The Role of Formal Methods in Software Procurement for the Railway Transportation Industry •

Umberto Foschi[1], Mauro Giuliani[1], Angelo Morzenti[2], Matteo Pradella[3], Pierluigi San Pietro[2]
(1) Rete Ferroviaria Italiana S.p.A., Italia; e-mail: {u.foschi, m.giuliani}@rfi.it;
(2) Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italia,
email: {morzenti, sanpietr}@elet.polimi.it;
(3) CNR Istituto di Elettronica e di Ingegneria dell'Informazione e delle Telecomunicazioni, Milano,
email: pradella@elet.polimi.it.

Abstract-- The present paper reports the experience of a joint project between Politecnico di Milano and Italian State Railway FS, Infrastructure Department(became Rete Ferroviaria Italiana S.p.A.: R.F.I. S.p.A.). The purpose of the project was to define procedures and rules for managing software procurement for safety-critical signalling equipment. The project covers all phases of system development, from requirements elicitation to implementation and final validation, providing requirements on methods, languages and tools to be used during software development, without any bias towards any particular technology or tool provider. The results are consistent with, and acceptable against, international standards. In particular, Requirements/Recommendations have been issued, tailored on various kinds of systems under examination, concerning: a) methods, techniques, languages and tools; b) organization of the provider company in terms of independence and responsibility of participating actors; c) documentation to be produced by the provider. A first experimental evaluation of formal specification methods applied to signalling systems is also reported, and we outline a further experimentation, where the results presented here will be applied on an industrial scale in the procurement, by RFI, of a complex signaling apparatus.
Keywords: Requirements Analysis, Verification and Validation, Command and Control Systems, Transportation Systems

I. INTRODUCTION

Computer based systems are getting ever more pervasive and in charge of critical missions. Mission criticality stems from economic, environmental, human-life-safety factors, as for example in systems for patient monitoring, flight control or automatic guidance. Often such systems are operated by organizations providing services (such as energy production and distribution, telecommunications, logistics, transportation), but that do not develop computer-based systems themselves. Rather, they act only as system integrators of systems purchased from external suppliers, typically hardware and/or software high tech companies.
Service providers have therefore the problem of managing acquisition and integration of purchased (sub)systems. Hence, such organizations often need clear, unambiguous, possibly formal, requirement specifications, to set requirements and make clear the responsibility of the purchaser and of the supplier. Moreover, both the purchaser and their suppliers must agree on rigorous acceptance procedures, based on verification (testing) and (final) validation, functional and safety assessment and safety approval. Last, uniform, possibly standardized documentation is essential to permit the monitoring of the development and facilitate the operation and the maintenance.
When dealing with safety-critical systems, the procurement task is made even harder by the requirement, under applicable national and international laws, that the systems must verify a suitable set of international standards, dictating procedures for design, deployment and maintenance. These standards must be applied under the legal responsibility of both the purchaser and their suppliers.
The present paper reports the experience of a joint project between Politecnico di Milano and Italian State Railway FS, Infrastructure Department (which recently became Rete Ferroviaria Italiana S.p.A: R.F.I. S.p.A.). The purpose of the project was to define procedures and rules for managing software procurement for safety-critical signalling equipment. The latter includes a broad range of devices, governing lines and tracks in stations, railway/road crossings, and train movements.
Various goals obtained in the project, which were imposed as additional constraints, are:

- the project covers all phases of system development, from requirements elicitation to implementation, final validation, approval and acceptance;
- the project provides requirements on methods, languages and tools to be used during software development, without any bias towards any particular technology or tool provider. The only general requirement is technical soundness and being up to date with respect to the current advances in computer science.
- the results are consistent with, and acceptable against, international standards (mainly the EN50128 standard for software [EN01]).
- choices are made (by imposing requirements) to obtain the best combination / trade off between needs of purchaser and provider.

• This work was partially supported by MIUR Project: "QUACK: Piattaforma per la qualità di sistemi embedded integrati di nuova generazione".

- the chosen methods are mature at industrial level, are supported by automatic tools, and are likely to gain acceptance by average engineers, both in the railway and computer technology domains.

In particular, Requirements/Recommendations have been issued, tailored on various kinds of systems under examination, classified according to the following three "dimensions":

1. complexity (low, medium, high)
2. degree of safety-criticality (SIL)
3. presence of temporal requirements (time independency, qualitative/quantitative time)

Recommendations concern:

a. methods, techniques, languages and tools;
b. organization of the provider company in terms of independence and responsibility of participating actors;
c. documentation to be produced by the provider.

In the paper we focus on point (a), occasionally mentioning results of kind (b).

Not surprisingly, the main result/contribution of the project concerns requirements specification, verification, and validation: these techniques support a correct interaction between purchaser and provider (design and implementation/coding are more mature and less critical: these choices can be left to providers). In particular, the project recommends the adoption of formal methods for the specification phase, when supported by suitable tools and verification and validation techniques. In the full paper, we report the results of a comparative evaluation of methods, tools and notations for (formal) requirements specification, starting from a discussion of their needed features.

As far as the evaluation of specification methods is concerned, we performed an experimental activity, the formal specification of a simple, but highly critical and time dependent, signaling apparatus. The paper also reports the main results of this.

We also report overall recommendations concerning:

- verification, e.g., testing through a suitable combination of functional and structural techniques, and adoption of coverage metrics;
- coding standards. e.g., choice of programming language (Ada as opposed to C), use of tools for static and program analysis;
- final validation, and the problem of minimizing its cost through a combination of requirements analysis and acceptance tests performed as much as possible in a simulated environment rather than in the field.

Our investigation differs from other apparently similar studies on application of formal methods to validation and verification of critical systems, such as [NASA95], in that it is very much finalized to the selection of methods that are commercially supported and at the same time have a defined level of automatic support to a given set of validation and verification activities.

In Section II, the paper describes the main recommendations issued by the project and in Section III shows the results of an experimental comparison of specification methods (Statecharts and SDL). Section 4 draws a few conclusions.

II. Main Results and Recommendations

A. System Classification and Responsibilities

As mentioned in the introduction, the recommendations were based on a classification of the systems under procurement according to three dimensions: complexity, criticality, and temporal requirements. Complexity was assumed to be conventionally determined by the purchaser, while the degree of criticality was determined, according to well known and widely recognized criteria, by the SIL (Safety Integrity Level) of the application.

The temporal requirements were classified in three categories: *time independent*, *qualitative time*, and *quantitative time*. The time independent category refers to systems without any particular temporal constraints, e.g. performing pure data or signal elaborations. The qualitative time category refers to systems that send to or receive from the environment time-ordered values and actions, without any quantitative information about time instants and time distances. Some improper real-time systems are in this category, systems with strict and binding requisites, but designed (e.g. by means of *ad hoc* protocols, synchronization or interlocking mechanism) to adequately manage every possible delay (or anticipation), or the absence of expected events. The quantitative time category comprises the properly called Hard Real Time (HRT) systems. These systems interact with processes that are not completely manageable or controlled, and that cannot avoid a quantitative expression of their temporal constraints (not just an order relation among events), without severe consequences. It is worth pointing out that the category of *qualitative time* is quite different from the so-called *soft real-time systems*, i.e., systems where missing some (quantitatively of qualitatively stated) time constraint is undesirable or annoying but does not cause unacceptable damage; also it does not correspond to *high-throughput* systems, which must have the capability of processing high quantities of data, but with time requirements that are expressed in statistical terms. This is because the systems under consideration were in any case critical for safety and economic reasons, so that missing time requirements (even when these are qualitative) is not admitted.

With reference to the various phases of the software development and to the correspondingly produced documents, the following responsibility roles are characterized: (i) specifier, (ii) software designer, (iii) programmer, (iv) person in charge of the requirement validation and final validation, (v) person in charge of the software verification. To obtain an effective and correct task subdivision and to favor independence and detachment of the acts of the responsible persons, it is required that some constraints are satisfied in assigning such roles to the persons who participate to the software development. For systems having SIL 3 or 4 and medium or high complexity, and for those of SIL 1 or 2 and high complexity, it is required that one given person cannot cover two of the above roles simultaneously; for example, the software designer cannot be also in charge of its verification; moreover, it is required that, for the systems with SIL 3 or 4 , the persons who cover roles (ii) and (iii) belong to one structure or organization (e.g., the division of planning) distinguished from those of sets (iv)

and (v) (e.g., the quality control division). These requirements for the personal incompatibility among the various roles are displayed in Figure 1, using boxes with long dashes. (The Figure, taken from [SM01], also shows connections the development phases and the documents there produced) The requirements are slightly lessened for systems having SIL 1 or 2 and medium or low complexity, and for systems having class of integrity 3 or 4 and low complexity (see boxes with short dashes in Figure 1): in this case the personal incompatibility is established among the following *set* of roles: {specifier}, {software designer, programmer}, {person in charge of the requirement validation and the final validation, person in charge of the software verification }; e.g., the software designer can be also a programmer, but a programmer cannot be the person in charge of the verification of the software. These less restrictive requirements consider the fact that some small or medium sized enterprises, though having a project staff of reduced size, might as well develop high quality software, if they correctly apply the suggested notations, methods, and tools.

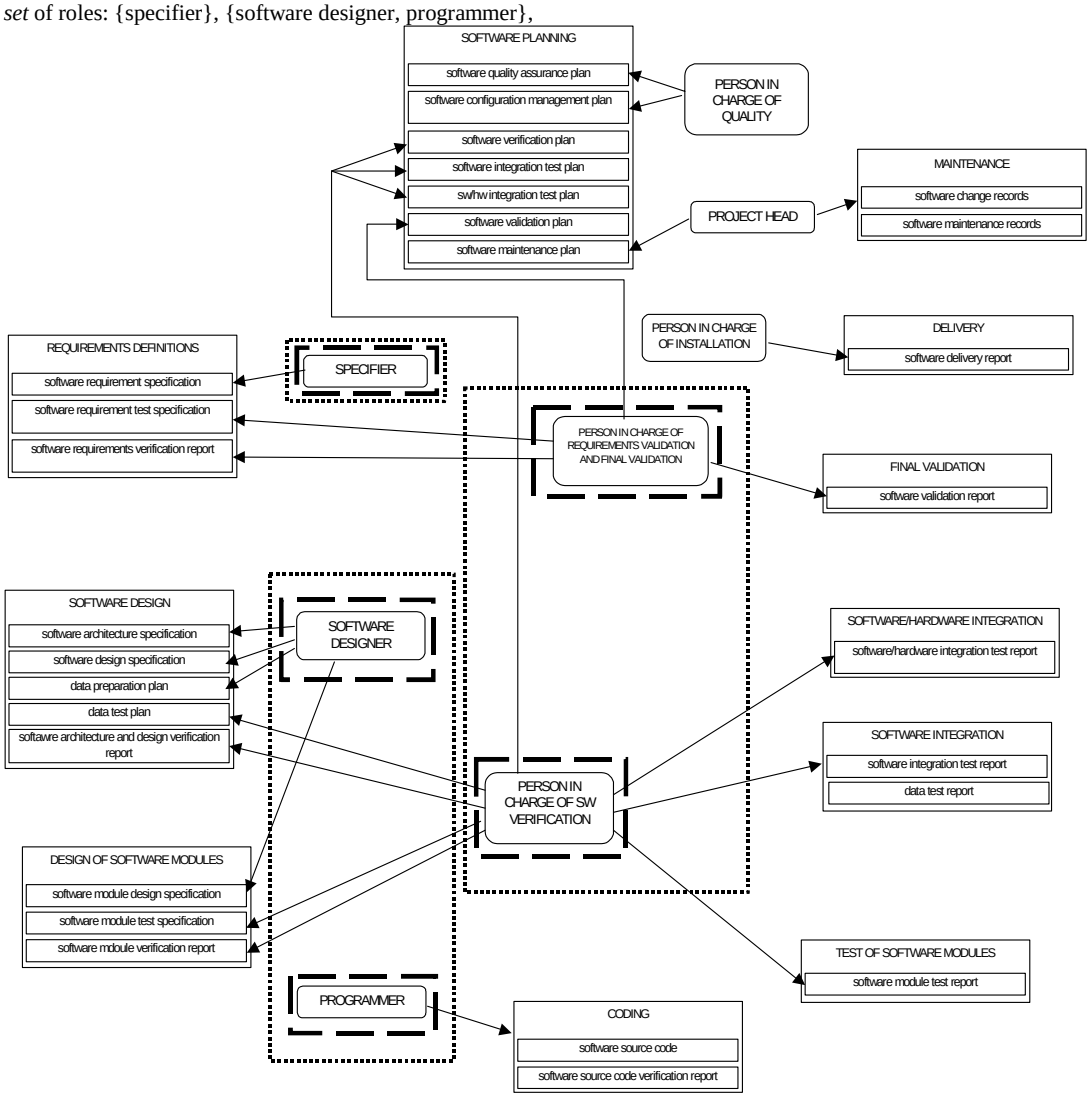

Figure 1: Phases/documents/persons-in-charge associations

B. Requirements Analysis and Specification

Table 1 shows the prescription on specification validation techniques and generation of functional test cases, depending on the integrity class of the system (class A includes SIL 3 and 4, class B includes SIL 1 and 2), its complexity and its temporal features. Table 2 illustrates the meaning of the terms used in Table 1.

Prescription \ System			CLASS		COMPLEXITY			TIME		
			A	B	LOW	MED.	HIGH	INDIP	QUAL	QUANT
Analysis	Simulation or traces generation		YES	YES	*	*	YES	*	YES	YES
	Property proof	no abstraction	*	YES(3)	*	*	*	*	*	*
		a bstraction	*	YES	*	*	*	*	*	*
		Generality	*	YES(3)	*	*	*	*	*	*
		Automation	*	SEMI	*	*	*	*	*	*
Syntax controls			YES	YES	*	YES	YES	*	YES	YES
Degree of spec. coverage			T	T	*	*	*	*	T(1)	T(1)
Validation accuracy degree			β	γ(2)	0	α	β	0	β(1)	γ(1)

Notes (see Table 2)

(1) The indicated coverage/accuracy is a minimum requirement for the temporal parts alone.

(2) It is recommendable, but not mandatory at the current state of the art, to use a method with degree of accuracy δ.

(3) It is recommendable, but not mandatory at the current state of the art.

The character * means that there is no recommendation, neither in favor nor against the adoption of this technique.

Table 1 Validation prescriptions.

ANALYSIS	The analysis activities are those that make possible th validation of the specification. The important activities for the validation are listed, except those possible with any notation (such as inspections and walkthroughs).
Simulation	The possibility to simulate/animate, also in interactive and semi-automatic way, the behavior of the system, generating the events and the actions included in the simulation in chronological order. **Values**: YES/NO
Traces generation	The possibility to generate (also in semi-automatic way) execution traces of the system according to the specification, and to verify automatically if the given traces are compatible with the specification. Differently from the simulation, events and actions are not necessarily generated in chronological order. **Values**: YES/NO
Property proof	The possibility to prove mathematically (by means of logical demonstrations or exhaustive analysis) that the specified system possesses suitable properties, e.g. of safety, absence of deadlock, etc. They are classified in the following according to the degree of certainty and generality.
Without abstraction	The proofs can be executed on the complete specification of the system. They have therefore a total degree of certainty: the specified system possesses without doubt the proved property. **Values**: YES/NO
With abstraction	The proofs can be executed, except in very simple cases, by introducing suitable approximations (abstractions) of the original specification, e.g. in the case of model checking when the actual data dealt by the system are ignored. Abstractions make proofs simpler but reduce the degree of certainty of the result. **Values**: YES/NO
Generality	The properties to be proved can be chosen by the user in a general and flexible way, using a suitable sufficiently expressive mathematical notation. **Values**: YES/NO
Automation degree	The support offered by the tools. **Values**: MAN (manual): proofs are carried out by hand; SEMI (semi-automatic): the tools support at least the verification that the proof is correct, and possibly prepare a structure of the proof (proof obligations) and/or automate the trivial parts and sub-proofs, but must be guided from expert users; AUTO: proofs are completely automatic. **Order**: MAN<SEMI<AUTO
Syntactic controls	Tool support in verifying that a specification is syntactically correct. **Values**: YES/NO
Specification coverage degree	**Values**: T (Total), when all the requirements have the same relevance and must therefore be specified; P (Partial), when some requirements, identified in unambiguous way and totally isolated from the others, do not have any influences on safety. **Order**: P<T.
Accuracy degree of validation	The degree of accuracy for the validation of the requirements specification to be carried out: it prescribes the available techniques of validation to be applied in order to catch up an adequate level of confidence. **Values**: O: informal inspections, walkthrough; α: syntactic controls of type, coherence between definition and use of the entities that compose the specification, i.e. the typical static controls carried out by the compilers of modern programming languages; β: at least one of the following: simulation, animation, generation of traces, symbolic analysis, reachability analysis, proofs of prefixed properties (e.g., absence of deadlock), proof of properties with abstraction; γ: same techniques as β, but made with a combination of at least two techniques of different nature and adopting suitable metrics in order to measure the coverage degree of the analyses; δ: statement and proof of general properties. **Order**: O<α<β<γ<δ.

Table 2 Legend for Table 1.

Finally, Table 3 shows the application of the prescriptions of Table 1 to a set of widely used formalisms, considering both language features and current tool support. The analyzed notations and formal methods are Z [Spi88], TRIO [GMM90], Statecharts [Har87], SDL [EHS97], UML [BRG99], LOTOS [EVD89], PN [Mur89], SCADE [BDS91], B [Abr96]. Table 3 is obtained by comparing, for each notation and corresponding method and tool environment, the characteristic features and the tool support with the requirements expressed in Table 1. It can be noted that, not surprisingly, the state of the art is still unsatisfactory, even for the methods and tools hat received the "best score", in the case of systems with quantitative timing features (so-called strict real time systems) and a high level of complexity.

In this case there is no “strongly recommended” method and tool, the existing ones being only “recommended”. This is due to the fact the currently available tools for analysis and verification of formal models are not certified neither validated by repeated and long-lasting application in an industrial setting. This is clearly the area where most significant theoretical and technical advances are needed and it is in fact a very active research area in the formal method international community. Table 3 will be subject to periodic revisions, since tool support may improve over time.

System / Method	INTEGRITY		COMPLEXITY			TIME		
	B	A	LOW	MED.	HIGH	IN	QL	QT
Z	Y	N (1)	Y	Y	N(1)	Y	Y	N
TRIO	N (1)	N (1)	Y	N (1)	N (1)	Y	Y	Y
STATE CHARTS	Y	Y	Y	Y	Y	Y	Y	Y (2)
SDL	Y	Y	Y	Y	Y	Y	Y	Y (2)
UML	N (3)	N	N (3)	N	N	Y	Y	N
PN	Y	N	Y	N	N	Y	Y	Y (2)
LOTOS	Y	N	Y	Y	N	Y	Y	Y (2)
SCADE	Y	N (1)	Y	Y	Y	Y	Y	Y (2)
B	Y	Y	Y	Y	Y	Y	Y	N

Legend: IN = Independent, QL: Qualitative, QT = Quantitative., Y = YES, N = No
(1) The NO answer derives from the unavailability of tools with a sufficiently consolidated level, that possess all the features required for the YES value.
(2) The method is recommendable for systems of class B. For the class A, this method is acceptable at the current state of the art, but not strongly recommended.
(3) The method is recommendable only for cases of class B with SIL=1.

Table 3. Prescriptions on specification methods.

C. Design, Coding, and Testing

Concerning the phases of high-level and detailed design the criterion was adopted to leave as much as possible to the supplier the choice of the notation and tools for performing and documenting the design steps, with the strong provision that all development steps be thoroughly documented. Adoption of UML notations, such as class diagrams, structure diagrams, deployment diagrams, was suggested, but also the use of more traditional, function-based rather than object based or object-oriented methods was admitted.
A much greater attention was devoted to the choice of the programming language adopted for the coding phase. A comprehensive investigation (also supported by the analysis of [Sto96]) led us to the conclusion that two languages are technically suitable for the development of critical software in the railway transportation field, namely Modula-2 and Ada. The former, however, suffers from a lack of suitable development tools and of a very quite limited industrial adoption. Hence the recommendation to use Ada for coding this category of computer based critical applications. We also had to consider the reluctance of many suppliers to adopt Ada, due to the perceived intricacies of the language and to the unavailability of professional programmers, and their preference for the C language. Therefore the use of the language C was also admitted, as a “second choice”, subject to the adoption of a set of strict coding standards and to the use of industrial strength tools for static verification of C programs (like PC-lint and QA C). The coding standards are meant to enforce a use of the C language in an “object oriented fashion”, and enclose the programming techniques, reported in [Hat95], concerning the major, well-known problematic features of the C language, i.e., unspecified behavior, undefined behavior, implementation-dependent behavior, and local behavior.
Recommendations on testing enclose the adoption of well-known techniques for white-box testing (coverage of statements, conditions, branches, paths, etc.) and for black box (i.e., functional) testing. We emphasize that functional testing must, at least in part, be based on test cases derived from formal specifications of the system and software requirements, thus exploiting the well known fact that formal models can be used for the two complementary and synergic operations of requirements validation and production of artifacts (e.g., functional test cases) to plan and support the verification process..
Functional, specification-based testing is also considered as complementary to structural testing, in that the degree of achieved structural coverage of the testing process, to be measured with the support of suitable software tools supporting the testing process, can also be reached by the application of functional test cases.

III. METHODS AND TOOLS: A COMPARISON

The last step of the joint project consisted of an experimental activity of comparison of two well-known methods and tools, namely the SDL tool suite by Telelogic and the Statemate tool suite by I-Logix. RFI provided both a natural language description of the adopted case study, a railway crossing signaling system, and a high-level schema. Very briefly, the analyzed system is in charge of checking the railway crossing current status and sending the suitable signals both to the station and to the train.
The experimental comparison was carried as follows. In a first meeting, trained engineers from RFI, supported by instructors from I-Logix, built a Statemate specification of the system, and then validated it using a simulator. A similar setting was arranged for the SDL tools. Afterwards, the authors performed some analyses, both qualitative (e.g. readability, ease of use), and quantitative (e.g. automatic proofs of a simple but critical properties, automatic test case generation) on the two different specifications. The analyses were carried out using the tools provided by Telelogic and I-Logix.

The results of the comparison, based on the elements defined in Table 4, have been summarized with reference to three main aspects: 1) naturalness and clarity of the notation; 2) availability of adequate validation and verification tools; 3) engineering level and documentation of the suite.

Specification	**readability**
	scalability
	composability
	coverage
	temporal aspects
	traceability
Validation & Verification	**simulation / traces**
	test cases
	proof of properties
Tools	**integration with O.S. / robusteness**
	notation completeness
	ease of use
	syntactic/ semantic checks
	validation and verification efficiency
	internal integration
	documentation

Table 4. Methods and tools comparison elements

Both SDL and Statemate were adequate with respect to point 3, with a slight advantage for SDL. As far as point 1 is concerned, the SDL notation resulted more cumbersome compared with Statemate. The RFI engineers found quite harder to express the system in SDL, while Statemate resulted more "natural" and easy to understand, and then to manage during the validation activity. On the opposite side, the quality of the tools (point 2) was at the time much better for SDL, while Statemate tools were incomplete and unable to effectively perform all the requested validation and verification activities. Nonetheless, we were able to test a really promising final prototype of a Statemate-based analysis tool, expected to be made available to the market in a few months.

IV. Conclusions

Various lessons have been learned during the project and among them we mention the following:

1. formal methods for specification and verification are—slowly and with difficulties—reaching some appreciation and use in the industrial environment: 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 the industry;
2. thorough verification of complex, hard real-time systems is still infeasible in practice using the (industrial strength) tools available at the time of the project; the verification technology is however rapidly evolving;
3. international standards like EN50128 [EN01] can have a positive role in promoting the adoption of systematic and technically sound development methods, but can also be technically outdated, obscure, ambiguous or too accommodating.

Based on the results of the project, the re-engineering of the specification of a large body signaling devices, that were historically developed based on electrical technology, can be done using the formal methods; applications can be developed by industry starting from the new specifications and applying the suggested new procedures; it also expected that the results of the joint project between Politecnico di Milano and FS - RFI can contribute to the planned upgrading of Cenelec EN 50128 standard..

A pilot application of the procedure defined in this project will start soon in RFI. This application, concerns the development of a new RFI Automatic Block System for train spacing. The procedure will be applied to the whole software lifecycle and its effectiveness will be carefully evaluated. Those parts of the procedure for which the pilot application evidences criticalities will be modified. The procedure, adequately finalized, will be proposed for the Cenelec EN 50128 upgrading process. The new Automatic Block System for train spacing is currently specified by a set of user requirements, expressed in textual form, concerning the main functions (train detection, train spacing, block orientation management, transmission to the on board cab signal of the signals aspect, recording of alarms and others juridical data), the performances (RAMS requirements) and the constraints (required level of E M Immunity, etc). The first action will be the formalization of the informal user requirements specification. The obtained formal requirements will be validated by simulation and the validated formal requirements will be the input for the following development phases, that also will be managed applying the defined procedure and adequately using the formal methods.

References

[Abr96] J. R. Abrial, The B-Book, Cambridge University Press, October 1996, ISBN: 0521496195.

[BDS91] Frédéric Boussinot, Robert De Simone, The Esterel Language. Another Look at Real Time Programming, Proc. of the IEEE, vol. 79, pp 1293-1304, 1991.

[BRG99] G. Booch, J. Rumbaugh, I. Jacobson, The Unified Modeling Language User uide. Addison-Wesley, 1999

[EHS97] Ellsberger, J.; Hogrefe, D.; Sarma, A., SDL - Formal Object-oriented Language for Communicating Systems, Prentice Hall Europe, 1997, ISBN 0-13-621384-7.

[EN01] EN 50128 Railway applications – Software for railway control and protection systems, CENELEC - 2001.

[EVD89] P. H. J. van Eijk, C. A. Vissers, M. Diaz (editors)

[GMM90] Ghezzi C., Mandrioli D., Morzenti A. TRIO a Logic Language for Executable Specifications of Real-time Systems, Journal of Systems and Software, June 1990.

[Har87] Harel, D. 1987. Statecharts: A visual formalism for complex systems. Science of Computer Programming 8, 231-274. Preliminary version: Tech. Report CS84-05, The Weizmann Institute of Science, Rehovot, Israel, February 1984.

[Hat95] Les Hatton, SAFER C: developing software for high-integrity and safety-critical systems, McGraw-Hill, London, 1995.

[Mur89] T. Murata, Petri Nets: Properties, Analysis and Applications Proceedings of the IEEE, Vol. 77, No 4, April, 1989, pp. 541-580.

[NASA95] NASA Office of Safety and Mission Assurance: Formal Methods Specification and Verification Guidebook for SW and Computer Systems. Vol I.-II. NASA GB 002-95, Washington 1995

[SM01] A/Morzenti, P. San Pietro, "Technical Specification: Software Life Cycle for Safety-Critical Signalig systems, http://www.elet.polimi.it/upload/sanpietr/pubs/rfi.pdf, or http://www.rfi.it/direzionetecnicainternet/specificheis/techSpecSafeSignalSw.pdf.

[Spi88] J. M. Spivey, Introducing Z: a Specification Language and its Formal Semantics, Cambridge University Press, Cambridge, 1988.

[Sto96] N. Storey, Safety-critical computer systems, Addison Wesley Longman, Edimburg, UK, 1996.

The formal description technique LOTOS Elsevier Science Publishers B.V., 1989.