Formal Specification and Automatic Analysis of Business Processes under Authorization Constraints: an Action-based Approach*

Alessandro Armando, Enrico Giunchiglia, and Serena Elisa Ponta

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

Abstract. We present an approach to the formal specification and automatic analysis of business processes under authorization constraints based on the action language $\mathcal{C}$. The use of $\mathcal{C}$ allows for a natural and concise modeling of the business process and the associated security policy and for the automatic analysis of the resulting specification by using the Causal Calculator (CCALC). Our approach improves upon previous work by greatly simplifying the specification step while retaining the ability to perform a fully automatic analysis. To illustrate the effectiveness of the approach we describe its application to a version of a business process taken from the banking domain and use CCALC to determine resource allocation plans complying with the security policy.

1 Introduction

Business processes are sets of coordinated activities subject to an access control policy stating which agent can access which resource. The specification of the policy is usually given in terms of a basic access control model (e.g. the RBAC model) possibly enriched with features providing the flexibility required by the application domain (e.g. delegation) and mechanisms that are necessary to meet mandatory regulations (e.g. separation of duty constraints). The complexity of the resulting system is usually so big that the ability to verify even basic properties such as the executability of the process w.r.t. the available resources cannot be done by manual inspection only. In previous works [1, 2] it has been shown that model checking can be profitably used for the automatic analysis of business processes. Yet their applicability to business processes of real-world complexity is problematic as state-of-the-art model checkers—being geared to the analysis of hardware designs—require the system to be modeled as the composition of independent (yet interacting) sub-components. On the contrary business processes and the associated security policies are best viewed as a collection of actions whose execution must satisfy a given workflow pattern and a set of access control rules.

* 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).

In this paper we present an approach to the formal specification and automatic analysis of business processes under authorization constraints based on the action language $\mathcal{C}$ [3]. The use of $\mathcal{C}$ allows for a natural and concise modeling of the business process and the associated security policy as well as for the automatic analysis of the resulting specification by using the Causal Calculator (CCALC) [4]. 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 model extended with delegation and separation of duty constraints. By using CCALC we have faced the problem of finding if there exists a resource allocation plan for the business process.

The rest of the paper is structured as follows. In the next section we provide a brief introduction to business processes under authorization constraints by describing our case study which will be used throughout the paper. In Section 3 we present the action language $\mathcal{C}$. In Section 4 we show how a business process under authorization constraints can be expressed in $\mathcal{C}$. In Section 5 we describe the analysis of the LOP carried out with CCALC. In Section 6 we discuss the related work and we conclude in Section 7 with some final remarks.

2 Business Processes under Authorization Constraints

Let us consider the Loan Origination Process (LOP) graphically presented in Figure 1. The workflow of the process is represented by means of an extended Petri net. Next to the basic concepts of Petri nets [5], i.e places, transitions, and arcs, an extended Petri net considers conditional arcs between places and transitions. Places from which arcs run to a transition are called the input places of the transition; places to which arcs run from a transition are called the output places of the transition. A transition of an extended Petri net can *fire* if there is a token in every input place of the transition and the conditions associated with the arcs between the input places and the transition hold.

Informally, 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. By means of the rating evaluation the bank establishes if the customer is suitable to receive the loan. To this aim, the execution may follow different paths: if the risk associated with the loan is low (`lowRisk`), then an internal rating suffices (`intRating`); otherwise the internal rating is followed by the evaluation of an external rating (`extRating`). The `lowRisk` condition denotes a situation in which the internal rating is ok and the amount of the loan is not high. The external rating is carried out by means of credit information stored and safeguarded by a Credit Bureau (CB), a third-party financial institution. As soon as it is ascertained that the external rating is needed, a request to obtain the credit information about the current customer is sent to a CB (`invoke creditBureau`). By using this information, the external rating evaluation is performed by executing the task `extRating`. Notice that the invocation of the CB and the execution of the task `extRating` are performed by the same

agent. In case there is a forbidden access to the information sent from the CB to the bank, i.e. an agent different from the one who sent the request has accessed the information exchanged, then the execution of the task is prevented and the rating evaluation is interrupted. In case of interruption, the director of the bank has to re-invoke the CB and to execute the task `extRating` for the rating evaluation to be successfully completed. 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, e.g 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.

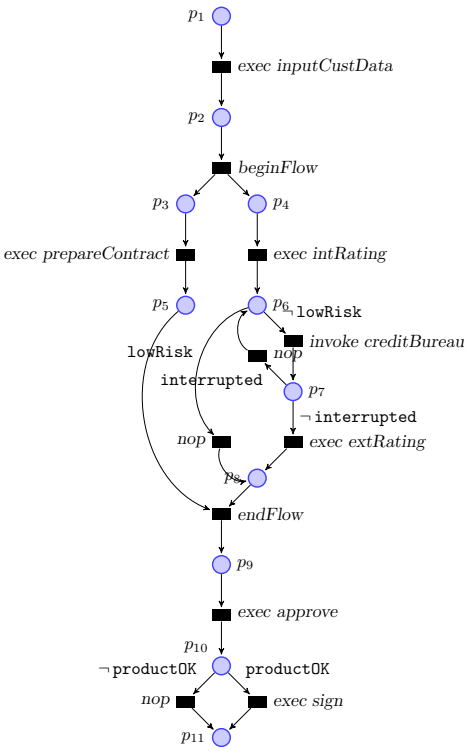

Fig. 1. Extended Petri net for the LOP

An agent can execute a task only if she has the required permissions. As it is common in the business domain, the access control policy of the LOP relies on an access control model based on RBAC [6] enhanced with delegations and separation of duty constraints. According to the RBAC model, to perform a task an agent must be assigned to a role 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 w.r.t. 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. Notice also that the invocation of the CB does not appear in Table 1 as this activity has to be executed by the same agent of the task `extRating` and, as a consequence, uses its permission assignment. We consider a static assignment of agents to roles subject to the following constraints: *(i)* there must be only one director, *(ii)* the director must not be assigned to any other role, *(iii)* an agent must not be assigned to roles hi-

Table 1. Permission assignment for the LOP

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

Table 2. SoD Constraints of the LOP

Name	Object	Critical Tasks
C1	customer's data	inputCustData, prepareContract, intRating, extRating
C2	internal rating	intRating, approve
C3	external rating	extRating, approve
C4	contract	prepareContract, approve, sign

erarchically related, e.g. an agent cannot be assigned to both roles supervisor and postprocessor.

Our RBAC access control model is enhanced with delegation as follows. A delegation rule is a quadruple of the form: $\langle Condition, ARole, DRole, Task\rangle$, where *ARole* and *DRole* are roles, $Task$ is a task, and $Condition$ is the applicability condition. A delegation rule states that if *Condition* holds and *ARole* is authorized to perform *Task* according to the permission assignment relation, then *ARole* can delegate *DRole* to execute *Task*. In our case study we consider the following delegation rules:

- D1: $\langle \neg$ highValue, supervisor, postprocessor, extRating$\rangle$,
- D2: $\langle$intRatingOK, director, supervisor, sign$\rangle$.

Finally, the RBAC access control model of our case study is enhanced with a mechanism that is necessary to meet separation of duty constraints. Separation of Duty (SoD) constraints are used for internal control and amounts to requiring that some critical tasks are executed by different agents. (See [1] for a survey on SoD properties.) In this paper we focus on a relaxed form of *object-based SoD* properties according to which an agent can access the same object through different roles as long as she does not perform all the tasks accessing that object. For each object involved in the LOP we define a set of critical tasks, i.e. all and only the tasks accessing to the same object. We then assume that the process implements a mechanism that prevents an agent from executing all the critical tasks for each object. The SoD constraints enforced by the presented mechanism in our case study are given in Table 2.

3 The Action Language $\mathcal{C}$

Action languages are high level formalisms for expressing *actions* and how they affect the world described with a set of atoms called *fluents*. Thus, the signature σ of the language is partitioned into the *fluent symbols* σ^{fl} and the *action symbols* σ^{act}. Intuitively, *actions* are a subset of the interpretations of σ^{act} while *states* are a subset of the interpretations of σ^{fl}. A formula of σ is a propositional combination of atoms. Each action language differs from the other for the constructs used to characterize how actions affect states.

$\mathcal{C}$ is an expressive propositional action language allowing for two kinds of propositions: *static laws* of the form

$$\textbf{caused } F \textbf{ if } G \tag{1}$$

and *dynamic laws* of the form

$$\textbf{caused } F \textbf{ if } G \textbf{ after } H, \tag{2}$$

where F, G, H are formulas of σ such that F and G do not contain action symbols. In a proposition of either kind, the formula F will be called its *head*.

An *action description* is a set of propositions. Consider an action description D. A *state* is an interpretation of σ^{fl} that satisfies $G \supset F$ for every static law (1) in D. A *transition* is any triple $\langle s, a, s' \rangle$ where s, s' are states and a is an action; s is the *initial* state of the transition, and s' is its *resulting* state. A formula F is *caused* in a transition $\langle s, a, s' \rangle$ if it is

- the head of a static law (1) from D such that s' satisfies G, or
- the head of a dynamic law (2) from D such that s' satisfies G and $s \cup a$ satisfies H.

A transition $\langle s, a, s' \rangle$ is *causally explained* according to D if its resulting state s' is the only interpretation of σ^{fl} that satisfies all formulas caused in this transition.

The *transition diagram* represented by an action description D is the directed graph which has the states of D as nodes and includes an edge from s to s' labeled a for every transition $\langle s, a, s' \rangle$ that is causally explained according to D.

Intuitively, if $\langle s, a, s' \rangle$ is a transition of the diagram, the concurrent execution of the atomic actions satisfied by a causes a transition from the state s to the state s'. Despite the fact that $\mathcal{C}$ consists of only two kinds of propositions, several others can be defined as abbreviations of either (1) or (2), modeling, e.g. actions' preconditions, actions' nondeterministic effects, fluents with default values, and inertial and exogenous fluents. The abbreviations used in this paper are given in Table 3, where F and G are fluent formulas, and H is an action formula.

4 Formal Modeling of Business Processes with $\mathcal{C}$

We now show how business processes under authorization constraints can be formally specified in the action language $\mathcal{C}$. This is done by using the LOP

Table 3. Abbreviations for Causal Laws

Abbreviation	Expanded Form	Informal Meaning
nonexecutable H **if** F.	**caused** $\bot$ **after** $H \wedge F$.	F is a precondition of H
H **causes** F **if** G.	**caused** F **if** $\top$ **after** $G \wedge H$.	F is true after H is executed in a state in which G is true
H **may cause** F **if** G.	**caused** F **if** F **after** $H \wedge G$.	F is true by default after H is executed in a state in which G is true
default F.	**caused** F **if** F.	F is true by default
constraint F.	**caused** $\bot$ **if** $\neg F$.	F must be true

Table 4. Fluents and their informal meaning

Fluent	Meaning
$\texttt{activated}(a, r)$	agent a is playing role r
$\texttt{accessed}(a)$	agent a has accessed the information sent by the CB
$\texttt{ua}(a, r)$	agent a is assigned to role r
$\texttt{delegated}(a, r, t)$	agent a is delegated by an agent in role r to perform task t
$\texttt{pa}(r, t)$	role r has the permission to perform task t
$\texttt{granted}(a, r, t)$	agent a obtained by means of role r the permission to perform task t
$\texttt{executed}(a, r, t)$	agent a has executed task t obtaining the authorization from role r
$\texttt{invoked}(a, r, e)$	agent a has invoked entity e obtaining the authorization from role r
$\texttt{senior}(r_1, r_2)$	role r_1 is more senior than or is as senior as r_2
`lowRisk`	the risk associated with the loan is low
`highValue`	the loan amount is high w.r.t. the financial status of the customer
`isIndustrial`	the customer is industrial
`intRatingOK`	the customer's internal rating is positive
`productOK`	both the customer and the bank agree on the contract
`interrupted`	the execution of the process is interrupted
$\texttt{p}_1, \ldots, \texttt{p}_{11}$	the places of the extended Petri net (cf. Figure 1)

as a case study. The set of fluents involved comprises all the fluents listed in Table 4 suitably instantiated for all tasks and roles in Table 1 and the set of available agents. The fluents `lowRisk` and $\texttt{pa}(r, t)$ are statically determined, that is their value is determined by means of static laws. The fluents $\texttt{activated}(a, r)$ and $\texttt{accessed}(a)$ are exogenous, i.e. they can arbitrarily change value in the transition from one state to the other (unless there is some other rule constraining their value). In addition, $\texttt{activated}(a, r)$ is subject to

$$\textbf{caused } \neg\, \texttt{activated}(a, r) \textbf{ if } \neg\, \texttt{ua}(a, r). \tag{3}$$

to state that an agent cannot be active in a role she is not assigned to, and $\texttt{accessed}(a)$ is subject to

$$\textbf{caused } \texttt{accessed}(a) \textbf{ after } \texttt{accessed}(a). \tag{4}$$

to state that the access to the information exchanged in the process by an agent is not reversible. All the other fluents in Table 4 are inertial, i.e. they keep their

value by default in the transition from one state to the other (unless there is some other rule constraining their value).

Each transition of the extended Petri net in Figure 1 (tasks execution, invocation of entities and empty activities) are actions. The preconditions of these actions have a common pattern, i.e. they always include the input places of the corresponding transition and the conditions associated with the arcs from the input places, if any. For example, the task `approve` is expressed by

$$\texttt{exec}(a, r, \texttt{approve}) \ \textbf{causes} \ \ \texttt{p}_{10} \wedge \neg\, \texttt{p}_9 \wedge \texttt{executed}(a, r, \texttt{approve}) \\ \textbf{if} \ \ \texttt{granted}(a, r, \texttt{approve}) \wedge \texttt{p}_9\,. \quad (5)$$

$$\texttt{exec}(a, r, \texttt{approve}) \ \ \textbf{may cause} \ \ \texttt{productOK} \\ \textbf{if} \ \ \texttt{granted}(a, r, \texttt{approve}) \wedge \texttt{p}_9\,. \quad (6)$$

where $\texttt{exec}(a, r, \texttt{approve})$ is the action symbol. Causal law (6) states that `productOK` is a nondeterministic effect of $\texttt{exec}(a, r, \texttt{approve})$. As it appears from (5), each transition corresponding to a task t requires $\texttt{granted}(a, r, t)$ as further precondition.

Some actions can also require to be executed as soon as their preconditions hold, e.g. `invoke creditBureau`. This is expressed by the following set of causal laws.

$$\begin{array}{l} \textbf{nonexecutable} \ \ \texttt{invoke}(a, r, \texttt{creditBureau}) \\ \qquad \textbf{if} \ \ \neg\, \texttt{granted}(a, r, \texttt{extRating}) \vee \neg\, \texttt{p}_6 \vee \texttt{lowRisk}\,. \\ \\ \textbf{caused} \ \ \texttt{invoke}(a, r, \texttt{creditBureau}) \\ \qquad \textbf{if} \ \ \texttt{granted}(a, r, \texttt{extRating}) \wedge \texttt{p}_6 \wedge \neg\, \texttt{lowRisk}\,. \\ \\ \texttt{invoke}(a, r, \texttt{creditBureau}) \\ \qquad \textbf{causes} \ \ \neg\, \texttt{p}_6 \wedge \texttt{invoked}(a, r, \texttt{creditBureau}) \wedge \texttt{p}_7\,. \end{array} \quad (7)$$

The execution of the task `extRating` is then expressed by

$$\texttt{exec}(a, r, \texttt{extRating}) \ \ \textbf{causes} \ \ \texttt{p}_8 \wedge \neg\, \texttt{p}_7 \wedge \texttt{executed}(a, r, \texttt{extRating}) \\ \textbf{if} \ \ \texttt{invoked}(a, r, \texttt{creditBureau}) \wedge \neg\, \texttt{interrupted}\,. \quad (8)$$

Unlike the other tasks, `extRating` does not have $\texttt{granted}(a, r, \texttt{extRating})$ in its preconditions as it is implicit in $\texttt{invoked}(a, r, \texttt{creditBureau})$. Notice that $\neg\, \texttt{interrupted}$ represents the condition associated with the arc between place $\texttt{p}_7$ and the current transition. The fluent `interrupted` is an inertial fluent subject to the following static laws for all $a_1 \neq a_2$ and $r \neq \texttt{director}$:

$$\begin{array}{l} \textbf{caused} \ \ \texttt{interrupted} \ \ \textbf{if} \ \ \texttt{p}_7 \wedge \texttt{invoked}(a_1, r, \texttt{creditBureau}) \wedge \texttt{accessed}(a_2). \\ \textbf{caused} \ \ \neg\, \texttt{interrupted} \ \ \textbf{if} \ \ \texttt{p}_7 \wedge \texttt{invoked}(a, \texttt{director}, \texttt{creditBureau}). \end{array}$$

As described in Section 2, the access control policy is given by an RBAC model enhanced with delegation and SoD constraints. The permission assignment relation is expressed by the statically determined fluents $\texttt{pa}(r, t)$ defined

by static laws according to Table 1, e.g.:

$$\begin{array}{l}\textbf{caused}\ \ \texttt{pa}(\texttt{supervisor},\texttt{extRating})\ \ \textbf{if}\ \ \neg\,\texttt{interrupted}\,.\\ \textbf{default}\ \ \neg\,\texttt{pa}(\texttt{supervisor},\texttt{extRating}).\\ \textbf{caused}\ \ \texttt{pa}(\texttt{director},\texttt{extRating})\ \ \textbf{if}\,\texttt{interrupted}\,.\\ \textbf{default}\ \ \neg\,\texttt{pa}(\texttt{director},\texttt{extRating}).\end{array}$$

The RBAC policy enhanced with delegation is given by static laws, e.g.:

$$\begin{aligned}\textbf{caused}\ \ \texttt{granted}(a,r,t)\ \ \textbf{if}\ \ &(\texttt{ua}(a,r)\wedge\texttt{activated}(a,r)\wedge\\ &\texttt{pa}(r_1,t)\wedge\texttt{senior}(r,r_1))\vee\texttt{delegated}(a,r,t).\end{aligned}\quad(9)$$

Each delegation rule originates a dynamic law. For example, rule D1 is modelled, for all $a_1 \neq a_2$, by

$$\begin{aligned}\texttt{d1}(a_1,a_2)\ \ \textbf{causes}\ \ &\texttt{delegated}(a_2,\texttt{supervisor},\texttt{extRating})\\ \textbf{if}\ \ &\texttt{ua}(a_1,\texttt{supervisor})\wedge\texttt{ua}(a_2,\texttt{postprocessor})\wedge\\ &\texttt{pa}(\texttt{supervisor},\texttt{extRating})\wedge\neg\,\texttt{highValue}\,.\end{aligned}\quad(10)$$

Finally, the SoD constraints of Table 2 are given by static laws constraining each set of critical tasks to be executed by at least two different agents. For example, constraint C3 is expressed by

$$\begin{aligned}\textbf{constraint}\ \ \neg(&\texttt{executed}(a,r_1,\texttt{prepareContract})\wedge\\ &\texttt{executed}(a,r_2,\texttt{approve})\wedge\texttt{executed}(a,r_3,\texttt{sign})).\end{aligned}\quad(11)$$

5 Experiments

We have fed CCALC with the specification of the LOP given in Section 4 and used it to determine whether the process can be run to completion. When this is the case a resource allocation plan (i.e. an assignment of agents to tasks) is extracted from the execution trace returned by the tool. Notice that if the assignment of agents to roles is not given (as it is the case in our case study) a possible agent-role assignment can also be obtained from the execution trace.

The process can take place in different scenarios characterized not only by different initial states (according to the initial value of `isIndustrial` and `highValue`), but also by the different nondeterministic effects determined by some actions and effects of exogenous fluents. As a result, in order to verify the executability of the process for all the possible scenarios, we run CCALC against all possible initial states and all possible successful process completions. The initial states are those in which the fluent $\texttt{p}_1$ and the fluents concerning the role hierarchy hold. The process ends successfully if it reaches a state in which both $\texttt{p}_{11}$ and `productOK` hold. We looked for a resource allocation plan for each of these scenarios. As further requirement, we looked for resource allocation plans involving the minimal number of agents. Thus we started by considering a single

agent and then we incremented the number of available agents until an execution trace was found by CCALC. CCALC could not find any way to complete the process when a single agent was considered. This is not surprising as the SoD constraints require at least two agents to perform the critical tasks. When we considered two agents, say a_1 and a_2, CCALC found an execution trace that suggests a resource allocation plan where a_1 evaluates the internal rating and prepares the contract, while a_2 inputs the customer's data, and approves and signs the contract. The execution trace also suggests an agent-role assignment, i.e. a_1 is both `preprocessor` and `supervisor` while a_2 is both `preprocessor` and `manager`. This resource allocation plan leads to successful completion of the process in a scenario where the customer is not industrial, the loan is not `highValue` and the internal rating is ok. However CCALC could not find an execution trace for all the scenarios by involving only two agents. When we considered three agents, CCALC found an execution trace for all the scenarios but the ones where the internal rating is not ok and the process is interrupted. This brought to a further inspection of the policy that highlighted that the process executability was not ensured when `intRatingOK` does not hold and `interrupted` does. In fact, in these scenarios a director is required to execute tasks `extRating` and `approve` while the SoD constraint C3 prevents the same agent from performing both tasks. As a consequence we modified the policy disabling the constraint C3 when the internal rating is not ok and the process is interrupted. Considering the new access control policy CCALC verified the process executability by finding an execution trace and a corresponding resource allocation plan for each scenario by involving three agents.

6 Related Work

The use of the action language $\mathcal{C}$ for the specification of business processes has been faced in [7] where the research objective is to describe, simulate, compose, verify, and develop Web services. On the contrary, our approach takes into account access control policies and focuses on model checking of business processes subject to an access control policy with the objective of verifying properties.

The problem of specifying a workflow process in the action language $\mathcal{C}$ has been also faced in [8]. In particular this paper considers activities with duration and the cost of a workflow execution but, differently from our approach, it does not take into account complex access control policies, mandatory requirements, and the problem of finding resource allocation plans.

The use of model checking for the automatic analysis of business has been put forward and investigated in [1]. The paper uses NuSMV, a state-of-the-art model checker originally designed for hardware circuits, to carry out the analysis of the LOP w.r.t. a variety of SoD properties. Our approach, by using an action language, allows for a natural and concise modeling of the business process and the associated access control policy which is much closer to the process being modeled. This greatly simplifies the specification process considerably reducing the probability of introducing bugs in the specification.

A formal framework that uses the SAL model checker to analyze SoD properties as well as to synthesize a resource allocation plan for business process has been put forward in [2]. However, differently from our approach, this framework does not offer a natural modeling and RBAC is the only access control model supported (with tasks rigidly associated with specific roles). Finally, the paper assumes an interleaving semantics while our approach allows multiple actions to be executed simultaneously.

7 Conclusions

The design and verification of business processes under authorization constraints is a time consuming and error-prone activity. Moreover, due to the complexity that business processes subject to an access control policy may reach, it can be difficult to verify even basic properties such as the executability of the process w.r.t. the available resources by manual inspection only.

In this paper we have presented an action-based approach to the formal specification and automatic analysis of business processes under authorization constraints. Our approach improves upon the state-of-the-art by greatly simplifying the specification activity while retaining the ability to perform a fully automatic analysis of the specifications by means of the Causal Calculator CCALC. Our experiments indicate that model checking of $\mathcal{C}$ specifications can be profitably used to verify the process executability and to identify resource allocation plans complying with the security policy.

References

1. Schaad, A., Lotz, V., Sohr, K.: A model-checking approach to analysing organisational controls in a loan origination process. In: SACMAT'06, ACM (2006) pp. 139–149
2. Cerone, A., Xiangpeng, Z., Krishnan, P.: Modelling and resource allocation planning of BPEL workflows under security constraints. Technical Report 336, UNU-IIST (2006) `http://www.iist.unu.edu/`.
3. Giunchiglia, E., Lifschitz, V.: An action language based on causal explanation: Preliminary report. In: AAAI-98, AAAI Press (1998) pp. 623–630
4. Texas Action Group at Austin: The causal calculator. (2008) `http://www.cs.utexas.edu/users/tag/cc/.`
5. Peterson, J.L.: Petri Net Theory and the Modeling of Systems. Prentice Hall PTR, Upper Saddle River, NJ, USA (1981)
6. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. Computer **29**(2) (1996) pp. 38–47
7. Chirichiello., A.: Two Formal Approaches for Web Services: Process Algebras & Action Languages. PhD thesis, "Sapienza" University of Roma. (2008)
8. Koksal, P., Cicekli, N.K., Toroslu, I.H.: Specification of workflow processes using the action description language $\mathcal{C}$. In: AAAI Spring 2001 Symposium Series: Answer Set Programming. (2001) pp. 103–109