






# The First Twenty-Five Years of Industrial Use of the B-Method

Michael Butler<sup>2</sup> , Philipp Körner<sup>1</sup>  , Sebastian Krings<sup>1</sup> , Thierry Lecomte<sup>3</sup>, Michael Leuschel<sup>1</sup>  , Luis-Fernando Mejia<sup>3</sup>, and Laurent Voisin<sup>4</sup> 

<sup>1</sup> Institut für Informatik, Universität Düsseldorf  
Universitätsstr. 1, D-40225 Düsseldorf, Germany  
{p.koerner,sebastian.krings,leuschel}@hhu.de

<sup>2</sup> University of Southampton  
University Road, Southampton, SO17 1BJ, UK  
mjb@ecs.soton.ac.uk

<sup>3</sup> CLEARSY  
320 avenue Archimède, 13100 Aix en Provence, France  
{thierry.lecomte,fernando.mejia}@clearsy.com

<sup>4</sup> Systereel  
1090 rue René Descartes, 13100 Aix-en-Provence, France  
laurent.voisin@systereel.fr

**Abstract.** The B-Method has an interesting history, where language and tools have evolved over the years. This not only led to considerable research and progress in the area of formal methods, but also to numerous industrial applications, in particular in the railway domain. We present a survey of the industrial usage of the B-Method since the first toolset in 1993 and the inauguration of the driverless metro line 14 in Paris in 1999. We discuss the various areas of applications, from software development to data validation and on to systems modelling. The evolution of the tooling landscape is also analysed, and we present an assessment of the current situation, lessons learned and possible new directions.

## 1 Introduction

The B-Method [4] for software and systems development and its successor Event-B [6] has a rich history. B has originally been developed as a successor to Z [10] by Jean-Raymond Abrial in the 1990s, focusing on two key concepts: refinement to gradually develop models and tool support, in particular proof and code generation.

More concretely, B is based on first-order-logic and set theory and follows the correct-by-construction approach. A formal B model consists of a collection of B machines. Each B machine may contain constants with properties and variables with invariants. The state of a B machine can be modified by operations, which may have preconditions associated with them. The invariants are a crucial concept of B, stipulating properties which hold initially and which must be preserved

by every operation. B machines can be refined, whereby, e.g., datatypes can be replaced by more concrete ones or non-determinism can be reduced or removed. A refinement machine is linked to an abstract machine via a gluing invariant, which stipulates how its states correspond to states in the abstract machine. An implementation machine is a refinement machine written in the B0 subset of the language. The B-Method also describes how to derive proof obligations to ensure that the invariants always hold and that every refinement machine correctly realises its abstraction. If all proof obligations are discharged, the implementation machines are “correct by construction”: they correctly implement the initial abstract B specification.

As of today, the industrial applications of B can be classified into three categories:

- the original B for software development (classical B) [4]: refine specifications until a low-level subset of B (B0) is reached where code generation is applied.
- B for system modelling (Event-B) [6]: model an entire system, not just a particular software component and then verify critical properties and understand why a system is correct.
- B for data validation: express properties in B and check data and configuration parameters in a certified manner.

In this article, we discuss these three classes of applications in turn (Sections 2, 3 and 4), focussing on railway applications where B had its most significant impact. We then discuss applications in other areas in Sect. 5, the tools behind the industrial applications in Sect. 6 and finally conclude with lessons learnt over the years in the final Sect. 7.

## 2 B for Software: The Early Days and Industrial Uptake

In the 80s, RATP (Régie Autonome des Transports Parisiens, Paris railway transport operator and infrastructure manager) and the consortium<sup>5</sup> in charge of the development of SACEM, the train protection system deployed on the Parisian RER Line A, faced the validation of the first control/command software of a safety critical railway signalling system ever operated in France. Without a real background in that domain they started using a tool based on Hoare Logic to verify assertions included in the code. Then they consulted Jean-Raymond Abrial who proposed the formalisation and verification of a formal specification of the software with what can be considered as a sketch of the B-Method. His proposal was accepted, validation engineers were trained, a formal specification was written and verified. Eventually, in 1988, SACEM was put into operation to the satisfaction of all.

In the same year, Abrial presented “The B Tool” [2] with an unnamed syntax. Most of the further developments, both on the language and on advanced tooling, were initiated during industrial projects in Paris.

---

<sup>5</sup> Consisting of Alstom (today Alstom), Compagnie des Signaux (today Hitachi), and Matra Transport (today Siemens Transportation, France).

After the SACEM experience, RATP, guided by Claude Hennebert, requested the use of the B-Method for the development of the safety critical software of the train protection system of the driverless Paris Métro Line 14. Alstom, with this project in mind, but also for their own developments, decided to use the B-Method. These are the origins of the development and use of the B-Method, and of formal methods in general, in the French railway industry.

In 1989, Alstom, RATP and SNCF (Société Nationale des Chemins de fer Français), willing to industrialise the B-Method, launched a project whose purpose was threefold: firstly, to train engineers in the principles of the method, secondly, to develop tools to support it and, thirdly, to create methodological guides to use it. This project, funded partially by the French government and driven by Alstom, established a close collaboration with J.-R. Abrial and the team of Ib Sørensen at British Petroleum then at B-Core. Additionally, Abrial was still in contact with a research group in Oxford and certainly was influenced in technical details.

After some training sessions given by J.-R. Abrial, an Alstom team started the development of railway applications with the first version of the B language and tools provided by J.-R. Abrial and his colleagues. The fundamental syntactic and semantic concepts of the language (VARIABLES, INVARIANT, INITIALISATION, OPERATIONS) were already present in this version, but in 1991 it turned out that evolutions of the language and tools were necessary to structure, analyse and prove software of industrial size and complexity. This is the reason why structuring (INCLUDES, IMPORTES, PROMOTES, EXTENDS), sharing (SEES, DEFINITIONS) and configuration (CONSTANTS, PROPERTIES, VALUES) clauses were introduced in the language. With the support of J.-R. Abrial, Alstom decided to develop its own set of tools for the new language. After two years of development, in 1993, the first version of what was called the B-Toolset was delivered internally, including a type checker, a proof obligation generator and a theorem prover able to manage software of industrial size and complexity. By that time, J.-R. Abrial proposed to Alstom that the Digilog company (then Steria, today CLEARSY) should industrialise these tools and make them available to RATP and to Matra Transport, the supplier that won the Paris Métro Line 14 contract (cf. Sect. 2.2). The work by Digilog on B started in 1995<sup>6</sup>, the contract related to L14 between RATP, SNCF and INRETS was signed in 1996. Alstom accepted and Digilog developed ATELIER B based on Alstom's B-Toolset.

The development of the language and of supporting tools was a very important aspect of the industrialisation of the B-Method. No less important was the definition of an effective and efficient development process and methodology for the new technology, while training engineers to use it.

## 2.1 Early Adoption

The introduction of the B-Method in an existing conventional software development environment necessarily induced the modification of the development pro-

---

<sup>6</sup> Which is the justification for the title of this article.

cess known and accepted by the development, verification and validation teams, the clients and the safety assessors. Doubts were numerous considering that the new process should comply with applicable railway standards, that it should be close to the existing process in order to reuse its infrastructure as much as possible, and consequently, that the activities related with the B-Method must be included within the phases of this process. Some questions were:

- Where should the definition of B abstract machines be included? In the software specification phase or in the software design phase?
- How should B components and formal proofs be documented?
- How should B components and formal proofs be verified, when and by whom?
- How should verification efforts be documented?
- How should module testing, integration and validation testing phases be modified in order to take advantage of the formal proof of B components?
- How should the development of the part of the software that does *not* need to be formally developed interact with the development of the part of the software that needs to be formally developed?

The companies that introduced the B-Method in their software development process reacted according to their own practices and experience. When it was decided to introduce it for the development of safety critical software of railway systems, the B-Method was neither taught nor used anywhere. Its first users were trained by J.-R. Abrial himself, who followed also the first developments. The methodology for using the B-Method and good practices were defined during these first developments. They address the following questions:

- How to create the architecture of a large B model?
- How to write the operations of abstract machines?
- How to refine abstract data with concrete data?
- How to refine operations?
- How to write loops?

User guides were written and some rules were automated with tools. Once the tools and good practices were developed and defined, training courses in the B-Method were given to all staff in the organisation dealing with software engineering: software development engineers, verifiers, validators, safety assurance engineers and their managers. The sustainability of the B-Method in industry has been made possible by the creation of an eco-system including RATP, the operator that requested the application of the method, CLEARSY, the company that maintains the tools supporting B in the long term, the engineering schools and universities that train engineers, conduct research and provide tools, and finally, the companies that provide B expertise and technical assistance. The existence of international conferences on formal methods and, particularly, on the B-Method and the participation of the industrial companies in these events contributes to the dissemination and sustainability of the method.

## 2.2 Driverless Metro Software: Météor and its successors

*Paris Line 14* The most well-known success story of B is the Parisian Métro Line 14. The main goal was to reduce the time interval between trains, yet ensuring the correctness and safety of the system. For this, the train control was automated and the trains are able to travel without a human driver. All safety critical components, i.e., the train control and the controllers for the automatic doors dividing the passengers from the tracks, have been formally developed using B. Since October 1998, the metro works flawlessly and not a single issue was caused by software. The same holds for the shuttle train at the Roissy Airport that drives since 2007.

The B models for the Line 14 and the Roissy Shuttle have 115 000 and 183 000 lines of code, 27 800 and 43 610 proofs and a manual proof percentage of 8.1 % and 3.3 %, respectively. More information concerning both metros and their full development statistics can be found in [5] and Sect. 4.2 of [62].

In neither case, unit testing was performed. Instead, formal development and proof gave enough confidence in the correctness of the generated Ada code which was used without change. The only tests performed were tests concerning the integration with non-critical software parts and global validation tests.

Early projects, such as the Line 14, pushed tool development. One example is semi-automatic refinement [22], which by now is included as BART in ATE-LIER B. Going through (data) refinement steps manually is tedious. Often, this work can be automated though: such a tool can drastically improve development speed, in particular if code generation from B0 is required.

*Canarsie CBTC* Siemens has evolved the product for Line 14 and installed it on many metro lines world-wide, notably on the Canarsie Line [31] in New York. 53 trains operate without interruption on 17 km of track consisting of 24 stations. In contrast to the Parisian Line 14, two different types of trains are mixed on the track: more modern trains are equipped with CBTC (computer- or communications-based train control) systems, whereas older trains are not. This results in a system that is far more complex than its counterpart in Paris.

Again, the software components of the system are split into parts that are safety-critical, and those that are non-critical for safety. All safety-critical parts have been developed in B; the only exceptions are components that cannot be expressed in a B model, including low-level communications, sensor, motors, breaks and file input/output.

The Canarsie Line was one of the first industrial applications that included the use of automatic refinement tools. Even though more proof obligations had to be discharged, these tools proved to speed up the process considerably. One of the key sentences concerning the usage of formal methods can be found in the description of the project:

*“Beyond the technological challenge of using such a complex formal method in an industrial context, it is now clear for us that building software using B is not more expensive than using conventional methods. Better, due to our experience in using this method, we can assert that using B is cheaper when considering*

**Table 1.** Overview of Alstom Projects

Product	Size (kloc)	First commissioning
Train speed controller for Calcutta Metro	Small (< 10)	1992
KVB, train protection system for French mainlines (no high-speed trains) trains.	Medium (10..50)	1995
SACEM extension for Paris RER Line A.	Small (< 10)	1996
Train speed controller for Cairo Metro.	Small (< 10)	1997
Speed controller for Lyon Metro.	Small (< 10)	1998
Lineside Electronic Unit (LEU) for mainlines in Australia, China, France, Greece, Italy, Spain, The Netherlands.	Small (< 10)	2000
URBALIS 200, train protection system for metro lines in Chile, China, Egypt, India, South Korea, Spain.	Medium (10..50)	2003
URBALIS 400, CBTC system for metro lines in Australia, Brazil, Chile, China, Dubai, France, Italy, Mexico, Panama, Qatar, Saudi Arabia, Singapore, Spain, The Netherlands, Vietnam.	Large (50..250)	2008

*the whole development process (from specification to validation and sometimes certification).*" [31]

*Safety-Critical Train Software at Alstom* Alstom has been a long time proponent of using the B-Method for safety critical software. Most Alstom trains now include some software which was produced from a B specification. Table 1 gives an overview of the important railway products where software was developed with the B-Method. The URBALIS 400 product, with its over 100 installations worldwide, represents currently the most widespread use of the B-Method for software in the world.

### 2.3 Code Generation for Hardware

There were a few research projects on using B for hardware, e.g., at the Atomic Weapons Establishment (AWE) [32] or within the PUSSEE research project [60], with applications for SmartCards (see Sect. 5 below). Only recently has B been used to develop hardware for railway applications, which we describe below.

*Platform Screen Doors Controllers* CLEARSY has developed several safety systems controlling the opening and closing of the Platform Screen Doors (PSD) installed in Metro stations in order to ensure passengers protection. These systems are independent of the train signalling and automatic operating systems; they can be installed in a Metro which is already in service. Due to the expansion of the urban population in most big cities in the world, PSD are a first step

towards full automation of a non-automatic, already existing metro line. The PSD controllers developed by CLEARSY are specified and programmed with B. First controllers used a dedicated translator from B to Ladder Logic [45], more recent ones use the CLEARSY Safety Platform. Paris lines 1 and 13, São Paulo lines 2 and 3, and Stockholm Citybanan metros have been equipped with such PSD controllers, certified SIL3 or SIL4.

*The CLEARSY Safety Platform* The CLEARSY Safety Platform [40] is both a hardware and software platform, aimed at easing the development and the deployment of safety critical applications, up to SIL4. It relies on the integration of the B-Method for programming (including mathematical proof), redundant code generation (to guard against hardware bugs or hardware failure) and compilation, and a hardware platform that ensures a safe execution of the software. Safety principles are built-in in the hardware and the safety library. The associated IDE is based on ATELIER B and the B language supported [43] has been specialised to address the specific hardware, to better ensure safety, and to minimise the proof effort. As of today, the CLEARSY Safety Platform has been certified 3 times with different certification bodies, for international railways applications.

### 3 B for System Modelling

*From Software to Systems: Event-B for System Modelling* The success of the Parisian Métro Line 14 showed that, given a set of software requirements, one could develop formally a program that fulfils it and prove it correct. But the software requirements used as input make some assumption on the environment in which the software is to be operated: logical interfaces to other pieces of software as well as electronic and physical devices such as motors.

Therefore, if the software requirements are wrong or do not fit the operational environment, the resulting system as a whole would malfunction. It was thus felt necessary to move the application of formal methods to an earlier phase in the system development process, namely in the system design phase. System design is performed by very capable engineers, but addresses very complex systems with a lot of moving parts and is difficult to reason about informally.

It was thus felt necessary to extend the B-Method to system design activities [9,3] and another notation was gradually derived from the B language, finally crystallising into Event-B, aiming for proven system studies where computation is distributed. In four EU projects concerning the development and industrialisation of Event-B, numerous industrial partners were involved, including Siemens, Bosch, SAP, Space Systems Finland, Alstom, CLEARSY, Gemplus, Leonardo and Critical Software Technologies.

*New York Flushing Line* CLEARSY used Event-B, supported by the ATELIER B tool, on two major industrial rail projects for New York City Transit (NYCT) to support safety assurance [57]. In the first project for the New York Flushing line,

formal models of a CBTC were developed and key safety properties were specified and proved at the system level. The main safety properties addressed were avoidance of train collisions, avoidance of trains traversing unlocked switches (causing derailment) and avoidance of over-speeding. The second project for NYCT involved an implementation of interlocking different from the first. Because the system level models were abstracted from details of the implementation, it was possible to reuse models from the first project in the second project, considerably reducing safety analysis effort in the second project. A key benefit of the system level formal analysis of [57] was the way precise properties required of the various sub-systems were identified in the design and the assumptions made in one sub-system about other sub-systems. Since different sub-systems were provided by different companies, this ensured that these assumptions were clearly communicated to relevant stakeholders at early stages of the development, avoiding problems later during the systems' integration phase.

*Octys* In [28], CLEARSY outline how they used Event-B supported by ATE-  
LIER B to perform safety analysis of an existing CBTC system called Octys for R ATP. Some key insights into the benefits of formalisation are described. For each safety property to be verified, the approach was to describe the property informally and an informal argument was developed to explain why the property held. This helped to frame the subsequent formal modelling and reasoning. It was found that it was very difficult to achieve a high level of rigour through the informal reasoning and that the formal reasoning filled in gaps in the reasoning, providing more complete arguments for safety. The formal reasoning also allowed the isolation of a minimal set of assumptions required to prove the desired property. This allowed for identification of gaps in the assumptions, whereby the informal safety requirements were improved.

*URBALIS 400 Zone Controller* In 2018 [29] the software for the Zone Controller of the Alstom URBALIS 400 CBTC developed using classical B (see Sect. 2.2 above) underwent a rigorous systems analysis. While the classical B method ensured that the implementation is correct wrt. the software specification, it does not guarantee that the algorithms themselves are correct wrt. system level requirements. The analysis was formalised with an Event-B model which links environment variables (the real position of the trains) with software variables (protection envelopes). ATE-  
LIER B and PROB were used to analyse the system and extract key properties that ensure the correct and safe functioning. These properties are of crucial importance when tuning or extending the algorithms of the zone controller.

*RailGround* As part of the EU H2020 Enable-S3 project, Thales Austria GmbH and the University of Southampton applied Event-B and UML-B to the *Rail-Ground* interlocking system [25]. The project used UML-B, which allows editing Event-B models using a UML-like graphical representation.

As well as demonstrating the feasibility of modelling a complex interlocking system in UML-B, the project also demonstrated benefits of using the UML



diagrammatic notation. The diagrammatic models were found to be easier to communicate to domain experts than the textual models.

*ETCS Hybrid Level 3* HL3 is a novel train control concept that aims at increasing the throughput of trains without additional rails. Thales Deutschland GmbH and Universität Düsseldorf used the B-Method to develop a reference model for a new approach to railway interlocking, Hybrid ERTMS/ETCS Level 3 (HL3), as part of a field demonstration of the feasibility of the HL3 principles [35]. The focus of the project was on the use of the model-checking and execution capabilities of PROB both for validation of the model *and* for use of the model as a reference implementation of the HL3 principles during the field demonstration. A graphical visualisation of the railway environment made it easy for the domain experts to provide feedback on the formal model, leading to improvements of the specification. A lot of the complexity of HL3 concerns degraded modes and corner cases, and the formalisation and validation approach allowed these to be addressed comprehensively. Execution of the B model on PROB was used to conduct field tests with real trains in realtime.

*EULYNX* Founded in 2014, EULYNX is a joint European project by several railway infrastructure providers aiming at a standardisation of interfaces and signaling systems. One of EULYNX members, the infrastructure division of the German railway company Deutsche Bahn, uses model-based systems engineering for their interlocking systems. Using SysML has led to improved specifications and thus increased the quality of the interlocking system. However, SysML is merely a semi-formal language. In consequence, within the European Shift2Rail project, an approach based on UML-B has been used to introduce an Event-B representation into the development process [17,55]. This effectively enables formal verification of interlocking systems specified in SysML.

## 4 B for Data Validation

In the last decade, the B language gained a new application area: aside from proving software correct, it can be used to ensure that *assumptions* about configuration data hold (often dubbed *data validation*). Indeed, a safety critical system often contains many data parameters which are instantiated differently for each particular deployment of the system. These parameters underlie restrictions to ensure the proper functioning of the system. When a system is incorrectly configured, this can lead to disaster. It turned out that the B language was very convenient to express properties for correct configuration.

This intuition gave rise to the development of the OVADO [1] tool for RATP, which took place in parallel to the early development of the RODIN platform. Prior to adopting such a tool, RATP used to have dedicated tools developed in order to check the correctness of configuration data for systems received from their suppliers. But these dedicated tools were expensive to develop and quite inflexible. Any change in the requirements made it necessary to change the software of the tool. In contrast, with a generic tool like OVADO, one just needs

to modify the B expression of the property to reflect the change and run the OVADO tool again.

In some cases, when the software was developed from a B specification, these properties were already expressed in B and used during the formal safety proof. This was, e.g., the case for the Paris Line 1 and 14 metro systems and other installations of the same system in Barcelona or São Paulo. This was one of the first industrial uses of data validation using PROB by Siemens [49,50,33], independently<sup>7</sup> conducted in 2008-2009 within the EU Deploy project. Before 2009, Siemens was using ATELIER B with custom proof rules and tactics, dedicated to deal with larger data values [19,20]. This, however, did not scale to many larger properties or data values, meaning that manual validation was required that was cost intensive and error prone. Indeed, the use of PROB did uncover at least one issue that was missed by the manual validation.

In order to better address industrial needs, tools developed dialects of the B language and domain specific data validation languages on top of B [44]. In the context of data validation string manipulations are important; hence PROB now allows usage of B's sequence operators on strings (e.g., for concatenation). Additionally, support for reading and writing XML was added to PROB during a case study in cooperation with Thales [36].

Data validation with B has now been applied to many railway systems worldwide, some of which are:

- Line 1 Paris, the second CDGVAL line LISA at the CDG airport in Paris, São Paulo line 4, ALGER line 1, Barcelona line 9, all by Siemens using a tool called RDV built-on top of PROB [49,50,33],
- more metro lines in Paris managed by RATP using OVADO, which includes a tool developed by CLEARSY called predicateB as first chain, and PROB as secondary tool chain [1,12].
- by Alstom for their URBALIS 400 CBTC system in 2014 using a tool based on PROB called DTVT developed by CLEARSY for various lines, e.g., in Mexico, Toronto, São Paulo and Panama [44].
- Alstom and SNCF also applied data validation for ETCS-Level 1 software in 2018 using another tool developed by CLEARSY using PROB.
- Together with Systerel, Alstom conducted data validation of the Octys CBTC for RATP in 2017 using the OVADO tool.
- by Thales using a tool based on PROB called Rubin for checking engineering rules of their ETCS Radio Block Centre (some aspects of Rubin are discussed in [36]).
- Other tools based on PROB were developed by CLEARSY such as Dave for General Electric or the latest generation tool called Caval.

An important aspect of these applications is the certification of the tools according to EN50128. Indeed, this norm stipulates that a data validation tool is of class T2, namely a tool that “*supports the test or verification of the design or executable code, where errors in the tool can fail to reveal but cannot directly*

---

<sup>7</sup> Initially the PROB team was unaware of the development of OVADO.

*create errors in the executable software*” [26]. The tools mentioned above satisfy the T2 requirements, e.g., by using a rigorous specification of the tool’s purpose and a rigorous testing process (see, e.g., [14]). The Caval tool obtained a T2 certificate in November 2019. The tools DTVT and OVADO even use a double chain: a primary tool that conducts the verification and a secondary tool that re-checks the result of the first tool.

## 5 Projects Outside the Railway Domain

Only few projects outside the railway industry are known to use B. Below, we present two additional areas of application.

*Modelling Vehicles* In the early 2000s, several projects were initiated to model vehicles, e.g., to improve the failure diagnostic of the first full-electronic multiplexed Peugeots as well as to ease the integration of the sub-systems of a one-time built military vehicle.

Due to the existence of the vehicles and the complexity of the design, the modelling adopted was a flat (no refinement) Event-B specification of the functional specification sided with a dictionary model providing additional semantics and natural language translation elements. A tool, Composys, was developed to automate validation and test functional architectures. It contained a static checking tool for B machines, a component-based consistency checking tool, and a natural language technical documentation generator.

*Smart Cards* When it comes to smart cards, the use of formal methods is mandatory for certification, if a high EAL security level is required. In this case, the functional specification of the software library is proved to comply with the security policy, both formalised with Event-B. Hence, application developers are assured that whatever the API calls, the smart card security is enforced i.e., no secret is disclosed. Several certifications have been obtained at the highest levels, in France and in Germany.

B was also used for embedded software development [39] while Event-B was used for hardware development [16]. The former used the default ATELIER B C code generator while the latter was based on a dedicated translator from Event-B to synthesisable VHDL.

## 6 B-Method Tools Throughout the Years

In this section, we will discuss tools for the B-Method that were developed throughout the years. As expected, not all of them survived. While some have been replaced by successors, others were only of academic interest. Given that most of the tools, including their features and peculiarities, are documented by various research papers and journal articles we keep things brief and reference the publications below.

*B-Toolkit* One of the first tools for use with the B-Method is the B-Toolkit [46,56] by B-Core. The B-Toolkit already was reasonably complete, offering editing, type checking, animation, proof obligation generation and discharge, documentation generation, and a first code generator targeting C. B-Toolkit is no longer supported. Its source code has been released under a BSD license at <https://github.com/edwardcrichton/BToolkit>.

*Click'n Prove* Click'n Prove [8] was an experimental user interface meant to explore new ways to interact with a prover (by clicking rather than command-line) and served as a basis for the RODIN interactive prover interface. Click'n Prove was built on top of XEmacs. Internally, it was using the ATELIER B tools for proving.

*ATELIER B* As mentioned above, the success of B in the railway domain drove the implementation and improvement of B-Method tools, such as ATELIER B, initially to be used for software validation [27,2].

In order to be useful for the safety-critical applications mentioned, ATELIER B needed to be verified and validated itself. To do so, several tasks were performed under the overall responsibility of RATP [41]:

- the theorem prover was subject to external expertise,
- a dedicated tableau-based prover was built to validate most of the theorem prover's mathematical rules,
- a committee was set up to demonstrate unprocessed rules by hand,
- a small automated prover was developed to verify the correctness of the dedicated tableau-based prover.

When it was created in 2001, CLEARSY gathered ATELIER B property rights from Alstom and RATP. ATELIER B is currently used by more than 30% CBTC-based automatic metros worldwide, for embedded and track-side safety software. This IDE is under continued development with new peripheral functions, e.g.:

- an automatic refiner tool, BART [42], similar to the one used by Siemens for the Canarsie line,
- a framework to automatically prove and review added mathematical proof rules, that generates a report for the safety case,
- a generic new proof obligation generator,
- integration of the PROB model checker, SMT solvers and the Why3 platform in the interactive prover,
- an improved C code generator targeting PIC32 microcontrollers,
- a compiler from B0 models to binary files for the CLEARSY Safety Platform.

While initially only supporting classical B, current versions of ATELIER B support Event-B machines as well. These Event-B machines are described using an adapted textual representation, with additional keywords (such as `ref`) and more liberal refinement which is dealt with by the proof obligation generator. This renders ATELIER B one of the two major IDEs for Event-B, the other one being RODIN.

*Rodin* RODIN [7] has been developed during the RODIN, Deploy and Advance projects. As a complete IDE, RODIN features the Event-B modelling database / storage, a type checker for Event-B and a proof engine [59] as well as different editors. Central parts of RODIN have been formally specified and proved using Click'n Prove [7].

As the team developing RODIN knew how much implementation effort went into building ATELIER B from the ground up, they were looking into ways to build RODIN on top of an existing framework and finally settled for Eclipse, from which RODIN inherits its main UI, the handling of workspaces and many internals. Just like Eclipse, RODIN is based on plugins and could (at least in theory) be extended to formalisms other than Event-B.

RODIN itself does not include a prover for Event-B. Instead, it just maintains proof trees in sequent calculus and allows reasoners and tactics to be added by plugins. In particular, ATELIER B's provers can be added to RODIN alongside others, e.g., SMT solvers. Influenced by the interactive control of provers implemented in Click'n Prove, RODIN provides a user interface for interactive proof, in which the different reasoners can be applied, proof tactics can be performed and the proof tree can be explored graphically. By doing so, provers can be used collaboratively inside a single proof.

Several code generators are available for RODIN, as are other extensions via the plugin mechanism. Among the most prominent ones are extensions for the composition and decomposition of models [23] and the theory plugin, which permits extending the mathematical core of Event-B by custom theories [52,51].

PROB PROB [48,47] is an animator, constraint solver and model checker for the B-Method. Its development started in 2001 with a first alpha release made in October 2003. It filled a gap in the B tooling landscape at the time, supporting the interactive and automatic validation of high-level specifications. Indeed, following classical B's correct-by-construction approach it is vital that the high-level specifications correctly capture the high-level requirements and functionality.

By then, only the B-Toolkit animator provided some very limited form of validation, and required the user to provide values for parameters and existentially quantified variables, the validity of which was checked by the BToolkit prover. This approach was justified by the undecidability of the B language, but was tedious for the user and prevented automated validation. In contrast, PROB allows fully automatic animation of specifications, i.e., values for constants, parameters are computed by PROB's constraint solver rather than explicitly given by users. Unknown to the PROB team at the time (around 2000), another team pursued similar ideas leading to the CLP-S solver [21] and the BZTT tool [11] based on it. This work also gave rise to a company (Lerios), which concentrated on model-based test-case generation and later ported the technology to an imperative programming language. Unfortunately, the development of BZTT and CLP-S has stopped; the tool is no longer available.

Using a variety of explicit-state and symbolic model checking approaches, PROB can be used to systematically check a specification for invariant violations [48]. Furthermore, PROB supports LTL model checking, and distributed

explicit state model checking. Model checking aside, the constraint-solving capabilities of PROB can also be used for model finding, symbolic model checking and deadlock checking as well as test-case generation and drive several of the animation, visualisation and data validation tools that will be discussed below.

*Animation & Visualisation Tools* For the industrial applicability of formal methods, visualisation and graphical model animation allow formal method experts to communicate with domain experts and enable them to identify errors. This may go as far as having a “management view”, that is easy to understand and hides all technical details [41]. Many visualisation and animation tools have been developed for B and Event-B. Among the first ones are BRAMA [58], which uses Flash to graphically visualise models. AnimB [54], a plugin for RODIN, also provides graphical visualisations based on Flash. Several tools were developed building on top of PROB: starting from an early prototype using Flash [15], BMotionStudio has been developed for editing and displaying visualisations [37]. Later on, BMotionStudio was superseded by BMotionWeb [38], an animation engine based on common web technologies, and the simpler VisB [61] based on SVG graphics. Another web-based animator was JEB [63], which was independently developed in JavaScript.

## 7 Discussion and Conclusions

Development of the B language and tools has been driven by industrial needs, which probably explains part of its success. A recent survey in the railway domain [13] cites mature tooling as the most important reasons to use a specific formal method. Mainly ATELIER B and PROB have been developed for a long time and have proven themselves in industry projects. They both are mature tools that also are actively maintained and further features are developed. The reader may also wish to consult older surveys on industrial use of formal methods in general such as [18,62,34,53].

*Current Situation of B* Here is our assessment of the current situation concerning the use of B in industry:

- B is arguably among the formal methods of greatest industrial impact, albeit mainly in the railway sector.
- There is still little industrial use of B in production outside of railways. B seems like a DSL for the railways: topologies can be well expressed using B relations, integer arithmetic is sufficient in many applications. In railways, we have clearly defined operating environments which enable exhaustive formal modelling and inductive reasoning.
- The flagship products of Alstom’s U400 and the successors of Météor are still operating and being installed on new lines. URBALIS 400 is running on over 100 lines and has 25% of the worldwide market in CBTC systems.<sup>8</sup>

<sup>8</sup> See the site (accessed 25/5/2020): <https://www.alstom.com/our-solutions/signalling/urbalis-cbtc-range-future-signalling-systems>

- Code generation for B has now moved to hardware level but the use of classical B for software (outside of hardware) is not increasing. It has not caught on in Siemens to other products and is not being applied to new products at Alstom anymore. One reason may be the need for experienced people. Moving from formal modelling to code generation requires a lot of extra resources. New tools like automatic refinement (BART) help to some extent, but one still spends a lot of time discharging proof obligations of little practical value (and it takes time to identify the really crucial proof obligations that pose essential problems).
- RODIN has had a lot of academic impact, but real industrial use for production systems is still somewhat disappointing. Several aspects of RODIN were stimulating academic research and experimentation, but were possibly detrimental for industrial use, e.g.: the use of Eclipse with its extension mechanism, the core language without sequences and machine inclusions, models being stored in an extensible database rather than a textual format. For example, the extension mechanism enabled experimentation, but it is confusing for industrialists to know which extensions are stable and are suitable for industrial use. The absence of a textual format was detrimental for team collaboration and versioning. Also, the tight link to Eclipse makes it more difficult to use RODIN as a stand-alone headless tool, in case a company's development practice is not centred around Eclipse.  
An exception here is UML-B [30] and Coda [24], where the tight integration with Eclipse enabled graphical modelling and industrial applications (cf. Sect. 3). Also, machine inclusion and textual format are now supported by the new CamilleX plug-in.<sup>9</sup>
- B for data validation has caught on and is being used for a wide range of railway products.
- There is an increased interest and activity by a wider range of industrial players for systems modelling with B. This is one area where we foresee considerable growth in the coming years, and where B could maybe move to more widespread use outside the railway domain.

*What made B successful in the beginning* The ability to specify programs rigorously and to demonstrate the compliance of their implementation with their specification was a major concern for RATP and Alstom in the beginning. The B-Method answered that need. The following factors also played a major role to enable B's initial breakthrough:

- the availability of tools to validate and verify the B formal notation,
- the more tractable proof obligations, thanks to the B machine structuring clauses and the use of successive refinement,
- the availability of code generators for B0, providing tangible benefits in terms of testing and certification efforts.

---

<sup>9</sup> See <https://wiki.event-b.org/index.php/CamilleX>

*Common Success and Fail Factors in Industrial Uses* We can identify the following common success factors for the industrial applications of the B-Method:

- *Tooling.* Formal methods can be of value without tool support, just as a technique to aid and focus human reasoning on complex systems. However, tooling provides many additional benefits and the availability of tools played a major role in most of B's successful industrial use cases.
- *Effectiveness of the method.* Thanks to the B-Method it was actually possible to specify, develop and prove formally programs and avoid a lot of tests.
- *Regulatory constraints.* The requirement (or at least strong steer) from the guiding authorities, e.g., RATP, for the suppliers to use formal methods is definitely a success factor in several of the case studies.
- *Expertise.* Formal method education is not widespread among programmers and engineers. Hence, the availability of B experts to provide support to a project team is crucial for success.
- *Documentation.* The availability of a methodological guide explaining how to model common artefacts in B (e.g., state machines, iterators) is important. This allows the spread of good practice among project teams and avoid each individual reinventing the wheel.

The following fail factors appeared multiple times.

- *Functional requirements.* Even formally developed programs may not fit the functional user needs. In other words, using the B-Method does not prevent developing the wrong program.  
This threat can be countered by early uses of animation on the high-level formal specification along with involvement of domain experts.
- *Investment.* Human investment is high and it is difficult to retain employees skilled with the B-Method. Some managers are convinced that formal modelling and proving is no fit for their “low-grade engineers”.
- *Predictability.* Proof is not full automatic, therefore, it is difficult to predict costs and delays of software development.
- *Scalability.* As far as Event-B is concerned, missing decomposition features and collaboration support in RODIN prevented modelling some more complex systems by larger teams.
- *Disruption.* The B-Method for software has a disruptive impact on existing software development practices; it basically requires to replace the existing software development cycle with a new one.
- *Business plans.* The development of bug-free software is often not an important objective of software companies, and may even be opposite to their business plans based on consulting and paid upgrades.

*Challenges* We see the following challenges which lie ahead to a successful broader adoption of the B methodology.

- *Proof and refinement.* More automation in proof is needed, for example by combining provers and constraint solvers, thereby reducing the need for



costly interactive proof. Better feedback for failed proof attempts and assistance in finding inductive invariants would lower barriers for users. The automation of refinement should be dramatically improved in order to reduce manual work.

- *Manage complex systems and models.* In particular, one challenge is to help users understand bigger formal models, e.g., by new visualisation techniques or extraction of knowledge guided by machine learning techniques. Automated correction and suggestions, especially helping non-formal methods experts.
- *Increase expressivity.* Guided by the work in data validation, it would be beneficial, e.g., to enable more convenient use of n-ary relations and associated projections, as the standard projections functions *prj1* and *prj2* on nested pairs are very cumbersome to use. Another challenge is to be able to express possibly recursive, higher-order functions so that proof, constraint solving, animation and execution are possible. Finally, a principled solution to replace the brittle DEFINITIONS of classical B should be sought.
- *Increase value of B formal modelling by making formal models executable.* Execution makes formal models accessible to domain experts and enables formal models to be used as cost-effective prototypes. Formal models can also play the role of interactive requirement or specification documents.
- *Marketing.* Efficient packaging of B tools such as the CLEARSY Safety Platform to both attract students and industry for practical applications.
- *Alternate approaches for B software development.* With the progress of model checking techniques, it will be maybe worth to develop conventionally and then verify formally, leading to less disruption of existing development practices.
- *Combine systems modelling with implementation.* Bridge the gap between Event-B systems modelling and classical B software development, providing a seamless process and along with supporting tools for combined systems modelling and implementation.

**Acknowledgements** We would like to show our gratitude to Jean-Raymond Abrial, who provided us with sources, discussions, insider information and knowledge from his personal experiences developing B and Event-B. We also thank the reviewers of FMICS for their extensive feedback and suggestions.

## References

1. R. Abo and L. Voisin. Formal implementation of data validation for railway safety-related systems with OVADO. In *Proceedings SEFM 2013*, volume 8368 of *LNCS*, pages 221–236. Springer, 2014.
2. J.-R. Abrial. The B Tool (Abstract). In *Proceedings VDM*, volume 328 of *LNCS*, pages 86–87. Springer, 1988.
3. J.-R. Abrial. Extending B without changing it. In *Proceedings B*, pages 169–190, 1996. ISBN 2-906082-25-2.

4. J.-R. Abrial. *The B-Book*. Cambridge University Press, 1996.
5. J.-R. Abrial. Formal Methods: Theory Becoming Practice. *Journal of Universal Computer Science*, 13(5):619–628, 2007.
6. J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.
7. J.-R. Abrial, M. Butler, S. Hallerstede, and L. Voisin. An Open Extensible Tool Environment for Event-B. In *Proceedings ICFEM*, volume 4260 of *LNCS*, pages 588–605. Springer, 2006.
8. J.-R. Abrial and D. Cansell. Click’n Prove: Interactive Proofs within Set Theory. In *Proceedings TPHOL*, volume 2758 of *LNCS*, pages 1–24. Springer, 2003.
9. J.-R. Abrial and L. Mussat. Introducing dynamic constraints in B. In *Proceedings B*, volume 1393 of *LNCS*, pages 83–128. Springer, 1998.
10. J.-R. Abrial, S. Schuman, and B. Meyer. Specification language. In *On the Construction of Programs: An Advanced Course*. Cambridge University Press, 1980.
11. F. Ambert, F. Bouquet, S. Chemin, S. Guenaud, B. Legeard, F. Peureux, M. Utting, and N. Vacelet. BZ-testing-tools: A tool-set for test generation from Z and B using constraint logic programming. In *Proceedings FATES*, pages 105–120, 2002. Technical Report, INRIA.
12. F. Badeau and M. Doche-Petit. Formal data validation with Event-B. *CoRR*, abs/1210.7039, 2012. Proceedings of DS-Event-B 2012, Kyoto.
13. D. Basile, M. H. ter Beek, A. Fantechi, S. Gnesi, F. Mazzanti, A. Piattino, D. Trentini, and A. Ferrari. On the industrial uptake of formal methods in the railway domain. In *Proceedings iFM*, volume 11023 of *LNCS*, pages 20–29. Springer, 2018.
14. J. Bendisposto, S. Krings, and M. Leuschel. Who watches the watchers: Validating the ProB Validation Tool. In *Proceedings F-IDE*, volume 149. EPTCS, 2014.
15. J. Bendisposto and M. Leuschel. A Generic Flash-Based Animation Engine for ProB. In *Proceedings B*, volume 4355 of *LNCS*, pages 266–269. Springer, 2007.
16. M. Benveniste. On Using B in the Design of Secure Micro-controllers: An Experience Report. *ENTCS*, 280:3–22, 2011.
17. R. Berghlehner, I. Cherif, and A. Rasheeq. An Approach to Improve SysML Railway Specification using UML-B and EVENT-B. Poster presented at RSSRail 2019.
18. J. Bicarregui, J. S. Fitzgerald, P. G. Larsen, and J. C. P. Woodcock. Industrial Practice in Formal Methods: A Review. In *Proceedings FM*, volume 5850 of *LNCS*, pages 810–813, 2009.
19. O. Boite. Méthode B et Validation des Invariants Ferroviaires. Master’s thesis, Université Denis Diderot, 2000. Mémoire de DEA de logique et fondements de l’informatique.
20. O. Boite. Automatiser les preuves d’un sous-langage de la méthode B. *Technique et Science Informatiques*, 21(8):1099–1120, 2002.
21. F. Bouquet, B. Legeard, and F. Peureux. CLPS-B - a constraint solver for B. In *Proceedings TACAS*, volume 2280 of *LNCS*, pages 188–204. Springer, 2002.
22. L. Burdy and J.-M. Meynadier. Automatic refinement. *Proceedings BUGM at FM’99*, 1999. [https://www.clearsy.com/wp-content/uploads/sites/7/dl/lilian\\_burdy/ug020003.pdf](https://www.clearsy.com/wp-content/uploads/sites/7/dl/lilian_burdy/ug020003.pdf).
23. M. Butler. Decomposition Structures for Event-B. In *Proceedings IFM*, volume 5423 of *LNCS*, pages 20–38. Springer, 2009.
24. M. J. Butler, J. Colley, A. Edmunds, C. F. Snook, N. Evans, N. Grant, and H. Marshall. Modelling and refinement in CODA. In *Proceedings Refine*, volume 115 of *EPTCS*, pages 36–51, 2013.

25. M. J. Butler, D. Dghaym, T. Fischer, T. S. Hoang, K. Reichl, C. F. Snook, and P. Tummeltshammer. Formal Modelling Techniques for Efficient Development of Railway Control Products. In *Proceedings RSSRail*, volume 10598 of *LNCS*, pages 71–86. Springer, 2017.
26. CENELEC. Railway Applications: Communications, Signalling and Processing Systems. Software for Railway Control and Protection Systems. *EN50128: 2001*, 2001.
27. ClearSy. *Atelier B, User and Reference Manuals*. Aix-en-Provence, France, 2009. Available at <http://www.atelierb.eu/>.
28. M. Comptier, D. Déharbe, J. M. Perez, L. Mussat, P. Thibaut, and D. Sabatier. Safety Analysis of a CBTC System: A Rigorous Approach with Event-B. In *Proceedings RSSRail*, volume 10598 of *LNCS*, pages 148–159. Springer, 2017.
29. M. Comptier, M. Leuschel, L. Mejia, J. M. Perez, and M. Mutz. Property-Based Modelling and Validation of a CBTC Zone Controller in Event-B. In *Proceedings RSSRail*, volume 11495 of *LNCS*, pages 202–212, 2019.
30. D. Dghaym, M. Dalvandi, M. Poppleton, and C. F. Snook. Formalising the Hybrid ERTMS Level 3 specification in iUML-B and Event-B. *Int. J. Softw. Tools Technol. Transf.*, 22(3):297–313, 2020.
31. D. Essamé and D. Dollé. B in Large-Scale Projects: The Canarsie Line CBTC Experience. In *Proceedings B*, volume 4355 of *LNCS*, pages 252–254. Springer, 2007.
32. N. Evans and W. Ifill. Hardware Verification and Beyond: Using B at AWE. In *Proceedings B*, volume 4355 of *LNCS*, pages 260–261. Springer, 2007.
33. J. Falampin, H. Le-Dang, M. Leuschel, M. Mokrani, and D. Plagge. Improving Railway Data Validation with ProB. In *Industrial Deployment of System Engineering Methods*, pages 27–43. Springer, 2013.
34. J. S. Fitzgerald, J. Bicarregui, P. G. Larsen, and J. Woodcock. Industrial Deployment of Formal Methods: Trends and Challenges. In *Industrial Deployment of System Engineering Methods*, pages 123–143. Springer, 2013.
35. D. Hansen, M. Leuschel, D. Schneider, S. Krings, P. Körner, T. Naulin, N. Nayeri, and F. Skowron. Using a Formal B Model at Runtime in a Demonstration of the ETCS Hybrid Level 3 Concept with Real Trains. In *Proceedings ABZ*, volume 10817 of *LNCS*, pages 292–306. Springer, 2018.
36. D. Hansen, D. Schneider, and M. Leuschel. Using B and ProB for Data Validation Projects. In *Proceedings ABZ*, volume 9675 of *LNCS*, pages 167–182. Springer, 2016.
37. L. Ladenberger, J. Bendisposto, and M. Leuschel. Visualising Event-B Models with B-Motion Studio. In *Proceedings FMICS*, volume 5825 of *LNCS*, pages 202–204. Springer, 2009.
38. L. Ladenberger and M. Leuschel. BMotionWeb: A Tool for Rapid Creation of Formal Prototypes. In *Proceedings SEFM*, volume 9763 of *LNCS*, pages 403–417. Springer, 2016.
39. J.-L. Lanet. The use of B for Smart Card. In *Proceedings FDL vol.2*, 2002.
40. T. Lecomte. The CLEARSY Safety Platform. <https://www.clearsy.com/en/our-tools/clearsy-safety-platform/>. Accessed: 2020-01-21.
41. T. Lecomte. Applying a Formal Method in Industry: A 15-Year Trajectory. In *Proceedings FMICS*, volume 5825 of *LNCS*, pages 26–34. Springer, 2009.
42. T. Lecomte. Return of Experience on Automating Refinement in B. In *Proceedings SETS*, 2014.
43. T. Lecomte. *Developing Safety Critical Applications*. CLEARSY Systems Engineering, 2019. Accessed: 2020-01-21.

44. T. Lecomte, L. Burdy, and M. Leuschel. Formally Checking Large Data Sets in the Railways. *CoRR*, abs/1210.6815, 2012. Proceedings of DS-Event-B.
45. T. Lecomte, T. Servat, G. Pouzancré, et al. Formal methods in safety-critical railway systems. In *Proceedings SBMF*, pages 29–31, 2007.
46. M. Lee and I. H. Sørensen. B-tool. In *Proceedings VDM*, volume 551 of *LNCS*, pages 695–696. Springer, 1991.
47. M. Leuschel, J. Bendisposto, I. Dobrikov, S. Krings, and D. Plagge. From Animation to Data Validation: The ProB Constraint Solver 10 Years On. In *Formal Methods Applied to Complex Systems: Implementation of the B Method*, chapter 14, pages 427–446. Wiley ISTE, 2014.
48. M. Leuschel and M. J. Butler. ProB: an automated analysis toolset for the B method. *STTT*, 10(2):185–203, 2008.
49. M. Leuschel, J. Falampin, F. Fritz, and D. Plagge. Automated Property Verification for Large Scale B Models. In *Proceedings FM*, volume 5850 of *LNCS*, pages 708–723. Springer, 2009.
50. M. Leuschel, J. Falampin, F. Fritz, and D. Plagge. Automated property verification for large scale B models with ProB. *Formal Asp. Comput.*, 23(6):683–709, 2011.
51. I. Maamria and M. Butler. Rewriting and Well-Definedness within a Proof System. In *Proceedings PAR*, volume 43. EPTCS, 2010.
52. I. Maamria, M. Butler, A. Edmunds, and A. Rezazadeh. On an Extensible Rule-Based Prover for Event-B. In *Proceedings ABZ*, volume 5977 of *LNCS*, pages 407–407. Springer, 2010.
53. A. Mashkoo, F. Kossak, and A. Egyed. Evaluating the suitability of state-based formal methods for industrial deployment. *Softw. Pract. Exp.*, 48(12):2350–2379, 2018.
54. C. Metayer. AnimB website. <http://www.animb.org/>.
55. A. Rasheeq. An Approach To Improve SysML Railway Specification Using UML-B And Event-B. Master’s thesis, Frankfurt University of Applied Sciences, 2019.
56. K. Robinson. The B method and the B toolkit. In *Proceedings AMAST*, volume 1349 of *LNCS*, pages 576–580. Springer, 1997.
57. D. Sabatier. Using Formal Proof and B Method at System Level for Industrial Projects. In *Proceedings RSSRail*, volume 9707 of *LNCS*, pages 20–31. Springer, 2016.
58. T. Servat. BRAMA: A New Graphic Animation Tool for B Models. In *Proceedings B*, volume 4355 of *LNCS*, pages 274–276. Springer, 2007.
59. L. Voisin and J.-R. Abrial. The Rodin Platform Has Turned Ten. In *Proceedings ABZ*, volume 8477 of *LNCS*, pages 1–8. Springer, 2014.
60. N. S. Voros, C. F. Snook, S. Hallerstede, and K. Masselos. Embedded System Design Using Formal Model Refinement: An Approach Based on the Combined Use of UML and the B Language. *Design Autom. for Emb. Sys.*, 9(2):67–99, 2004.
61. M. Werth and M. Leuschel. VisB: A lightweight tool to visualize formal models with SVG graphics. In *Proceedings ABZ*, LNCS, 2020. to appear.
62. J. Woodcock, P. G. Larsen, J. Bicarregui, and J. S. Fitzgerald. Formal methods: Practice and experience. *ACM Comput. Surv.*, 41(4):19:1–19:36, 2009.
63. F. Yang, J. Jacquot, and J. Souquères. JeB: Safe simulation of Event-B models in javascript. In *Proceedings APSEC, Volume 1*, pages 571–576. IEEE, 2013.