BSmart: A Tool for the Development of Java Card Applications with the B Method

Conference Paper · September 2008

DOI: 10.1007/978-3-540-87603-8_39 · Source: DBLP

3 authors, including:

David Déharbe
ClearSy System Engineering
114 PUBLICATIONS **940** CITATIONS

SEE PROFILE

Anamaria Martins Moreira
Federal University of Rio de Janeiro
62 PUBLICATIONS **222** CITATIONS

SEE PROFILE

BSmart: A Tool for the Development of Java Card Applications with the B Method

D. Déharbe[1], B. E. G. Gomes[1], and A. M. Moreira[1]

Federal University of Rio Grande do Norte; Natal, RN; Brazil
{david, bruno, anamaria}@consiste.dimap.ufrn.br

Abstract. BSmart is a tool to support a customized version of the B method for engineering Java Card software components for Smart Card applications, which require high-degrees of reliability and security.

1 Introduction

A smart card [1] is a portable computer device able to store data and execute commands in a highly secure way. Java Card [2] is a specialization of Java, providing vendor inter-operability for smart cards, and has now reached a *de facto* standard status in this industry. The strategic importance of this market and the requirement for a high reliability motivate the use of rigorous software development processes for smart card aware applications based on the Java Card technology. The B method [3] is a good candidate for such process, since it is a formal method with a successful record to address industrial-level software development.

In [4, 5], we proposed two versions of a Java Card software development method (called BSmart) based on the B method. Section 2 summarizes the main steps of the BSmart method.

In this paper, we present the current version of a tool (also called BSmart) to support the BSmart method. The BSmart tool must provide the automatable steps required by the method and some guidelines and *library machines* that can be useful during the development process. Further details are provided in Section 3. Related work and final considerations are drawn in Section 4.

2 The BSmart Method

Smart card applications [1] have a client-server architecture, with the services being provided by the card, and the client (also called host application), executing in a terminal to which the card is temporarily connected. Communication is carried out through the APDU (Application Protocol Data Unit) protocol, defined in smart card standards [1]. The main feature of the BSmart method is to abstract, as much as possible, such platform particularities from the developers of smart card aware applications.

The card services specifier only needs to apply some refinement steps to his abstract (implementation platform independent) B specification (this specification is called `API.mch` in the following). These refinements have the goal of adapting the specification to Java Card standards and introducing platform specific aspects gradually. Also, as usual in the B method, other refinement steps may be needed to make the substitutions

that define the operations directly translatable into Java Card code. Finally, the corresponding services provided by the card (Java Card code) will automatically be generated by the tool.

On the other side of the application, the developer of the host-side of the application will see a Java API, also generated by the tool, which corresponds to the interface of the original abstract specification from where the development started (`API.mch`). The client application code can then be developed in a completely platform independent way.

3 The BSmart Tool

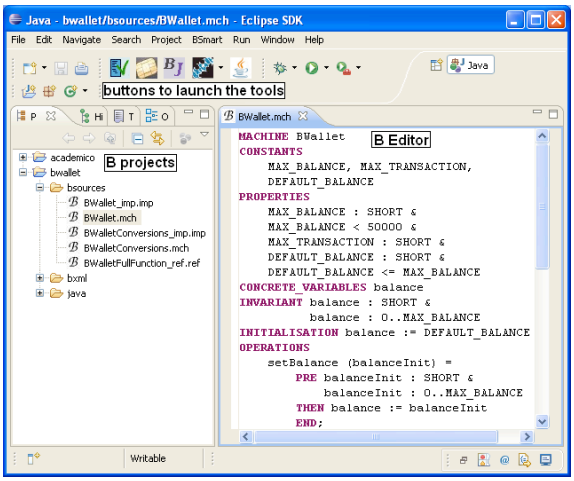

Fig. 1. A snapshot of the BSmart interface.

The BSmart tool is an environment connecting several software components, each responsible for implementing a different step of the BSmart method. BSmart is being developed as an Eclipse IDE plugin. The choice for the use of Eclipse was made mainly because it allows a faster development of the user interface and the easier and faster distribution of the tool. Also, with this choice, we align BSmart with the RODIN platform, also developed in Eclipse.

The different components making up the BSmart environment are:

B parser and type checker jBTools [6] is the front-end of the environment. Responsible for parsing and type-checking, it generates an XML intermediary format that is later retrieved for further processing.

Proof obligations (PO) generator POs are generated with Batcave [7], that generates proof obligations in textual and Harvey [1] formats.

BSmart modules Generator This component generates a *full function* refinement, i.e., a refinement where all non-typing pre-conditions of each operation are checked for explicitly, and exceptions are raised whenever they are not satisfied. It provides a wizard for user-defined exception names or can generate them automatically. The tool includes the creation of a specific B machine to specify exceptions and a module, called *Conversions*, which defines useful abstraction for both client and server sides.

B-Java Card conformity checker This component verifies that some Java Card related restrictions (e.g., number of defined methods) are satisfied by the B specification.

B to Java Card code translator This component translates all the B implementation modules into Java Card programming code. It is derived from the existing Java

[1] http://harvey.loria.fr/

synthesis component available in jBTools [6] by the introduction of Java Card specific rules. The tool generates the Java Card server application as well as an API for the *host* side client. This API is responsible for abstracting the details of the Java Card framework classes being used to communicate with the *card* side server. It also handles coding the method arguments into communication protocol data units and decoding the corresponding results.

4 Related Work and Conclusions

Related work [8] and tools, such as AtelierB and the BToolkit, concerning the generation of imperative code from B specifications, have been around for a while. The generation of object oriented code or models is however still a matter of current research as in, e.g., [9, 10], and a Java code generation tool has been developed [6]. This tool is also a product of a smart card development project (*Project BOM*[2]), and takes care of some memory optimization issues. The development of BSmart builds upon this previous work. However, in this previous work, the generated code needs to be manually modified to incorporate the communication and codification aspects particular to the Java Card platform, whereas BSmart provides automated support to handle communication aspects.

At the moment of writing this paper, the definition of the method is in a mature stage, and our attention is now focused on the implementation of more robust versions of the BSmart tools and packaging them in a user-friendly environment. The integration of the AtelierB provers in Batcave and of a B animation tool is also planned for a next release of the tool.

References

1. Rankl, W., Effing, W.: Smart Card Handbook. John Wiley (2003)
2. Chen, Z.: Java Card Technology for Smart Cards: Architecture and Programmer's Guide. Addison Wesley (2000)
3. Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge U. Press (1996)
4. Gomes, B., Moreira, A.M., Déharbe, D.: Developing Java Card applications with B. In: SBMF. (2005) 63–77
5. Deharbe, D., Gomes, B.G., Moreira, A.M.: Automation of Java Card component development using the B method. In: ICECCS, IEEE Comp. Soc. (2006) 259–268
6. Tatibouet, B., Requet, A., Voisinet, J., Hammad, A.: Java Card code generation from B specifications. In: ICFEM. Volume 2885 of LNCS. (2003) 306–318
7. Marinho, E., Jr, V.M., Tavares, C., Déharbe, D.: Batcave - um ambiente de Verificação Automática para o Método B. In: SBMF. (2007) 184–184
8. Bert, D., et al.: Adaptable translator of B specifications to embedded C programs. In: FME. Volume 2805 of LNCS. (2003) 94–113
9. Idani, A., Ledru, Y.: Object oriented concepts identifications from formal B specifications. In: FMICS. (2004)
10. Tatibouet, B., Hammad, A., Voisinet, J.C.: From abstract B specification to UML class diagrams. In: ISSPIT. (2002)

[2] `lifc.univ-fcomte.fr/RECHERCHE/TFC/rntl_bom.html`