Proceedings of the 2003 Winter Simulation Conference
S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds.

A META-THEORETIC APPROACH TO MODELING AND SIMULATION

Mamadou K. Traoré

LIMOS CNRS UMR 6158
Université Blaise Pascal
Campus des Cézeaux
63177 Aubière Cedex, FRANCE

ABSTRACT

We aim at building a methodological framework that integrates various methods and key concepts in a more coherent Modeling and Simulation architecture. The required flexibility for such a framework can be achieved by modeling the modeling process itself. The essence of this process lies in refining successive abstraction levels, each level into a lower one. The traversal of these abstraction levels, from the highest level (the more abstract) to the lowest one (the more detailed) involves two aspects: (1) System knowledge reside at different levels of a specification hierarchy; (2) many formalisms are often required for knowledge specification. As multi-formalism modeling provides a powerful means to deal with many formalisms, we show that modeling in addition the specification hierarchy provides the means to support many modeling processes. Both means are combined in a common meta-theoretic approach to enhance flexibility of the integrative framework.

1 INTRODUCTION

As an inter-disciplinary complex problem solving paradigm, Modeling and Simulation (M&S) addresses various applications, leading to diverse methods and underlying concepts. Consequently, it is desirable to integrate these methods and concepts in a coherent methodological framework. The M&S architecture in which our framework is growing is depicted in Figure 1. It attempts to unify major M&S issues, merging the following key approaches together and with various experiences:

- The M&S life cycle (Nance 1994).
- The synthesis of the relationships between a real system, its conceptual representation and its software simulator (Zeigler 1976).
- The analysis of M&S models design and execution (Fishwick 1995).

The identification phase, which initiates the M&S life cycle, starts with the announcement of a problem. The problem formulation is the process to identify both the real

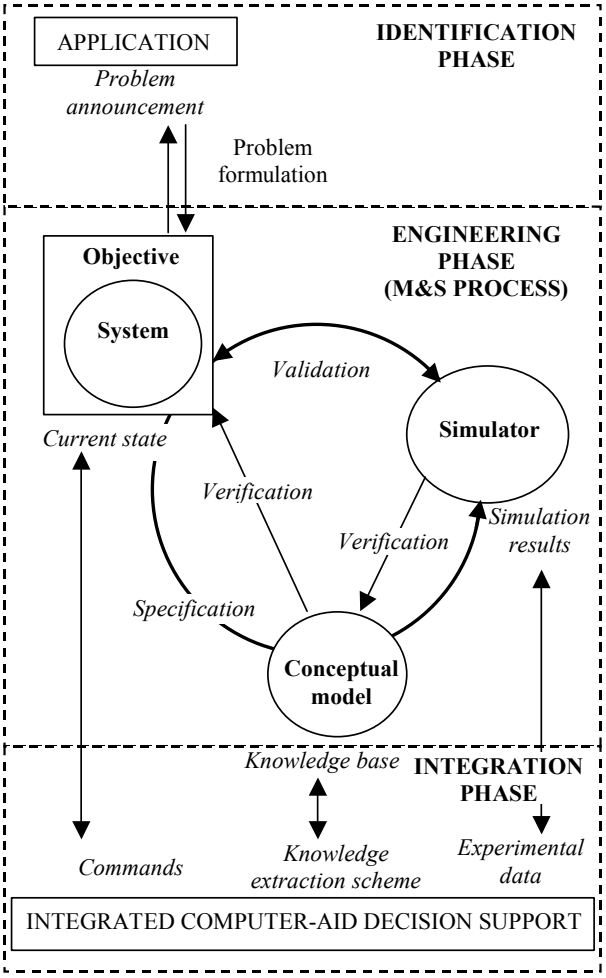

Figure 1: A Unified Architecture for the M&S Life Cycle

world system (entities and their relationship, environmental constraints, ...) and the objectives of the M&S study (e.g. criteria evaluation, performance optimization, future prediction, ...). This iterative (and sometimes multi-threaded) phase, which is often driven by exchanges between model designers and application domain experts,

leads to both quantitative knowledge (knowledge on numerical data gained from physical measures, observation, or experts), and qualitative knowledge (knowledge on the system structure and behavior). The concept of experimental frame refers to the limited set of circumstances under which these knowledge are relevant (Zeigler 1976).

The engineering phase, also known as the M&S process, is an iterative three steps model building process, which is based on the identified knowledge. From the system and objectives, a valid conceptual representation must be designed, and then translated into a valid executable program (i.e., the simulator). The means to accredit both conceptual and executable models within the context of the experimental frame are labeled VV&A, for Verification, Validation & Accreditation (Balci 1997, Robinson 1999). Verification is the process of checking the consistency of an abstraction level with respect to a previous one it is derived from, while validation is the process of comparing simulation results with real world experiments (at this stage, imperfect or missing knowledge could be adjusted).

The integration phase embeds the simulation system in a decision-making support (man or a computer-based monitoring environment). In a problem solving perspective, experiments are planned and driven by simulation, and the results are used to solve the problem, and eventually other problems of the same class; quantitative simulation provides a descriptive tool which is dissociated from and used by the problem solving knowledge (most of the time, this knowledge is built upon Operations Research or Artificial Intelligence tools (Ören 1989), while qualitative simulation embeds the problem solving knowledge (Wild and Pignatiello Jr 1994). In a knowledge base perspective, information can be extracted from the conceptual model to provide other integrated solvers with data for decision making. In a monitoring perspective, a monitoring environment (either in a real-time or an off-line context) gets the current state of the real world system, initiates simulation experiments, analyzes their results, and infers corrective commands to send to the system. In the case the real system has to be built or modified, these commands are made of design decisions.

The scope of our emerging methodological framework covers the M&S process. Section 2 recalls that the essence of this process is the traversal of abstraction levels, which is focused here (experimental frame, verification and validation issues are not covered by the present work). Various discipline-oriented methods and underlying concepts are often used to perform this traversal. A meta-theoretic approach is presented in section 3 to deal with them in a generic way. It combines both multi-formalism modeling and modeling of specification hierarchy. A conclusion is given in section 4, with some prospects on the framework supporting software platform.

2 METHODOLOGICAL FRAMEWORK FOR THE MODELING AND SIMULATION PROCESS

The key issue in simulation model engineering is how to put into correspondence a given system, its conceptual representation (in the context of the focused objectives) and a valid simulator. In other words, how does one start with a basic representation of the problem announced and refine the successive abstraction levels that depicts the solution at different levels of understanding? Various empirical methods are proposed in the literature to perform this traversal of abstraction levels. Some of them are very application-oriented. Some other adapt general methods and tools to simulation (e.g. UML class diagrams are widely used in object-oriented simulation studies).

To support diverse methods and underlying concepts, we aim at building the methodological framework presented in Figure 2. It relies on three major concepts:

- Orthogonal design, implementation and experimentation.
- Hierarchical specification.
- Multi-formalism.

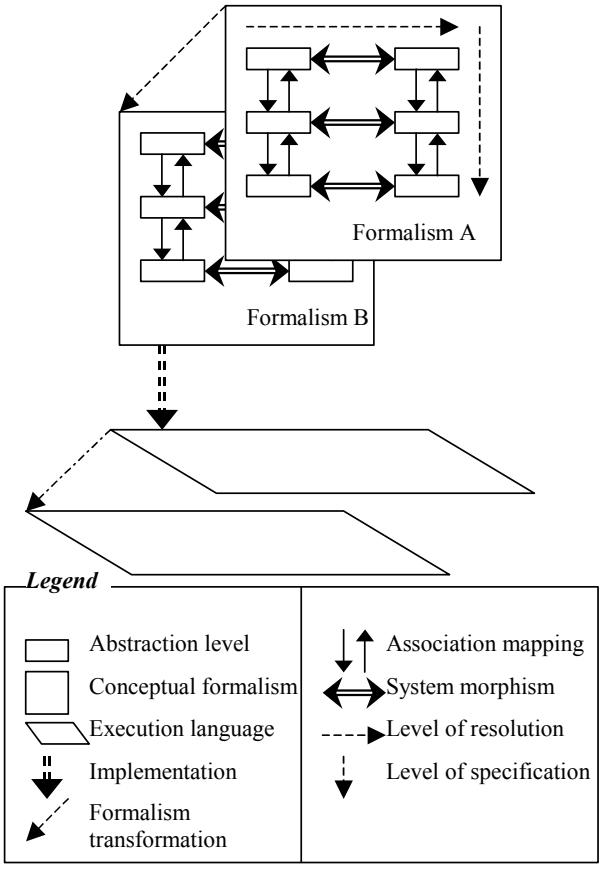

Figure 2: Methodological Framework for M&S Process

2.1 Orthogonal Design, Experimentation and Implementation

As mentioned previously, the traversal of abstraction levels is incremental in nature, and successive refinements are necessary to translate the initial problem into an executive solution. Along this process, it is admitted that the representation of a modeler's conceptual understanding should be separated from its translation into an executive simulation program, so as to avoid that execution details clutter the problem understanding (the so-called Dijkstra's principle of separation of concerns). This is why the framework shown in Figure 2 exhibits two orthogonal dimensions (each of them is composed of parallel sheets that represent different formalisms): one dimension for design (the upper part of the framework), and the other for implementation. The framework involves a third dimension, which is orthogonal to both the design and implementation dimensions, and which relates to model experimentation (for more clarity, this dimension is not shown in Figure 2; however, one can mention that experimental frames are formalized in this dimension, and that verification and validation issues are tackled through this formalization).

2.2 Hierarchical Specification

Systems theory distinguishes between system structure and behavior. Knowledge about these structure and behavior cannot be entirely specified at the same level. Indeed, various perspectives should be considered when specifying a system, among which are the major ones:

- Modularity, which key principles are decomposition (the system is decomposed into a set of components, which in turn, can be decomposable), and aggregation (existing components are aggregated into larger components, which in turn, can be aggregated).
- Functional behavior, which depicts the outer manifestation of the system.
- Internal connectivity, which describes the relationships inside the system.

A rigorous modeling method consists of a hierarchy of specification levels, each level being mapped into a neighbor level, and each level revealing more knowledge on system structure and behavior than the preceding one. The design dimension shown in Figure 2 exhibits an example of specification hierarchy, which three levels are mapped one into the following. A mapping describes the conditions under which one can move from a level to another one. Notice that a system can be specified at two different levels of resolution (i.e., one model of the system is more detailed than the other); in that case, there should be a system morphism that establishes a correspondence between these models at each level of the specification hierarchy (Zeigler et al. 2000).

A specification hierarchy is also required at the implementation level, even we do not show it in Figure 2 (for clarity purposes). A specification hierarchy that must be formalized at the implementation level is the one composed of the three so-called world views (the lower specification level addressing the event-oriented view, the middle level addressing the activity-oriented view, and the upper level addressing the process-oriented view).

2.3 Multi-Formalism

Various formalisms are often necessary during the traversal of abstraction levels. Specifying the entire knowledge, a single formalism can rarely fits the needs. Multi-formalism (i.e., the use of many formalism in the same M&S process) has pros and cons:

- At the design level, multi-formalism is a powerful means to capture all the aspects of a system. A multi-formalism model combines declarative, functional, constraint and spatial aspects (Fishwick 1995). Inversely, it is very hard to catch the overall semantics of a multi-formalism model. Also, VV&A processes are easier to process with a single-formalism model. Some formalisms (DEVS, Bond Graphs…) are expressive enough to subsume many other ones; then, they can be used to convert multi-formalism models into single-formalism models (Vangheluwe 2000).
- At the implementation level, multi-formalism refers to co-simulation, i.e. each sub-model of the overall model is simulated with a formalism-specific simulator (as already mentioned, each sheet of the framework presented in Figure 2 represents a formalism). This favors the reusing of existing simulators, while requiring at the same time the building of an often hard-to-achieve interoperability platform.

The idea behind our framework is the following: (1) allow a modeler to design his conceptual model, using multi-formalism (or a single formalism, in simple cases); (2) produce a semantic specification that is world-view independent, amenable to automated diagnosis and automatically transformable to an executable version; (3) allow the integration of existing simulators.

Multi-formalism modeling, also called formalism meta-modeling, has proven to be a powerful means to support many formalisms (Vangheluwe and De Lara 2002). It establishes a correspondence between data structures so that a model expressed in a given formalism can be translated into another formalism, using the data structures transformation rules. This movement (formalism transformation) is orthogonal and complementary to the movement through the specification hierarchy. Both movements constitute the traversal of abstraction levels. Hence, modeling the specification hierarchy, in addition to modeling the

formalisms, could provide the means to integrate many modeling processes in the same framework.

3 META-THEORETIC APPROACHES TO M&S

Here, we deal with meta-modeling. This concept covers diverse understanding in the literature, like the followings:

- A meta-model plays a role of a model for another model (Merkuryeva and Merkuryev 1999).
- Meta-models are approximations of simulation models (Dos Santos and Porta Nova 1999).
- Meta-modeling concerns the description of classes of models (Vangheluwe et al. 2002).
- A meta-model is a relatively small, simple model that approximates the "behavior" of a large, complex model (Davis 2003).

In our sense, meta-modeling is the process of building models of models. A specification hierarchy is a static representation of a modeling process; then a model of this hierarchy is a specification meta-model (as opposed to formalism meta-model). The traversal of abstraction levels requires both movements through specification hierarchy and formalisms space; hence our framework combines both the formalism and the specification meta-models.

3.1 Formalism vs. Specification Meta-Modeling

Table 1 compares the formalism and the specification meta-modeling. For the latter, we use UML instead of the Entity-Relationship (ER) formalism, used in (Vangheluwe and De Lara 2002) for formalism meta-modeling. Both formalisms, extended with the Object Constraint Language, are convenient for meta-modeling, but the use of UML is more coherent with the object-oriented software we intend to build as the framework support.

Table 1: Formalism vs. Specification Meta-Modeling

	Formalism	***Specification***
Meta-meta model	ER	UML
Meta-model	ER specification of FSA	UML class diagram of specification hierarchy
Model	FSA model	DEVS specification hierarchy
Execution	Simulation	Traversal of abstraction levels

A formalism meta-model is a model of formalism, e.g. an ER specification of FSA (Figure 3.a), while a specification meta-model is a model of specification hierarchy (Figure 3.b; the ER formalism is used here, only for the purpose of comparison). A specification meta-model can generate a specification hierarchy, as the one established in the DEVS framework, in the same way that an FSA meta-model can generate dynamic models. Figures 4.a and 4.b illustrate such generated models (in Figure 4.b, AMij denotes the association mapping from level i to level j, while JCij denotes the justifying conditions to climb up the hierarchy from level j to level i; an overview of the DEVS specification hierarchy is given in Appendix B). Also, similar to the principle of executing a graph grammar on a particular FSA model to produce a simulation trace, executing an adequate graph grammar on a specification hierarchy will result in a traversal of abstraction levels. This grammar describes a set of clauses related to the mapping conditions.

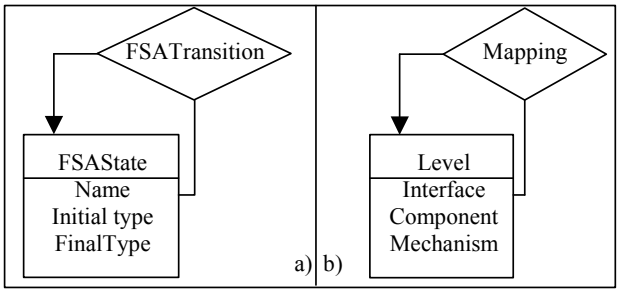

Figure 3: Meta-Models (Examples)

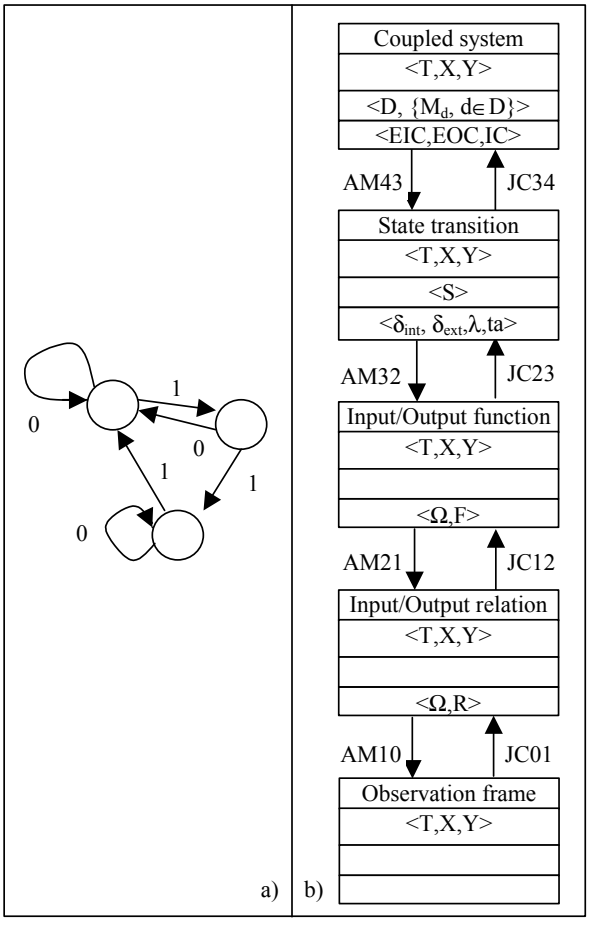

Figure 4: Models (Examples)

3.2 Modeling The Specification Hierarchy

Our specification meta-model is presented in Figure 5, using UML. A detail is given below:

- An abstraction level is composed of three aspects: the interface, mechanism, and composition abstractions.
- The interface abstraction defines the model's boundaries and the way the model can be used (e.g. objects, agents, modules).
- The mechanism abstraction describes the behavior of the model. It defines by either a set of operating algorithms, or a set of connecting rules (for the components of the model).
- The composition abstraction describes the abstract set of model's components, which can be simple state variables, or models in their turns.
- Each aspect of an abstraction level has attributes, i.e. a set of items that a modeler can define, add to or retrieve from the aspect. These items belong to the specification formalism that the modeler can also define, using formalism meta-modeling.
- A mapping establishes a correspondence between two abstraction levels. A mapping abstraction is composed of a set of clauses, each of which being composed of a condition and an action (to perform when the condition is satisfied).

A graph grammar is used to map a current abstraction level to a new one. Before showing the operational semantics of such grammar, we first propose (in the next section) a formalization of the specification meta-model.

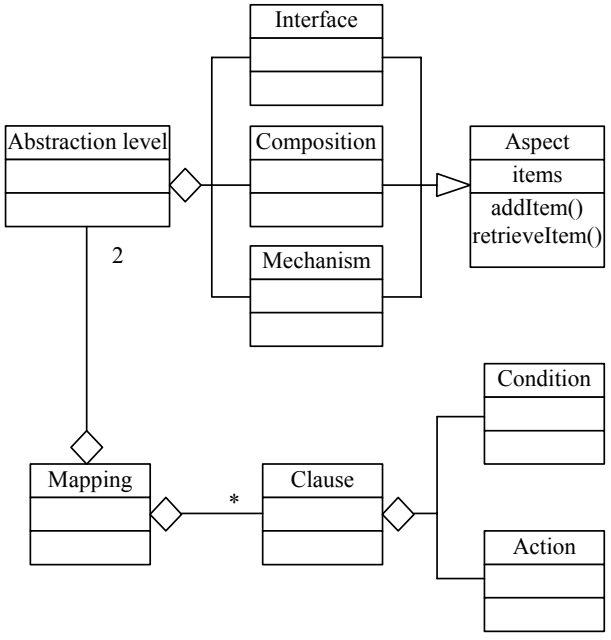

Figure 5: Specification Meta-Model (UML)

3.3 Formalization of the Specification Meta-Model

The specification meta-model can be used to produce a specification hierarchy in the design dimension shown in Figure 2, as well as in the implementation dimension. Notice that in this latter case, the interface abstraction relates to the partitioning of the simulation program, the mechanism abstraction relates to the simulation synchronization mechanisms, and the composition abstraction relates to the programming variables. One way to ease the manipulation of this specification meta-model is to formalize it in a clear abstract form. Both, abstraction level and mapping abstraction, are formalized as explained hereafter.

We define an abstraction level as a structure:

$$L = <I, C, M>$$

- I is the interface abstraction. It is a set of parameters (or inputs) that can be transmitted to the model (in the form of streams of commands, conditions or data), and results (or outputs) that are generated by the model (again in the form of stream of commands, conditions or data). An example of interface definition is the set $\{(\phi_k,\Omega_k,\rho_k)_{k=1,2,...}\}$ where ϕ_k is a stream, Ω_k the set of possible values that can be transmitted through this stream, and ρ_k the polarity of the stream (e.g. 1 for inputs, and –1 for outputs).
- C is the Composition abstraction. It is a set of models (notice that in an etymological sense, even a simple variable is a model).
- M is the mechanism abstraction. It is a set of rules that govern the model (how model inputs are transformed into its outputs and how model components interacts). All rules are described by clauses.

To enhance the understanding of this formalization, let us give hereafter three examples of abstraction levels:

- A simple variable x which can take its values in set E, can be defined as follows:

$$x = <\{(m,E,1), (r,E,-1)\}, \{x\}, \{R_m,R_r\}>$$

  - where m and r stands respectively for "memorization" and "restitution",
  - Rm is the rule to put the value in stream m to x, and Rr is the rule to put the value of x in stream r.
- An Object O, with an attribute a and a method set that operates on parameters in E, a method get that returns values in F, and a method run that operates on parameters in G and returns values in H, can be defined as follows:

$$O = <\{(s,E,1), (g,F,-1), (r,G,1), (r,H,-1)\}, a, S>$$

- where s, g and r stands respectively for "set", "get" and "run",
- S is $\{R_{set},R_{get},R_{run}\}\cup\{f_k\}_{k=1,2,...}$
- R_{set} is a rule to pick values from stream s and to update attribute a,
- R_{get} is a rule to translate the value of attribute a into the format of stream g,
- R_{run} is a rule to get values from stream r, compute new values and translate them into r's format,
- f_k is the k^{th} internal auxiliary rule which is useful for the other rules.

- A multi-agents system N, if viewed as a network of agents, can be specified as follows:

$$N = <\varnothing,\{A_1, A_2, A_3, ...\},\{R_k\}_{k=1,2,...}>$$

- where A1, A2, A3, ...are the agents of the network,
- Rk is the kth agents communication rule.

We define a mapping abstraction as a structure:

$$\nabla = < L_i, L_f, A, \{(c_i, a_i), i\in A\}>$$

- L_i and L_f are the initial and final levels,
- A is the set of clause names,
- c_i is a condition, and a_i is an action, for each $i\in A$.

A graph grammar is used to perform the mapping of the initial level into the final one (which is then created), by applying the actions of the mapping abstraction when their associated conditions are satisfied:

$L_i = <I_i, C_i, M_i> \text{———} \nabla \text{———}\rightarrow L_f = <I_f, C_f, M_f>$
c_1 is satisfied $\Rightarrow$ perform a_1 on I_f, C_f, and M_f
c_2 is satisfied $\Rightarrow$ perform a_2 on I_f, C_f, and M_f
...

3.4 Graph Grammars to the Traversal of Abstraction Levels

Here, we describe the operational semantics of a graph grammar associated to the traversal of abstraction levels in the DEVS specification hierarchy. We limits the discussion to the hierarchy climbing from level 1 to level 2. Readers unfamiliar to DEVS can find a summary of the formalism in Appendix A and an overview of the specification hierarchy in Appendix B (as already mentioned).

The justifying conditions established in (Zeigler 1976) for inferring knowledge at the level of "Input/Output Function Observation" from knowledge at the "Input/Output Relation Observation" (respectively level 2 and level 1) can be formalized in the following way: given all the pairs $(\omega_i,\rho_i) \in R$, the modeler has to provide a set $\{(a_j,b_j) / \forall j, \exists!\ i$ such that the input of ω_i when the system is in state a_j, produces the output ρ_i and let the system in state $b_j\}$. In other words, we can construct a graph, which nodes are the identified states of the system, and we link any node a_j to its corresponding node b_j with the corresponding edge (ω_i,ρ_i). The justifying conditions impose that:

- Every state of the graph must be reachable.
- Conflicting edges are not allowed, i.e. edges with the same ω and ρ, and starting from the same state and ending at different states (as shown in Figure 6 with doted arrows).

The graph grammar operates according to the principle illustrated in Figure 6. Given the left-hand-side (LHS) abstraction level (level 2), the modeler specifies a set of 4-uples that correspond to the observed pairs of initial and final states, associated to their causal Input/Output pairs. Conditions of the clauses are evaluated and their corresponding actions are performed. A state is said to be identified if it appears, once at least, as the third member of a 4-uple. The specific Λ node of the of the right-hand-side (RHS) abstraction level indicates identified states. The graph grammar produces the RHS abstraction level as the result of the mapping operation. The graph that appears there obviously defines the set of Input/Output function of the corresponding abstraction level.

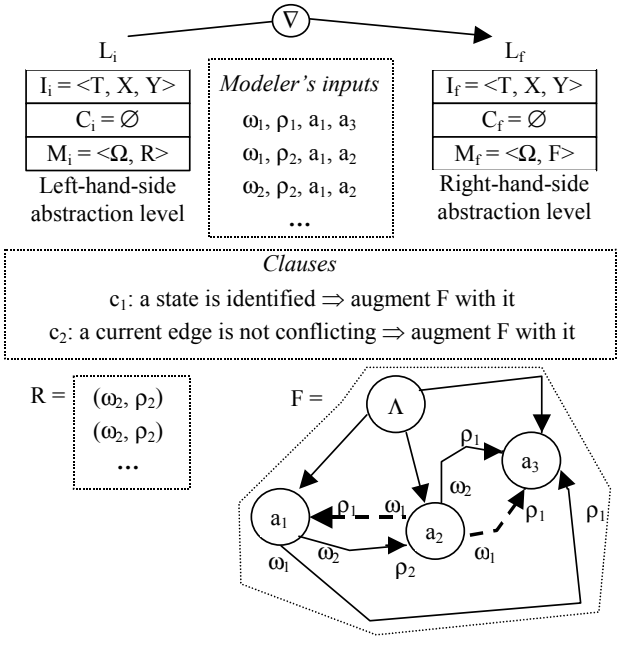

Figure 6: Operational Scheme for the Graph Grammar

4 CONCLUSION

We have presented a meta-theoretic approach to M&S, based on the modeling of the modeling process itself. It combines the known multi-formalism modeling approach to the modeling of the specification hierarchy (which is the

essence of the M&S process). Such a combination can achieve the required flexibility for the methodological framework that we have define to integrate many methods and their underlying concepts. This approach is being implemented in an object-oriented software platform developed in Java, with the following requirements:

- A *Design* layer (executable either in a web browser or in stand-alone), which supports graphical multi-formalism model design, execution, documentation and reuse.
- An *Execution* layer, which provides the implemented simulation libraries.
- A *Network* layer, which ensures simulators interoperability.

The approach does not escape from the following limitations that are inherent in using graph grammars:

- In the most general case, verifying the satisfaction of mapping conditions between two abstraction levels is a problem of sub-graph isomorphism, which is known to be NP-complete.
- In a practical point of view, it can be hard to express the mapping circumstances by clauses.

APPENDIX A: THE DEVS FORMALISM

Figure 7 summarizes a part of the evolution of the DEVS formalism. The initial DEVS model (that we name here GDM) has been introduced as:

$$GDM = \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

- S is a set of states, and the model is supposed to be at any time in some state $s \in S$.
- X is the set of input values (external events).
- Y is the set of output values (computed by the model).
- $e = ta(s)$ is the time elapsed since last change in the model. When it expires before any external event occurs, the system outputs the value $y = \lambda(s)$, and changes to state $\delta_{int}(s)$.
- $\delta_{int} : S \rightarrow S$ defines the internal transition function.
- $\lambda: S \rightarrow Y$ is the output function. An output is possible only before an internal transition.
- $ta : S \rightarrow R_{0,\infty}$ defines the time advance function (positive real values, including 0 and ∞).
- $\delta_{ext} : Q \times X \rightarrow S$ is the external transition function. If an external event $x \in X$ occurs before the expiration time, the system changes to state $\delta_{ext}(s,e,x)$.
- $Q = \{ (s,e) / s \in S, 0 \leq e \leq ta(s) \}$ is the total state set.

To make modeling easier and to allow the building of hierarchical coupled models, input and output ports have been introduced for receiving and sending messages.

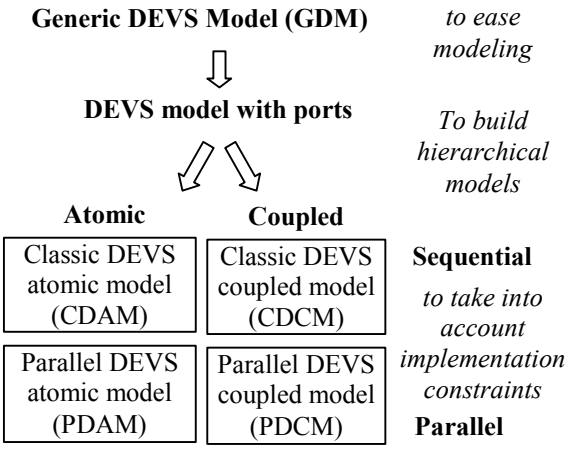

Figure 7: Evolution of the DEVS Formalism

Moreover, the following constraints have been established: (1) since two ports of an atomic model cannot simultaneously receive or send messages, input events (as well as output events) must be serialized. (2) In a coupled model, no direct feedback loop is allowed, i.e. no output port of a component may be connected to one of its input port. Atomic model with ports (CDAM) and coupled model with ports (CDCM) are respectively defined as:

$$CDAM = \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$
$$CDCM = \langle X,Y,D,\{M_d, d \in D\},EIC,EOC,IC, Select \rangle$$

- $X = \{ (p, v) / p \in InPorts, v \in X_p \}$ is the set of input ports and values,
- $Y = \{ (p, v) / p \in OutPorts, v \in Y_p \}$ is the set of output ports and values.
- D is the set of the component names,
- M_d is a DEVS model, $\forall d \in D$,
- EIC (External Input Coupling) connect external inputs to component inputs,
- $EIC \subseteq \{((N,ip_N),(d,ip_d)) / ip_N \in InPorts, d \in D, ip_d \in InPorts_d\}$
- EOC (External Output Coupling) connect component outputs to external outputs,
- $EOC \subseteq \{((d,op_d),(N,op_N)) / op_N \in OutPorts, d \in D, op_d \in OutPorts_d\}$
- IC (Internal Coupling) connect component outputs to component inputs,
- $IC \subseteq \{((a,ip_a),(b,ip_b)) / a \in D, b \in D, ip_a \in OutPorts_a, ip_b \in InPorts_b\}$
- Select : $2^D - \{\} \rightarrow D$ is the tie-breaking function that serializes the actions of concurrent imminent components.

A revision to these preceding models was introduced later to fit to the hardware and software evolution from sequential execution to parallel one. The former sequential DEVS models are labeled as being part of Classic DEVS,

while the coming parallel versions are labeled Parallel DEVS. Parallel DEVS models lead with bags of inputs and outputs instead of single ones. A confluent transition is used to define the next state of the model in case of collision between external and internal events. The tie-breaking function becomes obsolete (all imminent components generating their outputs which are distributed to their destination according to the coupling information). The parallel models (atomic: PDAM, coupled: PDCM) are defined as:

$$PDAM = \langle X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta \rangle$$
$$PDCM = \langle X, Y, D, \{M_d / d \in D\}, EIC, EOC, IC \rangle$$

- $\delta_{ext} : Q \times X^a \rightarrow S$ is the external transition function, where a = Card (InPorts),
- $\lambda : S \rightarrow Y^b$ is the output function, where
- b = Card(OutPorts),
- $\delta_{con} : Q \times X^a \rightarrow S$ is the confluent function.

APPENDIX B: THE DEVS SPECIFICATION HIERARCHY

The DEVS specification hierarchy consists of five increasing detailed levels (from level 0 to level 4):

- The Input/Output Observation frame (level 0) describes the system as a black box, and is defined as $IO = \langle T, X, Y \rangle$, where T is the time base, X is the input values set, and Y is the output values set.
- The Input/Output Relation Observation (level 1) describes the system behavior by a set of I/O pairs: $IORO = \langle T, X, \Omega, Y, R \rangle$, where T, X, and Y are the same as for IO, $\Omega \subseteq (X,T)$ is the set of allowable input segments, $R \subseteq \Omega \times (Y,T)$ is the IO relation, and $(\omega,\rho) \in R \Rightarrow dom(\omega) = dom(\rho)$.
- The Input/Output Function Observation (level 2) describes the set of functions that partitions the I/O relation set: $IOFO = \langle T, X, \Omega, Y, F \rangle$, where T, X, Ω and Y are the same as for IORO, $f \in F \Rightarrow f = \Omega \times (Y, T)$ is a function, and $\rho = f(\omega) \Rightarrow dom(\rho) = dom(\omega)$.
- The Input/Output System (level 3) describes the state set and the state transition functions of the system: $IOS = \langle T, X, \Omega, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$ where T, X, Ω and Y are the same as for IOFO, and S, δ_{int}, δ_{ext}, λ, ta are described in Appendix A.
- The Network System (level 4) describes the system as a set of interconnected components: $NS = \langle T,X,\Omega,Y,D,\{M_d, d \in D\},EIC,EOC,IC \rangle$ where T, X, Ω and Y are the same as for IOS, and D, M_d, EIC, EOC, and IC are described in Appendix A.

Mapping specification levels suggest two issues: (1) going from structure to behavior, i.e. converting a description at a level to a description at a lower level, and (2) inferring structure from behavior, i.e. climbing up the hierarchy. While the downward mapping is straightforward, the upward one is less so, and is possible only under special circumstances called *justifying conditions* (Zeigler 1976).

REFERENCES

Balci, O. 1997. Principles of Simulation Model Validation, Verification, and Testing. *Transactions of the SCS* 13 (1), 3-12.

Davis, P., and J. Bigelow. 2003. *Motivated Metmodels. Synthesis of Cause-Effect Reasoning and Statistical Metamodeling*. RAND Corporation, US.

Dos Santos, I. R. and A. M. O. Porta Nova. 1999. The Main Issues in Nonlinear Simulation Metamodel Estimation. In *Proceedings of the 1999 Winter Simulation Conference*, eds. Farrington, Nembhard, Sturrock, and Ewans, 502-509. Phoenix, AZ.

Fishwick, P. 1995. *Simulation Model Design and Execution. Building Digital Worlds*. Prentice Hall Inc., Englewood Cliffs, NJ.

Merkuryeva, G. and Y. Merkuryev. 1999. Metamodelling in Computer Simulation: State of Art. In *Preprints of the Advanced Summer Institute'99 on Life Cycle Approaches to Production Systems: Management, Control, and Supervision*, 2 p. Lueven, Belgium.

Nance, R. E. 1994. The Conical Methodology and the Evolution of Simulation Model Development. *Annals of Operations Research* 53, 1-45.

Ören, T. I. 1989. A paradigm for Artificial Intelligence in software engineering. In *Advances in Artificial Intelligence in Software Engineering*, ed. Ören, vol. 1, JAI Press, Greenwich, CN.

Robinson, S. 1999. Simulation Verification, Validation and Confidence : a Tutorial. *Transaction of the SCS* 16 (2), 63-69.

Vangheluwe, H. L. 2000. DEVS as a Common Denominator for Multi-formalism Hybrid Systems Modelling. *IEEE International Symposium on Computer-Aided Control System Design*, ed. Varga, 129-134. IEEE Computer Society Press. Anchorage, Alaska.

Vangheluwe, H. L., J. De Lara, and P. J. Mosterman. 2002. An Introduction to Multi-Paradigm Modelling and Simulation. In *Proceedings of the 2002 AI, Simulation and Planning in High Autonomy Systems*, eds. Barros and Giambiasi, 9-20. Lisbon, Portugal.

Vangheluwe, H. L., and J. De Lara. 2002. Meta-Models Are Models Too. In *Proceedings of the 2002 Winter Simulation Conference*, eds. Yücesan, Chen, Snowdon, and Charnes, 597-605. IEEE Computer Society Press. San Diego, CA.

Wild, R. H., and J. J. Pignatiello Jr. 1994. Finding Stable System Designs: a Reverse Simulation Technique. *Communications of the ACM* 37 (10), 87-98.

Zeigler, B. P. 1976. *Theory of Modelling and Simulation*. Wiley & Sons, N.Y.

Zeigler, B. P., H. Praehofer, and T. G. Kim. 2000. *Theory of modeling and simulation. Integrating discrete event and continuous complex dynamic systems*. 2nd Ed. Academic Press.

AUTHOR BIOGRAPHY

MAMADOU K. TRAORE is Associate Professor in Computer Science at the Blaise Pascal University of Clermont-Ferrand, France. His current research focuses on multi-concept modeling and simulation. He can be contacted by e-mail at <traore@isima.fr>, and his web page is <http://www.isima.fr/~traore>.