Foundations for Model-Based Systems Engineering and Model-Based Safety Assessment

Antoine B. Rauzy and Cecilia Haskins

Department of Mechanical and Industrial Engineering

Norwegian University of Science and Technology (NTNU)

Trondheim, Norway

Abstract

This article is a contribution to the INCOSE initiative for model-based engineering transformation. Its material has been presented at the ALSEE tour event in Oslo in September 2016. The ideas developed here come from the practical and theoretical experience of the authors in both industrial and academic frameworks. We organize the discussion around six theses that aim at establishing robust conceptual foundations for the model-based engineering transformation. We focus on model-based systems engineering, model-based safety assessment and the relationship between these two disciplines. We report on active research initiatives that implement these six theses via the S2ML+X paradigm. We conclude with suggestions about future research and teaching activities.

Keywords: model-based systems engineering, model-based safety assessment

1 Introduction

Technical systems designed by industry are increasingly complex and interconnected. The processes by which they are designed, produced, operated, decommissioned and the respective organizations that implement these processes are also complex. To address this complexity, the different engineering disciplines (mechanics, thermic, electric and electronic, software, architecture…) digitalize their efforts, i.e. they produce models. We can describe this transition as entering the era of Model-Based Systems Engineering [Haskins 2011]. Model-based systems engineering uses models to facilitate the communication and thinking activities of systems engineers. Models offer visualizations that augment the understanding of the problem domain beyond the textual statements of requirements, functions, performance, and verification. When properly used they assist the systems engineer in creating a consistent view of the problem and their work toward achieving a satisfactory solution [Long and Scott 2011]. The implication of this paradigm is that the design of each system involves dozens of models, all at the same time [Blanchard 2008, Walden 2015]. In addition, many of these models are embedded into the systems and used for their operation [Derakhshanmanesh 2014]. Paradoxically, most engineers perform modelling without any training in the theory of models. However, models must be taken seriously, and considered as first class citizens. The emerging science of complex systems is the science of models. This presents a number of challenges:

- Better understand the nature of models and their roles in industrial processes;
- Develop the “Art of Modeling” in each engineering discipline;
- Manage models throughout the lifecycle of systems;
- Design tools and methods to support the integration of engineering disciplines/processes through the alignment of models they produce; and,
- Teach modeling, clarify its value proposition its role to (future) engineers and systems designers.

Meeting these challenges is not possible without solid conceptual foundations. We present here six theses to organize this discussion about these foundations, with a focus on two engineering

disciplines: system architecture and safety analysis. These theses should be considered in the philosophical sense, i.e. they are short sentences that summarize our vision and that are supported by an organized set of hypotheses, arguments and conclusions.

We use the term “system architecture” to designate processes by which 1) missions, functions and the organization of the system are specified and justified, and 2) the different engineering disciplines contributing to the design of the system are integrated. This broad definition covers many activities. Alternative terms “system design” and “systems engineering” are sometimes used. One of the outcomes of system architecture process is a description of the architecture of the system. This architecture as well as the system architecture process may be organized according to architectural frameworks, such as DoDAF, ToGAF and other. In the sequel, model-based systems engineering refers to the model-based approach in system architecture. Model-based systems engineering is in some sense defined in opposition to the document-centric approach, although both approaches may involve documents and models. The definition is thus more a matter of degree than a radical separation.

We use the term “safety analyses” to cover all assessments of the performance of the system considering the uncertainties and random events (e.g. failures) the system will experience while in operation. This includes risk analyses, reliability analyses, maintainability analyses, and safety analyses. Safety analyses relies in essence on models. In the sequel, model-based safety assessment (or analyses) refers to safety analyses relying on high modeling languages as opposed to more traditional approaches relying on low level modeling formalisms such as fault trees or reliability block diagrams. Here again the definition is more a matter of degree than a radical separation.

System architecture and safety analysis are essential in system design and operation. They both consider systems at the same level of abstraction: system architecture aims at specifying how the system should work, while safety analyses aim at assessing the likelihood that something goes wrong, and the possible consequences of such an event. As of today, system architecture models use modeling formalisms such as SysML [Friedenthal 2011], while safety models are designed typically with formalisms such as fault trees, block diagrams, event trees, Markov chains or stochastic Petri nets (see e.g. [Kumamoto 1996, Rausand 2004] for reference textbooks).

The remainder of this article is organized as follows. Section 2 presents our six theses. Section 3 reports on active research initiatives that implement these six theses via the S2ML+X paradigm. Section 4 discusses the implications in terms of teaching of the above developments. Finally, Section 5 concludes the article.

2 Conceptual foundations for model-based engineering

Modeling is a simplex operation in the sense of Berthoz [Berthoz 2012], i.e. it reduces the complexity without removing it totally. It is a way to tame the complexity, not a magic wand to make it vanish. Models are thus complex and consequently need to be structured, documented, managed… This suggests that we need an engineering of models and raises a series of questions:

- How to structure models?
- What is a good modeling language?
- What is a good palette of modeling languages?
- How to manage versions and configurations of models through the lifecycle of systems?

To answer these questions, we need to consider models both for what they are, which requires studying modeling languages, and what they are used for, which requires studying assessment tools.

Each complex technical system designed by industry results in hundreds if not thousands of models. These models are used not only to design the system, but also to operate and even to decommission it. Designing, communicating and maintaining these models has a high cost. Therefore, it is of

primary importance to understand (and to challenge) their respective roles and benefits. This leads us directly to our first thesis.

2.1 Thesis 1 - The diversity of models is irreducible

Models are working tools, not (platonic) ideals: a model is an abstraction of the entity it describes and is useful only because it is an abstraction. The content and the level of abstraction of a model depends on what is to be observed, i.e. on the (virtual) experiments to be performed on that model. The meaning and practical consequences of this observation must be examined with care.

First, this implies that it is not possible to design all of the models of a complex technical system within a unified framework. Each model presents a specific view on the system. The different views reflect dissimilar needs. Each engineering discipline has (and needs to have) its own modeling methodologies, formalisms and tools, dedicated for the specific purposes of that discipline. In that respect, models designed for system architecture, are not different from models designed in other engineering disciplines. They reflect the vision of the system architects and designers and are dedicated to this very purpose.

Consequently, models are not compositional: the set of models of a system is not a model.

There cannot be such a thing as a unique model or even a master model of a complex system. We must accept reality and live with heterogeneous models. Ensuring that these models are related to the same system cannot be decided a priori, on the first try. Rather than to presume multiple modelers are using the same ontology, we must presume a cacophony of models then work to unify and harmonize them. The results reflect necessarily an organizational process, with all the inherent consequences this entails in terms of mutual understanding of individuals and groups having different cultures, of power relationships and diverging interests within and between organizations and so on.

This also applies to the relationships between system architects and safety analysts. Although both consider systems at the same level of abstraction, their visions and purposes are different and must remain so. Improving the way the two disciplines co-exist and co-influence each other is of great interest, but attempts to merge their activities and their models are dangerous “dreams”.

One of the main reasons for these difference stands in our second thesis.

2.2 Thesis 2 - There is an epistemic gap between pragmatic and formal models

Models are created at different levels of abstraction, for different purposes and in different modeling formalisms. There are two fundamental categories of models: pragmatic models that aim primarily at supporting the communication amongst stakeholders (see e.g. [Weilkiens 2015]) and formal models that aim primarily at calculating something, typically indicators. Formal models are used also for simulations or to generate artifacts such as computer code, or even physical objects (3D printing, additive manufacturing…).

Pragmatic models, when written in standardized graphical notations such as BMPN [White 2008], SysML [Friedenthal 2011] or OPM [Dori 2016], are sometimes called semi-formal. As their purpose is to facilitate communication, they keep implicit a lot of knowledge and take a broad outlook on the system under study. Formal models on the other hand essentially encode and organize (a given type of) mathematical equations. These models, typically designed in modeling languages such as Modelica [Modelica 2014], Lustre [Halbwachs 1991] or AltaRica [Prosvirnova 2013], make every detail explicit. They focus on some specific feature of the system under study. Obfuscation is a good test to separate the two categories of models. Take a pragmatic model and rename its elements with abstract names such as X, Y, Z. The model loses its meaning and interest because to understand and use it, you have to refer to the system under study. The virtual experiments performed on this type of model are primarily communications (brainstorming, negotiations…) that assume that the

stakeholders share a common knowledge about the described entity (in particular, names of its components). Now take a formal model and obfuscate it in the same way: nothing changes. The calculations performed on the model produce exactly the same results. It is even fully possible to delegate these virtual experiments to a third party (human or computer) that has strictly no idea of the described entity. Formal models have a semantics, i.e. they are interpreted as mathematical objects as opposed to pragmatics, i.e. an interpretation in the ``real'', ``physical'' world. Note that the distinction between semantics and pragmatics has important consequences in linguistics; see e.g. [Cruse 2011].

This epistemic gap between pragmatic and formal models must be examined with care, as it has important consequences for systems engineering processes.

First, pragmatic models and formals models have radically different natures and purposes. Both types of models are indeed useful, which means that systems engineering processes should rely on both.

Second, passing from pragmatic models to formal ones requires an engineering process. This process cannot be fully automated because it requires making explicit the heretofore-implicit knowledge. Attempts to "decorate" pragmatic models with formal information in the hope of generating formal models yields disappointing results: the source models are blurred and overloaded, losing their ability to support a seamless communication, while the generated formal models are incomplete and uselessly complex. This is the reason why attempts to generate safety models (which are formal) from system architecture models (which are pragmatic) have never been successful (this issue will be further discussed later in this paper). Therefore, the central question is not to generate one type of model to the exclusion of others, but to ensure a seamless coexistence of both types of models.

Third, as pragmatic models are computerized (thanks to the above-mentioned standardized notations), we can design tools to process them. These tools may perform consistency checks and other types of verification. They can also "execute" models, i.e. perform simulations, but this requires formalizing notations so to give them an executable semantics. It is also possible to trace impacts of changes in requirements on state machines describing the lifecycle of the system or on diagrams representing its functional architecture, see e.g. [Lebeaupin 2017]. In any case, these processing tools should be designed from a syntactic point of view as opposed to a semantic point of view, i.e. by considering models as structures in which it is possible to navigate and on which some structural operations can be performed, but without interpreting these structures (i.e., without assigning them any meaning). Note that the idea of considering objects under study from a purely syntactic point of view is hardly new: its power of has been demonstrated for instance in linguistics, indeed by Chomsky [Chomsky 1957] and in mathematical logic, via for instance Herbrand's interpretation of first order logic, see e.g. [Ebbinghaus 1996].

As we shall see, the syntactic point of view is also helpful to align heterogeneous models starting with our third thesis.

2.3 Thesis 3 - Behaviors + Structures = Models

Some readers may recognize that this thesis echoes the title of the famous book by Niklaus Wirth "Data structures + algorithms = programs" [Wirth 1976]. The modeling activity needs to be supported by concepts, methods and tools. Two important remarks are necessary at this point. First, to design a model, especially a formal model, we need a modeling language (would it be purely graphical), just as to design a program, we need a programming language. Second, models aim at simplifying in some way the complexity of systems. However, if the system under study is complex, the models of this system cannot be "just" simple because this simplification would lose too much information about the system.

We can make several observations here.

First, any formal modeling language is the combination of a mathematical framework to describe the behavior of the system under study and a structuring paradigm to organize the model. Pragmatic models generally are not based on a mathematical framework, even if they do support some simulation capability [Yaroker 2013]. Examples of underlying mathematical frameworks in formal models include for instance ordinary differential equations in Simulink and Modelica; Mealy machines in Lustre; guarded transitions systems in AltaRica and so on.

Second, the choice of the appropriate mathematical framework for a model depends on which aspect of the system we want to study, i.e. eventually what kind of virtual experiment we want to perform on the model, e.g. multi-physic simulation with Modelica, generation of embedded controllers with Lustre or probabilistic safety analyses with AltaRica.

Third, structuring paradigms are independent to a very large extent of the chosen mathematical framework. They can be studied on their own and applied to all mathematical frameworks. Structuring paradigms for modeling languages are derived from those of programming languages where object-orientation is dominant, if not hegemonic, in industrial practice (for reference monographs that strongly influenced, directly or indirectly, the design of most of modeling languages see e.g. [Meyer 1988, Abadi 1998, Rumbaugh 2005]).

For modeling languages, prototype-orientation seems particularly well suited. Prototype-orientation is a kind of object-orientation in which an object can be defined either via the class/instance mechanism (as in object-orientation) or directly via the notion of prototype [Noble 1999]. Ideas behind prototype-orientation stem from important works in cognitive science (see e.g. [Lakoff 1990]). They also echo works on how the knowledge is created (e.g. Hatchuel's CK-theory [Hatchuel 2009]). To summarize, prototypes are used when the system is analyzed with a top-down approach, i.e. the knowledge about the system is not stabilized yet, while the class/instance mechanism is used with a bottom-up model construction, i.e. when the knowledge about the system is sufficiently mature to develop extensive libraries of on-the-shelf modeling components. Reuse is also possible in the prototype/top down approach, but it is reuse of modeling patterns rather than reuse of modeling components (just as design patterns in software engineering [Gamma 1994] are different from libraries of reusable classes like the Qt, see e.g. [Lazar 2016]). In that respect, the system architecture and safety analysis processes are similar: they are essentially top-down approaches, relying on modeling patterns.

It is interesting to note that, from a historical perspective, the first object-oriented language, Simula [Kirkerud 1989], was actually a modeling language more than a programming language and that the first language that popularized object-orientation, Smalltalk [Goldberg 1983], which was strongly inspired by Simula, considers actually classes as modifiable objects, i.e. is very close to the prototype-orientation paradigm.

Object- and prototype-orientated constructs are also of great interest to represent structures without behavior, or more exactly structures without a formal semantics, and to perform operations on these structures (as suggested above). This is the reason why they should be used extensively to describe systems architectures. Hence our fourth thesis:

2.4 Thesis 4 – One should not confuse models with their graphical representations

At first, this thesis may seem at best extremely provocative, as most of the models in both system architecture and safety analyses (as well as in other engineering disciplines) are authored via graphical modeling environments and many practitioners just refuse to write a single line of code. However, graphical modeling is mainly useful to describe structural parts of models and systems (for an interesting discussion of graphical modeling see e.g. [Fuhrmann 2011]). It is hard to conceive how to author a differential equation or the probability distribution of the basic event of a fault tree graphically. Behavioral descriptions, such as Markov chains or Petri nets, can be represented graphically. However, as soon as models become large, which is the case for nearly any industrial-

scale system, their graphical representations become more problematic than useful: as they cannot fit into any reasonable space (computer screen or printed out paper), the analyst must visualize them by parts. This means that s/he must develop a global cognitive model to understand local graphical representations.

Moreover, it is often hard and counterproductive to split the model into parts *a priori*. This remark applies to models in general: 1) many details of models are better described by code (text) than graphics and 2) as they enlarge, it is not possible to present them in one visualization. In other words, models exist independently of their graphical representations. These graphical representations, even taken together, cannot fully describe the model, except in the most simplistic cases. Thus, it is often very convenient to have several partial graphical representations for the same information and to extract dynamically graphical representations according to one's needs.

Here again, the parallel with software engineering is fruitful. It is useful to represent the architecture of software by diagrams such those of UML [Rumbaugh 2005]. However, the software exists independently of these representations and the code is the ultimate reference. Moreover, below a certain level of abstraction, the code gives a more compact, more precise, in a word more useful, information than any drawing. At the end of the day, humans invented writing to overcome the lack of precision of drawing.

Neither systems architecture models nor safety analyses models are purely structural. They represent also systems behavior. Because they consider systems at the same level of abstraction, they tend to use the same type of behavioral description. This remark is formalized by our fifth thesis.

2.5 Thesis 5 - Discrete event systems are the (only) suitable mathematical framework to describe behaviors at a system level

We use here the term discrete event systems loosely. By discrete event systems we mean representations that assume that at any point of time the system is in a certain state and that it changes state when, and only when, a significant event occurs. This type of model is thus very different from differential equations that represent continuous changes of states.

Safety models are event-driven and probabilistic in essence. Probabilistic events are at the very core of (static and dynamic) fault trees, event trees, reliability blocks diagrams, Markov chains, stochastic Petri nets, AltaRica and all other modeling formalisms used in this domain.

Discrete events play also a significant role in modeling systems architectures, for instance the clocked transitions systems of OPM, and the UML and SysML state machine diagrams and sequence diagrams (inspired from message sequence charts of SDL).

However, these various mathematical frameworks are not equivalent. There are already subtle differences between Harel's Statecharts [Harel 1987] and UML and SysML state machine diagrams, see e.g. [Eshuis 2009] for a discussion. In the same vein, UML and SysML sequence diagrams and message sequence charts are slightly different, see e.g. [Harel 2003]. Computer science produced an incredible variety of formalisms to describe state machines, as well as process algebras (see e.g. [Milner 1989] for a classical textbook) that may also be a good candidate to represent certain aspects of systems' behaviors, see e.g. [Issad 2016] for recent developments.

It remains that representing systems behavior by discrete states that change under the occurrence of events is probably the right level of abstraction needed by both system architecture and safety analysis and more generally by any description considering systems at the same level. The future development of models engineering will tell us whether it is possible to get a more unified view of state machines. At least some well defined correspondences between the various formalisms should be established. Alignment of heterogeneous models is precisely the subject of our sixth and last thesis.

2.6 Thesis 6 - Abstraction + Comparison = Synchronization

As stated, the design, production, operation and decommissioning of a system involves the creation, maintenance and alignment of dozens, if not hundreds, of models. These models are designed by different teams in different languages at different levels of abstraction, for different purposes. Models mature also at different rates. The question is how to ensure that they describe the same system, i.e. how to synchronize them.

There are at least four distinct aspects in this question: a first one concerns the management of models in the context of the extended enterprise. This is the realm of collaborative data bases, product life cycle and product data management environments, see [Stark 2011]. The concept of "digital twin" is gaining popularity to designate systems in charge of models (and engineering data) management. A second aspect is related to the seamless cooperation of models of different abstraction levels. This is an important and difficult topic, see e.g. [Mainini 2012] for an interesting discussion. A third aspect regards the co-simulation of heterogeneous but compatible models, such as experiments from the Ptolemy project [Ptolemaeus 2014]. Our thesis is related to a fourth aspect, namely the alignment of heterogeneous models representing the system at about the same level of abstraction. A paradigmatic example of such alignment is indeed the alignment of system architecture models and safety models. This alignment is an industrial necessity and is required by Safety Standard such as IEC 61508 [IEC61508] and IEC 61511 [IEC61511].

The heterogeneity of these models makes it impossible to compare them directly. To compare them, we first have to abstract them into a common language, and then perform a comparison of their abstractions (see Figure 1). Once the comparison has been made, it is possible to go back to original models via a concretization mechanism. This principle is close to Cousot's abstract interpretation of programs [Cousot 1977]. As the behavioral part of models is purpose-dependent, the main way to compare models is to compare their structure. The structure of models reflects the structure of the system, though to a limited extent.

The abstraction, comparison, and concretization mechanisms can depend on the type and maturity of models. In the preliminary phase of a project, it may consist simply in comparing dictionaries (the names of components in each model). Later, more elaborate synchronization schemes can be put in place. Libraries of abstractors, comparators and "concretizators" could be defined. These ideas have been recently applied by Legendre on a realistic proof-of-concept (the design of the fire detection system of a military helicopter) [Legendre 2016]. To conclude this thesis, note that the objective of the whole synchronization process is not to align models fully. Rather, it is to identify areas of disagreements.

3 The S2ML + X paradigm

This section aims at presenting and discussing on-going research and development projects we are involved in that attempt to put into practice the six theses developed in the previous section. The guideline of these projects could be stated as the study from both a theoretical and a practical point of view of the S2ML+X paradigm.

3.1 System Structure Modeling Language (S2ML)

To study the interest of prototype-orientation in the context of modeling languages, the first author and his colleagues recently developed System Structure Modeling Language (S2ML), a small theoretical modeling language containing only structuring mechanisms [Batteux 2015]. S2ML gathers and organizes in a systematic way structuring constructs stemmed from both the object-oriented and the prototype-oriented programming. S2ML confirms that most of the structural relationships between the components/functions of a system are captured by four types of relationships: "is-part-of" (composition), "is-a" (inheritance), "uses" (aggregation) and "connect" (connection). Composition is the basic tool to decompose systems into subsystems (both from a functional and physical

standpoint). The system composes its subsystems, which in turn may compose sub-sub-systems and so on. Inheritance is the basic tool to describe abstraction: a car is a (inherits from) vehicle, meaning that it has all of the features of a vehicle, possibly specialized, plus some additional ones. In many situation, although a component or a function A requires another component or function B to perform its mission, B is not part of A. Rather, A uses B. Aggregation is the basic tool to describe such situations. Finally, connections are used the basic tool to describe how from of matters, energy or information circulate through the network of components or functions of a system. These abstract relationships are at the very core of systems architecture and models structure. Some additional mechanisms are also very important, such as the class/object (instance of class) mechanism as well as various notions of polymorphism.

Eventually, S2ML can be seen in two ways.

First, as a domain specific modeling language on its own, dedicated to architecture description, or more exactly to functional and physical decompositions of systems. The importance of domain specific languages for engineering is now well established, see e.g. [Fowler 2010, Selic 2007]. With respect to this first vision, S2ML clarifies and generalizes constructs found in other formalisms dedicated to structural descriptions, including for instance the structural diagrams of SysML.

Second and more importantly, as a complete and versatile set structuring paradigm that can be applied to any mathematical framework, as illustrated in Figure 2. With that respect, S2ML illustrates our third thesis, structure + behavior = model.

The richness of S2ML (even if no behavior is attached to its elements) makes it possible to develop quite complex models efficiently. This is the reason why a reference textual grammar has been developed for S2ML, and standardized graphical representations (close to those of SysML) are proposed to capture various aspects of models, illustrating in this way our fourth thesis.

We have also used S2ML in the context of three recent PhD theses. The first one was on reverse engineering of textual specifications using a scenario-based approach [Issad 2016]. S2ML was used there both to describe the system architecture (this model has been co-designed with the scenarios) and as a mean to structure scenarios. The second PhD thesis was on the agile development of corpuses of requirements [Lebeaupin 2017]. The idea was to apply on requirements the syntactic approach discussed Section 2.2 and to elicit them while designing S2ML models of the system architecture (and of course to link the syntactic structures of requirements with these models). Both PhD research efforts were supported by partnerships with industry (with Siemens for the first one and SAFRAN for the second one) and are illustrations of our first four theses. The third PhD thesis was on the use of the model synchronization approach on the previously mentioned example of a fire detection system of a military helicopter [Legendre 2016] and was therefore a proof-of-concepts for our sixth thesis. S2ML was used in this case as a pivot language between system architecture models (designed with SysML) and safety models and was conducted in a partnership with CEA List and DGA (the French Directorate General of Armaments).

The most advanced implementation of the S2ML + X paradigm is the design of the object-oriented modeling language AltaRica, which we shall describe here. But, before presenting AltaRica, we need to introduce the whys and wherefores of the so-called model-based safety assessment, as implemented by this modeling language.

3.2 The promise of the model-based approach in reliability engineering

The most widely used modeling formalisms for safety analyses lack either expressiveness, e.g., fault trees and event trees, or structure, e.g., Markov chains and stochastic Petri nets. Consequently, they are far from system specifications. These deficiencies make the models hard to design, hard to share with stakeholders, and even more importantly, hard to maintain through the entire lifecycle of systems.

Modeling systems in a more structured way and with suitable mathematical frameworks can reduce the distance between systems specifications and models, without increasing the complexity of calculations. This is the promise of the so-called model-based safety assessment. This approach provides the ability to animate/simulate models, to ease their validation, and to share them with stakeholders. Moreover, this presents the following important benefits for safety analyses *stricto sensu*:

- Such a model can address several safety goals, which eases versioning, configuration and change management;
- It can be assessed by several assessment tools, which increases versatility of assessments and quality-assurance of results (but with a cost);
- It allows fine grain analyses, which limits over-pessimism resulting from coarse grain analyses as performed for instance with fault trees.
- Its maintenance is alleviated significantly, as it is closer to systems specifications.
- The same formalism can be used to model simple static models as well as dynamic models, hence facilitating the acquisition of competences and the deployment of tools.
- The graphical animation of models makes it possible to share them with non-specialists.
- The same technology can be used not only for safety analyses but more generally to assess performance of systems (in terms of costs, delays, production levels…) subject to uncertainties.

There is of course an initial cost to pay for these benefits, but the return of investments is quickly positive. The idea of the AltaRica project is thus to use S2ML as a set of structuring constructs and a suitable mathematical framework for the description of behaviors.

3.3 The AltaRica 3.0 Project

As discussed above, the goal is to find an appropriate mathematical framework to perform safety analyses. Our fifth thesis asserts that stochastic discrete event systems provide such a framework. The first author proposed the generic notion of stochastic guarded transitions systems (GTS) for that purpose [Rauzy 2008]. GTS generalize, without sacrificing algorithmic rigor, all states/events formalisms used for safety analyses, including Markov chains and advanced implementation stochastic Petri nets (see e.g. [Signoret 2009]). Compared to state machines such as Statecharts, GTS are fully compositional. This is the reason why they should be preferred for safety analyses (it is anyway possible to translate any Statechart model into a GTS automatically).

The combination of GTS and S2ML is implemented into the modeling language AltaRica 3.0 [Prosvirnova 2013]. An integrated modeling environment for the third version of the language is currently under development as join effort of the Open-AltaRica team at IRT-SystemX (Paris, France) and the RAMS group at NTNU. Industrial partners (Airbus, Safran and Thalès) support this project, which aims to benefit from about fifteen years of successful industrial and academic experience using the two previous versions of the language. Assessment tools under development include tools that work directly on AltaRica models (such as a stochastic simulator) as well as compilers for lower level formalisms (such as fault trees and Markov chains). Figure 3 presents a snapshot of the AltaRica 3.0 project.

AltaRica 3.0 is significantly more powerful than the previous versions of the language. First, guarded transitions systems make it possible a larger category of systems than it was possible to deal with with the previous versions (notably looped systems) [Batteux 2017]. Second, structuring constructs stemmed from S2ML make much easier the design and the re-use of modeling patterns, which makes in turn the models easier to design, to debug, to communicate and to maintain [Batteux 2018]. Although AltaRica potentially represents the future of reliability engineering and system safety, it does not solve all of the problems of this domain. The assessment of safety models is actually provably hard. Some important questions, such as the equivalence of two discrete event systems, are

undecidable, see [Esperza 1998]. The calculation of probabilistic indicators is extremely resource consuming. This is not a problem of technology. It has been mathematically proven that they are computationally intractable (technically, the underlying problem is #P-hard as shown by Valiant) [Valiant 1979]. See also [Papadimitriou 1994] for a reference book on computational complexity. Practical assessment tools perform unwarranted approximations that may impact the significance of the results. Safety models always result in a tradeoff between the accuracy of the description and the ability to perform calculations. Finding a suitable compromise for a given system depends upon the expertise of the safety analyst. This is another reason why no simple translation scheme can be defined from system architecture models to safety models.

4 Discussion - Teaching Models Engineering

New concepts, techniques, and tools are introduced in industry often thanks to the injection of new generations of engineers trained to use these concepts, techniques, and tools at University. It is therefore of primary importance to train new generations of engineers to Model-Based Systems Engineering. However, current curricula are facing the following constraints:

1. Complex systems do not fit neatly in the time span of a typical academic semester, although some interesting experiments have been reported, e.g. [Haskins 2013];
2. Acquiring and maintaining sophisticated model-based engineering tools is out of the reach of most University budgets.
3. The design of new courses based on an emerging discipline takes place in the context of a historical revolution: a situation where the professor is not the only, nor even the main, knowledge holder.

The second author has applied systems engineering in a project-based course where there has been limited success introducing simple models. But, most often, once the scale of the models outgrows what can be easily visualized (e.g., a piece of A3 paper), the models stop growing or being used. Nevertheless, the collaborative approach this course promotes is extremely appreciated by students and is a key enabler to raise their interest for our discipline.

The first author introduced a few years ago a course on model-based systems engineering, first at Centrale Paris (Paris, France) for third year students (500 students) then at NTNU for master students (20 students), and at CentralePékin (Beijing, China) also at master level (50 students). He also has a solid experience in teaching model-based safety assessment. Here follow some remarks and lessons learned.

Modeling has much to do with programming. Models, as programs, are artefacts, sequences of symbols obeying a certain syntax (grammar) and having a certain semantics. Therefore, the science and engineering of models is tightly intertwined with computer science. In that respect, computer science can be viewed from three levels:

- As the study of discrete structures (graph, automata, languages…) which is at the core of the science of models, which means that a course on model-based systems engineering must allow adequate room for these concepts;
- As an engineering discipline (software engineering), which is quite far from models engineering;
- As a toolbox for the engineer (typically via scripting languages such as Matlab or Python), i.e. a nowadays essential productivity tool, especially for models engineering.

It is too early to draw definitive conclusions, but it is clearly difficult to teach model-based systems engineering to students who have no background in computer science or discrete mathematics. On the contrary, students with such background acquire quickly and relatively easily the concepts and methods of the discipline. The whole course is now evolving into an advanced introduction to discrete mathematics applied to model-based systems engineering. It relies for a part on the S2ML+X

paradigm as students are asked to design models in various small domain specific languages relying on this paradigm.

5 Concluding Remarks

As we enter the era of Model-Based Systems Engineering models must be considered as first class citizens and taken as the object of in-depth scientific studies. There will be no good model-based engineering without developing the science of models. Reciprocally, the science of models should be embodied in practice, by developing modeling concepts, languages, techniques and tools. Much remains to be done, although very significant progresses have been made in the recent years.

We are now facing a number of challenging issues to deal with new generations of systems as these systems are:

- Opaque: their states can be observed only by indirect means;
- Reflective: they embody models of their own behavior and environment;
- Deformable: their architecture changes throughout their mission.

We have to forge the concepts to face these challenges. We have to teach these concepts to new generations of scientists and engineers. A vast, exciting and inspiring program awaits.

6 References

Mauricio Abadi and Luca Cardelli. A Theory of Objects. Springer-Verlag. New-York, USA. ISBN 978-0387947754. 1998.

Michel Batteux, Tatiana Prosvirnova and Antoine Rauzy. System Structure Modeling Language (S2ML). AltaRica Association. 2015. archive hal-01234903, version 1.

Michel Batteux, Tatiana Prosvirnova and Antoine Rauzy. AltaRica 3.0 Assertions: the Why and the Wherefore. Journal of Risk and Reliability. Professional Engineering Publishing. September, 2017. doi:10.1177/1748006X17728209.

Michel Batteux, Tatiana Prosvirnova and Antoine Rauzy. AltaRica 3.0 in 10 Patterns. article submitted the International Journal of Critical Computer-Based Systems. 2018.

Alain Berthoz. Simplexity: Simplifying Principles for a Complex World. Yale University Press. New Haven, CT, USA. ISBN 978-0300169348. 2012.

Benjamin S. Blanchard and Wolter J. Fabrycky. Systems Engineering and Analysis. Pearson. Upper Saddle River, NJ 07456, USA. ISBN 978-0137148431. 2008.

Noam Chomsky. Syntactic Structures. Mouton. The Hague, The Nederland. ISBN ISBN 978-9027933850. 1957.

Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. ACM Press. pp. 238–252. New York, NY, USA. 1977. Los Angeles, California.

Alan Cruse. Meaning in Language: An Introduction to Semantics and Pragmatics. Oxford University Press. Oxford, U.K. ISBN 978-0199559466. 2011.

M. Derakhshanmanesh, J. Ebert, T. Iguchi, & G. Engels. (2014, September). Model-integrating software components. In International Conference on Model Driven Engineering Languages and Systems (pp. 386-402). Springer International Publishing.

Dov Dori. Model-Based Systems Engineering with OPM and SysML. Springer Verlag. Berlin, Heidelberg, New York. ISBN 978-1-4939-3294-8. 2016.

Heinz-Dieter Ebbinghaus, Jörg Flum and Wolfgang Thomas. Mathematical Logic. Springer-Verlag. New-York, USA. ISBN 978-0387942582. 1996.

Rik Eshuis. Reconciling statechart semantics. Science of Computer Programming. Elsevier. 74. pp. 65–99. 2009. doi:10.1016/j.scico.2008.09.001.

Javier Esperza. Decidability and Complexity of Petri Nets Problems - An introduction. Lectures on Petri Nets I: Basic Models. W. Reisig and G. Rozenberg Ed.. Springer. ISBN ISBN 3-540-65306-6. 1491. pp. 374–428. 1998

Martin Fowler. Domain Specific Languages. Addison-Wesley. Boston, MA 02116, USA. ISBN 978-0321712943. October, 2010.

Sanford Friedenthal, Alan Moore and Rick Steiner. A Practical Guide to SysML: The Systems Modeling Language. Morgan Kaufmann. The MK/OMG Press. San Francisco, CA 94104, USA. ISBN 978-0123852069. 2011.

Hauke A. L. Fuhrmann. On the Pragmatics of Graphical Modeling. Book on Demand. Norderstedt, Germany. ISBN 978-3844800845. 2011.

Adele Goldberg and David Robson. Smalltalk 80: The Language. Addison-Wesley Longman Publishing Co., Inc.. Boston, MA, USA. ISBN ISBN 0-201-11371-6. 1983.

Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. Design Patterns - Elements of Reusable Object-Oriented Software. Addison-Wesley. Boston, MA 02116, USA. ISBN 978-0201633610. October 1994.

David Harel. Statecharts: a visual approach to complex systems. Science of Computer Programming. 8:3. pp. 231-274. June 1987. doi:10.1016/0167-6423(87)90035-9.

David Harel and Rami Marelly. Come Let's Play. Springer. Berlin, Germany. ISBN 3-540-00787-3. 2003.

Cecilia Haskins. Getting Students Hooked on Systems Engineering. Procedia Computer Science. Elsevier. 13. pp. 976–982. 2013. doi:doi: 10.1016/j.procs.2013.01.102.

Cecilia Haskins, A historical perspective of MBSE with a view to the future. In INCOSE international symposium (June 2011, Vol. 21, No. 1, pp. 493-509). DOI: 10.1002/j.2334-5837.2011.tb01220.x

Nicolas Halbwachs, Paul Caspi, Pascal Raymond and Daniel Pilaud. The synchronous dataflow programming language Lustre. Proceedings of the IEEE. 79:9. pp. 1305–1320. 1991.

Armand Hatchuel and Benoit Weill. C-K design theory: an advanced formulation. Research in Engineering Design. Springer. 19:4. pp. 181–192. 2009.

International IEC Standard IEC61508 - Functional Safety of Electrical/Electronic/Programmable Safety-related Systems (E/E/PE, or E/E/PES). International Electrotechnical Commission. Geneva, Switzerland. ISBN ISBN 978-2-88910-524-3. April, 2010.

International IEC Standard IEC61511 - Functional safety - Safety instrumented systems for the process industry sector. International Electrotechnical Commission. Geneva, Switzerland. ISBN 978-2-8322-4752-5. April, 2016.

Mélissa Issad, Leïla Kloul and Antoine Rauzy. Incremental Modeling Methodology of Railway System Specifications. Proceedings of the Seventh International Conference on Complex Systems Design and Management, CSDM Paris 2016. Gauthier Fanmuy, Eric Goubault and Ed.. Springer International Publishing. ISBN 978-3-319-49102-8. pp. 95–111. 2016. doi:10.1007/978-3-319-49103-5-8.

Bjorn Kirkerud. Object-Oriented Programming With Simula. Addison Wesley. Boston, MA 02116, USA. ISBN 978-0201175745. 1989.

Hiromitsu Kumamoto and Ernest J. Henley. Probabilistic Risk Assessment and Management for Engineers and Scientists. IEEE Press. Piscataway, N.J., USA. ISBN 978-0780360174. 1996.

George Lakoff. Women, Fire, and Dangerous Things: What Categories Reveal About the Mind. The University of Chicago Press. Chicago, Illinois, U.S.A. ISBN 978-0226468044. April 1990.

David Long and Zane Scott. *A primer for model-based systems engineering*. Blacksburg, VA: Vitech Corporation. 2011.

Guillaume Lazar and Robin Penea. Mastering Qt 5. Packt Publishing Limited. Birmingham, United Kingdom. ISBN 978-1786467126. December 2016.

Benoît Lebeaupin, Antoine Rauzy and Jean-Marc Roussel. A language proposition for system requirements. Proceedings of 11th Annual IEEE International Systems Conference. IEEE. Montréal, Canada. April, 2017.

Anthony Legendre, Agnès Lanusse and Antoine Rauzy. Directions towards supporting synergies between design and probabilistic Safety assessment activities: illustration on a fire detection system embedded in a helicopter. Proceedings PSAM'13. IPSAM. Seoul, South-Korea. 2016.

Laura Mainini and Paolo Maggiore. Multidisciplinary Integrated Framework for the Optimal Design of a Jet Aircraft Wing. International Journal of Aerospace Engineering. Hindawi Publishing Corporation. 2012. doi:10.1155/2012/750642.

Bertrand Meyer. Object-Oriented Software Construction. Prentice Hall. Upper Saddle River, New Jersey, USA. ISBN 978-0136290490. 1993.

Robin Milner. Communication and Concurrency. Prentice Hall. Upper Saddle River, New Jersey, USA. ISBN 9780131150072. 1989.

Modelica - A Unified Object-Oriented Language for Physical Systems Modeling. Language Specification version 3.3. The Modelica Association. 2014. http://www.modelica.org.

James Noble, Antero Taivalsaari and Ivan Moore. Prototype-Based Programming: Concepts, Languages and Applications. Springer-Verlag. Berlin and Heidelberg, Germany. ISBN 978-9814021258. 1999.

Christos H. Papadimitriou. Computational Complexity. Addison Wesley. Boston, MA 02116, USA. ISBN 978-0201530827. 1994.

Tatiana Prosvirnova, Michel Batteux, Pierre-Antoine Brameret, Abraham Cherfi, Thomas Friedlhuber, Jean-Marc Roussel and Antoine Rauzy. The AltaRica 3.0 project for Model-Based Safety Assessment. Proceedings of 4th IFAC Workshop on Dependable Control of Discrete Systems, DCDS'2013. International Federation of Automatic Control. ISBN 978-3-902823-49-6. pp. 127–132. York, Great Britain. September, 2013.

Claudius Ptolemaeus. System Design, Modeling, and Simulation using Ptolemy II. Ptolemy.org. ISBN 978-1304421067. 2014. http://ptolemy.org/books/Systems.

Marvin Rausand and Arnljot Høyland. System Reliability Theory: Models, Statistical Methods, and Applications, 2nd Edition. Wiley-Blackwell. Hoboken, New Jersey, USA. ISBN 978-0471471332. January 2004.

Antoine Rauzy. Guarded Transition Systems: a new States/Events Formalism for Reliability Studies. Journal of Risk and Reliability. Professional Engineering Publishing. 222:4. pp. 495–505. 2008.doi:10.1243/1748006XJRR177.

James Rumbaugh, Ivar Jacobson and Grady Booch. The Unified Modeling Language Reference Manual. Addison Wesley. Boston, MA 02116, USA. ISBN 978-0321267979. 2005.

Bran Selic. A systematic approach to domain-specific language design using UML. In Object and Component-Oriented Real-Time Distributed Computing, May 2007. ISORC'07. 10th IEEE International Symposium on (pp. 2-9). IEEE.

Jean-Pierre Signoret. Dependability & safety modeling and calculation: Petri nets. Proceedings of 2nd IFAC Workshop on Dependable Control of Discrete Systems DCDS'09. IFAC Proceedings, Volume 42, Issue 5, pp 203-208. Bari, Italy, June 2009.

John Stark. Product Lifecycle Management: 21st Century Paradigm for Product Realisation (2nd ed). Springer London Ltd. London, England. ISBN 978-0857295453. August, 2011.

Leslie G. Valiant. The Complexity of Enumeration and Reliability Problems. SIAM Journal of Computing. SIAM. 8:3. pp. 410–421. 1979.

INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, fourth edition. David D. Walden, Garry J. Roedler, Kevin J. Forsberg, R. Douglas Hamelin and Thomas M. Shortell Ed. Wiley-Blackwell. Hoboken, NJ, USA. ISBN 978-1118999400. August 2015

Tim Weilkiens, Jesko Lamm, Stephan Roth and Markus Walker. Model-Based System Architecture. Wiley-Blackwell. Hoboken, New Jersey, USA. ISBN 978-1118893647. 2015.

Stephen White and Derek Miers. BPMN Modeling and Reference Guide: Understanding and Using BPMN. Future Strategies Inc. Lighthouse Point, FL, USA. ISBN 978-0977752720. 2008.

Niklaus Wirth. Algorithms + Data Structures = Programs. Prentice-Hall. Upper Saddle River, New Jersey 07458, USA. ISBN 978-0130224187. 1976.

Yevgeny Yaroker, Valeriya Perelman, and Dov Dori. "An OPM conceptual model-based executable simulation environment: Implementation and evaluation." Systems Engineering 16.4 (2013): 381-390.

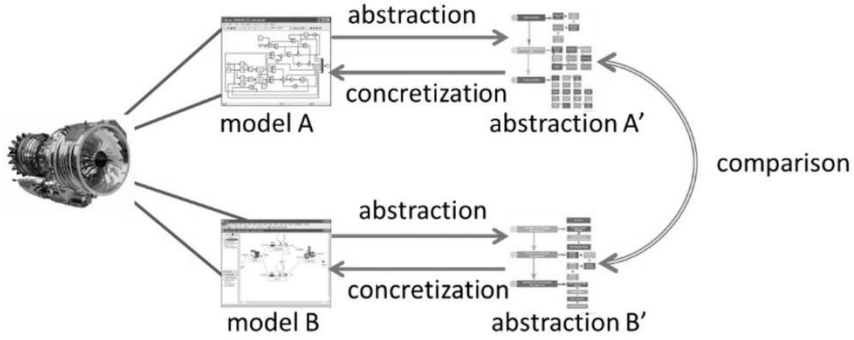

Figure 1. The model synchronization process

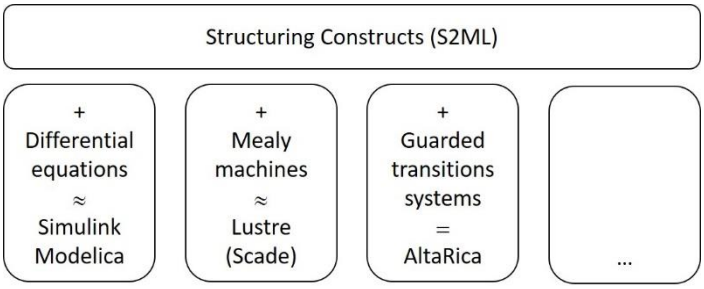

Figure 2. S2ML as a structuring paradigm for modeling languages

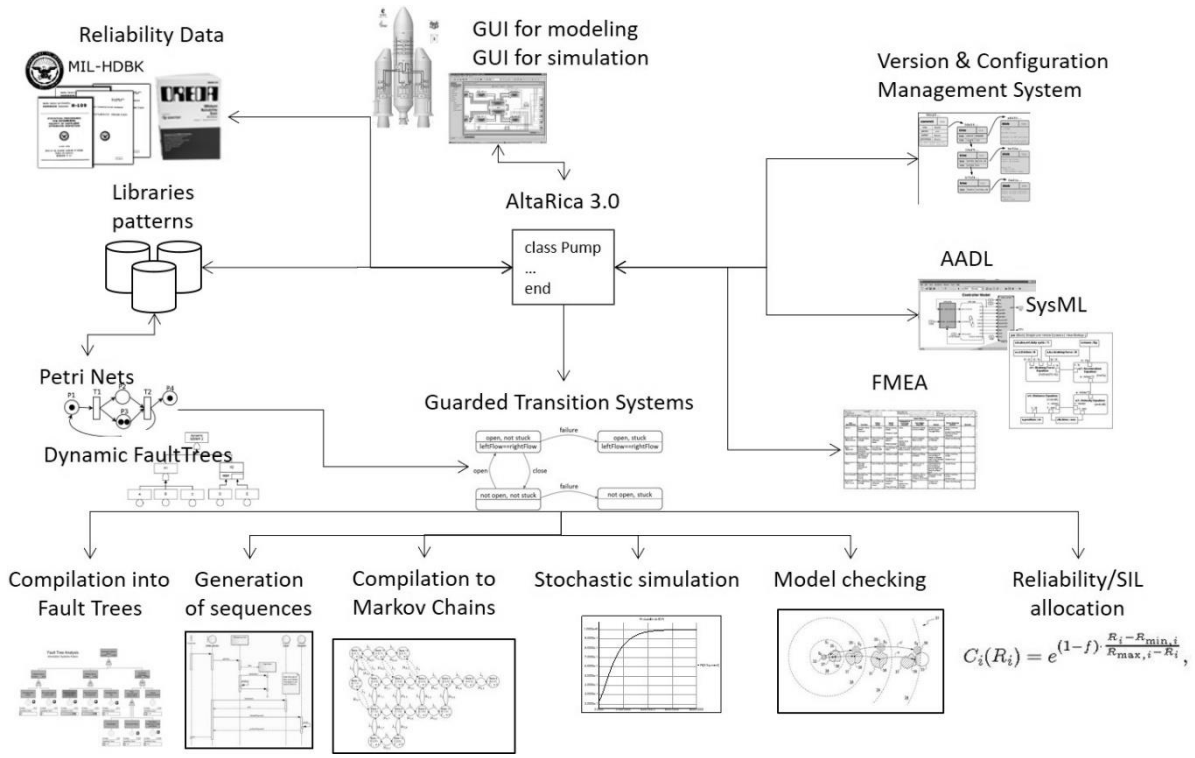

Figure 3. A snapshot of the AltaRica 3.0 project