Integration of Security Policy into System Modeling

Nazim Benaïssa[2,4,5], Dominique Cansell[1,5], and Dominique Méry[2,3,5,⋆]

[1] Université de Metz
cansell@loria.fr
[2] Université Henri Poincaré Nancy 1
[3] mery@loria.fr
[4] benaissa@loria.fr
[5] LORIA
BP 239
54506 Vandœvre-lès-Nancy
France

Abstract. We address the proof-based development of (system) models satisfying a security policy. The security policy is expressed in a model called OrBAC, which allows one to state permissions and prohibitions on actions and activities and belongs to the family of role-based access control formalisms. The main question is to validate the link between the security policy expressed in OrBAC and the resulting system; a first abstract B model is derived from the OrBAC specification of the security policy and then the model is refined to introduce properties that can be expressed in OrBAC. The refinement guarantees that the resulting B (system) model satisfies the security policy. We present a generic development of a system with respect to a security policy and it can be instantiated later for a given security policy.

Keywords: refinement, integration, security policy.

1 Introduction

One of the most challenging problems in managing large networks is the complexity of security administration. Role-based access control has become the predominant model for advanced access control because it reduces the complexity and cost of security administration in large networked applications. Other models, like OrBAC [1], have been introduced by providing a structure based on the application domain and by introducing the concept of organisation. Networks or software systems can be abstracted by action systems or event B models; however, security requirements should be integrated into the proof-based design of such systems and we address the integration of security policy - expressed in a security model OrBAC - in the final systems. This leads us to deal with security properties like permissions and prohibitions. We leave obligations as

⋆ The research is supported by the DESIRS project of the ACI Sécurité et Informatique of the Ministry of Research. The first author has obtained a partial PhD grant from the Région Lorraine.

J. Julliand and O. Kouchnarenko (Eds.): B 2007, LNCS 4355, pp. 232–247, 2006.

out of the scope of the current work. J.-R. Abrial [2] contributes to the access control problem: the study consists of elaborating a system that controls access to a building for different persons. He does not refer to a security model but his work influences our current work.

1.1 Integration of Security Policies in System Development

When a system is under development, it is necessary to consider requirements documentation. The document is either written in a natural language, or in a semi-formal language, or in a formal language and it may include different aspects or views of the target system. Security policy is a possible part of this document and it may be expressed in a specific modelling language designed for expressing permissions, prohibitions, recommendations, obligations, ... related to the target system. Now, a key question is to ensure that the resulting system conforms to the security policy and it appears to us that in existing systems the link between the system and its security policy is not clearly established and formally validated, as shown in figure 1: the satisfaction relation should be established in a formal way. We illustrate the problem to be solved by considering two modelling languages:

- the OrBAC modelling language for security policy
- the event B modelling language for systems

Another important point is that we focus on the access control problem and as shown in figure 1.2, we describe several steps to obtain an implementation of the system from the statement of the security policy:

1. Generating a B model OM from the security policy O: the translation relation is explained in the current paper and can be mechanized.
2. Generating a B model RM by refining OM and by adding progressively details of the document which are not yet integrated into the current model: the refinement of B models is the key concept ensuring the validation of the satisfaction relation.
3. Writing a system model SYS from the last B model: the implementation of a refined B model into a system language can be directed by transformations over events.

1.2 Proof-Based Incremental Modelling

Proof-based development methods [3] integrate formal proof techniques in the development of software systems. The main idea is to start with a very abstract model of the system under development. Details are gradually added to this first model by building a sequence of more concrete ones. The relationship between two successive models in this sequence is that of *refinement* [3,8,4]. The essence of the refinement relationship is that it preserves already proved *system properties* including safety properties and termination. A development gives rise to a number of, so-called, *proof obligations*, which guarantee its correctness. Such proof obligations are discharged by the proof tool using automatic and interactive proof procedures supported by a proof engine [10].

The goal of the paper is to address the proof-based development of models satisfying a security policy. The security policy can be expressed in a formal language and it is possible to analyse the security policy, especially the consistency of the policy. The refinement ensures the correctness of the satisfaction relation: the system satisfies the security policy.

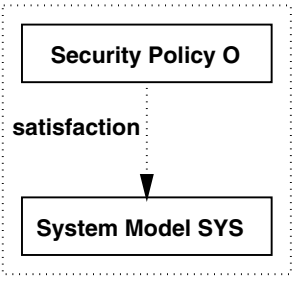
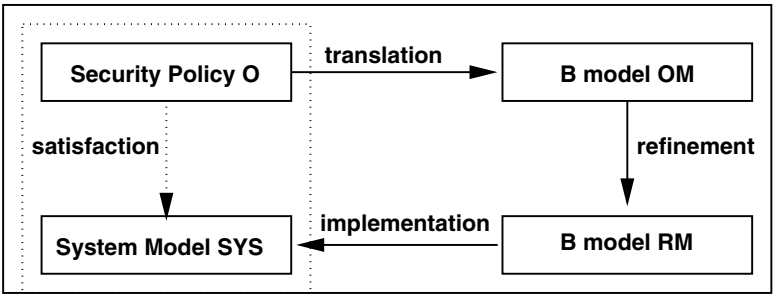


Fig. 1. The satisfaction relation



At the most abstract level it is obligatory to describe the static properties of a model's data by means of an "invariant" predicate. This gives rise to proof obligations relating to the consistency of the model. These are required to ensure that data properties which are claimed to be invariant are preserved by the events or operations of the model. Each refinement step is associated with a further invariant which relates the data of the more concrete model to that of the abstract model and states any additional invariant properties of the (possibly richer) concrete data model. These invariants, so-called *gluing invariants*, are used in the formulation of proof obligations related to the refinement.

The goal of a B development is to obtain a *proved model*. Since the development process leads to a large number of proof obligations, the mastering of proof complexity is a crucial issue. Even if a proof tool is available, its effective power is limited by classical results over logical theories and we must distribute the complexity of proofs over the components of the current development, e.g. by refinement. Refinement has the potential to decrease the complexity of the proof process whilst allowing for traceability of requirements.

B models rarely need to make assumptions about the *size* of a system being modelled, e.g. the number of nodes in a network. This is in contrast to model checking approaches [9]. The price to pay is to face possibly complex mathematical theories and difficult proofs. The re-use of developed models and the structuring mechanisms available in B help in decreasing the complexity. Where B has been exercised on known difficult problems, the result has often been a simpler proof development than has been achieved by users of other more monolithic techniques.

2 Models for Security Policy

The interaction of people with IT systems generate various security needs to guarantee that each system user benefits of its advantages without trespassing on another user's rights. These needs vary according to the activity field required. It could be regarding: Confidentiality (Non disclosure of sensitive information to non authorised persons), Integrity (Non alteration of sensitive information), Availability (Supply of information to users according to their rights of access these information), Auditability (The ability to trace and determine the actions carried out in the system).

Such requirements usually result in setting up an access control model that expresses security policies, defining for each user his permissions, prohibitions and obligations. Users (or subjects) are active entities operating on objects (passive entities) of the system.

Several access control models have been proposed: DAC [14], MAC [5,6], RBAC [12,15,13] or OrBAC [1]. In the Role-Based access control model, the RBAC model, security policy does not directly grant permissions to users but to roles [12]. A role is an abstraction for users. Each user is assigned to one or several roles, and will inherit permissions or prohibitions associated with these roles. Such a security model states security properties on the target system and on a hidden state of the current system. The hidden state clearly expresses dynamic properties related to permissions and prohibitions. The classical role-based models have no explicit state variable; the context information might be used to express the state changes but we think that a state-based approach like B provides a simpler framework for integrating security policy specification in the design of a system. Moreover, the refinement may help us in introducing security properties in a proof-based step.

2.1 Organization-Based Access Control Model: OrBAC

The OrBAC (Organization-Based Access Control model) for modelling the security policies is an extension of the RBAC model. OrBAC is based on the concept of organization. The specification of the security policy is completely parametrized by the organization such that it is possible to handle simultaneously several security policies associated with different organizations [1]. Another advantage of the OrBAC model compared to other models is that it makes it possible to express contextual permissions or prohibitions.

OrBAC takes again the concept of role such as it was defined in RBAC. Users are assigned to roles and inherit their privileges. The concept of *view* (or object's groups) is also introduced as an abstraction of the objects of the system. The construction of these groups of objects must be semantically well founded, this construction is related to the way in which the various roles carry out various actions on these objects. It should be noted that there are similarities with the concept of *view* in relational databases where it is a question of gathering objects which have similar properties. Just as for the objects, the actions are also gathered in activities, this implies that there are two levels of abstraction in OrBAC:

- Abstract level: roles (doctor, nurse), activities (management) and views (patient files, administration files) of the system on which various permissions and prohibitions are expressed.
- Concrete level: subjects (Paul, Peter, John), actions (create, delete) and objects (patient_file1, patient_file2) of the system.

Subjects, actions and objects are respectively assigned to roles, activities and views by relations defined over these entities(see figure 2). We detail relations in the next sub-section.

Empower, Use and Consider. *Assignment of subjects to roles*: subjects are assigned to one or more roles in order to define their privileges. Contrary to RBAC, subjects play their roles in organizations, which implies that subjects are assigned to roles through a ternary relation including the organization:

$empower(org, s, r)$: means that the subject s plays the role r in the organization org.

Assignment of actions to activities: As for roles and subjects, activities are an abstraction of various actions authorized in the system. The relation binding actions to activities is also a ternary relation including the organizations:

$consider(org, a, act)$: means that the action a is considered as an activity act in the organization org.

Assignment of objects to views: As in relational databases, a view in OrBAC corresponds to a set of objects having a common property. The relation binding the objects to the views to which they belong is also a ternary relation including the organization:

$use(org, o, v)$: means that the organization org uses the object o in the view v.

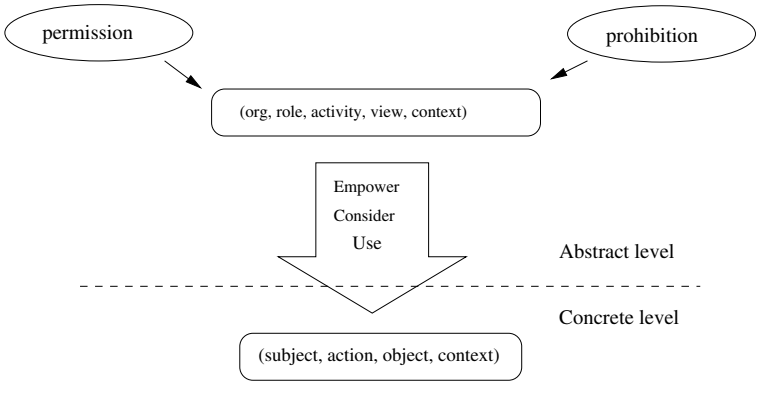


Fig. 2. Abstract and Concrete level of OrBAC

Modeling a Security Policy with OrBAC. When subjects, actions, and objects are respectively assigned to roles, activities and views, it is now possible to describe the security policy. It consists of defining different permissions and prohibitions:

- $permission(org, r, act, v, c)$: means that the organization org grants to the role r the permission to carry out the activity act on the view v in context c.
- $prohibition(org, r, act, v, c)$: means that the organization org prohibits the role r to carry out the activity act on the view v in the context c.

The concept of context, which did not exist in RBAC, is important in OrBAC, since it makes it possible to express contextual permissions (or prohibitions). Let us consider the example of a security policy in a medical environment. If one wants to restrict the access to patients records or files to their attending practitioner, the following permission should be added to the security policy:

$$permission(hospital, physician, consult, patient\_file, attending\_practitionar)$$

If there is no context: $permission(hospital, physician, consult, patient\_file)$. A physician could therefore access the file of any patient, which needs to be avoided. To be able to use this concept of context, a new relation $define$ should be introduced:

$Define(org, s, a, o, c)$: means that within organization org, the context c is true between subject s, the object o and action a.

Hierarchy in OrBAC. The OrBAC model makes it possible to define role hierarchies (as in RBAC) but also with respect to the organization hierarchies. The hierarchies allow the inheritance of the privileges (permissions or prohibitions), if for example $r2$ is a sub-role of $r1$, for an organization org, an activity av and a view v in the context ctx:

- When $permission(org, r1, av, v, ctx)$ holds then $permission(org, r2, av, v, ctx)$ holds.
- When $prohibition(org, r1, av, v, ctx)$ holds then $prohibition(org, r2, av, v, ctx)$ holds.

In the same way for the organizations, if $org2$ is a sub-organization of $org1$ then, for a role r an activity av and a view v in the context ctx:

- When $permission(org1, r, av, v, ctx)$ holds then $permission(org2, r, av, v, ctx)$ holds.
- When $prohibition(org1, r, av, v, ctx)$ holds then $prohibition(org2, r, av, v, ctx)$ holds.

The concept of inheritance is a key concept in OrBAC, since it allows gradual building of the security policy. Indeed, it is necessary to start by establishing a flow chart of the organizations (and roles) and defining the privileges on the basic organizations, it will then be enough to add gradually the privileges of the sub-organisations(sub-roles).

3 Event B Models from OrBAC

A complete introduction of B can be found in [7]. The question is to integrate the event B method and the OrBAC method; we have shortly introduced the event B concepts and the OrBAC concepts. In a B model, we should define the mathematical structures on which is based the development and the system under development; this information can be used to derive further properties that will be used in the validation of models. The B models have a static part and a dynamic part and in the specification of a security policy in OrBAC one has to state dynamic properties and to check the consistency of the resulting theory. The MOTOrBAC tool [11] provides a framework for defining a security policy and for checking the consistency of the set of facts and rules in a PROLOG-like style; this approach is clearly based on a fixed-point definition of permissions. The question of expressing administration model in OrBAC is also crucial and it is very simple to express the administration of security policy in B, since one can model the permissions as a variable satisfying the security policy expressed in an invariant. These points will be recalled when we present the effective translation of OrBAC models into event B models.

The current status of the work is as follows:

- We assume that we have an OrBAC description of the security policy.
- The security policy is supposed to be stable and consistent; the consistency is checked using tools like, for instance MOTOrBAC.
- The security policy states permissions and prohibitions.

The problem is to translate OrBAC statements into the event B modelling language. The translation of the security policy into event B includes several successive stages. A first B model is built and then other successive refinements are made as shown by figure 3. The first refinement validates the link between the abstract level (role, ...) and the concrete level (subject, ...).

The approach is based on refinement and each model or refinement model is enriched either by a constraint required by the OrBAC specification or by constraints like workflow constraints or separation of duties. Each constraint is attached to an invariant. The invariant becomes stronger through the refinement steps.

3.1 Abstract Model with Permissions and Prohibitions

As presented in the paragraph 2.1, the OrBAC specification has two levels of abstraction (see figure 2). The first step consists of an event B model modelling the abstract part of the security policy, i.e. initially, only concepts of organization, role, view, activity and context are considered. In the first model, permissions and prohibitions of the OrBAC model should be described.

- The clause SETS in the event B model contains basic sets such as organisations, roles, activities, views and contexts: ORGS, ROLES, ACTIVITIES, VIEWS, CONTEXTS.

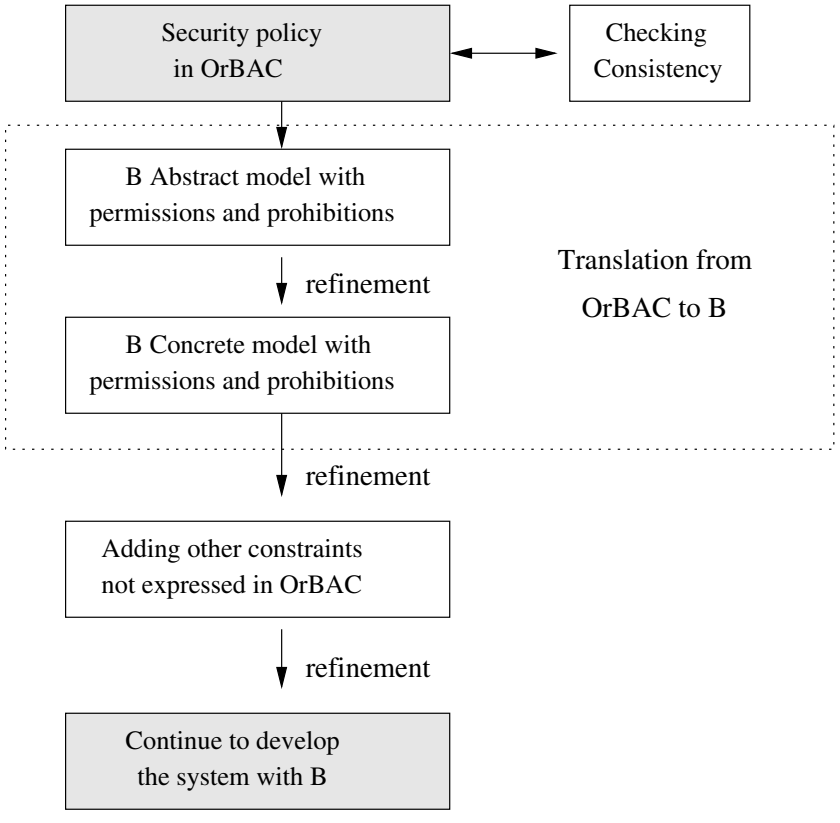


Fig. 3. Steps of the passage from OrBAC to a B event-based model

– The clauses CONSTANTS and PROPERTIES contain the constants like *permission* and *prohibition* that will contain privileges of the OrBAC description. One of the most important concepts contained in OrBAC is the concept of hierarchy: whether it is organization hierarchy or role hierarchy. Two new constants *sub_role* and *sub_org* are introduced to take into account respectively the role and organization hierarchy. It is enough to specify which roles and which organizations are concerned with inheritances, and the permissions and prohibitions corresponding to inheritances are deductively generated.

SETS
$ORGS$;
$ROLES$;
$ACTIVITIES$;
$VIEWS$;
$CONTEXTS$

CONSTANTS
$permission$,
$prohibition$,
sub_org,
sub_role,
$default$ /* $default\ context\ value$ */

PROPERTIES
$permission \subseteq ORGS \times ROLES \times ACTIVITIES \times VIEWS \times CONTEXTS$
$prohibition \subseteq ORGS \times ROLES \times ACTIVITIES \times VIEWS \times CONTEXTS$

$sub\_org \subseteq ORGS \times ORGS$
$sub\_role \subseteq ROLES \times ROLES$

$default \in CONTEXTS$

$/* \ Organization\ hierarchies \ */$
$\forall(org1, org2, r, av, v, ctx).($
$(org1 \in ORGS \wedge org2 \in ORGS \wedge$
$r \in ROLES \wedge av \in ACTIVITIES \wedge$
$v \in VIEWS \wedge ctx \in CONTEXTS \wedge$
$(org1 \mapsto org2) \in sub\_org \wedge$
$(org2 \mapsto r \mapsto av \mapsto v \mapsto ctx) \in permission)$
$\Rightarrow$
$(org1 \mapsto r \mapsto av \mapsto v \mapsto ctx) \in permission)$

$/* \ Role\ hierarchies \ */$
$\forall(org, r1, r2, av, v, ctx).($
$(r1 \in ROLES \wedge r2 \in ROLES \wedge$
$org \in ORGS \wedge av \in ACTIVITIES \wedge$
$v \in VIEWS \wedge ctx \in CONTEXTS \wedge$
$(r1 \mapsto r2) \in sub\_role \wedge$
$(org \mapsto r2 \mapsto av \mapsto v \mapsto ctx) \in permission)$
$\Rightarrow$
$(org \mapsto r1 \mapsto av \mapsto v \mapsto ctx) \in permission)$

$/* Same\ properties\ for\ prohibitions \ */$

For a given particular case, it is enough to initialize sets in the clause SETS by entities, organizations, roles, views, activities, contexts. Properties of constants, like *permission*, *prohibition*, *sub_role* and *sub_org*, should also be set in the clause PROPERTIES. Consequently, permissions and prohibitions can not be modified, since they are defined as constants; the OrBAC definitions are expressing properties satisfied by a consistent theory of permissions and prohibitions. We will address later the administration of OrBAC.

Introducing State Variables. An event B model expresses properties over state and state variables; the main problem is effectively that OrBAC has no explicit variables. In fact, OrBAC users are using some kind of state modifications but no explicit state exists in OrBAC, even if contexts might be used to model it. Variables are used to model the status of the system with respect to permissions and prohibitions:

– The clause VARIABLES contains two variables, the state variable *hist_abst* that contains the history of system activities; the variable *context* determines the running context of the system.

Variables satisfy the following properties added to the invariant:

INVARIANT
$context \in CONTEXTS$
$hist\_abst \subseteq ORGS \times ROLES \times ACTIVITIES \times VIEWS \times CONTEXTS$
$hist\_abst \subseteq permission$

The initial values of the two variables are set as follows:

$$context := default \parallel hist\_abst := \emptyset$$

As the security policy is supposed to be consistent, we should be able to prove in the clause ASSERTIONS :

ASSERTIONS
$pemission \cap prohibition = \emptyset$
$hist\_abst \cap prohibition = \emptyset$

– The clause EVENTS contains the following events :
  - The event *action* models when an authorization request for the access of a subject to an object of the system occurs.
  - The two events *set_default* and *set_context_value* are attached to the changes of the system context.

action $\widehat{=}$
any org, r, v, av **where**
$org \in ORGS$
$r \in ROLES$
$v \in VIEWS$
$av \in ACTIVITIES$
$(org \mapsto r \mapsto av \mapsto v \mapsto context) \in permission$
then
$hist\_abst := hist\_abst \cup \{(org \mapsto r \mapsto av \mapsto v \mapsto context)\}$
end

set_context_default $\widehat{=}$
begin
$context := default$
end

set_context_value $\widehat{=}$
begin
$context :\in CONTEXTS - \{default\}$
end

The invariant should be preserved and it means that any activity in the system is controlled by the security policy through the variable *hist_abst*.

3.2 First Refinement: Concrete Model with Permissions and Prohibitions

One of our goals is to use the refinement to validate the relation between security models; OrBAC defines two levels of abstraction and the current model is refined into a concrete model. The refinement introduces subjects, actions and objects:

sets $SUBJECTS$, $ACTIONS$ and $OBJECTS$ contain respectively subjects, actions and objects of the system under development. The clause CONSTANTS includes the following constants: *empower* (assignment of *subjects* to *roles*), *use* (assignment of objects to *views*) and *consider* (assignment of *actions* to *activities*). Properties of constants are stated as follows:

PROPERTIES
$empower \subseteq ORGS \times ROLES \times SUBJECTS$
$use \subseteq ORGS \times VIEWS \times OBJECTS$
$consider \subseteq ORGS \times ACTIVITIES \times ACTIONS$

Concrete Variables. A new variable *hist_conc* models the control of the system according to the security policy; it contains the history of the actions performed by a subject on a given object. The context in which the action occurred is also stored in this variable.

The relation between *hist_conc* and the variable *hist_abst* of the abstract model is expressed in the gluing invariant; the first part of the invariant states properties satisfied by variables with respect to permissions.

INVARIANT
$\forall(s, a, o, ctx).($
$(s \in SUBJECTS \wedge a \in ACTIONS \wedge$
$o \in OBJECTS \wedge ctx \in CONTEXTS \wedge$
$(s \mapsto a \mapsto o \mapsto ctx) \in hist\_conc)$
$\Rightarrow$
$(\exists(org, r, av, v).($
$org \in ORGS \wedge r \in ROLES \wedge$
$av \in ACTIVITIES \wedge v \in VIEWS \wedge$
$(r \mapsto s) \in empower \wedge$
$(v \mapsto o) \in use \wedge$
$(av \mapsto a) \in consider \wedge$
$(org \mapsto r \mapsto av \mapsto v \mapsto ctx) \in hist\_abst)))$

INVARIANT
$\forall(s, a, o, ctx).($
$(s \in SUBJECTS \wedge a \in ACTIONS \wedge$
$o \in OBJECTS \wedge ctx \in CONTEXTS \wedge$
$(s \mapsto a \mapsto o \mapsto ctx) \in hist\_conc)$
$\Rightarrow$
$(\forall(org, r, av, v).($
$org \in ORGS \wedge r \in ROLES \wedge$
$av \in ACTIVITIES \wedge v \in VIEWS \wedge$
$(r \mapsto s) \in empower \wedge$
$(v \mapsto o) \in use \wedge$
$(av \mapsto a) \in consider)$
$\Rightarrow$
$(org \mapsto r \mapsto av \mapsto v \mapsto ctx) \notin prohibition)))$

The invariant states that each action performed by the system satisfies the security policy. For the prohibitions, when a subject s wants to carry out an action a on an object o in an organization org, it is necessary to check that no prohibition exists for that action. The second part of the invariant states properties satisfied by variables with respect to prohibitions:

action $\widehat{=}$
  any s, a, o, org, r, v, av **where**
    $s \in SUBJECTS \wedge a \in ACTIONS \wedge o \in OBJECTS \wedge$
    $org \in ORGS \wedge r \in ROLES \wedge$
    $av \in ACTIVITIES \wedge v \in VIEWS \wedge$
    $(r \mapsto s) \in empower \wedge$
    $(v \mapsto o) \in use \wedge$
    $(av \mapsto a) \in consider \wedge$
  $/* \ permission \ */$
    $(org \mapsto r \mapsto av \mapsto v \mapsto ctx) \in permission \wedge$
  $/* \ prohibition \ */$
    $(\forall (orgi, ri, avi, vi).($
    $(orgi \in ORGS \wedge ri \in ROLES \wedge$
    $avi \in ACTIVITIES \wedge vi \in VIEWS \wedge$
    $(ri \mapsto s) \in empower \wedge$
    $(vi \mapsto o) \in use \wedge$
    $(avi \mapsto a) \in consider)$
    $\Rightarrow$
    $((orgi \mapsto ri \mapsto avi \mapsto vi \mapsto ctx) \notin prohibition))$
  then
    $hist\_conc := hist\_conc \cup \{(s \mapsto a \mapsto o \mapsto context)\}$
  end

The Events. The abstract model should consider the permissions and the prohibitions for a subject s that asks to perform an action a on an object o.

Discussion on Contextual Security Policies. In the different cases we studied, it appeared that the context notion has two different aspects. The first aspect concerns the contexts that are global to the system. An example of a global context is a system managing accesses to a building in a company. We may have a permission (or a prohibition):

$$permission(company, agent, access, building, opening\_hours)$$

In this permission, the context *opening_hours* is global to the system, i.e. the whole system is, at a given moment, in the context $default$ or *opening_hour*. A state variable *context* indicating the running context of the system is used in this case. On the other hand, in the case, for example, of a system managing the access to the patient files in a hospital, we may have permissions of the form:

$$permission(hospital, physician, consult, patient\_file, attending\_practitioner)$$

In this permission, the context *attending_practitioner* (that means that the permission is valid only if the physician is the attending practitioner of the patient) is not global to the system but links subjects to the objects. In this case a new constant $define$ (as in OrBAC) is used. This constant defines links between objects and subjects with respect to some actions, and has the following form :

$$define \subseteq ORGS \times SUBJECTS \times ACTIONS \times OBJECTS \times CONTEXTS$$

In order to give the system designer the possibility of expressing contextual permissions of each type, modifications must be made to the B model. If $context\_value$ is a value of a global context, the invariant should be modified as follows:

INVARIANT
$\forall(s, a, o, ctx).($
$(s \in SUBJECTS \wedge a \in ACTIONS \wedge$
$o \in OBJECTS \wedge ctx \in CONTEXTS \wedge$
$(s \mapsto a \mapsto o \mapsto ctx) \in hist\_conc)$
$\Rightarrow$
$(\exists(org, r, av, v).($
$org \in ORGS \wedge r \in ROLES \wedge$
$av \in ACTIVITIES \wedge v \in VIEWS \wedge$
$(r \mapsto s) \in empower \wedge$
$(v \mapsto o) \in use \wedge$
$(av \mapsto a) \in consider \wedge$
$(((org \mapsto s \mapsto a \mapsto o \mapsto ctx) \in define) \vee (ctx = context\_value)) \wedge$
$(org \mapsto r \mapsto av \mapsto v \mapsto ctx) \in hist\_abst)))$

3.3 Second Refinement: Adding Other Constraints Not Expressed in OrBAC

The state variables of the event B model give additional information on the system which was not available with OrBAC. It was impossible to know at a given moment the state of the system and, for example, which member of a company staff consulted or modified which file. This point is important since in practice the security policies are increasingly complex and new types of constraints appear. The passage towards B allows us to implement the security policy such as it was established in OrBAC, and enrich it with the possibility of introducing new constraints such as workflow constraints or the duty separation.

Workflow Constraints. Workflow constraints express properties on the task scheduling in a system. For instance, a rule for a given workflow states that an action *act* should be executed, only if a set of actions *act*1, *act*2..., *actn* have already executed. Those constraints can not be expressed in OrBAC, because, when a subject is assigned to a given role, it obtains its complete privileges. A permission is systematically delivered to execute the action *act*, if one of the roles to which a subject is assigned has the appropriate privilege, even if one of the actions *act*1, *act*2,..., *actn* has not yet been executed. The implementation of these constraints in a B model leads to the following invariant:

$$
\begin{array}{l}
\text{INVARIANT} \\
\forall(s, o, ctx).(\\
\quad (s \in SUBJECTS \wedge \\
\quad o \in OBJECTS \wedge \\
\quad ctx \in CONTEXTS \wedge \\
\quad (s \mapsto act \mapsto o \mapsto ctx) \in hist\_conc) \\
\Rightarrow \\
\quad (\exists(sw, cw).(sw \in SUBJECTS \wedge cw \in CONTEXTS \wedge \\
\qquad (sw \mapsto act1 \mapsto o \mapsto cw) \in hist\_conc) \wedge \\
\quad \exists(sw, cw).(sw \in SUBJECTS \wedge cw \in CONTEXTS \wedge \\
\qquad (sw \mapsto act2 \mapsto o \mapsto cw) \in hist\_conc) \wedge \ldots \\
\quad \exists(sw, cw).(sw \in SUBJECTS \wedge cw \in CONTEXTS \wedge \\
\qquad (sw \mapsto actn \mapsto o \mapsto cw) \in hist\_conc)))
\end{array}
$$

The refinement provides a way to add such a constraint to the model and proof obligations ensure the correctness of the transformation. Another refinement can be done to introduce specific rules for aspects such as duty separation.

3.4 Separation of Duties

Separation of duties aims to prevent fraud and errors by disseminating an action's execution privileges among different subjects. To implement a system satisfying this type of constraints, it is necessary that when a subject asks for the authorization to execute an action on an object, to be able to check if it did not already act throughout the process, which is impossible to do with OrBAC in a simple way. However, there is a form of separation of duties known as static separation of duties (implemented with RBAC [12]). This one consists of preventing a subject from accumulating several important functions, and it can be achieved when subjects are assigned to roles. In the B model, the following assertion should be proved to guarantee that no subject accumulates two critical given roles $r1$, $r2$. In the clause ASSERTIONS:

$$
\begin{array}{l}
\forall s.((s \in SUBJECTS \wedge (org \mapsto r1 \mapsto s) \in empower) \\
\qquad \Rightarrow (org \mapsto r2 \mapsto s) \notin empower)
\end{array}
$$

Proceeding this way may be too rigid in some cases. A subject s can cumulate several functions if it does not intervene many times in the management of the same object o. To prevent a subject s executing two critical actions $act1$, $act2$ on an object o with $act1 \neq act2$, the following invariant has to be proved:

$$
\begin{array}{l}
\text{INVARIANT} \\
\forall(s1, s2, o, ctx1, ctx2).(\\
\quad (s1 \in SUBJECTS \wedge s2 \in SUBJECTS \wedge \\
\quad o \in OBJECTS \wedge \\
\quad ctx1 \in CONTEXTS \wedge ctx2 \in CONTEXTS \wedge \\
\quad (s1 \mapsto act1 \mapsto o \mapsto ctx1) \in hist\_conc) \wedge \\
\quad (s2 \mapsto act2 \mapsto o \mapsto ctx2) \in hist\_conc) \\
\Rightarrow \\
\quad (s1 \neq s2))
\end{array}
$$

The separation of duties and workflow constraints are only particular cases of constraints where instant system state must be known in order for them to be expressible.

4 Conclusion and Open Issues

The development of software systems satisfying a given security policy should be based on techniques for validating the link between the security policy and the resulting system. The link between the security policy and the system is called *satisfaction* and we have used the event B method, especially refinement, for relating the security policy expressed in OrBAC and the final system. The link between the two levels of abstractions in OrBAC is proved to be a B refinement. Our work is greatly influenced by the case study developed by J.-R. Abrial [2]; he shows how a system for controling access to buildings can be derived by refinement, and he starts by expressing the essence of the access control. In our case, we use an elaborate formalism OrBAC for expressing the security policy and for checking its consistency; we derive a mathematical theory from OrBAC specification and we define an explicit state of a system which is not explicit in OrBAC. The refinement provides us with a way to develop a list of models which progressively integrate details that do not seem to be possible to express in OrBAC: workflow constraints, for instance. Our models are generic with respect to the security policy and can be reused to develop a real system. A crucial question would be to use our models for developing an infrastructure for controlling an existing system with respect to a security policy. Moreover, security policy expresses permissions and prohibitions but it remains to consider obligations which are very difficult to refine because they are close to liveness properties and should be expressed on traces. Moreover, the administration of security policy (ADOrBAC) leads to modifiable permissions and prohibitions: our constants should be transformed into state variables and decisions should be taken to handle situations, which are not satisfying the invariant (some person might become undesirable whilst in a building, when security policy is modified by the administration). Finally, case studies should be developed using these models.

References

1. A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access Control. In *4th IEEE International Workshop on Policies for Distributed Systems and Networks (Policy'03)*, June 2003.
2. J.-R. Abrial. Etude système: méthode et exemple. http://www.atelierb.societe.com/documents.html.
3. J.R. Abrial. *The B Book - Assigning Programs to Meanings.* Cambridge University Press, 1996. ISBN 0-521-49619-5.
4. R.-J. Back and J. von Wright. *Refinement Calculus.* Springer-Verlag, 1998.
5. D. E. Bell and L. J. LaPadula. Secure computer systems: Unified exposition and multics interpretation. MTR-2997, (ESD-TR-75-306), available as NTIS AD-A023 588, MITRE Corporation, 1976.
6. K. Biba. Integrity consideration for secure computer systems. Technical Report MTR-3153, MITRE Corporation, 1975.

7. Dominique Cansell and Dominique Méry. Logical foundations of the B method. *Computers and Informatics*, 22, 2003.
8. K. M. Chandy and J. Misra. *Parallel Program Design A Foundation*. Addison-Wesley Publishing Company, 1988. ISBN 0-201-05866-9.
9. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 2000.
10. ClearSy. *Web site B4free set of tools for development of B models*, 2004.
11. F. Cuppens. Orbac web page. http://www.orbac.org.
12. D.F. Ferraiolo, R.Sandhu, S.Gavrila, D.R. Kuhn, and R.Chandramouli. Proposed nist standard for role-based access control. *ACM Transactions on Information and System Security*, 4(3):222–274, 2001.
13. Serban I. Gavrila and John F. Barkley. Formal specification for role based access control user/role and role/role relationship management. In *ACM Workshop on Role-Based Access Control*, pages 81–90, 1998.
14. Butler Lampson. Protection. In *Proceedings of the 5th Annual Princeton Conference on Information Sciences and Systems*, pages 437–443, Princeton University, 1971.
15. R.Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.