A FOUR-CONCERN-ORIENTED SECURE IS DEVELOPMENT APPROACH

Michel Embe Jiague[1,2], Marc Frappier[1], Frédéric Gervais[2], Pierre Konopacki[1,2], Régine Laleau[2], Jérémy Milhau[1,2] and Richard St-Denis[1]

[1]*GRIL, Département d'informatique, Université de Sherbrooke, 2500 boulevard de l'Université Sherbrooke J1K 2R1, Québec, Canada*

[2]*Université Paris-Est, LACL, IUT Sénart Fontainebleau, Département Informatique Route Hurtault, 77300 Fontainebleau, France*

Keywords: Information system, Security policy, Access control, Formal method, Model checking, Process algebra, ASTD.

Abstract: In this paper, we advocate a strong separation of four aspects of information systems: data, dynamic behavior, security data and access control behavior. We describe how to model each of these aspects using formal methods. An abstract specification of each part of an information system is defined. The presented approach can be used when building a system from scratch but can also be applied to implement a security controller for an existing system. In parallel with models, properties of the system are written. These properties are checked against the system's models to ensure they hold using model checking techniques.

1 INTRODUCTION

Our aim is the formal specification of information systems (IS). Roughly speaking, an IS helps an organization to collect and manipulate all its relevant data. In this context, several characteristics and properties must be captured by the specification: definition and structuration of data collected and used in the IS, data integrity constraints, list of actions that can be invoked by end-users, views that can be displayed by end-users, action ordering constraints, data access control, security policies, etc. We strongly believe that formal methods can improve the quality of IS. Formal notations being amenable to automated analysis, they can be used for model-based software development, code generation and efficient interpretation, automated verification and testing, thereby reducing development costs, fostering reuse and increasing quality. In order to capture all these aspects, it seems that several layers of modeling should be considered. The proposed approach is inspired from the concept of *separation of concerns* (Parnas, 1972). The latter is generally used in the context of programming, like in aspect-oriented software development (AOSD) (Win et al., 2002). It consists in separating a program into distinct *concerns* like security, data logging, optimizations, etc. However, this paradigm can imply cross-cutting effects. For instance, the implementation of a concern can be dispatched over several parts, in that case, the program is no longer modular. The code of a concern can also be merged with code that implements other concerns. Such negative effects are common issues in aspect-oriented programming, probably because of the low level of abstraction. A survey of AOSD methods is available in (Schauerhuber et al., 2007). The main contribution of this paper is to model with formal notations the functional part and the access control part of an IS, verify properties on each part independently and on the union of all parts ensuring quality and reliability, and derive an implementation of the complete IS using systematic translations or symbolic execution of models.

1.1 The SELKIS and EB^{3}SEC Projects

The SELKIS project (ANR-08-SEGI-018)[1] from the French national research agency aims to define a development strategy for secure health care networks IS from requirements engineering to implementation. The results of this project can be applied to any type of secure IS, but the medical field was chosen because of the complexity and diversity of security requirements. EB^{3}SEC is a project of the Canadian research agency NSERC which aims to secure IS by enforcing

[1]http://lacl.fr/selkis/

Embe Jiague M., Frappier M., Gervais F., Konopacki P., Laleau R., Milhau J. and St-Denis R..
A FOUR-CONCERN-ORIENTED SECURE IS DEVELOPMENT APPROACH.
DOI: 10.5220/0003619604640471
In *Proceedings of the International Conference on Security and Cryptography* (MPEIS-2011), pages 464-471
ISBN: 978-989-8425-71-3

an access control policy modeled using formal notations without modifying a currently deployed system. Use cases of the EB^{3}SEC project are about banking IS that can involve many access control rules. For both projects, an approach based on formal methods was chosen in order to build specifications which can be rigorously validated by proofs and model checking over properties of the specification. In the approach of these projects, a separation is made between access control rules and the functional model at the requirement, specification and implementation levels. An access control filter is produced. It intercepts actions before they are executed. If the action and other parameters match access control rules, the action can be executed by the IS. In the other case, the execution is rejected and an error message is returned to the user.

1.2 The Approach

The key idea of our approach is to reuse the AOSD paradigm, but at a higher level of abstraction, more precisely at the specification level. We aim at dividing the IS specification into four concerns, which are here called *models*:

1. **Functional Models:**
   (a) The *Functional Data Model*: this concern deals with the definition of entities, associations, their attributes and the actions related to them. It also includes the safety properties (invariants) that the data attributes must satisfy.
   (b) The *Functional Dynamic Model*: this part describes the event ordering constraints and the dynamic properties. It provides a specification of the IS behavior.
2. **Access Control Security Models:**
   (a) The *Security Data Model*: this third part also deals with entities, associations and attributes and defines a static access control policy composed of rules which describes *who* can do *what* in the system. Static rules are state invariant, *i.e.* they are not based on the state of the system.
   (b) The *Security Dynamic Model* also called access control model: this last concern also deals with access control policies, but adds the description of *when* (in which context) an action can be performed. Such rules can refers to previously executed actions or previous states of the IS. This kind of policy allows the IS to be more secure, enabling access only to those that are entitled to at the right moment in the business process of the organization.

By using formal notations, verification techniques may be used to check properties on each model and to verify consistency between the four parts. The chosen separation of models is justified as follows. Since we are dealing with IS, end-users are naturally focused on data. The dynamic model provides more information than the sole data model. It can be viewed as the controller of the IS. Data and dynamic models constitute the functional model of the system. Distinguishing these two parts is somehow analogous to comparing state-based and event-based models. The two security models are justified by the dependency of most organizations on the proper functioning of their IS, to prevent corruption, loss of data or breaches in confidentiality, which may have serious consequences. Such a separation of concerns also simplify system evolution. Access control policies often change independently of the functional model of the system. Maintenance teams experience a high turnover rate during the lifetime of an IS. These four models provide a well-decomposed specification which is easier to grasp and maintain.

Even if other choices could have been made (for instance, security could be split into several distinct models, one for each ACIT - Availability, Confidentiality, Integrity, Traceability - feature), one of the key elements in our approach is the incremental building of IS models. First, the data model only provides data definition and structure. Hence, there is no restriction on the potential behavior of the IS, only excluding data values violating integrity constraints. Then, the dynamic behavior restricts the potential IS event orderings. Finally, the security dynamic model reduces the number of possible behaviors by taking the security facets into account.

Fig. 1 summarizes our proposal and introduces properties such as INV, SCE, SCEF and NCE that will be explained in Section 4. The rest of this paper is structured as follows. Section 2 describes the software development process of the 4CO approach, putting a special emphasis on model-driven engineering, rather than traditional software design and implementation. Section 3 describes the techniques used for specifying data, behavior and security policies. Section 4 describes the techniques used for specifying properties on data, behavior and security policies, and for verifying them. Finally, Section 5 reviews related approaches and research projects.

2 IS DEVELOPMENT PROCESS

A four-concern-oriented approach is proposed to formally specify safe and secure IS. This section pro-

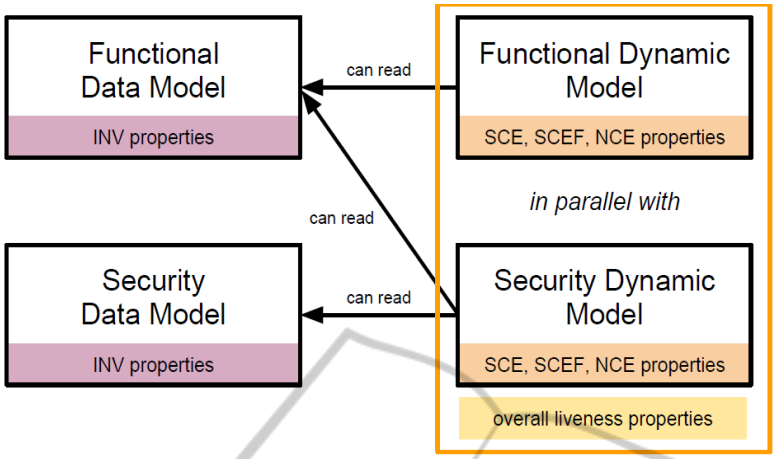

Figure 1: Our proposal: a four-concern-oriented secure IS development approach.

vides a general overview of the target methodology, which consists of nine steps. Most of them can already be supported by existing languages and tools from the EB^{3}SEC (Embe Jiague et al., 2010) project that we have developed in our research groups.

The whole specification is intended to be built incrementally, but some exceptions may occur, especially when maintenance or monitoring issues are considered. For the moment, we only consider an IS specification from scratch. Specifying concerns for an existing IS is discussed at the end of this section.

Our approach is composed of the following steps:

1. **Specifying the Functional Data Model.** The first step consists in defining the main entity types, associations and attributes. In our approach, this part is described with class diagrams and attribute definitions.
2. **Specifying the Functional Dynamic Model.** The next step is the definition of the IS controller, expressing ordering constraints on actions. We have a choice of two complementary, but equivalent notations. The EB^3 process algebra allows us to represent the dynamic behavior using purely algebraic expressions (Frappier and St-Denis, 2003). Algebraic state transition diagram (ASTD) (Frappier et al., 2008) is an hybrid automata-based and algebra-based formal language with a graphical notation which allows the specifier to describe behavior using hierarchical automata that can be freely combined with process algebra operators. These formal languages can be used to specify the functional dynamic model.
3. **Verifying Data Integrity Constraints.** Once the functional data model is defined, we use a translation into Event-B (Abrial, 2010) for checking invariant properties like data integrity constraints. We have developed systematic translation rules from EB^3 process expressions to ASTD and from ASTD to Event-B machines (Milhau et al., 2010).
4. **Verifying Dynamic Properties.** The aim of this step is to verify safety properties that involve actions, such as necessary conditions to enable an action, and liveness properties, such as sufficient conditions to enable or reach an event. Model checking and proving dynamic properties on our models is presented in (Frappier et al., 2010).
5. **Specifying Access Control Data.** The first two models constitute the IS functional specification. Next, security policies have to be defined. We use a class diagram to represent static access control policies, in a way similar to *-RBAC approaches (Ferraiolo et al., 2003).
6. **Specifying Access Control Security Policies.** More complex access control rules, involving workflows and action ordering, can be expressed in this model as presented in (Konopacki et al., 2010).
7. **Verifying Security Properties.** In this step, verification techniques are performed to verify specific security properties.
8. **Verifying Overall Liveness.** Finally, we need to verify that the functional model coupled with the security model does not prevent actions from being executed.
9. **Maintaining Models.** IS designers may have to modify the IS models after some time. Our approach offers a way to update the system without modifying every models, even if the system is running.

This process is not linear. Some iterations are required for obtaining a complete specification. For instance, if the security requirements are too strong, some users can no longer perform desired actions. In that case, IS specifiers have to take some decisions to weaken the rules of the security model.

Since our approach is incremental, one can apply steps 2, 4 – 8 to an existing IS. A prerequisite is a data model coming from the IS documentation. Then, a controller can be executed for monitoring the valid sequence of IS actions. This is relevant, for instance, in a service-oriented approach, where our development process could be used for service orchestration and choreography.

3 SPECIFICATION

In this section we introduce a class diagram, the ASTD notation and a case study that we use to illustrate our approach. Specification refers to points 2, 5 and 6 of our approach.

3.1 Security Data Model

We need to model access control policies with various constraints such as permissions, prohibitions, SoD (Separation of Duty) and obligations. SoD constraints are dynamic and defined as a workflow in the next section. Sometimes only permissions or prohibitions are sufficient to model a static access control policy. A more complete description of permissions, prohibitions, SoD and obligations is given in (Konopacki et al., 2010).

The class diagram depicted in Fig. 2 is highly inspired by the RBAC standard (ANSI, 2004). Entities *User*, *Role* and *Action* respectively depict users involved in the IS, the role they play and actions of the IS. The association *Play* depicts who is allowed to play a role in the IS. Relations *Permission* and *Prohibition* depict permission and prohibition in the IS. Actions are granted or prohibited to a role that can be played by different persons. The semantics of the class diagram is expressed by a predicate, called the static predicate $sp(\sigma)$, which determines if an event σ satisfies the constraints expressed in the class diagram. For instance, an RBAC-like static predicate would state that an event $\sigma = < u, r, a >$ is valid iff

$$(u,r) \in Play \wedge (r,a) \in Permission \wedge (r,a) \notin Prohibition$$

3.2 The ASTD Notation

The ASTD notation is a formal and graphical notation for specifying IS functional dynamic behaviors. The notation was introduced in (Frappier et al., 2008) as an extension of Harel's Statecharts with process algebra operators. Each ASTD type corresponds to either a process algebra operator or a hierarchical automaton. Automaton states can be elementary or an ASTD of any type.

ASTD specifications combine benefits of graphical representation and process algebra: easy to read representation of states from automata; control behavior, hierarchical structure and the power of abstraction of process expressions. An ASTD has a static topology, *i.e.* its structure, and a state that evolves according to actions executed. Contrary to a process expression, when an ASTD is executed, only its state evolves, whereas the execution of an action on a process expression returns a new process expression, whose structure is sometimes more complex due to unwinding of quantifications, closures and process calls. Hence, an animation of the specification for validation with end-users is more difficult to understand. Fig. 3 provides a small example of an ASTD specification for a library IS managing members, books and loans. Each component is described using an automaton and combined using process algebra operators from EB^3 (Frappier and St-Denis, 2003). Each event can has several parameters. The first two parameters denote security attributes as *role* and *user* from the security data model. Any relevant attribute could be added to the class diagram and then used in ASTD events as a security parameter.

ASTD is an executable notation, with reasonable performance to handle large numbers of entities and instances, and thus can be used for controlling IS with low response time constraints. Using *i*ASTD, our ASTD interpreter, an implementation of the model can be produced. It can stand as a controller of the system. It updates its state to reflect the changes induced by the actions executed.

The ASTD notation can also be used to specify security dynamic models, by introducing new parameters to actions such as user identifier, role, organization or any additional parameter needed to express dynamic security rules. Sophisticated access control rules can be expressed by using process expressions. In our library IS example, an SoD constraint is added to request and acquire actions. The separation of duty comes from the guard $uId \neq uId'$ implying that the user performing the action request must be different from the user performing the action acquire. For instance, the user must play the role *professor* to be granted to make a book request. An obligation constraint is also added to acquire and discard. In other words, the request for a book must be done by a professor, and another user must acquire it. Those con-

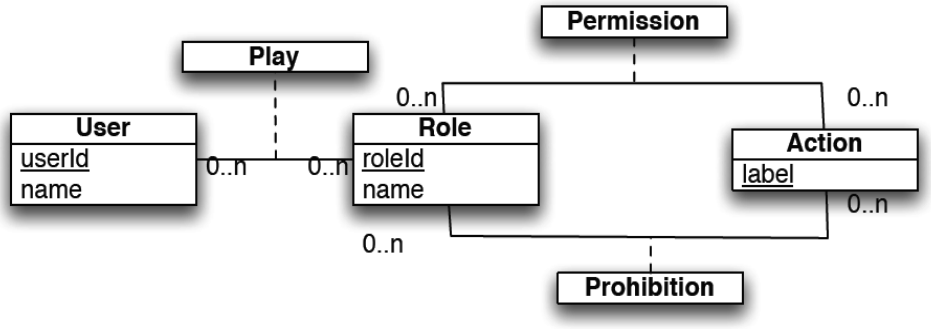

Figure 2: Class diagram used to model access control policies.

straints are expressed in the ASTD specification presented in Fig. 4.

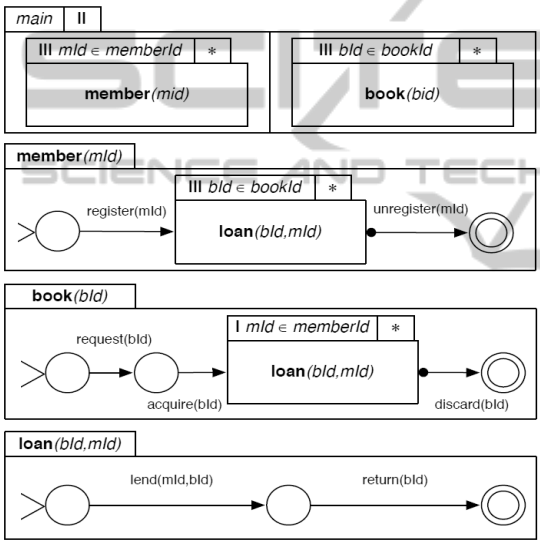

Figure 3: A complete ASTD specification for a library IS.

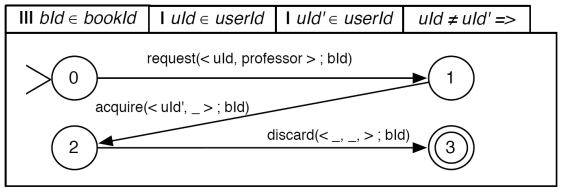

Figure 4: An ASTD specification for a security rule of a library IS.

4 VERIFICATION OF PROPERTIES

This section describes points 3, 4, 7 and 8 from the approach where properties are verified against produced models.

4.1 Classification of Properties

One advantage of using formal methods is the ability to verify properties over system models, whether they are functional or security models. Properties are traditionally partitioned into three categories:

1. *Liveness Properties* state that something good eventually happens. They sometimes state that an action implies a reaction from the system; this is however rarely used in IS, where actions are human-driven (one cannot force a user to trigger specific actions). Deadlock, in the traditional sense of reaching states where no transition is possible, is rarely an issue.
2. *Safety Properties* state that something bad does not happen. IS safety properties usually describe invariant properties of data, *necessary conditions* to enable an action, or what a user is *not allowed* to do with the system at a given point.
3. *Fairness Properties* state that an enabled action will be executed at some point. Since end-users determine which actions they want to execute, IS typically do not have fairness constraints.

In (Konopacki et al., 2010), we have identified four sub-categories which frequently occur in IS requirements, and for which we want to develop a verification strategy.

4.1.1 Liveness

1. **SCE: Sufficient State Condition to Enable an Event.** Such properties state that end-users should be able to execute an action when a condition holds. "An action is enabled" means that there is a transition for this action in the current state. Examples are: i) a book can always be acquired by the library when it is not currently acquired; ii) a member can always return a borrowed book. Thus, the IS must accept acquire(b) when book b does not exist; the IS must accept return(b) when b is borrowed.

2. **SCEF: Sufficient State Condition to Enable an Event in the Future.** Such properties state that there is a sequence of actions that leads to the execution of the desired action. Examples are: i) ultimately, there is always a procedure that enables a member to leave library membership (the procedure depends on the loans and reservations of the member); ii) a member can always ultimately borrow a book (the procedure depends on the status of the book and its reservations). Thus, the action can become enabled if proper steps are taken.

4.1.2 Safety

1. **INV: Invariant State Property.** These properties hold for each state of the system. For example, a member cannot borrow more books than a certain limit.
2. **NCE: Necessary State Condition to Enable an Event.** These properties state that if the IS can accept an action, then a condition must hold. Examples are: i) a book can only be acquired if the buying request was made by a professor; ii) a book cannot be reserved by the member who is borrowing it.

4.2 Specifying Properties

4.2.1 SCE and SCEF

CTL is well suited to specify SCE and SCEF properties, due to its branching nature which is useful for expressing *enabledness* of actions. Ideally, a CTL-like (*i.e.*, branching) language should support both states and actions, as in PROB (Leuschel and Butler, 2003), since SCE and SCEF properties typically refer to both states and actions in their natural language formulation. Classical LTL is not suitable, because of its inability to express *enabledness*. Particular features of some model checkers (like the enabled operator in ProB and location labels in SPIN (Holzmann, 2004)) slightly extend LTL such that some SCE and SCEF properties can be specified in LTL. Stable-failure refinement, as in FDR2, can also be used to specify SCE and SCEF, but not as easily and naturally as in CTL. Plain first-order logic can also be used for SCE and SCEF when actions are specified as before-after state predicates. In (Frappier et al., 2010), we have shown how ALLOY (Jackson, 2006), FDR2 and PROB can be used to specify and check SCE and SCEF properties.

4.2.2 NCE and INV

These properties are more widely supported. For example CTL, LTL, first-order logic and trace refinement can all be used.

4.3 Generic Security Properties

Specifying different kinds of properties on a given model and checking them on this model ensure a high level of correctness. Properties expressed by designers are called *specific properties*. In addition to these properties we have identified properties called *generic properties*. They can be checked on models to ensure their consistency and liveness. Ten generic properties have been identified as for example:

Access Control Policy Typing checks that all events used in process expressions are also instances of the class *Action*, otherwise the static predicate cannot be verified. This property can be statically checked over the access control model.

Role Feasibility checks that all instances of the class *Role* can be used by a user to execute at least one action. This property is generalized to all classes of the access control data model. They are used to check that all instances are useful. Furthermore, we have equivalent properties for associations of the access control data model. These verifications are also used for association *permission*, to check that all permissions depicted in the data model are reachable in the dynamic model.

Prohibition Unfeasibility checks that what is prohibited in the class *prohibition* cannot be executed by the access control dynamic model.

Reachability of all Security Events checks that all instances of class *Action* in the access control data model can be executed by the access control dynamic model.

Reachability of all IS Events checks that the access control policy is not too restrictive, *i.e.*: all events, that the IS could execute, can be executed at least once by the access control model.

All these properties can be formalized. For example, *Reachability of all IS events* can be expressed using CTL as follows:

$$\forall e \in \Sigma_{FN} : \mathcal{M}_{FN} \models \mathsf{EF}(e) \Rightarrow \exists \sigma \in \Sigma_{SEC} : \sigma.e = e \wedge sp(\sigma) \wedge \mathcal{M}_{SEC} \models \mathsf{EF}(\sigma)$$

In this formula, Σ_{FN} (Σ_{SEC}, resp.) is the set of events used in the functional dynamic model $\mathcal{M}_{FN}$ (security

dynamic model $\mathcal{M}_{SEC}$, resp.); E and F are symbols used in CTL to respectively express *exists* and *finally*; σ is a security event (*i.e.*: a *n*-tuple) and $\sigma.e$ is the event of σ. There are two problems to make this property usable in a model checker : i) the quantification is over events, and most model checkers do not support such quantifications; thus they must be unfolded, ii) two distinct models (*i.e.*: functional and access control) are used, thus two checks are done in sequence in the unfolding.

4.4 Verifying Properties

Model checking is an interesting technique to verify IS specifications for several reasons: it provides broader coverage than simulation or testing, it requires less human interaction than theorem proving, and it has the ability to easily deal with both safety and liveness properties.

The separation of concerns illustrated in Fig. 1 allows for a modular verification of properties. Invariant properties can be checked separately in the functional model and the security model, since one model cannot modify the state of the other. NCE can also be checked locally in each model: the functional model and the security model are composed in parallel and must synchronize on common actions; thus the traces of the parallel composition of these two models are included in the traces of each component, when restricted to their respective alphabets. SCE and SCEF can be checked locally, but the parallel composition of the two models may further restrict the enabledness of actions, thus violating these properties. The specifier must identify which properties should still hold in the parallel combination and re-check them. Some of the generic properties guaranty liveness of the combination and must be checked over the parallel composition of the functional model and the security model.

5 RELATED WORK

Few methods have been developed to formally specify and verify secure information systems. If we consider only the functional aspect, we have pointed out in (Fraikin et al., 2005) that it is interesting to combine state-based and event-based models to express all kinds of properties of IS. State-based methods have been extensively used because they are suitable to describe states of IS, with invariant predicates to represent static integrity constraints, two important aspects of IS. Dynamic properties are more tricky to specify. Even if complex ordering of operations can be expressed by using additional variables and pre-conditions, this kind of properties are more naturally and clearly expressed by event-based methods. In our work, we combine the B method and EB^3. Other combinations can be considered. In (Evans et al., 2008), we propose to combine B and CSP, mainly because of existing tools.

With regard to the security aspect, there exist many formalisms to specify security properties (Konopacki et al., 2010). However most of them consider basic access controls. Few research projects deal with separation of duty. The work in (Basin et al., 2010) uses CSP with an RBAC model. Because CSP does not support natively state variables, it is not well adapted for IS. Moreover, the access control model is RBAC and cannot be extended or tailored to specific IS as we propose in our work.

Finally, when combining functional and security aspects, comparable research projects are (Basin et al., 2009) and (Kallel et al., 2009). The first one is based on SecureUML (Basin et al., 2006) and expresses security properties with OCL. OCL being a state-based method, dynamic properties, such as constraints on workflows, may require to encode workflow manually and describe the evolution of the state for each action executed. Moreover, verification tools associated with OCL are less powerful than ProB, Alloy and FDR2. The second one is based on RBAC, Z, temporal logic (LTL) and aspect-oriented languages for implementation purposes. Regarding RBAC, the same limit as mentioned above can be stated. In addition, LTL is not sufficient to express important security properties, as described in Section 4 where we point out that at least CTL has to be used.

6 CONCLUSIONS

We presented our approach of secure IS design, in which a strong separation of four essential parts of a system is performed at the specification level. Each of these parts is described by a model expressed with a formal notation. When writing a model, properties that must be checked are also expressed. Our approach alternates between model specification and model checking in order to ensure consistency and cohesion between all aspects of the system. Model-specific and overall verifications are performed. The relative independency between models also helps IS designers to update the system after it is deployed. As validation of our approach, we have developed interpreters, systematic translation procedures and implementations based on models described in this paper. Presently we can interpret access control policy spec-

ifications in order to provide a filter that will grant or deny the execution of actions. In (Milhau et al., 2010) we present systematic translation rules from the ASTD notation to Event-B (Abrial, 2010). We are currently working on a translation of such Event-B access control policy specifications into Business Process Execution Language (BPEL) that can be enforced in a service oriented architecture (SOA) system (Embe Jiague et al., 2011). As future work we plan to provide tools in order to model and verify ASTD specifications and implement the algorithms of (Milhau et al., 2010).

ACKNOWLEDGEMENTS

This research is funded by ANR (France) as part of the SELKIS project (ANR-08-SEGI-018) and by NSERC (Canada).

REFERENCES

Abrial, J.-R. (2010). *Modeling in Event-B*. Cambridge University Press.

ANSI (2004). *American national standard for information technology – role based access control.* ANSI INCITS 359–2004.

Basin, D., Burri, S. J., and Karjoth, G. (2010). Dynamic enforcement of abstract separation of duty constraints. In *Computer Security – ESORICS 2009*, LNCS. vol. 5789, pp. 250–267, Springer, Berlin Heidelberg.

Basin, D., Doser, J., and Lodderstedt, T. (2006). Model driven security: From UML models to access control infrastructures. *ACM TOSEM*, 15(1):39–91.

Basin, D. A., Clavel, M., Doser, J., and Egea, M. (2009). Automated analysis of security-design models. *Information & Software Technology*, 51(5):815–831.

Embe Jiague, M., Frappier, M., Gervais, F., Konopacki, P., Milhau, J., Laleau, R., and St-Denis, R. (2010). Model-driven engineering of functional security policies. In *INSTICC Press*, volume Information Systems Analysis and Specification, pages 374–379, Funchal, Madeira.

Embe Jiague, M., Frappier, M., Gervais, F., Laleau, R., and St-Denis, R. (2011). From ASTD access control policies to WS-BPEL processes deployed in a SOA environment. In Chiu, D. K. W. and al., editors, *WISS 2010 Workshops*, LNCS. vol. 6724, pp. 126–141, Springer, Berlin Heidelberg.

Evans, N., Treharne, H., Laleau, R., and Frappier, M. (2008). Applying CSP - B to information systems. *Software and System Modeling*, 7(1):85–102.

Ferraiolo, D. F., Kuhn, D. R., and Chandramouli, R. (2003). *Role-Based Access Control*. Artech House, Inc., Norwood, MA, USA.

Fraikin, B., Frappier, M., and Laleau, R. (2005). State-based versus event-based specifications for information system specification: a comparison of B and EB^3. *Software and Systems Modeling*, 4(3):236–257.

Frappier, M., Fraikin, B., Chossart, R., Chane-Yack-Fa, R., and Ouenzar, M. (2010). Comparison of model checking tools for information systems. In Dong, J. and Zhu, H., editors, *Formal Methods and Software Engineering*, LNCS. vol. 6447, pp. 581–596, Springer, Berlin Heidelberg.

Frappier, M., Gervais, F., Laleau, R., Fraikin, B., and St-Denis, R. (2008). Extending statecharts with process algebra operators. *Innovations in Systems and Software Engineering*, 4(3):285–292.

Frappier, M. and St-Denis, R. (2003). EB^3: an entity-based black-box specification method for information systems. *Software and Systems Modeling*, 2(2):134–149.

Holzmann, G. J. (2004). *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley.

Jackson, D. (2006). *Software Abstractions*. MIT Press.

Kallel, S., Charfi, A., Mezini, M., Jmaiel, M., and Klose, K. (2009). From formal access control policies to runtime enforcement aspects. In Massacci, F., Zannone, N., and Redwine, S. T., editors, *Engineering Secure Software and Systems*, LNCS. vol. 5429, pp. 16–31, Springer, Berlin.

Konopacki, P., Frappier, M., and Laleau, R. (2010). Expressing access control policies with an event-based approach. Technical Report TR-LACL-2010-6, LACL, Université Paris Est.

Leuschel, M. and Butler, M. (2003). ProB: A model checker for B. In Araki, K., Gnesi, S., and Mandrioli, D., editors, *FME 2003: Formal Methods*, LNCS. vol. 2805, pp. 855–874, Springer, Berlin Heidelberg.

Milhau, J., Frappier, M., Gervais, F., and Laleau, R. (2010). Systematic translation rules from ASTD to Event-B. In Méry, D. and Merz, S., editors, *Integrated Formal Methods*, LNCS. vol. 6396, pp. 245–259, Springer, Berlin H.

Parnas, D. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12).

Schauerhuber, A., Schwinger, W., Kapsammer, E., Retschitzegger, W., Wimmer, M., and Kappel, G. (2007). A survey on aspect-oriented modeling approaches. Technical report, Vienna University of Technology.

Win, B. D., Vanhaute, B., and Decker, B. D. (2002). How aspect-oriented programming can help to build secure software. *Informatica (Slovenia)*, 26(2).