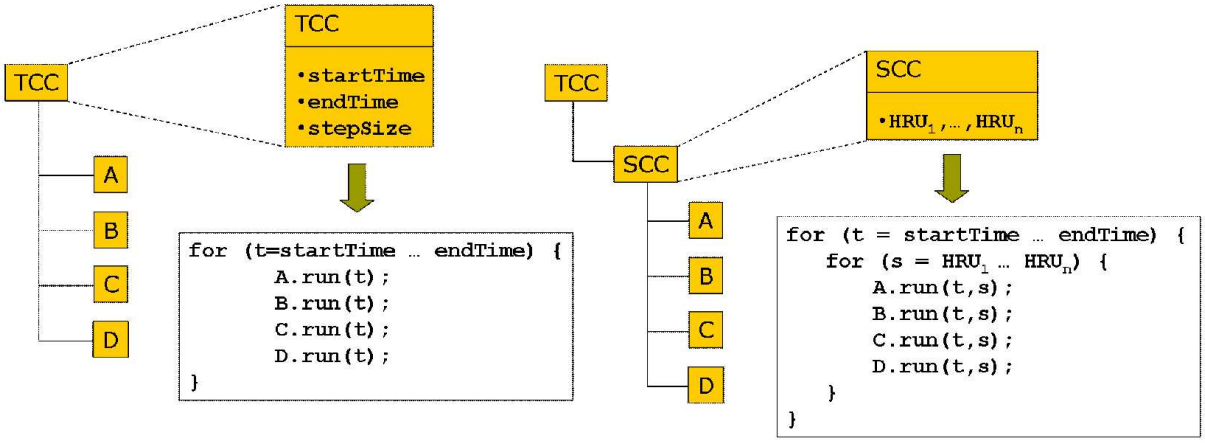

Fig. 2. Application of compound components for representation of temporal and spatial contexts.

developer to concentrate on the process implementation without thinking too much about system functions.

– **Model builder**: The model builder supports the assemblage and configuration of complex models from single model components with an easy to use graphical user interface. This interface offers capabilities for the mapping of component output parameters to input parameters of subsequent components.

  With the model builder different model configurations can be stored and managed. Once a model has been assembled and configured inside the model-builder it can be easily passed to other users or executed in other computing environments.

– **Update center**: The update center is a standard NetBeans component and provides functionality for easy update and download of existing OMS modules. All developers of OMS model components can encapsulate their work within NetBeans modules. By offering them via the Internet they can make their work available to other OMS users who can then retrieve and install them through the use of the update center. OMS-tailored NetBeans modules include not only the model components themselves, but they can also provide additional data like parameter-sets and documentation.

  All model components can be supplied with additional signature keys and licensing information to protect them against modification without permission and to secure the property rights of the developer.

– **User interface components**: The OMS offers well designed user interface (UI) components which provide a number of visualization features for developers and users. As an example, the 2d plot component offers miscellaneous 2d representations for modelling results in form of graphs or xy-plots. Additional UI components can easily be added by implementing them as NetBeans modules and integrating them into the framework.

2.2 OMS model components

The OMS system components are complemented by the OMS model components which form the building blocks for all models created within the framework. For each model component the following properties are prototyped by the OMS and have to be implemented by the developer:

– The model component implements four common methods: *register()*, *init()*, *run()* and *cleanup()*. The *register()* method comprises commands and functionality that needs to be executed once during the model initialization stage, like loading native libraries containing model functionality. The *init()* method includes code to be executed at the first invocation of a module, mostly for presetting parameters to initial values. Code contained in the *run()* method is executed at each module invocation and contains the real functionality, e.g. the Penman-Monteith equation for calculation of the potential evapotranspiration. At the end of model execution, the *cleanup()* method of each model component is executed in order to free resources used by the component.

– The read and write access of each variable is supervised by the OMS model component. To secure consistency during runtime, each variable with read access must have been written by a preceding module during the model execution. With this information, the runtime system is able to synchronise variable values between the executions of two successive model components and thus guarantee a harmonised data flow.

Whereas the model components themselves are implemented as Java classes, they may include calls to functions from native libraries using the Java Native Interface (JNI). Thus, a large amount of pre-existing process implementations can be accessed from the OMS with minor reprogramming efforts.

In addition to the model components for the implementation and execution of process representations, OMS provides

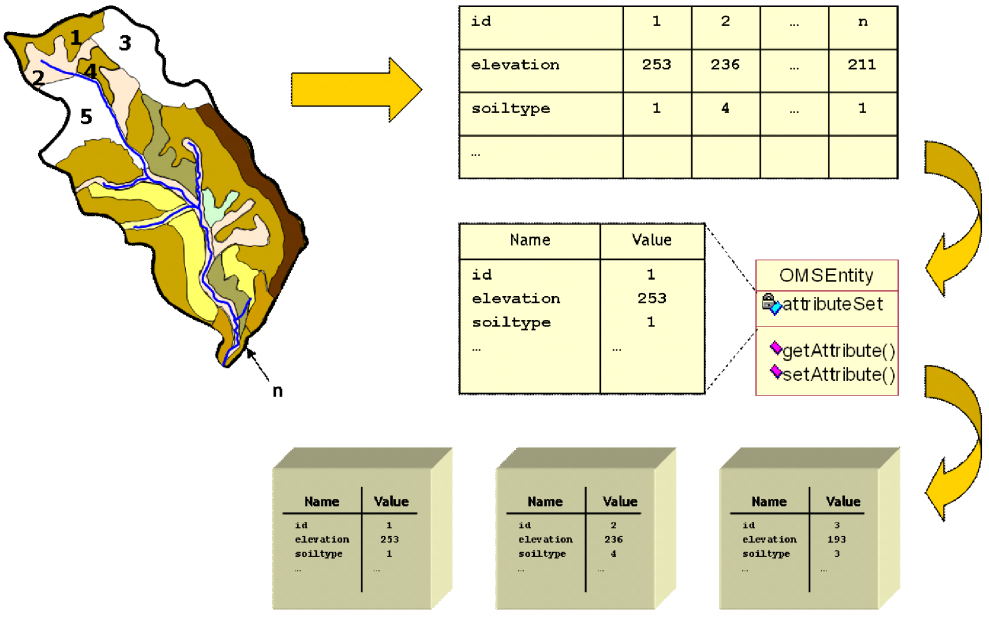

Fig. 3. Representing spatial model entities as OMSEntity objects.

specialised compound components. These work as containers for other model components and can be used to represent hierarchical structures. Each compound component provides an internal iterator which controls if and how the contained elements are enumerated. By modifying this iterator, arbitrary control structures like conditional or iterated execution of contained components can be realised. In case of a conditional operator the representing compound component simply decides wether or not the contained components are executed – dependent on some external condition checked by the component. For example, if the compound component represents a while-operator, the compound component consecutively iterates over all contained components as long as a specific external condition is valid. With these compound components, the runtime behaviour of the models can be very effectively structured and implemented.

Because of the representation of temporal or spatial contexts in many models, the iterated execution of model components is of special interest. Therefore predefined compound components for representation of temporal contexts (TimeCompoundComponent) and spatial contexts (SpatialCompoundComponent) are already implemented in OMS. TimeCompoundComponents (TCC) own a specific attribute that represents a user defined time interval and step size. With such data, TCC can create discrete points in time. On the other hand, SpatialCompoundComponents (SCC) represent discrete points in space by explicitly listing predefined spatial entities like Hydrological Response Units (HRU) or raster cells. Figure 2 shows an example of the execution of model components within a TCC on the left side. Here the process modules A to D are executed iteratively inside a time loop provided and controlled by the TCC. On the right side of Fig. 2, a SCC is integrated into the TCC. Again the modules A to D are executed iteratively but now across time and space. This example also shows how compound components can be assembled in a hierarchical manner.

2.3 Representing space

In order to represent spatial model entities, the system core uses the data type OMSEntity. OMSEntities work as an abstract container for arbitrary attributes of spatial entities. These data which can vary for each OMSEntity object are stored in tables that map attribute names to their respective values. With this approach attribute sets of spatial model entities can easily be expanded if additional data are provided by new model components or external sources. Figure 3 shows an example of how spatial model entities (polygons or raster cells) can be obtained and represented as OMSEntity objects. The figure shows a basin with elevation and soil information. Each entity has a unique ID and a set of elevation and soil-type values. Specific getter and setter functions of the OMSEntity can be used to access the OMSEntity's attribute set. This attribute set is then used during model execution to provide the process modules with the required spatially distributed information.

When a model is assembled with the model builder, the single process modules can extend the OMSEntity set by process specific state variables and attributes. For example, a soil water process module with two different soil storages extends the attribute set of each OMSEntity object by two variables representing the storages during initialisation. Additionally, specific getter and setter routines are set up to retrieve or change the current state of these storages according to the running processes during the model execution.

The idea of representing spatial entities as abstract containers for arbitrary data opens the possibility of implementing model components which are not bound to a specific spatial discretisation (e.g., polygons or raster cells). Furthermore, they can work on OMSEntity data objects without consideration or knowledge of the underlying discretisation concept. This feature, of course, cannot solve problems of process validity and compatibility on specific spatial and temporal scales, which still have to be considered by the module or model developer.

3 Application of the OMS

In order to test the described system against an existing model, a number of model components have been implemented. The basis for these components was provided by the J2000 model (Krause, 2001, 2002) which is a conceptual fully distributive hydrological modelling system. The J2000 uses the topological HRU approach for catchment discretisation and implements the single processes of the hydrological cycle (ETP, snow, soil water, groundwater, lateral routing between the HRUs and channel routing) as encapsulated process modules. Therefore the J2000 already provides a number of cleanly implemented process descriptions written in Java. These were transferred into OMS model components with only minor adjustments in the initialisation and cleanup routines. Additionally, readers for transferring the J2000 parameter files (which describe the spatial entities J2KParaReader) and for the input data files (which contain the driving variables xxxDataInput) have been integrated into the OMS. Figure 4 shows the J2000 model setup inside the OMS Model Builder.

During initialisation within the OMS/J2000 model, the J2KParaReader component reads the J2000 two spatial model entity types (Hydrological Response Units and river reach units) together with their describing parameters from external data sources (files). The objects descriptions are then translated into two lists of OMSEntitiy objects, one for the HRUs and one for the river reaches. These lists can then be accessed by two different Spatial Compound Components which are used to create iterators over the lists. The attributes of the single objects in the OMSEntitiy lists are then updated by the process modules with additional attributes and state variables together with getter and setter routines to assess their content. Both SCC are then integrated into the temporal context of one TCC which iterates in daily time steps during model execution.

During model application for each iteration step of the TCC, the two SCCs execute a number of different process modules embedded in their context in a sequential order (Fig. 4). During the execution of the single process modules, the state of the spatial objects passed to them are changed according to the process implementation inside the modules and the updated states are given back to the system. An example of the run routine of a module implementing the interception process is shown in Fig. 5. In the beginning, the

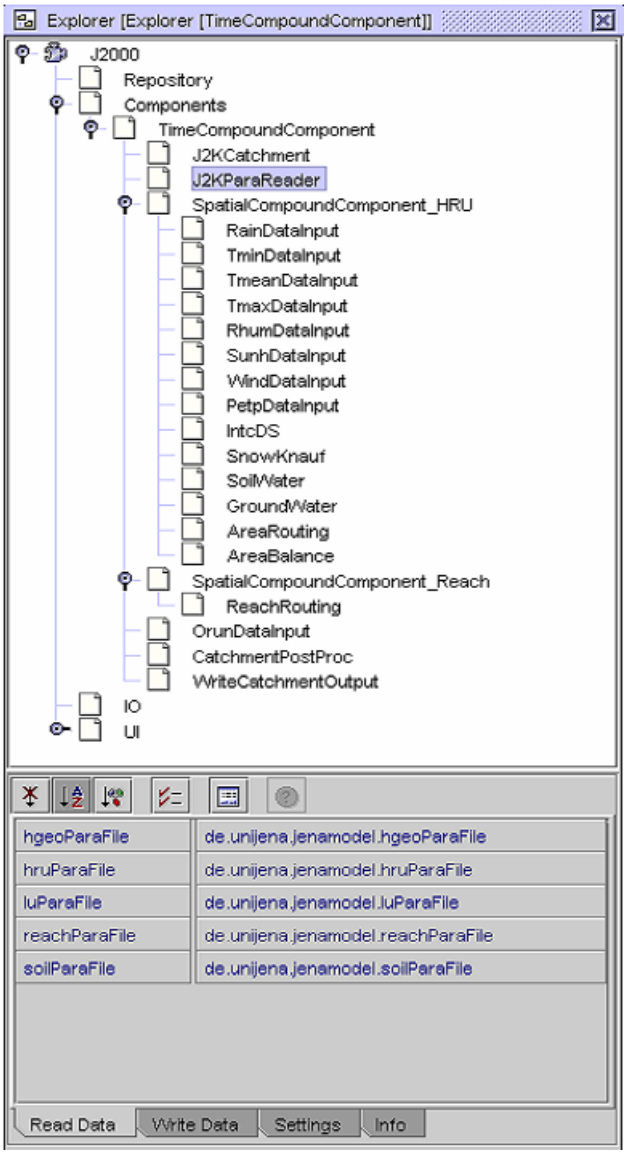

Fig. 4. Composition of J2000 components with the model builder.

relevant HRU attributes are passed to local variables. Next, the process implementation follows and is terminated at the end by the return of the updated local variables back to the HRU object. Preliminary test runs of the OMS/J2000 model show that the system produces nearly identical results as the original J2000 model. The differences are mostly determined by different treatment of the input variables and slightly differing parameter sets.

4 Conclusion and outlook

This implementation of a complete hydrological model into the OMS has shown the suitability of the system for model development and application. For the integration of the J2000 modules and their distribution, concept spatial entities have been developed and integrated into the OMS.

```
public class IntcDS extends OMSComponent {
transient OMSTimeInterval time;
transient OMSEntitySet es;

public int run() {
//pass hru variables to local variables
OMSEntity currentEntity = this.es.current;
int julday = (int) time.current.getDayInYear();
double[] LAIArray = (double[])currentEntity.getAttribute("LAI");
double LAI = LAIArray[julday-1];
double area = currentEntity.getDoubleAttribute("area");
double dailyRain = currentEn-
tity.getDoubleAttribute("dailyRain");
double dailyTmean = currentEn-
tity.getDoubleAttribute("dailyTmean");
double dailyPetp = currentEn-
tity.getDoubleAttribute("dailyPetp");
double dailyAetp = currentEn-
tity.getDoubleAttribute("dailyAetp");
double deltaETP = dailyPetp - dailyAetp;
double actIntcStorage = currentEn-
tity.getDoubleAttribute("actIntcStorage");
double throughfall = 0;
double dailyIntc = 0;

//calculate interception parameters
double alpha = 0;
if(dailyTmean < -2.0)
    alpha = 0.5;
else
    alpha = 0.2;

double maxIntcCap = (LAI * alpha) * area;

if(actIntcStorage > maxIntcCap){
    throughfall = actIntcStorage - maxIntcCap;
    actIntcStorage = maxIntcCap;
}
double deltaIntc = maxIntcCap - actIntcStorage;
if(deltaIntc > 0){
    double saveRain = dailyRain;
    if(dailyRain > deltaIntc){
      actIntcStorage = maxIntcCap;
      dailyRain = dailyRain - deltaIntc;
      deltaIntc = 0;
      dailyIntc = (saveRain - dailyRain);
    } else{
      actIntcStorage = (actIntcStorage + dai-
lyRain);
      dailyIntc = dailyRain;
      dailyRain = 0;
    }
} if(deltaETP > 0){
    if(actIntcStorage > deltaETP){
      actIntcStorage = actIntcStorage -
deltaETP;
      dailyAetp = dailyAetp + deltaETP;
      deltaETP = 0;
    } else{
      deltaETP = deltaETP - actIntcStorage;
      dailyAetp = dailyAetp + (dailyPetp -
deltaETP);
      actIntcStorage = 0;
    }
}
//return hru variables from local variables
currentEntity.setDoubleAttribute("throughfall",
throughfall);
currentEntity.setDoubleAttribute("netPrecip",
throughfall + this.dailyRain);
currentEntity.setDoubleAttribute("dailyAetp",
dailyAetp);
currentEntity.setDoubleAttribute("actIntcStorage",
actIntcStorage);
currentEntity.setDoubleAttribute("dailyIntc",
dailyIntc);

return 0; }
```

Fig. 5. Example of the run routine of a OMS/J2000 process module.

Great care was taken to implement the OMSEntities as flexible and open as possible to guarantee their reusability for other process modules from other models. During the refactoring of the J2000 modules for the use inside OMS, only minor parts had to be adapted to ensure proper module installation and initialisation. The original process implementations were left untouched. Test runs of the OMS/J2000 implementation showed nearly identical results when compared to the original model results.

The comparison of the OMS/J2000 against the original implementation also showed that the OMS performance is much slower than the original implementation. This lack of performance is related to the flexibility gained by the use of dynamic attribute sets for model entities inside the OMS. To keep them updated and consistent during model execution, a lot of time consuming computing, like type casting, has to be performed. The optimisation of model performance and elimination of time consuming operations during model execution therefore forms the subject of ongoing research. A possible solution could be the utilization of alternative data structures for storing entity attributes. For example. the GNU Trove library (http://trove4j.sourceforge.net/) provides a fast, lightweight implementation of the Java Collections API to realize this.

Edited by: P. Krause, S. Kralisch, and W. Flügel
Reviewed by: anonymous referees

References

David, O., Markstrom, S. L., Rojas, K. W., Ahuja, L. R., and Schneider, I. W.: The Object Modeling System, in: Agricultural System Models in Field Research and Technology Transfer, edited by: Ahuja, L., Ma, L., Howell, T. A., Lewis Publishers, CRC Press LLC, 317–331, 2002.

European Union (2000): Directive 2000/60/EC of the European Parliament and of the Council of 23 October 2000 establishing a framework for Community action in the field of water policy; Official Journal L 327, p. 0001–0073, 2000.

Krause, P.: Das hydrologische Modellsystem J2000 – Beschreibung und Anwendung in großen Flußgebieten (The hydrological modelling system J2000 – Documentation and application in large river basins); Schriften des Forschungszentrums Jülich, Reihe Umwelt/Environment, Band 29, 2001.

Krause, P.: Quantifying the impact of land use changes on the water balance of large catchments using the J2000 model; Physics and Chemistry of the Earth, 27, 663–673, 2002.