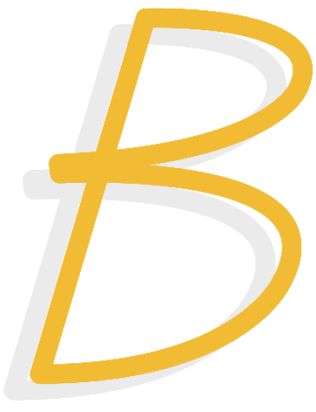
WHITE PAPER Mai 2020

FORMAL METHODS IN ACTION IN THE RAILWAYS

CLEARSY
Safety Solutions Designer

Formal methods and industry are not so often associated in the same sentence as the former are not seen as an enabling technology but rather as difficult to apply and linked with increased costs. In the 1990s, the introduction of the B method and the Event-B language into several industrial development processes was witnessed with more or less success, when new tools and new practices were available to ease acceptance by industry. At that time, a number of research projects and non-trivial industrial applications had backed these two formal methods. Almost 10 years later, after several real size experiments in diverse application domains, the situation has slightly evolved and this white paper intends to make clear how the B method, the Event-B language and the Formal Data Validation have contributed to a safer world.

The B Method



The B Method was introduced in the late 80's to correctly design safe software.
It is a formal method to develop software mathematically proven to comply with its specification. It relies on a mathematical model of the software, containing both what the software is expected to do and its algorithm.
The software model is decomposed into smaller models in order to manage the complexity ("divide and conquer").
The model is proved: the algorithm doesn't contradict its specification.
The software code is generated from the implementation model. Code is readable, very close to the model and is easily checked. The final software application is made of parts developed with B and parts not developed formally.

The main idea was to avoid introducing errors by proving the software while being built, instead of trying to find errors with testing after the software was produced. Promoted and supported by RATP[1], B and Atelier B have been successfully applied to the industry of transportation, through metros automatic pilots installed worldwide. Paris Meteor line 14 driverless metro is one of the main reference applications with over 110,000 lines of B models, translated into 86 000 lines of Ada. No bugs were detected after the proof was completed, neither at the functional validation, at the integration validation, and at the on-site testing, nor since the beginning of the metro line operation (October 1998). For years, Alstom Transportation Systems and Siemens Transportation Systems (representing a major part of the worldwide metro market) have been the two main industrial players in the development of safety-critical B software. Both companies have a product based strategy and reuse as much as possible existing B models for future metros. As an example, the Alstom Urbalis 400 CBTC (Radio communication based train control) equips more than 100 metros in the world, representing 1250 km of lines and 25 %[2] of the CBTC market.

For such applications including driverless metros, B modelling is used for safety critical functions for both trackside (zone controller, interlocking) and on-board (automatic train pilot or ATP) software. The interlocking part has to avoid having two trains on the same track section. It computes Boolean equations that represent the tracks status as seen from diverse sensors. The automatic pilot is mainly in charge of triggering the emergency brake in case of over-speed. It requires several functions such as the localisation that involve several graph-based algorithms, and the energy control which computes the braking curve of the train, based on the geometry of the tracks. Data types used are integer for the energy control, Booleans for the interlocking and tables of integers for the tracks.

[1] Paris metro authority

[2] Source : http://www.alstom.com/products-services/product-catalogue/rail-systems/signalling/products/urbalis-400/

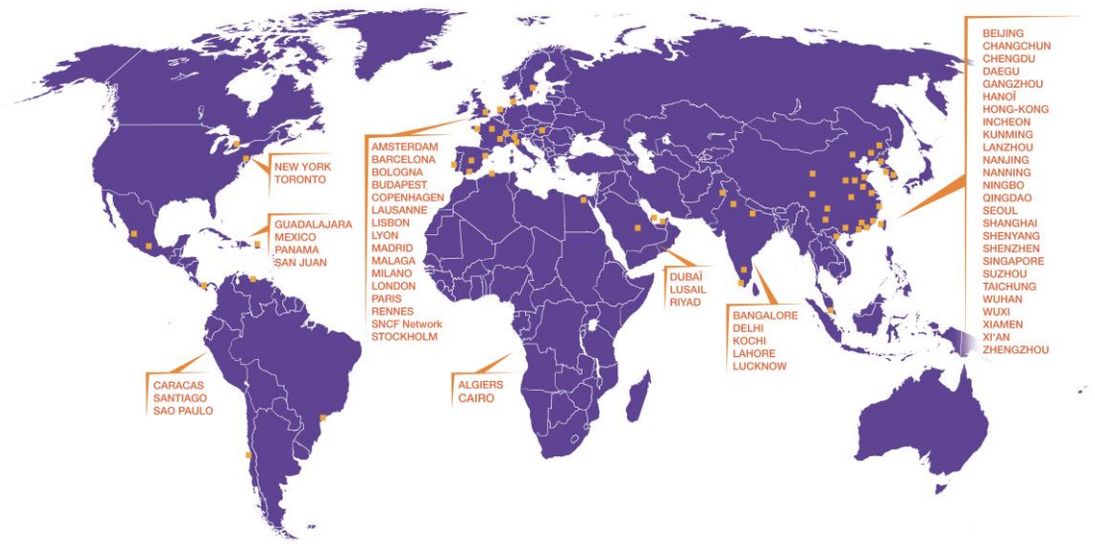


Figure 1: automatic driving metro subsystems, based on the B method, installed worldwide (Alstom Urbalis, Siemens Mobility Trainguard)

To date, the biggest B software is an XML compiler enabling the execution of safety critical embedded applications by an interpreter. The B models generate more than 300,000 lines of Ada code, for this SIL4 T3-compliant (EN50128) program. The method is not limited to 300,000-lines of software code and has not met any bottleneck until now. Therefore, the method is likely to scale up to larger, non-threaded software.

At the other end of the scale, with platform screen doors (PSD) or remote inputs/outputs controllers, less demanding in terms of computation, smaller applications are generated for both programmable logic controllers (PLC) and PIC32 microcontrollers, with a maximum of 64 KB in memory per software. SIL3 and SIL4 controllers, in charge of opening and closing platform screen doors have been (or will be) installed in Paris (L1, L4, L13), Stockholm (Citybanan) and Sao Paulo (L2, L3, L15 Monorail).

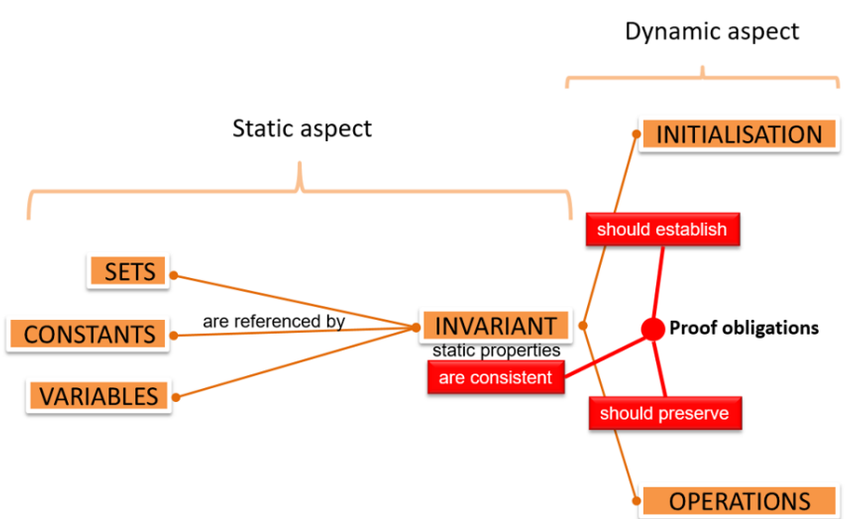


Figure 2: the relations between the B modelling elements

This modelling approach is slightly specific but comes along many other interesting features. The “specification before code” motto imposes a top-to-bottom approach (or by decomposition). Software developers are encouraged to specify first, from natural language requirements. It does not prevent reuse of existing software but avoids asking the dangerous question “what do I get if I gather all these software components together?”

The target software is cyclic and mono-threaded. No interrupt should modify the state variables. Full integer arithmetic is supported (non-trivial floating-point arithmetic is practically not provable) as well as Boolean predicates and equations (and arrays of integers and Booleans).
The models are text-based. The same mathematical language (B) is used for the specification model and the implementation model, based on the set theory and predicate logic. The model contains both the software properties (the static aspect) and its behavior (the dynamic aspect). A proved model means that the specification is consistent (no contradiction) and the implementation complies with its specification. *A minima*, the software is proved to be programming error-free.

There are many reasons to use B for safety critical software development:

- Improved level of confidence, brought by the mathematical formalism and the proof. The use of B removes ambiguities as the mathematical model captures the meaning of the software.
- Early error detection. Errors are discovered by proof during the modelling and not by test once the software is built.
- Most testing is useless. Proof replaces testing. Mathematical proof is exhaustive while testing is not.
- Avoid redundant software development. For highest safety integrity level, only one B model is required, compared with two software applications developed more traditionally by two independent teams.
- Accepted for certification. Several industrial standards recommend or strongly recommend the use of formal methods (EN50128, IEC61508).

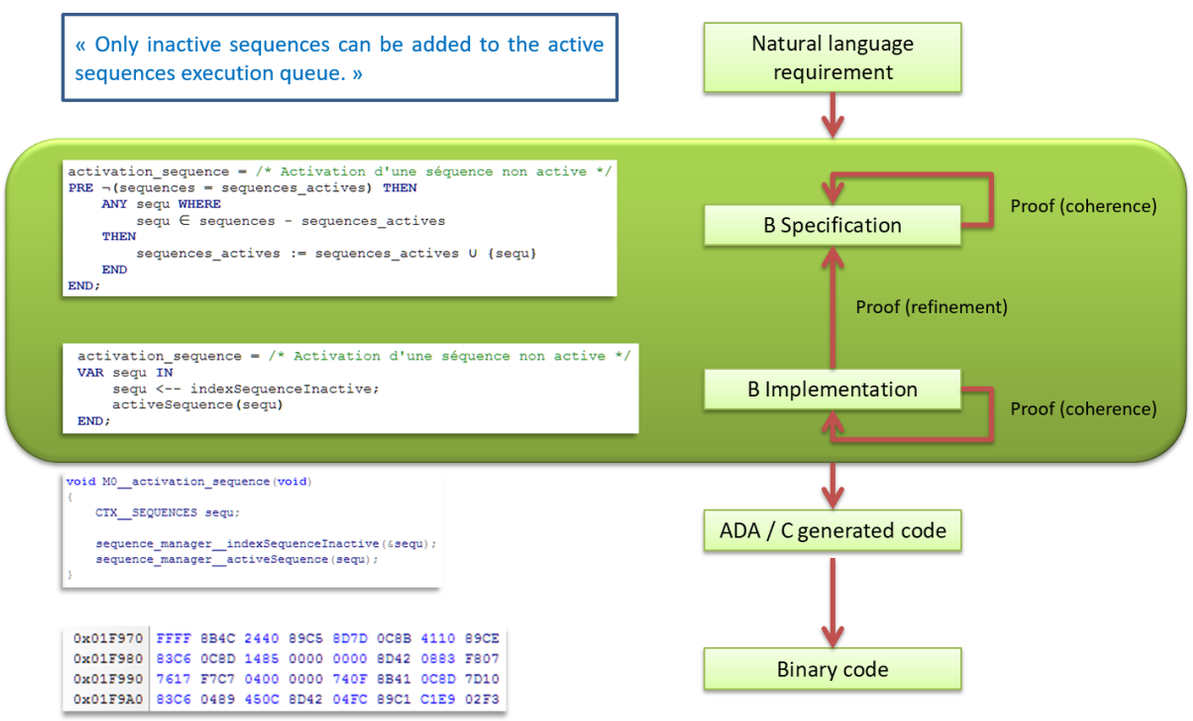
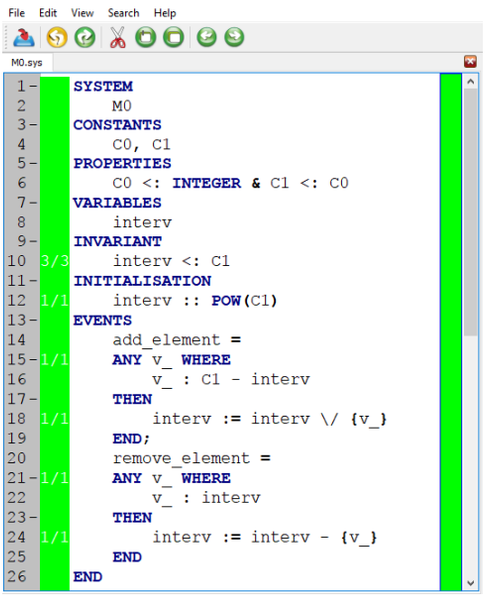


Figure 3: the complete path from requirements to binary code.
The proof-by-construction principle applies to the green area (specification, implementation). Conformance crosschecks (requirements and source code) require other means.

The Event-B Language



A broader use of B appeared in the mid `90s, called Event-B, to analyse, to study and to specify not only software, but also systems[3]. It extends the usage of B to systems that may contain software, hardware and equipment, environment, and also to intangible objects like process, procedure, business rule, etc. In that respect, one of the outcomes of Event-B is the proved definition of systems architecture in their environment and, more generally, the proved development of, so-called, “system studies", which are performed at the beginning, before the specification and design of the software. This enlargement allows to perform failure studies right from the beginning, even in a large system development.

Event-B is used for formal modelling to progressively analyse and verify by proof system-level specifications. It relies on a mathematical model of the system, containing both the properties of the system and its evolution rules. The evolution rules are encoded in the form of a collection of asynchronous events that may be triggered based on conditions and may modify the system state variables. The modelling is progressive as the model is made more and more detailed, and complexity is added gradually. The top-level model is simplified (abstracted) with few state variables modelled. Modelling details are added to successive models (refinements). Events and properties have to be rewritten to consider these details. For example, a train can be seen as a point moving on a line then can be refined by adding details like the number of cars, the length of train, its braking capability, the traction model, etc.

“The model is proved” means that the evolution rules enforce the properties of the system.

Similarly, there are many reasons to use Event-B to model systems:

- Improved level of confidence. It is brought by the mathematical formalism and the proof. It enables the assessment of complex specification (structure, behaviour) in the early stages. The model may be checked against scenarios. Finally, better software specifications are derived from this modelling.
- Ambiguities are removed. The mathematical model captures the meaning of the system specification.
- Easier test definition. The modelling allows defining which tests have to be performed for subcomponents acceptance and before daily operation.
- Accepted for certification. Several industrial standards either recommend or strongly recommend the use of formal methods (EN50128, IEC61508), or require the use of formal methods (Common Criteria).

[3] system is here considered in its widest definition

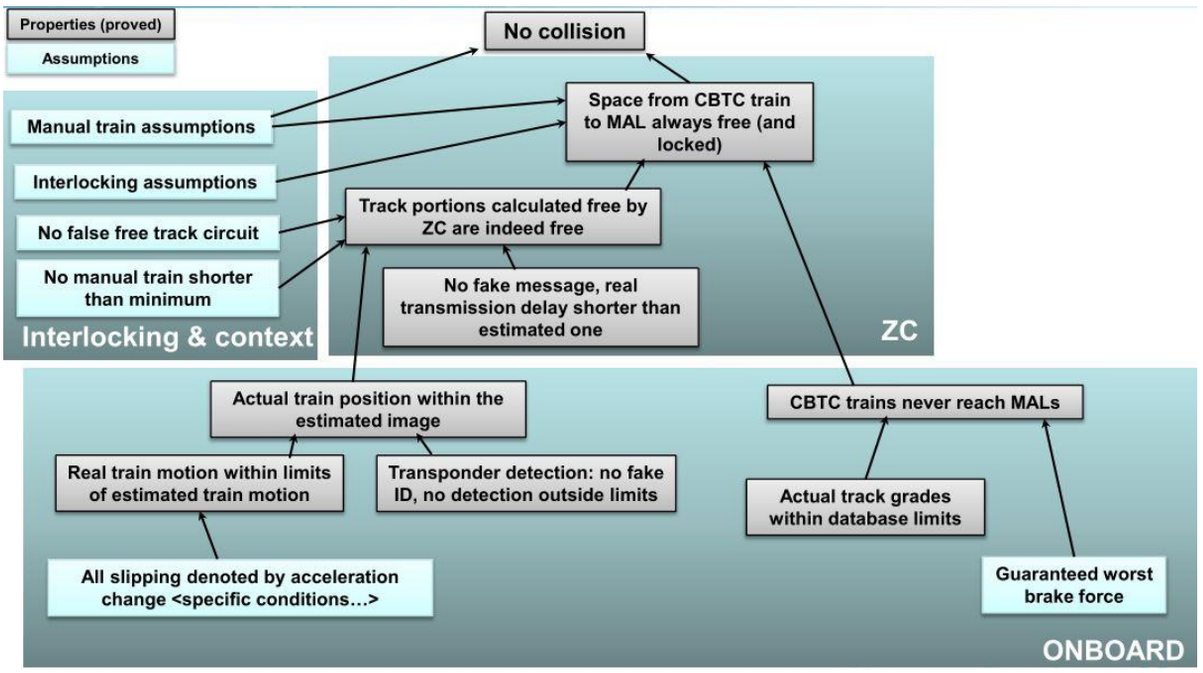


Figure 4: NYCT line 7 modernisation project - the structure of the formal proof for the main safety properties of the system: no collision and no over-speeding. Event-B/Atelier B was used to support the mathematical demonstration.

Event-B is used in a number of safety cases. The fundamental goal is to extract the rigorous reasoning establishing that the considered system ensures its requested properties and is safe, and to assert that this reasoning is correct and fully expressed. At system level, this rigorous reasoning involves the properties of different kind of subsystems (from computer subsystems to operational procedures), that the formal proof shall all encompass. Event-B is used to formalise the reasoning with a collection of separate models: each model is readable and understandable by a non-expert without digging into hundreds of events and tens of refinement levels. This approach was used for the formal system verification for the CBTC of New York subway line 7 in 2012 and Flushing in 2014 (effort divided by two due to models reuse). It was also deployed by SNCF to design a new signalling system, based on a degraded version of ERTMS, aimed at low-traffic, regional lines. At this moment, RATP is making use of it for the formal system verification of the CBTC of Paris subway line 4.

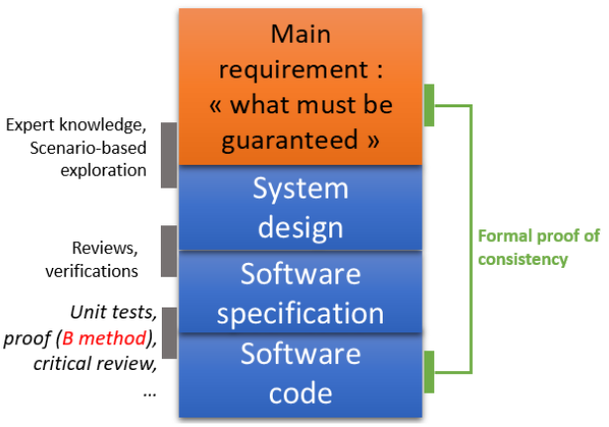
The formal software analysis

CLEARSY proposes a new innovative analysis approach to establish with mathematical proof that all or part of a software is compliant with respect to a functional or a safety requirement.

This approach establishes a direct formal link between the source code of the software and the properties of the system that integrates that software. It is now possible to detect any kind of noncompliance that may have been introduced in the design phase: from the identification of algorithms during the system definition phase, up to their concrete realization, taking into account possible implementation specific constraints.

This approach is particularly suitable when the traditional verification and validation activities show to be lacking:

- Scenario-based verification is possibly incomplete for systems having too many states.
- Bugs are discovered late in the development cycle, especially when they stem from errors from the system design.



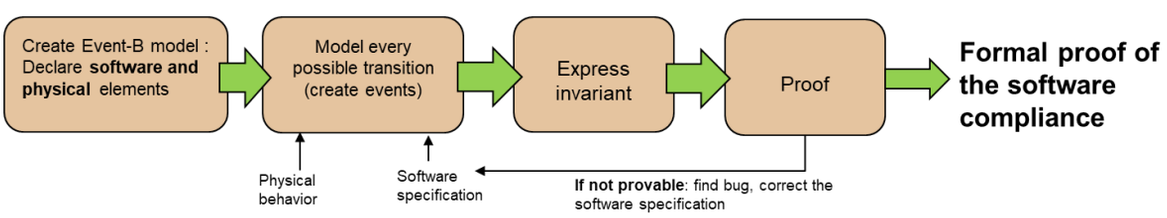
In comparison, the formal analysis approach proposed by CLEARSY is complete: it is guaranteed to cover all possible functional behaviors, including as a matter of fact all system dysfunctions and failures that have not explicitly been discarded. This is the benefit of using a method based on mathematics and a property-based approach, instead of a case-by-case approach.

To do so, the formal software analysis method consist in building an event-b model in which both physical and software elements are described and every change of state is modelled as an event of the b model:

- Software variables: every possible transition is described by the software specification.
- Physical elements: it requires to model its possible behavior.

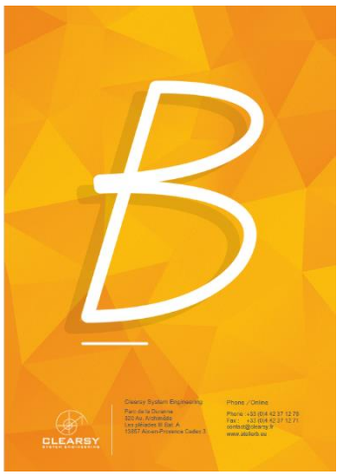
The expected property is expressed as an invariant of the model and the formal proof of compliance is obtained using inductive reasoning:

- The initial state should be compliant to the invariant.
- Then, every possible transition (which means every event in the b model) shall ensure that it will not break the expected property.



- Figure 8: Event-B modelization and proof process

Atelier B



Atelier B is the reference tool from CLEARSY, freely available and fully functional:

- for the development of (safety critical) software. It supports the B method and the B language.
- for the modelling of systems. It supports the Event-B language.

It includes several model editors, proof tools and code generators. It has been used for certified applications up to SIL4 (EN50128) in the railways and EAL6+ (Common Criteria 3.1) in the smartcard industry.
A dedicated support (more frequent Atelier B releases, privileged access to beta features for evaluation, short-term bug correction) is provided for maintenance contract holders. CLEARSY also proposes training courses and services to support its customers.

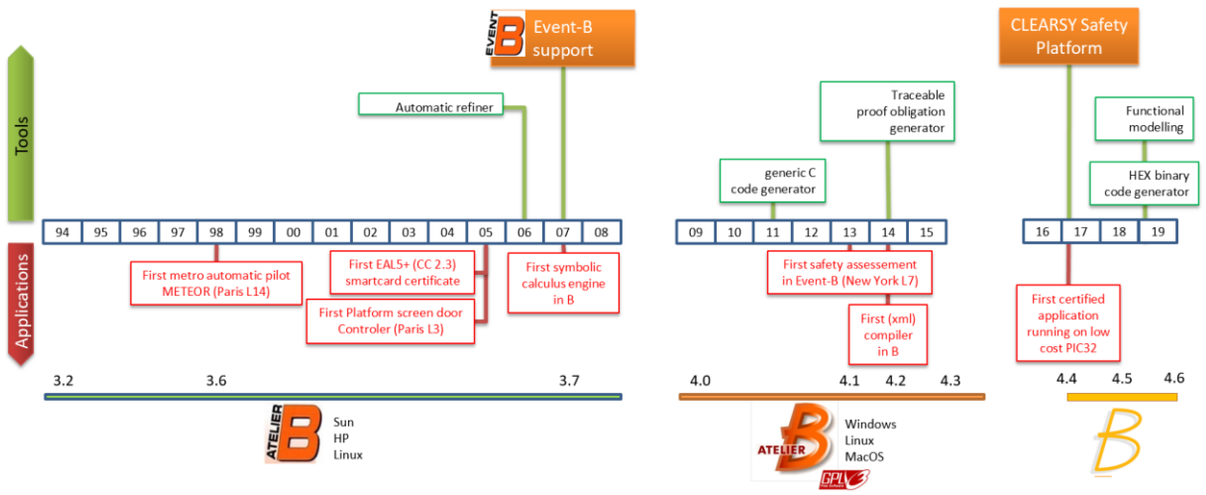


Figure 5: Atelier B timeline showing the major improvements (in green) and the first new kinds of applications (in red), since its creation.

Formal Data Validation

In the railways, software applications are usually developed and validated independently from the parameters or constant data that fine-tune their behaviour. For example, the track topology, signal and point positions, kilometer points, etc. are constant data used by an automatic pilot to compute braking curves and to determine when to trigger the emergency brake. In order to avoid a new compilation if the data are modified but not the software, two different processes define the software and the data validations.

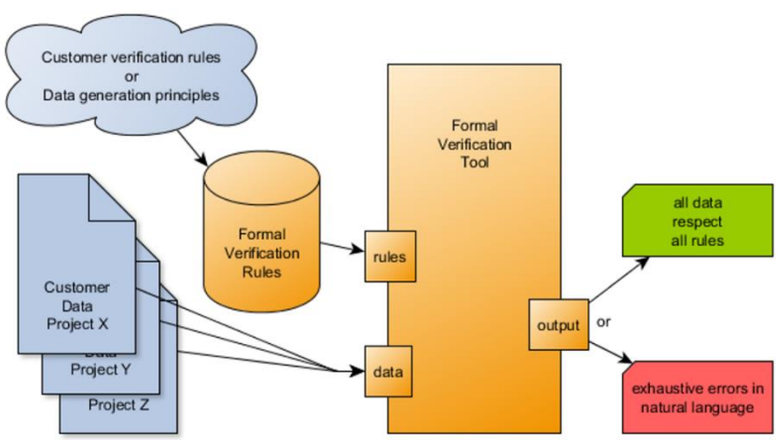
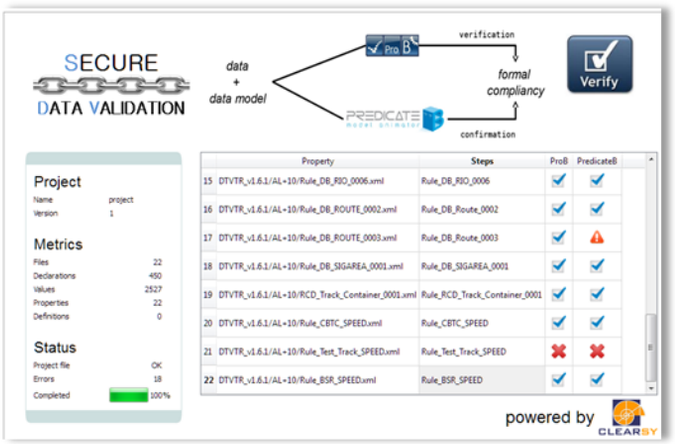


Figure 6: Formal data validation scheme

Data validation consists of checking an heterogeneous data collection[4] against a set of properties / rules[5] issued from regulation, exploitation constraints, train manufacturer product design, etc. Manual data validation used to be entirely human, leading to painful, error-prone, long-term activities (requiring several months to check manually up to 100,000 items of data against 1,000 properties / rules).



Formal data validation is the natural evolution of this human-based process into a more secure one where:

- the properties / rules are formalised, to constitute a formal data model (mathematical, based on the B language). It is built from natural language inputs.
- the verification of conformance between the data collection and the formal data model is performed by a formal tool (or by a combination of redundant formal tools if required)

This approach has been invented by CLEARSY, thanks to its deep knowledge and skills on formal method technology and associated tools.
The benefit of this formal approach is diverse:

- It is fast: up to 10x faster than a pure human verification, a couple of hours is enough for validating a complete railway project
- It is automatic, push-button and repeatable at will
- It removes human errors, as it makes use of certified formal techniques
- It allows a strong reuse from one project to another (capitalisation of the knowledge)

Formal data validation is industry ready. Several major players currently have deployed it like:

- Alstom – more than 20 metros (Urbalis) and tramways
- General Electric – for the Singapore underground
- RATP (Paris metro authority) – several metros in Paris
- SNCF (French Railways) – for checking the interlocking tables on the main lines (Mistral NG) and for checking balise data on ERTMS freight corridors
- Siemens Mobility – for metros (Trainguard)
- Thales – for metros

[4] CBTC or ETCS configuration data, IXL or RBC parameters, etc.

[5] For example, “successive track circuits should have continuously increasing kilometer points”, “there exists a path between two distinct track circuits”, “signals should be positioned a minimum 100 meters before the point they protect”, etc.

CLEARSY Safety Platform

The CLEARSY Safety Platform is aimed at easing the development and the deployment of safety critical applications, up to SIL4. It relies on the smart integration of formal methods, redundant code generation and compilation, and a hardware platform that ensures a safe execution of the software.

The CLEARSY Safety Platform is made of an integrated software development environment (IDE) and a hardware platform that natively integrates safety principles. Hence the developer has only to focus on the functional design while mathematical proof replaces unit and integration testing. There is no need for independent software development teams: redundant software is automatically produced from a single model.

Provided with a certification kit, the CLEARSY Safety Platform obviously lowers the cost to develop, certify and deploy a safety critical application. The hardware platform is available either as a starter kit or as a daughter board to be integrated into in-house developments.

"The safety principles are out of reach of the developer who cannot alter them"

The safety principles are built-in, both at software level and at hardware level (2OO2 hardware, 4OO4 software).

The functional correctness is ensured by mathematical proof. The detection of any divergent behaviour among the two processors and the four instances of the software is handled by the platform. The safety verification include cross checks between software instances and between microcontrollers, memory integrity, microcontroller instruction checker, etc.

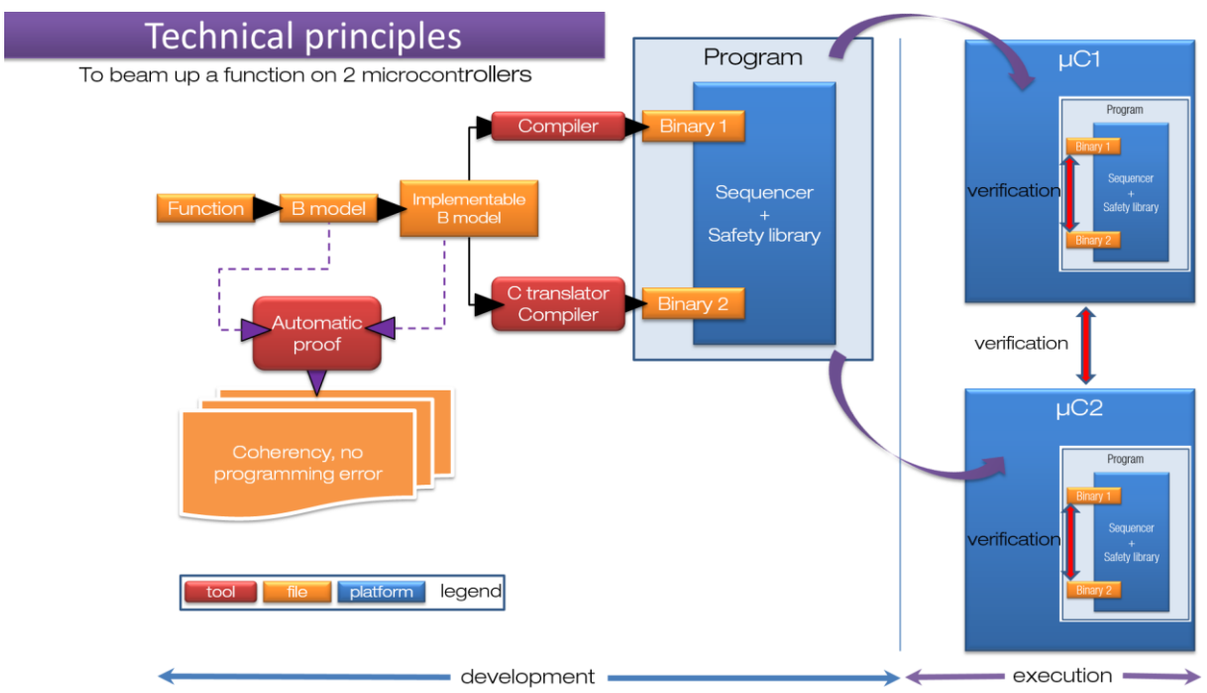


Figure 7: Development and deployment

The B formal method, at the core of the development process, reduces development, deployment and certification costs. Mathematical proof ensures that the software complies with its specification and guarantees the absence of programming errors while avoiding unit testing and integration testing.

Moreover, only one functional model is used to produce automatically the redundant software, avoiding the need to have two independent teams for its development. The formal model may be developed manually or be the by-product of a translation from a Domain Specific Language to B (BXml open API).

The CLEARSY Safety Platform targets control-command applications. In its current form, it allows developing cyclic applications (read inputs and current time, perform computations, command outputs), run directly on the hardware without any underlying operating system. There is no predefined cycle time to comply with: the application is run as fast as possible and the time information is managed directly by the application software.

With PIC32 microcontrollers, the platform offers up to 100 MIPS for lightweight applications handling Boolean and INTEGER data.

The building blocks of this technology have already been certified (SIL3/SIL4) in several railway projects worldwide, in particular the COPPILOT systems (open and closing platform screen doors) in São Paulo and Stockholm.

Once every event is proved, it is guaranteed that the modelized system can never reach a state which is not compliant with the invariant. As a consequence, the software behaviour is compliant with the expressed expectation.

Bibliography

- Benveniste, M.V.: On using B in the design of secure micro-controllers: An experience report. Electr. Notes Theor. Comput. Sci. 280 (2011)
- Hansen, D., Schneider, D., Leuschel, M.: Using B and ProB for data validation projects. In: Abstract State Machines, Alloy, B, TLA, VDM, and Z - 5th Int'l Conf., ABZ 2016, Linz, Austria, May 23-27, 2016
- Lecomte, T.: Safe and reliable metro platform screen doors control/command systems. In: FM 2008: Formal Methods,15th Int'l Symposium on Formal Methods, Turku, Finland, May 26-30, 2008
- Lecomte, T.: Applying a formal method in industry: A 15-year trajectory. In: Formal Methods for Industrial Critical Systems, 14th Int'l Workshop, FMICS 2009, Eindhoven, The Netherlands, November 2-3, 2009
- Lecomte, T.: Double cœur et preuve formelle pour automatismes sil4. 8E-Modèles formels/preuves formelles-sûreté du logiciel (2016)
- Lecomte, T., Burdy, L., Leuschel, M.: Formally checking large data sets in the railways. CoRR abs/1210.6815 (2012)
- Sabatier, D.: Using formal proof and B method at system level for industrial projects. In: Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification - 1st Int'l Conf., RSSRail 2016, Paris, France, June 28-30, 2016
- Lecomte, T., Deharbe, D., Prun, E., Mottin, E.: Applying a Formal Method in Industry: A 25-Year Trajectory In: SBMF 2017, Recife, Brazil, November 29 – December 1, 2017
- Lecomte, T.: Disruptive Innovations for the Development and the Deployment of Fault-Free Software In: FM 2018, Oxford, UK, July 17, 2018

Executive Summary

Formal methods are now industry ready, able to scale up to real size railway projects and to provide a real support for successfully completing safety demonstrations.

In this white paper, we show that several formal methods (B, Event-B, formal data validation, formal software analysis) directly contribute to safety critical software and system development, system-level specification analysis and constant parameters validation, with the help of mathematical proof.

Based in Aix en Provence, Lyon, Paris and Strasbourg, CLEARSY is a French SME, which specialises in developing SIL1 to SIL4 level safety systems and software. The company develops complex systems, from their design to putting them into service, and undertakes the interim stages of validation, checking and safety testing. CLEARSY is particularly active in the development of systems and software in the railway, car, military, nuclear energy and space industries.

http://www.clearsy.com/en/

http://www.fersil-railway.com/en/

http://www.atelierb.eu/en/

CLEARSY
Safety Solutions Designer