Discrete-Event Multimodels and their Agent-Supported Activation and Update

Levent Yilmaz

M&SNet: Auburn Modeling & Simulation Laboratory
Department of Computer Science and Engineering
Auburn University
Auburn, AL 36849, U.S.A.

Tuncer I. Ören

M&SNet: Ottawa Center of the MISS
School of Information Technology and Engineering (SITE), University of Ottawa
Ottawa, ON K1N 6N5, CANADA

Abstract

A broad range of problems involve multiple levels and resolutions of scientific or engineering phenomena. Hence, the significance of simulation modeling at multiple levels, scales, and perspectives is well recognized. Multiscale modeling is highly interdisciplinary, with progress occurring independently across fields. Yet, disparate efforts involving the design and development of various types of multimodels result in incoherent solutions that lack a common unifying framework. This paper provides a rationale for a taxonomy of discrete-event based multimodel methodology that is equally applicable for continuous and discontinuous-change models expressed in ordinary differential equations and hence explores the viability of realization of a unified multimodel formalism. A multimodel is perceived as a container component that can be (re)configured with alternative model families. Technical challenges in model updating and agent support in model relationship management across levels and scales in multimodels are discussed. Novel agent-augmented model design patterns are presented to illustrate plausibility of interchangeable strategies for the realization of different types of multimodels within a single coherent framework.

1 Introduction

Many real-world phenomena can not be modeled by one single model; rather, they require the use of a set of complementary models that together are able to describe the whole process. Various efforts in computational science (Weinan and Engquist 2003) and social science (Gilbert and Troitzsch 1999) are indicative of the need for tools that facilitate development of models that span multiple scales to integrate the continuum from the micro to macro resolution of the same phenomena. While simulation modeling and computational science communities are actively developing model representation strategies that can capture multiple scales, formalisms (Zeigler at al. 2000), resolutions (Davis and Bigelow 2002), and perspectives (Davis 2000) in terms of existing conventional simulation tools, there does not yet exist a unifying formalism that subsumes most, if not all, of these formalisms. The premise of this paper is that, by developing a generalized multimodel formalism that can be configured with alternative strategies one can instantiate various types of multimodels within the same model development framework. Such a strategy is in sharp contrast with the current trend in developing formalisms for different modeling needs, where distinct solutions for multiple representations of scale, resolution, and staging are defined. We argue that agent theory that is well recognized as a useful paradigm for model representation can also be used in the development of the theory and methodology of the design of multimodels and their simulators.

The paper is organized as follows. In section 2 we provide a brief overview of current efforts in discrete-event multimodel development. Section 3 suggests a generalized multimodel formalism based on the variation of submodel structure and activation policies. Section 4 presents a structural design strategy toward the development of a DEVS-based multimodel formalism. Section 5 extends the presented structure with observer and scheduler agents that support submodel activation.

2 Related Work on Multimodels

A multimodel is a modular model that subsumes multiple submodels that together represent the behavior of a phenomenon. Multimodel formalism was originally developed by Ören (1987, 1991) as a generalization of discontinuity in piecewise continuous systems. In the first formulation, a generic architecture for continuous-change models described by ordinary differential equations was developed and documented as templates. The representation was also extended to discrete-change models as well as to memoryless models, i.e., models without state variables.

The formalism influenced the development of combined simulation, which entails the integration of continuous and discrete-event simulations within the same system description. For instance, as a special case of multimodel, coupled multiformalism specification developed by Praehofer (1992) extended DEVS (Zeigler et al. 2000) formalism to provide a simulation environment for combined continuous/discrete-event modeling for hybrid simulation. Fishwick and Zeigler (1992) developed a FSA-controlled multimodel to simulate qualitative dynamics of physical systems. Davis and Bigelow (2002) define multiresolution as building a single model, a family of models, or both, to describe the same phenomenon at different levels of resolution in a mutually consistent way. Various efforts are underway within the computational biology community (Takahashi et al. 2004) to develop tools that are capable of capturing multiple algorithms, formalisms as well as levels and scales in representing cellular systems.

3 Multimodels

Most complex systems have several aspects –some dormant– with mutual influence. To be able to study more than one aspect of reality and their interactions at the same time or sequentially, one needs modeling methodologies such as multimodels. By expanding our horizon based on the variations on the multimodel structure and submodel activation behavior, we can identify various types of models.

3.1 The Rationale for a Multimodel Taxonomy

Taxonomy of multimodel types based on various plausible constraints imposed on the submodel structure and activation policies is introduced in (Yilmaz and Ören 2004). As shown in the synopsis in Appendix A, we consider two main criteria, i.e., structure and activation behavior of the submodels. Based on the submodel structure of a multimodel, we consider the number of submodels active at a given time and the variability of the structure. Conventional multimodels, where only one model is active at a time can be characterized as single aspect (sequential) multimodels. Depending on the nature of the activation information, multimodels can be constraint-driven, pattern-directed, or goal-directed. The structure of Appendix A also facilitates preparation of systematic glossaries of the terms –in this case types of multimodels– by providing a systematization of the concepts. For example, cyclic multimodel is a pattern-directed multimodel where there is a cycle in the selection of submodels. Similarly, an acyclic multimodel is a pattern-directed multimodel where there is no cycle in the selection of submodels.

This paper illustrates the adaptivity and flexibility required in realizing multimodels in terms of constraint-driven multimodels that have stationary or adaptive control policies for staging or switching models. Most complex phenomena operate not only over multiple scales and levels, but also stages. At each stage, the phenomena can be described in terms of an ensemble of models that represent the system at various levels and from different aspects. Dynamic model updating is fundamentally critical to multimodel design, as it is necessary to switch among models that represent distinct stages of the problem. Staging, if necessary, can be used to switch among models that represent different aspects, levels, and scales of the same system, unless they are defined as a coherent unit (ensemble). A unifying generalized multimodel formalism should enable both strategies.

3.2 Technical Requirements in Multimodel Design

The challenges in dynamic model updating in multimodel formalism are the issues involved in substituting a new model or submodel without taking the simulator offline. The following five conditions present the basic requirements (Litmus test) for dynamic model replacement.

- **Activation:** Submodel replacement must be initiated, either internally within the multimodel or externally by the simulator.
- **Integrity:** The consistency of submodels undergoing replacement needs to be preserved. The event scheduling and simulation protocol need to be restricted or regulated to facilitate interleaving of submodel replacement and/or update activities with the simulation events.
- **Submodel Instantiation:** The new (or selected) submodel must be dynamically loaded and linked into the run-time environment of the simulator (simulation engine). This requires new model and simulator decoupling strategies that avoid persistent connections.
- **State Reconstruction:** The state of a model must be reconstructed or at least resume from a specific state when re-instantiated after an update operation. This requires externalization through abstraction, state saving, transmission, and reconstruction after the update operation.
- **Simulator Rebinding:** Once a model is loaded and linked to the run-time environment, the simulator needs to be bound to the new model.

4 The *Structure* for the Macro Architecture of Multimodels

A multimodel can be seen as an adaptive and customizable container that includes a number of submodels, each one of which has their own dependencies. That is, a unified multimodel needs to be configured by alternative set of submodels depending on the type of the problem and simulation objectives.

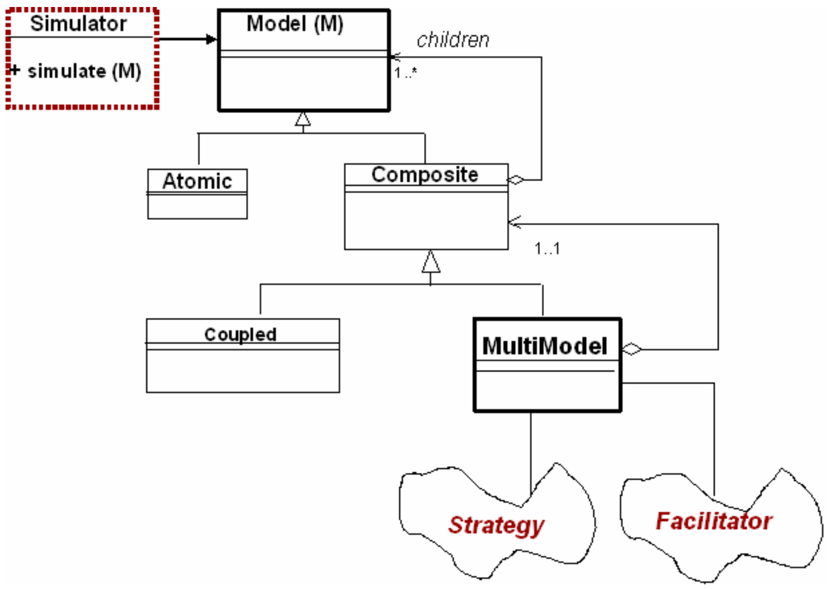

Figure 1: Extending DEVS Model Hierarchy with Multimodels

We use the DEVS framework as a basis to explore multimodel design strategies, as the separation of model, simulator, and experimental frame within the DEVS framework (Zeigler at al. 2000) provide convenient decoupling mechanisms that can be extended to facilitate realization of dynamic model and simulation updating. (As was mentioned the original specification of multimodels is applicable to continuous-change models described by ordinary differential equations (Ören 1987)). To this end, Figure 1 depicts a multimodel as a wrapper that aggregates submodels that are either atomic or coupled. Being a type of model, a multimodel provides the same uniform model interface to the simulator. The *facilitator* component aims to decouple the multimodel from the intricate details of instantiating a family of submodels to avoid explicit assumptions and facilitate its seamless reconfiguration with alternative ensembles of submodels. A common abstract interface is provided within the facilitator agent, as shown in Figure 2, to delegate the responsibility of initializing a family of models and their coupling relations to specialized constructor components.

assigned to decouple its construction from the generalized multimodel component. By selecting a different constructor component, separate alternative coupling configurations of the same model family can be produced by the controller components A_1 through A_n, as shown in Figure 2. Each one of the constructors refers to possible alternative stages of the same phenomena. The submodels associated with constructors designate the aspects, levels, and scales that are relevant to the given stage. As the simulator generates the behavior of the model, depending on the staging decisions, the multimodel needs to switch to an alternate submodel that produces and consumes the internal and external events. This requires avoiding permanent binding between the simulator and the multimodel. As such, the facilitator component includes a **facilitator agent** that acts as a bridge between the simulator and the submodels.

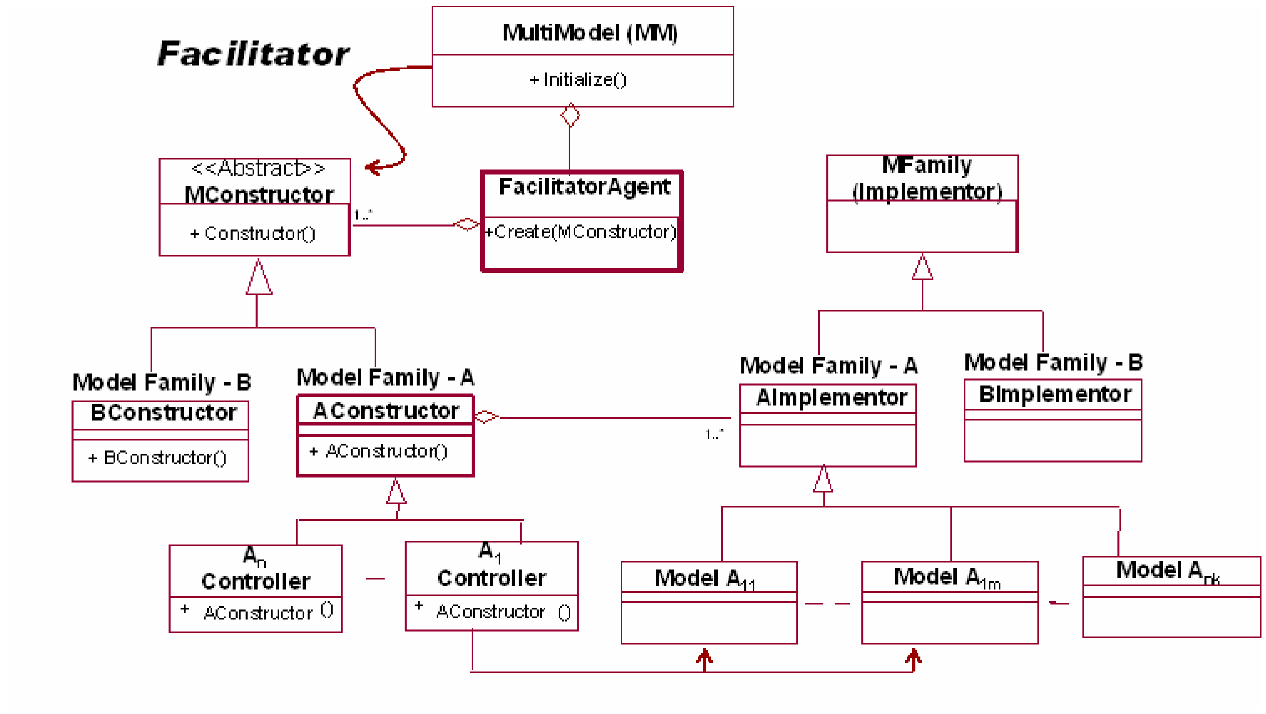

Figure 2: The Facilitator Agent

By the use of a common abstract interface, the multimodel does not need to hardwire a protocol for the construction of submodel components as well as their coupling across levels, scales, and aspects being modeled. For each family of submodels, a separate concrete constructor is

The facilitator agent provides a reference to the controller of the ensemble of active submodel(s) that can be updated via *update* request submitted by the **scheduler agent** discussed in section 5. The multimodel includes a reference to the abstract model family constructor that can be instantiated by any of the specialized constructor selected by the model developer at the time of initialization. The facilitator is parameterized with the same model family constructor at the time of its instantiation.

The concrete constructor component (i.e., AConstructor), using a reflective API, traverses each one of its child nodes (i.e., A_1Controller) to create controller objects that have the responsibility to instantiate ensemble of models that represent distinct stages, each one of which constitutes submodels that specify the phenomena at different levels, scales, and aspects. Model families within the constructor hierarchy that do not have more than one stage has only one controller object. Once all the controller objects associated with the model family are created, their references are returned back to the facilitator agent that switches among them under emerging conditions as the simulation unfolds. Notice that neither the multimodel nor the facilitator has direct reference or knowledge about the individual models and their coupling relationships. The facilitator agent brings a level of indirection, by which permanent connection between the simulator and submodels of the multimodel are avoided. Achieving the integrity and consistency among multiple models within an ensemble require managing relations among the submodels. The model family implementor abstract interface shown in Figure 2 provides a programmatic interface to a list of submodels that are active or latent within the multimodel.

The facilitator agent interleaves discrete-event simulation events with the update events to assure the integrity of the multimodel. An update task scanning phase within the controller object uses a set of update rules to determine if there exist any imminent updates due to modified state of any of the submodels in the ensemble to assure consistency across scales, levels, and aspects. This set of rules is independent of the submodel activation policy managed by the strategy component, by which the multimodel is configured.

5 Agent Support in Submodel Update and Activation in Multimodels

Depending on the activation policy of submodels, multimodels are classified into various types, including constraint-driven, pattern-directed, goal-driven multimodels. Each one of these multimodel types requires a distinct protocol and mechanism for activating submodels.

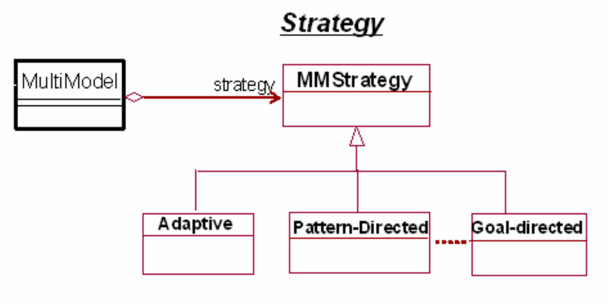

Figure 3: Submodel Activation Policies

While a pattern-directed strategy involves selecting submodels based on a predefined order, a goal-directed strategy involves run-time model qualification based on a planning mechanism. As such, the strategy component of the multimodel design shown in Figure 1 entails various mechanisms by which the multimodel can be configured at the time of its instantiation to realize the designated multimodel type (i.e., goal-directed, adaptive, multi-aspect). Figure 3 presents a number of strategy components as they relate to the multimodel container component. The approach entails the definition of a family of multimodel control strategies (protocols), encapsulating each one, and making them interchangeable. As such, the strategy component lets the submodel scheduling policy vary independently from the multimodel that uses it.

Figure 4 presents the gross organizational layout of the multimodel components with the emphasis on the context of the strategy component within the multimodel architecture. An *abstract model* is defined to represent the implementation-independent state space representation of the multimodel. This abstract model is constantly updated by the facilitator as the behavior is generated.

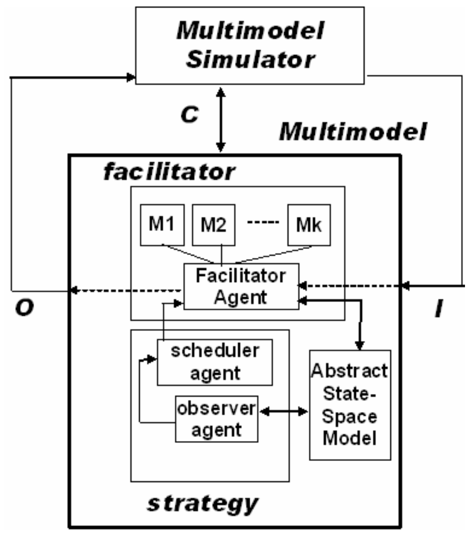

Figure 4: Components of a Multimodel

The facilitator also uses the same abstract model to enable continuity between distinct models. That is, the *state reconstruction* requirement listed in section 3.2 is handled by using the abstract state model to (re)-initialize an activated model. Translation of (to) the state of submodels to (from) the abstract state space is handled by the facilitator agent. A constraint-driven adaptive multimodel needs to provide a controller that decides the conditions under which submodels can be switched for activation. Changes in the state-space are observed by an *observer agent* that subscribes to updates to the data components of the abstract model. The notifications provided to the observer agent are evaluated to match the transition conditions of the reactive controller (i.e., finite state model) embedded within the scheduler agent. The satisfiability of a transition condition associated with the current state of the scheduler designates a target state that represents a subsequent submodel that needs to be activated to switch to a new stage within the problem space. Once the scheduler decides to select a new submodel, the facilitator updates its active submodel with the new selection. The new active submodel is then used to generate the behavior of the multimodel. The input (I), output (O), and controller (C) channels depicted in Figure 4 are used to derive the behavior of the multimodel consistent with the original DEVS simulator protocols (Zeigler et al. 2000). The input channels are coupled with the facilitator agent that decides to which submodel the message needs to be delegated. Similarly, the output of the active submodel is passed to the output channels of the multimodel via the facilitator agent. The controller channel is used to communicate model update decisions to the simulator so that it switches to the corresponding simulator (i.e., basic simulator or coordinator as defined in the DEVS formalism) that is associated with the type of the active submodel (atomic or coupled model).

The control policy within the scheduler agent can be defined either as a stationary or adaptive protocol with reinforcement learning mechanism. A stationary protocol can be defined as

$$SP = \langle S, M, P, action, next \rangle,$$

where

$$M=\{M1, M2, M3, \ldots, Mk\}$$

denotes the set of submodels, and *S* depicts the set of internal states, each one of which is labeled with $m \in M$. *P* denotes the *percepts* provided by observer agent. The observer agent maps the abstract state-space onto percepts that refer to the transition conditions within the controller policy. The action selection function is a mapping,

$$action: S \rightarrow A,$$

from internal states to actions. Actions are outputs of the scheduler agent that are used to update the facilitator agent with the submodel designated by the label of *S*. The function *next* maps an internal state and percept onto an internal state:

$$next: S \times P \rightarrow S.$$

The behavior of a state-based scheduler for a stationary constraint-driven multimodel can be summarized as follows: The scheduler starts in its initial state depicting the initial submodel that represents the multimodel. The observer agent observes the changes in the abstract state space of the multimodel and generates a percept. The internal state of the scheduler is then updated based on the type of the percept. An action is performed to update the facilitator to select a new submodel. Often, a percept and the following action denote a change in the stage of the problem, which then requires the selection of a new submodel.

6 Conclusions

Exploring a phenomenon at multiple levels as well as temporal and spatial scales is becoming a significant concern in various scientific fields. This paper explored the challenges and requirements in multimodel design, where dynamic model updating is considered to be critical. The significance of decoupling a multimodel and simulator from the submodels via new levels of indirection is argued to be critical in flexible staging and switching among submodels that represent the behavior of a phenomenon at different aspects, levels, and scales. The viability of realizing various types of multimodels (i.e., multiscale, multiresolution, multiaspect) within a single unifying multimodel formalism is emphasized. The notion of parameterized multimodels that can be reconfigured with alternative strategy and control objects is presented as a plausible approach to develop a unified and generalized multimodel framework.

References

Davis K. P. (2000). “Exploratory Analysis Enabled by Multiresolution, Multiperspective Modeling," in Jeffrey A. Joines, Russell R. Barton, K. Kang, and Paul A. Fishwick (eds.)*, Proceedings of the 2000 Winter Simulation Conference*, 2000.

Fishwick P. and B. E. Zeigler. (1992). “A Multimodel Methodology for Qualitative Model Engineering,” *ACM Transactions on Modeling and Simulation*, vol. 2, no. 1, pp. 52-81.

Gilbert N. and K. G. Troitzsch (1999). *Simulation for the Social Scientist*. Open University Press.

Ören T.I. (1987). “Model Update: A Model Specification Formalism with a Generalized View of Discontinuity,” In: *Proceedings of the Summer Computer Simulation Conference*, Montreal, Quebec, Canada, 1987 July 27-30, pp. 689-694.

Ören T.I. (1991). “Dynamic Templates and Semantic Rules for Simulation Advisors and Certifiers,” In: *Knowledge-Based Simulation: Methodology and Application*, P.A. Fishwick and R.B. Modjeski (Eds). Springer-Verlag, Berlin, Heidelberg, New York, Tokyo, 53-76.

Praehofer H. (1992). *System theoretic foundations for combined discrete-continuous system simulation*. Ph.D. dissertation, Johannes Kepler University, Linz, 1991.

Takahashi, K., Kaizu, K., Hu, B., and Tomita, M. A multi-algorithm, multi-timescale method for cell simulation *Bioinformatics* (2004) 20(4):538-546.

Weinan E and B. Engquist, "Multiscale Modeling and Computation", *Notices of the AMS*, **50**(9), 1062-1070 (2003).

Yilmaz L and T. Ören (2004). "Dynamic Model Updating in Simulation with Multimodels: A Taxonomy and Generic Agent-Based Architecture," In Proceedings of the SCSC'04. pp. 3-8.

Zeigler B. P., H. Praehofer, T. G. Kim (2000). Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems, Academic Press.

Appendix A. A Synopsis of Multimodel (MM) Formalisms

Based on	Additional Criteria				Type of multimodel (MM) (Synonyms are represented within parentheses)
Structure of submodels	**Number** of submodels active at a given time			Only one	Single aspect MM (Sequential MM)
				2 or more	Multiaspect MM
	Variability of structure (variability of number of submodels)	**Static**			Static-structure MM
		Dynamic (Dynamic-structure MM) (Variable-structure MM)	Number of submodels	Extensible	Extensible MM
				Depends on model's stage	Multistage MM
			Alterations of submodels	No	Non-mutational MM
				Yes	Mutational MM
					Evolutionary MM
Behavior (activation) of submodels	**Nature of knowledge** to activate submodels	Constraint-driven			Constraint-driven MM (Adaptive MM)
		Pattern-directed (Pattern-directed MM) (Metamorphic MM)	Submodel selection is cyclic	No	Acyclic MM
				Yes	Cyclic MM
		Goal-directed			Goal-directed MM (Exploratory MM)
	Location of knowledge to activate submodels	Within the MM (Internal activation of submodels)			Active MM (Internally activated MM)
		Outside the MM (External activation of submodels)			Passive MM (Externally activated MM)