A B Formal Framework for Security Developments in the Domain of Smart Card Applications

Frédéric Dadeau, Marie-Laure Potet, Régis Tissot

Abstract We propose in this paper a formal framework based on the B method, that supports the development of secured smart card applications. Accordingly to the Common Criteria methodology, we focus on the formal definition and modelling of access control policies by means of dedicated B models expressing, on one hand, the access control rules, and, on the other hand, the dynamics of the system. These models are then weaved to produce a security kernel. From there, we propose a conformance relationship that aims at establishing whether a concrete representation of the system complies, at the security level, with the security kernel. This embraces both a well-defined notion of security conformance as well as traceability allowing to relate basic events appearing at the level of applications with abstract security policies. This approach is put in practice on an industrial case study in the context of the POSÉ project, involving both academic and industrial partners.

Key words: Access Control, B Method, Security Model, Traceability, Common Criteria, Conformance Relation

1 Introduction

Security requirements, as functional aspects, must be taken into account along the global development process of applications. For instance, in the Common Criteria (CC) approach, security is specified as a set of security

Frédéric Dadeau, Régis Tissot
Laboratoire d'Informatique de Franche-Comté, 16 route de Gray, F-25030 Besançon cedex, e-mail: {dadeau,tissot}@lifc.univ-fcomte.fr

Marie-Laure Potet
Laboratoire d'Informatique de Grenoble, BP. 72 – F-38402 Saint-Martin d'Hères cedex, e-mail: Marie-Laure.Potet@imag.fr

functional components [7] and the development process must then supply some assurances relatively to these security requirements [8]. The main assurance classes are relative to the design of the application to be evaluated (ADV), how functional testing have to be conducted (ATE) and to the vulnerability analysis (AVA). The result is a level of confidence, based on the measures taken during the development process. Furthermore, based on the functional and assurance components which are selected, CC evaluations rely on the principle of presentations of evidences, which explain how security functionalities are really guaranteed. For instance, testing assurance must establish how security requirements are covered. On the other hand, functional specifications must be established complete and consistent with the security functionalities requirements. The CC norm is then strongly based on the notion of specifications (models) and traceability (presentations of evidences).

On the other hand, smart cards play an important role in the information systems security. They supply a medium for authentication, confidentiality and integrity. Security in smart cards is based on hardware mechanisms and operating system protections and, at the level of applications, on some security properties that can be established. Because smart cards become a central piece of every day citizen security, as high-security identification cards, public transport, payment or healthcare cards, it is crucial to produce some evidences in term of security. Due to the increasing number of such applications, methodologies must be elaborated, in order to dispose of validation processes which can be reproduced and automated.

This paper presents a formal framework dedicated to the development of secure smart cart applications, based on the B method [2]. This work has been developed in the national french ANR project named POSÉ that aims at proposing an effective approach, adapted to security and development engineer practices. Section 2 describes the context of the POSÉ project and the proposed approach. Section 3 describes the security model level and Section 4 shows how conformance between an application and a security model can be characterized. Then, Section 5 illustrates our approach on a real smart card case study and its use in a model-based testing process. Finally, conclusion and perspectives are presented in Sect. 6.

2 The Context

The POSÉ project[1] is dedicated to the validation of smart card applications, using a model-based testing approach [22]. Model-based testing consists in using the formal model to support the computation of test cases and the associated test oracle, namely, the result expected after the execution of the test case. Then, abstract test cases have to be concretized to be run on the system under test. In this process, the major difficulty is to relate the abstract data

[1] `http://www.rntl-pose.info`

(operations signatures and abstract values) to the concrete data of the implementation (method signatures and concrete values). The partners[2] implied in this project are Gemalto, the world leader in smart cards technology suppliers, LEIRIOS Technologies and its model-based test generation LTG tool, SILICOMP-AQL an accredited organization in security certification and two academic partners, the LIG (Grenoble) and LIFC (Besançon).

2.1 The POSÉ Approach

A model-based testing approach is deployed at Gemalto, using the Leirios Test Generator tool [14] based on B. The POSÉ project addresses the extension of this approach to take specific security properties into account. More precisely, addressed topics are how security requirements can be formalized and linked to a functional model, in order to be exploited for security model-based testing. One of the major challenge was to re-use the concretization platform, which links the abstract specifications with the APDU communication level [13]. Furthermore, also for industrial reasons, a compatibility with the Common Criteria norm was expected, in order to re-use the methodology for validation testing, to satisfy evaluations levels EAL 4, 4+ or 5 [9].

The POSÉ project focuses on access control policies for several reasons. First, as pointed out previously, data protection is a central piece of security in smart card applications. Furthermore, this aspect becomes more important when smart card standardized platforms are considered. For instance, the POSÉ case study is based on the notion of objects (data, file and directory files) which carry their own access control rules. Thus, the correctness of the access control implementation is very crucial, as well as the personalization phase which instantiates the platform for a given set of controlled objects. Access control which is considered here is the control of subjects executing some operations on some of the protected objects. These rules depend on security attributes, such as the files life cycle. Second, we are interested in the dynamic aspects of access control policy, i.e., how permissions evolve during the system execution. Thus, a security model will describe both the access control entities (subject, object, conditional rules) and the dynamic part of the controlled operations. In this way, access control can be specified as a set of admissible traces. A similar approach is adopted by F. Schneider [19] who characterizes access control by security automata.

When security is dedicated to the control of operation execution, traceability and conformance consist in establishing a correspondence between behaviors admitted by the security model and behaviors admitted by the application. Contrary to functional conformance, where a specification and an implementation are compared, there is here no direct correspondence between secured operations and the interface of applications. Thus, a mapping

[2] `http://www.gemalto.com`, `http://www.leirios.com`, `http://www.aql.fr`

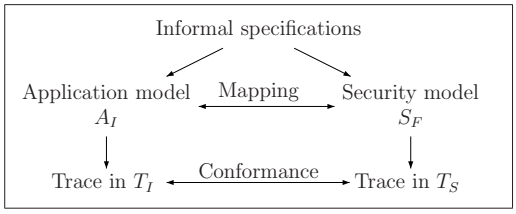

Fig. 1 Principle of the approach

relation must be given, in order to decompose application interfaces in terms of secured operations. The proposed approach is depicted in Fig. 1.

Intuitively, traces accepted by the application can also be accepted by the security model, through the mapping relation. Thus, the conformance reduces to the inclusion, apart from the mapping. In sections 3 and 4, we describe the formal framework capturing both modelling and conformance. This framework is based on the B method because B was already used in the existing model-based testing approach [6] and it is also well-suited to the definition of conformance.

2.2 A Brief Introduction to the B *Method*

The B method [2] is dedicated to the formal development, from high level specification to implementable code. Specifications are based on three formalisms: *data* are specified using a set theory, *properties* are first-order predicates and the *behaviors* are specified by *Generalized Substitutions*. A formal development process is supported through a refinement relation. The B method has been applied in industrial applications, such as the railway domain [3] and in the context of JavaCard application or environment [17, 5].

Generalized Substitutions can be defined by the Weakest Precondition (WP) semantics, introduced by E.W. Dijkstra [10], and denoted here by $[S]R$. $[x := e]\,R$ is the substitution of free occurrences of x in R by e. Table 1 presents other useful WP definition examples (in which z is a fresh variable).

From generalized substitutions, the following predicates can be computed (x being the state variables attached to the substitution S and x' the values of x after the substitution):

$\mathsf{trm}(S)$	$\hat{=}$	$[S]true$	termination
$\mathsf{prd}_x(S)$	$\hat{=}$	$\neg[S]\neg(x' = x)$	before-after predicate
$\mathsf{fis}(S)$	$\hat{=}$	$\exists x\ \mathsf{prd}_x(S)$	feasibility

Operation definitions are of the form $o \leftarrow op(i) \hat{=}$ PRE P THEN S END. An operation is characterized by its termination and its before-after predicate. The Event B extension [1], dedicated to dynamic aspects, is based on another execution model. Events are of the form SELECT P THEN S END.

$[x, y := e, f]\, R$	$\Leftrightarrow$ [z:=f][x:=e][y:=z]	multiple substitution
[skip] R	$\Leftrightarrow R$	null substitution
[PRE P THEN S END] R	$\Leftrightarrow P \wedge [S]\, R$	preconditioned substitution
[SELECT P THEN S END] R	$\Leftrightarrow P \Rightarrow [S]\, R$	guarded substitution
$[S_1 \ ; \ S_2]\, R$	$\Leftrightarrow [S_1]\, [S_2]\, R$	sequential substitution
[CHOICE S_1 OR S_2 END] R	$\Leftrightarrow ([S_1]R) \wedge ([S_2]R)$	bounded choice substitution
[VAR x IN S END] R	$\Leftrightarrow \forall$ x $[S]\, R$	substitution with local variable

Table 1 Some Weakest Precondition calculus definitions

An event is characterized by its before-after predicate and as soon as the event is feasible, it can be enabled. Feasibility is considered here as a guard.

Abstract models can be proved and refined (see Fig. 3 and 5 for examples). First, invariants can be stated: the proofs consist in showing that invariants are established by the initialization part and preserved by each operation definitions. Then, the refinement process consists in building a more concrete model and establishing the refinement relation. The refinement is based on a gluing invariant linking abstract and concrete variables. Refinement proof obligations consists in showing that the concrete initialization refines the abstract one, and that each concrete operation refines its abstract definition. A substitution S is refined by a substitution T, with respect to the gluing invariant L ($S \sqsubseteq_L T$) if and only if:

$$\boxed{L \wedge \ \mathsf{trm}(S) \ \Rightarrow \ [T]\neg[S]\neg L}$$

3 Formal Security Models

As stated in Sect. 2.1 we are interested by both access control rules and the dynamic evolution of security attributes. In order to be compatible with the Common Criteria security functional requirements, a security model will be constituted by two parts: a *rule-based model*, describing classical aspects of access control and a *dynamic model*, describing how security attributes evolve. The rule-based model corresponds to components of families FDP_ACC and FDP_ACF (access control policy and access control functions) of the Common Criteria, and the dynamic model corresponds to components of family FMT_MSA (Management of Security Attributes) [7]. These two models will be stated by means of B specifications. In this way, security properties can be proved as invariants relative to object, subject and security attributes.

3.1 B *Security Model*

The rule-based model describes which subjects are authorized to execute which operations on a controlled object, depending on some conditions relative to security attribute values. Permissions (the only kind considered here)

```
MACHINE e_purse_rules
SETS
   SUBJECTS={admin, bank, pda}; OPERATIONS={checkPin, credit, ...};
   MODE={perso, use, invalid}
CONSTANTS
   permission,
     /*  Security attributes:  */
   mode, isHoldAuth
PROPERTIES
   mode ∈ MODE ∧ isHoldAuth ∈ BOOL ∧
   permission ∈ (SUBJECTS ↔ OPERATIONS) ∧
     /*  Access control rules:  */
   (mode=use ⇒ (bank ↦ checkPin) ∈ permission) ∧
   ((mode=use ∧ isHoldAuth=TRUE) ⇒ (bank  ↦ credit) ∈ permission) ∧
   ...
END
```

Fig. 2 Formal model expressing the e-purse security rules

are defined as triplets belonging to a relation of the form `SUBJECTS` $\leftrightarrow$ `OPERATIONS` $\leftrightarrow$ `OBJECTS`, where $A \leftrightarrow B$ stands for a binary relation between A and B, and $a \mapsto b$ is the representation associated to pairs. Security attributes are specified as abstract constants and conditions are predicates on these constants. An example is given in Fig. 2.

In smart card applications, subjects generally correspond to the type of authentication and access control depends on the life cycle of the card or the applet. We illustrate this with an example of an electronic purse (e-purse), in which some operations may only be executed from specific terminals that represent the subjects. We distinguish three kinds of terminals: administrative terminals dedicated to personalization, bank and pda terminals. An access rule of the security policy states, for example, that the `checkPin` operation, that compares the holder PIN code for its authentication can only be executed from a bank terminal. In the same way, the holder authentication is also a security attribute. Another rule states that a `credit` command can be executed only if the holder has been authenticated.

The dynamic model describes how objects, subjects and security attributes evolve through a set of basic operations, including controlled operations. The dynamic model should fulfill several properties relative to the rule-based model: all controlled operations must be defined in the dynamic model; moreover, this latter must contain two special entities, named *subject* and *object*, denoting respectively the current subject and object values. The dynamic description of the `checkPin` operation is given in Fig. 3. Because, in this example, the access control does not imply any object, their related definitions are omitted.

```
MACHINE e_purse_dynamic
SETS
   SUBJECTS={admin, bank, pda}; MODE={perso, use, invalid}
VARIABLES
   subject, mode, isHoldAuth
INVARIANT
   subject ∈ SUBJECTS ∧ mode ∈ MODE ∧ isHoldAuth ∈ BOOL
INITIALISATION
   subject :∈ SUBJECTS || mode := perso || isHoldAuth := FALSE
OPERATIONS
   res ← checkPin(p) ≙
      PRE p ∈ 0..9999 THEN
          CHOICE isHoldAuth := TRUE || res := success
          OR isHoldAuth := FALSE || mode := invalid || res := blocked
          OR isHoldAuth := FALSE || res := failure
          END
      END
   ...
END
```

Fig. 3 Formal model expressing the e-purse dynamics

3.2 Security Kernel and Admissible Traces

From a rule-based model and a dynamic model, a security kernel, enforcing the rules of the first model on the second one, can be automatically generated by the Meca tool [12]. Let $out \leftarrow op(i) \hat{=}$ PRE P THEN S END be the definition of operation *op* in the dynamic model. Let $C \Rightarrow (s \mapsto op \mapsto o) \in$ *permission* be the unique rule associated to operation *op* (to simplify). The generated kernel contains the operation given in Fig. 4, describing how the execution of the operation *op* is controlled.

The security kernel specifies behaviors that are secure. Traces can be syntactically represented as sequences of occurrences of execution calls, stated as triplets (op, v, r) where *op* is an operation name, v a valuation of input parameters and r a valuation of output parameters. Then, a trace associated to a model M is written $< init\ ;\ (c_1, v_1, r_1)\ ;\ \ldots\ ;\ (c_n, v_n, r_n) >$ with $c_i \in Oper(M)$. In order to define admissible traces, the event corresponding to the execution of an operation call has to be defined. Let $out \leftarrow op(i)$

```
out, rs ← exec_op (i) ≙
   PRE pre_typ THEN /* typing of parameters */
      IF subject=s ∧ object=o ∧ C ∧ P
      THEN S || rs := OK ELSE rs := KO
      END
   END
```

Fig. 4 General format of an operation in the security kernel

be an operation defined by the substitution PRE P THEN S END. The event $exec(op, v, r)$, corresponding to the execution of the call $op(v)$ returning the value r, can be defined by the substitution:

SELECT $[i := v]P$ THEN
VAR out IN $[i := v]S$; SELECT $(r = out)$ THEN skip END END
END

Substitution into substitution, as $[i := v]S$, is defined as in [2]. As described in Sect. 2.2, a substitution can be characterized by its prd predicate. Here, we have $\mathsf{prd}(exec(op, v, r)) \equiv \exists\, out' \,/\, [i := v](P \wedge \mathsf{prd}(S) \wedge out' = r)$, that exactly describes the effect of an operation call for input v producing the value r as result. Now, let t be a trace of the form $< init\,;\ (c_1, v_1, r_1)\ ;\ \ldots\ ;\ (c_n, v_n, r_n) >$. This trace is *admissible* for the model M ($t \in T_M$) if and only if the condition $\mathsf{fis}(init\ ;\ exec(c_1, v_1, r_1)\ ;\ \ldots ; exec(c_n, v_n, r_n))$ holds.

For instance, $< init\,;\, S\,;\, (\texttt{checkpin}, < 1234 >, < success, OK >) >$ is an admissible trace if the predicate $mode = use \wedge subject = bank$ can be established after the sequence $< init\ ;\ S >$. We now present a conformance relation, based on this formal framework that aims at establishing whether or not an application conforms to a security model.

4 Conformance Relationship

Smart card applications are generally built as a set of commands, in the APDU format [13]. *APDU commands* supply the card with instructions to be executed and their parameters. *APDU responses* return results and a status word that contains the result of the command execution. Values of the status word are standardized; for instance, SW=9000 indicates that the commands terminated in the right way. Conformance is then based on some relations between APDU commands and abstract controlled operations.

Relatively to the security properties which are considered, ie., the control of commands execution, the granularity between operations which are controlled and the operations of the application is the same. Nevertheless, operations at the level of application and commands designed in the access control differ. In the first case, operations are defensive and can be invoked in any case (authorized case, security error or functional restriction) whereas operations of the security model are not executed if access control conditions do not hold, since they are considered as preconditions. Moreover, although status words are standardized, APDU responses are not always predictable. In case of multiple errors (for instance two security defaults or a functional restriction and a security error) application specifications do not impose any choice. This indeterminism is very important in the sense that implementations are free to favour one cause over another. A too precise specification could introduce a cause of channel side, making the implementation behavior

too predictable. As a consequence the mapping correspondence must support a form of underterminism to deal with multiple errors.

4.1 Mapping Security and Functional Models

We propose, hereafter, a definition of a mapping which is suitable for our application domain. A mapping is a set of rules stating how application calls can be related to controlled operations of the security kernel. In a more general case, a rule takes one of the two following forms:

1. $(op_{\mathrm{A}pp}, < v_{\mathrm{A}pp} >, < r_{\mathrm{A}pp} >) \rightarrow (op_{\mathrm{S}ec}, < v_{\mathrm{S}ec} >, < r_{\mathrm{S}ec}, \ \mathrm{OK} >)$
2. $(op_{\mathrm{A}pp}, < v_{\mathrm{A}pp} >, < r_{\mathrm{A}pp} >) \rightarrow (\mathsf{skip}, < \mathrm{KO} >)$

The first case maps an authorized behavior with a security behavior that describes a set of possible security attributes change. The second case corresponds to non authorized calls, in particular security attributes and the current subject and object must not be modified, in any way. $< v_{\mathrm{A}pp} >, < r_{\mathrm{A}pp} >, < v_{\mathrm{S}ec} >, < r_{\mathrm{S}ec} >$ denotes sequences of values or free variables. In the following, $(c_{\mathrm{A}pp}, c_{\mathrm{S}ec}) \in \ R$ means that there exists a rule $l \mapsto \ r$ and a substitution σ such that $\sigma(l) = c_{\mathrm{A}pp}$ and $\sigma(r) = c_{\mathrm{S}ec}$.

Here, we consider a restrictive case where the name of operations are identical and the input parameter values are equal. In this case, a mapping consists in establishing, for each application level command, a correspondence between results which are returned at the application level and associated result in the security model. Thus, a mapping takes the form:

$$\{< r^1_{\mathrm{A}pp} >, \ldots, < r^n_{\mathrm{A}pp} >\} \mapsto \ \{< r^1_{\mathrm{S}ec} >, \ldots, < r^k_{\mathrm{S}ec} >\}$$

Figure 5 describes a functional specification of our e-purse. The mapping R_1 hereafter is based on the fact that a thin observation of authorized behaviors is possible (`success`, `failure`, `blocked`):

$$\{< \texttt{9000} >\} \rightarrow \{< success, OK >\} \quad \{< \texttt{9202} >\} \rightarrow \{< blocked, OK >\}$$
$$\{< \texttt{9201} >\} \rightarrow \{< failure, OK >\} \quad \{< \texttt{9401} >, < \texttt{9402} >\} \rightarrow \{< KO >\}$$

A mapping can be non-deterministic, meaning that some results of the application level can belong to two sets. In this case, one result at the application level can correspond to different abstract results. Non-determinism is a way to deal with multiple errors. Suppose now that the dynamic part of our example (see Fig. 3) introduces a precondition of the form $p \in \mathbb{N}$ and a condition of the form IF $p \notin 0..9999$ THEN res := data_error ELSE ... END. The mapping R_1 is changed to R_2 in which $\{< \texttt{9401} >\} \rightarrow \{< KO >\}$ is replaced by $\{< \texttt{9401} >\} \rightarrow \{< data\_error, OK >\}$. Nevertheless a problem arises when the two conditions $p \in 0..9999$ and $mode = use$ do not hold. Depending on the order in which the verifications are performed in the ap-

```
sw ⟵ checkPin(p) ≙
  PRE p ∈ ℕ
  THEN
     IF p ∈ 0..9999
     THEN IF mode = use ∧ terminal = bank
          THEN
               IF p = pin
               THEN isHoldAuth := TRUE ‖ hptry := 3 ‖ sw := 9000
               ELSE isHoldAuth := FALSE ‖ hptry := hptry - 1 ‖
                     IF hptry - 1 = 0
                     THEN mode := invalid ‖ sw := 9202
                     ELSE sw := 9201
                     END
               END
          ELSE sw := 9402    /* mode ≠ use ∨ terminal ≠ bank */
          END
     ELSE sw := 9401    /* p ∉ 0..9999 */
     END
  END
```

Fig. 5 A functional model of the `checkPin` command

plication level, the result may differ. To overcome this problem, R_2 has to be extended, introducing a non-deterministic mapping, by adding the rule $\{< 9401 >\} \rightarrow \{< KO >\}$ (cf. Sect. 4.2).

4.2 Conformance Definition

Intuitively, an application conforms to a security model if and only if its traces are accepted by the security model, through the mapping relation. Due to the considered security policies, it means that: (*i*) all sequences of positive calls (associated to an effective execution of operations) can also be played by the security model, and, (*ii*) the application level can refuse more executions than the security level, in particular for functional reasons.

More formally, let $t_A = < init_A \ ; \ c_A^1 \ ; \ldots \ ; c_A^n >$ be a trace relative to the application and let $t_S = < init_S \ ; \ c_S^1 \ ; \ldots \ ; c_S^n >$ be a trace relative to the security model. A mapping relation R can be extended to traces in the following way:

$$(t_A, t_S) \in \ R \ \text{ iff } \ (c_A^1, c_S^1) \in \ R \ \wedge \ldots \wedge (c_A^n, c_S^n) \in \ R.$$

In this way, the set of traces associated to an application trace t_A can be computed. Now, operation calls that return KO can be assimilated to stuttering steps [16], because they do not modify security attributes. The operation *Stut* hereafter erases such calls.

$$\begin{array}{ll} Stut(< (\mathsf{skip}, < \text{KO} >) \ ; \ s >) & \hat{=} \ Stut(< s >) \\ Stut(< (c, < v >, < r, \text{OK} >) \ ; \ s >) & \hat{=} < (c, < v >, < r, \text{OK} >) \ ; \ Stut(< s >) > \\ Stut(<>) & \hat{=} <> \end{array}$$

Finally, the conformance between an application A and a security model S, through a mapping relation R, is defined by:

$$\forall\ ta\ (ta \in T_A \Rightarrow \exists\ ts\ ((ta, ts) \in R \wedge Stut(ts) \in T_S))$$

With this definition, it is possible to implement some part of the access control in a wrong way, for instance in making a mistake during the update of a security attribute. The conformity relation that we propose (only) verifies that a wrong implementation can not be used to obtain rights that are not authorized. Nevertheless, it is the main expected characteristics in security: a security failure which can not be exploited in any way is not really a problem.

The relevance of the proposed approach is based on the correctness of R. For a left part $(c_{\mathrm{A}pp}, v_{\mathrm{A}pp}, r_{\mathrm{A}pp})$ of a mapping rule, let $\{(c^i_{\mathrm{S}ec}, v^i_{\mathrm{S}ec}, r^i_{\mathrm{S}ec}) \mid i \in 1..n\}$ be the set of right parts associated to it. Correctness must ensure that modification of security attributes evolve in the same way, at the application and security levels. Then correctness must be stated by:

$$\begin{array}{c}\text{CHOICE}\ \ exec(c^1_{\mathrm{S}ec}, v^1_{\mathrm{S}ec}, r^1_{\mathrm{S}ec})\ \text{OR}\ \ldots\ \text{OR}\ exec(c^n_{\mathrm{S}ec}, v^n_{\mathrm{S}ec}, r^n_{\mathrm{S}ec})\ \ \text{END} \\ \sqsubseteq_L\ exec(c_{\mathrm{A}pp}, v_{\mathrm{A}pp}, r_{\mathrm{A}pp})\end{array}$$

with L the relation linking security attributes, subjects and objects with their representation at the application level (see Sect. 2.2). For instance if the correspondence $\{< \texttt{9401} >\} \rightarrow \{< KO >\}$ is omitted in R_2 then the correctness of R_2 (cf. Sec. 4.1) does not hold, because we can not establish that $subject = terminal \wedge\ p \notin 0..9999 \Rightarrow mode = use \wedge\ terminal = bank \wedge\ p \notin 0..9999$, where $subject = terminal$ is the gluing invariant linking variables of the security level with variables of the application level (other variables do not differ).

5 Applications in the POSÉ Context

In order to support interoperability and security, standards have been proposed by the main manufacturers. These norms define open platforms upon which standardized consumer and business applications can be built. It is thus very interesting to propose a methodology associated to the development and validation of applications based on such platforms. The IAS application [11], chosen as the case study of the POSÉ project, offers a notion of security data objects –SDO– that carry their own access control rules –SDO security header. An application developed on IAS consists in a personalization phase, giving a set of SDOs with their instantiated security headers. The IAS platform has been chosen in the French Administration project Adèle (`https://www.ateliers.modernisation.gouv.fr/ministeres/projets_adele/a125-ter-developpement/public`).

5.1 Description of the Models

Initially, only a functional model of the application was available. This model had previously been used to validate an implementation that has been established as conform to the functional model. Since this latter is very large (60 operations for about 15000 lines of B code), it was necessary to ensure its conformance w.r.t. the security requirements. We started by designing a model of the dynamics of the system, and, separately, we considered the access control rules. The resulting model was much simpler and more abstract than the original model (13 operations for about 1000 lines of B code). This model focuses on the file life cycle and access conditions based on pin authentication. Because of the limitation of the animator tool we use, these two models are deterministic. Let us consider an example, extracted from the case study.

We consider the *VERIFY* command which works as the *checkPin* operation of the e-purse example (Fig. 3) but it is parametrized by any PIN object. VERIFY permits either to get the authentication of a PIN object, if a PIN value is given, or to check its authentication state, if no PIN value is given. In the security model, the success of the VERIFY command depends on the existence of the PIN object and the validation of the access conditions which protect the PIN. The security model also deals checks the expected PIN value and the value of its tries counter.

In the security model, the status words of the command VERIFY are abstracted to *success, blocked, failure*, whose meanings are similar to those of the `checkPin` command. We define a mapping $M1$ (given hereafter) between the status word of the implementation and the abstracted ones –in 63Cx, x represents the number of remaining tries.

Status word on functional model	Meaning	Mapping with security model
9000	Success	$\{< success, OK >\}$
6983	SDO PIN unverified and no more tries	$\{< blocked, OK >\}$
6984	SDO PIN tries counter reached 0	$\{< failure, OK >\}$
63Cx	User authentication failed or not done	$\{< failure, OK >\}$
6A88	SDO PIN not found	$\{< KO >\}$
6982	Secure messaging erroneous or invalid access conditions	$\{< KO >\}$
6700	PIN value length is out of bounds	$\{< KO >\}$

Due to the complexity of the models, the mapping relation correction was not established by proof. It has been established by a review process based on the analysis of each branch of the code. Such form of validation seems to be at the level of smart card application developers because developers must understand both the security model and the application description, stating which status word corresponds to which internal behaviors.

5.2 Testing Methodology

The conformance relation that we have proposed is able to establish whether, or not, a trace is correct w.r.t. security requirements, in our case, the access control policy. In the POSÉ project, the model based validation approach is based on the Leirios Test Generator (LTG) tool [14] that offers an animator for B specifications and technics for generating tests from B specification. Animation will be both used to verify that a sequence can be played by a model and to compute parameter values or preambles to build a correct sequence. Moreover, a script has been developed in order to apply a mapping relation to any traces relative to the functional model.

In a first (ascending) approach, tests are produced at the application level and confronted to the security model kernel. In this way, the confidence of the functional model w.r.t the security one is improved. For instance, let us consider a functional model in which we have omitted to cancel the authentication on a PIN when an subsequent authentication fails –due to an erroneous PIN value. This error is observable in the example of Fig. 5. Then, the test sequence:

< (VERIFY,<pin1,1234>) ; (VERIFY,<pin1,4321>) ; (VERIFY,<pin1,_>) >

produces the output sequence <<9000> ; <63C1> ; <9000>>. But the animation of the sequence s for the mapped output sequence <<success, OK> ; <failure, OK> ; <success, OK>> fails to be established on the security kernel model. In order to experiment the defined conformance relation, we have performed mutations on model and checked that tests sequences generated from this model did not conform to the security model.
As pointed out before, only deterministic models can be taken into account by LTG. As a consequence, an abstract sequence computed from an application sequence can be easily animated by this tool, provide a very effective procedure for a test oracle. If unbounded non-deterministic models are considered, the feasability of the abstract sequences can be computed through a proof process. Finally, if the mapping relation is non-deterministic, at least one sequence must be accepted by the security model.

A second (descending) approach has been experimented in the POSÉ project. It consists in exploiting the security model to generate abstract test cases, which are completed with the help of the functional model. This approach is well-suited to the industrial process, since tests are built at the security level. Let s be an input sequence constituted by a sequence of command invocation and their input values. s can be played at the security level in order to obtain a result sequence r. Now, sequence s is played by LTG at the functional level. Notice that operations at the security level may have abstracted –and removed– parameters w.r.t. the functional level; these parameters are added when replaying the sequence s at the functional level. Thus, the animator looks for an instantiation of input parameters that makes it possible to successively execute each operation so that this latter results

in one of the expected outputs in r (modulo the mapping relation). If the instantiation is possible, we have the guarantee that the test conforms to the security requirements, and thus it can be played on the implementation. Otherwise, if the sequence is not be executable at the functional level, we not conclude on the conformance of the functional level w.r.t. the security level; indeed it is possible that the functional level is more restrictive than the security level, and requires additional operations of the functional level to be inserted along the sequence.

6 Conclusion and Future Work

As stated in the introduction, the approach proposed here has been developed in the framework of the RNTL POSÉ project, dedicated to verification and development of certifiable smart card applications. The B method has already been proved to be well-suited for smart card industries [6] and also, here, for modelling main entities of access control. Based on this method, Security Policy Model required from Common Criteria EAL5, can be easily specified, including dynamic aspects as preconized by some data protection class components. Due to the expressiveness of the B method, dynamic aspects can be captured in a more or less precise way. Moreover, notions of observability and refinement attached to the B models has been easily exploited in order to define a conformance relation including data refinement. This relation can be used both for testing or formal development approaches, as preconized by the ADV and ATE classes, particularly for high EALs.

The B method has already been used as a support for access control policies [4, 20]. In [4], the authors propose a form of modeling attached to Or-BAC access control and characterize behaviors which are conform to a given access control. The approach proposed here, can be seen as an extension of [4] and [20], in which dynamic conditions are taken into account as well as the observation of inputs, outputs and data refinement. In this way we have relate models which are stated at different levels of abstraction, as it is imposed by the Common Criteria approach. In [18], the authors use Labeled Transition Systems (LTS) to describe test purposes from Or-BAC rules specifying access control. They act as an oracle for the test execution, using based on the ioco conformity relation [21]. Our approach is similar, since they both rely on trace inclusions, and our notion of stuttering is close to the notion of quiescence. Nevertheless, our relation is not exclusively destined to be used as a test oracle. Indeed, by establishing preservation properties on our relation, it would be possible to prove properties on the implementation through the abstract security model.

We are currently leading experiments on using a combinatorial testing approach in order to generate test cases that exercise the security of the system. In this context, we plan to use the conformance relation as a test oracle, as illustrated previously.

Acknowledgements This work was supported by the RNTL POSE project (ANR-05-RNTL-01001), and partially funded by the Région Franche-Comté.

References

1. J-R. Abrial and L. Mussat. Introducing Dynamic Constrains in B. In D. Bert, editor, *Proceedings of the 2nd Int. B Conference*, volume 1393 of *LNCS*. Springer, 1998.
2. J.R. Abrial. *The B-Book*. Cambridge University Press, 1996.
3. P. Behm and all. Météor: A Successful Application of B in a Large Project. In *FM'99 - Formal Methods*, volume 1708 of *LNCS*, pages 348–387. Springer, September 1999.
4. N. Benaissa, D. Cansell, and D. Mery. Integration of Security Policy into System Modeling. In Julliand and Kouchnarenko [15].
5. D. Bert, S. Boulmé, M-L. Potet, A. Requet, and L. Voisin. Adaptable Translator of B Specifications to Embedded C programs. In *FME 2003: Formal Methods*, volume 2805 of *LNCS*. Springer, 2003.
6. F. Bouquet, F. Celletti, G. Debois, A. De Lavernette, E. Jaffuel, J. Julliand, B. Legeard, J. Lidoine, J.-C. Plessis, and P.-A. Masson. Model-based security testing, application to a smart card identity applet. In *eSmart 2006, 7th Int. Conf. on Smart Cards*, Sophia-Antipolis, France, September 2006.
7. Common Criteria for Information Technology Security Evaluation, Part 2: Security functional components. Technical Report CCMB-2006-09-002, version 3.1, sept 2006.
8. Common Criteria for Information Technology Security Evaluation, Part 3: Security assurance components. Technical Report CCMB-2006-09-003, version 3.1, sept 2006.
9. Common Criteria for Information Technology Security Evaluation, version 3.1. Technical Report CCMB-2006-09-001, sept 2006.
10. E.W. Dijkstra. *A discipline of Programming*. Prentice-Hall, 1976.
11. The Gixel web site. `http://gixel.fr`.
12. A. Haddad. Meca: a Tool for Access Control Models. In Julliand and Kouchnarenko [15].
13. Smart Card Standard: Part 4: Interindustry Commands for Interchange. Technical report, ISO/IEC, 1995.
14. E. Jaffuel and B. Legeard. LEIRIOS Test Generator: Automated Test Generation from B Models. In Julliand and Kouchnarenko [15].
15. J. Julliand and O. Kouchnarenko, editors. *B 2007: Formal Specification ans Development in B*, volume 4355 of *LNCS*. Springer, 2007.
16. Lamport. A temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, may 1994.
17. J-L. Lanet and A. Requet. Formal Proof of Smart Card Applets Correctness. In *CARDIS'98*, number 1820 in LNCS. Springer, 1998.
18. K. Li, L. Mounier, and R. Groz. Test Generation from Security Policies Specified in Or-BAC. In *COMPSAC – IEEE International Workshop on Secuirty in Software Engineering (IWSSE'07)*, Beijing, July 2007.
19. Fred B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, 2000.
20. N. Stouls and M-L. Potet. Security Policy Enforcement through Refinement Process. In Julliand and Kouchnarenko [15].
21. J. Tretmans. Conformance testing with labelled transition systems: Implementation relations and test generation. *Computer Networks and ISDN Systems*, 29(1):49–79, 1996.
22. M. Utting and B. Legeard. *Practical Model-Based Testing - A tools approach*. Elsevier Science, 2006. 550 pages.