

Model Checking of Security-sensitive Business Processes^{*}

Alessandro Armando and Serena Elisa Ponta

DIST, Università di Genova, Italy
{armando,serena.ponta}@dist.unige.it

Abstract. Security-sensitive business processes are business processes that must comply with security requirements (e.g. authorization constraints). In previous works it has been shown that model checking can be profitably used for the automatic analysis of security-sensitive business processes. But building a formal model that simultaneously accounts for both the workflow and the access control policy is a time consuming and error-prone activity. In this paper we present a new approach to model checking security-sensitive business processes that allows for the separate specification of the workflow and of the associated security policy while retaining the ability to carry out a fully automatic analysis of the process. To illustrate the effectiveness of the approach we describe its application to a version of the Loan Origination Process featuring an RBAC access control policy extended with delegation.

1 Introduction

A business process is a set of coordinated activities carried out concurrently by different entities and using a set of resources with the aim to achieve a goal or to deliver a service. The design and development of business processes is a non trivial activity as they must meet several contrasting requirements, e.g. the compliance with mandatory regulations and the ability to support a wide range of execution scenarios.

Security-sensitive business processes are business processes in which security requirements play a significant role. In this paper we focus on security requirements on authorization. Failure to meet authorization constraints may lead to economic losses and even to legal implications. The evolution from static, well established processes to dynamic ones—a current trend in the development of business processes—may seriously affect their security and often this occurs in subtle and unexpected ways. As an example consider a business process in which agents can be dynamically delegated to perform tasks they were not initially authorized to execute. This is desirable as delegation provides additional flexibility to the process, but it also offers new ways to circumvent security. Since these

^{*} This work was partially supported by the FP7-ICT-2007-1 Project no. 216471, “AVANTSSAR: Automated Validation of Trust and Security of Service-oriented Architectures” (www.avantssar.eu).

kinds of vulnerabilities are very difficult to spot by simple inspection of the workflow and of the associated security policy, security-sensitive business processes are a new, promising application domain for formal methods.

Model checking is a technique for the automatic analysis of concurrent systems. Given a formal model of the system formally specified by a finite transition system and the expected property specified by a formula in some temporal logic, e.g. LTL, a model checker either establishes that the system enjoys the property or returns an execution trace witnessing its violation.

Model checking has been remarkably successful in many key application areas, such as hardware and protocol verification and, more recently, software and security protocol verification. A natural question is whether model checking can be profitably used for the automatic analysis of security-sensitive business processes. Previous works [1–3] provide a positive answer to this question by showing that business processes under authorization constraints can be formally specified as transition systems and automatically analyzed by model checkers taken off the shelf. However the manual definition of the transition system starting from the workflow and the associated security policy is a complex and error-prone activity, which—if not carried out correctly—may undermine the significance of the whole method.

In this paper we present a new approach to the specification and model checking of security-sensitive business processes that comprises the following steps:

1. formal modeling of the security-sensitive business process as an access-controlled workflow system;
2. formal modeling of the expected security property as an LTL formula ϕ ;
3. automatic translation of the access-controlled workflow system into a planning system, a formal framework amenable to automatic analysis; and
4. model checking of the planning system to determine whether it enjoys ϕ .

An *access-controlled workflow system* is a formal, yet natural framework for specifying security-sensitive business processes and results from the combination of a workflow system and of an access control system. A *workflow system* supports the specification of the control flow of the business process by extending Petri Nets [4] through a richer notion of state that accounts not only for the concurrent execution of the tasks but also for the effects that their execution has on the global state of the system. An *access control system* provides a declarative, rule-based language for specifying a wide variety of security policies and operations for updating them. Our approach thus facilitates the modeling activity, while supporting a fully automatic analysis of the process.

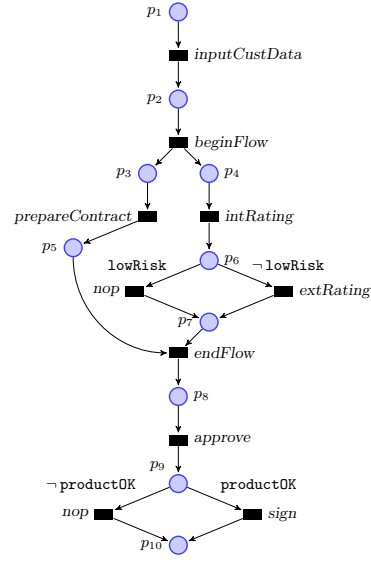
To illustrate the effectiveness of the approach we have applied it against a version of the Loan Origination Process (LOP) that features an RBAC access control policy extended with delegation. By using SATMC [5, 6], a model checker for planning systems, we have detected serious flaws in our original specification of the LOP and this has led us to the definition of a new, improved version of the business process.

```

<process name="LOP" .../>
  <sequence>
    <invoke> inputCustData </invoke>
    <flow>
      <invoke> prepareContract </invoke>
      <sequence>
        <invoke> intRating </invoke>
        <if>
          <condition>  $\neg$ lowRisk </condition>
          <invoke> extRating </invoke>
        </if>
      </sequence>
    </flow>
    <invoke> approve </invoke>
    <if>
      <condition> productOK </condition>
      <invoke> sign </invoke>
    </if>
  </sequence>
</process>

```

(a) BPEL Program



(b) Workflow System

Fig. 1: Loan Origination Process

Our approach improves upon existing works on model checking of business processes by simultaneously supporting:

- the separate specification of the workflow and of the associated security policy;
- the formal and declarative specification of a wide range of security policies;
- the specification of tasks with non-deterministic effects;
- LTL to specify complex security properties at the level of access-controlled workflow system, a higher level than that provided by transition systems;
- full automation of the analysis.

To the best of our knowledge no other model checking platform encompassing all the above features exists.

2 Security-Sensitive Business Processes

Let us consider the BPEL [7] specification of the Loan Origination Process (LOP) given in Fig. 1a. The process starts with the input of the customer's data (`inputCustData`). Afterwards a contract for the current customer is prepared (`prepareContract`) while the customer's rating evaluation takes place concurrently. The rating enables the bank to determine whether the customer can be granted the requested loan. To this end, the execution may follow different

Table 1: Permission assignment for the LOP

Task	Role
<code>inputCustData</code>	<code>preprocessor</code>
<code>prepareContract</code>	<code>postprocessor</code>
<code>intRating</code>	<code>if (isIndustrial) then supervisor else postprocessor</code>
<code>extRating</code>	<code>supervisor</code>
<code>approve</code>	<code>if (isIndustrial) then manager else supervisor</code>
<code>sign</code>	<code>if (intRatingOK) then manager else director</code>

paths: if the risk associated with the loan is low (`lowRisk`), then an internal rating suffices (`intRating`); otherwise the internal rating is followed by an external evaluation (`extRating`) carried out by a Credit Bureau, a third-party financial institution. The `lowRisk` condition indicates a situation in which the internal rating is positive and the amount of the loan is not high. The loan request must then be approved (`approve`) by the bank. Subsequently, if the customer and the bank have reached an agreement, the contract is signed (`sign`). Notice that the execution of a task may affect the state of the process. For example, the task `approve` modifies the state of the execution by issuing a statement asserting if the proposed product is suitable or not for the customer.

An agent can execute a task only if she has the required permissions. As it is common in the business domain, the LOP relies on an access control model based on RBAC [8] extended with delegation. According to the RBAC model, to perform a task an agent must be assigned a role that is enabled to execute the task and the agents must be also active in that role. The roles used in our case study are given in Table 1 together with the tasks they are enabled to execute. Roles can be organized hierarchically. In our case study, a `director` is more senior than a `manager` and a `supervisor` is more senior than a `postprocessor`. Senior roles inherit the permission to perform tasks assigned to more junior roles. As a consequence, an agent can execute a task if her role (*i*) is directly assigned the required permissions or (*ii*) is more senior than a role owning such permissions. The permission assignment relation in Table 1 associates each task of the LOP with the most junior role entitled to execute it.

Following the idea of conditional delegation presented in [9], we consider *delegation rules* of the form: $\langle PreConds, ARole, DRole, Task \rangle$, where *ARole* and *DRole* are roles, *Task* is a task, and *PreConds* is a set of conditions that must hold for the delegation to be applicable. A delegation rule states that if *PreConds* holds and *ARole* is authorized to perform *Task* according to the permission assignment relation, then *ARole* can delegate *DRole* to execute *Task*. Notice that this is a task delegation rather than a role delegation. In fact, the delegated agent does not acquire a new role but she only obtains the permission to perform *Task* by means of *ARole*. Examples of delegation rules considered in our case study are:

- D1: $\langle \text{intRatingOK}, \text{manager}, \text{supervisor}, \text{approve} \rangle$,
- D2: $\langle \text{intRatingOK}, \text{manager}, \text{supervisor}, \text{sign} \rangle$,

As far as the security requirements are concerned, here we focus on Separation of Duty (SoD) properties which are used for internal control and are probably the most common application-level properties that business processes must comply with. SoD amounts to requiring that some critical tasks are executed by different agents. This can be achieved by constraining the assignment of roles (*Static SoD*), their activation (*Dynamic SoD*) or even the execution of tasks [1]. In this paper we focus on *Object-based SoD (ObjSoD)* and *Operational SoD (OpSoD)*. The former requires that no agent performs all the tasks accessing the same object, while the latter requires that no agent performs all the tasks of the workflow.

Object-based SoD for the LOP. Since `intRating`, `extRating`, and `approve` access and deal with the rating of the customer they form a set of critical tasks and the LOP is thus expected to meet the following ObjSoD property: “*If the process terminates successfully, then no single agent has performed all the critical tasks (namely intRating, extRating, and approve).*”

Operational SoD for the LOP. The OpSoD for the LOP can be expressed as follows: “*An agent cannot perform all the tasks of a successful process execution.*” Notice that in this case the set of critical tasks depends on `lowRisk` whose value cannot be predicted in advance. If `lowRisk` is false then the set of critical tasks comprises all the tasks of the process, otherwise it contains all tasks but `extRating`.

3 Access-Controlled Workflow Systems

At the core of our approach lies the notion of *access-controlled workflow system*, a formal framework supporting the separate specification of the workflow of the business process and of the associated security policy in terms of a workflow system and of an access control system respectively. A *workflow system* is a Petri Net extended with a richer notion of state that accounts not only for the concurrent execution of tasks but also for the effects that their execution has on the global state of the system. An *access control system* supports the formal and declarative specification of the access control policy (stating which agent can perform which task) including advanced but often used features such as delegation. In this section we provide a detailed account of our specification framework, a trace-based semantics, and a temporal logic that allows for the formal statement of properties of business processes.

3.1 Workflow Systems

A *fact* is an atomic proposition. Let \mathcal{F} be a set of facts. A *literal over \mathcal{F}* is either a fact in \mathcal{F} or the negation of a fact in \mathcal{F} . The set of literals over \mathcal{F} is denoted by $\mathcal{L}(\mathcal{F})$. A set of literals L is *consistent* if and only if L does not contain a fact and its negation. A *formula over \mathcal{F}* is a propositional combination of facts and

Table 2: Mapping BPEL programs into workflow systems

$\langle \text{invoke} \rangle$ t $\langle / \text{invoke} \rangle$		$\langle \text{sequence} \rangle$ p_1 p_2 $\langle / \text{sequence} \rangle$	
$\langle \text{if} \rangle$ $\langle \text{condition} \rangle$ c $\langle / \text{condition} \rangle$ p $\langle / \text{if} \rangle$		$\langle \text{flow} \rangle$ p_1 p_2 $\langle / \text{flow} \rangle$	

Legenda:
 t task

 p, p_1, p_2 BPEL programs

place

transition

 c literal

 workflow associated with BPEL program p
 nop any operation s.t. $\pi(\text{nop}) = \mu(\text{nop}) = \nu(\text{nop}) = \emptyset$

the propositional constant **true** using the usual propositional connectives (i.e. \neg , \wedge , \vee , and \Rightarrow).

Let \mathcal{F} be a set of facts and \mathcal{T} be a set of transitions. A *workflow system* over \mathcal{F} and \mathcal{T} is a tuple $WS = \langle \mathcal{P}, \mathcal{F}, \mathcal{I}_{WS}, \mathcal{T}, In, Out, \gamma, \alpha \rangle$, where \mathcal{P} is a set of *places*, $In \subseteq (\mathcal{P} \times \mathcal{T})$, $Out \subseteq (\mathcal{T} \times \mathcal{P})$, γ is a function that associates the elements of In with formulae over \mathcal{F} expressing applicability conditions, $\alpha : \mathcal{T} \rightarrow 2^{\mathcal{L}(\mathcal{F})} \times 2^{\mathcal{L}(\mathcal{F})} \times 2^{\mathcal{L}(\mathcal{F})}$ is a partial function mapping transitions into sets of literals corresponding to their *preconditions* $\pi(t)$, their *deterministic effects* $\eta(t)$, and their *non-deterministic effects* $\nu(t)$ respectively, i.e. $\alpha(t) = \langle \pi(t), \eta(t), \nu(t) \rangle$, where $\pi(t)$ and $\eta(t)$ are consistent. We call *tasks* the transitions for which α is defined. If $T \subseteq \mathcal{T}$ is a set of transitions, then T_α denotes the set of tasks in T . A *marking* is a function $M : \mathcal{P} \rightarrow \mathbb{N}$. A *state of WS* is a pair (M, L) , where M is a marking and $L \subseteq \mathcal{L}(\mathcal{F})$ is a maximally consistent set of literals, (i.e. a truth-value assignment to the facts in \mathcal{F}). \mathcal{I}_{WS} is the set of *initial states* of WS .

The *preset of a transition* t , in symbols $\bullet t$, is the set $\{p \in \mathcal{P} : (p, t) \in In\}$. The *set of preconditions of a transition* t , in symbols $*t$, is the set $\{\gamma(p, t) : (p, t) \in In\}$. The *postset of a transition* t , in symbols t^\bullet , is the set $\{p \in \mathcal{P} : (t, p) \in Out\}$.

Let $T \subseteq \mathcal{T}$ be a set of transitions. We define $\pi(T) = \bigcup_{t \in T_\alpha} \pi(t)$, $\eta(T) = \bigcup_{t \in T_\alpha} \eta(t)$, and $\nu(T) = \bigcup_{t \in T_\alpha} \nu(t)$. A *step* is a set of transitions $T \subseteq \mathcal{T}$ such that $\pi(T)$ and $\eta(T)$ are consistent. A *step* T is *enabled in a state* (M, L) iff

- $L \models \gamma(p, t)$ for all $p \in \mathcal{P}$ and $t \in T$ s.t. $(p, t) \in In$, where \models is the consequence relation in classical propositional logic.
- $\pi(T) \subseteq L$, and
- $M(p) \geq \sum_{t \in T} In(p, t)$ for all $p \in \mathcal{P}$.

Notice that here and in the sequel we use $In(p, t)$ and $Out(t, p)$ to denote also the characteristic function of In and Out relations respectively.

It is worth pointing out that workflow systems extend Petri nets by providing a richer notion of state given by the pair (M, L) , that accounts not only for the concurrent execution of the component activities (by means of the marking M) but also for the effects that these activities have on the state of the system (represented by the set of literals L). For this reason workflow systems are a natural formal model for business processes. The mapping in Table 2 shows how the constructs occurring in the BPEL program of Fig. 1a can be mapped to workflow system templates. The mapping can be readily extended to support other basic activities (e.g. `<receive>`, `<reply>`) as well as more complex BPEL structured activities (e.g. `<while>` and the `<link>` construct which is used to join activities occurring in different branches of a `<flow>`).

3.2 Access Control Systems

An *access control system* over \mathcal{F} and \mathcal{T} is a tuple $ACS = \langle \mathcal{F}, \mathcal{I}_{ACS}, \mathcal{A}, \mathcal{T}, \mathcal{U}, \alpha, H \rangle$ where \mathcal{F} , \mathcal{A} , \mathcal{T} , \mathcal{U} are sets of facts, agents, tasks, and policy updates respectively, $\alpha : \mathcal{U} \rightarrow 2^{\mathcal{L}(\mathcal{F})} \times 2^{\mathcal{L}(\mathcal{F})}$ is a function mapping policy updates into consistent sets of literals corresponding to their *preconditions* $\pi(u)$ and their *effects* $\eta(u)$ respectively, i.e. $\alpha(u) = \langle \pi(u), \eta(u) \rangle$, and H is a set of rules of the form $\ell_0 \leftarrow \ell_1, \dots, \ell_n$ with $\ell_i \in \mathcal{L}(\mathcal{F})$ for $i = 0, \dots, n$ and $n \geq 0$. A *state of ACS* is a maximally consistent set of literals $S \subseteq \mathcal{L}(\mathcal{F})$ such that $H(S) \subseteq S$ where $H(S) = \{\ell_0 : (\ell_0 \leftarrow \ell_1, \dots, \ell_n) \in H \text{ and } \ell_i \in S \text{ for all } i = 1, \dots, n\}$. We assume that the set \mathcal{F} of an *ACS* always contains a fact `granted(a, t)` for all $a \in \mathcal{A}$ and $t \in \mathcal{T}$ expressing the authorization of an agent a to execute a task t . $\mathcal{I}_{ACS} \subseteq \mathcal{L}(\mathcal{F})$ are the *initial states of ACS*.

Let U be a set of policy updates. We define $\pi(U) = \bigcup_{u \in U} \pi(u)$ and $\eta(U) = \bigcup_{u \in U} \eta(u)$. A *step of ACS* is a set of policy updates $U \subseteq \mathcal{U}$ such that $\pi(U)$ and $\eta(U)$ are consistent. A *step U is enabled in a state L* iff $\pi(U) \subseteq L$.

3.3 Access-Controlled Workflow Systems

An *access-controlled workflow system* over \mathcal{F} and \mathcal{T} is a pair $AWS = \langle WS, ACS \rangle$ where WS is a workflow system over \mathcal{F} and \mathcal{T} and ACS is an access control system over \mathcal{F} and \mathcal{T}_α . A *state of AWS* is a pair (M, L) , where M is a marking of the workflow system WS and L is a maximally consistent set of literals in $\mathcal{L}(\mathcal{F})$. The *initial states of AWS* = $\langle WS, ACS \rangle$ are the initial states (M, L) of WS such that L is also an initial state of ACS . We use \mathcal{I}_{AWS} to denote the set of initial states of AWS . A *task allocation for T_α* is a total function $\lambda : T_\alpha \rightarrow \mathcal{A}$. A *step W of AWS* = $\langle WS, ACS \rangle$ is a triple (T, U, λ) where $T \subseteq \mathcal{T}$ and $U \subseteq \mathcal{U}$ such that both $\pi(T) \cup \pi(U)$ and $\eta(T) \cup \eta(U)$ are consistent and λ is a task allocation for T_α . A *step W = (T, U, λ) is enabled in a state (M, L)* iff `granted(λ(t), t)` $\in L$ for all $t \in T_\alpha$, T is enabled in (M, L) , and U is enabled in L . If a step $W = (T, U, \lambda)$ is enabled in $S = (M, L)$, then the occurrence of W in S leads to a new state

$S' = (M', L')$, in symbols $S[W]S'$. A literal l is caused by a step transition $S[W]S'$ iff at least one of the following conditions holds:

- $l \in \eta(T) \cup \eta(U)$, i.e. l is a deterministic effect of some transition or update,
- $l \in \nu(T)$ and $l \in L'$, i.e. l is a non-deterministic effect of some transition,
- there exists $l \leftarrow l_1, \dots, l_n \in H$ with $l_i \in L'$ for $i = 1, \dots, n$, i.e. l is the head of a rule in H whose body holds in L' ,
- $l \in L$ and $l \in L'$, i.e. the truth-value of l is left untouched by the occurrence of W .

A step transition $S[W]S'$ is causally explained according to AWS iff S' coincides with the set of literals caused by the occurrence of W in S . (The notion of causal explanation is adapted from [10].) An execution path χ of AWS is an alternating sequence of states and steps $S_0W_0S_1W_1\cdots$ such that $S_i[W_i]S_{i+1}$ are causally explained step transitions for $i \geq 0$. We use $\chi^s(i)$ and $\chi^w(i)$ to denote S_i and W_i respectively. An execution path χ is initialized iff $\chi^s(0) \in \mathcal{I}_{AWS}$. A state S is reachable in AWS iff there exists an initialized execution path χ such that $\chi^s(i) = S$ for some $i \geq 0$. A state (M, L) is n -safe iff $M(p) \leq n$ for all $p \in \mathcal{P}$. An access-controlled workflow system AWS is n -safe iff all its reachable states are n -safe.

Notice that the mapping of Table 2 yields 1-safe access-controlled workflow systems, provided that the initial states are 1-safe. This trivially follows from the observation that every place of the workflow system has at most one incoming edge. In the sequel we will restrict our attention to 1-safe access-controlled workflow systems.

3.4 A Logic for Access-Controlled Workflow System

Properties of access-controlled workflow system can be expressed by means of LTL formulae. Let AWS be an access-controlled workflow system. The set of LTL formulae associated with AWS is the smallest set containing \mathcal{F} , an atomic proposition for each place in \mathcal{P} , an atomic proposition $\mathbf{exec}(a, t)$ for each $a \in \mathcal{A}$ and $t \in \mathcal{T}_\alpha$, an atomic proposition $\mathbf{exec}(t)$ for each transition $t \in \mathcal{T} \setminus \mathcal{T}_\alpha$, an atomic proposition $\mathbf{exec}(u)$ for each policy update $u \in \mathcal{U}$, and such that if ϕ and ψ are LTL formulae, then also $\neg\phi$, $(\phi \vee \psi)$, $(\phi \wedge \psi)$, $(\phi \Rightarrow \psi)$, $\mathbf{X}\phi$, $\mathbf{F}\phi$, and $\mathbf{G}\phi$ are LTL formulae. Let χ be an initialized path of AWS. An LTL formula ϕ is valid on χ , in symbols $\chi \models \phi$, if and only if $(\chi, 0) \models \phi$, where $(\chi, i) \models \phi$, for $i \geq 0$, is inductively defined as follows:

- if ϕ is a fact then $(\chi, i) \models \phi$ iff $\chi^s(i) = (M, L)$ with $\phi \in L$,
- if ϕ is an atomic proposition corresponding to a place $p \in \mathcal{P}$, then $(\chi, i) \models \phi$ iff $\chi^s(i) = (M, L)$ with $M(p) \geq 1$,
- if ϕ is an atomic proposition of the form $\mathbf{exec}(a, t)$, then $(\chi, i) \models \phi$ iff $\chi^w(i) = (T, U, \lambda)$ with $t \in \mathcal{T}_\alpha$ and $\lambda(t) = a$,
- if ϕ is an atomic proposition of the form $\mathbf{exec}(o)$, then $(\chi, i) \models \phi$ iff $\chi^w(i) = (T, U, \lambda)$ with $o \in (\mathcal{T} \setminus \mathcal{T}_\alpha) \cup \mathcal{U}$,
- $(\chi, i) \models \neg\phi$ iff $(\chi, i) \not\models \phi$,

- $(\chi, i) \models (\phi \vee \psi)$ iff $(\chi, i) \models \phi$ or $(\chi, i) \models \psi$,
- $(\chi, i) \models \mathbf{X}\phi$ iff $(\chi, i + 1) \models \phi$, and
- $(\chi, i) \models \mathbf{F}\phi$ iff there exists $j \geq i$ s.t. $(\chi, j) \models \phi$.

The semantics of the remaining connectives readily follows from the following equivalences: $(\phi \wedge \psi) \equiv \neg(\neg\phi \vee \neg\psi)$, $(\phi \Rightarrow \psi) \equiv (\neg\phi \vee \psi)$, and $\mathbf{G}(\phi) \equiv \neg\mathbf{F}\neg\phi$. A formula ϕ is valid in AWS, in symbols $\models_{AWS} \phi$, iff $\chi \models \phi$ for all initialized execution paths χ of AWS.

By using LTL we can now give a formal definition of the SoD properties presented in Sect. 2.

ObjSoD for the LOP. If the process terminates successfully, then for all agents $a \in \mathcal{A}$ there exists a task $t \in T$, where $T = \{\text{intRating}, \text{extRating}, \text{approve}\}$, such that a does not perform t :

$$\mathbf{F}(p_{10} \wedge \text{productOK}) \Rightarrow \bigwedge_{a \in \mathcal{A}} \bigvee_{t \in T} \mathbf{G} \neg \text{exec}(a, t). \quad (1)$$

OpSoD for the LOP. For all agents $a \in \mathcal{A}$ there exists at least a task in the workflow that is never executed by a :

$$\bigwedge_{a \in \mathcal{A}} \left(\begin{array}{l} \mathbf{G} \neg \text{exec}(a, \text{inputCustData}) \vee \mathbf{G} \neg \text{exec}(a, \text{prepareContract}) \vee \\ \mathbf{G} \neg \text{exec}(a, \text{intRating}) \vee \mathbf{G}(\text{lowRisk} \vee \neg \text{exec}(a, \text{extRating})) \vee \\ \mathbf{G} \neg \text{exec}(a, \text{approve}) \vee \mathbf{G} \neg \text{exec}(a, \text{sign}) \end{array} \right). \quad (2)$$

4 Model Checking Access-controlled Workflow Systems

A planning system is a formal framework for the specification of concurrent systems inspired by the model used by the AI community to specify planning domains: states are represented by sets of literals and state transitions by actions, where an action is specified by preconditions (i.e. literals that must hold for the action to be executable) and effects (i.e. which literals are possibly affected by the execution of the action); the possible behaviors of the system can also be constrained by means of rules. It must be noted that the ability to specify rules and actions with non-deterministic effects goes beyond the expressiveness of traditional planning languages, e.g. STRIPS, but are supported by more recent developments [10, 11]. In this section we formally define a planning system and provide a trace-based semantics, a temporal logic for specifying properties, and a formal translation from access-controlled workflow systems to planning systems.

4.1 Planning Systems

A *planning system* is a tuple $PS = \langle \mathcal{F}_P, \mathcal{I}_{PS}, \mathcal{O}_P, \omega, H_P \rangle$, where \mathcal{F}_P is a set of facts, \mathcal{O}_P is a set of planning operators, $\omega : \mathcal{O}_P \rightarrow 2^{\mathcal{L}(\mathcal{F}_P)} \times 2^{\mathcal{L}(\mathcal{F}_P)} \times 2^{\mathcal{L}(\mathcal{F}_P)}$ is a function mapping planning operators into sets of literals corresponding to their *preconditions* $\pi(o)$, their *deterministic effects* $\eta(o)$, and their *non-deterministic*

effects $\nu(o)$ respectively, i.e. $\omega(o) = \langle \pi(o), \eta(o), \nu(o) \rangle$, where $\pi(o)$ and $\eta(o)$ are consistent, and H_P is a set of rules over \mathcal{F}_P . A *state of PS* is a maximally consistent set of literals over \mathcal{F}_P . \mathcal{I}_{PS} is the set of *initial states* of PS . If O is a set of planning operators, then $\pi(O) = \bigcup_{o \in O} \pi(o)$, $\eta(O) = \bigcup_{o \in O} \eta(o)$, and $\nu(O) = \bigcup_{o \in O} \nu(o)$. A *step* O of PS is a set of planning operators such that $\pi(O)$ and $\eta(O)$ are consistent. A *step* O is *enabled in state* S iff $\pi(O) \subseteq S$. If a step O is enabled in a state S , then the occurrence of O in S *leads to* a new state S' , in symbols $S[O]S'$. A literal l is *caused by a step transition* $S[O]S'$ iff at least one of the following conditions holds:

- $l \in \eta(O)$, i.e. l is a deterministic effect of some planning operator,
- $l \in \nu(O)$ and $l \in S'$, i.e. l is a non-deterministic effect of some planning operator,
- there exists $l \leftarrow l_1, \dots, l_n \in H_P$ with $l_i \in S'$ for $i = 1, \dots, n$, i.e. l is the head of a rule in H_P whose body holds in S' ,
- $l \in S$ and $l \in S'$, i.e. the truth-value of l is left untouched by the occurrence of O .

A step transition $S[O]S'$ is *causally explained according to PS* iff S' coincides with the set of literals caused by the occurrence of O in S . An *execution path* χ of PS is an alternating sequence of states and steps $S_0O_0S_1O_1 \dots$ such that $S_i[O_i]S_{i+1}$ are causally explained step transitions for $i \geq 0$. We use $\chi^s(i)$ and $\chi^o(i)$ to denote S_i and O_i respectively. An execution path χ is *initialized* iff $\chi^s(0) \in \mathcal{I}_{PS}$. A state S is *reachable* iff there exists an initialized execution path χ such that $\chi^s(i) = S$ for some $i \geq 0$.

Let PS be a planning system. The set of LTL formulae associated with PS can be defined analogously to the set of LTL formulae for an access-controlled workflow system (cf. Sect. 3.3) by using facts and planning operators as atomic propositions. The validity of an LTL formula ϕ of PS on an execution trace also closely follows the one given for access-controlled workflow systems. Finally, we say that ϕ is *valid in PS*, in symbols $\models_{PS} \phi$, iff $\chi \models \phi$ for all initialized execution paths χ of PS .

4.2 From Access-controlled Workflow Systems to Planning Systems

Let $AWS = \langle WS, ACS \rangle$ be an access-controlled workflow system over \mathcal{F} and \mathcal{T} with $WS = \langle \mathcal{P}, \mathcal{F}, \mathcal{I}_{WS}, \mathcal{T}, In, Out, \gamma, \alpha \rangle$ and $ACS = \langle \mathcal{F}, \mathcal{I}_{ACS}, \mathcal{A}, \mathcal{T}_\alpha, \mathcal{U}, \alpha, H \rangle$. The *planning system associated with AWS* is $PS = \langle \mathcal{F}_P, \mathcal{I}_{PS}, \mathcal{O}_P, \omega, H_P \rangle$, where \mathcal{F}_P is obtained from \mathcal{F} by adding a new fact for each place in \mathcal{P} ,¹ \mathcal{I}_{PS} contains a state $L \cup \{p : M(p) = 1\} \cup \{-p : M(p) = 0\}$ for each initial state (M, L) of AWS , $H_P = H$, \mathcal{O}_P contains

- a planning operator $\mathbf{exec}(a, t)$ for each $a \in \mathcal{A}$ and $t \in \mathcal{T}_\alpha$,
- a planning operator $\mathbf{exec}(t)$ for each transition $t \in \mathcal{T} \setminus \mathcal{T}_\alpha$,
- a planning operator $\mathbf{exec}(u)$ for each policy update $u \in \mathcal{U}$,

¹ We will not bother distinguishing between a place and the corresponding fact.

and ω is such that:

- for all $a \in \mathcal{A}$ and $t \in \mathcal{T}_\alpha$, $\pi(\mathbf{exec}(a, t)) = \pi(t) \cup \bullet t \cup *t \cup \{\mathbf{granted}(a, t)\}$,
 $\eta(\mathbf{exec}(a, t)) = \eta(t) \cup t \bullet \cup \neg \bullet t$, $\nu(\mathbf{exec}(a, t)) = \nu(t)$, where $\neg F = \{\neg f : f \in F\}$
for $F \subseteq \mathcal{F}$;
- for all $t \in \mathcal{T} \setminus \mathcal{T}_\alpha$, $\pi(\mathbf{exec}(t)) = \bullet t \cup *t$, $\eta(\mathbf{exec}(t)) = t \bullet \cup \neg \bullet t$, and $\nu(\mathbf{exec}(t)) = \emptyset$;
- for all $u \in \mathcal{U}$, $\pi(\mathbf{exec}(u)) = \pi(u)$, $\eta(\mathbf{exec}(u)) = \eta(u)$, and $\nu(\mathbf{exec}(u)) = \emptyset$.

Notice that the set of LTL formulae used to specify the properties of PS coincides with the set of LTL formulae used to specify the properties of AWS .

Theorem 1. *Let ϕ be an LTL formula, AWS be an access-controlled workflow system and PS be the planning system associated with AWS , then $\models_{AWS} \phi$ iff $\models_{PS} \phi$.*

This allows us to reduce the problem of checking whether AWS enjoys a given property ϕ to the problem of checking whether PS enjoys ϕ . The proof of the theorem (available in [12]) amounts to showing that AWS and PS are bisimulation equivalent.

We have developed a prototype implementation of the above translation from access-controlled workflow systems to planning systems within SATMC, a SAT-based bounded model checker for planning systems. SATMC [5, 6] is one of the back-ends of the AVISPA Tool [13] and has been key to the discovery of serious flaws in security protocols [14, 15]. Given a planning system PS , an LTL formula ϕ , and a positive integer k as input, SATMC builds a propositional formula whose models (if any) correspond to initialized execution paths χ of PS of length at most k such that $\chi \models \phi$. (We have recently extended SATMC so to handle planning systems as defined in Sect. 4.1, i.e. planning systems featuring rules as well as operators with non-deterministic effects.) The propositional formula is then fed to a state-of-the-art SAT solver and any model found by the solver is translated back into a counterexample. It can be shown (see, e.g., [16]) that the encoding time (i.e. time required to build the propositional formula) is polynomial in the size of the planning system and of the goal formula for any given value of $k > 0$.

Given an LTL formula ϕ and an access-controlled workflow system AWS , the translator automatically reduces the problem of checking whether $\models_{AWS} \phi$ to that of checking whether $\models_{PS} \phi$, where PS is the planning system obtained by applying the translation to AWS . The resulting planning system PS is given as input to SATMC. Therefore by the addition of the above translator, SATMC is now capable to model check access-controlled workflow systems.

5 Experiments

We have formalized the LOP of Sect. 2 as an access-controlled workflow system AWS in a scenario characterized by the following agents: **davide**, the director, **maria** and **marco**, managers, **pierPaolo**, who can act both as preprocessing clerk

and as postprocessing clerk, **pierSilvio**, who can act both as preprocessing clerk and as supervisor, **pietro**, postprocessing clerk, and **stefano**, supervisor. (See [12] for more details.) We have then automatically translated *AWS* into the corresponding planning system and by using SATMC we have analyzed the planning system of the LOP w.r.t. the SoD properties (1) and (2).

ObjSoD for the LOP. Let AWS_0 be the access-controlled workflow system for the LOP we presented in Sect. 2. Our first experiment was to check whether \models_{AWS_0} (1). SATMC found a counterexample where **pierSilvio** can execute all the tasks **intRating**, **extRating** and **approve** through his role **supervisor**, thereby violating the property. By inspecting the intermediate states of the trace it is easy to conclude that the violation occurs if the customer is not industrial, the internal rating is not ok, and the loan is neither **highValue** nor **lowRisk**. Indeed, in this scenario the permission assignment relation, together with the seniority relation between **supervisor** and **postprocessor**, allows a **supervisor** to perform all the critical tasks.

It is easy to see that this violation can be prevented by restricting the permission assignment relation. However this solution has the negative effect of reducing the flexibility of the business process. We therefore considered an alternative solution based on the idea of implementing in the LOP a mechanism that prevents an agent to activate a role if she had already executed **intRating** and **extRating** in the same role. Notice that the mechanism has the effect of restricting the execution paths to those satisfying the following LTL formula:

$$\bigwedge_{a \in A} \bigwedge_{r \in \mathcal{R}} \mathbf{G}(\text{executed}(a, r, \text{intRating}) \Rightarrow \mathbf{G}(\text{executed}(a, r, \text{extRating}) \Rightarrow \mathbf{G} \neg \text{activated}(a, r))) \quad (3)$$

where \mathcal{R} is the set of roles involved in the process and $\text{executed}(a, r, t)$ abbreviates the formula $(\text{granted}(a, r, t) \wedge \mathbf{X} \text{exec}(a, t))$. Thus instead of changing AWS_0 into a new access-controlled workflow system AWS_1 implementing the above mechanism and checking whether \models_{AWS_1} (1), we asked SATMC to check whether $\models_{AWS_0} ((3) \Rightarrow (1))$. SATMC did not find the previous counterexample any more, but found a new one. In the new counterexample the agent responsible for the violation is **stefano**, who executes **intRating** and **extRating** as **supervisor**, but can nevertheless execute **approve** because a manager, **maria**, delegates him to approve the document by means of the delegation rule D1 and this leads to the violation. A further inspection of the intermediate states of the trace shows that the violation occurs if the customer is industrial, the internal rating is ok, and the loan is **highValue** and not **lowRisk**.

To avoid this new violation we constrained the applicability of rule D1 by conjoining its applicability condition with the literal $\neg \text{highValue}$. By changing AWS_0 in this way we obtained a new access-controlled workflow system, say AWS_2 , and asked SATMC to check whether $\models_{AWS_2} ((3) \Rightarrow (1))$. SATMC did not find any counterexamples to this formula.

OpSoD for the LOP. We then checked \models_{AWS_2} (2) and SATMC found a new violation. The violation occurs if the customer is not industrial, the internal rating

is ok, and the loan is `lowRisk` and not `highValue`. Notice that in this situation the loan is `lowRisk` and `extRating` is not performed and therefore the `ObjSoD` is ensured. However `pierSilvio`, who is assigned to roles `preprocessor` and `supervisor`, successfully completes the LOP and thus violates the `OpSoD`. Notice that the user and permission assignment relations do not enable `pierSilvio` to sign the contract. In fact `sign` must be executed by an agent who is at least manager. However a manager, `maria` in the trace, delegates `pierSilvio` to sign the document by executing the delegation rule `D2`.

By inspecting the counterexample leading to the violation, it appears that `pierSilvio` inherits the permission to execute `prepareContract` and `intRating` by the more junior role `postprocessor`. Thus, a possible solution is to modify the role hierarchy by breaking the seniority relation between `supervisor` and `postprocessor`. An alternative solution is to restrict the applicability condition of `D2` by conjoining it with the fact `highValue`. In fact, when the loan is not `highValue`, the access control policy is less restrictive and the application of `D2` must be prevented. SATMC does not find any violation in the access-controlled workflow systems obtained by modifying AWS_2 in both ways.

The experiments were performed on a notebook with an Intel Core 2 Duo processor with 1.50GHz clock and 2GB of RAM memory. All the vulnerabilities were found by SATMC in little times: the encoding time ranges from 2.08 sec for \models_{AWS_0} (1) to 6.43 sec for \models_{AWS_2} (2) while the solving time is less than 0.5 sec for all the experiments considered.

6 Related Work

An approach to the automatic analysis of security-sensitive business processes is put forward in [1]. The paper shows that business processes with RBAC policies and delegation can be formally specified as transition systems and that SoD properties can be formally expressed as LTL formulae specifying the allowed behaviors of the transition systems. The viability of the approach is shown through its application to the LOP and the NuSMV model checker is used to carry out the verification. Our approach provides the user with a level of abstraction which is much closer to the process being modeled and provides a number of important advantages including (i) the separate specification of the workflow and of the access control policy and (ii) the formal specification of the security properties as LTL formulae that specify the allowed behavior of the access-controlled workflow system. This is not the case in the approach presented in [1], where even small changes in the workflow or in the access control policy may affect the specification of the whole transition system and the specification of the security property is relative to the (low level) transition system. In our approach the compilation of the access controlled workflow system and of the expected security properties into the corresponding planning system and properties (resp.) can be done automatically and proved correct once and for all as we have done in this paper. Our approach therefore greatly simplifies the specification process, reduces the

semantic gap, and considerably reduces the probability of introducing bugs in the specification.

A formal framework that integrates RBAC into the semantics of BPEL and uses the SAL model checker to analyze SoD properties as well as to synthesize a resource allocation plan has been presented in [2]. However the approach supports the RBAC model with tasks rigidly associated with specific roles, while our approach supports the specification of a wide variety of access control models and policy updates. Moreover the semantics of BPEL adopted in [2] does not take into account the global state of the process and assumes an interleaving semantics. This is not the case in our approach as it accounts for a global state that can be affected by the execution of the tasks as well as for the simultaneous execution of actions.

An approach to the combined modeling of business workflows with RBAC models is presented in [3]. The paper proposes an extended finite state machine model that allows for the model checking of SoD properties by using the model checker SPIN. It considers a simple RBAC model only based on previous activation (or non-activation) of roles and it does not take into account delegation.

Another approach to the automated analysis of business processes is presented in [17]. The paper proposes to model workflows and security policies in a security enhanced BPMN notation, a formal semantics based on Coloured Petri nets, an automatic translation from the process model into the Promela specification language and the usage of SPIN to verify SoD properties. However no provision is made for the assignment of an agent to multiple roles, role hierarchy, delegation, and the global state of the process.

An approach based on model checking for the analysis and synthesis of fine-grained security policies is presented in [18]. The framework supports the specification of complex policies (including administrative policies), but mutual exclusion in the user assignment relation is not supported, nor it is possible to express role inheritance. Moreover, the modeling of the workflow is not in the scope of [18], whereas modeling and analyzing the interplay of the workflow and the access control policy is one of the main objectives of our work.

7 Conclusions and Future Work

We have presented a new approach to the formal modeling and automatic analysis of security-sensitive business processes that greatly simplifies the specification activity while retaining full automation. Our approach improves upon the state-of-the-art by supporting the separate specification of the workflow and of the security policy as well as a translation into a specification framework amenable to automatic analysis. Our experiments confirm that model checking can be very effective not only to detect security flaws but also to identify possible solutions.

The analysis of business processes via reduction to planning is currently being implemented and integrated by SAP within SAP NetWeaver BPM [19] by using SATMC as back-end. The version of SATMC currently integrated supports a simpler definition of planning systems than that given in Sect. 4.1 (i.e. it does

not feature rules nor operators with non-deterministic effects). The integration of the latest version of SATMC will enable the SAP platform to support the advanced features that have been presented in this work.

References

1. Schaad, A., Lotz, V., Sohr, K.: A model-checking approach to analysing organisational controls in a loan origination process. In: SACMAT, ACM (2006) 139–149
2. Cerone, A., Xiangpeng, Z., Krishnan, P.: Modelling and resource allocation planning of BPEL workflows under security constraints. TR 336, UNU-IIST (2006) <http://www.iist.unu.edu/>.
3. Dury, A., Boroday, S., Petrenko, A., Lotz, V.: Formal verification of business workflows and role based access control systems. In: SECURWARE'07 201–210
4. Peterson, J.L.: Petri Net Theory and the Modeling of Systems. Prentice Hall(1981)
5. Armando, A., Compagna, L.: SATMC: a SAT-based model checker for security protocols. In: JELIA. Vol 3229 of LNAI., Springer-Verlag (2004) 730–733
6. Armando, A., Compagna, L.: Sat-based model-checking for security protocols analysis. In: IJIS, Springer (2007)
7. OASIS: Web Services Business Process Execution Language Version 2.0. (2007) <http://docs.oasis-open.org/wsbpel/2.0/0S/wsbpel-v2.0-0S.html>
8. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. Computer **29**(2) (1996) 38–47
9. Atluri, V., Warner, J.: Supporting conditional delegation in secure workflow management systems. In: SACMAT, ACM Press (2005) 49–58
10. Giunchiglia, E., Lifschitz, V.: An action language based on causal explanation: Preliminary report. In: AAAI-98, AAAI Press (1998) 623–630
11. Ferraris, P., Giunchiglia, E.: Planning as satisfiability in nondeterministic domains. In: AAAI-00 and IAAI-00, AAAI Press / The MIT Press (2000) 748–753
12. Armando, A., Ponta, S.E.: Model checking of security-sensitive business processes (2009) <http://www.ai-lab.it/serena/tr090724.pdf>.
13. Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuellar, J., Drielsma, H.P., Heám, P.C., Mantovani, J., Mödersheim, S., von Oheimb, D., Rusinowitch, M., Santiago, J., Turuani, M., Viganò, L., Vigneron, L.: The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In: CAV, Springer-Verlag (2005)
14. Armando, A., Carbone, R., Compagna, L.: LTL model checking for security protocols. In: CSF-20, IEEE Computer Society (2007) 385–396
15. Armando, A., Carbone, R., Compagna, L., Cuéllar, J., Tobarra, M.L.: Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based single sign-on for google apps. In: FMSE, ACM (2008) 1–10
16. Kautz, H., McAllester, H., Selman, B.: Encoding Plans in Propositional Logic. In: KR, (1996) 374–384
17. Wolter, C., Miseldine, P., Meinel, C.: Verification of business process entailment constraints using SPIN. In: ESSoS, Springer LNCS (2009)
18. Guelev, D.P., Ryan, M., Schobbens, P.Y.: Model-checking access control policies. In: ISC, Vol 3225 LNCS, Springer (2004) 219–230
19. SAP NetWeaver Business Process Management. <http://www.sap.com/platform/netweaver/components/sapnetweaverbpm/index.epx>