International Environmental Modelling and Software Society (iEMSs)
2012 International Congress on Environmental Modelling and Software
Managing Resources of a Limited Planet, Sixth Biennial Meeting, Leipzig, Germany
R. Seppelt, A.A. Voinov, S. Lange, D. Bankamp (Eds.)
http://www.iemss.org/society/index.php/iemss-2012-proceedings

Domain Specific Languages for Modeling and Simulation: Use Case OMS3

Olaf David[23], Wesley Lloyd[23], James C. Ascough II[1], Timothy R. Green[1], Kevin Olson[3], George H. Leavesley[3], Jack Carlson[3]

[1] USDA-ARS-NPA, Agricultural Systems Research Unit
2150 Centre Ave., Bldg. D, Suite 200, Fort Collins, Colorado 80526 USA
[2] Colorado State University, Dept. of Computer Science,
Fort Collins, Colorado 80523 USA
[3] Colorado State University, Dept. of Civil and Environmental Engineering,
Fort Collins, Colorado 80523 USA
odavid@colostate.edu

Abstract: A domain-specific language (DSL) is usually a concise, declarative language that strongly emphasizes a particular problem domain. DSL methods and implementations in general are widely prototyped and applied for creating elegant ways to express properties, relationships, and behavior of real-world models. This paper introduces DSL use for creating models and simulations within the Object Modeling System 3 (OMS3) modeling framework. In OMS3, various DSL concepts have been adopted to complement general-purpose modeling languages such as Java, FORTRAN, and C. Design patterns, such as the builder pattern, have been adopted through the DSL to support the setup of complex simulations for various applications such as Ensemble Streamflow Prediction (ESP), model calibration (Luca), configuration of model efficiency calculation, or output visualization. Experience has shown that a well-balanced adoption of DSL principles, complemented with general-purpose language elements, enhances the efficiency of model application while reducing development effort for model developers and users. In addition, adoption of DSL principles provides a viable alternative to complex graphical user interface development.

***Keywords**:* Component-based modeling, Domain Specific Languages, Modeling frameworks, Simulations; Calibration.

1. INTRODUCTION

A domain-specific language (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, 1997; Deursen et al., 2000). DSLs provide concise and strongly focused methods for data management and configuration setup of a program. They let users write simple business rules for a particular task. DSL designs are geared towards easy readability and DSL constructs often read like natural languages. The declarative DSL approach is motivated by the desire to allow coding without actually promoting it. The Object Modeling System 3 (OMS3) takes advantage of the DSL “builder” design pattern as provided by the Groovy programming language and DSL extension (Dearle, 2010). OMS3 defines simulation DSLs for various purposes. Basic model application, parameter estimation, sensitivity analysis, or ensemble streamflow prediction are a few examples of DSL variants in OMS3. These simulation DSLs allow the creation and

configuration of run-time simulations for OMS3; however, the simulation DSL is not inherently bound to this framework.

2. SIMULATION DSLs

What constitutes a simulation in the OMS3 context? A simulation DSL declares 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. A simulation DSL reads like structured text, such as XML or JSON. However, it is still executable code that might contain general purpose language constructs. The following sections present three different simulation types: (i) basic simulation, (ii) ensemble stream flow prediction, and (iii) model calibration using Luca.

2.1 Basic Simulation

Listing 1 shows a typical simulation DSL file (*.sim) for OMS3 which is directly executable using the OMS3 runtime. 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 in the Java/Groovy programming language. The ability to parse and process a simulation DSL is part of the underlying Groovy runtime in OMS3. 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 1, as used within the Thornthwaite monthly water balance model presented in Lloyd et al. (2011a), provides underlying knowledge about component connectivity. Model components are specified within the `components{}` section in Listing 1. Component connection is represented via `connect{}` statements, indicating data flow from source to target.

Listing 1
Simulation DSL example in OMS3 for a 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")
        }
    }
}
```

The `connect{}` section lists line-by-line data flow handled by the system. For example, the line `'climate.temp' 'soil.temp'` in Listing 1 connects the '`temp`' output field of the '`climate`' component with the '`temp`' input field of the '`soil`' component. This is the only information required to define and establish data flow between components, which internally have those fields annotated using `@In` and `@Out` annotations (see http://oms.javaforge.com for additional description of OMS3 annotations). The entries in each line of the `connect{}` section 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. OMS3 performs

conversions of field types if both the source and target field types are different. However, a conversion service is offered by OMS3 when two components are connected using fields which are incompatible otherwise. For example, one component providing an Open GIS Consortium (OGC) simple feature collection as output can feed into another one requiring “well known text” (WKT, ASCII encoding of geometries) input with no problems if a conversion class or service exists and is offered via the service provider interface. The same mechanism is also used for unit conversion or alignment of temporal and spatial scales between fields.

2.2 Ensemble Streamflow Prediction DSL

ESP is a simulation type for Ensemble Streamflow Prediction. It implements a modified version of the National Weather Service’s ESP procedure (Day, 1985). ESP uses historic or synthesized meteorological data as an analogue for the future. These time series are used as model input to simulate future conditions. The typical application of ESP is streamflow forecasting. The initial hydrological conditions of a watershed, for the start of a forecast period, are assumed to be those simulated by the model for that point in time. Typically, multiple hydrographs are simulated from this point in time forward, one for each year of available historic data. For each simulated hydrograph, the model is re-initialized using the watershed conditions at the starting point of the forecast period. The forecast period can vary from a few days to an entire year. A frequency analysis is then performed on the peaks and/or volumes of the simulated hydrograph traces to evaluate their probabilities of exceedance.

Listing 2 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 simulation’s top level element defines it as an ESP DSL. It uses elements from a basic simulation, such as `model{}`, `outputstrategy{}`, or `resource{}`. In addition, ESP specific simulation parameters are added for defining the forecasting period and the historical years to be used.

Listing 2
Simulation DSL example in OMS3 for an Ensemble Streamflow Prediction.

```
esp(name:"TetonEsp") {

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

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


    // define model
    model(classname:"model.PrmsDdJhXyz") {
       // parameter
       parameter (file:"$oms_prj/data/teton/teton18_xyz_test.csv") {
          inputFile  "$oms_prj/data/teton/teton_OMS_Data.csv"
          outFile    "out.csv"
          sumFile    "basinsum.csv"
          out        "summary.txt"

          startTime "2003-10-01"
          endTime   "2005-04-30"
       }
    }
```

```
    // the number of forecast days after the end of the simulation period
    //forecast_days 15
    // as an alternative you can pecify the end of the forecast period
    // as date.
    // choose one of th two options,

     forecast_end  "2005-08-31"

    // historical years for to be used for traces
    // years are inclusive
    first_year 1981
    last_year  2004

    analysis(title:"Trace analysis") {

       // relative path name, last output
       esptraces(title:"teton", dir:"%last", var:"basin_cfs")
    }
 }
```

2.3 'Luca' Model Calibration

Luca (Hay, 2006) is a multiple-objective, stepwise, automated procedure for model calibration. The calibration procedure uses the Shuffled Complex Evolution (SCE) global search algorithm to calibrate any OMS3 model. Luca defines an OMS simulation type for building and performing a procedure to calibrate parameters for a (hydrological) model. It integrates the following components: (i) multiple-objective step-wise calibration, (ii) SCE global-search parameter optimization, and (iii) OMS model interoperability.

The PRMS model has been configured for parameter estimation using the USGS Luca parameter estimation method (Hay, 2006). 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.

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)
        }
    }
}
```

An OMS3 Luca simulation contains the usual elements for defining the model, model parameter, and other resources. The main Luca elements are the round and step definitions, where each step declares the parameter bounds with a calibration strategy as well as the objective functions to be applied for this step.

3. DISCUSSION

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.

4. SUMMARY

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).

REFERENCES

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., and 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.

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

Deursen van A., Klint, P., Visser, J., Domain-Specific Languages: An Annotated Bibliography. ACM SIGPLAN Notices, 35(6):26-36, June 2000.

Donatelli, M., Rizzoli, A., 2007. A design for framework-independent model components of biophysical systems. In: Hatfield, J., Donatelli, M., Rizzoli, A. (Eds.), Farming Systems Design 2007: An International Symposium on Methodologies for Integrated Analysis of Farm Production Systems, Catania, Sicily, Italy, Vol.2, pp. 208-209.

Donatelli, M., Rizzoli, A., 2007. A design for framework-independent model components of biophysical systems. In: Hatfield, J., Donatelli, M., Rizzoli, A. (Eds.), Farming Systems Design 2007: An International Symposium on Methodologies for Integrated Analysis of Farm Production Systems, Catania, Sicily, Italy, Vol.2, pp. 208-209.

Hay, L.E., Umemoto, M., 2006. Multiple-objective stepwise calibration using Luca: U.S. Geological Survey Open-File Report 2006-1323, 25p.

Leavesley, G.H., Markstrom, S.L., Viger, R.J., 2006. USGS Modular Modeling System (MMS) - Precipitation-Runoff Modeling System (PRMS). In: Singh, V.P. and 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. 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.

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: Environmental Modelling and Software and Decision Support – Developments in Integrated Environmental Assessment (DIEA), Vol. 3, A.J. Jakeman, A.A. Voinov, A.E. Rizzoli, and S.H. Chen (Eds.), pp. 101-118. Elsevier, The Netherlands.

Tulach, J., 2008. Practical API Design: Confessions of a Java Framework Architect. Springer, New York, NY.