A Kernel Approach for Interactive-oriented Model Construction in JAVA

Olaf David
Department of Geography / Geoinformatics
Friedrich-Schiller-University Jena
Loebdergraben 32, D-07743 Jena, Germany
Tel: +49 3641 632732, Fax: +49 3641 632727
email:olaf.david@geogr.uni-jena.de

August 12, 1997

Abstract

Models in environmental hydrological simulation and research are widely accepted for supporting the decision process in landscape planing management. In contrast to special purpose models with limited universal application modeling systems became popular. Such kind of systems deals with a component oriented approach in terms of their capability constructing a model from a set of library components residing in a (hydrological) modeling library. Our efforts in modeling system development are tending to introduce the JAVA architecture as the basic architecture for such kind of modeling systems in simulations. The Object Modeling System OMS is characterized by a pure object-oriented architecture on the top of the JAVA virtual machine in terms of the fundamental interaction principle of the modeling objects, which is closely to the design of pure object-oriented systems and languages. This paper will introduce a modeling system architecture for an interactive oriented development of hydrological simulation models, where a scenario variation in simulation (data + algorithm variability) gets interested as well as execution performance.

1 Introduction

Modeling and simulation in hydrology is general used explaining complex environmental coherence immediately closed to soil/water/atmosphere transition. In recognition of processes observed in past and validated in software models forecasts are possible. Different scenarios in hydrological system balance are worth to evaluate under different hydrological questions to a model.

Modeling systems [MHH+96] [Uni93] where developed claiming flexibility and variability in module usage under these concerns. But in designing and implementing such a kind of prototyping oriented development systems with regard to its application in hydrology inputs from software construction and technology are requested to construct an 'state of the are' modeling tool. But this challenge for software construction cannot be solved using (i) conventional modular approaches and (ii) with (modeling) programming languages like FORTRAN. Such languages where especially designed for optimizing the model execution with limited respect to design oriented issues like modularization in model construction. A well tuned usage of programming languages becomes necessary to optimize the *model generation time* as well as the *model run time*.

2 Object Modeling System Overview

The Object Modeling System which is currently under development at the Dept. of Geoinformatics addresses these issues. It provides an support for a simple integration of optimized FORTRAN and C code due to an automatic wrapper generator for interface mappings to JAVA as well as an integrated Graphical User Interface for an interactive combination of components (JAVA classes) on the top of the JAVA virtual machine.

This system provides a simple (modeling) user access to new technologies in software handling in terms of an experimental prototyping oriented development process. Modularization and the more restricted view in pure object-orientation was one of the main underlying methodological principles in architecture design.

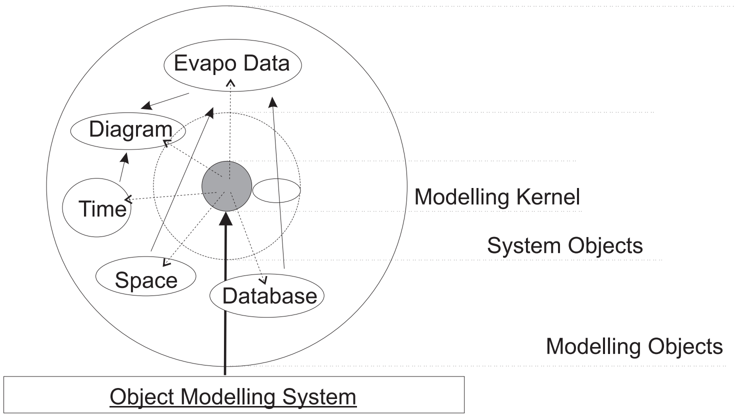

Figure 1: Object Modeling System Architecture

According to the micro kernel design in operating system research a tool kit approach was used to construct the modeling system. The principle architecture of the system is shown in Figure 1. The system is general divided into (i) a compact *Modeling Kernel* responsible for model generation and execution and (ii) a surrounding *Model Implementation* representing the developed model as a set of interacting objects.

In complex system understanding this approach replaces a monolithic system using "hard-wired" build in functions with a distributed system of loosely coupled objects. The man purpose of the modeling kernel is to realize the *(i) creation* and *(ii) the type secure interaction* of JAVA objects. The kernel doesn't know anything about the internal behavior of the objects. Objects stands like for example a database, hydrological evaporation object, a interception object, a visualization object and so on. The mechanisms (i) and (ii) are working independently from the complexity and size of objects and are operating only using the visible object interfaces.

The general functionality of the whole system is determined by the number of available components for creating objects residing in (modeling) libraries. Than more universal and powerful the modeling library in terms of modeling approaches for different (hydrological) processes then more convenient is the whole system. Therefor such an *open* system design is much more suited for flexible extension based on modeling libraries than monolithic systems providing mostly a limited set of build in capabilities (e.g. time iteration management, database access, presentation of results). In contrast to such system designs the Object Modeling System architecture does not provide any built in functionality. There are only some modeling and related libraries as a kind of system libraries usable in most cases of model developing. These libraries can be used as an originator for the implementation of own modeling library components. Currently the modeling system library consists of classes providing:

Database access classes based on the JDBC API. A universal database connectivity to any relational database supporting the ODBC interface is given. This enables any model developer in OMS an access to a wide range of relational databases for data retrieve

Graphical presentation classes for output visualisation. Some classes realising different kinds of graph output for online observation of modeling process behaviour.

Time management classes for general time handling within models. These classes provide a uniform time interface to (hydrological) models. Iterations in different time steps in a defined time interval are typical purposes of such capabilities.

These libraries should illustrate some issues in flexibility using objects for any purpose. Nothing is limited by the Modeling System, all functionality is provided via modeling libraries. The Modeling System itself is only *handling* these modeling libraries using the *Modeling Kernel* which is explored in more detail in next section.

2.1 Modeling Kernel Architecture

The Modeling Kernel is the component of the Object Modeling System responsible for correct and type secure object interaction described in last section. The main purpose of the Kernel is:

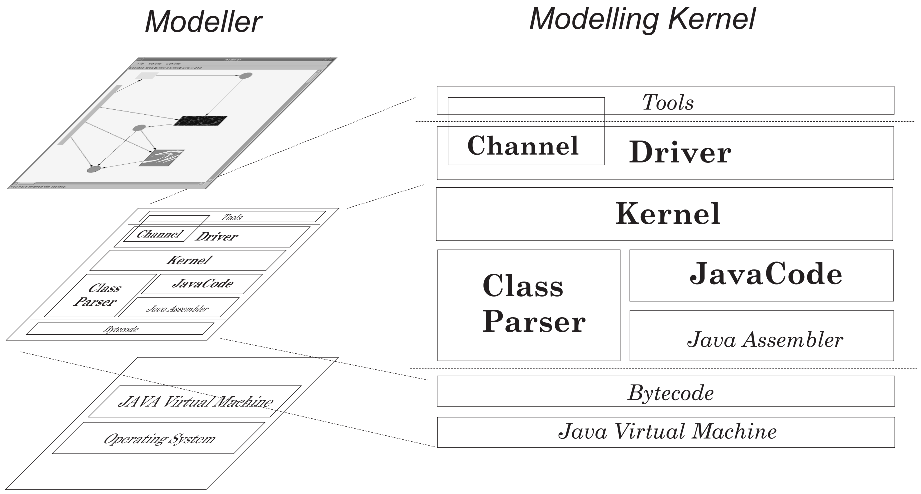

Figure 2: Modeling Kernel

- *generating* and setting up an executable model from a given input which is strong related to generate the code for object creation and connection binding and
- *perform* and control the execution of a generated model

The Modeling Kernel is located as the central component of the *Modeler tool* that allows the graphical oriented assembling of models in a kind of a visual programming language. The Modeler tool became the front end for user interaction. On the lowest layer there is the JAVA execution environment residing on the top or embedded into the operating system. The detailed structure of the Modeling Kernel is shown on the right side of the Figure 2. It is full written in JAVA and is separated into several layers implementing different abstraction levels in object management.

The *Driver* layer on the top of the kernel implements a language interface for model description defined in the syntax of the *Object Modeling Language OML* (described in next sextion). It encapsulates an interface for communication with the Modeler GUI.

The *Kernel* layer is responsible for object management in terms of meta info storage of classes and objects, type checking, access validation of classes and methods independently from any language in Driver definition and performs semantic checks of OML.

The *ClassParser* layer realizes the full structure analysis of JAVA class files with extraction of several meta information of them like for example a list of public methods, parameter lists of each method with regard to overloaded methods.

The *JavaCode* layer is managing all code related information necessary for construction of executable models in JAVA bytecode. It calls assembler operations for building correct class files with all internal structures.

2.2 Well tuned Language Application

In our opinion there is a need for a well tuned application of tools in modeling process design and model assembling using the OMS. We have implemented an Object Modeling Language (OML) within the Modeling Kernel in extension to the JAVA usage. Both languages are able to create JAVA code but the development purpose and background determines the application of one.

1. For *developing modeling libraries* the JAVA language should be used as most as possible. Then the portable application of the library under different operating systems is guaranteed. For developing number crunching parts of a module, native C and FORTRAN implementations can be used. Therefore we provide a tool as a part of the OMS which generates the C and JAVA wrapper for a given FORTRAN source. In this scheme the wrapper generator is responsible for the correct parameter format usage (e.g. equivalent data types in C/FORTRAN/JAVA, call by reference/value) not the user.

2. The OML is designed to support the interactive prototyping process in terms of *assembling a model from a set of available modules* from a library. OML is used to express the object interaction as shown in Figure 1. The OML interface is embedded in the top level layer of the Modeling Kernel. It provides the modeler a pure object-oriented scripting language interface, but the interface is hided within the Modeler tool. Only a graphical view to the model is given. A maximal support of interactive work in terms of model testing, observation and changing is provided by short model development cycles (assemble model - test the model) using efficient and incremental techniques in compiling.

3 Object Modeling Language

The Object Modelling Language was designed with inspiration by pure object oriented languages and systems [Cha91, GR83, LCD+95, Cha93, Szy92]. This OML provides a subset of basic features of object-oriented languages which are useful to extend and support the final process in model construction, the combination and interaction of objects. Therefor the design of the OML and the design of the modeling kernel was closely coupled.

The design of OML was driven by a global tendency in language design in terms of migrating essential features of conventional programming language to an associated library to provide (1) a simple and slender language construction and implementation and (2) a better support for user defined extensions of the language only by extending the library [OL92, LCD+95]. Such an approach allows the adoption of a programming language to user demands like for example implementing parallel iterators or operator overloading without any language changes.

OML allows the user to *combine* any complex objects with full regard to object-oriented features like multi-methods, inheritance or static typing. Objects in OML can be created from any JAVA class.

According to pure object oriented systems only the *(1) creation of objects* and *(2) sending a message to an object* is defined in the language. Because there is a garbage collection in the JAVA system, releasing an object was omitted from the language definition. To ensure the realization of a control flow in model execution (conditionals, iterations) the encapsulation of statements in blocks was introduced in OML. Blocks can be used as parameters for messages as well as objects and basic literals (floating and fixed point numbers, strings, booleans,). Blocks seems to be an extension to the underlying JAVA architecture but there is an transparent implementation of Blocks in the modeling kernel system library. In OML

Finally each statement of a valid OML program can be in principle either:

1. the object creation : `Classname OBJECT` or
2. the message send: `OBJECT` $\leftarrow$ `Methodname Parameters`
   where parameter can be
   (a) `OBJECT` (any object identifier)
   (b) `Literals` (e.g. 1.2, 1234, true, ...)
   (c) `Block` (a Sequence of statements (1) and (2), embedded into braces)

`Classname` and `Methodname` comes from any public JAVA class in a (modeling) library. The ClassParser layer in the modeling kernel is responsible for extracting this information from a binary class file. In general the kernel layer structure reflects the different responsibilities as shown in section 3.

Finally the support of native JAVA Methods in OML is given as well because native methods are transparent on the bytecode level and are visible on `.class` level as ordinary JAVA methods with a modifier flag indicating their native implementation [Sun95b].

4 An OML Example

The pure object-oriented interaction paradigm described in last section id powerful enough building hydrological simulation models. A simple example in OML is given below. There are used different kind of objects to realize a simulation of evapotranspiration using the HAMON equation [JBA90]. Data input, the modeling computation loop and the data output are represented as full typed object interaction.

Data input
: For Data Input a Database object is used to establish a connection to a relational data base. In out example we are using a ODBC access to ORACLE 7 database using the JAVA JDBC and a bridge to OCBC. When a database object gets created a connection to the `jdbc:odbc:qeor7` established. Using the `query` method a SQL statement can be submitted to the database object. Another object is used for time control. The `Time` object is representing a interval from 1971 to 1973.

Computation
: For hydrological simulation there are some modeling classes available within the modeling library of the OMS residing in several JAVA packages. There are a *Data objects* (`Evapo, Temp, Precip, ...`), *Process objects* (`Hamon, Haude, ...`) and some related classed implementing physical units and constants. The main part of the simulation consists of a Loop. The `TimeLoop` method of the `Time` object is called with a computation sequence as parameter. This sequence gets executed for each time step. The type of the `Time` objects shows that the `TimeLoop` is working on a daily interval. Inside the `TimeLoop` the actual temperature data is obtained from the data base, the `Evapo` object is set with the computed evaporation.

Data Output
: For data output a two dimensional diagram window object (`Diag`) is used. After object creation, the method `appendPoint` can be used for appending a point at the end of the painted diagram graph. In our case the computed evaporation is drawn.

As defined in the real syntax of the OML classes are denoted with enclosed `< .. >`, followed by the object name with parameters (e.g. `<gui.Dbtool.Database> DBase("jdbc:odbc:qeor7", "ga", "geoadmin"); )` . A method call of an object is invoked by sending a message cognizable with the arrow (e.g. `Diag<- appendPoint( Time, Evapo<-getMM());`). This syntactical definition differs from the JAVA standard as well as the block parameter of the `TimeLoop` message. Replacing the standard control statements of the JAVA language by such an approach leads to a more simpler constructed compiler for code generation, which is used in the Modeling Kernel.

The example above illustrates some advantages of such an approach. There is no embedded functionality of data base access, time management or presentation 'hard wired' within the modeling system or the kernel. Only on the top of these classes the modeler is able to develop his own modeling classes and finally connecting them to complex models. It is not bounded to any limitations in resource handling. If the modeler needs for example a simultaneously connection to more than one database he can realize this by creating different parameterised database objects.

Another flexibility is given by the problem domain specific creation of object for nested iteration in models. In such a similar way on different time scales can be operated.

The full type checking mechanism of the kernel allows only well typed parameters in object-oriented semantics. In the example above, the HAMON equation is used in daily context. In such a arrangement of variables and parameters the Kernel can determine the correct usage of an process approach in a given time or space scale context.

```
// Database initialization
<gui.Dbtool.Database> DBase('jdbc:odbc:qeor7', 'ga', 'geoadmin');
DBase<-query('select bytesequence from measurementvar where variablename
= tempe rature');

// setting the time daily interval
<lib.fsu.System.TimeControl.DayInterval> Time(1971,1,1,1973,1,1);
<lib.fsu.Models.EvapoTrans.EvapoTransData> Evapo();
<lib.fsu.Models.Temperature> Temp();
<lib.fsu.Models.Constant> HC(0.0055);
<lib.fsu.Models.Angle> Ang();
<lib.fsu.Models.EvapoTrans.Hamon> Hamon();
<lib.pub.Diagram.DiagramWindow> Diag('Graph output', 'Evapo',
640,480,'day','mm' );

Ang<-setDeg(44.2);

Time<-TimeLoop([
   Temp<-setGradC(DBase<-getAsDouble(1));
   Evapo<-set(Hamon<-computeEvapo(Time,Temp,Ang,HC));
   Diag<-appendPoint( Time, Evapo<-getMM());
]);

DBase<-CloseConnection();
```

Figure 3: Hydrological Model Example in OML

Whenever the modeler wants to observe a internal state of a variable, maybe the computed evaporation on each time step, he is able creating and using a new diagram object for tracing. In same way modeling process approaches can be replaced. Like for example if the hydrological modeler wants to examine the behavior of it's model using the PENMAN-MONTEITH[JBA90] modeling approach instead of the HAUDE [JBA90] equation for estimating an evaporation he has to substitute this modeling object with attention to the needed parameters. When a new data access is necessary to use like for example to make a network communication to a remote GIS database, it can be added as a new library component without any changes in the Modeling System itself.

5 Conclusion

Motivation to establish the Object Modeling System on the top of the JAVA architecture is driven by the wide acceptance of JAVA in scientific application. The easy combination of basic simulation modules to a complex simulation model is needed in hydrological simulation to realize variable scenario management. JAVA addresses some of the issues in dynamic simulation generation and management with its API (e.g. dynamic class loading).

The support of interactive model setup for execution to play with different simulation scenarios requires well designed tools for this purpose. The Object Modelling System architecture addresses these things with performance oriented and interactive oriented design of the modeling kernel architecture and the OML and pure object-oriented combination of JAVA Modules into models.

Our current efforts are focused on a GUI for the automatic and user invisible generation of OML model scripts and the embedded compilation and execution in a complex JAVA Modelling Framework of OMS. The acceptance in

JAVA usage for hydrological modeling, where almost all models where written in FORTRAN, is determined how easy FORTRAN Code can be integrated into the Modelling process. Therefor we provide an transparent way with automatic C/JAVA wrapper generation for FORTRAN code using the Native Interface of JAVA.

References

[Cha91] C. Chambers. Making Pure Object-Oriented Languages Practical. In *Proceedings of the OOPSLA Conference*, 1991.

[Cha93] C. Chambers. The Cecil Language: Specification and Rationale. Technical Report TR-93-03-05, University of Washington, Dept. of Computer Science and Engineering, March 1993.

[Dav95] O. David. Interactive Environmental Modeling using an Object-Shell. In *Proc. International Congress on Modelling and Simulation MODSIM 95*, volume 4, pages 185–190. MSSA, MSSA, November 1995.

[Dav96] O. David. Experiences Building a Pure Object-oriented Modelling System Architecture. In P. Zannetti and C.A. Brebbia, editors, *Development and Application of Computer Techniques to Environmental Studies VI*, pages 605–615. Computational Mechanics Publications, Southampton, Boston, September 1996.

[Gla96] R.L. Glass. The Relationship Between Theory and Practice in Software Engineering. *Communications of the ACM*, 39(11):11–13, November 1996.

[GR83] A. Goldberg and R. Robson. *Smalltalk 80:The Language and its Implementation.* Addison Wesley, 1983.

[JBA90] Editors: M.E. Jensen, R.D. Burman, and R.G. Allen. *Evapotanspiration and Irrigation Water Requirenments.* American Society of Civil Engineers, 1990.

[Kra96] D. Kramer. *The JAVA Platform, A White Paper.* JavaSoft, Mountain View, California, 1996.

[LCD+95] B. Liskov, D. Curtis, M. Day, S. Ghemawat, R. Gruber, P. Johnson, and A. S. Myers. *Theta Reference Manual.* Programming Methodology Group Memo 88, MIT, Cambridge, MA, February 1995.

[MHH+96] R.L. McCown, G.L. Hammer, J.N.G Hargreaves, D. Holzworth, and D.M. Freebairn. APSIM: a Novel Software System for Model Development, Model Testing and Simulation in Agricultural Systems Research. *Agricultural Systems*, 50:255–271, 1996.

[OL92] S. Omohundro and C-C. Lim. The Sather language and libraries. Technical Report TR-92-017, International Computer Science Institute, March 1992.

[Sun95a] Sun Microsystems, Inc., Mountain View, California. *The JAVA Language Specification*, 1995.

[Sun95b] Sun Microsystems, Inc., Mountain View, California. *The JAVA Virtual Machine Specification*, 1995.

[Szy92] C. A. Szypersky. *Insight Ethos: On Object-Orientation in Operating Systems.* PhD thesis, ETH Zuerich, 1992.

[Uni93] University Colorado at Boulder. *Modular Modeling System Users's Manual*, January 1993.

[Zan96] P. Zannetti. Environmental Modelling - next generation. In P. Zannetti and C.A. Brebbia, editors, *Development and Application of Computer Techniques to Environmental Studies VI*, pages 3–13. Computational Mechanics Publications, Southampton, Boston, September 1996.