

# Modular Synthesis of Enforcement Mechanisms for the Workflow Satisfiability Problem: Scalability and Reusability

**Daniel Ricardo dos Santos<sup>1,2,3</sup>,**  
Serena Elisa Ponta<sup>2</sup>, Silvio Ranise<sup>1</sup>

<sup>1</sup>Fondazione Bruno Kessler (FBK)

<sup>2</sup>SAP Labs France

<sup>3</sup>University of Trento



## SECENTIS

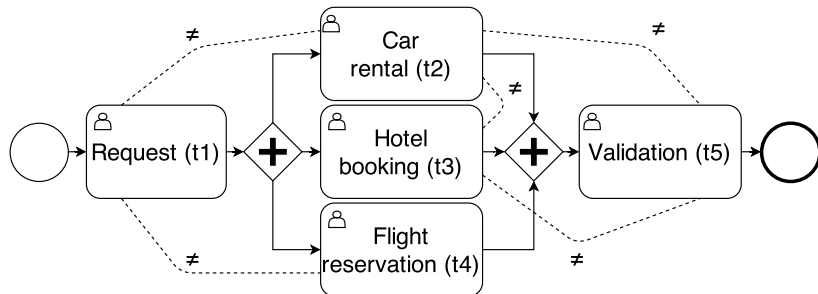
A European Industrial Doctorate on Security and Trust

# Outline

- 1 Introduction
- 2 Modular Design and Enactment
- 3 Applications
- 4 Conclusion

# Context

- A **workflow** specifies a collection of **tasks** and the causal relationships between them
- Authorization **policy** specifies which users can execute which tasks
- Additional **constraints**, such as Separation/Binding of Duty, further restrict the execution of tasks by users
  - Important to avoid errors and fraud, limiting opportunities for abuse
- We call those workflows **security-sensitive**



task	roles
t1	r3
t2	r2
t3	r2
t4	r1
t5	r2

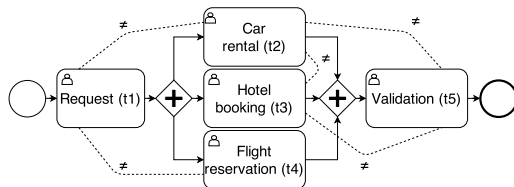
roles	users
r1	a
r2	a, b, c
r3	a, b

# Workflow Satisfiability Problem

## WSP

Is there an **assignment** of users to tasks such that a workflow **terminates** while satisfying all authorization constraints?

# Problem

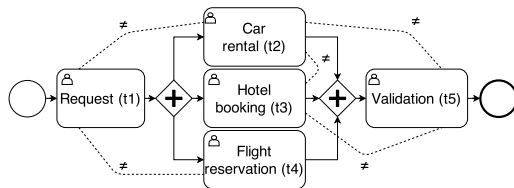


task	roles
t1	r3
t2	r2
t3	r2
t4	r1
t5	r2

roles	users
r1	a
r2	a, b, c
r3	a, b

Execution = t1(a), t2(b), t3(c), **X**

# WSP solution



task	roles
t1	r3
t2	r2
t3	r2
t4	r1
t5	r2

roles	users
r1	a
r2	a, b, c
r3	a, b

Execution = t1(b), t2(a), t3(c), t4(a), t5(b)

# Run-time WSP

## Run-time WSP

Answering sequences of user **requests** at execution time **ensuring termination** with the satisfaction of authorization constraints

WSP is computationally hard, but run-time version allows us to divide the problem in two steps



# Solving the WSP - previous work<sup>1</sup>

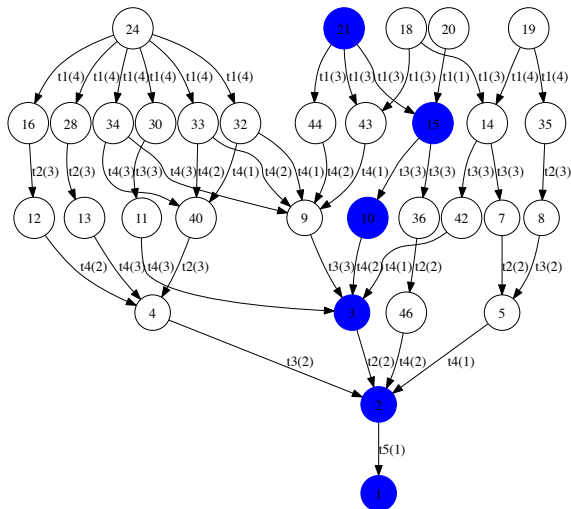
- Technique for the synthesis of **run-time monitors** for the WSP in two steps:
  - **Off-line**: takes as input the specification of control-flow and constraints
  - **On-line**: takes as input the authorization policy
- Off-line result is based on a set of **symbolic users** and an unconstrained policy
  - Supports different authorization policies at run-time
- On-line result is a set of queries  **$can\_do(u, t)$**  that encode the conditions under which a user can perform a task while ensuring satisfiability

<sup>1</sup> - C. Bertolissi, D.R. dos Santos, S. Ranise, "Automated Synthesis of Run-time Monitors to Enforce Authorization Policies in Business Processes", ASIACCS 2015

# Off-line

- A **symbolic transition system**  $S = (V, Tr)$  is derived from a workflow specification
  - $V$  contains variables related to execution and authorization constraints
  - $Tr$  contains transitions  $t(u) : en \rightarrow act$
- $S$  is used to compute a **symbolic reachability graph**  $RG$ 
  - Nodes labeled by a symbolic representation of the set of states from which it is possible to reach a state in which the workflow successfully terminates
  - Edges labeled by task-user pairs (symbolic users)
- Any path from a leaf node to the root corresponds to a **symbolic execution**

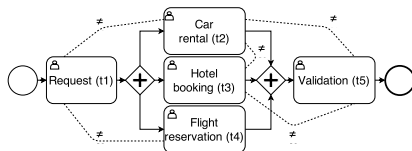
# Off-line - Symbolic Reachability Graph



# On-line

- A Datalog program is derived from  $RG$  by building clauses from the formulae **labeling the nodes**
- Each clause invokes predicates *auth* (interface to the authorization policy) and *h* (keeps track of which user has executed which task)
- If the monitor **grants a request** (clause is satisfied), then the user can execute the task and the workflow can terminate while satisfying the authorization policy and the authorization constraints

# Run-time WSP solution



task	roles
t1	r3
t2	r2
t3	r2
t4	r1
t5	r2

roles	users
r1	a
r2	a, b, c
r3	a, b

#	History	Query	Answer
0	$\emptyset$	$can\_do(a, t1)$	deny
1	-	$can\_do(b, t1)$	grant
2	$h(t1, b)$	$can\_do(c, t3)$	grant
3	$h(t3, c)$	$can\_do(a, t4)$	grant
4	$h(t4, a)$	$can\_do(b, t2)$	deny
5	-	$can\_do(a, t2)$	grant
6	$h(t2, a)$	$can\_do(b, t5)$	grant
7	$h(t5, b)$	-	-

# Problems with the previous solution

- How to scale monitor synthesis and handle **large workflows**?
- How to reuse already specified modules **across processes**?
- How to **specify and enforce** authorization constraints that span **multiple modules**?

# Contributions of this paper

- Define security-sensitive workflow **components** equipped with **interfaces** that allow to glue components together
- Define **gluing assertions** to specify the constraints between components (**control-flow and authorization**)
- Automatically **synthesize run-time monitors** from workflow components, ensuring that all tasks can be executed without violating the policy or the constraints

# Outline

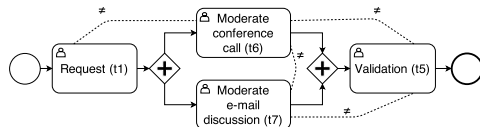
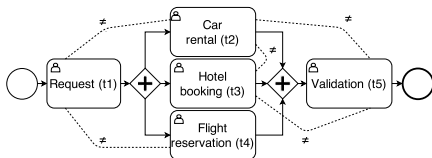
- 1 Introduction
- 2 Modular Design and Enactment
- 3 Applications
- 4 Conclusion



# Modularity

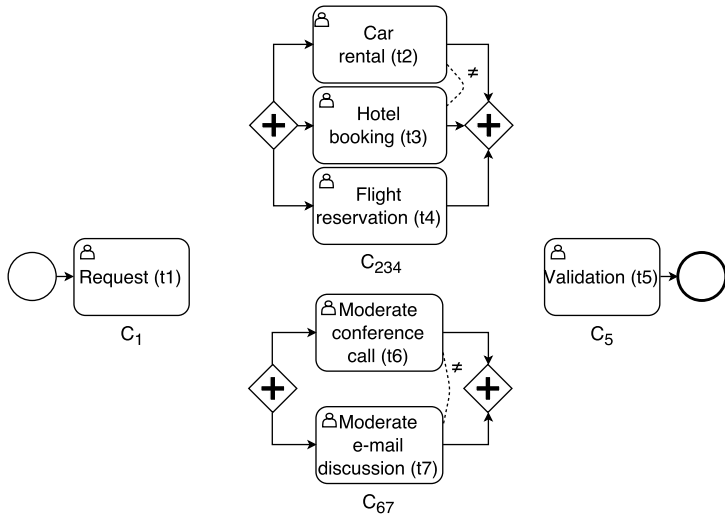
- **Modularity** is important for better and more reusable models
- Business process libraries and **repositories**
  - Allow designers to **store, retrieve and combine** reusable workflow models
- **End-to-end** processes spanning multiple workflows
  - **Composition** of smaller models, e.g., Purchase Order and Warehouse Management
  - Inter-module constraints

# Example

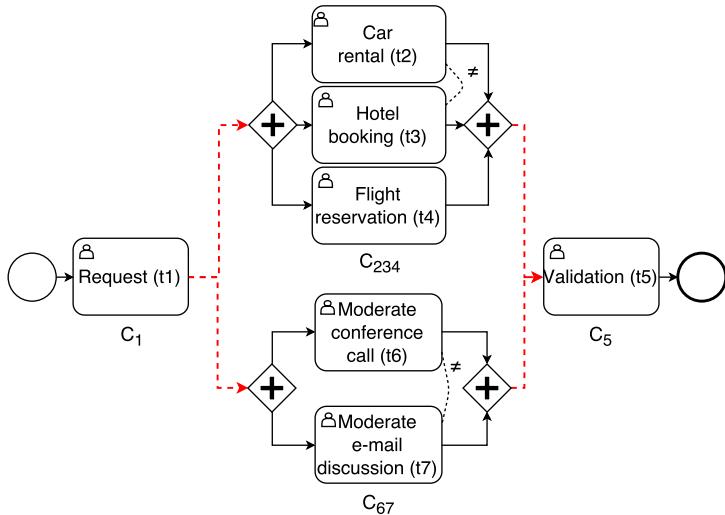


- Designer would like to reuse  $t1$  and  $t5$  for both workflows, while modeling from scratch only  $t2, t3, t4$  and  $t6, t7$
- Define them as **components** and specify complete workflows by combining the reused components with the new ones
  - $C_1 \oplus_G C_{234} \oplus_G C_5$  and  $C_1 \oplus_G C_{67} \oplus_G C_5$
- Obtain a monitor for the composed workflows

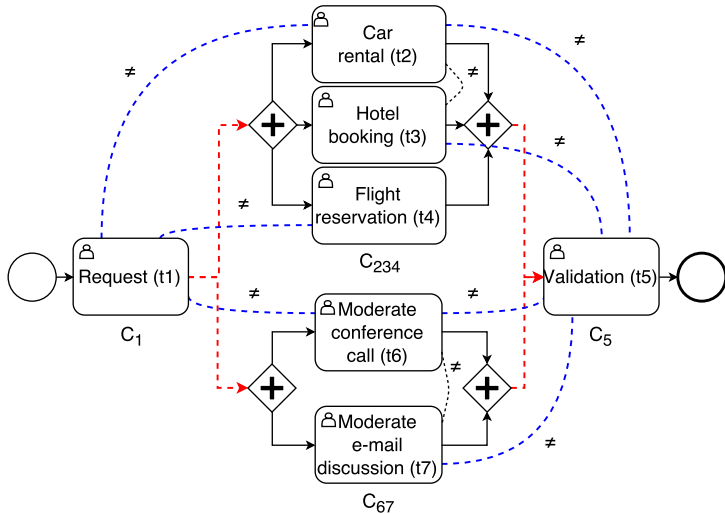
# Example - components



# Example - control-flow



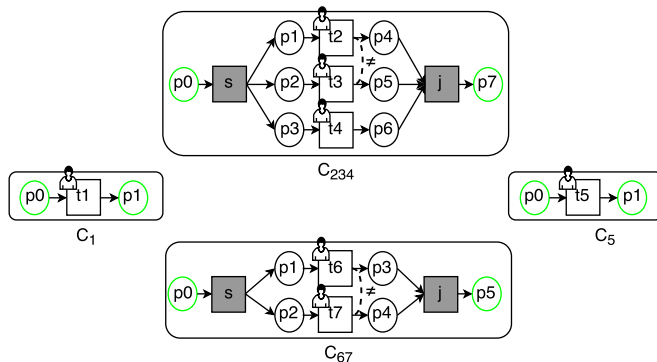
# Example - constraints



# Enforcement - monitor synthesis

- What to do:
  - Translate component specifications to transition systems
  - Synthesize and store monitors for new components
  - Retrieve and combine monitors for reused components
- How to do it:
  - Refine the transition systems from previous work
  - Define Components, Interfaces, and Gluing assertions
  - Use the same monitor synthesis procedure

# Components



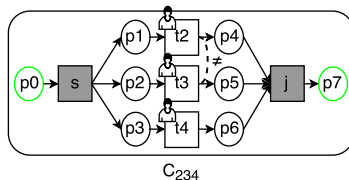
Component  $C = (S', Int)$  contains a refined transition system  $S'$  and an interface  $Int$

# Transition system refinement

- $S = (V, Tr)$  refined to  $S' = ((P, D, A, H, C), Tr', B)$ 
  - $V = P \cup D \cup A \cup H \cup C$
  - $B$  specifies internal authorization constraints
  - Transitions in  $Tr'$  contain variables from  $V$  and  $B$
- $S'$  is a refinement for composition, and  $S$  can be obtained back from  $S'$



# Example - Transition system refinement



## Transitions

$$t2(u) : p1 \wedge \neg d_{t2} \wedge a_{t2}(u) \wedge c_{t2}(u) \wedge c_{t2}^i(u) \rightarrow$$

$$p1, p4, d_{t2}, h_{t2}(u) := F, T, T, T$$

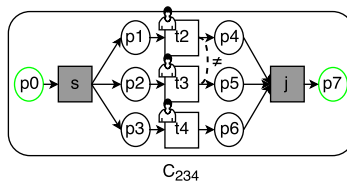
## Constraints

$$B = \{\forall u. c_{t2}(u) \Leftrightarrow \neg h_{t3}(u)\}$$

# Interface

- $Int = (A, P^i, P^o, H^o, C^i)$  defines the variables of  $S'$  that can be set by another component (**input**) and those that are set by the component itself (**output**)
  - $A$  is defined by the authorization policy
  - $P^i, P^o$  are the places that connect components (control-flow)
  - $H^o$  transfers the execution history between components
  - $C^i$  are the inter-component authorization constraints

# Example - Interface

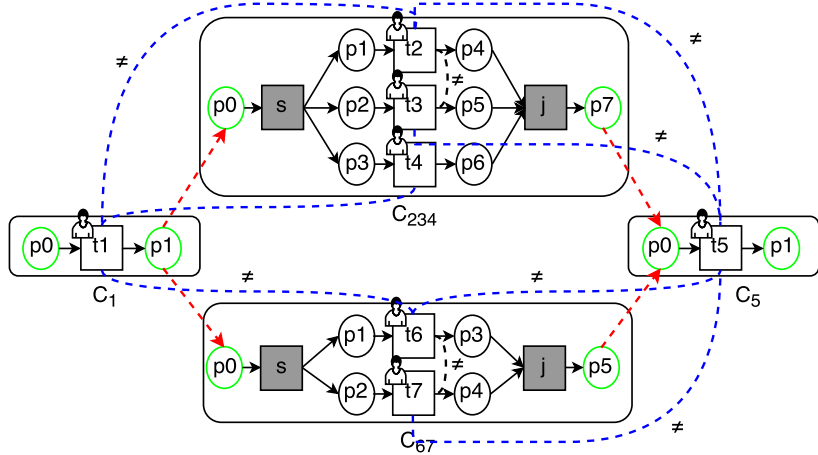


## Interface

$$P^i = \{p0\}, P^o = \{p7\},$$

$$H^o = \{h_{t2}, h_{t4}\}, C^i = \{c_{t2}^i, c_{t4}^i\}$$

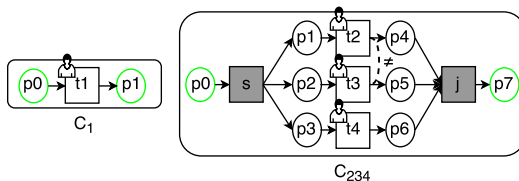
# Gluings



# Gluing

- $G = G_{EC} \cup G_{Auth}$  is a set of gluing assertions over  $Int_1$  and  $Int_2$ 
  - inter-module execution constraints:  $p_1^i \Leftrightarrow p_2^o$
  - inter-module authorization constraints:  $C_1^i \cup H_2^o$
- $S = S_1 \oplus_G S_2$ 
  - $\oplus_G$  combines the variables and transitions in  $S_1$  and  $S_2$  to form a new component  $S$
  - commutative and associative: result is independent of the order

# Example - Gluing



## Assertions

$$G_{EC} = \{p1_1 \Leftrightarrow p0_{234}\}$$

$$G_{Auth} = \{\forall u. c_{t2}^i(u) \Leftrightarrow \neg h_{t1}(u), \forall u. c_{t4}^i(u) \Leftrightarrow \neg h_{t1}(u)\}$$

# Monitor Synthesis

- Workflow systems (**original**) and workflow components (**refined**) are equivalent
  - Define a procedure  $\mathcal{M}$  that takes as input a component  $S' = ((P, D, A, H, C), Tr', B)$  and returns a transition system  $S = (V, Tr)$
- Let  $\mathcal{RM}$  be the procedure (from previous work) that takes as input a transition system and returns a Datalog program  $\mathcal{RM}(S)$ 
  - Then  $\mathcal{RM}_C(S_C) = \mathcal{RM}(\mathcal{M}(S_C))$

# Monitor synthesis

- A monitor for components  $C_1$  and  $C_2$  is obtained by **concatenating** the Datalog monitors for the constituent **components** and adding clauses derived from the **gluing assertions**
- Monitors for **components** are computed by considering **all possible values** for the variables in the interfaces
- Gluing **assertions** consider a **sub-set** of these values:
  - how **execution** flows between components
  - how **authorization** further constraints possible executions



# Outline

- 1 Introduction
- 2 Modular Design and Enactment
- 3 Applications**
- 4 Conclusion

# Applications

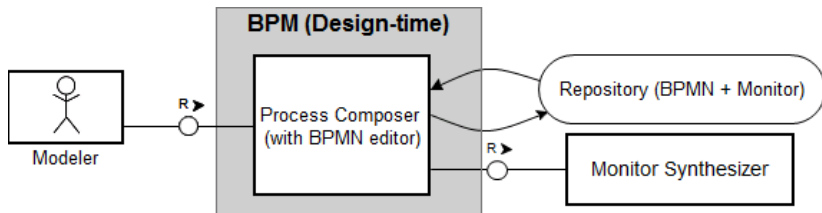
- The two main applications are: **scalability and reuse**
- Scalability allows the synthesis of monitors for **large workflows**
  - Modularly **specified**
  - Automatically **decomposed**
- Reuse allows designers to **store/retrieve and combine** workflow models alongside monitors in a **repository**

# Scalability

- Experimented with a random workflow generator
  - Generate  $n$  workflows of 5 tasks and compose them sequentially
  - Varying number of constraints (intra- and inter-modules)
- Workflows of up to 500 tasks (100 components), with 5%, 10%, and 20% of the tasks in a constraint
  - Results show a linear growth in time to synthesize monitors, as opposed to the exponential explosion when using a monolithic technique
- Decomposition is left as future work

# Reuse

- **General architecture** for WFMS integration
  - **Back-end** in Python using the MCMT model checker
  - **Front-end** adapted for different WFMS systems (e.g., SAP HANA Operational Intelligence)



# Outline

- 1 Introduction
- 2 Modular Design and Enactment
- 3 Applications
- 4 Conclusion

# Conclusion

- **Modular** approach to the **specification** of security-sensitive workflows and **synthesis** of run-time monitors for the WSP
  - It provides **scalability and reuse**
- Step towards **practical** enforcement mechanisms for security-sensitive workflows

Thank you!

[dossantos@fbk.eu](mailto:dossantos@fbk.eu)

[www.secentis.eu](http://www.secentis.eu)



UNIVERSITY  
OF TRENTO