Jul 1st, 12:00 AM

A framework for integrated modeling using a knowledge-driven approach

Jean-Pierre Müller

Follow this and additional works at: https://scholarsarchive.byu.edu/iemssconference

https://scholarsarchive.byu.edu/iemssconference/2010/all/57

This Event is brought to you for free and open access by the Civil and Environmental Engineering at BYU ScholarsArchive. It has been accepted for inclusion in International Congress on Environmental Modelling and Software by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.
A framework for integrated modeling using a knowledge-driven approach

Jean-Pierre Müller*
*CIRAD-GREEN, Campus International de Baillarguet, 34398 Montpellier cedex 5, France (jean-pierre.muller@cirad.fr)

Abstract: In this paper, we argue that integrated modeling is about integrating disciplinary discourses reflecting various points of view on a given system rather than only setting up a very complex model from the start, or putting together independently designed models. In our approach, we propose an integrated modeling environment, based on ontologies, to represent and articulate the various points of view. Accordingly the proposed framework, called MIMOSA, is based on two components: 1) a declarative component based on ontologies to describe the concepts, the modeled system using the described concepts, and its dynamics using an extensible set of formalisms, 2) an executive component based on DEVS (Discrete Event System), used to map the declarative component into. Additionally, we describe the architecture to initialize and to observe the simulation. The originality lies in the way we are using ontologies and declarative modeling to design the model, the use of DEVS, as well as in supporting the whole process within a software platform. The approach is illustrated with a socio-ecological model for assessing the sustainability of community-based regulations in Madagascar.

Keywords: complex system modeling, ontology, discrete-event simulation, conceptual model, modeling platform, simulation platform, multi-modeling.

1. INTRODUCTION

The first question is: what do we mean by modeling? Minsky [1965] provides a very broad but accurate definition: "A is a model of B for X if manipulating A allows to answer the questions of X on B". In our case A is any formal structure (set of differential equations, computer programs, etc.) that represents the aspects of B that are necessary and sufficient for answering a question of X by only manipulating A. If A represents aspects of B, therefore A is a form of knowledge representation in the semantic sense.

The second question is: what do we mean by integrated modeling? Taking for granted that the question mainly arises for complex systems, a possible answer is to create a sophisticated model still retaining and articulating all the relevant aspects of B. But designing A directly from what we know about B appears as a challenge. This challenge can be overcome by building distinct models A' of parts of B and then by plugging them together (object-oriented or multi-agent models appear as being the extreme of this idea). This multi-modeling approach raises the issues of merging multiple formalisms and of data compatibility among the models A'. However the multi-modeling approach is more than a technical problem. It primarily is a multi-disciplinary problem where the question is the elicitation of the concepts used by the various disciplines (and even within a discipline from various points of view), their mutual understanding and, in fine, their articulation (if not theoretically incompatible).

It exists a number of modeling and simulation platforms that allow sophisticated modeling like Swarm (Minar [1996]), Repast (Collier [2002]) or, more specifically, SME (Costanza et al. [2004]). Others provide the functionalities for combining models, inclusively written with heterogeneous formalisms or programming languages, like HLA (Kuhl et al. [1999]), platforms based on DEVS (Zeigler et al. [2000]) or, more specifically, OpenMI (Moore et al. [2005]). Finally some platforms provide easy (graphical) ways to specify the behaviors.

---

* This definition also entails the living models used by the biologists, e.g. the laboratory mouse.
like Repast Symphony, Stella (Richmond [2001]) or Atom (de Lara & Vangheluwe [2002]) using meta modeling. The later platforms illustrate what (Villa et al. [2009]) call declarative modeling. In the same reference, declarative modeling is opposed to semantic modeling where the formalization of the domain discourse is central. In most approaches the formalization of the domain discourse is made using ontologies (Gruber [1993]). However, few platforms actually propose to support knowledge-driven modeling by starting from ontologies, by using these ontologies to describe concrete systems, possibly combining various points of view, and finally by generating the simulation model (i.e. the mathematical or programming structure). The objective of this paper is to propose such a framework, called Mimosa, using ontologies for the system structure, declarative modeling for the dynamics, and using DEVS as the target formalism.

The next section shall describe the overall framework including 1) the declarative component to represent the concepts, the dynamics, and their instanciation to describe the system, 2) the executive component to run the simulations, 3) the mapping between these two components, and finally 4) the initialization and the monitoring of the simulation. Then, we shortly present an application illustrating the framework possibilities before we conclude and present a number of perspectives.

2. THE MIMOSA FRAMEWORK

The MIMOSA framework includes two components. The declarative component is concerned with knowledge representation and therefore provides tools to describe what one knows about the system under study. It is called the declarative component because it is assumed to be endowed with a denotational semantics over the modeled system. The executive component is a formal structure endowed with an operational semantics that defines how the simulation is produced. A mapping between the declarative and the executive components is defined.

2.1 The declarative component

The declarative component is aimed at representing the knowledge we have about the modeled system and its dynamics. As argued in the introduction, we consider that modeling a system is beforehand a knowledge representation process. Consequently, it is not enough to describe a particular system, we have to also describe the concepts or categories we are using to describe the modeled system. We use the ontologies for knowledge representation. According to Tom Gruber at Stanford University (Gruber [1993]), an ontology in the context of computer science is "a description of the concepts and relationships that can exist for an agent or a community of agents." The ontologies are based on descriptive logics where there is a distinction between the terminological box (or T-Box) and the assertion box (or A-Box).

The T-Box recursively defines new categories from a set of primitive categories using set theoretic operators (complementation, union and intersection) as well as semantic relationships called roles. The roles correspond to the relationships and attributes, an attribute being a relationship with a primitive category. The recursive definitions build the taxonomy of categories. The A-Box describes a particular system as a set of linked individuals as instances of the categories and the roles defined in the T-Box. A standard textual language based on XML exists for representing ontologies (OWL [2004]), but there is no agreed upon graphical representations. In Mimosa, we have chosen to use a simplified UML-like class diagram (Bommel and Müller [2007]). In the following, we shall call the T-Box the conceptual model (named T) and the A-Box the concrete model (named A). A conceptual model \( T \) is a tuple \(<PC,C,D,R>\) where \( PC \) is the set of primitive category names, \( C \) is the set of the non primitive category names, \( D \) defines each element of \( C \) from other categories of \( PC \cup C \), \( R \) are tuples of \( \text{RoleName}\times C \times C \) defining the semantic relations. A tuple of \( R \) between an element of \( C \) and an element of \( PC \) is called an attribute. A concrete model \( A \) is a tuple \(<I,L>\) where \( I \) is a set of individuals instances of \( PC \cup C \), and \( L \) are tuples of \( \text{RoleName}\times I \times I \).

In figure 1, the conceptual model for the stupid model (Railsback et al. [2006]) is graphically represented on the left and its corresponding concrete model is represented on the right. In the conceptual model, the category "Space" is described as a structured set of "Cell"s with a neighborhood relationship. The category "BugPopulation" is described as an
unstructured set of "Bug"s. The category "BugPosition" defines a mapping between the population of bugs and the space of cells. In the concrete model, we only have three individuals: an instance of "BugPopulation", an instance of "Space" and an instance of the position relationship.

The use of ontologies was already suggested to formalize the modeling process itself like in the FEARLUS project (Pignotti [2005]), to annotate existing simulation components and to build a catalog of simulation formalisms and component types like in DeMO (Miller and Baramidze [2005], called semantic annotation in Villa & al. [2009]), or to type the events exchanged among simulation components (Rizzoli & al. [2008], called semantic mediation in Villa & al. [2009]). However, it has been rarely proposed directly to specify simulation models.

The obvious limitation of ontologies is their inability to specify dynamics. In fact, it would be possible to provide ontologies for the various formalisms (differential equations, automata, petri nets, state charts, rule-based systems, etc.) and to use it to instantiate concrete process descriptions as it is made in AToM³ (de Lara & Vangheluwe [2002]). Villa & al. [2009] propose to use the ontologies by adding the causal relationships to describe the dynamics. However, it is very heavy or not fully integrated with the ontology of the system structure.

We decided to directly associate to each category the chosen formalism and the associated declarative process specification. As an example, we can associate to the BugPopulation a population growth dynamics using a difference equation \[ \text{pop}(t+1) = \text{pop}(t) \times \text{popGrowthRate} \], to the Bug a food search strategy and to the Cell a food growth equation. We have chosen to associate the concrete processes to the categories because we consider that the process is homogeneous for each category in a model. Although the parameters can be different for each individual and the processes and the formalisms can be completely heterogeneous among the various categories.

We finally come up with the declarative structure \( DS \) being a tuple \( <T,A,D> \), where \( T \) is the conceptual model made of categories and taxonomic and semantic relationships, \( A \) is the concrete model made of linked individuals as instances of the categories and the semantic relationships, and \( D \) is a mapping of the categories of \( T \) (\( C(T) \)) into the pairs \( <f_i,p_i> \) (where \( f_i \) is a formalism name and \( p_i \) a process specification of which form depends on \( f_i \)). Therefore, we assume, the existence of a set \( F \) of formalism names with for each \( f_i \) an associated set of possible process specifications \( P(f_i) \).

2.2 The executive component

The executive component is based on coupled DEVS models (see Ziegler [2000]) with some extensions as described by Müller [2009] together with the associated operational semantics. The operational semantics is based on discrete-event system simulation. A DEVS model is a tree structure of which leaves are atomic DEVS models and the root and intermediate nodes are coupled DEVS models.

Structurally, an atomic DEVS model, we shall call an entity, is a box with input and output ports. A coupled DEVS model is such a box that contains a set of DEVS models with interconnected ports that include the input/output ports of the enclosing box (see figure 2).
Figure 2. DEVS models: (a) atomic, (b) coupled.

Operationally, an atomic DEVS model is defined by a tuple:

\[ <X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda_{int}, \lambda_{ext}> \]

where \( X \) is the set of input events, \( Y \) is the set of output events, \( S \) is the set of (possibly continuous) states, \( \delta_{ext} \) the transition function in response to incoming external events (\( \delta_{ext}: S \times X \times S \)), \( \delta_{int} \) the transition function in response to the internal event (\( \delta_{int}: S \rightarrow S \)), \( \delta_{con} \) the confluent function when both incoming external events and the internal event happen at the same time, \( \lambda_{int} \) the function producing the outgoing external events, \( \lambda_{ext} \) the function producing structural changes. There are some slight changes in the definition with respect to the standard one:

- We make the distinction between the structural description (the entities) and the process specification (the operational DEVS model);
- It is allowed to receive and produce sets of events as in //−DEVS in order to manage simultaneity;
- \( \lambda_{int} \) is another name for the time advance function usually called \( ta \);
- \( \lambda_{ext} \) produces events of which effects are to change the DEVS structure (creation/deletion of DEVS models and connection changes). This functions allows to describe dynamical systems with dynamical structures.

Semantically, \( \delta_{int} \) defines the reaction to incoming events. If nothing happens for a given time (\( \lambda_{int} \)), events are issued (\( \lambda_{ext} \)), an internal transition is made (\( \delta_{int} \)) and the structure is possibly changed (\( \lambda_{int} \)). The operational semantics of a coupled DEVS model is recursively defined on the operational semantics of its components. In practice, the hierarchical structure is flattened defining a DEVS-bus where the atomic DEVS models are plugged and interconnected.

We finally come up with the executive structure as a tuple \( <E, L, P> \), where \( E \) is a set of entities, \( L \) is made of tuples \( <e_i, out_{ij}, e_k, in_{kl}> \) where \( e_i \) and \( e_k \) are entities and \( out_{ij} \) and \( in_{kl} \) are their output and input port respectively, \( P \) associates to each element \( e_i \) of \( E \) its operational DEVS structure \( <X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda_{ext}, \lambda_{int}, \lambda_{str}> \).

2.3 Mapping the declarative structure into the executive structure

The mapping from the descriptive structure into the executive model consists in mapping the annotated ontology \( <T, A, D> \) in the executive structure \( <E, L, P> \). For the time being, it is defined as follows:

- Each individual \( a_i \) of \( I(A) \) is mapped onto an entity \( e_i \) in \( E \);
- Each link between two individuals of \( A \): \( <r_i, a_i, a_j> \) is mapped onto a tuple \( <e_i, r_i, e_k, in > \) in \( L \), where \( r_i \) is an output port of \( e_i \) and in is the default input port for all the entities;
- For each individual \( a_i \) in \( A \), if it is an instance of the category \( c_j \) in \( C \), then the operational DEVS structure \( <X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda_{ext}, \lambda_{int}, \lambda_{str}> \) is derived from the specification \( D(c_j) \) and attached to \( e_i \).

Of course, the way the operational structure is derived depends on the formalism in which the dynamics is expressed. If the formalism is DEVS itself, the derivation is one to one. Otherwise, it must exist for each available formalism:

- Either a way to transform directly the process expressed in this formalism into the corresponding operational structure (compilation);
- Or an already defined operational structure parameterized by the process description (interpretation).

For example, for the differential equation formalism, it is enough to define \( \lambda_{int} \) as the integration step, \( \delta_{int} \) as performing the integration, \( \delta_{ext} \) getting the variable values the
equation depends on and \( \lambda_{\text{ext}} \) issuing the newly computed variable values. It has been shown that any formalism can be mapped into DEVS allowing a multi-formalism approach to modeling (Zeigler [2000]).

Currently, we only consider the interpretation of the process specifications. The following formalisms are currently implemented: direct scripting of the DEVS functions using Python, Java, Scheme, etc., markov processes and state chart diagrams. More can be added when needed through a plugin architecture.

### 2.4 Initializing and observing

The executive component only partially defines the initial state of the simulation (the entities and their interconnections). It remains to define the initial state of each entity. By default, the initial state is generated from the attributes of the corresponding individuals and their values. However any source of information to set the initial state could be considered: files, databases, geographical maps, generators of various sorts (for example generators of random graphs). To achieve this, we introduce the possibility to annotate the individuals with initialization methods (see figure 3 where they appear as pseudo UML stereotypes of the corresponding individuals). The set of available initialization methods can also be extended through a plugin mechanism.

Last but not least, the executive component must be monitored to understand the system dynamics and answer the question. Two strategies are possible for observing the evolution of the simulation state: 1) the active strategy consists in having each entity signaling its state changes, 2) the passive strategy consists in having each entity providing information on its state only on demand.

The first strategy is systematic but very time consuming. The second strategy requires to separate the definitions of the sampling policy from the executive model execution. In MIMOSA, we provide the two possibilities: 1) for the active strategy, it is possible to associate to each process a set of probes to be issued at each state change. We added an additional output function \( \lambda_{\text{probe}} \) to the operational DEVS structure called after each transition, 2) for the passive strategy, a pair of functions has been added to the operational DEVS structure in order to request and to get information on an entity state: \( \delta_{\log} \) and \( \lambda_{\log} \). Therefore, any entity can monitor the state of another entity with any strategy (fixed time steps, when some events occur, etc.).

These observations have to be output in a certain way: visualized, saved in files, databases or maps, or directed to other programs. We, therefore, provide an extendable set of output components to which the probes can be directed (see figure 3).

![Figure 3. The concrete model with the initialization and output specifications.](image-url)

Finally, the parameters of the initialization method as well as the components of the visualization output can be grouped into a control panel specification that is used to generate the simulation control panel like in NetLogo or Stella (figure 4).
3. AN APPLICATION

As an illustration, we shall present the architecture of a real-life application of the Mimosa platform: MIRANA a socio-ecological model for assessing the sustainability of community-based regulations in Madagascar. The Malagasy local communities managing forest resources have difficulties in assessing the impacts of the management plans they decide upon. To help them, we have designed an integrated model with the ecological processes, the various norms (zoning, quota, etc.) and the resulting inhabitants behavior in order to explore the impacts of scenarios.

In MIRANA, the conceptual model is made of the set of ontologies (about 150 categories in total) describing 1) the actors of the system (households, community, etc.) provided with needs (food, money, etc.) and objectives (conservation, production, etc.), 2) the resources they are acting on (lands, animal and vegetal species, etc.), 3) the actions carried out by the actors on the objects (hunting, cultivating, etc.), and 4) the various norms regulating the actions. Figure 5 shows the ontology structure. For the dynamics, the actors are provided with planning mechanisms and the resources are provided with spontaneous processes (fertility dynamics, growth of biomass, etc.) as difference equations. In figure 5 the stereotypes under the concept names denote the used formalisms.

The concrete model (figure 6) contains essentially the territory of the local communities, the households and the administrative structures. The structure of the actual landscape and the placement of the households and species in the landscape are generated from a database that describes the site. The configuration of a particular site with a given set of activities, ecological dynamics parameters, objectives and regulations is made using GIS and excel files loaded into a database to generate the simulation model through initialization methods (as stereotypes of the instances).

The question is about the sustainability of the management plan in ecological, social and economical terms. Therefore the simulation produces indicators directly related to the sustainability issue. They are visualized and stored in the database through a set of output components.

For the time being, only the surfaces of the habitats, the surface of cleared land, the soil fertility and the satisfaction rate of rice need are computed, and some sensitivity analysis are made by launching the Mirana model in batch mode using MatLab®.

4. CONCLUSION AND PERSPECTIVES

We have presented a modeling and simulation framework, called MIMOSA, in which we are seriously considering knowledge-driven modelling in its broader sense. The ontologies are used for knowledge representation about the system structure, with the concepts that we need to describe the system (the conceptual model), and the instantiation of these concepts to model a particular system-(the concrete model). Because modeling is also about describing the dynamics of the system, we propose to attach the description of the various processes as <formalism, specification> pairs to the relevant concepts. For simulation, the target formalism and architecture DEVS has been chosen and a way to map the discourse onto the target architecture is presented. Additionally, we described an extensible
architecture to add new formalisms, initialization methods and output components (visualization, data storage). Finally, we shortly described a full fledge application to illustrate the possibilities of the proposed framework. An unexpected outcome of using Mimosa is the possibility with the conceptual model to actually describe a class of systems and, in doing this, to pave the way towards hierarchies of more or less generic and/or reusable models. An expected outcome of using Mimosa is the elicitation of the concepts used by various disciplines, opening the possibility to support not only multi-modeling but, more importantly, multi-disciplinary modeling.

The perspectives are numerous. At the multi-disciplinary modeling level, a tool like Mimosa needs a methodology to actually design these conceptual models in interaction with a multiplicity of experts and even stakeholders. At the architectural level, we mentioned the possibility to also use the ontologies as meta-models to describe the process formalisms themselves. It is planned to actually add this possibility to Mimosa using the results of Atom³ (de Lara & Vangheluwe [2002]). In the same vein, we are using more and more ontologies to also describe the actions and the events that, in fine, are exchanged among the DEVS entities. Therefore, semantic mediation as mentioned in Villa & al. [2009] is also possible. The related perspective is to make the mapping from the declarative structure into the executive structure more sophisticated in order to actually translate these
event descriptions as event data types at the simulation level. Currently, the initialization methods are only used to specify the initial state of each entity but not parts of the coupled DEVS structure. An obvious perspective is to be able to generate sub-networks from the specifications (for the example, the complete cellular automaton structure) before running the simulation.

REFERENCES


