A Workflow Instance-based Model-checking Approach to Analysing Organisational Controls in a Loan Origination Process[1]

Andreas Schaad

SAP Labs France, Security & Trust Group
805, Avenue du Dr Maurice Donat
06250 Mougins, FRANCE
andreas.schaad@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 systems, 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. We expand on some of our earlier work reported in [44], which was restricted to single workflow models and not arbitrary workflow instances of one or several models. This initial restriction to model-checking at the workflow model level meant that we were not able to analyse delegation and revocation of tasks and access rights in the detail as required in some of our earlier conceptual models [11, 12]. 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. Possibly required separation of duty properties were translated into a set of LTL-based constraints.

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 and our earlier work on model-checking access control in workflow systems [44] was a first step into that direction.

Accordingly, this paper presents an extended 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. Extended means that, instead of single models only, we are now able to analyse behaviour within arbitrary workflow instances of one or several models. The earlier restriction to model-checking at the workflow model level meant that we were not able to analyse delegation and revocation of tasks and access rights in the detail of our conceptual models established in [11, 12]. We derived information about the workflow from BPEL specifications and business object repositories. This was captured in a

[1] The work of Andreas Schaad was sponsored under the EU FP6 IP project "R4eGov". The work of Karsten Sohr was sponsored under the BMBF funded project "ORKA"

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. However, we need to again stress that the formalisms behind our approach have already been published elsewhere [11, 12].

The scope of this paper is to give an overview of how capture some of these formalisms within a model-checking environment as well as instantiate and analyse them in the context of real-world scenarios. The rest of the paper will provide some more required background information regarding the 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). Then we give a formal role-based access control model for workflows which serves as the basis for the model checking process (Section 4). After a brief summary of the current state of the art in the area of model-checking (Section 5) we then specify the banking workflow in SMV together with a defined subset of the constraints in LTL (Section 6). We then discuss some results of our analysis and provide some final conclusions and future research directions (Sections 7).

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).

Fig. 1. Separation Taxonomy (I)

2 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 alone.

An exhaustive overview is provided in [44] as well as [19 – 25] and we only summarise properties we modelled in LTL in Figure 1.

2.2 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.

2.3 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).

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

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.

3 Delegation and Revocation 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). 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). The customer and Bank finally come to an agreement expressed in the signature of the client and Bank representative.

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.

Based on the previous descriptions and delegation 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. One example would be to check that a principal p1 can only delegate a task instance to another principal p2 if p2 has the same role as p1. In fact, our underlying formal model requires that a task instance can only be assigned to a principal if the role of that principal has been assigned to the corresponding task type of the instance at the workflow model level.

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 Identification	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 Management System	post print request ()	Print Opening Form	Contract
Customer	-	sign ()	Customer signs form	Contract
Manager	-	sign () update ()	Bank signs form	Contract
Clerk Postprocessor	Account Management System	open ()	Open Account	Account

Table 2. Assignments of rights, roles and tasks

.

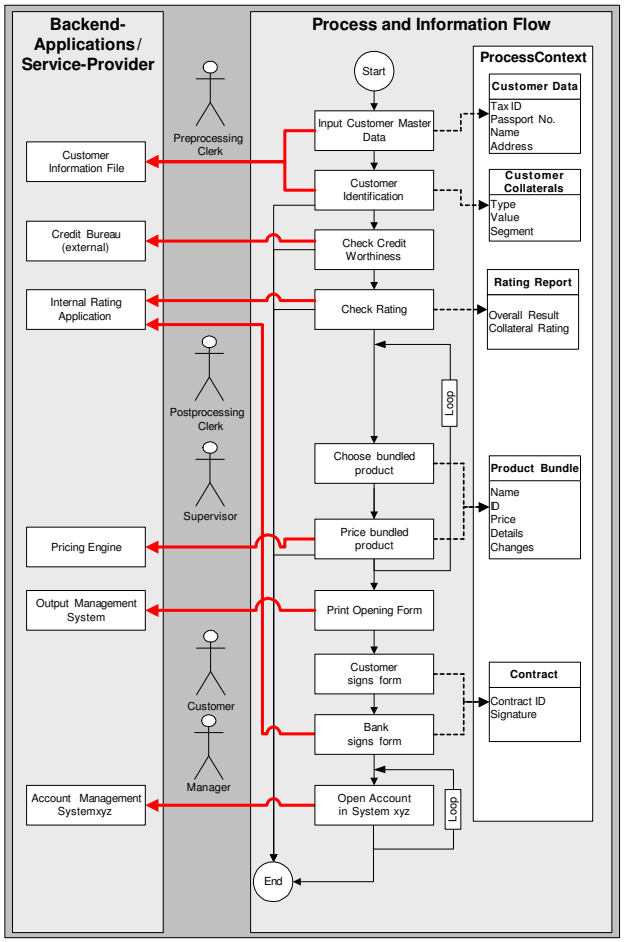

Fig. 2. Loan Origination Workflow

Another example of properties we would like to check for is that a revocation of a task instance requires a prior delegation of the same instance.

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 or demonstrate that an object is not accessed. 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].

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 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?)

Fig. 3. General Safety Properties

4 A Role-based Access Control Model for Workflows

In this section, we present our role-based access control model for workflows. Then, we show in later sections how to translate this model into finite state machines (so-called Kripke structures), which will serve as input to a model checker. The access control model is as general as possible and is independent of a specific workflow such in contrast to the approach described in [44].

We use first-order linear temporal logic (LTL) [42] in order to formally specify our role-based access control model for workflows. We use LTL because a workflow system can be considered as a reactive system. Specifically, the role-based security policies for workflows can be regarded as dynamic. For example, dynamic SoD properties are often required in the context of workflows and furthermore, due to task delegation and revocation the task assignment may vary on runtime. In addition, LTL allows for talking about things such as the execution history or order of executions, as in [26], while still being much simpler than [26]. Finally, LTL is often used by standard model checkers such as SPIN [45] and NuSMV [38] to express properties to be checked.

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

A temporal first-order signature consists of a set of sorts, a set of function symbols and a set of predicate symbols (each symbol coming with a string of argument sorts and, for function symbols, a result sort). Predicate symbols are partitioned into rigid and flexible symbols: the former do not change over time, while the latter may vary. Models live over discrete time, indexed by the natural numbers as time steps. They interpret the sorts with (time-independent) carrier sets, function and rigid predicate symbols with time-independent functions and predicates of appropriate types, and flexible predicate symbols with families of functions and predicates, where the families are indexed by natural numbers.

Sentences are the usual first-order sentences built from equations, predicate applications and logical connectives and quantifiers ∀, ∃. Additionally, we have the modalities, G ("globally in the future"), F ("sometimes in the future"), X (in the next step), and U ("until"). Often the corresponding past modalities H ("historically"), O ("once in the past"), Y ("one step before"), and S ("since") also exist. Satisfaction is defined inductively for a given time step, where the modalities allow for referring to other time steps. A sentence is satisfied in a model if it is satisfied in the time step zero.

Figure 4 now gives an overview of the relevant sets, predicates, and axioms needed for describing our role-based access control model for workflows. In particular, we assume a CASL-style formalization as used in [41]. In Table 4 and Table 5, the meaning of the sets and the predicates which are used in the formal specification from Figure 4 can be found.

We assume that all sets are finite, which is essential for the model checking process described later (cf. Section 5). Moreover, we assume an RBAC96-style role-based model for workflows [15]. However, we omitted the session concept and introduced a predicate __Active_for__ instead. r__Active_for__u indicates that role r has been activated by user u.

Further sorts $Sort_1$, …, $Sort_n$ and predicates P_1, …, P_m may also be necessary which are used to express workflow dependencies. These sorts and predicates are clearly workflow-specific. In a certain banking business process, for example, we may discriminate between private and industrial customers. Thus, we may introduce the sort Customer and a predicate isIndustrial: Customer. Depending on whether that predicate is evaluated to true or false different tasks may be executed in the workflow, i.e., different paths are chosen.

There are several axioms defined in the RoleBasedWorkflow specification. For example, it is stated that a role can only be activated if the appropriate user assignment has been done before and that a user may only perform an operation on an object if she gains the appropriate permission from a role. Moreover, from the axiom Exec(u,op,o) ∧ Exec(u',op',o') ⇒ u=u' follows that per time step only one user may execute access rights.

SPEC RoleBasedWorkflow
sorts U, R, T, TT, Op, Obj , $Sort_1$, …, $Sort_n$;

flexible preds
- UA: U × R;
- PA: Op × Obj × R;
- __Active_for__: U × R;
- UT: U × T;
- TTA: TT × R;
- Exec: U × Op × Obj;
- ExecT: U × T;
- If required, further predicates P_1, …, P_m that express workflow dependencies

rigid preds
- __IsTaskTypeOf__: TT× T;

forall u:U, r:R, t:T,tt,tt'T, op,op':Op, o,o':Obj;
- u __Active_for__ r ⇒ UA(u,r)[2]
- Exec(u,op,o) ⇒ ∃ r:R.(r__Active_for__u ∧ PA(op, o, r))
- Exec(u,op,o) ∧ Exec(u',op',o') ⇒ u=u'
- tt __IsTaskTypeOf__ t ∧ tt' __IsTaskTypeOf__ t ⇒ tt=tt'
- ExecT(u,t) ∧ tt __IsTaskTypeOf__ t ⇒ ∃ r:R.(r__Active_for__u ∧ TTA(tt,r))
- ExecT(u,t) ⇒ UT(u,t)

SPEC RoleBasedWorkflowWithDelegation= RoleBasedWorkflow **then**

flexible preds
- ExecDelR: U × R × U;
- ExecDelT: U × T × U;
- ExecRevWeakLocalR: U × R × U;
- ExecRevWeakLocalT: U × T × U;

forall u,u':U, r:R, t:T;
- ExecDelT(u,t,u')⇒ X(UT(u',t) ∧ ¬UT(u,t)) ∧ UT(u,t) ∧ ∃ r:R.(UA(u',r) ∧ TTA(tt,r))
- ExecDelR(u,r,u')⇒ X(UA(u',r) ∧ ¬UA(u,r)) ∧ UT(u,r)
- ExecRevWeakLocalT(u,t,u')⇒ X(¬UT(u',t) ∧ UT(u,t)) ∧ O ExecDelT(u,t,u')
- G ((ExecDelT(u,t,u') & ¬ExecRevWeakLocalT(u,t,u') & F ExecRevWeakLocalT(u,t,u')) ⇒ (¬ExecRevWeakLocalT(u,t,u') U ExecRevWeakLocalT(u,t,u'))
- similar axioms for ExecRevWeakLocalR

Fig. 4. Role-based access control model for workflows, specified in a CASL style notation

[2] Strictly speaking, we ought to introduce here the modality G, but due to the Necessitation axiom this is not necessary [42].

Set	Meaning
U	Users
R	Roles
Obj	Objects
Op	Operations
T	Task (instances)
TT	Task types

Table 4. The sets involved in the access control model for workflows and their meanings

Predicate	Meaning
UA(u,r)	Assignment of user u to role r
PA(op, o,r)	Assignment of the permission (op,o) to role r
r __Active_for__ u:	Role r is active for user u
UT(u,t)	Assignment of user u to task t
TTA(tt,r)	Assignment of task type tt to role r
tt __IsTaskTypeOf__ t:	tt is the task type of task instance t
Exec(u,op,o)	User u executes operation op on object o
ExecT(u,t)	User u executes task t
ExecDelR(u,r,u')	User u delegates role r to user u'
ExecDelT(u,t,u')	User u delegates task r to user u'
ExecRevWeakLocalR(u,r,u')	User u revokes role r from user u'
ExecRevWeakLocalT(u,t,u')	User u revokes task t from user u'

Table 5. The predicates defined for the role-based access control model for workflows and their meanings

As pointed out elsewhere [46], the task concept is fundamental to workflow models. Hence, we introduce a task set T and several task-related predicates. Specifically, the predicate UT means that user u is obliged to execute task t and ExecT(u,t) means that user u executes task t. Nevertheless, we also model the execution of single access rights with the help of the Exec predicate in order make available a finer granularity. This way, our model allows one to specify SoD properties on access rights and not only on tasks or roles.

As discussed above, our model also comprises the notion of task types. This is reflected by the TT sort and the predicate isTaskTypeOf. The axiom tt __IsTaskTypeOf__ t ∧ tt' __IsTaskTypeOf__ t ⇒ tt=tt' states that the task type of a task instance is always unique. Note that we define this predicate as rigid, i.e., the predicate does not vary over time. This correlates to the practical assumption that a task type of a task instance may not change and is known in advance. The other predicates, however, are flexible. For example, the __Active_for__ predicate may vary over time because roles may be activated or deactivated at the discretion of users and may not be statically activated by an administrator.

In our model, tasks can only be executed if the task instance in question and the (unique) task type of this task instance are assigned to the user. This is reflected by the axioms

ExecT(u,t) ⇒ UT(u,t) and

ExecT(u,t) ∧ tt __IsTaskTypeOf__ t ⇒ ∃ r:R.(r__Active_for__u ∧ TTA(tt,r)).

The second part of the specification in Figure 1 comprises several delegation- and revocation-related predicates. We support two types of delegation at the moment, namely delegation of task instances and delegation of roles. We could have also defined the delegation of task types. However, this is already covered by the delegation of roles because in our model task types are assigned to roles and not to users directly. In case of task delegation, we explicitly demand that the user u' who receives the task instance must be assigned to a role r with the appropriate task type. Otherwise, the task instance could not be executed by u' later.

After a successful delegation step, the relations UT (in case of task delegation) and UA (in case of role delegation) are changed appropriately. Clearly, we demand as a prerequisite that the delegating user possesses the task/role at the delegation step. In addition, we demand that only that person may revoke a task/role who has delegated the task/role in question before. Both aforementioned properties are summarised in the axiom ExecRevWeakLocalT(u,t,u')⇒

X(¬UT(u',t) ∧ UT(u,t)) ∧ O ExecDelT(u,t,u').

The last axiom in the delegation section of Figure 4 says that no further revocation may occur between a delegation step and a revocation step. For reasons of simplicity, we describe only weak and local revocation (cf. Section 2) here at the moment.

5 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* (also called Kripke structure) consists of a finite set of states; a set of initial states (a subset of the set of states); a total transition relation (states are accessible from the current state and for all states, at least one successor state exists); a function mapping each state to the atomic propositions holding in this state.

The aim of model checking is to automatically verify that the Kripke structure in question satisfies certain properties. Often those properties can be formulated in propositional LTL such that the dynamic behaviour of the system can be investigated.

NuSMV variable	Predicate application
isIndustrialCustomer_c, greater100k_credit, …	IsIndustrialCustomer(c), Greater100k(credit)
UA_u_r	UA(u,r)
PA_p_r	PA(p,r)
TTA_tt_r	TTA(tt,r)
UT_u_t	UT(u,t)
activefor_u_r	r_Active_for__u
exec_u_op_o	Exec(u,op,o)
exec_u_t	ExecT(u,t)
exec_delR_u1_r_u2	ExecDelR(u1,r,u2)
exec_revR_u1_r_u2	ExecRevWeakLocalR(u1,r,u2)
exec_delT_u1_t_u2	ExecDelT(u1,t,u2)
exec_revT_u1_r_u2	ExecRevWeakLocalT(u1,t,u2)
isTaskTypeOf_tt_t	tt __IsTaskTypeOf__ t
s	N/A (current task execution, delegation or revocation step)

Table 6. Predicate applications and the corresponding NuSMV variables

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.

5.1 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 Kripke structure 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 Kripke structure 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 a Kripke structure 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.

5.2 LTL Model Checking with NuSMV

As pointed out above, we can specify the Kripke structure with the help of the SMV input language. However, we also need a way to specify the properties which the Kripke structure should satisfy. NuSMV offers two formalisms for this purpose, namely CTL (computation tree logic) and propositional LTL. In the following, we will use *propositional* LTL (in contrast to first-order LTL as introduced in Section 4) for the specification of dynamic SoD properties. As pointed out in [41], LTL is well-suited to specifying dynamic access control policies.

NuSMV makes available the modalities mentioned in Section 4 such as `G`, `F` and `X`. So we do not need to repeat the details here.

LTL characterises each linear path induced by a Kripke structure. NuSMV allows for specifying temporal properties in an extra section called LTLSPEC. It is possible to define several LTL properties for a Kripke structure at the same time.

6 Model Checking Role-based Workflows

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 and tasks 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 a Kripke structure 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 task execution as well as the delegation and revocation steps are specified by means of a Kripke structure, and then the properties to be verified are specified in LTL. In the following, this approach is discussed in more detail, in

particular it is shown how a role-based workflow is translated into a Kripke structure.

6.1 Modeling the Workflow in SMV

Owing to the fact that workflows may include branches and loops we model the workflow directly as a Kripke structure. Note that in our access control model for workflows all sets are finite and all predicates work on those finite sets. Finite sets and predicates are essential for the model checking process [37].

For all the predicates defined in Figure 4, we now introduce corresponding state variables as shown in Table 6. More exactly speaking, a state variable `pred_x1_x2_…_xn` is added for every relevant predicate application `pred(x1, …, xn)`. In the following, we give some examples of the state variables we have introduced in order to describe role-based access control policies for workflows:

- For each user-role assignment, we introduece a variable `UA_u_r`. `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_op_o_r` is true iff `PA(op,o,r)` is true.
- For each role activation `r __Active_for__ u`, we define a state variable `Activefor_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`.
- Similarly, we define a variable `Exec_u_t` for every user performing a task, i.e., iff `ExecT(u,t)` is true.

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 `Customer Data` (cf. Table 2). Thus, our model presented in Section 4 supports the two predicates `Exec` und `ExecT` for execution. Correspondingly, we introduced two kinds of variables `Exec_u_op_o` and `Exec_u_t`. In our example, we may then have the variables `exec_u_inputcustomerdata` as well as `exec_u_query_customerdata` and `exec_u_update_customerdata`, respectively.

Furthermore, we introduced an additional scalar state variable `s` with values `s1,…, sn, success, failure`. This variable indicates the current workflow step (state). The special values `success` and `failure` are introduced because the transition relation must be total according to the aforementioned definition of Kripke structures. Totality means in this context that for all states there *must exist* a successor state. In order to guarantee this condition, we define `failure` as a default case if there is no successor task execution, delegation or revocation step, i.e., if the workflow blocks at a certain time step. However, if we have successfully finished the workflow instance, we jump to the `success` state. If then the special states `success` and `failure` are reached, we stay in those states forever.

Beyond the RBAC-related and the step variables, we define control flow variables which govern the execution flow and correspond to the additional predicates P_1, …, P_m of our workflow access control model. For example, we have introduced a Boolean variable `greater100k_credit` 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_credit):=greater100k_credit` without initialization. This means we can choose the value of `greater100k_credit` for the workflow at random, but once chosen, the value does not change any more.

To obtain a better understanding of the resulting Kripke structure, we give an excerpt of the loan origination workflow in Figure 5 showing how the steps 3 to 5 from Figure 2 have been mapped to the Kripke structure.

```
next(s):=
case
…
 s=s4& exec_u2_checkcreditworthiness:s5;
 s=s5 & exec_u2_checkrating:s6;
    s=s6     &     exec_u2_choosebundledproduct     &
 !greater100k_credit:s7;
    s=s6     &     exec_u2_choosebundledproduct     &
 greater100k_credit:s8;
…
```

Fig. 5. Excerpt of the NuSMV specification of the loan origination workflow.

6.2 Modelling Delegation and Revocation in SMV

We have also modeled delegation and revocation policies as discussed in Section 2. Specifically, we can handle two kinds of delegation: task instance delegation and role delegation.

Similarly to the `exec_u_op_o` or `exec_u_t` state variables, we introduce the variables `exec_delR_u1_r_u2` and `exec_delT_u1_t_u2` to express both types of delegation. Similarly, we have the variables `exec_revR_u1_r_u2` and `exec_revT_u1_t_u2` to represent the corresponding revocation steps in our Kripke structure.

As mentioned above, delegation and revocation are regarded as a single step within the workflow. For example, if `u1` delegates the task `Input Customer Data` in step `s3` of the workflow, we can specify this in the NuSMV input language as follows:

```
next(s):=
case
…
s=s3 &
exec_delT_u1_inputcustomerdata_u2:s4;
…
esac;
```

If the delegation has been performed successfully, the `UT` relation must be adapted appropriately, e.g.:

```
next(UT_u2_inputcustomerdata):=
case
exec_delT_u1_updatecustomerdata_u2:1;
1:0[3];
esac;
```

Hence, `UT` is a dynamic relation changing on certain points of time as mentioned before.

6.3 Model Checking Workflows

Having outlined the Kripke structure for the role-based access control policies of workflows, we demonstrate now how various properties can be checked by NuSMV. Specifically, we can handle the following three kinds of questions:

- Are the axioms we defined for our role-based access control model fulfilled (cf. Figure 4)?
- Are certain SoD properties violated?
- What are the consequences of delegation and revocation steps (e.g., regarding SoD policies)?

Subsequently, we discuss these three aspects in more detail. We also show how a critical-state analysis can be carried out by NuSMV.

6.3.1. Verification of Axioms

In Figure 4, we gave several axioms that our role-based access control model for workflows must satisfy. For example, a task can only be executed by a user if that user is assigned to the appropriate task type (through a role). We could check now if our Kripke structure adheres to those axioms. Alternatively, we could assume that the Kripke structure has been constructed in a way that satisfies all the axioms. Then, there would be no need to check the axioms in question by means of the model checker.

In order to demonstrate how to check such properties by NuSMV, we take the following axiom as an example:

```
Exec(u,t) ∧ tt __IsTaskTypeOf__ t ⇒
∃ r:R.(r__Active_for__u ∧ TTA(tt,r)).
```

Assume that we have a user `u` and that the task type `InputCustomerDataType` is assigned to the role `Clerk Preprocessor`. Then, we have the following LTL property against which our Kripke structure could be checked:

```
G (exec_u_t &
 isTaskTypeOf_inputcustomerdatatype_
 inputcustomerdata
 -> activefor_u_clerkpreproc &
    TTA_inputcustdatatype_inputcustdata);
```

Similarly, we can check if a user has the adequate access rights to perform a task assigned to her. Once again, let us take the task `Input Customer Data` as an example. This task consists of the `query` and `update` operations which are executed on object `Customer Data`. Thus, we have the condition

```
G( exec_u_inputcustomerdata ->
       exec_u_query_customerdata &
       exec_u_update_customerdata);
```

Moreover, it must be checked whether u has the adequate permission to execute `update` and `query`. For example, we must verify the following condition in case of the `update` operation

```
G (exec_u_update_customerdata) ->
     (PA_clerkpreproc &
     activefor_u_clerkpreproc);
```

Concerning the revocation of tasks, we can also demonstrate that the user who revokes a task must have delegated that task before. We obtain the following NuSMV specification for this property:

```
G (exec_revT_u1_t_u2->
   O exec_delT_u1_t_u2)
```

6.3.2. Verification of SoD properties

First, we demonstrate how general SoD properties can be specified in LTL. The Kripke structure describing the access control policy of the workflow can then be checked against these properties. Subsequently, we discuss SoD properties, arising in the context of the loan origination workflow as defined in Figure 2 and formulate them in propositional LTL. Due to space limitations, we give only three examples for SoD properties.

Simple Dynamic SoD (SDSoD):

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

```
G(!(activefor_clerkpreproc_u &
     activefor_clerkpostproc_u));
```

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:

```
G((activefor_u_clerkpreproc ->
     ! F activefor_ clerkpostproc_u));
```

Task-based Dynamic Separation of Duties (TSoD):

Botha and Eloff introduced in [19] the concept of task-based dynamic SoD stating that a user must not execute two conflicting tasks within a workflow instance. Due to the fact that we added the concept of task instances to our role-based access control model for workflows we can now easily specify such properties. For example, the tasked-based SoD property "If customer is an industrial customer, master data must be verified by an independent clerk." can be formulated as follows

```
G (isIndustrialCustomer_c
     ->!(exec_u_verifycustomerdata &
            Y exec_u_inputcustomerdata));
```

Note that an additional task `Verify Customer Data` has been introduced.

[3] The label `1` represents in the NuSMV input language the default case, i.e., `UT_u2_inputcustomerdata` is false in that default case.

Other SoD Rules of the Loan Origination Process:
Similar to the previous examples, the SoD rules not mapped to the taxonomy, e.g., given in [9] 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 `isRatingOKCB_Customer`, indicating whether the rating is positive, we can express this in LTL:

```
G(!isRatingOKCB_customer ->
 (exec_u_post_querycreditbureau
    & ! X exec_u_query_ratingreport));
```

6.3.3. Critical-state Analysis of SoD properties

By means of NuSMV, we can also carry out a critical-state analysis of SoD properties. To take an example, let us consider the two mutually exclusive roles `Clerk Postprocessor` and `Supervisor` with an SDSoD constraint. Let us further assume that no user may execute both the `update product bundle` and `commit product bundle` operations within a workflow instance. Rather this latter SoD property matters in this scenario than the fact that both the aforementioned roles are mutually exclusive. Thus, we can allow a user to violate the SDSoD constraint as long as the second SoD property still holds. We can formulate this less restrictive property in LTL as follows:

```
G (activefor_u_clerkpostproc &
     F activefor_u_supervisor ->
         exec_u_update_productbundle &
         !F exec_u_commit_productbundle).
```

The first part of this specification represents the negated SDSoD property. The second part states that after executing the `update product bundle` operation, `commit product bundle` must not be performed by a user `u`.

If the model checker does not find any path through the Kripke structure that violates the aforementioned property, one can conclude that the violation of the SDSoD property might be alarming but not critical. Therefore, we might allow user `u` to activate both the `Clerk Postprocessor` and `Supervisor` roles. In contrast, if this property does not hold, we forbid the activation of both roles within a workflow instance.

6.3.4. Delegation and SoD properties

In this subsection, we sketch how delegation and SoD can interfere with each other. For this purpose, we take the aforementioned TSoD property. To maintain this property, we must consider the task access history and not the object access history as in the case of object-based SoD [24]. If a certain task `t1` has been performed within a workflow instance, a conflicting task `t2` must not be executed thereafter. Clearly, this may have effects on the delegation and revocation of tasks. Assume that we have the two conflicting tasks `Input Customer Data` and `Verify Customer Data`. Let us further assume that user `u1` has already executed `Input Customer Data` and user `u2` is obliged to perform `Verify Customer Data`. If now `u2` falsely delegates her task to `u1` in case of an industrial customer, then obviously either this task could not be executed by `u1` or the TSoD property would be violated. With the help of NuSMV one can construct such a Kripke structure and can then easily check this Kripke structure against the TSoD property in question.

7 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.

We expanded on some of our earlier work reported in [44], which was restricted to single workflow models and not arbitrary workflow instances of one or over several models. This initial restriction to model-checking at the workflow model level only meant that we were not able to analyse delegation and revocation of tasks and access rights in the detail as required in some of our earlier conceptual models [11, 12].

Based on a formal framework, we are now in a position that allows us to pose a variety of security related questions with respect to and over arbitrary instances of complex workflow models.

Future work will focus on a more structured and in depth analysis of possible security / safety properties that need to be maintained as well as the automated translation of workflow (BPEL) and organisational structure (LDAP) models into a specification fit for model-checking.

8 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.

44. Schaad, A., Sohr, K., Lotz, V. *Model-checking Separation of Duty properties in a Loan Origination Workflow* ACM SACMAT 2006, Lake Tahoe
45. Holzmann, G.-J: *The Model Checker SPIN*. IEEE Trans. Software Eng. 23(5): 279-295 (1997)
46. Kang, M.H., J.S. Park, J.N. Froscher: *Access control mechanisms for inter-organizational workflow*. ACM SACMAT2001:66-74