Security Validation of Business Processes via Model-checking *

Wihem Arsac[1], Luca Compagna[1], Giancarlo Pellegrino[1], Serena Elisa Ponta[12]

[1] SAP Research Sophia-Antipolis, 805 Avenue Dr M. Donat, 06250 Mougins, France
{wihem.arsac, luca.compagna, giancarlo.pellegrino, serena.ponta}@sap.com
[2] U. of Genova, Viale Causa 13, 16145 Genova, Italy serena.ponta@dist.unige.it

Abstract. More and more industrial activities are captured through Business Processes (BPs). To evaluate whether a BP under-design enjoys certain security desiderata is hardly manageable by business analysts without tool support, as the BP runtime environment is highly dynamic (e.g., task delegation). Automated reasoning techniques such as model checking can provide the required level of assurance but suffer of well-known obstacles for the adoption in industrial systems, e.g. they require a strong logical and mathematical background. In this paper, we present a novel security validation approach for BPs that employs state-of-the-art model checking techniques for evaluating security-relevant aspects of BPs in dynamic environments and offers accessible user interfaces and apprehensive feedback for business analysts so to be suitable for industry.

1 Introduction

Business Processes (BPs) are at the core of many organizations and industrial sectors. While this increases business agility, flexibility and efficiency, it also requires BPs to be carefully designed and executed so to be sure that important Key Performance Indicators such as compliance with respect to regulations and directives, fraud prevention, end-user acceptance and confidence are kept high within organizations. From this, emerge critical security desiderata that have to be properly tackled within BP scenarios. Established standards for Business Process Management (BPM), in primis the Business Process Modeling Notation (BPMN),[3] recognize this necessity. Industrial BPM systems aim to enforce these assignments, but are of little help in guaranteeing business process analysts that the BP they defined fulfills the security desiderata. This, combined with the high dynamicity of the environments in which BPs are run (e.g., delegation of task assignment), makes these security desiderata assessment a very hard task for the business analyst. Though testing and tuning the BP under-design improves its final quality, they hardly provide the level of assurance required. Model checking [1] can provide such a higher level of assurance, as it allows for validating all

* This work was partially supported by the FP7-ICT Projects AVANTSSAR (no. 216471, www.avantssar.eu) and SPaCIoS (no. 257876, www.spacios.eu)

[3] http://www.omg.org/spec/BPMN/2.0

the potential execution paths of the BP under-design against the expected security desiderata. Clear obstacles for the adoption of model checking techniques to validate security desiderata in industry systems include *(i)* the generation of the formal model on which to run the analysis as well as *(ii)* how to feed back the model checking results to a business analyst that is neither a model checking practitioner nor a security expert. In this paper we present a model-checking approach for the automatic validation of security-relevant aspects of BPs and we discuss how it can be integrated within industrial BPM systems and indeed used by business analysts via accessible user interfaces and apprehensive feedback. Thus our approach makes model checking techniques usable by business analysts and allows their integration in real industrial environments. As proof of concept we have implemented our approach as an Eclipse plug-in that has been integrated within the SAP NetWeaver Business Process Management system (NW BPM, [2]). Furthermore we tackle relevant security desiderata including data-related properties as they are of importance in the proper execution of the workflow. A business process example from the banking domain is used to assess the contributions of our work, i.e. *(i)* the viability of our security validation approach, *(ii)* a push-button technology featuring accessible user interfaces, *(iii)* the way we reduce the gap between high level industrially-suited languages for BPs and formal languages via an automatic translation, *(iv)* the way we validate security desiderata and in particular data-related properties under dynamic resource allocation. Considering strong market drivers such as compliance and customer acceptance, the advantage of our approach and tool clearly lies in increasing the quality, robustness and reliability of BP under-design, thus mitigating the risk of deploying non-compliant BP.

The paper is organized as follows. We start to set the motivation of our work by presenting a BP example from the banking domain and critical security desiderata in Section 2. In Section 3 we overview our approach and in Section 4 we show how to formally define BPs and security desiderata. Section 5 assesses our results on a real case study. In Section 6 we discuss the related work and we conclude in Section 7 outlining future research directions.

2 A Motivating Example

Let us consider a scenario where a BP analyst is asked to model a Loan Origination Business Process (LOBP) for a bank. To achieve such a business goal, one has to design all the sequences of tasks executed by business resources so as to perform the loan process. We assume the bank is running NW BPM to handle BPs and the SAP NetWeaver Identity Management (NW IdM) to operate a role-based access-control (RBAC, [3]) enhanced with delegation [4] on BPs. Thus the features of the scenario we consider rely on the expressiveness and behavior of the real system used to model it. The business analyst uses NW BPM to design the LOBP depicted in Figure 1. The LOBP aims to evaluate and possibly grant a customer request for a loan amount. This can be operated by the flow of tasks outlined hereafter. After receiving and identifying the customer, the bank carries

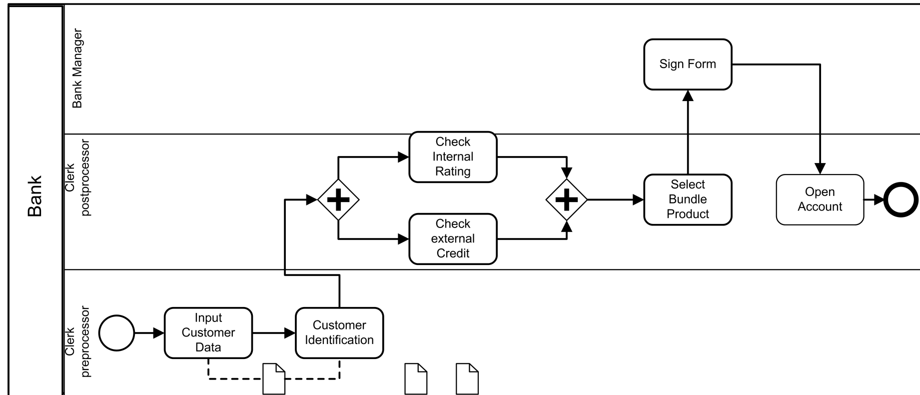


Fig. 1: The Loan process within the Bank

out a careful evaluation of the customer's rating through internal mechanisms and by asking assurance to external agencies called credit bureaus. Subsequently, a bundled product is selected and, if both the customer and the bank agree, a contract is signed. The flow of human tasks that have to be manually realized by bank's users to achieve the LOBP's business goal are visible within the bank's pool. In there, three BPMN lanes capture the three main bank roles that are supposed to be active in the execution of the LOBP: the *pre-processors* who receive and identify the customer's request and update/create the customer data in the banking system (`Input Customer Data` and `Customer Identification` tasks); the *post-processor* who analyze the credit worthiness of the customer internally and externally via interaction with a credit bureau (`Check Internal Rating` and `Check External Credit Worthiness` tasks), identify the most appropriate bundle product (`Select Bundle Product` task), submit a loan proposal to the customer, and, provided that the contract is signed, they are the ones opening the customer bank account in the bank system (`Open Account in System` task); *the managers* who are responsible for the local agency of the bank and intervene to provide their approval to the loan (`Sign Form` task).

The assignment of roles to a user provides him/her with the authorizations needed to fulfill specific activities within an organization. The access control managing the execution of the process that we consider is based on RBAC enhanced with delegation. The bank's NW BPM is connected to the bank's NW IdM where roles, users, user-to-role assignments or delegations rules are defined for the bank organization. The information within NW IdM is exploited in two main ways by NW BPM. First, at design time, it is possible to associate to each human task of the BPMN model, a set of potential owners and a set of excluded owners where a owner is a principal, i.e. either a user or a role. Then, at runtime, a user can claim a task ready-to-be-executed if *(c1)* the potential owners of the task comprise either that user or a role to which he/she is assigned, and *(c2)* the excluded owners of the task do not comprise that user nor a role to which he/she is assigned. The user that has claimed the task is normally the one that will execute it. If for some reasons he/she cannot execute the task,

that user can delegate the task to a delegatee provided that the excluded owners of the task do not comprise the delegatee nor a role to which the delegatee is assigned. (This is normally referred to as delegation of execution [4].) It is thus clear that the environment in which the BP runs is highly dynamic. Notice that a more complex model of delegation could be considered, however we rely on the one in place in NW BPM, the real system on which we assess our approach. Though not visible in Figure 1, we can imagine that the business analyst has defined `preprocessor`, `postprocessor` and `manager` as potential owner of the lane pre-processing clerk, post-processing clerk and manager, respectively.

Another aspect of BPs is the data which are internally accessed to complete the tasks. For instance, in the LOBP, customer data profiles are created and stored in the bank system. The customer data is a data object, made of several fields, gathering all personal and financial information of a customer, including, e.g., the `name`, `gender`, and `ethnic group`. The input of a task normally needs to access certain data fields so as the completion of a task may output data fields. These relations are specified by the business analyst for each task within the BPMN model. It is worth noticing that only a few of these relations are graphically represented on the model itself. Figure 1 captures the `Input Customer Data` task writing in the `Customer_Data` object data fields that are then consumed by the `Customer Identification` task.

Let us imagine now that the business analyst receives some additional requirements. In particular, the bank requires the LOBP to be compliant with regulations (e.g., Basel II, http://www.basel-ii-risk.com/Basel-II), EU directives (e.g., directive 95/46/EC for the protection of individuals' personal data) related to the protection of individuals with regard to the processing of personal data and on the free movement of such data, and the bank internal security policy. Not surprisingly this gives raise to a number of security desiderata for the LOBP that are not so easy to be checked and evaluated by the business analyst. Hereafter a few important security principles the bank wants to enforce in its premises and thus on the LOBP:

(S1) need-to-know: users should access only those sensitive data strictly necessary to accomplish their tasks. For a critical task, data can be defined that should not be known by the principal executing the task. E.g., the user selecting the loan bundle product should perform it objectively and thus should not have access to personal customer information such as the `name` or the `gender`.

(S2) dual control: it aims to mitigate the risk of fraud by dividing the responsibility in executing processes. Separation of duty (SoD) is one means to implement this principle. E.g., it is desirable that the bank user selecting the loan offer is not the same signing the contract.

(S3) data confidentiality: the access to sensitive data should be restricted to certain users. E.g., pre-processing clerks should not access the sensitive fields of the data objects `loanContract` and `account`, such as the loan rate, the contract duration, or the monthly rate.

It is indeed difficult for the business analyst to be sure that security desiderata as the ones expressed above are fulfilled by the BP under-design given the high dynamicity of the environment in which the BP is executed. In other words, the dynamicity of the resource allocation makes the design of the access control on data and task assignment a complex and error-prone activity. In fact, at design time it is very hard to foresee all the behaviors that the access control mechanism, e.g., the RBAC model provided by NW IdM, may originate. The scenario is made even more complex by delegation which may offer unexpected ways to circumvent security desiderata (e.g., when delegating a task, its related data object access is also delegated so that the delegatee may gain access to data fields he/she was not supposed to). We propose an approach providing tool support to business analysts that mitigates the design of BPs non-compliant with security desiderata. In doing this, we focus on security aspects combining both data-flow (such as need-to-know or data confidentiality) and resource allocation (such as separation of duty).

3 An outline of our approach

Our approach, outlined in Figure 2, integrates a security validation procedure within standard BP modeling environments enabling the validation of security desiderata. First of all, the business analyst uses the modeling environment to define the *BP model* (e.g., the LOBP) intended here as the workflow, the data objects as input and output to tasks, task assignment to users and roles, etc. The `Security Desiderata Specification` module offers then the business analyst with accessible user interfaces, i.e. with an easy way to specify industrially relevant *Security Desiderata* (e.g., need-to-know, data confidentiality, dual control) that the BP model is required to satisfy. The rest of the security validation procedure automatically checks whether the BP model enjoys these desiderata or not. In particular, a *Formal Model* capturing both the BP model and the security

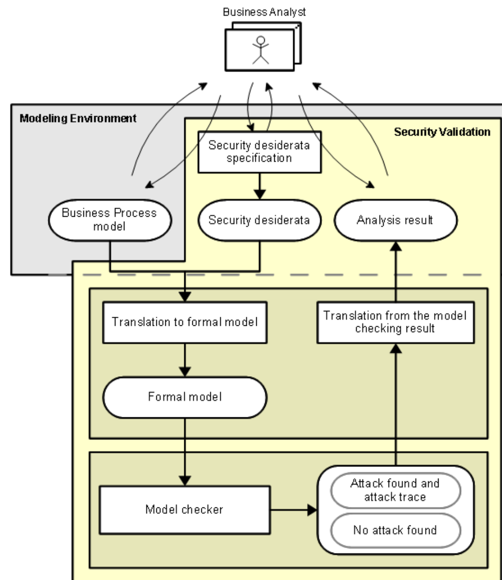


Fig. 2: Our Security Validator within a BPM modeling environment

desiderata is generated (see Section 4) and then analyzed via a `Model Checker` that completely explores all the potential execution paths of the BP model. The results of the analysis, either *no attack found* or *attack found* coupled with the *attack trace*, are retrofitted to the business analyst upon a translation of the difficult-to-read model checking output to an easy-to-understand *Analysis Result* highlighted in a graphical way. This transformation, carried out by the `Translation from the Model Checking result` module, allows the business analyst to quickly understand the attack reported (if any) and to take proper counter-measures to fix it in the BP model. Once implemented within a BPM modeling system our approach offers a push-button security validation procedure that employs state-of-the-art automated reasoning techniques for evaluating security-relevant aspects of BPs in complex dynamic environments and still features accessible user interfaces and apprehensive feedback to be usable for industrial people and business analysts.

4 Formalization

We now show how security relevant aspects of BPs and security desiderata can be formally specified in a model suitable for formal analysis. In particular we use the AVANTSSAR Specification Language (ASLan, [5]) to define a model checking problem where a transition system will be evaluated w.r.t. the security desiderata, i.e. security properties that the process must achieve. States of the transition system are sets of facts, i.e. atomic propositions, that have to satisfy a set of Horn clauses and transitions are captured as rewrite rules. An ASLan specification comprises an initial state, Horn clauses, rewrite rules, and security properties to be defined as attack states. Horn clauses offer a declarative way to specify dependencies among facts, while rewrite rules allow for the specification of transition rules. Let $f, g_1, \ldots, g_n$ be facts, Horn clauses of the form

$$\texttt{hc}\ \ hornClauseName\ := f\ :-\ g_1, \ldots, \mathrm{g}_n$$

assert that in each state where the body holds (i.e., $g_1, \ldots, g_n$ hold), then also the head of the Horn clause holds (i.e., f holds). Conditional rewrite rules have the form

$$\texttt{step}\ rewriteRuleName := \ PFs.NFs\ \&\ Conds\ \ =>\ RHS$$

where PFs and RHS are sets of positive facts, NFs is a set of negated facts, and *Conds* represents a set of possibly negated atomic conditions, e.g. $\texttt{equal}(t_1, t_2)$ that holds if term t_1 is equal to term t_2. Given a state where PFs, NFs, and *Conds* hold, i.e. all the positive facts and conditions hold and the negative facts and conditions do not, the application of the rewrite rule consists in replacing PFs with RHS in the next state. Thus the main difference between Horn clauses and rewrite rules is that the head of an Horn clause is inferred in the very same state where the body holds, while the facts in RHS of a rewrite rules are inferred in the next state if PFs, NFs, and *Conds* hold in the current state and the rule

Table 1: Facts and their informal meaning

Fact	Meaning
userToRole(a, r)	user a is assigned to role r
poto(p, t)	principal p has the permission to perform task t
exo(p, t)	principal p cannot execute task t
canExecute(a, r, t)	user a can execute task t by means of role r
granted(a, r, t)	user a claimed the right to perform task t via role r
ready(t)	task t is ready to be executed
done(t)	task t has been executed
executed(a, t)	user a executed task t
taskToData(t, in, out)	task t accesses data in in input and data out in output
aknows(a, d)	user a knows data d
contains(s, d)	set s contains data d

is applied. Given a set of Horn clauses H, a state S and a rewrite rule R whose PFs, NFs, and $Conds$ hold in S, the transition relation of the transition system leads to a state S' obtained by substituting PFs with RHS and adding the heads of the Horn clauses in H whose body holds (parallel execution of rules can be permitted to increase the efficiency of the approach). More details on the syntax and semantics of ASLan can be found in [5]. A simple way to describe properties of a transition system is by defining a subset of so-called bad states or attack states, i.e. sets of facts in which the desired properties are violated. The attack states specification in ASLan is syntactically similar to a rewrite rule, only that there is no RHS. As a result if an attack state is reached, then the set of rules leading to the attack state define the violation to the security properties. Notice that Horn clauses, rewrite rules, and attack states can be defined with variables of given types. In this way the facts in there used, e.g. in the Horn clauses head and body, can be parameterized to express families of Horn clauses, rewrite rules, and attack states respectively.

The ASLan facts we use to specify BPs in ASLan and their security desiderata are summarized with their informal meaning in Table 1. We illustrate the translation by using the LOBP and security desiderata presented in Section 2. In the ASLan specification, the process flow is expressed by means of rewrite rules managing the facts `ready(t)` and `done(t)`. For example, the sequential execution of task `custIdentification` after task `inputCustData` can be specified as follows

```
step w_custIdentification :=
    done(inputCustData) => ready(custIdentification)
```

All the other sequential tasks are treated in a similar way. As it appears from Figure 1, the LOBP is characterized by a parallel flow of execution, i.e. an AND-split gateway that allows activities `checkIntRating` and `checkExtCreditWorthiness` to be executed simultaneously, and an AND-join that converges the flow. This behavior can be specified by using rewrite rules to express the AND-split and AND-join, i.e. a rule which adds two process-flow-related facts that will be used to state when the tasks in the parallel flow are ready, and a rule that adds

a process-flow-related fact when the tasks of all the incoming paths are completed and will be used to make the task following the parallel flow ready, i.e. `selectBundledProduct`. Other BPMN elements, such as XOR-split and XOR-join, have been modeled as well but for the sake of brevity they are not presented here as they are not used in the scenario considered.

Once the control flow has stated that a task is ready to be executed, the actual execution still requires a user willing to and even more important authorized to perform the task according to the access control policy. As described in Section 2, the core of the access control policy relies on the concepts of potential and excluded owners defined for users or roles. The potential and excluded owner relations as well as the authorization of users to claim the execution of a task (i.e. $\texttt{canExecute}(a, r, t)$) are expressed in ASLan by means of Horn clauses. As an example, the fact that the task `inputCustData` has the role `preprocessor` as a potential owner can be expressed by the Horn clause:

$$\texttt{hc poto_inputCustData} := \texttt{poto}(\texttt{preprocessor}, \texttt{inputCustData})$$

Notice that $\texttt{poto}(\texttt{preprocessor}, \texttt{inputCustData})$ always holds as the body of the Horn clause is empty, i.e. the potential owner relation does not depend on any logical expression. Excluded owner relations can be expressed analogously. To ensure that users assigned to an excluded role for a task t cannot acquire the permission to execute t via other roles, the excluded owner relation involving roles is flattened to users by the following Horn clause

$$\texttt{hc user_forbidden_task}(a, r, t) := \texttt{exo}(a, t) \ :- \ \texttt{userToRole}(a, r), \texttt{exo}(r, t)$$

The authorization of users to claim the execution of a task is captured by Horn Clauses that express the RBAC authorization model and the direct assignment of users to tasks as follows:

$$\texttt{hc rbacAc}(a, r, t) := \texttt{canExecute}(a, r, t) : -\texttt{userToRole}(a, r), \texttt{poto}(r, t)$$

$$\texttt{hc directAc}(a, r, t) := \texttt{canExecute}(a, r, t) : -\texttt{userToRole}(a, r), \texttt{poto}(a, t)$$

The claim of a task can be expressed in ASLan by the following rewrite rule

$$\begin{aligned}&\texttt{step authorizeTaskExec}(a, r, t) := \\ &\quad \texttt{canExecute}(a, r, t).\texttt{ready}(t).\texttt{not}(\texttt{exo}(a, t)) => \texttt{granted}(a, r, t)\end{aligned}$$

where $\texttt{granted}(a, r, t)$ expresses the fact that agent a, having the possibility to perform task t ($\texttt{canExecute}(a, r, t)$), claimed the execution of t and is then in charge of performing it. As already presented in Section 2, users can delegate a task they claimed to a delegatee provided that the excluded owners of the task do not comprise the delegatee nor a role to which the delegatee is assigned. Delegation of execution can be expressed in ASLan by means of a rewrite rule stating that user $a1$ receives from a, who claimed task t, the duty to perform t, as follows:

$$\begin{aligned}&\texttt{step delegationOfExec}(a, a1, r, r1, t, n) := \\ &\quad \texttt{granted}(a, r, t).\texttt{userToRole}(a1, r1).\texttt{not}(\texttt{exo}(a1, t))\&\texttt{not}(\texttt{equal}(a, a1)) \\ &\quad => \ \texttt{granted}(a1, r, t).\texttt{userToRole}(a1, r1)\end{aligned}$$

Then, the execution of tasks by an authorized user is expressed by the following rewrite rule

$$\texttt{step taskExec}(a, r, t, in, out) := \texttt{granted}(a, r, t).\texttt{taskToData}(t, in, out) \Rightarrow \\ \texttt{executed}(a, t).\texttt{done}(t).\texttt{taskToData}(t, in, out).\texttt{aknows}(a, in).\texttt{aknows}(a, out) \quad (1)$$

stating that the task t is executed by a user a, i.e. $\texttt{executed}(a, t)$, if a obtained the right to execute task t acting in role r, i.e. $\texttt{granted}(a, r, t)$. Notice that as a result user a accessed, and thus knows, the sets of data input and output of the task, i.e. $\texttt{aknows}(a, in)$, $\texttt{aknows}(a, out)$. As already said, performing a task is often related to accessing the data objects involved in the BP. In particular each task takes a set of data in input and returns a set of data containing the task results in output, e.g. the customer's rating is a result of the task `checkIntRating`. More in detail the data in input and output are sets of fields of possibly different business objects. These data fields are a static information that can be expressed in ASLan by facts which are included in the initial state and hold during the whole process execution. As an example, the fact that task `inputCustData` outputs, among others, the data field `name` can be expressed by the fact $\texttt{contains}(\texttt{out_inputcustdata}, \texttt{pair}(\texttt{customerdata}, \texttt{name}))$ where $\texttt{pair}(\texttt{customerdata}, \texttt{name})$ associates the data field, `name`, to the data object containing it, `customerdata`, and `out_inputcustdata` is the set of data output of `inputCustData`. The association between tasks and its inputs and outputs is done via facts included in the initial state. E.g., $\texttt{taskToData}(\texttt{inputCustData}, \texttt{in_inputcustdata}, \texttt{out_inputcustdata})$ associates the task `inputCustData` to its input and output sets respectively. Notice that, since executing tasks means accessing data (as shown in (1)), the delegation of a task execution can let the delegatee access data objects he/she was not supposed to.

The security desiderata presented in Section 2 can be expressed in ASLan as attack states. As an example, the need-to-know principle (S1) according to which the user selecting the loan offer should not have access to personal customer information such as the `name` or `gender` can be expressed in ASLan by the following attack state for all $Field \in \{\texttt{name}, \texttt{gender}\}$

$$\texttt{attack_state needToKnow}(a, set) := \texttt{executed}(a, \texttt{selectBundledProduct}). \\ \texttt{aknows}(a, set).\ \texttt{contains}(set, \texttt{pair}(\texttt{customerdata}, Field)) \quad (2)$$

As another example (S2), i.e. to check if an agent can perform both the tasks `selectBundledProduct` and `signForm`, can be specified via an attack state containing facts $\texttt{executed}(a, \texttt{selectBundledProduct})$ and $\texttt{executed}(a, \texttt{signForm})$. Finally, the security desiderata (S3), i.e. pre-processing clerks should not access the loan rate, neither the duration of the contract, nor the amount monthly paid to ensure data confidentiality, can be expressed in ASLan by attack states containing $\texttt{userToRole}(a, \texttt{preprocessor})$, $\texttt{aknows}(a, set)$, and $\texttt{contains}(set, \texttt{pair}(Object, Field))$ where $Object$ and $Field$ have to be replaced with the data objects and fields specified in (S3).

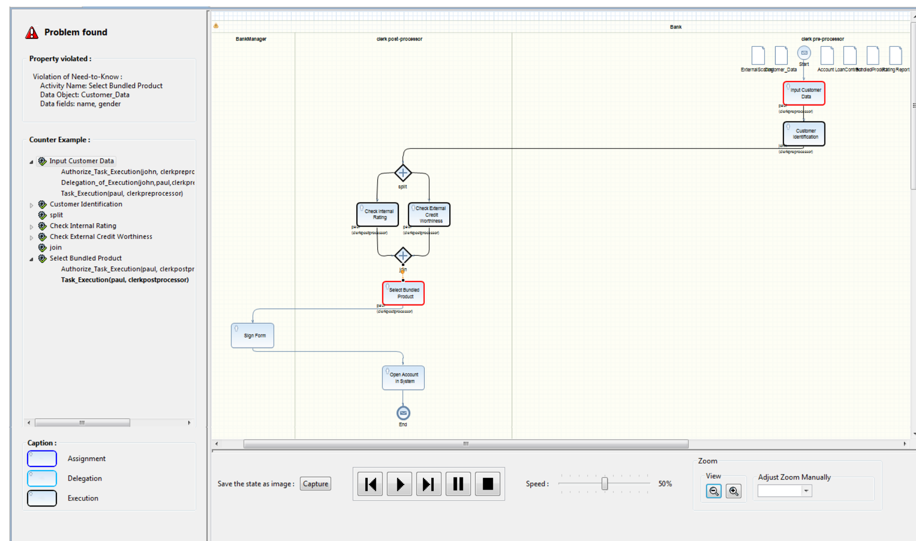
5 Assessment

In order to assess our work, we have implemented our approach and formalization within the NW BPM industrial environment which provides, besides others, a Process Composer module that enables process architects and developers (referred in our paper as business analysts) to design and deploy executable BP models. We have enhanced the Process Composer with a Security Validation plug-in that implements our approach and that business analysts can smoothly run to validate security desiderata for the BP under-design. We consider the business analyst has designed the LOBP in NW BPM as discussed in Section 2 and that he/she is left with the problem of evaluating that the LOBP enjoys the security desiderata (S1), (S2), and (S3) required by the bank. By running the Security Validation plug-in, the business analyst accesses a wizard where he/she can easily create the security desiderata to be validated, i.e. an implementation of the `Security Desiderata Specification` module. It would be clearly unfeasible to expect the business analyst to specify security desiderata as attack states in ASLan, e.g. (2). We have thus invested effort to devise user-friendly GUIs through which the business analyst can express his/her security desiderata without being neither a formal methods practitioner nor a security expert. For instance, in order to validate the need-to-know security desiderata (S1) the business analyst can simply provide the key information, i.e. the task and the data fields that the user performing the task should not have access to, and press a button. The security-relevant aspects of the BP under-design are gathered from the BPMN model in NW BPM and from the other relevant sources (e.g., the NW IdM deployed at the bank where bank users, roles, and user-to-role assignments are specified) and are automatically translated together with the security desiderata into a formal model (see Section 4). Model-checking functionalities are provided by the AVANTSSAR Platform (AVP) as a service[4]. In particular our Security Validator plug-in invokes the SAT-based Model-Checker [6] service. If there exists an execution path of the BP under-design that violates one of the security desiderata, such a violation is discovered by the model checker and returned to our plug-in. The output from the model checker service is not very human-readable and it is thus transformed back into a graphical representation that the business analyst can easily digest and play with. This is critical to allow the business analyst to understand the root cause of the violation (if any) and to take proper counter-measures to fix the issue in the model. By running our Security Validator to check the LOBP for (S1), (S2), and (S3), the SATMC service returns the counter-example listed in Figure 3a. The counter-example is automatically presented via the `Analysis result` GUI of Figure 3b to the business analyst that can see which desiderata has been violated—in this case (S1)—and play the attack trace step-by-step in a movie-like fashion. The attack trace shows to the business analyst that user *paul* can execute the task `inputCustData` accessing in this way to the `name` and `gender` of the customer and can later on execute the task `selectBundledProduct`, thereby violating the need-to-know

[4] http://satmc.ai.dist.unige.it/avantssar/

```
GOAL needToKnow(paul,out_inputcustdata)
[w_inputCustData]
[authorizeTaskExec(john,preprocessor,inputCustData)]
[delegationOfExec(john,paul,preprocessor,postprocessor,inputCustData)]
[taskExec(paul,preprocessor,inputcustomerdata,...,out_inputcustdata)]
[...]
[w_selectBundledProduct]
[authorizeTaskExec(paul,postprocessor,selectBundledProduct)]
[taskExec(paul,postprocessor,selectBundledProduct,...)]
```

(a) Analysis result: SATMC counter-example



(b) Graphical representation of the analysis result

Fig. 3: need-to-know property counter-example

desiderata. Moreover, by playing the attack trace step-by-step it is easy for the business analyst to see that *paul* executes the task `selectBundledProduct` through his role `postprocessor` and is able to perform the task `inputCustData` by being delegated by the pre-processing clerk *john*. Figure 3 clearly shows the benefits of the graphical representation of the attack with respect to the raw output of the model checker which is not suitable for business analysts. It is worth noticing that there is a clear correspondence between the two representations. In particular the graphical attack trace shows the execution flow, i.e. steps of the form `w`*_task* in Figure 3a, by highlighting the activities currently executed and gives key information about the action performed by using different color to highlight the activities, e.g., dark blue for task claim, black for task execution, and blue for delegation. Furthermore some other information are placed with the BP model to provide, e.g., the user claiming the task, the user executing the task, and the user being delegated. To avoid the violation in Figure 3, the business analyst can think to set role `postprocessor` as an excluded owner for task `inputCustData`. Though this prevents the attack trace

just presented, the security desiderata are not fulfilled and a new run of the Security Validation plug-in spots a violation where the pre-processing clerk *john* executes task `inputCustData` and is then delegated by *paul* to perform task `selectBundledProduct`. This new violation stresses the fact that the execution of tasks can be delegated to users of any other role. Thus the business analyst has to define excluded owners for the two tasks involved in the violation such that no roles other than the ones defined as potential owner can be delegated to execute them. New runs of the Security Validator plug-in spot attack traces to properties (S2) and (S3). The business analyst must then analyze the attack traces and modify the process model, e.g. by adding excluded owners. This process is iterated until either no attack traces are detected or there is not a way to fix the problem in the BP model, i.e. the business analyst acknowledges the risk and hence can put in place monitoring systems at run-time in order to prevent frauds. The work was conducted jointly with practitioners of BPM, thus profiting from their expertise in designing user-friendly GUIs for business analysts. The experiments were performed on a desktop computer with an Intel Core 2 Duo processor with 2.66GHz clock and 4Gb of RAM memory. The plug-in takes at most 26 secs to detect the violations to (S1), (S2), and (S3). We are currently considering larger specifications so as to evaluate the scalability of the approach. The experiments performed so far show that, as model checking techniques suffer of the state explosion problem, the time required for the analysis increases with the depth of the attack trace, i.e. the number of steps required to reach the attack state, from 0.62 sec for a trace of depth 1 to 875.02 sec for depth 12, on a business process characterized by 33 tasks with a potential owner each, an AND-split, 5 XOR-splits, and 4 roles. To mitigate the time increase we are currently working on some optimization in the translation of the model so that by reducing the time required for the analysis would ensure the scalability of the approach. Preliminary results are promising both on the toy example presented in this paper as well as on a complex BP for aviation featuring over 60 tasks.

6 Related Work

The use of model checking for the automatic analysis of business processes has been put forward and investigated in [7]. The paper proposes the use of the NuSMV model checker to formalize and verify business processes with RBAC policies and delegation. However it does not take into account the data-flow and the need to make the approach accessible to business analysts which are not familiar with formal methods. Another approach based on model checking to the analysis of business processes is presented in [8]. It considers workflows and security policies modeled in a security enhanced BPMN notation and a formal semantics based on Coloured Petri nets (CP-nets). It then presents 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, delegation, and data-flow. Another approach which uses CP-nets in this context is [9]. It presents a formal technique to

model and analyze RBAC using CP-nets which can be composed with context-specific aspects of the application of interest. However the approach does not take into account data and thus data related properties. An approach based on model checking for the analysis of access control policies is presented in [10]. It supports the specification of complex policies where permissions are the ability of agents to access data to read or write, however the scope is not dedicated to business processes as it does not take into account the process workflow. In [11] the authors consider workflow and authorization policies for privacy purposes. In particular they propose the use of Color-X diagrams to represent the process which is then translated in Prolog to perform an analysis of privacy relevant properties over data, e.g. need-to-know principle. However, their approach is restricted to privacy and does not take into consideration critical properties, such as dual control. A data-aware language for expressing compliance requirements is presented in [12]. It extends the BPMN-Q query language to express compliance rules, e.g. how the content of data should influence the workflow. The rules are expressed as queries and then formalized by mapping them into past linear temporal logic formulae which are then model checked against the process models to decide about compliance. In particular BPMN-Q allows users to express compliance requirements in a visual way very similar to the way processes are modeled and whenever a compliance rule is violated, execution paths causing violations are visualized to the user. However the paper does not take into account the access control policy in place and thus security properties related to the access to data by unauthorized users or dual control. An extension of UML for secure systems development, i.e. UMLsec, and a tool for the analysis of software configurations that ensures compliance with security policies are presented in [13, 14] respectively. In particular the approach allows to check extended UML models of business applications and their security permissions for vulnerabilities to security rules (such as separation of duty). The approach and motivations are similar to ours, however we here focus on aiding the design of security relevant aspects of business processes modeled by using BPMN which is broadly employed in industrial business process management systems. Nonetheless, it is worth to further investigate this approach in the future.

7 Conclusion and Future Work

Is this critical BP task executable in an objective way within my organization? Are these sensitive BP data accessible to the manager director only? These are recurrent difficult-to-answer questions a business analyst may face during the design of a BP. In this paper we have detailed a novel security validation approach for BPs that employs state-of-the-art model checking techniques for evaluating security-relevant aspects of BPs in dynamic environments, and still features accessible user interfaces and apprehensive feedback for business analysts.

At the core of our approach lies a general way to formally capture security-relevant aspects of BPs, not only those that have been well-studied so far (e.g., dual control), but also new important ones from the area of data-flow such as

need-to-know and data confidentiality. As proof of concept we have implemented our approach as an Eclipse plug-in within NW BPM and assessed it against a BP example from the banking area. All in all, our approach increases the quality, robustness and reliability of BP under-design, mitigating the risk of deploying non-compliant BPs.

In the early future, we want to capture in our formal specification other aspects of BPs including service invocation and consumption as well as data objects shared among several BP instances. Some optimizations in the translation to the formal model are under development in order to scale the approach on industrial-scale BPs. Preliminary results run on a BP from the aviation domain are promising despite the over 60 tasks executed in that BP. This makes more concrete a potential productization of our approach within industrial tools such as NW BPM. We are currently discussing with SAP business units in that respect.

References

1. Clarke, E.M., Grumberg, O., Peled, D.: Model checking (2000)
2. Karch, S., Heilig, L.: SAP NetWeaver. 1. aufl edn., Galileo Press (2004)
3. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. Computer **29**(2) (1996) 38–47
4. Giorgini, P., Massacci, F., Mylopoulos, J.: Modeling security requirements through ownership, permission and delegation. In: RE, IEEE Press (2005) 167–176
5. AVANTSSAR: Deliverable 2.1: Requirements for modelling and ASLan v.1. Available at http://www.avantssar.eu (2008)
6. Armando, A., Carbone, R., Compagna, L.: LTL Model Checking for Security Protocols. In: JANCL, special issue on Logic and Information Security. (2009)
7. 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
8. Wolter, C., Miseldine, P., Meinel, C.: Verification of business process entailment constraints using SPIN. In: ESSoS, Springer LNCS (2009) 1–15
9. Rakkay, H., Boucheneb, H.: Security analysis of role based access control models using colored petri nets and cpntools. (2009) 149–176
10. Zhang, N., Ryan, M., Guelev, D.P.: Evaluating access control policies through model checking. In: ISC. (2005) 446–460
11. Teepe, W., van de Riet, R., Olivier, M.: Workflow analyzed for security and privacy in using databases. J. Comput. Secur. **11**(3) (2003) 353–363
12. Awad, A., Weidlich, M., Weske, M.: Specification, verification and explanation of violation for data aware compliance rules. ICSOC-Service Wave (2009)
13. Jan, J.: Secure Systems Development with UML. Springer-verlag edn., Springer Academic Publishers (2005)
14. Höhn, S., Jürjens, J.: Rubacon: automated support for model-based compliance engineering. In: ICSE. (2008) 875–878