International Congress on Environmental Modelling and Software

2nd International Congress on Environmental Modelling and Software - Osnabrück, Germany - June 2004

Jul 1st, 12:00 AM

Metadata and Modeling Frameworks: The Object Modeling System Example

Olaf David

I. W. Schneider

George Leavesley

Follow this and additional works at: https://scholarsarchive.byu.edu/iemssconference

Metadata and Modeling Frameworks:
The Object Modeling System Example

O. David[a], **I. W. Schneider**[b], **and G. H. Leavesley**[c]

[a] *Colorado State University, Fort Collins, CO, U.S.A.*

[b] *USDA Agricultural Research Service, Great Plains System Research, Fort Collins, CO, U.S.A*

[c] *US Geological Survey, Denver CO, U.S.A*

Abstract: The main motivation for the usage of modeling frameworks for environmental simulation software is to manage and simplify the interoperability of (loosely) coupled simulation components. Conventional approaches in collaboration are using an Application Programming Interface (API). Recent developments in simulation frameworks focus on introspecting architectures for simulation components, where components become passively explored and integrated in to the framework. Such solutions seem to be more flexible to support the framework evolution because components are less tight to a specific framework API. The Object Modeling System (OMS) is an introspecting simulation framework, which uses metadata in annotated components such as (i) spatial and temporal constraint specification, (ii) data annotation for variables and parameters to specify simulation related data like runtime constraints for range validation, unit conversion, or automated testing. The OMS utilizes metadata annotation (i) at model construction time to support proper spatial and temporal model assembly (ii) and at model runtime to support proper data linkage. This paper will emphasize on metadata access to support model linkage to simplify the development of simulation components for environmental scientists and will give application examples based on the Object Modeling System

Keywords: Modeling; Framework; Metadata; Components; Java

1. INTRODUCTION

Meta data in the context of modelling and simulation traditionally represents knowledge about the application of simulation models. For example, the proper setup of parameter sets to run a model is only possible with the correct knowledge about parameter range constraints, physical units, etc. Such information usually resides in documentation or is a part of the user interface.

For modelling frameworks meta data has further meaning and importance. Meta data can be used to support the process of model (i) construction, (ii) testing and (iii) application.

This paper introduces the Object Modeling System with respect to its underlying component concept.

2. THE OBJECT MODELING SYSTEM

The Object Modeling System (OMS) is a Java-based modeling framework that facilitates simulation model development, evaluation and deployment. OMS models are treated as hierarchical assembled components representing building blocks. Components are independent and reusable software units implementing processing objects for simulation models. In general, OMS consists of a library of science, control, and database modules and a means to assemble the selected modules into an application-specific modeling package. The framework is supported by data dictionary, data retrieval, GIS, graphical visualization, and statistical analysis utility modules. Current challenges in natural resource management have created demand for integrated, flexible, and easily parameterized hydrologic models. Most of these monolithic models are not

modular, thus modifications (e.g., changes in process representation) require considerable time, effort, and expense.

OMS addresses these needs by using an object-oriented, component based approach for model development to:

- Reduce duplication of modeling efforts

- Improve the quality and currency of model code

- Make natural resource models much easier to build, access, understand and use;

- Facilitate long-term maintainability of existing and new natural resource models;

- Lead to greater consistency of modeling for particular problems and scales;

- Enhance response and delivery times in scientific modeling projects;

- Ensure creditability and security of model implementations; and

- Function on any major computing platform.

2.1 OMS Components

Components are the basic building blocks of a model. They represent usually a unique concept in a model like a physical process, a management practice, a remote data input, etc. Components can be fully understood just by exploring the component metadata. Therefore OMS defines a comprehensive set of metadata which should be bundled with the component.

Meta data attachment to components is a technique which was introduced to the Java Programming language by means of formal documentation annotation support. Classes, fields, and methods can have documentation header, which are comments according to the Java Language specification but contain useful additional info for the java documentation tool 'javadoc' [???]. Javadoc annotations are primarily used to generate API documentation (HTML and other formats) out of java sources. Such an approach helps managing a consistent source code documentation by keeping code and code documentation in the same file.

The javadoc style annotation of java language elements was also designed to be extensible. It is legal to define custom meta data annotations for java classes. Parsing tools for java such as javadoc, 'qdox', or the Netbeans Java Source API provide comprehensive support to lookup and manage documentation annotations.

The javadoc approach was used to support the meta data annotation of components. The meta data can be categorized in two conceptual groups

- *Meta data to document the component in an informal way.* A component contains for example metadata about its creation time, the author, the organization, or references to publications related to the code.

1. *Formal meta data to annotate a component for model integration or testing.* Such meta data is required by OMS to support the proper component integration into a model at design time. Such meta data enables OMS to deal with formal processing requirements.

There are two main levels of meta data annotation for an OMS component.

1. Component meta data annotates the entire component. It contains information about its overall purpose, authorship, version control, literature references, temporal or spatial scale etc.

2. Attribute meta data annotates data requirements of the component per public attribute. Such metadata captures additional information about the attribute, such as physical units, data constraints, data flow, default values, etc.

Figure 1 shows the component meta data for a forage component. OMS related meta info is tagged with a '@oms.' at the beginning to avoid 'namespace' conflicts with potential other tag definitions.

```
/**
 * Forage.
 *
 * This module estimates daily
 * plant growth for five functional
 * groups in rangelands.Daily growth is
 * driven by average daily temperature
 * relative to the optimum and base
 * temperatures for cool season grasses
 * (C3), warm season grasses (C4),
 * legumes, shrubs, and weeds. The module
 * is adapted from GPFARM.
 *
 * @oms.author  Allan Andales
 * @oms.version $Id: Forage.java 294 2004-
 *              05-24 21:11:15Z david $
 * @oms.created April 1, 2004, 2:39 PM
 * @oms.name    Forage Component
 * @oms.category  Plants
 * @jni.files RunForage.f90 PlantMod.f90
 *            ForSite.f90 Forage.f90
 */
public class Forage extends …
```

Figure 1: 'Forage' Component Definition

Component meta data in OMS affects several phases in component integration and model development which are mentioned briefly.

2.1.1 Component Documentation.

Meta data annotations of components are used in OMS to generate documentation. Unlike the default javadoc tool, OMS generates XML Docbook (Welsh, 1999) to document components. Docbook was chosen as the main document format because of its flexibility to transform component documentation into other formats more easily. Formal and informal meta data is used for documentation generation. Resulting component XML specification can be processed with any tool supporting docbook or they can be transformed in other XML representations. The inclusion of component specifications into an XML component library data base is one of many examples.

2.1.2 Component Testing

Formal meta data is being used to support automated testing. OMS facilitates black box testing of components. Each component attribute can be annotated with test related information for its input and output. OMS consumes these annotations in a testing phase to test the component by

1. Generating input data according to given test annotations. Test data will be generated using several available random or sequence generators.

2. Checking output data according to given test annotations. Output data is usually expected to be in a certain range.

2.1.3 Component Integration/Model Design Time

Model design refers to the process of model construction based on model building blocks, known as components. A component needs to tell the modelling framework if it fits into model, which the model developer is going to built. The component has to offer information about application scales, data dependencies and other information, which are important for a consistent integration of the component into the model. If for example data requirements cannot be resolved, the modelling framework will guide the user to select alternative or additional components for the model.

2.1.4. Component Execution/Model Runtime

At model run time meta data availability about the models data is useful as well. If for example the components data has additional meta data about physical units of the components input data, unit conversion can be performed dynamically. Run time checks of model state variables can be specified with metadata.

Metadata affects the communication between the modeling framework and the component. Components want to see data handled by the modelling system under a certain name, in a certain unit, etc. Components interact with the framework to get the data in the right structure and format.

2.2 API vs. Introspection

There are two major communication principles of components. In an *API* based communication the component interacts with the system by using a well known API. This is the common and traditional method for interoperability. Component developers are using framework API calls to get data in a requested format. Such an approach can be used in any kind of programming language. A component using a framework API is then tightly coupled, technically and conceptually. Such a component can hardly be reused in another context.

The communication between the modeling framework and the component can also be based on a more flexible schema for interaction, called introspection. Here, the modeling framework is capable to explore the structure and content of a component with respect to data and metadata. This approach works only on architectures, which do have building support for such introspection and exploration. Java and C# are the examples for such architectures.

2.2.1 OMS Attributes

An attribute is a basic data type in OMS. It extends the concept of a typed data element with its ability for metadata annotation and management.

Attributes are representing model state variables or model parameter. They are component input or output. A component interacts with attributes only. The usage of attributes in a model determines, if such an object is considered to be a

parameter or a variable. There is no such terminology like a parameter or state variable at the component level in OMS.

OMS uses introspection to deal with meta data for design and run time purposes. All data and meta data belonging to the attribute called '*nitstress*' is shown in Figure 2. The object '*nitstress*' represents a stress factor for nitrogen in a OMS component called 'Forage'. There is the data declaration given as object of type 'Attribute.Double'. The does have such classes for all reqired basic data types (floating point, fixed point, boolean). These are actually Java interfaces and no classes.

The data declaration is preceded by a section containing all the metadata about the attribute. This section is enclosed in '/** .. */'. It is actually a comment according to the Java Language specification as mentioned before.

```
/** Nitrogen stress factor.
 * @oms.name_de      Sickstoff     Stress
                     Faktor
 * @oms.unit         kg/kg
 * @oms.access       read
 * @oms.constraint   0.0..1.0
 * @oms.default      1.0
 */
private Attribute.Double nitstress;
```

Figure 2: Attribute Definition

The metadata annotation section starts with the name of the attribute in English. It is followed by OMS annotation tags all starting with "@oms.". There are several tags recognized by OMS:

- `@oms.name_<loc>`
  This optional tag contains the localized name. If the design or run time environment is localized for a certain lanuage, this name gets used.

- `@oms.unit`
  This optional tag refers to the unit of the attribute. It uses the UCAR units notation specification (Emmerson et al. 2001) .

- `@oms.access`
  This required tag specifies the data flow. Either the component reads or writes this attribute.

- `@oms.constraint`
  This optional tag controls additional value constraints of an attribute. In the example above nitstress is a factor wich

has a valis range of 0 to 1. Any other value is considered to be incorrect according to the definition of nitstress. Constrains become handy for run time verification of a model. If a value violates the constraint specification the system will notify the user and might indicate a problem with input data. Other valid values for the constraint tag are: '<=0' or '>-273.15'

- `@oms.default`
  This optional tag contains the default value for this attribute.

There are other metadata annotation tags not used in this example.

- `@oms.dim`
  If the attribute is an Array (eg. Attribute.DoubleArray) this tag points to a numerical constant or to another attribute representing the dimension of that array.

- `@oms.test`
  This specific tag supports the automated testing of components. The content of this tag is similar to the constraint tag. OMS perform a component test, which generates data according to the requirements specified in this tag. In the given example a useful test related to 'nitstress' would be '@oms.test 0.0..1.0'. A randomizer would generate input data in this range. The component could then be verified automatically.

Attribute types such as 'Attribute.Double' are realized as Java interfaces as supposed to classes. There are several internal classes in OMS which are implementing these interfaces. The system choses the right implementation for the given set of metadata and will pass the system generated attribute to the component. It uses reflectively the property getter and setter methods to access the attribute. Setter and Getter are implemented according to the Java Beans Standard.

5. CONCLUSIONS

Meta data is far more than a semantic add-on for modelling frameworks. Properly designed, a strong support for meta data helps modelling frameworks to support model developer and user

in the process of model construction and application.

The Object Modeling System OMS uses annotations of components to capture information about the models data in addition to data names and types. Such annotations provide the benefit of keeping data and metadata close together and allow for the application of different tools for executing testing, and documenting.

The metadata annotations in OMS were used for the development of model components out of the Root Zone Water Quality Model (RZWQM), the Precipitation Runoff Modeling System (PRMS).

The future Java annotation support specified in the JSR 175 and implemented in JDK 1.5 offers the a more straightforward implementation of metadata management in the Java virtual machine, which is expected to be used for OMS.

6. ACKNOWLEDGEMENTS

7. REFERENCES

David O., S.L. Markstrom, K.W. Rojas, L.R. Ahuja, and I.W. Schneider (2002). The Object Modeling System, In: Agricultural System Models in Field Research and Technology Transfer, L. Ahuja, L. Ma, T.A. Howell, Eds., Lewis Publishers, CRC Press LLC, 2002: 317—331.

Leavesley, G. H., R. W. Lichty, et al. (1983). Precipitation-runoff modeling system--User's manual, U.S. Geological Survey: 207.

Leavesley, G. H., G. E. Grant, et al. (1996). A modular modeling approach to watershed analysis and ecosystem management. Watershed '96 A National Conference on Watershed Management, U.S. Government Printing Office.

Bloch J. : A Program Annotation Facility for the Java Programming Language. JSR-175 Public Draft Specification. [http://www.jcp.org/en/jsr/detail?id=175]

Emmerson S. el al. (2001): Units Specification. JSR-108 Public Draft Specification. [http://www.jcp.org/en/jsr/detail?id=108]

Walsh, N., Muellner L. (1999). DocBook: The Definitive Guide. O'Reilly & Associates, Oct 1999.

JavaDoc, Core JavaDoc Tool [http://java.sun.com/j2se/javadoc/]

Arnold,K.; J. Gosling, D. Holmes (1998): The Java Programming Language, Addison Wesley, 1998