Convergence in integrated modeling frameworks

[1]Frits van Evert, [2]Dean Holzworth, [3]Robert Muetzelfeldt, [4]Andrea Rizzoli, [5]Ferdinando Villa

[1]Plant Research International, Wageningen, The Netherlands, frits.vanevert@wur.nl
[2]APSRU, Toowoomba, Australia
[3]Simulistics, Edinburgh, United Kingdom
[4]IDSIA, Manno, Switzerland
[5]University of Vermont, USA

Keywords: *modeling framework; semantic linking.*

EXTENDED ABSTRACT

There are three important reasons for the use of modeling frameworks in environmental science: dealing with complexity, re-using modules for different models, and providing support for commonly needed services. Literally dozens of modeling frameworks are being used by environmental scientists, several of which are under active development. It was our objective to determine just how much common ground there is between these frameworks. A review of how the various frameworks support decomposition of the modeling problem, specification of the model (including sub-models and compositions of sub-models), event-handling, numerical integration and run-time execution of models, revealed that it is helpful to make a distinction between implementation-level and modeling-level frameworks. An implementation-level framework's primary purpose is to link existing model implementations. With an implementation-level framework, the user is responsible for fully specifying links between modules and ensuring their correctness with respect to intention and logic. Differences between the various implementation-level frameworks are relatively unimportant. However, representing even a moderately complex system with an implementation-level framework requires the user to spend much effort to make sure that the necessary modules are properly connected. Modeling-level frameworks attempt to unburden the user by allowing domain-specific terms to be used to specify which models or modules should be used and how they should be linked. There is a direct, if usually implicit, link between both kinds of frameworks: before a model specified with a modeling-level framework can be executed, it must be translated into a general-purpose programming language, which will be done using an implementation-level framework. It is argued that users and builders of frameworks in environmental science will benefit from making the distinction between implementation-level and modeling-level frameworks and that they should be explicit about the implementation-level frameworks targeted by the code generators of modeling-level frameworks. If at all possible, a widely accepted target implementation-level framework should be chosen. Then, at some future time, we may see re-use and extension of an existing framework instead of the creation of YAMF (Yet Another Modeling Framework).

1. INTRODUCTION

There are three important reasons for the use of modeling frameworks in environmental science. First, the systems of interest to environmental scientists, and models of these systems, are complex. An effective way of dealing with complexity is to decompose the system into smaller systems, and possibly repeating this process, until sub-systems of manageable complexity have been identified. Supporting the composition of a model of the system of interest by combining models of sub-systems obtained in this way, is the most important task of a modeling framework. Second, it is often the case that two different systems of interest have one or more sub-systems in common. A modeling framework enables the creation of models of such systems by drawing on a common pool of sub-models. Third, even modules that represent completely different biological or physical systems, have common characteristics and requirements, both at the level of modeling (distribution in space and time) and at the level of implementation (I/O, numerical integration). A framework can provide support for such common requirements.

To date, literally dozens of frameworks have been proposed for use by environmental scientists, with several under active development at the time of writing. Each new framework claims some unique merits and while it is probably true that a new framework is needed from time to time, it will be sad indeed if the field of environmental science is so fragmented that dozens of frameworks are needed to satisfy all its modeling requirements. In the light of this, we consider in this paper several of these frameworks and try to determine just how much common ground there is between them. In our (somewhat arbitrary) selection of frameworks to be considered, we are slightly biased toward agronomic and hydrological simulation, but the analysis should be applicable outside these areas. The following frameworks are considered: IMA (Villa, 2005), OpenMI (Moore et al., 2004), APSIM (Keating et al., 2003), MODCOM (Hillyer et al., 2003), TIME (Rahman et al., 2003), SELES (Fall and Fall, 2001), MMS (Leavesley et al., 1996), SME (Maxwell and Costanza, 1995) and FSE (Van Kraalingen, 1995). In the following sections we investigate how the various frameworks support decomposition of the modeling problem, specification of the model (including sub-models and compositions of sub-models), event-handling, numerical integration and run-time execution of models.

2. DECOMPOSITION OF THE MODELING PROBLEM

Modeling frameworks support construction of model implementations by connecting two or more sub-model implementations. The sub-model implementations or modules are the result of a decomposition of the modeling problem. How to perform the composition is left to the user of the framework. Various authors have adopted an approach to model decomposition strongly influenced by software engineering principles. This results in the definition of sub-models such that there is much interaction within each sub-model, and little interaction between them. The most commonly selected strategy by far is to decompose by structure. A simulation can then be redefined by replacing one module with another module that represents the same part of the modeled system. This is true whether the modules represent crops (APSIM), catchments (TIME), or population dynamics of a species (SME). But other strategies are possible. For example, Van Evert et al. (2003) use two instances of the sub-model "DairyCows" to represent one group of animals during two (housing and grazing) periods of the year.

In summary, decomposition by structure is by far the most commonly used decomposition strategy.

3. MODEL SPECIFICATION: MODULES

Most frameworks take as the unit of decomposition the model function, adorned with such housekeeping info as number and names of parameters. An exception is OpenMI, where modules additionally contain machinery to buffer states and may perform spatial (dis)aggregation as well as averaging and interpolation over time.

Some frameworks work only with the interface of modules. In other words, they treat modules as black boxes, in keeping with the object-oriented paradigm which stresses encapsulation and focuses on behaviour. An advantage of this method is that modules can be developed and tested separately from the application software in which they will be deployed and can then be delivered in binary form: C^{++} classes (Tarsier, (Watson and Rahman, 2004)), Microsoft COM classes (MODCOM), .NET classes (OpenMI, TIME, and the .NET version of MODCOM), or Java classes (OpenMI).

From the above it is apparent that classes are the customary mapping of models to model implementations. This is appropriate, because even though a model itself is just a function, a model implementation needs to communicate, among other things, number and names of parameters to other program units. The resulting construct is best expressed as a class. Frameworks such as APSIM, MMS and FSE, where modules are typically written in Fortran, define modules as a set of functions, grouped in such a way that the effect is that of a class.

Some frameworks allow specification of modules in a run-time environment using a scripting

language (VBScript in MODCOM, MickL in ICMS (Rahman et al., 2004)). Such modules are still black-box modules because this script is not processed by the framework or by other modules.

In contrast to black-box modules, white-box modules provide processable information about the model to the framework. SME uses the Modular Modeling Language (Maxwell and Costanza, 1997), SELES and IMA also define declarative modeling languages. Other declarative languages used in modeling include Stella (ISEE Systems, Lebanon, NH, USA), Simile (Simulistics Ltd., Edinburgh, United Kingdom), and Modelica (*http://www.modelica.org*). The declarative specification of a model can be used to document the model, allow drill-down through composite models, and present models in graphical form. In addition, from a model defined in a declarative language, code can be generated that will run in specific frameworks (Argent and Rizzoli, 2004; Ford et al., 2005; Muetzelfeldt, 2002).

In summary, many frameworks use black-box modules. These are commonly expressed as classes, either at the source code level or at the binary level. Several frameworks use declaratively specified models. While this has some benefits at the level of module specification, we shall see in the next section that declaratively specified models are most useful in model composition.

4. MODEL SPECIFICATION: COMPOSITIONS

In a model composition, two or more modules (black-box or white-box) are connected. A black-box module has pre-defined inputs and outputs. This is typically true for white-box modules, too, although the fact that the full model specification is available in processable format means that the model could be inverted during the code generation phase that follows. IMA in particular makes a distinction between a model (with, for example, state variables and parameters) and a workflow element (with inputs and outputs).

A composition of black-box modules is bound to the execution platform that is targeted by its modules. On the other hand, a composition of white-box modules (sometimes called a "meta-model") can be deployed on different platforms by using appropriate code generators. For example, different SME translators target a supercomputer and an MPI-mediated cluster of machines. The meta-model frameworks considered in this paper (IMA, SME, and SELES) can also make use of black-box modules; this is indicative of the need to be able to incorporate "legacy"-modules or modules that are not easily expressed in the declarative modeling language of the meta-model's framework.

There are good reasons for using white-box modules and for using black-box modules. Not all applications require the more expressive white-box modules; consequently, no convergence to frameworks that support white-box modules is taking place.

5. SPECIFYING LINKS

When two model implementations are linked, the semantics of output and input (physical quantity and units, time, space, etc.) as well as the representation (integer, double, etc.) must match. Frameworks such as MODCOM and TIME place no restrictions on links on the basis of semantics or representation, although both attempt to guide the user by allowing textual mark-up of inputs and outputs. TIME additionally supports "Raster" and "TimeSeries" datatypes and provides a library of operations, but the framework does not allow for the expression of spatial relationships between modules. OpenMI has a mechanism that requires the discretization in time and space of the requested information to be specified when a link is created. If the information is not available at the requested discretization, the module can either employ framework-supplied methods to provide a mapping, or generate the mapping using its own methods. SME's MML and SELES's modeling language both allow grid-spatial semantics to be expressed. IMA is by far the most ambitious of the frameworks: it can load and use arbitrary ontologies (Guarino and Giaretta, 1995) to fully specify the semantics of all models. Machine reasoning is then used during the code generation phase to produce algorithms to solve the models, inserting converters (accumulators, unit translators, aggregators) where necessary to address scale mismatches such as different time and space granularities.

There is considerable interest in the semantics of model compositions. That is convergence. But there is a marked divergence in the approaches used to express the semantics of model compositions, with on the one hand frameworks such as TIME, OpenMI, SME and SELES that offer classes and interfaces to support to some extent the semantic specification of links, and on the other hand the ontology-based approach supported by IMA.

6. EXECUTING LINKS

During the execution of a simulation run, information is exchanged between modules. A variety of techniques is used by the various

frameworks. FSE uses variables that are accessible to all modules; HUME (Kage and Stützel, 1999) uses pointers to object members. Both methods are fast, but the first requires source-code level linking and the second introduces tight dependencies between modules. APSIM and MMS use a black-board mechanism where modules publish and retrieve information through a functional interface. This allows linking of executable modules, but incurs significant overhead. TIME and MODCOM achieve linking through pointers to objects. This incurs little overhead and is very flexible.

Declaratively represented models can be linked by generating source code in which the linkages are already established. The translated ("flat") form of a Modelica or Simile model is an example of this; how links are represented in the generated executable code depends on the code generator used. Links in IMA-generated executables are represented as pointers to objects.

Implementation-level links between modules are most commonly represented by pointers to objects in an object-oriented framework.

7. NUMERICAL INTEGRATION

The various modules of a composite model implementation represent different parts of the same system. If these modules contain differential equations and represent closely coupled parts of the system, it may be desirable to integrate them together, as in the case of populations of a predator and a prey occupying the same territory. In other cases, the problem at hand may call for modules to be integrated with different step sizes and/or algorithms.

FSE offers several integrators but can use only one in a given simulation; the chosen integrator is used for all modules. MODCOM offers several integrator implementations; any number can be used in a given simulation; and any number of modules can be specified to be integrated together. OpenMI and TIME don't offer explicit support to integrate modules together, but the effect can be achieved by compositing two or more modules.

In summary, whether support for integration is offered, and the different ways and methods of integration supported, are features of a particular framework.

8. EVENTS

Three kinds of events can be recognized in simulation: time-, discrete- and state-events. Pritsker (1986) describes a simple yet realistic model with all three types of events; see Zeigler et al. (2000) for a full theory. Many frameworks allow for time events ("plowing will take place on 15 October") as well as discrete events. FSE (version 2.1) and MODCOM handle state events. Handling state events is not completely straightforward in OpenMI, but at one of the training workshops the first author was shown that it can be done. IMA can handle events by using an event ontology and appropriate software; however, this has not been implemented to date.

Events are required to adequately model many systems, yet not all frameworks considered are able to handle the full range of events. The original version of FSE was not able to handle state events, but the current one is.

9. RUN SIMULATION

Running a simulation comprising two or more modules involves coordinating the repeated execution of the modules (applying the transfer function of the model implemented by the module). The order of execution of modules is determined by dependencies between modules and discretization of time and space. The use of an event list, where an event specifies which module needs to be run at which time, is common to many frameworks (SME, MODCOM, TIME). The event list may be simply a loop specifying time steps of equal size, with all modules being called at each time step (FSE), or it may be a dynamic list, allowing events spaced at arbitrary intervals, as well as insertion and deletion of events (TIME, MODCOM). In IMA an event list is created by the runtime to account for the discretization of the states implied by the observation contexts, such as time and space, defined by the modeller. Unique among modelling frameworks, IMA provides a generalization of the observation context that extends beyond time and space; as an example, an input parameter can be defined to have two alternative values, according to two different studies that have measured it. This multiplicity propagates automatically through the model structure and results in the calculation of two different sets of results, one per each study. The semantically explicit IMA allows independent developers to extend the observation context semantics and provide their own views of them, including alternative conceptualizations of time and space.

OpenMI stands apart from the other frameworks in that it does not employ an event list. Instead it uses a pull-mechanism where each module performs calculations in response to requests. In the process of fulfilling a request, a module may "pull" values from one or more other modules, leading to a cascade of pulling actions. When a module can calculate the requested value without pulling

values from other modules, it does so and then calculated results start flowing up the chain of requests.

An event list is by far the most common method of coordinating the execution of modules in the frameworks considered. OpenMI is the only framework that uses a different mechanism.

10. DISCUSSION AND CONCLUSIONS

With the feature-by-feature comparison of the preceding sections in hand, it becomes clear that the frameworks considered in this paper must be classified into two main groups. We will call these groups "implementation-level" frameworks and "modeling-level" frameworks. An implementation-level framework's primary purpose is to link existing model implementations. Thus, implementation-level frameworks (TIME, MODCOM, FSE, MMS) often treat their modules as black boxes. The user is responsible for specifying links between modules and ensuring their correctness with respect to intention and logic. Differences exist between the various implementation-level frameworks with regard to platform, language, efficiency, and the support of functionality such as state events. These differences are relatively unimportant and are related to implementation choices and completeness of the implementation; there are no fundamental differences between the implementation-level frameworks considered in this paper.

While the implementation-level frameworks between them can represent the systems currently of interest to environmental scientist (and have been used to do so), representing even a moderately complex system with such a framework requires the user to spend a lot of effort to make sure that the necessary modules are properly connected. The OpenMI framework significantly unburdens the user by offering a facility whereby a module can specify the discretization in time and space of the values it requests from another module. Somewhat more generally, yet still in the realm of implementation-level frameworks, John Bolte (2001, personal communication to Van Evert) has suggested using software tools to translate, for example, a spatially explicit model to an arrangement of objects in the MODCOM framework.

Modeling-level frameworks formalize the attempt to unburden the user by allowing domain-specific terms to be used to specify which models or modules should be used and how they should be linked. SME and SELES use declarative specifications of models in the domain of spatial (landscape) modeling. IMA extends this concept in that it is extensible to any domain, provided that such a domain is formalized through an ontology.

Thus, it seems that there is a clear separation between implementation-level and modeling-level frameworks, with users forced to choose between one or the other. In reality, however, there is a direct link between both kinds of frameworks. Before a model specified with a modeling-level framework can be executed, it must be translated into a general-purpose programming language. Because this is done through software, the output of such a process represents an implementation-level framework, whether intentionally or not. We are not aware of references that shed light on the frameworks targeted by the code generators of SME, SELES or IMA, which leads us to believe that the implementation-level frameworks used by these modeling-level frameworks should perhaps be called "accidental" frameworks.

The link between implementation-level and modeling-level frameworks can be made explicit by specifying the target implementation-level framework. Such an implementation-level framework must be capable of representing the entire range of discrete event and continuous simulation. For example, frameworks based on the DEVS formalism (Zeigler et al., 2000) are widely recognized to meet this goal. Interestingly, little reference is made to DEVS in environmental science literature, although Filippi and Bisgambiglia (2004) applied a DEVS-based implementation-level framework to problems in environmental science. Levytskyy et al. (2003) have presented work on translating Modelica models to the DEVS formalism using the Atom3 tool.

In conclusion, we suggest that users and builders of frameworks in environmental science will benefit from making the distinction between implementation-level and modeling-level frameworks, and by being explicit about the implementation-level frameworks targeted by the code generators of modeling-level frameworks. If at all possible, a widely accepted target implementation-level framework should be chosen. DEVS may be helpful in this respect. Then, at some future time, we may see re-use and extension of an existing framework instead of the creation of YAMF (Yet Another Modeling Framework).

11. ACKNOWLEDGEMENTS

This publication has been partially funded under the SEAMLESS Integrated Project (European Commission, DG Research, contract no. 010036-2). The comments of an anonymous reviewer were helpful in improving this paper.

12. REFERENCES

Argent, R.M., and A.E. Rizzoli. 2004. Development of Multi-Framework Model Components. International Environmental Modelling and Software Society Conference 2004, University of Osnabrück, Germany.

Fall, A., and J. Fall. 2001. A domain-specific language for models of landscape dynamics. Ecological Modelling 141:1-18.

Filippi, J.B., and P. Bisgambiglia. 2004. JDEVS: an implementation of a DEVS based formal framework for environmental modelling. Environmental Modelling & Software 19:261-274.

Ford, R.W., G.D. Riley, M.K. Bane, C.W. Armstrong, and T.L. Freeman. 2005. GCF: A General Coupling Framework. Concurrency and Computation: Practice and Experience.

Guarino, N., and P. Giaretta. 1995. p. 25-32 Towards Very Large Knowledge Bases. IOS Press.

Hillyer, C., J. Bolte, F. van Evert, and A. Lamaker. 2003. The ModCom modular simulation system. European Journal of Agronomy 18:333-343.

Kage, H., and H. Stützel. 1999. HUME: An objekt oriented component library for generic modular modelling of dynamic systems. Modelling cropping systems. Symposium of the European Society of Agronomy., Lleida, Spain.

Keating, B.A., P.S. Carberry, G.L. Hammer, M.E. Probert, M.J. Robertson, D. Holzworth, N.I. Huth, J.N.G. Hargreaves, H. Meinke, Z. Hochman, G. McLean, K. Verburg, V. Snow, J.P. Dimes, M. Silburn, E. Wang, S. Brown, K.L. Bristow, S. Asseng, S. Chapman, R.L. McCown, D.M. Freebairn, and C.J. Smith. 2003. An overview of APSIM, a model designed for farming systems simulation. European Journal Of Agronomy 18:267-288.

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

Levytskyy, A., E.J.H. Kerckhoffs, E. Posse, and H. Vangheluwe. 2003. Creating DEVS components with the meta-modelling tool AToM3. 15th European Simulation Symposium (ESS), October 2003. Delft, The Netherlands. Society for Modeling and Simulation International (SCS), Delft, The Netherlands.

Maxwell, T., and R. Costanza. 1995. Distributed Modular Spatial Ecosystem Modelling. International Journal of Computer Simulation 5:247–262.

Maxwell, T., and R. Costanza. 1997. A language for modular spatio-temporal simulation. Ecological Modelling 103:105-113.

Moore, R., I. Tindall, and D. Fortune. 2004. Update on the Harmonit Project – The OpenMI Standard for Model Linking 6th International Hydroinformatics Conference, Singapore.

Muetzelfeldt, R.M. 2002. Using Simile to make MODCOM components [Online] http://www.decmod.org/documents/simile_modcom/ (verified 11 August 2005).

Pritsker, A.A.B. 1986. Introduction to simulation and SLAM II. Halsted Press, New York.

Rahman, J.M., S.M. Cuddy, and F.G.R. Watson. 2004. Tarsier and ICMS: two approaches to framework development. Mathematics and Computers in Simulation 64:339-350.

Rahman, J.M., S.P. Seaton, J.-M. Perraud, H. Hotham, D.I. Verrelli, and J.R. Coleman. 2003. It's TIME for a New Environmental Modelling Framework MODSIM 2004 International Congress on Modelling and Simulation, Townsville, Australia.

Van Evert, F., H. Ten Berge, H. Van der Meer, B. Rutgers, T. Schut, and J. Ketelaars. 2003. FARMMIN: Modeling Crop-Livestock Nutrient Flows. ASA/CSSA/SSSA Annual Meetings, Denver, Colorado, USA.

Van Kraalingen, D.W.G. 1995. The FSE-system for crop simulation, version 2.1 Report no 1. C.T. de Wit Graduate School for Production Ecology, Wageningen University.

Villa, F. 2005. A semantic framework and software design to enable the transparent integration, reorganization and discovery of natural systems knowledge. Journal Of Intelligent Information Systems (Available online at http://ecoinformatics.uvm.edu/papers/villa_jiis.pdf). In press.

Watson, F.G.R., and J.M. Rahman. 2004. Tarsier: a practical software framework for model development, testing and deployment. Environmental Modelling & Software 19:245-260.

Zeigler, B.P., H. Praehofer, and T.G. Kim. 2000. Theory of modeling and simulation. 2nd ed. Academic Press, San Diego.