Model Checking Access Control in Business Workflow

Technical Report
SAP Research – CRIM (Computer Research Institute of Montreal)

Produced as extension of results of SAP-CRIM collaboration

Version finale

CRIM-06/10-11

A. Dury, CRIM
S. Boroday, CRIM
A. Petrenko, CRIM
V. Lotz, SAP

October 26, 2006

Collection scientifique et technique

ISBN-10 : 2-89522-084-0
<table>
<thead>
<tr>
<th>Table of Content</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Abstract</strong> ........................................................................................................... 5</td>
</tr>
<tr>
<td>1. <strong>Introduction</strong> ........................................................................................................ 6</td>
</tr>
<tr>
<td>2. <strong>Related Work</strong> ........................................................................................................ 7</td>
</tr>
<tr>
<td>2.1 <strong>Model Checking - an Efficient Verification Technique</strong> .............................................. 7</td>
</tr>
<tr>
<td>2.1.1 <strong>Spin – One of the Most Successful Model Checkers</strong> ............................................. 7</td>
</tr>
<tr>
<td>2.1.2 <strong>User Friendly Property Specification</strong> .................................................................... 8</td>
</tr>
<tr>
<td>2.1.3 <strong>Abstraction Techniques</strong> ..................................................................................... 9</td>
</tr>
<tr>
<td>2.1.4 <strong>Fairness Constraints</strong> .......................................................................................... 12</td>
</tr>
<tr>
<td>2.2 <strong>Workflow Model Checking</strong> .................................................................................... 13</td>
</tr>
<tr>
<td>2.3 <strong>RBAC Abstraction and Model Checking</strong> .................................................................. 15</td>
</tr>
<tr>
<td>3. <strong>An Approach to the Verification of Business Workflows against RBAC</strong> ....................... 18</td>
</tr>
<tr>
<td>3.1 <strong>Business Workflow and RBAC</strong> ............................................................................... 18</td>
</tr>
<tr>
<td>3.1.1 <strong>Business Workflow Model</strong> .................................................................................. 18</td>
</tr>
<tr>
<td>3.1.2 <strong>RBAC Roles</strong> ...................................................................................................... 20</td>
</tr>
<tr>
<td>3.1.3 <strong>RBAC Rules and Constraints for Role Activation</strong> .................................................... 20</td>
</tr>
<tr>
<td>3.1.4 <strong>Safety Properties</strong> ............................................................................................... 22</td>
</tr>
<tr>
<td>3.2 <strong>“Procure to Stock” Business Workflow</strong> .................................................................. 22</td>
</tr>
<tr>
<td>3.2.1 <strong>Business Workflow Description</strong> .......................................................................... 22</td>
</tr>
<tr>
<td>3.2.2 <strong>Roles and Role Assignment</strong> ................................................................................ 24</td>
</tr>
<tr>
<td>3.2.3 <strong>Abstracted Business Workflow</strong> .......................................................................... 25</td>
</tr>
<tr>
<td>3.3 <strong>Naive Model of Business Workflow and RBAC</strong> ....................................................... 26</td>
</tr>
<tr>
<td>3.4 <strong>Multi EFSM Model</strong> ............................................................................................... 28</td>
</tr>
<tr>
<td>3.4.1 <strong>Business Workflow Transformation</strong> .................................................................... 28</td>
</tr>
<tr>
<td>3.4.2 <strong>RBAC Constraints and User Modeling</strong> ................................................................ 29</td>
</tr>
<tr>
<td>3.4.3 <strong>User Choice and Synchronization</strong> ...................................................................... 31</td>
</tr>
<tr>
<td>3.5 <strong>Properties</strong> ........................................................................................................... 32</td>
</tr>
<tr>
<td>3.6 <strong>State Space Explosion</strong> .......................................................................................... 33</td>
</tr>
<tr>
<td>4. <strong>Experiments</strong> ........................................................................................................... 33</td>
</tr>
<tr>
<td>4.1 <strong>Setup</strong> .................................................................................................................... 34</td>
</tr>
<tr>
<td>4.2 <strong>Reachability Property</strong> ............................................................................................ 34</td>
</tr>
<tr>
<td>4.2.1 <strong>Property Formulation</strong> ........................................................................................ 34</td>
</tr>
<tr>
<td>4.2.2 <strong>Results</strong> ............................................................................................................. 35</td>
</tr>
<tr>
<td>4.3 <strong>Separation of Duties Property</strong> ................................................................................ 35</td>
</tr>
<tr>
<td>4.3.1 <strong>Property Formulation</strong> ........................................................................................ 36</td>
</tr>
<tr>
<td>4.3.2 <strong>Results</strong> ............................................................................................................. 36</td>
</tr>
<tr>
<td>5. <strong>Alleviating State Explosion</strong> .................................................................................... 37</td>
</tr>
<tr>
<td>5.1 <strong>Data Encoding</strong> ...................................................................................................... 38</td>
</tr>
<tr>
<td>5.1.1 <strong>Experimental Evaluation</strong> .................................................................................... 39</td>
</tr>
<tr>
<td>5.1.2 <strong>Discussion</strong> ....................................................................................................... 40</td>
</tr>
<tr>
<td>5.2 <strong>Property Reformulation</strong> ........................................................................................ 40</td>
</tr>
<tr>
<td>5.2.1 <strong>Results</strong> ............................................................................................................. 41</td>
</tr>
</tbody>
</table>
ABSTRACT

While RBAC is an increasingly popular and efficient security solution, RBAC design and maintaining of policies is a challenging problem in a dynamic business environment. Thus, automated RBAC verification techniques are needed. In this report, we investigate the capability of one of the most successful verification technologies, model checking. Based on the analysis of the state of the art in RBAC model checking, we propose our solution to model checking RBAC that also takes into consideration a particular business workflow, addressed by the RBAC. Efficient and practical techniques are proposed to fight state explosion. Experiments show that without aggressive abstractions, RBAC model checking could be difficult for large workflows and user pools. However, with appropriate abstraction, it is still feasible for small to moderate size problems.
1. INTRODUCTION

Role Based Access Control (RBAC) is an increasingly popular and efficient security solution. While RBAC systems are able to address classified information protection, the major advantage of RBAC is its ability to constraint risk related to malicious fraudulent or erroneous user behavior, typically using a concept of separation of duties (SoD), i.e., allowing several persons required to complete a critical task, with no single person having excessive control. For decades, if not for hundreds of years, significant efforts were devoted to impose SoD in accounting, banking, military, and even in political realms (as separation of powers). However, in software engineering, the relevant experience is rather limited. Currently, RBAC design and maintenance of RBAC security policies are challenging problems as company structure, roles, user pools, business flows, internal and external (legal) security requirements are always changing. In this context, automated RBAC verification techniques can contribute both to product integrity and time to market. While early RBAC verification methodologies rely on visualization of constraints [Tidswell and Jaeger, 2000] and graph transformation [Koch and Parisi-Presicce, 2003], modern, more formal and powerful approaches are usually based on automated reasoning techniques, such as model checking.

Model checking is a verification technique that allows one to automatically check whether a property, typically expressed in a temporal logic, holds over a (typically) finite state model, derived from software, hardware, or, in our case, workflows and RBAC systems. Model checking tools proved themselves reliable and capable of detecting subtle bugs, not detected by other quality assurance methods and previously unknown security vulnerabilities.

Classic model checking deals with large, though finite, systems. While the classical “full” model checking involves a complete unfolding of a whole high level model into a low level finite state model (such as a Kripke structure), on-line (on-the-fly) verification is performed by unfolding the composition of properties with the system model and avoiding exploration of states not relevant to the property verification.

In this work, we aim at model checking of RBAC used in the context of workflows of business applications. We first present an overview of previously developed approaches to RBAC and workflow verification. We then describe our approach that is based on model checking of business workflow considered together with a RBAC. Since a major issue with model checking is usually state explosion, we pay a special attention to simplification (abstraction) methods. RBAC is often seen as an application and a domain specific security framework. In SAP systems, RBAC is used to control access to data from business applications, while access to other data, such as unstructured portal content, is controlled by a different mechanism [Zibulski, 2005]. Thus, we model RBAC in conjunction with workflow processes, in a setting derived from a real application
context. We check compatibility of RBAC with a given workflow and validate security properties against the given RBAC constraints set and workflow. Our work differs from the “light-weight” set-theoretic model checking efforts [Schaad and Moffett, 2002], [Zao et al, 2003], [Mankai and Logrippo, 2005], [Hughes and Bultan, 2004], [Park and Kwon, 2005] by considering the workflow on which the RBAC is imposed and order-dependent constraints. Unlike most work on full-scale model checking with Spin [Ahmed and Tripathi, 2003], [Hansen and Oleshchuk, 2005], [Herrmann, 2003], [Nguyen and Rathke, 2005], we elaborate our approach in details, at the same time, we elaborate several techniques which fight the state explosion problem. We provide the results of experimental evaluation and comparison of the proposed techniques.

The document is organized as follows. Section 2 explains the concepts of model checking and abstraction and discusses prior work on RBAC and business workflow model checking. Our approach to RBAC model checking is elaborated in Section 3. Section 4 illustrates the approach with a case study. Techniques for alleviating state explosion effect are presented along with their experimental evaluation in Section 5. Verification of RBAC with unbounded user pool and other extensions are discussed in Section 6. We conclude in Section 7.

2. RELATED WORK

2.1 Model Checking - an Efficient Verification Technique

2.1.1 Spin – One of the Most Successful Model Checkers

Most recent research works on RBAC model checking [Ahmed and Tripathi, 2003], [Hansen and Oleshchuk, 2005], [Herrmann, 2003], [Nguyen and Rathke, 2005], employ Spin [Holzmann, 2003]. The next popular choice is a “light-weight”\(^1\), structural model checker Alloy Analyzer [Schaad and Moffett, 2002], [Zao et al, 2003], [Mankai and Logrippo, 2005], [Hughes and Bultan, 2004], [Park and Kwon, 2005]. While results obtained with Alloy appear to be more mature, elaborated, and confirmed by experimental data, modeling of workflows is rather difficult in Alloy, which lacks dynamic features.

Spin is one of the most successful on-line model checker. It supports the PROMELA language, inspired by Dijkstra’s guarded command notation, Hoare CSP I/O operations, and SDL. A PROMELA specification defines process types and instances, which could be seen as extended automata communicating via

\(^1\) Alloy Analyzer is usually considered as a model checker in the technical literature. However, it is not a model checker in the strict (mathematical) meaning of the word, but rather a constraint solver or “model finder”.
messages and shared variables. The properties are usually expressed either as “never claim” Buchi automata or in propositional Linear Temporal Logic (LTL). LTL extends the propositional logic with operators of eventuality $\langle\rangle$, universality $[]$, until $U$, and next $X$ (note that next support requires recompilation of Spin with the corresponding option and is incompatible with some of Spin optimization techniques) [Holzmann, 2003].

Model checking difficulties are scalability (state explosion) and user friendly property specification. We discuss how these issues, and a related problem of fairness, are tackled both in general model checking literature, and in devoted RBAC verification literature. Also we briefly discuss the related problem of fairness.

### 2.1.2 User Friendly Property Specification

Model checkers usually employ a propositional temporal logic. At the same time, the first order linear temporal logic (with quantification over values) is a more convenient means to express properties, including separation of duties constraints in RBAC systems. Fortunately, for a given finite object set, a first order formula could be unfolded (grounded) into a (large) propositional formula. As a more scalable solution, model checking could be performed for propositional formulae under quantification for each possible combination of variables, and then the truth value of the first order formula could be evaluated. While the task is not trivial in the most general case, in most RBAC properties, only universal quantification is used. In this case, a workaround could consist in non-deterministic instantiation of users or objects for checking a temporal property in the designated initialization automaton. Note that properties could often be simplified by instantiating a unique representative user of a class of users with identical constraints [Ahmed and Tripathi, 2003].

Proprietary operators and “syntactic sugar”, such as used in Time Rover's Metric Temporal Logic [Drusinsky, 2000], could be helpful for specifying properties. Dwyer proposes a specification pattern repository [Dwyer et al, 1999] that embraces the most often used property patterns. The repository maps the patterns into LTL, CTL, and regular expressions. Independent mappings exist for other property specification languages, for example, into the extended automata language of Object Geode model checker [Hallal et al, 2003]. In the context of business workflow property specification, several additional patterns useful for workflow analysis are developed within TestBed tool [Janssen et al, 1999]. Others argue that patterns are more easily expressed and customized using automata, while a friendly GUI or a controlled subset of English could be used to facilitate property specification for non-experts [Smith et al, 2003]. In certain model checkers, like Spin, properties are expressed using a kind of extended finite state machines.
2.1.3 Abstraction Techniques

Probably, among most worthy RBAC specific abstraction techniques are pruning of role activations [Jha et al, 2005] and exploiting user symmetry to reduce a general property to a user specific one [Ahmed and Tripathi, 2003]. Otherwise, general abstraction techniques, described in the following sections are applicable.

Formal verification usually requires certain simplification tricks. At the same time, such tricks can be incorporated within rigorous abstraction process. Formally, abstraction is the reduction of checking \( A \models \varphi \) to a simpler \( A' \models \varphi' \). Several general theoretical frameworks have been proposed to relate properties of concrete (original) and abstracted systems. Probably, the most known one is the abstract interpretation framework, which originates in program analysis and consists in giving a non-standard (approximated) semantic for a program, usually by replacing variable domains and operations with abstract domains and corresponding abstract semantic operations [Cousot and Cousot, 1977]. Similar ideas have been developed previously as pseudo-evaluation, data flow analysis, and symbolic execution. A series of Cousot papers, starting from [Cousot and Cousot, 1977] provides a sound abstract interpretation framework rooted in partial order and lattice theory. Since the difference between control and data is rather subjective, the apparatus of abstract interpretation could be used to provide a general (not necessarily data specific) property preserving abstraction framework [Bensalem et al, 1992].

More recently, yet another program analysis approach, called program slicing [Tip, 1995], has been adapted for model checking. Adapting program abstraction techniques, a special care should be taken on message passing, non-determinism, and other features of specification languages that are not widespread in the programming languages.

Usually, an abstraction preserves a property (class of properties) in one way or another (i.e., it preserves property, property violations, or both). It is said that an abstraction preserves a property if whenever the (abstract) property holds on the abstract system the original (concrete) property holds on the original (concrete) system. While one could argue that such an abstraction preserves property violations of the original system, we adhere to the standard terminology [Bensalem et al, 1992]. An abstraction, which preserves both, property and property violation, is precise (w.r.t. to the property). Sometimes the same approach (e.g., variable bounding) could be considered as over, under, or precise abstraction depending on nuances of the process or class of properties under consideration.

For property preserving abstraction, CEGAR (Counterexample Guided Abstraction Refinement) process [Clarke et al, 2000] is applicable. It is an iterative process using abstract counterexamples not corresponding to any counterexamples in the original system to refine the abstraction. The process is
performed until either a valid counterexample of the original system is obtained or the abstract system passes the model checking process.

Probably, the most known type of abstraction is the so called “conservative” abstraction, which in a sense preserves all the executions of the original system. It preserves a large class of so-called universal properties that includes all LTL properties. A property is called universal if and only if it holds on all the executions of all the systems for which the property is true. While conservative abstraction could be used to prove universal properties, more refined abstraction types are required for non-universal properties.

Most typical conservative abstraction methods are state merging, variable and data abstraction. One popular and easy to automate conservative abstraction methodology is the “invisible” variables technique, which converts some variables into free input variables. If an abstract counterexample does not correspond to any counterexample of the original system (concrete counterexample), the number of invisible variables is refined in a CEGAR process.

The dual of conservative abstraction is restriction or reduction of some part of the specification, without adding to the behavior. Reduction of states or transitions in the underlying Kripke structure preserves reachability properties (negatives of safety properties). In certain cases, when indeed irrelevant parts are abstracted, the reduction could be precise.

Below we detail several abstraction techniques:

- **State merging.** If only states, labeled with the same atomic proposition (used in the property) are merged, safety properties are preserved. Roughly speaking, safety properties are those which state that something bad (e.g., deadlock) could not happen. Finer state merging approaches could disregard state agreement on presence of atomic propositions that occur only in the positive form in a formula (under an even number of negations) or only in the negative form (under an odd number of negations). The technique is known also as state folding or quotient. The main problem of state merging is how to determine states to be merged. Thus, direct state merging is rarely supported by existing tools. However, many other abstraction methods could be interpreted as state merging of underlying Kripke structure of the specification. Among few exceptions, there is the Concurrency Workbench tool, which could perform state merging according to a bisimilarity criterion [Cleaveland et al, 1993].

- **Deleting (some or all) variables** [Berard et al, 2001]. According to this technique, variable updates and guards are simply ignored. The approach is rarely used due to coarseness, especially in the case of complete deletion of all variables. In such a case, it could be difficult to handle properties which are related to variable values, at least without some ad-hoc coding. The
technique is perfectly appropriate for so called data independent systems, where computational flow does not depend on data. Variable deletion could be seen as a kind of state merging. Keeping some variables would result in a more fine-grained abstraction, but it is less trivial due to a need for an analysis of dependencies of variables.

- **Variable abstraction**, aka abstract interpretation [Cousot and Cousot, 1977], is more fine-grained than variable deletion. It usually consists in domain or data type abstraction, as follows
  - integers are mapped into even and odd values, or more generally, integer counting is replaced by counting modulo $n$ [Beard et al];
  - interval arithmetic;
  - bounded domain (e.g., integer could be represented by interval $[1..10]$ and interval bounds) [Beard et al];
  - relational domains that keep track of equality relations are more computationally difficult, but more precise than non-relational abstractions, such as interval arithmetic. Most popular relational abstractions are convex polyhedra (HyTech), octagons, difference bound matrices, linear equalities.

- **Multi-valued logic** could be used to perform a fine abstraction that loses neither property nor its violation. 3MVC tools uses three valued logic to abstract and verify a Java program with dynamically allocated objects and threads [Yahav, 2001].

- **Predicate Abstraction.** This technique combines model checking and theorem proving [Saidi and Graph, 1997]. A related technique is splitting [Dams et al, 1993]. Abstraction is induced by a set of predicates. CEGAR could be used for predicate discovery [Das and Dill, 2002]. Predicate abstraction could deal with infinite systems. Predicate abstraction of C programs is used by Microsoft for static verification of device drivers [Ball et al, 2006]. While the original technique relies on theorem proving, few recent tools, such as SatAbs [Clarke et al, 2005], involve alternative techniques, such as SAT solving.

- **Symmetry reduction.** Usually, a kind of state merging: a Kripke structure is transformed into “Quotient Structure” where each state represents a class of symmetric states of the original model [Clarke et al, 1993].

- **Partial Order Reduction.** Often, an arbitrary order on the events of loosely coupled processes could be imposed [R. P. Kurshan et al, 1998].

- **Irrelevant process/thread deletion.**
- **Abstraction of the number of process instances.**

- **Compositional model checking**, when reasoning on components could lead to a conclusion on the property of the whole system (e.g., the assume guarantee approach [McMillan, 1997]).

- **Slicing** [Tip, 1995]. It is removal of parts of a program (or, in the model checking context, of a specification), which are irrelevant to the property. Usually, it involves building a dependence relation between various parts of the program (or specification). Most often, next-free LTL (which is obviously stuttering close) is considered [Millet and Teitelbaum, 1998], [Brückner and Wehrheim, 2005]. Spin tool supports static (i.e., prior to execution) slicing. Abstract slicing uses abstract interpretation, model checking, and predicate abstraction to achieve finer slices [Huong, 2005].

- **“Live variable”**. It is a simple slicing technique, which somewhat resembles garbage collection. Typically, when a variable is “dead”, i.e., will never be used again in and does not appear in the property, it is set to a designated value (zero). A “live” optimization package comes with IF toolset [IF], [Bozga et al, 1999].

- **Restriction**. It consists in elimination of some executions [Berard et al, 2002]. Usually, states or transitions of underlying Kripke structure are eliminated. For example, guards are strengthened or integer domains are replaced with intervals without taking care of bounds. Restriction does not preserve the safety properties, so CEGAR cannot be applied to verification of safety properties. Restriction preserves so-called stable properties violations (once true forever true), such as reachability properties. Slicing could be seen as a kind of restriction.

- **Observers** [Groz, 1986]. Observers and filters allow one to restrict allowed behavior with a designated “observing” or “testing” automaton. Unlike previous methods, the behavior is reduced without simplifying the structure (state and transitions) of the specification. In some cases, the total state/transition number can, unfortunately, increase. Rich observers use variables, history, and counters. Observers, such as Spin “never claims”, are often used not only to simplify the model, but also as a powerful means to express properties.

Several alternative taxonomies of model abstractions in a wider simulation context are known [Frantz, 1995], [Lee and Fishwick, 1996].

### 2.1.4 Fairness Constraints

Some workflow experts argue that an accurate workflow analysis should support strong fairness constraints. Several workflow languages do not allow for
variables, thus finite cycles are often abstracted into cycles which allow for infinite execution. Also, a workflow without guarantied cycle exit could be seen as an abstraction for a set of workflows with finite (but different) cycle exit conditions. However, workflows with infinite loops are seldom needed. A solution to exclude such unneeded executions is to impose so-called fairness constraints. These constraints could be either encoded in the property itself, or incorporated into the model, provided that the language of the model checker supports them. While developing the UML activity diagram based workflow analysis tool TATD [TATD], the authors failed to reach acceptable results with an existing model checker. To solve the problem, they develop a fair version of the nuSVM model checker [Eshuis and Wieringa, 2002]. We argue that while the fairness problem exists in workflow analysis, model checking support for fairness is not always needed. Moreover, the experimental setup was not exactly fair in those experiments. Namely, the different constraints were specified in LTL and in fair nuSMV language. In the former case, the authors encode exit for each particular cycle, while in fair nuSMV they just define eventual termination of workflow. In our opinion, one could specify eventual termination of workflow in LTL, as

\[ \langle (\text{final or deadlock}) \rangle \rightarrow \text{property} \]

could suffice. Another alternative could be augmenting cycles with counters and cycle exit conditions. While counters are likely to add to state explosion for simple workflows, in the case of modeling a workflow constrained by a RBAC, the effect of counters on the state space is not clear. The counter approach could nevertheless face another limitation. Eshuis and Wieringa [Eshuis and Wieringa, 2002] argue that infinite cycles could occur even in the modeling of acyclic workflows. While we believe that modeling of acyclic workflow could accurately be performed without infinite cycles, in modeling of a workflow extended with a RBAC in style of [Hansen and Oleshchuk, 2005], it might be difficult to avoid infinite cycles even in acyclic workflows. For example, a cycle could occur on a path, where a candidate action is always denied by a SoD constraint. Safety and reachability properties do not depend on presence of infinite cycles.

### 2.2 Workflow Model Checking

Workflows reflect organizational aspects of a work procedure, such as structure, synchronization and ordering (flow) of tasks, information flow, etc.

Higher-level security properties of business applications result not only from access control mechanisms, but also from business workflow implementations. For example, if the authorization of a purchase request precedes (in all or some executions) the purchase request in a given workflow, then the purchase
request can be approved before being completely filled. Such a workflow violates the security property “no carte blanche”. It should also be noted that some overly restrictive security policies may render certain workflows or important execution scenarios non-feasible. Thus, workflow verification is relevant to security analysis.

Usually workflow modeling is based on graphs, automata, Kripke structures, Petri Nets, and more rarely on constraint solving, data-flow pointer analysis, BDD, propositional and temporal logic [Wainer et al, 2004].

There are a number of workflow notations and supporting tools. However, despite active research in the area, available verification tools are not numerous. It should be noted that several workflow verification tools, such as Woflan (Work Flow Analyzer) [Woflan], do not support the use of data. Even cycles are not always fully supported, for example, by Testbed Studio (a Spin front-end) [Janssen, 1999] and are considered as a challenge in workflow modeling.

Significant academic research is performed on advanced constructs of business workflow languages with complicated semantics, such as non-local operators (or-forking, cancellation). For example, the problem, known as “vicious cycles”, was identified in early formalizations of non-local semantics of the ARIS workflow description language, EPC. The tool, called EPC Tools [EPC], is based on a formal semantics which resolves such deficiencies [Kindler, 2003]. Unfortunately, so far EPC tools support only consistency (soundness) checking and simulation, rather than full fledged property verification.

Petri Net is a particularly popular formalism for workflow modeling, since it allows representing the process forking/merging in a natural way. A variant of Petri Net is promoted as a workflow net [Aalst, 1997]. Colored Petri Nets verification tools [CPNTOOLS] could be used to model and verify EPC workflows extended with probabilities, post and pre conditions, variables, time etc [Hee et al, 2005].

Since suspension of workflows is often difficult and undesirable, the problem of dependable introduction of policy changes without interrupting currently executed workflows arises [Bose and Matthews, 2001]. The proposed solution is based on Spin model checking of workflows against properties such as “jobs are billed if and only if they are shipped” [Bose and Matthews, 2001].

A great deal of research is devoted to workflow model checking related to web service verification [Koshkina and Breugel, 2003]. Among available specialized web service/workflow model checking tools we could mention WSAT tool [WSAT], that supports several web service languages, including BPEL4WS, and could use both Spin and NuSMV as a backend.
Infinite model checkers could handle unbounded variables, though they face scalability problems even on relatively simple workflows [Fu et al, 2002].

Thus, certain workflow notation features might be difficult to model accurately, but their modeling is not impossible. There exist supporting tools that translate workflow models into input languages of model checking tools (usually automata or Petri net based), so in this work we assume an automata model of a workflow is already available, rather than dealing with a specification in a complicated workflow notation.

2.3 RBAC Abstraction and Model Checking

Even a relatively simple case of verification of static SoD policies (such as at least \( n \) user are required to perform a task) from static role mutual exclusion constrains (such as a given user cannot be assigned to more than \( k \) roles from a given set of allowed roles) is computationally difficult (coNP-complete) [Li et al, 2004]. Thus, “heavy artillery” of model checking that involves sophisticated optimization techniques to cope with hard problems (SAT-solvers, BDD, partial order reduction etc…) is justified for RBAC verification.

Remark. In our work, as well as in most RBAC publications, abovementioned “role mutual exclusion constraints” are considered to be SoD constraints. However, in order to draw a clear distinction between the objective and the mechanism, Li et al [Li et al, 2004] prefer to call them “role mutual exclusion constraints”. Such a clarification of the standard terminology may be justified for abstract fundamental research on complexity of RBAC policies. However, hereafter, we prefer a more usual wider understanding of SoD that encompasses both, the objective and the mechanism. In our framework, the objective is usually referred as a property (though one could analyze neutral or even undesirable properties). The low-level mechanism to fulfil objectives, in our framework, is a rule. The constraints are intermediate between rules and properties.

Most of known publications apply model checking to RBAC policies, abstracting from implementation details and control mechanisms.

[Hansen and Oleshchuk, 2005] verifies application and organization specific RBAC “policy implementations” against selected security properties that represent a high level enterprise security policy. Security policies may be of a higher level and evolve independently of their RBAC implementations. Among verified properties are static, dynamic (for one or all sessions), operational, object, history based SoD, prerequisite, cardinality, and, user-user\(^2\) constraints. Apparently, the RBAC implementation PROMELA model supports a set of workflow constraints plus several SoD constraints. Similarly, some of the verified history based properties appear to be more relevant to business logic than to security (e.g., a recorded invoice is eventually verified). The RBAC implementation model consists of four concurrent processes: the first process selects user, the second selects a role, the third – a permission, and the fourth

\(^2\) A user-user constraint usually prevents two users from activating or being assigned the same role.
verifies relevant constraints, and, if no constraint is violated, models the operation, associated with the permissions by recording the user id into a designated “history” array. While history array does not scale well, alternative approaches, such as a blacklist [Crampton, 2003] of operations for each user, or enabled and disabled states for each permission assigned to a role are proposed.

Ahmed and Tripathi [Ahmed and Tripathi, 2003] verify Computer Supported Cooperative Work System, CSCW, using the model checker Spin against various security properties. While manual specification of CSCW security properties in LTL might be difficult, “conversion functions” that facilitate the translation of SoD constraints to LTL are developed. Property specific abstraction is used to fight state space explosion. Four different Spin models are developed to verify four different aspects, namely, task flow (e.g., each operation could be executed), role constraints (e.g., each role could have a member), information flow (e.g., non inference), and administrative role assignment. To cope with state explosion, following measures are suggested:

- Some operations could be excluded.
- Verification based on a specific user often can be generalized to verification of global properties (user symmetry).
- Abstraction of internal data structure. For example, in many cases, the model does not need to maintain the count of the events in the precondition that contributes to a property. Instead, a bit signifying that the precondition is satisfied is maintained.
- For a faster verification, role constraints could be specified in LTL.

Specification language cTLA, derived from Lamport’s Temporal Logic of Action (TLA), is used to formalize and after translation to PROMELA to verify RBAC [Herrmann, 2003].

Spin and temporal logic could be used for goal-elaborating policy refinement of a higher level policy to a low level policy [Rubio-Loyola, 2005].

Nguyen and Rathke [Nguyen and Rathke, 2005] use Spin to verify multithreaded functional programs (rather than workflows or management systems) against security policies, expressed in form of a “policy automaton” which, generally speaking, could be used to express RBAC policies [Covington et al, 2000].

A popular open-source model checker NuSMV is used to solve so-called safety analysis problem (whether only trusted users could violate a given security property) [Jha et al, 2005]. Furthermore, a more general Administrative Insider Threat Assessment Problem (AITAP) is formulated and verified. Access control schemes are formally defined as state-transition systems (i.e., labeled Kripke
structures). The NuSMV model of RBAC is straightforward, except for an efficient model abstraction, based on pruning of irrelevant rules and role activations. In the reported case studies, the model checking approach is compared with the logic–programming approach, based on a Prolog like language. Experimental data favor the model checking approach. While state space explosion persists, RBACs with tens of roles and rules are verified.

A popular RBAC analysis tool is Alloy Analyzer, a model checker that supports a light-weight structural specification language Alloy, based on the first order logic [Zao et al, 2003], [Mankai and Logrippo, 2005], [Schaad and Moffett, 2002], [Hughes and Bultan, 2004]. While language features are limited, and modeling of complex history based properties or workflows is difficult, it is perfectly appropriate for describing organizational structures and simple SoD constraints.

A designated policy verification model checker RW is developed [Zhang et al, 2004], [Guelev et al, 2004].

Besides model checking, RBAC schema/policy verification could be performed using theorem proving [Drouineaud et al, 2002], graphs, binary decision diagrams (BDD) [Fistler et al, 2005], constraints solvers, and integer programming.

Several criteria for composing policies that preserve certain policy properties, such as compatibility with a given application (i.e., workflow) are known [Gligor et al, 1998]. Possibly, they could be used to simplify RBAC policy verification by composition or decomposition.

Summarizing the above review of existing literature, we notice that there are few reports on applying the model checking technology for verifying RBAC on workflows. Even when a workflow, on which a RBAC is imposed, is somehow modeled, technical details and experimental data are rarely reported. Here we try to elaborate this idea in detail. More specifically, we suggest several approaches for verifying a set of properties (typically related to separation of duties and reachability concerns) over a Business Workflow and a given RBAC, using the model of Extended Finite State Machine (EFSM), which is often used in input languages of model checking tools. EFSM includes a set of states, actions, and guarded transitions along with optional variables and parameters. We describe a naïve single EFSM model and a multi EFSM model. Several techniques alleviating state explosion caused by a growing number of users are experimentally evaluated.
3. AN APPROACH TO THE VERIFICATION OF BUSINESS WORKFLOWS AGAINST RBAC

3.1 Business Workflow and RBAC

The models of a Business Workflow and RBAC are presented in this section. The model of a Business Workflow is first elaborated, and then the notion of roles is introduced, along with the rules and constraints that can be defined on them. We conclude this section by describing properties to be verified over the Business Workflow and RBAC and discussing role activation constraints that can alleviate the state explosion problem. More sophisticated means for fighting this problem are elaborated in Section 5.

3.1.1 Business Workflow Model

A Business Workflow is a workflow, whose application domain pertains to business activities. Such activities may be related to buying or selling goods, contracting services to or from another company, etc. While different languages can be used to model workflows, we assume that a workflow is modeled by an Extended Finite State Machine. One particular state of the model is the initial state of the Business Workflow. Transitions between states are labeled with actions of the Business Workflow. Message input, output, arithmetic operations on Business Workflow-specific variables are considered to be actions. Transitions can be guarded by conditions. Transitions with no action are called tau-transitions (aka internal transitions).

Such choice is not unique, various workflow notations (often based on Colored Petri Nets) have been proposed, and usually could be mapped to EFSM (note that transformation of Extended Finite State Machines to Petri Nets is supported by various tools, such as Model Checking Kit [Kit]). Actually, translation of workflows to an automata/EFSM like language is required by most workflow model checking tools. Our working Business Workflow example is presented in Section 3.2.

We show later that EFSM can be used to model not only a Business Workflow, but also a RBAC. Note that while in most EFSM definitions and supporting tools, such as Spin, EFSMs communicate via message exchange, we do not use this option here. Hereafter, messages are sent or received to or from the environment, and not exchanged between EFSMs. Some actions, called user actions, are performed by users, others, called system actions, are performed without user involvement. We consider that user actions are the subject of additional RBAC constraints or rules. EFSM does not provide a designated exception mechanism, thus all the exceptions (such as “quantity of goods received is incorrect”) are modeled as message receives. Thus, we consider
EFSMs that progress independently and communicate only via shared variables. Since shared variables are used in most mainstream programming languages, this communication mode could easily be understood even by the reader without formal background. An EFSM could be associated with one or more EFSM parameters. When in an EFSM a parameter, such as user identifier, variance threshold value, or purchase price is instantiated to a particular value, the resulting EFSM is called an instance of the parameterized EFSM. Moreover, messages could also be associated with parameters.

Unlike our previous work [Boroday et al, 2002], [Petrenko et al, 2004], we do not put any additional restrictions on the EFSM. Thus, transition labels (guard, variable updates, message receive or send) are completely optional. Still, we believe that the methods of conservative variable deletion, described in our previous work [Boroday et al, 2002], could be applied to Business Workflow EFSM as well.

A sequence of consecutive transitions is called a path of EFSM. Unless the first state of a path is clearly indicated, we assume that such path starts from the initial state. A path, associated with variable and parameter valuations that conform to guarding conditions and variable updates is called an execution, and a path for which such valuations of variables exist, is called executable. Not all paths are executable. Due to the absence of queued EFSM message exchange, unlike in [Boroday et al, 2002], an asynchronous composition of several EFSMs could be defined. Here, an execution of such EFSM composition is called an execution of a set of EFSMs.

**Remark.** EFSM composition could be defined as follows. Without loosing generality, we assume that sets of local variables of component EFSMs do not intersect (if they do, they can be renamed). Each variable of each component is a variable of the composition. Each state of a composition is a tuple of states of component. The initial state of the composition is a tuple of all the initial states of components. Whenever there is transition of a component from a state \( s \) to a state \( s' \), for each state of the composition that contains \( s \) there exists a transition of the composition from the above state of the component into a state of composition, obtained by replacing \( s \) with \( s' \). The guard and actions of the composition’s transitions are the same as those of the component transitions. However, in this work, we do not have to compose EFSMs manually, relying on the model checking tool. Our definition is consistent with Spin model checker, though Spin builds a finite state (rather than EFSM) composition of a system of communicating EFSMs with a property automaton.

---

3 EFSM parameters are valuated only at the beginning of the execution, while message parameters could be valuated several times in course of the execution.
3.1.2 RBAC Roles

The central notion of RBAC is the notion of role. Each role is associated with a non-empty set of user actions. In a formal set-theoretic RBAC definition, permission to role assignment relation is usually defined [NIST]. Every user action of the Business Workflow is associated to at least one role, though one user action can be associated to several roles at the same time. The concept of role refines the previously known concept of a user group.

A user to role assignment defines which roles each user could activate during the execution of Business Workflow. Role activation resembles a “login” procedure, by which a user identifies himself to the system, with a given set of rights. Role activation can be allowed either in any state of the workflow, in specific states of the workflow (such as only in the initial state for instance), or can be constrained by a more elaborate restriction. Between those two extremes (activation in any or only in initial state), one meaningful role activation mode is to allow activation of a role only in Business Workflow states that have at least one emanating transition, labeled by an action of the role. This mode is assumed in this work from this point on. Options to extend our results to other role activation modes are discussed in Section 6.

Each role can only be activated once and no de-activation is allowed during the execution of a given Business Workflow. We allow a user to activate several roles. The working example of roles for our Business Workflow and user to role assignment is described in Section 3.2.

3.1.3 RBAC Rules and Constraints for Role Activation

We distinguish broadly two types of RBAC constraints: dynamic and static. Static ones are imposed on the actions of the administrator, who assigns roles to users. Dynamic constraints are imposed on role activation during Business Workflow execution. Thus, dynamic ones could be in conflict with a user to role assignment. Dynamic constraints are also called weak or run-time constraints. Since static constraints are better understood and well supported by existing tools, our major focus is on the modeling of dynamic constraints. Our approach is related to rule-based RBAC systems, such as OASIS (Open Architecture for Securely Interworking Services) [Bacon et al, 2002] in which role activation is governed by rules, which resemble Horn clauses: namely, a set of preconditions is associated with each role. Contrary to OASIS, which does not yet support negative preconditions (such as “Role 1 is not active”), which are needed to implement separation of duties constraints, we allow such preconditions in our RBAC model. Conditions that are not based on roles (e.g., temporal, or user specific conditions) are not considered in this work. No priority

---

4 For simplicity we do not introduce objects here (thus, we assume that “open an account” and “open a window” are different actions, rather than the same action on different objects).
is defined on rule executions: rules could be applied in arbitrary order. We show how the typical constraints on roles could be translated into rules. Moreover, using our approach, one could verify whether an action based constraint could be imposed by a set of role-based rules on a given Business Workflow. We also verify state reachability properties. State reachability is a way of verifying compatibility of Business Workflow with a set of rules [Gligor et al, 1998]. It is especially important to ensure the reachability of a success state, which corresponds to a normal execution of Business Workflow. However, reachability of states that correspond to certain important functionalities might also be required.

Several types of constraints could be used in RBAC. Cardinality constraints could limit the number of users for a role. For example, only one CEO could be allowed. Cardinality constraints could also restrict the number of roles for a user. For example, a low ranked user could be restricted to one single role at a given time. Simple separation of duties constraints prevent any user from performing actions belonging to mutually exclusive roles in the same Business Workflow execution. For example, it could be forbidden for the same person to issue and approve a purchase request. Prerequisite constraints require one role to already be active for a user before another role could be activated by the user. For example, one should have an account in a bank (active or be assigned to the role of bank account holder), before receiving a credit (to be assigned to or activate a debtor role). While some constraints could be trivially encoded into sets of rules, other constraints are less trivial, e.g., rules for cardinality constraints could become very cumbersome unless auxiliary variables are used. We explain how this could be resolved in Section 3.4.2.

The above examples are not an exhaustive description of possible constraints. While it is understood that simple SoD constraints are not always sufficient and there is a need for order (history) dependent constraints, a precise range of the supported constraints differs from system to system. Actually, an NIST document [NIST] indicates that most of the systems currently in use and implementing static constraints limit themselves to order-independent constraints on role activation. Obligations are not considered as a part of RBAC here. RBAC constraints express that some undesirable behavior should not happen. This type of constraints is called in the model checking literature “safety properties” (the notion is used in a more specific meaning in security literature; however, here we use it only in the above sense). While various preconditions could occur in rules of the RBAC, we consider only role based, i.e., those that are related to previous activation (or non-activation) of other roles, or propositional logic formulas that use negation, conjunction, or disjunction over atomic propositions that denote activity of certain roles. The integration of preconditions based on previously executed actions is a straightforward extension of our work.
Note that even if a rule allows role activation, the role is not necessarily activated during a given Business Workflow execution. However, no role can be activated, unless permitted by its corresponding rule.

### 3.1.4 Safety Properties

While other studies are concerned with properties of a particular RBAC, constraints set or Business Workflow in isolation, here we are mostly concerned with properties of Business Workflow, on which a RBAC with a given set of constraints is imposed.

While, usually, constraints on role activation could directly be enforced by a set of RBAC rules, the relation between role activation and actions is less straightforward. Thus, we concentrate on verification of properties on actions.

Here we consider mainly safety properties (which state that some undesirable behavior never happens). While safety properties have a clear meaning, non-safety properties, such as the eventuality of reaching a certain state, allow for various interpretations. The valuation of non-safety properties would depend on many assumptions, such as whether the user always attempts to perform only actions allowed by the RBAC and Business Workflow or may attempt to perform arbitrary actions, whether one user always activates the relevant roles to progress etc. For this reason, we do not tackle non-safety properties in this work.\(^5\)


### 3.2 “Procure to Stock” Business Workflow

#### 3.2.1 Business Workflow Description

The Business Workflow used as a running example is called “Procure to Stock” and describes the procurement of goods from the creation of the purchase request, up to the final delivery of the goods to the stock or the termination of the scenario in several possible unsuccessful end states. Figure 1 shows the Business Workflow EFSM. Message receives are used to model exceptions during the execution of the Business Scenario. For instance, there is an exception called \textit{PriceVarianceException}, which occurs whenever the price

\(^{5}\) Note that any property considered only on successful executions is a safety property, e.g., property in the form (\langle\neg\text{success state}\rangle\rightarrow\phi) is a safety property. (Personal communication with Qing Fan)
received on the invoice differs from the price that was settled between the buyer and the seller, when the purchase order was placed. In this exception, the parameter *PriceVariance* describes the difference between the two prices.

In Business Workflow in Figure 1:

- User actions are set in bold. They are *CreatePR*, *ConfirmPR*, *CancelPR*, *?NoSupplier* (“?” indicates a message reception), *!ReqPrice* (“!” indicates a message sending), *!ReqExpert*, *?BestPrice*, *?BestExpert*, *SendPO*, *RecInvoice*, *RecCreditNote*, *NoCreditNoteRec*, *PaymentProcess*, *RecPaymentConf*, *BlockGoods*

- The only variable, *Counter*, is in italic.

- Parameters of the Business Workflow are also in italic; there are five integer parameters *Cost*, *T1*, *T2*, *T3*, *T4* and one Boolean parameter *SupplierIndicated*.

System actions, which are not attributed to any user, are also set in italic, there are three of them: counter increments, *?PriceVarianceException*, and *?QuantityVarianceException*. The latter (exception message receives) are associated with input parameters *PriceVariance* and *QuantityVariance* (*QttyVariance* on the Figure 1.), respectively.

In the next section, a role set and role assignment for this Business Workflow are presented.
3.2.2 Roles and Role Assignment

Five roles are defined for the Business Workflow:

- The only action of Role 1 is **CreatePR**.
- The actions of Role 2 are **ConfirmPR, CancelPR**.

The actions of Role 4 are RecInvoice, RecCreditNote, NoCreditNoteRec.

The actions of Role 5 are PaymentProcess, RecPaymentConf, BlockGoods.

The following users are defined (for an imaginary small company):

• The first user, Alice, the CEO; she can activate all the roles.

• The second user, Bob, the Supervisor; he can activate Role 1, Role 2, Role 3 and Role 5.

• The third user, Carol, the Financial Manager; she can activate Role 4 and Role 5.

• All the other users are Employees and can only activate Role 1 and Role 4.

3.2.3 Abstracted Business Workflow

Model checking of a Business Workflow with unbounded parameters and variables, in some cases, is impossible. In this section, we describe abstractions which applied to the example Business Workflow make model checking feasible.

The variable counter, the integer parameters cost $T_1$, $T_2$, $T_3$, $T_4$, as well as the input parameters QuantityVariance and PriceVariance, are abstracted conservatively. The abstraction of these variables, thus, introduces a non-deterministic choice instead of guarding (branching) conditions related to these variables. Deletion of the counter variable adds an infinite cycle absent in the original Business Workflow (this is the only added path in the EFSM of the abstracted Business Workflow which is non-executable in the original Business Workflow). Actually, the added cycle does not prevent a model checker from verifying any safety properties on actions or state reachability. To verify other properties (not related to safety), the CEGAR approach could be followed or for this particular cycle a fairness constraint could be used.

Other approaches not explored in this work are restricting parameter values, coverage analysis, and the Lee-Yannakakis method for transforming an EFSM into an equivalent EFSM without unexecutable paths [Lee and Yannakakis, 1992], although the method may not work for EFSM with infinite variable domains.
3.3 Naïve Model of Business Workflow and RBAC

A naïve approach to model Business Workflow with RBAC constraints is to add RBAC related variables, guards, transitions into the given Business Workflow EFSM. Such a simple model of a Business Workflow and RBAC is a single EFSM. Namely, in order to impose a RBAC on a Business Workflow, the Business Workflow EFSM is augmented with auxiliary variables, transitions, and additional guards on the transitions. For \( n \) users and \( r \) roles a set of \( n \times r \) Boolean variables in the EFSM encodes roles activation for users. Role activations are modeled by looping transitions. Such activations modify the value of the variables representing the role activations for the users. Guards are defined on role activation to impose RBAC constraints and action activations to implement the role definition (i.e., which action belongs to which role). In each state of the EFSM, any user can be chosen for executing a next action. Such choice is represented by a specific transition for each user.

Figure 2 shows a fragment of such an EFSM for two users to illustrate the idea. Hereafter, conditions (guards) and actions could be present on the same transition. To keep distinction between actions and guards, we use \( \text{if/then} \) keywords (this corresponds to PROMELA \( d\_step \) which evaluates a guard and performs several actions in a single atomic step in order to reduce state space).
Figure 2 Fragment of "Procure To Stock" Business Workflow Constrained by a RBAC

Such a model is intuitive, but is somewhat cumbersome due to a significant number of user-choice and role activation transitions (transitions related to user choice are present in each state) and suboptimal state-space\(^6\) during model checking. As the user choice is non-deterministic, the verification process may select a user for which no actions are currently executable. In such a case, several non-deterministic user choices may happen before a user that can actually perform at least one action is finally chosen. In the absence of fairness constraints, this could even result in infinite loops. Moreover, each of these choices may result in additional states, contributing to state explosion.

The model, a single (large) EFSM, could become more tractable if decomposed onto three EFSMs, one for the Business Workflow, second (single state) EFSM for non-deterministic user choice, and a third one for role activations. However, such decomposition may complicate the extension of the model toward Business Workflow-state dependent RBAC modeling and require special efforts to limit role activation only to relevant states.

One way of solving the problem without adding to the size of the model is described in the next session.

---

\(^6\) Strictly speaking, the number of reachable configurations, i.e., combinations of states with variable valuations
3.4 Multi EFSM Model

We describe in this section a refined model of Business Workflows, roles, users, along with the RBAC.

The idea is to use a single state EFSM (one per user), where each state of the Business Workflow is represented by a corresponding value of a designated variable shared among the EFSMs. This allows us to resolve some of the above mentioned problems of the naïve model as follows. First, any role activation transition now appears only once in the EFSM. Second, each user is represented by a dedicated EFSM, but all user EFSMs share the same Business Workflow state and role activation variable. Choosing a user is performed by the model checker, when it decides which EFSM executes an action in the next step.

While such choice increases the number of EFSM instances, Spin model checker language allows a compact representation of such a set of parameterized EFSM instances, via the process type mechanism. The following section details the approach and illustrates it with examples.

3.4.1 Business Workflow Transformation

We transform the abstracted Business Workflow EFSM into an (in a sense equivalent) EFSM with a single state and one variable State (while preserving behavior in terms of user actions). The variable State contains the identifier of the current state of the Business Workflow. Transforming a state of the Business Workflow EFSM into a variable enables the explicit synchronization of several users executing the same Business Workflow, as we describe below in Section 3.4.3. Each variable of a Business Workflow is preserved. The variable State is used as a guard on transitions of the EFSMs defining the executable actions in the corresponding state of the Business Workflow. Action transitions that are not looping transitions in the Business Workflow set the value of the State variable to the next relevant state. Thus, each transition of the Business Workflow from state $s_i$ into state $s_j$ with the guard $g$ and action $a$ is represented by a loop of the single state EFSM with the guard $g$ and $State == s_i$ and action $a'$ which combines the action $a$ and assignment $State = s_j$. The example below refers to the Business Workflow described in the previous section (with no information on user or roles for a moment):
3.4.2 RBAC and User Modeling

We represent each user by an EFSM, which is an instance of a Business Workflow parameterized EFSM that encodes in a local variable, the user identifier. The user EFSM is obtained from the single state Business Workflow EFSM, described above, by complementing it with Boolean variables which represent currently activated roles, adding transitions for role activation and user actions and guards to constrain the execution of actions to specific states.

For each existing role $R$ that can be assigned to the user $U$ we define a corresponding Boolean $\text{Activated}(R, U)$, which is true when the role $R$ is currently activated for the user $U$ and false otherwise. For a given role and a state, in which the role can be activated a new looping transition is created in the EFSM. The action of the transition is the activation of the role, while the guard is a predicate, constraining the activation.

The predicate expresses the RBAC rules defined for this role. An example is given in Figure 4, implementing the following RBAC rule: “A user $U$ can only activate Role 2 if and only if he previously did not activate Role 3 and Role 4”. Note that this rule is not a mutual exclusion constraint for Role 2 and Role 3 as well as for Role 2 and Role 4, because the user is allowed to activate Role 2 and then Role 3 or Role 4.
We represent a mutual exclusion of roles (SoD constraint) by the two following rules:

- A user $U$ can only activate $Role 2$ if and only if he did not activate $Role 1$.
- A user $U$ can only activate $Role 1$ if and only if he did not activate $Role 2$.

Role cardinality constraints (such as the constraint “Role $R$ cannot be activated by more than $K$ users in any execution of the Business Workflow”) are modeled by a variable $number\_users\_for\_role\_R$, which is shared by all the user EFSMs. The variable is updated and used on guards for role activation in a similar manner (note that role cardinality constraints cannot be encoded into variable-free rules).

User cardinality constraints are represented by a counter $number\_role\_activated$ in each user EFSM, incremented each time a new role is activated by the user. Guards of role activation transitions are extended with the conjunction “and $number\_role\_activated < K$”, where $K$ is the maximum number of roles allowed to activate. In the presence of a cardinality constraint, each role activation requires an auxiliary guard, which prevents repetitive activation of the same role.

Each transition with a user action is constrained by a guard, which states that a role containing the action is active. An example is given in Figure 5, where the action $CreatePR$ is enabled for users of $Role 1$ and action $ConfirmPR$ for users of $Role 2$. 

Figure 4 A Transition that Represents a RBAC Rule

If $State = s$ and $Activated(3,U) = \text{false}$ and $Activated(4,U) = \text{false}$ then $Activated(2,U) = \text{true}$
Arbitrary not-conflicting rules addressing the same role could also be combined, by making conjunction of their preconditions and merging their variable updates.

### 3.4.3 User Choice and Synchronization

![Figure 5 Transitions Augmented with RBAC Related Guards](image)

![Figure 6 Model Overview](image)
As shown in Figure 6, each user is represented by a dedicated EFSM. The choice of the user to execute a next action of the Business Workflow is, thus, performed by the model checker. As only an EFSM, which has an enabled transition, can be chosen, no unneeded states (corresponding to the selection of a user that cannot perform any action, as was possible in the naïve model) is explored. The variable `State`, shared by all EFSMs (as well as workflow variables and parameters) enforces EFSM synchronization. `Activated()` array represents the current roles activated by users. For simple RBAC constraints, this array might be local, but, in the general case, sharing the array might be required. Alternatively, auxiliary shared variables are used to share information on active roles (e.g., to encode cardinality constraints).

### 3.5 Properties

Several types of properties can be formulated for a Business Workflow and a RBAC such as compatibility, redundancy, cardinality, prerequisite, separation of duties, minimum number of users, RBAC requirements, secrecy and specific company policies.

Compatibility properties express conditions under which the RBAC is compatible with the Business Workflow (i.e., it is not overly constrained, for instance). Redundancy properties state that a constraint is redundant and can safely be removed from the RBAC. Cardinality properties ensure that no role can be activated by more than a given number of users. Prerequisite ensures that a role can only be activated, if a specific set of roles has previously been activated. Separation of duties enforces a mutual exclusion of roles or actions for a single user or several users (“buddies”). For instance, a simple SoD property states that no user could perform two given actions in the same execution. Another type of properties is that a given number of users is indeed the minimum required to successfully complete a Business Workflow for a given RBAC. Requirements of the RBAC design could be verified by using properties on the rules of the RBAC. Secrecy properties pertain to the potential information leaks in a given Business Workflow to users having insufficient roles privileges to access such information. Finally, some specific properties related to company policies could also be considered.

However, here we consider only two types of properties: state reachability and simple separation of duties properties. State reachability properties are used to verify the reachability of a specific state (e.g., a successful end state of the Business Workflow under specific conditions, such as a given parameter combination or a given user-role assignment). Separation of duties properties are used to check the possibility for a set of users to execute several specific actions during any Business Workflow execution. Several properties and their formulations are discussed in Section 4.
We investigate two ways of representing properties, using history array and LTL.

History array records the actions performed by each user during an execution of the Business Workflow. The approach faces difficulties with cyclic workflows, where the number of executed actions is unlimited. Even if cycles are augmented with counters, the size of the history array could be a serious issue. Thus, in our experiments, for a given action only the last user having performed such action is recorded, which imposes a limitation onto properties that could be verified. The size of the array equals the number of user actions. The property is stated as a predicate on the values of the cells of the history array.

The second approach, instead of recording executed actions, uses LTL for formulating properties.

### 3.6 State Space Explosion

State explosion limits a user pool any model checker can handle for a given Business Workflow and RBAC. Thus, there is a need to alleviate its effect.

In the remaining part of this document, we present several techniques for fighting state explosion in model checking Business Workflows with RBAC and experimentally evaluate their effectiveness. To this end, we first present experimental data characterizing the model checking process of the above described model without applying those techniques.

### 4. EXPERIMENTS

We present in this section the results obtained with the multi EFSM model introduced in the previous section. We describe the setup used for the experiments, namely, two different RBAC systems and then present the experimental data for the two types of properties: reachability and separation of duties. In the following experiments, we assume, as described in Section 3.2.2, that the first user, Alice, is the CEO, user 2, Bob, is the Supervisor, user 3, Carol, is the Financial Manager, and 4th to nth users are all Employees. For model checking with one to three users not all users’ types are, thus, present. In all the performed experiments described below, the parameter supplierIndicated is always set to false, forcing the model checker to consider states from s4 to s12. Obtained data, thus, characterize the worst case for model checker, as it is forced to explore the whole state space.

---

7 A refined version of the history based approach might use only few variables, which record only essential to property verification information [Hansen and Oleschuk, 2005].
4.1 Setup

All experiments are performed using Spin model checker, version 4.2.3 under Windows XP, on a computer with 2Gb of RAM, of which 1Gb is allocated to Spin. Experiments are performed with an increasing number of users until the verification can no longer be performed within the given memory. In order to evaluate model checking complexity with a different size of RBAC, we define two RBACs: a minimal one with only one rule and a more complex one with four rules restricting user/roles assignments.

The simple RBAC, hereafter RBAC1, is as follow:

- **Role 5** cannot be activated after **Role 1**.

Another RBAC, hereafter RBAC2, is as follow:

- **Role 2** cannot be activated after **Role 1**.
- **Role 3** cannot be activated after **Role 2**.
- **Role 5** cannot be activated after **Role 3**.
- **Role 5** cannot be activated after **Role 4**.

For simplicity, we assume that the user pool contains one or more Employee, but CEO, Supervisor, and Financial Manager are single users. These assumptions reflect a typical small or medium company situation.

4.2 Reachability Property

In order to ensure that a state space is fully verifying a state reachability property, we introduce in the Business Workflow an auxiliary unreachable (in the presence of RBAC) state, s25 and check its reachability. We connect the newly added state s25 to the state s23 via transition labeled by an user action, called ReachTest, which is not part of any role and is, thus, not executable.

4.2.1 Property Formulation

The reachability property is expressed in a negative form as Linear Temporal Logic formula "<> (state==s25)". The formula expresses that, in any execution, Business workflow execution should never reach state s25. As this state is indeed unreachable, Spin is forced to explore the complete state space of the model, and we can, thus, determine the maximal state space size for any reachability property on this Business Workflow.
4.2.2 Results

Figure 7 illustrates the impact of the RBACs (simple RBAC1 and complex RBAC2) on the numbers of users that the model checker can handle and the corresponding size of the explored state space.

RBAC2, by restricting the assignments of users to roles, limits the number of possible executions of the Business Workflow and thus reduces the state space explored for a given number of users. While eight users can be handled with RBAC1, this limitation is relaxed to nine users for RBAC2.

4.3 Separation of Duties Property

We define the following separation of duties property: "No one can perform both CreatePR and ReceivePaymentConf". The property relates to actions that are "distant" from each other in the Business Workflow and belong to Role 1 and Role 5. RBAC1 has only one rule which prevents the violation directly, by forbidding someone to play both Role 1 and Role 5. No rule of RBAC2 prevents such the violation of the SoD, but the property is nevertheless respected due to the interplay of the user-to-role assignments and the rules of the RBAC2. This
forces Spin model checker to explore the full state space. The obtained data correspond to the worst case situation: the number of states explored when a violation occurs is typically inferior to that reported in the following experiments. Next we consider property specification using a history array and LTL.

### 4.3.1 Property Formulation

Using history array, the property is easily formulated: the cell of the history array representing the user identifier that performed an action should not have the same content as the cell representing the user identifier that performed another action, provided that this content is not null (if the identifier is null, no user has performed the action). We specify in PROMELA the following property, which should never become true:

\[
\text{history[CreatePR]} == \text{history[ReceivePaymentConf]} \quad \text{and} \\
\text{history[CreatePR]} \neq 0
\]

Using LTL, we formulate that any user having executed an action \( a \) should not execute another action \( b \) in the same execution. We introduce two auxiliary variables, lastUser and lastAction, that contain the identifier of the last user that has performed an action and the identifier of the action. The formula for the \( i \)-th user becomes:

\[
<>((\text{lastUser}==i)\text{and}(\text{lastAction}==\text{CreatePR})) \rightarrow \\
!<>(\text{lastUser}==i)\text{and}(\text{lastAction}==\text{ReceivePaymentConf})
\]

For \( n \) users, we use a conjunction of \( n \) such formulae.

### 4.3.2 Results

We illustrate in Figure 8 how the state space explored verifying the SoD property formulated using LTL and a history array with RBAC1 and RBAC2 depends on the number of users. Model checking with RBAC2 can handle seven users for both property specifications (history array and LTL), while model checking with RBAC1 only handles five or six, depending on the formulation. The difference in the number of users between the two RBACs is two for the history array property (five users with RBAC1 and seven with RBAC2) and one for the LTL property (six and seven users, respectively). In both cases, the use of the LTL property is more efficient than the history array, either in terms of the maximum number of users (for RBAC1) or in terms of the size of the explored state space for a given number of users (for both RBACs).
Note that the maximum number of users is still relatively low for real-life applications (the number of users between 10 and 20 would be more reasonable for a given Business Workflow and typical SME). We next study several ways for alleviating the state explosion problem in model checking the Business Workflow and RBAC.

5. ALLEVIATING STATE EXPLOSION

While we target reachability and SoD properties, most of the techniques alleviating state explosion described in this work apply directly to arbitrary safety properties on actions.

We consider five types of techniques, which are related to

- data encoding (an efficient data representation),
- property reformulation,
- RBAC restriction,
• users’ activities restriction, and
• Business Workflow reduction.

An economical representation of data reduces memory consumption in model checking. One technique for achieving this is bit coding of an array of Boolean variables (in our case, the array represents the current roles activated by users). The technique does not lead to any reduction in the size of the explored state space, though it reduces the amount of memory needed to explore a given state space. It is property and RBAC independent.

Property reformulation is a simplification of the property, taking into account a given Business Workflow. In particular, we demonstrate how one can, in some cases, take advantage of the specifics of a Business Workflow to reformulate another, equivalent, but easier to verify, property. The technique is relevant mainly to SoD properties.

RBAC restriction pertains to the representation of the RBAC rules, taking into account a given Business Workflow and safety property (namely, a SoD). We elaborate a technique of restricting role activations to fewer states, reducing the complexity of model checking while preserving correctness of the results. It could also be applied to cardinality based property, but details are left for future study.

Techniques based on users’ activities restriction constrain the roles activation defined by the RBAC and allowed actions of such roles in order to further restrain model checking to executions, in which the violation may happen. The proposed technique addresses specific users, contrary to the previous one which was defined at the role activation level. The technique targets properties in terms of actions of a single user. More work is needed to adopt the technique for properties addressing actions of several users (such as cardinality constraints).

Finally, techniques addressing Business Workflow reduction modify its representation, while taking into account the property to verify and the associated RBAC, in order to produce a smaller Business Workflow, preserving correctness of the results. Two such techniques are proposed, which differ in tool support and complexity. They apply to safety properties on actions and state reachability. Extension toward cardinality properties may not be trivial.

5.1 Data Encoding

The activation of a given role for a specific user can be modeled by a Boolean variable, and activation of several roles for a user by a Boolean array. The role activation array is the most memory-consuming variable of the user EFSMs and thus, the primary target for memory reduction. Unfortunately, the PROMELA
Boolean variables require eight bits of memory, instead of only one. Thus, we apply a bit coding technique [Ruys, 2000] representing eight roles activations by a single byte value (several bytes would be needed for a number of roles greater than eight). Each bit of the byte represents the activity of a role. Access to individual bit requires bit masking. Bit masks are values that are the power of two (1, 2, 4, 8). The technique affects the low level representation of data in PROMELA; however it does not affect the role activation model itself. While the technique does not reduce the state space of a system, it reduces the state size and, thus, the memory consumption, which, in turn, partially alleviates the state explosion problem.

5.1.1 Experimental Evaluation

The compact role representation slightly reduces the memory consumed by the model checker for the reachability property (see Figure 9), and this improvement allows the model checker to handle one extra user for RBAC2. Considering that the memory consumption grows exponentially with the number of users, this result is nevertheless essential.

![Figure 9 State Space Explored for Reachability Property Using Data Encoding](image)
The data with the SoD shown in Figure 10 also demonstrate a slight improvement in memory consumption, but not in terms of the number of users.

![Figure 10 State Space Explored for Separation of Duties Property Using Data Encoding](image)

5.1.2 Discussion

The data encoding technique results in a moderate improvement: at best, it can only allow the model checker to handle one extra user. Given the fact that the memory consumed by the model checker grows exponentially with the number of users, this result is nevertheless essential. Moreover, this technique is completely independent of Business Workflow, RBAC or property and, thus, should always be used.

5.2 Property Reformulation

Given a specific Business Workflow, some LTL properties can be replaced with simpler properties equivalent for the particular Business Workflow. As an example, a simple SoD property forbidding a user $u$ to perform two actions $a$ and $b$ is stated in LTL as follows:
◊ u(a) -> ! ◊ u(b)

For a Business Workflow, which forbids any occurrence of a before b, it could be replaced with:

□ (u(a) -> ! ◊ u(b))

The first formula rejects any execution containing either the sequence a…b or b…a, while the second only matches the sequence a…b and will be verified faster. The same approach can be applied for a set of users, instead of a given user, by building a LTL formula which is an enumeration of such single-user formula. The approach requires model checking or static analysis of the Business Workflow in order to determine if a can occur before b, which, however, is less computationally expensive than the verification of RBAC along with Business Workflow\textsuperscript{8}. Such technique is relevant to separation of duties properties, but not to reachability properties.

5.2.1 Results

![Figure 11 State Space Explored for Separation of Duties Property Using Property Reformulation](Image)

\textsuperscript{8} By taking the RBAC into account, one may find additional opportunities to apply such technique.
Data shown in Figure 11 indicate that for RBAC2, reformulating property allows the model checker to handle one extra user. For the RBAC1, it more than halves the number of states explored, but this is not sufficient in this case to handle more users.

5.2.2 Discussion

The property reformulation technique targets action based SoD properties. However, it could be extended to role based SoD properties (such as “a user should not activate both Role 1 and Role 2 in a given execution”): the preliminary verification checks if one of the sequences of activations “Role 1 then Role 2” or “Role 2 then Role 1” is excluded by the Business Workflow and if so, the property is then reformulated to be checked only in the remaining feasible sequence. In a similar manner, the check is performed on the Business Workflow only, without taking into account the RBAC or any user pool. The technique is not applicable to simple cardinality based properties (such as “no user should activate more than two roles at the same time”), which have to be verified in all the states of the Business Workflow. Complex properties, involving cardinality and order dependency (such as “no user should activate more than two roles and Role 4 among them in any execution”), could also be targeted by the technique, but more work is required to adapt it.

5.3 RBAC Restriction

The idea is close to role pruning applied in the context of insider threat assessment in [Jha et al, 2005]. It consists in forbidding some activation of certain roles in given states. We present the technique first for a Business Workflow with non interleaved role activation and then generalize the approach for interleaved role activation.

5.3.1 Non Interleaved Role Activation

We first consider a Business Workflow, in which there is no interleaving of roles, i.e., on each path, between two actions of the same role, no other action of another role can be executed. For instance, a scenario when Role 1 is activated and then all its actions are completed before Role 2 is activated, then Role 3, 4, and so on, until the end of the Business Workflow, respects such constraint. We also assume that all roles in preconditions are negated. Then, activation of a given role could be restricted to a set of states, where each state is the first state of a path of the Business Workflow, whose emanating transition is labeled by an action of this role.
For our running example, this technique would restrict role activation to a single state. For instance, activation of Role 4 could be restricted to state s13. However, in a more general case, the Business Workflow contains several possible paths on which a role can be activated first in different states of the Business Workflow. We, thus, restrict the activation of such a role to several states. For instance, Role 3 can be restricted to states s7 and s10, because on any possible execution, where actions of Role 3 are executed, the first states of those executions are s7 and s10.

The idea is generalized in the Section 5.3.3 to the case of interleaved roles and arbitrary rules.

**5.3.2 Results**

As Figure 12 and Figure 13 show, RBAC restriction technique reduces by up to a third the number of explored states, and this improvement allows one extra user to be handled in the case of RBAC2 for the reachability property.
5.3.3 Interleaved Role Activation

We generalize the approach of restricting RBAC to arbitrary Business Workflow, roles, rules, and verification of reachability properties and properties on actions. In our RBAC definition in Section 3.1, role activations in a given state are restricted to roles associated with actions on the outgoing transitions of the state. Here we iteratively suppress some role activations following the rule:

For a state $s$ activation of role $R$ is forbidden if on each path to $s$, there exists a state $s'$, $s' \neq s$, such that

- activation of role $R$ is allowed in $s'$, and,

- after state $s'$, in all the states of the path, roles with preconditions containing $R$ negated an odd number of times, are forbidden.

To illustrate the procedure we consider a simple Business Workflow in Figure 14.
Figure 14 A Business Workflow with Interleaved Roles

Assume that each role contains just one action, \textit{Role 1} - \textit{a}1, \textit{Role 2} - \textit{a}2, \textit{Role 3} - \textit{a}3, and \textit{Role 4} - \textit{a}4. RBAC forbids the user to activate of \textit{Role 3} when \textit{Role 4} is activated for him. Thus, activation of \textit{Role 1} could be restricted to state \textit{s}1, \textit{Role 2} to \textit{s}3, however, activation of \textit{Role 4} is allowed in all the states, except \textit{s}3, because states \textit{s}1, \textit{s}2, \textit{s}4, and \textit{s}5 have outgoing transitions labeled with the action of \textit{Role 4}.

With the procedure, described above, activations of \textit{Role 4} could be restricted to \textit{s}1 and \textit{s}4. Activation of \textit{Role 4} could be disallowed in \textit{s}2 and \textit{s}5, because on all paths, leading to \textit{s}2 and \textit{s}5, there is a state, namely \textit{s}1 and \textit{s}4, respectively, where activation of \textit{Role 4} is allowed, and activation of \textit{Role 3} (preconditions of which involve negated \textit{Role 4}) is not allowed on the rest of the path. Note, activation of \textit{Role 4} cannot be forbidden in \textit{s}4. If one disallows activation of \textit{Role 4} in \textit{s}4, certain safety properties, such as «no user could perform actions \textit{a}3 and \textit{a}4 in the same execution», held in the original model, would be violated.

More work is needed to handle rules that use variables (e.g., to express cardinality constraints).
5.3.4 Discussion

The proposed RBAC restriction technique consists in restricting role activations to certain states of the Business Workflow. In the non interleaved role activation case, role activation is allowed only in those states, where the role can be activated first, for at least one path of the Business Workflow. A finer technique, based, for example, on replacing states by their common predecessor state, may further reduce the number of such states. More research is needed to elaborate and evaluate such refinements.

5.4 Users’ Activities Restriction

The idea of restricting users’ activities is to impose additional constraints to disallow role activations and actions of specific users that are not relevant to a property to be checked. Restrictions should exclude from exploration those global states of the Business Workflow and RBAC model, where the property cannot be violated. The idea is applicable at the least to the following case:

- Property could be formulated in terms of actions of a single user, i.e., the first order LTL [Emerson, 1990] formalization of the property contains at most one quantification over users (for instance, a simple SoD property), and
- each action of the SoD property cannot be executed more than once in any given execution.

In such case, we introduce a new constraint that prevents a user unable to execute all mutually exclusive actions of the SoD property (e.g., due to role assignment) from actually performing any of them. The reduction of the number of possible actions for a user reduces the state explosion. Furthermore, if all actions of a given role $R$ for a given user cannot be executed due to the newly introduced constraints, we then introduce another constraint that prevents the user to even activate $R$, further reducing state explosion.

For our SoD property example, we introduce the following two constraints:

- User 3, Financial Manager is prevented from executing action `ReceivePaymentConf` of Role 5. The constraint is enforced by an additional predicate on the guard of the transition labeled by this action in the EFSM of the user, whose role is Financial Manager.
- Users Employees are prevented from executing action `CreatePR`. Such action is the only action of Role 1 and only this role, so the role activation can be disallowed instead. An additional predicate is added to the guard of the transitions corresponding to activation of Role 1 in all the EFSM of users, whose roles are Employee.
5.4.1 Results

The data in Figure 15 correspond to the users’ activities restriction technique applied to our running “Procure to Stock” example. The improvement in terms of number of users handled is 33% for RBAC1 (eight users instead of six) and 57% for RBAC2 (11 users instead of seven).

![Figure 15 State Space Explored for Separation of Duties Property Using Users’ Activities Restriction Technique](image)

5.4.2 Discussion

The users’ activities restriction technique is relevant to action based SoD properties, but not to reachability properties, because it relies on the actions defined in the SoD. Extension to role based SoD properties and cardinality properties could be considered in a future work. The technique is efficient and allows the model checker to handle between two to four additional users.
5.5 Combining State Explosion Alleviating Techniques

In this section, we evaluate the cumulative effects of all the above elaborated techniques for fighting state explosion: data encoding, property reformulation, RBAC restriction and users' activities restriction. The state space explored for our running example and the SoD property are represented in Figure 16. For RBAC1 the number of users handled by the model checker within 1Gb of memory is increased from six to eleven and from seven to fourteen for RBAC2. Compared to the model checking of the Business Workflow together with RBAC1, the use of all the techniques results in the reduction of the state space by 98.79% for the same number of users (at most six in this case). Similarly, for RBAC2, the reduction is 99.56% for seven users. Considering that the state space is increasing exponentially with the growth of the number of users, doubling the number of users handled in model checking is an important result.

![Figure 16 State Space Explored for Separation of Duties Property Using All Techniques](image)

5.6 Business Workflow Reduction

A simple idea of reducing a Business Workflow is to prune parts of its EFSM model irrelevant to a property to be verified, while avoiding pruning states or transitions, where the property can be falsified. One may then try to further merge adjacent states irrelevant to the property.
5.6.1 Business Workflow Pruning

5.6.1.1 Pruning Technique

Pruning the model of a given Business Workflow is, in fact, a kind of slicing techniques, generally known and well understood. In our running example, the reachability property considered in the previous sections is about state $s_{25}$, so we can prune paths not leading to $s_{25}$.

For a simple SoD property such as

$$\forall u \Diamond u(a) \rightarrow ! \Diamond u(b),$$

where $u$ is a user and $u(a)$ denotes that the user $u$ performs action $a$, a transition can be removed from Business Workflow, if it does not belong to any path, on which both actions $a$ and $b$ occur.

For simple SoD properties, the pruning could be performed by a static analysis, in a manner similar to slicing procedures. For more complex properties, model checking of the Business Workflow could be used. Users are abstracted from such analysis: the pruning is only based on the presence or absence of specific actions and on their order on paths of the Business Workflow. Such verification can, thus, be performed with much less stringent memory requirements than that of the BS with the complete user pool. Whenever the conjunction of the abstract property and the requirement to execute a transition does not hold for the Business Workflow, the transition should be abstracted. If as a result of transition pruning, some states become unreachable, they could also be removed. The approach is simpler than CEGAR, since no counterexample is used to refine a model, and model checking of an abstract property on an abstract model (Business Workflow) leads to a straightforward simplification rather than to a refinement of Business Workflow.

We obtain, for the reachability and SoD property, the pruned Business Workflow (because both properties can be falsified on the same sub graph of the Business Workflow), shown in Figure 17.
Figure 17 Pruned Business Workflow
5.6.1.2 Results

Data of Figure 18 and Figure 19 show that even when the pruning is limited to few states, it still allows handling an additional user for RBAC2. Results with RBAC1 do not change in terms of the number of users, but slightly improve in terms of the number of states.

![Figure 18 State Space Explored for Reachability Property Using Business Workflow Reduction](image-url)
5.6.1.3 Discussion

In our Business Workflow, pruning does not significantly reduce the number of explored states, due to the small number of pruned states. The technique could, perhaps, be more efficient for Business Workflow and properties in case of a more substantial pruning. In the running example, it could be the case for verification of reachability of state $s_3$.

5.6.2 States Merging

5.6.2.1 Merging technique

In this section, we consider merging of adjacent states that are not relevant to a property for a given RBAC. The goal is to remove from the Business Workflow states and transitions that cannot have any impact on the verification of the considered property. This technique can be applied for both, SoD and reachability, properties.

If a given role does not appear in the pre- or post- conditions of any rule of the RBAC, the actions of the role are not involved in the property definition and do not affect variables, used in the guards (directly or indirectly), then those actions do not have any impact on the execution flow of the rest of Business Workflow.
(for instance, they cannot stop the execution) and are irrelevant to verification of the property.

Therefore, transitions that correspond to actions of irrelevant roles can be stripped from labels, i.e., labeled as internal, tau-transitions. Note that removing guards may render abstraction imprecise, though conservative. Then, states connected only by unlabeled transitions can be merged. Note that instead of such iterative state merging, a standard technique for determinizing non-deterministic automata can be applied. If the guards and variable updates in the remaining transitions of the Business Workflow depend on variables in deleted transitions, further abstraction is required.

Applying the outlined technique to our working example with RBAC1 and the SoD property, we obtain the reduced Business Workflow for RBAC1, shown in Figure 20.

![Figure 20 Business Workflow after State Merging](image)

The technique somewhat resembles slicing, though slicing typically does not merge states. In our experiments, with a proper action coding, Spin slicer suggests removing only very few actions, and was not able to perform the pruning presented here.

If we use both, pruning and merging, techniques in our example, we obtain the Business Workflow, shown in Figure 21.
The reduced Business Workflow is then used for property verification. Assuming a next-free safety property, if no violation is found in the reduced Business Workflow, then no violation exists in the original one, but the converse is not true. In our case, the abstraction is precise with respect to the two considered properties, since any existing execution path in the abstracted Business Workflow is represented in the original one (pruned in previous section).

However, in a more general case, should a violation be found, it must be confirmed in the original Business Workflow. The approach is, thus, similar to the CEGAR approach (though no iterative refinement is actually performed for now).

We plan to eventually extend the technique to handle more complicated rules, such as those of RBAC2, as, in the current form, this technique cannot be applied to it in our example, because such RBAC affects all the roles. Such technique is, therefore, suits better RBAC with a low number of rules, which are usually the most difficult to verify using a model checking approach, as our experimental data show.

5.6.2.2 Results

As shown in Figure 22, state merging considerably increases the number of users that can be handled, namely from eight to eighteen for RBAC1 for the reachability property and from six to fourteen for the SoD property.
5.6.2.3 Discussion

The experiments with the reachability property verification indicate that, while RBAC2 ("Complex RBAC") requires exploration of fewer number of states for any given number of users than RBAC1, this is no longer the case, as soon as the Business Workflow has been reduced. The obtained data indicate that verification with RBAC1 and reduced Business Workflow requires fewer states than RBAC2. The explanation is as follow: only Role 1 and Role 5 are kept in the Business Workflow reduced by the proposed technique, and RBAC1 is the only RBAC, constraining activations of the Role 5 after Role 1. While RBAC2 contains more rules than RBAC1, rules of RBAC2 are no longer relevant to the reduced Business Workflow, because they all pertain to the roles absent in the reduced Business Workflow.

We note that, while the Business Workflow reduction and the states merging techniques can be used with RBAC1, this is not the case with RBAC2, for which only the first one can be used (application of state merging with RBAC2 results
in no further improvement). Finally, the technique only guarantees that when no violation is found in the reduced model, the original model has no violation either.

5.7 Conclusion

We elaborated several techniques to combat the state explosion problem: data encoding, property reformulation, RBAC restriction, users’ activities restriction, and Business Workflow reduction. Data encoding is independent of any rules, user pool, and property to verify. The technique can reduce by one third the memory consumption, allowing to handle, at best, one extra user during model checking. Our experimental data show that the most efficient technique is the Business Workflow reduction, it reduces by up to 99.9% the number of explored states and doubles the number of handled users. However, the technique is property and rule dependent. Several techniques can be applied together, and our experiments show that the first four techniques double the number of handled users. More work needs to be done to study interactions of various techniques, the range of Business Workflows, rules, and properties, on which they can be applied.

6. DISCUSSIONS

6.1 Handling Unbounded Number of Users

Our approach deals with a finite number of users. While RBAC is rarely applied for large companies, it is still interesting to verify a property for an arbitrary number of users. The reachability properties are monotone, in the sense that if one adds users, the reachable states of the model stay reachable by definition. However, combination of reachability and minimum cardinality properties, for instance, a reachability property combined with a cardinality property, such as “all users should be involved” would no longer be necessarily monotone. In this case, the property could be violated, if the number of users exceeds the maximum length of the path leading to the desired state. SoD properties could be violated by adding users: with more users some new paths, which violate a property, could be added. It is easy to see that for simple SoD constraints and properties, if a property holds for a number of users that exceeds or equals the number of roles (and each role is assigned at the least to one user), the property holds for any number of users. However, it may not be the case for cardinality and order-dependent constraints. While it is difficult to build such upper limits on the number of users in the most general setting, it could be done for typical RBAC constraints properties.
The study of RBAC composability could help in solving these issues. However, so far the problem of property preserving RBAC user pool and policy composition is studied [Gligor et al, 98] only with a simplified assumption of unordered scenarios (when actions of Business Workflow could be executed in any order).

6.2 Role Activation Alternatives

Role activation in this work is restricted to states in which at least one emanating transition is labeled with an action of the given role. Two other obvious options are to allow role activations for all the roles in every state and to allow them only in specific states (such as the initial state of the Business Workflow). While the later choice is very restrictive, the first one seems more interesting.

Note that counterexamples to a (LTL) property on the considered role activation mode are always counterexamples for abovementioned alternative modes, though the converse is not true. For instance, while in our running example SoD property holds for RBAC1, the SoD property would be violated on RBAC1 with any of the two other role activation alternatives.

Still, the techniques suggested in this work could be extended toward the abovementioned options and possibly several other RBAC role activation modes as well. Indeed, the problem of verification of a RBAC with state-independent role activation could be easily encoded into verification of a RBAC with our role activation mode. Perhaps, the most straightforward encoding is to add to each role a designated user action \textit{nop} (no-operation) and define in each state of the Business Workflow a looping transition, labelled with action \textit{nop} and a guard that always evaluates to false (such as “1=0”).

6.3 Further Means for Fighting State Explosion

Apparently, an efficient technique for fighting state explosion is to exploit symmetry for users with similar roles to reduce the user pool necessary for property verification. While symmetry was already used to simplify properties, it might be even more useful for model simplification. Some of the proposed techniques could further be refined based on the user symmetry. We may broaden the scope of application of the proposed techniques to other kinds of rules and properties (cardinality, information leakage, for instance). The use of Counter-example Guided Abstraction Refinement (CEGAR) is not yet explored in our work. At the same time, we believe that some ideas stated in Section 5.2 could be used to improve CEGAR process itself, by allowing not only an
iterative refinement of the specification abstraction, but also simplification of a property to be verified (at least, for model checking frameworks that involve evaluation of each sub-formula in each state).

7. CONCLUSION AND FUTURE WORK

We proposed several approaches to model check RBAC on Business Workflow and suggested techniques for alleviating state explosion inherent to real-life Business Workflows and RBAC to improve the scalability of the model checking approach. A somewhat surprising finding is that a larger set of constraints is easier to model check, due to a stricter restriction it imposes on possible user activities, and, thus, reducing the state space. The proposed techniques are experimentally evaluated on an abstract Business Workflow, which possesses important features of real-life Business Workflow, such as presence of input parameters, messages exchanges, non-determinism, and loops. We, thus, believe that the obtained results can be applied to a wide range of real Business Workflows. It would be interesting to assess the proposed modeling approaches and state explosion alleviating techniques on Business Workflows supported by typical SAP applications.
REFERENCES


P. Cousot, R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixedpoints. 4th POPL, 1977. ACM.


D. Dams, O. Grumberg, R. Gerth. Generation of reduced models for checking fragments of CTL. CAV 93. LNCS 1066.


D. Drusinsky. The temporal rover and the ATG rover. SPIN Model Checking and Software Verification. LNCS 1885. 2000.


NIST Role Based Access Control Implementation Standard. Version 0.1
January 2006.

J. Rubio-Loyola, J. Serrat, M. Charalambides, P. Flegkas, G. Pavlou. A
functional solution for goal-oriented policy refinement. Seventh IEEE
International Workshop on Policies for Distributed Systems and Networks
(POLICY’06), 2006.

T.C. Ruys. Low-fat recipes for Spin, Spin model checking and software
verification, 7th International SPIN Workshop, Springer LNCS no 1885, 2000.

A. Schaad, J. Moffett. A lightweight approach to specification and analysis of
role-based access control extensions. 7th ACM Symposium on Access Control


B. Shafiq, J.B.D. Joshi, E. Bertino. A. Ghafoor. Secure interoperation in a
multidomain environment employing RBAC policies. IEEE Transactions on
Knowledge and Data Engineering. 17(11): 2005.

B. Shafiq, A. Masood, J. Joshi, A. Ghafoor. A role-based access control policy
verification framework for real-time systems. 10th IEEE Intl. Workshop on

Extended RBAC and the coloured Petri Net modeling. IEICE Transactions. E88-

R.L. Smith, G.S. Avrunin, L.A. Clarke. From natural language requirements to
rigorous property specifications. Monterey Workshop on Software Engineering

composition of access control and application features. 10th ACM Symposium

G. Spafford. Sergegate duties to lessen security risks. Datamation. January,
2006. (online) http://itmanagement.earthweb.com/

J.E. Tidswell, T. Jaeger. An access control model for simplifying constraint
expression. 7th ACM conference on Computer and Communications Security,
2000.

Languages. 1995.


Brown University PLT. Margrave. An API for XACML policy verification and change analysis (online) http://www.cs.brown.edu/research/plt/software/margrave/.

Bandera http://bandera.projects.cis.ksu.edu/.


Woflan tool (on-line) http://is.tm.tue.nl/research/woflan/.

TATD (online) http://wwwhome.cs.utwente.nl/~tcm/tatd.html.


Model-Checking Kit http://www.fmi.uni-stuttgart.de/szs/tools/mckit/