Community modeling systems: classification and relevance to hydrologic modeling

Bo Lu and Michael Piasecki

ABSTRACT

Numerical modeling in the water sciences has been shifting from developing single or specific purpose-oriented tightly intertwined model applications to integrated model systems addressing more complex and interlinked geo-physical, -chemical and -biological processes across all strata of the critical zone geo-volume. This is a response to a number of important issues that span from preservation of legacy software, to a higher degree of development cost efficiency, to the realization that processes in one strata depend on others, to harmonizing software system usage, and to improving code provenance and repeatability of model runs. Consequently, a number of community modeling systems (CMS) have either been proposed or are being developed with individual communities typically taking the lead to develop a CMS for their constituency. While the development of CMS is a major step forward in trying to harmonize modeling efforts, chosen approaches vary with numerous efforts underway to arrive at a workable and functional CMS. This review seeks to provide an overview of these efforts, with a focus on those that address processes located in the critical zone, and tries to assess their degree of success based on some general criteria for the development of CMS.

Key words | community modeling system, hydrologic modeling, legacy software, software benchmarks, software libraries, standard interfaces

Bo Lu
HydroChina,
Beijing,
China

Michael Piasecki (corresponding author)
Department of Civil Engineering,
City College of New York,
New York,
USA
E-mail: *mpiasecki@ccny.cuny.edu*

INTRODUCTION

Over the past decade the focus of the hydrologic modeling community has been shifting away from developing new models that address isolated aspects of water movement to more holistic view models that also include chemical, physical and biological processes within a layered geo-volumes framework. While there is no clear definition of where the boundaries of these geo-volumes are, in our definition the geo-volume is best aligned with that of the 'critical zone' of the Earth, which extends from a layer approximately 100 m in the subsurface to the 500 mBar layer in the atmosphere of the hydrologic cycle (Brantley *et al.* 2006). The critical zone is therefore the target realm in which we seek to track the movement of water and try to determine stores and resident times of where the water resides. This in turn requires an integrated modeling approach rather than a piecemeal execution of software packages originally developed for isolated layers in this zone (Hewett *et al.* 2010).

Development of such integrated modeling systems is typically beyond the scope of individual researchers, instead requiring team efforts to assemble software systems which can address processes at the envisioned complexity levels (Abbott & Vojinovic 2009; Wagener *et al.* 2009). It is not surprising then that considerable team effort has been spent on seeking solutions to create integrated modeling approaches within a community system context. The trajectories chosen for creating these systems are however quite diverse, and are motivated by the many objectives and interest spectra that exist within the hydrologic community. These include: promoting 'popular' software systems for more widespread use; creating link environments for standalone software packages so they can run simultaneously;

doi: 10.2166/hydro.2012.060

developing comprehensive representations of all geo-layers in the critical zone and coupling them through deterministic means (i.e. a formal representation of the physics via ordinary or partial differential equations); developing complex software code structures in which modules can be added or subtracted depending on the specific modeling objectives which need to be addressed; and creating collections of different modeling approaches including datasets where a modeling community rallies around a geospatial context with specific scientific questions (e.g. Chesapeake Bay) rather than a set of software packages or a specific modeling environment.

In this context we will speak of community modeling systems (CMS), a popular term to describe efforts to develop complex evolving and adaptable modeling systems through collaborative partnerships (Dickinson *et al.* 2002; Voinov *et al.* 2008). We include in this list types of software systems that have been developed by a single entity, but have acquired a large following and can thus be added to the CMS naming umbrella.

While it is impossible to give a complete global overview of all legacy software packages and modeling systems ever developed, we seek to address those that have achieved a certain degree of popularity, that promise to address a new approach or paradigm and that are different from the previously mentioned to show the breadth of what has and is being done in this field. We first outline our thoughts on how to arrive at the suggested classification system, followed by an attempt to discuss some of the pros and cons of the (three) classes we have identified. Within these three classes we place a higher degree of focus on the third, coupling systems, because it has seen by far the highest level of activity. While we realize that this skews the balance of this review somewhat in favor of this class, we think it important to work the distinct differences with a little more emphasis on some of the underlying technologies used. We stress however that our focus will be on the general idea of a chosen approach and not on elucidating in depth the subtle differences of the technical underpinnings of each approach (which exceeds the scope of this article). For example, we will not address in detail metadata issues that govern data ingestion into models, exchange between model components, interfaces between models or the complexity of data (both in terms of quantity and quality) needed to feed and drive any of the modeling systems. While we realize that this could be of interest and be used as an additional classifier rubric, it is beyond the space limitations of this article to address them at a sufficient level. Finally, we will provide an outlook of future approaches that could be adopted to build a CMS.

CLASSIFICATION OF CURRENT CMS

In trying to bring some order to the many approaches that exist, the authors decided to seek a simple classification system within which to group the many approaches and also to provide a context to discuss some of the pros and cons of each approach. There are many aspects that could be used to draw classification lines through the collection of modules, models and technical approaches used for creating a CMS, thus complicating the matter. However, the authors found that a reasonable broad classification with a few subcategories could be achieved when examining software structures/type and mode of community engagement.

Using these two guidelines we have been able to identify three subclasses. The first class concerns those approaches that are based on geospatial context. In this approach, the CMS is not at all defined by technical specifications or software package preferences but rather by the common interest in a specific geographic region such as an estuary (e.g. Chesapeake Bay) or a lake region (e.g. the Great Lakes). Consequently, the CMS consists of a collection of many different models that use different software applications and grid configurations with as much different temporal coverage and scope definitions of what each modeling effort is/was aiming at achieving. A key opportunity in this approach is to deliberately generate duplication, so extensive comparisons can be made between different modeling approaches addressing the same question or objective. Another advantage is the ability to embrace and welcome a substantial degree of modeling diversity into the CMS without requiring adaption of software packages or specific modeling approaches. In a way, this is an environment where everybody is welcome regardless of their background within the region; the associated science questions are the only common denominator. Disadvantages of this approach include the lack of interoperability between models, no

means to couple and link and no means of a formalized trajectory to improve modeling capability (i.e. addressing more complex modeling questions using more holistic problem definitions) and also no mechanism to add and contribute new software segments.

The second class concerns the use of a single software assembly which, in contrast to the first class, is entirely void of any geospatial reference. This class can be divided into two methods. The first uses a single software package (often quite massive) in which subcomponents can be switched on or off using flags and switches defined in input files, often accompanied by a collection of auxiliary programs that help in the pre- and post-processing tasks. The second uses a more modular system where components can be linked and activated during the compilation and link steps and can also be based on different programming languages. The system offers a suite of modules from which to pick and can therefore be tailored to encompass many modeling domains that are grouped in sequential order. A major advantage of these systems is the fact that code control is handled by a few dedicated individuals, thus ensuring a high degree of bug-free code assembly and the ability to use the same software version across different applications. Community involvement for these systems occurs via feedback loops such as bug postings, blogs and community chat lines where suggestions and reports and even software segments are funneled back to the developers for inclusion within the next release.

The third class takes an entirely different approach to bringing modeling functionality together and, as for the second class type, uses no reference to common geospatial interests. In this class CMS are based on linking frameworks or 'glue' layers that allow software systems to communicate with each other during runtime. A key feature here is that legacy software programs remain largely untouched and simply need to adhere to a communications interface that permits exchange of variables and data during execution. The major advantage is that the link system does not concern itself at all with the underlying physics and therefore does not make any preconceived decisions about what type of physical world representation must be used, thus allowing maximum flexibility when assembling model configurations. While this approach opens wide the door for legacy code to be integrated, it also exerts little control on the appropriateness of modules used: a key advantage in the category-two approach.

In having focused on just two criteria to draw demarcation lines, it is important to point out other aspects that influence CMS development. From the perspective of modelers, essential issues coming to fore in the development of CMS are the proliferation of individual models and modules, the credibility of model/legacy software, the integration of independent models, the interoperability of both model components and accompanying datasets, the infrastructure maintenance and the question of how to best provide a means for the community to add to the CMS. For example, in the area of hydrology, a plethora of models has been developed over the past few decades that are employed in a wide spectrum of areas ranging from watershed management to engineering design (Singh & Woolhiser 2002). Since those code stacks are mostly standalone and were not intended to be linked to other computational kernels at the time of their creation, it is typically quite difficult to integrate two or more code implementations. It is even more challenging to pull components from different modeling environments and assemble them into a brand new software package. The technical impediments include lack of modular model structure, intertwining of user interfaces and computing kernels, varying computer languages used to encode the modeling kernel, distinct input and output data structures and poor documentation of source codes, to name a few (Rizzoli *et al.* 1998).

Some legacy codes have proven to be quite popular over the years, such as the models developed by the Hydrologic Engineering Center (HEC, http://www.hec.usace.army.mil/) at the US Army Experiment station in Vicksburg, Mississippi (e.g. HEC-River Analysis System, HEC-Hydrologic Modeling System). However, there are also a vast number of research legacy software programs that have only been used for individual research applications and have never had exposure to formal verification procedures. While Argent (2004) argues that harvesting and incorporating legacy codes (which represents countless man hours of effort in addition to being a very rich knowledge source) is an appropriate and even necessary step, it is also clear that this is not straightforward because of a typical lack of documentation, lack of credibility of the algorithms or methods encapsulated in the codes, incompatible

programming languages and a general lack of good coding practices (e.g. avoidance of FORTRAN-based GOTO statements, hardwired constants or structures that defy modularization and parallelization).

A dilemma often faced is whether to invest effort in vetting and restructuring legacy software for common use (after all, these software systems represent a fairly rich knowledge base), or to launch the creation of new software assemblies and leave the legacy codes behind. This decision will depend on the quality of the software systems versus the difficulty of recreating the software contents.

USING GEOSPATIAL CONTEXT TO DEFINE A CMS

The first type of CMS is regionally limited and only targets the study of processes within a particular region. It commonly involves a collection of independent third-party model systems or tools along with regional datasets that drive these models. In this case, community members enrich the CMS by submitting their models and accompanying data that have been collected and compiled for specific purposes and time frames. A typical representative of this group is the Chesapeake Community Modeling Program (CCMP), hosted by the Chesapeake Bay Research Consortium (CRC, http://ches.communitymodeling.org/documentation/pdf/ModelPreamble.PDF), an open source system of watershed and estuary models that are dedicated to the study of the Chesapeake Bay region on the eastern shore of the USA. It contains an assembly of watershed, hydrodynamic and biogeochemical models and additional modeling tools, along with the Chesapeake Bay Environmental Observatory (CBEO) data. However, the involved models are standalone executable programs, are sometimes license restricted and generally do not provide a true 'side-by-side' placement having used similar grid or mesh assemblies, model runtime frames and identical datasets to drive the model runs. Rather, the current effort of the Chesapeake Bay modeling community is more focused on the application and amelioration of those models, instead of construction of more comprehensive model systems via integration. However, a set-up of this structure permits the addition of other modeling efforts and the community can easily agree on a certain event or time frame that needs attention with several spawned independent modeling efforts running side by side.

A similar effort exists for the Great Lakes System that features the Great Lakes Observing System (GLOS 2010), which also engages in data monitoring efforts, similar to the Chesapeake Bay Research Consortium. It should be noted that entries within this class are few, hence the limited length of this section. Despite its obvious shortcomings, the idea has however been working well for the cited efforts because of its characteristic of being loosely coupled and at the same time all embracing, but also because of its low level investment of resources. The authors found this idea compelling enough to devote a separate section to it, especially since its underlying motivation is distinctly different from the other two classes.

USING MONOLITHIC CODE FRAMEWORKS TO FORM A CMS

A CMS within this category centers on the use of one specific or monolithic software package, sometimes organized using modular software architecture. While this model architecture typically features some degree of flexibility by allowing the on- and off-switching of a set of predefined modules, it makes the extension more cumbersome as change requests need to be submitted which are then integrated (Kuo *et al.* 2004). In this case the development group will determine the usefulness of the requests and then possibly extend the code, which makes this a potentially time-consuming process. Of course, the advantage lies in the fact that the development team can control version creep as well as test and validate prior to release. A typical example of this type of CMS can be found in the meteorological community, within which the MM5 (Penn-State/UCAR Mesoscale Modeling 5th Generation, http://www.mmm.ucar.edu/mm5) or the more recent Weather Research and Forecasting model (WRF, http://wrf-model.org/index.php) have been designed for mesoscale numerical weather prediction (Michalakes *et al.* 1998, 2001). The WRF model development is conducted through a set of 16 lead groups, each of which concentrates on one particular task such as the development of numerical software, the maintenance of model architecture and the integration of

models from related domains. A similar case is the family of three-dimensional ModFlow (USGS 2009) or ParFlow (LLNL, https://computation.llnl.gov/casc/parflow/overview.html) codes used to model subsurface flows. Their modular structure has enabled the integration of some additional simulation capabilities, for instance, simulation of surface-water, solute transport, aquifer-system compaction and land subsidence.

Approaches which use a generic component-based modeling framework that can integrate models and build up multi-component model systems, thus permitting a substantial degree of flexibility, also fall within this class. Examples are the Community Surface Dynamics Modeling System (CSDMS, Peckham 2008), the Partnership for Research Infrastructures in Earth System Modeling (PRISM, Valcke *et al.* 2006) and the Earth System Modeling Framework (ESMF, http://www.earthsystemmodeling.org/). The CSDMS uses a strategy called Common Component Architecture (CCA), which involves a set of tools and standards for modularizing component modules (Bernholdt *et al.* 2006). It also contains a language-interoperability tool called BABEL (Dahlgren *et al.* 2007) and a graphical user interface (GUI) for linking component modules within the high-performance computing environment called Ccaffeine (http://www.cca-forum.org/ccafe). BABEL can generate 'glue codes' for component modules written in different programming languages, including C, C++, Java, Fortran and Python. The CSDMS currently involves a variety of terrestrial, marine, coastal and hydrological modules that were submitted by community members, and have been or will be modularized as linkable component modules. The PRISM framework employs a standalone coupler called OASIS to handle synchronized data exchanges between numerical codes, a Standard Compile Environment (SCE) to retrieve and compile source codes and a Standard Running Environment (SRE) to maintain model execution (Valcke *et al.* 2004). Finally, the ESMF framework is a hierarchical collection of components that can be combined to form larger-scale models such as the atmospheric circulation model (GEOS5) that NASA deploys at its Goddard Space Center. Components can comprise physical domains on the Earth's surface such as hydrologic features (lakes, rivers, etc.), but also chemistry, vegetation and catchment processes as well as atmospheric turbulence and radiation models. The system permits the use of parallel computing environments and, through its coupler functionality, the module execution in sequential or concurrent mode (Hill *et al.* 2004; Collins *et al.* 2005).

There is also another group of models belonging to this class and concerns software systems for the simulation of river, estuarine and coastal process. This large group includes codes such as DELFT3D (the newest member added in January 2011 from DELTARS, http://www.deltares.nl/en), ROMS, TOMS, EFDC, SELFE, CH3D and ADCIRC, to name just a few. Each of these software packages has its group of followers or user community. We do not seek to explore this line of reasoning however, because there are quite a number of software systems in this subdomain (unlike for example ModFlow/ParFlow or MM5/WRF) and the estuarine/coastal modeling community is in fact quite divided over whether a specific program is better than another. Also, for many of these software systems the community is not engaged in a coordinated effort of improvement; they are either left alone as they are or are being converted into a myriad of derivatives as the original source code is altered by each individual who downloads the latest-known documented version. Because this specific software landscape is so diverse, it is not included in our discussion. Commercial modeling environments such as the MIKE family of the Danish Hydraulic Institute or the suite of codes used by the Dutch company DELTARES are also omitted, because this group is not license-free software.

Yet another aspect of this group is community-of-practice type environments. These are online hubs that seek to provide a forum to exchange information about models, download models and share modeling experiences via blogs and chat features. These hubs also offer the possibility of benchmark dataset creations or definition of a common testbed environment in which to compare model approaches against measured and validated data. Examples of this type of CMS include the Community Modeling and Analysis System (CMAS, http://www.cmascenter.org/) funded by the US Environmental Protection Agency which offers a suite of different application models, and the Integrated Environmental Modeling hub (iemHUB, http://iemhub.org/) with a similar outlook. A European effort which falls into this category is the European Network for

Earth System Modeling (ENES, https://verc.enes.org/), which '… aims to provide for the European Earthsystem modeling community a marketplace to discuss, inform and interact …' and '… collects and prepares information on the European Earth System Models and commonly used software tools'.

While the idea of forming interest groups to develop a new monolithic code structure has the advantage of bringing many minds to bear on the development, thus ensuring substantial intellectual focus and breadth, it is a fairly time-consuming task because of the large development group and the need for an organized versioning system. In addition, the tight source code control typically delays the transfer to other operating systems and also prevents the harnessing of a much broader community for software contributions. This in turn limits the incorporation of modules and externally developed software components (including legacy software) and also the porting and integration of software segments written in other programming languages.

USING COUPLING FRAMEWORKS TO DEVELOP A CMS

Coupling frameworks seek to provide a software layer into which external software systems can be embedded or linked in an attempt to overcome difficulties in the process of model integration, i.e. disparate model interface definitions, mixed programming languages, difference in data semantics and incompatible spatial and temporal scales (Holzworth *et al.* 2010). There is also the notion that the advancement of coupling frameworks and increased software commonality are key to facilitating the creation of more cohesive and collaborative communities (Killeen *et al.* 2006).

Components of coupling frameworks

The literature contains abundant examples of how coupling frameworks could look. We will try to give an overview on those that seem to be the most prominent. In a somewhat simplistic view, coupling frameworks are software layers that 'glue' together modules during runtime in such a fashion that data can be moved in and out of these components together with time (time manager) and spatial (re-gridding or spatial interpolation) control. In other words, as long as each of the components or modules abides by the rules and protocols set forth by the interface (or 'glue') definitions, any software system can be linked to another software package during runtime. This is a preferred approach when trying to link legacy software and requires the need to write so-called wrappers that mimic the coupling framework interface and hide the historic input/output (I/O) definitions of the legacy codes. While the range of features of a coupling framework largely depends on the requirements of its problem domain, we have tried to identify the most common as follows.

- *Model standard or protocol*: the kernel of coupling frameworks that commonly comprises standard interfaces that component modules should comply with, descriptions of model structure, data model, metadata tags and some other abstract standards.
- *Module library*: modules represent context-independent software units that can be separated from their original code base and turned into standalone executables with moderate change (Ciupke & Schmidt 1996). They are standardized, portable and are usually made available in the form of a dynamic-link library (DLL) or a component object model (COM) object. A module can encapsulate scientific concepts and algorithms or just be a service module.
- *Data analyst*: contains tools for data analysis, for instance, geospatial data processing, data statistics, data interpolation or extrapolation, etc.
- *Toolbox*: contains sophisticated tools and utilities that facilitate the development of component modules such as optimizer, ordinary differential equation (ODE) solver, unit converter, tools for wrapping or converting legacy software, data flow monitors, etc.
- *Workbench*: a platform for model linkage, execution and management, which usually supports graphical icon-based model construction (Maxwell & Costanza 1996).

Studies on coupling frameworks can be traced back to the 1990s, of which the Modular Modeling System (MMS, Leavesley *et al.* 1996a) is an early attempt. The MMS represents a hybrid approach between a traditional standalone model system and a component-based coupling framework. It is similar to the former where modules stay

as source code files and will be compiled and linked as executables during the linking process. The only exception is the absence of the standard interfaces, that contribute to make the modules compatible. Bongartz *et al.* (2003) pointed out that the MMS is an object-based rather than an object-oriented coupling framework, which does not support features such as abstraction, inheritance and encapsulation.

David *et al.* (2002) subsequently adopted the basic idea of MMS and presented a pure Java object-oriented framework called Object Modeling System (OMS). One highlight of the OMS is that it takes advantage of the introspection feature of the Netbeans (http://netbeans.org/) Independent Development Environment (IDE), thus supporting integration of component modules via metadata tagging and reflection techniques (David *et al.* 2004). The same principle is also evident in the Interactive Component Modeling System (ICMS, Reed *et al.* 1999; Cuddy *et al.* 2002) and The Invisible Modeling Environment (TIME, Rahman *et al.* 2003, 2004a, 2004b). While the former is built by a self-developed C-like language along a debugger called ICMSBuilder, the latter is born from the .NET IDE (Meyer 2001). The introspection mechanism embodies the concept of inducting the declarative language into model development.

Fekete *et al.* (2009) presented the idea of developing a declarative framework called the Next generation Framework for Aquatic Modeling of the Earth System (NextFrAMES), which attempts to provide a high-level abstraction of the scientific tasks. It provides an eXtensible Markup Language (XML) schema for describing model structure, along with a runtime engine that interprets the modeling XML, loads the modules, establishes the linkage and executes the model (Lakhankar *et al.* 2008). NextFrAMES uses the declarative language to integrate component modules, whereas the Spatial Modeling Environment (SME, Maxwell & Costanza 1994), another early attempt at a coupling framework, focuses on integrating component modules encoded in declarative language. It employs commercial declarative modeling environments such as STELLA (http://www.iseesystems.com/softwares/Education/StellaSoftware.aspx) to create modules that perform computations over a spatial unit, e.g. a grid cell. Those modules can then be loaded to the library, converted to C++ objects and executed within the geospatial context (MaxWell & Costanza 1995; Maxwell 1999). The declarative modeling approach, or system modeling, is advantageous in linking elements declared in the model with entities declared in a distributed knowledge ontology; however, the absence of a standard and common declarative modeling language often slows its reuse in other applications (Argent 2004; Argent & Rizzoli 2004).

Coupling frameworks that incorporate the common features summarized above are the Open Modeling Interface and environment (OpenMI, http://www.openmi.org/; Gregersen *et al.* 2005; Moore & Tindall 2005), the ModCom framework (Hillyer *et al.* 2003) and the Tarsier environment (Watson *et al.* 2001; Watson & Rahman 2004). These linking kernels represent a set of standard interfaces that describe, link and run compatible models (Knapen *et al.* 2009). Since integration work is handled solely by the interfaces (and as such remains de-coupled from the 'scientific' modules), module development is fairly uncomplicated with few and manageable constraints. Variants of the OpenMI approach have been presented by Castronova & Goodall (2010) in which they further simplify the standard interface of OpenMI, and present the so-called Simple Model Wrapper (SMW) which defines three methods for components, i.e. *Initialize*(), *PerformTimeStep*() *and Finish*() (Goodall *et al.* 2011). These three functions are designed to integrate a service-oriented interface that incorporates the web processing service (WPS) specifications and the OpenMI standard interface and enables models to be exposed as web services. An additional notable effort has been proposed by the Consortium for the Advancement of Hydrologic Sciences Inc. (CUAHSI, http://www.cuahsi.org) effort that features the Community Hydrologic Modeling Platform (CHyMP, http://www.cuahsi.org/chymp.html). The component-based CMS CHyMP comprises modular and linkable components for integrated water cycle modeling. The proposed CHyMP incorporates features of linking models across domains such as atmospheric, oceanic, ecological and environmental models, interfacing with databases of the CUAHSI Hydrologic Information System and providing functionalities of pre- and post-modeling analysis (Famiglietti *et al.* 2008, 2010). It will take advantage of existing coupling frameworks to link components, data and models together, thus allowing the community to focus on the development of model components.

Another alternative has been presented by Campbell & Hummel (1998) who introduced the Dynamic Information Architecture System (DIAS) which uses standard abstract classes to specify 'entity objects' and their dynamic behaviors. The entity objects conceptualize the real-world entities in ecological systems such as atmosphere, ocean and fish, and the dynamic behaviors represent simulation models. The DIAS allows the building, manipulation and simulation of complex ecological systems in which multiple objects interact via multiple dynamic environmental and ecological behaviors (Hummel & Christiansen 2002).

Modes of integration

Bulatewicz (2006) summarized four approaches based on how model codes are integrated: the monolithic approach, the scheduled approach, the component approach and the communication approach. As pointed out earlier, the monolithic approach uses pieces of code that are taken from different programs and merged together to form a new program. As a result of introducing the concept of modular and hierarchical decomposition of models and the subsequent emergence of the object-oriented programming design as the dominant programming methodology, monolithic models have become much more modular which has served as a successful base for the advancement of integration approaches (Padulo & Arbib 1974; Gamma *et al.* 1995). The other three approaches are similar in their use of software blocks and as such resemble the modular structure of the monolithic software approach. They differ in their code functionality and arrangement, however. In the scheduled and communication approach, the software components are independent models conducting scientific computations, and execute in a scheduled order and through messaging passing. The referred models can be further decomposed into a set of fine-grained software components, each of which is responsible for a specific function; this subdivision of a subdivision depicts the main idea of the component approach. Of the coupling frameworks mentioned here, the NextFrAMES falls into the scheduled approach group, the SME, DIAS and Tarsier adopt the communication approach and the others utilize the component approach.

In contrast to the previous classification approach, Brandmeyer & Karimi (2000) described a classification of methodologies using a five-layered pyramid with ascending order of complexity and sophistication: one-way data transfer at the pyramid base, loose coupling, shared coupling, joined coupling, and finally a layer on top for the tool coupling approach, as shown in Figure 1.

The lowest level (i.e. the one-way data transfer), in which two completely separate models have a single directional data transfer requiring manual user control, may not even be considered as a genuine coupling method. In this approach it is the users' responsibility to access the output of one model and adapt or re-format it to be the input of another model, an approach that is very common. The next level upgrade of this method is the loosely coupled approach, which supports bi-directional data transfer and provides an entirely hands-off data transfer. This approach works off a list of standardized file structures in which one program is expecting a certain file to appear in a specific subdirectory at some point during runtime, and vice versa. In the shared coupling approach, models are integrated with the aid of a shared component, for example, through a GUI or a database. While models in these first three approaches remain and execute as independent applications, the two remaining higher-level coupling schemas feature a higher degree of integration. In the joined coupling method, one model takes the dominant role and the other models are integrated via plug-ins, thus becoming secondary code insertions into the main or lead code. The concept of the top-level tool coupling is a hybrid approach representing a combination of the ideas embedded in the shared and joined couplings.

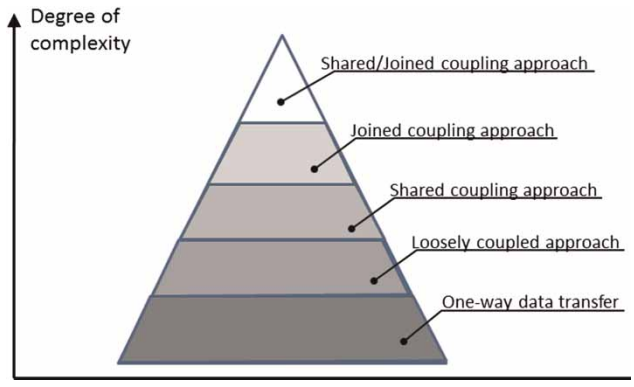

Figure 1 | Classification of coupling methods according to Brandmeyer & Karimi (2000).

In our opinion, all the classification schemas are valid as they provide an attempt to systematically list the various approaches. Inevitably, some of the software packages, frameworks and coupling approaches mentioned in this manuscript can be classified according to these two schemes. It is more important to simply outline the underlying ideas of the approach rather than to classify each within the schemas presented.

Component model standardization

Ideally, component modules should be independent standalone software entities that can be analyzed separately and then merged to form more complicated model systems (Voinov *et al.* 2008). For some of the coupling frameworks introduced earlier, i.e. OpenMI, Tarsier and ModCom, component modules are compiled into DLLs which can be loaded by the execution manager at runtime. In other coupling frameworks, component modules can be pre-compiled source codes (as in MMS) or metadata-based models specified by declarative languages (in SME) (Abel *et al.* 1994).

While a component module commonly remains as a 'black box' to the external environment, some degree of standardization can be achieved fairly easily if the component module is modified to expose an agreed-upon set of information. There are two requirements for this type of standardization. First, while component modules of an integrated model typically execute in sequence or parallel and are controlled by a runtime manager, the underlying technology is based on a small set of designated methods embedded in the source codes. These codes are called to perform initialization, main computation and termination. In this approach, each component module belonging to a coupling framework must use consistent method signatures which the runtime manager can recognize even if it requires customization. For example, the standard interface of the OpenMI (http://www.openmi.org/) contains the methods *Initialize*(), *Prepare*(), *GetValues*(), *Finish*() and *Dispose*() etc., whereas that of the ICMS contains *initialization*(), *main*() and *finalization*(). A summary of these and other methods is listed in Table 1.

The second requirement is that component modules should expose their input and output definitions so that a linkage can be set up by mapping the output of one module to the input of another explicitly or implicitly. Some coupling frameworks adopt a metadata-based approach, in which component modules are required to self-document their source codes where input/output properties will be assigned to certain variables along with other metadata. The TIME and the ICMS frameworks introduced earlier feature this type of communication approach. While TIME relies on the metadata-tagging feature given by the third-party .NET environment, ICMS achieves this through a proprietary declarative programming language called MickL (Rahman *et al.* 2004b). The OMS and the NextFrAMES approaches are similar cases, but instead of embedding metadata annotations in the source codes they define input/output quantities using XML-encoded declarations in external files. A more common approach is to specify exchange items in standard interfaces. For example, component modules using OpenMI should implement the *IExchangeItem* interface, which specifies exchanging items such as *Quantity* and *Elementset*. A *Quantity* contains metadata of a variable, such as ID, description and unit, etc., while an associated *Elementset* provides spatial information about the *Quantity*. ModCom uses an approach that is quite similar to OpenMI, where component modules need to implement the *SimObj* interface. The input/output variables are defined as SimData class and can be exposed via the *ISimObj.Input* or *ISimObj.Output* properties (Hillyer *et al.* 2003).

The above requirements for standardizing component modules are discussed under the assumption that they are written using a programming language that is supported by the coupling framework. For example, for the .NET version of OpenMI, component modules can be encoded using VB.NET or C# programming languages, which permit the runtime manager to invoke the embedded methods and parse input/output items. In contrast, OpenMI is not directly usable for a large number of modules that are written in other languages. The mixed-language issue could be addressed by performing language translations and re-coding, either manually or using translation tools (e.g. JNBridge, www.jnbridge.com). However, neither manual transformations (which can be complicated and prohibitively time-consuming) nor the translation tools (which often have difficulties in adequately translating complex programs) provide a high degree of success.

Table 1 | Summary of coupling framework technologies

Framwork	Model builder/ simulation control	Component model format (acceptable source codes)	Methods in standard interfaces	Input/output specified in component models	Communication mechanism
MMS	GUI (Xmbuild)	Executables/source codes file (Fortran/C)	declare(), initialize(), run(), main()	N/A	File transfer or sharing central database
SME	Configuration file	SMML Object(C++/Java (Fortran,C))	N/A	Specified in Frame classes	Message passing
ICMS	GUI (ICMSBuilder)	Plugin (MickL)	initialization(), main(), finalization()	Variable/fields attributes	Input/output mapping
DIAS	GUI (GeoViewer)/ Context Manager	N/A (Any)	N/A	Register input/ output parameters	Indirect communication via domain objects
Tarsier	GUI (Tarsino)	DLL (Bortland C++)	execute()	RegisterFields function	Data sharing and message passing
OMS	GUI (Model Editor)	Jar (Java/Fortran/C)	init(), run(), cleanup()	Attribute Editor	Input/output mapping
ModCom	GUI/ISimEnv interface	DLL (Any supporting COM)	StartRun(), EndRun(), HandleEvent()	Input/Output Class	Message passing interface and I/O actions from/to disk files
TIME	GUI	DLL (VB/C#/Fortran/ C++/Java)	runTimeStep()	Variable/fields attributes	Input/output mapping
OpenMI	GUI (Configuration Editor)	XML file (.omi) + DLL (Any)	initialize(), Prepare(), GetVaues(), Finish(), Dispose()	Input/Output Class	Request-reply mechanism
NextFrAMES	XML file	Plugin (C/C++)	initialize(), execute()	XML statements	Input/output mapping

Another alternative approach is the use of web services which support machine-to-machine integration and interoperation of web-based applications. This idea would entail the creation of web-services-based modules that can be accessed via standard interfaces, such as the WPS, an Open GeoSpatial Consortium standard, or customized web services designed for model communications (Horak *et al.* 2008). Figure 2 depicts a simple example where Model A is a standardized model located in one machine and Model B represents a web-service-based model located in a remote machine. On the local side, an additional wrapper would be needed in order to tap into the web services of model B.

Communication mechanisms

Data exchange is the primary communication task between modules within a modeling system. In some early integrated models, modules achieved integration by transferring data files or only sharing databases. For example, Leavesley *et al.* (2005) coupled the BOR RiverWare model (Fulp *et al.* 1995) with the MMS via a shared relational database. The MMS

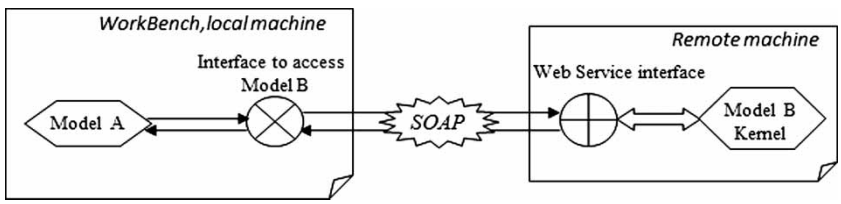

Figure 2 | Communication of web-service-based models (SOAP: simple object access protocol).

simulated streamflows and wrote the results into a database, while the RiverWare model read them and proceeded to evaluate reservoir-management strategies. To facilitate this set-up, customized data management interfaces (DMIs) needed to be written to assist the database to bridge the communications among the participating actors (Leavesley *et al.* 1996a, 1996b). This repetitive reading from and writing to a database approach suffers from long execution times, slowing down the overall model progression. As a result of IT infrastructure growth on data pipelines and improved data and machine communication protocols, most coupling frameworks now seek approaches that allow component modules to communicate dynamically and seamlessly in addition to using new paradigms of how data collections can drive physical models (Gourbesville 2009).

One approach adopted by some of the coupling frameworks is based on a request-reply mechanism. An example of this type of approach is again the concept behind OpenMI. During the initialization phase, it requires each compliant model to implement the 'GetValues' method which allows the retrieval of data from another OpenMI compliant module during runtime. In the new version of OpenMI (V2.0), links between components or models are established by passing the data definitions in the *GetValues*() call. This simplifies the communication because there is no need to predefine expected links. The OpenMI uses a purely single-threaded architecture where only one data request is handled at any time (Fortune *et al.* 2008); however, it now includes both quantitative (numbers) and qualitative values (such as 'dry' or 'wet') as descriptors.

Another approach can be described as input-output mapping. In some coupling frameworks, the linkage between two component modules can be explicitly specified by matching the output of one module with the input of the other using GUI tools. For example, the ICMS provides a dragging-and-dropping platform where component modules or model objects (e.g. subcatchment, stream) can be connected via a link-arrow. The underlying connection is then defined by configuring the link properties. As a modification to this approach, some coupling frameworks such as NextFrAMES specify those linkages implicitly. For the case of NextFrAMES, this is done by defining input/output components separately in a modeling XML file including the definition of variables passed. The variable names are then mapped to related variables declared in the source codes of the plug-in embedded in a component model (Fekete *et al.* 2009). This type of semantic link enables the data transfer from one module to another, with the added advantage that point-to-point mappings often avoid semantic ambiguity and confusion.

Note that this approach requires the need for a sequential execution of the component modules to ensure that a data provider model is executed before the next-in-line data consumer. From this point of view, this approach lacks some of the flexibility inherent in the request-reply mechanism, which supports triggering the execution of a data provider model whenever it is needed. The last approach in this line is based upon in-memory data sharing and a message-passing mechanism. The Tarsier framework is an example in which the model structure and communication protocols are based on the 'observer' pattern of client-supplier computing (Gamma *et al.* 1995). Data can be registered as '*usees*' and shared among models and tools called '*users*'. If two *users* use the same *usee*, they are implicitly linked. When one *user* changes the *usee*, it will send a message to the other *user* which will respond immediately to the data changes (Watson *et al.* 2001). The message passing is handled by the *SendMessage* and *ReceiveMessage* methods implemented by *users*.

A key prerequisite for establishing valid communications between modules is that these modules are interoperable. Howie *et al.* (1996) defined interoperability as the ability of different programs to share and process information irrespective of their implementation language and platform. In the hydrologic domain, the component modules should also be interoperable with respect to spatial and temporal scales which could differ in format, resolution and reference system. For example, regional climate models mostly provide meteorological estimates (such as precipitation, temperature, air humidity and wind speed) with a spatial resolution of a few kilometers, whereas distributed hydrologic models are normally built up for higher-resolution analysis with grid sizes in the tens of meters range. When coupling them together, a step of downscaling transformation should be included in order to reconcile the scale difference. Downscaling can be carried out using different approaches. For example, Marke *et al.* (2011) present a general statistical approach that introduces a

correction of biases which has been applied for the downscaling of precipitation (Früh *et al.* 2006). Cubasch *et al.* (1996) investigated approaches of direct interpolation of the nearest gridpoints, time-slice and statistic downscaling in climate change experiments.

It is clear that the degree of interoperability will therefore hinge on the degree of sophistication that has been implemented in the interoperability tool box. Unit conversions, syntactic (format) transformations, temporal and spatial interpolation and extrapolation capabilities and semantic mediation (e.g. of keywords and variable names) are all services that are ideally embedded into this tool box which should be able to automatically act whenever it detects an incompatibility. Currently, the majority of coupling frameworks only feature a limited set of transformation tools.

Coupling frameworks and their use in CMS

As pointed out in the previous sections, coupling frameworks exist in a number of forms using different implementation strategies. They basically fall into two categories: those that use declarative statements in which the number and type of modules are recorded as well as the sequence in which the modules will be called; and those that use a GUI of some sort allowing the construction of module execution sequences using visual aids. The former typically uses a set of configuration files to describe the overall model structure, i.e. component modules and their connections. An engine accesses the configuration file at runtime, parses the model hierarchy, loads specified component modules, invokes the model computations and performs the model I/O. We can group NextFrAMES, SME and PRISM within this category. For a large model composed of many component modules however, the preparation of the configuration file(s) can be quite complicated and time-consuming in addition to requiring users to have a clear understanding of the coupling hierarchy.

Working with a GUI workbench is more user-friendly of course and, from a user point of view, visual aids for dragging and dropping connections supported by drop-down menus to navigate module libraries is simpler than encoding configuration files. Examples of coupling environments providing GUIs include OpenMI, ICMS and OMS. In most of these systems, an integrated model can be exported as a project file which can be re-loaded to the workbench, thus providing some degree of repeatability and provenance.

Whatever the adopted approach, we believe that coupling frameworks can serve very well as the backbone of a CMS. One of the key advantages lies in the fact that legacy code can be migrated into the environment (admittedly with some work) so that it complies with the necessary I/O and interface definitions. However, the convenience of legacy migration is not the only aspect to consider as there is also the frequency with which the coupling framework may experience upgrades and changes as it matures further. Additional points to consider are the extent of the existing library of compliant modules and codes, whether or not the system enjoys a large group of developers or an active user community from which to draw support, a rich support library that contains peripherals and visualization applications and if the framework supports high-performance computations on parallel machines or the cloud.

Among the discussed coupling frameworks, NextFrAMES is still under development and some of the others such as the OMS and the DIAS only have a few applications to date (Sydelko *et al.* 2001; Kralisch *et al.* 2005). While developed by a (small) group of researchers and therefore having some manpower behind it, the ModCom framework appears to have found little acceptance in practice; we found little evidence in the form of published references during the literature search that would demonstrate use of it in the wider modeling community. Also, being built on a Unix-based platform, MMS and SME may not have the chance to gain extensive popularity in the largely Windows-based hydrologic user community. Those frameworks developed using less-compatible languages such as ICMS (using MickL) and Tarsier (Borland C++) may be hindered in their degree of utilization and spread in the community. However, TIME and OpenMI have attracted a fairly large group of researchers, with TIME having incorporated most of the well-known hydrologic models in Australia (Bari *et al.* 2009; Rassam *et al.* 2009).

SUMMARY AND FUTURE WORK

In this paper we have reviewed current efforts in building CMS for the hydrologic community and to summarize

some of the salient points, both advantages and disadvantages, of the systems that we have been able to review. We deliberately focused on systems that are aimed at the hydrologic community with some efforts on the hydrologic periphery, such as atmospheric or estuarine/coastal modeling systems. We felt that these efforts have a place in this review as they are related to the hydrologic realm and also demonstrate some basic features that are potentially common to any CMS, regardless of the specific community.

We have classified current CMS into three categories. The first uses a specific region as an organizing principle, for which any number of models can be developed with no particular requirement as to what these models should be. This somewhat loose organizing principle has the advantage that additions are quite easily accomplished and the exchange of data, information and results is fairly straightforward. With this approach, models can be easily compared on defined grid structures and defined boundary conditions with the purpose of determining any performance advantages and any shortcomings of modeling assumptions. The real strength, however, lies in the ability to bring together researchers from many disciplines to discuss and learn from each other, perhaps the best forum for true interdisciplinary discussions, thus advancing modeling approaches. The disadvantages are also quite obvious; there is no software development component, no focus on just one software package, no use of coupling mechanisms and no ability to receive technical assistance unless from the developers of the software system that is currently being used. Holistic approaches are limited to whatever a single program can do, meaning that coupling and execution in parallel is not possible.

The second class is characterized by the use of a single software structure as the underlying organizing principle; i.e. instead of a single-region-multiple-software-applications frame, it is based on the idea of single-software-package-any-region. A key advantage of this system is that a single software package used by the entire community allows for a high degree of sophistication of this software; the pooled suggestions for improvements could be vast while at the same time the efforts for source code updating could be the responsibility of a few (expert) individuals. In addition, it limits the need for a large set of pre- and post-processors as the same utilities are used by everybody, and easy exchange of I/O data is possible between fellow users. The use of these approaches is also advantageous for modelers who do not want to be embedded in the technical details of software development and implementation. While this approach typically requires a team of dedicated developers, and as such is cost intensive, it will result in models that are robust and therefore have a high potential for drawing in a large user community. In addition, the development team may be able to provide high-performance computing facilities for which the software releases have been tested (an invaluable advantage).

The third class concerns coupling frameworks that seek to combine the computations of processes in many geo-volume strata addressing the linkages that exist between the movements of water and the many bio-, chemical- and physical (and even economical and life-cycle) processes that are present in the hydrologic realm. The key advantage of these systems is clearly in their potential to provide more holistic views of the environment through coupling legacy codes, without making preconceived assumptions and decisions about how the underlying physical, chemical and biological representations should look. Community contributions to software development or benchmark case studies are possible in these systems, as many of them feature gateways for externally developed code insertions in case the default modules need to be replaced.

While highlighting some of the pros and cons of the geographic and monolithic CMS, this paper also addressed the features of coupling frameworks and how these features pertain to CMS development. In short, it is fair to say the coupling framework approach offers a number of advantages for the development of a CMS. (1) The focus on mediation layers enables the CMS to incorporate new component/modules easily, especially when derived from legacy software systems. This feature can make CMS a powerful tool and also create the spirit of a true community environment. (2) Coupling frameworks provide infrastructure for mediating the execution and communication between quite disparate models; community members therefore only need to focus on the migration of their own models. (3) The portability and reusability features of compatible modules enable users to construct more complex software assemblies that have a wide range of applications. (4) Legacy programs can be wrapped to become compliant

modules without the need for extensive code modifications by using development tools. (5) Some coupling frameworks support parallel computations, which can be a venue for improving modeling efficiency. (6) A potentially large user community provides a good feedback pool that can be used to improve the coupling framework.

There are also a few disadvantages, however. First, some of the frameworks are established at a basic computational level, often requiring a somewhat steeper learning curve for using the chosen coupling framework. For example, a coupling framework may provide a standard set of interfaces encoded in a certain programming language, which makes it difficult to adopt or use if the user has little to no experience using this language. Second, when wrapping a piece of legacy software, the effort to wrap the code increases dramatically if the code is complex and monolithic in its structure and features complex data models that are not easily modified to work well with the chosen interface definitions. Additional work may be necessary to separate intertwining interfaces and to partition and modularize monolithic software systems. Third, coupling frameworks have been criticized for not seizing the opportunity to formulate a new modeling paradigm that seeks to couple governing equations of the hydrologic processes present in each geo-volume strata in a holistic fashion. This is to say, the coupling framework may perpetuate the existence of inadequate models (and the errors they produce) by coupling one inadequate model to another, thus producing a seemingly better outcome while in reality only adding the faulty results of one model to the faulty results of another. In other words, a coupling framework may facilitate data exchange during runtime, but does not link these models on a more conceptual and theoretical platform.

We propose that future work on the CMS development is likely to happen along a more Darwinian evolutionary track: many of the systems presented here are still in the process of being developed, others already have some degree of acceptance, while others may or may not survive in the future. In other words, future work will be spread across many different pathways and will involve testing as well as observing what happens as systems develop. In light of this, it is consequently difficult to arrive at a verdict as to what is best, or most promising, versus what has little chance of surviving or meeting community expectations. For this, much of the work currently carried out is still in its infancy.

It is also difficult to predict what technological developments will bring in the future and how these developments may impact one CMS versus another. For example, the increased use of internet technologies such as web services or cloud computing may open up previously unconsidered opportunities. There are also other frameworks that could be explored for CMS development such as scientific workflow engines. These are relatively new arrivals on the computing scene (at least for hydrologic modelers) and have not yet been investigated for their suitability to serve as a backbone for a CMS despite obvious advantages such as: a means to record provenance; automatic versioning system; smart connections to web services and any type of data store; ease of library development; and coupling of workflows from pre- to post-processing, to name just a few. The questions that could be answered here are how easily legacy software could be transferred into scientific workflow activities (or actors), how computing performance would suffer if executed through workflow engines or how applicable or universally transferrable a CMS is when attempting to use it any location across the world (or initially perhaps the USA). Future studies could also focus on: development of a CMS which could be linked through middleware; middleware that places any CMS computation within the high-performance computing arena; or frameworks that help to overcome semantic disparity between models in general and CMS in particular.

REFERENCES

Abbott, M. B. & Vojinovic, Z. 2009 Applications of numerical modelling in hydroinformatics. *Journal of Hydroinformatics* **11** (2–4), 308–319.

Abel, D. J., Davis, P. J. & Davis, J. R. 1994 The systems integration problem. *International Journal of Geographical Information Systems* **8** (1), 1–12.

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

Argent, R. M. & Rizzoli, A. E. 2004 Development of multi-framework model components. In: Transactions of the Second Biennial Meeting of the International Environmental Modelling and Software Society, Osnabrück, Germany, 14–17 June, pp. 365–370.

Bari, M. A., Shakya, D. M. & Owens, M. 2009 LUCICAT Live A modeling framework for predicting catchment management options. In *18th World IMACS/MODSIM Congress*, Cairns, Australia, 13–17th July, pp. 3457–3463.

Bernholdt, D. E., Allan, B. A., Armstrong, R., Bertrand, F., Chiu, K., Dahlgren, T. L., Damevski, K., Elwasif, W. R., Epperly, T. G. W., Govindaraju, M., Katz, D. S., Kohl, J. A., Krishnan, M., Kumfert, G., Larson, J. W., Lefantzi, S., Lewis, M. J., Malony, A. D., McInnes, L. C., Nieplocha, J., Norris, B., Parker, S. G., Ray, J., Shende, S., Windus, T. L. & Zhou, S. 2006 A component architecture for high-performance scientific computing. *International Journal of High Performance Computing Applications, ACTS Collection Special Issue* **20** (2), 163–202.

Bongartz, K., Flugel, W. A., Krause, P. & Taddei, U. 2003 Geoinformatics toolset for integrated river basin management (IRBM) of the Saale river, Thuringia, Germany. In *International Congress on Modelling and Simulation (MODSIM)*, Townsville, Australia, 14–17 July.

Brandmeyer, J. E. & Karimi, H. A. 2000 Coupling methodologies for environmental models. *Environmental Modeling and Software* **15**, 479–488.

Brantley, S. L., White, T. S., White, A. F., Sparks, D., Richter, D., Pregitzer, K., Derry, L., Chorover, J., Chadwick, O., April, R., Anderson, S. & Amundson, R. 2006 Frontiers in Exploration of the Critical Zone: Report of a workshop sponsored by the National Science Foundation (NSF), October 24–26, 2005, Newark, DE, p. 30.

Bulatewicz, T. F. 2006 Support for model coupling: An interface-based approach. Phd Thesis, Department of Computer and Information Science, University of Oregon.

Campbell, A. P. & Hummel, J. R. 1998 The dynamic information architecture system: an advanced simulation framework for military and civilian applications. In *Advanced Simulation Technologies Conference*, Boston, MA, USA, 5–9 April.

Castronova, A. M. & Goodall, J. L. 2010 A generic approach for developing process-level hydrologic modeling components. *Environmental Modelling and Software* **25**, 819–825.

Ciupke, O. & Schmidt, R. 1996 Components as context-independent units of software. In *Special Issues in Object-Oriented Programming, Workshop Reader of the 10th European Conference on Object-Oriented Programming*, Heidelberg, Germany, pp. 139–143.

Collins, N., Theurich, G., DeLuca, C., Suarez, M., Trayanov, A., Balaji, V., Li, P., Yang, W., Hill, C. & da Silva, A. 2005 Design and implementation of components in the earth system modeling framework. *International Journal of HPC Applications* **19**, 341–350.

Cubasch, U., Storch, H., Waszkewitz, J. & Zorita, E. 1996 Estimates of climate changes in southern Europe using different downscaling techniques. *Climate Research* **7**, 129–149.

Cuddy, S. M., Letcher, R. A. & Reed, M. B. 2002 Lean interfaces for integrated catchment management: rapid development using ICMS. In: *Integrated Assessment and Decision Support* (A. E. Rizzoli & A. J. Jakeman, eds). Proceedings of the First Biennial Meeting of the International Environmental Modelling and Software Society, 3, pp. 300–305.

Dahlgren, T., Epperly, T., Kumfert, G. & Leek, J. 2007 *Babel User's Guide, March 23 2007 Edition*. Center for Applied Scientific Computing, US Dept of Energy and University of California Lawrence Livermore National Laboratory.

David, O., Markstrom, S. L., Rojas, K. W., Ahuja, L. R. & Schneider, I. W. 2002 The object modeling system. In: *Agricultural System Models in Field Research and Technology Transfer* (L. R. Ahuja, L. Ma & T. A. Howell, eds). Lewis Publishers, New York, pp. 317–330.

David, O., Schneider, I. W. & Leavesley, G. H. 2004 Metadata and modeling frameworks: the object modeling system example. In *Proceedings of International Environmental Modelling and Software Society*, Osnabrück, Germany, 14–17 June.

Dickinson, R. E., Zebiak, S. E., Anderson, J. L., Blackmon, M. L., DeLuca, C., Hogan, T. F., Iredell, M., Ji, M., Rood, R., Suarez, M. J. & Taylor, K. E. 2002 How can we advance our weather and climate models as a community? *Bulletin of the American Meteorological Society* **83** (3), 431–434.

Famiglietti, J. S., Murdoch, L., Lakshmi, V. & Hooper, R. 2008 Community modeling in hydrologic science. *EOS, Transactions, American Geophysical Union* **89** (32), 292.

Famiglietti, J. S., Murdoch, L., Lakshmi, V. & Hooper, R. 2010 Towards a framework for community modeling in hydrologic science. Report from the 2nd Workshop on a Community Hydrologic Modeling Platform (CHyMP), Memphis, TN, 31 March–1 April.

Fekete, B. M., Wollheim, W. M., Wisser, D. & Vorosmarty, C. J. 2009 Next generation framework for aquatic modeling of the Earth System. *Geoscientific Model Development Discussions* **2**, 279–307.

Fortune, D., Gijsbers, P., Gregersen, J. & Moore, R. 2008 OpenMI-Real progress towards integrated modeling. In: *Practical HydroInformatics: Computational Intelligence and Technological Developments in Water Applications* (R. J. Abrahart, L. M. See & D. P. Solomatine, eds). Springer, Berlin Heidelberg, pp. 449–464.

Früh, B., Schipper, J. W., Pferiffer, A. & Wirth, V. 2006 A pragmatic approach for downscaling precipitation in alphine-scale complex terrain. *Meteorologische Zeitschrift* **15** (6), 631–646.

Fulp, T. J., Vickers, W. B., Williams, B. & King, D. L. 1995 Decision support for water resources management in the Colorado River regions. In: *Workshop on Computer Applications in Water Management* (L. Ahuja, J. Leppert, K. Rojas & E. Seely, eds). Colorado Water Resources Research Institute, Fort Collins, CO, Information Series No. 79, pp. 24–27.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. 1995 *Design Patterns: Elements of Reusable Object Oriented Software*. Addison Wesley, Reading, Mass.

Goodall, J. L., Robinson, B. F. & Castronova, A. M. 2011 Modeling water resource systems using a service-oriented computing paradigm. *Environmental Modelling and Software* **26**, 573–582.

Gourbesville, P. 2009 Data and HydroInformatics: New possibilities and challenges. *Journal of Hydroinformatics* **11** (3–4), 330–343.

Gregersen, J. B., Gijsbers, P. J. A., Westen, S. J. P. & Blind, M. 2005 OpenMI: the essential concepts and their implications for legacy. *Software Advances in Geosciences* **4**, 37–44.

Hewett, C. J. M., Doyle, A. & Quinn, P. F. 2010 Towards a HydroInformatics framework to aid decision-making for catchment management. *Journal of Hydroinformatics* **12** (2), 119–139.

Hill, C., DeLuca, C., Balaji, V., Suarez, M. & Silva, A. D. 2004 The architecture of the earth system modeling framework. *Computing in Science and Engineering* **6**, 18–28.

Hillyer, C., Bolte, J. & Lamaker, A. 2003 The ModCom modular simulation system. *European Journal of Agronomy* **18**, 333–343.

Holzworth, D. P., Huth, N. I. & de Voil, P. G. 2010 Simplifying environmental model reuse. *Environmental Modelling & Software* **25**, 269–275.

Horak, J., Orlik, A. & Stromsky, J. 2008 Web services for distributed and interoperable hydro-information systems. *Hydrology and Earth System Sciences* **12**, 635–644.

Howie, C. T., Kunz, J. C. & Law, K. H. 1996 *Software Interoperability*. Kaman Sciences Corporation, Bloomfield, CT, USA.

Hummel, J. R. & Christiansen, J. H. 2002 The dynamic information architecture system: a simulation framework to provide interoperability for process models. In *Proceedings of Simulation Interoperability Workshop*, Orlando, FL, 8–13 September.

Killeen, T., DeLuca, C., Toth, G., Gombosi, T., Stout, Q., Goodrich, C., Sussman, A. & Hesse, M. 2006 Integrated frameworks for earth and space weather simulation. In *American Meteorological Society Meeting*, Atlanta, GA, 29 January–2 February.

Knapen, J. J. R., Verweij, P., Wien, J. E. & Hummer, S. 2009 OpenMI: The universal glue for integrated modeling? In *18th World IMACS Congress and MODSIM09 International Congress in Modeling and Simulation*, Cairns, Australia, 13–17 July, 895–901.

Kralisch, S., Krause, P. & David, O. 2005 Using the object modeling system for hydrologic model development and application. *Advances in Geosciences* **4**, 75–81.

Kuo, Y. H., Klemp, J. B. & Michalakes, J. 2004 Mesoscale numerical weather prediction with the WRF Model. In *Symposium on the 50th Anniversary of Operational Numerical Weather Prediction*, University of Maryland, pp. 14–17.

Lakhankar, T., Fekete, B. M. & Vörösmarty, C. J. 2008 Semantic web infrastructure supporting NextFrAMES modeling platform. In *American Geophysical Union*, Fall Meeting, San Francisco, CA, 15–19 December, abstract #IN11B-1035.

Leavesley, G. H., Markstrom, S. L., Brewer, M. S. & Viger, R. J. 1996a The modular modeling system (MMS): the physical process modeling component of a database-centred decision support system for water and power management. *Water, Air and Soil Pollution* **90**, 303–311.

Leavesley, G. H., Restrepo, P. J., Markstrom, S. L., Dixon, M. & Stannard, L. G. 1996b The Modular Modelling System (MMS): User's Manual. Report 96-151, US Geological Survey, pp. 175.

Leavesley, G. H., Markstrom, S. L., Viger, R. J. & Hay, L. E. 2005 The Modular Modeling System (MMS): a toolbox for water- and environmental-sources management. US Geological Survey and International G-WADI Modelling Workshop, February–March 2005. National Institute of Hydrology, Roorkee, India.

Marke, T., Mauser, W., Pfeiffer, A. & Zangl, G. 2011 A pragmatic approach for the downscaling and bias correction of regional climate simulations-evaluation in hydrological modeling. *Geoscience Model Development Discussions* **4**, 45–63.

Maxwell, T. 1999 A paris-model approach to modular simulation. *Environmental Modelling and Software* **14** (6), 511–517.

Maxwell, T. & Costanza, R. 1994 *Spatial Ecosystem Modeling in a Distributed Computational Environment. Towards Sustainable Development: Concepts, Methods, and Policy*. Island Press, Washington, DC, pp. 111–138.

Maxwell, T. & Costanza, R. 1995 Distributed modular spatial ecosystem modelling. *International Journal of Computer Simulation* **5** (3), 247–262.

Maxwell, T. & Costanza, R. 1996 Facilitating high performance, collaborative spatial modeling. In *3rd International Conference on Integrating Geographic Information Systems and Environmental Modelling*. Santa Barbara, CA, 21–25 January.

Meyer, B. 2001 .Net is coming. *Computer* **34** (8), 92–97.

Michalakes, J., Dudhia, J., Gill, D., Klemp, J. & Skamarock, W. 1998 *Design of a Next-generation Regional Weather Research and Forecast Model. Towards Teracomputing*. World Scientific, River Edge, New Jersey, pp. 117–124.

Michalakes, J., Chen, S., Dudhia, J., Hart, L., Klemp, J., Middlecoff, J. & Skamarock, W. 2001 Development of a next generation regional weather research and forecast model. In *Proceedings of the 9th ECMWF Workshop on the Use of High Performance Computing in Meteorology*, Singapore, 25–29 October, pp. 269–276.

Moore, R. V. & Tindall, C. I. 2005 An overview of the open modeling interface and environment (the OpenMI). *Environmental Science and Policy* **8**, 279–286.

Padulo, L. & Arbib, M. A. 1974 *Systems Theory: a Unified State-Space Approach to Continuous and Discrete Systems*. W.B. Saunders, Philadelphia, PA.

Peckham, S. 2008 CSDMS handbook of concepts and protocols: A guide for code contributors. Available from: http://csdms.colorado.edu/wiki/Tools_CSDMS_Handbook.

Rahman, J. M., Seaton, S. P., Perraud, J. M., Hotham, H., Verrelli, D. I. & Coleman, J. R. 2003 It's TIME for a new environmental modelling framework. In *Proceedings of International Congress on Modelling and Simulation (MODSIM03)*, Townsville, Australia, 14–17 July, pp. 1727–1732.

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

Rahman, J. M., Cuddy, S. M. & Watson, F. G. R. 2004b Tarsier and ICMS: two approaches to framework development. *Mathematics and Computers in Simulation* **64** (3–4), *February 2004*, 339–350.

Rassam, D. W., Pickett, T. & Knight, J. H. 2009 Incorporating floodplain groundwater interactions in river modeling. In *18th World IMACS/MODSIM Congress*, Cairns, Australia, 13–17 July, pp. 3116–3122.

Reed, M., Cuddy, S. M. & Rizzoli, A. E. 1999 A framework for modeling multiple resource management issues – an open modeling approach. *Environmental Modelling and Software* **14**, 503–509.

Rizzoli, A. E., Davis, J. R. & Abel, D. J. 1998 Model and data integration and re-use in environmental decision support system. *Decision Support Systems* **24** (2), 127–144.

Singh, V. P. & Woolhiser, D. A. 2002 Mathematical modeling of watershed hydrology. *Journal of Hydrologic Engineering* **7**, 270–292.

Sydelko, P. J., Hlohowskyj, I., Majerus, K., Christiansen, J. & Dolph, J. 2001 An object-oriented framework for dynamic ecosystem modeling: application for integrated risk assessment. *The Science of the Total Environment* **274**, 271–281.

USGS 2009 *Status of ModFlow versions and ModFlow-related programs available on USGS Web Pages*. Office of Groundwater, US Geological Survey. Available from: http://water.usgs.gov/nrp/gwsoftware/modflow-status.pdf.

Valcke, S., Declat, D., Redler, R., Ritzdorf, H., Schoenemeyer, T. & Vogelsang, R. 2004 The PRISM coupling and I/O system. In *Proceedings of the 6th International Meeting: High Performance Computing for Computational Science*, Valencia, Spain, 28–30 June, pp. 845–850.

Valcke, S., Guilyardi, E. & Larsson, C. 2006 PRISM and ENES: A European approach to earth system modelling. *Concurrency and Computation: Practice and Experience* **18** (2), 231–245.

Voinov, A., Zaslavskiy, I., Arctur, D., Duffy, C. & Seppelt, R. 2008 Community modeling, and data-model interoperability. In *Proceedings of 4th Biennial Meeting of International Environmental and Software Society*, Barcelona, Catalonia, 7–10 July, pp. 2035–2047.

Wagener, T., Reed, P., van Werkhoven, K., Tang, Y. & Zhang, Z. 2009 Advances in the identification and evaluation of complex environmental systems models. *Journal of Hydroinformatics* **11** (3–4), 266–281.

Watson, F. G. R. & Rahman, J. M. 2004 Tarsier: a practical software framework for model development, testing and deployment. *Environmental Modelling and Software* **19** (3), 245–260.

Watson, F. G. R., Rahman, J. & Seaton, S. 2001 Deploying environmental software using the Tarsier modelling framework. In *Proceedings of 3rd Australian Stream Management Conference*, Brisbane, Australia, 27–29 August, pp. 631–637.

First received 13 May 2011; accepted in revised form 29 March 2012. Available online 29 May 2012