Environmental Modelling & Software 39 (2013) 201–213

Contents lists available at SciVerse ScienceDirect

Environmental Modelling & Software

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

A software engineering perspective on environmental modeling framework design: The Object Modeling System☆

O. David [a,b,*], J.C. Ascough II [c], W. Lloyd [a,b], T.R. Green [c], K.W. Rojas [d], G.H. Leavesley [a], L.R. Ahuja [c]

[a] Dept. of Civil and Environmental Engineering, Colorado State University, Fort Collins, CO 80523, USA
[b] Dept. of Computer Science, Colorado State University, Fort Collins, CO 80523, USA
[c] USDA-ARS-NPA, Agricultural Systems Research Unit, 2150 Centre Ave., Bldg. D, Fort Collins, CO 80526, USA
[d] USDA-NRCS, 2150 Centre Ave., Bldg. A, Fort Collins, CO 80526, USA

ARTICLE INFO

Article history:
Received 12 September 2011
Received in revised form
24 February 2012
Accepted 9 March 2012
Available online 14 June 2012

Keywords:
Object Modeling System
Environmental modeling frameworks
Modeling and simulation
Software engineering
Software design

ABSTRACT

The environmental modeling community has historically been concerned with the proliferation of models and the effort associated with collective model development tasks (e.g., code generation, data transformation, etc.). Environmental modeling frameworks (EMFs) have been developed to address this problem, but much work remains before EMFs are adopted as mainstream modeling tools. Environmental model development requires both scientific understanding of environmental phenomena and software developer proficiency. EMFs support the modeling process through streamlining model code development, allowing seamless access to data, and supporting data analysis and visualization. EMFs also support aggregation of model components into functional units, component interaction and communication, temporal-spatial stepping, scaling of spatial data, multi-threading/multi-processor support, and cross-language interoperability. Some EMFs additionally focus on high-performance computing and are tailored for particular modeling domains such as ecosystem, socio-economic, or climate change research. The Object Modeling System Version 3 (OMS3) EMF employs new advances in software framework design to better support the environmental model development process. This paper discusses key EMF design goals/constraints and addresses software engineering aspects that have made OMS3 framework development efficacious and its application practical, as demonstrated by leveraging software engineering efforts outside of the modeling community and lessons learned from over a decade of EMF development. Software engineering approaches employed in OMS3 are highlighted including a non-invasive lightweight framework design supporting component-based model development, use of implicit parallelism in system design, use of domain specific language design patterns, and cloud-based support for computational scalability. The key advancements in EMF design presented herein may be applicable and beneficial for other EMF developers seeking to better support environmental model development through improved framework design.

© 2012 Elsevier Ltd. All rights reserved.

Software availability

Name of software: Object Modeling System Version 3 (OMS3)
Description: OMS3 is an environmental modeling framework that utilizes new advances in framework software engineering approaches including a non-invasive lightweight framework design and the use of domain specific languages.
Developer: Olaf David
Source language: Java

Contact address: Dr. Olaf David, USDA-ARS, ASRU, 2150 Centre Ave., Bldg. D, Suite 200, Fort Collins, CO 80526 USA.
E-mail: odavid@colostate.edu

1. Introduction

Designing domain specific software frameworks is challenging; however, frameworks and libraries are essential to enable better software engineering processes in support of scientific modeling. There are many examples of framework use in software engineering including: 1) a web developer uses a content management system to develop a data warehouse application for enabling communication with a remote data store; 2) a user interface developer selects a widget framework to create a rich cross

☆ Thematic Issue on the Future of Integrated Modeling Science and Technology.
* Corresponding author. USDA-ARS, ASRU, 2150 Centre Ave., Bldg. D, Suite 200, Fort Collins, CO 80526, USA. Tel.: +1 970 492 7316; fax: +1 970 492 7310.
E-mail address: odavid@colostate.edu (O. David).

1364-8152/$ – see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.envsoft.2012.03.006

platform desktop application for image processing; 3) a Geographical Information System (GIS) specialist uses a geo-processing library to process geometric data layers for optimal solar panel placement; 4) a biologist uses a statistical framework and library tools to test the correlation between gene expression and phenotypes; and 5) a climate researcher accesses array-oriented scientific data (e.g., precipitation and temperature) which is distributed to multiple computers using the Message Passing Interface library. Software developers use different frameworks and libraries for many reasons often revolving around trust (the software application has already been developed and appears to work correctly), complexity (to avoid dealing with complex and multifarious details), and efficiency (the work of others is easier to build upon). Effective software frameworks improve not only developer productivity but also the quality and reliability of the software product itself, and allow developers to focus development efforts on supporting unique application requirements (as opposed to developing application infrastructure).

Frameworks and libraries abstract common software functionality to provide application developers with a means to achieve reuse. An Application Programming Interface (API) is an abstraction layer that provides access to the underlying functionality of software frameworks. Frameworks and libraries always provide one or more APIs; however, frameworks and libraries are not used in the same fashion. Many frameworks adhere to the notion of the "Inversion of Control" design pattern where the framework directs program execution flow. When using a framework that supports Inversion of Control, application classes extend framework-specific data types which are invoked dynamically at runtime by the framework. In contrast, use of software libraries entails instantiation of library data types and invocation of library functions by application code. Developers must have familiarity with framework-specific classes, types, and methods to harness the functionality provided.

Designing an environmental modeling framework (EMF) requires consideration of a broad spectrum of modeling approaches including discrete event simulation, agent-based modeling, and the use of genetic algorithms. Current EMFs evolved using simulation models originating from hydrology, biology, climatology, and economic scientific domains (Argent, 2004). They typically reflect domain origins in their design and allow abstract modeling within the various scientific disciplines. What else differentiates EMFs? Environmental modelers are typically not software engineers. Scientific understanding requires detailed knowledge which contradicts full reliance on black-box programming interfaces that can hide information related to input/output management, code relationships, etc. Additionally, scientific knowledge is needed to develop or choose an adequate set of equations, algorithms, and mathematical constructs to create environmental models. Unique social and cultural characteristics of the environmental model development process include:

- Environmental modelers desire to understand scientific details. Traditional framework designs often do not sufficiently describe all the features of a science component in that: 1) emphasis is placed on algorithmic relationships where data object implementations reduced to traditional typing systems may be too simplistic; and 2) relevant meta-information describing data objects is often part of the external documentation and not an integral part of the model.
- Environmental model development needs to be creative, i.e., new approaches have to be explored that go beyond the boundaries of given programming languages, data structures, algorithms, and existing architectures. EMFs should foster creativity and not constrain the modeler to the framework developer's view.
- Environmental models evolve within their own "engineering ecosystem." Many current environmental modeling efforts involve tens of thousands of lines of scientific code that are resistant to refactoring and efforts to improve code design because of resource, cultural, and reward constraints. High quality model code is often a secondary priority since scientific reward is frequently based on the accuracy of model output, rather than long-term model code maintainability and reusability.

Individuals and organizations increasingly realize the opportunities of collaborative and open modeling efforts, taking advantage of current state-of-the-art software development practices employed by open source projects, industry, etc. Environmental model development can be expensive and time consuming, e.g., various natural resource models produced by the U.S. Department of Agriculture-Agricultural Research Service (USDA-ARS) such as the Soil and Water Assessment Tool (SWAT, Arnold et al., 1993), Water Erosion Prediction Project (WEPP, Flanagan and Nearing, 1995), and the Root Zone Water Quality Model (RZWQM, Ahuja et al., 2000) have cost roughly US $15-20M each to design, implement, evaluate, and deploy. EMFs have emerged as a valuable tool to streamline diverse modeling projects, allowing modelers to more easily develop and integrate models while simultaneously taking advantage of advances in information processing and data management. The major objectives of this paper are to present key EMF design goals/constraints and describe the development and application of the Object Modeling System Version 3 (OMS3) including the rationale behind the nontraditional OMS3 framework design approach. OMS3 represents a significant departure from previous versions of the framework and also from traditional framework design approaches for environmental modeling. The "non-invasive" design of OMS3, inspired by web application frameworks, focuses on: 1) supporting graceful adaptation of source code from existing environmental models; 2) elevating the importance of metadata; and 3) avoidance of imposing complex programming interfaces on a modeler. OMS3 design features for utilizing components as model building blocks (including ways to aggregate components into environmental models) are discussed, followed by a description of how OMS3 arranges complex scenarios in a concise but scalable approach using domain specific language (DSL) principles.

2. EMF design goals and constraints

Driving forces for framework adoption within the environmental modeling community include, but are not limited to: saving time and reducing costs, providing quality assurance and quality control, re-purposing model solutions for reuse in new models, ensuring consistency and traceability of model results, and supporting computational scalability to enable complex modeling. The bottom line is that an EMF should help the developer more efficiently implement and deliver environmental simulation models. Rizzoli et al. (2008) conclude that a return on investment should be realized through adopting an EMF.

Designing EMFs has traditionally been fraught with challenges since a broad variety of modeling approaches, conceptualizations, and abstractions exist. Addressing a certain modeling domain helps framework developers meet modeler expectations. Therefore, framework development requires accommodating software engineering skill sets of framework users, anticipating target architectures, supporting high performance computing needs, and handling constraints of the programming languages and

software architectures common to the specific modeling domain. These factors support freedom of choice for a modeler through an EMF. Existing institutional and cultural settings that may foster or limit the acceptance of EMFs should also be considered. Furthermore, EMF designs should aim to support model development while minimizing the divergence from existing modeling practices within a research organization. Fostering organizational awareness of software engineering best practices may prove advantageous before moving to a framework-based modeling approach. Adopting software coding standards, using version control systems to manage source code, and performing code peer reviews can be important steps towards improving model code development that will eventually help expedite adoption of modeling frameworks. EMFs should promote the creative talent of modelers to construct new models and modeling approaches while eliminating tedious, time-consuming aspects of model development, commonly understood as the software engineering aspects.

An EMF should allow the modeler to retain intellectual ownership of models and associated source code. The model source code should not be "owned" by the framework, i.e., it must exist and be sustainable outside the framework to ensure independent and ongoing development. This requirement differs significantly from other solutions such as web-based applications, graphical user interfaces, and high performance computing applications. EMFs should support interaction with existing legacy models. General-purpose programming languages (e.g., Java, C, FORTRAN) that have a wide infiltration with active developer communities which provide free compilers, tools, and development environments should be given preference over specialized modeling and simulation languages. Adoption of widely available and supported general purpose languages for model development helps ensure longevity of model implementations. Using free, open source programming languages (e.g., the GNU Compiler Collection) supports collaboration on environmental model development since they are more widely available at a lower cost than proprietary languages. EMFs should minimize source code that is required by infrastructure constraints, underlying platforms, operating systems, or programming language specifications. Additionally, an EMF should promote semantic density of modeling solutions, i.e., coding is expected to be straightforward and concise for the modeler even if additional development effort is required on the part of the framework developer. If the EMF streamlines difficult and tedious tasks (e.g., database queries, parallelization of algorithms, time/date management, graphing/visualization algorithms), it is more likely to be adopted as this lowers the programming burden for the modeler. Programming concept redundancy should be avoided, i.e., if a programming language provides data types and functions then custom equivalents should not be reintroduced via the framework. For example, operator overloading in C++ already provides a language extension path, therefore developing an extra programming interface for "specific" data creation or conversion is unnecessary. The likelihood of adopting an EMF for environmental modeling may be significantly increased if common, native programming language concepts are implemented by the framework.

EMF design goals are multi-faceted and often include many aspects of general software framework development. An EMF needs to foster development productivity, interoperability with other modeling tools, protocols, and programming languages, thereby minimizing the integration effort. The EMF should be semantically powerful, intuitive to use, and not impose a steep learning curve. In addition, the EMF should enable flexible model development rather than enforce complex and rigid development concepts. Most functional features offered by current EMFs are closely related to interoperability aspects. Facilitating data exchange between software units representing parts of a whole model is a key feature. These units called modules, classes, or components represent core building blocks supporting the principles of modular and component-based software construction. Some modeling frameworks provide class libraries, a library programming interface, or have a set of classes and recommended object-oriented design patterns encompassing connectivity. Robust EMFs should not only enable module connectivity but also manage data exchange while considering the type and physical scale of geospatial objects, support conversion based on physical units, and account for the location of data in distributed computing environments.

In summary, even though EMFs are yet another technical part of enabling technology for modeling, their design has to consider many organizational, intellectual, and social specifics of the scientific community. Furthermore, their adoption for developing environmental models is an imprecise process that is often driven by many internal and external factors, some of which may not be related to the particular technical strengths and weaknesses of the framework.

3. Object Modeling System (OMS)

The Object Modeling System (OMS) is a framework for environmental model development that provides a consistent and efficient way to: 1) create science simulation components; 2) develop, parameterize, and evaluate environmental models and modify/adjust them as science advances; and 3) re-purpose environmental models for emerging customer requirements. OMS is an open source software project (i.e., all framework code is freely available) enabling members of the scientific community to collaborate to address complex issues associated with the design, development, and application of environmental models. The OMS architecture has been designed so that it is inter-operable with other frameworks supporting environmental modeling globally. OMS is currently being used within the United States for water supply forecasting (Leavesley et al., 2010) and agro-ecosystem modeling projects (Ascough et al., 2012), and in Northern and Central Africa for groundwater modeling using isotope traces.

3.1. Development history

The OMS development effort started in the early 2000's as a vehicle to migrate the design principles of the Modular Modeling System (MMS) (Leavesley et al., 2006), written in C/Motif, into a reusable environmental modeling framework. MMS originated from the Precipitation-Runoff Modeling System (PRMS) hydrologic model (Leavesley et al., 1983), a distributed watershed model. The principles of process-based assembly for PRMS FORTRAN modules using a C interface were re-created as Java classes under OMS Version 1 (OMS1, Ahuja et al., 2005) based on core MMS technologies. PRMS was chosen since it provided the foundation for MMS and was already disaggregated into components (modules) representing physical processes in hydrology. OMS1 was a procedural system with framework support for daily model time stepping which focused primarily on data exchange and management. A simple user interface allowed the definition of parameter sets represented in data dictionaries. OMS Version 2 (OMS2), initially released in 2004, delivered major improvements and was integrated into the USDA-Natural Resources Conservation Service (USDA-NRCS) information technology architecture in 2008. OMS2 harnessed the Netbeans™ rich client platform application to deliver an integrated modeling development environment offering project management, data visualization,

parameter editing, and a visual model builder. As development continued, both the OMS2 and Netbeans™ API improved but also became large and complex. Emerging requirements such as web-service deployment and the need to support model creation in other integrated development environments resulted in redesign of OMS2 in favor of the more flexible and "lightweight" OMS3 framework design. OMS3 (David et al., 2010) provides a new API which leverages existing concepts of component-based modeling. The new OMS3 architecture was driven by the need for: 1) elegance and simplicity in component design; 2) performance and scalability to support implicit multi-threading for component execution; 3) faster adaptation of legacy code; and 4) flexible tool extensions to implicitly integrate model calibration and sensitivity/uncertainty analysis methods, resulting in the utilization of DSLs for simulation setup and execution.

As shown in Fig. 1, OMS3 is comprised of four primary architectural foundations including modeling resources, the system knowledge base, development tools, and modeling products. The OMS3 core consists of an internal metadata knowledge base for model and simulation creation. A simulation in OMS is defined as a collection of resources (parameter sets, input data, modeling components, model execution method, etc.) required to produce desired modeling outputs. The system supports harnessing metadata from various sources including natural resources databases (e.g., land use/cover, soil), web-based data provisioning services, version control systems, and/or other code repositories, which is incorporated into the framework knowledge base that various OMS3 development tools employ to create modeling products. OMS3 modeling products include science components and complete models, simulations supporting parameter estimation and sensitivity/uncertainty analysis, output analysis (e.g., statistical evaluation and graphical visualization) tools, modeling audit trails (i.e., reproducing model results for legal purposes), and miscellaneous technical/user documentation. As with any EMF, fully embracing the OMS3 architecture requires a commitment to a structured model development process which may include the use of a version control system for model source code management or databases to store audit trails. Such features are important for institutionalized adoption of OMS3 but less critical for adherence by a single modeler.

3.2. Framework invasiveness and OMS3

The degree of dependency between a framework and simulation model code can be described as "framework invasiveness", defined by Lloyd et al. (2011a) as the degree to which model code is coupled to the underlying framework. Framework to modeling code invasiveness occurs due to several factors, including the use of a framework API consisting of data types and methods/functions which developers use to harness framework functionality, the use of framework-specific data structures (e.g., classes, types, and constants), or the implementation of framework interfaces and extension of framework classes. Framework to application invasiveness is a type of code coupling; object-oriented coupling (i.e., coupling between classes in an object-oriented program) has been shown to correlate inversely with the likelihood of a mistake in the code (Briand et al., 2000). Mistakes in model code negatively impact the functional correctness of the code, thereby reducing the functional aspects of code quality. Other important aspects of model code quality include "non-functional" quality attributes such as maintainability, portability/reusability, and understandability. Lloyd et al. (2011a) demonstrated that code invasiveness incurred by using a modeling framework is correlated to non-functional code quality metrics. The impact of this invasiveness can be considered as the degree of dependency imposed by the modeling framework for a specific modeling problem.

Why is a non-invasive framework approach important for OMS3? Most environmental modelers are natural resource scientists frequently with only self-taught experience in programming and little or no proficiency with software architecture and design. Most environmental modeling development projects do not have the luxury of employing experienced software engineers or computer scientists who are able to understand and apply complex design patterns, UML diagrams, and advanced object-oriented techniques such as parameterized types, higher level data structures and/or object composition. The use of object-oriented design principles for modeling can be productive for a specific modeling project that has limited need for external reuse and extensibility. Extensive use of object-oriented design principles can be difficult for scientists to adopt in that adoption often entails a steep learning

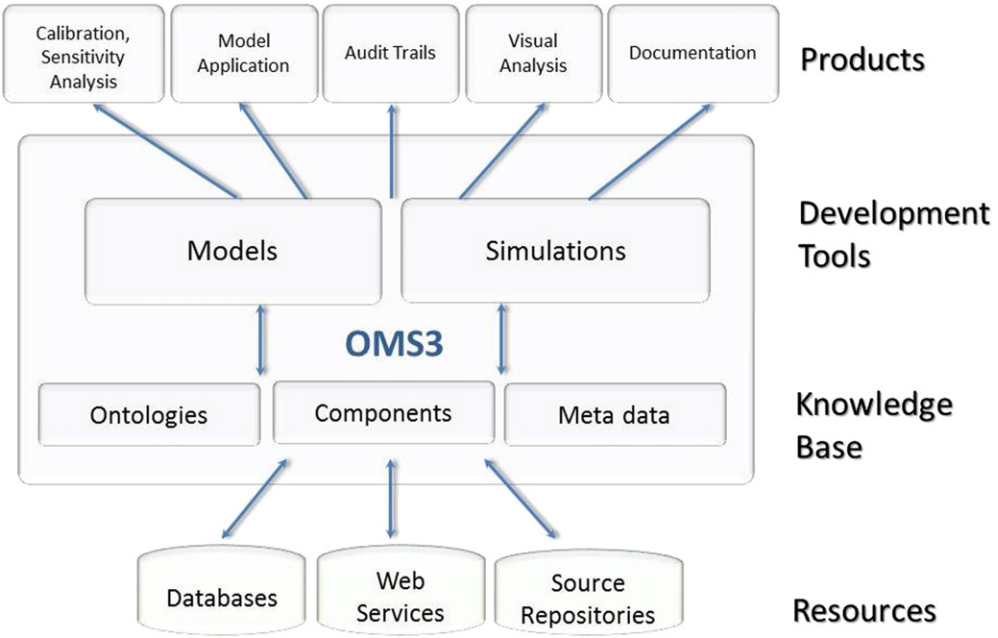

Fig. 1. OMS3 principle framework architecture.

curve requiring management of complex programming concepts (e.g., polymorphic execution flow in an inheritance hierarchy). Object-oriented design principles have been promoted for nearly two decades as a promising technology for supporting environmental modeling. As experience has shown, well-designed object-oriented models are difficult and time consuming to develop, particularly with a design objective to support reuse, maintainability, and understandability. Design of complex systems requires experience, anticipation of future use cases by providing extension points, freedom to refactor poor aspects of a design, and adequate time and resources. Environmental modeling perspectives, concepts, and approaches vary and are not easy to capture in a single object-oriented design. The environmental modeling community maintains many legacy models still in use based on algorithms and equations developed decades ago. What has changed and continues to change are the hardware and software infrastructures that house and deliver the output from environmental models. Devices such as smart phones and cloud computing (described later) are emerging technologies which non-invasive EMFs can readily support.

EMFs can also be classified as heavyweight or lightweight based on various design characteristics (Lloyd et al., 2011a). The primary difference between these framework types is how they present functionality to the developer. Heavyweight frameworks, e.g., traditional object-oriented frameworks such as Java's Swing Application Framework for graphical user interface development, provide developers with an API that is can be large and unwieldy requiring developers to spend considerable time becoming familiar with before writing model code. The lightweight framework design approach adopted by OMS3 originated from various web application and enterprise frameworks (Richardson, 2006). In contrast to heavyweight frameworks, lightweight frameworks offer functionality to the developer using a variety of techniques aimed at reducing the API's overall size and developer dependence on the API. A lightweight EMF adapts to an existing model's source code, resulting in a less steep learning curve as there is no complex API to understand or special data types to manage. This provides several practical implications for environmental modelers as there is no major paradigm shift since existing modeling code and libraries are being used. Adopting and using a lightweight framework is easier since modeling components used by the lightweight framework can still function and continue to evolve outside the framework.

Enabled by the lightweight and non-invasive characteristics of the OMS3 modeling framework, creating a modeling object becomes a rudimentary task as there are no interfaces to implement, no classes to extend, no polymorphic methods to override, and no specialized framework-specific data types to use. OMS3 uses metadata by means of language annotations to specify and describe "points of interest" amongst existing data fields and class methods of the model. To verify the improvement in code quality of using annotation-based components and models versus traditional API approaches, Lloyd et al. (2011a) conducted a study comparing the implementation of several component-based hydrology models within different languages and EMFs. They applied several code quality metrics to quantify code characteristics of the different model implementations and found that the non-invasive framework approach of OMS3 enabled more concise model implementations, in terms of number of lines of code and lower code complexity, for environmental model development. For example, the OMS3 implementation of the Thornthwaite model required only 295 lines of code whereas other EMFs required between 450 and 1635 lines of code. All of the model implementations had identical functionality and produced the same modeling results. Furthermore, implementation of the PRMS model in OMS3 required only 10,163 lines of code compared with 16,997 for OMS2 (Lloyd et al., 2011a). Software engineering research suggests that reduction in code size typically results in lower model development and maintenance costs over the lifetime of a model (Briand et al., 2000). Outside of the environmental modeling domain, similar success stories have been observed which are currently driving the popularity of lightweight web services frameworks such as JBoss Seam and Spring (Yuan et al., 2009).

3.3. OMS3 component-based modeling concepts

Like other modeling frameworks such as OpenMI (Blind and Gregersen, 2005; Gregersen et al., 2007), Common Component Architecture (CCA) (Bernholdt et al., 2003), Earth System Modeling Framework (ESMF) (Collins et al., 2005), and Common Modeling Protocol (CMP) (Moore et al., 2007), OMS3 uses classes as the fundamental model building block while embracing principles of component-based software engineering for the model development process. The advantages of constructing modular software are well known in the software engineering field. Individual modules can be developed using standardized interfaces supporting module communication. Partitioning a system into modules typically helps to minimize coupling, which should lead to code that is easier to maintain.

The term component refers to self-contained, separated software units that implement independent functions in context-independent manner. In this paper, we refer to components as modeling entities which implement a single conceptual modeling concept. A component can be hierarchical in that it may orchestrate interaction among other finer-grained components through the use of different categories of annotations in OMS3. Components expose framework-relevant aspects via metadata, and each component should provide a sufficient level of complexity within a model's component hierarchy. Fig. 2 shows the principle layout of components as supported in OMS3 including managing data flow/execution phases and building component hierarchies. Like many EMFs, OMS3 provides core features including functional encapsulation supporting isolation of individual computational aspects into components, facilitation of directed data flow (input/output slots or exchange items), and management of various execution states within components including "Initialize/Run/Finalize" as described by Peckham (2008).

While object-oriented methods focus on abstraction, encapsulation, and localization of data and methods, their use can also lead to simulation systems where objects are highly co-dependent. To

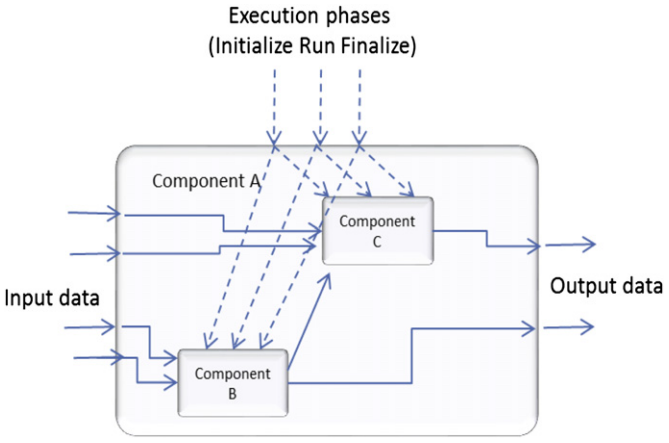

Fig. 2. OMS3 component architecture including data flow, execution phases, and encapsulation.

remove this limitation, a more streamlined model development process is supported by OMS3 that emphasizes the component as an object-oriented software unit which can be developed and tested independently, and delivered for use as a service. Component implementations have been shown to support reuse more efficiently than simply relying on object-oriented language features or methods (Schmidt, 1999). The following are salient benefits of using components for building complex systems:

- Components are designed with a standard, well-defined interface in mind. A published interface hides the implementation of the component logic and forces an abstraction level which separates provided functions from implementation.
- Components are self-contained units. They can be developed and tested individually, and can be packaged and delivered to be used in multiple applications.
- The use of components simplifies the construction of building complex models since their use fundamentally changes the way systems are built. As opposed to programming an entire model in a "whole-block" monolithic fashion, models can be composed of "building-block" components that originate from both new code and legacy code.

Environmental modeling applications have traditionally been designed as large blocks of hand-crafted code which in turn result in monolithic models. These models were not designed to be easily re-purposed if a related application is required in the future. A major disadvantage of building monolithic simulation models is that conceptual boundaries within the model are not supported as there is no separation between concepts in the code. A component-based modeling approach helps address the challenge of building complex simulation models by reducing model software complexity and overcoming the limitations of monolithic, highly coupled model implementations. Components were the main simulation model building blocks in OMS1 which offered library classes that were over-ridden as subclasses or instantiated directly (Table 1a). OMS1 defined a limited set of data types that could be exchanged between model components. This was an acceptable approach but modelers were limited in their ability to use and share the data types. For legacy code integration, as well as for future model development, this constraint is undesirable. Functionality to support unit, type, and data transformation resulted in large amounts of complex framework code. Model components developed in OMS1 had to be subclasses of framework classes making blending classes from different models impossible since a delegator/wrapper for existing components was always needed.

Table 1
Comparison of declarations between OMS framework versions.

(a) OMS1	/** Elevation. @unit ft @access read **/ public OMSDouble elevation;

(b) OMS2	private Attribute.Double elevation;
	…
	/** Elevation.
	@unit ft
	@access read
	**/
	public void setElevation (Attribute.Double e) {
	elevation = e;
	}

(c) OMS3	@Description ("Elevation") @Unit (ft) @In public double elevation;

OMS2 simplified component design by allowing models/components to use interfaces instead of extending classes, i.e., no framework interface implementation was accessible from within the modeling component (Table 1b). This "design-by-contract" implementation allowed OMS2 to offer variants of framework data types with extensible implementations for unit conversion, remote data access, data transformations, etc. while being fully transparent to the component. Switching to interfaces improved the overall quality and robustness of the framework and resulted in a reduced API size. However, data types supported by OMS2 were still framework dependent. For OMS Versions 1 and 2, framework data types for component data transfer objects were included in addition to the existing Java language counterparts. This represented type redundancy for *double* and *Double* by providing an *OMSDouble* class implementation (OMS1) or an *Attribute.Double* (OMS2).

OMS3 provides a simplified approach for environmental model component design. It enables Plain Old Java Objects (POJOs) that are annotated to be used within the framework (Table 1c). Annotated POJOs are easy to create since they are basic Java classes enriched with language level annotations. Annotations provide the framework with metadata to interpret the component as a model building block. Technologies that enable this framework design include:

- Runtime *introspection* on class structures, exploring fields, methods, and their values – enabling exploration of component internals to find framework entry points for data flow and execution.
- Language level *annotations* on classes, methods, and fields – for describing data flow fields and tagging Initialize/Run/Finalize methods.
- *Reflective access* to object fields and *reflective method invocation* – for component-to-component data transfer and indirect method execution.

Any language or platform that supports the above features, such as Java or C#, should be sufficient to implement this type of architecture. OMS3 is a major advancement from Versions 1 and 2 in that it departs from the traditional API-based framework approach for component design in favor of a more lightweight, non-invasive implementation (Lloyd et al., 2011a). The following section introduces the concept of annotations as implemented in OMS3.

3.4. OMS3 annotations

Programming language annotations define and capture metadata to describe elements such as classes, fields, or methods. In OMS3, framework functionality is integrated as annotations to facilitate component connectivity and provide for data flow. Annotations are non-invasive to the simulation model code and represent a native construct of the Java platform and language. They introduce a language extension mechanism that is not available in traditional languages such as C or C++. In C# annotations are called attributes; the Groovy programming language also supports annotations similar to Java. Non-invasive lightweight framework principles based on plain objects have proven successful in other application domains (e.g., Richardson, 2006) and can be used to help improve environmental modeling. There are three categories of annotations in OMS3 (Table 2):

Table 2
Example declaration of the Hydrologic Response Unit (HRU) area parameter in OMS3.

```
@Description("HRU area, Area of each HRU")
@Role("PARAMETER")
@Unit("hectare")
@Bound ("nhru")
@In public double[] hru_area;
```

1. *Mandatory Execution Annotations* provide essential information for component execution. They describe method invocation points and data flow between components. Examples are @In or @Out for describing input and output "ports."
2. *Supporting Execution Annotations* support component execution by providing additional information about data flow, physical units (@Unit), and range constraints (@Range) that might be used during execution.
3. *Documentation Annotations* are used to create documentation, and map to databases/archives and other content management systems and tools. Examples are @Description, @Author, @VersionInfo, etc.

In OMS3, all parameter and variable declarations within a component are supplemented through the use of annotations. For example declaration of the parameter "hru_area", the physical area of a Hydrologic Response Unit (HRU), includes information about its role as a parameter, description or definition, unit of measure, the bounding dimension of its array size, and use as a double precision input array (Table 2). Listing 1 shows a simple component for lookup table computation in the OMS/J2K watershed model (Ascough et al., 2012). All annotations start with an (@) symbol and have the following features in OMS3:

- OMS3 package dependencies only exist for annotations (oms3.annotations.*). No API calls are required, a characteristic which supports the Inversion of Control design pattern.
- Annotations enable automatic generation of component documentation. Individual fields are annotated to describe their specific purpose. Data flow annotations are used to generate documentation.
- A component adheres to the Initialize/Run/Finalize cycle by tagging methods with corresponding annotations. The compute method of a component is "tagged" with the @Execute annotation making the name of the method insignificant.
- Data flow indications (i.e., component linkages) are provided using the @In and @Out annotations. No explicit marshaling or un-marshaling of component variables is needed, i.e., an assignment is sufficient to pass them on to the receiving component.

With respect to data flow, it should be noted that OMS3 explicitly manages the data transfer protocol between connected components. It has no knowledge about the semantics or structure of transferred data, which could be a multi-dimensional array, a scalar parameter, a time representing calendar object, a database connection, etc. Any data type can be exchanged across components, a feature provided by Java's "reflection mechanism." This results in conceptually simple and clean transfer of data while providing maximum flexibility for the modeler. In cases where transformation of data during an exchange is desired because of otherwise incompatible data types, physical units, scales, or resolution, a service provider interface (SPI) exists to allow altering the exchanged data objects.

In summary, annotations provide an integrated and contextually safe method for capturing modeling metadata including units, ranges, etc. The Java platform provides API support for annotations making them easier to comprehend, manage, and process than other metadata representation schemes, for example XML. Although not discussed in this paper, the annotation-based approach for component integration in OMS3 is also supported for programming languages such as FORTRAN, C, or C++, allowing for the same descriptive integration scheme within those languages.

3.5. OMS3 multi-threading support

Another core design aspect of OMS3-based components is support for multi-threading, a common technique to parallelize internal processing within a software application on one machine. Unlocking multi-threading for mainstream environmental modeling techniques is challenging due to the inherent complexity, however, it is highly advantageous given the fact that almost all modern day computers have multi-core CPUs, thus supporting parallel computations. This requires EMFs to support multiple execution threads to fully utilize available processing resources and enable hierarchical scaling of environmental models. For environmental modeling applications, OMS3 code is executed using individual threads which are managed by the framework runtime. Thread communication occurs through data flow between the @Out annotated fields of one component to the @In annotated fields of another component. The component executes when all inputs are present and satisfied. This is accomplished by native language synchronization features using wait() or notify(). OMS3 mediates data flow using a producer/consumer-like synchronization pattern, and protects itself from dead-locks caused by incorrect specification of @In and @Out methods. OMS3 execution is multithreaded by design, i.e., each model component is executed in its own separate thread. Once all input fields have valid values, the component runs the execution method (@Execute). No explicit specification of execution order is required because this design supports implicit parallelism at the component level. In addition, no explicit knowledge of parallelization mechanics and threading patterns is required for a model developer. To our knowledge, this is unique for existing EMFs, however workflow systems provide for similar concepts.

In addition to multi-threading, support for hierarchical scaling is a major function provided by OMS3. That is, OMS3-based models implicitly scale to cluster and cloud-based environments without code changes. Additionally, geospatial models can share core model data structures (e.g., an HRU) and process them in parallel within a model. Studies currently are being conducted as described in Section 4 to evaluate scalability for different models, model data sets, and deployment configurations using multi-core, cluster, and cloud-based infrastructures.

3.6. OMS3 domain specific language (DSL) simulation capability

OMS3 departs from the desktop-based integrated modeling development environment in OMS2 by allowing flexible integration into different development environments and general platform integration (e.g., JGrass, web-services, etc.). The power of DSLs is leveraged to provide a flexible integration layer above the components for modeling and simulations. A DSL, in contrast to a general purpose programming language, is a programming language or specification language dedicated to a particular problem domain, a particular problem representation approach, and/or a particular solution technique (Deursen van, 1997; Deursen van et al., 2000). DSLs provide data and configuration to a program and let users write business rules for a particular task. The DSL approach is motivated by the desire to allow coding without actually promoting it. OMS3 takes advantage of the DSL "builder

design-pattern" as provided by the Groovy programming language (Dearle, 2010). This "Simulation DSL" allows the creation and configuration of runtime simulations for OMS3; however, the Simulation DSL is not inherently bound to the framework.

What constitutes a *simulation* in the OMS3 context? A simulation defines the resources needed to run an environmental model for a given purpose. A basic simulation in OMS3 consists of: 1) the component executable binaries, 2) model-specific parameters and other (e.g., climate) input data in files or databases, 3) strategies for handling model output, and 4) performance evaluation methods, e.g., simple graphing/plotting or formal evaluation statistics. Additional information and resources may be required if the simulation includes parameter estimation, sensitivity analysis, or uncertainty analysis.

Listing 2 shows a typical Simulation DSL file (*.sim) for OMS3 which is directly executable within the OMS3 runtime state. It has a hierarchical structure resembling some XML design approaches with two significant differences: 1) it is not as verbose as XML and is executable as a script, and 2) it may contain programming statements as accepted in the Java/Groovy programming language. The ability to parse a Simulation DSL is part of the OMS3 and underlying Groovy runtime. The "*.sim" file defines the model by listing all model components, defining connectivity of component fields, and providing initial parameter definitions. The simulation script file in Listing 2, as used within the Thornthwaite monthly water balance model presented in Lloyd et al. (2011a), provides underlying knowledge about component connectivity. Model components, shown as blue boxes, are specified within the components{} section in Fig. 3. Connectivity arrows are all connect {} statements, indicating data flow from source to target.

The connect{} section lists line by line data flow handled by the system. For example, the line <'climate.temp' 'soil.temp'> in Fig. 3 connects the 'temp' output field (@Out annotated) of the 'climate' component with the 'temp' input field (@In annotated) of the 'soil' component. This is the only information required from a modeler to define and establish data flow between components using @In and @Out to annotate component fields. The entries in each line follow the '<object>.<field>' notation. The field name can be omitted in the target reference if it exists in both components with the same name. The system performs conversions of field types if both the source and target field types are different, but a conversion service is offered by OMS3. Two components can be connected using fields which are type incompatible. For example, one component providing an Open GIS Consortium (OGC) simple feature collection as output can seamlessly feed into another one requiring "well known text" (ASCII encoding of geometries) input if a conversion class or service exists and is offered via a SPI. The same mechanism is also used for unit conversion or alignment of temporal/spatial scales between fields.

Listing 3 shows a Simulation DSL file for the PRMS (Leavesley et al., 2006) Java-based model used for the USDA-NRCS water supply forecasting system in the western United States. The PRMS model has been configured for parameter estimation using the USGS Luca parameter estimation method (Hay and Umemoto, 2006). OMS3/Luca is a multi-objective, stepwise, automated procedure for model calibration that uses the Shuffled Complex Evolution global search algorithm to calibrate OMS3-based models. As shown in Listing 3, in addition to the standard simulation elements such as *outputstrategy, resource, model, output, parameter*, etc., a Luca DSL simulation, executable within OMS3, defines additional elements for the calibration parameter bounds for each step or objective function type. Besides Luca, there are other available OMS3 DSL simulation types such as Fourier Amplitude Sensitivity Test sensitivity analysis (Saltelli, 2002), Dynamically Dimensioned Search parameter estimation (Tolson and Shoemaker, 2007), and Ensemble Streamflow Prediction (Kim et al., 2006).

To summarize, Simulation DSLs are easily adjustable to new simulation types (e.g., parameter estimation or uncertainty analysis methodology) and provide the model user with a high degree of freedom in setting up complex simulations (e.g., batch processing of multiple watersheds for stream flow or water quality prediction). DSL scripts allow very concise and understandable expressions although they can contain "traditional" programming code. This flexibility makes them very attractive for simulation integration and superior to "data-only" static representations for capturing modeling metadata such as XML.

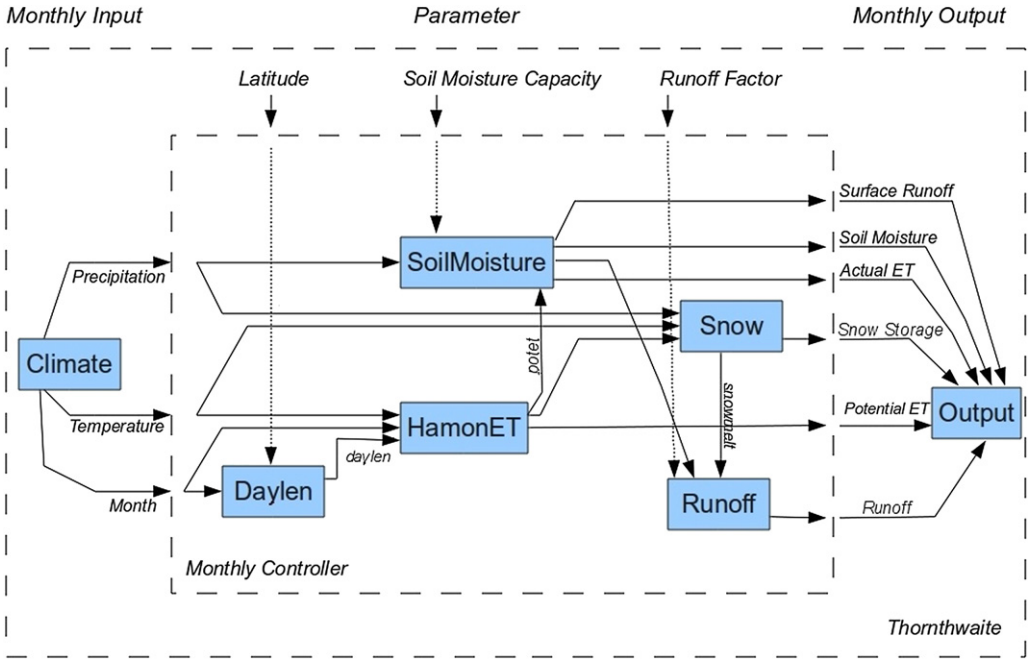

Fig. 3. Thornthwaite model components and data flow (from Lloyd et al., 2011a).

4. OMS3 applications

4.1. Cloud Services Innovation Platform (CSIP)

Recent attention has been directed towards enabling OMS3 to seamlessly scale models through the use of cloud infrastructures and service-oriented architectures (SOAs). Cloud computing is emerging as a viable and attractive solution for scientific computing (Hoffa et al., 2008). The main goal is to scale parallel processing of components beyond individual computers to harness networks of virtual computers while not requiring significant code changes. For that purpose, the USDA-NRCS has initiated the Cloud Services Innovation Platform (CSIP) (Fig. 4).

The goal of CSIP is to develop a scalable, modular, cost effective, and open deployment platform for simulation models to deliver legacy and research simulation models as cloud-based web-services. To provide developers with a robust SOA environment, CSIP incorporates existing USDA infrastructure components including OMS3 and soil, management, climate, and other databases to support environmental modeling within both managed private clouds and public clouds (e.g., Amazon EC2 or Rackspace). A primary research goal for CSIP development is to gain experience and implement sustainable strategies for model and data services in a cloud environment. The use of OMS3-based annotation interfaces under CSIP accelerates the migration of legacy environmental models for resultant cloud-based deployment.

Initial CSIP research has developed a web-service implementation of the RUSLE2 (Revised Universal Soil Loss Equation Version 2, Foster et al., 2001) model for estimating sediment production on upland areas. This effort is in support of the USDA-NRCS Conservation Delivery Streamlining Initiative program in collaboration with the USDA-ARS. CSIP is currently leveraged to run field- and watershed-scale models in a scalable compute cloud environment to assist the USDA Conservation Effects Assessment Program (Duriancik et al., 2008). RUSLE2 has historically been used as a Windows™-based desktop application to guide conservation planning and inventory erosion rates over large areas. The model provides a reusable computational engine that can be used without a user interface for model runs in other applications. RUSLE2's computational engine was integrated into an OMS3 model to support efficient execution and was enhanced using innovative non-relational database approaches. The resulting RUSLE2/OMS3 erosion component was embedded into a RESTful web service (Richardson and Ruby, 2007) for input data management; data retrieval for soils, climate and management records; data conversion; and data caching. A single server manages access to cloud-based compute nodes. RUSLE2 model tests on the order of 100K+ model runs has been completed using hundreds of cloud nodes to verify the utility of a cloud-based deployment (Lloyd et al., 2011b).

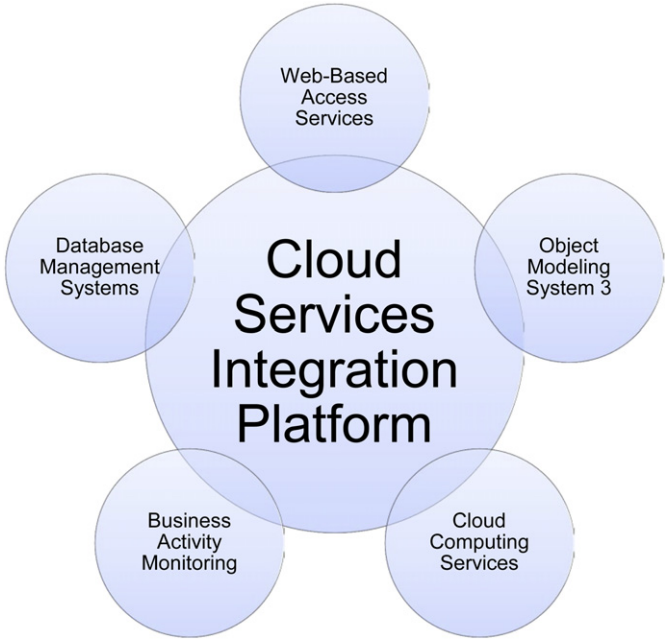

Fig. 4. Cloud Services Innovation Platform (CSIP) software architecture.

To demonstrate cloud-based support for environmental modeling, a prototype application was developed to showcase running the RUSLE2 model under CSIP from an Android™ mobile device (Fig. 5). The interactive workflow shows the parameterization of RUSLE2 erosion transects by accepting manual input or using USGS elevation services. Mobile mapping features are utilized to visualize location information available via global positioning system, transect direction, or latitude/longitude information. CSIP-based geospatial databases are queried to determine location-specific land use/land cover management options. Model runs are performed using CSIP supported by cloud compute node(s). Upon model completion, the mobile device displays erosion values for the given input parameters.

CSIP development remains ongoing. The flexibility offered by OMS3 components using the annotation-based integration method was essential for a RESTful web services development. RESTful service definition is enabled using web-service specific annotations on the OMS3 modeling components.

4.2. Water supply forecasting and watershed modeling

There are several operational and research-focused OMS3 model applications to date. The National Water and Climate Center of the USDA-NRCS is augmenting seasonal, regression-equation based water supply forecasts with shorter-term forecasts based on the use of distributed-parameter, physical process hydrologic models and an Ensemble Streamflow Prediction (ESP) methodology. The primary ESP model base (Leavesley et al., 2010) is built using OMS3 and the PRMS hydrological watershed model. The model base will be used to address a wide variety of water-user requests for more information on the volume and timing of water availability and to improve water supply forecast accuracy. The PRMS/ESP methodology is a modified version of the ESP procedure developed by the National Weather Service (Day, 1985) which uses historical or synthesized meteorological data as an analog for the future with the timeseries data used as model input to simulate future stream flow.

A visualization tool running under OMS3 is available for visual display of user-selected ESP output traces. The tool performs a frequency analysis on the peaks and/or volumes of the simulated hydrograph traces and displays a list of all the historic years used with their associated probability of exceedance. Different options are available in applying frequency analysis. One assumes that all years in the historic database have an equal likelihood of occurrence. Alternative schemes for weighting user-defined periods, based on user assumptions or *a priori* information, are also being investigated. El Niño, La Niña, and Pacific Decadal Oscillation (PDO) periods have been identified in the ESP procedure, and these can be sorted and extracted separately for analysis. The PRMS/ESP tool running under OMS3 will provide timely forecasts for use by the agricultural community in the western United States where snowmelt is a major source of water supply.

Another modeling application currently being developed under the OMS3 framework is the component-oriented AgES-W (Agro-Ecosystem-Watershed) model. AgES-W is a fully distributed

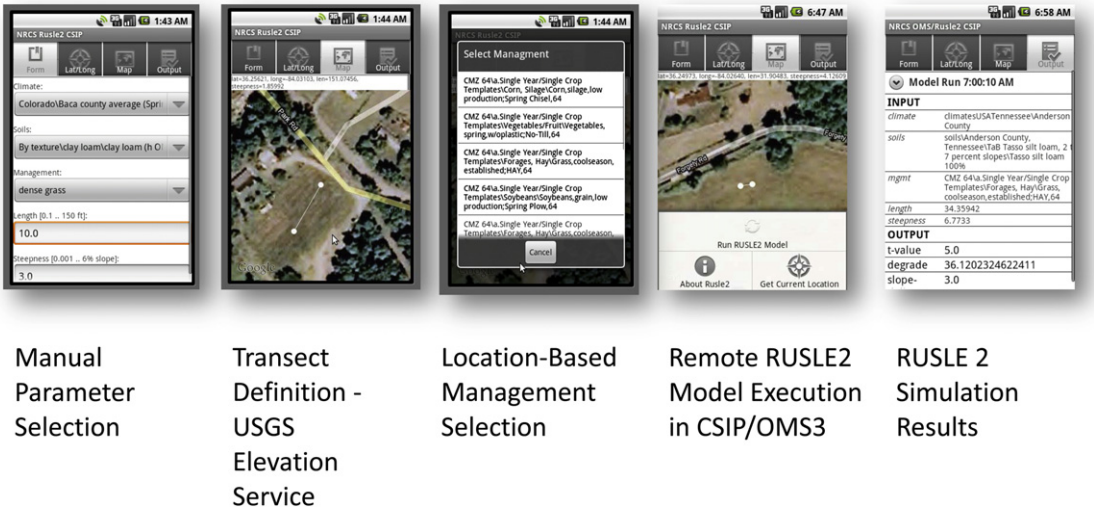

Fig. 5. CSIP/OMS3-based mobile RUSLE2 erosion model application.

simulation of water quantity and quality in large watersheds (Ascough et al., 2010). AgES-W consists of Java-based simulation components (80+ representing interception; snow processes; soil water balance; nutrient (nitrogen and phosphorus) cycling; erosion; lateral flow and groundwater movement; and runoff concentration, flood, and chemical routing in channels) from the J2K-S (Krause et al., 2009), SWAT, RZWQM2, and WEPP models. AgES-W simulates conjunctive stream flow and groundwater interaction, carried out by HRUs which are connected by a lateral routing scheme to simulate lateral water transport processes. This permits fully distributed hydrological modeling of river basins. AgES-W performance for stream flow prediction was evaluated recently (Ascough et al., in press) for the Cedar Creek Watershed in northeastern Indiana, USA, one of 14 benchmark watersheds in the USDA-ARS Conservation Effects Assessment Project watershed assessment study. Future plans are to enhance AgES-W for: 1) diverse cropping system responses to water deficits, 2) model uncertainty analyses and scaling, and 3) plant responses to atmospheric CO_2. New OMS3 tools currently under development to facilitate AgES-W application include HRU delineation, new sensitivity/uncertainty analysis methods and spatial visualization tools, and web-based cloud computing as described in the previous section.

5. Summary and conclusions

Environmental modeling frameworks streamline and accelerate the model development and implementation process; however an initial learning curve for EMF-based modeling always exists. Resource and in-kind institutional support is important for the acceptance of an EMF, but it is up to the modeler to adopt an EMF so that it becomes an integral part of the model development workflow. Apart from social and cultural barriers, EMF developers are further challenged technically to develop frameworks which are less cumbersome for modelers to adopt. Web service and database framework projects outside the modeling community have demonstrated that model developers will adopt a software development framework if it is easily understood, enables seamless integration of existing codebases and workflows, and does not invalidate existing institutional software development practices.

For the above reasons, designing EMFs and associated programming interfaces is an extremely challenging task. Numerous modeling frameworks are currently under development worldwide with the primary purpose of integrating existing and future environmental models into common, inter-operable, and flexible systems. One such framework, the Object Modeling System Version 3 (OMS3), represents a persuasive choice for adoption with its inherent non-invasive and scalable implementation. OMS3 development leverages successful framework designs and software engineering principles originating from various general purpose and web-based application frameworks. In OMS3, the internal complexity of the framework has been reduced by adopting a lightweight design, thereby resulting in a less steep learning curve as there are fewer complex technical details for the model developer to absorb. Parallelism in OMS3 is achieved using multi-threading on multi-core CPUs and research is ongoing to further extend the scalability of OMS3 by adopting MapReduce-based large-scale distributed computing environments (Dean and Ghemawat, 2004) such as Hadoop™. OMS3 can be considered a non-invasive modeling framework for component-based model and simulation development on multiple platforms. As shown by Lloyd et al. (2011a), the straightforward component integration structure allows rapid implementation of new models and an easier adaptation of existing models and components. Other studies have shown that this approach leads to models with less overhead and a more intuitive design. By embracing the use of non-intrusive language annotations for modeling metadata specification and framework integration in favor of traditional APIs, OMS3-based models keep their identity outside of the modeling framework. Annotations enable multi-purposing of components, which is difficult to accomplish with a traditional API design. In OMS3, annotations provide component connectivity, data transformation, unit conversion, and automated generated of model documentation. In addition to the Java programming language, the annotation-based approach for component integration in OMS3 is also supported for programming languages such as FORTRAN, C, C++, and C#.

OMS3 introduces an extensible and lightweight layer for simulation description that is expressed as a Simulation DSL based on the Groovy framework. DSL elements are simple to define and use for basic model applications, or for more complex setups for parameter estimation, sensitivity/uncertainty analysis, etc. The use of DSLs for “programmable” configuration eliminates core programming language “noise” and is efficacious for many different types of modeling applications (e.g., distributed watershed

modeling to support automated setup of multiple batch model runs).

In summary, the overall development goal of OMS3 has been to provide features to make it easier for modelers to create contemporary, inter-operable, scalable and lightweight models by fully leveraging computing resources, data stores, and infrastructure opportunities. By harnessing lightweight framework design principles, model development can become a more efficient and rewarding exercise for scientists while model users can experience the benefits of scalable, contemporary modeling applications.

Acknowledgments

We would like to thank the Departments of Civil and Environmental Engineering and Computer Science at Colorado State University for contributing to research and development of the OMS framework, and the Department of Geography at Friedrich Schiller University for contributing to development of associated OMS-based environmental models.

Listing 1.
Simulation component example in OMS3.

```
package climate;

import oms3.annotations.*;
import static oms3.annotations.Role.*;

@Author
  (name = "Peter Krause, Sven Kralisch")
@Description
  ("Calculates land use state variables")
@Keywords
  ("I/O")
@SourceInfo
  ("$HeadURL: http://svn.javaforge.com/svn/oms/branches/oms3.prj.ceap/src/
   climate/CalcLanduseStateVars.java $")
@VersionInfo
  ("$Id: CalcLanduseStateVars.java 1050 2010-03-08 18:03:03Z ascough $")
@License
  ("http://www.gnu.org/licenses/gpl-2.0.html")
@Status
  (Status.TESTED)

public class CalcLanduseStateVars {

  @Description("Attribute Elevation")
  @In public double elevation;

  @Description("Array of state variables LAI ")
  @In public double[] LAI;

  @Description("effHeight")
  @In public double[] effHeight;

  @Description("Leaf Area Index Array")
  @Out public double[] LAIArray;

  @Description("Effective Height Array")
  @Out public double[] effHArray;

  @Execute
  public void compute() {
    LAIArray = new double[366];
    effHArray = new double[366];
    for(int i = 0; i < 366; i++){
      LAIArray[i] = calcLAI(LAI, elevation, i+1);
      effHArray[i] = calcEffHeight(effHeight,elevation, i+1);
    }
  }
  // code for calcLAI and calcEffHeight
}
```

Listing 2.
Simulation DSL example in OMS3 for the Thornthwaite monthly water balance model.

```
sim(name:"TW") {
build(targets:"all")
  // define output strategy: output base dir and
  // the strategy NUMBERED|SIMPLE|TIME
  outputstrategy(dir: "$oms_prj/output", scheme:SIMPLE)
  // define model

  model(iter:"climate.moreData") {
    components {   // listing of all model components
      climate   'tw.Climate'
      daylen   'tw.Daylen'
      et   'tw.HamonET'
      out   'tw.Output'
      runoff   'tw.Runoff'
      snow   'tw.Snow'
      soil   'tw.SoilMoisture'
    }
    connect {   // component connectivity: 'source' 'target'
      // climate
      'climate.temp'   'soil.temp'
      'climate.temp'   'et.temp'
      'climate.temp'   'snow.temp'
      'climate.precip'   'soil.precip'
      'climate.precip'   'snow.precip'
      'climate.time'   'daylen.time'
      'climate.time'   'et.time'
      'climate.time'   'out.time'

      // daylen
      'daylen.daylen'   'et.daylen'
      'daylen.daylen'   'out.daylen'

      // soil
      'soil.surfaceRunoff'   'out.surfaceRunoff'
      'soil.surfaceRunoff'   'runoff.surfaceRunoff'
      'soil.soilMoistStor'   'out.soilMoistStor'
      'soil.actET'   'out.actET'

      // PET
      'et.potET'   'soil.potET'
      'et.potET'   'snow.potET'
      'et.potET'   'out.potET'

      // Snow
      'snow.snowStorage'   'out.snowStorage'
      'snow.snowMelt'   'runoff.snowMelt'

      // runoff
      'runoff.runoff'   'out.runoff'
    }

    parameter {   // initial model parameter 'comp.field' value
      'climate.climateInput' "$oms_prj/data/climate.csv"
      'out.outFile'   "$oms_prj/output/TW/out/output.csv"
      'runoff.runoffFactor'   0.5
      'daylen.latitude'   35.0
      'soil.soilMoistStorCap' 200.0
    }
  }
  // model efficiency (optional)
  efficiency(obs:"precip",sim:"runoff",
      precip:"precip", methods:NS+ABSDIF+TRMSE)
  // compute annual summary for runoff 'on-the-fly' (optional
  summary(time:"time", var:"runoff",
   moments:COUNT+MEAN+MIN, period:YEARLY)
  analysis(title:"Model output") {
    timeseries(title:"Monthly waterbalance", view: COMBINED) {
      x(file:"%last/output.csv", table:"tw", column:"date")
      y(file:"%last/output.csv", table:"tw", column:"runoff")
      y(file:"%last/output.csv", table:"tw", column:"daylen")
    }
  }
}
```

Listing 3.
Simulation DSL example in OMS3 for Luca parameter estimation.

```
/* Luca calibration.*/
luca(name: "EFC-luca") {

  // define output strategy: output base dir and
  // the strategy NUMBERED|SIMPLE|DATE
  outputstrategy(dir: "$work/output", scheme:NUMBERED)

  // for class loading: model location
  resource "$work/dist/*.jar"

  // define model
  model(classname:"model.PrmsDdJh") {
    // parameter
  parameter (file:"$work/data/efc/params_lucatest.csv") {
    inputFile   "$work/data/efc/data_lucatest.csv"
    outFile   "out.csv"
    sumFile   "basinsum.csv"
    out   "summary.txt"

    startTime "1980-10-01"
    endTime   "1984-09-30"
    }
  }

  output(time:"date", vars:"basin_cfs,runoff[0]",
    fformat="7.3f", file:"out1.csv")
  calibration_start "1981-10-01"   // Calibration start date
  rounds 2      // calibration rounds, default 1

  // step definitions
  step(name:"Et param") {
    parameter {
      jh_coef(lower:0.001, upper:0.02, strategy:MEAN)
    }
    optimization(simulated:"out1.csv|EFC-luca|basin_cfs",
      observed:"$work/data/efc/data_lucatest.csv|obs|runoff[0]") {
    of(method:ABSDIF, timestep:DAILY)
    }
  }

  step(name:"soil param" {
    parameter {
      ssrcoef_sq(lower:0.001, upper:0.4, strategy:MEAN)
      soil2gw_max(lower:0.001, upper:0.4, strategy:MEAN)
  }
  optimization(simulated:"out1.csv|EFC-luca|basin_cfs",
      observed:"$work/data/efc/data_lucatest.csv|obs|runoff[0]") {
    of(method:ABSDIF, timestep:DAILY)
    }
  }
}
```

References

Ahuja, L.R., Rojas, K.W., Hanson, J.D., Shaffer, M.J., Ma, L. (Eds.), 2000. The Root Zone Water Quality Model. Water Resources Publications LLC, Highlands Ranch, CO USA.

Ahuja, L.R., Ascough II, J.C., David, O., 2005. Developing natural resource modeling using the object modeling system: feasibility and challenges. Advances in Geosciences 4, 29–36.

Argent, R., 2004. An overview of model integration for environmental applications – components, frameworks and semantics. Environmental Modelling & Software 19 (3), 219–234.

Arnold, J.G., Allen, P.M., Bernhardt, G., 1993. A comprehensive surface-groundwater flow model. Journal of Hydrology 142, 47–69.

Ascough II, J.C., David, O., Krause, P., Fink, M., Kralisch, S., Kipka, H., Wetzel, M., 2010. Integrated agricultural system modeling using OMS 3: component driven stream flow and nutrient dynamics simulations. In: Swayne, D.A., Yang, W., Voinov, A.A., Rizzoli, A., Filatova, T. (Eds.), Proc. Fifth Biennial Conference of the International Environmental Modelling and Software Society, Modelling for Environment's Sake, ISBN 978-88-9035-741-1, pp. 1089–1097. Ottawa, Canada, July 5–8, 2010.

Ascough II, J.C., David, O., Krause, P., Heathman, G.C., Kralisch, S., Larose, M., Ahuja, L.R., Kipka, H., 2012. Development and application of a modular watershed-scale hydrologic model using the Object Modeling System: Runoff response evaluation. Transactions of the ASABE 55 (1), 117–135.

Bernholdt, D.E., Elwasif, W.R., Kohl, J.S., Epperly, T.G.W., 2003. A component architecture for high-performance computing. In: Proc. of the Workshop on Performance Optimization for High-Level Languages and Libraries (POHLL-02). New York, NY, 22 June, 2003.

Blind, M., Gregersen, J.B., 2005. Towards an Open Modeling Interface (OpenMI) the HarmonET project. Advances in Geosciences 4, 69–74.

Briand, L.C., Wust, J., Daly, J., Porter, D.V., 2000. Exploring the relationships between design measures and software quality in object-oriented systems. Journal of Systems & Software 15 (3), 245–273.

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 High Performance Computing Applications Fall/Winter 2005 19, 341–350.

David, O., Ascough II, J.C., Leavesley, G., Ahuja, L.R., 2010. Rethinking modeling framework design: object modeling system 3.0. In: Swayne, D.A., Yang, W., Voinov, A.A., Rizzoli, A., Filatova, T. (Eds.), Proc. Fifth Biennial Conference of the International Environmental Modelling and Software Society. Modelling for Environment's Sake, Ottawa, Canada, July 5–8, 2010, pp. 1183–1191.

Day, G.N., 1985. Extended streamflow forecasting using NWSRFS. Journal of Water Resources Planning and Management (ASCE) 111, 157–170.

Dean, J., Ghemawat, S., 2004. MapReduce: Simplified Data Processing on Large Clusters. Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.

Dearle, F., 2010. Groovy for Domain-Specific Languages. Packt Publishing, Birmingham, UK, 312 pp.

Deursen van, A., Klint, P., Visser, J., 2000. Domain-specific languages: an annotated bibliography. ACM SIGPLAN Notices 35 (6), 26–36.

Deursen van, A., 1997. Domain-specific languages versus object-oriented frameworks: a financial engineering case study. In: Smalltalk and Java in Industry and Academia, STJA'97. Ilmenau Technical University, pp. 35–39.

Duriancik, L.F., Bucks, D., Dobrowolski, J.P., Drewes, T., Eckles, S.D., Jolley, L., Kellogg, R.L., Lund, D., Makuch, J.R., O'Neill, M.P., Rewa, C.A., Walbridge, M.R., Parry, R., Weltz, M.A., 2008. The first five years of the conservation effects assessment project. Journal of Soil and Water Conservation 63 (6), 185A–197A.

Flanagan, D.C., Nearing, M.A. (Eds.), 1995. USDA-Water Erosion Prediction Project: Hillslope Profile and Watershed Model Documentation. NSERL Report No. 10. West Lafayette, IN: USDA-ARS National Soil Erosion Research Laboratory.

Foster, G.R., Yoder, D.C., Weesies, G.A., Toy, T.J., 2001. The design philosophy behind RUSLE2: evolution of an empirical model. In: Ascough II, J.C., Flanagan, D.C. (Eds.), Soil Erosion Research for the 21st Century, Proc. Int. Symp., 3–5 January 2001, Honolulu, HI, USA. ASAE, St. Joseph, MI, pp. 95–98.

Gregersen, J.B., Gijsbers, P.J.A., Westen, S.J.P., 2007. OpenMI: Open modelling interface. Journal of Hydroinformatics 9 (3), 175–191.

Hay, L.E., Umemoto, M., 2006. Multiple-objective Stepwise Calibration Using Luca, U.S. Geological Survey Open-file Report 2006-1323, 25 pp.

Hoffa, C., Mehta, G., Freeman, T., Deelman, E., Keahey, K., Berriman, B., Good, J., 2008. On the Use of Cloud Computing for Scientific Workflows. SWBES 2008, Indianapolis, IN. 10–12 December 2008.

Kim, Y.-O., Jeong, D., Ko, I.H., 2006. Combining rainfall-runoff model outputs for improving ensemble streamflow prediction. Journal of Hydrologic Engineering 11 (6), 578–588.

Krause, P., Bende-Michl, U., Fink, M., Helmschrot, J., Kralisch, S., Künne, A., 2009. Parameter sensitivity analysis of the JAMS/J2000-S model to improve water and nutrient transport process simulation – a case study for the Duck catchment in Tasmania. In: Anderssen, R.S., Braddock, R.D., Newham, L.T.H. (Eds.), 18th World IMACS Congress and MODSIM09 International Congress on Modelling and Simulation. Modelling and Simulation Society of Australia and New Zealand and International Association for Mathematics and Computers in Simulation, July 2009, ISBN 978-0-9758400-7-8, pp. 3179–3186.

Leavesley, G.H., Lichty, R.W., Troutman, B.M., Saindon, L.G., 1983. Precipitation-Runoff Modeling System – User's Manual. U.S. Geol. Surv. Water Resour. Invest. Rep. 83-4238.

Leavesley, G.H., Markstrom, S.L., Viger, R.J., 2006. USGS modular modeling system (MMS) – Precipitation-runoff modeling system (PRMS). In: Singh, V.P., Frevert, D.K. (Eds.), Watershed Models. CRC Press, Boca Raton, FL, pp. 159–177.

Leavesley, G., David, O., Garen, D., Goodbody, A., Lea, J., Marron, J., Perkins, T., Strobel, M., Tama, R., 2010. A Modeling Framework for Improved Agricultural Water-supply Forecasting. In: Proc. Joint 9th Federal Interagency Sedimentation Conference and 4th Federal Interagency Hydrologic Modeling Conference, June 27 – July 1, 2010, Las Vegas, Nevada.

Lloyd, W., David, O., Ascough II, J.C., Rojas, K.W., Carlson, J.R., Leavesley, G.H., Krause, P., Green, T.R., Ahuja, L.R., 2011a. Environmental modeling framework invasiveness: analysis and implications. Environmental Modelling & Software 26 (10), 1240–1250.

Lloyd, W.J., Pallickara, S., David, O., Lyon, J., Rojas, K., 2011b. An Exploratory Investigation on the Migration of Multi-tier Applications to Virtualized Cloud-based Infrastructures. Paper presented at Grid 2011: The 12th IEEE/ACM International Conference on Grid Computing, 21–23 September, Lyon, France.

Moore, A.D., Holzworth, D.P., Herrmann, N.I., Huth, N.I., Robertson, M.J., 2007. The Common Modelling Protocol: a hierarchical framework for simulation of agricultural and environmental systems. Agricultural Systems 95 (1–3), 37–48.

Peckham, S., 2008. CSDMS Handbook of Concepts and Protocols: A Guide for Code Contributors. http://csdms.colorado.edu/wiki/Help:Tools_CSDMS_Handbook (accessed 2.02.12.).

Richardson, L., Ruby, S., 2007. RESTful Web Services. O'Reilly Media, Sebastopol, CA, 448 pp.

Richardson, C., 2006. POJOs in Action: Developing Enterprise Applications with Lightweight Frameworks. Manning Publications Co., Greenwich, CT, 456 pp.

Rizzoli, A.E., Leavesley, G.H., Ascough II, J.C., Argent, R.M., Athanasiadis, I.N., Brilhante, V.C., Claeys, F.H., David, O., Donatelli, M., Gijsbers, P., Havlik, D., Kassahun, A., Krause, P., Quinn, N.W., Scholten, H., Sojda, R.S., Villa, F., 2008. Chap. 7: integrated modelling frameworks for environmental assessment and decision support. In: Jakeman, A.J., Voinov, A.A., Rizzoli, A.E., Chen, S.H. (Eds.), 2008. Environmental Modelling and Software and Decision Support – Developments in Integrated Environmental Assessment (DIEA), vol. 3. Elsevier, The Netherlands, pp. 101–118.

Saltelli, A., 2002. Making best use of model evaluations to compute sensitivity indices. Computer Physics Communications 145 (2), 280–297. doi:10.1016/s0010-4655(02)00280-1.

Schmidt, D.C., 1999. Why software reuse has failed and how to make it work for you. C++ Report 11 (1), 1999.

Tolson, B.A., Shoemaker, C.A., 2007. Dynamically dimensioned search algorithm for computationally efficient watershed model calibration. Water Resources Research 43, W01413. doi:10.1029/2005WR004723.

Yuan, M.J., Orshalick, J., Heute, T., 2009. Seam Framework: Experience the Evolution of Java EE, second ed. Prentice Hall Publishers, Upper Saddle River, NJ, 504 pp.