Encapsulating Environmental Models and Data using Java and XML

A. E. Rizzoli[a], R. M. Argent[b], M. Manglaviti[c], and M. Mutti[c]

[a] *IDSIA, Manno, Switzerland (andrea@idsia.ch)*

[b] *Centre for Environmental Applied Hydrology; Department of Civil and Environmental Engineering The University of Melbourne, Australia (R.Argent@unimelb.edu.au)*

[c] *University of Applied Sciences of Southern Switzerland, Department of Informatics and Electronics, Manno, Switzerland*

Abstract: Advances in distributed and component-based computing, particularly via the Internet, are paving the way for a client-server approach to simulation modelling. In this approach, components are provided as 'services' on Internet sites, from which model-building clients select and run only those components necessary to meet the modelling needs of a given problem situation. This approach enhances model component re-use, but creates significant difficulties in model component specification and communication. A range of component communication protocols, such as OMG's CORBA, Microsoft's DCOM (now ".NET"), and Sun's Java RMI are available. This paper discusses these approaches and tools and then addresses the problem of effectively separating model interfaces from their implementations, irrespective of the communication solution adopted. This problem is addressed using a "data-binding" solution based on XML (eXtensible Markup Language) and Java. The solution is based on using an XML-schema to describe the input and output interface of the component, and the design and implementation of dedicated "schema-mapper" applications which read the schema and generate the code needed for component data import and export. Once a schema for a model is written, model developers can access and run the model simply by accessing its "published" interface.

Keywords: Model integration; XML schema; Component-based modelling; Distributed modelling

1. INTRODUCTION

Advances in component-based computing and component distribution, particularly via the Internet, are paving the way for a client-server approach to simulation modelling. Advantages include easier model component maintenance on dedicated servers, thereby freeing scientist and manager end-users from much local hardware and software specific model management. Distributed computing also means that model components can be seen as Internet "services" from which users can select only those components required to construct a model for a particular application. This approach also enhances model component re-use. Nevertheless, the Internet and distributed computing have also brought with them many headaches. The requirement for components to effectively communicate and work together has spawned a wide range of standards and competing approaches.

This paper discusses these approaches and tools and then addresses the problem of effectively separating model interfaces from their implementations, irrespective of the communication solution adopted. This problem is addressed in the context of automatically encapsulating a model component into a given distributed computing standard. A "data-binding" solution based on XML (eXtensible Markup Language) and Java is proposed, although this approach can be logically extended to other systems. The proposed solution is based on using an XML-schema to describe the *semantics* and the *syntax* of the input and output interface of the component, and the design and implementation of dedicated "schema-mapper" applications which read the schema and generate the code needed for component data import and export. Thus, once a schema for a model is given, the model interfaces can be automatically generated. In such a way, developers of different distributed computing environments provide "schema-mappers", and the modeller need not write a "driver" for each component, but rather writes a detailed description of the data in XML.

2. DISTRIBUTING RESOURCES ON THE INTERNET

At the beginning of the "information-era", after WWII, computers were rare, and resources were centralised around individual machines. The advent of personal computers in the late 1970s changed this, and initiated a move towards distributed resources, albeit through small computer clusters in local area networks. This was an overall positive evolution in the history of computing, since it moved computing resource availability and affordability out of the realm of research centres and universities and into offices and homes.

This revolution also had an impact in the field of environmental modelling and software. Through the use of PC-based compilers, environmental managers, researchers, and students were able to design, implement and run mathematical models, enjoying an unprecedented flexibility and freedom of experimentation.

The downside of these developments was the proliferation of the programming languages and data standards adopted by researchers in coding their models. Two effects made users take a more serious look at what was going on: first the acknowledgement that re-implementing models every time a computer operating system was upgraded, was not the way to go. Too much effort has been spent - and it is still currently spent - in trying to re-use and encapsulate "legacy models" into new modelling system. This paper aims to help in reducing the effort of this endeavour. The second effect was the widespread adoption of the Internet in the early 90s that made possible online exchange of data and models.

The technological advances brought by the Internet raised the attention of researchers towards two orthogonal approaches to distributing information, namely:

- distributed data, metadata, and models; and
- distributed computing resources, providing model solvers and simulators as Internet services.

In this paper we focus on the latter feature, in particular investigating the possible developments implied by distributing computing resources, from collaborative and distributed computing to model and data interchange and re-use.

3. THE BASIC ARCHITECTURES FOR NETWORK COLLABORATION

Architectures for distributed computing have been under development for some twenty years (Figure 1). Long before web browsers became available, computing connectivity was being pursued through examples such as the Unix `rpc` (remote procedure call) library routines [Nelson, 1981]. Using the `rpc` library, a C programmer could make procedure calls on other machines, passing parameters along the way. While this feature found little application in environmental modelling, it paved the way to many of the later Internet applications.

Microsoft developed various communication features in their flagship Windows operating system, starting from the Windows for Workgroups evolution of Windows 3.11, released in 1992. They introduced Object Link Embedding (OLE) [Microsoft Corporation, 1987-1992] and Dynamic Data Exchange (DDE) [Petzold 1992]. Dynamic Data Exchange defines applications as *clients* and *servers* and, as the name implies, allows the client and the server to exchange data. These concepts evolved into Microsoft's COM (Component Object Model) architecture, which enables development of client/server applications based on the object-oriented paradigm.

In 1989, a group of software and hardware vendors formed a consortium, the Object Management Group (OMG), committed to "creating a component-based software marketplace by hastening the introduction of standardised object software". The major by-product of their effort was the Common Object Request Broker Architecture (CORBA), the first specification of which was published in October 1991. An Object Request Broker is a software device that allows sharing of software objects on a network. OMG also developed IIOP (Internet Inter-ORB protocol), which allows ORB to work on the Internet. CORBA provides a language (IDL, Interface Definition Language) for writing interfaces for existing applications, written in different languages, which run on different machines on a network.

Basically, OMG invented the architecture which is found in Microsoft's DCOM (Distributed COM) and Sun's Java RMI (Remote Method Invocation). Both these approaches have similarities to the CORBA architecture; the differences are evident at a lower level, when examining the inner workings of these

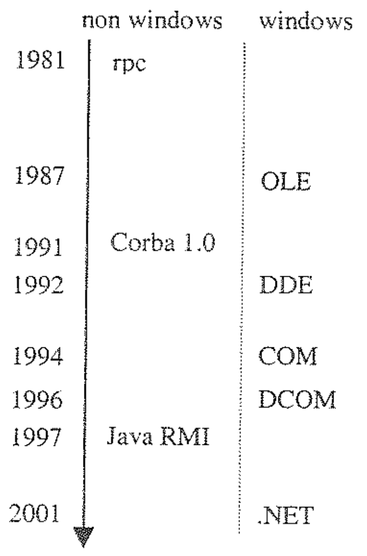

Figure 1. A timeline for distributed computing.

architectures [Tallman and Kain, 1998; Raj, 1998].

All these architectures provide valuable tools that support development of software systems which exploit the Internet as a medium to publish, exchange and develop environmental models and data. In the next section we will present some of these applications.

4. PUTTING DISTRIBUTED MODELS AND DATA TO WORK

In this section we present different approaches to distributed modelling, and to the use of the Internet as a support to modelling and simulation activities. There are available a range of examples where components and applications reside or are executed at either the server or client end. Some of these examples include:

- web-based simulations, where data and programs reside on the server. In these, a server-side application can also provide an interface to the program by means of CGI-scipts or Java Applets.
- distributed simulation, where different simulation components (often legacy applications) are separated, possibly on different machines, and integrated model execution is undertaken by passing input and output between applications
- distributed model bases, where model repositories are available on Internet sites. Users can download and run models from these sites to meet particular needs.

4.1. General Purpose: DMSO HLA

The Defense Modeling and Simulation Office (DMSO) has promoted and developed the High Level Architecture (HLA), a general purpose architecture for simulation reuse and interoperability. The main DMSO objective was to create a software environment in which distributed simulation was possible and where they could also re-use legacy models. HLA is now an open standard (IEEE Standard 1516) and has also been chosen as the facility for Distributed Simulation Systems by the OMG.

An example of a system intended to be fully compliant with the HLA is DIAS, the Dynamic Information Architecture System [Sydelko et al., 1999]. This architecture underlies the object-oriented implementation of the Integrated Dynamic Landscape Analysis and Modelling System (OO-IDLAMS) and was also the architecture in the Dynamic Environmental Effects Model (DEEM), used in a number of military-related environmental effect applications in the US. DIAS supports the integration of a variety of models, databases and information processing applications. DIAS also contains an expert system that assists users in developing new components to meet particular information management needs, based on consideration of the context and scale of the problem situation. In DIAS-based applications, integration of legacy models is undertaken through a 'registration' process. Legacy applications do not directly communicate amongst themselves, but interact indirectly via domain-specific entity objects. This approach provides a generally stable modelling environment as additions or deletions of models do not affect the core operation. Legacy applications are also executed in their native languages (e.g., FORTRAN, C, etc.), and, using an ORB-based approach, can operate in a distributed manner across the Internet.

4.2. Web-Based Processes

Examples of web-based server-end environmental applications are becoming increasingly available, albeit in generally simple mapping and request-response forms. Simple examples can be found in the air and water monitoring web sites provided by various national and state environmental protection agencies and authorities. Many of these sites (eg http://www.sca.nsw.gov.au/) can be queried to provide the outputs from statistical models related to "State of the Environment" style indicators and produce outputs that report rated or percentile results.

4.3. Distributed Simulation

An example of distributed simulation is found in the Modular Modelling System (MMS). This is a framework for modelling that can be used to develop, support and apply any dynamic model [Leavesley, 1996]. It is a database-centric system with major components for input data pre-processing, development and application of models, and post-process analysis and visualisation. Communication between models is performed via exchange of ASCII files with a central database. The strengths of the system are that is supports a significant selection of legacy systems, although this is offset by the reduced re-usability that arises by having complete models, rather than components, encapsulated and linked.

4.4. Distributed Model Bases

ECOBAS is an example of a model-base, and can be described as a system that produces model documentation that is easily accessible, complete, standardised, comparable and transferable to different applications [Hoch et al., 1998]. Documentation includes the basic model equations and variables as well as additional information about the application context and validity issues for model use. The system provides some of the semantic requirements for selecting appropriate models from distributed sources, and also supports some of the technical requirements.

5. ATTACHING SEMANTICS TO DATA

Distribution of services (models and data), such as those described above, leads to problems in communication among these services. There is a need for a software infrastructure that enables data and model integration on the Internet and, in general, enables Internet applications (models in our case) to retrieve and process information, without knowing in advance what this information will look like. Therefore this infrastructure must provide *semantics* to data and to applications.

The problem of attaching semantics to Internet data and applications has been pursued for some time and there are some promising efforts that incorporate decades of Knowledge Representation research [Berners-Lee et al., 2001]. The suggestion is to use *ontologies* to describe the knowledge domain. In this situation, an ontology is a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents [Gruber, 1993]. Ontologies provide a way to interpret knowledge, and can be used to provide information to external users for the interpretation of the modelling domain.

Example ontologies for various domains have been developed, such as the Unified Modeling Language for software engineering [Booch et al., 1999], Modelica for control engineering [Mattsson et al. 1998], Structured Modelling for management sciences [Geoffrion, 1987], Ecobas for ecological modelling [Hoch et al., 1998], and the open modelling engine for environmental modelling [Rizzoli et al., 1998].

The next step is to automate the interpretation of ontologies, creating web *agents* able to read an ontology and then to interpret data structured according to that ontology. Among the various alternatives, the World Wide Web Consortium is promoting the Resource Description Framework (RDF), as a standard to represent the *meaning* of Internet resources [W3C, 2001a].

5.1. Applying the Semantic Web Framework to the Case of Distributed Modelling and Simulation

It is expected that, given current directions in Internet use and distributed computing, model components and data will become increasingly available as resources on the web, allowing their use in construction of models that support environmental decision making.

While RDF provides a way to represent the semantic structure of data and models, a software infrastructure to put RDF into action is not readily available. Until such an infrastructure becomes available, the following approach provides one method for achieving the desired result. The approach uses an *XML schema* [W3C, 2001b] to design and implement the specification of the model interface, rather than RDF.

5.2. From Interface Specification to Automatic Code Generation using an XML Schema

Models process data. For example, dynamic models take initial states, parameter values and time series as inputs, and generally produce output time series. The user must know the input data format to feed the model and the output data format to interpret the result. To embed such a model in another application, or publish it as an applet in a web page, a developer must write "drivers" (data conversion programs) to transform the user input into the correct input format. This task can be tedious and time consuming and would benefit from automation. We offer a solution to this based on XML documents and XML schemas.

The first step was to write an XML schema to define the syntax rules to specify the model interface. An XML schema describes the elements that are used to build XML data files. Another option would have been to write an XML Document Type Definition (DTD) [W3C, 2000] to write the grammar rules of the model interface. However the XML DTD is limited, since it does not allow specification of the exact format of a string, which is fundamental to define the different data types, nor does it allow definition of repeated elements, which are essential for time series representation.

In Figure 2 we show a fragment of the XML schema that defines the format of the interface for the hydrological rainfall-runoff model IHACRES [Jakeman and Hornberger, 1993]. Note the "record-like" structure of the element `Input Data Element` in the XML schema - it is composed of two measurements, rainfall and temperature, and the date (day, month, year) of sampling. The `Input Data Element` can represent a time series using the `<sequence>` tag in the schema to indicate the possibility of repeated elements.

```
<complexType name="IH_input">
<sequence>
<element name="InputDataElement">
  <complexType>
    <element name="Date">
      <complexType>
        <attribute name="day"
                   type="integer"/>
        <attribute name="month"
                   type="integer"/>
        <attribute name="year"
                   type="integer"/>
      </complexType>
    </element>
    <element name="Temperature">
      <complexType>
        <attribute    name="centigrades"
                         type="double"/>
      </complexType>
    </element>
    <element name="Rainfall">
      <complexType>
        <attribute name="quantity"
                   type="double"/>
      </complexType>
    </element>
  </complexType>
</element>
…
```

Figure 2. A fragment of an XML schema.

Provided every model has a schema, such as that shown in Figure 2, for its input and output interface it is possible to generate XML data to feed it, and also to interpret the XML output. Using the XML schema indicated in Figure 3 it is possible to generate the XML document shown in Figure 4.

In the following, we describe the other necessary component of this approach - how to parse XML data and map it into the model memory, that is, how we automatically generate the model driver starting from the XML schema.

5.3. Generating Model Drivers with the "Schema Mapper"

The following assumes model source code written in Java, but, in general, the model can be written in any language, provided that an interface between the model original language and Java is provided.

For sake of simplicity and ease of development, we used the Java Remote Method Invocation (RMI) [Sun Microsystems, 1999] as the backbone of our client-server application. It would be relatively easy to switch to CORBA, thus accessing the wealth of language interfaces it provides.

The Schema Mapper is a Java application that reads the XML schema and generates a set of Java classes, which are then used by the Marshaller and the Unmarshaller, as described in the XML Data Binding Specification [Sun Microsystems, 2001]. The Unmarshaller parses the XML document and maps the data into Java Objects. The Marshaller, on the other side, "flattens" Java Objects in a text format as an XML document. The relationships between the Schema Mapper, the Marshaller, and the Unmarshaller are described in Figure 3. The classes generated by the Schema Mapper provide a set of methods to access, in both read and write mode, all the components of the model interface, as specified in the XML interface specification.

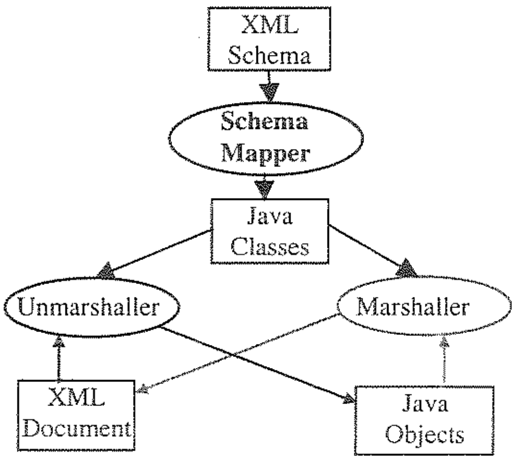

Figure 3. The XML schema is used to generate a set of Java classes, which enable data "marshalling" and "unmarshalling".

```
<?xml version="1.0" encoding="UTF-8"?>
<simulation>
  <parameters a1="0.1" a2="0.2"
b1="0.4" tmax="0.7" b0="0.3" tau="0.6"
c="0.5" />
  <InputDataElement>
    <rainfall quantity="5.0" />
    <date month="2" day="3" year="1"
/>
    <temperature centigrade="4.0" />
  </InputDataElement >
  <InputDataElement >
    <rainfall quantity="10.0" />
    <date month="7" day="8" year="6"
/>
    <temperature centigrade="9.0" />
  </ InputDataElement >
...
```

Figure 4. A fragment of an XML document.

These classes must be distributed both to the model server and to the model client. In the case of the IHACRES model, the Schema Mapper will automatically generate the classes `IH_input` and `IH_output` (from the XML schema in Figure 2, where the element `IH_input` is defined; the element `IH_output` is defined later in the schema in an analogous way). These classes will contain the methods required to read and write all the fields of the IHACRES model interface.

5.4. Writing a Model Server

The model author must write only a very short piece of code, that is, a method that can be invoked from an external application. This method will take as input an object of type `IH_input` and return an object of type `IH_Output`.

The clear advantage is that the model driver must be *written only once by the model developer*. There is no need for reverse engineering by third party modellers to re-use this model. In our case study, the IHACRES model, it is sufficient to write the `IH_run` method that maps the data structures contained in the `IH_input` to the internal data structures of the simulation model and vice-versa for the output.

Once the developer has written this simple interface, the model can be either distributed as java bytecode or published on an Internet model server. We have written an RMI application that acts as a model server, and that publishes models on the network. This model server can provide remote access to many models, due to the Java RMI architecture.

5.5. Writing a Model Client

While the model author has to do a bit of work to use the `IH_input` and `IH_output` classes, the task on the client side is much simpler and could be easily automated. The model user writes an application that calls the "run" method of the server (in our case,

```
public class IhacresClient {
public static void main(String args[]) {
  try {
   // looks for an Ihacres model
   // on the RMI naming service
     Ihacres ih = (Ihacres)Naming.lookup(
       "rmi://www.modelserver.com/Ihacres
       Service");
     // load the XML data from file
     IH_Input inputdata = getIn-
      put(args[0] + ".xml");
     // call the remote model
     IH_Output outputdata =
      ih.RunIhacres(inputdata);
} // end try
...
```

Figure 5. A fragment of the model client code.

IH_run). The application is very simple and the relevant code fragment is reported in Figure 5.

The application is structured as follows: 1) unmarshal the data from an XML file, which complies to the XML schema of the model, into an input interface java object; 2) pass the java object to the run method; 3) get back a java object structured according to the output interface, and 4) marshall this object into an XML file which is then further processed (e.g. graphic/text display of data).

6. CONCLUSIONS

With increases in the provision of distributed modelling components, it has become apparent that simply selecting a communication protocol will not solve the problems of communication and execution using distributed services, such as across the Internet. One of the key problems lies in the area of separating models from their interfaces, and specific input and output formats, irrespective of a selected communication protocol. The approach presented here, whereby an XML-schema describes the input and output interface for a component and a schema-mapper application generates the associated data management code, provides a method that relieves the coding burden on the developer and supports a consistent method for encapsulating environmental models.

7. REFERENCES

Berners-Lee, T., J. Hendler, and O. Lassila, The semantic web. *Scientific American*, May 2001.

Booch, G., J. Rumbaugh and I. Jacobson, *The unified modeling language user guide.* Addison-Wesley, Reading, 482 pp., 1999

Geoffrion, A.M., An Introduction to Structured Modeling. *Management Science*, 33(5), 547-588, 1987.

Gruber, T.R., A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199-220, 1993.

Hoch, R., T. Gabele and J. Benz, Towards a standard for documentation of mathematical models in ecology. *Ecological Modelling*, 113: 3-12, 1998.

Jakeman, A.J. and G.M. Hornberger, How Much Complexity is warranted in a Rainfall-Runoff Model?, *Water Resources Research*, 29(8), 2637-2649, 1993.

Leavesley, G.H., S.L. Markstrom, M.S. Brewer and R.J. Viger, The modular modeling system (MMS) - The physical process modeling component of a database-centred decision support system for water and power management. *Water, Air and Soil Pollution*, 90(1-2),303-311, 1996.

Mattsson, S.E., H. Elmqvist and M. Otter, Physical system modeling with Modelica. *Control Engineering Practice*, 6, 501-510, 1998.

Microsoft Corporation, *Microsoft Windows version 3.1 Software Development Kit Guide to Programming.* MSDN Library Archive Edition, 1987-1992.

Nelson, B.J., Remote Procedure Call, *XEROX PARC CSL-81-9*, May 1981.

Petzold, C., *Programming Windows 3.1.* 3d ed. Microsoft Press, MSDN Library Archive Edition, 1992.

Raj, G.S., A Detailed Comparison of CORBA, DCOM and Java/RMI (with specific code examples), http://www.execpc.com/~gopalan/misc/compare.html, 1998.

Rizzoli, A.E., J.R. Davis, and D.J. Abel, A model management system for model integration and reuse, *Decision Support Systems*, 24, 127-144, 1998.

Sun Microsystems, JSR-31 - XML Data Binding Specification, http://jcp.org/jsr/detail/031.jsp, 2001.

Sun Microsystems, RMI: Java Remote Method Invocation - Distributed Computing for Java http://java.sun.com/marketing/collateral/javarmi.html, 1999.

Sydelko, P.J., K.A. Majerus, J.E. Dolph and T.N. Taxon, A dynamic object-oriented architecture approach to ecosystem modeling and simulation, American Society of Photogrammetry and Remote Sensing Annual Conference, Portland, OR., USA, 410-421, 1999.

Tallman, O. and J.B. Kain, COM versus CORBA: a decision framework. *Distributed Computing*, September-December 1998.

W3C, Extensible Markup Language (XML) 1.0 (Second Edition) http://www.w3.org/TR/2000/REC-xml-20001006, 2000.

W3C, Semantic Web Activity: Resource Description Framework (RDF), http://www.w3.org/RDF/, 2001a.

W3C, XML Schema. http://www.w3.org/XML/Schema, 2001b.