Modeling and Reasoning about Business Processes under Authorization Constraints: a Planning-based Approach

A. Armando[1,2], E. Giunchiglia[2], M. Maratea[2] and S. E. Ponta[3]

[1]Security & Trust Unit, Fondazione Bruno Kessler, Trento, Italy

[2]DIBRIS, University of Genova

[3]SAP Research Sophia-Antipolis, Mougins, France

ICAPS 2013
Rome, June 13th 2013

Context and Motivations

Business processes under authorization constraints are

- sets of coordinated activities (workflow),
- subject to an access control policy stating which agent can execute which activity.

Analysis of Business Processes

- Difficult to get right.
- Traditional verification techniques do not ensure the needed level of assurance.

Context and Motivations

Business processes under authorization constraints are

- sets of coordinated activities (workflow),
- subject to an access control policy stating which agent can execute which activity.

Analysis of Business Processes

- Difficult to get right.
- Traditional verification techniques do not ensure the needed level of assurance.

Action Languages

- High level formalisms for expressing actions and how they affect the world, described by states, i.e. sets of atoms called fluents.
- Expressive languages, e.g. non-determinism, indirect, and conditional effects.
- Automated analysis support through existing tools.
- In the context of business processes: Allow for the *separate* specification of workflow and security policy in a natural way.

Contributions

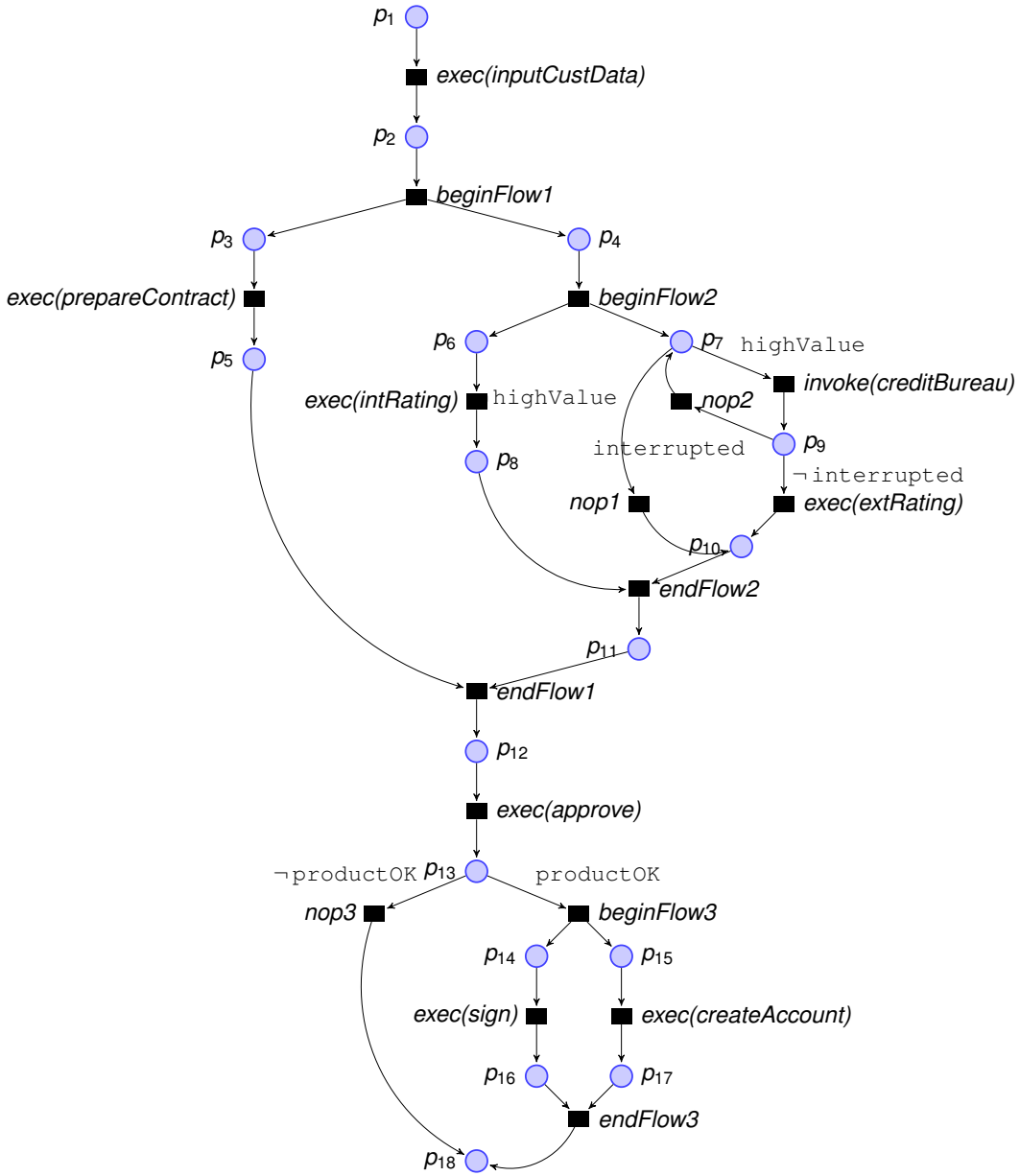
- Formal specification of business processes under authorization constraints based on the action language $\mathcal{C}$ that allows for a natural and concise modeling.
- Automatic analysis of the formal specification by using the Causal Calculator (CCALC).
- Effectiveness of the approach assessed applying it against a Loan Origination Process (LOP), a business process from the banking domain.

Outline

LOP - Workflow

Business Processes under authorization constraints are

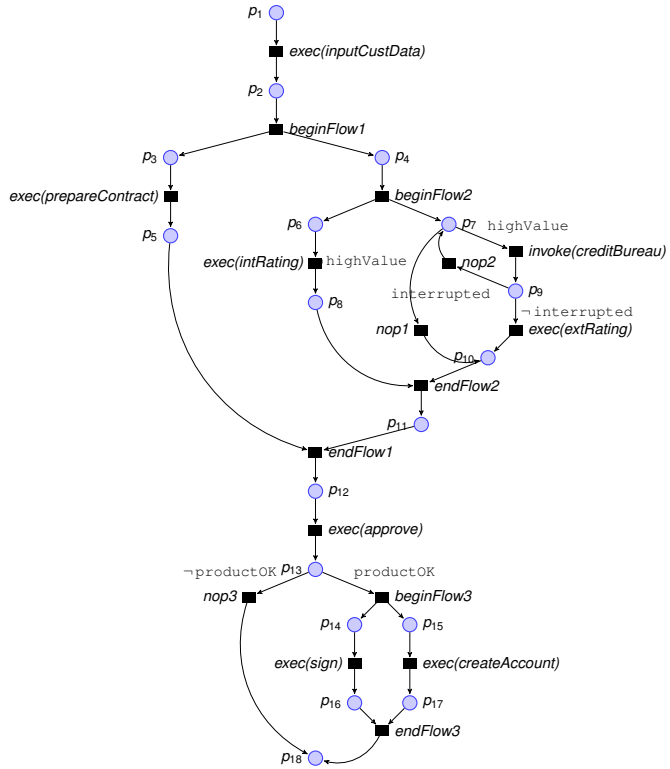
- sets of coordinated activities (workflow), here of the LOP represented with a Petri net.
- subject to an access control policy stating which agent can execute which activity.



LOP - Access Control Policy

Business Processes under authorization constraints are

- sets of coordinated activities (workflow), , here represented with a Petri net.
- subject to an access control policy stating which agent can execute which activity.



- specified by an access control model, e.g. RBAC,
- delegation rules,
- additional security constraints, e.g. Separation of Duty (SoD).

LOP - RBAC Model

- User Assignment (Alice is a director)
- Permission Assignment (a director can execute the approval task)
- Hierarchy of roles (a director is more senior than a manager)

Examples of 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 (`lowrisk`) then `manager` else `director`
`sign`	if (`isindustrial`) then `director` else `manager`

LOP - RBAC Model

- User Assignment (Alice is a director)
- Permission Assignment (a director can execute the approval task)
- Hierarchy of roles (a director is more senior than a manager)

Examples of 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 (lowrisk) then manager else director
sign	if (isindustrial) then director else manager

LOP - Delegation of Tasks

A *delegation* represents a typical flexibility requirement. It can be expressed with

$$\langle \mathit{PreCondition}, \mathit{ARole}, \mathit{DRole}, \mathit{Task} \rangle$$

- If *PreCondition* holds and
- *ARole* is authorized to perform *Task* according to the permission assignment relation,
- then *ARole* can delegate *DRole* to execute *Task*.

Example of delegation rule for the LOP

- D: $\langle \texttt{intRatingOK}, \texttt{director}, \texttt{manager}, \texttt{approve} \rangle$.

Separation of Duty

Critical tasks must be executed by different agents.

Widely used in the financial setting to prevent frauds.

SoD Constraints of the LOP

Name	Critical Tasks
C1	`inputCustData`, `prepareContract`, `intRating`, `extRating`
C2	`intRating`, `approve`
C3	`extRating`, `approve`
C4	`prepareContract`, `approve`, `sign`

Separation of Duty

Critical tasks must be executed by different agents.

Widely used in the financial setting to prevent frauds.

SoD Constraints of the LOP

Name	Critical Tasks
C1	`inputCustData`, `prepareContract`, `intRating`, `extRating`
C2	`intRating`, `approve`
C3	`extRating`, `approve`
C4	`prepareContract`, `approve`, `sign`

Outline

The Action Language $\mathcal{C}$

Expressive propositional action language given by

- fluent and action symbols,
- causal laws, i.e. propositions
  - static laws **caused** F **if** G
  - dynamic laws **caused** F **if** G **after** H

Some useful abbreviations:

Abbreviation	Expanded Form	Informal Meaning
nonexecutable H **if** F.	**caused** $\bot$ **after** $H \wedge F$.	$\neg F$ 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

The CCALC tool supports automated reasoning of $\mathcal{C}$ specifications.

The Action Language $\mathcal{C}$

Expressive propositional action language given by

- fluent and action symbols,
- causal laws, i.e. propositions
  - static laws **caused** F **if** G
  - dynamic laws **caused** F **if** G **after** H

Some useful abbreviations:

Abbreviation	Expanded Form	Informal Meaning
nonexecutable H **if** F.	**caused** $\bot$ **after** $H \wedge F$.	$\neg F$ 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

The CCALC tool supports automated reasoning of $\mathcal{C}$ specifications.

The Action Language $\mathcal{C}$

Expressive propositional action language given by

- fluent and action symbols,
- causal laws, i.e. propositions
  - static laws **caused** F **if** G
  - dynamic laws **caused** F **if** G **after** H

Some useful abbreviations:

Abbreviation	Expanded Form	Informal Meaning
nonexecutable H **if** F.	**caused** $\bot$ **after** $H \wedge F$.	$\neg F$ 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

The CCALC tool supports automated reasoning of $\mathcal{C}$ specifications.

Outline

Expressiveness of the approach

In the context of business processes, with $\mathcal{C}$ it is possible to model:

- execution of tasks,
- (deterministic, non-deterministic, conditional) effects of tasks,
- delegation rules,
- access control policy,
- SoD constraints,
- ...

LOP in $\mathcal{C}$

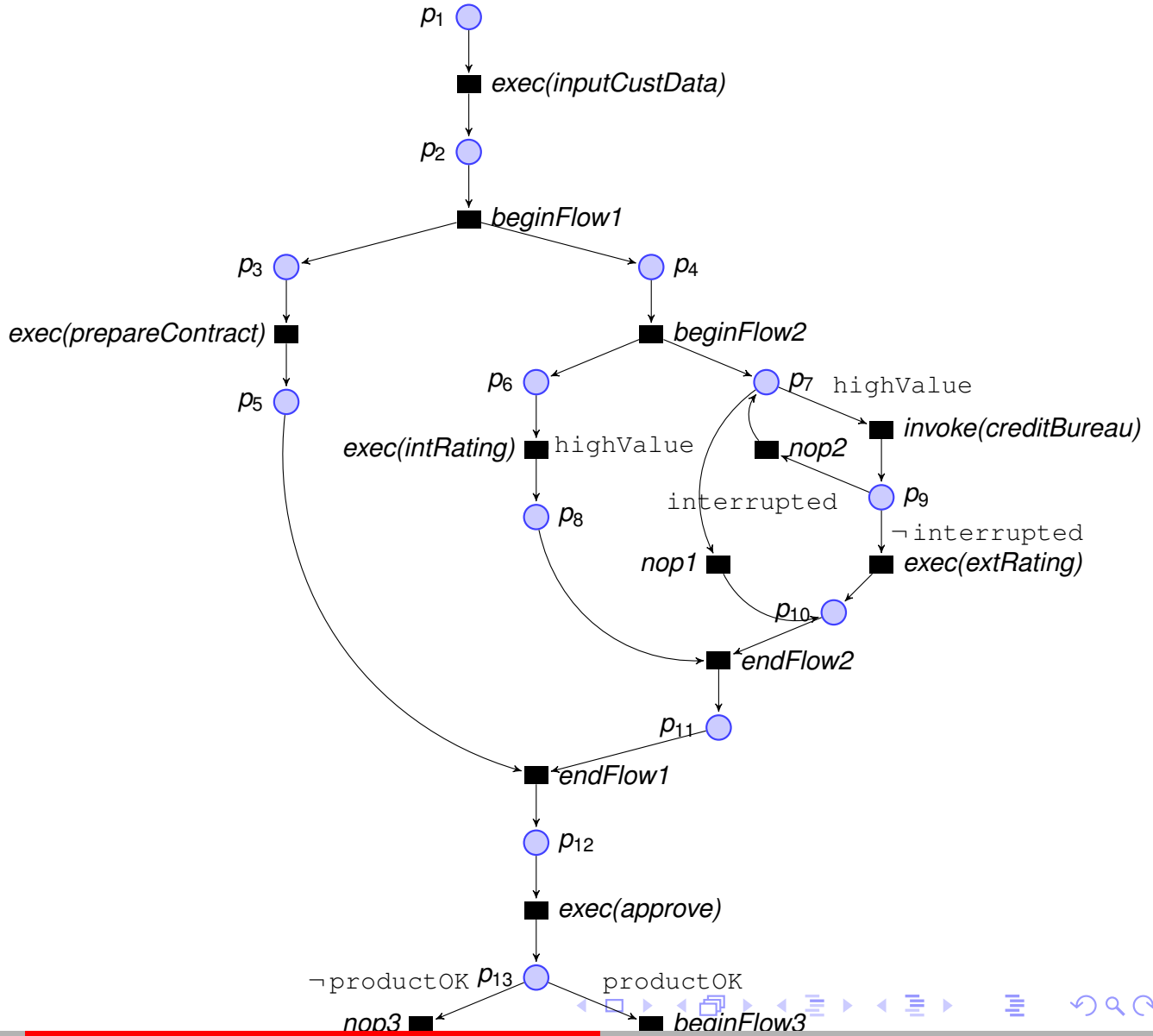
We show how the LOP can be specified in $\mathcal{C}$.

Fluent	Meaning
$\texttt{activated}(a, r)$	agent a is playing role r
$\texttt{ua}(a, r)$	agent a is assigned to role r
$\texttt{pa}(r, t)$	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
$\texttt{delegated}(a, r, t)$	agent a is delegated by an agent in role r to perform task t
$\texttt{executed}(a, r, t)$	agent a has executed task t through role r
$\texttt{senior}(r_1, r_2)$	role r_1 is more senior than or is as senior as r_2
$\texttt{lowrisk}$	the risk associated with the loan is low
$\texttt{interrupted}$	the execution of the process is interrupted
$\texttt{p}_1, \ldots, \texttt{p}_{18}$	the places of the extended Petri net
$\ldots$	$\ldots$

Action	Meaning
$\texttt{exec}(a, r, t)$	execution of task t by agent a by means of role r
$\texttt{d}(a_1, a_2)$	agent a_1 delegates agent a_2 by means of delegation rule d
$\texttt{disable}(a_1, a_2, t)$	agent a_1 disables agent a_2 from performing task t
$\ldots$	$\ldots$

Execution of Tasks in $\mathcal{C}$

Example: task `intRating`



Execution of Tasks in $\mathcal{C}$

Example: task `intRating`

Preconditions

nonexecutable $\texttt{exec}(a, r, \texttt{intRating})$
if $\neg(\texttt{p}_6 \wedge \texttt{granted}(a, r, \texttt{intRating}))$

Deterministic effects

$\texttt{exec}(a, r, \texttt{intRating})$
causes $\texttt{p}_8 \wedge \neg\texttt{p}_6 \wedge \texttt{executed}(a, \texttt{intRating})$

Non-deterministic effects

$\texttt{exec}(a, r, \texttt{intRating})$ **may cause** $\texttt{intRatingOK}$

Conditional effects

$\texttt{exec}(a, r, \texttt{intRating})$
may cause $\texttt{highProfileIndCust}$ **if** $\texttt{isindustrial}$

Delegation Rules in $\mathcal{C}$

Example: delegation rule D
$\langle \texttt{intRatingOK}, \texttt{director}, \texttt{manager}, \texttt{approve} \rangle$.

Preconditions

nonexecutable $d(a_1, a_2)$ **if** $\neg(\texttt{ua}(a_1, \texttt{director}) \wedge \texttt{ua}(a_2, \texttt{manager}) \wedge \texttt{pa}(\texttt{director}, \texttt{approve}) \wedge \texttt{intRatingOK})$

Deterministic effects

$d(a_1, a_2)$ **causes** $\texttt{granted}(a_1, \texttt{director}, \texttt{approve}) \wedge \texttt{delegated}(a_2, \texttt{director}, \texttt{approve})$

for all $a_1 \neq a_2$.

Access Control Policy in $\mathcal{C}$

RBAC policy establishes which agent can perform which task

caused $\texttt{granted}(a, r, t)$
if $(\texttt{ua}(a, r) \wedge \texttt{activated}(a, r) \wedge \texttt{pa}(r_1, t) \wedge \texttt{senior}(r, r_1))$

(and a corresponding law for "causing" $\neg\,\texttt{granted}(a, r, t)$).

A specific permission assignment

caused pa(director, approve) **if** ¬ lowrisk
default ¬ pa(director, approve)
caused pa(manager, approve) **if** lowrisk
default ¬ pa(manager, approve)

Task	Role
...	...
approve	if (lowrisk) then manager else director

Access Control Policy in $\mathcal{C}$

RBAC policy establishes which agent can perform which task

caused $\texttt{granted}(a, r, t)$
if $(\texttt{ua}(a, r) \wedge \texttt{activated}(a, r) \wedge \texttt{pa}(r_1, t) \wedge \texttt{senior}(r, r_1))$

(and a corresponding law for "causing" $\neg\,\texttt{granted}(a, r, t)$).

A specific permission assignment

caused $\texttt{pa}(\texttt{director}, \texttt{approve})$ **if** $\neg\,\texttt{lowrisk}$
default $\neg\,\texttt{pa}(\texttt{director}, \texttt{approve})$
caused $\texttt{pa}(\texttt{manager}, \texttt{approve})$ **if** $\texttt{lowrisk}$
default $\neg\,\texttt{pa}(\texttt{manager}, \texttt{approve})$

Task	Role
...	...
approve	if (lowrisk) then manager else director

Access Control Policy in $\mathcal{C}$

RBAC policy establishes which agent can perform which task

caused $\texttt{granted}(a, r, t)$
if $(\texttt{ua}(a, r) \wedge \texttt{activated}(a, r) \wedge \texttt{pa}(r_1, t) \wedge \texttt{senior}(r, r_1))$

(and a corresponding law for "causing" $\neg\,\texttt{granted}(a, r, t)$).

A specific permission assignment

caused $\texttt{pa}(\texttt{director}, \texttt{approve})$ **if** $\neg\,\texttt{lowrisk}$
default $\neg\,\texttt{pa}(\texttt{director}, \texttt{approve})$
caused $\texttt{pa}(\texttt{manager}, \texttt{approve})$ **if** $\texttt{lowrisk}$
default $\neg\,\texttt{pa}(\texttt{manager}, \texttt{approve})$

Task	Role
...	...
approve	if (lowrisk) then manager else director

Example: SoD constraint C4

constraint $\neg(\texttt{executed}(a, r_1, \texttt{prepareContract}) \wedge \texttt{executed}(a, r_2, \texttt{approve}) \wedge \texttt{executed}(a, r_3, \texttt{sign}))$

Name	Critical Tasks
⋮	⋮
C4	prepareContract, approve, sign

Expressiveness of the approach

With $\mathcal{C}$ it is also possible to model:

- internal dependencies within the process. (e.g. **caused** `lowrisk` **if** $\neg$`highValue` $\wedge$ `intRatingOK`)
- internal and external events, which are not explicitly represented in the model (e.g. activation of a role).
- invocation of entities, e.g. invocation of Credit Bureau,
- all remaining parts of the workflow, e.g. “dummy” activities to begin/end a flow,
- exceptions

Outline

Experiments

Motivation: Given the resulting specifications, and the interplay with the security policy, it may not be trivial to establish if other desirable security properties hold (e.g. because entailed by the already enforced security policy), or if it is necessary to revise the model and/or the security policy in order.

Scenario for the LOP: We defined a complete permission and user assignment, with a number of delegation rules and SoD constraints (comprised the ones seen before).

Problem (Verification of Security Properties)

If the process terminates successfully, then no single agent has performed all the tasks `intRating`, `extRating` if `highValue`, `approve` and `sign`.

We have

1. run CCALC on the related specifications of the problem,
2. to determine whether there exists a path leading to a state in which the same agent performs the tasks `intRating`, `extRating` if `highValue`, `approve`, and `sign`.

Experiments Results

The path holds if the following property is satisfied

$$\begin{aligned} &p_{18} \wedge \texttt{productOK} \wedge \texttt{executed}(a, \texttt{intRating}) \wedge \\ &\quad (\neg \texttt{highValue} \vee \texttt{executed}(a, \texttt{extRating})) \wedge \\ &\quad \texttt{executed}(a, \texttt{approve}) \wedge \texttt{executed}(a, \texttt{sign}) \end{aligned} \quad (1)$$

CCALC found the trace, in few seconds.

The trace shows that (1) is satisfied, and from the trace it is possible to reconstruct the reasons for the violation, i.e. which agent has performed the violation performing which tasks.

Other contributions

- We experimented with other reasoning tasks:
  1. *Synthesis of the Permission Assignment*: for a given number of agents, synthesize a security policy for the business process under given security requirements. In particular, we synthesize the permission assignment for an RBAC model for the LOP given some requirements for the user and the permission assignment.
  2. *Resource Allocation Plan*: for a given number of agents and for all execution flows, find (if any) an assignment of activities to agents ensuring the completion of the business process according to the given security policy.
- Comparison between $\mathcal{C}$ and SMV on this setting.

Outline

Conclusions

Action-based approach to the formal specification, and automatic analysis of business process under authorization constraints that

- is natural and coincise, and
- allows to perform an automatic analysis.

Effectiveness shown by applying it against a business process taken from the banking domain.

Our experiments shows that the proposed approach can be used in a number of reasoning tasks.

References

More deatils on:

A. Armando, E. Giunchiglia, M. Maratea, S.E. Ponta

An Action-based Approach to the Formal Specification and Automated Analysis of Business Processes under Authorization Constraints.

Journal of Computer and Systems Sciences, Vol. 78(1), pg. 119-141, 2012.

Semantics of $\mathcal{C}$

A state is an interpretation of the fluent symbols that satisfies $G \supset F$ for every static law in the action description D.

A transition is any triple $\langle s, a, s' \rangle$ where

- s (s', resp.) is the initial (resulting, resp.) state, and
- a is an action;

A formula F is caused in a transition $\langle s, a, s' \rangle$ if it is

- the head of a static law from D such that s' satisfies G, or
- the head of a dynamic law 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 the fluent symbols that satisfies all formulas caused in this transition.

The *transition diagram* represented by 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.

Abbreviations for Causal Laws

Abbreviation	Expanded Form	Informal Meaning
nonexecutable H **if** F.	**caused** $\bot$ **after** $H \wedge F$.	$\neg F$ 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
inertial F.	**caused** F **if** F **after** F.	F is an inertial fluent
constraint F.	**caused** $\bot$ **if** $\neg F$.	F must be true

Immediate Invocation of Entities in $\mathcal{C}$

- Actions executed as soon as their preconditions hold.

Example: invocation of the Credit Bureau

nonexecutable $\texttt{invoke}(a, r, \texttt{creditBureau})$
if $\neg\,\texttt{granted}(a, r, \texttt{extRating}) \vee \neg\,\texttt{p}_6 \vee \texttt{lowrisk}\,.$

caused $\texttt{invoke}(a, r, \texttt{creditBureau})$
if $\texttt{granted}(a, r, \texttt{extRating}) \wedge \texttt{p}_6 \wedge \neg\,\texttt{lowrisk}\,.$

$\texttt{invoke}(a, r, \texttt{creditBureau})$
causes $\neg\,\texttt{p}_6 \wedge \texttt{invoked}(a, r, \texttt{creditBureau}) \wedge \texttt{p}_7\,.$

Internal Dependencies in $\mathcal{C}$

- The value of fluents can be determined by the interaction of other process features.

Example: the interruption of the process

caused $\texttt{interrupted}$ **if** $\texttt{p}_7 \wedge$
$\texttt{invoked}(a_1, r, \texttt{creditBureau}) \wedge \texttt{accessed}(a_2)$.

caused $\neg\texttt{interrupted}$ **if** $\texttt{p}_7 \wedge$
$\texttt{invoked}(a, \texttt{director}, \texttt{creditBureau})$.

for all $a_1 \neq a_2$ and $r \neq \texttt{director}$ and where $\texttt{interrupted}$ is an inertial fluent

Events in $\mathcal{C}$

- Events influencing the process whose causes often do not need to be modeled in the scope of the process.
- Expressed by exogenous fluents (fluents that can arbitrarily change value in the transition (unless there is some other rule constraining their value).

Example: the non reversible access to the information exchanged with Credit Bureau by an unauthorized agent

caused `accessed`(a) **after** `accessed`(a).

where `accessed`(a) is an exogenous fluent.

The Causal Calculator (CCALC)

Given a $\mathcal{C}$ specification, an initial state and a desired state, CCALC automatically

- establishes if there exists an execution path starting for the initial state and reaching the desired state, and
- in case the execution path exists, CCALC returns a trace representing the execution path.

Causing $\neg$ granted

$$\textbf{caused} \ \neg \texttt{granted}(a, r, t) \ \textbf{if} \ \neg \texttt{ua}(a, r) \vee \neg \texttt{activated}(a, r) \vee$$

$$\left(\bigwedge_{r_1 \in \mathcal{R}} (\neg \texttt{pa}(r_1, t) \vee \neg \texttt{senior}(r, r_1)) \right)$$

where $\mathcal{R}$ is the set of roles involved in the business process.

Some advantages of *C* vs NuSMV

- 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 policy are best viewed as a collection of actions subject to a given workflow pattern and a set of independent access control rules.
- Moreover $\mathcal{C}$ provides modelers with the ability to specify the system incrementally: This is an important feature that is less supported by the specification languages of state-of-the-art model-checkers.