Declarative modelling for architecture independence and data/model integration: a case study

Ferdinando Villa, University of Vermont. USA (ferdinando.villa@uvm.edu)
Marcello Donatelli, ISCI, Bologna, Italy
Andrea Rizzoli, IDSIA, Lugano, Switzerland
Peter Krause & Sven Kralisch, University of Jena, Jena, Germany
Frits K. van Evert, PRI, Wageningen, The Netherlands

Abstract: The need for integrating dynamic models with independently developed datasets and other models has long been recognized. Only recently, advances in modelling technologies, knowledge representation and protocols for remote communication of structured content have made this goal practical. The same advances make it possible to decouple the representation of a model from its executable implementation in ways that allow unprecedented levels of architecture independence and explicit, transportable model declaration. These developments are crucial to the creation of repositories of models where the models' lifetime is not tied to that of specific modelling paradigms, execution architectures, or storage technology. In this contribution, we describe a case study involving the declarative representation of model structure in a web-based knowledge base, its extraction through standard URLs as XML content, and the automated translation of the XML via dedicated components into source code models to be compiled for three different target architectures. We present the minimal declarative specification of the model interface and its key components, and discuss how it can be translated into executable form. We also describe the advantages of linking of each model component to explicit, external ontologies for matching model inputs and outputs to datasets and to other models. The advantages of the declarative specification are discussed in the light of ongoing, large-scale projects in agricultural and biodiversity modelling.

Keywords: Declarative modelling, ontologies, code generation, platform independent models

1. INTRODUCTION

Mathematical models provide a formal way to express our knowledge on how observations can be processed to infer new data. Models, like data, always conform to a conceptualization. According to Genesereth and Nilsson (1987) a conceptualization is the objects, concepts, and other entities that are presumed to exist in some area of interest and the relationships that hold them. Thus, a conceptualization capable of describing scientific models requires, at minimum, to express the notion of linkage between scientific observations and their change in time and space by causal relationships.

The set of abstractions that allows conceptualizing and expressing those cause-effect relationships and their results is the adopted *modelling paradigm*, exemplified by notions like ordinary differential equations, stock-and-flow, or individual-based modelling.

Very often we speak of "models" without any distinction between the model equations, the implementation of the equations in a software code. Sometimes, even complex software applications, which include a graphical user interface a database management system, are called "models". This confusion can be solved by the wording *declarative modelling*, which has been used to identify a specification of models that is based on the semantics of the natural systems being modelled rather than the algorithms that calculate their changing states. As such, declaratively expressed models are independent on architectural and software details, and need to be implemented into working algorithms before they can be simulated. A domain specific case of declarative modeling is for instance given by the platform independent model at the base of UML, which needs a platform specific implementation to become a software. The implementation of a declarative model is then delegated to an algorithm, rather than a human programmer, and this brings in a number of advantages.

First, the modeller can focus on the modelling details, rather than the implementation ones: the modeller is forced into adopting tight, well-reasoned, and usually better description paradigms compared to, e.g., the definition of a model implicit in a FORTRAN program. The main advantage, though, lies in the fact that declarative models only depend on the conceptualization (paradigm), and are thus easier to be exchanged and communicated as long as the basic conceptualization is agreed upon. In this paper we focus on the role of declarative specifications to provide portability and

facilitate integration between independent, uncoordinated data and models.

2. DECLARATIVE MODELLING

The main tool to support declarative specification of a model is a formal statement of the underlying conceptualization. Such a statement is performed by relating model entities (such as. variables, equations, parameters) and their meaning (e.g. an input variable is a temperature) to a set of primitive concepts, which are shared as the common understanding of the modelling domain. This is done implicitly in our brain, when we read a set of model equations, but it must be explicitly stated to a machine.

In recent years, the use of *ontologies* (Gruber, 1993) has become commonplace to express formal conceptualizations in mutually understandable ways. Ontologies are expressed using common representational languages such as XML and they are optimized for web-based access and sharing. A concept is expressed by its properties and the values of the properties, and concepts are put in relation by links, which themselves have properties that take values.

Declarative modelling can be supported by ontologies since they can provide, at the same time, schemata for model declaration and meaning for these schemata. Given an ontology describing model concepts, *instances* are defined to declaratively express specific models by referring to concepts laid out in the ontologies. As an example, an ontology may provide the definition of the *stock* and *flow* concepts, and a model is declared by defining specific stock and flow variables with their names, initial states, and equations. Such declaration contains enough information to enable a software execution environment to simulate the behaviour of the model over a temporal and spatial extent.

Moreover, thanks to the rich meaning made possible by current ontology frameworks, an execution environment can be programmed to properly connect models to data, and feed quantities calculated by simulation to other models in the same environment, without risking the error resulting from matching inputs and outputs that specify different natural-world entities.

2.1 The knowledge base

A database that contains both the ontology and the instances that populate it is usually called a *knowledge base*. Several storage options are available to handle a knowledge base through extensions to well-established database technology. The knowledge base can contain information on data, models, scientific workflows, and, in general, on all the concepts that are relevant to our modelling domain.

Storing a model in a knowledge base involves expressing its logical structure by subdividing the model into components and mapping each component to a concept defined in the same storage. As an example, a model may contain a parameter that needs to be generated at each run from a Weibull distribution with given parameters. In order to enable its declarative specification, the knowledge base must contain the definition of a Weibull variate with the specification of the parameters necessary to fully specify it. This definition will capture the conceptual essence of the distribution, and leave it to the software execution environment the task to create the call to an actual software routine that will produce the values from a pseudo-random generator that can simulate the distribution with the given parameters. Sophisticated ontologies may provide enough detail about the concept of a statistical distribution to move from the "black box" concept illustrated above to a more complete statement that can actually inform an experimenter of what a Weibull distribution, and ultimately a statistical distribution in general, is and how it is useful in scientific investigation. The more sophisticated the ontologies, the more flexibility can be achieved in integrating data and models from external sources, and the less detail must be provided to software execution environments in order to make them capable to simulate models expressed in declarative terms.

2.2. Logical vs. procedural views

The declarative representation of a model can be summarized into (1) reference concepts and properties to define the identity of each modelled entity, and (2) the properties that capture the causal relationships. The first point is the easiest to achieve, and familiar as embedded in common software packages such as STELLA (Costanza, 1998) or Simile (Muetzelfeldt and Masseheder, 2003), where model entities (such as stocks and flows) are laid out graphically. Yet, the second point is harder and essential to the concept of model, since the difference between data and models is that the latter establishes causal relationships among data.

Causality in conventional models is usually expressed through equations, defined to calculate the value of variables. Equations, by naming the values of other variables, implicitly define causal relationships that are viewed as dependencies from a processing point of view. An ontology-based framework can make these dependency relationships explicit, and add meaning to them by means of specializing the kind of relation. So, for example, a generic *depends-on* relationship can be further specialized into a *flows-into* relationship between a state variable and a flux variable (rate), and the underlying software architecture may react

to that by understanding that the flux must be integrated over time and added. The notion of variable, so central to conventional approaches, can similarly be enriched and made dependent on the modelled entity. For example, in an individual-based paradigm, variables describe quantitative traits of modelled individuals, but maintain the link to the individual, which is the main entity considered. No conflicts need to exist between paradigms, whose conceptual boundaries often become blurred when a explicit knowledge-based approach is used, particularly if notions of scale are formally defined.

2.3. From to knowledge base to code

In order to perform a translation from a declarative specification to a working software component, the software infrastructure must share and understand the conceptualization. The most sophisticated systems can read explicit ontologies and infer the simulation algorithms by reasoning on the contents of the ontologies. Other, simpler systems can be envisioned that are tied to one or a few specific ontologies (such as one that defines notions like the stocks and flows defined above) and produce algorithms that calculate them in a high- or low-level programming language, refusing to handle any knowledge that can't be interpreted in those terms.

An XML schema (Sperberg-McQueen and Thomson, 2000) can be seen as a relatively informal ontology, where the meaning is suggested, if not formally identified, by the names of the node identifiers, and basic relationships are captured by the structure of containment of nodes within other nodes. This is the case discussed in this paper, where the declarative structure is defined by means of a formally specified XML schema.

The case studies discussed below all use the same XML schema, which approximates an ontology of stocks and flows, and translate the specification of a model into three different high-level languages that can be compiled and executed to simulate the model. Figure 1 summarizes the process.

The knowledge base contains the ontology that specifies the *domain concepts* and the *model concepts*. The domain concepts represent the entities (the variables and parameters), which are used by the model concepts (the causal relationship, expressed as equations) to describe the model itself. `Temperature` is a concept, `airTemperature` and `greenHouseTemperature` are instance of such concept which are also described via specific attributes, and `greenHouseTemperature = airTemperature * k` is a model concept.

These concepts can be expressed using the XML Schema Definition Language (XSD), and the instances of the concepts are XML files which can be processed and translated into software components targeting various modelling framework platforms.

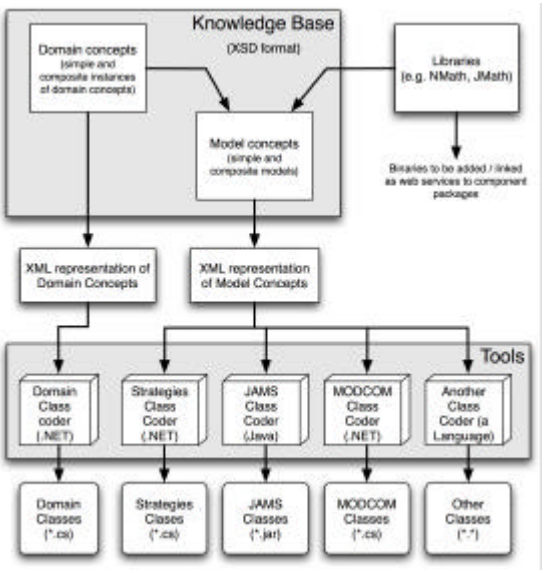

Figure 1. A schematic representation of the translation from declarative representation to binary implementations of models.

In the following paragraphs we illustrate a simple model and then provide details of its automated translation into system architectures available to investigators worldwide, each targeting a different modelling framework and programming language.

3. CASE STUDY

We present a model and show how it can be declaratively represented and translated to three different target architectures.

The case study is based on static models chosen to represent two hierarchical levels: several *simple* models, which depend solely on inputs, and a model which uses the simple models to build a *composite* model. Although the example is fairly simple, it represents the type of hierarchical structure that can be further expanded. We target a fine granularity of discrete model units, assuming that "users" may select the composite model, whereas "modellers" may plug in and out different simple models to study improvements in the estimation. Note that this substitution process has actually happened over the years with the composite model structures, which are described below.

3.1 The model

We use as example the models to estimate reference evapotranspiration according to the FAO approach (Allen et al., 1998). The composite model, i.e. the Penman-Monteith equation, is then split into a number of simpler models (Donatelli et al., 2006a). The reference evapotranspiration model is:

$$ET_0 = \frac{1}{l} \frac{s\left(R_n - G\right) + K_t \dfrac{VPD(e_a, e_s)\ r\ C_p}{r_a}}{s + g\left(1 + \frac{r_c}{r_a}\right)} \quad (1)$$

Where:

ET_0 = reference crop evapotranspiration (mm d^{-1});
l = latent heat of vaporization (MJ kg^{-1});
s = slope of the saturation vapour pressure-temperature relationship (kPa $°C^{-1}$);
R_n = net radiation (MJ m^{-2} d^{-1});
G = soil heat flux (MJ m^{-2} d^{-1});
K_t = unit conversion factor (86400 s d^{-1} for ET_0 in mm d^{-1});
VPD = vapour pressure deficit of the air (kPa);
e_a = actual vapour pressure (kPa);
e_s = saturation vapour pressure (kPa);
r = mean atmospheric density (kg m^{-3});
C_p = specific heat of the air (MJ kg^{-1} $°C^{-1}$);
r_a = aerodynamic resistance;
γ = psychrometric constant (kPa $°C^{-1}$);
r_c = canopy resistance.

While some of the above quantities are numerical parameters, some others are obtained by simpler models, which we list in Table 1. Those models will be stored in the knowledge base and they will be implemented according to a specific design, language and platform by the tools of Figure 1.

Table 1: Discrete model units stored in the knowledge base and implemented from their XML representation.

Simple models	**Symbol**	**Explanation**
DADSmith	r	Atmospheric density
DAVPRHFAO	e_a	Actual vapour pressure
DSVPTetens	e_s	Saturation vapour pressure
DSVPDTetens	s	Slope of saturation vapour pressure deficit
DPsychrometric Constant	γ	Psychrometric constant
DARFAO	r_a	Aerodynamic resistance
DVPDFAO	VPD	Vapour pressure Deficit (requires e_s and e_a)
DNRFAO	R_n	Net radiation
Composite model		
DRETFAO56	ET_0	Reference evapotranspiration

3.2. Principles of conversion

The conversion of a model from its declarative format into a procedural form, ready to be compiled and executed, can be automated, following a principled approach.

In our case studies, the essential ingredients are a set of XML schemata describing the Domain Concepts and the Model Concepts used in the formulation of our modelling exercise.

A model concept relies on the definition of entities such as inputs, states, parameters and outputs. A model is also characterised by its state transition equation and output transformations. These model concepts need to be mapped onto the domain concepts to associate a meaning with them. For instance, the model concept labelled R_a needs to be mapped into the domain concept Net Radiation to assume a numeric value characterised by a unit and a dimension.

The conversion algorithm, implemented in the tools of Figure 1, will then parse all the model concepts, retrieving the data type information from the domain concepts in order to create the interface of the model component class in the target language or modelling framework. The model classes are implemented as components, exposing an interface for their use in various contexts and environments. The interface varies according to the target framework, but in general it will include accessor methods, to set input values and get outputs, execution methods, to fire the model equations, and test methods, including pre and post-conditions.

The code of the body of the methods is then generated according to the content of the state transition and output transformation equations.

In case the model is composite, some model variables can be obtained by computing sub-models. The XML schema of the model concepts allow to specify such a condition and the generated code for the model equation can refer either to another declaratively generated code or to a call to a binary component, implementing, for instance, a specialised numerical routine.

Finally, we assume that the target modelling framework complies to a pattern where the simulation algorithm is separated from the model equations. In other words, the model components simply compute the rate of change of the state variables and the output values at time t, while the simulation algorithm takes care of using the state and rate values to perform the numerical integration (Rizzoli et al. 1998).

Note that in the case study presented in this paper, the model is static and therefore it does not make use of state variables, the model will only need its output transformations.

3.3. Conversion frameworks

The information provided in the XML file can be used to generate code targeted either at a specific framework, or to components with no dependency to a specific framework and which can be used in different frameworks via a wrapper class.

Although platform independent in principle, when a reference to external to the knowledge base

binaries is needed, an attribute specifying the target platform is added and selectively used by different code generation applications. The applications to generate code for the NET platform make use of the classes in the `System.CodeDom` namespace of the .NET 2 framework, whereas Java code is generated using e.g. JAXB.

3.3.1 To non-framework specific components

The classes generated implement a design to build extensible components via the design pattern strategy (Mesketer, 2004), use extensible domain classes in the interface (Athanasiadis et al., 2006), implement interfaces also to facilitate the discovery of types and properties via reflection, and implement test of pre and post conditions (Meyer, 1997). An example implementation of this design is in Donatelli et al., (2006b).

A model class, called "strategy", includes the definition of parameters using the same type used to define variables in Domain Classes (VarInfo); the VarInfo values are static properties of the class. The `IStrategy` interface implemented by these classes contains the methods to run the model, test pre and post conditions, reset model outputs, and set parameters default values. The application Strategy Class Coder requires as inputs, besides the XML model representation, a class name and the namespace, and the domain class name used to instantiate the input output object of the `IStrategy` implementation. The composite model generated is associated with the simple strategies.

3.3.2 To JAMS

The Jena Advanced Modelling System (JAMS) provides a framework for the development, composition and application of distributed and process oriented environmental models. JAMS implements the system core which provides spatial and temporal contexts in form of containers called compound-components into which single components can be embedded which implement the calculation procedure of single more or less complex processes. Each process component defines in a declarative part what input it needs and what output it provides followed by the process implementation in which the calculation takes place. The single process components are implemented in knowledge libraries, which are linked to JAMS during runtime.

The description of a model for the JAMS environment is made by an XML file in which the spatial and temporal contexts are defined by compound component entries which contain the process component entries in their correct order. In the process entry the variables and parameters for each component they need during runtime are defined by their names and the provider (e.g. another component or context) which is able to deliver them:

```
<jamsvar name="airTempMax"
provider="HRUContext"
providervar="currentEntity.maxTemp"/>
```

During runtime the XML file is parsed and the components are processed by calling their run() routine which contain the system independent process implementation.

3.3.3 To ModCom

ModCom is a modelling framework that was first described by Hillyer et al. (2003). Recent developments, including a C# implementation, are available online (Anonymous, 2006). A ModCom simulation model consists of a number of independent component models. The component models of a composite model communicate through input- and output-ports. Ports are discovered and connected through the properties and methods of the `ISimObj` interface which is implemented by all component models.

In this study, `ISimObj`-derived classes were generated for each of the simple component models listed in Table 1, after which the composite model was created by coupling instances of the simple models using the port mechanism. The following is a C# statement effectuating a single link between two component models:

```
crop.Inputs["airTempMax"].Data =
 metReader.Output["airTempMax"].Data;
```

All prototype applications and sample files are available at: http://craisci.icamodelling.it/codegen

4. DISCUSSION

While by no means trivial, the software aspects of knowledge-based systems are well within reach, and prototypes of knowledge-based modelling systems are in use today. An immediate, yet important advantage of declarative modelling is the portable specification exemplified by the case studies above, which yields uniformity of representation to models that can be adapted to the infrastructure of choice without a need to coordinate with the model development. Such models can be understood, extended and improved collaboratively by groups working with different architectures with no concern for technical detail.

On a higher-level vision, modelling at the conceptual level allows users to employ a language that is tailored to the knowledge domain of reference, adding the necessary dynamic information to the definition so obtained, and letting the software infrastructure infer an appropriate computing workflow. So-called domain ontologies can reflect the terms and relationships proper of different disciplines and make disciplinary modellers feel at home while at the same time allowing automated integration between

independently developed disciplinary data and models.

In the longer-term, the vision for declarative modelling merges with the more general one of a *Semantic Web* (Antoniou and van Harmelen, 2004) where model content, similar in structure to other content available online such as data or text documents, can be searched, retrieved and simulated from the web with no more specialized knowledge than it is currently necessary to access any web page. The opportunity to achieve such a vision lies in the formal expression of model content according to homogeneous ontologies and in the capability of associating specific conceptualizations with ad-hoc infrastructure capable of using it transparently.

5. CONCLUSIONS

The approach presented here illustrates a first step towards building a knowledge base of model and data content that can be used across infrastructures.

The XML representation chosen is a first simplification of a platform independent model to be used for biophysical modelling, but it has been useful to approach an operational use of the knowledge base.

Making this knowledge base homogeneous with the larger semantic web will provide services to modellers whose implications in terms of scientific sharing, integration, collaborative development and peer review we can only begin to envision.

6. 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). F. Villa is also funded by the US National Science Foundation (grant 0225676, ITR-SEEK).

6. REFERENCES

Allen, R.G., L.S. Pereira, D. Raes, M. Smith. 1998. Crop evapotranspiration: Guidelines for computing crop water requirements. *Irr. & Drain*. Paper 56. UN-FAO, Rome, Italy.

Anomymous, 2006. ModCom [Online]. http://www.modcom.wur.nl (verified on February 6, 2006).

Antoniou, G. van Harmelen F. 2004. *A Semantic Web Primer*. MIT Press, Cambridge, MA.

Athanasiadis I., A. Rizzoli, M. Donatelli, L. Carlini. 2006. Enriching software model interfaces using ontology-based tools. In: Proc. of the iEMSs Congress, Vermont, USA, July 2006.

Costanza, R. Duplisea, D., Kautsky, U. 1998. Introduction to Special Issue Ecological Modelling on modelling ecological and economic systems with STELLA. *Ecological Modelling*, Vol. 110, No. 1, pp. 1-4.

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

Genesereth, M. R., and Nilsson, N. J. 1987. *Logical Foundations of Artificial Intelligence*. San Mateo, CA: Morgan Kaufmann Publishers.

Gruber, T.R. 1993. A translation approach to portable ontology specifications. *Knowledge Acquisition*, Vol. 5, No. 2, pp. 199-220.

Hillyer, C., J. Bolte, F. van Evert, and A. Lamaker. 2003. The ModCom modular simulation system. *European Journal of Agronomy*, Vol. 18, No. 3-4, pp. 333-343.

Mesketer, S. J., 2004. *Design Patterns in C#*. Addison – Wesley, Boston, MT, USA, pp. 247-256

Meyer, B. 1997 *Object-oriented software construction, 2nd edition*. Prentice Hall, Upper Saddle River, NJ, USA,

Muetzelfeldt, R., J. Massheder. 2003. The Simile Visual Modelling environment. *European Journal of Agronomy*, Vol. 18, No. 3-4, pp. 345-358.

Rizzoli A.E., J.R. Davis, D.J. Abel . 1998. Model and data integration and re-use in environmental decision support system. *Decision Support Systems*, Vol. 24, No. 2, pp. 127-144.

Sperberg-McQueen, C. M. and H. Thompson. 2000. XML schema [Online], http://www.w3.org/XML/Schema (verified on February 23, 2006).