Jul 1st, 12:00 AM

Enhancing Model Reuse via Component-Centered Modeling Frameworks: the Vision and Example Realizations

Marcello Donatelli

Iacopo Cerrani

Davide Fanchini

Davide Fumagalli

Andrea-Emilio Rizzoli

International Environmental Modelling and Software Society (iEMSs)
2012 International Congress on Environmental Modelling and Software
Managing Resources of a Limited Planet, Sixth Biennial Meeting, Leipzig, Germany
R. Seppelt, A.A. Voinov, S. Lange, D. Bankamp (Eds.)
http://www.iemss.org/society/index.php/iemss-2012-proceedings

Enhancing Model Reuse via Component-Centered Modeling Frameworks: the Vision and Example Realizations

Marcello Donatelli[1,2], Iacopo Cerrani[1], Davide Fanchini[1], Davide Fumagalli[1], Andrea Emilio Rizzoli[3]

[1]European Commission, Joint Research Centre, Institute for Environment and Sustainability, Via E. Fermi 2749, I-21027 Ispra (VA), Italy
[2]CRA – Agriculture Research Council, Bologna, Italy
[3]Dalle Molle Institute for Artificial Intelligence, Manno, Switzerland
[1]marcello.donatelli@jrc.ec.europa.eu

Abstract: Model frameworks have represented a substantial step forward with respect to monolithic implementations of biophysical models. However, the diffusion of such frameworks, as model development environment, beyond the groups developing them has been very modest. The reusability of models has also proved to be modest. The reason for the latter was attributed also to the lack of standardization toward few frameworks. Emphasis has been placed on the framework and even new implementations of models have been made targeting a specific framework, likely assuming that the reusability of the model unit would have been directly proportional to the quality of the framework. In any case, the goal of several projects has been to make available the framework. Developers in the operational arena, but even in research, have reacted by developing their own framework. Still, the problem of model reuse has been largely unsolved; estimating that increasing the flexibility for reuse would have added a costly overhead, in terms of both complexity and possibly as lack of efficiency in the operational use.
The focus on frameworks has made software architects overlooking on the requirements of reusability per se of model units. The component oriented programming paradigm allows targeting intrinsic reusability of discrete model units, and makes room for enabling advances functionalities in simulation systems. This paper firstly present the abstract architecture of a component oriented framework articulated in independent layers: Model, Composition, and Configuration. The Application layer may link to any of these, to develop from simple console applications to sophisticated MVC applications. Proofs of concept are presented for each layer, including the BioMA framework of the European Commission used for agriculture and climate change studies.

Keywords: component oriented programming; model discretization; model composition

1. INTRODUCTION

Since many years, model frameworks have represented a substantial step forward with respect to monolithic implementations of biophysical models [e.g. Hillyer et al. 2003]. The separation of algorithms from data, the reusability of services such as I/O procedures and integration services, the target of isolating a modeling solution in a discrete unit have brought a solid advantage in the development of simulation systems as in OMS3 [David et al. 2011], and TIME [Rahman et al. 2003]. However, the diffusion of such frameworks, as model development environment, beyond the groups developing them, can be considered modest. The reusability of models has

also proved to be modest; a model unit for a given framework is not used in other frameworks. The reason for the latter appears to be attributed also to the lack of standardization toward few frameworks; the acronym YAMF – Yet Another Model Framework - was created summarizing the envisioned goal of standardizing as much as possible modelling frameworks. All of this has in fact created an obstacle for model reuse. Emphasis has been placed on the framework and even new implementations of models have been made targeting a specific framework, likely assuming that the reusability of the model unit would have been directly proportional to the quality of the framework. In any case, the goal has been to make available the framework. Also, legacy model boxes have been interfaced either to modeling frameworks via framework specific wrappers, of via intermediate layers like OpenMI [Gregersen et al. 2005, 2007].

Increasing the flexibility of use of a specific framework, even within a domain, requires an increasing level of complexity, which makes the cost of using it resources demanding, especially if the user targets the development of a specific application, which would likely use a subset of the features made available. Decreasing the flexibility, within a specific domain, increases its possible reusability by adding specific components and utilities, and by limiting the generic overhead claimed from a very generic framework. However, goals and functionalities, even within a domain, may partially vary across groups, and even the choices for I/O procedures and data persistence may vary. Developers in the operational arena, but even in research, have de facto reacted by developing their own framework, in spite of the constantly improved, in terms of quality, offer of modeling frameworks. Still, the problem of model reuse has been largely unsolved, and two possibilities have been at hand: either 1) the use of discrete units of software, often being strongly limited by the fact that such units were not developed for composition, hence not making available a number of possible functionalities, summarized by frameworks which have as explicit target purely linking data from one component to another, or 2) reimplementation.

The focus on frameworks has made software architects at least partially overlooking on the requirements of reusability per se of model units. Also, if the development of a customized framework may be estimated to some extent as unavoidable, building a framework based itself on framework-independent components (e.g. plugins which may be reused via an adapter) would facilitate this task. The component oriented programming paradigm allows targeting intrinsic reusability of discrete model units, and makes room for enabling advances functionalities in simulation systems. It also impacts on (specialized) model frameworks development, causing a shift in requirements and consequently an impact on architecture. However, the component oriented programming is not sufficient per se in fostering model units reuse.

This paper explores the shift of paradigm represented by putting the focus on components, rather than on frameworks, and presents concept and concrete examples, used also at operational level.

2. FROM MODELS TO VIEWERS

A component can be defined as: "*A unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject by composition by third parties*" [Skyperski et al. 2002]. The tools and utilities of a framework must respect the requirement of context specific dependencies, to enhance reusability. If the design of a framework is based on layers, its building block units must avoid dependencies from the framework, and layers must be independent from each other.

According to the envisioned use and to the features requested for a specific software realization, we propose a simulation system to be discretized in layers, each with its own features and requirements. Such layers can be the Model Layer (ModL), where fine granularity models are implemented as discrete units, the Composition Layer (CompL), where basic models are linked into more complex,

aggregated models, and the Configuration Layer (ConfL), which allows providing context specific parameterization (in the software sense) for operational use. Applications can span from simple console applications to user-interacting applications based on the model-view-controller pattern, in the simplest cases linking either directly to either the ModL or the CompL, or accessing model ConfL. In all cases, the component oriented architecture allows implementing a set of functionalities which impact on the richness of functionality of the system and on its transparency. Layers implement no dependency among them, hence facilitating the independent reuse of tools, utilities, and model components in different applications and frameworks.

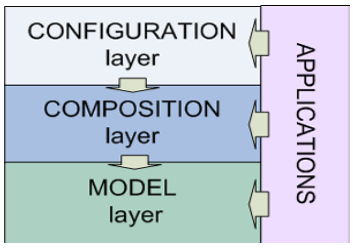

Figure 1. Layer-based software simulation system: **Model layer:** fine grained/composite models implemented in components; **Composition layer:** modeling solutions from model components; **Configuration layer:** adapters for advanced functionalities in controllers; **Applications:** from console to advanced MVC implementations.

When looking to the component model diagram of a possible modeling solution, we can identify the three layers as in the example below: model components are realizations of the model layer, and have no dependency from the other layers; the package composing (linking them and providing services) is a realization of the composition layer, again with no dependency to the others; finally, the package making a modeling solution is a realization of the configuration layer which allows building articulated applications. The configuration allows building an adapter to use the modeling solution, whereas the composition could be used via different adapters in different configuration frameworks. Similarly, each single model component could be reused independently of the composition layer.

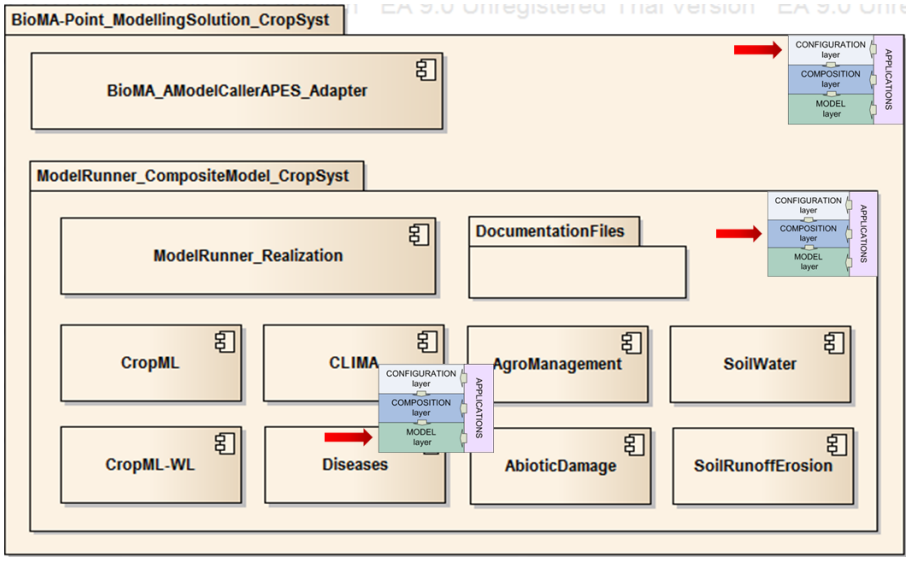

Figure 2. Example of modeling solution making use of the three layers, and reusable outside the framework as either model composition or single components.

Typically, a legacy unit of code would be implemented via an adapter to the configuration layer; however, such implementation may not make accessible all the functionalities identified as requirements by the model and composition layers.

2.1 The model layer

One possible definition of a model, relevant to the work of developing components for biophysical simulation, is a conceptualization of a process. When developing code using the OOP paradigm, a model can be implemented in a class, providing the estimation or generation of a variable (or a set of interrelated variables), obtaining a fine level of granularity. There might be more than one way to do this. If two different models estimate variable "A", those two models are alternatives for estimating variable A even if they have different input requirements and different parameters. As a consequence, the two models must be available as separate units, and their input, parameters and output must be defined. Such units are here called **strategies**, from the related design pattern introduced below. A way to have a set of models available in a component, via the same call, including alternate approaches, is the implementation of the design pattern Strategy (Mesketer, 2004). This offers the user of the component algorithms which are alternative options (strategies) for doing the same thing. When building a biophysical model component, this allows in principle alternative options to be offered for estimating a variable or, more generally, to model a process. This often-needed feature in the implementation of biophysical models, if implemented in this way, comes with two very welcome benefits from the software side: 1) it allows easier maintenance of the component, by facilitating the addition of other algorithms, 2) it allows the easy addition of further algorithms from the client side, without the need to recompile the component, while keeping the same interface and the same call. The basic point here is that a strategy (a model class) encapsulates a model, the ontology of its parameters, and the test of its pre- and post-conditions. It can be used either directly as a strategy (in this case we call it simple strategy, where simple indicates that does not use other strategies as part of its implementation), or it can be used as a unit of composition, as described below. A **composite strategy** differs from a simple strategy because it needs other (simple) strategies to provide its output(s). The list of inputs is given which includes all the inputs of all classes involved (except those which are matched internally). The list of outputs includes all outputs produced by each strategy and the ones specific to the composite class (if any). The list of parameters needed includes those of the classes associated with and the ones (if any) defined in the composite class. When the value of a parameter is set, if the parameter belongs to an associated class, it is set on that class. The test of pre/post conditions makes use of the methods available in each associated simple strategy class, plus the new tests specified in the composite class. If a violation of pre/post condition occurs in one of the associated classes, the message informs the user not only about the violation that has occurred, but also in what class it has occurred. Composite strategies do not differ in their use compared to simple strategies. The interface used for models is the same for all modelling solutions in the component, implementing the Composite pattern to hide the complexity of model solutions based on composite strategies. This leads to their being a single signature for internal and extended models. Composite strategies too can be added to the components without requiring a re-compilation of their code, thus providing a way to extend component models fully autonomously by third parties. The adoption of the Composite pattern also allows implementing the Create-Set-Call [Cwalina and Abrahams 2006] pattern to maximize the application programming interface simplicity; once a developer gets familiar with the interface of models of a specific domain, their use is immediate, including extensions made autonomously by third parties. If a component is made available consisting of strategies (and unit tests as requirement), a third party will not change the code of the components, instead, it will provide either alternate solutions to the existing modeling problems, or solutions to other modeling problems in the same domain. In other terms, what is normally achieved via code

sharing (co-development and debug) is proposed as two separate steps, in which debug is done and certified by code owners, and extension may be provided by third parties. As a side effect, this allows a clearer mechanism for preserving intellectual property rights that what is implicit in open-source policies; however, further discussing these aspects is out of the scope of this paper. Composite strategies are solutions to modelling problems at a coarser granularity (in principle) with respect to simple strategies. In other terms, a composite strategy is a "closed" modelling solution which makes use of selected models of finer granularity as units of composition (simple strategies). Such a closed solution is not proposed as the unique solution for a specific modelling problem. An example of composite strategy is presented in Villa et al. [2006]. This kind of composite models provide in fact a sound foundation to select modeling approaches to be used at operational level. A third type of strategy is the **context strategy**. In this type of modeling unit, logic is implemented to select the strategy to be used at run time. Alternate model approaches might be chosen, for instance, in response to values of inputs, or based on the presence/absence of inputs. Details on the requirements of the model layer are provided by Donatelli and Rizzoli, [2008]. The papers by Bregaglio et al. [2012] and Confalonieri et al. [2012b] show examples of how the architecture of the modeling layer fosters model development and reuse. The papers of Carlini et al. [2006], Donatelli et al. [2006, 2006b], and Bregaglio [2012], present examples of model components. A library of model components freely downloadable is available with dedicated software development kit for reuse [Components 2012]

2.2 The composition layer

The composition layer is where models from different components are composed to build a modelling solution. A concrete realization of the composition layer is a modeling solution, which can however be made of one component only in this case just for using the composition layer services. A model solution is developed and used for a specific purpose (e.g. a "crop model" in which we link crop, soil water and other sets of models to simulate water limited production of crops).
The layers are connected via implementation of the design pattern Adapter, and, in fact, a realization of the configuration layer may be done connecting directly an object from the model layer (hence not linking components within the layer at all).

The composition layer must include:

- Time handling, hence allowing for calls to models at the time step chosen for communication across components in the modeling solution (the time step chosen for communication is not necessarily the time step of the modeling approaches used);
- Provide events handling (even if we can think of a framework like the one of the composition layer as always event driven; in this case we refer to actions which are triggered not at all time steps).

The composition layer may include:

- Integration services;
- Data services (in principle excluding persistence, which is part of the configuration, hence belonging to the configuration layer and context specific).
- Visual tools can be developed to assist creating code units to be compiled and used by applications.

Functional requirements:

- Must allow re-use of components data-types;
- Allow transfer of modeling options/run options to/from the higher level (Configuration level, Application);
- Require simple implementation of adapters of components to an instance of the layer;
- Allow multiple exchange of data across components within time step;

- Implement an initialization and finalization method;
- Have its own scalable logging.
- Allow discovering via reflection links between components, and on the quantities involved
- Allow discovering via reflection the components used
- Allow discovering via reflection inputs, outputs, and parameters of modeling solutions
- Allow discovering via reflection modeling options made available as part of specific modeling solutions

2.3 The configuration layer

Once a modeling solution has been developed, there is the need of providing it with all the data necessary for its run, e.g. a weather series or soil data in the case of a crop growth simulation model. These data can originate from various deployment environments, for example a database, xml files, or remote web services.
All these ways of providing a modeling solution with needed data are abstracted in the concept of a configuration for a run of the modeling solution itself. This concept is addressed in the Configuration Layer. This layer must expose functionalities to code using the object needing to be configured, because this code must be able to correctly configure it before the run. Also, the configuration layer must expose handles to run a modeling solution iteratively, as it is requested for instance in sensitivity analysis or during optimization.

Functional requirements

- Fill in values for items constituting the configuration.
- Verify their validity with respect to the environment of execution.
- Save a configuration for later reloading.
- Create recursive configuration structures, in case one of the items constituting the configuration needs in turn to be configured (e.g.: once chosen a database reader as the provider for a data series, fill in credentials to connect to it). An implication of this requirement is that not only a modeling solution must own a configuration, but also the non-trivial components constituting the configuration itself.
- Support callback functions when the status of a configuration changes, to refresh views attached in a Model View Controller architecture.

Non functional requirements

- A modeler should have to write as less code as possible to implement these functionalities, allowing him to concentrate on the business logic of a Modeling Solution. Hence, for the functional requirements, as much implementation as possible should be provided in advance.
- To ease the realization of Modeling Solutions adapting third party models, the implementation should not be provided in advance by means of an abstract class, i.e., the common super type of each Modeling Solution must be an interface.

3. EXAMPLE REALIZATIONS

Several model components have been developed, some presented or used in papers in these proceedings. Compositions were also made and adapters to the BioMA – Biophysical Model Applications configuration layer realization developed [BioMA 2012]. Applications were developed leveraging of the layers described, and depending on the Configuration layer. The modelling solutions shown in Fig. 3 are visible to the application via adapters of the configuration layer. The application developed is currently being used to run multiple modelling solutions in the domain

of biophysical models in agriculture for the European Commission. The following papers contain examples of heterogeneous modelling solutions being used in the biophysical domain and all available in the proceedings of the conference: Manici et al. [2012], Confalonieri et al. [2012], Donatelli et al. [2012], Maiorano et al. [2012].

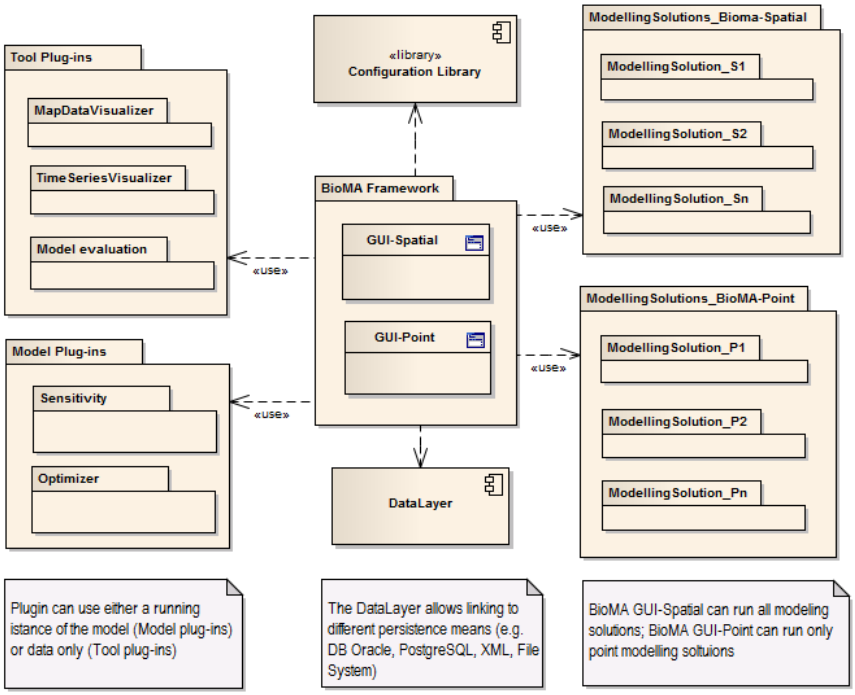

Figure 4. The BioMA application for biophysical simulation against explicit spatial units. Single components and modeling solutions can be reused outside the platform.

4. CONCLUSIONS

Adopting a the component oriented design, implementing models at fine granularity to foster reuse and hybridization in modeling solutions, and avoiding dependencies to modeling frameworks concretely fosters models re-use. The platform and applications developed within the BioMA project are made available as concrete examples for solution to operative problems, but, and of no lesser importance, as single components made available which allow independent extensibility by third parties. We claim that a greater focus on reuse, instead of specific framework, can effectively allow achieving the goal of avoiding duplication of modeling engines, sharing components of known quality.
The realizations of the architecture have been used for analysis delivered to the European Commission covering Europe and Latin America with different levels of abstraction. The overhead for the implementation of the architecture presented has proved to be a modest toll with respect to the increased operational capabilities achieved, as in many modeling frameworks. However, the requirements of the model layer and of the composition layer make the discrete model units produced reusable outside the framework, and provide advanced functionalities for their use and their composition.

REFERENCES

BioMA 2012 Biophysical Models Applications
http://bioma.jrc.ec.europa.eu/bioma/help/ [verified May 15, 2012]
Bregaglio, S., 2012. PhD Thesis
ftp://mars.jrc.ec.europa.eu/public/marcello/Bregaglio/ [verified May 15, 2012]

Bregaglio S., Donatelli M., Confalonieri R., Acutis M. 2012. Comparing modelling solutions at submodel level: a case on soil temperature simulation. *These proceedings.*

Carlini L., Bellocchi G., Donatelli M. 2006. Rain, a software component to generate synthetic precipitation data. Agronomy Journal 98, 1312-1317

Components http://agsys.cra-cin.it/tools/ [verified May 15, 2012]

Confalonieri R., Bregaglio S., Stella T., Negrini G., Acutis M., Donatelli M., 2012. An extensible, multi-model software library for simulating crop growth and development. *These proceedings.*

Confalonieri R., Bregaglio S., Donatelli M., Tubiello F., Fernandes E. 2012. Agroecological Zones Simulator (AZS): A component based, open-access, transparent platform for climate change – crop productivity impact assessment in Latin America. *These proceedings.*

Cwalina K., B. Abrams, 2006. Aggregate components. In Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries. Addison-Wesley, Courier in Westford, Massachusetts, USA. 235-271.

David, O., Ascough, J., Leavesley, G., and Ahuja, L.: 2011 Rethinking modeling framework design: object modeling system 3.0, in: En- vironmental Modeling International Conference Proceedings, 5- 8

Formetta, G., Mantilla, R., Franceschi, S., Antonello A., and R. Rigon, The JGrass-NewAge system for forecasting and managing the hydrological budgets at the basin scale: models of flow generation and propagation/routing, Geoscientific Model Development Volume: 4 Issue: 4 Pages: 943-955

Donatelli M., Bellocchi G., Carlini L. 2006. Sharing knowledge via software components: models on reference evapotranspiration. European. Journal of Agronomy 24, 186-192.

Donatelli M., Carlini L., Bellocchi G. 2006b. A software component for estimating solar radiation. Environmental Modelling and Software 21, 411-416.

Donatelli M., Rizzoli A. 2008 A design for framework-independent model components of biophysical systems International Congress on Environmental Modelling and Software iEMSs 2008 Proceedings of the iEMSs Fourth Biennial Meeting, Barcelona, Spain, July 2008: 727-734

Donatelli M., Srivastava A., Duveiller G., Niemeyer s. Estimating Impact Assessment and Adaptation Strategies under Climate Change Scenarios for Crops at EU27 Scale. *These proceedings.*

Gregersen, J.B., Gijsbers P. J. A., Westen S. J. P., Blind M., OpenMI: the essential concepts and their implications for legacy software, Advances in Geosciences, 4, 37-44, 2005

Gregersen J. B., Gijsbers P. J. A., Westen S. J. P. 2007. OpenMI: Open modelling interface. Journal of Hydroinformatics Vol 9 No 3 pp 175–191

Hillyer C., Bolte J., van Evert F., Lamaker A., 2003. The MODCOM modular simulation system. European Journal of Agronomy, 18, 3-4, 333-343.

Maiorano A., Donatelli M., Fumagalli D. Potential distribution and phenological development of the Mediterranean Corn Borer (Sesamia nonagrioides) under warming climate in Europe. *These proceedings.*

Manici L., Donatelli M., Fumagalli D., Lazzari A., Bregaglio S. 2012 Potential Response of Soil-Borne Fungal Pathogens Affecting Crops to a Scenario of Climate Change in Europe. *These proceedings.*

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: MODSIM 2004 International Congress on Modelling and Simulation, Modelling and Simulation Society of Australia and New Zealand Inc,Townsville, Australia, pp. 1727-1732Szypersky C., D. Gruntz, S. Murer, 2002 Component software - beyond object-oriented programming. 2nd Ed. Addison-Wesley, London, United Kingdom, 2002.

Villa, F., M. Donatelli, A. Rizzoli, P. Krause, S. Kralisch, F. K. van Evert 2006. Declarative modelling for architecture independence and data/model integration: a case study iEMSs Third Biennial Meeting, Vermont, July 2006.