MODEL-DRIVEN ENGINEERING OF FUNCTIONAL SECURITY POLICIES

Michel Embe Jiague[1,2], Marc Frappier[1] Frédéric Gervais[2], Pierre Konopacki[1,2],
Régine Laleau[2], Jérémy Milhau[1,2], Richard St-Denis[1]
[1]*GRIL, Département d'informatique, Université de Sherbrooke, Sherbrooke (Québec), J1K 2R1, Canada*
{*Michel.Embe.Jiague,Marc.Frappier,Pierre.Konopacki, Jeremy.Milhau,Richard.St-Denis*}*@USherbrooke.ca*
[2]*LACL, Université Paris-Est, IUT Fontainebleau, 77300 Fontainebleau, France*
{*Frederic.Gervais,Laleau*}*@u-pec.fr*

Keywords: Security model, security policy, specification, verification, process algebra, hierarchical state transition diagram, EB^{3}SEC, EB^3, MDA, SOA, BPEL.

Abstract: This paper describes an ongoing project on the specification and automatic implementation of functional security policies. We advocate a clear separation between functional behavior and functional security requirements. We propose a formal language to specify functional security policies. We are developing techniques by which a formal functional security policy can be automatically implemented. Hence, our approach is highly inspired from model-driven engineering. Furthermore, our formal language will enabled us to use model checking techniques to verify that a security policy satisfies desired properties.

1 Introduction

Information systems (ISs) are prevalent in today's economy. Public companies as well as private corporations have most of their ISs on-line and rely on data exchange with customers and partners to carry out their day-to-day business. Due to their nature, they must be highly accessible over Internet.

Strict security and privacy regulations are imposed on financial and health sectors. For instance, health software applications in the USA must comply with HIPAAA (Health Insurance Portability and Accountability Act); those in Canada with PIPEDA (Personal Information Protection and Electronic Documents Act). A financial software application must satisfy a security policy according to the Sarbane-Oxley law in the USA.

We distinguish between *functional security*, which deals with security requirements at the *business level*, and *architectural security*, which deals with software design level issues, like authentication, encryption, and secure communication protocols. Functional security essentially determines *who* can do *what*, *when* and *where*.

The aforementioned laws and regulations mostly deal with functional security. Unfortunately, functional security is poorly managed in most organisations. Requirements are often vaguely described and handled in various disconnected parts of a software systems. For instance, basic role-based access control, RBAC (Ferraiolo et al., 2003) is used to grant access to a service. But RBAC does not depend on the state of the service; authorization is granted solely on the basis of the role of the user and the service requested. Additional functional security requirements are thus handled in the service itself. For instance, business requirements that determine who can do what and when are mixed with functional requirements and implemented together in the service code. As most organisation and popular agile software processes advocate little documentation, the code is the ultimate description of functional security policies. It is then highly difficult to demonstrate that a software satisfies security regulations. It is even more difficult to evolve a security policy to satisfy new requirements.

To deal with this issue, we propose a new approach that is based on the following principles. **1)** Business requirements should be clearly separated in two parts: i) functional behavior, which states what the system services should do; ii) functional security, which states who, when, and where can these services be used. **2)** Functional behavior and functional security should be separately implemented in distinct

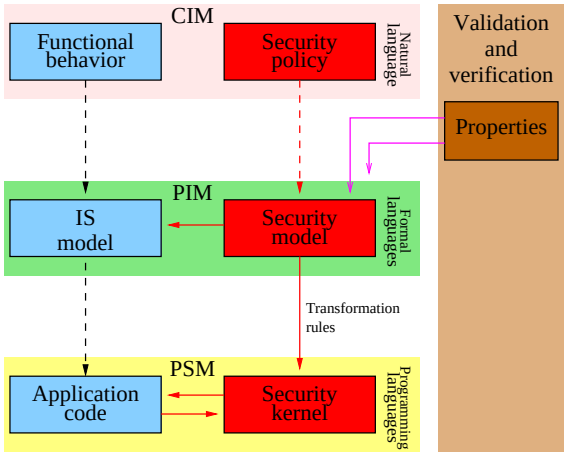

Figure 1: The proposed MDE approach

components of the system. **3)** The implementation of functional security policies should be completely automatic, to ensure reliability, maintainability and correctness with respect to regulations.

To implement this approach, we use the principles of *model-driven engineering* (MDE), which recommends to elaborate abstract models independently of any implementation techniques. It includes three steps as shown in Fig. 1. The first step focuses on the creation of a *computation independent model* (CIM) of the IS by considering various kinds of requirements (including security) formulated in a natural language. The second step consists in the development of a *platform independent model* (PIM) that allows the IS to be defined at an abstract level using appropriate domain-specific languages, preferably formal languages. The third step refines the latter in a concrete model according to a *service oriented architecture* (SOA) environment, which results in a *platform specific model* (PSM) that takes into account implementation issues. Generally, the security requirements of an IS are collected at the CIM level by means of a security policy, that consists of high-level security rules, specified more precisely at the PIM level and enforced at the PSM level by a security kernel. Figure 1 emphasizes the separation of security aspects and functional aspects in all the three steps of the MDA approach. First, the functional security policy is stated separately from functional behavior requirements. Second, the PIM consists of two abstract models derived from business rules and the security policy, respectively. Third, the security kernel, responsible for the enactment of the security policy, is independent of the application code, enabling modifications of security rules without having to change the application code. The proposed approach also includes activities for the validation or verification of the three main models (CIM, PIM and PSM). For instance, the security model can be checked with respect to some properties in order to guarantee that it does not restrict unduly the behavior of the IS.

There are several ongoing projects and initiatives related to security of Web-based systems. *ORKA* (ORKA, 2009) explores ways to bridge the gap between organizational control and access rights management. It includes the development and implementation of integrated security concepts founded on role-based security policies while considering organizational control principles. The *NGN-44* (ICTI, 2010) project is an initiative inside the *Carnegie Mellon|Portugal Program*. It consists in the development of new notions of semantically rich interface languages, and an associated programming language as well as logic-based verification techniques with the aim of enforcing security, integrity, and correctness requirements on distributed extensible Web-based applications. Finally, Meinel proposes a security model that enables the description of security policies as a set of abstract security intentions, which can be translated automatically into concrete security policies. An identity model, which makes possible the propagation of identity information to all services, and methods to describe trust between two unrelated parties in order to carry out sensitive transactions are also under investigation (Meinel, 2009). Since all these projects are in their infancy, it is difficult to appraise their real impact and compare the efficiency of the proposed tools.

The solutions introduced in this paper are in the spirit of recommendations of a re-engineering project initiated by one of our partners in Canada in the banking industry This firm is currently examining the functional security aspects of its ISs, particularly those interacting with brokers, customers and external financial systems to manage investment portfolio and trade financial products like stocks and options. These solutions will also be validated in a companion French project, called SELKIS, which targets medical ISs (SELKIS, 2009).

This paper focuses on the formalization of functional security rules by using the EB^{3}SEC method and automatic translation of EB^{3}SEC specifications into *algebraic state transition diagram* (ASTDs) in order to implement security policies efficiently and verify their consistency. Two techniques are investigated for the implementation of security policies. ASTDs are either refined into BPEL models executed by a BPEL engine or simulated by an interpreter with persistent objects, the BPEL engine and the interpreter being integrated into Web services of an SOA environment.

2 Specification of the Security Model with the EB^{3}SEC Method

An EB^{3}SEC specification includes a data model and a process expression for the definition of functional security rules at two granularity levels: atomic services (actions or transactions) and business processes (collections of related, structured atomic services). Permissions related to the execution of actions are described in the former. Security rules of the latter are defined by a process expression in the EB^{3}SEC language, which is inspired from process algebras. The process expression can refer to the data model to specify state-oriented constraints. The EB^{3}SEC method is an extension of EB^3 (Frappier and St-Denis, 2003), a method specifically devised to derive the input-output behavior of an IS (i.e., the IS model in Fig. 1). There is another granularity level suitable for the static permissions for users to access data attributes. The EB^{3}SEC method is not really concerned with this lowest granularity level. Nevertheless, it is taken into account during the refinement of the PIM into the PSM.

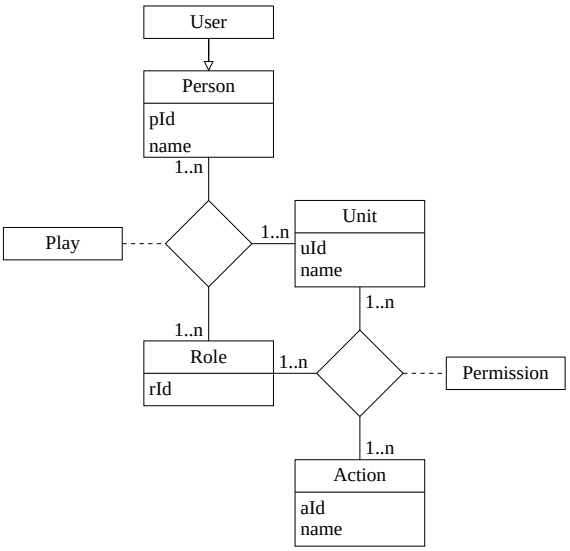

Figure 2: Class diagram of the security model

The example data model represented by the class diagram of Fig. 2 includes four main entity types and two associations. The entity types `Unit`, `Person`, `Role` and `Action` correspond respectively to the units of an organization (e.g., hospital wards or bank branches), the users or customers of the IS, the roles assigned to persons (e.g., patient, nurse, doctor) and the visible operations of the IS, more precisely the atomic operations that come from the IS's class diagram of an EB^3 model (e.g., *Register*, *Open_File*). The association `Play` allows to define the role of a person in a unit. Finally, the association `Permission` provides a means to define functional security rules at the granularity levels of atomic services.

The data model allows one to borrow concepts from role-based access control models, such as RBAC (Ferraiolo et al., 2003) and OrBAC (Kalam et al., 2003). In return, one is not limited to the framework of these models, and can include any entity deemed relevant for the security problem at hand. The concept of role is important to grant access rights to users according to their roles. Each user in a given role can then access the only part of the system (atomic services and data attributes) circumscribed by his access rights or privileges. These technologies have been extended to take into account temporal constraints on user-role assignments (X-GTRBAC (Joshi et al., 2005)) and Web services as protected resources (WS-RBAC (Bhatti et al., 2007)). They are, however, not adapted to long running processes, as those encountered in an SOA environment, and lack the expressive power to refer to past actions (which are elements of traces in process algebra notations). Nonetheless, the EB^{3}SEC language solves this problem. Indeed, ordering constraints on actions can be easily formulated while considering the associations `Permission` and `Play`. For example, the following security requirement include ordering constraints of actions:

> *A professor must approve a book prior to its acquisition and prior to its discard.*

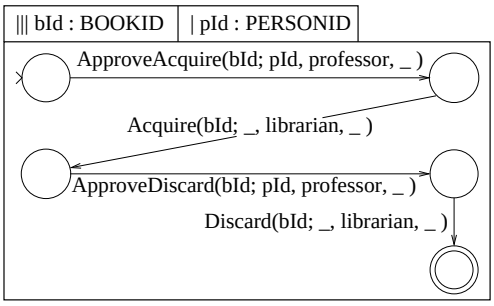

Figure 3: Specification of the security rule using the ASTD notation

These constraints on the acquisition and discard of books is formalized as follows in EB^{3}SEC:

$$
\begin{aligned}
&\mathit{Approval\_Rule}() = \\
&\quad |||\mathit{bId} : \mathit{BOOKID} : \ \ | \mathit{pId} : \mathit{PERSONID} : \\
&\qquad \langle \mathit{pId}, \mathit{professor}, \_ , \mathit{ApproveAcquire}(\mathit{bId}) \rangle . \\
&\qquad \langle \_ , \mathit{librarian}, \_ , \mathit{Acquire}(\mathit{bId}) \rangle . \\
&\qquad \langle \mathit{pId}, \mathit{professor}, \_ , \mathit{ApproveDiscard}(\mathit{bId}) \rangle . \\
&\qquad \langle \_ , \mathit{librarian}, \_ , \mathit{Discard}(\mathit{bId}) \rangle
\end{aligned}
$$

In this process expression, each action is embedded in a quadruple, which also includes the identification number of a person (`pId`), a role (`professor`, `librarian`), and a unit. The symbol "_", used as a wildcard, means that the corresponding field is not constrained by any value. The actions `Acquire` and `Discard` can only be executed by librarians (not necessarily the same), and actions `ApproveAcquire` and `ApproveDiscard` by a professor for a given book. The sequential composition operator "**.**" is used to specify the execution order of actions for a given book. The quantified interleaving ("|||") takes into account all entities of type `Book` while the quantified choice ("|") indicates the entity of type `Person` involved for every book.

The valid system input trace, which records the actions in their execution order, must match the traces defined by the security rules. The following process expression puts the security rules together:

$$main() = rule_1() \parallel rule_2() \parallel \cdots \parallel rule_n(),$$

where the operator "||" is the parallel composition (i.e., CSP's synchronization on shared actions).

3 Translation of EB^{3}SEC Specifications into ASTDs

EB^{3}SEC and EB^3 are trace-based formal languages. To make easier the verification of properties with model-checking techniques and interpretation of functional security policies at the PSM level, the state-based formalism of ASTDs (Frappier et al., 2008) has been adopted. This choice has no impact on the formalization of security policies. In fact an EB^{3}SEC specification can be easily translated into an ASTD because there is a one-to-one correspondence between syntactical elements of EB^{3}SEC and those of the ASTDs notation. Figure 3 shows the ASTD for the *Approval_Rule* introduced in the previous section. This representation, closely related to process algebras and statecharts, acts as a pivot language.

A translator and a prototyping version of an ASTD interpreter have been written in the OCaml programming language. The current state of the interpreted ASTD is stored in a relational database in order to ensure that the IS can recover after a crash. This interpreter benefits from EB3PAI (Fraikin and Frappier, 2002; Fraikin and Frappier, 2009), an interpreter of EB^3 processes. Another version of the ASTD interpreter written in Java is under development with the sole intention to integrate it more easily into an SOA service. OCaml and Java versions should be compared with respect to efficiency and access time criteria. Such a comparison may lead to better optimizations of the underlying algorithms and data structures.

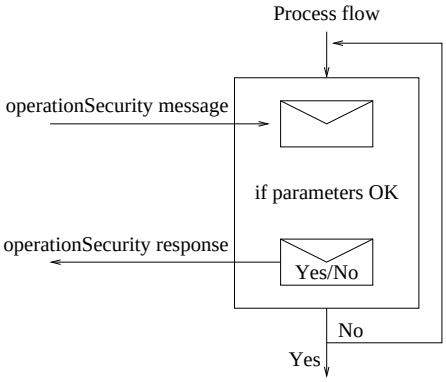

Figure 5: Communication protocol between a PDP and a security process

4 Implementation of the Security Model with BPEL

In the context of an IS along with security mechanisms, the security kernel is responsible for the enforcement of the security policy. Its primary components is a WS-BPEL engine and a *policy decision point* (PDP). The former is a back end server for the latter as illustrated in Fig. 4. The WS-BPEL engine runs *security processes* or *security workflows* when using BPEL as a specification language for security policies. The numbers in Fig. 4 indicate the order of messages when a consumer requests the service *A*, which in turn requests the service *B*.

A security process cannot be arbitrarily defined. It must comply with a BPEL model based on the semantics of ASTDs as well as with the BPEL standard and the interaction schema between security processes and the PDP (as illustrated in Fig. 5). It should also be possible to deploy each security rule as a service in order to promote its reuse.

Since a security process must be executed by a WS-BPEL engine, an ASTD specification must be transformed into one or more security processes. Such a transformation is founded on a behavioral and semantic equivalence rather than a simple syntactical mapping, even though a first evaluation has revealed similarities between some ASTD operators (e.g., sequential composition, (quantified) choice, (quantified) interleaving, synchronization) and BPEL constructs (e.g., `sequence`, `pick`, `flow`) or combinations of tags (e.g., `sequence` with `correlationSet` on *var* for a quantified choice). The resulting BPEL process needs to be efficient, particularly when dealing with quantified operators, like quantified interleaving

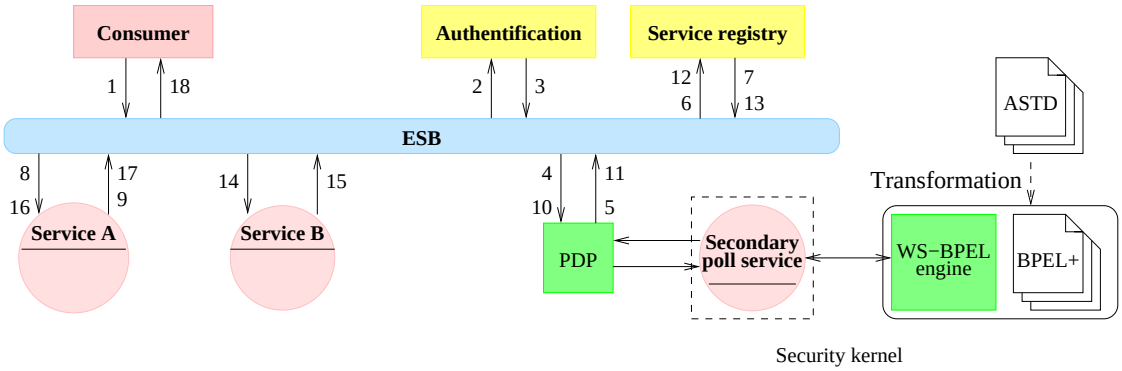

Figure 4: SOA environment for the enforcement of security policies using BPEL

over large sets of data. In addition, the data model is translated into an XSD and permissions related to the execution of actions are accessed via services.

There have been many attempts to transform specification written in process algebra like languages. Amstel and al. (van Amstel et al., 2008) developed a transformation from the process algebra ACP (Algebra of Communicating Processes) into UML state machines while preserving the semantics as much as possible. The execution context of ACP specifications is, however, the usual one, in the sense that events are received from the environment and then accepted or discarded by the interpreter based on the current state of the process. If the event is accepted, the process evolves into another state where it waits for specific events to happen. Chirichiello and Salaün designed Web services using a process algebra and encoded them into WS-BPEL processes (Chirichiello and Salaün, 2007). In the case of the combination ASTD/BPEL, the execution schema is quite different and must be considered when dealing with transformation. An ASTD specification does not model the IS itself but rather a policy, which is enforced during the execution of the IS.

Using the communication protocol depicted in Fig. 5, the following XML code shows a part of the BPEL security process obtained from the ASTD of Fig. 3:

```
<sequence>
  <repeatUntil name="repUtlAppAcq">
    <sequence name="seqAppAcq">
      <receive name="recAppAcqSec"
               createInstance="yes"
               operation="appAcqSec"
               variable="inAppAcq">
        <correlations> ... </correlations>
      </receive>
      <assign name="assappAcqSec">
        <copy><from>
'professor' = $inAppAcq.credential/ns0:role/ns0:roleName
               </from>
          <to variable="outAppAcq"
                part="canExecute"/>
        </copy>
      </assign>
      <reply name="repappAcqSec"
             operation="appAcqSec"
             variable="outAppAcq">
      </reply>
    </sequence>
    <condition>$outAppAcq.canExecute</condition>
  </repeatUntil>
  [...]
</sequence>
```

5 Verification of the PIM

A number of typical properties that a security policy should satisfy have been identified. For instance, if a user u can play a given role r in an organisation o according to association *Play* in the security class diagram, then process expression *main* (or the ASTD) should allow him to execute at least an action under this role. This is a reachability property which can be expressed in temporal logic (CTL) as follows:

$$\forall t \in Play : \exists \sigma, a : \\ \sigma = \langle t.u, t.r, t.o, a \rangle \wedge sp(\sigma) \wedge main \models EF(\sigma))$$

Another typical class of properties is permission feasibility: at least one person can execute a given action allowed for a role in an organisation.

$$\forall t \in Permission : \exists \sigma, u : \\ \sigma = \langle u, t.r, t.o, t.a \rangle \wedge sp(\sigma) \wedge main \models EF(\sigma)$$

We are currently experimenting the verification of these properties using model checking techniques.

6 Conclusion

We have presented an MDE solution to functional security management in service-oriented IS. Security policies can be comprehensively described in a single specification language called EB^{3}SEC, which integrates traditional RBAC-like access control policies, using its security data model, and stateful control rules using a process algebra or ASTDs. EB^{3}SEC can be automatically implemented using symbolic computation, thereby streamlining policy evolution. EB^{3}SEC is amenable to automated analysis using model-checking techniques. We are currently experimenting several model checkers to determine the most appropriate one. We are also investigating the translation of EB^{3}SEC policies into BPEL policies, to provide a more standard implementation.

EB^{3}SEC subsumes RBAC organizational control principles like separation of duties, delegation of rights and hierarchical structuring of concepts. These can all be express as elements of a security class diagram and their semantics defined using a first-order predicate. EB^{3}SEC goes beyond access control rules typically expressed in RBAC-like policies and XACML, by taking into account stateful rules, *i.e.* rules that can deal with the history of service requests to determine the authorization of the next requests. These stateful rules are abstractly described using a process algebra or ASTDs. In an RBAC or XACML approach, stateful business rules are separately described in a conventional programming language, hence are hard to modify and analyse. Approaches based on workflows can also be expressed in EB^{3}SEC. Our process algebra and ASTDs offer more powerful modeling mechanisms than BPEL, which streamlines the specification of workflow constraints. For instance, synchronisation, quantification, and access to state variables in guards can be used to model complex ordering constraints. Timing constraints can also be represented through the use of action timestamps, guards and time attributes in the security class diagram.

REFERENCES

Bhatti, R., Sanz, D., Bertino, E., and Ghafoor, A. (2007). A policy-based authorization framework for web services: Integrating xgtrbac and ws-policy. In *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pages 447 –454.

Chirichiello, A. and Salaün, G. (2007). Encoding process algebraic descriptions of web services into bpel. *Web Intelli. and Agent Sys.*, 5(4):419–434.

Ferraiolo, D., Kuhn, D., and Chandramouli, R. (2003). *Role-based access control*. Artech House Publishers.

Fraikin, B. and Frappier, M. (2002). EB3PAI: an Interpreter for the EB^3 Specification Language. In Haneberg, D., Schellhorn, G., and Reif, W., editors, *5th Workshop on Tools for System Design and Verification (FM-TOOLS 2002), proceedings*, Reisensburg Castle, Günzburg, Germany.

Fraikin, B. and Frappier, M. (2009). Efficient symbolic computation of process expressions. *Science of Computer Programming*, 74(9):723 – 753. Special Issue on the Fifth International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA'06).

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.

ICTI (2010). Carnegie mellon | portugal program. interfaces - certified interfaces for integrity and security in extensible web-based applications `http://www.cmuportugal.org/tiercontent.aspx?id=1564&ekmensel=568fab5c_68_0_1564_6`.

Joshi, J., Bertino, E., Latif, U., and Ghafoor, A. (2005). A generalized temporal role-based access control model. *Knowledge and Data Engineering, IEEE Transactions on*, 17(1):4 – 23.

Kalam, A. A. E., Benferhat, S., Miège, A., Baida, R. E., Cuppens, F., Saurel, C., Balbiani, P., Deswarte, Y., and Trouessin, G. (2003). Organization based access control. *Policies for Distributed Systems and Networks, IEEE International Workshop on*, 0:120.

Meinel, C. (2009). Soa — security. hasso-plattner-institut für softwaresystemtechnik. `http://www.hpi.uni-potsdam.de/meinel/research/security_engineering/soasecurity1.html`.

ORKA (2009). The orka consortium. germany. `http://www.organisatorische-kontrolle.de/index-en.htm`.

SELKIS (2009). Project anr-08-segi-018. france. `http://lacl.fr/selkis/`.

van Amstel, M. F., van den Brand, M. G. J., Proti, Z., and Verhoeff, T. (2008). Transforming process algebra models into UML state machines: Bridging a semantic gap? In *Theory and Practice of Model Transformations*, volume 5063 of *Lecture Notes in Computer Science*, pages 61–75. Springer Berlin / Heidelberg.