Development of Multi-Framework Model Components

Robert M Argent[a] and Andrea E Rizzoli[b]

[a] Cooperative Research Centre for Catchment Hydrology, The University of Melbourne, 3010, Australia. R.Argent@unimelb.edu.au

[b] IDSIA, Manno, Switzerland, andrea@idsia.ch

Abstract: A number of environmental modelling frameworks have been developed recently, and plans for new frameworks are under way. Examples such as TIME, OpenMI, SME and OMS share an approach to environmental modelling based on model components, and offer improved model development and deployment. These approaches have methods for ensuring model component-linking compatibility using manual and machine processes either internal or external to the model component. Examples include matching output to input and checking data type compatibility. Semantic integration is also possible, such as with the OpenMI, where a component requests and receives particular data. However, each framework does model component checking in a different way and interoperability between model components of different frameworks is limited. To improve the use of model components it is necessary to consider the development of multi-framework model components (MFMC). Existing software standards enable communication at a low level, but many problems remain at high levels. This paper discusses development of an MFMC in each of TIME and the OMS, that can be accessed from the other framework. Additionally, the requirements for further framework compatibility, such as the OpenMI, are considered. Six main approaches are described, covering methods relevant to both between- and cross-platform compatibility, which range from re-implementation, through Web Services, to declarative modelling. Web services are suggested as a viable option for the problem considered here, although the other techniques warrant further investigation in particular cases.

Keywords: Component-based modelling; Modelling frameworks; Model development, Multi-framework model components.

1. INTRODUCTION

Development of new environmental modelling ideas has been taking place over recent years. Two primary ideas, of simplified modelling focussing on dominant processes and using parsimonious methods, are being combined with modular or component-based modelling ideas. In this approach components are created that represent discrete system functions, and these functions are then combined flexibly to address particular problems using a tailored modelling approach.

These developments have been associated with the production of various support tools, the most significant of which are the modelling frameworks, or development environments, within which developers can build and test components, and click components together, and together with data and visualisation components, to build multi-component models. One of the problems in this approach, and a major problem of the environmental modelling science world, is that a component build by and for one framework is not directly compatible with another framework.

The problems of making algorithms from one component into a useable form for other components can be addressed using a range of approaches. This paper explores these approaches for the development of model components for porting between three specific modelling frameworks – the OpenMI, OMS, and TIME. Furthermore, the semantic issues are explored further to identify some of the critical issues that need to be addressed before appropriate components for use in multiple frameworks can be identified and developed.

2. PORTING MODEL COMPONENTS

We have identified at least six methods to port a model component across modelling frameworks, but we must first distinguish between whether or not the modelling frameworks are implemented using the same software platform.

Where components are based on the same platform, with binary level compatibility, porting options are a) *interface extension*, and b) *on-demand data binding*. For different platforms the options are: c) *manual re-implementation*; d) *cross-compilation*; f) *web-servicing*, with declarative interfaces; and e) *declarative modelling.*

We now examine the advantages and the disadvantages of these approaches.

2.1 Binary Compatible Modelling Frameworks

This is the case where the modelling frameworks use the same software architecture (e.g. two frameworks are implemented using the .NET software architecture).

2.1.1 Interface Extension

This approach proposes development of a different model component interface for each framework. The model component code stays mostly unchanged when being used in a new framework, and an interface or wrapper has to be developed. This activity, to be done by hand, may take a lot or little time depending on the requirements and features of the frameworks.

2.1.2 On-Demand (XML) Data Binding

This approach involves the specification of the model component interface by means of a standard XML description. The description is then parsed by software tools that automatically generate the source code for the model component interface, targeted to the appropriate modelling framework. The source code is then compiled and linked into the modelling framework. This approach has been called XML data binding (McLaughlin, 2002)

This option, while possibly requiring more effort up-front in the definition of an XML schema to define the interface data formats, may be better in the long term, as new components can be readily created to be multi-framework compatible. It also has the advantage of being easy to implement both in J2EE, using JAXB (Java Architecture for Data Binding: http://java.sun.com/xml/jaxb) and in .NET, using `xsd`, the W3C Schema Definition Tool. A nice feature of this approach is that the same XML interface can be used to generate code for modelling frameworks based on both Java and C#.

2.2 Heterogeneous Modelling Frameworks

This is the case where the model components are designed and implemented to work under incompatible modelling frameworks (e.g. one modelling framework is based on the .NET architecture, and one on J2EE).

2.2.1 Manual Re-implementation

This is the oldest and most common approach to make a model component re-usable in another modelling framework: make public the algorithms, code and explanation of the science in the component. This model component can then be manually transformed for use in another framework. This transformation will be of varying complexity depending on the nature of the algorithms, the language/s used for the components and the features and capacities of the alternative frameworks. It is the most straightforward approach as it requires only a single unit block of work to produce a workable solution. However, over time this may become the most time consuming, as every model component needs to be broken open, re-written, de-bugged and tested.

2.2.2 Cross-Compilation and Translation

To limit the programming effort, model components written for one modelling framework can be ported to another using cross-compilation and translation tools. For instance, JNBridge (www.jnbridge.com), an implementation of the .NET Remoting wire-level protocol, and Remotesoft's Java.NET (www.remotesoft.com) allow translation of a model component written in Java into a corresponding C# component. However, in this approach the interfaces may be not compliant with the specifications issued by the target modelling framework, and therefore interface extension or XML data binding might be required.

2.2.3 Web-Servicing

Model components can be implemented as web-services, providing a published interface that is remotely callable. The use of a standard protocol, such as SOAP, to exchange data, and the fact that the model component resides on a remote server, enables interoperability among non binary-

compatible modelling frameworks. This approach requires that the modelling framework knows about the interface of the remote model component. Thus, we fall back into one of the two approaches for binary compatible systems: interface extension or XML data binding. An example of the latter can be found in Rizzoli *et al.* (2001).

2.2.4 Declarative Modelling

This is the most generic approach to component porting, but is also the most difficult to bring into common practice and to implement.

The basic idea is to shift from a *procedural* approach to modelling to a *declarative* one. In the procedural approach, models are written as sets of instructions for simulating the model, written in the programming language of the modelling framework, for instance C# or Java. In the declarative approach, models are represented as a set of statements defining the structure of the model. These statements are written in a text file, using a standard and open format (again, an XML schema can be useful). Experiences of declarative modelling are found in the Simile modelling environment (Muetzelfeldt and Massheder, 2003) and in the Integrating Modelling Architecture (Villa, 2000). A declarative model can be processed and transformed automatically, by means of a model compiler, into a model component targeting any modelling framework.

The main disadvantage with this approach is that every modelling framework needs to adhere to a standard and common declarative modelling language, which currently exists only in some domain. A major advantage is the ability to link elements declared in the model with entities declared in distributed ontologies, thus reducing the risk of ambiguities and misuse of data, which are quite frequent in all other approaches

This overview provides a range of techniques for developing multi-framework model components, with the primary determinants being the platform and software structure used by the associated modelling frameworks. The following explores development of a simple component model in the TIME framework, and the requirements for operating this component in the OMS framework and making the component compatible with the OpenMI.

3. TIME – THE INVISIBLE MODELLING ENVIRONMENT

TIME is an environmental modelling framework constructed using .NET (Rahman *et al.*, 2003). Its primary features are a thin architecture and a strong capability to use model metadata. By adding metadata about parameters and variables, control of the model by the TIME system can be automated in many ways. One of these is to support automatic creation of user interfaces. If a model has a declared parameter with metadata regarding range and default value, then a user interface consisting of, say, a slider bar ranging between the extremes of the allowable range, can be readily generated. Also, if metadata information on data types relating to inputs and outputs is given, then the linking behaviour of two component models can also be controlled by an intelligent model management system, by means of XML data binding. Another feature is the use of component technology, in that components are designed to be multi-purpose, either being run with command line techniques, to support multiple runs for, say, stochastic modelling, run remotely, for Web Service applications, or have attached an automatic GUI, providing control over parameter values for scenario exploration.

3.1 The Component

The selected component was one of the more simple components that can be developed, and one which has often been used for examples in development of the TIME modelling environment (Rahman *et al.*, 2003). This is a rainfall runoff model that uses a runoff ratio to create runoff from rainfall (equation 1), which can be run once or for every time step in a temporally dynamic simulation, and which has spatial application to a point, polygon or any cell in a raster.

$$Ro = RR * (Rain - ET) \quad (1)$$

Where

Ro is runoff

RR is runoff ratio

Rain is precipitation

ET is evapotranspiration

In a Java version of the component in TIME, the component code is as follows:

```
package
TIME.Models.Examples.RainRunoffCoefft;

import TIME.Core.*;

/**
 * Rainfall runoff coefft test.
 * @author David Verrelli
 */
```

```
public class RunoffCoeff extends
Model
{
  /** @attribute Input() */
  double Rainfall, PET;

  /** @attribute Parameter()
  *  @attribute Minimum(0.0)
  *  @attribute Maximum(1.0) */
  double Coeff;

  /** @attribute Output() */
  double Runoff;

  //Constructor
  public RunoffCoeff()
  {
   super();  //This is implicit
  }

  public void runTimeStep()
  //This runs at every timestep.
   {
   Runoff = Coeff * Math.max(Rainfall
   - PET, 0.0);
   }
 }
```

This kind of straightforward TIME routine has three basic sections:

- An initiation section, containing basic declarations of the package, any external library requirements, class declaration and, if required, parent class declaration
- declaration section for parameters and variables, and
- run section, wherein lies the core code for the algorithm

The component above extends the abstract parent class `Model`, which controls the timestepping of the model through a `runTimeStep()` abstract method, and also keeps track of the progress of the model run. `Model` has a `Subject` parent class that, together with an `Observer` class, provides a communication structure built on the Observer pattern (Gamma *et al.*, 1994). `Data` is also an abstract parent class, sharing the `Subject` parent class with `Model`. All child classes of `Data` implement the `setItem (in i: int, in val: double)` method and the `item( in i: int)` query to provide a common data interface (Rahman *et al.*, 2003).

A similar code structure to the above could be used to write the model component in other languages, such as C#, FORTRAN95, Eiffel, and Visual Basic, each of which would access the same parent classes and methods.

4. THE OBJECT MODELING SYSTEM - OMS

The OMS is a modelling framework written in Java, developed by members of the USGS, USDA and Friedrich-Schiller University. Modular modelling lies at the core of the OMS with a structure that clearly separates the system core, system extensions, and the user interface. The core provides the functionality for basic module operation, data handling, input-output, visualisation and remote access. Extensions cover features such as module development, application construction and the management of 'dictionaries' that covers, to some degree, the semantics of interaction between modules.

A model component, which performs a similar computation to the one examined above, can be implemented in the OMS modelling framework. The model component also contains three basic sections: an initialisation section, a run section and a clean-up or handover section, as shown below.

```
/**
 * RunOffCoeff.java
 * @author adapted from Sven Kralisch
 */

package de.unijena.jenamodel;
import org.omscentral.data.*;
import org.omscentral.model.*;
import java.io.ObjectInputStream;
import org.j2k_io.j2kBinFileHeader;

public class RunOffCoeff extends
OMSComponent {
  transient  OMSTimeInterval time;
  transient  OMSEntitySet es;

  /** @attribute Input() */
  transient public double Rainfall =
0;
  transient public double PET = 0;

  /** @attribute Parameter() */
  transient public double Coeff = 0.2;

  /** @attribute Output() */
  transient public double Runoff = 0;

  // Constructor
  public RunOffCoeff() {}
  public void register() {}
  public int init() {return 0;}
  private void initData() {
    OMSEntity currentEntity =
this.es.current;

  try {
```

```
  Coeff = Double.parseDouble((String)
currentEntity.getAttribute("Coeff"));
  Rainfall =
Double.parseDouble((String)
currentEntity.getAttribute("Rainfall"
));
  PET = Double.parseDouble((String)
currentEntity.getAttribute("PET"));

}
 catch
(org.omscentral.data.OMSEntity.NoSuch
AttributeException nsae)
{System.out.println("Attribute not
found");
}
}

  public int run() {
    initData();
    double Temp = 0.0;
    if(Rainfall> PET)
      Temp = Rainfall-PET;
    else
      Temp = 0.0;
    Runoff = Coeff * Temp;
cleanup();
return 0;
  }

public boolean cleanUp(){
  OMSEntity currentEntity =
this.es.current;
try {
currentEntity.setAttribute("Runoff",
new Double(Runoff));
}
 catch
(org.omscentral.data.OMSEntity.NoSuch
AttributeException nsae) {
System.out.println("Attribute not
found");
}
return true;
}
}
```

Comparison between this component and that built for TIME highlights some similarities and differences between the two. In terms of control structures and looping or stepping through time and space, both have a similar approach, with the 'run part' of the component, containing the operational algorithm, being separate from these structures. The differences arise from more framework-specific attributes, such as the type and arrangement of parent classes.

These differences influence the selection of an approach for accessing a TIME component from the OMS, and indicate that the web service approach offers the most efficient approach in this case. Undertaking this would be done through the core support in OMS for accessing remote components. In the opposite direction, that of operating an OMS component from within TIME, a web service option is also suggested. The .NET system, upon which TIME is based, has native support for remote component access, and implementation of these within TIME would be straightforward.

This straightforward use of a web service approach is a reasonably elegant approach to framework interaction, although it has the difficulties of remote operation. Despite these difficulties, web services are growing in use in areas outside of environmental modelling, and there is a considerable potential for providing modelling services in this way.

An alternative approach is to support a common component interface within each framework, and to create local component interfaces that conform to this standard.

5. THE OPEN MODELLING INTERFACE (OpenMI)

The OpenMI is an approach to components and models that focusses on the linking of models, rather than internal model or component operation or construction (Gijsbers *et al.*, 2002). The OpenMI has arisen from consideration of existing environmental modelling and the needs of the European Water Framework Directive, and consists of a set of interfaces and concrete classes that specify the requesting and exchange of data between models or components.

In the OpenMI a model component, identified as a *linkable* component, is populated with persistent data by an initialisation method (`Initialise()`), then data are obtained using a `GetValues()` run method invoked from a calling component. The main difference from the above two components lies in the pull mechanism, which allows a calling component to perform the computation and extract the results with one call to the `GetValues()` method.

Given this, the OpenMI approach could be used to support component interaction between different frameworks. To achieve this, each framework would separately contain the methods and classes necessary to use the OpenMI, and components, created as discrete objects with published interfaces, would be made compatible with OpenMI. This approach, or that of web services, provides a way of allowing components constructed in different frameworks to be used together in a confederated model. Additional difficulties arise,

however, over the semantics of the data exchange between components.

6. SEMANTIC CONSIDERATIONS

In component-based modelling a common problem is that of the *meaning* of data, variable and parameter names that are used by and passed between components. When components are built by different people using different frameworks, and then offered for use by other people with other frameworks, misunderstandings can arise unless a clear meaning is given to data, parameters and variables. To do this a clear language needs to be established covering not only these, but also the modelled concepts.

This problem is well recognised within disciplines, and approaches such as formal ontologies have been proposed. As environmental management expands and becomes more multi-disciplinary, components are necessarily re-constructed to fit new scales or conceptual structures. This brings with it an increased problem on accurate information exchange between components. Technical solutions to this include a greater use of metadata and meta-information, supported by XML. In particular, the declarative modelling approach appears to be well suited to this purpose. The Semantic Web initiative of the W3C (http://www.semanticweb.org) also offers some hope, through improved definition and communication of the 'meaning' of web information.

7. CONCLUSION

The area of multi-framework model components is one of considerable challenge, although the range of technical solutions listed here, and the development styles of those working on environmental modelling frameworks offers an indication that the difficulties are not insurmountable. Future key areas for investigation and practice include testing of the approaches listed here, identification of the advantages and disadvantages of implementation, and extension of these ideas beyond the limited range of frameworks considered here.

7. REFERENCES

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley: Reading, MA.

Gijsbers, P. J. A., Moore, R. V., Tindall, C. I., 2002. HarmonIT: Towards OMI, an Open Modelling Interface and Environment to harmonise European developments in water related simulation software, Hydroinformatics 2002. Fifth International Conference on Hydroinformatics: Cardiff, UK, IAHR, p. 7pp.

McLaughlin, B. 2002, *Java & XML Data Binding*, O'Reilly.

Muetzfeldt, R., Massheder, J., 2003. The Simile visual modelling environment. *European Journal of Agronomy*, 18, 345-358.

Rahman, J. M., Seaton, S. P., Perraud, J.-M., Hotham, H., Verrelli, D. I., Coleman, J. R., 2003. It's TIME for a new environmental modelling framework. In: Post, D. A., (Ed.), MODSIM 2003 International Congress on Modelling and Simulation: Townsville, Modelling and Simulation Society of Australia and New Zealand Inc., p. 1727-1732.

Rizzoli, A.E., Argent, R.M., Manglaviti, M., Mutti, M. 2001. Encapsulating Environmental Models And Data: A Self-Consistent Approach. In: Ghassemi, F. et al.. (eds). Proceedings of *MODSIM 2001, International Congress on Modelling and Simulation*, 10-13 December 2001, The Australian National University, Canberra, Australia, 1649-1654.

Villa, F., 2001. Integrating modelling architecture: a declarative framework for multi-paradigm, multi-scale ecological modelling. *Ecological Modelling*, 137 (1), 23-42.