# Guest Editorial: Special Issue on Computer Automated Multi-Paradigm Modeling

Modeling and simulation are becoming increasingly important enablers for the analysis and design of complex systems. To tackle problems of ever growing complexity, the focus of modeling and simulation research is shifting from simulation techniques to modeling methodology and technology. This trend is also visible in the software engineering community. Most noticeable is the shift from programming to modeling. Examples of this are the use of the Unified Modeling Language (UML) in the context of the Model Driven Architecture (MDA) and of (domain specific) tools to automatically generate prototype and even production quality code from high-level models.

In this special issue, the emerging field of Computer Automated Multi-Paradigm Modeling (CAMPaM) is presented. Because of the heterogeneous nature of embedded systems and the many implementation technologies, multi-paradigm modeling is a critical enabler for holistic design approaches (such as mechatronics) to avoid overdesign and to support system integration. Multi-paradigm techniques have been successfully applied in the field of software architectures, control system design, model integrated computing, and tool interoperability.

Multi-paradigm modeling spans the study of physical as well as software systems and combinations thereof. It adresses and integrates three orthogonal directions of research:

(1) *model abstraction*, concerned with the relationship between models at different levels of abstraction;
(2) *multi-formalism modeling*, concerned with the coupling of and transformation between models described in different formalisms;
(3) *meta-modeling*, concerned with the description (models of models) of classes of models. More explictly, the specification of formalisms.

Multi-paradigm modeling explores the possible *combinations* of these notions. It combines, transforms and relates formalisms, generates maximally constrained domain- and problem-specific formalisms, methods, and tools, and verifies consistency between multiple views.

## Modeling and Simulation

At a first glance, it is not easy to characterize the field of modeling and simulation. Certainly, a variety of application domains such as fluid dynamics, energy systems, and logistics management make use of it in one form or another. Also, a plethora of techniques from mathematics, artificial intelligence, numerical analysis, etc. are used. As a paradigm, modeling and simulation is a way of representing our *knowledge* about structure and behavior of systems and *answering questions* about them. Whereas the goal of modeling is to *meaningfully* describe a system, presenting information in an understandable, re-usable way, the aim of simulation is to be *efficient and accurate*. Crucial to the modeling and simulation endeavour is that there is a *homomorphic* relation between model and system: building a model of a real system and subsequently simulating its behavior should yield

the same results as performing a real experiment followed by observation and codifying the experimental results.

### Abstraction

Models of system behavior can be described at different levels of abstraction or detail as well as by means of different formalisms. The particular formalism and level of abstraction chosen depends on the background and goals of the modeler as much as on the system modeled. This level of abstraction, which may be different for each of the components or views of a complex system, is determined by the available knowledge, the questions to be answered about the system's behavior, the required accuracy of answers, etc.

In "Modeling Methodology for Integrated Simulation of Embedded Systems" by **Ledeczi, Davis, Neema, and Agrawal** (forthcoming in a regular issue of TOMACS), it is shown how the execution (or interpretation) of models at different levels of abstraction can be facilitated. A model interpreter translates the model into an executable form. This multi-grained simulation allows simulation of individual subsystems in isolation as well as replacing detailed behavior of aggregate subsystems by a high-level behavioral model that is more efficient. It even allows concurrent hardware/software simulation. Here, synchronous and asynchronous dataflow constitute the glue to combine the separate parts. Because modelers frequently switch between different levels of abstraction to answer different questions, inherent support for this is indispensable in modern system design projects.

Introducing hierarchy and subsequently replacing parts of the hierarchy by atomic components is an intuitively appealing way to "abstract" models. Abstracting can be seen as a special kind of transformation which preserves some properties of the system. The challenge is to model these transformations, and use these models to automate model abstraction and abstraction level selection.

### Formalism

Orthogonal to the choice of model abstraction level is the selection of suitable *formalisms* in which the models are described. The choice of formalism is related to the abstraction level, the amount of data that can be obtained to calibrate the model, the availability of solvers/simulators for that formalism, as well as to the kind of questions that need to be answered.

A formalism *specifies* all valid models that are part of it. Models in a formalism may be represented in a textual or graphical fashion, or both, depending on the formalism. A formalism typically has both syntactic and semantic parts. The syntactic part pertains to the form and structure of valid models whereas the semantic part pertains to the meaning of the models. In general, the syntax of a formalism, i.e., the elements of its language, is separated into a concrete and an abstract part. The concrete part concerns the actual appearance (e.g., whether an assignment is written as ':=' or '=' or whether a sum of three variables is written in infix 'x+y+z' or in prefix 'plus(x,y,z)' notation). The abstract part concerns how the language elements are connected together (e.g., that an assignment has a left-hand side and a right-hand side).

On the one hand, the meaning of a model can be expressed in an operational fashion by explicitly describing how a model can be simulated/executed/solved. On the other hand, meaning can be expressed in a denotational or transformational fashion by mapping the model onto an equivalent representation in a "known" formalism. For example, the meaning of a bond graph model can be obtained by mapping it onto a set of differential and

algebraic equations (DAE). Note how even the operational approach is in essence a transformation, transforming a model onto a behavior trace that is a model in an appropriate state trajectory formalism.

Robin Milner, in his 1991 Turing Award lecture, rejects the idea that there can be a unique conceptual model, or one preferred formalism, for all aspects of something as large as concurrent systems. Rather, many different levels of explanation, different theories, and languages are needed. This is even more true in the general context of complex systems modeling, where the need for different formalisms, as well as their combination and integration is driven by their increasing heterogeneity. This multifarious character emerges for one from the integration of different implementation technologies in the present-day design process. For example, the mechatronics design process advocates such an integrated system view to achieve optimal performance, given demanding cost contraints. In addition, models at different levels of abstraction are used to quickly evaluate design decisions, even combining hardware and software, both in production as well as in prototype form. Thus, high level behavior may have to be studied in conjunction with low level detailed effects for which different formalisms are best suited.

Not only do multiple formalisms need to be used, but a modeler must also be able to combine models in different formalisms through coupling and transformation. Within CAMPaM, this is called multi-formalism modeling. The Ptolemy project described in "A Component-Basesd Approach to Modeling and Simulating Mixed-Signal and Hybrid Systems" by **Liu and Lee** provides a sound theoretical basis for modeling different models of execution by dataflow based on token exchange. It shows how hierarchy can be the basis for using multiple models of computation in conjunction with one another. At each level in the hierarchy, a component acts as part of a model of computation but it may be implemented by a different model of computation at the more refined level. This is also referred to as heterogeneous refinement. Liu and Lee show in particular how the finite state machine (FSM), discrete event (DE), and continuous time (CT) formalisms operate and interact.

Here, the continuous time formalism takes a special place because it is fundamentally different from discrete event formalisms. Even though its execution is often performed by discretization in time, continuity assumptions govern the quality of sophisticated numerical solvers. Combining such continuous behavior with discrete state changes leads to *hybrid dynamic systems*, which entail very specific problems. In "Modeling, Simulation, Sensitivity Analysis and Optimization of Hybrid Systems" by **Barton and Lee** an overview of these is given including event detection and location, re-initialization of continuous state variables, sensitivity to parameter variations and the effect on optimization.

### Meta-modeling

A proven method to achieve flexibility for a modeling language to support many formalisms is to model the language itself. Such a model is called a meta-model. If we consider a modeling language as a concrete representation of a formalism, a meta-model describes a formalism. As a meta-model is a model in its own right, it must be expressed in the context of some formalism. This meta-formalism may again be described in a meta-meta-model. Traditionally, meta-models specify the syntax of a class of models. This notion can be generalized to include the semantics of the models (for example, by means of models of transformation). Such a meta-model specification of syntax and semantics is sufficient to automatically generate a full domain- and problem-specific modeling, and

possibly simulation, environment.

The advantages of such an approach are numerous. For example, by adapting the model of a formalism, and automatically generating a prototype modeling environment, design choices can be rapidly evaluated. Constructing problem-specific formalisms is preferable to the construction of all-encompassing, unifying formalisms. Not only is it hard to make the latter consistent and to provide a sufficiently powerful simulation engine, but too much flexibility puts the burden of restraint on the user. This is especially important in collaborative projects typical in industry where style guides are commonplace to give models a uniform look and feel that is easily interpreted by others. In addition, domain specific constraints provide an initial level of confidence in the designed model (e.g., that it satisfies cconservation of energy in case of a physical system).

The first project to extensively apply meta-modeling notions as its very foundation, aimed to support the exchange of models and data between Computer Aided Software/Systems Engineering (CASE) tools, and was called CDIF (CASE Data Interchange Format). It allowed the use of "best-of-class" tools for any one particular task without the need for duplicate data entry, and also facilitated "live" coupling of different tools. The original motivation of CDIF is presented in the overview "Metamodeling in EIA/CDIF - Meta-Metamodel and Metamodels" by **Flatscher**, which shows the CDIF technology to be of industrial strength. It adopted a four layer meta-modeling architecture: the base layer represents the data, followed by a model layer which contains the models to produce this data. On top of this, the first meta-model layer captures the modeling languages. This could include models of formalisms for control design such as the block diagram language with its gain blocks, integrators, and summing blocks, but it could also model formalisms such as bond graphs which have a different language consisting of junctions, dissipation, and storage. Each of the layers in this architecture is an "instance of" a layer one level up.

In order to exchange models between tools, the formalism these models are expressed in needs to be available. The strength of CDIF is that this information can be included in the exchange data as a model itself, i.e., as a *meta-model*. The CASE tool first processes the meta-model so it "understands" the data that follows. For this scheme to work, only the formalism to describe the meta-model has to be shared across tools. This shared formalism can then be a more compact language based on entities, attributes, and relations, that proves itself powerful enough to model any of the domain-specific formalisms that are used for the exchanged models.

Because of its successful use, this four layer architecture was adopted in the UML as well. The instantiation relation introduced by the object-oriented nature of UML, however, is different from the one between the meta-levels in the four layer architecture. In "Rearchitecting the UML Infrastructure" by **Atkinson and Kühne**, this is discussed in detail and the orthogonality of the two relations is presented to put forward a sound theory of meta-modeling in the context of object-oriented languages. Indeed, when the concept of meta-modeling is used loosely, i.e., it is not a strict requirement of instantiation between meta levels, it reduces to little more than a "packaging" notion as employed by modern programming and modeling languges. In an extension of the meta-modeling research, Ledeczi *et al* discuss the compositionality of meta-models.

### Transformation

Transformation of models relates the three orthogonal directions of CAMPaM: formalisms, abstraction, and meta-modeling. Some of the interesting model manipulations are:

—Formalism transformation: given a model in a certain formalism, these transformations convert it into a model expressed in another formalism. For modeling and simulation, possible transformations can be presented in a Formalism Transformation Graph (FTG).
—Model optimization: these transformations do not change the formalism in which the model is expressed. Their application results in a reduction of model complexity.
—Code Generation: these transformations produce a textual representation of the model (subject to syntactic constraints) suitable for interpretation by a simulator.
—Simulator specification: these specifications give the operational semantics of the model.

Formalism transformation has many uses. One use is the answering of particular questions about the system. Some questions can only be answered in the context of a particular formalism. For example, the hybrid dynamic systems formalism formulated by Barton and Lee is based on hybrid automata, and, therefore, well suited for numerical analyses. The $\chi$ language discussed in "Declaration of unknowns in hybrid system specification" by **Van Beek, Bos and Rooda** (forthcoming in a regular issue of TOMACS), on the other hand, is at a higher level than hybrid automata, and, therefore, more amenable to the modeling task: though the introduced "unknown" operator allows convenient specification of model initialization, it is less applicable from the point of view of numerical analysis.

An important characteristic of the work by Barton and Lee and Van Beek *et al* is that it facilitates general differential and algebraic equation (DAE) modeling as opposed to requiring an explicit ordinary differential equation (ODE) representation. The use of DAEs is especially important for plant modeling, a field that historically and currently applies continuous time models quite extensively. Discrete event models have deep roots in control theory, where continuous time models are captured by the ODE formalism that is amenable to execution in a discrete event framework. This is discussed by Liu and Lee for Ptolemy, as well as in "Dynamic Structure Multi-Paradigm Modeling and Simulation" by **Barros** (forthcoming in a regular issue of TOMACS), whose Heterogeneous Flow System Specification also facilitates continuous-time simulation.

In contrast, plant modeling based approaches rely on a dedicated continuous-time numerical solver for differential and algebraic equations. In addition to the support for implicit modeling, it also allows increased accuracy and the use of dedicated solvers depending on the characteristics of the simulation trajectories (for example, particular types of stiffness), instead of building the solver type into the model of computation. Here, the transformation of a model into a trajectory is more complex but the model itself may be simpler. This demonstrates how the complexity of the model-solver combination is invariant under behavior-preserving formalism transformations.

As mentioned earlier, the formalism transformation approach is typically applied to map the execution of a *new formalism* onto the (nearest) existing executable formalism. If this can be achieved, a simulation engine may be readily available. Typically, it is meaningful to introduce a new formalism for a specific application, encoding particular properties and constraints of the application. Often, translation involves some loss of information. This loss may be a blessing in disguise as it entails a reduction in complexity, hopefully leading to an increase in (simulation) performance. Usually, the aim of multi-step transformation is to eventually reach the trajectory level. In a denotational sense, performing a sequence of formalism transformations makes the *semantics* of models in formalisms explicit: the meaning of a model/formalism is given by mapping it onto some known formalism (whose meaning may in turn be given by mapping it onto an even more basic formalism and so

on). Though a multi-step mapping may seem cumbersome, it can be perfectly and correctly performed by tools. Note how the larger the number of intermediate formalisms becomes, the higher the potential for optimization (i.e., model complexity reduction) along the way.

It is critical that the selected model of execution to be associated with the model of a syntax be sufficiently powerful to capture the required interpretation idiosyncrasies. For example, a state transition diagram has no notion of time, and therefore, would not be suitable to model ordinary differential equations. In general, a formalism with extensive computation facilities may introduce overspecification, however. If, for example, a global notion of time is inherent in the model of computation, it imposes additional constraints on the implementation which may not be desirable.

Many formalisms have proven capable of representing the execution semantics of a wide variety of discrete event formalisms. The underlying formalism used by Van Beek *et al* is that of Communicating Sequential Processes (CSP). In "Formal Modeling of Discrete Event Systems: A Holistic and Incremental Approach Using Petri Nets" by **Bobeanu, Kerckhoffs and Van Landeghem** (forthcoming in a regular issue of TOMACS), it is place/transition Petri nets with time extensions (timed transitions with atomic firing), and in the work by Barros it is a variant of a discrete event system specification (DEVS) model.

Implementation of transformations depends on the formalisms involved. However, since models determined by some meta-model can always be described as graphs (subject to the constraints given by the meta-model), transformation may be performed by a generic graph-transformation. Therefore, it makes sense to combine meta-modeling and graph-grammars in a unifying framework. Meta-models determine the classes of graphs that are allowed on the left-hand side (LHS) and right-hand side (RHS) of a graph-grammar rule. A graph rewriting system will match LHS patterns in a host graph (the model) and replace them by the RHS. A graph-grammar can be viewed as a model in the graph-grammar formalism, which itself is specified in a meta-model.

## In Closing

This issue touches upon many aspects of the entire spectrum of modeling & simulation, mostly from a theoretical/methodological perspective. The interested reader is referred to a forthcoming special issue of IEEE TRANSACTIONS ON CONTROL SYSTEM TECHNOLOGY on the same topic, for specific applications of Computer Automated Multi-Paradigm Modeling (CAMPaM) in the field of control system technology.

We sincerely enjoyed putting this Special Issue together. We hope it provides a clear perspective on the activities in the field of CAMPaM, and the way it addresses present day issues in modeling & simulation. We wish to thank the authors who responded to the call for papers as well as the numerous referees who kindly provided in-depth reviews of the manuscripts. Thanks also to David Nicol, the editor-in-chief, for the opportunity to present CAMPaM to the Modeling & Simulation research community through ACM TOMACS and to John Konkle, editorial assistant, for his abundant and efficient organizational support.

PIETER J. MOSTERMAN
*The MathWorks, Inc.*
*Natick, Massachusetts*

HANS VANGHELUWE
*McGill University*
*Montreal, Canada*