A Story About Formal Methods Adoption by a Railway Signaling Manufacturer

Stefano Bacherini[2], Alessandro Fantechi[1],
Matteo Tempestini[2], and Niccolò Zingoni[2]

[1] Università degli Studi di Firenze, Dipartimento di Sistemi e Informatica
[2] General Electric Transportation Systems
fantechi@dsi.unifi.it, Stefano.Bacherini@trans.ge.com,
Matteo.Tempestini@ge.com, Niccolo.Zingoni@ge.com

Abstract. This paper reports the story of the introduction of formal methods in the development process of a railway signaling manufacturer. The first difficulty for a company is due to the many different formal methods proposals around; we show how this difficulty has been addressed and how the choice of a reference formal specification notation and of the related tools has been driven by many external factors related to the specific application domain, to the company policies, to european regulations. Cooperation with University has been fundamental in this process, which is now at the stage in which internal acceptance of the chosen formalisms and tools is established.

1 Introduction

Railway signaling has been often considered as one of the most successful areas for the industrial application of formal methods, reporting many success stories.

There are two main reasons for this success. On the one hand, railway signaling has always generated the interest of formal methods researchers: its safety requirements with the implied need to avoid any kind of errors, the discrete nature of typical control computations and the absence of very hard real-time constraints, have made it a promising application field, in which the different formal specification and verification techniques can be conveniently applied. On the other hand, railways have always had a very strong safety culture, based on simple fail-safe principles. In electromechanical equipments, used in most signaling systems before the introduction of computers, gravity was used to bring a system to the fail-safe state (e.g. all signals to red) in any occurrence of a critical event. On the other hand, the impossibility of predicting in general the effects of the occurrence of faults in computer-based equipment, has long delayed the acceptance of computer-controlled signaling equipment by railway companies. The employment of very stable technology and the quest for the highest possible guarantees have been key aspects for the adoption of computer-controlled equipment in railway applications. Formal proof, or verification, of safety has been therefore seen as a necessity.

J. Misra, T. Nipkow, and E. Sekerinski (Eds.): FM 2006, LNCS 4085, pp. 179–189, 2006.

In this paper, we offer some insight into the actual industrial usage of formal methods in this field, describing the experience of a railway signalling company, namely the railway signaling division of General Electric Transportation Systems (GETS), confronted with the need to adopt formal specification and development techniques in the development cycle of safety-related equipment.

We will see how the choice of which formalism and tool to adopt inside the company development cycle has been influenced by several factors. The choice is indeed not easy: there are many notations, methods, and (prototypal) tools originating from the academia, which however lack industrial strength in terms of tool stability, documentation and user support. On the other hand, there are very few technically sound methods and tools coming from industry. Indeed, the combination of several external factors, such as specific characteristics of the application domain, the general company policies, the european safety regulations, and the trends over the last years of the main actors of the application domain (namely, railway operators, railway infrastructure owners, railway signalling industries), has actually facilitated the choice, narrowing the range of preferred formalisms and tools.

In section 2, the EN50128 guidelines by the European Committee for Electrotechnical Standardization regarding the development of software for railway signaling are discussed, with regards to the adoption of formal specification techniques. Section 3 reports more information of the recent evolution of the context in which GETS operates. Section 4 discusses some first experiments that have been conducted in cooperation with academy in order to correctly address the issue. Section 5 discusses the choice made by GETS to adopt Stateflow of the Matlab environment as the reference formalism and tool.

2 CENELEC Guidelines

The EN50128 guidelines [6], issued by the European Committee for Electrotechnical Standardization (CENELEC), address the development of "Software for Railway Control and Protection Systems", and constitute the main reference for railway signaling equipment manufacturers in Europe, with their use spreading to the other continents and to other sectors of the railway (and other safety-related) industry.

The EN50128 document is part of a series of documents regarding the safety of railway control and protection systems, in which the key concept of Software Safety Integrity Level (SWSIL) is defined. One of the first steps indicated by these guidelines in the development of a system is to define a Safety Integrity Level (SIL) for each of its components, on the basis of the level of risk associated, by means of a risk assessment process. Assigning different SILs to different components helps to concentrate the efforts (and therefore the production costs) on the critical components. The SILs range from 4 (very high), to 1 (low), and 0 (not safety-related).

The EN50128 guidelines dictate neither a precise development methodology for software, nor any particular programming technique, but they classify a wide

range of commonly adopted techniques in terms of a rating (from "Forbidden" to "Highly Recommended" and "Mandatory") with respect to the established SIL of the component. Formal methods (in particular CCS, CSP, HOL, LOTOS, OBJ, Temporal Logic, VDM, Z and B are cited as examples) are rated as highly recommended for the specification of systems/components with the higher levels of SIL. Formal proof is also highly recommended as a verification activity. Anyway, formal techniques are not classified as mandatory, since alternative, more traditional techniques are also accepted. We should notice however that this is the first time (the first edition of EN50128 dates back to 1994) that a strong indication about the usage of formal methods appears in standard guidelines.

Indeed, despite CENELEC directives and success stories, formal methods have not permeated the whole railway signaling industries, where much software is still written in traditional ways. This is due to the investments needed to build up a formal methods culture, and to the high costs of commercial support tools. Moreover, equipment can conform to CENELEC without applying formal methods. Verification by thorough testing can be claimed compliant to EN50128. But relying only on traditional testing shifts an enormous effort (usually more than 50% of the total development effort) on the shoulders of the testing department. This becomes a risk for a company that is more and more required by the market to be CENELEC compliant. Indeed, since testing activities are performed in late phases of product life cycle, bugs detection and fixing activities imply reviews of early phases with, consequently, high costs and stretched time. The only solution is to shift back the effort to the design team, by introducing formal methods in the specification and design phases. This is why the railway signalling division of General Electric Transportation Systems (GETS) has taken the decision to adopt formal methods in the development cycle of SIL 4 equipments.

3 The Context

Historically, the ancestors of GETS, similarly to many railway industries all over Europe, had a strict collaboration with Italian State railways. The design of new equipment were carried on as a single team between the railway operator and the equipment providers. The evolution and liberalization of the European market has clearly separated the roles of the operator, which issues equipment specifications, and providers, which implement the specification, but also needs to produce addressing the global market. Hence the specification themselves have gained more importance, in particular with respect to the possibility to have unambiguous, formally specified, specifications.

Indeed, this new trend has become evident inside a joint project between Politecnico di Milano and Italian State Railway FS, Infrastructure Department (which recently became Rete Ferroviaria Italiana S.p.A.). The purpose of the project was to define procedures and rules for managing software procurement for safety-critical signaling equipment [8]. One of the aims of the project was to select and classify formal methods that were sufficiently mature for industrial usage, were supported by automated tools, and were likely to gain acceptance by average engineers, both in the railway and computer technology domains.

One of the indications emerging from this project was that Statecharts [11] and SDL [5] were perceived as the most suitable formalisms according to various parameters, including those cited above.

Another event to be noted is the launching of the Eurointerlocking project by a consortium among the main European infrastructure companies, with the aim of developing a standard interlocking system at a European level, with the purpose to reduce costs, by means of use of standardized components and standardized interlocking rules. Inside Eurointerlocking, we can cite the interesting EIFFRA (Euro-Interlocking Formalised Functional Requirements Approach) activity [13], where, together with an attention to textual requirements, and requirement management tools, such as Telelogic DOORS, model-based requirements are addressed, by proposing UML [19] state diagrams and Statecharts to describe the behaviour, and OCL [20] to describe properties of the interlocking systems.

We can also mention another experience inside Eurointerlocking, by SNCF-RFF, which has modeled their national (relay based) interlocking logic principles using Statecharts and Statemate [15].

In conclusion, the trend that we can note within the railway signaling field is towards state machine - based formalisms, such as SDL and Statecharts, the latter in their various dialects (UML, Statemate, etc...). The graphical syntax and the availability of commercial support tools are considered as positive discriminant factors.

4 The Experiments

GETS has addressed the problem of introducing formal methods in its development process by contacting experts at University of Florence. Collaboration with the Faculty of Engineering of the University of Florence was indeed a tradition, already established on mechanics and electronics. Facing the problem of addressing software certification along CENELEC guidelines, and given that exhaustive testing, possible on the small software systems of the beginnings, was no more viable, GETS has asked to the University experts to establish a common project of technology transfer about formal methods.

The project has followed the indications of the already cited RFI procurements guidelines [8] . In particular some first experiments, have been attempted, modeling in SDL some already produced systems [1,7], with specific attention to the issues of validation coverage [2] and of code generation [3].

Though modelling with SDL allowed a formal methods culture to start to consolidate inside GETS, it was not felt that this was the definitive choice, both for some difficulties emerged with the language itself (the asynchronous nature of communication, inherited by the original mission of SDL to describe communication protocols, and some other characteristics of the messages management have been perceived as difficulties by the designers) and for the not clear future of the language and its support tools, which were going to be merged into the UML 2.0 world.

Following the trends that have been noted in the international railway signalling arena, mainly inside the Eurointerlocking effort, later experiments have switched to Statecharts, at first in their Statemate dialect. At that time, it seemed that also the major GETS client, namely RFI, was inclined to use Statemate Statecharts for drawing their systems specifications. The experiments consisted in the formal specification of a railway signalling system for the objects detection in level crossing areas. The system, named PAI-PL, was developed and homologated SIL 4 by GETS using a customer paper based requirements specification. During the experiment, that specification was translated in a Statemate model and analysed using the related model checker tools. The results showed both that formal methods could be used in specification activities and that could also permit to find mistakes or ambiguous aspects in requirements. Nevertheless after some time, and a quite dense dialog with RFI, it has appeared that no clear decision had already been taken, and that the railway infrastructure company was not ready to abandon its traditional way of developing specifications in favour of formal statecharts specification. This is also because, GETS apart, most of the others signalling companies did not replied positively to formal methods quest by RFI.

The choice of the formal method and support tools were now back in the hands of GETS. The experience acquired on Statecharts indicated that a natural candidate tool to acquire was ILogix Statemate tool [12]. At this point, however, other factors, mostly related to costs, have been taken in consideration. We should recall that the quest for the adoption of a formal method for the specification of systems were coming mainly from the V&V department. Inside the company the high investment needed to acquire the tools would have been therefore not shared among all the departments. Design departments were more keen to adopt instead more flexible tools that could aid several aspects of the design, and not only the specification by statecharts of the "discrete" behaviour of a system.

5 The Choice of Stateflow

An attractive competitor appeared on the scene, in the form of the Stateflow component [16] of the Matlab modeling environment [17] . Indeed Stateflow statecharts share most of the characteristics of other dialects of statecharts, but their semantics have some restrictions, especially in comparison with that described in [11]. Indeed, Statemate semantics is based on three different views (behavioural, functional and structural) of a system, which are related to three corresponding charts (statecharts, activity-charts and module-charts) in a model. Instead, Stateflow semantics permits to represent only the behavioural view, while there is no special formalism to represent the other ones. These and the interactions with the behavioural view can be partially and sometimes with difficulty made up using Simulink formalism. It is in particular a very hard task to develop a model compound by nested functional and behavioural blocks.

From the behavioural point of view, the most peculiar characteristic of Stateflow semantics is the use of the "clockwise rule" to evaluate the transitions from the same state. If no user-defined priority rule is given, transitions from the same state are ordered first on the form of their guards (transitions guarded by an event are evaluated before those guarded only by a condition, and unguarded transitions come last): remaining unordered transitions from the state (i.e. sharing the same form of the guards) are ordered by their graphical appearance: the first transition is the one whose arc starts closest to the upper left corner of the source state, and the others follow clockwise. We refer to [10] for a complete formal description of Stateflow semantics. The clockwise rule has two main implications:

- the Stateflow semantics is completely deterministic, since outgoing transitions are always deterministically ordered. The problem is that determinism in some intricate cases (e.g. involving overlapping boolean conditions) cannot be immediately perceived by the user, who naturally considers them as non-deterministic. On the other hand, while Statemate or other statecharts tools are able to identify (statically or by model-checking) possible sources of nondeterminism, this is not possible in Stateflow, where such critical situations perceived by the user as nondeterministic, are actually resolved only at simulation time.
- porting specifications from Statemate or UML Statecharts to Stateflow and vice-versa (by simple manual redrawing or by some import/export tool through a XML/XMI format) is not immediate, and care should be taken that the intended meaning of the specifications is preserved during the porting.

We can observe however that the delays of the major client in adopting formal specifications, referred in the previous sections, have moved the focus away from waiting for specifications from the client, towards the proprietary production and use of specifications, for a later sharing with and approval by the client. Hence, the issue of porting has no more been considered crucial for GETS.

The semantic disadvantages of Stateflow had their counterpart in the possibility offered by Matlab, and by lots of tools compatible with Matlab and Simulink environment, of modelling and simulating several aspects of a system: this possibility was felt as very attractive by many groups inside the design department. Moreover, Matlab was already widely used at corporate level, so that knowledge about it could be easily retrieved over the corporation intranet. Again, several modeling experiments were started, which allowed a better knowledge of the peculiar characteristics of Stateflow. In Figure 1, the main statechart extracted by the model of the already cited PAI-PL system is shown; actually, the represented states are phases of the execution of the system, defined in conformance with the customer requirement specification, and are hierarchically subdivided in lower level statecharts.

The experiments have shown the capability of Stateflow to formally describe the behaviour of a system, allowing simulation and integration in a complete model of the system. The experiments have actually revealed the semantic

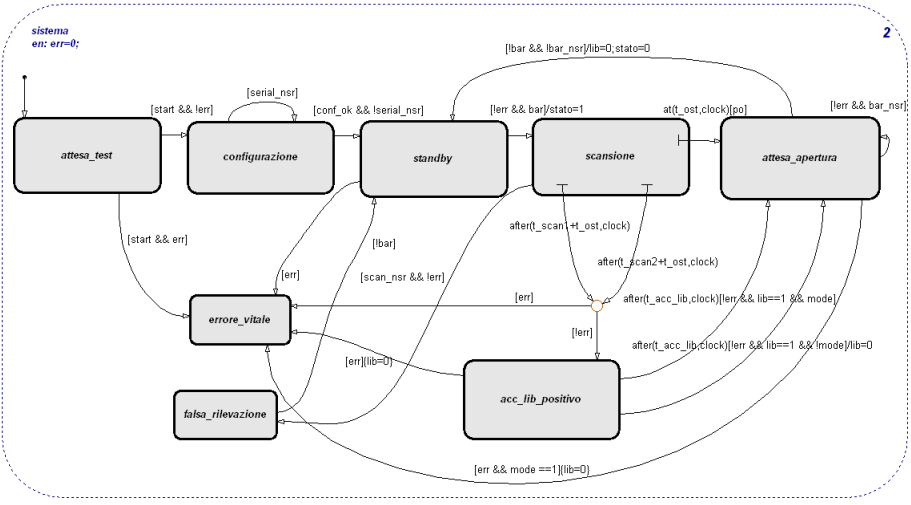

Fig. 1. The Stateflow model of PAI-PL phases

problems that plague Stateflow, but the expected advantages were valued as positively counterbalancing the negative aspects; hence, a decision to adopt Stateflow was taken.

For the first time, Stateflow was actually adopted in the design of a new system, while previous experiments were mainly playing with the specifications of systems already in production. The tool was successfully used to formally define the high levels requirements of this new system and to share the specification with the customer. A more detailed model was developed to define the software requirements of the system and used to design and write down some functions of the application code. Moreover the model was used to carry out functional system tests and to identify corner-case scenarios.

Currently, the use of Matlab has still to become widespread inside the company, and this is planned to occur incrementally on a project by project base. The collaboration with University is still active, and is now focused over the added value that can be obtained from Stateflow specifications, in terms of early validation (through model checking), test case generation and code generation These are listed in order of priority for GETS: namely, first guaranteeing an early validation of Stateflow models, then guaranteeing the consistency between developed code and the model by means of high coverage testing, and last investigating the possibility of automating the code generation from the Stateflow model.

5.1 Model Checking over Stateflow Specifications

Model checking the Stateflow specifications in search of inaccuracies or to guarantee the exhaustiveness of their verification and the compliance with customer

requirements is the first goal for our research activity. Actually, several experiments have already been carried on, using SF2SMV [14], a conversion tool developed at Carnegie Mellon University under a contract of the locomotive branch of GETS. The tool allows to convert a Stateflow specification so that it can be given as input to the popular SMV model checker [18].

The experiments have been quite satisfactory, but have revealed two weaknesses, namely the missing maintenance of the conversion tool, and the problem (common to any other format translator) that counterexamples given by SMV must be traced by hand on the original Stateflow specification, and this is certainly a not immediate step.

We are currently investigating the possibility of writing a translator from Stateflow to the UMC on the fly model checker [9], developed at ISTI in Pisa, which takes as input UML State Diagrams. Obviously, the translator should be able to encode the Stateflow semantics into the UML semantics. In return, we are confident to obtain a better back tracing of counterexamples to the original Stateflow specification.

An alternative that is also taken in consideration is the use of commercial model checkers for Stateflow specifications, such as TNI's SCB and OSC's Embedded Validator. An evaluation of such tools is also planned.

5.2 Test Case Generation from Stateflow Specifications

The model developed in GETS allowed to identify corner-case scenarios of software behaviour and to adopt them as test cases during real system testing. Nevertheless the tests were not collected with a formal methodology that permits to measure the coverage of the model, but were identified during model simulation activities with the only purpose of defining software requirements. Therefore it was impossible both to evaluate the correctness of the model and to completely test the conformance of the real system to the software requirements through the model. These two aspects showed the need to develop a test case generation strategy. Indeed a test generation tool can help the user, together with a model checking activity, in the model validation; moreover, it can be used to strengthen the relations between the model and the software: this can be done by testing the same scenario on the model and the system and comparing the outputs. The purpose is to reduce the time to define the test cases and increase the detection of corner case scenarios. Therefore test case generation can be used to reduce the execution time of the functional tests. For this purpose automatic procedures, such as parallel execution of the model and the system with outputs comparison, will be investigated. An analysis of test generation tools as TNI's STB or T-VEC's Test Generation for Simulink is also planned.

5.3 Code Generation from Stateflow Specifications

Automatic code generation is usually considered as a natural output of a software formal specification because it can be easily obtained from a model using a proper tool. This is true in several application domains, but not in railways,

where code generation is viewed not less suspiciously than formal methods. Indeed railways safety-related systems are based upon architectures designed with safety more than performance targets in mind. Moreover, the operating software is wrote down mainly to satisfy testing, synchronization and other safety issues. Therefore, the application software needs to be written down following strict constraints, to be seamlessly integrated with the hardware and the operating software. Integration can be very hard, using the code generated from a model: evaluating and understanding how much hard is the object of future work. The evaluation could be done starting with a simple model and analysing the code generated with tools such as ADI's Beacon for Simulink/Stateflow, to understand which language structures it uses and how readable and "linked" to the model it is.The idea is that only using a special precaution during the model development is possible to generate a usable code that can be successfully integrated with the existing one. If the experiment gives good results, the code generation could be used for development of some application functions. For these functions, the effort could be shifted in the early phases of software life cycle (the model development) and most bugs could be fixed during model validation (test generation and model checking). Of course this will not replace standard software testing activities, but will reduce the time of the software life cycle and will guarantee the conformance of the software developed to the model used as software requirements specification.

6 Lessons Learned and Conclusions

The industrial acceptance of formal methods has always been difficult; though many success stories are reported, formally developed software is still a small percentage of the overall installed software. Application domains where safety is a major concern are the ones where industrial formal method applications are more easily found; in particular, railway signaling is considered one of the most successful area of formal methods diffusion. However, the choice among so many different formal methods proposals is not an easy task for a company; the risk of early choices of methods that are not suitable or are not widely accepted by the company departments is high. The experience we have reported has profited of many enabling factors that have in the end facilitated the choice:

- collaboration with academic experts;
- no time–to–market pressure (due to the longer time span of projects w.r.t. other application domains), which has allowed a long experimental phase before selection;
- European regulations asking for formal methods;
- a market evolution pushing for formal methods adoption;
- indications from the major clients about the preferred formalisms for specification.

However, even in this favourable setting for the growth of a formal method culture and in spite of standard and customers indications, the choice was still

not easy. Cost factors and company policies necessarily have driven, or even imposed, the choice. In the story we have told, the final choice of Stateflow has on one side followed a trend that has recently emerged in railway signaling, that is, a shift towards behavioral, state-machine based formalisms; on the other hand, this choice was favoured by the industrial quest for formalisms supported by commercial integrated environments, which have a broader scope of application. Tools that give the ability of simulating and model-checking specifications, and of generating code from them provide an interesting added value. The current stage of the adoption of Stateflow in GETS is that the tool is being used for specification and simulation in several new projects. Still more experiments are needed to better evaluate benefits and deficiencies of using model-checking, test case generation and code generation in GETS'products life cycle, and to choose industrial-strength tools offering such functionalities. Hence, it is too early to draft a final balance of the experience: the return of the ongoing analysis will actually be seen in several years.

References

1. S. Bacherini, S. Bianchi, L. Capecchi, C. Becheri, A. Felleca, A. Fantechi, E. Spinicci, *Modelling a railway signalling system using SDL.* In *Proceedings 4th Symposium on Formal Methods for Railway Operation and Control Systems (FORMS'03)*, Budapest. L'Harmattan Hongrie, 2003.
2. M. Banci, M. Becucci, A. Fantechi, E. Spinicci, Validation Coverage for a Component-based SDL model of a Railway Signalling System, Electr. Notes Theor. Comput. Sci. 116: 99-111 (2005).
3. M. Becucci, A. Fantechi, M Giromini, E. Spinicci "A Comparison between Hand-written and Automatic Generation of C Code from SDL using Static Analysis", Software: Practice&Experience, vol. 35, n 114, 2005, pp. 1317-1347.
4. G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.
5. J. Ellsberger, D. Hogrefe, and A. Sarma. *SDL - Formal Object-oriented Language for Communicating Systems*. Prentice Hall, 1997.
6. European Committee for Electrotechnical Standardization. EN 50128, Railway Applications Communications, Signaling and Processing Systems Software for Railway Control and Protection Systems, 2001.
7. A. Fantechi, E. Spinicci, *Modelling and Validating a Multiple-configuration railway signalling system using SDL.* Electronic Notes in Theoretical Computer Science 82 No. 6 (2003)
8. U. Foschi, M. Giuliani, A. Morzenti, M. Pradella, and P. San Pietro. The role of formal methods in software procurement for the railway transportation industry. In *Proceedings 4th Symposium on Formal Methods for Railway Operation and Control Systems (FORMS'03)*, Budapest. L'Harmattan Hongrie, 2003.
9. S. Gnesi, F. Mazzanti. "On the fly model checking of communicating UML State Machines" Second ACIS International Conference on Software Engineering Research Management and Applications (SERA2004), Los Angeles, USA, 5-7 May 2004.
10. G. Hamon, J. M. Rushby. An Operational Semantics for Stateflow. FASE 2004: 229-243, LNCS 2984

11. D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.
12. D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, M. Trakhtenbrot, *STATEMATE: A Working Environment for the Development of Complex Reactive Systems*, IEEE Transactions on Software Engineering, Vol. 16, N. 4, April 1990, pp. 403-414
13. N.H. König and S. Einer. The Euro-Interlocking formalized functional requirements approach (EIFFRA). In *Proceedings 4th Symposium on Formal Methods for Railway Operation and Control Systems (FORMS'03)*, Budapest. L'Harmattan Hongrie, 2003.
14. B. Krogh, C. Spencer, "Formal Verification of Stateflow Diagrams Using SMV", http://www.ece.cmu.edu/ webk/sf2smv/
15. P. Le Bouar. Interlocking SNCF functional requirements description. Euro-Interlocking Project, Paris, May 2003.
16. The Mathworks: Stateflow and Stateflow Coder, Users Guide. (2005)
17. The Mathworks: MATLAB 7 Users Guide. (2005)
18. K.L. McMillan. Symbolic Model Checking, Kluwer Academic Publishers, 1993.
19. Object Management Group, 1999, *Unified Modeling Language Specification, Version 1.5* http://www.omg.org/technology/documents/formal/uml.htm
20. J. Warmer and A. Kleppe. OCL: The constraint language of the UML. Journal of Object-Oriented Programming, 12(1):10–13,28, Mar. 1999.