Dynamic Model Updating in Simulation with Multimodels: A Taxonomy and a Generic Agent-Based Architecture

Levent Yilmaz
M&SNet: AMSL (The Auburn Modeling and Simulation Laboratory)
Computer Science and Engineering
Auburn University
Auburn, AL, USA
yilmaz@eng.auburn.edu

Tuncer I. Ören
M&SNet: OC-MISS (Ottawa Center of the McLeod Institute of Simulation Sciences)
School of Information Technology and Engineering (SITE), University of Ottawa
Ottawa, ON, Canada
oren@site.uottawa.ca

Abstract
Dynamic model update and replacement are critical capabilities for modeling problems, the underlying processes of which are uncertain. The trajectory of most complex social phenomena, including conflicts, for instance, is never fixed and predictable. Symbiotic simulation is yet another emergent challenge that requires model adaptivity in response to emerging conditions within the actual system. In such experiments simulation parameters, as well as scenarios may shift, invalidating a priori assumptions or analysis. Multimodel formalism and runtime dynamic model updating are suggested as potential approaches to deal with uncertainty. A taxonomy of multimodels, a generic agent-based architecture for static structure, and single aspect multimodels with exploratory (goal-directed) behavior are presented to explore issues in multimodel simulation.

Keywords: agent, composability, methodology, multimodel, multisimulation.

1.0 Introduction
For most realistic modeling problems, the nature of the problem changes as the simulation unfolds. Initial parameters and models can be irrelevant under emergent conditions, or there may be a need for switching to new set of models (i.e., multi-stage processes). For instance consider the water and ice in a mix: we have two submodels to represent: ice and water. Both submodels exist at the same time; we add (or subtract) calories; ice melts or water freezes and accordingly the two masses change. Each submodel is governed simultaneously by different (physical) laws (melting of the ice, heating of the water). Once all ice melts, only one subsystem is active; the ice subsystem becomes latent and the only active subsystem is the one representing the water. We continue adding calories; water is heated –under the present pressure. If the pressure is the one corresponding to sea level and if the water is pure, then at 100º C (or 212º F) the water starts evaporating. This may require a new submodel to study the dynamics of evaporation and heating of the steam. Adaptivity in simulations and scenarios (not just parameter spaces), is necessary to deal with emergent conditions and evolve systems in a flexible manner. Uncertainty and change is more pervasive in social phenomena (i.e., social conflicts). The trajectory of a realistic conflict scenario is never fixed due to changing attitudes, emotions, and motives between adversarial parties. Multi-stage games [Bennett 1987], for instance, characterize conditions where preference of actors change during a conflict, resulting in (simulation) games with new options and strategies to choose from. Extension and generalization of the multimodel formalism, which is originally formulated in [Ören 1987], can create new vistas to explore and study such multi-stage games. Symbiotic simulation is yet another development area of concern raised in a recent conference [Fujimoto et al. 2002] as a grand challenge with relevance to various application domains. In symbiotic simulation information about the actual performance and accuracy characteristics of the system are acquired during actual simulation rather than before. The immediate factor raised by these challenges is that there is a need for run-time switching of models based on interpretation of emergent, potentially unforeseen conditions to facilitate dynamic run-time simulation composition.

This paper explores multimodel formalism and suggests potential strategies for their simulation. To this end, in section 3 we introduce a taxonomy of multimodel types. Requirements for dynamic model updating in multimodel simulation are also discussed in section 3. Section 4 presents the rationale for an agent-based approach to multimodel simulation. In section 5 we conclude by discussing potential avenues of future research.

2.0 Background on Multimodel Formalism

A multimodel is a modular model that subsumes multiple submodels that together represent the behavior of the model. Multimodel formalism was originally introduced by [Ören 1987, 1991] to facilitate generalization of discontinuity in piecewise continuous systems. The multimodeling concept influenced the development of combined simulation, which entails the integration of continuous and discrete simulations within the same system description. For instance, as a special case of multimodel, coupled multiformalism specification developed by Praehofer [1992] extended DEVS formalism to provide a simulation environment for combined continuous/discrete-event simulation. Fishwick and Zeigler [1992] applied a special case of the multimodel to simulate qualitative dynamics of a physical system. Yet, in each one of these multimodel formalizations, submodels share the same address space, and they are updated based on switch statements; hence, existing formalisms are not only inflexible (i.e., hardwired), but also do not operate on truly distinct arbitrary models.

3.0 Issues in the Specification and Simulation of Multimodels

To systematically analyze possible multimodel design space, we present a preliminary ontology of multimodels and elaborate on the requirements for dynamic model updating to realize multimodel formalism.

3.1 An Ontology of Multimodels

Figure 1 illustrates a taxonomy of possible multimodel types based on various plausible constraints imposed on the submodel structure and activation policies. Based on the submodel structure of a multimodel the taxonomy considers the number of submodels active at a given time and the variability of the structure as the criteria. Conventional multimodels [Ören 1987, 2001], where only one model is active at a time can be characterized as single aspect (sequential) multimodels. In both cases, the states of the dormant models need to be saved and resumed once they are activated. Yet, re-instantiation of these latent models with a new up-to-date state information is needed to facilitate continuity in the overall model behavior. Multiaspect models [Ören 2001] operate under conditions, where more than one submodel can co-exist simultaneously to represent distinct aspects of the same phenomena. Multimodels can have static or dynamic structures. Dynamic structure multimodels enable not only variation of the number of submodels, but also their alteration (i.e., evolution). Extensible multimodels enable inclusion of new submodels that are unforeseen at the design time, while alteration of existing submodels results in evolutionary behavior or mutation of submodels. The ability of a model to respond to and alter its own structure or behavior is plausible through reflective models. Computational reflection can be used to facilitate introspective access or operate on (alter) model's own computation through intercessory reflection [Maes 1987]. Different model update policies and mechanisms result in various design options for multimodels. The activation policy of a multimodel is based on the nature of information necessary for the activation of submodel(s) along with the location of this information. Depending on the nature of the activation information multimodels can be constraint-driven, pattern-directed, or goal-directed.

Constraint-driven multimodels are controlled by stationary [Fishwick and Zeigler 1992] or adaptive transition policies that can have learning capabilities. Various reinforcement learning strategies can be used to maximize utilities, payoffs, or rewards in constraint-driven multimodel simulation. By choosing those actions in a given state that maximize a scalar reinforcement or feedback received after each action, the policy can adapt its action award structure by using a Markov decision process. Pattern-directed multimodels, on the other hand, follow a fixed cyclic or acyclic pattern (i.e., sequence) of submodel activations. Model selection can also be based on goal-directed (teleological) activation. Such models with planning facilities enable exploring a state space with alternative submodels, the preconditions of which are consistent with the observed conditions. In this viewpoint submodels are simply operators in a state space search algorithm implemented by the planner that is interfaced with the multimodel. Pattern-directed activation can guide selection of known submodel(s). A simple case is metamorphic models where the number of submodels is finite and their sequence is fixed (i.e., egg, larva, pupa, butterfly metamorphosis requires 4 submodels). Based on the location of information necessary for the activation of a submodel, we have to consider two cases. Active multimodels initiate updates due to internal emergent conditions that satisfy the transition conditions embedded with their specification [Ören 1987]. In externally activated multimodels (i.e., passive multimodels), the decision to qualify and switch to a submodel is external to enable flexible and customizable update mechanisms or protocols [Fishwick and Zeigler 1992].

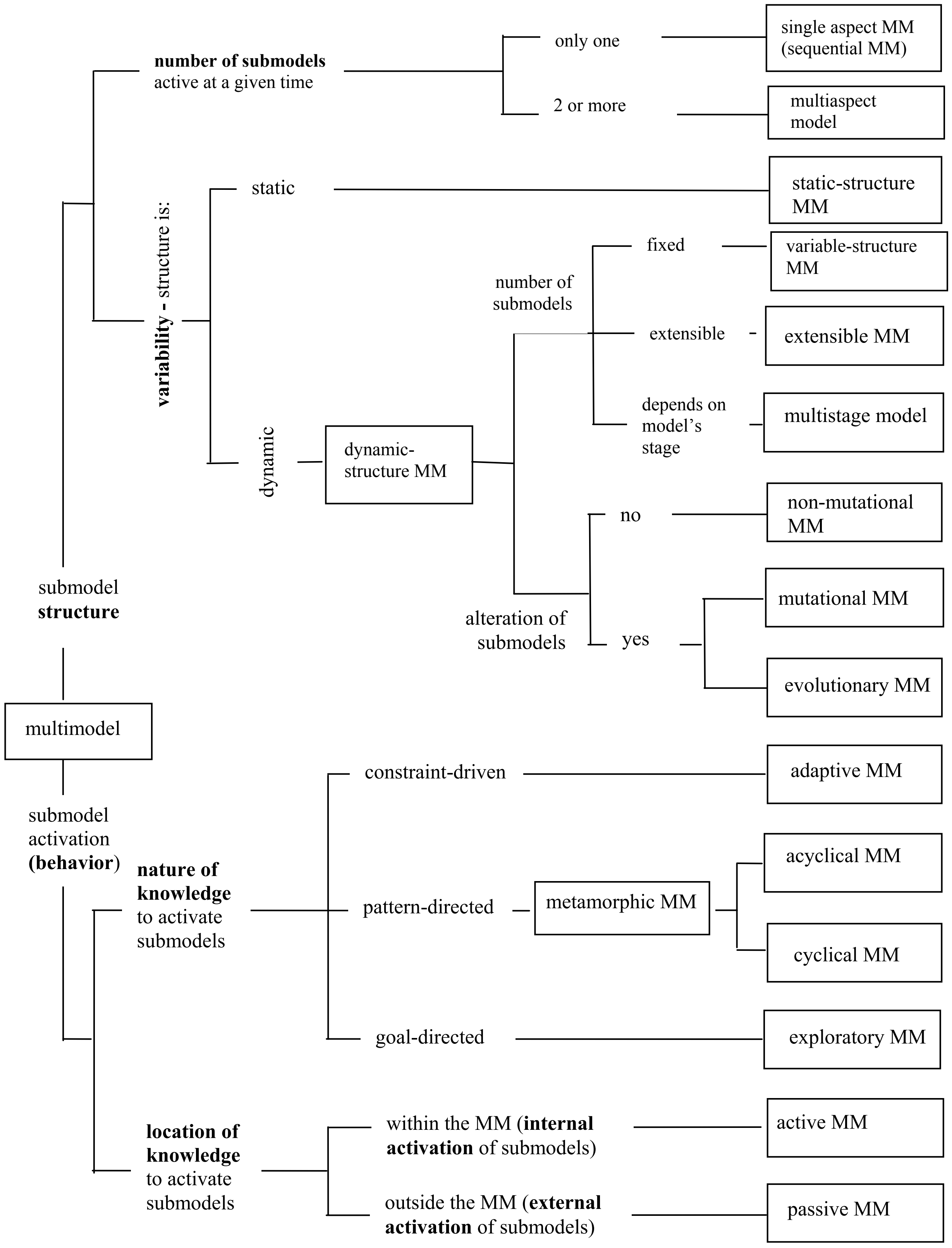

Figure 1. A Taxonomy of Multimodels (MMs)

3.2 Requirements for Dynamic Model Update

The challenges in dynamic model updating in multimodel formalism are the issues involved in substituting a new model 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 by a submodel or externally by a scheduler.
- **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 module replacement 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. Also, the intricate details of complex submodel construction process should be as independent of the multimodel as possible to enable flexible update.
- **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.

4.0 A Conceptual Basis for Agent-Based Multimodel Simulator

Given the requirements above we now outline the components of a generic agent-supported approach that can facilitate realization of static structure, single aspect multimodels with exploratory behavior.

4.1 Agent-Supported Simulation of Multimodels

The requirements of dynamic model replacement requires computational support to *observe* simulation state, *reason* to qualify models for update, *facilitate* model (re)binding, and *plan* for goal-directed activation by exploring potential paths within the state space of the problem domain. Each submodel can be viewed as an operator that transforms the state of the simulation. Agent paradigm provides the necessary computational infrastructure to attain these objectives. Agents are capable of observing, perceiving, and reasoning about their environment to act or proact with goal-driven responses [Ferber 1999]. The notion of mediator and facilitators are widely used as a semantic integration solution for disparate information systems. The underlying rationale for using facilitators is to bridge the gap between resources through a level of indirection. The same argument applies to model binding and rebinding, as well. That is, flexible update mechanisms need decoupling between a simulator and models that it operates on. The following sections suggest the requisite components of a generic architecture for multimodel simulators.

4.2 Dynamic Model Replacement with Facilitator Agents

A facilitator agent, as shown in Figure 2, is a decorator (wrapper) [Gamma et al. 1996] that decouples the simulator from submodels to enable run-time composition. In the envisaged approach the facilitator instantiates a model and sends back the simulator a *logical handle* for the model (i.e., M_I) as opposed to actual physical reference. This provides the facilitator with the necessary flexibility to update the model seamlessly without changing its interface. When the simulator (i.e., DEVS simulator instance) needs to perform an operation on the model object, the invocation is made on the facilitator, with the logical handle as the parameter. The facilitator uses the handle to delegate the operation to the model instance (i.e., DEVS model object) it encapsulates. The facilitator maintains a permanent level of indirection between the simulator and the model. The premise of the approach is that the facilitator acts as model facility that is parameterized to switch from one model to another by a parameter update.

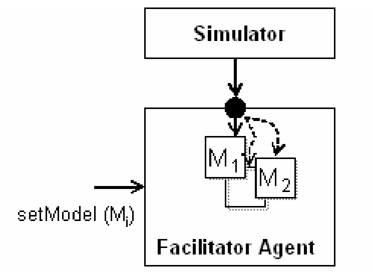

Figure 2: Facilitator Agent

A *setParameter* (i.e., setModel) operation on a facilitator object, as shown in Figure 2, can enable rebinding the handle with a new parameter. The parameterization of facilitators is reminiscent of the power of templates in modern modeling and simulation construction languages. Yet, the main difference is the runtime (re)binding of parameters to facilitate dynamic model updates.

4.3 Goal-Directed Exploration with Scheduler Agents

External activation of submodels requires facilities to guide the submodel selection process. While a predefined pattern for model switching is possible, goal-directed exploration may be needed in cases where uncertainty exists.

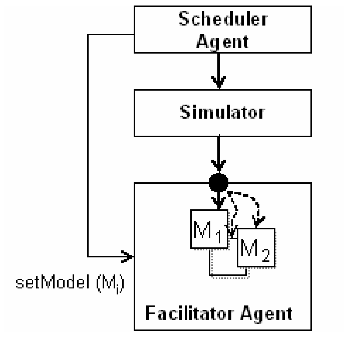

Figure 3: Scheduler Agent

A scheduler agent, as shown in Figure 3, can have planning capability that drives a simulator with alternative models by updating the parameterized model facility (i.e., facilitator). A scheduler agent extends the FSA-controlled multimodel [Fishwick and Zeigler 1992] approach by incorporating deliberative reasoning facilities [Ferber 1999] to explore the state space of the multimodel. A hypothetical planning process within a scheduler agent is shown in Figure 4. We assume a rule-based reasoning process, where the activation conditions of submodels are defined in terms of a production system.

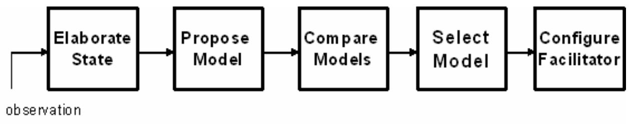

Figure 4: Submodel Qualification

The SOAR agent system [Rosenbloom et al. 1993] provides the rationale and significance of such an approach for deliberative reasoning. The input to the process refers to the observed condition (see section 4.4) used by the *elaborate state* component of the planner. The elaboration phase maps (or generalizes) the observed concrete simulator conditions to the abstract state of the multimodel so that planning can be performed in terms of abstract value (state space) manipulation.

Note that the qualifier conditions (precondition) of submodels are defined in terms of abstract predicates that facilitate submodel selection. During the model proposition phase all submodels, the preconditions of which are implied by the current observed state of the multimodel are identified. Formally, model qualification can be defined in terms of a specification matching function, *M*, defined as follows: $M\text{: } Spec \times Spec \rightarrow \{T, F\}$. A specification match is useful if it can ensure that the qualified submodel satisfies a query *Q* (i.e., output of the elaboration phase), if and only if, for any submodel *S* and *Q*, $M(A,Q) : \{Q_{pre}\} \Rightarrow \{S_{pre}\}$, which means that submodel *S* has a weaker condition compared to the query (observed state). A more strict condition can be defined by taking the postconditions (if available) of submodels into account: $M(A,Q) : [\{Q_{pre}\} \Rightarrow \{S_{pre}\}]$ and $[\{S_{post}\} \Rightarrow \{Q_{post}\}]$. An agent-based planning layer connected to a simulator would be capable of identifying, qualifying, and, if necessary, choosing an operator that represents a specific model based on the specified preferences and options. Furthermore, in the case of an impasse or lack of knowledge on preferences, a planning layer can guide exploring alternative contexts in some order. Plausible models can be maintained by *focus points*. A focus point manages branch points in the model scheduling stack. Suppose that an observed abstract state (i.e., transition condition) is at the top of the stack. If only a single model qualifies for exploration, then it is selected for the update operation. Yet, if more than one model matches the condition, a simulation focus point is generated to manage newly created simulation branching (discontinuity) points, each one of which would have its own contexts. When a path is exhausted, the closest focus point selects the next available model to instantiate the simulation frame or return to the context that generated the focus point. As simulation state space is explored, a network of focus points is generated. Determining which focus point should be active at any given time is the responsibility of the scheduler agent.

4.4 Observer Agents for Update Notifications

Dynamic submodel replacement requires monitoring simulation conditions to determine if any of the potential state variables or objects of interest are changed. State changes within the simulator objects of interest could be an indicator of a scenario or phase change in the simulation; hence, the need for dynamic model update and replacement may arise. While polling the simulation state is possible to detect changes, continuous scanning of the simulation state space would incur an unnecessary cost that will degrade the efficiency of the simulation engine.

Furthermore, low and high granularity polling levels can cause inaccuracy and performance problems, respectively. Low polling rate may result in missed significant changes, whereas high polling rate could potentially result in unnecessary processing. Using observer agents, as shown in Figure 5, with a subscription and notification facility could enable more efficient monitoring of a simulation as long as simulator provides a notification interface.

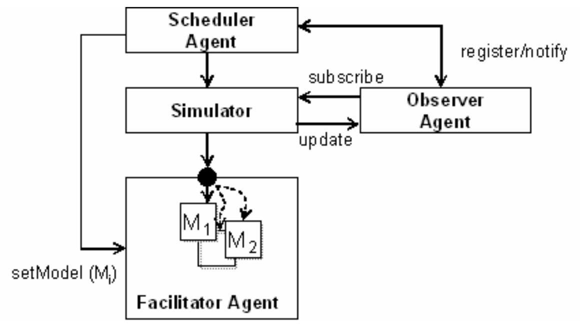

Figure 5: Monitoring with Observer Agents

4.5 Momento Agents for State Restoration

A momento agent captures and externalizes a model's internal state, so that it can be restored to its state later. The challenge is to avoid violation of the model encapsulation. In dynamic model replacement it is critical for the scheduler to assure the integrity of the simulation during updates. We suggest the association of a momento agent with each model. The scheduler agent requests from the simulator to access the momento agent of the present model to save its state. The state of a model can simply be the values of its variables, including its current phase. The momento agent can simply be a proxy [Gamma et al. 1996] for the model so that it can present the interface of the model, along with other facilities such as state externalization. Next time the model is instantiated, the simulator can set its state by simply invoking a *setState* message on the momento.

5.0 Conclusions

Multimodeling formalism influenced the development of several methodologies, including combined simulation [Preahofer 1992], FSA-controlled multimodeling [Fishwick and Zeigler 1992], and MOOSE [Cupert and Fishwick 1997]. Yet, we are still scratching the surface of what is possible with the very basic dynamic model update concept. The taxonomy presented in this paper is an indicator for the potential for various types of multimodels, each one of which can be useful for variety of problem domains, such as multi-stage conflict analysis. The generic agent-based approach described in section 4 presents a reasonable strategy to realize multimodel simulators. Yet, more research is needed to identify intricate details of *observing* the simulation state, *reason* to qualify models for update, *facilitate* run-time model rebinding, and *plan* for goal-directed activation of submodels.

References

Bennett G. P. (1987). *Analyzing Conflict and Its Resolution*: *Some Mathematical Contributions*. Clarendon Press, Oxford.

Cubert R. M. and P. A. Fishwick (1997). "Moose: An Object-Oriented Multimodeling and Simulation Application Framework," *Simulation* vol 70, no. 6, pp. 379-395.

Ferber J. (1999). *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Harlow, UK: Addison Wesley.

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

Fujimoto R., D. Lunceford, E. Page, A. Uhrmacher (2002). *Technical Report of the Dagstuhl-Seminar Grand Challenges for Modelling and Simulation*.

Gamma E., R. Helm, R. Johnson, J. Vlissides. (1996). *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley.

Maes P. (1987). "Concepts and Experiments in Computational Reflection," *ACM SIGPLAN Notices.* vol . 22 no. 2, pp. 147-155.

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.

Ören T.I. (2001). Towards a Modelling Formalism for Conflict Management. In: Discrete Event Modeling and Simulation: A Tapestry of Systems and AI-based Theories and Methodologies. H.S. Sarjoughian and F.E. Cellier (eds.), Springer-Verlag, New York, pp. 93-106.

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

Rosenbloom P. S., J. E. Laird, and A Newell (1993). *The Soar Papers: Research on Integrated Intelligence*. Cambridge, MA: MIT Press.