Experimenting with diversity in the model driven development of a railway signaling system

Michele Banci
ISTI–CNR
Via G. Moruzzi 1
56124 Pisa, Italy
michele.banci@isti.cnr.it

Alessandro Fantechi
DSI - Università degli Studi di Firenze, Via S. Marta, 3, 50139 Firenze, Italy
fantechi@dsi.unifi.it

Stefania Gnesi
ISTI–CNR
Via G. Moruzzi 1
56124 Pisa, Italy
stefania.gnesi@isti.cnr.it

Giovanni Lombardi
ISTI–CNR
Via G. Moruzzi 1
56124 Pisa, Italy
giovanni.lombardi@isti.cnr.it

ABSTRACT

In this paper we discuss how we have introduced elements of diversity in the experimental model driven development process of a railway signalling system. The experience has been done inside a larger industrial project undertaken to evaluate the feasibility of employing formal modelling and automatic code generation in the development of a new generation of railway signalling systems hosted by a new fault-tolerant platform. The diversity is introduced at the level of the compilation of the generated code, and is aimed to discover possible faults due to the compilation environment or to the underlying operating system. Other forms of diversity will be then experimented in a step by step fashion.

Categories and Subject Descriptors

I.2.2 [**Design Language**]: state machines, automatic generation code;
D.2.1[**Software Engineering**]: Requirements/Specifications;
D.2.2 [**Software Engineering**]: Design Tools and Techniques;

General Terms

Design

Keywords

Formal modelling, automatic code generation

EFTS'07, September 4, 2007, Dubrovnik, Croatia
 ISBN 978-1-59593-725-4/07/09...$5.00

1. INTRODUCTION

We present some experiments made about the introduction of diversity in a development process of a safety-critical embedded system, based on formal modelling and automatic code generation. Indeed, within a collaboration with ALSTOM, ISTI-CNR has undertaken a project for evaluating on a proprietary embedded architecture the performance issues of code automatically generated from a formal model of the interlocking, in terms of number of controlled entities (switches, signals,…) that can be dealt with, within the real-time constraints established for the specific application.

In this paper we report instead how the introduction of diversity in the adopted model driven development process helps to improve safety of the produced equipment. The diversity is introduced at the level of the compilation of the generated code, and is aimed at discovering possible faults either due to the compilation environment or to the underlying operating system. Other forms of diversity are planned to be experimented in a step by step fashion.

The hardware platform that has been used in the present paper has been realized by ALSTOM for supporting communication among the different units constituting their interlocking systems. The platform is a proprietary 2 out of 2 architecture using two Freescale Coldfire processors. Hw & Sw diversity are used to bring the system to a fail shutdown in the case the two processors do not behave consistently.

The typical applications of the redundant platform are in the distribution and concentration of safe data links over a geographically distributed area.

The main purpose of the on-going collaboration of ISTI-CNR and ALSTOM is the evaluation of performance issues in the case the

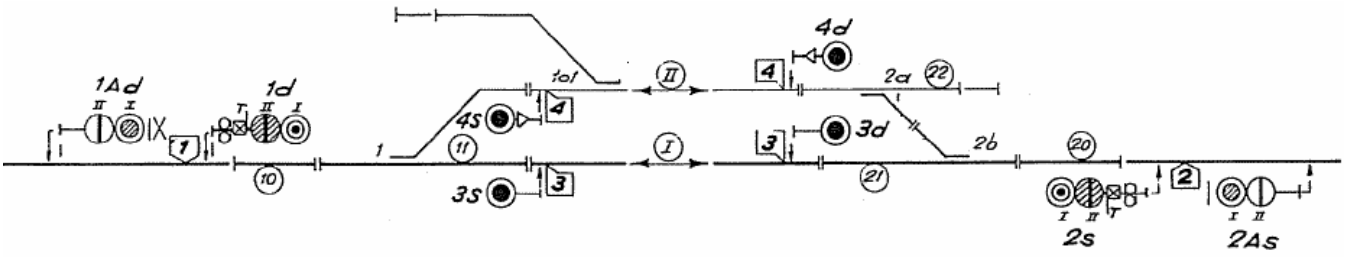

Figure 1. 019 system

platform is used as a basis to support the full functionality of an interlocking system, by means of code automatically generated from a formal model. In this paper, we address instead only issues related to the introduction of diversity in the software development cycle, hence we will not discuss any more hardware architectural issues.

The paper is structured as follows: in this section we have introduced the redundant architecture that hosts the designed interlocking system, which is described in Section 2. In Section 3 the adopted development process is presented, together with the used formalisms and tools. In Section 4 the same development process is revisited in terms of the safety issues, in conformance to the CENELEC safety guidelines for railway signalling systems. Such issues suggest the adoption of diversity for some phases of the development process, which is done in Section 5 for what concerns compilation. Further applications of diversity are discussed in the conclusive section.

2. THE CASE STUDY

In the experience described in this paper, we have used as a case study a real Italian interlocking system referring to a typical layout of a simple station (see [10]).

This typical layout is referred in the documents of Italian railways with the number 019. This example (a single track line station) is constituted by eight allowed routes, two switches, eight signals, six track circuits and two automatic blocks. In Figure 1 line segments represent track segments in the station; some of them have track circuits, that is, sensors that detect the presence of a train, which are numbered inside circles, joints between segments representing switches. Lollypop-like drawings represent signals of various type. Numbered labels are attached to each important part of a route. Interlocking rules are obviously the core of the system, so their correctness is the main objective to be addressed. The rules aim at allowing only the safe combinations of switches positions, signals and trains in a station in order to avoid collisions. The signal indications, handled by the interlocking system, govern the correct use of the routes, authorizing the movement of trains. The rules usually enforce a predefined sequence of actions: issuing a route request command usually triggers a check that all the track elements involved in the route are free. In this case, commands are issued for the positioning of switches for that route and for locking the track elements. This phase may be followed by a global centralized control over the correct state of the commanded elements, after which the route is locked and signal indications for the route are set.

A route can be set free only if all switches on it are in the correct position, and no trains are present.

The signals can be set to green only if the route in front of them is set to free. The above set of rules expresses two examples of generic principles that hold for every interlocking systems. Actually, the precise and complete set of such rules depends on the particular station or railway yard, and also on national policies traditionally established by railway companies or regulatory boards.

The development of computer controlled Railway Interlocking Systems has seen an increasing interest in the use of Formal Methods, due to their ability to precisely specify the logical rules that guarantee the safe establishment of routes for trains through a railway yard, as witnessed by the considerable literature about formalization of interlocking systems (see for example, [8][9][3][1][7]).

The application of formal methods in the rigorous definition and analysis of the functionality and the behaviour of a system, promises the ability of showing that the system is correct. Given such a promise, that is already out since several years, it is astonishing to see how little formal methods are actually used in the safety critical system industry, though the use of formal methods is increasingly required by the international standards and guidelines for the development of safety critical computer-controlled systems.

Industrial acceptance of formal methods is strictly related to the investment needed to introduce them, to the maturity of tool support available, and to the easiness of use of formal methods and tools.

Nowadays, the industrial trend is directed to the adoption of formal verification techniques to validate the design, integrating them within the existing development process.

Industries are more keen to accept formal verification techniques assessing the quality attributes of their products, obtained by a traditional life cycle, rather than a fully formal life cycle development, due to the lower training and innovation costs of the former.

Several approaches to the application of formal methods in the development process have been proposed, differing for the degree of involvement of the method within it. Starting from rigorous specifications, formal methods can be used for the derivation of test cases, or as a validation technique aimed at proving that the specification satisfies the requirements, or as an auxiliary technique in the automated generation of code.

3. THE DEVELOPMENT PROCESS

The development process adopted for the case study follows the cited trends, basing on the modelling of the 019 system using the tool SCADE of Esterel Technologies, by means of state machines (called SSM – Safe State Machines) and data-flow diagrams, starting from the original specifications of the 019 system.

The SCADE (Safety Critical Application Development Environment) tool-suite by Esterel Technologies is a set of tools able to support a whole model-based development method. SCADE is mostly used in automotive and avionics applications, and relies on the use of diagrams and state machines, representing an approach to formal modelling based on a formal graphical description of the specification of a system. Its graphical

modelling formalism benefits from deterministic formal semantics, allowing the derivation of a clean mathematical model from a SCADE design to the synchronous paradigm of the Lustre [2] language. The same deterministic model could be used for correct-by-construction automatic code generation and formal verification [11].
SCADE provides a verification technique based on formal verification tools over the model as well. It is based on the synchronous data-flow paradigm. Inputs and outputs of a SCADE block are typed data-flows. The type of a data-flow can be simple (bool, int, real) or structured (a structure or tuple made of a set of typed fields).
The model produced for the 019 system is made up by 68 SSMs. Every SSM is very simple and it is formed by two states, being the model directly derived by the relay circuits schematics that constitute the official specification of the 019 system. The model cannot be reproduced here for lack of space: we only give a description of a portion of the model, as it appears in a window of the SCADE tool (Figure 4).
The model has been simulated by means of the SCADE simulator, employing as simulation cases an already existing suite of test cases for the 019 system.
Model checking over the SCADE state machines by means of the native model checker Design Verifier tool is another way to gain confidence in the model, which has only preliminary been attempted at the current stage of the project.
After the SCADE model has been developed and simulated, the KCG 5.1 code generator, available in the SCADE tool suite, has been used to derive C code implementing the model; this code generator has obtained a certification for the Safety Integrity Level A of the DO-178B avionic guidelines. The obtained code can be directly used on the architecture target without any modifications.

The generated code is then compiled and loaded on the duplicated processors, by means of the CodeWarrior for Coldfire compiler and its facilities for host/target development.

The same interface[1] (see Figure 3) is used for providing inputs and reading outputs to/from the SCADE model, and to send input commands and to receive output values to/from the embedded system, the same tests can be in this way performed on the two implementations. We speak of simulation also in the case of test execution over the target platform since we are anyway simulating the real environment.

Simulation of the SCADE model and of the implementation contributes to give further confidence over the validated code generator.

[1] Actually, two equivalent interfaces have been built, one written in Java and one written in Visual Basic, to better accommodate the different used environments.

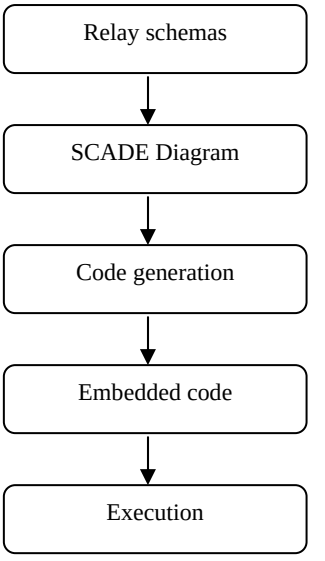

Figure 2 - Development process

4. SAFETY ISSUES IN THE DEVELOPMENT PROCESS

CENELEC guidelines recommend the usage of specific techniques to increase the safety of railway signalling systems in spite of faults that can have a negative impact on the safety of the system, at the hardware level (EN 50129) [5], at the software level (EN 50128) [4] and at the system level (EN 50126) [6]. Form the lower level to the most abstract one, the main techniques used at each step of the development process.
As we have already said, a 2 out of 2 architecture is used to detect hardware faults and to bring the system in a fail-safe state. Actually, many other techniques are used to improve reliability, and hence safety, of the hardware. Moreover, various applications of this architecture employ two replicas of this fail-safe processor, for improving availability.
The real-time operating system, is considered as “proven in use” and hence acceptable for the guidelines: system testing and simulation with the SCADE model help to cover possible problems related to the use of this operating system

The same can be said for the compiler, which is also considered proven in use; again, system testing and co-simulation with the SCADE model help to cover possible problems related to the use of this particular compiler.
We have already reported that the SCADE code generator has been validated for the high levels of safety integrity. Also in this case, system testing and co-simulation on the model and on the generated code with the same tests help in gaining confidence over the generated code.
We have also reported that confidence in the produced models is obtained by means of simulation, and model checking.
The weaker phase in this process, for what concerns the safety issues, appears to be the Compiler/Operating System issue, the confidence on which is based only on simulation and testing. Also a better confidence over the modelling phase, delegated only to simulation and model checking, could be desirable.

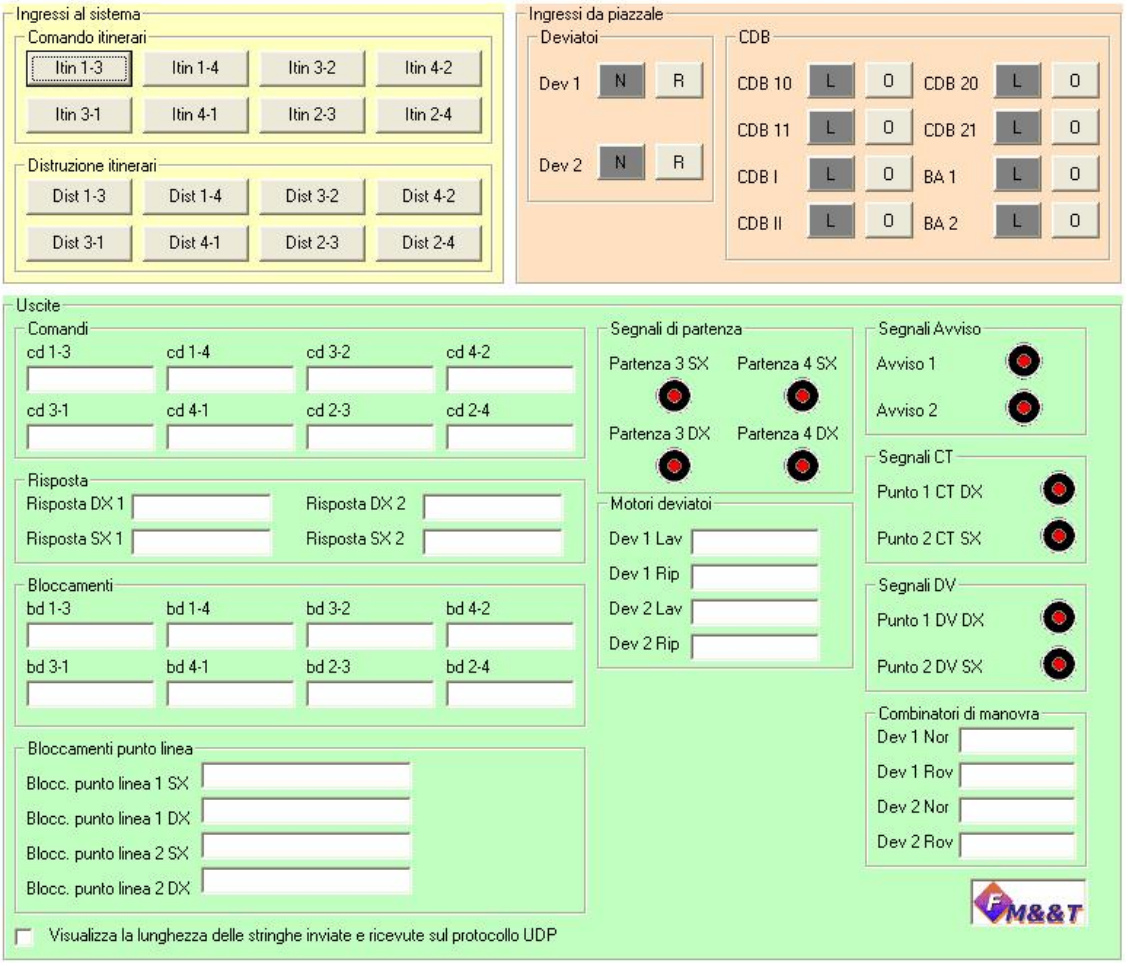

Figure 3. 019 input-output panel

5. INTRODUCTION OF DIVERSITY IN COMPILATION

Diverse design and diverse programming has been often adopted as a method to build systems that can tolerate design faults.

Software faults are essentially due to design problems, hence in order to provide means to tolerate software faults diverse design can be exploited at several levels, for example requiring hardware redundancy as well.

In particular, diversity is to be applied where other forms of guaranteeing the absence of faults are not in place.

As we have said, inside the development process presented in the previous section, a weakness can be identified in the Compiler and Operating System support. We have applied diversity at this step by compiling the generated code with a different compiler over a different operating system. In particular, a Window implementation is obtained by means of the Visual C programming environment.

The obtained code has been executed on the host Windows machine, exercising it by means of the same interface designed for the other implementation, and the execution of this implementation has been compared, on the identified test cases, with the execution on the target platform. In other words, we have executed a simulation as already done for the model on the embedded implementation.

Since the two implementations differ for the compiler used and for the operating environment (processor and operating system), co-simulation allows possible faults depending on such components to be detected. Obviously, although exhaustiveness of testing cannot be reached, the use of an extensive test suite derived from the application domain allows to reach a high degree of confidence. Indeed, with the (limited) amount of testing conducted as far on the case study, we were not able to identify discrepancies. At present, all the testing activity has been done manually, but the co-simulation and testing can be appropriately automated in the production process, accommodating much more extensive test suites.

The other area of development process in which we have identified a weakness form the point of view of guaranteeing safety is the modelling itself.

The adoption of diversity also in this case is under study, giving two diverse models of the same 019 system developed with two formalisms, for example with SCADE and Statemate, and to compare them both by simulation and by testing and comparing the generated codes, which would be generated by diverse code generators as well.

The two models will be co-simulated to look for discrepancies, against exploiting an extensive test/simulation suite. This particular application of diversity aims at discovering of faults due to erroneous interpretation of system requirements,

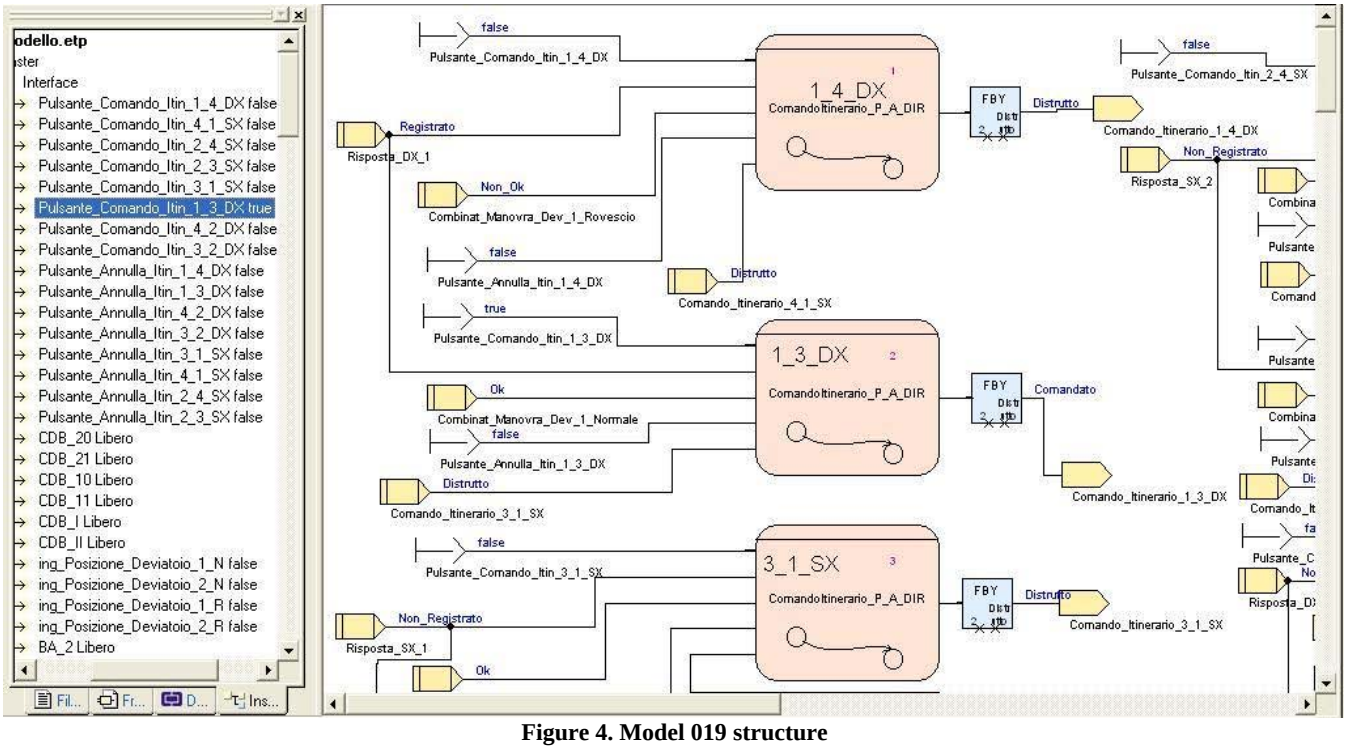

Figure 4. Model 019 structure

especially those which could have been facilitated by the specific modelling paradigm enforced by the adopted formalism and tool. Comparing a synchronous SCADE model with an asynchronous Statemate model will make it possible to discover this kind of faults. The two models can even produce, by diverse automatic generator tools, two diverse versions of C code, which again can be tested over the same test suite.

It would be even possible in principle to load the code generated starting from the diverse models in the two processors of the Smart I/O platform: this possibility needs however to be studied in detail, since synchronization issues between the two replicas become an important problem.

6. CONCLUSIONS

The availability of the complete environment needed to produce an application at industrial level from a formal model to the code has been exploited to carry on some experiments on the introduction of diversity in the development process.

The first experiments have been encouraging due to their relatively low cost (limited to re-compilation and re-testing), that can positively effect the industrial acceptance of the approach. We are planning new experiments to extend the adoption of diversity, to other steps of the development process

7. ACKNOWLEDGMENTS

We wish to thank ALSTOM for the technical support given to the experiments and for having provided the hardware platform and the access to the needed software tools.

8. REFERENCES

[1] A. Cimatti, F. Giunchiglia, G. Mongardi, D. Romano, F. Torielli and P. Traverso, 1998, *Formal Verification of a Railway Interlocking System using Model Checking*, Formal Aspects of Computing, Vol 10, 361-380.

[2] Alain Le Guennec, Bernard Dion, Esterel Technologies, *Bridging UML and Safety-Critical Software Development Environments.*

[3] C. Bernardeschi, A. Fantechi, S. Gnesi, S. Larosa, G. Mongardi and D. Romano. *A Formal Verification Environment for Railway Signaling System Design*, Formal Methods in System Design, Vol. 12, 2, pp. 139-161, 1198.

[4] CENELEC European Committee for Electrotechnical Standardization. *Railway applications: software for railway control and protection systems.* European Standard, June 1997. prEN 50128.

[5] CENELEC European Committee for Electrotechnical Standardization. *Railway applications: safety railway related electronic systems for signalling.* European Standard, December 1999. prEN 50129.

[6] CENELEC European Committee for Electrotechnical Standardization, *Railwayapplications: software for railway control and protection systems. European Standard,* June 1997. prEN 50126.

[7] F. J. van Dijk, W. J. Fokkink, G. P. Kolk, P. H. J. van de Ven and S. F. M. van Vlijmen, *EURIS, a specification method for distributed interlockings*, in (W. Ehrenberger, ed) Proc. 17th Conference on Computer Safety, Reliability and Security -

SAFECOMP'98, Heidelberg, Lecture Notes in Computer Science 1516, pp. 296-305, Springer (October 1998)

[8] M. Banci, A. Fantechi, *Geographical vs. Functional Modelling by Statecharts of Interlocking Systems.* FMICS Ninth International Workshop on Formal Methods for Industrial Critical Systems, Linz, September 20-21, 2004. Electronic Notes in Computer Science (Elsevier).

[9] P. Behm, P. Benoit, A. Faivre, and J.M. Meynadier. *Meteor: A Successful Application of B in a Large Project.* FM'99, Toulouse, Sept. 1999, LNCS 1708, pp.369-387

[10] P. E. Debarbieri, F. Valdambrini and E. Antonelli, A.C.E.I. *Telecomandati per linee a semplice binario, schemi I0/19. CIFI Collana di testi per la preparazione agli esami di abilitazione*, Quaderno 12, 1987.

[11] Pascal Raymond, PhD thesis : Compilation efficace d'un langage d´eclaratif synchrone: le generateur de code Lustre-v3, Institut National Polytechnique de Grenoble, 1991.