A Workflow-based Model-checking Approach to Inter- and Intra-analysis of Organisational Controls in Service-oriented Business Processes

Andreas Schaad[1], Karsten Sohr[2] and Michael Drouineaud[3]

[1]SAP Research, Security & Trust Research Group,
Vincenz-Priessnitz-Str.1, 76131 Karlsruhe, Germany
andreas.schaad@sap.com

[2]Technologie-Zentrum Informatik, Universität Bremen,
Bibliothekstr.1, 28359 Bremen, Germany
sohr@tzi.de

[3] Technologie-Zentrum Informatik, Universität Bremen,
Bibliothekstr.1, 28359 Bremen, Germany
mdruid@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. complex separation of duty requirements but at the same time support for discretionary delegation of business process tasks and rights).

To aid in the analysis of safety problems in business process-based ERP systems, this paper presents a model-checking based approach [5]. This is done in the context of a real-world banking business process requiring static and dynamic separation of duty properties as well as delegation of task instances.

This paper extends earlier work we have done in this area [6]. In particular, we now present a more structured analysis approach covering the inter- and intra-relationships between business process definitions and their instances. A fundamental notion is that of a business object which may be manipulated by one or several principals over multiple business process definitions and instances. This, in combination with the ability to delegate and revoke tasks, the possibly required rights, as well as maintaining dynamic separation of duty properties cannot be done manually any more and motivates our model-checking approach.

***Keywords*:** model-checking, access control, workflow, separation of duties, organisational control

1. Introduction

Regarding the current trend towards the development of service-oriented architectures, we observe that the notion of re-usable and composed services as well as defined repositories of business objects will eventually allow an application expert to identify business processes at their clients, identify the underlying assets and choose, compose and map the required services and business objects to eventually realize the functionality demanded by a customer.

While this greatly reduces the time to market as well as emphasizes re-use of tested services and standardized business objects this may also generate new problems with respect to access control.

Resources to which access ultimately needs to be controlled are now part of a defined and fixed business object repository (such as accounts payable or customer master data). These business objects are manipulated by a fixed set of defined services, access to which may in turn be controlled through policy decision and policy enforcement points at various layers in a system (such as the business process or application layer).

Without automated support it may, however, become difficult to understand to which business objects a chosen service may actually provide access due to its application independence. In the past during the development of standard 3-tier business system (workflow, application, database) an engineer eventually had to do all mappings of applications, roles, access rights and database tables by hand. Though this was very costly and laborious, a business object (database table) could only be accessed through a purpose-specific designed program.

In the times of service re-use and service composition, it may well be the case that a service in the context of some business process provides access to a business object as intended by an application designer and also to the same business object which has, however, been instantiated in the context of a completely different business process. This is in fact only a very small example in a far greater research domain we now see emerging regarding vulnerabilities induced by service composition and re-use.

To further define this research area we now present a structured analysis approach covering the inter- and intra-relationships between business process definitions (models) and their instances. The fundamental notion is that of a business object at modeling time and its instances which may be manipulated at run-time by one or several principals over multiple business process definitions and instances. This complex setup in combination with the ability to delegate and revoke tasks, the possibly required rights, as well as maintaining dynamic separation of duty properties motivates our model-checking approach.

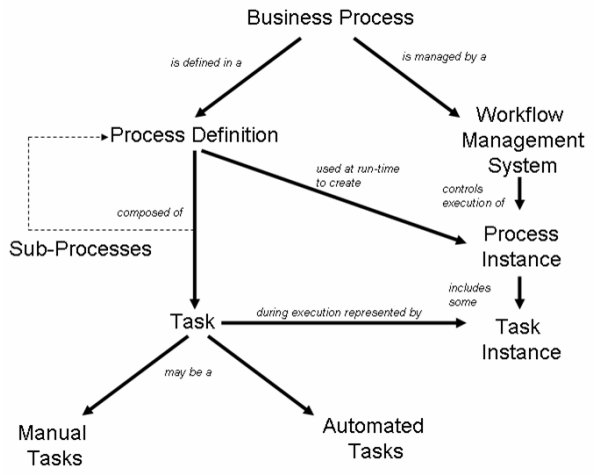

Figure 1: A business process taxonomy

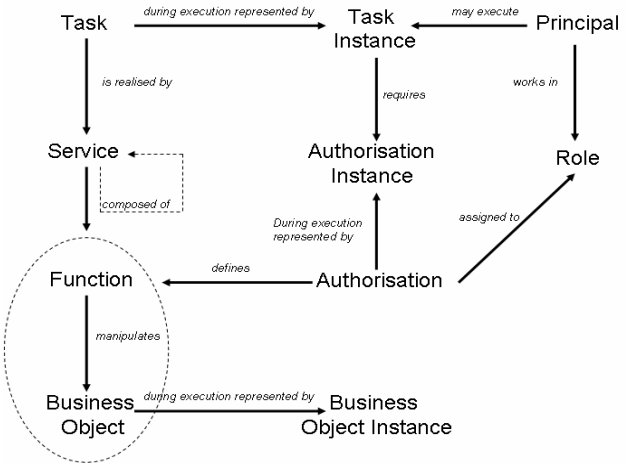

Figure 2: A basic conceptual task and role model

The rest of the paper will provide some more required background information on business processes, separation of duties as well as delegation and revocation of tasks and rights (Section 2). We intentionally work on the basis of a very basic conceptual business process model (Sections 2.1 and 2.2) and limited subset of separation of duty and delegation properties (Sections 2.3 and 2.4). Section 3 instantiates such properties within the context of two real-world financial workflows, namely a loan origination workflow in Section 3.1 and a life insurance process in Section 3.2. We provide a structured analysis approach looking at inter- and intra-relationships between workflow models and their instances as part of Section 3.3. Section 4 provides some details into our corresponding NuSMV and LTL specifications, in particular the definition of the delegation and revocation functions, some selected SoD properties, and the required resource model that provides the link between tasks, required rights and the business object instances that will eventually be manipulated when performing a task. We then discuss some results of our analysis in Section 5, specifically, the analysis of intra- and inter-relationships of process definitions and their instances. We also describe a macro mechanism which allows us to reuse LTL specifications of frequently recurring dynamic separation of duty properties. We provide some conclusions and future research directions in Section 6.

2. REQUIRED BACKGROUND

2.1 Business Processes

Business systems are often process oriented. A process definition, a representation of what should happen, is created, and it typically comprises some sub-processes until the required level of refinement is reached.

A business process might be to ship goods for a customer or to grant a new loan. In the first case this process would be split into sub processes such as reviewing a shipment order for completeness, checking custom regulations, booking space on a ship, insuring the freight and eventually shipping it. Each process and sub-process eventually comprises some task. A task is a single logical step in the process. Making a payment or not making a payment is a task. It is sometimes not feasible to automate all tasks during a single project.

Though a process definition will describe all tasks whether they are automatic or manual, a workflow management system may only executed a subset of these in an automated fashion. When a process definition has been developed, tested and deployed, instances of that process flow through the system. The process definition describes how a loan is generally provided by a bank. The instance is that of actually processing a loan for a specific client. The instance of a process comprises task instances that will include work items that are passed to an individual (or workflow participant, or user) for action, or to another process for action. So, a clerk might receive all documents relating to a loan application, and he may be asked to approve the loan.

2.2 A basic conceptual task and role model

Based on the business process related terminology it needs to be discussed how performing a task is eventually related to accessing objects. We therefore include the notion of a service as tasks may be regarded as an abstraction for some specific functionality to be performed in an application context. A process definition and its tasks reference one or more services, which in turn may be composed of a set of other services. Services do provide some functionality through a defined API or web service interface which eventually results in the manipulation of a business object instance. In line with the RBAC [7] model, authorizations do refer to a function / object relationship, allowing a principal acting in some role to invoke the functions of a service. When a process instance and its task instances are created at execution time, those tasks instances requiring human interaction are presented to a principal through a work list. However, neither core RBAC nor the standards like WfmC [8] do further detail the relationship of a task instance to an authorization. We assume that the concept of an authorization instance is required to complement the distinction between task definition at process modeling time and task instance at process execution time. For example, the meaningful delegation of a task instance to a principal without the required authority supports our assumption as we will later show. As the scope of this paper is on the use of model checking techniques for constraint analysis we do not further analyze and compare relevant prior work on workflows and (role-based) access control, e.g [9, 10, 11].

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 instance(s) 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 instance(s).

Figure 3: Separation Taxonomy

2.3 SoD - General introduction and overview

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 authorizing an order, recording the arrival of the item, recording the arrival of the invoice and finally authorizing 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.

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. [7, 12, 13, 14, 15, 16, 17, 18] and distributed systems management, e.g. has led to the definition of taxonomies and frameworks, that can, however not be reviewed in the course of this section. We point to [21] for a more detailed summary and comparison of existing taxonomies and formal definitions.

Within the scope of this paper we will base our later formal specification and analysis on the separation taxonomy as part of Figure 3. Separation properties can be expressed and enforced at different technical layers. What is, however, commonly required as contextual information is the following:

1. A notion of roles and maintenance of exclusive relationships.
2. A notion of authorizations and maintenance of exclusive relationships.
3. A reference to a business object (as well as unique instance identification at runtime).
4. A notion of a business process execution engine which may provide the following data:
   a) Information about a business process definition (model)
   b) Information about the execution path within an instance of the process definition
   c) Information about the time of manipulation of a business object instance
   d) 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.

As already informally discussed in sections 2.1 and 2.2, our task and role model provides the necessary relationships for supporting items 1 – 4 while our later specification supports item 5.

2.4 Delegation and revocation

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

In [21] we have provided formal models for the delegation of tasks (obligations) and the required rights (authorizations), based on the conceptual models provided in [15, 16, 17]. We introduced the concepts of review and supervision [22] as obligations on delegated general and specific obligations (tasks at the workflow model level and specific task instances). The formalization in a predicate logic also showed that the delegation of authorizations, 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 authorization 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 modeling such information is essential for providing revocation functionality as we will later show in our LTL specification.

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 task or authorization 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 [27]. In this framework different revocation schemes for delegated access rights are classified against the dimensions of propagation, dominance and resilience.

Role	Service	Function	Task	Business Object
Clerk Preprocessor	Customer Relationship Management	query () update () commit()	Input Customer Data	Customer Data
Clerk Preprocessor	Customer Relationship Management	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	Loan Product Bundle
Clerk Postprocessor	Pricing Engine	modify () commit <100k	Price Bundled Product	Loan Product Bundle
Supervisor	Pricing Engine	commit >100k	Price Bundled Product	Loan 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 1: Assignments of rights, roles and tasks in the loan origination workflow

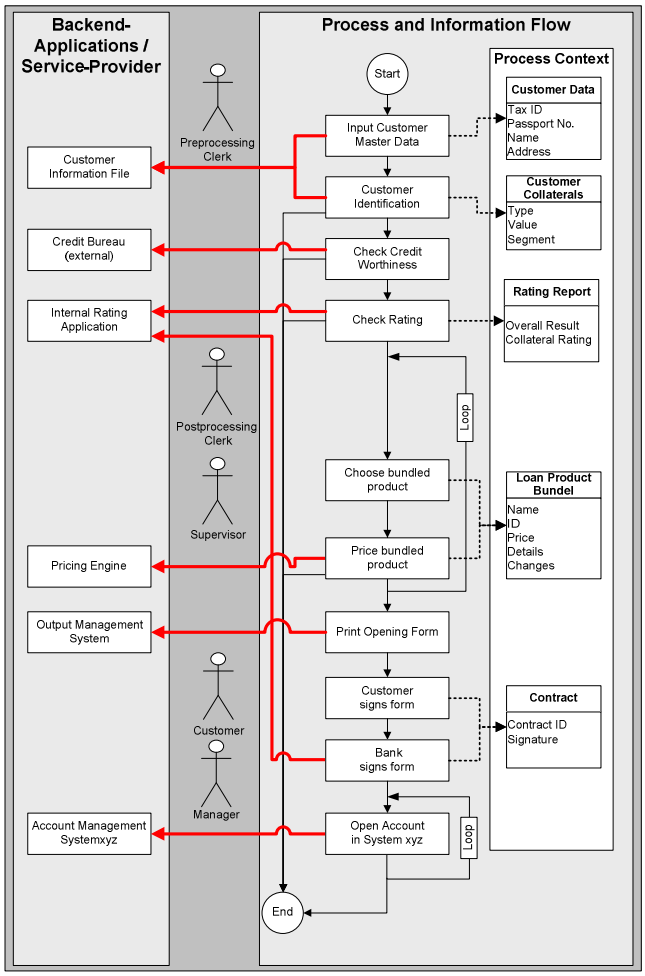

Figure 4: Loan Origination process

3. SoD, DELEGATION AND REVOCATION IN FINANCIAL PROCESSES

3.1 A loan origination process

Figure 4 shows a typical loan origination process in the banking domain, similar to that described in [22]. The supporting Table 1 summarizes some of the required roles, the general service, the functions and associated tasks and business objects as modeled 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.

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 and services (e.g. ABAP transactions) and are mapped to system-level guided procedures and rules.

Table 2 lists some basic separation of duty properties we would like to maintain with the context of the loan origination workflow. An example of a simple static separation of duty property would be that no single person may be assigned to the two exclusive roles pre/post processor. A more relaxed dynamic separation of duty property requiring run-time evaluation of a process instance would be that a person may be assigned to the two exclusive roles pre/post processor but must not activate them. A more complex object-based separation property requiring the inspection of involved business objects may be that a clerk may only price bundled product if he did not perform the operation modify () with respect to a specific offer.

Delegation may happen at various stages in the process. For example, a preprocessing clerk may delegate the task of customer identification to another clerk acting in the preprocessor role as well. This would be a simple example to as no delegation of authorization would be required due to equal roles. A more complex case would be that of a post-processing clerk to delegate the task of doing the credit bureau rating to a pre-processor (perhaps due to lack of time and to many pending credit checks). This would require the delegation of an authorisation and authorisation instance for that specific task instance.

Customer tailored Process Product Bundling		
Possible SoD property	***Type (as defined in Fig. 3)***	***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 (execute commit()) a bundled loan 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).	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
Any clerk (pre/post) may only work with some customer data in a loan origination process if he is not working with the same file in an insurance process.	Inter process ObjSoD	The required contextual information is that of another process definition regarding any insurance workflow as well as monitoring access to a customer’s file through such processes. Such a requirement is driven by the danger that there is a clerk selling insurances to a customer ignoring certain obligations of the client regarding the payment of a credit.
Any commit() of a pricing engine operation after a modify() one the same business object must be performed by different clerks	Inter process ObjSoD	This is the same requirements as in a single or multiple process instances of the same definitions but must now also take all other process definitions into consideration where pricing services are used.

Table 2: SoD properties in a loan origination process

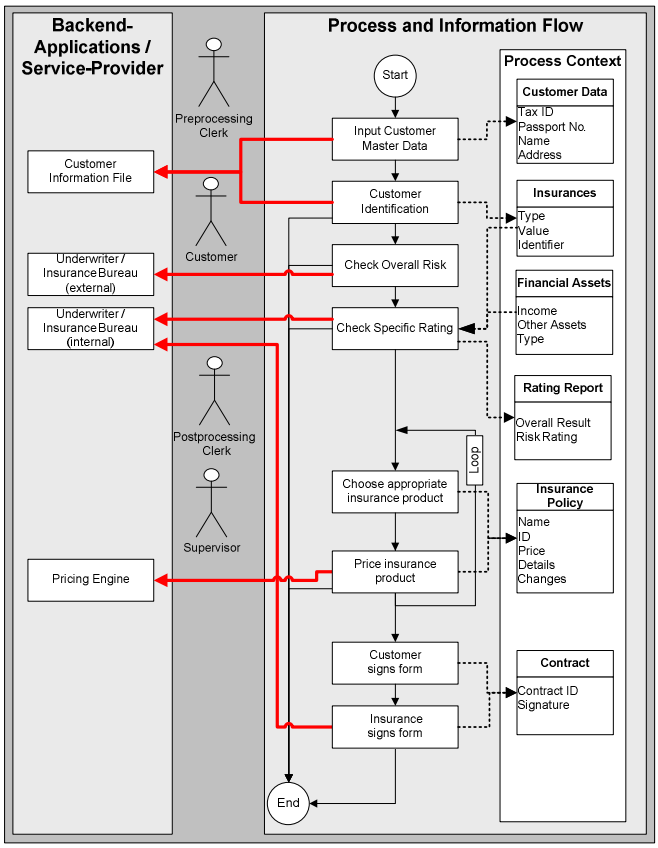

Figure 5: Insurance process

3.2 An insurance process

Figure 5 details a typical life insurance process, in fact very similar from its process flow to the loan origination process. We will need this process to later demonstrate the potential access of a business object over different process definitions and their instances. This may happen due to reuse of a service and its offered functions for realizing a task. The two examples we will later focus on are those of the customer relationship management service and the pricing engine service.

Both are abstracted examples of a service that may be defined as an independent component and deployed in the context of different workflow process definitions. Note that we chose these simplistic examples to demonstrate the problem of inter- and intra workflow dependencies where a reused service may potentially be used to violate existing separation properties. Current trends in service oriented driven design and deployment of business systems are likely to reveal far less obvious dependencies.

The basic process describes that of issuing a life insurance, where a customer asks for insurance and a clerk performs all the necessary checks regarding external underwriters and the internal risk rating offices. For doing so he will need access to a customer’s master data file as well as his financial assets required, for example, if the customer has a bank account with the same company.

The required roles and services are part of Table 3 which, however, only shows those associations as required in our later analysis.

Role	Service	Function	Task	Business Object
Clerk Preprocessor	Customer Relationship Management	query () update () commit()	Input Customer Data	Customer Data
Clerk Postprocessor	Pricing Engine	modify () commit <100k	Price Bundled Product	Product Bundle
Supervisor	Pricing Engine	commit >100k	Price Bundled Product	Product Bundle
…	…	…	…	….

Table 3: (Shortened) Assignments of rights, roles and tasks in the life insurance issuing workflow

Due to recent mergers of Banks with Insurance companies (e.g. Dresdner Bank and Allianz AG in Germany), insurances and financial products like a loan may be sold together, perhaps even in the same branch and by the same clerk. In fact, our two examples are inspired by a possible scenario where a life insurance is used as a financial instrument to secure and pay a mortgage.

For simplicity reasons an example of a simple static separation of duty property would be that no single person may be assigned to the two exclusive roles pre/post processor. A more complex object-based separation property requiring the inspection of involved business objects may be that a post-processing clerk may only price (ie. that is commit()) an insurance if he did not perform the operation modify () with respect to a specific offer proposed by the system. As we will later discuss on more detail, in our formal analysis, such separation properties may not be sufficient to enforce control over the processes and involved business objects.

Despite the presence of some defined object-based separation properties for a single process definition and its instances some undesired behavior of principals may occur across process definitions and instances due to some shared services. For example, the customer relationship service may be available to a preprocessing clerk who works in a loan origination as well as insurance process. He may now exercise the functions query() and update() on a specific customer data file in the loan process while he performs the commit() on the same data file but through the insurance process. Another example may be that of a pricing engine service. Here a clerk may perform both functions modify() and commit() but each within the context of a different process definition.

Though the above example may not necessarily lead to a fraud, they are representative of the basic dilemma a clerk finds himself in when selling a life-insurance secured long-term loan to a customer. The clerk's commission for the loan will be based on a high interest to be paid by the customer, while his commission for the life insurance will be based on a low interest to be paid by the bank.

3.3 Analysis Criteria

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.

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 sx (with an assignment of access rights and tasks) a defined state sy can be reached.

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?)

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 authorization 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 summarized 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.

	Process Definition		Process Instance		Separation of Duties				
					SSoD	SDSoD	ObjSoD	OpSoD	HistSoD
Layer	1	n..	1	n..					
L1	Y	N	N	N	X	/	/	/	/
L2	Y	N	Y	N	/	X	X	X	X
L3	Y	N	N	Y	/	X	X	X	X
L4	N	Y	N	N	X	/	/	/	/
L5	N	Y	N	Y	/	X	X	X	X
L6	N	Y	Y	N	not realistic				

Table 4: Inter- and Intra workflow security analysis layers

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 [23].

Table 4 identifies 5 layers of analysis we perform with respect to existing separation of duty properties based on the possible combinations of one or multiple process definitions and instances in combination with the one static and four dynamic separation of duty properties defined in Figure 3. In fact, as we will later show, current specifications of, for example, object-based separation of duty need to be adopted with respect to layers L3 and L5.

4 Model checking workflow access control policies

Often, dynamic access requirements exist in the context of workflows. For example, various kinds of dynamic SoD properties as those described in the context of the loan origination workflow are to be satisfied. In addition, the access rights available to a user may change over time due to delegation and revocation.

Since workflows (for example, due to loops and branches) can be quite complex, an automated analysis of workflows against such dynamic access control properties is desirable. For example, the question arises whether a particular workflow instance satisfies dynamic SoD properties or certain access rights leak to unauthorized users. Specifically, due to delegation and revocation, undesirable access control properties may arise such as the violation of dynamic SoD. Model checking tools may give the policy designer the opportunity to detect such undesirable properties in order to change the access control policy appropriately.

In particular, we can regard a workflow instance as a reactive system, which must satisfy various dynamic access control requirements. Due to the fact that model checkers are well-suited to analyzing reactive systems, we use the widely-used model checker NuSMV [24] for the analysis of access control policies for workflows. For this purpose, workflows are represented in NuSMV's input language (as so-called Kripke models).

Before demonstrating how to express workflows in the NuSMV specification language, a formal representation of the conceptual task and role model for workflows is given. This serves as the basis for the NuSMV specification.

4.1 A formal representation of the conceptual task and role model

This section defines resp. explains the notions "workflow process definition" and "workflow instance". Furthermore, it is discussed how these two notions relate to each other.

Let us start with a short introduction of some basic notions. Operations resp. functions[1] are defined on a (finite) set of general objects. A general object represents a certain business object, e.g., a credit. An object instance is then a specific instance of a (general) business object, e.g., a credit for an individual customer. An operation may only be applied to an object instance, if it is an instance of a general object on which the operation is defined.

A workflow process definition on operation level is basically a finite Kripke frame [25] with a unique initial point and a unique end point of a (first-order) modal logic[2]. The Kripke frame determines the possible sequence(s) of operations from the initial to the final point. The Kripke frame demands the application of certain operations resp. methods to certain general objects.

Operations can be assembled to services. Services are collections of operations. Generally speaking, a task then is an obligation for an assigned user, which requires (partial) access to the capabilities of certain services, and possibly includes some directives concerning the sequence of services resp. operations to be used in certain situations. From the perspective of workflow process definition on operation level, a task (in terms of Kripke semantics) has the same structure as the complete workflow.

On the other hand, we can describe a workflow process definition in terms of possible task sequences, regarding tasks as atomic units. This would correspond to a Kripke frame of (first-order) modal logic for tasks. We call this a workflow process definition on task level.

For the sake of simplicity, we assume here that there are no regulations concerning the sequence of services resp. operations within a task. Hence, the workflow process definition on task level contains in principle the same information as the workflow definition on operation level,

[1] From now on operations and functions shall be regarded as synonyms.

[2] For reasons of simplicity, one may assume that this Kripke frame corresponds to a finite set of closed formulae.

i.e., the Kripke semantics for a single task would be trivial.

Successful completion of a workflow means fundamentally processing one or several object instances such as a loan for an individual customer from the initial point to the final point of the process definition such that the requirements of every task in the actual sequence of tasks are met.

So besides the workflow process definition there are workflow instances representing the possible realizations of the process definition. If a workflow process definition corresponds to a Kripke frame, then a complete workflow instance is the equivalent of a linear ordered sequence starting at the initial point and ending at the final point in a Kripke model (specific kind of a finite state machine). However, model checking allows one to examine the Kripke model as a whole.

A task instance consists of a task resp. task identifier and a list of object instances (being instances of appropriate general objects). A permission is a pair [operation, (general) object], a permission instance is a permission containing only an object instance. An authorization is a triple [user, operation, (general) object]. Basically such authorizations are derived from permissions via the roles a user is assigned to. An authorization containing an object instance is called authorization instance. The reader may notice that a general object may be a composite of simpler units (e.g., a list).

In this paper, we will use a first-order linear temporal logic (LTL) [25] as a formalism. Sentences are the usual first-order sentences built from equations, predicate applications, logical connectives and the quantifiers ∀, ∃. Additionally, we have the modalities `G` (always in the future), `F` (sometimes in the future), `X` (in the next step). The corresponding past modalities are `H` (historically), `O` (once), `Y` (in the preceding step).

In this formalism, authorizations resp. authorization instances are represented by the `Auth` predicate. The `Exec` predicate indicates the actual application of an operation to an object instance by an individual user for the accordant triple [user, operation, object instance]. Considering a complete workflow instance (given as a linear ordered sequence of tasks resp. operations applied to appropriate object instances), one concludes that the `Exec` predicate must be true for the pertinent triples. This is no surprise since a complete workflow instance is equivalent to a successful realization of a workflow.

We demand that no user can apply an operation to an object instance, unless he is authorized to do so, i.e., `Auth` is true for the concerning triple. In our formalism, this can be expressed as the following rule assuming that the modalities `H` and `G` include also the present point of time:

```
∀ u,op,object_instance:
  (H(Exec(u,op,object_instance) ->
     Auth(u,op,object_instance))) &
  (G(Exec(u,op,object_instance) ->
     Auth(u,op,object_instance))).
```

Of course `u` is meant to be a user, `op` stands for an operation, the term `object_instance` should be self-explaining. Naturally, an authorization for a general object implies the authorization for all object instances of the general object in question. It easily follows that for an arbitrary user `u` the possibilities of applying a given operation `op` to an object instance `object_instance` are determined by the truth values of the predicate `auth`.

Set	Meaning
U	Users/Principals
R	Roles
Op	Operations
Obji	Object instances
Obj	(General) Objects
S	Services
T	Task
Ti = TaskID × List of Obji	Task instances

Predicate	Meaning	Domain
UA	User assignment	U × R
UT	User task instance assignment	U × Ti
PA	Permission assignment	Op × Obj × R Op × Obji × R
Auth	Authorization	U × Op × Obj U × Op × Obji
Exec	Execution of operations	U × Op × Obji
Active_for	Activated role	U × R
IsInstanceOf	Object instance belongs to general object	Obji × Obj
ExecDelTi	Task instance delegation	U × Ti × U
ExecDel	Delegation of an authorization instance	U x Op x Obji × U
ExecRevTi	Task instance revocation	U × Ti × U
ExecRev	Revocation of an authorization instance	U x Op x Oi × U

Table 5: Overview of the sets and predicates

The `exec` predicate makes no sense in the context of workflow process definitions. On the other hand, the `Auth` predicate does. If we consider a workflow instance as a *dynamic* process, then the involved user(s) would "see" the `Exec` predicate true in the past for the already applied operations w.r.t the accordant triples and the remaining possibilities to continue the workflow instance would be indicated by the value true of the auth predicate at the present point of time. From this point of view, the duty of a WfMS is to calculate the truth values of the `auth` predicate for all required triples at runtime and to enforce the above rule.

In addition, we define predicates for delegation and revocation, namely, `ExecDelTi`, `ExecDel`, `ExecRevTi`, and `ExecRev` (cf.[6]). We support both delegation of task instances and delegation of single authorizations at the moment. Delegation of task instances only means that the obligation of performing the task instance in question is delegated. A user may obtain a new entry for that task instance in his work list. In contrast, the delegation of authorization instances is required in order to make available the appropriate authorizations for executing task instances. After a successful task instance delegation step, the relation `UT` is changed appropriately; in case of the delegation of single authorization instances, the `auth` relation is changed.

As a summary, the sets and predicates involved in our formal role and task model is given in Table 5.

4.2 Modelling workflows in NuSMV

NuSMV [24] is a symbolic model checker, which is an extension of McMillan's SMV system [26]. A Kripke model can be specified by an intuitive input language. Since Kripke models are finite, the only data types are finite ones, namely, Booleans, scalars, and fixed arrays.

NuSMV variable	Meaning
`step`	Represents the current workflow step
`valuegt3000, …`	Control variables for the workflow
`UA_u_r`	UA-related variables
`PA_op_o_r, PA_op_oi,r`	PA-related variables
`activefor_u_r`	User u activates role `r`
`auth_u_op_oi`	User `u` possesses the authorization instance on `op` and `oi`
`exec_u_op_oi`	User `u` executes `op` on object instance `oi`
`execdel_u1_ti_u2`	Execution of a delegation step w.r.t. task instances
`execrev_u1_ti_u2`	Execution of a revocation step w.r.t task instances
`execdel_u1_op_oi_u2`	Execution of a delegation step w.r.t. authorization instances
`execrev_u1_op_oi_u2`	Execution of a revocation step w.r.t authorization instances

Table 6: The NuSMV variables and their meaning

```
step=s4 & exec_u2_prepare_ratingreport1
& exec_u2_release_ratingreport1
& exec_u2_post_ratingreport1:s5;

step=s5 & exec_u2_query_ratingreport1:s6;

step=s6
& exec_u2_queryavailableproducts_productbundle1
& !greater100k:s7;
step=s6
& exec_u2_queryavailableproducts_productbundle1
& greater100k:s8;
...
```

Figure 6: Excerpt NuSMV specification of the workflow

Reusable components can be specified by modules. The primary purpose of NuSMV's input language is to describe the transition relation of the Kripke model 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. Using the `init` function, we can also define initial values for state variables.

Due to the fact that workflows may include branches and loops we model workflows (workflow instances) directly as Kripke models. For this purpose, we introduce a certain scalar state variable `step` with values `s1,…,sn`. This variable indicates the current workflow step to be performed. The following state variables have been introduced in order to describe the role-based resource model for workflows/workflow instances:

- For each user-role assignment, a variable `UA_u_r` is 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_op_o_r` is true iff `PA(op,o,r)` is true. Note that `o` is a (general) business object according to our terminology from Figure 2. If required, we can additionally define state variables `PA_op_oi_r` which take object instances into consideration.
- For each authorization instance belonging to a user `u`, we create a variable `auth_u_op_oi`. An authorization instance contains an object instance and not a general business object.
- For each role activation `active_for(u,r)`, we define a state variable `activefor_u_r`.
- We express the fact that user `u` performs operation `op` on the object instance `oi` with a state variable `exec_u_op_oi`.

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.

In order to obtain a better overview, a summary of the relevant NuSMV variables and their meaning is given in Table 6. To obtain a better understanding of the resulting Kripke structure, we give an excerpt of the loan origination workflow in Figure 6 showing how the steps 3 to 5 from Figure 4 have been mapped to the Kripke structure. Note that the workflow is in this example specification represented by the executions of operations and not by task executions (which could also have been done).

4.3 Modelling delegation and revocation in SMV

We model delegation and revocation as discussed in Section 2.4 and can handle the delegation of task instances (task delegation) and single authorization instances. Similarly to the `exec_u_op_oi` state variable, we introduce the variables `execdel_u1_ti_u2` and `execrev_u1_ti_u2` to express the predicates for the delegation of task instances in NuSMV.

For the delegation and revocation of authorization instances, we introduce the NuSMV state variables `execdel_u1_op_oi_u2` and `execrev_u1_op_oi_u2`. Note that `u2` is then authorized to execute the operation `op` on the object instance `oi` after a successful delegation step. This can be expressed in the Kripke model the following way[3]:

```
next(auth_u2_op_oi):=
case
    execdel_u1_op_oi_u2:1;
  ...
esac;
```

We can now check with the help of NuSMV if certain SoD properties are violated by delegation and revocation steps or if a user possesses the appropriate authorizations to execute the task instance. The former case will be explained in Section 5.2. Owing to the fact that through the delegation and revocation predicates principals/users can obtain

[3] NuSMV keywords are indicated in **bold font**.

authorization instances at run-time we have to deal with dynamic access control policies. This further motivates using propositional LTL and model checkers.

4.4 Specifying SoD properties in LTL

Having outlined the Kripke model for our access control policy for workflows, we now demonstrate how SoD properties can be specified in LTL. The Kripke model describing the access control policy of the workflow can then be checked against these properties. In the following, we present two examples of SoD properties.

Simple Static SoD (SSSoD)

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

```
G(!(UA_u_clerkpreproc &
                  UA_u_clerkpostproc)).
```

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).
```

5 Analysis

We now present some of the results of our model checking-based approach to the analysis of access control policies for workflows. Specifically, we demonstrate how SoD properties can be checked taking into consideration the different layers mentioned in Table 4. We then analyze the access control policies in the context of delegation. At the end of this section, we also introduce a schema concept. This way, one can define macros for SoD properties and instantiate the macros when needed in a concrete workflow.

As explained in [6], the basic idea of our model-checking approach is as follows: The workflow access control policies (e.g., user-role assignments, user-task assignments), the workflow instance itself and the delegation and revocation steps are specified by means of a Kripke model (cf. Section 4.1), and the properties to be checked such as SDSoD are specified in propositional LTL. NuSMV then verifies whether the properties hold for the Kripke model in question or not.

5.1 Different layers of SoD property analysis

In this section, we discuss the analysis of SoD properties w.r.t. the layers introduced in Table 4. Both example properties mentioned in Section 4.4 can be regarded as examples for an L1 (SSoD) and L2 (SDSoD) analysis. We first give a further example for the analysis on layer L2, namely, to check an ObjDSoD property. Thereafter, we also give examples of an L3 and L5 analysis.

Layer 2: ObjDSoD

Generally, we have to consider the query: Given a single process definition and a single instance is it possible that a user can access the same object instance `o1` through acting in `r1` and `r2`?

Considering our loan origination workflow, we could introduce an ObjDSoD 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 ObjDSoD the activation of both the roles is not forbidden. On the other hand, no user/principal is permitted to execute the `modify_productbundle1` access right in the `Clerk Postprocessor` role and the `commit_productbundle1` access right in the `Supervisor` role. In LTL, we can formulate this the following way:

```
G (UA_u_clerkpostproc & UA_u_supervisor
  -> G(exec_u_commit_productbundle1
    ->!O exec_u_modify_productbundle1)).
```

Note that `productbundle1` is an instance of the business object `product bundle`.

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.

Layer 3: Multi_Instance_ObjDSoD

We can also carry out an inter-instance analysis of workflows by means of NuSMV. For this purpose, we use the `process` statement provided by NuSMV. Processes are used to model interleaving concurrency. A process is a NuSMV module which is instantiated using the keyword `process`.

A NuSMV module declaration is an encapsulated collection of declarations and specifications. A module declaration also opens a new identifier scope. Once defined, a module can be reused as many times as necessary. Modules are used in such a way that each instance of a module refers to different data structures. Furthermore, a module can be called with parameters. The actual parameters can be regarded as data of a higher level scope which are handed over to the module in question.

```
Module main
VAR
LoanOriginationWF1 process LoanOriginationWF(…);
LoanOriginationWF2 process LoanOriginationWF(…);
…

Module LoanOriginationWF(…)
…   -- Workflow specification as displayed
    -- in Figure 6
```

```
Module main
…
LoanOriginationWF1 process LoanOriginationWF(…);
InsuranceWF1 process InsuranceWF(…);
…
Module LoanOriginationWF(…)
…
Module InsuranceWF(…)
…
```

Figure 7: Excerpt of the NuSMV specification for workflow definitions and process instantiations

The program now executes a step by non-deterministically selecting a process, then executing all of the assignment statements (such as next statements) in that process in parallel. It is implicit that if a given variable is not assigned by the process, then its value remains unchanged.

In the upper part of Figure 7, we give an example module as an excerpt (LoanOriginationWF). For the sake of simplicity, we have left out the concrete workflow specification in this figure. One can imagine that the module represents the NuSMV specification of our loan origination workflow. Furthermore, we have two process calls, i.e., module instantiations with the `process` statement. The two calls of the `LoanOriginationWF` module correspond to two workflow instances of our "Loan Origination" workflow. These workflow instances are regarded as running concurrently. The "…" within the parentheses stands for parameters, which are handed over to the processes, such as the `exec_u_op_oi` and `auth_u_op_oi` state variables.

For an L3 analysis of ObjDSoD we have to consider the following query: Given several instances of a single process definition can an object instance be accessed by a user acting in two exclusive roles? This means: Is it possible that a user can access an object instance `o1` through acting in role `r1` in instance `i1` and `r2` in instance `i2`?

Regarding the loan origination workflow, we again can check the ObjDSoD property mentioned above – but now the LTL specification is applied to two *different* instances of the *same* workflow definition. This way, the model checker detects if a user performs the `modify` function on `productbundle1` in one workflow instance and then commits this product bundle in another workflow instance.

Layer 5: Multi_Process_Multi_Instance_ObjDSoD

In case of an L5 analysis, we have to check the following query: Given several instances of several process definitions, can an object instance be accessed by a user acting in two exclusive roles? As an example, we now assume a secondary workflow "Insurance" as introduced in Section 3.2. Let us further assume that we have the object instance `custdata1` which is an instance of the business object `customer data`. This object instance can be manipulated by the `modify` function in the loan origination workflow (with the help of the `Customer Relationship Management` service) and can be confirmed by the `commit` function in the insurance workflow (again with the help of the `Customer Relationship Management` service).

The ObjDSoD property then states that the customer data object instance `custdata1` must not be accessed in different workflow instances of different process definitions. In particular, we consider here workflow instances of the loan origination and of the insurance workflow definitions (cf. the NuSMV specification in Figure 7, lower part). The ObjDSoD property to be checked can then be formulated in propositional LTL as follows:

```
G(UA_u_clerkpreproclo &
  UA_u_clerkpreprocin ->
    G(exec_u_crm_commit_custdata1 ->
    !O exec_u_crm_modify_custdata1)).
```

`crm` represents the `Customer Relationship Management` service shared by both workflows. In fact, this example demonstrates the influence of the service context for access decisions (in contrast to the other examples). Note that we have introduced here a new state variable `exec_u_service_op_oi` in order to express the service context. This is in accordance with our formal model for RBAC workflows introduced in Section 4.1.

5.2 Analysis of delegation

We can also analyse SoD properties in the context of delegation. In particular, this analysis can be carried out on the different layers introduced above. This will be sketched in the following.

For this analysis, let us again consider the aforementioned ObjDSoD example while carrying out an L5 analysis. In addition, we have two users `u1` and `u2`. `u1` is entitled to perform the task `Input Customer Data` in the loan origination instance and `u2` is to perform the task `Input Customer Data` in the insurance workflow instance. Now, assume that `u2` delegates her task to `u1` (including the appropriate access rights). As a consequence of this delegation, `u1` is now additionally authorized for executing the `modify` function on `custdata1` within the insurance workflow instance, i.e., `auth_u1_crm_modify_custdata1` is true after the delegation (cf. Section 4.3). Then, `exec_u1_crm_modify_custdata1` can also be true.

We can now check the ObjDSoD property in this situation and again obtain the following LTL specification:

```
G(UA_u1_clerkpreproclo  &
  UA_u1_clerkpreprocin ->
    G(exec_u1_crm_commit_custdata1 ->
    G!O exec_u1_crm_modify_custdata1)).
```

NuSMV gives a counterexample in this case because `u1` can execute both functions on `custdata1` due to the delegation.

5.3 Macro mechanism for the specification of SoD properties

The reader may have noticed that the LTL specifications for the ObjDSoD properties to be checked look similar or are even the same in the different cases of our analysis. However, the process executions (i.e., the Kripke models) are different in each case.

This observation leads us to the introduction of a schema concept for SoD constraints (or in general, for authorization constraints). Then, we can define constraint schemas that can be given parameters. Hence, these schemas would allow for a parameterization of the LTL specifications. This resembles the macro mechanism introduced in the C programming language. Let us again take ObjDSoD as an example. Let us further skip the conjunction part of the ObjDSoD constraint for reasons of simplicity. Then, we can define a macro `ObjDSoD(u,oi,op1,op2)` the following way

```
G(exec_u_op1_oi ->!O exec_u_op2_oi).
```

We can instantiate this macro with the concrete parameters as required for a concrete workflow. In the aforementioned example, we obtain `ObjDSoD(u,commit,modify,productbundle1)` which can then be expanded by a macro pre-processor to

```
G(exec_u_commit_productbundle1 ->
     !O exec_u_modify_productbundle1).
```

In the other example, we would have to consider the macro

```
ObjDSoD(u,crm_commit,
        crm_modify,custdata1).
```

The macro mechanism helps the policy designer to reuse already identified constraint types relevant to domain-specific workflows such as banking, insurance and logistics workflows. For each domain in question, one can imagine a library consisting of the macros relevant to that domain. A security officer then would not have to newly identify domain-specific authorization constraints.

6 Summary and Conclusion

This paper presented a structured analysis approach to model-checking organisational controls in workflows, covering the inter- and intra-relationships between business process definitions (models) and their instances. The fundamental notion was that of a business object at modeling time and its instances which may be manipulated at run-time by one or several principals over multiple business process definitions and instances. This complex setup in combination with the ability to delegate and revoke tasks, the possibly required rights, as well as maintaining dynamic separation of duty properties motivated our model-checking approach. This paper extended earlier work we have done in this area [6].

Technically, we extended our model checking approach in several directions. First, we gave a formal representation of our conceptual role and task model for workflows. Moreover, we used the process concept provided by NuSMV. This allows for an inter-instance analysis of dynamic SoD properties with workflow instances (both of the same process definition or even of different process definitions). Additionally, we introduced a macro mechanism which can be employed to reuse LTL specifications of SoD properties. This would now allow us to check arbitrary and service-based business process definitions such as delivered by BPEL or xPDL workflow specifications.

Future work will focus on the further refinement of our conceptual model, specifically on the notion of a service. Here we will have to introduce a notion of a service definition as well as service instance. The overall challenge here is to keep the model small enough to be made subject to automated analysis and avoid state explosion at model-checking time. Furthermore, we can perform a more in-depth analysis of service composition. This way, a kind of covert-channel analysis can be carried out. We also have not yet developed the full algorithms for revocation in the context of multiple business process (instances). We also intend to investigate other types of authorization constraints such as cardinality constraints, prerequisite roles, or temporal constraints in the sense of Bertino et al. [28]. Temporal constraints would also allow us to support certain kinds of conditional delegation in our workflow concept. In order to deal with temporal constraints, we can then employ the UPPAAL model checker [29], which has a built-in time concept.

Acknowledgment

The work of Andreas Schaad was sponsored under the BMBF-funded project “ORKA” and the European IP “R4eGov”. The work of Karsten Sohr and Michael Drouineaud was sponsored under the BMBF-funded project “ORKA” and the DFG-funded project “ForRBAC” (Contract number SO 515/2-1).

References

[1] Samarati, P. and S. Vimercati, *Access Control: Policies, 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] Clarke, E., O. Grumberg, and D. Peled, *Model Checking*. 2000: The MIT Press.

[6] Schaad, A., V. Lotz, and K. Sohr. *A model-checking approach to analysing organisational controls in a loan origination process*. In Proceedings of the 11th ACM Symposium on Access Control Models and Technologies, Lake Tahoe, CA, 2006.

[7] Sandhu, R., et al., *Role-based access control models.* IEEE Computer, 1996. 29(2): p. 38-47.
[8] Workflow Management Coalition *The workflow Reference Model*, Document Number TC00-1003 published 19th January, 1995, WfmC
[9] Bertino, E., E. Ferrari. The specification and enforcement of authorization constraints in workflow management systems. ACM Transactions on Information and Systems Security and Privacy, 2(1):65-104, Februar 1999.
[10] Atluri, V. and W. Huang, *An Authorization Model for Workflows.* Lecture Notes in Computer Science, 1996. 1146: p. 44-64.
[11] Botha, *Separation of duties for access control enforcement in workflow environments.* IBM SYSTEMS JOURNAL, 2001. 40(3).
[12] Sandhu, R. *Separation of Duties in Computerized Information Systems*. in *IFIP WG11.3 Workshop on Database Security*. 1990. Halifax, UK.
[13] Simon, R. and M. Zurko. *Separation of Duty in Role-Based Environments*. in *Computer Security Foundations Workshop X*. 1997. Rockport, Massachusetts.
[14] Sandhu, R. *Separation of Duties in Computerized Information Systems*. in *IFIP WG11.3 Workshop on Database Security*. 1990. Halifax, UK.
[15] Nash, M. and K. Poland. *Some Conundrums Concerning Separation of Duty*. in *IEEE Symposium on Security and Privacy*. 1990. Oakland, CA.
[16] Baldwin, R. *Naming and Grouping Privileges to Simplify Security Management in Large Databases*. in *IEEE Symposium on Security and Privacy*. 1990. Oakland.
[17] 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.
[18] Ferraiolo, D., J. Cugini, and D. Kuhn. *Role-Based Access Control (RBAC): Features and Motivations*. in *Computer Security Applications*. 1995.
[19] 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.
[20] 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.
[21] Schaad, A., *A Framework for Organisational Control Principles*, *PhD Thesis*, in *Department of Computer Science*. 2003, University of York.
[22] 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.
[23] Loer, K. and M. Harrison. *Towards Usable and Relevant Model Checking Techniques for the Analysis of Dependable Interactive Systems*. in *ASE*. 2002.
[24] 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.
[25] Goldblatt, R., *Logics of Time and Computation, 2nd Edition, Revised and Expanded.* CSLI Lecture Notes, 1992. 7.
[26] McMillan, K., *The SMV system, Symbolic Model Checking - an approach* 1992, Carnegie Mellon University CMU-CS-92-131.
[27] Hagstrom, A., et al. *Revocations - A Categorization*. in *Computer Security Foundations Workshop*. 2001: IEEE.
[28] Bertino E., P.A. Bonatti, E. Ferrari: TRBAC: A temporal role-based access control model. ACM Trans. Inf. Syst. Secur. 4(3): 191-233 (2001)
[29] Bengtsson, J., K.G. Larsen, F. Larsson, P. Pettersson, W. Yi. *Uppaal - a Tool Suite for Automatic Verification of Real-Time Systems.* In Proceedings of the 4*th* DIMACS Workshop on Verification and Control of Hybrid Systems, New Brunswick, New Jersey, 22-24 October, 1995.

Author Biographies

Andreas Schaad, PhD, CISSP works for SAP Research Germany in the Security and Trust group. Prior to joining SAP Germany he obtained his PhD at the University of York, UK followed by work for Ernst & Young, UK as an Information Systems auditor and SAP Labs France as a senior researcher. He is actively serving on several ACM and IEEE conference committees and his research interests include all aspects of organisational control.

Dr. Karsten Sohr works at the Center for Computing Technologies (TZI) of the Universität Bremen, Germany. Prior to joining the TZI he received his doctoral degree from the Universität Marburg, Germany. He is currently coordinator at the TZI for the research area security. His research interests include role-based access control and security of mobile applications.

Michael Drouineaud works at the Center for Computing Technologies (TZI), Universität Bremen. He has received a graduate degree of mathematics and a post graduate degree of information technology from the Swiss Federal Institute of Technology Zurich. He is currently employed within the BMBF-funded research project ORKA. His research interests include the application of mathematical methods to role-based access control (RBAC). He is specifically engaged in employing logic-based formalisms for the validation of RBAC security policies.