Météor: A Successful Application of B in a Large Project

Patrick Behm, Paul Benoit, Alain Faivre, and Jean-Marc Meynadier

Matra Transport International
48-56 rue Barbès 92120 Montrouge
tel : +33 (0) 1 49 65 70 00 fax : +33 (0) 1 49 65 70 93
{behm,benoit,faivre,meynadier}@matra-transport.fr

Abstract. The automatic train operating system for METEOR, the first driverless metro in the city of Paris, is designed to manage the traffic of the vehicles controlled automatically or manually. This system, developed by Matra Transport International for the RATP, requires a very high level of dependability and safety for the users and the operator. To achieve this, the safety critical software located in the different control units (ground, line and on-board) was developed using the B formal method together with the Vital Coded Processor. This architecture thus ensures an optimum level of safety agreed with the customer. This experience with the METEOR project has convinced Matra Transport International of the advantages of using this B formal method for large-scale industrial developments.

1 Introduction

Matra Transport International has developed railway-automated systems with very strong dependability and safety needs (Sacem, Val, Maggaly, Météor). The complexity of the new systems leads to realise most of the functions by software even for safety functions. The validation of the software design is therefore of the utmost importance to guarantee the absence of catastrophic situations. To prevent software design errors and to reach *zero-fault*, Matra Transport International has chosen the B method to develop and validate safety critical software. The formal process has been fully applied for the first time to develop the automatic train operation system, which equips Météor, the new metro line 14 in Paris (France).

1.1 Historic

The use of formal method in the french railway industry was introduced for the SACEM system, an automatic train control system asked by RATP, the transit authority in Paris, in the beginning of 1980. The consortium of manufacturers in charge of this project decided to develop single software using a new technology to secure it, the VCP[1] technique, instead of diversified software using the

[1] Vital Coded Processor

redundancy concept. The VCP technique consists in protecting each software information by a redundant code, checked at run-time. Because the software was not doubled, a *zero default* design was required. Moreover, validation by testing was not considered as sufficient by RATP; so, they asked manufacturers to use a new approach based on formal method.

The process was the following one:

1. the functional software requirements were re-written in a formal language;
2. the software source code, written in Modula 2 language, was completed with *pre-assertions* and *post-assertions* and checked with a partially automated program proof activity;
3. a check between formal specification and code assertions was manually performed.

Despite the heaviness of this process and its weak automation, the confidence achieved by this first application had convinced RATP. For the following tender, the Météor line, they demanded the use of formal method for safety critical software development. Matra Transport International was chosen in 1992 to develop the automatic train operation system for the Météor line, using the B method [Abr96] for the safety critical software.

1.2 Météor

The new metro line, number 14, in Paris (France), is in operation since October 15th 1998 between *Tolbiac-Nationale* at south-east of Paris and *Madeleine* at north-west. It is connected with two main junctions of the metro network, *Gare de Lyon* and *Chatelet*. This line was designed to reach traffic of 40.000 passengers per hour and per direction with an interval between train down to 85 s. on "peak hours". This new line is managed by the automatic train operation system developed by Matra Transport International. This automatic guideway transportation system allows to manage automatic driverless trains together with non-equipped manually driven trains. It was designed to meet strong requirements such as a high quality of service (user's point of view), an easy operation management and a strong safety level.

The automatic train system is made up of four main subsystems:

1. *PA-SIG*[2], the automatic pilot and signalling system,
2. the operation control centre,
3. the platform doors,
4. the audio-video system.

The *PA-SIG* is the only subsystem that includes safety critical software. This subsystem controls the running and stopping of every trains, and controls opening and closing of doors located in trains and platforms. It ensures that the speed

[2] Pilote Automatique et SIGnalisation

of trains is safely guaranteed and controls electrical traction power, routes, doors and sends back alarms from passengers to the operation control centre. The automatic train operation system has a distributed architecture with:

1. line equipment located in the operation control centre,
2. wayside equipment dispatched along the track,
3. on-board equipment.

All of them are linked together through a high-speed transmission network to exchange information. At any point of the track, wayside equipment and on-board equipment are also able to mutually exchange numerical information.

The operation control centre allows the operators to monitor the pre-established operating train programs for each day and to supervise all the events received by the network. To ensure their work, operators have:

1. an optical track diagram which displays on run time a detailed picture of the track with the position of every trains,
2. video screens which display what is happening inside trains or on platforms,
3. communication means to speak with passengers in train or in station,
4. terminals which allow them to act on the system by sending orders.

Platform doors prevent people from falling on to the track. Exchange of passengers between trains and platforms is authorised when trains are correctly stopped in stations. In this case, platform doors and train doors are just facing together. In case of failure, emergency doors allow evacuation of the passengers when the trains are not correctly stopped.

Audio and video communication allow the security of passengers with the help of supervision or direct communication between passengers and operators in the operation control centre.The system has to meet a high level of availability.

1.3 Safety Critical Software

The automatic pilots (on-board, wayside and line equipment) run on a safety computer using the VCP technique. The VCP environment delivers basic services such as inputs/ouputs, hardware/software interfaces and safe computing. According to safety analysis performed at system and equipment levels, the software of the automatic pilots is split into two products (Figure 1):

1. safety critical software which corresponds to vital functionalities,
2. non safety critical software.

Each of them is running on a specific microprocessor board. The safety critical software is sequential and cyclic (with a basic cycle of 360 ms), single task and non-interruptible. This feature enables the use of a formal method such as B.

In the other hand, considering a validated *Software Requirements* document, software failures can be classified into two categories:

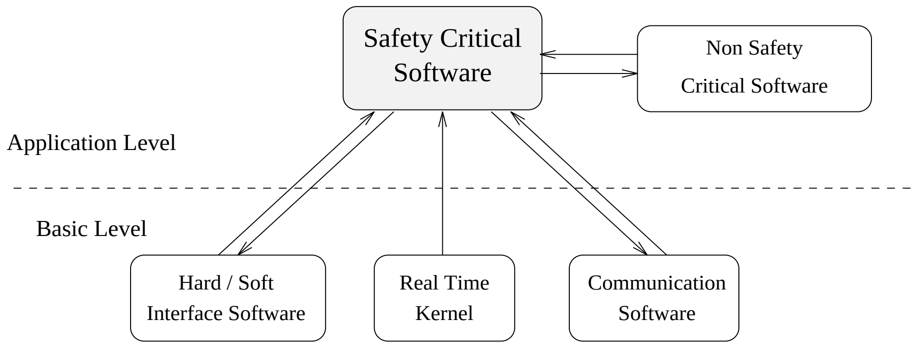

Fig. 1. Safety Critical Software of Météor

1. *design or coding errors.*
   In that case, the source code does not fulfil the *Software Requirements* document.
2. *errors in the code production chain (compiling, linking,...) or resulting from hardware failures.*
   In that case, program execution is not correct with respect to the meaning of the source code.

The VCP technique detects the second category of errors. The goal of using the B formal method is to protect against the first one by developing a single software program aiming to be bug-free.

2 Industrialisation of the B Method

Evidently, at the beginning of the Météor project, the B technology was far from being mature enough for a large industrial application. As it stood, B provided a set of mathematical modelling tools, formal validation obligations and a prototype toolkit to control and to prove rather small models. The first trials pointed out how great was the risk of elaborating, from the first level of abstraction, models very difficult to prove and even more difficult to refine into code, and revealed that the toolkit could not deal with large software components. In particular, the feasibility of a 100% proven development was not guaranteed.

Moreover, there was no answer to the question of introducing such a method into an existing lifecycle in the context of an industry. Usually, the realisation of safety critical software involves two teams:

1. the development team, which elaborates the specification of the software, designs it, and validates it, from a functional point of view;
2. the validation team, which carries out specific safety analysis, such as *Functional Safety Analysis* or *Software Errors Effect Analysis* to demonstrate that the execution of the software will not lead to a dangerous situation.

How the use of a formal method impacts this organisation and their two processes? Moreover, we were concerned about the level of education required for the use of B. Could our engineers be efficient with it, or is it reserved for experts?

An important work program was required and it was decided to go for the following objectives:

1. to define the new processes for the development and the validation teams, benefiting from the use of B;
2. to elaborate a methodological reference book enabling an efficient application of B, both for the design and the validation processes;
3. to bring the toolkit performances to the required level to handle large volumes of modelling, and particularly to improve proof techniques;
4. to link the method with the protection technique of code compilation and execution used at Matra Transport International.

In the next paragraphs we present the results of a 4-year program of industrialisation of B, associating RATP, Matra Transport International and Stéria Méditérranée.

2.1 The Development Process

The formal development cycle is shown in Figure 2 against the conventional development cycle.

This development cycle consists of the following two main steps:

1. the modelisation phase in which an abstract model of the informal software requirements is elaborated;
2. the design phase which, in successive steps, transforms the abstract model into a concrete one, ready to be translated into code.

The Modelisation Phase. The input document of this phase is the *Software Requirements* document which specifies the requirements to be met by the software product, in compliance with the design choices decided at the system, subsystem and equipment levels. It includes the functional requirements, i.e. what the software has to do, as well as operational and implementation constraints (e.g. the use of the VCP technique for the software product and the maximum time allowed for it).

This document is written in natural language. The aim of the modelisation phase is to formalise the functional requirements: an abstract model is elaborated using the B notation. This model must include all the functional features included in the *Software Requirements* document, regardless, as far as possible, of how it will be implemented afterwards.

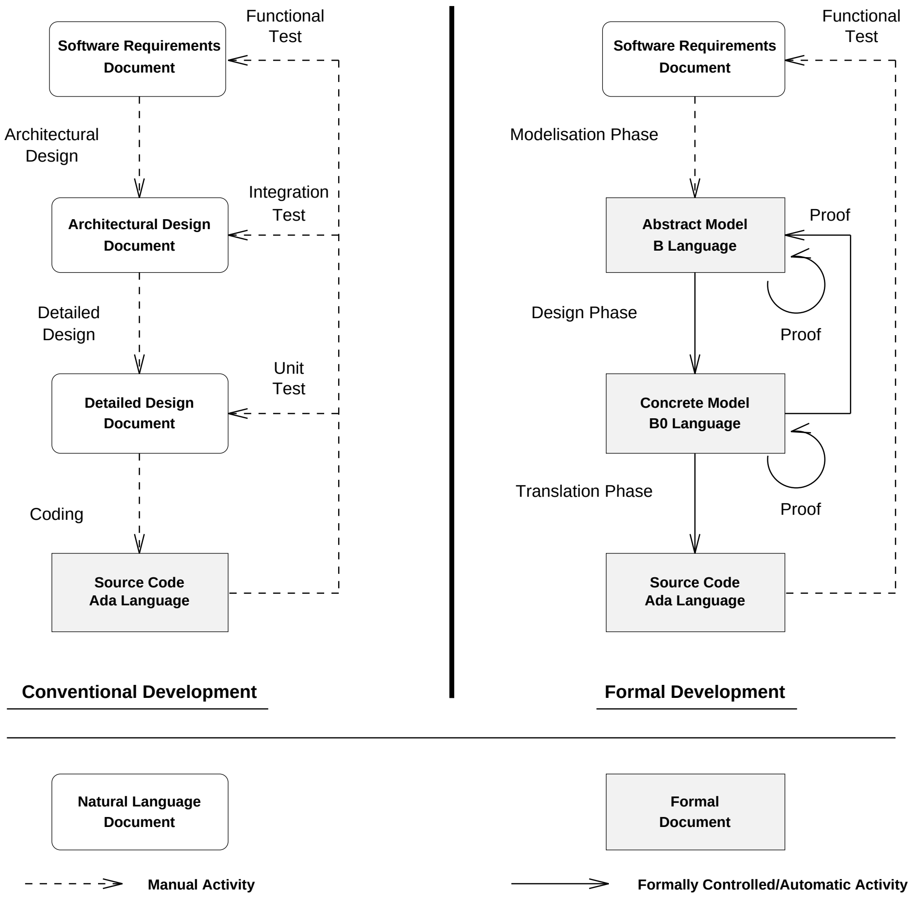

Fig. 2. Formal development cycle vs. conventional development cycle

Although the two lifecycles may look similar, there is an important difference with a conventional development. In a formal development, the abstract model is complete, i.e. no other functional information is required to implement the software product, and it gives only the highest level of the software architecture. On the contrary, the architectural design document of a conventional development gives most of the architecture, but is not concerned with the precise semantics of each box of the architecture.

Moreover, the abstract model includes properties, enabling a consistency proof of the model. The safety properties which are explicitly expressed in the *Software Requirements* document are formalised in the model. They may appear in the invariant clause of the abstract model if they must be satisfied by all the states of the system; if they deal with the dynamics of a particular function, they are included as post-conditions in the specification of the operation which formalises the function.

The consistency of the abstract model is carried out by means of the following tools:

1. the proof obligation generator; this tool produced all the mathematical theorems (called *lemmas*) to be proven;
2. the automatic and the interactive proof tools; the automatic proof tool attempts to prove each lemma. The automatic proof of the most difficult lemmas may fail; then, the interactive proof tool is used for these lemmas, the developer guides the proof by defining special proof tactics or by adding mathematical rules. These rules are validated within the framework of the formal validation activities.

As said before, the elaboration of an abstract model, completed with respect to the *Software Requirements* document, totally proven and good enough to be the firm basis for the design, is a very complex activity; it would not be effective without an efficient methodological reference environment.

A *B Method Book* [BMB99] was elaborated and enriched for several years to address this problem. It included guidelines for each step of the development. As far as the modelisation phase is concerned, the book includes:

1. rules to build the abstract model. In particular it provides :
   – restrictions to the B language (e.g. *Machines* without parameters),
   – architecture rules (e.g. the *INCLUDES* and *USES* clauses are not used),
   – specific rules for the modelisation of data and functions,
   – modelling examples relating to the notions often used in the *Software Requirements* document (e.g. several automaton model schemas are developed),
   – modelisation rules to ease the proofs (i.e. to reduce the number of lemmas produced and to get more simple lemmas).
2. rules to interface the formally specified software with software products which are not formally developed and/or reused;
3. guidelines for the interactive proofs.

The *B Method Book* enables better control of the modelisation phase. Beyond this step, it can be considered that an extremely hard to prove model stems from a defect of the model itself. This is why the rate of automatic proof is used as an indicator of the quality of the models: an abstract model must reach the 90% level of automatic proof before it is allowed to be implemented.

A very interesting demonstration of the efficiency of this book is that, in our experience, very few feedback coming from the design phase demand modifications of the abstract model, in spite of the difference of these two levels.

The Design Phase. The objective of this phase is to transform the abstract model into a concrete model, written in a subset of the B language, the B0 language, which enables automatic translation into code. During this phase, major design decisions have to be taken, fitting the operational requirements of the software product.

From a technical point of view, the refinement work from abstraction to pseudo-code includes several aspects:

1. data reification: the set of abstract data must be transformed into a set of data of basic type (integer, boolean, array). Links between these two sets are precisely given to enable the proof;
2. architecture refinement: the architecture of the abstract model has to be completed to get the implementation architecture;
3. structure refinement: the dynamics of the operations must be expressed using basic control structures (sequence, alternative, loop);
4. the validation of the concrete model with respect to the abstract one.

Again, our *B Method Book* provides a vital help to achieve the design phase. Starting from the idea that many design choices may be systematically decided, it describes the process the user has to follow:

1. the book recommends to complete first the architecture refinement in a very systematic way;
2. the structure refinement is then guided with predefined schemas for each substitution (*if*, *case*, non deterministic substitution, etc.); a special chapter explains how to build loops and how to implement abstract iterators;
3. the data reification is postponed as far as possible; for this step, a data reification dictionary is provided; it gives schemas of refinement for each type of abstract data which usually occurs in the abstract models, together with formal link between the abstraction and the suggested concrete data.

Validation of the concrete model with respect to its abstraction is again carried out using the proof obligation generator, the automatic and interactive provers. Proof activities are widely facilitated by the use of the standard schemas for architecture refinement and data reification of the *B Method Book*: specific tactics have been introduced for them. Again, the rate of automatic proof is used as a quality indicator: concrete models are reworked until 80% of the produced lemmas are automatically proven[3].

Translation into Ada Code. The translation of the concrete model, written in the B0 language, into Ada code is automatic and made by a double chain

[3] These rates express our current practice of the method in a given release of the toolkit. We hope they will continue to increase progressively as the power of the proof techniques and the efficiency of our methodological referential will improve.

of translators. The tool takes into account most of the *Vital Coded Processor* constraints, which were previously handled by the developer[4].

Evidently, the Ada code produced can be read, but not modified: otherwise the proof effort would be ruined. Beyond this restriction, the Ada code is considered as an intermediate language used to interface the compilation line: for the user's point of view, B0 is the implementation language.

2.2 The Validation Process

The validation process takes into account all the activities necessary to obtain the conviction that the critical functions are safely implemented. These activities follow and complete the safety analyses achieved beforehand on system, subsystem and equipment specifications.

This validation process springs from the following observations:

1. The proof activity, carried out during the formal development phase, guarantees the internal consistency of the first abstract B model and the consistency between the various phases of the formal development.
2. The proof activity, realised by automatic and interactive tools, is considered as a double chain, that is, a validation chain which is independent from the design activity of the B models carried out by the development teams.

In that context, the validation tasks to be performed are as follows:

1. verification of the abstract B model, resulting from the modelisation phase with respect to the informal *Software Requirements* document,
2. verification of interfaces between B models and operating features developed conventionally,
3. validation of all the mathematical rules, added during proof activities of the development phases,
4. verification of the translation of concrete B0 models into Ada source code,
5. functional test based validation.

This highlights the test based validation phase compared with that of a conventional development. In particular, the validation by proofs during modelisation and design phases makes unnecessary:

1. the unit tests, which validate the code of the procedures against their detailed design;
2. the integration tests, which validate, step by step, that the modules fit to one another, according to the design.

[4] Some constraints cannot be automatically carried out; they must be handled in the concrete model.

Furthermore, it is interesting to notice that one major advantage of the B method is to enable the validation during the development phases, and not after the elaboration of the code.

As for development activities, a *B Validation Book* [BVB99] was elaborated and enriched during the project. It details the validation activies and provides guidelines for each step of the validation.

Verification of Models Coming from the Formal Development Phase. This task guarantees the consistency and the completeness of the abstract B model coming from the formal development phase in comparison with the informal *Software Requirements* document. These requirements, and particularly the safety properties, can be converted into invariant properties or substitutions in the operations of the B models.

This is the most important step in validation activities since the following software products generated during design phase until software execution on a target machine, will be under the control of the B Method and then of the VCP technique.

This is also the most complicated step because of the informal aspect of the initial *Software Requirements* document which prevents any automation of that verification.

A detailed and complete analysis is carried out on B models compared with functional specifications. The proof activities guarantee that concrete B models stemming from the design phase strictly refined the validated abstract B models. Therefore, no additional activity is necessary to validate these concrete models.

Verification of B/Non-B Interfaces. Ada components produced by the formal B process can be interfaced with software components conventionally developed in Ada. Among these components are, for example, the operating features which are not classified as safety critical or data files which describe topology of railroad and which are calculated by others tools independently from the B development.

For each of these components, not expressed in B, and interfaced with B models, there is a corresponding formal specification, called basic machine. These basic machines contain all the information necessary for the proof of the B components which refer to them, that is, the assumptions necessary for the correct execution of the automatically generated code. Not observing these assumptions may generate a risk of inconsistency between the code corresponding to the basic machines, which has been manually generated, and the code of the components, which has been automatically generated through the toolkit.

Basic machines represent two main types of operating features:

1. Non safety critical operating features: from a safety point of view, nothing can be guaranteed concerning their results. In this case, the associated basic machines stipulate that the B components which call upon these operating features must be capable of accepting any execution result.
2. Safety critical operating features, not developed in B: in this case, the associated basic machines specify the properties expected by the proof from these operating features.

Example 1 ((Safety properties about stations and platforms)). A platform belongs to a single station and a station is associated with a maximum of two platforms.

Example 2 ((Safety properties about shunting.)). If the position of a shunting is not uncontroled, its value must be right or left.

A detailed and complete conformity analysis is equally carried out on corresponding Ada code compared with basic machines and *Software Requirements* document.

Validation of the Added Proof Rules. The rules added by the developers during the interactive proof step must be correct, that is, they must preclude wrongfully proving lemmas.

The validation of these rules includes various preliminary steps which make it possible to ensure that they are correctly written and really useful. Next, the actual mathematical proof is carried out.

In an initial step, a tool attempts to automatically prove the largest number of added rules. The rules which are not proven in this phase are then listed and undergo a manual mathematical demonstration which is as complete and as rigorous as possible: each step in the proof must be accompanied by an explicit reference to the *B-Book* (definition, theorem, property) or to general mathematical knowledge which corroborates it.

Any rule that cannot be proven either because it is false, or because it is too complex, can result from an incorrect formulation of the informal functional requirements.

Verification of Translated Ada Source Code. The safety of translation between B0 and VCP-Ada is guaranteed by the use of a double chain: two translators B0/VCP-Ada were independently developed. The correctness comparison between results issued on both sides of the double chain is assured by the VCP technique itself.

Functional Test Based Validation. The objective of the functional tests is to validate the code with respect to the *Software Requirements* document. Although this activity is redundant with all previous validation activities, we have kept it within the framework of Météor in order to assess the setting up formal methodology. This validation activity is divided between the development team for nominal tests and the validation team which completes test coverage with borderline cases. Notice that, if it were technically possible, tests on the abstract model, rather than on the code, could suffice.

2.3 Industrialisation of the Toolkit

The initial release of the toolkit we had at the end of 1994 was far from satisfying the needs of Météor. An ambitious program of evolution has enabled to take its performances to the level required. Among qualitative improvements, several evolutions of the B language were implanted (e.g. abstract constants, use of variables in B0). Especially, quantitative improvements have been realised from tests bench defined in terms of volume of formal models, time of processing and automatic proof cover. Among qualitative improvements, several evolutions of the B language were implanted (e.g. abstract constants, use of variables in B0). This last point is fundamental since the totality of the safety critical applications of Météor was expected to represent about 30,000 lemmas. A 10% increase of the automatic proof coverage saves 3,000 lemmas!

Table 1. Adequacy of releases for Météor

Release	*Date*	*Type checking of B models*	*Automatic and interactive proofs*	*Translation into Ada code*	*Safe translation into Ada code*
1.6	12/94	Inadequate	Inadequate	No translator	NO
2.4	3/96	Uncomplete	Inadequate	No translator	NO
2.9	9/96	OK	Difficult	OK	NO
3.3	10/97	OK	OK	OK	OK

On the basis of an internal benchmark of lemmas, here are the coverage of the automatic prover for the different releases of the toolkit.

Table 2. Number of lemmas to be proven interactively

Release	*Automatic proof coverage*	*Expected number of lemmas to be proven interactively*
1.6	35%	19,500
2.9	60%	12,000
3.3	80%	6,000

3 Météor

3.1 Organisation

The development and validation activities of the safety critical software are carried out by three teams: the development teams, the support team, the validation team.

Development Teams. Each safety critical software product (for wayside, onboard, control-centre equipment) is developped by one development team. A development team is in charge of the whole development of their software from the *Software Requirements* document to the software/hardware integration. A team includes about two third of engineers trained in B.

Concerning the formal model, the teams carry out the following activities:

1. *production of the abstract and concrete models.*
   Two main documents are used for this:
   (a) the *Software Requirements* document, to produce the abstract model,
   (b) the *B Method Book*, in order to use B efficiently.
   The models are elaborated in a rigorous compliance to these documents.
2. *cross-reading of the abstract model.*
   Every component of the abstract model written by a developer is cross-read by another developper of the same team in order to verify its conformity with regards to the software requirements.
3. *automatic and interactive proof of these models.*
   A developper makes the complete proofs of his models. The interactive proof begins only when the model is mature enough in order to avoid to make proof several times. During the interactive proof, the developper may add new mathematical rules.
4. *documentation of the models.*
   Two documents are produced. The *Abstract Model* document gives the traceability of the abstract model with the *Software Requirements* document and justifies the modelisation choices. The *Concrete Model* document mainly justifies the design decisions taken in the refinement steps.

Validation Team. Validation activities on safety critical software are realised by a software validation team in a RAMS department, strictly independent from development teams, the general mission of which is to guarantee the dependability (reliability, availability, maintainability and safety) of the automated transportation systems and automatic train control systems developed by Matra Transport International.

This team carries out the following activities:

1. *verificatin of consistency and completness of the abstract models with regards to the* Software Requirements *document.*

2. *verification of the interfaces between B and non-B.*
   The Ada code of each basic machine is verified to be conform with the corresponding B machine and with the *Software Requirements* document.
3. *validation of the added proof rules.*
   Each rule that is not proven by the *automatic rules prover* is proven by hand.
4. *verification of the proof.*
   All the proof generation process is validated. The proof activity is automatically run again in order to verify that all the lemmas are proven.

Support Team. This team is made of engineers expert on the B method. It carries out the following activities:

1. *support of the development and validation teams.*
   If one encounters any problem when using the method or the tools he can ask to the support team.
2. *reviews of the formal models.*
   Each component produced by the developpers is reviewed by the support team in order to verify its conformity to the rules of the *B Method Book*. It will guarantee in particular that the proof will keep feasible and as easy as possible.
3. *reviews of the added mathematical rules.*
   Each rule added by a developper is reviewed in order to verify that it is mathematically correct. At this step the verification is not complete: the reviewer just read the rule and convinces himself that the rule seems right. If the rule is considered as wrong, the developper will have to remove or change the rule and make another proof.
4. *cross-reading the abstract model.*
   Each component of the model written by a developper is cross-read by the support team in order to verify that it is provable and easy to read.
5. *reviews of the verification analysis.*
   Each analysis produced by the validation team during verification activities is reviewed by the support team in order to control its conformity to the rules of the *B Validation Book*.

Links Between the Teams. During the development of the software the support team works together with the development team. The development is considered as finished when:

1. the system and software requirements specification are considered as fully mature,
2. all the reviews have been passed,
3. all the proof have been done,
4. all the nominal functional tests have been passed successfully.

Then the products (formal models, added rules, ada sources) are given to the validation team.

The link between the support and the development or validation team is strong. On the other hand, development and validation teams work independently.

Training. Three courses are provided by the support team.

1. *Basic common course.*
   The objective is to make people familiar with B models, in order for them to be able to read and understand B models, on the basis of the *B-Book*. People from development and validation teams are attending this one-week course.
2. *Development course.*
   The objective of this one-week course is to teach how to elaborate large abstract models in B, to prove them and to derive proven concrete models, on the basis of the *B method Book*.
3. *Validation course.*
   The objective of this one-week course is to teach how to validate a development with B, on the basis of the *B Validation Book*.

4 Results

In the following paragraphs, we give figures and considerations about the three safety critical software products of the Météor project.

Ada. Table 3 gives the size of each software in lines of Ada (blank lines and commentary lines are excluded), instructions of Ada and Ada packages.

Table 3. Metrics about Ada

Software Product	*Lines of Ada*	*Instructions of Ada*	*Ada Packages*
Wayside	37,000	22,000	305
On-Board	30,000	20,000	160
Line	19,000	15,000	165
Total	86,000	57,000	630

B Model. Table 4 gives the size in lines of B (blank lines and commentary lines are excluded), number of lemmas generated and number of rules added. Moreover 556 common rules (not specific to a software product) have also been added.
It is interesting to note that we have about 1 lemma for 4 lines of B.
For each equipment, the number of lines of B is greater than the number of lines of Ada because it includes the number of lines of B in the concrete *and* the abstract model. The ratio between the number of lines of Ada and that of B0 is about 1.1.

Table 4. Metrics about B

Software Product	*Lines of B*	*Lemmas*	*Rules*
Wayside	50,000	13,600	477
On-Board	40,000	7,600	322
Line	25,000	6,600	1
Total	115,000	27,800	800

Proof. Table 5 gives the number of lemmas automatically proven by the tool customized by the 556 commun rules and the dedicated proof tactics and the number of lemmas proven interactively.

Table 5. Proof metrics

Software Product	*Lemmas*	*% Automatically Proven Lemmas*	*% Automatically Proven Lemmas Tool customized*	*Interactively Proven Lemmas*
Wayside	13,600	79 %	89 %	1,463
On-Board	7,600	84 %	90 %	775
Line	6,600	80 %	99 %	16
Total	27,800	81 %	92 %	2,254

Moreover, nearly 60% of the added rules are automatically proven by tools.

Proof of Abstract and Concrete Model. Table 6 gives the repartition of the number of lemmas between the abstract model and the concrete model.

Table 6. Proof metrics for the Abstract Model and the Concrete Model

Software Product	*Abstract Model Lemmas*	*Concrete Model Lemmas*	*% Abstract Model Lemmas*	*% Concrete Model Lemmas*
Wayside	2,400	11,300	18 %	82 %
On-Board	2,300	5,300	30 %	70 %
Line	1,500	5,000	23 %	77 %
Total	6,200	21,600	22 %	78 %

We can notice that the concrete model generates about 3/4 of the lemmas.

Errors Found by Proof. Many errors have been found during proof activities. Nevertheless, as development and proof are closely linked, no figure is available.

Several errors have been found by the validation team on the mathematical rules added. It is interesting to notice that they did not correspond to bugs in the models but only bugs in the proof: they were fixed without any change in the model and, consequently, in the Ada code.

Errors Found by Testing. We have obtained the following results:

1. functional validation on host computer: no bug found;
2. integration validation on target computer: no bug found;
3. on-site tests: no bug found;
4. since the line operates: no bug found.

This is clearly a very unusual situation: in comparaison with a previous unformal development of less complex safety critical software products, the different phases of validation have revealed several tens of errors. Several releases were necessary before the acceptance tests.

5 Conclusion

5.1 B: An Efficient Method for Industrial Applications

The Météor project has demonstrated how a formal development method such as B could be successfully applied in the context of a large industrial application. This success is established by several points.

B Is Not a Method Reserved for Experts. Even if a team of B experts has supported the development and validation works, most of the engineers of the development and the validation teams were newcomers in formal method (with only two weeks for training). Moreover, most of our engineers get quickly efficient and enjoy the method.

B Can Be Integrated in an Industrial Process of Development. In particular, the usual preliminary/detailed design phases can be substituted by the modelisation/design phases and the validation activies can be adapted. Moreover, very few feedback from the concrete models asked rework on the abstract models.

The Complete Proof of a Complex Software Is Feasible. 100% of proof obligations and added rules were proven, in spite of the large amount of lemmas and the complexity of many of them.

Using B, You May Expect that No Bug Will Be Found after the Development. Since no bug was found out by the validation work on host computer and target computer, nor during tests on the real line, the *zero-fault* objective can be claimed to be reached.

Formal Development Is Cost Effective. The formal development and validation were kept within the initial budget and delay. We have established that a formal development is not more expensive than a conventional development with a far better level of quality, regardless the gain obtained in the maintenance phase. This result is all the more significant as Météor is our first industrial application.

The main key points that led to the previous statements are the following:

1. a very careful preparation was needed to define how a formal method could be integrated into an existing organisation, taking into account the previous practices of software development and validation;
2. efficient methodological guidelines were developed and improved for development and validation activities: the *B Method Book* and the *B Validation Book*;
3. the toolkit was continuously improved to reach the operational level requested by Météor, thanks to a work program between Matra Transport International, RATP and Stéria Méditérranée.

Above all, the best evidence of success is that the use of B is generalised for the development of safety critical software in Matra Transport International.

5.2 What's Next

Producing the Concrete Models Automatically. An analysis of the Météor models pointed out that many refinements from abstract to concrete models were similar. Moreover, since all the semantics of the program are defined in the abstract model, it seemed possible to automatically produce in many cases the concrete models according to the abstract model. Promising results have already been obtained [BM99], allowing important cost saving.

Using B on Non Safety Critical Software. The development of the safety critical software of Météor using the B method does not make costs rise. Since reductions of cost are still possible (e.g. using the automatic production of the concrete models), it may be worth to use the B method even for non safety critical software. Experiments are in progress to demonstrate this point.

B at the System Level. The B method is currently used at the software level. Nevertheless, it seems very interesting to bring the formalisation effort up to the highest project level, in order to better prepare the system architecture and to rigorously check its validity. Others experiments, using extensions of the B language, are at the moment in progress on system level.

References

[Abr96] Abrial J.R., *The B-Book* (1996).

[BMB99] Burdy L., Dollé D., *B Method Book*; Internal document ref. DRL/XT/39.2617.98/DD/AA (1999).

[BVB99] Faivre A., Milonnet C., *B Validation Book*; Internal document ref. DSF/XT/32.1374.96/CM/CM (1999).

[BM99] Burdy L., Meynadier J.-M., Automatic Refinement; BUGM at FM'99 (1999)