BRILLANT: an open source platform for B

Samuel Colin[1], Dorian Petit[2], Georges Mariano[3], and Vincent Poirriez[2]

[1] scolin@hivernal.org
[2] Univ Lille Nord de France, UVHC
LAMIH, CNRS UMR 8530
F-59313 Valenciennes Cedex 9, France
dorian.petit@univ-valenciennes.fr, vincent.poirriez@univ-valenciennes.fr
[3] INRETS-ESTAS
National Institute for Transport and Safety Research
20 rue Elisée Reclus - B.P. 317
F-59666 Villeneuve d'Ascq, France
georges.mariano@inrets.fr

Abstract. This article presents an open-source platform, the ***BRILLANT*** project, with a focus on some of its most prominent components.

Keywords: B method, tool support, XML

1 Introduction

The software industry adopted B because of the availability of software tools supporting all phases of the B development process (semantics verification, refinement, proving, automatic code generation). Unlike most software tools, B support tools resulted from prototypes developed by industrial partners rather than by the academic community. In fact, from 1993 to 1999, the "Atelier B" development project was funded by the "Convention B", which was a collaborative effort of the RATP (Parisian Autonomous Transportation Company), SNCF (the French National Railway Society), INRETS (the National Institute for Transport and Safety Research) and Matra Transport (now Siemens Transportation Systems), among others.

We think it is important to have solutions as open as possible so as to allow the research community to experiment upon the B method, hence we have proposed the *BRILLANT* [1] framework, showing the feasibility of a safe software development system that ranges from semi-formal specification (UML) to contract-equipped code generation (i.e., equipped with assertions from the abstract model). This framework consists in a central core that provides basic tools for abstract manipulations of the B language. Various components can be plugged into this central core.

Figure 1 shows the BRILLANT platform's current organization. This platform is the result of several years (the first developments goes back to 1997) of academic research and development by a few Masters degree and PhD students. The original impetus for the research was the lack of open tools allowing to manipulate and experiment with the B method. Thus, our first project was to implement a parser for the B language. We immediately realized that academia, or at least a part of it, was just waiting for such an open tool since soon after its first release, two studies [2,3] used and built upon our tool.

Later on, a proof obligations generator was developed as the B method is, at its center, proof-related. At this point of time, we had exclusively used Objective Caml [4] as a meta-language in our platform development. It made sense to us that a cooperative development platform could not remain language-dependent, which led us to define and use an XML description for B in order to facilitate connections with the tools developed by others.

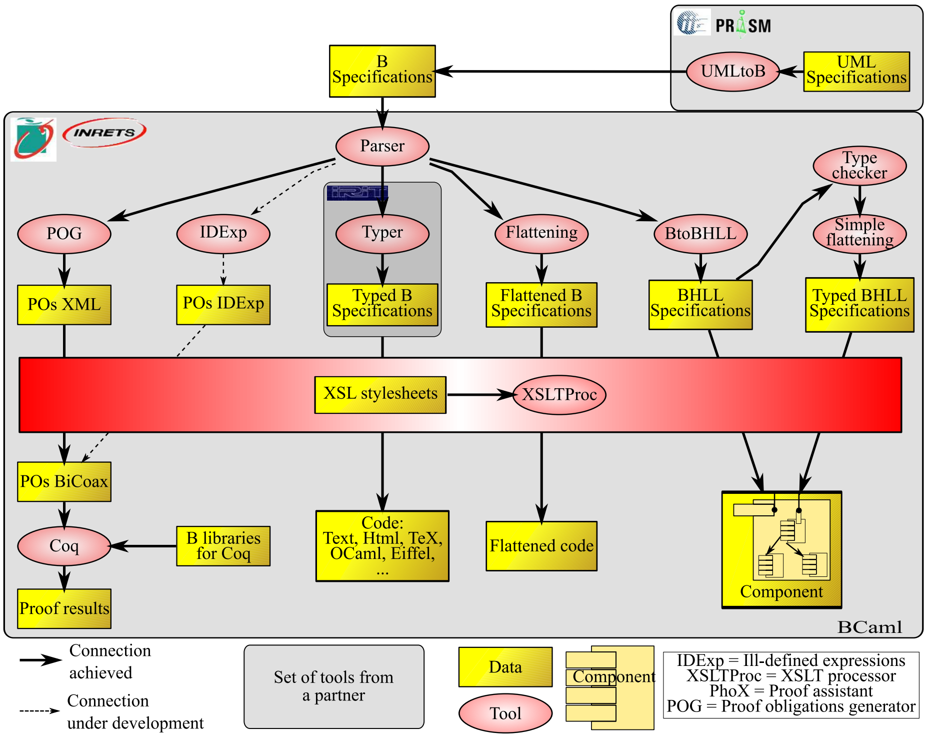

Fig. 1. The overall organization of *BRILLANT*

2 Some specificities of BRILLANT

Specifying with B ultimately entails the generation of proof obligations and their demonstration. In this section we will focus on two parts of the BRILLANT platform that deal with these two aspects of the B method.

2.1 The Proof Obligations Generator (POG)

Extending the language of B or adding new concepts often ends up in the extension of the semantics of B: in that case, new kinds of proof obligations may appear. As a result, the validation of these works requires the availability of a POG for experimenting.

For the development of the POG, we chose to follow the approach defined in the B-Book [5]: reducing B substitutions to their smallest syntactic and semantic set (i.e., generalized substitutions or GSL). Their definition in ***BRILLANT*** corresponds to the use of an abstract data type that is then used for generating proof obligations according to the rules of [5, appendix E].

The corresponding ***BRILLANT*** code was written with readability in mind, making it easy to match the code with the rule from which it is derived. Ideally, this makes the POG of ***BRILLANT*** a developer-friendly target for experimenting with new proof obligations rules, assuming some familiarity with the OCaml language.

Once the proof obligations in the XML format are available, they can be exported to other tools by using XSL stylesheets. For instance, the proof obligations can be converted into LaTeXfiles ; into text files, which are easily read by humans; into HTML files, which improve the readability of the formulas; or into a format suitable for a prover, in order to verify the proof obligations.

2.2 The prover

BiCoax is a set of COQ libraries defining the mathematical constructs presented in [5] and containing at least all properties and theorems described therein. Historically, it is the descendant of another tool based on a similar prover (PhoX): the theory files were translated to COQ and reorganized so as to follow the outline of [5].

At the time of writing, BiCoax implements all constructs and all properties of the first two chapters of [5] and half of the third chapter, up to and including the definitions of natural numbers on set-theoretic terms.

Our implementation let us find a few mistakes in the BBook. A good part of them are typographical errors. Some of them came from misusing or forgetting hypotheses when proving a property, leading to false properties, whose incidence was fortunately minor on the rest of the theory. These errors are documented in [6] and in an ERRATA file distributed along with BiCoax.

In the end, although BiCoax is not complete yet as the last half of the third chapter of the BBook is not implemented, it is already integrated in the toolchain for designing and proving B projects. As such it illustrates the relevance of the choices made in the upper part of the toolchain such as the use of an XML format as an exchange format between loosely-connected tools. Despite its incompleteness, we can consider that its initial goal of providing a proof tool for B is reached. Thanks to the loose dependency towards the rest of the ***BRILLANT*** platform, BiCoax can also be used on its own for experimenting on B extensions.

3 Development and distribution of BRILLANT

The first choices of development for ***BRILLANT*** consisted in the use of the Objective Caml language in a Unix-like development environment. The reasons behind these choices were practical: the Master's students were familiar with developping with OCaml for having studied it in college and were also familiar with the development environment for the same reason. Development included the use of a versioning system, which was made necessary because of the number of people involved. Another reason behind the development environment is the fact that commercial B tools ran on the same platforms.

These choices brought their share of advantages and inconvenients. The use of OCaml alleviated the weight of abstract-heavy handling of the B language, making the implementation of the various parts of the B toolchain relatively easy for the students. The development in a Unix-like environment made porting the tool to other platforms easier as well. This fact was also facilitated by the fact that OCaml was/is available on the targetted platforms: Linux, Solaris, MacOSX and Windows+Cygwin for the OS, Intel, PowerPC and Sparc for the architectures. Finally, the use of a versioning system made the development of experimental branches and their reintegration into the main code easier.

The inconvenients are on par with the advantages. The use of OCaml makes it more difficult for the newcomer to compile the tools without prior knowledge of OCaml development. We tried to simplify this part by using a standard way of compiling on Unix-like platforms ("make; make install"). The choice of such a neutral development environment also makes the installation tricky, unusual or not experimented on some platforms.

The interaction between the tools suffers from a similar impediment. Because of the variety of existing shells in the aforementioned platforms, instead of using shell scripts for making the tools interact with each other, we use a Makefile script. This makes the dependency-based generation of the various files of a project easier, as producing one file from another is simply a matter of Makefile rule activation. This unfortunately has the unfortunate side-effect of making any error in the process difficult to understand and debug for newcomers.

In the end, we feel that the benefits overcome the difficulties. Although ***BRILLANT*** has not been tweaked for each of the platform, it still has this potential because we tried to make the most neutral and safe choices for development in the first place.

4 Discussion, conclusion and perspectives

The *BRILLANT* platform design has two principal advantages: it uses open and standardized formats, and the source codes for its tools are openly available. In addition, it can be used to test and/or validate B-related experiments, and in fact, we were the first users of many of the prototypes now available for the platform (e.g., bparser, bgop, btyper, BiCoax). We have been working to finetune the platform to help it meet the needs of other theoretical research projects, including but not limited to extending the B language, improving the current tools, providing couplings with other provers (e.g., Coq, Harvey), and offering other validation formalisms (e.g., model-checking).

Several other projects, these more related to the fundamental research currently under way, also offer interesting perspectives for the future, such as UML/OCL/B coupling [7], temporal extensions for B [8], and safe software component generation [9]. Much work remains to be done, and the platform developers will be happy to provide their assistance to those who would like to try to use the tools in the context of their own research. All the necessary resources for building *BRILLANT* are available on the web site dedicated to collaborative free software development [1].

References

1. : (BRILLANT) http://gna.org/projects/brillant.
2. Bodeveix, J.P., Filali, M.: Type synthesis in B and the translation of B to PVS. In: ZB'2002 – Formal Specification and Development in Z and B. Volume 2272 of Lecture Notes in Computer Science (Springer-Verlag)., Grenoble, France, LSR-IMAG (2002) 350–369
3. Laleau, R., Mammar, A.: A generic process to refine a B specification into a relational database implementation. In: ZB'2000 – International Conference of B and Z Users. Volume 1878 of Lecture Notes in Computer Science (Springer-Verlag)., Helsington, York, UK YO10 5DD (2000) 22–41
4. Leroy, X., Doligez, D., Garrigue, J.and R<E9>my, D., Vouillon, J.: The objective caml system. Technical report, INRIA (2005) Software and documentation available on the Web http://caml.inria.fr/.
5. Abrial, J.R.: The B Book - Assigning Programs to Meanings. Cambridge University Press (1996)
6. Colin, S., Mariano, G.: BiCoax, a proof tool traceable to the BBook. In: From Research to Teaching Formal Methods - The B Method (TFM B'2009). (2009)
7. Marcano, R., Levy, N.: Using B formal specifications for analysis and verification of UML/OCL models. In: Workshop on consistency problems in UML-based software development. 5th International Conference on the Unified Modeling Language, Dresden, Germany (2002)
8. Colin, S., Mariano, G., Poirriez, V.: Duration calculus: A real-time semantic for B. In: First International Colloquium on Theoretical Aspects of Computing, UNU-IIST (2004) Guiyang, China.
9. Petit, D., Poirriez, V., Mariano, G.: The B method and the component-based approach. Journal of Design & Process Science: Transactions of the SDPS **8**(1) (2004) 65–76 ISSN 1092-0617.