

# Model-Based Development and Formal Methods in the Railway Industry

Alessio Ferrari, CNR-ISTI

Alessandro Fantechi, Università di Firenze

Stefania Gnesi, CNR-ISTI

Gianluca Magnani, General Electric Transportation Systems, Florence

*// The transition from a code-based process to a model-based one isn't easy, particularly for companies that operate in the safety-critical sector. A railway signaling manufacturer adopted general-purpose, model-based tools aided by formal methods to develop its products, facing challenges and learning lessons along the way. //*



**GENERAL ELECTRIC TRANSPORTATION SYSTEMS** (GETS), Florence, Italy, is a medium-sized branch of a global railway signaling manufacturer.

Approximately 10 years ago, to address safety-critical industries' rising interest in formal methods,<sup>1</sup> the company started experimenting with formal


modeling and verification. To this end, it contacted experts from the local university to support some initial experiments.

The team evaluated several formal tools, but the developers preferred a semiformal tool suite—namely, Simulink/Stateflow.<sup>2</sup> The Simulink language uses a block notation to define continuous-time dynamic systems; the Stateflow notation is based on David Harel's statecharts<sup>3</sup> and supports the modeling and animation of event-based, discrete-time applications. The main drivers for this choice included the large amount of packages available in the tool suite—packages that developers could use throughout the development process—and the widespread knowledge about the tools found within the company.

Initially, the developers used the models designed through Simulink/Stateflow solely for requirements elicitation, but in 2007, the company wanted to explore using such models for code generation as well. One year later, this technology became part of the development process, but changing the development paradigm from code-based to model-based required additional changes in the verification process. The company adopted model-based testing and abstract interpretation, as well as language restrictions to reduce the tool suite's semiformal semantics to a formal semantics.<sup>4</sup>

The new model-based approach sped up development and allowed the company to handle more complex systems. As projects grew in size, they required new technologies that could rigorously handle system requirements. The company selected SysML, a unified modeling language for system development, to address this issue. After three years with SysML, the company established a formal development approach that integrates SysML and Simulink/Stateflow.

In this article, we describe the



challenges and lessons learned by the company throughout this 10-year experience.

## Challenges

Over the course of this experience, GETS faced several challenges that deserve some extra attention.

### Modeling Language Restriction

The code used in safety-critical systems must conform to specific safety standards, so companies typically use coding guidelines to avoid using improper constructs that might be harmful from a safety viewpoint. When a safety-critical company adopts modeling and autocoding, the generated code must conform to the same standards as handcrafted code. The adopted code generator—Simulink®<sup>5</sup>—induces a tight relation between the generated code and any modeling language constructs. Hence, the identification of a safe subset of the modeling language enables the production of code that complies with the guidelines and that can be successfully integrated with the existing code.

GETS did this by first defining an internal set of modeling guidelines for Simulink/Stateflow—specifically, these guidelines were practical recommendations on language construct usage. The idea was that any model-generated C code following the guidelines would comply with the company's coding standard.

The company based the initial guidelines on a code analysis generated from a model previously designed for requirement elicitation. Because this preliminary set of guidelines had the limit of being derived from a specific model, it could lack generality, so in the projects that followed, GETS extended it with other recommendations borrowed from the automotive domain.<sup>5</sup>

To ease formal analysis, the company decided to complete the modeling style guidelines by restricting the

Stateflow language to a semantically unambiguous set. To this end, it used studies that focused on translating a subset of Stateflow into the Lustre formal language.<sup>6</sup> The company's current models are therefore independent from the simulation engine, a choice that opened the door to formal verification.<sup>4</sup>

### Generated Code Correctness

Safety-critical norms, such as CENELEC EN 50128, the European standard for railway software, ask for a certified or proven-in-use translator. In the absence of such a tool, like in the case of available code generators for Simulink/Stateflow, a strategy must be defined to ensure that code behavior fully complies to model behavior, and that no additional improper functions are added during the code synthesis phase. The objective is to perform verification activities at the abstract model's level, minimizing or automating any operations on the code.

GETS adopted a model-based testing approach called translation validation,<sup>7</sup> completed by static analysis via abstract interpretation.<sup>8</sup> In translation validation, you execute test scenarios based on functional objectives at the

this overall approach suitable for bypassing the tool qualification required in current safety regulations. (Railway norms aren't as specific about tool qualification as, say, avionics norms are,<sup>10</sup> so companies in the railway sector must agree on possible strategies with certification authorities.)

### Multiple Formalisms

Safety-critical systems are large, complex platforms with several interacting units and architectural layers. To manage such complexity, their development is based on multiple levels of abstraction, a setup that requires different models with different granularities. Indeed, a model used for code generation is hardly usable for reasoning at the system design level. Simulink/Stateflow don't support a flexible hierarchical development approach, so system designers must adopt other modeling languages that can express the higher abstractions that the process inherently requires.

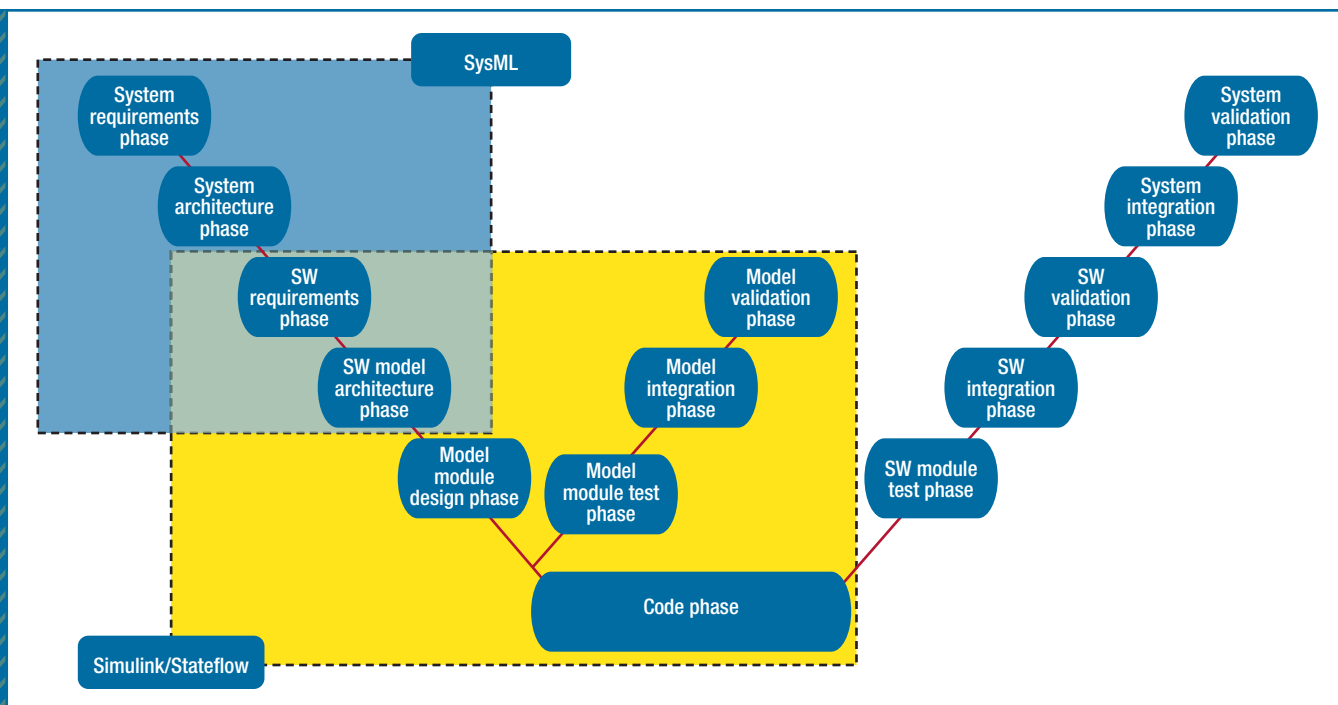
GETS addressed this issue by adopting SysML ([www.sysml.org](http://www.sysml.org)). After an initial experience with the TOPCASED tool for SysML ([www.topcased.org](http://www.topcased.org)),

When a company adopts autocoding, the generated code must conform to the same standards of handcrafted code.

model level. Then, you repeat the same tests on the generated code, checking that the model's outputs and the corresponding code are consistent. To ensure runtime error freedom, the company uses the Polyspace tool to perform abstract interpretation.<sup>9</sup> This final step verifies a program's correctness on an overapproximation of the range of program variables.

Certification authorities considered

which at that time wasn't considered mature enough for industrial usage, the company adopted the Magic Draw platform ([www.nomagic.com/products/magicdraw.html](http://www.nomagic.com/products/magicdraw.html)). Early GETS projects that used code generation didn't use SysML support—the need came when the systems the company produced started to radically increase in terms of complexity, for example, when the LOC exceeded 100,000.



**FIGURE 1.** Overview of the model-based development process that GETS adopted. The left side of the V shape includes the design activities, while the right side includes the verification activities. The small right branch on the left side of the V includes the verification activities performed at the model level.

SysML's current role is as follows. Immediately after requirement elicitation, requirements appear as unstructured Post-It notes on the requirements manager's whiteboard. The manager identifies high-level system requirements among this initial set and expresses them in the form of SysML requirement diagrams. These diagrams allow the requirements manager to specify hierarchical relationships and dependencies among single requirements, replacing the chaotic Post-It view with a structured graph-like model.

Next, the design team uses block diagrams to specify the interfaces of the system modules that are supposed to implement the requirements. The team adopts an approach based on decomposition, which allows specializations of each module into submodules to realize the actual implementation. Each

module has some high-level requirements apportioned to it, and if needed, the design team can refine requirements into lower-level requirements when modules are specialized.

SysML models are structured as packages in a single root model, where each package corresponds to a phase of the V-based development process prescribed by the CENELEC norm for railway safety-critical systems. Therefore, a well-defined mapping between process phases and the diagrams are used in each phase.

SysML could also be used to define the actual implementation's behavior and to generate code. However, modeling and simulation at the behavioral level is much faster and more flexible with Simulink/Stateflow, and the generated code has higher quality. Therefore, SysML's role ends at the software architecture level.

## Process Integration

Companies develop products via processes, which define a framework made of tasks, artifacts, and people. The introduction of new technologies in an established process requires adjustments to the process structure, which has to maintain its coherence through these changes. This is particularly true in the case of safety-critical companies, whose products must be validated according to normative prescriptions. Hence, a sound process must be defined to integrate modeling and code generation within the existing framework.

As Figure 1 shows, GETS defined an enhanced V-based process that embeds two verification branches: one for the activities performed on the models, and the other for the tasks concerning source code and the system. In the figure, we highlight the parts that strictly concern software development (based

on Simulink/Stateflow modeling) and the parts related to system development (based on SysML modeling). The two process fragments overlap in the SW Requirements phase and in the SW Model Architecture phase. Indeed, software requirements are expressed in SysML, as well as in the software architecture. An equivalent architecture is expressed through Simulink in the form of interacting blocks, which are the model's functional modules—the components. We can manually trace SysML requirements to the Simulink model. The Model Module Design phase refines the Simulink blocks into Stateflow statecharts.

Note that this process is somehow adaptable to both manual coding and to autocoding. After completing the SysML modeling activities, you can decide to adopt either handcrafted code or Simulink/Stateflow modeling to develop the application. Indeed, in some applications—for example, firmware, systems with limited software, or platforms with strong dependencies from legacy code—the code generation technology is considered inconvenient, so developers use handcrafted code.

## Lessons Learned

Facing all the challenges in model-based development allowed GETS to learn some important lessons.

### Abstraction

Models let you work at a higher level of abstraction, and they can be manipulated more easily than code. The company experienced the actual relevance of this statement in the transition from code-based to model-based testing. The model-based testing approach it adopted allowed developers to define behavioral test scenarios at the component level without disrupting the model structure. This approach would have been impracticable on hand-crafted code. Indeed,

with handcrafted code, it's common to perform tests on single functions, but it's much more difficult to identify the functions that contribute to a software component's behavior. With models, you build a system in terms of its components. Therefore, component identification and testing happens in a natural way.

GETS also learned that abstraction is a delicate concept that must be carefully handled, with the proper degree of abstraction clearly identified for an artifact to be useful. For example, in their

With strictly defined modeling guidelines, you can look at the generated code as if it were written by the same programmer.

initial experience with SysML, designers adopted natural language requirements throughout the process until they reached the lowest level of model detail. At that point, their content was basically equivalent to the Simulink/Stateflow models. Such requirements' level of abstraction had to be raised because they appeared to be redundant: any slight modification to the models would have implied a modification to the requirements.

### Expressiveness

Graphical models are closer to natural language requirements, yet they're also an unambiguous way of exchanging or passing artifacts among developers. The GETS team experienced this observation first hand when the project passed from its initial developer to another developer within a month and without much support.<sup>4</sup>

Up to that point, if someone in the company was a piece of software's father, he would have remained the one and only repository of knowledge

about that software. This is a common problem in many small- and average-sized companies that negatively affects both the company itself, which has to rely on a single person to modify and reuse its core software, and the developer, who normally wants to extend his competencies beyond his initial fragments of code.

### Cohesion and Decoupling

Automatically generated software is composed of modules with higher internal cohesion and better decoupling

with respect to manual coding. Interfaces among functionalities are based solely on data, and the control flow is simplified because there's no cross-calling among different modules. Decoupling and well-defined interfaces help ease the outsourcing of the modeling activity, which is a relevant aspect in the development of products that have to tackle time-to-market issues. The company acknowledged these advantages in the course of its development process.

Structured development gives developers greater control over components and ultimately leads to software with fewer bugs. At GETS, developers experienced this when the number of bugs found through verification dropped from 10 bugs to three per module after the company introduced a rigorous model-based process.<sup>4</sup>

### Uniformity

Generated code has a repetitive structure that facilitates automated verification activities. When you have strictly defined

modeling guidelines, you can look at the generated code as if it were consistently written by the same programmer. Thus, any code analysis task can be tailored on an artificial programmer's design habits. The abstract interpretation procedure adopted to reveal runtime errors worked only on the generated code because systematic analysis on handcrafted code was made harder by its variable structure and programming style.

ually define the traceability links, automatically generating the traceability matrixes. In a traditional process, the developer manually edits traceability matrixes without any tool support, leading to maintainability issues.

At GETS, customer-issued change requests normally involve system-level requirements. The tool support available in the company's model-based approach allows changes to be

When passing from traditional code unit testing to model-based testing, verification costs fell approximately 70 percent.

SysML also guarantees uniformity at the process level. The use of a unified modeling language—and a single tool—in most of the development phases eases the development in all of the activities that involve interfaces among phases. Indeed, in a V-based process, a phase's output artifact is the input artifact for the following phase. The use of SysML makes this handover more rigorous.

### Traceability

Software modules are directly traceable to their corresponding blocks in the specification modeled with Simulink/Stateflow. Traceability is a relevant issue in the development of safety-critical systems because any error must be traced back to the process task or artifact defect that produced it. With the support of Simulink/Stateflow, GETS introduced a structured development approach that lets developers define navigable links between single code statements and requirements.

At the SysML level, traceability involves the links between requirements diagrams and related SysML diagrams. Simple drag-and-drop operations man-

traced from such requirements to the module-level requirements and the corresponding models. Therefore, both developers and the requirements managers have a complete view of a change request's impact. Contrast this with the traditional process, in which someone would have to inspect the traceability matrix and check for artifacts affected by the change request, an activity that can be rather time-consuming and error-prone (unless supported by automated tools).

Automatic traceability support among SysML and Simulink/Stateflow models is still an open issue because no tool currently implements such a feature.

### Documentation

For safety-critical systems, the official documentation associated with each process phase and artifact is as important as the actual system. Process certification is essentially based on an external authority's inspection of such documentation. It's therefore important to have documentation that's formal, expressive, and up to date on product status.

In the process that GETS currently uses, both SysML and Simulink/Stateflow models provide documentation for process artifacts. Simulink/Stateflow models with proper comments automatically generate software documentation, thus keeping documentation and software totally aligned. In addition, SysML diagrams are integrated into the manually edited documentation. Documents can be automatically generated from SysML as HTML pages, but certification authorities typically require paper-like documents focused on text, rather than navigable HTML documents. The main reason is that the certification authority normally enters at the end of the development process to validate compliance with standards and wants to analyze the process as a sequential history—with a paper trail—not as an interwoven graph of HTML pages.

The integration of SysML models into the documents does pose maintainability issues. Indeed, if the model changes, that change isn't automatically reflected in the documentation. However, the one-to-one correspondence between SysML packages and process phases—and the associated documents—eases the effort of manually updating the documentation. Furthermore, the traceability links between models in different packages help maintain the cross dependencies among documents. When a model changes, its package clearly identifies the document that must be modified as well. Anyone with access rights can follow the traceability links and retrieve the other models affected by the change. Such models belong to packages with associated documents, so the link among models indirectly creates a relationship among those documents, and the overall SysML model becomes a sort of navigable index for the process documentation.



## Verification Cost

The introduction of a new development process at GETS reduced some of the costs associated with verification activities, while ensuring greater confidence in product safety. When passing from traditional code unit testing based on structural coverage objectives to testing based on functional objectives aided by abstract interpretation, verification costs fell approximately 70 percent. The new approach was comparable to the previous one in terms of compliance with CENELEC EN 50128 requirements on verification, but the results were much more cost-effective.<sup>9</sup>

Although GETS has achieved consistent cost improvements, manual test definition still bottlenecks the process, requiring approximately 60 to 70 percent of the whole unit-level verification cost. Preliminary experiments with formal verification applied at the unit level demonstrate that this technology might considerably reduce verification costs for most requirements. Indeed, recent experiments with formal verification via Simulink Design Verifier have shown that verification cost can further drop by 50 to 66 percent.<sup>4</sup>

## Complexity

The main drawback of introducing automated code generation is the resulting software's size and overall complexity. Although these aspects don't complicate verification activities, they pose challenges from the performance viewpoint.

Real-time constraints for railway signaling systems aren't as demanding as they are for other kinds of embedded systems, since the typical required response times are in the range of hundreds of milliseconds. But they're still reactive systems that need to activate failure recovery procedures in a brief amount of time to reach a safe state, should a failure occur. Execution time influences reaction time. In the first

experiments with automated code generation at GETS, this execution time took four times longer compared to the time required to execute the corresponding handcrafted code. So as not to abandon the advantages of auto-coding, a hardware upgrade solved this timing discrepancy.

But to design new, more complex systems, this issue must be taken into account when defining the hardware architecture. The hardware designer has to consider that the code is both larger in size and less flexible in terms of source-level optimization (recall that compiler-level optimizations aren't recommended for safety-critical systems): when designing the platform, you must plan for a larger amount of memory if you want to use automatic code generation.

## Knowledge Transfer

From the GETS effort's outset, one research assistant from the university who operated within the company was fully focused on the technology to be introduced and an internal development team put that research into practice on real projects after the ex-

teams or even entire research divisions, but medium-sized companies often have to use the same personnel both to keep the organization on track for market needs and to take care of daily software development. We argue that the research management model adopted for GETS can be adapted to other medium-sized companies with comparable results.

**T**he people behind GETS were able to understand the benefits of a model-based process aided by formal methods thanks to the initial enthusiasm associated with automated code generation. Such technology showed its potential in a few months, and its adoption was relatively straightforward. After that, a butterfly effect occurred that brought forward the easy adoption of other techniques, such as model-based testing, abstract interpretation, and system modeling with SysML.

Formal verification isn't part of the GETS development process yet. But we've observed that formal verification with model checking is often the focus

When designing the platform, you must plan for a larger amount of memory if you want to use automatic code generation.

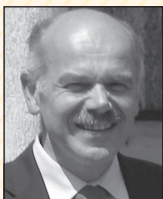
ploratory studies indicated success was possible.

The results obtained during this experience wouldn't have been possible via intermittent collaborations alone. On the other hand, to separate the research effort from the time-to-market issues, the research assistant's independence from the development team had to be preserved. Large companies can profit from dedicated internal research

of a company's first experiments with something more formal, especially in the safety-critical systems domain. In most cases, those companies don't go much further than these initial experiments, notwithstanding the achieved evidence of lower verification costs. Indeed, the adoption of formal verification without intermediate steps isn't common: the difficulties associated with the steep learning curve required



**ALESSIO FERRARI** is a researcher at CNR-ISTI (Consiglio Nazionale delle Ricerche—Istituto di Scienza e Tecnologia dell'Informazione, Pisa). His research interests are primarily requirements engineering and formal modeling. Ferrari received a PhD in computer engineering from the University of Florence. Contact him at [alessio.ferrari@isti.cnr.it](mailto:alessio.ferrari@isti.cnr.it).



**ALESSANDRO FANTECHI** is full professor at the University of Florence, where he's an active researcher in the areas of formal development and verification of safety-critical systems, with a particular emphasis on railway signaling systems. Contact him at [fantechi@dsi.unifi.it](mailto:fantechi@dsi.unifi.it).



**STEFANIA GNESI** is director of research at CNR-ISTI (Consiglio Nazionale delle Ricerche—Istituto di Scienza e Tecnologia dell'Informazione, Pisa) and head of its Formal Methods and Tool group. Her research interests include development of new logics for the formal specification and verification of concurrent systems and the application of model-checking techniques. Contact her at [stefania.gnesi@isti.cnr.it](mailto:stefania.gnesi@isti.cnr.it).



**GIANLUCA MAGNANI** is a software engineer at General Electric Transportation Systems, Florence, signaling division. His technical interests concern the application of SysML modeling to the development of railway systems. Contact him at [gianluca.magnani@ge.com](mailto:gianluca.magnani@ge.com).


Context," *Proc. 1st NASA Formal Methods Symp.*, NASA, 2009, pp. 166–170.

6. N. Scaife et al., "Defining and Translating a 'Safe' Subset of Simulink/Stateflow into Lustre," *Proc. 4th ACM Int'l Conf. Embedded Software*, ACM, 2004, pp. 259–268.
7. M. Conrad, "Testing-Based Translation Validation of Generated Code in the Context of IEC 61508," *Formal Methods in System Design*, vol. 35, no. 3, 2009, pp. 340–389.
8. P. Cousot and R. Cousot, "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints," *Proc. 4th ACM SIGACT-SIGPLAN Symp. Principles of Programming Languages*, ACM, 1977, pp. 238–252.
9. A. Ferrari et al., "Adoption of Model-Based Testing and Abstract Interpretation by a Railway Signaling Manufacturer," *Int'l J. Embedded and Real-Time Communication Systems*, vol. 2, no. 2, 2011, pp. 42–61.
10. A.J. Kornecki and J. Zalewski, "Certification of Software for Real-Time Safety-Critical Systems: State of the Art," *Innovations in Systems and Software Eng.*, vol. 5, no. 2, 2009, pp. 149–161.



Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.

by formal methods often tend to discourage industrial practitioners and managers, who need to see the evidence of productivity gains within a short time frame.

The experience we report here shows that it might be more effective to start with less formal tasks—automated code generation—and then later adopt more formal tasks, such as verification, when the company has matured into full awareness of the actual benefits of "being formal." 

## References

1. J. Woodcock et al., "Formal Methods: Practice and Experience," *ACM Computing Surveys*, vol. 41, no. 4, pp. 19:1–19:36.
2. S. Bacherini et al., "A Story about Formal Methods Adoption by a Railway Signaling Manufacturer," *Proc. 14th Int'l Symp. Formal Methods*, LNCS 4085, Springer, 2006, pp. 179–189.
3. D. Harel, "Statecharts: A Visual Formalism for Complex Systems," *Science Computer Programming*, vol. 8, no. 3, 1987, pp. 231–274.
4. A. Ferrari et al., "The Metrô Rio Case Study," to be published in *Science Computer Programming*, 2013; doi: 10.1016/j.scico.2012.04.003.
5. A. Ferrari et al., "Modeling Guidelines for Code Generation in the Railway Signaling

# IEEE Software

FIND US ON  
**FACEBOOK  
& TWITTER!**

[facebook.com/  
ieeesoftware](https://facebook.com/ieeesoftware)

[twitter.com/  
ieeesoftware](https://twitter.com/ieeesoftware)