Environmental Modelling & Software 68 (2015) 110–121

Contents lists available at ScienceDirect

Environmental Modelling & Software

journal homepage: www.elsevier.com/locate/envsoft

A lightweight framework for rapid development of object-based hydrological model engines

David Kneis [1]

University of Potsdam, Institute of Earth and Environmental Sciences, Germany

ARTICLE INFO

Article history:
Received 13 October 2013
Received in revised form
9 February 2015
Accepted 12 February 2015
Available online

Keywords:
Modeling framework
Generic model
Hydrology
ECHSE

ABSTRACT

Computer-based simulation models are frequently used in hydrological research and engineering but also in other fields of environmental sciences. New case studies often require existing model concepts to be adapted. Extensions may be necessary due to the peculiarities of the studied natural system or subtleties of anthropogenic control. In other cases, simplifications must be made in response to scarce data, incomplete knowledge, or restrictions set by the spatio-temporal scale of application. This paper introduces an open-source modeling framework called ECHSE designed to cope with the above-mentioned challenges. It provides a lightweight infrastructure for the rapid development of new, re-usable simulation tools and, more importantly, the safe modification of existing formulations. ECHSE-based models treat the simulated system as a collection of interacting objects. Although feedbacks are generally supported, the majority of the objects' interactions is expected to be of the feed-forward type. Therefore, the ECHSE software is particularly useful in the context of hydrological catchment modeling. Conversely, it is unsuitable, e. g., for fully hydrodynamic simulations and groundwater flow modeling. The focus of the paper is put on a comprehensible outline of the ECHSE's fundamental concepts and limitations. For the purpose of illustration, a specific, ECHSE-based solution for hydrological catchment modeling is presented which has undergone testing in a number of river basins.

Availability of the software

The ECHSE software is freely available to interested users and developers under the terms of the MIT license. The website of the project is http://echse.bitbucket.org. From this site, one can download the recent documentation files covering (1) the basic design and use of the modeling framework, (2) the installation of model engines, (3) the existing classes for rainfall–runoff modeling, as well as (4) theory and use of pre- and post-processing utilities (Kneis, 2012a,b,c,d).

The mentioned website also provides a link to the ECHSE's code repository. By cloning the repository, one gets access to

1. the latest generic source code (see Section 3.2.4),
2. the declaration and code of a set of classes for hydrological catchment modeling (see Section 4.2),
3. model engine definition files specifying the classes used by a particular engine,
4. scripts for code generation and compilation,
5. the code generator and other utilities (as self-documenting R-packages or C++ sources),
6. the documentation files.

The primary way to contribute to the project is to improve the existing classes or to design new classes and model engines being of potential interest to the community. As opposed to the application-specific classes, it is not planned to permanently maintain different forks (i. e. concurrent versions) of the software's generic core.

1. Introduction

1.1. Terminology

The term *model* is used in many different ways and contexts. To minimize the chance of confusion, the following convention is used throughout this paper:

E-mail address: david.kneis@tu-dresden.de.
[1] Present address: Technical University Dresden, Institute of Hydrobiology, Germany.

http://dx.doi.org/10.1016/j.envsoft.2015.02.009
1364-8152/

A hydrological *model* is a computer-based tool to simulate the dynamics of a *particular* hydrological system. Example: A rainfall–runoff model of the Amazon River Basin.

A model usually comprises of two well-separated parts: (1) a generic simulation software and (2) the data describing the particular system (parameters, forcings, etc.). The first part, i. e. the plain software, is called the *model engine*. A rainfall–runoff *model engine*, for example, is applicable to many different river basins.

Finally, a *modeling framework* is a software aimed at facilitating the implementation of *model engines*. The terms *generic model (engine)* or *simulation environment* are sometimes used synonymously with *modeling framework*. The concept of modeling frameworks is introduced in the next section.

1.2. Software concepts for process-oriented modeling

In the context of water resources management, computer models proved to be quite useful tools (see, e. g. Liebscher and Mendel, 2010; Beven, 2012). This applies in particular to process-oriented, conceptional models. Such models represent a consistent collection of ideas about the functioning of real systems and they help to reveal gaps in current knowledge. Practical uses range, for example, from the assessment of management options to the generation and verification of operational forecasts.

Given the diversity of scientific and practical questions in the fields of hydrology and aquatic ecology, the water research community agrees upon the fact that a single 'all-purpose model engine' does not (and cannot) exist (Teutsch and Krüger, 2011). This is because the appropriate complexity (variables and interactions) as well as the spatio-temporal resolution of a model depends on many factors, including (1) the research question, (2) the system's characteristics, (3) the spatial and temporal scale of application, as well as (4) the available data and computational resources. Consequently, a very large number of individual model engines has been developed over the last decades (e. g. Singh and Frevert, 2002a,b; Liebscher and Mendel, 2010). They are usually specialized on the representation of selected processes or compartments.

A rather new focus of research has been set on the development of more flexible model engines. Ideally, they can be adapted to the problem of interest, for example, by considering additional processes/compartments and/or by dropping unnecessary ones. Likewise, the mathematical description of the hydrological processes, can be adapted mindful of the present knowledge, existing data, and available computer resources. The remainder of this section gives a short overview on major lines of development.

1.2.1. Multi-compartment model engines

The idea is to simulate all compartments (typically soil, subsurface, surface waters) and the related hydrological processes within a single, fully integrated software. Well known examples are HydroGeoSphere (Therrien et al., 2010; Sudicky et al., 2008), the Penn State Integrated Hydrological Model (PIHM; Qu and Duffy, 2007; Bhatt et al., 2014), Système Hydrologique Européen (SHE; Refsgaard et al., 2010) as well as the Soil and Water Assessment Tool (SWAT; Arnold et al., 1994; Arnold and Fohrer, 2005) and its derivatives. The integrated simulation of all major hydrological processes makes these tools applicable to a broad range of complex problems. A drawback, however, is the complexity of the software and the high demand for data and computation time in particular. This may effectively hamper the application in many real-world situations.

1.2.2. Component-based model engines

This concept uses largely independent model engines (components) for the simulation of individual compartments and/or processes. To allow for the necessary exchange of data between the components at run time, a communication standard is introduced. This strategy is mainly represented by the Open Modeling Interface, OpenMI (Gregersen et al., 2007) and the Framework for Risk Analysis in Multimedia Environmental Systems, FRAMES (Whelan et al., 2014) but see Dozier et al. (2014) for alternatives. The subdivision of the software into well-separated components is an evident advantage in terms of maintenance and collaborative development. The approach is, however, primarily designed for the coupling of existing model engines. It does not facilitate the development of new engines and requires profound knowledge of low-level details of both the communication standard (Castronova and Goodall, 2010) and the coupled models.

1.2.3. Modeling frameworks

A modeling framework is a software aimed at facilitating the development of different model engines. Thus, the framework is applicable to a *range of problems*. The user of the framework must define (or select) the modeled compartments including the respective state variables, forcings, and parameters. He must also specify equations to describe all processes acting on the state variables. As with component-based simulation software, framework-based model engines are well-structured. They are easily maintained, modified or extended. The framework-based approach fits well with the typical work flow where a preliminary model engine is developed rapidly with the intention of stepwise improvement. Last but not least, modeling frameworks are perfect for studying uncertainties related to the model (engine's) structure. References to a number of existing modeling frameworks can be found in Section 2.

None of the three approaches outlined above is superior in all thinkable situations. Consequently, it seems desirable to further invest in all three lines of development and to accept its co-existence.

1.3. Focus of this paper

This paper puts the focus on the third approach addressed in Section 1.2. A short survey on existing modeling frameworks is presented in Section 2 before a newly developed framework called ECHSE[2] is introduced in Section 3. It's most prominent target application is hydrological catchment modeling. Nevertheless, bearing some inherent limitations in mind (Section 3.2), the framework can be used productively in other contexts too. For example, the ECHSE could aid in the implementation of object-based water quality models for river-lake systems (see, e. g. Kneis, 2007). Likewise, the model engine used by Förster et al. (2005) to simulate flood storage in a system of interconnected reservoirs could have been implemented very easily using the ECHSE framework.

The specific objectives of software development are outlined in Section 3.1. Fundamental concepts and limitations of the framework are presented in 3.2. The subsequent sections 3.3 and 3.4 address the practical steps of model engine development and set-up. An example of use is given towards the end of the paper (Section 4). Conclusions are drawn in Section 5 and information on software availability is contained in Availability of the software.

2. Existing water-related modeling frameworks

Due to compelling benefits of generic simulation tools in terms of flexibility and re-usability, a vast number of software solutions

[2] Abbreviation for Eco-Hydrological Simulation Environment.

emerged in recent decades. This includes modeling languages designed to serve various fields of science and technology (e. g. Fritzson, 2004). In the subsequent paragraphs, the focus is on tools having a clear scope of application in the dynamic simulation of environmental processes and hydrological systems in particular.

The simulation of water fluxes at the *catchment scale* (possibly including fluxes of matter) constitutes an important field of application for modeling frameworks. River basins are spatially large, complex systems subject to considerable spatial heterogeneity (e. g. with respect to infiltration rates and mechanisms). A strictly physically-based description is practically impossible and a unified, transferable theory for mathematical modeling is, therefore, not available. As a consequence, many alternative, conceptual formulations have been developed for phenomena such as, for example, runoff generation or evapotranspiration. Modeling frameworks not only facilitate the selection of a single 'best' formulation for a particular case but they also provide an efficient means for running multi-model ensembles.

Software solutions that were designed (or can be used) to model catchment-scale eco-hydrological processes include, for example, the Interactive Components Modeling System (Reed et al., 1999) and Tarsier (Watson and Rahman, 2004) both compared in Rahman et al. (2004), The Invisible Modelling Environment (TIME, Rahman et al., 2003), the E2 framework (Perraud et al., 2005; Argent et al., 2009) and its successor WaterCAST (Argent et al., 2008) as well as LIQUID (Branger et al., 2010) and Source IMS (Welsh et al., 2013).

The Object Modeling System (Kralisch et al., 2005; David et al., 2010, 2013) exploited by, e. g., Formetta et al. (2014) provides another typical example of a modeling framework. Like many other frameworks (and particular model engines), it is based on the paradigms of object-orientated design outlined in Zeigler (1990). In such a design, the fundamental building blocks of a model engine are *application-specific classes*. Typically, a class represents a specific *type of* compartment including all associated variables and methods for data manipulation. Ideally, suitable class code can be picked from an existing library. In other cases, the model engine developer must design classes with the required functionality himself.

In comparison to the aforementioned software, the rainfall–runoff modeling toolkit (Wagener et al., 2001), the frameworks presented by Wang et al. (2005) or Gattke and Pahlow (2006), as well as FUSE (Clark et al., 2008) and SUPERFLEX (Fenicia et al., 2011) have a narrower focus on conceptual hydrological modeling. They are of primary interest to scientific users in the context of hydrological model structure identification, hypothesis testing, and uncertainty exploration.

The use of modeling frameworks is most beneficial in situations where a model engine needs to be either extended or simplified for a particular application. This is a typical case in river and lake water quality modeling where the set of constituents (e. g. organic load, nutrients, toxic chemicals, organisms) and relevant turnover processes are highly case- and site-specific. Dedicated modeling frameworks are presented, for instance, in Reichert (1994) or DHI (2013). Regnier et al. (2002) describe another generic approach to reactive transport simulation and another Framework for Aquatic Biogeochemical Models, FABM, was recently published (Bruggeman and Bolding, 2014). Also, Mooij et al. (2014) present a strategy to separate ecological knowledge (expressed in a set of differential equations) from the actual model implementation.

To sum up, one can say that a considerable pool of generic, eco-hydrological simulation software is available. Since the scientific background of the developers and the intended user groups are heterogeneous, the various tools differ in fundamental aspects, some of which are pointed out below:

- Generic character: Is the developer free to implement new state variables and processes or is he restricted to select from predefined set(s)?
- Language: What computer language, if any, does the developer need to know? Is this a custom language or a general-purpose one?
- Code generation: In case of extensible frameworks, what is the balance between manual coding and automatic code generation?
- Code segregation: What part of the generic framework is (or needs to be) exposed to the developer of a specific model engine?
- Code execution: Is the engine-specific code parsed at run-time (interpreter approach) or is it compiled and linked with the engine's generic kernel?
- Numerics: What methods to integrate differential equations does the framework permit or force (see, e. g. Clark and Kavetski, 2010)?
- Spatial discretization: Does the framework support spatially distributed modeling, e. g. by providing dedicated classes/types?
- Platform dependence: Does the framework run on different operating systems?
- Availability: What policy/license applies to the generic part of the software?
- Peripheral functions/tools: How does the framework support tasks like visualization or parameter estimation?

From the author's point of view, the vision of a single, unifying eco-hydrological modeling framework is similarly unrealistic as the vision of an all-purpose model engine. Given the range of different application, the co-existence of alternative modeling frameworks, each with specific pros and cons, is not surprising (see, e. g. Argent et al., 2006). The diversity of approaches may also be regarded as a driver of continuous evolution.

3. The ECHSE modeling framework

3.1. Background and objectives

The ECHSE modeling framework introduced in the remainder of this paper was born out of experience with the development and/or use of model engines for different purposes. This includes water quality simulation (Kneis et al., 2006, 2009), flood retention modeling (Förster et al., 2005), and runoff forecasting (Kneis et al., 2012). At a certain level of abstraction, the specific model engines used in the cited studies were actually identical. In particular, all engines were designed to represent a spatially distributed system by a collection of interacting objects. Likewise, all engines aimed at simulating the system's dynamics by numerical integration of ordinary differential equations. Retrospectively, it appears that the above-mentioned studies could have taken significant profit from the use of a single, dedicated modeling framework.

Development of the ECHSE framework formally started in 2010 based on an earlier attempt to create an extensible water quality model for shallow freshwater systems (Kneis, 2007). The development was guided by the following primary objectives:

(1) Creation and modification of model engines should be as simple and transparent as possible. This was considered a necessity for successful applications not only by experienced developers but also by master and PhD students from non-IT disciplines. A particular goal was to use well-established principles of object-orientated design internally while limiting the required skill at the user's side to standard

procedural programming. Automatic code generation was considered a suitable technology to achieve this goal.

(2) The framework should be lightweight in the sense that the functionality of the created model engines is restricted to actual simulation. The decision was guided by the fact that, for many peripheral tasks such as visualization or parameter estimation, sophisticated, highly flexible, platform-independent tools are freely available, e. g. through the R project (R Development Core Team, 2009). Hence, a focus was put on the creation of a generic, non-interactive user interface to allow for convenient integration of ECHSE-based model engines with pre- and post-processing tools.

(3) The fitness of ECHSE-based model engines for operational applications (e. g. in the realm of flood forecasting) or computationally demanding uses (e. g. in the context of optimization) was of particular concern. Consequently, special attention was paid to aspects governing computational efficiency. For example, the framework should offer capabilities for parallel processing and the user should have control over ODE integration strategies to achieve a suitable tradeoff between speed and accuracy. Finally, consistent exception handling, including the generation of easy-to-interpret stack trace information, was regarded as an indispensible feature for serious applications.

(4) With the premise of easy, non-restrictive use of the framework in, e. g., science and education, the software, including documentation and sources should be freely available. Applicability on different operating systems was considered as an important aspect too.

In addressing the objectives outlined above, the ECHSE modeling framework comes with a unique combination of features (Table 1) which is, according to the author's knowledge, not found in existing simulation environments. Nevertheless, it is emphasized that the ECHSE does not claim at being generally superior to existing solutions. Like any modeling framework, it has specific strengths and limitations being revealed in subsequent sections.

3.2. Implementation

3.2.1. Object-oriented approach

Model engines built with the ECHSE framework largely comply with the object-oriented approach to software design (see, e. g. Meyer, 1997). The fundamental building blocks of the software are *application specific classes*. Using the language of traditional (procedural) programming, such a class may be considered as a *compound type* supplemented by a set of dedicated *functions* to manipulate instances (i. e. *variables*) of that particular type. The instances of a class are called *objects*.

The properties of ECHSE-based model engines can be summarized as follows:

- The compartments and/or spatial sub-units of a simulated system are treated as individual *objects*. Objects of different classes typically co-exist. All objects of a particular class are collectively managed as an *object group* (Fig. 1).
- All the application-specific classes are derived from an *abstract base class* following the rules of inheritance. The base class provides functionality which is common to all objects. It is also the key to seamless iteration over all objects (of any class).
- The state of an object is described by one or more *state variable(s)*. The dynamics of the state are typically controlled by internal and/or external forcings. Internal forcings represent *interactions* between objects. An interaction is equivalent to the exchange of information (in the model) and the transfer of mass, energy, or information (in the real system).
- The purpose of the software is to compute the evolution of the states of all objects for a sequence of discrete time steps. Typical outputs are the values of state variables and/or fluxes rates.

3.2.2. Interactions (internal forcings)

As stated above, the term *internal forcing* refers to a situation where a particular object receives information – potentially affecting its state – from another simulated object. In the remainder of the text, the equivalent but more common term *interaction* is preferably used to denote such inter-object exchange of information.

In object-based models, two different types of object interactions can exist: *feed-forward* interactions and *feedbacks*. In hydrological models, a feed-forward interaction is characterized by a well-defined upstream–downstream relation between the involved objects. The upstream object only forwards information to the downstream object but there is no reverse link (Fig. 2, buckets A and B). For a real-world example, consider a river (1st object) flowing into a lake (2nd object). As long as the slope of the river bed

Table 1
The ECHSE framework: major aims and concepts.

Objectives	Concepts
• Rapid and transparent development of model engines from scratch or by re-implementation of existing codes • Modification of existing model engines without the danger of non-obvious side effects	• Strict separation between generic, re-usable code and application-specific code • Automatic generation of code sections • Simple and safe-to-use interface between generic and application-specific code with clear, restrictive rules for data access
• Attractiveness for developers with limited programming skills	• No need for object-oriented programming • Low-level details of the framework are hidden
• High computational efficiency	• Use of compiled C++ code • Built-in parallel processing • Support for analytical and numerical solutions
• Reliable error diagnostics	• Consistent error handling using exceptions • Extensive checks of input data • Generation of complete and precise traceback information
• Simple, generic user interface of model engines	• Keyword-driven input of configuration data via command line and/or files • Data I/O through standard file formats (tabular text by default; support for HDF5 is planned)
• Lightweight and free software	• Clear source code restricted to core functionality, i. e. dynamic simulation • Publication of the source code under a non-restrictive license • Framework, model engines, and any required free software tools run on different platform

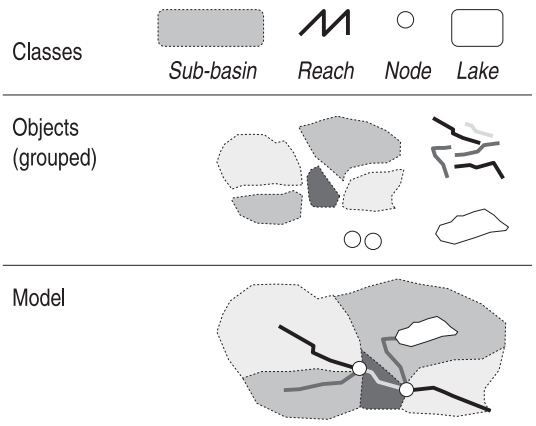

Fig. 1. Relation between classes, objects, and object groups with the example of a hydrological catchment model.

is sufficiently large, the flow rate at the object's interface is practically insensitive to the lake's water level or storage.

In contrast to that, information needs to be exchanged in both directions where feedback interactions occur (Fig. 2, buckets B and C). As a realistic example, consider a lake water quality model, where water body and sediment are treated as separate objects. The flux of dissolved matter across the interface would be controlled by both the concentrations in the water body and the sediment's pore water, i. e. the states of the two objects.

Numerical issues related to interactions are discussed below (Section 3.2.3). The technical implementation of internal forcing variables is addressed in Section 3.2.5.

3.2.3. Numerical aspects and limitations

The distinction between the two types of interactions discussed in Section 3.2.2 is important in terms of numerical treatment. In general, the dynamics of an *individual* object's state variables are determined by a system of ordinary differential equations (ODE; Eqn. (1)).

$$\dot{y} = Q^T \cdot r \tag{1}$$

In this notation (e. g. Reichert et al., 2001), y is the vector of the state variables and $\dot{y}$ represents their derivatives with respect to time. Furthermore, r is a vector of rate expressions reflecting all considered processes and Q^T is the transpose of the so-called stoichiometry matrix.

Potentially, the ODE systems of all simulated objects are coupled. Thus, looking at a particular object, Q and/or r may contain references to the state variables y of *any* object(s) considered in the model. For example, in a model with a feedback interaction between two objects A and B, the circular references $\dot{y}_A = f(y_B)$ and $\dot{y}_B = f(y_A)$ exist. Hence, the ODE systems of the two objects form a joint system requiring *simultaneous* integration.

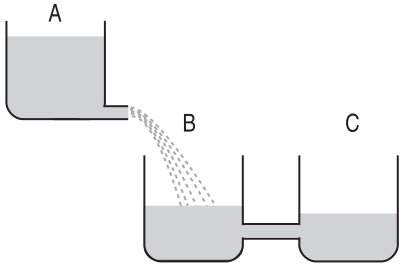

Fig. 2. Three leaking bucket objects whose storage levels are linked through a feed-forward interaction (A & B) and a feedback interaction (B & C).

In a model with feed-forward interactions only, the situation is much simpler. If $\dot{y}_A = f(y_B)$, it is guaranteed that $\dot{y}_B \neq f(y_A)$. Due to that, the ODE systems of all the individual objects can be integrated *sequentially*. Thus, in each time step, the objects can be simulated one after another as long as the processing order reflects the upstream–downstream relations (upstream first).

As a simplification, ECHSE-based model engines stick to the principle of sequential processing of the individual objects in every time step disregarding the present types of interactions. Therefore, the model engines are primarily applicable to systems dominated by feed-forward interactions. Nevertheless, feedbacks can be simulated but the ODE systems of the involved objects are still treated as *temporarily independent*. That is, information is exchanged between the interacting objects only at the begin/end of each model time step. The corresponding numerical solution is, therefore, of Eulerian type and subject to first-order accuracy. Another consequence of sequential processing is the need for artificial *observer objects* to properly simulate a feedback between two objects A and B (Fig. 3). The observer's role is to supply information on the current state of A to B and vice versa. If A and B would query each others state directly, it was impossible for the two objects to gather synchronous information.

From the above discussion it should be clear that ECHSE-based model engines should not be applied to feedback-dominated simulation problems. In particular, the objects of an ECHSE-based model are not a suitable substitute for the spatial sub-units used by codes that solve partial differential equations. For those types of problems, including groundwater flow and open channel flow with significant backwater effects, other software designs are more appropriate in terms of accuracy and computational efficiency.

In ECHSE-based model engines, the numerical method used to solve the ODE system of a particular object is a feature of the underlying class. Hence, it is up to the developer of the class to choose a suitable algorithm. If no analytical solution is available, a numerical scheme must be selected with an eye on accuracy and efficiency. A first order scheme is always easily implemented. For more stable and accurate solutions, the ECHSE framework provides a built-in Runge–Kutta solver with automatic time step control adopted from Press et al. (2002). It is planned, however, to replace the current solver by the LSODA routine (Hindmarsh, 1983; Petzold, 1983) which automatically detects and handles stiff ODE systems.

Generally, special attention must be paid to the exchange of information between objects. This is true even for simple interactions where objects are coupled in a feed-forward manner. Consider, for example, the two buckets A and B from Fig. 2. One the

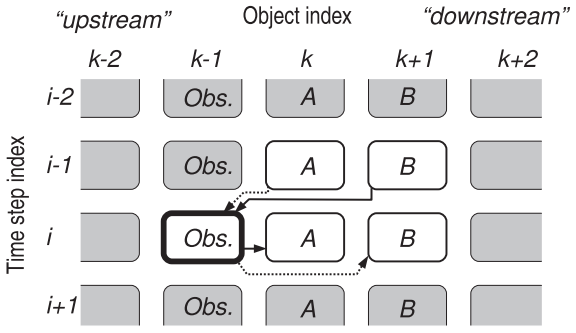

Fig. 3. Practical handling of a feedback between two objects A and B: First, an auxiliary observer object (Obs.) queries the current state of A and B. Then, the two objects obtain synchronous information on each others state from the observer. This is repeated in every time step.

one hand, it seems adequate to pass information from A to B that guarantees conservation of mass, e. g. a cumulated flux rate or an equivalent time-step average. On the other hand, however, it may be desirable to pass information on flow variations at a sub-time step level to properly capture the dynamics of B's input. The ECHSE framework does not force a particular standard regarding what information is passed from A to B and how it is used by B. Therefore, it is up to the model engine developer to choose a proper strategy, taking into account accuracy, computational burdens, and the envisaged length of simulation time steps. In simple cases, passing just time step-averaged information may be sufficient. In other cases, the parameters of interpolation functions, holding information on temporal variations, may be more adequate exchange items. In any case, the developer is responsible for declaring the necessary output variables in the class of the sender object (A) as well as associated internal forcing variables in the class of the receiver object (B). See Section 3.2.5 for details on a class' data members (i. e. variables) and Section 3.3.2 for a practical example.

3.2.4. Categories of source code

As mentioned in Table 1, an ECHSE-based model engine combines

1. generic source code which is re-used in any model engine, and
2. specific code which distinguishes this particular engine from others.

The *generic code* is responsible for basic operations like memory allocation, reading, conversion, and checking of all inputs as well as the generation of outputs. It also manages the iteration over the time steps of the simulation period (outer loop) as well as the iteration over the simulated objects (inner loop), possibly using multiple threads. The generic code is 100% re-usable.

The *specific code* is equivalent to the definition of the application-specific classes which can be used to instantiate the modeled objects. This part of the code must be provided by the model engine developer. The ECHSE framework relies on the technique of *code generation* to automatically create essential parts of the specific code. In accordance with the major objectives (Table 1, 1st row), this reduces the effort for manual coding to a minimum. At the same time, the generated code is guaranteed to be compatible with the generic parts. Last but not least, the use of generated code eliminates the risk of some capital programming mistakes as it sets narrow restrictions on data access.

Together with the generic source code, the *code generator* forms the main infrastructural service provided by the ECHSE modeling framework. The code generator is a stand-alone program currently implemented as an add-on package for the R software (R Development Core Team, 2009). The interplay of (1) generic, (2) generated, and (3) hand-written code is illustrated in Fig. 4. Automatic code generation and the nature of the hand-written parts are further discussed in Section 3.3 in the context of a basic example.

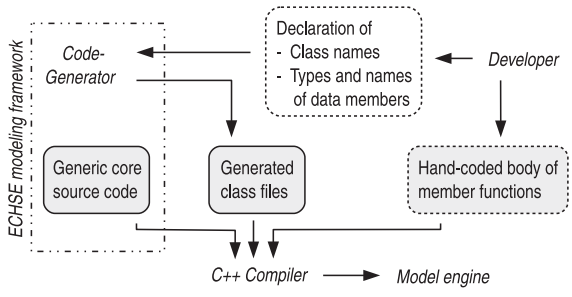

Fig. 4. Origin of the three categories of source code (shaded boxes) that form an ECHSE-based model engine. Information to be contributed by the developer is shown with dashed borders. The two items in the dash-dot box on the left constitute the actual ECHSE modeling framework.

3.2.5. Members of application-specific classes

This section introduces the variables (*data members*) and methods (*member functions*) of an arbitrary application-specific class. Currently, nine types of data members are to be distinguished (Table 2). All of these variables actually represent arrays of variable length. In this way, a class can have any number of state variables, forcings, parameters, etc. (even zero).

Usually, the *state* of an object is captured by a set of scalar state variables. Their values are taken as representative for the entire object. In a fictive reservoir class, the storage volume or the depth-averaged temperature could be declared as scalar state variables, for example. Vector-valued state variables provide a means to take the spatial variability *within the boundaries of an object* into account. A vertical profile of temperature is a typical example. If the length of the vector(s) is not constant (i. e. the spatial discretization is dynamic), the use of vector-valued state variables is in fact indispensable. In many other cases, a sub-division of the original class is an alternative. With respect to a stratified reservoir, for instance, one could use a 'layer' class with only scalar state variables.

Changes in the values of state variables are often triggered by *forcings*. An *external* forcing is just a variable whose dynamics is predefined. The respective time series data are read from files as outlined in Section 3.2.6. *Internal* forcings are equivalent to object interactions (see Sections 3.2.1 & 3.2.2). The internal forcing variables associated with a particular object are simply references to the output variables of other simulated objects. Technically, the internal forcing variables are implemented as a vector of pointers.

Parameters appear in the differential equations that govern the state variables' dynamics. In ECHSE-based model engines, parameters may be declared as either object-specific or class-specific. In the first case, individual values may be assigned to the objects of a class. In the second case, the values are associated with object groups rather than with the individual objects (recall Fig. 1). Thus, the values are shared among all objects of a particular class. Table 2 also distinguishes between numerical parameters (plain scalar constants) and parameter functions (lookup tables). The latter allow for the representation of single-argument functional relationships that do not fit well into analytical structures (rating curves, storage characteristics of reservoirs, etc.).

Output variables are used to 'publish' numerical data about an object. Only those 'visible' data can be accessed by *other* objects in the context of internal forcings. Furthermore, the export of time series can only be requested for declared output variables.

Apart from the data members discussed above, an application-specific class provides a number of methods (Table 3).

The `simulate` method is responsible for the updating of the state and output variables at the end of each modeling time step.

Table 2
Types of data members of an application-specific class.

Type	Sub-type	Short name
State variable	Scalar	stateScal
	Vector-valued	stateVect
Forcing	External	inputExt
	Internal	inputSim
Scalar parameter	Object-specific	paramNum
	Class-specific	sharedParamNum
Parameter function	Object-specific	paramFun
	Class-specific	sharedParamFun
Output variables	–	output

Table 3
Member functions of an application-specific class.

Method	Purpose
`simulate`	Updates state and output variables
`derivs`	Computes derivatives of state variables
get-methods	Read-access to data members from Table 2
set-methods	Write-access to state and output variables

While the method's interface is generated automatically, the body code must be filled in by the model engine developer (recall Fig. 4). This body code provides the actual model equations (Eqn. (1)) together with an appropriate analytical or numerical solution. In the case of numerical integration, the body of the `derivs` method must also be filled with code. This auxiliary method is used to pass expressions for the state variables' derivatives to the generic, built-in ODE solver. It has to be emphasized that the body code of the `simulate` and `derivs` methods typically consists of simple variable declarations, assignments, function calls, and conditional expressions. For a model engine developer it is, therefore, sufficient to know the basics of procedural programming but no understanding of object-oriented concepts is required.

The get- and set-methods listed in Table 3 are composed of generic and generated source code (Fig. 4). They provide fast but safe access to an object's state variables, forcings, and parameters within the body of `simulate`. To enforce a transparent, human-readable source code, all these data items and functions are accessible *by name*.

3.2.6. Handling of external forcings

A model engine must be capable of reading and handling user-supplied time series of external forcings. Typically, the data also have a spatial dimension, thus, they can be regarded as a matrix where each row corresponds to a time interval and each column holds the data of a particular location. The model engine is responsible for assigning adequate forcing data to every simulated object. This is often complicated due to a mismatch between the external data and the objects in terms of their spatial location and/or resolution (Fig. 5).

In general, the external data either refer to observation points (Fig. 5, case c) or grid cells (cases a and b). The simulated objects, however, may be of any desired shape or extent. In a hydrological model, for example, the objects typically correspond to sub-basins (Fig. 5, cases b and c) or cells of a regular grid (case a).

ECHSE-based model engines use a generic approach to cope with most of the thinkable cases in a computational efficient way:

1. With respect to a particular forcing variable, every modeled object may be linked to one or more *external input locations* (EILs).
2. Each of those links is associated with a weight w subject to $0 \leq w \leq 1$.
3. For a particular object and forcing variable, the sum of the weights is 1.

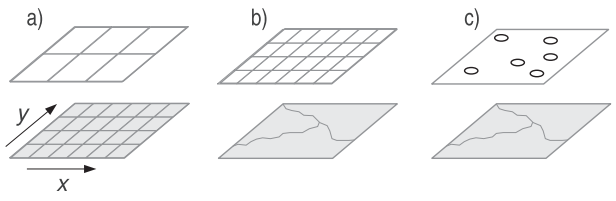

Fig. 5. Typical spatial discretizations of a distributed hydrological model (bottom row) and spatial resolutions of meteorological forcing data (top row).

The simplest case is a nearest-neighbor approach where each object is linked to a single EIL ($w = 1$). Other techniques of spatial interpolation, including inverse-distance and Kriging methods, require a single object to be linked with multiple EILs.

The weights w must be calculated externally. For all objects and forcing variables, the non-zero weights must be supplied in a model input file along with the names of the respective EILs. The apparent inconvenience is more than compensated by two advantages. First, virtually any approach to data regionalization may be applied without modification of the generic model engine's core. Second, the precomputed weights can be re-used whenever the model is run with identical external forcing. This can reduce computation times drastically, for example during parameter calibration.

It is a prerequisite of the presented approach that both the links and the corresponding weights are constant in time. Thus, neither the number and nor the spatial position of the EILs must change during the simulation. This also implies that the time series at all EILs must be free of gaps. Therefore, it is necessary to complete any missing values before feeding time series data into an ECHSE-based model, e. g. by means of interpolation in space and/or time. The lack of built-in mechanisms to manage the filling of gaps appears inconvenient at first glance. However, it enables the modeler to choose estimation algorithms adapted to the specific data at hand while, at the same time, the framework's source code remains generic and concise.

ECHSE-based models expect external forcing data to be in interval-based form. Both time-step averages (e. g. for temperature data) or sums (e. g. daily precipitation) are supported. The temporal resolution of the external time series must be identical to (or a multiple of) the modeling time step. Time series of variable resolution are acceptable as long as the above condition is fulfilled.

If the temporal resolution of the forcing data is incompatible with the desired model time step, the external time series need to be pre-processed using adequate techniques of aggregation, disaggregation, or interpolation. The same applies to time series of instantaneous values that need to be transformed into interval-based series before use in a model. As in the case of gap-filling, such data processing is outsourced to external tools for the sake of both maximum flexibility and simplicity of the ECHSE framework.

Neither the temporal resolution nor the length of the time series have a relevant impact on the model's memory consumption. The latter depends on the total number of EILs only.

3.3. Steps of model engine development

3.3.1. Outline and definition of a toy example

The development of a new model engine from scratch comprises the following steps:

1. Names have to be invented for all application-specific classes.
2. For each class, the data members (Table 2) must be declared in an ordinary tabular text file.
3. The code generator must be run on the set of class declaration files established in step 2.
4. The processes have to be implemented by filling the bodies of the `simulate` method and (optionally) the `derivs` method with C++ code.
5. The generic, generated, and hand-written parts of the source code must be compiled into the executable model engine.

In subsequent sections, the steps 2, 3, and 4 of the above list are illustrated with the example of a *linear reservoir* class. Such a reservoir can be thought of as a bucket whose outflow rate q_{ex} is proportional to its storage volume v. Taking into account the

mass balance, the dynamics are described by Eqn. (2) where q_{in} is the inflow rate and k is a retention parameter with the unit of time.

$$\frac{dv}{dt} = q_{in} - q_{ex} = q_{in} - \frac{v}{k} \tag{2}$$

As long as q_{in} is a simple function of time t, Eqn. (2) can be integrated analytically. The particular solution for the state variable v presented in Eqn. (3) assumes that q_{in} is constant over a time step of length Δt.

$$v(t_0 + \Delta t) = (v(t_0) - q_{in} \cdot k) \cdot e^{(-\Delta t/k)} + q_{in} \cdot k \tag{3}$$

3.3.2. Declaration of data members (step 2)

The necessary *declaration* of the data members of the example class is presented in Fig. 6. The information is held in a tabular text file for later processing by the code generator. Each record declares a single data item characterized by its name and type (cf. rightmost column of Table 2). Note that, in this example, two output variables were declared to publish information on both average and instantaneous outflow rates. In fact, only the two leftmost columns in Fig. 6 are processed by the code generator. The displayed auxiliary columns are useful, however, for the purpose of documentation.

3.3.3. Code generation (step 3)

Using a declaration of data members as in Fig. 6 together with the corresponding class name (e. g. '`LinReserv`'), the code generator outputs several sections of C++ source code, split over different files. These generated sections comprise:

- the declaration of a derived class '`LinReserv`' where the bodies of the `simulate` and `derivs` methods (see Section 3.2.5) are blank, except for `#include` directives to import hand-written code (see Section 3.3.4 below);
- the declaration of static constants for use with the get/set methods (Table 3), enabling efficient access to data members by its names (as specified in the second column of Fig. 6);
- code to facilitate the handling of simulated objects in general and the management of class-specific parameters (see Table 2) in particular.

For the example presented here, the generated code consists of about 150 lines. Real-world model engines typically deal with a whole set of different classes (see, e. g. Section 4.2) and the generated code easily spans thousands of lines then.

3.3.4. Writing of body code (step 4)

The hand-coded body of the example class' `simulate` method is shown in Fig. 7. For clarity, the names of all data access methods (last two lines in Table 3) are printed in italic letters. The generated

0	*type*	*name*	*comment*	*unit*
1	stateScal	v	Storage volume	m^3
2	inputExt	qin	Inflow rate	m^3/s
3	paramNum	k	Retention constant	1/s
4	output	qex_end	Qutflow at end of time step	m^3/s
5	output	qex_avg	Time-step averaged outflow	m^3/s

Fig. 6. Declaration of data members of a linear reservoir class. The inflow is assumed to be an external forcing.

```
1 double v_new= (stateScal(v) - inputExt(qin) * paramNum(k))
2   * exp(-delta_t / paramNum(k)) + inputExt(qin) * paramNum(k);
3 set_output(qex_avg)= inputExt(qin) - (v_new - stateScal(v))
4   / delta_t;
5 set_output(qex_end)= v_new / paramNum(k);
6 set_stateScal(v)= v_new;
```

Fig. 7. Body of the `simulate` method of a linear reservoir class adopting the analytical solution from Eqn. (3) (C++ code).

constants used to access the data *by name* appear in bold italic letters. Finally, the bold-faced identifier `delta_t` is a formal argument of `simulate` representing the user-defined model time step.

The initial two lines in Fig. 7 represent Eqn. (3). Here, the value of the right hand side expression is stored in a temporary variable. This is because the values of both the initial and the updated storage volume appear in the mass balance used to calculate the time-averaged outflow rate (lines 3–4). The instantaneous outflow rate is set in line 5 and the value of the state variable is finally updated in line 6.

Fig. 7 demonstrates that basic knowledge of procedural programming is sufficient to implement a simple class. This is particularly true in cases where ODEs can be solved analytically (e. g. Eqn. (3)) and, consequently, the body of the `derivs` method remains blank.

3.3.5. Design considerations

The actual art of model engine development is to write re-usable model components. With respect to ECHSE-based model engines, this can be achieved in two ways. Firstly, one should split very specific, complex classes into sub-classes. This increases the chance that the entire (sub)-classes can instantly be re-used in different model engines. Secondly, it is advisable to wrap potentially re-usable code sections into functions and to collect those functions in a library. The functions can then be accessed by all classes importing the library.

3.4. Set-up of an ECHSE-based model

The set-up of a model for a specific system is a straightforward process comprising the following steps:

1. A table has to be provided to declare all objects (e. g. sub-basins, river segments, etc.) by specifying a unique identifier and the corresponding class name.
2. A table with information on the objects' interactions must be set up. Each record in such a table maps an internal forcing variable of a *target object* to the output variable of a *source object*.
3. External forcings must be assigned to the objects. In particular, the weights discussed in Section 3.2.6 need to be specified.
4. Initial values must be assigned to all state variables.
5. Parameter values have to be provided in a number of class-specific tables.
6. A configuration file must be set up. This keyword-driven file is read by the model engine at the very beginning of execution. It contains, for example, information on the simulation period, the model time step, and it also specifies the names of all other input files. Alternatively, the configuration data (or a subset thereof) can be supplied at the command line.

All the above information is held in plain text files of tabular format. Although the files could be prepared manually, this is

hardly feasible in the case of distributed models for larger areas. Therefore, a number of preprocessing tools have been developed in recent years with a focus on hydrological catchment modeling. The most frequently used tools are called TOPOCATCH and METEOFILL (Kneis, 2012b).

The R-package TOPOCATCH provides a rather complete set of methods to pre-process spatial data for the purpose of hydrological catchment modeling. In particular, it is capable of identifying all basic objects (sub-basins, reaches, nodes) including their attributes and upstream–downstream relations from a digital elevation model (DEM). TOPOCATCH also provides methods for the extraction of river cross-section from a DEM as well as for the spatial regionalization of the cross-sections' hydraulic characteristics.

The METEOFILL tool is specialized on the filling of gaps in time series of meteorological variables (recall Section 3.2.6). It uses an inverse-distance approach to interpolate in space, optionally combined with linear regression (additive residual interpolation). METEOFILL is a C++ application optimized for efficient batch processing.

4. Application example

4.1. Model engine HYPSO-RR

The ECHSE framework has primarily been used to develop rainfall–runoff model engines. Here, a particular engine called HYPSO-RR is presented. It was developed as a component of a hydrological forecasting framework. HYPSO-RR is designated for the continuous simulation of stream flow in meso-scale river basins. It belongs to the category of conceptual model engines which make use of both strictly physical equations and empirical relations. Where possible, computational efficiency was given priority over a very sophisticated representations of processes. This was necessary to allow for real-time optimization (namely state updating) and uncertainty estimation (e. g. using precipitation forecast ensembles). Many of the basic equations were borrowed from existing model engines (see references in Section 4.2).

The data requirements of HYPSO-RR are adapted to the situation in Germany, where observed records of major meteorological variables are typically available. Nevertheless, the model engine was also applied successfully in other parts of the world where data are more scarce (Section 4.3).

A comprehensive documentation of the classes and the mathematical formulations of hydrological processes can be found in Kneis (2012a). The subsequent sections provide a brief outline of the important application-specific classes.

4.2. Classes and processes

4.2.1. Sub-basin class

The sub-basin class is responsible for simulating the water balance at sub-basin level. At present, only three basic types of land-use are distinguished: Vegetated soil, impervious surfaces, and water. For vegetated areas, runoff generation is computed based on the water balance of a homogeneous soil column. In the model, the generation of direct runoff (surface runoff, sub-surface stormflow) is bound to the occurrence of saturated zones. The areal percentage of those zones is estimated with the method of Zhao et al. (1980) and the corresponding rate of direct runoff is obtained using the analytical solution presented by Todini (1996, signs corrected). The slower components (interflow, groundwater recharge) are computed as functions of the soil water content as in LARSIM (Ludwig and Bremicker, 2006). To account for snow storage and melt, the water and energy balance of the snow cover is simulated. A collection of the relevant equations can be found in Tarboton and Luce (1996) or Kneis (2012a).

A simple approach is used to estimate evapotranspiration (ET). First, a potential rate of ET is estimated from short-wave (gross) radiation and temperature data using the Makkink equation (de Bruin, 1987). Then, the actual rate is obtained by multiplication with a crop factor (see e. g. Feddes, 1987) and a linear term accounting for the soil water content. The leaf area index is used as a proxy for the crop factor and a linear correlation is assumed between the two quantities. Compared to the Penman–Monteith method, the obtained estimates of ET are probably less accurate. However, the described approach has the advantage of being applicable with a minimum set of predictor variables.

Runoff concentration at the sub-basin level is modeled separately for the individual runoff components. As in LARSIM (Ludwig and Bremicker, 2006), the generated runoff for each component is routed through a single linear reservoir to emulate the distribution of travel times. The retention parameters of the linear reservoirs (symbol k in Eqn. (2)) are obtained as the product of a concentration time index (CTI) and dimensionless calibration factors. The CTI is derived from the sub-basin's size, shape, and topography while initial estimates of the calibration factors can be gathered from recession analysis.

The following state variables are considered in the sub-basin class:

1. Soil water content (−),
2. Snow water equivalent (m),
3. Energy content of the snow cover (kJ/m^2),
4. Albedo of snow surface (−),
5. Storage volumes of the linear reservoirs controlling runoff concentration (m^3).

The set of meteorological forcings comprises precipitation, air temperature, short-wave radiation, air pressure, wind speed, relative humidity, and cloudiness. For regions without snow cover, only the first four variables are relevant. The leaf-area index is also treated as an external forcing in order to allow for seasonal variation.

4.2.2. Reach class

In HYPSO-RR, a river reach is approximated by a linear reservoir with a variable retention parameter. The linear reservoir approach is adopted for its attractive analytical solution. The particular solution presented in Eqn. (4) allows for a linear change of the inflow rate q_{in} during a discrete time step of length Δt.

$$v(t_0+\Delta t) = v(t_0)\cdot x + \frac{a\cdot(x-1)}{b^2} + \frac{q_{in}(t_0)\cdot(x-1) - a\cdot\Delta t}{b} \qquad (4)$$

with

$$a = (q_{in}(t_0+\Delta t) - q_{in}(t_0))/\Delta t$$
$$b = -1/k$$
$$x = e^{(-\Delta t/k)}$$

For a linear reservoir, the relation between storage volume v and outflow q_{ex} is linear, namely $k = v/q_{ex}$. For a real channel section, however, this relation is generally non-linear. In order to apply the solution from Eqn. (4) anyway, the relation is locally linearized, i. e. $k \approx \Delta v/\Delta q$.

Using Manning's equation, it is fairly simple to establish approximate functions $k = f(v)$ and $k = f(q)$ from information on the cross-section's geometry, bed slope, and roughness. These functions are then used to select an appropriate k for use in Eqn. (4)

reflecting the current state $v(t_0)$ as well as the inflow rate q_{in} at times t_0 and $t_0 + \Delta t$.

4.2.3. Node classes

Node objects are responsible for merging water flows originating from different sources (i. e. other objects). They are used, for example, to represent junctions in the channel network. A typical node object has multiple internal forcing variables (representing the inflow rates) and (at least) one output variable holding the sum of all inflow rates to be made accessible for a downstream object. A node object does not have any state variables.

4.2.4. Reservoir class

This class can be used to model lakes and reservoirs based on the stage–storage relationship and a rating curve. Precipitation and lake evaporation are also taken into account. The water balance equation is integrated numerically to allow for arbitrarily complicated bathymetry data and rating curves.

4.3. Selected case studies

HYPSO-RR was used to set up rainfall–runoff models for the river basins and stream gages listed in Table 4. The calibration parameters were identified semi-automatically by running a sequence of Monte-Carlo simulations (see e. g. Kneis et al., 2014). The Nash–Sutcliffe index and the percentage bias served as independent objective functions.

The model for the small mountainous catchment of the Wilde Weißeritz was primarily set-up for benchmark tests. In particular, a comparison was made with an existing model (Kneis et al., 2012) based on LARSIM (Ludwig and Bremicker, 2006). This model engine is operationally used by many German and some Austrian flood forecasting centers. It was found that both LARSIM and HYPSO-RR capture the observed runoff with a similar goodness-of-fit. Using HYPSO-RR, Nash–Sutcliffe indices $\geq$0.75 were obtained during calibration and even higher values ($\geq$0.84) were found during validation in more recent years. In this particular catchment, the goodness-of-fit was found to be closely linked with the proper simulation of snow accumulation and melt.

The applications in the Philippines have their focus on short-term flood forecasting using radar-based precipitation estimates. For the Marikina Basin upstream of Manila, a Nash–Sutcliffe index of about 0.75 was obtained for the validation period (year 2012, rain-gage data as forcing). The hydrograph of the August 2013 flood that affected large parts of the Philippine capital was also reproduced well (Nash–Sutcliffe index 0.89). The skill of runoff forecasts was found to be generally positive for lead times up to 6 h. For up to 3 h, the forecast quality appears to be sufficient for practical uses, e. g. flood warning.

Table 4
Current application sites of the HYPSO-RR model engine and used time steps (Δt). DE: Germany, IN: India, PH: Philippines.

Basin/country	Δt	Stream gage	km^2
Weißeritz/DE	1 h	Rehefeld	15
		Ammelsdorf	49
Neckar/DE	1 h	Kirchtellinsfurt & 10 upstream gages	2317
Marikina/PH	1 h	Montalban	380
C. de Oro/PH	30 min	Kabula Bridge	1074
Mahanadi/IN	3–8 h	Salebhata	4400
		Kesinga	12,200
		Kantamal	20,900
		Tikarpara	127,000
		Mundali	134,000

The model for the Mahanadi River basin (Kneis et al., 2014) represents the first large-scale application of HYPSO-RR. A reasonable match between observed and simulated stream flow was obtained by model calibration (Nash–Sutcliffe indices between 0.77 and 0.86). For the purpose of validation, the optimized parameter sets were exchanged among the gaged sub-catchments. In most cases, the Nash–Sutcliffe index decreased only slightly (range 0.71–0.83). The Mahanadi study was also targeted at analyzing the computational efficiency of the ECHSE's generic core and HYPSO-RR in particular. For that purpose, models of different spatial discretization were set up. Within the considered limits (1500–150,000 simulated objects), computation times and memory consumption were found to be almost linearly related to the number of objects. The built-in shared memory parallelization allowed for a significant reduction of computation times on true multi-processor machines.

5. Conclusion

The presented modeling framework offers an infrastructure for the development of well-structured, object-based model engines. The ECHSE framework may be used productively in cases where custom solutions are required because existing ready-to-use model engines are either too complex or too simple. In particular, a use of the ECHSE framework may be advantageous in situation where a once established model structure needs to be safely modified later. This is the case, for example, if a preliminary model has to be developed rapidly with the option for later improvement. Another potential field of application lies in the exploration of structural model uncertainty.

ECHSE-based model engines are primarily applicable to systems dominated by feed-forward interactions between the simulated objects. The framework is thus suitable to develop engines for rainfall–runoff modeling or many water quality simulation problems, for example. Although feedback interactions are supported too, the accuracy of the corresponding numerical solutions is currently limited. Therefore, it would be unwise to use the ECHSE framework in a context where a large number of computational sub-units is coupled through feedbacks. This is the case, for example, in groundwater flow simulation or hydrodynamic problems requiring a solution of the full Saint Venant equations. It will be a focus of future work to examine numerical methods that ease the mentioned limitations without causing a general loss of computational efficiency. A predictor–corrector approach, specifically applied to the objects involved in feedbacks, might be a possible option. Nevertheless, feed-forward systems with a moderate number of feedbacks will remain the major field of application for ECHSE-based model engines.

In view of the framework's inherent limitations, future enhancements also need to be targeted at better support for the coupling of ECHSE-based models with external models (e. g. multi-dimensional PDE codes). This is to ultimately combine the strengths of both the framework-based and the component-based modeling approach outlined in Section 1.2. In its present version, the ECHSE framework does not provide a dedicated infrastructure, i. e. any inter-model exchange of data must be managed via files. Also, any mismatches with respect to the temporal resolution of the coupled models need to be handled by external pre-/post-processors. A possible way to go is to make ECHSE-based model engines compliant with a software component interface standard such as openMI (Gregersen et al., 2007). This requires that a model engine's key functionalities (e. g. initialization, update of inputs, integration of state variables over a finite time step, update of outputs) are exposed through an API as separate routines. Fortunately, the ECHSE's generic source code can be decomposed rather easily into

functions providing these functionalities. However, extensions are necessary, for example, regarding the updating of an object's external inputs. The dedicated methods still need to support data import from files (Section 3.2.6) but they must also be capable of exploiting exchange items supplied by linked external models.

Acknowledgments

The author thanks the current users of the ECHSE software for their feedback and motivation. The presented work was funded by the German Ministry of Education and Research (BMBF) through the PROGRESS project (grant no. 03IS2191A, group D2.2).

References

Argent, R., Brown, A., Cetin, L., Davis, G., Farthing, B., Fowler, K., Freebairn, A., Grayson, R., Jordan, P.W., Moodie, K., Murray, N., Perraud, J.-M., Podger, G.M., Rahman, J., Waters, D., 2008. WaterCAST User Guide. eWater CRC, Canberra.

Argent, R., Voinov, A., Maxwell, T., Cuddy, S., Rahman, J., Seaton, S., Vertessy, R., Braddock, R., 2006. Comparing modelling frameworks – a workshop approach. Environ. Model. Softw. 21 (7), 895–910.

Argent, R.M., Perraud, J.-M., Rahman, J.M., Grayson, R.B., Podger, G.M., 2009. A new approach to water quality modelling and environmental decision support systems. Environ. Model. Softw. 24 (7), 809–818.

Arnold, J., Williams, J., Srinivasan, R., King, K., Griggs, R., 1994. SWAT, Soil and Water Assessment Tool. USDA Grassland, Soil and Water Research Laboratory.

Arnold, J.G., Fohrer, N., 2005. SWAT2000: current capabilities and research opportunities in applied watershed modelling. Hydrol. Process. 19, 563–572.

Beven, K., 2012. Rainfall-runoff Modeling - the Primer, second ed. Wiley.

Bhatt, G., Kumar, M., Duffy, C.J., 2014. A tightly coupled GIS and distributed hydrologic modeling framework. Environ. Model. Softw. 62, 70–84.

Branger, F., Braud, I., Debionne, S., Viallet, P., Dehotin, J., Henine, H., Nedelec, Y., Anquetin, S., 2010. Towards multi-scale integrated hydrological models using the LIQUID ® framework. Overview of the concepts and first application examples. Environ. Model. Softw. 25 (12), 1672–1681.

Bruggeman, J., Bolding, K., 2014. A general framework for aquatic biogeochemical models. Environ. Model. Softw. 61, 249–265.

Castronova, A.M., Goodall, J.L., 2010. A generic approach for developing process-level hydrologic modeling components. Environ. Model. Softw. 25 (7), 819–825.

Clark, M.P., Kavetski, D., 2010. Ancient numerical daemons of conceptual hydrological modeling: 1. fidelity and efficiency of time stepping schemes. Water Resour. Res. 46 (10), W10510.

Clark, M.P., Slater, A.G., Rupp, D.E., Woods, R.A., Vrugt, J.A., Gupta, H.V., Wagener, T., Hay, L.E., 2008. Framework for understanding structural errors (FUSE): a modular framework to diagnose differences between hydrological models. Water Resour. Res. 44 (12), W00B02.

David, O., Ascough II, J.C., Leavesley, G.H., Ahuja, L., 2010. Rethinking modeling framework design: object modeling system 3.0. In: Proceedings of the 2010 International Congress on Environmental Modelling and Software. International Environmental Modelling and Software Society (iEMSs).

David II, O., Ascough, J., Lloyd, W., Green, T., Rojas, K., Leavesley, G., Ahuja, L., 2013. A software engineering perspective on environmental modeling framework design: the object modeling system. Environ. Model. Softw. 39, 201–213.

de Bruin, H.A.R., 1987. From Penman to Makkink. In: Hooghart, J.C. (Ed.), Evaporation and Weather: Proceedings and Information No. 39. TNO Committee on Hydrological Research, The Hague.

DHI, 2013. MIKE 21/3 Ecological Modeling (MIKE 21/3 ECO Lab FM Module), Short Description. URL. http://www.dhisoftware.com.

Dozier, A., David, O., Zhang, Y., Arabi, M., 2014. Modpi: a parallel model data passing interface for integrating legacy environmental system models. In: Ames, D., Quinn, N., Rizzoli, A. (Eds.), Proceedings of the 7th International Congress on Environmental Modelling and Software, June 15–19, San Diego, California, USA.

Feddes, R.A., 1987. Crop factors in relation to Makkink reference-crop evapotranspiration. In: Hooghart, J.C. (Ed.), Evaporation and Weather: Proceedings and Information No. 39. TNO Committee on Hydrological Research, The Hague.

Fenicia, F., Kavetski, D., Savenije, H.H.G., 2011. Elements of a flexible approach for conceptual hydrological modeling: 1. motivation and theoretical development. Water Resour. Res. 47 (11), W11510.

Formetta, G., Antonello, A., Franceschi, S., David, O., Rigon, R., 2014. Hydrological modelling with components: a GIS-based open-source framework. Environ. Model. Softw. 55, 190–200.

Förster, S., Kneis, D., Gocht, M., Bronstert, A., 2005. Flood risk reduction by the use of retention areas at the Elbe River. Int. J. River Basin Manag. 3 (1), 21–29.

Fritzson, P., 2004. Principles of Object-oriented Modeling and Simulation with Modelica 2.1. John Wiley & Sons, Ltd.

Gattke, C., Pahlow, M., 2006. Using object oriented methods for adaptive hydrological model development and uncertainty estimation. In: Proceedings of the 7th International Conference on Hydroinformatics, HIC 2006, Nice, France.

Gregersen, J.B., Gijsbers, P.J.A., Westen, S.J.P., 2007. OpenMI: open modelling interface. J. Hydroinform. 9 (3), 175–191.

Hindmarsh, A.C., 1983. Scientific Computing. North-Holland, Amsterdam, Ch. ODEPACK, A Systematized Collection of ODE Solvers, pp. 55–64.

Kneis, D., 2007. A Water Quality Model for Shallow River-lake Systems and Its Application in River Basin Management. Ph.D. thesis. University of Potsdam, Institute of Geoecology. URL. http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-14647.

Kneis, D., 2012a. Eco-hydrological Simulation Environment (ECHSE) - Documentation of Model Engines. University of Potsdam, Institute of Earth- and Environmental Sciences. URL. http://echse.bitbucket.org/downloads/documentation/echse_engines_doc.pdf.

Kneis, D., 2012b. Eco-hydrological Simulation Environment (ECHSE) - Documentation of Pre- and Post-Processors. University of Potsdam, Institute of Earth- and Environmental Sciences. URL. http://echse.bitbucket.org/downloads/documentation/echse_tools_doc.pdf.

Kneis, D., 2012c. Eco-hydrological Simulation Environment (ECHSE) - Documentation of the Generic Components. University of Potsdam, Institute of Earth- and Environmental Sciences. URL. http://echse.bitbucket.org/downloads/documentation/echse_core_doc.pdf.

Kneis, D., 2012d. Eco-hydrological Simulation Environment (ECHSE) - Installation and Administration Guide. University of Potsdam, Institute of Earth- and Environmental Sciences. URL. http://echse.bitbucket.org/downloads/documentation/echse_install_doc.pdf.

Kneis, D., Bürger, G., Bronstert, A., 2012. Evaluation of medium-range runoff forecasts for a 50 km2 watershed. J. Hydrol. 414–415, 341–353.

Kneis, D., Chatterjee, C., Singh, R., 2014. Evaluation of TRMM rainfall estimates over a large Indian river basin (Mahanadi). Hydrol. Earth Syst. Sci. 18, 2493–2502.

Kneis, D., Förster, S., Bronstert, A., 2009. Simulation of water quality in a flood detention area using models of different spatial discretization. Ecol. Model. 220, 1631–1642.

Kneis, D., Knösche, R., Bronstert, A., 2006. Analysis and simulation of nutrient retention and management for a lowland river-lake system. Hydrol. Earth Syst. Sci. 10, 575–588.

Kralisch, S., Krause, P., David, O., 2005. Using the object modeling system for hydrological model development and application. Adv. Geosci. 4, 75–81.

Liebscher, H.-J., Mendel, H.G., 2010. Vom empirischen Modellansatz zum komplexen hydrologischen Flussgebietsmodell – Rückblick und Perspektiven (From empirical approaches to complex hydrological catchment models – Review and prospects). Bundesanstalt für Gewässerkunde (in German).

Ludwig, K., Bremicker, M. (Eds.), 2006. The Water Balance Model LARSIM - Design, Content and Application. Freiburger Schriften zur Hydrologie, vol. 22. University of Freiburg, Institute of Hydrology.

Meyer, B., 1997. Object-oriented Software Construction, second ed. Prentice Hall.

Mooij, W.M., Brederveld, R.J., de Klein, J.J., DeAngelis, D.L., Downing, A.S., Faber, M., Gerla, D.J., Hipsey, M.R., 't Hoen, J., Janse, J.H., Janssen, A.B., Jeuken, M., Kooi, B.W., Lischke, B., Petzoldt, T., Postma, L., Schep, S.A., Scholten, H., Teurlincx, S., Thiange, C., Trolle, D., van Dam, A.A., van Gerven, L.P., van Nes, E.H., Kuiper, J.J., 2014. Serving many at once: how a database approach can create unity in dynamical ecosystem modelling. Environ. Model. Softw. 61, 266–273.

Perraud, J.-M., Seaton, S., Rahman, J., Davis, G., Argent, R., Podger, G., 2005. The architecture of the E2 catchment modelling framework. In: Zerger, A., Argent, R. (Eds.), International Congress on Modelling and Simulation (MODSIM 2005), pp. 690–696.

Petzold, L.R., 1983. Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations. Siam J. Sci. Stat. Comput. 4, 136–148.

Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P., 2002. Numerical Recipes in Fortran 90-The Art of Parallel Scientific Computing, second ed. Cambridge university press.

Qu, Y., Duffy, C.J., 2007. A semidiscrete finite volume formulation for multiprocess watershed simulation. Water Resour. Res. 43, W08419.

R Development Core Team, 2009. R: a Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, ISBN 3-900051-07-0. URL. http://www.R-project.org.

Rahman, J., Cuddy, S., Watson, F., 2004. Tarsier and ICMS: two approaches to framework development. Math. Comput. Simul. (MATCOM) 64 (3), 339–350.

Rahman, J., Seaton, S., Perraud, J.-M., Hotham, H., Verrelli, D., Coleman, J., 2003. It's time for a new environmental modelling framework. In: International Congress on Modelling and Simulation (MODSIM 2003), pp. 1727–1732.

Reed, M., Cuddy, S.M., Rizzoli, A.E., 1999. A framework for modelling multiple resource management issues – an open modelling approach. Environ. Model. Softw. 14 (6), 503–509.

Refsgaard, J., Storm, B., Clausen, T., 2010. Système hydrologique européen (SHE): review and perspectives after 30 years development in distributed physically-based hydrological modelling. Hydrol. Res. 41 (5), 355–377.

Regnier, P., Vanderborght, J.P., Steefel, C.I., O'Kane, J.P., 2002. Modeling complex multi-component reactive-transport systems: towards a simulation environment based on the concept of a knowledge base. Appl. Math. Model. 26, 913–927.

Reichert, P., 1994. Aquasim - a tool for simulation and data analysis of aquatic systems. Water Sci. Tech. 30 (2), 21–30.

Reichert, P., Borchardt, D., Henze, M., Rauch, W., Shanahan, P., Somlyody, L., Vanrolleghem, P.A., 2001. River Water Quality Model No. 1. IWA Publishing.

Singh, V., Frevert, D. (Eds.), 2002a. Mathematical Models of Large Watershed Hydrology. Water Resources Publications, Highlands Ranch, CO.
Singh, V., Frevert, D. (Eds.), 2002b. Mathematical Models of Small Watershed Hydrology and Applications. Water Resources Publications, Highlands Ranch, CO.
Sudicky, E.A., Jones, J.P., Park, Y.-J., Brookfield, A.E., Colautti, D., 2008. Simulating complex flow and transport dynamics in an integrated surface-subsurface modelling framework. Geosci. J. 12 (2), 107–122.
Tarboton, D.G., Luce, C.H., 1996. Utah energy Balance Snow Accumulation and Melt Model (UEB). Tech. rep. Utah State University and USDA Forest Service.
Teutsch, G., Krüger, E.H. (Eds.), 2011. Water Science Alliance – Prioritäre Forschungsbereiche (White Paper on Priority Fields of Water Research). Helmholtz Zentrum für Umweltforschung (UFZ) (in German).
Therrien, R., McLaren, R.G., Sudicky, E.A., Panday, S.M., 2010. HydroGeoSphere – a Three-dimensional Numerical Model Describing Fully-integrated Subsurface and Surface Flow and Solute Transport (Draft). Université Laval & University of Waterloo, Canada.
Todini, E., 1996. The ARNO rainfall-runoff model. J. Hydrol. 175, 339–382.
Wagener, T., Boyle, D.P., Lees, M.J., Wheater, H.S., Gupta, H.V., Sorooshian, S., 2001. A framework for development and application of hydrological models. Hydrol. Earth Syst. Sci. 5 (1), 13–26.
Wang, J., Endreny, T.A., Hassett, J.M., 2005. A flexible modeling package for topographically based watershed hydrology. J. Hydrol. 314 (1–4), 78–91.
Watson, F.G.R., Rahman, J.M., 2004. Tarsier: a practical software framework for model development, testing and deployment. Environ. Model. Softw. 19, 245–260.
Welsh, W.D., Vaze, J., Dutta, D., Rassam, D., Rahman, J.M., Jolly, I.D., Wallbrink, P., Podger, G.M., Bethune, M., Hardy, M.J., Teng, J., Lerat, J., 2013. An integrated modelling framework for regulated river systems. Environ. Model. Softw. 39, 81–102.
Whelan, G., Kim, K., Pelton, M.A., Castleton, K.J., Laniak, G.F., Wolfe, K., Parmar, R., Babendreier, J., Galvin, M., 2014. Design of a component-based integrated environmental modeling framework. Environ. Model. Softw. 55, 1–24.
Zeigler, B.P., 1990. Object Oriented Simulation with Hierarchical Modular Models: Intelligent Agents and Endomorphic Systems. Academic Press.
Zhao, R.-J., Zuang, Y.-L., Fang, L.-R., Liu, X.-R., Zhang, Q.-S., 1980. The Xinanjiang model. In: Hydrological Forecasting, Proceedings of the Oxford Symposium, vol. 129. IAHS-AISH Publ. IAHS Press, Wallingford, UK, pp. 351–356.