LOGY TRANSFER

luene and Hexahy-

CHAPTER 15

The Object Modeling System

Olaf David, Steven L. Markstrom, Kenneth W. Rojas, Lajpat R. Ahuja and Ian W. Schneider

CONTENTS

INTRODUCTION

The problems facing both users and developers of natural resource models are becoming much more complex. Understanding human management issues such as farming practices, erosion control, pesticide and fertilizer application, reservoir management and habitat restoration become compounded when viewed within the physical, hydrological, chemical and biological responses of the natural world. Computer simulations for conceptualization, prediction and management of agricultural areas, watersheds, water supply and environmentally sensitive sites are likewise becoming more complex. The interdisciplinary nature of these problems usually requires taking into account a significant number of different models, alternatives, data sources and domain experts.

Much domain-specific disciplinary expertise has been developed and captured in the form of computer simulation models. These models usually represent the efforts of an individual or a small

1-56670-0563/02/$0.00+$1.50

group of scientists and are consequently very focused in their scope. Several of these models must often be applied within the context of a single project application. Although these models may share much in common, there may be conflicting data, scales. methodologies, file formats, or even computer hardware and software requirements for these models. To overcome these limitations they are often rebuilt to meet an appropriate system delineation.

On the other hand, some large, monolithic process models have been designed and implemented to cover a range of simulation alternatives (Abel, 1994). Many of these models are closed, stand-alone systems. They require fixed input/output data format with no capability to interface with other models. Many of these models are still operating in batch mode. Modifications and extensions to a model are often handled as additions made to the main body of code developed years ago. Such a tightly wrapped structure limits this integration with other models and makes their update and maintenance difficult.

An important effort in environmental simulation model design, therefore, is to enhance modularity, reusability and interoperability of both science and auxiliary components. Leavesley et al. (2002) pointed out that models applied for different systems or sub-systems required different levels of detail and comprehensiveness, which are driven by problem objectives, data constrains and spatial and temporal scales of application.

Reusability can be increased by establishing standard simulation module libraries. These libraries are comprised of components of simulation models and provide basic building blocks for a number of similar applications. They are designed to allow interoperability, which is essential for the incorporation of various scientific disciplines. Module libraries have been successfully applied in several domains such as manufacturing systems, transport and other systems (Top et al., 1997; Breunese et al.. 1998; Praehofer, 1996). An advanced modular modeling framework based on the creation of a library of science and utility modules offers an exciting possibility for developing customized agricultural system models in the 21st Century. These models will use the best science available for their purpose and will be easy to update and maintain. The library may lead to standardization of science, tools and interfaces and will serve as a reference and coordination mechanism for model developers and future research.

This chapter gives an overview of the efforts made in adopting modular modeling principles for the construction and application of natural systems models through the Object Modeling System (OMS) framework. The chapter presents:

1. OMS project background and objectives
2. Basic OMS principles
3. OMS architecture and implementation
4. Results of a model application example

Background and Objectives of the OMS Project

The OMS Project is an interagency project between the USDA-ARS (Agricultural Research Service), USGS (U.S. Geological Service) and USDA-NRCS (Natural Resource Conservation Service). The past experiences of the agencies indicated that the development of comprehensive, multidisciplinary simulation models was a very expensive process ($15 to $30 million per model). Many development activities were duplicated among different modeling projects; for example, the hydrologic components in crop models, water quality models and erosion models. The duplicated components generally used different levels of detail and time scales, whereby they did not give the same results in hydrology and, thus, other outputs. Furthermore, the maintenance of several large, monolithic models has been a problem. These considerations led to the initiation of the OMS Project.

The overall project goal is the development of OMS, an integrated modeling framework, which allows integration of models, founded on a standard library of components, tools and data. Models developed within OMS will be based on representing different scientific approaches with regard

to components that addres
of the OMS project are t

1. Identify modeling li
simulation models.
2. Formalize the linka
3. Develop generic so
4. Develop the framev

To provide compreh
part of the framework:

1. A module-building
framework (this ad
2. A module reposito
model (types of m
access- and systen
3. A model builder tl
verify data connec
4. A dictionary fram
extended semantic
5. An extensible use
development and
for database mana

Modularity

The fundamental a
models. New scientific
dated by other scientist
development. Unfortur
software constructs, sci
tation there is usually

It is possible that b
gained through better s
into components may r
most notably Zeigler (1
This is related to a mc
DeRemer and Kron (1
system is an essential
individual modules." V
modules whose interac
complexity inherent in
easier to understand c

Modularization has
inherent structure of res
out different degrees
loosely coupled model
conceptualizations of r

to components that address data constrains and spatial/temporal scale of application. The objectives of the OMS project are to:

1. Identify modeling library parts (modules or components) and glean them from existing nonmodular simulation models.
2. Formalize the linkages between these components to support model building.
3. Develop generic software tools to support models and modeling.
4. Develop the framework which supports these objectives.

To provide comprehensive modeling assistance, the following functional components will be part of the framework:

1. A module-building component that will facilitate the integration of existing (legacy) code into the framework (this adaptation support will simplify the technical procedure for module implementation)
2. A module repository that will contain modules that can be readily utilized to assemble a working model (types of modules in the library will include science-, control-, utility-, assessment-, data access- and system modules)
3. A model builder that will assemble modules from the module library into executable models and verify data connectivity and compatibility in scale and comprehensiveness
4. A dictionary framework that will manage extended modeling data type information and provide extended semantics checking for module connectivity verification
5. An extensible user interface that will facilitate an appropriate user interaction for general model development and application (it will be supported by a number of contributing software packages for database management, visualization and model deployment)

BASIC OMS PRINCIPLES

Modularity

The fundamental and underlying approach of OMS is to apply modular design to simulation models. New scientific results are generally developed piecewise; each step is reviewed and validated by other scientists in the community. A similar approach should be taken to model software development. Unfortunately, when formalizing scientific methods and research into operational software constructs, scientists tend to start from a scratch. Moreover, when it comes to implementation there is usually a lack of understanding of appropriate software design.

It is possible that better understanding of the behavior of complex physical systems could be gained through better software practices. Specifically, dissaggregation of large and complex systems into components may reveal inter- and intra- relationships. This has been the interest of many authors, most notably Zeigler (1990), who called this "modeling in the large," and of Cota and Sargent (1992). This is related to a more general software concept known as "programming in the large." Earlier, DeRemer and Kron (1976) pointed out: "... structuring a large collection of modules to form a system is an essentially distinct and different intellectual activity from that of constructing the individual modules." With programming in the large, the emphasis is on partitioning the work into modules whose interactions are precisely specified. Modularity is the overall key to coping with the complexity inherent in large systems. Disaggregating these systems into smaller subsystems that are easier to understand can be achieved on a "divide and conquer" (Pidd and Castro, 1998) strategy.

Modularization has not been a common technique in practical model development, although the inherent structure of resource simulation models support modularization. Leavesley et al. (2002) pointed out different degrees of modularization: fully process modules and models; fully coupled models; loosely coupled models; and uncoupled models for decision support systems, which are all important conceptualizations of model integration and must be design principles for any modeling framework.

Several different modular approaches have been applied to watershed models (Singh, 1995). One of the earliest modular model development efforts was done for the SHE (European Hydrological System) Model (Abbot et al., 1986; Ulgen et al., 1991). The Precipitation Runoff Modeling System (PRMS) (Leavesley et al., 1983) was modularized from a monolithic version and implemented in the Modular Modeling System (MMS) (Leavesley et al., 1996). MMS was an early attempt by the U.S. Geological Survey to support interactive model construction, requiring a specified module structure. Recently, interest in the standardization of model software design has increased. Projects such as APSIM (McCown, 1995) are based on these principles. Model standardization is also gaining momentum within the environmental research and regulatory organizations and agencies (Whelan et al., 1997).

OMS development focuses on the following points to achieve a maximum benefit from modularization:

- Modularization is the key concept to simulation model development. OMS provides an application programming interface (API) for creating new modules. A master library of modules is also available for simulating a variety of water, energy and biochemical processes.
- OMS allows the interactive construction of complex models. OMS graphical user interface (GUI) components facilitate control of module connection, validity, consistency and completeness.
- An extended interface description for modules needs to be developed to specify data semantics. This is important for interdisciplinary module development.
- OMS supports automated module documentation generation. A module communicates with other modules through its public interface. Documentation generated from the module interface specification is sufficient to judge its suitability in a given simulation context.
- OMS potentially supports model scenario management and model customization through module exchangeability. This requires a set of module alternatives stored in a specific module library for that purpose. The library must organize this information and make the model fragments available for reuse.

The validation of module compatibility requires an advanced semantic representation of data objects (variables, parameter, etc.) used to connect modules. This leads to the concept of data dictionaries covering extended modeling related type information such a units, value ranges and description.

OMS Framework

The OMS framework is a domain-specific, reusable architecture with a set of interdependent classes in an object-oriented language. The primary benefits of application frameworks in general are modularity through well-defined and stable interfaces, reusability by using generic components, extensibility through hook methods and inversion of control through a reactive dispatching mechanism. The different kinds of frameworks are distinguished by the ways of adapting the framework: "black-box" and "white-box" frameworks. The black-box framework contains abstract elements, which needs to be specialized in the application. The white-box framework instead consists of already implemented specific components, which needs to be selected and customized in an application. OMS is fundamentally a white-box system as classified by the abstract hook method (Fayad and Schmidt, 1997).

OMS SYSTEM CHARACTERISTICS

The OMS has the following characteristics:

1. OMS models are treated as hierarchical assembled components representing building blocks. Components are independent and reusable software units implementing processing objects for simulation models. They reside in a model library and are categorized into data access components,

science components, c
described in the "OMS
2. OMS is able to integrat
code, components writ
level more easily. This
3. The "knowledge"-back
variables and paramet
application. Dictionari
the component connec
This is described in th
4. OMS is extensible. Ext
tion. Extension packag
dictionary framework a
5. OMS scales from a fu
visualization and anal
Model Views" subsect

The approach being us
modeling. It is an approach
i.e., system components w
coupled in a hierarchical
This conceptual approach
Because objects lack an ex
data flow was applied. An
hierarchical modules.

OMS Modules and API

An OMS module is a p
be complex or a single equ
the module will address. A
data objects, which are m
defining the interaction of
this interface. An OMS m
methods. It consists of the

- The OMSComponent
mentation based on i
- A module encapsulat
objects and other su
intermodule commur
may obtain a data ob
- A module encapsulat
Fortran, or C prograr
- Each OMS module r
  - Init — The init m
the data objects.
  - Run — The run m
objects and compo
the simulation ru
  - Cleanup — The c
resources (i.e., m
method is called

science components, control components, utility components and system components. This is described in the "OMS Modules and API" subsection.
2. OMS is able to integrate legacy code components. Due to automated wrapper generation for legacy code, components written in languages such as Fortran can be embedded into OMS at the function level more easily. This is described in the "Legacy Code Implementation" subsection.
3. The "knowledge"-backbone of OMS is the dictionary framework. It enables OMS to verify state variables and parameters according to scientific nomenclatures during model development and application. Dictionaries are also used to specify parameter sets, model control information and the component connectivity. They are implemented in the Extensible Markup Language (XML). This is described in the "Dictionary Framework" subsection.
4. OMS is extensible. Extension packages exist for different aspects in model development and application. Extension packages are used for visual model assembly, model application, an interface to the dictionary framework and GIS. This is described in the "Modeling System GUI Design" subsection.
5. OMS scales from a full-featured, stand-alone development system with tools for model assembly, visualization and analysis to a runtime Web service environment. This is described in the "OMS Model Views" subsection.

The approach being used for component definition is based on modular, hierarchical system modeling. It is an approach to complex dynamic system modeling where modular building blocks, i.e., system components with a well-defined interface in the form of input and output ports, are coupled in a hierarchical manner to form a complex system (Zeigler, 1990; Praehofer, 1996). This conceptual approach was implemented by using object-oriented programming paradigms. Because objects lack an explicit output interface the concept of input and output ports representing data flow was applied. An application programming interface (API) was designed to implement hierarchical modules.

OMS Modules and API

An OMS module is a piece of software implementing a specific function. This function might be complex or a single equation. The scope and complexity of the module depends on the problem the module will address. An OMS module is implemented as an object-oriented class that defines data objects, which are manipulated by the module's methods. A module has a public interface defining the interaction of the module with its environment. All communications must go through this interface. An OMS module has a single purpose, so it is restricted to a fixed set of interface methods. It consists of the following components:

- The OMSComponent class of the OMS core library provides the API for custom module implementation based on inheritance.
- A module encapsulates modeling objects. These data objects can be input data objects, output data objects and other submodules. The data objects are variables or parameter that are used for intermodule communication. The module can be the creator and owner of the data object or it may obtain a data object from other parts of OMS.
- A module encapsulates modeling logic. OMS provides the tools to implement the code in the Java, Fortran, or C programming language.
- Each OMS module must implement the following interface methods:
  - Init — The init method initializes the modules data objects. It is used to set the initial state of the data objects. This method is called prior to a model simulation run.
  - Run — The run method implements the computational part of a module. It operates on all data objects and components with the results of the module processing. This method is called during the simulation run and implements the process logic.
  - Cleanup — The cleanup method performs finalizing tasks for the module. It will release system resources (i.e., memory and sockets). It can also be used for writing summary reports. This method is called past model simulation run.

This OMS module definition API is implemented in the Java programming language (Arnold et al., 2000).

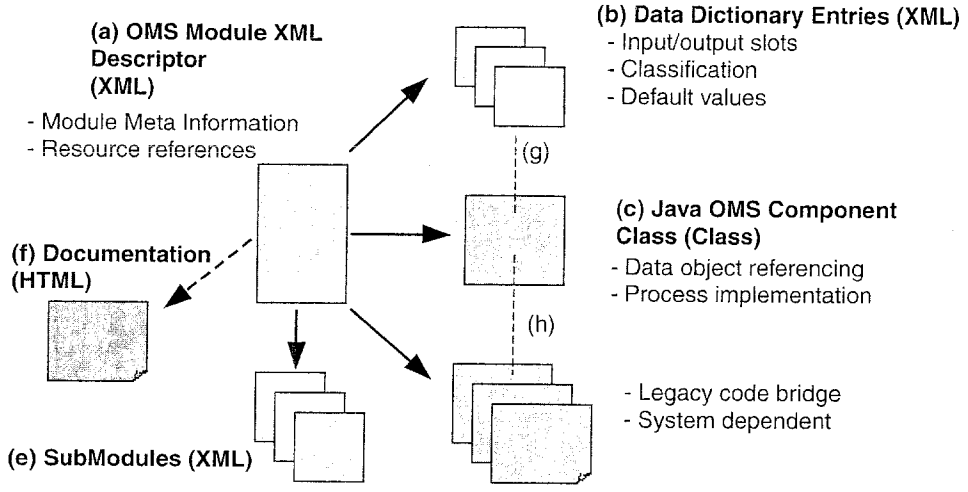

Figure 15.1 Module components and relations in OMS.

From an OMS framework point of view, a module consists of several components important for model development. The two main parts of a module include (see Figure 15.1):

- **Module Interface Specification** — This is represented by an XML descriptor (Figure 15.1a), the public interface is exposed. This includes the input and output data as specified by the data dictionary (Figure 15.1b), a reference to the module implementation component (Figure 15.1c) and sub-module descriptors (Figure 15.1e).
- **Module Implementation** — The implementation of the Java OMSComponent class (Figure 15.1c) and the native libraries (Figure 15.1d) if there is a legacy code attached to the module.

This design has several implications:

1. Module connectivity can be validated by using its interface only. Because modules are referencing data dictionary entries (e.g., parameter or variables), two modules can be connected if both are referencing the same entry. In addition, an extended "type" verification of data objects allows a more semantic type verification of proper data references among modules extending a simple "data type-based" verification (Figure 15.1g).
2. Because the module interface is expressed in XML, modules can be implemented in different programming languages. The design also takes architecture-dependent variations of resources belonging to the model into account. Native libraries are referenced for different operating systems and architectures (Figure 15.1h).
3. OMS uses automated documentation generation for the module interfaces based on XML style sheets into HTML (Figure 15.1f).

In summary, the benefit for an XML/Java combined description of modules results in sophisticated system handling of this module for verification, architecture adaptation and documentation based on a common resource.

Module data objects reside as attributes of each module. A module may encapsulate a number of data objects depending on its complexity. Several factors need to be taken into considerations when adding data objects to a module:

- Data objects may be
  publishing these data
- Public data objects ca
  a previously publishe
  the modeling framew

```
                                                    //
                                                    pu
Module Declaration                                  ..
                                                    ON
Declared objects.                                   ON
Initialized with data                               ON
generated by                                        ON
OMS system
calls based on                                      ON
data dictionary content                             ON
                                                    ..
                                                    }
```

Figure 15.2 Data object dec

The declaration of dat
based on OMS API system
in Figure 15.4).

Legacy Code Impleme

Wrapping is a techniqu
architecture. Mapping inte
the access to this legacy c
tools, which generate glu
advantages:

- Keeping and reusing
- Moving step by step
- Accelerated introduc
- "Separation of conce

A module contains a se
ponent. A module may imp
run( ) and **cleanup**( ) for
specific editors for setting
menting a kind of preproc
OMS uses a sophistic
OMS framework tools su
Figure 15.3 shows how m
variables and parameters

Dictionary Framework

The OMS dictionary
model data, model param
phase, it provides and ma

- Data objects may be declared either private or public. Accessibility and visibility is realized by publishing these data objects to the OMS framework.
- Public data objects can be shared among different modules. In this case a module is referencing a previously published data object. The module is responsible for requesting the data object from the modeling framework. If a data object is not shared, it is local only to the module.

```
                    // Declares basin and HRU physical parameters.
Module Declaration  public class basinprms extends OMSComponent  {
                    ...
Declared objects.   OMSDouble o_basin_area = getOMSDouble ("basin_are");
Initialized with data  OMSDoubleArray o_hru_area = getOMSDoubleArray ("hru_area", "nhru")
generated by        OMSDoubleArray o_hru_imperv = getOMSDoubleArray ("hru_imperv", nhru")
OMS system          OMSDoubleArray o_hru_percent_imperv = getOMSDoubleArray
calls based on              ("hru_percent_imperv", "nhru")
data dictionary content  OMSDoubleArray o_hru_perv = getOMSDoubleArray ("hru_perv", "nhru");
                    OMSDimension o_nhru = getOMSDimension ("nhru");
                    ...
                    }
```

Figure 15.2 Data object declaration and system initialization.

The declaration of data objects in a module is shown in Figure 15.2. Objects are generated based on OMS API system calls. The identifiers must match the data dictionary entries (see example in Figure 15.4).

Legacy Code Implementation

Wrapping is a technique to embed existing nonobject-oriented software into an object-oriented architecture. Mapping interfaces of conventional systems into an object-oriented syntax enables the access to this legacy code. This mapping procedure can be supported by wrapper generation tools, which generate glue code for both software "worlds." Wrapping provides some general advantages:

- Keeping and reusing the existing software infrastructure as much as possible
- Moving step by step toward object orientation
- Accelerated introduction of object orientation
- "Separation of concerns" (graphical user interface vs. number crunching part, etc.)

A module contains a set of methods. Some of them are inherited from the super class OMSComponent. A module may implement additional methods. A module has to override the methods **init**( ), **run**( ) and **cleanup**( ) for interface functionality. A method for customization is used to provide specific editors for setting up the module data objects. Such editors may be applications implementing a kind of preprocessing prior model run.

OMS uses a sophisticated method to integrate native languages such as Fortran and C. Some OMS framework tools support the integration of native code into modules. The example given in Figure 15.3 shows how module data objects (input/output/local data objects) are mapped to Fortran variables and parameters by mapping Java object properties to Fortran variables..

Dictionary Framework

The OMS dictionary provides and manages dictionary resources for model construction like model data, model parameters, modules and models. In addition, during the model development phase, it provides and manages parameter scenarios, time series data in model application. It also

Intialization routine
Property map section connects Java data objects to FORTRAN variables
FORTRAN computation code implements process logic

```
public native int init ( )  /*@F77
    @propertymap o_basin_area.value                -> basin_area;
    @propertymap o_hru_area.value1D                -> hru_area;
    @propertymap o_hru_imperv.value1D              -> imperv;
    @propertymap o_hru_percent_imerv.value1D       -> percent_imperv;
    @propertymap o_hru_perv.value1D                -> hru_perv;
    @propertymap o_nhru_value                      -> nhru
    @{
      real *8 totarea
      integer i
      real *8 diff

      totarea = 0.
      do 100 i = 1, nhru
        hru_imperv (i) = hru_percent_imperv (i)  * hru_area (i)
        hru_perv (i) = hru_area (i) - hru_imperv (i)
        totarea = totarea + hru_area (i)
100 continue
      diff = (totarea - basin_area)/basin_area
      if (abs (diff).ge. .01) then
       omsExeption ('Sum of hru areas is not equal to basin area')
       return
      end if
    @}  init = 0
*/;
```

Figure 15.3 Legacy code integration.

Data dictionary entry ID
Description
Version
Data category, range and default value
Data type and unit
Author reference

```
<coms: data id="hru_elev">
    <oms:descrption>
        <oms:short>Mean elevation</oms:short>
        <oms:full>Mean elevation for each HRU </oms:full>
    </oms:description>
    <oms:version>
        <oms:cdate>2001-08-15 02:24:39</oms:cdate>
        <oms:mdate>2001-08-15 02:24:39</oms:mdate>
        <oms:revision>1.0</oms:revision>
    </oms:version>
    <oms:value cat="par am">
        <oms:min>-300.</oms:min>
        <oms:max>30000</oms:max>
        <oms:default>0.</oms:default>
    </oms:value>
    <oms:unit cat="length">feet</oms:unit>
    <oms:type>float*8</oms:type>
    <oms:author href="rojas" />
<oms:data>
```

Figure 15.4 Data dictionary element.

handles the interaction between several dictionary data sources with database management systems and the modeling system. OMS dictionaries are written in XML.

The OMS dictionary design addresses two major issues:

1. The OMS dictionary XML resource file structure defines a generic and open scheme for describing and processing sets of (heterogeneous) dictionary content.
2. The OMS dictionary GUI tool operates with dictionary XML resources and manages its content in coordination with the other OMS components of the framework to construct and apply a model.

Figure 15.4 shows
tion about the paramet
requested by modules

OMS Dictionary Fr

A major dictionar
Format is available fo
for handling dictionari
These issues were tak

- A dictionary co
the dictionaries
to a tabular vie
- A permission sc
developers have
- Dictionaries m
query language
feature. Each d
XML as the pr
should be able
systems. There
database dictio
- Dictionaries sh
must drive the
- The dictionary
directional inte
to construct or
dictionary as r
- The logical str

Other design cor
as possible, both for
resource definition f
OMS tools is possit

Modeling System

The OMS fram
supports different ty
opment and applicat
with a basic set of
extensions.

The OMS Comi
dling of common el
using a python shell.
look and feel but al
One powerful featu
interaction of the C

OMS extension
according to model
of modeling:

Figure 15.4 shows a dictionary element, which encapsulates all meta-information and information about the parameter "hru_elev." Based on this record the system is able to instantiate objects requested by modules (see Figure 15.1) and initialize them properly.

OMS Dictionary Framework Architecture Design Considerations

A major dictionary design goal was to maximize transparency, openness and extensibility. Format is available for data, module, parameter, or scenario dictionaries. The preferred solution for handling dictionaries is a generic scheme where all of these special dictionaries could be mapped. These issues were taken into account when designing the overall dictionary architecture:

- A dictionary consists of data sets, which may be either homogenous or heterogeneous. Because the dictionaries are written in XML, data sets are not document-oriented and should not limited to a tabular view.
- A permission scheme is required to control the operations associated with the dictionary. Dictionary developers have the responsibility to set up appropriate permissions.
- Dictionaries may contain large sets of XML entries. Well-known approaches such as the XML query language can filter information from an XML data set. A dictionary must deal with this feature. Each dictionary must provide applicable queries to filter specific information. The use of XML as the primary dictionary data format opens a wide range of other data sources. The system should be able to generate XML dictionary documents from relational database management systems. There must also be uniform and transparent support for local, remote (via a URL) and database dictionaries.
- Dictionaries should be self-describing in terms of their data structures. The dictionary data content must drive the behavior of any GUI tools.
- The dictionary GUI tool should be able to operate with other tools in OMS in an easy way. A bi-directional interaction should be supported, so dictionary entries may be "dragged" into other tools to construct or run the model. New entries, such as modeling results, may be "dropped" into the dictionary as modeling results.
- The logical structure of a dictionary must be mapped into a physical tool representation.

Other design considerations including keeping the dictionary file as simple and as transparent as possible, both for humans and for processing in the OMS framework. By using XML as dictionary resource definition format and XPath for accessing elements of a dictionary, a slim architecture for OMS tools is possible.

Modeling System GUI Design

The OMS framework uses a common core user interface (CommonUI). The CommonUI supports different types of OMS extensions. Extensions implement different tools for model development and application. OMS is open and configurable through the extension package. OMS comes with a basic set of extensions, but also offers system developer an API to implement custom extensions.

The OMS CommonUI provides the following functionality for all extension: GUI resource handling of common elements; system logging; and a console interface to interact with the system by using a python shell. The CommonUI enables extensions with slim implementation overhead, common look and feel but also provides maximum flexibility to adapt the overall appearance of the system. One powerful feature of the CommonUI is the python-scripting interface. It allows the dynamic interaction of the CommonUI with extensions, based on shell commands.

OMS extensions implement the toolbox elements that can be used to customize the CommonUI according to modeler needs. OMS comes with a basic set of extensions dealing with various facets of modeling:

- **Development Extension** — allows the interactive assembly of models based on a module library. It provides support for module search, module retrieval and module integration from local and remote sites according model requirements. It allows the validation of module connectivity using input/output data constraints graphically. This extension delivers runable models.
- **Application Extension** — deals with the interactive model application of runable models. It provides automatically generated GUI elements for model parameterization, output customization, graphical components for visualization of variables and parameter scenario management. This extension delivers model results in terms of graphs and numbers.
- **Dictionary Framework Extension** — manages the modeling dictionaries. They are used to handle the following modeling resources: module interfaces; parameters and variables; parameter sets: and meteorological data set descriptors. Other extensions use this extension to validate the proper module connectivity (development extension) or for parameter value assignment (application extension).

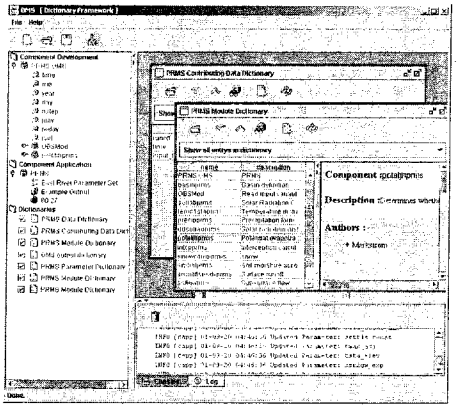

Figure 15.5 OMS CommonUI and extensions.

Other types of extensions could be developed for post-run results analysis, GIS connectivity, legacy code integration and model deployment using Web services. Figure 15.5 shows the OMS CommonUI loaded with the extensions for model development, model application and the dictionary framework. Each extension appears as a folder node in the model resource tree on the left side of the window. All extension related resources are sub-nodes of their folder. The common logging panel and the console command line interface appear as tabbed panels on the bottom of the window. The main part of the CommonUI occupies the extension desktop in the center of the window. The desktop is customized according to the extension implementation purpose.

OMS Model Views and Deployment

Enabling simulation models to run under different architectures and computing environments becomes more and more important, especially with the increasing demand for Web-based application of simulation models. OMS is designed to cover a variety of application and deployment and execution paradigms:

- The OMS application gets deployed by Java Web Start Technology over the Internet. This ensures the local client installation, automated update and security of OMS. Models being developed with OMS in such a scenario are running within OMS. This setting is typical for a development application, where the modeler needs flexibility to change the models module structure, parameter sets, as well as input data.
- The application of OMS models as canned models is required when validated OMS models are applied in projects or application scenarios. There is no need to change the structure of the model

or have the flexibil
of an adaptive, mo
- OMS models can
runtime environm
models. Preconfig
client site deals or

Within OMS it is pc
changing the model stru
run under a Web-only e

Figure 15.6 Loaded PRM

For proof of conce
interaction. The hydrolo
OMS. The goal was to
resource descriptors and

PRMS is a determi
A watershed is divide
conceptualization of PF
of interlinked reservoir
of PRMS reflect the va
tested) consists of 14 p

The test application
River basin, a subcatchi
was used for validation.
sel tool (Leavesley et al.

Figure 15.6 shows (
node in the resource tr
loaded PRMS runtime
generating GUI elemen

Although the PRM
required for the Java en
generation could preve
following steps in code

or have the flexibility of automatically generated GUI components for parameterization. The usage of an adaptive, model-specific GUI directs the model user.

- OMS models can be executed in a server-centric environment. Handled by a Web server, an OMS runtime environment produces a generic Web interface to enable a Web browser application of models. Preconfigured parameter and data sets can be accessed at the server side, whereas the client site deals only with a Web browser for model input and output.

Within OMS it is possible to transfer an existing model to different execution schemes without changing the model structure. Models being developed with the OMS environment are capable to run under a Web-only environment.

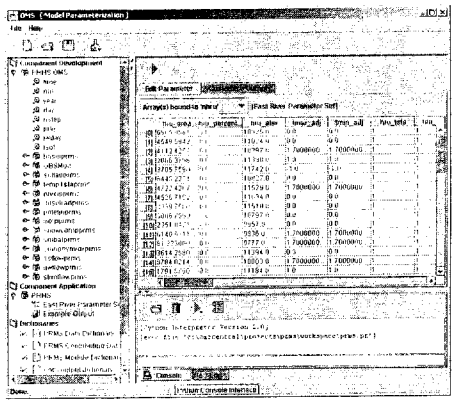

Figure 15.6 Loaded PRMS modules and PRMS generic parameterization GUI.

EXAMPLE APPLICATION OF PRMS

For proof of concept, a prototype model was developed to test component integration and interaction. The hydrological model PRMS, (Leavesley et al., 1983) was selected for integration into OMS. The goal was to transform the model structure into OMS modules and their related XML resource descriptors and compare the model output with the validated version of the model in MMS.

PRMS is a deterministic, distributed, continuous hydrological model (Leavesley et al., 1983). A watershed is divided into homogeneous hydrological response units (HRUs). The modular conceptualization of PRMS is reflected by simulating the hydrological system as a vertical series of interlinked reservoirs. Each hydrological process is represented as a separate module. Variants of PRMS reflect the variable characteristics of different catchments. The basic PRMS model (as tested) consists of 14 process modules (Figure 15.6).

The test application for OMS also contained a meteorological data and a parameter set for the East River basin, a subcatchment of the Gunnison River basin in Colorado. A 20-year set of climate data was used for validation. The East River parameter set was preprocessed and generated by the GIS–Weasel tool (Leavesley et al. 1997). Both, the parameter set and the time series input where in MMS formats.

Figure 15.6 shows OMS with modules forming the PRMS model. The Component Development node in the resource tree contains the PRMS OMS model build upon the 14 sub-modules. The loaded PRMS runtime was configured with the East-River Parameter Set. OMS is automatically generating GUI elements for parameter input such as spreadsheets for one-dimensional arrays.

Although the PRMS source code was already modularized for MMS, some recoding was required for the Java environment. PRMS modules are implemented in Fortran77. Automated code generation could prevent many of the errors that were introduced by the manual procedure. The following steps in code migration were performed:

1. The MMS runtime library was modified for capturing each modules interface calls (reading/writing to variables/parameter/dimensions). XML-intermediate module descriptors for this information were generated.
2. The Java OMSComponent file and the XML component descriptor were generated from the intermediate descriptor. Data objects referenced by MMS system calls in MMS/PRMS modules resulted in OMS/PRMS data objects referencing data dictionary elements. This transformation was done automatically.
3. The Java OMSComponents were extended with Fortran science code, embedded into the initialization, processing and cleanup section of each component. Local variables were identified (compiler test run) and added to the code.
4. OMS tools and compiler were used to generate binary executable code (Java class files, dynamic link libraries).
5. A total of 456 parameters, variables and dimensions were identified and generated. For each of them, an XML data dictionary element was generated and added to a PRMS data dictionary.
6. The East River parameter set was converted into an XML OMS parameter file and added to a parameter dictionary.

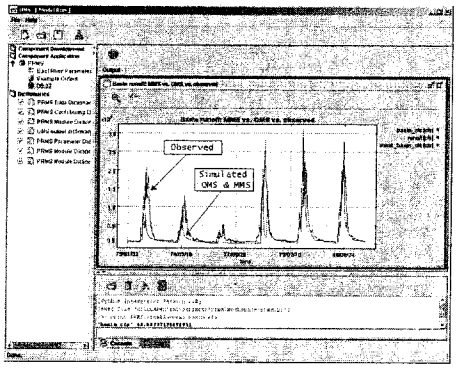

Figure 15.7 PRMS run in OMS — comparison of basin runoff from OMS vs. MMS and observed data.

The modules were loaded into a running OMS framework. The driving variables were traced for each component to verify the input. The output variables were plotted and compared with corresponding MMS/PRMS variables. Finally, the simulated basin runoff for the OMS and the MMS version of PRMS was plotted with the observed runoff. Figure 15.7 shows the result, matching graphs for MMS and OMS values for the basin runoff.

CONCLUSION

The OMS introduced in this chapter is a framework that facilitates the development of customized models from a library of science, data and utility modules, as well as their testing, application and deployment. OMS features component integration techniques, graphical user interface components, graphical visualization features and other utilities supporting model construction and application. The design of OMS was influenced and driven by the needs of agricultural and natural resource agencies to optimize the models, development process and maintenance. Due to the increasing complexity of the simulation problems encountered in natural resource and environmental management model development can only be efficient by using methods and tools such as OMS. Due to its dedication to modularized model development, OMS offers the potential for science building block reuse and easy update and maintenance.

Model management and transfer practices from the research unit development level to the field office application level were considered in OMS design. Hence, the goal of this project is the design

and implementation of a modeling system that emphasizes interoperability, connectivity, scalability and reuse of environmental simulation modules. The PRMS model was adapted into OMS, which lead to a significant reduction of module implementation code. Results showed that a major fraction of GUI components for model parameterization could be generated automatically from module data components. Researchers are in the process of disaggregating another large model, the Root Zone Water Quality Model (RZWQM), into OMS modules.

Future Developments

Future OMS development efforts will be leveraged by application and project demands. The integration of GIS capabilities by means of an OMS extension will enable interactive geo-processing in the context of model application. GIS implementation efforts are underway. The overall focus of the GIS extension will be driven by the need of modeling related features. This comprises model parameterization in terms of spatial parameter sets as well as spatial visualization of model progress and results.

Another major OMS effort will focus on a deployment of simulation models using Web services and Universal Description Discovery and Integration (UDDI). This will enable simulation models to be discoverable and directly callable by other simulation environments. Other efforts include post-run analysis extensions that will be integrated with descriptive statistics.

ACKNOWLEDGMENT

The OMS Project is supported and funded by the USDA-ARS, in collaboration with the USDA-NRCS and the U.S. Geological Survey.

REFERENCES

Abbot, M.B., J.C. Bathurst, J.A. Cunge, P.E. O'Connell and J. Rasmussen. 1986. An introduction to the European Hydrological System — Système Hydrologique Européen (SHE), structure of a physically based, distributed modelling system, *J. Hydrology,* 87:61–77

Abel, D.J. and P.J. Kilby. 1994. The systems integration problem, *Int. J. Geographical Inf. Syst.,* 8(1):1–12.

Arnold, K., J. Gosling and D. Holmes. 2000. *The Java™ Programming Language, Java™ Series,* 3rd ed. Addison-Wesley, Reading, MA.

Balmer, D.W. and R.J. Paul. 1990. Integrated support environments for simulation modelling, *Proc. of Winter Simulation Conf.,* December 9–12, New Orleans, Louisiana, 243–249.

Breunese, A.P.J., J.L. Top, J.F. Broenink and J.M. Akkermans. 1998. Library of reusable models: theory and application. Simulation Series, Simulation Councils Inc. 71(1):7–22.

Cota B.A. and Sargent R.G., 1992. A modification of the process interaction world view, *ACM Trans. on Modeling and Computer Simulation,* 2(2):109–129.

DeRemer, F. and H.H. Kron. 1976. Programming-in-the-large vs. programming-in-the-small, *IEEE Trans. Software Eng.,* SE-2(N.2):114–121.

Fayad, M.E. and D. Schmidt. 1997. Object-oriented application frameworks. *Commn. ACM,* 40(10):32–38

Leavesley, G.H., P.J. Restrepo, S.L. Markstrom, M. Dixon and L.G. Stannard. 1996. The modular modeling system (MMS) — user's manual, U.S. Geological Survey Open-File Report 96-151.

Leavesley, G.H., R.J. Viger, S.L. Markstrom and M.S. Brewer. 1997. A modular approach to integrating environmental modeling and GIS, *Proc. 15th IMACS World Congr. on Scientific Computation, Modelling and Applied Mathematics,* August 24–29, Berlin, Germany.

Leavesley, G.H., R.W. Lichty, B.M. Troutman and L.G. Saindon. 1983. Precipitation-runoff modeling system — user's manual, U.S. Geological Survey Water Resources Investigation Report 83-4238.

Leavesley, G.H., S.L. Markstrom, P.J. Restrepo and R.J. Viger. 2002. A modular approach to adressing model design, scale and parameter estimation issues in distributed hydrological modeling, *Hydrological Processes,* 16(2):173–187.

McCown, R.L., G.L. Hammer, J.N.G. Hargraves, D.L. Holzworth and D.M. 1995. APSIM: a novel software system for model development, model testing and simulation in agricultural systems research, *Agricultural Syst.*, 50:255–271.

Panagiotis, K.L., S. Molterer and B. Paech. 1998. Re-Engineering for Reuse: Integrating Reuse Techniques into the Reengineering Process. citeseer.nj.nec.com/linos98reengineering.html.

Pidd M. and R. Bayer Castro. 1998. Hierarchical modular modeling in discrete simulation, *Proc. 1998 Winter Simulation Conference,* D.J. Medeiros et al., Eds., IEEE, Washington, D.C., 383–390.

Praehofer, H. 1996. Object-oriented modeling and configuration of simulation programs. In *Eur. Meet. on Cybernetics and Syst. Res.,* Vienna, Austria, April, 259–264.

Singh, V.P. 1995. *Computer Models in Watershed Hydrology,* Water Resources Publications, Highlands Ranch, CO.

Top, J.L., A.P.J. Breunese, J.F.Broenink and J.M. Akkermans. 1997. Structure and use of a library for physical system models, *Proc. ICBGM'95, 2nd Int. Conf. Bond Graph Modeling and Simulation,* Las Vegas, Nevada, January 15–18, SCS Publishing, San Diego, CA, Simulation Series, 27(1):97–105.

Ulgen, O.M., O. Norm and T. Thomasma. 1991. Reusable models: making your models more user-friendly, *Proc. 1991 Winter Simulation Conf.,* December 8–11, Phoenix, AZ, 148–151.

Whelan, G., K. J. Castleton, J.W. Buck, G.M. Gelston, B.L. Hoopes, M.A. Pelton, D.L. Strenge and R.N. Kickert. 1997. *Concepts of a Framework for Risk Analysis in Multimedia Environment Systems (FRAMES),* PNNL-11748, Pacific Northwest National Laboratory, Richland, WA.

Zeigler, B.P. 1990. *Object-Oriented Simulation with Hierarchical, Modular Models,* New York Academic Press, San Diego.

F

Jerry L. Hatfield and

CONTENTS

Simulation models
These models have ran
within the soil–plant–a
generated from a wide
of processes and to be
Ritchie (1991) assemb
Since that time there h
models for agricultural
perfect model is to be
explore some of the c
systems models.

Models are represer
with much more predi
predict the trajectory c
into space and to inte
chemical reactions bec
to predict biological sy
gations of our unders
equations. We learn fr

1-56670-0563/02/$0.00+$1.50