Softw Syst Model (2013) 12:223–225
DOI 10.1007/s10270-013-0346-4

EDITORIAL

The evolution of modeling research challenges

Robert France · Bernhard Rumpe

Published online: 14 May 2013

In 2007 ICSE hosted a track called "Future of Software Development" (FOSD). We were invited to write and present a paper on the future of modeling for the track. The resulting paper [1] described the state of modeling research, identified some major challenges and proposed a research road map. Parts of this road map are currently being explored, and progress has been made in addressing some of the challenges we identified. However, there is still significant research "to be done" with respect to the challenges outlined in that paper. It is not our intent to discuss the progress the community has made with respect to the road map in this editorial (our apologies for deflating expectations in this regard; an editorial is simply not the place for such discussions). Rather, we would like to use this editorial to stimulate discussions around some of the challenges that have arisen since we wrote that paper.

Before we get into identifying specific challenges, let us first take a step back and ask "What is the grand challenge addressed by researchers in the SoSyM community?" The answer has typically been "To significantly reduce the time, cost, and effort required to develop complex software-intensive systems that meet stringent quality requirements through use of models that are fit-for-purpose." This is still the focus of many research programs, but we are also seeing a broadening of the grand challenge to include not only development problems, but problems that occur throughout the lifetime of a complex system, from conception to retirement. For example, in the FOSD paper we made reference to emerging work on the use of models to manage software at runtime (now referred to as models@run.time). The grand challenge is thus evolving to encompass a wider range of problems that occur in the life cycle of complex systems, as it should. It is interesting to note that in other disciplines the use of models to manage work on artifacts throughout their life cycles is receiving significant attention. For example, in the construction domain building information modeling (BIM), techniques and standards are being developed to support the management to built environments from conception, to architectural design, to onsite construction, to building occupancy and finally to building demolition. In this case models are the primary means for managing work throughout a building's lifetime.

Given this broader view of the role that models can play, we have identified the following research opportunities and challenges:

- Systems that integrate mechanical, software and electrical engineering subsystems (e.g., cyber-physical systems (CPS)) will become commonplace, and thus, there is a need for work on how the modeling approaches used by experts in these diverse domains can be integrated. This is a very challenging problem, and thus, there is a need for significant intellectual investment in developing integrated modeling languages and associated tooling. With respect to CPS, there are two core issues that require our attention as researchers:
  - Mechanical, electrical and software engineering rely on entirely different theories (e.g., theories based on continuous mathematics versus those based on event-based automata), and thus engineers in these domains use very different approaches. The challenge is to develop good bridges across these theories.

R. France (✉)
Colorado State University, Fort Collins, Colorado, USA
e-mail: france@cs.colostate.edu

B. Rumpe
RWTH Aachen University, Aachen, Germany
e-mail: Bernhard.Rumpe@sosym.org

  - Proper integration of two very different development processes is needed. The development processes should be decoupled to the extent possible, for example, if software is the dominating factor (and risk), software development should not be made dependent on the mechanical development process. Currently, this is sometimes not possible because software developers usually need the mechanical parts (physical or electrical) to exist before starting development. What can help in this respect is a front-loading process that disconnects software and mechanical development through the use of models of the mechanical systems to facilitate early simulations.

- Virtual construction and testing through simulation will become increasingly important. Simulating the context of a system or subsystem to understand its behavior is particularly important (where the context may include, or exclude, mechanical parts). This might require extensive understanding of the context, which can be facilitated by models.
- Tooling for advanced artifact management including information and dependency tracing, variabilitymodeling, advanced consistency checking and analysis will have to improve, possibly by orders of magnitude, to better support effective use of models across a system's lifetime.
- Models are more effective if they are properly integrated into a system's life cycle processes. In particular, system development processes have a high degree of potential innovation, and thus, these processes need to be adaptable to project, product and developer's needs. At the same time, they must also be controlled in a predictable manner. Explicit modeling of the processes, as is done for business processes, is needed if we want to tightly integrate processes with models.
- Adaptive systems that require new levels of security and safety are beginning to emerge. For example, dynamic adaptation or updating is necessary because customization by users demands not only system flexibility, but also frequent security updates. Apps from smartphones and software plugins are manifestations of this adaptability. Even their versioning is independent of the underlying technologies. Explicit use of models during runtime to allow customer adaptation could prove to be beneficial. Safety and security issues could call for modeling the possible bandwidth of customization through appropriate constraint specification languages that are inherently underspecified.
- Cloud systems will soon be integrated with non-cloud-based systems, e.g., for storage and handling of masses of sensor data of various sources (car, home, body, factory, energy, city, state and governance), and will have an impact on systems that connect to the Internet ("Internet of things"). There is thus a need to investigate ways in which cloud-based concerns can be addressed in models.
- Multicore may play increasingly important roles in embedded systems, e.g., to provide redundancy on chips (for safety reasons) and support reliable parallel computations at least of parts of the software. Can modeling help here, e.g., by explicitly orchestrating parallelization, redundancy and security levels?
- "Correctness by construction" will become even more feasible. A prominent example is the use of a deterministic, real-time bus for the implementation of communication, such that models that include time constraints can faithfully be mapped to the implementation and by construction communication scheduling fits the desired needs. A second example is the use of an executable DSL (or programming language) again with explicit time constraints combined with a restricted expressivity for loops and recursion for both data structure and functionality. This allows generators to predict worst-case computation times of the generated program and thus automated checking of reliability of specified timing constraints. Fortunately, these techniques help engineers tackle new problems, e.g., by means of the above-mentioned multicore and CPS technologies. Combinations of frameworks and hardware technology with models to customize them will certainly be of help here. We expect more techniques to solve problems by construction instead of postmortem analysis to come in the increasingly critical security and safety systems domain.
- Formal methods reloaded in lightweight forms provide "just enough" analysis to meet development needs in many situations. These techniques are typically model-based, e.g., Sat-Solvers and Model Checking, and they provide developers with feedback expressed in familiar languages. We anticipate that further advanced forms of automatic analysis and, in particular, constructive synthesis will come. Some of them are the following:

  - Automatic synthesis of complete models from views.
  - Automatic mappings of logical models to a distributed hardware architecture according to functionality needs and reliability.
  - Automated correctness checking of certain kinds of algorithms.
  - Generation of proof obligations along with execution of refactorings (behavior-preserving transformations) on models and ideally automated proofs.
  - Automated checking of consistency of a set of selected model elements to a configuration in a software product line.
  - Automated checking of conformance of behavior of a subclass/subcomponent implementation versus the super-specification it replaces.

  - Proof that decomposition of a more complex functionality into time-consuming subsystems in total still serves the desired quality of service.

- The community still lacks tool infrastructures that enable agile model-based development. These infrastructures will allow us to use DSLs in more agile ways. However, the challenging question is "Who is going to provide the tooling for these DSLs?"

In summary, there are still significant challenges that we need to address. We hope to see more submitted papers that go beyond the traditional modeling boundaries.

Reference

1. France, R., Rumpe, B.: Model-driven development of complex software: A research roadmap. In: Future of software engineering 2007 at ICSE, Minneapolis, pp. 37–54. IEEE (2007)