MMMDE: Workshop on Mathematical Models for Model-Driven Engineering

(*Read MMMDE as "triple em dee e"*)

Zinovy Diskin*, Vadim Zaytsev†, Rick Salay‡, Bernhard Schätz§,
*Department of Computing and Software, McMaster University, Canada,
*Department of Electrical and Computer Engineering, the University of Waterloo, Canada,
†Instituut voor Informatica, FNWI, Universiteit van Amsterdam, The Netherlands,
‡Department of Computer Science, University of Toronto, Canada,
§Institut für Informatik, Technische Universität München, Germany,
§Software & Systems Engineering Department, fortiss GmbH, Germany,
*diskinz@mcmaster.ca, †vadim@grammarware.net, ‡rsalay@cs.toronto.edu, §schaetz@informatik.tu-muenchen.de

***Abstract*—Software engineering (SE) strives to learn from matured engineering disciplines, such as mechanical and electrical engineering (*physical engineering*, PE), and MDE is an essential step in this direction. Mathematical models are fundamental for PE, but should it be so for SE? What are similarities and differences in the development and use of mathematical models in SE vs. PE? How can SE and MDE benefit from a better understanding of these similarities and differences? These questions become even more challenging when we recognize that mathematical modelling and formalisation are not identical (although closely related), and the abundance of formal models in SE may actually hide the lack of mathematical models with all its negative (but perhaps negligible?) consequences.**

Questions above are seldom discussed in the MDE literature, but we believe they deserve special attention. The MMMDE Workshop aims at gathering together MDE experts who are concerned with developing mathematical foundations for MDE, understanding the role of mathematical modelling in engineering in general and SE in particular, and with relating these general thoughts to practical MDE problems. We want to "test the waters", and try to solidify broadly formulated concerns outlined above into several well-focused research questions or directions.

I. Motivation and Scope

There are essential similarities and essential differences between PE and SE, which essentially influence the value and roles of mathematical models. Below we present several observations about the subject (one subsection per observation), and the corresponding questions that we think are useful to discuss at the workshop.

Observation 0: Luxury vs. Necessity

While mathematical modelling is a commonplace practice in PE, SE can (arguably) survive without mathematical models. Indeed, building a bridge or a car without an a priori analysis would be too costly, and hence their mathematical modelling is a must. In contrast, testing and debugging can (pretend to) replace modelling, analysis and other mathematical means of providing correct-by-construction software. This leads to the following major question: *what should a mathematical model provide to be accepted and used in SE?*

Observation 1: Verification vs. Design

In PE, mathematical models are used for both checking the design and suggesting the design (Fig. 1 shows how it works). In SE, checking seems to be understood and exploited much more than suggesting. This distinction is the more so surprising as design in SE is much less bounded by external physical restrictions than in PE. Perhaps, this is a sign of the lack of mathematical rather than formal models in SE (see the next observation).

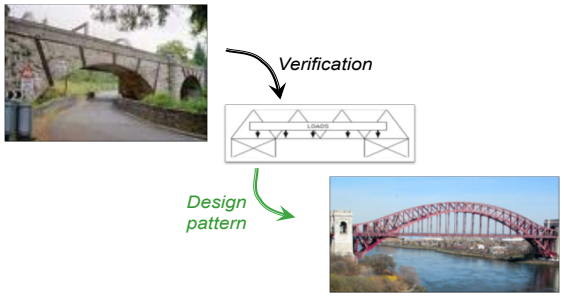

Fig. 1. From verification to design

Observation 2: Mathematical vs. Formal

Discrete domains (prevalent in SE) are much better amenable to formalisation than continuous ones (in PE); moreover, in design models, the object to be formalised (code) is itself a formal object. This may lead to an abundance of formal models in a typical SE process, but formal models are not necessarily mathematical models. An assembly program or byte code are formal objects, but they need mathematical models of their intended semantics if we are interested in understanding *what* these programs do on the specification level beyond sequences of machine instructions. Similarly, a Java program is a formal (or almost formal) but not a mathematical object if we are looking for a higher-level specification beyond sequences of programming operators. In a sense, the very non-trivial activity of *reverse engineering* can be seen as a transition from the formal to the mathematical world. On the other hand, quasi-mathematical but semi-formal models are widely used in PE. Of course, true mathematical models are themselves formal, but the benefits they provide often go beyond formalisation as such: specification and design patterns, consistent conceptual frameworks and terminological and notational frameworks based on them, ways of thinking about and understanding the domain are typical implications of mathematical rather than just formal modelling.

The differences between mathematical and formal are not always well understood in SE and MDE too; the latter often lacks mathematical models but the problem is not recognised as mathematical models are substituted by formal ones.

Observation 3: From Engineering to Physical to Mathematical to Formal

An important facet of the issue is how to bridge the (enormous) gap between engineering thinking and intuition on the one side, and mathematical formalisms on the other. Classical PE models like a pendulum and a mass hanging from a spring in mechanics, a bending beam and a shell in mechanics of materials, an RCL-contour or the entire electrohydraulic analogy in electricity are intuitive and manageable by an average PE-engineer, but each of them encapsulates fairly complex mathematical and bulky formal structures. These models can be seen as simple interfaces to the underlying mathematical structures. In fact, the transition from the engineering domain to its formalisation is mediated by a whole chain of models provided by different disciplines: from Engineering theories in Mechanics/Electricity to General Physics to Theoretical Physics to Mathematical Physics (only here we are in the realm of mathematics) to Numerical Methods, which employ their own terminological, conceptual, and mathematical frameworks.

We have somewhat similar "physical" models bridging the gap in SE too: automata of different types, Petri nets, grammars and rewriting systems of different types, different sorts of tables providing interfaces to different logics, control flow and data flow nets, dependence graphs, and also class diagrams, ER-diagrams, statecharts and message sequence charts constitute the "golden modelling fund" of SE. There are, however, essential differences between engineering models in PE and SE. Some of the SE-models above are indeed engineering interfaces to the underlying mathematics, others still lack an agreed formal semantics. Also, it appears that a typical MDE chain from engineering to mathematical is shorter and seemingly more primitive than in PE: from the engineering domain to an engineering model to its formalisation in code.

Observation 4: Forests vs. Trees

Models in PE are integrated into systems—model libraries, in which each model has its precisely defined goals and application conditions. It seems that models in SE are less integrated, and their applicability is less accurately specified. Software engineers often tend to master one modelling framework and supporting tooling, and apply it as broadly as possible without too much worrying about the very applicability of the framework to a given domain problem. Models in PE are integrated into "forests", whereas models in SE look more like isolated trees. For example, work on behaviour modelling done with Petri nets is not well related to work on behaviour modelling done with statecharts, the same for structural modelling with ER-diagrams, class diagrams and ontologies. Even worse is that the isolationism of models gives rise to the isolationism of the respective communities, and the problem becomes unmanageable.

Observation 5: The MDE Paradox

Model-driven engineering placed modelling and models at the centre of software development process. Researchers and practitioners learned to appreciate abstraction by observing the technical benefits it provides in the context of model-based requirement engineering, software design, and testing. However, even though this

appreciation of abstraction was widely promoted and praised, the use of actual mathematical models did not receive significant attention. While the model-centred MDE culture requires a greater mathematical precision and semantic accuracy, many modelling theories have limitations that are left unexplored or unknown to software developers, and a typical MDE tool can place an advanced technology on top of surprisingly weak semantic foundations. The result is that MDE enabled us to create, manipulate and use many kinds of models with extreme ease and effectiveness, but also brought to life many *implicitly* inconsistent or/and inadequate to their intended goals models, which often require manual adjustment after their automatic processing. By significantly increasing the demand for semantic and mathematical accuracy in modelling and models, but leaving the latter on basically the same level as before the pre-MDE era, MDE created a paradoxical impression of a decreased level of mathematical support in software development.

Observation 6: Categorical Thinking

Category theory (CT) is a mathematical theory of structures, and hence is an obvious candidate to mediate the transition from a structurally complex engineering domain to its mathematical models to formalisation. Moreover, categorical modelling by its very nature tends to integrate mathematical models into a consistent mathematical framework. However, CT is not a part of a common SE curriculum, its methods are (although quite natural but) unusual, and there are no good textbooks suitable for a software developer; the result is that CT is often considered as being excessively complicated. It seems there are no good and fast ways to remedy the problem, whose deeply rooted conservative educational and cultural factors hinder its technical solution.

II. Objectives and the Intended Audience

Observations above show enough differences in the value and application of mathematical models in PE and SE to justify (re)thinking and discussing the interaction of mathematical modelling and MDE and SE. Such a discussion seems need greater clarity in understanding the core issues than we have today — hence, this workshop.

The MMMDE Workshop aims at gathering together MDE experts who are concerned with developing mathematical foundations for MDE, understanding the role of mathematical modelling in engineering in general and SE in particular, and with relating these general thoughts to practical MDE problems. We want to "test the waters" and try to solidify broadly formulated concerns above into several well-focused research questions or directions, and perhaps make them accessible to the community via a publication. Perhaps, we could continue the workshop with the next edition of MoDELS in a more traditional setting with paper submission and reviewing process.

The intended audience is assumed encompassing three main groups:

- MDE researchers and practitioners with an affinity to mathematical and formal methods.
- Applied mathematicians or computer scientists applying (or wishing to apply) their research skills to MDE and having trouble in bridging the conceptual gap.
- SE/MDE practitioners who seek help in mathematical techniques but do not want to pursue mastery in the underlying theories.

III. Tentative Schedule

The workshop will consist of three parts: introduction and the keynote (first session, 9:00–10:20); the invited talks (10:45–12:00 and 13:15–15:00) and the panel with conclusive discussion (15:30–17:00). A tentative schedule is below (each talk slot assumes 5 minutes for a brief discussion).

Time	**Presenter**	**Topic**
9:00–9:20	Organisers	Introduction and goals
9:20–10:15	Tom Maibaum	Why Modelling Succeeds in Engineering: a Cookbook Approach to Modelling in MDE
10:15–10:45	Break	
10:45–11:15	Don Batory	$3x = 12$ means $x = 6$: Einstein's First Equation
11:15–11:45	Harald König	Why do more elaborate IDEs lead to worse software?
11:45–12:00	Discussion	Free exchange
12:00–13:00	Lunch	
13:15–13:30	Richard Paige	A set of provocative statements
13:30–14:00	Martin Gogolla	Observations on Support for Logical Reasoning on UML Models
14:00–14:15	Bran Selic	Think or Swim: On Engineering Methodology
14:15–14:30	Rick Salay	Why does math in SE make work harder when in other engineering disciplines it makes it easier?
14:30–15:00	Zinovy Diskin	What's bad with a bad mathemtical model?
15:00–15:30	Break	
15:30–16:30	Panel with Bran Selic, Don Batory, Tom Maibaum, Harald König	What should a mathematical model provide to be accepted and used in SE?
16:30-17:00	General discussion	+ Conclusions "What's next"?