International Environmental Modelling and Software Society (iEMSs)
2010 International Congress on Environmental Modelling and Software
Modelling for Environment's Sake, Fifth Biennial Meeting, Ottawa, Canada
David A. Swayne, Wanhong Yang, A. A. Voinov, A. Rizzoli, T. Filatova (Eds.)
http://www.iemss.org/iemss2010/index.php?n=Main.Proceedings

Rethinking Modeling Framework Design: Object Modeling System 3.0

O. David[a], J.C. Ascough II[b], G.H. Leavesley[c], L. Ahuja[b]

[a]Depts. of Civil and Environmental Engineering and Computer Science, Colorado State University, Fort Collins, CO 80523 USA (e-mail: odavid@colostate.edu)
[b]USDA-ARS-NPA, Agricultural Systems Research Unit, Fort Collins, CO 80526 USA
[c]Dept. of Civil and Environmental Engineering, Colorado State University, Fort Collins, CO 80523 USA

Abstract: The Object Modeling System (OMS) is a framework for environmental model development, data provisioning, testing, validation, and deployment. It provides a bridge for transferring technology from the research organization to the program delivery agency. The framework provides a consistent and efficient way to create science components, build, calibrate, and evaluate models and then modify and adjust them as the science advances, in addition to re-purposing models for emerging customer requirements. OMS was first released in 2004 and version 3.0 represents a major milestone towards an easier to use, more transparent and scalable implementation of an environmental modeling framework. OMS3 development is the result of an in-depth analysis of successful framework designs and software engineering principles as provided by general-purpose modeling frameworks. Like any modeling framework, OMS3 is *enabling* technology for modeling. The main goal of OMS3 development is an easier integration of model source code based on language annotations while being flexible to adopt existing legacy models. In OMS3, the internal complexity of the framework itself was reduced while allowing models to implicitly scale from multi-core desktops to clusters to clouds, without burdening the model developer with complex technical details.

***Keywords**:* Modeling framework; Software design; Component modeling; Noninvasive framework.

1. INTRODUCTION

Frameworks in general support efficient software development; environmental modeling frameworks (EMF) (Rizzoli et al, 2008) target modelers to help implement models that are state-of-the-art in science and software engineering, operate across domains, scale with problem complexity, and can be integrated with standard methods for calibration, optimization, and sensitivity/ uncertainty analysis. Using an EMF for model implementation has advantages not just for the model developer(s) - organizations not initially involved in the model creation process can benefit from EMFs as well. EMF-based models are more interoperable (since they typically adhere to some programming standard), can be coupled and integrated, and new model uses can be implemented more quickly. In addition, EMF-based models have an enforced structure that is provided by the framework so they can be better understood by developers outside of the initial development team. All the above benefits are available once a model is implemented using an EMF, or a legacy model is adapted to an EMF.

An initial learning curve for EMF-based modeling always exists. Depending on the scope and comprehensiveness of the EMF, the modeler typically has to adopt programming techniques such as structured programming and modular decomposition (always a good practice), and be proficient in object-oriented programming or even architectural design patterns. If the framework targets high performance computing, a basic understanding is helpful towards dividing a complex problem in such a way that many CPUs can process models with measurable performance gain. The institutional support and acceptance of an EMF might help providing resource and in-kind support, but it is up to the modeler to

adopt an EMF in a way that is an integral part of the modeling workflow. Besides cultural and social barriers, there is also the technical challenge for the framework developer to lower the burden for successful framework adaptation by modelers. Web service or database framework projects outside the modeling community have demonstrated that model developers will adopt a framework if it is easy to approach, fits into an existing codebase and workflow seamlessly, and therefore does not fully invalidate existing practices of software development.

OMS in its previous versions (2.x and earlier) represented the traditional framework development approach, i.e., building a framework supporting object-oriented or more specifically component-based modeling. It provided, like many EMFs, a core Application Programming Interface (API) that featured framework data types, base classes, support classes for time/space management, process management, data manipulation and I/O in a library that the model used via inheritance, interface implementation, and plain calls.

The evolution of the OMS3 framework design is very much related to the design of model components, and is also driven by usage experience:

- OMS framework versions 1.x offered traditional library classes that a model or component had to either subclass or instantiate directly. The framework defined a limited set of data types that could be exchanged between model components. This was a simple approach, however, it limited modelers in their ability to use and share custom data types. For legacy code integration, this type of constraint is undesirable. Moreover, needed built-in flexibility for unit, type, and data transformation resulted in complex and large amounts of internal framework code. In addition, since model components had to be subclasses of the framework, blending into existing class hierarchies was impossible and a delegator/wrapper for existing components was always needed.

```
/** Elevation.
    @unit ft
    @access read
 **/
private OMSDouble elevation;
```

- OMS 2.x simplified component design by allowing a model/component to use only interfaces instead of classes, i.e., no framework interface implementation was accessible from within the modeling component. As a result, the modeling component became more lightweight. This "design-by-contract" implementation allowed OMS 2.x to offer variants of framework data types with variable 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 reduced API size. However, the data types being supported by the framework were still a fixed set:

```
@Description("Elevation")
private Attribute.Double elevation;
```

For any OMS version prior to 3.0, *framework data types* for component data transfer objects had to be introduced in addition to the existing Java language counterparts. This represented a redundancy for types such as `double` and `Double` by providing an `OMSDouble` class implementation (OMS1.x) or an `Attribute.Double` (OMS2.x)

OMS 3.x departs from the traditional API-based framework approach for component design in favor of a more lightweight, non-invasive implementation. Software developers in general and model developers specifically, are in favor of lightweight frameworks in contrast to heavyweight frameworks. Heavyweight frameworks, e.g. traditional object-oriented frameworks, provide developers with an API that is often large and developers typically spend considerable time becoming familiar with framework APIs before writing model code. In contrast, 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. Programming language annotations which capture metadata are used to identify specific points in the model code where framework functionality is integrated. The following section introduces the component design and layout in OMS3.

2. MODELING COMPONENTS ARE PLAIN OBJECTS

Like in other modeling frameworks, an OMS3 modeling component implements a *domain specific simulation concept* as software code. The term component refers to a concept in software engineering which extends the reusability of code from the source level to the binary executable. Components are context-independent, both in the conceptual and technical domain. They represent self-contained software units that are separated from the surrounding framework environment.

OMS3 provides a radically simplified approach for model component design. It allows Plain Objects (Plain Old Java Objects) or POJOs that are annotated to be used within the framework. Annotated POJOs are easy to create since they are normal classes enriched with Java language level annotations. They provide the framework with hints to use the component as a model building block. Technologies that enable this type of framework design include:

- Runtime *introspection* on object and class structure, fields, methods, and classes - for exploring the internals of a component and finding framework entry points for data flow and execution.
- Language level *annotations* on classes, methods and fields – for describing dataflow fields and tagging IEF (Init/Execute/Finalize) methods.
- *Reflective access* on object fields and *reflective method invocation* – for component-to-component data transfer and indirect component 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.

Listing 1 shows a simple component for lookup table computation in the OMS3/J2K-S watershed model (Ascough II et al., 2010). All annotations start with an (@) symbol.

```
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 1. Component example in OMS3.

Annotations have the following features in OMS3:

- OMS3 package dependencies only exist for annotations (`oms3.annotations.*`). No API calls are needed ("Inversion of Control").
- Annotations exist to document the whole component; single fields have them as well for data flow specification and documentation.
- A component adheres to the IEF cycle by tagging methods with the corresponding annotations. The computational method of this component is "tagged" with the `@Execute` annotation, the name of the method is not significant.
- Data flow indications are provided by using `@In` and `@Out` annotations.
- No explicit marshalling or un-marshalling of component variables is needed, i.e., an assignment is sufficient to pass them on to the receiving component.

Annotations provide a more integrated and context safe way for adding modeling specific metadata to code such as units, ranges, etc. Since there is API support within the Java platform, they are easier to comprehend, manage, and process that other metadata representation, for example XML.

Another core design aspect of components in OMS3 is the support of multithreading from the ground up. Since almost every new computer has multi-core processing, the inherent support of multiple execution threads via the modeling framework is required develop and deploy models which scale with processing resources.

Even for simple model applications, each OMS3 component is executed in its own separate thread which is managed by the framework runtime. Thread communication happens through data flow from the `@Out` field of one component to the `@In` field of another component. The component will execute if all inputs are present and satisfied. This is accomplished by native language synchronization features using `wait()/notify()`. OMS3 internally only mediates data flow in a producer/consumer-like synchronization pattern, and also protects itself from dead-lock situations caused by an incorrect Out/In setup. This ultimately leads to implicit parallelism at the component level, i.e., no explicit knowledge of parallelization mechanics and threading patterns is required for a model developer.

In addition to multi-threading, OMS3 also scales into cluster and cloud environments without any model recoding. Being able to implicitly scale models was also one of the major OMS3 refactoring accomplishments. Using Distributed Shared Objects (DSO) in Terracotta, geospatial models can share core model data structures (e.g., a hydrologic response unit, HRU) and process them in parallel within a model. Studies are being conducted to revaluate scalability for different models, model data sets, and setups on multi-core, cluster, and most interestingly in cloud infrastructures.

3. SIMULATIONS AS DOMAIN SPECIFIC LANGUAGES

OMS3 departs from the very GUI-centric design in version 2.2, by allowing more flexibility for integration into different development environments (IDEs) and general platform integration (e.g., JGrass and web-services stack). A flexible integration layer above the modeling components is provided by leveraging the power of a Domain Specific Language (DSL) for modeling and simulations. A DSL, in contrast to general purpose programming languages, is a programming language or specification language dedicated to a particular problem domain, a particular problem representation technique, and/or a particular solution technique. DSLs have the task of providing data (configuration or otherwise) to a program and let users write business rules for a particular task. This is usually motivated by the desire to allow coding without actually promoting it. OMS3 takes advantage of the builder design-pattern DSL, as provided by the Groovy programming language. This newly developed "Simulation DSL" allows the creation and configuration of simulations for OMS3; however, the Simulation DSL is not inherently bound to the framework.

What is a simulation in the OMS3 context? A simulation defines all the resources that are needed to run a model for a given purpose. A basic simulation in OMM3 consists of: (i) the model and component executables, (ii) model-specific parameter and other (e.g., climate) input data in files or databases, (iii) some strategy for handling output, and (iv) a method to evaluate model performance with simple graphing/plotting or formal evaluation statistics. There might be additional information and resources required if the simulation includes parameter estimation, sensitivity analysis, or uncertainty analysis.

Listing 2 shows a simulation using the Precipitation Runoff Modeling System (PRMS, Leavesley et al., 2006) Java-based model (`PRMSHdJh`) used for USDA-Natural Resource Conservation Service (NRCS) water supply forecasting in the western United States. The PRMS model has been set up for parameter estimation using the USGS Luca method. OMS3/*luca* is a multiple-objective, stepwise, automated procedure for model calibration that uses the Shuffled Complex Evolution global search algorithm to calibrate any OMS3-based model in multiple rounds and calibration steps. As shown in Listing 2, 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, model parameter bounds for each step, or objective function type. Besides *luca*, there are other OMS3 DSL simulation types, such as *fast* (FAST sensitivity analysis), *dds* (Dynamic Dimensioned Search parameter estimation), or *esp* (Ensemble Streamflow Prediction).

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

   // workspace directory
   def work = System.getProperty("oms3.work");

   // 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)
    }
  }
}
```

Listing 2. Luca DSL parameter estimation example in OMS3.

Our experience in developing and working with simulation DSLs has shown that they are easy to adjust to new simulation types (e.g., parameter estimation or uncertainty analysis) 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). DSLs look very much descriptive (and in fact they are), and can also contain any Java language statements. This type of behavior makes them very attractive for flexible integration of simulations and superior over "data-only" and static representations such as XML.

4. OMS3 MODEL APPLICATIONS

Using OMS3, a number of science model implementations and applications are currently being tested and applied. All the models are open source and available on the OMS Javaforge project site (http://oms.javaforge.com).

4.1. OMS3/J2K-S Watershed Model

The OMS3/J2K-S watershed model is based on the European J2K-S model (Krause, 2002; Krause et al., 2006). OMS3/J2K-S (Ascough II et al., 2010) is a modular, spatially distributed hydrological system which implements hydrological processes as encapsulated process components. OMS3/J2K-S operates at various temporal and spatial aggregation levels throughout the watershed. For example, runoff is generated at the HRU level with subsequent calculation of runoff concentration processes (through a lateral routing scheme) and flood routing in the stream channel network. Research is currently being performed within the USDA-Agricultural Research (ARS) to develop, improve, and evaluate this model for selected watersheds within the Conservation Effects Assessment Program (CEAP) initiative.

4.2 NRCS Water Supply Forecasting

Seasonal water supply forecasts are an important function of the NRCS National Water and Climate center (NWCC). The forecasts are produced in cooperation with the National

Weather Service, are developed for hundreds of basins in the western United States, and are used by the agricultural community to optimize water use during the irrigation season. The NWCC has historically developed seasonal, regression-equation based forecasts of estimated seasonal stream flow volume. To address the agricultural community's requests for more information on the volume and timing of water availability and to improve forecast accuracy, the NWCC is now developing the capability to use distributed-parameter, physical process hydrologic models to provide forecasts of daily, weekly, as well as seasonal stream flow using an Ensemble Streamflow Prediction (ESP) methodology. The primary objective of this model-based forecast effort is the development and implementation of an OMS3 PRMS-based model family (and associated methods and tools) to enable the provision of timely, improved, and more frequently updated water supply forecasts.

4.3. Conservation Delivery Streamlining Initiative (CDSI)

USDA- NRCS field consultants currently use an array of analytical tools when providing technical assistance, including the Web Soil Survey, Revised Universal Soil Loss Equation, Wind Erosion Equation, Nutrition Balance Analyzer, Soil Quality Index, Pesticide Screening Tool, Phosphorus Index, Energy Estimators, Cost and Returns Estimator, among several others. They will increasingly use more comprehensive process-based modeling tools, such as the Agricultural Policy Extender (APEX) and the Agricultural Nonpoint Source (annAGNPS) models, or at least the technology contained within them, for on-farm system analysis. Unfortunately, each model comes with its own data provisioning requirements, unique user interface, and processing requirements. Currently, field consultants have hit a wall of complexity and resource constraints, and the existing tools are not used to their full potential (Carlson, 2010).

To remedy this problem, the NRCS has initiated a Conservation Delivery Streamlining Initiative (CDSI) to integrate technology components with the workflows of the field consultant. CDSI will provide the framework and common user interface for the field consultant. The USDA-ARS is leading the development of models in OMS3 to integrate science components across multiple models and tools into various model bases, one of which will integrate with the CDSI framework. The purpose of the OMS-CDSI model base is to deliver science deployed as services available to the CDSI workflow.

4.4 JGrass – Horton Machine and NewAge

Recently, OMS3 was used as the modeling engine for JGrass through a collaboration effort between HydroloGIS, Colorado State University, and the University of Trento, Italy. JGrass is an open source front end for the Grass Geographic Information System. It is based on the uDig framework, which is maintained by HydroloGIS and CUDAM (University Centre for Advanced Studies on Hydrogeological Risk in Mountain Areas, University of Trento, Italy). Grass has a long history, emerging from a federal agency-sponsored project into an open source project supported by a community of researchers, consultants and the Open Source Geospatial Foundation (OSGeo). The OMS3 runtime engine was brought into JGrass to provide for the execution of any OMS3 model within this environment. The core components of the Horton Machine, a hydro-geomorphological tool box, were implemented under OMS3. Example Horton Machine components include a peak flow discharge model, watershed stream network delineation model, and an energy balance model. All components were integrated with core GIS processing through "Sprint"-like development sessions. The fact that data type handling for modeling components in OMS3 is open to Open Geospatial Consortium (OGC) types for raster and vector data allowed a straightforward and efficient implementation. In addition to the Horton Machine, the NewAge semi-distributed hydrological model was also ported into OMS3. NewAge consists of new conceptualizations where geographic features and modeling are mixed in an innovative way. Besides modeling the whole hydrological cycle, NewAge is also able to take into account human artifacts like reservoirs, intake and outtakes.

5. CONCLUSION

OMS3 is a lightweight and non-invasive modeling framework for component-based model and simulation development on multiple platforms. The simplified structure for components allows rapid implementation of new models and an easier adaptation of existing models and components. Recent studies have shown that this type of approach leads to models with less overhead and a more intuitive design. By fully embracing language annotations for modeling metadata in favor of traditional APIs, OMS3-based models keep their identity outside of use within the modeling framework. Annotations also enable multi-purposing of components which is hard to accomplish with a traditional API design. In OMS3, annotations provide for component connectivity, data transformation, unit conversion, and automated documentation model generation (Docbook5+). Agencies such as USDA-NRCS use those annotations for creating traceable simulation audits based on secure hash algorithm (SHA 256).

OMS3 also introduces implicit multithreading and 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 calibration, sensitivity analysis, etc. This type of "programmable" configuration that eliminated core programming language "noise" turned out to be an extremely useful tool for current projects to support aspects such as automated, distributed setup of multiple batch models runs, integration into the open source GIS JGrass console, or IDE integration to name a few.

Ongoing work with several modeling applications in OMS3 has shown the efficacy and efficiency of a non-invasive framework approach. The OMS3 development goal is to provide features to the modeler to make it easy to create contemporary, inter-operable, scalable and lightweight models that take full advantage of computing resources, data stores, and infrastructure opportunities while keeping things simple and intuitive. Additional information about OMS3 and associated models applications can be found on http://oms.javaforge.com.

REFERENCES

Ascough II, J., David, O., Krause, P., Green, T., Heathman, G., Kralisch, S., and L. Ahuja, A component-based distributed watershed model for the USDA CEAP Watershed Assessment Study, paper presented at Farming Systems Design 2009, Monterey, California USA, 23-26 August, 2009.

Ascough II J.C., O. David, P. Krause, M. Fink, S. Kralisch, H. Kipka, and M. Wetzel Integrated agricultural system modeling using OMS3: Component driven stream flow and nutrient dynamics simulations, 2010 International Congress on Environmental Modelling and Software Modelling for Environment's Sake, Fifth Biennial Meeting, Ottawa, Canada, 5-8 July 2010.

Carlson, J.R., D.B. Butler, O. David, K.W. Rojas, J.C. Ascough II, G.H. Leavesley, W.R. Oaks, L.C. Price, and L.R. Ahuja, Conservation ontology and knowledge base to support delivery of technical assistance to agricultural producers in the United States, 2010 International Congress on Environmental Modelling and Software Modelling for Environment's Sake, Fifth Biennial Meeting, Ottawa, Canada, 5-8 July 2010.

David, O., S.L. Markstrom, K.W. Rojas, L.R. Ahuja, and I.W. Schneider, The Object Modeling System. In: Agricultural System Models in Field Research and Technology Transfer, L.R. Ahuja, L. Ma, and T.A. Howell (Eds.). Lewis Publishers, Boca Raton, FL, 317–330, 2002.

Leavesley, G.H., S.L. Markstrom, and R.J. Viger, USGS Modular Modeling System (MMS) - Precipitation-Runoff Modeling System (PRMS). In: V.P. Singh and D.K. Frevert (Eds.), Watershed Models. CRC Press, Boca Raton, FL, pp. 159-177, 2006.

Rizzoli, A.E., G.H. Leavesley, J.C. Ascough II, R.M. Argent, I.N. Athanasiadis, V.C. Brilhante, F.H. Claeys, O. David, M. Donatelli, P. Gijsbers, D. Havlik, A. Kassahun, P. Krause, N.W. Quinn, H. Scholten, R.S. Sojda, and F. Villa, Chap. 7: Integrated modelling frameworks for environmental assessment and decision support. In: A.J. Jakeman, A.A. Voinov, A.E. Rizzoli, and S.H. Chen (Eds.), Environmental Modelling

and Software and Decision Support – Developments in Integrated Environmental Assessment (DIEA), Vol. 3, pp. 101-118. Elsevier, The Netherlands, 2008.