Imperial College London

Security as a Resource in Process-Aware Information Systems

Michael Huth

14 October 2011

Joint Work with:

- Jason Crampton
  Information Security Group
  Royal Holloway
- Jim Huan-Pu Kuo
  Department of Computing
  Imperial College London

Imperial College
London

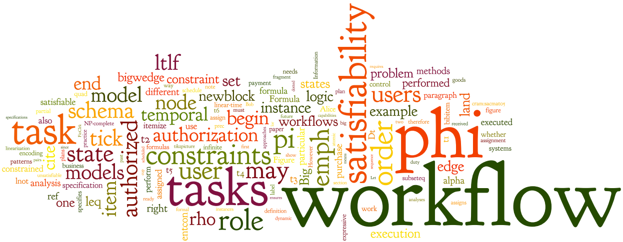
Outline of talk

Imperial College London

Authorized Workflows

Imperial College
London

Wordle from Relevant Paper



What are workflows to us?

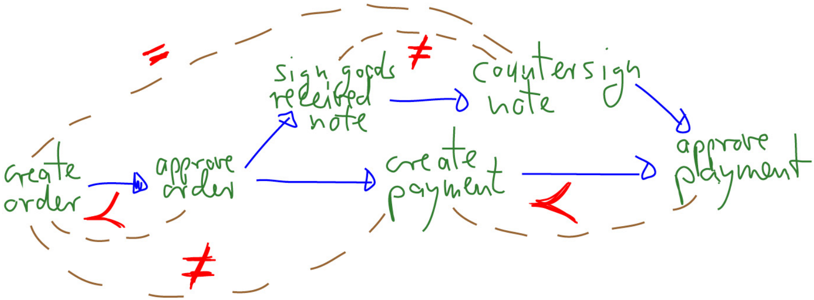
- Plans or schedules that map users or resources to tasks
- Such mappings may be constrained, e.g. Binding of Duty
- Security policy may prevent some user/task combinations
- Business objectives or legal requirements may further constrain workflow
- Temporal order of tasks may be constrained

A workflow is such a plan that meets all constraints.

Why are workflows interesting?

- Important technology, e.g.
  - Business process management systems
  - Cloud-based collaboration services, e.g. `inkspotscience.com`
- Industrial practice of workflows is
  - often flawed and uses ad hoc methods
  - rarely takes into account security considerations
- Academic methods brittle under change of models
- Most analysis problems NP-hard
- Model-based approaches to design and analysis of workflows have potential impact

Example workflow specification



Blue edges: temporal constraints. Binding of users to tasks constrained by equality $=$, inequality $\neq$, and seniority $\prec$.

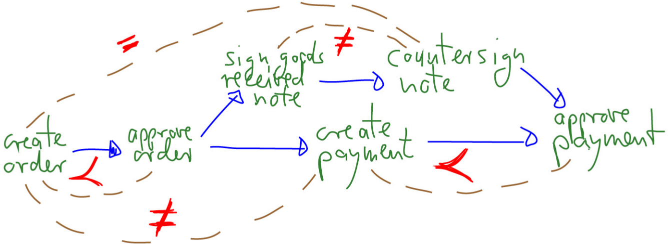
Representative specification formalism

Specification of authorization system $\mathcal{AS}$ comprised of:

- $(T, \leq)$ finite partial order of tasks:
  $t < t'$ means t has to precede t'
- U set of users
- $A \subseteq T \times U$ where (t, u) in A means:
  u authorized to execute task t
- C set of entailment constraints of form $(D, t \rightarrow t', \rho)$
  - $D \subseteq U$ and $\rho \subseteq U \times U$
  - meaning: if u in D and assigned to task t, then user u' assigned to t' is such that (u, u') is in ρ
- e.g. $=$ as ρ and D as U gives Binding of Duty

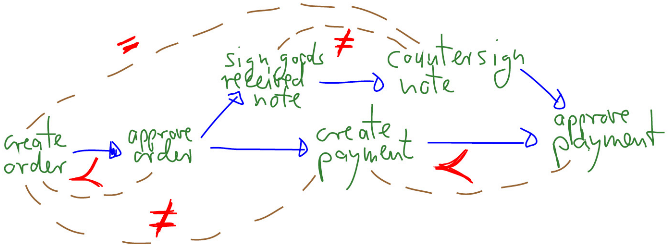
Unrealizabile workflow example

- Alice hasRole FinAdmin, Bob hasRole FinClerk
- FinAdmin authorized to approve orders and payments
- FinClerk authorized to all other tasks
- Workflow below not realizable: Alice is most senior person



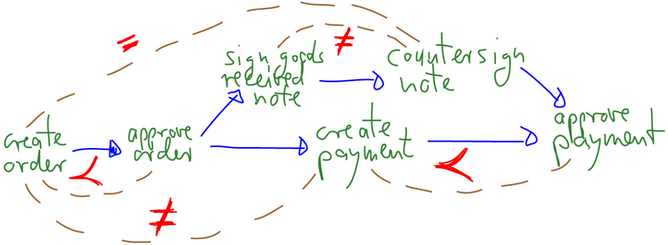
Unrealizabile workflow: details

- If Alice creates order, no senior person can approve it
- If Bob creates order, Alice needs to approve it ($\prec$)
- But Alice also has to create payment because of $\neq$
- But then there is no senior person to approve it



Repair advice for unrealizabile workflow

- Realizable by adding Carol hasRole FinClerk:
  - Alice approves order and payment
  - Bob creates order and countersigns note
  - Carol creates payment and signs goods received note



Imperial College London

Synthesizing Authorized Workflows

Synthesizing secure workflows in LTL(F)

- Translate a workflow specification $\mathcal{AS}$ into formula $\phi_{\mathcal{AS}}$ of NP-complete linear-time temporal logic fragment LTL(F)
- Show: authorized workflow translates into model of $\phi_{\mathcal{AS}}$
- Conversely, show that any model of $\phi_{\mathcal{AS}}$ translates into authorized workflow
- So we can synthesize authorized workflows for $\mathcal{AS}$ by
  - generating $\phi_{\mathcal{AS}}$ from $\mathcal{AS}$
  - running a model checker on the fully connected model ...
  - ...with the negation of $\phi_{\mathcal{AS}}$ as query

Temporal logic LTL(F)

- Syntax where p is from set of atomic propositions AP:

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \phi \mid \mathsf{F}\,\phi$$

- F temporal connective "Future", and "Globally" $\mathsf{G}\,\phi$ is defined as $\neg\mathsf{F}\neg\phi$
- Semantics via infinite sequence of states $\pi = s_0 s_1 \ldots$ where each s_i subset of AP:

$\pi \models p$ iff $p \in s_0$

$\pi \models \neg\phi$ iff not $\pi \models \phi$

$\pi \models \phi_1 \wedge \phi_2$ iff $(\pi \models \phi_1$ and $\pi \models \phi_2)$

$\pi \models \mathsf{F}\,\phi$ iff there is $i \geq 0$ with $\pi^i \models \phi$,
where π^i is the infinite suffix $s_i s_{i+1} \ldots$ of π

Formula $\phi_{\mathcal{AS}}$ for model checker

$$\phi_{FT} = \bigwedge_{t\in T} \mathsf{F}\, t \qquad \phi_{GT} = \mathsf{G}\left(\bigvee_{t\in T} t\right) \qquad \phi_{GU} = \mathsf{G}\left(\bigvee_{u\in U} u\right)$$

$$\phi_{\leq} = \bigwedge_{t\in T} \mathsf{G}\left(t \to \mathsf{G}(\bigvee_{t'\not\leq t} t')\right)$$

$$\phi_{seU} = \bigwedge_{u\in U} \mathsf{G}\left(u \to \bigwedge_{u'\in U\setminus\{u\}} \neg u'\right)$$

$$\phi_{seT} = \bigwedge_{t\in T} \mathsf{G}\left(t \to \bigwedge_{t'\in T\setminus\{t\}} \neg t'\right)$$

$$\phi_{A} = \bigwedge_{t\in T} \mathsf{G}\left(t \to \bigvee_{(t,u)\in A} u\right) \qquad \phi_{C} = \bigwedge_{(D,t\to t',\rho)\in C} \phi_{(D,t\to t',\rho)}$$

$$\phi_{(D,t\to t',\rho)} = \bigwedge_{u\in D} (\mathsf{F}\,(t\wedge u)) \to \mathsf{G}\left(t' \to \bigvee_{(u,u')\in\rho} u'\right)$$

$$\phi_{\mathcal{AS}} = \phi_{FT} \wedge \phi_{GT} \wedge \phi_{GU} \wedge \phi_{\leq} \wedge \phi_{\checkmark} \wedge \phi_{seU} \wedge \phi_{seT} \wedge \phi_{A} \wedge \phi_{C}$$

Imperial College London

Experimental setup: declare tasks and users

```
MODULE main

VAR

createPurchaseOrder : boolean;
approvePurchaseOrder : boolean;
signGoodsReceivedNote : boolean;
createPayment : boolean;
countersignGoodsReceivedNote : boolean;
approvePayment : boolean:

bob : boolean; alice : boolean; carol : boolean;
...
```

Experimental setup: declare behavior and spec

```
...
INIT   -- all states are initial ones
TRUE

TRANS  -- all states transition to all states
TRUE

-- claim that all paths satisfy negation of phi_AS
-- "counterexample" is realizability witness
LTLSPEC ! ( phi_AS)
```

Parameterized analysis tool

- Model-checking algorithm works for all formulas of LTL(F)
- No need to invent new analyses, if written in LTL(F), e.g.
- Schedulability with constraints across workflow instances:
  - write $\phi'_{\mathcal{AS}}$ for $\phi_{\mathcal{AS}}$ with each p replaced by p'
  - check two instances of workflow are realizable where . . .
  - . . . task t executed by different users in each instance:

$$\phi_{\mathcal{AS}} \wedge \phi'_{\mathcal{AS}} \wedge \bigwedge_{u \in U} (\mathsf{F}\,(t \wedge u) \to \mathsf{G}(t' \to \neg u'))$$

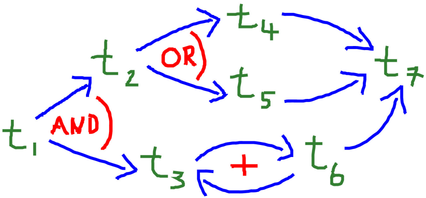
- But how to use a model checker to compute repair advice?

Imperial College London

More Expressive Workflows

Task choices and task iteration

- Need to support conjunction of paths (default in temporal order)
- Need to support disjunction in paths: adaptive, non-determinstic flows
- Need to support bounded iteration of tasks



Domain-specific languages

- Choice of language driven by use context, e.g.
- visual and control-flow oriented for front-end modeling language (e.g. commercial ones such as BPMN)
- textual and tool-independent intermediate language or
- languages comitted to particular tool and modeling paradigm such as Petri nets or process algebras

Example DSL: a "process algebra"

V, W ::=		Workflows
	t	(Atomic Workflow)
	$W^{\leq m}$	(Bounded Iteration)
	$V; W$	(Sequential Composition)
	choose k from $\mathbf{W}$	(Threshold Choice)

- choose k from $\mathbf{W}$ means exactly k workflow specifications from set $\mathbf{W}$ scheduled
- gives OR-fork and OR-join for $k = 1$
- gives AND-fork and AND-join for $k = |\mathbf{W}|$

Example DSL: textual choices and iteration

- Interaction between non-deterministic choice and iteration:
- if task `t` is not chosen, we may want to ignore multiplicities

```
TASK_CHOICES {
  (t1 && t2 && t3 && !t4) || (!t1 && t2 && t3)
}

TASK_MULTIPLICITIES { -- default is [1,1]
  t1[2,4];
  t3[1,3]
  t4[1,2];
}
```

Encoding task choices in LTL(F)

- Tasks declared as atomic propositions
- Task choices declared as Boolean formula over tasks
- Wrap each atomic formula into a Future modality
- For example, the task choice declaration

$$(t_1 \wedge t_2 \wedge t_3 \wedge \neg t_4) \vee (\neg t_1 \wedge t_2 \wedge t_3)$$

- ... has as LTL(F) encoding the formula

$$(\mathrm{F}\, t_1 \wedge \mathrm{F}\, t_2 \wedge \mathrm{F}\, t_3 \wedge \neg \mathrm{F}\, t_4) \vee (\neg \mathrm{F}\, t_1 \wedge \mathrm{F}\, t_2 \wedge \mathrm{F}\, t_3)$$

Encoding task multiplicities in LTL

- Need to reflect on possibility that task is not chosen
- Need to enforce lower bounds if chosen
- Need to enforce upper bounds in any event
- Declaration `t[2,3]`, e.g., has LTL encoding

$$((\mathsf{F}\, t) \rightarrow \mathsf{AtLeast}(2, t)) \wedge \mathsf{AtMost}(3, t)$$

- ... where $\mathsf{AtLeast}(k, t)$ and $\mathsf{AtMost}(k, t)$ are defined next

Encoding lower bounds on occurrence in LTL

- AtLeast(k, t) says t occurs at least k many times
- encoding uses operators Strong Until U and Next X:
- specify encoding in generality (above, ϕ is t):

$$\begin{aligned} \mathrm{AtLeast}(0, \phi) &= \texttt{true} \\ \mathrm{AtLeast}(k+1, \phi) &= (\neg\phi\,\mathrm{U}\,(\phi \wedge (\mathrm{X}\,\mathrm{AtLeast}(k, \phi)))) \end{aligned}$$

Encoding upper bounds on occurrence in LTL

- AtMost(k, t) says t occurs at most k many times
- encoding now uses Weak Until operator W and Next X:

$$\begin{aligned} \mathrm{AtMost}(0, \phi) &= (\mathrm{G}\, \neg\phi) \\ \mathrm{AtMost}(k+1, \phi) &= (\neg\phi\, \mathrm{W}\, (\phi \wedge (\mathrm{X}(\mathrm{AtMost}(k, \phi))))) \end{aligned}$$

Advantages of parameterized approach

- many, e.g., `alice executes t1[0..3]`
- may model that Alice can execute task `t1` at most three times
- can encode this as

$$\mathrm{AtMost}(3, \mathit{alice} \wedge t_1)$$

- But: LTL model checking is exponential in nesting of Untils, i.e. in size of multiplicities

(De)composition

Composing Workflows

- Workflows specified as process terms:
  - Can draw from work on process algebras
  - But this mostly deals with composition of control flow
- When workflows are also constrained:
  - How should we compose constraints?
  - How should we compose constraints and control flow?

Composition example

- Let tasks `t1` and `t2` be such that different users need to execute them
- Say that both tasks can happen at most four times
- There are at least two senses in which we could think of task repetition as a composition:
  - All users who execute all instances of task `t1` are different from all users that execute all instances of task `t2`
  - The users are different across specific pairs of instances of tasks `t1` and `t2`
- Composition mechanisms should be able to accommodate and articulate both views

Information Security as Resource

- Ideally, capture information security as constraints
- Then compose these constraints with other models
- Composition should guarantee desired security policies
- Well understood in types, e.g.: enrich a normal type with a security label

How to do this for secure workflows?

Challenges for Authorized Workflows

- If temporal constraints independent from other constraints: can decompose scheduling of tasks and solving of constraints
- Even solving of other constraints alone is NP-hard (presence of $=$ and $!=$ suffices)
- Alternatives, e.g. modal logics with nominals or description logic equivalents are often undecidable

First experimental results (work in progress)

- Randomly generated workflows, models with 10-100 tasks and 5-60 users
- Code generator generates such models and transforms them into NuSMV models
- Then we do LTL model checking on those models
- For models with about 30 tasks, model checking may take only minutes, but it can take hours
- Also, model checkers do not report back an “unsatisfiable core” for diagnostics

Decomposition (Work in progress)

- Do not encode temporal order of tasks in LTL(F), indpendent topological sort
- Allow sets of tasks and users at states
- Constraints solution now maps tasks t to user sets U_t (any choice from U_t will work)
- Potentially much shorter paths than the number of tasks, e.g. three states for six tasks:

$$\begin{aligned} s_0 &= \{approveOrder, approvePayment, Alice\} \\ s_1 &= \{createOrder, countersignNote, Bob\} \\ s_2 &= \{createPayment, signreceivedNote, Carol\} \end{aligned}$$

- Experiments now solve models with up to 100 tasks in time that ranges from seconds to minutes

Wrapping Up

Conclusions

- We presented workflows and ways in which to enrich them with security constraints
- We saw how to encode realizability of secure workflows as an LTL satisfiability problem
- We discussed pluses and minuses of such an approach to realizability analysis
- We speculated about more expressive workflows and (de)composition principles
- And we reported first experimental results

Future Work

- What are effective tools and algorithms for reasoning about realizability of secure workflows?
- How can one compute repair advise for unrealizable secure workflows?
- How should one model composition of secure workflows?
- Is it beneficial to think of administrative security management as a secure workflow?
- How should collaboration under imperfect information be modeled in secure workflows?

Imperial College London

References

- Crampton, J., and Huth, M. (2011) On the Modeling and Verification of Security-Aware and Process-Aware Information Systems. Proc. Int'l BPM Workshop WfSAC. Springer Lecture Notes in Business Information Processing. To appear.
- Rozier, K. Y., and Vardi, M. Y. (2010) LTL satisfiability checking. Software Tools and Technology Transfer 12:123-137
- van der Aalst, W., Pesic, M., Schonenberg, H. (2009) Declarative workflows: Balancing between flexibility and support. Computer Science - R & D 23(2):99-113
- van der Aalst, W., and ter Hofstede, A. (2005) YAWL: yet another workflow language. Information Systems 30(4):245-275
- Sistla, A. P., and Clarke, E. M. (1985) The complexity of propositional linear temporal logics. Journal of the ACM 32:733-749

Q & A

Q & A

Q & A

Q & A