A Model-checking Approach to Analysing Organisational Controls in a Loan Origination Process

Andreas Schaad, Volkmar Lotz
SAP Labs France, Security & Trust Group
805, Avenue du Dr Maurice Donat
06250 Mougins, FRANCE
{andreas.Schaad, volkmar.Lotz}@sap.com

Karsten Sohr
Universität Bremen, Technologie-Zentrum Informatik
Bibliothekstraße 1
28359 Bremen, Germany
sohr@tzi.de

ABSTRACT

Demonstrating the safety of a system (ie. avoiding the undesired propagation of access rights or indirect access through some other granted resource) is one of the goals of access control research, e.g. [1-4]. However, the flexibility required from enterprise resource management (ERP) systems may require the implementation of seemingly contradictory requirements (e.g. tight access control but at the same time support for discretionary delegation of workflow tasks and rights).

To aid in the analysis of safety problems in workflow-based ERP system, this paper presents a model-checking based approach for automated analysis of delegation and revocation functionalities. This is done in the context of a real-world banking workflow requiring static and dynamic separation of duty properties.

We derived information about the workflow from BPEL specifications and ERP business object repositories. This was captured in a SMV specification together with a definition of possible delegation and revocation scenarios. The required separation properties were translated into a set of LTL-based constraints. In particular, we analyse the interaction between delegation and revocation activities in the context of dynamic separation of duty policies.

Categories & Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection – *access controls*

General Terms

Security

Keywords

Model-checking, organisational control, separation, delegation, revocation

1 INTRODUCTION

Within some of our earlier research we focused on modelling and achieving “organisational control” [5] by integrating new and already existing work on workflow based systems [6], the required access rights [7, 8], the definition of separation of duty policies [9, 10] and the delegation and revocation of access right/authorisations and tasks/obligations [11, 12]. This led to the partial implementation of such concepts in the SAP Research workflow stack [13]. In particular, we implemented a security enforcement point for a workflow tasklist manager, automated support for delegation and revocation schemes [14] and specification and enforcement of separation of duty policies using the JESS and iLog rule systems.

This further confirmed our already obtained insights into the possibly existing unwanted relationships between such components.

In particular, we had already observed at a formal level [5] that delegation and revocation features may be used to “circumvent” separation of duty properties, thus providing potentially undesired access to resources. However, “Enterprise Resource Management” means providing people with the ability to perform their work according to economic principles. It is thus a partially contradictory aim to build systems that provide flexibility (e.g. delegating tasks and possibly required access rights) at the same time aiming to strictly preserve safety. We believe that only a mix of a well-designed access control system and a set of (compensating) controls at configuration, deploy and run-time can allow us to achieve an acceptable level of organisational control and flexibility. Analysis tools at the various stages and system levels are required to assist us.

Accordingly, this paper presents a model-checking based approach for automated analysis of delegation and revocation functionalities in the context of a workflow requiring static and dynamic separation of duty properties. We derived information about the workflow from BPEL specifications and business object repositories. This was captured in a SMV specification together with a definition of possible delegation and revocation scenarios. The required separation properties were translated into a set of LTL-based constraints. The results appear to be promising enough to further continue the automated translation of workflow and other context-relevant policies and information for a model-checking based analysis.

The rest of the paper will provide some more required background information on separation of duties as well as delegation and revocation of tasks and rights (Section 2). We then instantiate such properties within the context of a real-world loan origination process and informally discuss constraints that need to be maintained (Section 3). After a brief summary of the current state of the art in the area of model-checking (Section 4) we then specify the banking workflow in SMV together with a defined subset of the constraints in LTL (Section 5). We then discuss some results of our analysis (Section 6) and provide some conclusions and future research directions (Sections 7 and 8).

2 BACKGROUND AND RELATED WORK

2.1 SoD - General introduction and overview

Separation controls are probably the so far best understood type of application-level constraint, as indicated by the variety of existing work. Specifically research in the areas of role-based access control, e.g. [15] and distributed systems management, e.g. [16] has led to the definition of taxonomies and frameworks, that will be reviewed in the course of this section. Although the origins of this principle cannot be clearly identified, it is obvious that the development of organisational theory, e.g. [17, 18], and internal control and accountancy frameworks helped in their definition and possible ways of implementation. Application areas are the prevention of fraud due to the misuse of powers and the preservation of integrity.

One classic example when talking about separation controls is that of preventing fraud committed by the purchasing officer in a company. If he could perform all the necessary steps of creating and authorising an order, recording the arrival of the item, recording the arrival of the invoice and finally authorising the payment, it would be easy for him to place an order with a fictitious company he owns, record a non-existing arrival, pay to the company, and add the non-existing goods to the books. Only the end-of-the-year inventory would reveal the discrepancy between the books and the physical stock. Enforcing a separation control in this context may be to not let a principal have all the necessary authorisations for each required step in this process. A more relaxed variation may be to not allow him to perform all the steps on his own. This is sometimes referred to as a dual control since two or more people are needed for the execution of a critical process.

Often, the term "separation of duties" is used in the context of examples like the above. Although commonly used, we believe that it does in many cases not really reflect its actual interpretation. This is because in its most general definition the separation of duties may be best described as a means of preventing (in)advertent error and fraud through the general or context-dependent limitation of a principal's authority. Thus, "separation of authority" may be a more precise term. It has already been realised by others that there is still a gap in the support of current workflow / ERP systems for supporting separation of duty properties [19].

We now provide a more specific review over the development of separation controls in information systems. This will reveal that there are various ways of specifying separation controls (e.g. exclusive roles or rights or tasks) at different conceptual levels (e.g. application, middleware, database level). The actual choice will be clearly dependent on the organisational context.

Separation of duties (initially "Separation of privilege") as a design principle for the protection of information in computer systems, was first referred to by Saltzer and Schroeder [20]. Clark and Wilson [21] then introduced the separation of duties as a mechanism to control fraud and error and ensure the consistency of the data objects. They later describe two distinct types of separation of duty called the static and dynamic separation of duty.

Transaction control expressions are a notation for the description and implementation of static and dynamic separation of duty controls [22]. An information object is associated with the transaction control expression. The execution of operations on the object causes each transaction control expression to be converted into history. A partial history for transient objects (which have a short lifetime, such as a cheque) and complete history for persistent objects (with a long lifetime such as an account) is maintained. One possible separation of duty is enforced by the rule that for transient objects different transactions must be executed by distinct users. This approach is later extended in [23], discussing more refined dynamic separation controls, the presence of role hierarchies and the possibility of substituting the attribution for a principal and a task. Later criticism by Nash and Poland [24] addresses the fact that Sandhu did initially not consider the possibility of overlapping assignments of roles to transactions. They further discuss the object-based separation of duty where every transaction against an object must be performed by a different user. Using Sandhu's transaction control expressions they show how to maintain a history of object access. Parallel to this, named protection domains (NPD) are presented as a different approach to expressing separation of duty controls [25]. Instead of grouping individuals, privileges are grouped in a NPD. Such privileges might be all the operations needed to run the accounts-receivable portion of a general-ledger application. Only one NPD can be activated at a time, avoiding conflicting privileges or their misuse and making it possible to express separation of duties. It is further suggested to group individuals on the basis of the tasks they perform and not according to organisational hierarchies, an approach which may be seen as a precursor to current role-based approaches.

It was soon realised that organisational roles could be used as a suitable concept for expressing more elaborate and fine-grained separation controls. Thus, it is not surprising that work on the separation of duties received particular attention with the progress made in role-based access control [10]. One main reason for this is the established concept of mutually exclusive roles as initially discussed. These are any roles that for some organisational reason have been declared to be exclusive, which may then have an effect on their relation to principals and assignment with policies.

In [9] the separation of duties is defined as a multi-person control policy requiring that two or more different people are responsible for the completion of a task. This is done by spreading authority and responsibility for an action or task over multiple people. However, choosing a role concept similar to RBAC96 as the basis for the definition of a set of separation of duty controls, they effectively consider authority only. The underlying mechanism they use for implementing separation controls is the mutual exclusion of roles. Constraining the role membership, role activation and role use, leads to two main dimensions along which separation controls can be specified. These are referred to as static separation of duties (strong exclusion) and dynamic separation of duties (weak exclusion), where a further distinction is made between several kinds of dynamic separation. Two of these dynamic controls are of importance in our later banking example, namely the object-based and operational separation. These identified separation controls are listed informally in Figure 1.

These observed controls are further formalised in [26]. Here, separation controls or policies are seen in the form of conjunctions of constituent properties of system states and state transitions. A distinction is made between three types of general policy properties called access-attribute (e.g. user-group membership invariants); access-authorisation (e.g. evaluation of access queries); and access-management (e.g. granting of access right).

1. Static Separation of Duties
   - (Simple) Static Separation of Duties (SSSoD)

   A principal may not be a member of any two exclusive roles.
2. Dynamic Separation of Duties
   - (Simple) Dynamic Separation of Duties (SDSoD)

   A principal may be a member of any two exclusive roles but must not activate them at the same time.
   - Object-based Separation of Duties (ObjSoD)

   A principal may be a member of any two exclusive roles and may also activate them at the same time, but he must not act upon the same object through both.
   - Operational Separation of Duties (OpSoD)

   A principal may be a member of some exclusive roles as long as the set of authorisations acquired over these roles does not cover an entire workflow.
   - History-based Separation of Duties (HistSoD)

   A principal may be a member of some exclusive roles and the complete set of authorisations acquired over these roles may cover an entire workflow, but a principal must not use all authorisations on the same object(s).

Figure 1: Separation Taxonomy (I)

These properties show dependencies, and any omission or incorrect modification of dependent properties may result in ineffective separation policies. Based on these observations eleven types of separation of duty controls are identified. These are formalised and extended variations of those listed in [9] and earlier work reported in [21, 24, 27]. It is further shown how these relate to each other and may be composed if, for example, static or dynamic separation of duties is enforced. Parallel to this work, a more formal framework for implementing separation of duty controls is presented in [28]. As in [9] the chosen underlying mechanism is the mutual exclusion of roles. However, the discussion is taken further by analysing relationships between different types of separation controls; properties of mutual exclusion of roles; and constraints on possible role hierarchies forbidding the existence of a 'root' role containing all other system roles, which may be compared with the activation hierarchies described in [15] and the role graph approach outlined in [29].

In earlier work we have provided a more detailed, unifying specification of existing separation taxonomies approaches, showing possible composition dependencies using a constraint satisfaction approach [5].

2.2 Contextual information required to define and maintain SoD properties

Having clarified the general background and various conceptual interpretations of separation of duty properties, these can be expressed and enforced at different technical layers. What is commonly required as contextual information is the following:

1. A notion of roles and maintenance of exclusive relationships.
2. A notion of authorisations and maintenance of exclusive relationships.
3. A reference to a business object (its type as well unique identification at runtime).
4. A notion of a workflow and execution engine which may provide the following data:
   - Information about a workflow model
   - Information about the execution path within an instance of the model
   - Information about the time of manipulation of an object
   - Information about future possible paths that can be taken (To allow for a predictive analysis and, for example, switching to an extended audit when a suspicious but not critical state is reached)
5. A monitor keeping track of delegated and revoked tasks.

2.3 Delegation of tasks and rights

Delegation may be used as a term for describing how duties and the required authority propagate through an organisation, usually in terms of the refinement of a high-level organisational goal into manageable policies which eventually lead to the execution of some task [30, 31]. This is often referred to as decentralisation or Management by Delegation [18] where delegation considers the passing of policy objects from one principal to another with respect to the performance of some activity and attainment of some common organisational goal.

However, often the term delegation is also used to describe how a principal passes some specific object on to some other principal, because the current structure does not allow the achievement of a goal one or both of these principals have [17]. If such delegation activities occur frequently, have a regular pattern or principals delegate some object indefinitely, then this indicates that the current organisational structure and procedures do not reflect the goals of the involved principals. An initially temporary and ad-hoc delegation must now become part of the regular administrative delegation activities shaping the formal organisational structure. There may be different factors motivating such general administrative delegation or ad-hoc delegation between specific principals. We thus distinguish between two types of delegation that need to be clarified: Administrative delegation (administration) and ad-hoc delegation (delegation).This distinction is often not made clear, e.g. [32]. Both cause some sort of policy object assignment to be changed, where administration has a high degree of similarity, regularity and repeatability, and conversely ad-hoc delegation has a low degree of these. We argue that delegation may be seen as distinct from administration. Three characteristics can be used to support this distinction. These are the representation of the authority to delegate; the specific relation of a principal to an object; and the duration of this relation.

In [11] we have provided formal models for the delegation of tasks (obligations) and the required rights (authorisations), based on the conceptual models provided in [16]. We introduced the concepts of review and supervision as obligations on delegated general and specific obligations (tasks at the workflow model level and specific task instances). The formalisation in a predicate logic also showed that the delegation of authorisations, as well as general and specific tasks can be based on one general delegation function. This function will also maintain a history of delegation and object access activities over a sequence of states, recording properties such as multiple delegations of an authorisation to the same principal by different delegating principals or the dropping a delegated task/obligation by a delegating principal.

We noted that an explicit distinction between delegating tasks types and their instances needs to be made. For example, a task instance may only be delegated to some principal in a role associated with the corresponding task type. Maintaining and modelling such information is essential for providing revocation functionality as we will later show in our LTL specification.

2.4 Revocation of tasks and rights

In general, revocation of an object is based on its previous delegation and thus requires the following pieces of information [1]: The principals involved in previous delegation(s); the time of previous delegation(s); the object subject to previous delegation(s).

Our SMV specification provides this information and may thus support the various forms of revocation as described in the revocation framework of [33]. In this framework different revocation schemes for delegated access rights are classified against the dimensions of resilience, propagation and dominance. Since resilience is based on negative permissions, we do not consider this here, as there is no corresponding concept for the policy objects in our model. The remaining two dimensions may be informally summarised as follows:

1. Propagation distinguishes whether the decision to revoke affects
   - only the principal directly subject to a revocation (local); or
   - also those principals the principal subject to the revocation may have further delegated the object to be revoked to (global).
2. Dominance addresses conflicts that may arise when a principal subject to a revocation
   - has also been delegated the same object from other principals. If such other
   - delegations are independent of the revoker then this is outside the scope of revocation.

If, however, such other delegations have been performed by principals who, at some earlier stage, received the object to be revoked via a delegation path stemming from the revoker, then the revoking principal may only revoke with respect to his delegation (weak) or revoke all such other delegations that stem from him (strong).

Based on these two dimensions, we work on the basis of 4 different revocation schemes which, due to the absence of the resilience property, are a subset of those described by [33], summarised in Table 1. A full formal treatment of revoking delegated tasks and rights is part of [12] and we will now investigate how far these schemes can be expressed and integrated with respect to our banking workflow.

No	Propagation	Dominance	Name
1	No	No	Weak local revocation
2	No	Yes	Strong local revocation
3	Yes	No	Weak global revocation
4	Yes	Yes	Strong global revocation

Table 1: Revocation Taxonomy

Role	Service	Access Right	Workflow Step	Business Object
Clerk Preprocessor	Customer Information File	query () update ()	Input Customer Data	Customer Data
Clerk Preprocessor	Customer Information File	query ()	Customer Identificati on	Customer Data
Clerk Postprocessor	Credit Bureau	prepare () release <100k post ()	Check Credit Worthiness	Rating Report
Supervisor	Credit Bureau	release >100k	Check Credit Worthiness	Rating Report
Clerk Postprocessor	Internal Rating	query ()	Check rating	Rating Report
Supervisor	Internal Rating	update ()	Bank signs form	Rating Report
Clerk Postprocessor	Product Database	query available products ()	Choose Bundled Product	Product Bundle
Clerk Postprocessor	Pricing Engine	modify () commit <100k	Price Bundled Product	Product Bundle
Supervisor	Pricing Engine	commit >100k	Price Bundled Product	Product Bundle
Clerk Postprocessor	Output Manageme nt System	post print request ()	Print Opening Form	Contract
Customer	-	sign ()	Customer signs form	Contract
Manager	-	sign () update ()	Bank signs form	Contract
Clerk Postprocessor	Account Manageme nt System	open ()	Open Account	Account

Table 2: Assignments of rights, roles and tasks

3 SOD AND DELEGATION IN BANKING WORKFLOWS

Figure 2 shows a typical loan origination process in the banking domain, similar to that described in [34]. The supporting Table 2 summarises some of the required roles, the general service, the required access rights and associated workflows steps and business objects as later modelled in SMV.

The loan origination process describes a customer wanting to buy a bundled product. If he is not an existing customer, his master data and other identification-relevant data need to be entered into the system. Several external and internal ratings then need to be obtained by the processing clerk in order to check the credit worthiness of the client (e.g. based on sums of liabilities, sums of assets, reasons for rating etc.). The system will then propose a preconfigured bundled product to the clerk and customer (e.g. original price, customer segment special conditions, customer company special conditions, asset limit for price etc.). The customer and Bank finally come to an agreement expressed in the signature of the client and Bank representative.

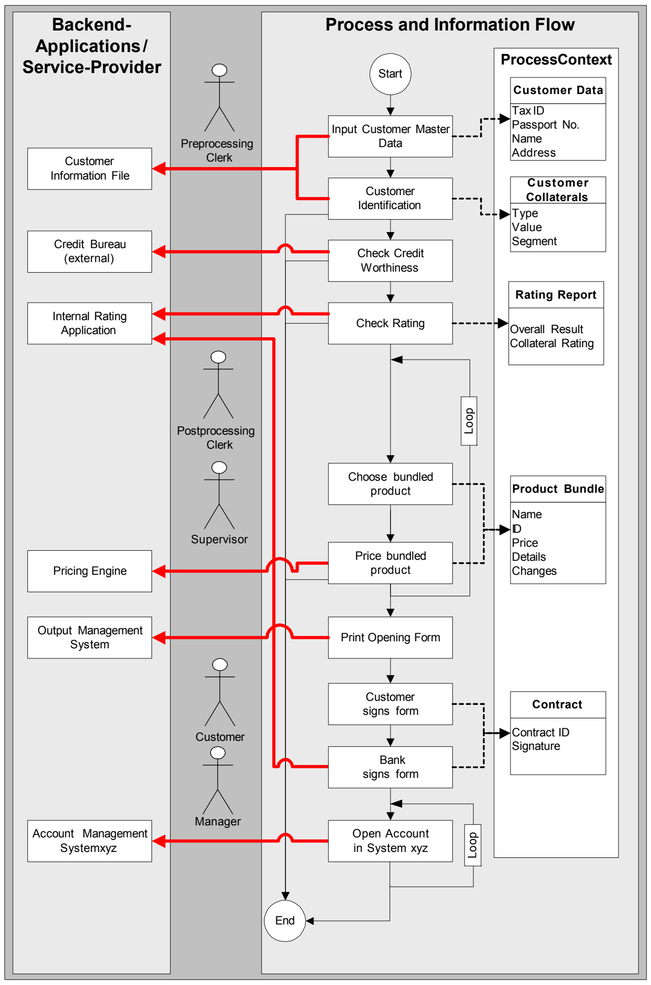

Figure 2: Loan Origination Workflow

Within the context of this paper we can only provide a high-level perspective and abstract the roles and access rights required on some external backend-application (left hand-side). Process-context information and the specific business objects access to which requires to be controlled are explicitly mentioned (right-hand side). Each of the workflow steps in this process will in turn be realised within several components (e.g. ABAP transactions) and are mapped to system-level guided procedures and rules.

Table 3 now defines a set of possible separation of duty properties, partially derived from the taxonomy in Figure 1. These are subset of the properties we discussed with SAP Banking Solution Architects.

Based on the previous descriptions and properties there are now several questions we would like to be able to ask and later formally specify for automated verification by the model checker. However, we explicitly exclude workflow analysis related questions (e.g. possible deadlocks) as this has already been sufficiently researched [35].

1. **SoD-based Safety:** Given a set of static and dynamic separation of duty policies, are these maintained over a finite sequence of states?
   - Can a desired state x not be reached due to these policies?
   - Can an explicitly excluded state be reached?
2. **Delegation and Revocation-based Safety:** Given the ability to delegate and revoke, can a principal obtain a certain right at some state?
   - What is the valid initial authority state to prevent a principal p obtaining a right?
   - Can a separation of duty property be "circumvented"?
   - Can a principal always revoke what he delegated? (Without blocking, e.g. an existing SoD property)
3. **Task-based Safety:** Given a set of tasks requiring access rights, will a principal be able to perform these tasks?
   - What is the valid initial authority state to allow a principal to perform his tasks?
   - Is it possible to have an "optimal" / least privilege system?
   - What is the valid initial authority state (with respect to assignment of the right to delegate) to allow a principal to perform his tasks? (So he could get the right from a colleague?)

Figure 3: Safety Properties

The general safety question considers whether given an initial state s_x (with an assignment of access rights and tasks) a defined state s_y can be reached. We would thus like to be able to check whether a principal can obtain a specific right at some stage; whether he can exercise this right on some object; and whether a desired authorisation state (at reference monitor evaluation time) cannot be reached due the initial authority state (ie. initial access control matrix setting). We thus group the safety properties to be verified according to the following three groups as informally summarised in Figure 3.

We would also like to be able to perform some "critical" state analysis, e.g. during run-time of the system a state occurs that is alarming but not critical if there is a set of possible future paths that introduce a mitigating factor; demonstrate that an object is not accessed; or that a dynamic SoD is maintained. In a similar fashion we would like to be able to perform some reverse trace analysis to determine what initial configurations and possible paths exist given any of the above properties and some undesired state x? This is similar to work performed in the area of safety critical systems analysis [36].

Customer tailored Process Product Bundling		
Possible SoD property	***Type (as defined in Figure 1)***	***Possible required Contextual information***
No person may be assigned to the two exclusive roles pre/post processor	SSSoD	Role Directory vs. User Dictory
A person may be assigned to the two excusive roles pre/post processor but must not activate them	SDSoD	This would mean to check for two things: a) they are not activated at any state, b) they have not been activated one after the other
If customer is industrial customer, master data must be verified by independent clerk	Application specific	This property would require the existence of a rule linked to the type of a customer account. Secondly, a notion of workflow is required to trigger the independent verification.
If credit bureau rating is negative then internal rating must be performed by another clerk	Application specific	This is a rule that would need to be attached to the workflow step of receiving the result.
If internal rating is negative, then case must be confirmed by supervisor.	Application specific	This is a rule that would need to be attached to the workflow step of receiving the result.
Clerk may only price bundled product if he did not perform operation “modify ()” wrt to the specific offer	ObjSoD	This is an example of a dynamic separation of duty property that requires contextual information about the execution path of a workflow and the specific business object (bundled product) that was manipulated.
If this is an industrial customer, then a clerk may perform tasks 1.-9. or 10 but not both for the same customer	OpSoD	This is an example of a dynamic separation of duty property that requires contextual information about the execution path of a workflow and the specific case (customer) that was manipulated.
A principal may be a member of the two exclusive roles pre/post processor and the complete set of authorisations acquired over these roles may cover a critical authorisation set, but a principal must not use all authorisations on the same object(s).	HistSoD	This is like ObjSoD and OpSoD together. We require to check the execution path and object access versus the critical authorisation set.
A principal p1 may be assigned to the two exclusive roles post processor and supervisor. He may also activate them but not use them on the same object (Product Bundle). (Compare in detail with section 5.3)	ObjSoD + Application specific	We should interpret this as two exclusive roles not having the same rights on a Business Object Type (not a particular instance). If we check for the property then we should get two traces: a) at step 6 the pricing was done for less then 100k – this is ok no violation of property as supervisor is not involved. b) at step 6 the pricing was done for more then 100k – this is ok only if not p1 in the supervisor role does commit operation

Table 3: SoD properties in a loan origination process

4 MODEL CHECKING

In order to aid in the automated analysis of complex systems and properties as described in the previous sections we apply model-checking techniques [37]. Such techniques have already been used and refined in other domains such as safety-critical systems analysis, e.g., to verify the correctness of railway control systems or aircraft controllers. Model checking is a technique for the automated verification of *finite* state-based (concurrent) systems. The proof of a property is entirely carried out by the machine. In case the property does not hold, the model checker will construct a counter-example suitable for failure diagnosis.

In mathematical terms, the considered (finite) systems are represented as finite state-based transition graphs (Finite State Machine, FSM). A *Finite State Machine* consists of a finite set of states; a set of initial states (a subset of the set of states); a transition relation (states are accessible from the current state); a function mapping each state to the atomic propositions holding in this state.

The aim of model checking is to automatically verify that the FSM in question satisfies certain properties. Often those properties can be formulated in propositional linear temporal logic (LTL) such that the dynamic behaviour of the system can be investigated.

Various model checking tools exist. For a reference see [37]. In the following section, we discuss the NuSMV model checker which will be later employed for the verification of workflow SoD properties.

4.1 The Model Checker NuSMV

The NuSMV [38] is a symbolic model checker, which is an extension of McMillan’s SMV system [39]. Beyond SMV’s BDD-based model checking NuSMV now supports also model checking techniques based upon propositional satisfiability. This way Bounded Model Checking (BMC) [40] can also be supported. BMC is an optimisation such that the search is restricted to a finite time interval instead of searching the whole time bar.

The FSM can be specified by an intuitive input language. Since it is intended to describe FSMs, the only data types are finite ones, namely Booleans, scalars, and fixed arrays. In addition, reusable components can be specified by modules. The primary purpose of NuSMV’s input language is to describe the transition relation of the FSM in question. For this purpose, `next` expressions can be used. For example, if we have specified `next(b):=1;` for a Boolean state variable `b`, this means that in the following state `b` is true.

Moreover, with the help of the `init` function, we can also define initial values for state variables (remember that an FSM has a set of initial states). It is also possible to define variables which do not change over time and variables which are completely unrestricted. The unrestricted variables are called input variables (keyword `IVAR`) and can change arbitrarily.

In order to specify asynchronous systems (e.g., distributed systems or hardware circuits), a `process` statement can be used. Due to the fact that we do not need this statement in our current workflow model, we do not describe it here. If, however, we intend to consider multiple workflow instances as intended in future work, the `process` statement might be helpful.

4.2 Linear Temporal Logic LTL

As pointed out above, we can specify the FSM with the help of the SMV input language. However, we also need a way to specify the properties which the FSM should satisfy. NuSMV offers two formalisms for this purpose, namely CTL (computation tree logic) and propositional LTL. In the following, we will use LTL for the specification of dynamic SoD properties. As pointed out in [41], LTL is well-suited to specifying dynamic access control policies.

LTL [42] uses the familiar Boolean operators like $\wedge$ and $\vee$. On the other hand, special temporal operators have been introduced:

- `F p` (sometimes in the future holds `p`),
- `G p` (globally in the future holds `p`),
- `p U q` (`p` holds until `q`), and
- `X p` (`p` is true in the next step).

Moreover, corresponding past modalities are also available (such as `H` – historically, `O` – once in the past, `Y` – one step before). To sum up, LTL characterises each linear path induced by an FSM. NuSMV allows specifying properties in an extra section called LTLSPEC. It is possible to define several LTL properties for an FSM at the same time.

5 MODEL CHECKING WORKFLOW ACCESS CONTROL POLICIES

As indicated earlier in this paper, we often must deal with *dynamic* security policies in the context of workflows. One example are the various kinds of dynamic SoD policies as those described in the context of the loan origination workflow. Due to delegation and revocation the access rights available to a user may change over time. Since workflows (for example, due to loops and branches) can be quite complex, an automated analysis of such policies is desirable. For example, the question arises whether a particular workflow instance satisfies dynamic SoD policies or certain access rights leak to unauthorised users. Specifically, due to delegation and revocation, unwanted security properties may arise such as the violation of dynamic SoD. Hence, model checking tools like the NuSMV may give the policy designer the opportunity to detect such as undesirable properties and to change the policy appropriately.

There are other model-checking based approaches for the verification of access control policies such as [43]. However, our approach is tailored towards SoD, delegation and revocation policies, specifically in the context of workflows. Due to the fact that we would like to directly map the workflow access control policies to an FSM we decided to use a model checker that allows one to directly encode the workflow. The RW language described in [43] is not primarily designed towards such needs.

In summary, our model checking-based approach for policy verification works as follows: The workflow access control policies (e.g. user-role assignments), the workflow and the delegation and revocation steps are specified by means of an FSM, and then the SoD properties are specified in LTL. In the following, this approach is discussed in more detail.

5.1 Modelling the workflow in SMV

Due to the fact that workflows may include branches and loops we model the workflow directly as an FSM. For this purpose, we introduced a certain scalar state variable `step` with values `s1,…, sn`. This variable indicates the current workflow step to be performed.

Furthermore, we assume a RBAC96-style role-based access control approach for workflows, i.e., if a user executes a certain workflow task, he is assigned to an appropriate role; moreover, this role must be activated on executing the task in question. In addition, note that a task may consist of more than one operation to be performed. For example, the `Input Customer Data` task of our loan origination process consists of the `query` and `update` operation on the business object `Input Customer Data`. The following state variables have been introduced in order to describe role-based access control policies for workflows:

- For each user-role assignment, a variable `UA_u_r` has been introduced. `UA_u_r` is true iff the predicate `UA(u,r)` is true for a user u and role r.
- Similar state variables are introduced for permission assignment, i.e., `PA_p_r` is true iff `PA(p,r)` is true.
- For each role activation `activate(u,r)`, we define a state variable `activate_u_r`.
- As proposed in [41], we also express the fact that user `u` actually performs operation `op` on object `o` with a state variable `exec_u_op_o`.

Beyond the RBAC-related variables, we define control flow variables which govern the execution flow. For example, we have introduced a Boolean variable `greater100k` indicating that we deal with a credit exceeding the 100k threshold. Due to the fact that we do not want to restrict this variable in advance and that on the other hand the variable should be constant during the whole workflow instance, we use the following trick of specifying `next(greater100k):=greater100k` without initialisation. This means we can choose the value of `greater100k` for the workflow at random, but once chosen, the value does not change any more.

The RBAC-related state variables must be set appropriately in the FSM. For example, if `exec_u_op_o` is true in a certain step, then there must be at least one role `r` activated which contains the access right `PA_op_o_r` at this step. Hence, the FSM must reflect this condition correctly. Moreover, RBAC-related state variables can be defined as unrestricted in certain cases in order to check different scenarios at one run of the model checker. However, this can lead to the so-called state explosion problem.

In addition, NuSMV offers a simulation functionality which allows exploring possible executions of the system in question. By means of this functionality, runs of the workflow can be simulated, i.e., certain unrestricted variables (control flow variables or RBAC-related variables) could be set. This way, the behaviour of the access control policy can be checked.

In Figure 4 an excerpt of the loan origination workflow is given showing how the steps 3 to 5 have been mapped to the FSM. In order to obtain a better overview, a summary of the relevant NuSMV variables and their meaning is given in Table 4.

NuSMV variable	Meaning
`step`	Represents the current workflow step
`greater100k,ratingokcb, …`	Control variables for the workflow
`UA_u_r`	UA-related variables
`PA_p_r`	PA-related variables
`activate_u_r`	User u activates role r
`exec_u_op_o`	User u executes op on object o
`exec_delegate_u1_r_u2`	Execution of a delegation step
`exec_revoke_u1_r_u2`	Execution of a revocation step

Table 4: The NuSMV variables and their meaning

```
step=s4 & exec_u2_prepare_ratingreport
& exec_u2_release_ratingreport
& exec_u2_post_ratingreport:s5;

step=s5 & exec_u2_query_ratingreport:s6;

step=s6
& exec_u2_queryavailableproducts_productbundle &
!greater100k:s7;
step=s6
& exec_u2_queryavailableproducts_productbundle &
greater100k:s8;
...
```

Figure 4: Excerpt of the NuSMV specification of the loan origination workflow.

5.2 Modelling Delegation and Revocation in SMV

We have also modelled delegation and revocation policies as discussed in section 2.3. Specifically, we can handle two kinds of delegation: We can delegate all the access rights required to execute a task (task delegation) and secondly, we can delegate single access rights irrespective of delegating a corresponding task.

Delegation and revocation are carried out with the help of temporary roles `r`, which are delegated from one user `u1` to another user `u2`. These roles contain the groups of rights to be delegated. As a prerequisite, user `u1` must clearly hold all the access rights belonging to role `r` on the delegation process. Once again, the FSM must be defined in a way that this condition holds. Two further predicates are used for delegation and revocation:

- `exec_delegation(u1,r,u2)`, meaning `u2` obtains role `r` from `u1` by delegation and
- `exec_revocation(u1,r,u2)`, meaning `u1` revokes role `r` from `u2`.

Similarly to the `exec_u_op_o` state variables, we introduce the variables `exec_delegate_u1_r_u2` and `exec_revoke_u1_r_u2` to express the aforementioned predicates. Furthermore, delegation and revocation are regarded as a single step within the workflow. Hence, if `u1` delegates the temporary role `updatecustomerdata` in step `s3` of the workflow, we can specify this in the NuSMV input language as follows:

```
next(step):=
case
  …
  step=s3 &
  exec_delegate_u1_updatecustomerdata_u2:s4;
  …
esac;
```

If the delegation has been successfully performed, the `UA` relation must be adapted appropriately:

```
next(UA_u2_updatecustomerdata):=
case
   exec_delegate_u1_updatecustomerdata_u2:1;
   1:0[1];
esac;
```

Hence, `UA` is a dynamic relation changing on certain points of time. Alternatively, we could have added a further predicate `UAD` to our model, indicating the delegated user assignment. With the help of NuSMV, it can now be checked if certain SoD properties are violated by delegation and revocation steps. This will be explained in Section 6.4. In addition, it can be investigated if certain access rights leak to unauthorised persons.

5.3 Specifying SoD properties in LTL

Having outlined the FSM for the role-based access control policies of workflows, we demonstrate now how various SoD properties can be specified in LTL. The FSM describing the access control policy of the workflow can then be checked against these properties. Subsequently, we discuss several SoD properties, arising in the context of the loan origination workflow as defined in Figure 1 and formulate them in propositional LTL.

Simple Static SoD (SSSoD):

No user/principal may be a member of both the exclusive roles `ClerkPreProcessor` and `ClerkPostProcessor`. In LTL, we have then the following formulation for principal `u`:

```
G(!(UA_u_ClerkPreProcessor &
           UA_u_ClerkPostProcessor)).
```

Simple Dynamic SoD (SDSoD)

A principal may be a member of any two exclusive roles but must not activate them at the same time:

```
!(activate_u_clerkpreproc &
        activate_u_clerkpostproc).
```

There is a loophole with this property: The exclusive roles could be activated one after another. Hence, a better version for SDSoD would be, for example:

```
(activate_u_clerkpreproc ->
        ! F activate_u_clerkpostproc).
```

Operational Separation of Duties (OpSoD)

A principal may be a member of some exclusive roles as long as the set of authorisations (operations) acquired over these roles does not cover an entire workflow.

Here we need to relax “entire workflow” to steps 1.-9. of the banking workflow only. This can theoretically be done by the two

[1] The label `1` represents in the NuSMV input language the default case, i.e., `UA_u2_updatecustomerdata` is false in that default case.

roles `ClerkPreProcessor` and `ClerkPostProcessor` only (if we do not exceed the 100k thresholds). This would mean to check for two things:

1. they are not assigned over two roles at any state,
2. they have not been delegated and revoked one after the other over some states such that never at any state all authorisations cover 1.-9.

Once again, the second variant – which is stronger than the first one – prevents a principal from circumventing OpSoD by delegation and revocation. A discussion follows in Section 6.4. In propositional LTL, this second variant can now be expressed as follows:

```
!(F auth_u_update_customerdata &
 F auth_u_query_customerdata &
 F auth_u_prepare_ratingreport &
 F auth_u_release_ratingreport &
 F auth_u_post_ratingreport &
 F auth_u_query_ratingreport &
 F auth_u_queryavailproducts_productbundle &
 F auth_u_update_productbundle &
 F auth_u_commit_productbundle);
```

Note that we introduced here further state variables indicating that principal `u` is authorised to execute the operations in question such as update, query, or prepare.

Object-based Separation of Duties (ObjSoD)

A principal may be a member of any two exclusive roles and may also activate them at the same time, but he must not act upon the same object through both.

Considering our loan origination workflow, we could introduce an ObjSoD restriction for the Price Bundled Product task. The roles `Clerk Postprocessor` and `Supervisor` are exclusive in this case. Further assume that user `u` has activated both the `Clerk Postprocessor` role and the `Supervisor` role. According to ObjSoD the activation of both the roles is not forbidden. On the other hand, no user/principal is permitted to execute the `update product bundle` access right in the `Clerk Postprocessor` role and the `commit product bundle` access right in the `Supervisor` role. In LTL we can formulate this the following way:

```
G ((exec_u_update_productbundle &
   activate_u_clerkpreproc) ->
       ! F(  activate_u_supervisor &
             exec_u_commit_productbundle))
```

We have checked for that property, and got two traces:

1. the pricing was done for less than 100k (`!greater100k`) – no violation of the property as the supervisor is not involved
2. the pricing was done for more than 100k – this is only ok if `u` does not commit in the `Supervisor` role.

This shows that we cannot blindly check for generic separation properties but must take application specific constraints (e.g., monetary thresholds) into consideration.

History-based Separation of Duties (HistSoD)

A principal may be a member of some exclusive roles and the complete set of authorisations acquired over these roles may cover an entire workflow, but a principal must not use all authorisations on the same object(s). This is a combination of OpSoD and ObjSoD and can be specified in LTL because (past) temporal operators are available.

Other SoD Rules of the Loan Origination Process:

Similar to the previous examples, the SoD rules not mapped to the taxonomy in Figure 1 have been specified and checked by the NuSMV system. For example, consider the rule “If credit bureau rating is negative, then internal rating must be performed by different clerk.” Assuming we have introduced a flow variable `ratingokcb`, indicating whether the rating is positive, we can express this in LTL:

```
(!ratingokcb ->
(exec_u_post_querycreditbureau
    & ! X exec_u_query_ratingreport));
```

6 ANALYSIS

Based on the defined separation properties and defined delegation and revocation functions we can summarise the following results and insights gained from our analysis.

6.1 Maintaining SSSoD

The direct delegation and revocation of authorisations will not have a effect on an existing SSSoD property since this only looks at mutually exclusive roles. However, from an administrative viewpoint we would want to periodically check over the lifetime of a system that the respective subset of the access control matrix of a principal does not entail the same authorisations as they would result out of computing the access control matrix for the SSSoD property. This, however, would prove difficult if principals also do have the ability to revoke authorisations such that the full set would never exist. It is for this reason that we suggest to also consider the delegation and revocation history logs and derive whether in a given period the critical set of authorisations was available to a principal. We had already demonstrated a similar scenario in [5].

6.2 Maintaining SDSoD

The same observations as for the SSSoD property hold for the SDSoD property. The only difference is that here we need to check for the activation of roles. This would mean that if delegation and revocation functionality is available to a principal then we would have to periodically check that within the activation phase of a role, the principal does not receive the set of authorisations as defined by a mutually exclusive role. Again, the logs for delegation and revocation activities have to be considered.

6.3 Maintaining ObjSoD

This property differs to the previous properties as it explicitly considers object access. This means that for maintaining the property, two checks will have to be made. The first would need to consider the object access history and compare the used authorisations to the critical authorisation set resulting out of the mutually exclusive roles. With respect to existing delegation and revocation functionality, we would have to check whether the principal used any delegated authorisations for the object access, whether these were revoked, and whether these correspond to the critical authorisation set as defined within an existing ObjSoD property.

6.4 Maintaining OpSoD

To maintain this property requires an additional piece of information: The underlying workflow model within a specific instance of which objects are accessed. We then need to compare the set of required authorisations for a workflow with the critical set of authorisations computed from a principal's exclusive roles. In addition, we need to check for any delegation and revocation activities within the duration of a workflow and whether these resulted in the acquisition of the critical authorisation set.

During our analysis we have encountered a scenario by means of NuSMV where the aforementioned naïve version of OpSoD (cf. Section 5.3) can be circumvented whereas the second variant of OpSoD cannot. In this scenario Principal `u1` delegates the access rights of his `Clerk PreProcessor` role to `u2` one after the other. After `u2` has exercised this access right, `u1` immediately revokes it. We further assume that `u2` has the role `Clerk PostProcessor`. If we did not exceed the 100k limit, then the supervisor is not involved, and `u2` could execute all operations covering the steps 1.-9. of the banking workflow. This way, OpSoD could be violated. In our test run, the first property is evaluated to true by NuSMV whereas the second is false. In fact, the first constraint is satisfied at every time step: `u2` never has all the authorisations of the critical set at a point of time. NuSMV also gives a counterexample (trace) in case of the second (stronger) property.

In Figure 3, various safety properties have been given, which can be checked. For example, we can verify if a principal can revoke what he delegated or if certain SoD constraints may prevent a delegation or revocation step from being executed (cf. [5], for example). By our analysis, we can rule out such useless delegation and revocation steps in advance. For example, assuming that revocation should be executed at step `s5`, we can encode this directly within the FSM:

```
next(step):=
case
   …
   step=s5 & next(OpSoD) &
   exec_revoke_u1_r_u2:s6;
   …
esac;
```

Note that `OpSoD` is only a placeholder for the concrete operational SoD constraint (propositional expression) that must be satisfied. In order to check now if revocation always succeeds, we can check if step `s6` can be reached. Thus, we can consider the following LTL formula: `F step=s6`. Clearly, the other SoD properties can be verified in a similar way.

6.5 Maintaining HistSoD

The same observations for the ObjSoD and OpSoD properties hold with respect to maintaining the HistSoD property. A combination of the required checks and comparisons of static critical sets, object access and dynamically acquired and revoked authorisations will need to be performed.

7 PRACTICAL IMPLICATIONS

Regarding the transformation of workflow models into an SMV model, we envisage this to be fully automated without any difficulties as the SMV input language is powerful enough to capture the semantics of standardised workflow modelling languages like BPEL.

Our delegation model is still very rudimentary and will fail once we start to consider delegation of task types and task instances. We will thus continue with the full adoption of the formal delegation model as defined in [11, 12]. This point will at the latest need to be resolved once we start to address enforcement of separation properties over several workflow models and instances.

The most challenging area is that of mapping the system resources at modelling- and run-time. For example, at the modelling level concepts such as exclusive roles can be defined in a SAP system and can be easily extracted in form of relational tables. This will also be possible at run-time (e.g. to determine activation of exclusive roles) but is assumed to be quite costly and thus realistically only desirable for selected critical applications and processes (e.g. procurement to stock and approval).

Another aspect is that of specification of rules (such as Separation of Duties) in a declarative language at the application level and transformation into LTL. This would allow for quantification of generic SoD properties over, for example, business object types. Many resource planning and access management systems like SAP or SAM Jupiter already link to rule engines like iLog or JESS. We have, however, not yet analysed the semantics of the different available rule languages and first-order LTL.

8 SUMMARY AND CONCLUSION

This paper has presented a model-checking based approach for automated analysis of delegation and revocation functionalities. This was done in the context of a real-world banking workflow requiring static and dynamic separation of duty properties.

As we intended to focus to a great deal on the underlying business case, we only outlined a possible approach of how to capture the workflow in a SMV model amended by a LTL-based specification of the Separation of Duty properties.

The results of our analysis confirmed some of the already existing insights into the possible unexpected use of delegation and revocation functionality to "circumvent" dynamic separation properties, dependent on the application context. As it is our overall goal to provide people with the ability to perform their work according to economic principles, the traces given by the model-checker gave us useful insights into different scenarios how a principal can accomplish his work.

REFERENCES

1. Samarati, P. and S. Vimercati, *Access Control: Polcies, Models and Mechanisms*, in *Foundations of Security Analysis and Design*, R. Focardi and R. Gorrieri, Editors. 2001, Springer Lecture Notes 2171. p. 137-196.
2. Harrison, M., W. Ruzzo, and J. Ullman, *Protection in Operating Systems.* Communications of the ACM, 1976. 19(8): p. 461-471.
3. Jaeger, T. and J. Tidswell, *Practical safety in flexible access control models.* ACM Transactions on Information and System Security (TISSEC), 2001. 4(2).
4. Crampton, J. *A reference monitor for workflow systems with constrained task execution.* . in *10th ACM Symposium on Access Control Models and Technologies*. 2005.
5. Schaad, A., *A Framework for Organisational Control Principles, PhD Thesis*, in *Department of Computer Science*. 2003, University of York.

6. Atluri, V. and W. Huang, *An Authorization Model for Workflows.* Lecture Notes in Computer Science, 1996. 1146: p. 44-64.
7. Rits, A., B. deBoe, and A. Schaad. *XacT: A bridge between resource management and access control in multi-layered applications*. in *Software Engineering for Secure Systems – Building Trustworthy Applications (SESS'05)*. 2005. St. Louis, MO, USA.
8. Sohr, K., L. Migge, and G. Ahn. *Articulating and enforcing authorisation policies with UML and OCL*. in *Software Engineering for Secure Systems - Building Trustworthy Applications (SESS'05)*. 2005. St. Louis, MO, USA.
9. Simon, R. and M. Zurko. *Separation of Duty in Role-Based Environments*. in *Computer Security Foundations Workshop X*. 1997. Rockport, Massachusetts.
10. Ahn, G. and R. Sandhu, *Role-based authorization constraints specification.* Information and System Security Journal, 2000. 3(4): p. 207-226.
11. Schaad, A. *An Extended Analysis of Delegating Obligations.* in *IFIP DBSec* 2004.
12. Schaad, A. *Revocation of Obligation and Authorisation Policy Objects.* in *IFIP DBSec 2005*. 2005.
13. Schulz, K. and M. Orlowska, *Facilitating cross-organisational workflows with a workflow view approach.* Data Knowl. Eng. , 2004. 51(1): p. 109-147.
14. Frossard, A., *Delegation of Tasks in Workflow Management Systems*, in *School of Computer and Communication Sciences (IC)*. 2005, Ecole Polytechnique Fédérale de Lausanne (EPFL) Lausanne, Switzerland.
15. Sandhu, R., et al., *Role-based access control models.* IEEE Computer, 1996. 29(2): p. 38-47.
16. Damianou, N., et al. *The Ponder Policy Specification Language*. in *Policies for Distributed Systems and Networks*. 2001. Bristol: Springer Lecture Notes in Computer Science.
17. Pugh, D., *Organization Theory: Selected Readings*. 4th ed. Penguin Business. 1997: Penguin Books.
18. Mintzberg, H., *The structuring of organizations*, ed. E. Cliffs. 1979, NJ: Prentice-Hall.
19. Botha, *Separation of duties for access control enforcement in workflow environments.* IBM SYSTEMS JOURNAL, 2001. 40(3).
20. Saltzer, J. and M. Schroeder. *The protection of Information in Computer Systems*. in *IEEE*. 1975.
21. Clark, D. and D. Wilson. *A Comparison of Commercial and Military Security Policies*. in *IEEE Symposium on Security and Privacy*. 1987. Oakland, California.
22. Sandhu, R. *Transaction Control Expressions for Separation of Duties*. in *4th Aerospace Computer Security Conference*. 1988. Arizona.
23. Sandhu, R. *Separation of Duties in Computerized Information Systems*. in *IFIP WG11.3 Workshop on Database Security*. 1990. Halifax, UK.
24. Nash, M. and K. Poland. *Some Conundrums Concerning Separation of Duty*. in *IEEE Symposium on Security and Privacy*. 1990. Oakland, CA.
25. Baldwin, R. *Naming and Grouping Privileges to Simplify Security Management in Large Databases*. in *IEEE Symposium on Security and Privacy*. 1990. Oakland.
26. Gligor, V., S. Gavrila, and D. Ferraiolo. *On the Formal Definition of Separation-of-Duty Policies and their Composition*. in *IEEE Symposium on Security and Privacy*. 1998. Oakland, CA.
27. Ferraiolo, D., J. Cugini, and D. Kuhn. *Role-Based Access Control (RBAC): Features and Motivations*. in *Computer Security Applications*. 1995.
28. Kuhn, R. *Mutual exclusion of roles as a means of implementing separation of duty in role-based access control systems*. in *Proceedings of the second ACM workshop on Role-based access control*. 1997.
29. Nyanchama, M. and S. Osborn, *The role graph model and conflict of interest.* Transactions on Information Systems Security, 1999. 2(1): p. Pages 3 - 33.
30. Muller, J., *Delegation and Management.* British Journal of Administrative Management, 1981. 31(7): p. 218-224.
31. Moffett, J.D., *Delegation of Authority Using Domain Based Access Rules*, in *Dept of Computing*. 1990, Imperial College, University of London.
32. Zhang, L., G. Ahn, and C. B. *A Rule-based Framework for Role-Based Delegation*. in *6th ACM Symposium on Access Control Models and Technologies*. 2001. Chantilly, VA, USA.
33. Hagstrom, A., et al. *Revocations - A Categorization*. in *Computer Security Foundations Workshop*. 2001: IEEE.
34. Schaad, A. and J. Moffett. *Separation, review and supervision controls in the context of a credit application process: a case study of organisational control principles*. in *ACM SAC* 2004.
35. Janssen, W., et al., *Model Checking for Managers.* Lecture Notes in Computer Science, 1999. 1680.
36. Loer, K. and M. Harrison. *Towards Usable and Relevant Model Checking Techniques for the Analysis of Dependable Interactive Systems*. in *ASE*. 2002.
37. Clarke, E., O. Grumberg, and D. Peled, *Model Checking*. 2000: The MIT Press.
38. Cimatti, A., et al. *NuSMV2: an Open Source Tool for Symbolic Model Checking* in *QA075 Electronic computers. Computer Science http://eprints.biblio.unitn.it/archive/00000085*. 2002.
39. McMillan, K., *The SMV system, Symbolic Model Checking - an approach* 1992, Carnegie Mellon University CMU-CS-92-131.
40. Biere, A., A. Cimatti, and Y. Zhu, eds. *Symbolic model checking without BDDs*. Tools and Algorithms for the construction and analysis of systems Vol. 1579. 1999, Springer LNCS.
41. Mossakowski, T., M. Drouineaud, and K. Sohr. *A temporal-logic extension of role-based access control covering dynamic separation of duties*. in *TIME-ICTL*. 2003. Cairns, Queensland, Australia.
42. Goldblatt, R., *Logics of Time and Computation, 2nd Edition, Revised and Expanded.* CSLI Lecture Notes, 1992. 7.
43. Zhang, N., M. Ryan, and D. Guelev. *Evaluating Access Control Policies Through Model Checking*. in *ISC*. 2005.