Jul 1st, 12:00 AM

Enriching software model interfaces using ontology-based tools

Ioannis N. Athanasiadis

Andrea-Emilio Rizzoli

Marcello Donatelli

Laura Carlini

Enriching software model interfaces using ontology-based tools

I. N. Athanasiadis [a], A. E. Rizzoli [a], M. Donatelli [b], and L. Carlini [b]

[a] *Dalle Molle Institute for Artificial Intelligence, Lugano, Switzerland*

[b] *CRA –Research Institute for Industrial Crops, Bologna, Italy*

Abstract: Common practice has proven that software implementations of environmental models are seldom reused by broader communities or in different modelling frameworks. One of the reasons for this situation is the poor semantics of model interfaces. Model interfaces describe a critical amount of the modellers' knowledge, but their software implementations fail to represent the complexity of model assumptions in software terms. In this paper, we present an ontology-driven approach that aims to enrich software model interfaces with advanced semantics. A generic ontology for defining environmental model variables has been developed along with two families of tools for supporting the modellers' community to share their knowledge and software codes in an easy, efficient and sound way. The first family of tools consists of a web-based ontology editor for sharing knowledge related to environmental model components and their interface variables. The second set of tools exploits the knowledge stored in the ontology by generating source code in an automated fashion. Thus, it is shown how ontologies, accompanied by a set of supporting tools, can be used for promoting the reuse of environmental models.

Keywords: Declarative and semantic modelling, Ontologies, Model linking and integration, Code generation

1. INTRODUCTION

Writing a model of an environmental system is a complex process which aims at providing an abstraction of the real world processes, using a given formalism, and exploiting a wide collection of techniques originating from general systems theory, to economics and social sciences.

A model, being an abstraction, in order to tame the complexity of the real world, approaches its subject from a specific point of view; particular assumptions and hypotheses about the phenomena involved are made. We therefore neglect the full extent of causal chains and driving forces of the phenomena of interest and we strive for simplification, focalization and modularisation of the model construction process.

When we implement the model on a computer, we introduce more assumptions, more limitations (for instance, the model is forced to a discretization) and therefore the software implementation of a model should be considered as a poor realization of the original formalisation. Such an approximation states only implicitly the assumptions made for building it. For instance, the spatial discretization of a model variable can only be inferred by a close inspection of the data type used to implement it.

During the last decades, a number of models have been designed and implemented, and it has become natural to assemble them together in order to try to address more and more complex problems. Integrated assessments are becoming increasingly common in environmental management and therefore we are faced with the problem of integrating models across scales and disciplines. This is neither an easy, nor a straightforward process.

Software Engineering promotes the concepts of reusing "components-off-the-shelf" (Szyperski et al 2002, Egyed et al. 2005), distributed computing (Attiya and Welch, 2004), agent-based computing (Luck et al. 2005), service-oriented architectures and web services (Erl, 2004) to support the development of modular applications. The very same concepts are meant to be used to develop modular and integrated environmental software applications.

However, software integration is not the sole necessary condition for a proper assemblage of environmental models. In other words, if a set of (good) software model implementations are working together, this is not at all a sign that the compound model makes any sense from a modelling point of view and generates credible results. Dif-

ferent authors have tried to target the issue of quality assurance in the development of environmental models (Refsgaard et. al. in press, Jakeman et al. in press), but their main focus is on the quality of the modelling process.

This paper argues that sound integration of environmental models also requires automated coupling of the knowledge hidden behind each software implementation. In particular, in Section 2 we investigate a model structure and identify its knowledge components, typically implicit both in the model interface and implementation. Section 3 focuses in the utilization of ontologies for specifying model interfaces, while in Section 4 we present a web-based tool for communal ontology authoring.

2. MODEL KNOWLEDGE AND LINKING

2.1 The knowledge encapsulated in a model

The result of the modelling process is a formalisation that encapsulates knowledge related to both the interactions of the modelled system with its surrounding environment (model interface and data exchange), and the internal behaviour of the system (model equations, or endogenous variables). Consequently, a software component implementing a model will consist of two parts, the interface and the implementation. The interface defines the inputs, outputs and parameters of a model, while the implementation defines the model equations.

Declarative modelling aims to separate the algorithms which execute the numerical solution of the model equations from the 'declarations' of the equations themselves and the variables and parameters occurring in the equations. Prior work has focused on the equation part (Muetzelfeldt, 2004), whereas in this paper we concentrate on a declarative approach for describing model interface to facilitate model linking and integration using ontologies. Ontologies provide a formal support to express conceptualisations (Gruber, 1993), and a number of tools support the creation of ontologies. Furthermore, model knowledge stored in the ontology can be used both for formal documentation and provide functionalities which go beyond the computation of model variables.

2.2 Sound model linking and integration

Easy model linking and integration is a key feature that is advertised by most modelling frameworks. However, we advocate that simple integration in software terms is not enough for sound model integration. A software implementation of an environmental model does not take into account the semantics of the software interface. The information associated with the inputs, states, outputs and parameters is limited to their data type. For instance,

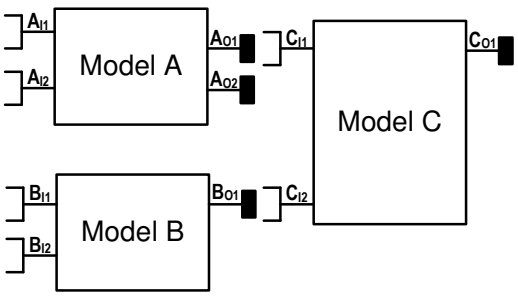

Figure 1. A model linking example.

a typical software implementation expounds as model interface arrays of doubles, integers, and strings, whose context is described in the software documentation, or, even worse, only in the variable names. However, this practice requires that someone has to read the documentation in order to understand how to reuse this model properly. This is because the model's knowledge related to its interface is not encapsulated in the actual interface of the model implementation in a self-explained fashion.

Consider for example the case depicted in Figure 1, where Models A and B are linked to another Model C. Model C exposes to inputs C_{I1} and C_{I2}, which are to be linked to model outputs A_{O1} and B_{O1}. Let assume, without loss of generality, that all these variables are simple floats. In software terms, integration can be achieved simply if both C_{I1} and C_{I2} are linked to any software component output that provides a float. However, from a modelling point of view, each model input or output is not simply a float, instead it measures a specific quantity in a specific temporal and spatial context (i.e. it could be a car's velocity or an ambient air pollutant's concentration at ground level, and so on). Moreover, even if two models correctly link a variable expressing the same element, the model receiving the variable as an input may be able to handle only a sub-range of the values provided as outputs (due to model assumptions). It becomes evident that standard software interface conventions are not enough for encapsulating the full knowledge of the model interface.

The vision of reusing model software implementations as off-the-shelf components requires the assumptions on the model interface to be represented implementation in a machine readable format. Following the previous example, suppose Models A, B, and C are supplied by diverse vendors. In order to achieve sound model integration, each linkage should be verified not only at the low level of data type matching (which is the unique requirement for software integration), but also against the actual semantics (context and assumptions) related to model interface. To elaborate it a bit more, let Model A (of the previous example) exposes a sin-

gle float A_{O1} that represents the calculated rainfall output, while Model C has a water pressure input C_{I1}, to be also a float. Suppose that someone tries to make a link from A_{O1} to C_{I1}. In such case, as both variables are represented as single floats, the integration is feasible in software terms, though it makes no sense form a modelling perspective. The same holds for less semantically diverse cases, where we could have model variables expressing the same concept, but with mismatches in characteristic times, units, pre- and post- conditions, temporal or spatial dimensions and sampling rates.

We need to express all the knowledge related to the model interface in a declarative way, using an ontology, as we show in the following section.

3. TOWARDS AN ONTOLOGY FOR SPECIFYING MODEL INTERFACES

3.1 Models and model types

In order to enrich model interfaces with advanced semantics, we developed an ontology, called the *Model Interface Ontology* that aims to encapsulate our knowledge on the model interface in a declarative fashion. In this paper, we consider biophysical agricultural models. As agricultural biophysical processes occur through time and space, they are usually modelled using stocks and flows, following the system dynamics approach. A model interface exposes both *stocks* (states) and *flows* (rates of inputs and outputs) and it can be used by a simulation engine (numerical integrator) for calculating the stocks as an accumulation of flows over the simulation time horizon.

These concepts are declared in our ontology as follows: We identify two types of models: *Static* and *Dynamic* models. The first kind of models does not expose any states and rates, as they are not required to be integrated over time. The opposite holds for the dynamic models. All inputs, outputs, states and rates of models are types of an abstract *Measurement* concept (ontology class), which is used for defining their semantics in different contexts (space, time units, and so on). The Measurement class is detailed below. Figure 2, illustrates the relations between the two model types in the ontology.

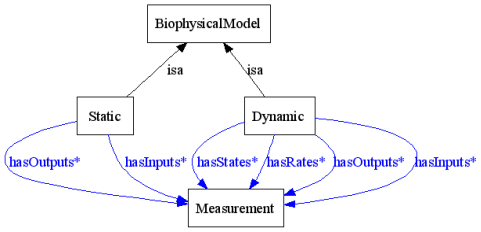

Figure 2. The relations between the model type concepts of the model interface ontology.

3.2 Model interface elements as Measurements

The Measurement class is the key instrument for conceptualising the model interface elements. The Measurement class specifies the following properties of a model interface element:

- The observed quantity
- The spatial observation context
- The temporal observation context
- The sampling frequency
- Value conditions (minimum, maximum and default value and default unit)

A *Quantity* can be considered as the result of applying a physical dimension on a subject of interest. For example, *AirTemperature* can be considered as a physical quantity that represents the *Temperature* dimension of air. *Spatial* and *Temporal contexts* are used to define the dimensionality of a measurement in space and time. *Sampling frequency* associates the tempo-spatial dimension of a measurement to a sampling rate and grid size. Finally, *value conditions* are used for defining boundary conditions for a measurement's allowed values. An abstract view on the Measurement class and its relationships with the rest concepts in the ontology is presented in Fig. 3.

Utilizing such a conceptual schema, we can detail a model interface element. For example, a measurement called "*HourlyAirTemperature*" can be defined by referring to *AirTemperature* quantity, be measured at a *point* in *space* and *time*, on an *hourly* basis, having as default unit degrees *Celsius* and be consistent to some value conditions (min, max, and

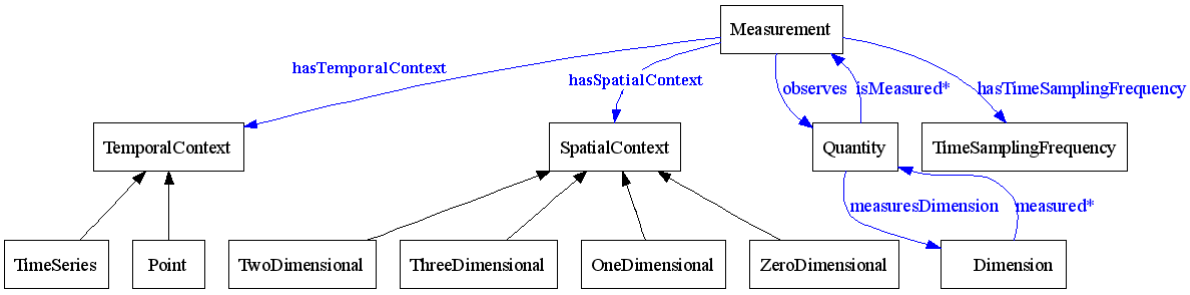

Figure 3.The relations of the Measurement concept.

default values). Consequently, such an instance of *Measurement class* can be attached to a model interface.

Note that the developed *Model Interface Ontology* has been realized using the Web Ontology Language (OWL, McGuinness, D.L and F. van Harmelen 2004), through the Protégé ontology editor (http://protege.stanford.edu/plugins/owl/). OWL–DL expressivity was enough for conceptualizing this domain. The specifications of units and dimensions were based on the SWEET ontologies (2006). Finally, the *Model Interface Ontology* is available online (at: http://seamless.idsia.ch/ontologies/mio.owl).

In the previous sections, we advocated the potential of publishing model interfaces in a declarative format and proposed an ontology for capturing the semantics of model interface elements. This approach was undertaken by the Seamless-IP project and the community of Agricultural Production Externalities Simulator (APES) modellers. A set of tools have been developed to enable modellers to: (a) share their knowledge related to environmental model components and their interface variables, and (b) exploit the knowledge stored in the ontology by generating source code in an automated fashion.

4. AgrOntologies: A WEB-BASED TOOL FOR COMMUNAL ONTOLOGY AUTHORING

The process of setting up an ontology, and populating it with modellers' knowledge was not straightforward. The major problems experienced, were related to managing modeller's conflicting views and the complexity of the domain at hand. In order to tackle such issues and to facilitate knowledge elicitation within a community of more than ten modelling teams involved in APES, we built a web-based tool, called AgrOntologies, for communal ontology authoring. A key issue of this process is that modellers are required to make their model interfaces explicit and communicate them in a formal, yet comprehendible way to others. Through the AgrOntologies portal, a modeller can (a) specify model variables in detail, or even reuse existing variables defined by others, (b) define model interfaces and ultimately, (c) put together models together in components.

Note that the AgrOntologies portal presents information to the users in a "natural" way for them, not as they are represented within the ontology using description logics. In this sense, modellers are not required to be exposed to all the complexity of the internal ontology structure; rather they are allowed to register their models through an easy to use portal.

We are currently evaluating the ontology design and populating the ontology with actual model specifications. A screenshot of the developed portal is shown in Figure 4.

5. DCC: A TOOL FOR GENERATING MODEL SOURCE CODE

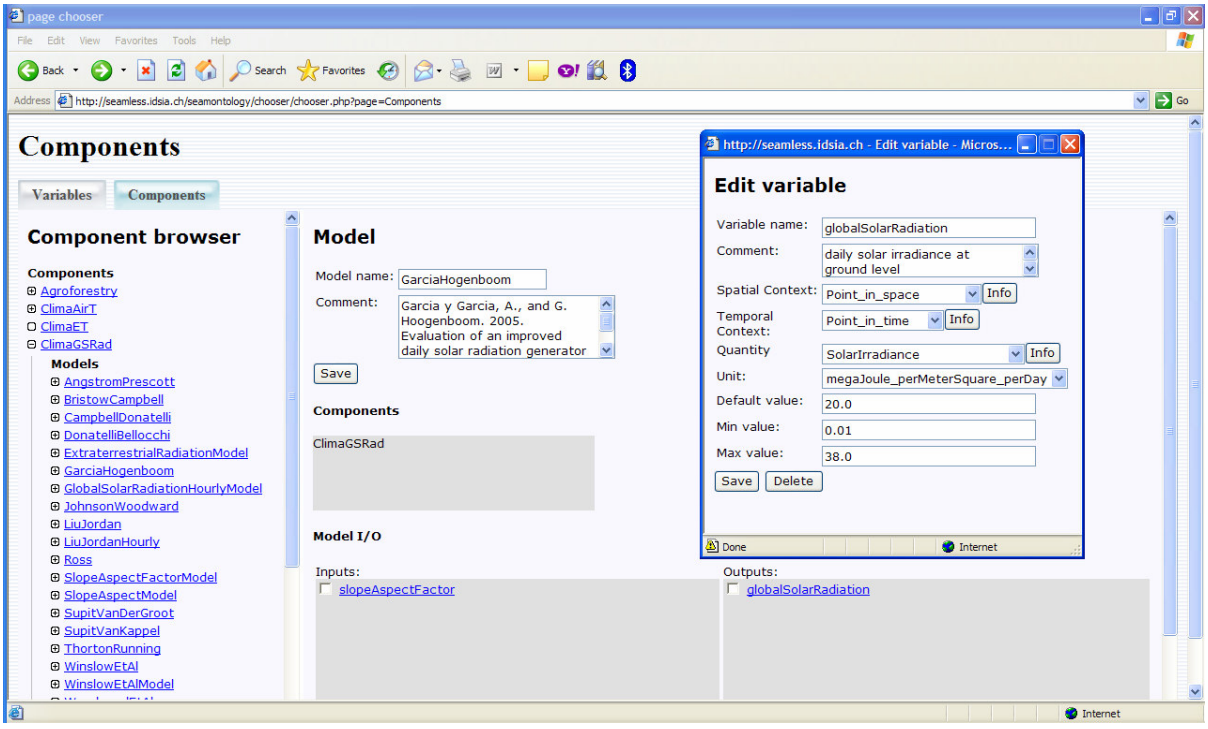

Figure 4.The relations of the Measurement class.

The use of the definition of concepts and their instances goes beyond documentation and model component linking. The attributes values associated with each variable can in fact be used to provide to components information needed to test the adequacy of values at run time. This can be done via the implementation of the design-by-contract approach to test pre-conditions (e.g. Donatelli et al, 2006a and 2006b). Making available variables attributes in an implementation of model components has multiple uses, because it allows: 1) validating inputs to the component, 2) using bounds for model parameters in automatic calibration, 3) defining sub-ranges of allowed variables to account for specific model limitation, and 4) provide attribute values as simulation output for auto-documentation of results.

A software design which allows implementing the information available in components makes use of an abstract data type called the domain class, following the approach by Rizzoli et al. (1998). The domain class is characterised a set of data attributes, which are the inputs, states, outputs of the model and a set of access methods to set and get the attribute values. The data attributes contain the numerical value, the variable's range, the default value, and the measurement units. Defining a domain class also allows setting the boundaries of the domain to be modelled, providing the information to model according to the approach chose. Multiple models implemented in a component can make use of the same domain class.

The application Domain Class Coder (DCC) is a windows application which, from an input files extracted from the ontology application described in Section 4, generates the C# code of twin classes. Such classes are a type to hold values, and a companion class to hold variables attributes. The former is an abstract class to be used as type in the component interface, which then allows extensions via subclassing of its default implementation. The other class, conventionally called with the postfix VarInfo to the value class name, contains attribute values which are declared as static properties and have only the get access method. VarInfo values are used by a component to test pre and post conditions which uses the VarInfo type, (CRA.core.preconditions.dll, available as the DCC, at http://www.isci.it/tools; DCC is available the page XP Utils). The XML schema of the latter type is shown in Fig. 5. From the XML schema it becomes evident that the information realized in the domain class is less compared to that stored in the ontology, but it is functional to the purpose describe above.

Once the input file is loaded (either as an XML or as a tab separated ASCII file), the user can change minimum and maximum values to account for specific model limitations (if any) with respect to the values stored in the ontology. The user must also specify the domain class name, and the namespace of the class. The output is given by the C# code of the two classes described above, which implement interfaces which allows discovering types and attributes via reflection. The package which can be downloaded also contains a sample input file which allows generating the relevant classes.

When these classes are included in a component assembly, its content can be browsed via reflection using the application Model Component Explorer. This component allows discovering the domain classes, their attributes and types, and the VarInfo values for each attribute. The component is available in the same page of the DCC.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
  - <xs:element name="VarInfoDomainClass">
    - <xs:annotation>
        <xs:documentation>VarInfo domain
          class</xs:documentation>
      </xs:annotation>
    - <xs:complexType>
      - <xs:sequence maxOccurs="unbounded">
        - <xs:element name="VarInfoType">
          - <xs:complexType>
            - <xs:sequence>
                <xs:element name="name" />
                <xs:element name="description" />
                <xs:element name="currentValue" />
                <xs:element name="maxValue" />
                <xs:element name="minValue" />
                <xs:element name="defaultValue" />
                <xs:element name="units" />
                <xs:element name="varType" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

Figure 5. The XML Schema of the VarInfo Domain Class.

6. DISCUSSION AND FUTURE WORK

Various Environmental Management Information Systems have exploited ontologies mainly for information processing. Most of them focus on seamless integration of environmental data repositories, e.g. related to coastal zone management (Cristophides et al. 1999), weather (Dance and Gorman 2002), or water management (Felluga et al. 2003). More generic approaches for environmental data fusion as Infosleuth (Nodine 2000), Buster (Neumann et al. 2001) and AMEIM (Athanasiadis et al 2005) utilized ontologies too. However, none of those systems use ontologies for environmental model linking and model component integration. This is the major contribution of this paper, where we introduce ontologies as a medium for efficient model integration. The Model Interface Ontology was proposed for enriching model interfaces in a declarative fashion. Also, clear path for building

reusable components was defined, and the use of ontologies, accompanied by a set of supporting tools, was exemplified.

Parallel efforts (Villa *et al*. 2006) are focusing on extending the current framework by specifying model equations using semantic modelling primitives. Ontology representations of both model interfaces and equations may lead us to a fully declarative modelling and simulation environment.

7. ACKNOWLEDGEMENTS

This publication has been partially funded under the SEAMLESS integrated project, EU 6th Framework Programme for Research, Technological Development and Demonstration, Priority 1.1.6.3. Global Change and Ecosystems (European Commission, DG Research, contract no. 010036-2). Seamless project website: http://www.seamless-ip.org. The AgrOntologies portal tool was implemented by David Huber, AntOptima SA.

8. REFERENCES

Athanasiadis, I., Solsbach, A., Marx Gómez, J., Mitkas, P., 2005. An agent-based middleware for environmental information management, *In* 2nd Conf. on Information Technologies in Environmental Engineering (ITEE'2005), pp. 253-267.

Attiya, H. and Welch, J., 2004. Distributed Computing: Fundamentals, Simulations, and Advanced Topics, Second Edition, Willey.

Christophides, V., Houstis, C., Lalis, S., and Tsalapata, H. 1999. Ontology-driven Integration of Scientific Repositories. In: Proc. of the Fourth Workshop on Next Generation Information Technologies.

Dance, S, and Gorman, M., 2002. Intelligent Agents in the Australian Bureau of Meteorology. In: Proc. of the 1st International Workshop on Challenges in Open Agent Systems to be held at AAMAS'02.

Donatelli M., G. Bellocchi, L. Carlini. 2006a Sharing knowledge via software components: models on reference evapotranspiration European Journal of Agronomy. Vol 24, No.2, pp.186-192

Donatelli, M., G. Bellocchi, L. Carlini, 2006b. A software component for estimating solar radiation. Environmental Modelling and Software. Vol. 21, No. 3, pp. 411-416.

Egyed, A., Müller, H.A., and Perry D. E., 2005. Integrating COTS into the development process, IEEE Software, 22 (4): 16-18.

Erl, T., 2004. Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services, Prentice-Hall.

Felluga, B. *et al*. 2003. Environmental Data Exchange for Inland Waters Using Independed Software Agents. Report 20549 EN. Institute for Environment and Sustainability, European Joint Research Centre, Ispra, Italy.

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

Jakeman, A.J., Letcher, R.A., Norton, J.P. 2006. Ten Iterative Steps In Development and Evaluation of Environmental Models. Environmental Modelling & Software. In press.

Luck, M., McBurney, P. Shehory, O., Willmot, S. (eds.), 2005. Agent Technology: Computing as interaction, AgentLink.

McGuinness, D.L and F. van Harmelen (eds), 2004. OWL Web Ontology Language Overview, W3C Recommendation, www.w3.org/TR/owl-features/

Muetzelfeldt, R.I. 2004. Declarative Modelling in Ecological and Environmental Research. European Commission Directorate-General for Research, Position Paper no. EUR 20918. European Commission, Brussels, Belgium.

Neumann, H., Schuster, G., Stuckenschmidt, H., Visser, U., and Vögele, T. 2001. Intelligent brokering of environmental information with the buster system. In Hilty, L.M., and Gilgen, P. W. (eds), Intl. Symposium Informatics for Environmental Protection., pp. 505-512.

Nodine, M, Fowler, J., Ksiezyk, T., Perry, B., Taylor, M., Unruh, A.. 2000. Active Information Gathering in InfoSleuth. International Journalof Cooperative Information Systems, 9 (1-2):3-28.

Rizzoli, A.E., Davis, J.R., Abel, D.J. 1998. A model management system for model integration and re-use. Decision Support Systems,Vol. 4, No. 2, pp. 127-144.

Refsgaard, J.C., J.P. van der Sluijs, J. Brown, P. van der Keur. 2006. A framework for dealing with uncertainty due to model structure error. Advances in Water Resources. In press.

SWEET Ontologies, 2006. Semantic Web for Earth and Environmental Terminology (SWEET), url: http://sweet.jpl.nasa.gov, Last updated on: Jan 26, 2006.

Szyperski, C., Gruntz, D., Murer, S. 2002. Component Software – Beyond Object-Oriented Programming, Second Edition. ACM Press, New York, NY.

Villa, F., M. Donatelli, A. Rizzoli, P. Krause, F.K. van Ewert, 2006. Declarative modelling for architecture independence and data/model integration: A case study, In 3rd Biennial meeting of the International Environmental Modelling and Software Society, July 9-12, 2006.