The use of B for Smart Card

Jean-Louis LANET

Gemplus Research Lab, Av. du Pic de Bertagne,

13881 Gémenos cedex BP 100.

jean-louis.lanet@gemplus.com

1. Introduction

In a previous paper [Lan-00] we stated that smart cards could be the ideal domain for applying formal methods. We said that the need of formal methods has three origins: mastering the complexity of the new operating systems, certifying at a high level a part of the smart card and reducing the cost of the validation. We believed that these reasons were enough to introduce formal methods in the software live cycle. Unfortunately the efforts for integrating data and behavior in a same framework for generating automatically test cases with model checkers, have not yet been successful. But we strongly believe that some others solutions for example JML Java Modeling Language [JML] can solve partially the test problem [Bur-02]. For the certification, the certification obtained by Multos did not encourage the other smart card manufacturers to propose high level certification due to the costs, even if Gemplus got an EAL5+ certificate. If certification helps to introduce formal methods this is just as a side effect. Finally it is the complexity of the operating systems and the need to avoid vulnerabilities that initiated the GemClassifier [Lan-02] smart card development.

We believe that a clean methodology with related metrics and tools improvements will consequently help the integration of formal methods and in particular the B method [Abr-96] in the software process [Cas-02]. It is important to have guidelines for the specifications and proofs that help the designers. For this purpose we joined a European project, named MATISSE[1]. The approach of the MATISSE [Mat-01] project is to exploit and enhance existing generic methodologies and associated technologies that support the correct construction of software-based systems. In particular, a strong emphasis was placed on the use of the B Method. Within this project, we evaluate the advantages and the drawbacks of using formal methods in our specific domain by applying a dedicated methodology on our case study.

2. The Gemplus case study

For the Java Card security, it is important that an applet can not have access to the data of other applets by using the sharing mechanism, or access to the code of the operating system.

---

[1] European IST Project MATISSE *IST-1999-11435*

The verifier is a key component of the Java security architecture. It examines incoming code in order to ensure that it respects the syntax of the byte code language and the language typing rules. The verifier checks statically that the control flow and the data flow do not generate run time error. Other components are responsible for protecting system resources from abuse but they depend on the verifier as they rely on language features such as access restrictions (private, protected, final, *etc*). It is obvious to say that a vulnerability in this component would be catastrophic for the card. We have specified and implemented such a verifier with all the Java Card byte code features treating the subroutine off-card [Cas-02b].

1.1. The sister developments

With this project, we wanted to determine if formal methods can be used in developing smart cards in such a way that gains in quality come at predictable and acceptable cost. By stating this goal we defined the expected effects of using formal methods. The hypotheses to be tested are detailed enough to make clear what measurements are needed to demonstrate the effects. We set up an evaluation plan that explained exactly the hypotheses to be tested and which metrics need to be recorded during developments in order to validate our claim.

In order to compare the effectiveness of formal methods in term of cost or quality from the case study result, it is necessary to have two developments to determine the differences. The formal development in the case study must have a sister project using the Gemplus procedure for development. The measurements take into account developments, proofs, reviews, tests and documentation for traditional and formal developments.

There is a small bias in this scheme due to the fact that the two development do not cover exactly the same algorithm. Both are byte code verifiers (type verification only) but the informal development relies on the regular algorithm specified by SUN [Lin-96] (we call it stand alone byte code verifier) while the formal one relies on the PCC algorithm[Nec-97]. This one is similar to the KVM verifier or to the Java lightweight verification that could be used for Java Card to perform similar verification on card while the regular algorithm is the one used with the standard Java Virtual machine. A complete description of the formal model has been presented in [Cas-02c]. At the end, both software have been embedded into smart card. The code is stored in the program area (ROM) while downloaded applet to be verified are stored in the data area (EEPROM).

In order to validate the developments, a common test plan has been specified and evaluated. The entry point in this process is the requirements that are derived from Sun on byte-code verification. These requirements only describe needs, without specifying any technical solution. Using them and Sun know-how on verification, informal specifications are generated. The informal specifications are formalized when performing the *Formalization*, in order to obtain a B model refined in B0 models. The B0 model is translated in C code, which is then optimized and/or compiled to obtain embedded code. Some of the activities are performed manually and as a consequence, errors can be introduced at some steps. A specific team was dedicated to the test, while a third team performed some reviews (informal specification).

A second bias has been introduced here, because the teams where not separated and some information flows occurred. The conventional development took benefit from the formal specification phase and also from test plan development.

1.2. The methodology

For the formal development, the methodology is well described in [Cas-02d], and we give here an insight on how to optimize the proof process which can be time consuming. We propose to ensure that each refinement step, from the formal specification to the formal implementation is coherent regarding its previous step. That is why the review of the specification is so important. The first step of the model has to be trusted and specification review is used to obtain this confidence. Then, thanks to the refinement process and the proof of each refinement process, we ensure the coherence of the whole model. Our proof process is divided in two parts: The first part of the proof activity is to tune the models in order to correct the models and to fit most of the *Atelier B* automatic prover constraints. The second part is the interactive proof of each remaining lemma, that cannot be discharged by the automatic prover

In the first step of the proof, the idea is to take in entry the models coming from the formal development. This development is relatively straightforward, going from the specification through its refinements and finally the formal implementation. During this development, the proof was not really considered. But now, correction to models has to be performed: the prover is mainly used as a debugger that indicates the errors of the development. The interactive prover helps to identify the lacks of the invariant in the models and in the loops, and indicates how to construct them. The goal of this part is to increase the automatic proof rate in order to remove as much as lemma as possible. The regular process is to look at the lemmas and if they seem to be true to not prove them with the interactive prover and to postpone this phase unless the model development is complete. At the end of the first step of our proof process, the models have been corrected and adapted to the *Atelier B* prover, the remaining lemmas have been analyzed and are supposed correct.

Then, the second part of the proof activity can start. It consists in manually proving the remaining lemmas thanks to the interactive prover interface provided by the *Atelier B*. By guiding the prover, the formal developer can help it to prove the lemmas. Of course, proof rules can still be developed to ease the manual proof. Once all the lemmas have been proved, the last task to perform is the proof of the added proof rules. Then, when all the identified tasks have been performed, we can claim that our model is entirely proved.

1.3. Main results

More detailed information about the results of the project can be found in [Lan-02b] and [Bur-03]. The different reviews discovered several errors in the informal specification and in the most abstract model of the B specification. Other errors where discovered during the proof process. They have generated hundred of unprovable lemmas that have not been immediately identified as false but have been corrected during the development phase. The test campaign exhibits errors of different origins. The first one is linked with the specification process (14 errors). If an error occurs during the translation from informal specification to formal specification the proof process is unable to detect them. The second type of errors (9 errors) were due to bugs into the translator. At the end of the refinement process, the final component is an implementation in a subset of the B language the B0 which is very close to C. The translator we used was a proprietary prototype with faults. This prototype has not been qualified according to the standard process.

	Formal development	Standard development procedure
Development workload	12 weeks	12 weeks
Proof workload	6 weeks	NA
Test workload	1 week	3 weeks
Integration	1 week	2 weeks
Total	**20 weeks**	**17 weeks**
Bugs discovered by review	13	24
Bugs discovered by proof	29	-
Bugs discovered with tests	32	71
Total	**74**	**95**

All the bugs discovered with test where at the boundaries of the formal method: at the beginning, the specification process and at the end, the code translation.

3. Conclusions

With this project we clearly demonstrate that the B method can be use for developing smart card components. It increases the quality of the development at an affordable cost. The overhead is not too important. But we have also to pay attention to the other software engineering techniques (review, test…) that must be combined with the formal method. In fact the key point is the correct integration of such a methodology in our software life cycle.

In this project we also formalize other parts that are not described here. It shows that not all the parts of the program need to use formal methods. For example, some low-level modules were entirely developed with B, requiring for the proof process significant efforts. Those modules could have been developed with the standard development procedure without reducing the confidence in the code. Formal method helps mastering the complexity of software and should be used in correlation with other techniques. But only the critical part of the software need to use such a method.

What has not been covered with this project is the application level. In fact, the customer buy a solution a platform and an application. The first point was to be sure to be able to built

trusted platform and then we need to provide trusted services to our customer. To which extend it is possible to model application with real object oriented features like Java card applet with the same level of confidence ? Is B the adequate solution to our problem or are they other solutions ? Then the last point will be to address the complete system and provide to the customer a global trusted system.

References

[Abr-96] J.R Abrial, *Assigning Programs to Meanings*, Cambridge University Press 1996.

[Bur-02] L. Burdy, A. Requet, *JACK: The Java Applet Correctness Kit*, to appear in GDC'02, November 2002, Singapore.

[Bur-03] L. Burdy, L. Casset, A. Requet, *Développement formel d'un vérifieur embarqué de byte code Java*, submitted in RSTI-TSI special issue on "*Développement rigoureux de logiciel avec la méthode B*".

[Cas-02] L. Casset, J-L. Lanet *Increasing Smart Card Dependability*, EW2002, St – Emilion, September 2002.

[Cas-02b] L. Casset, L. Burdy, A. Requet, *Formal Development of an Embedded Verifier for Java Card Byte Code*, DSN 2002, June 2002, Washington.

[Cas-02c] L. Casset, *Development of an Embedded Verifier for Java Card Byte Code using Formal Methods*, in Proceeding of FME 2002, Copenhagen, Denmark, July 2002

[Cas-02d] L. Casset, *Construction correcte de logiciels pour carte à puce*, Ph.D., Université de la Méditerranée, 2002.

[JML] The JML Home Page :http://www.cs.iastate.edu/~leavens/JML.html

[Lan-00] J.-L. Lanet, *Are Smart Card the Ideal Domain for Applying Formal Methods*, ZB 2000, August 2000, York.

[Lan-02] J.-L. Lanet, *GemClassifier a formally developed Smart Card*, HCSS conference, Linthicum, Maryland, March 2002.

[Lan-02b] J.-L. Lanet, L. Casset, D. Deville, *Java Card ™ Platform On-Card Byte code Verifier, the Ultimate Step*, Java One 2002, San-Francisco, http://servlet.java.sun.com/ javaone/sf2002/conf/sessions/display-2538.en.jsp

[Lin-96] T. Lindholm, F. Yellin, *The Java Virtual Machine Specification*, Addison Wessley 1996.

[Mat-01] Matisse Project IST-1999-11435, *E0, the Common Evaluation Plan*, 2001, http://www.matisse.qinetiq.com.

[Nec-97] G. Necula, P. Lee, *Proof Carrying Code*, 24th ACM SIGPLAN Symposium on Principles of Programming Languages, pp. 106-119, Paris, 1997