Überprüfung auf Einhaltung von Instanzumspannenden Einschränkungen bei Arbeitsabläufen durch Anwendung auf

Event Logs

Inhaltsangabe

Zitat “Companies spend millions of dollars on firewalls, encryption and secure access devices, and it’s money wasted, because none of these measures address the weakest link in the security chain.”
– *Kevin Mitnick*

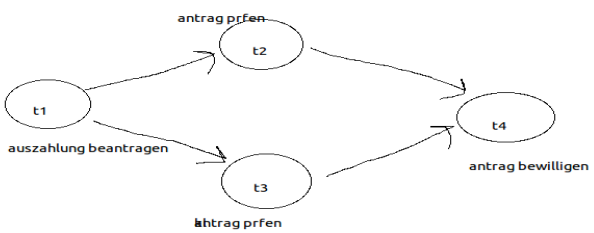
Abstact in deutsch und englisch???

Einleitung

(Motivation, Beispiel, Konventionen in Arbeit, Gliederung der Arbeit)

Um internet Betrug in Unternehmen zu verhindern, wird bei vielen Authorisierungsmodellen das Separation of duty Prinzip beachtet. Dieses besagt, dass es bestimmte Aufgaben gibt, die nicht von ein und der selben Person eledigt werden können. Dazu gibt es verschiedene Ansätze, die ermöglichen, Einschränkungen auf die Authorisierung zu definieren, die je nach Art bereits in der Entwurfsphase des Workflows oder dynamisch während der Laufzeit erzwunden wird, indem sich die informationen dynamische aktualisieren (Hier ein paar Quellen angeben). Allerdings erlauben die aktuellen Modelle nur Einschränkungen innerhalb eines Workflows, was aber nicht ausreichend ist.

Betrachten wir folgendes Beispiel:



Um eine Zahlung zu tätigen, muss die Zahlung zuerst beantragt werden. Dieser Antrag muss zweimal geprüft werden. Sollte die Prüfung positiv verlaufen, wird der Antrag bewilligt und die Zahlung wird getätigt.

Es ist allerdings unerwünscht, dass t3 und t4 von der selben Person ausgeführt wird, die die Zahlung beantragt. Ferner sollten auch keine Verwandten dieser Person daran arbeiten, da die Bewilligung aus reiner Gefälligkeit erfolgen könnte.

Diese Einschränkungen, die sich jeweils auf einen Arbeitsablauf beziehen, sind aber nicht ausreichend, um Betrug zu verhindern.
Es könnte sich eine Gruppe bilden, die sich in mehreren Instanzen des Arbeitsablaufs gegenseitig die Auszahlungen bewilligen.
Eine zusätzliche Einschränkung wäre somit, dass eine Gruppe nur 5mal an t1 und t2 zusammen arbeiten darf.

In dieser Arbeit wird eine Grammatik entwickelt, die es erlaubt, solche Einschränkungen zu definieren. Zusätzlich wird ein Programm vorgestellt, das MXML Logs einliest und sie auf die definierten Einschränkungen hin überprüft.

Folgende Konventionen werden hier eingehalten: *kursive Begriffe* bezeichnen Fachbegriffe, Blockschrift kennzeichnet Pseudocode,...

In Kapitel 2 werden wichtige Begriffe erläutert. In Kapitel 3 widmen wir uns der Herleitung von Einschränkungen und der Definition einer antsprechenden Grammatik. Diese wird in Kapitel 4 in ein Programm integriert, welches Event Logs auf die Verletzung von Einschränkungen prüft. Dazu wird der Algorithmus vorgestellt. Kapitel 5 und 6 schließen mit einer Untersuchung der Korrektheit und einem Ausblick in weitere Forschung.

Wichtige Begriffe

In diesem Kapitel werden wichtige begriffe vorgestellt, die im Verlauf der weiteren Arbeit von Bedeutung sind.

Workflow

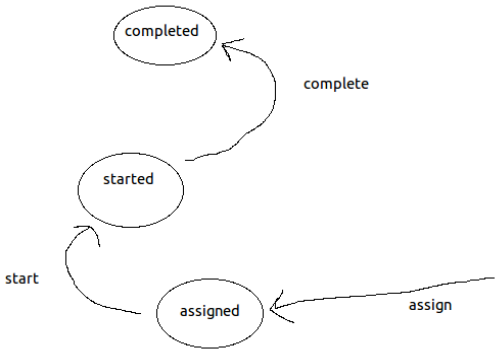
- Bild zu Workflow UML

Ein Workflow ist eine definierte Abfolge von abgeschlossenen Arbeitsaufgaben, den *Tasks.*

TODO: suche nach einer offiziellen Definition von workflows

Tasks

Tasks sind Aufgaben, die in sich abgeschlossen sind. Sie besitzen 3 verschiedene Zustände: Assigned, Started und Completed welche durch Events getriggert werden. Das MXML Modell unterstützt 13 verschiedene Events, die sich in ihrer Bennenung bei den einzelnen Programmen unterscheiden können. Events können auch feiner gegliedert sein, bzw es kann auch sein, dass nur eine Teilmenge davon verwendet wird. In dieser Arbeit gehen wir davon aus, dass es folgende Events gibt: *started*, *assigned*, *completed*, ...



Authorisierung

Rollenmodell

Rollen sind eine Menge von ...

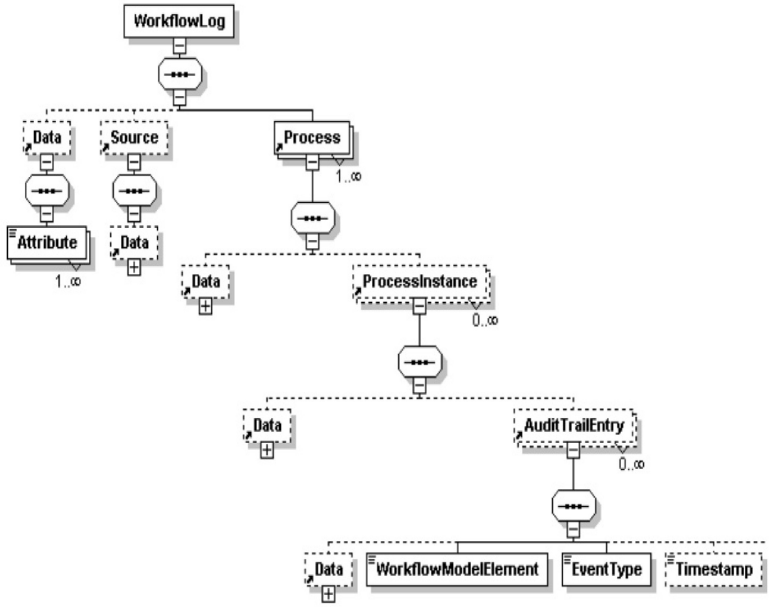
Zeitmodell

`Zeitpunkte,...`

Event Logs

EventLogs sind Abbildungen von Workflows. Ein EventLog kann durchaus unvollständig sein bzw Fehler beinhalten.

Das in dieser Thesis verwendete Format für Event Logs ist der Standard MXML



Beispiel für einen Log:

CaseID	Task	User	Role	Timestamp	EventType
0	Approach check	'Mark'	'Admin'	1999-12-13T12:22:15	start
0	Pay check	'Theo'	'Azubi'	1999-12-13T12:22:16	start
1	Approach check	'Lucy'	'Azubi'	1999-12-13T12:22:17	start
1	Pay check	'Mark'	'Admin'	1999-12-13T12:22:18	abort
0	Revoke check	'Theo'	'Clerk'	1999-12-13T12:22:19	start

Schutzziele ??

Authorisierung

Nur berechtigte Personen dürfen Zugriff auf eine Resource haben bzw eine Aktion ausführen.

Authentifizierung

Integrität

Verwandte Arbeit

Dieser Text sollte aus Sicht des Stoffes sein:

Kurze Geschichte:

Business Rules und Auditierung ??

Um die Zuverlässigkeit von Abläufen zu gewährleisten, wird schon lange Auditierung betrieben. Ursprünglich ging es darum, den korrekten Ablauf nachzuweisen. Man kann damit aber auch die zulässige Ausführung von Ereignissen prüfen.

Inter-Instance Constraints

In dieser Arbeit wird der theoretische Ansatz von … um inter instance constraints erweitert und ist der Ausgangspunkt meiner Arbeit. Allerdings gehen sie davon aus, dass die Constraints für Authorisierungszwecke gedacht sind, und unterscheiden deswegen zwischen statischer Analyse zur Entwurfszeit und dynamischer Analyse während des Ablaufs. Bei der dynamischen Analyse werden die Informationen nach jeder Aktion aktualisiert und unter Umständen die Liste von verbotenen Nutzern / Rollen angepasst.

Da ich Auditierung benutze, wird statically_checked weggelassen

Erklären, was ich anders mache

ProM

SCIFF

Sciff wurde von Marco Alberti, Federico Chesani und Marco Gavanelli im Rahmen des SOCS Projekts entwickelt. Es arbeitet ebenfalls mit Prolog, kann aber keine Inter Instance Constraints.

Theorie

Ein praktisches Beispiel mit vielen Einschränkungen

- Bild zum Workflow + Erklärung
- schriftliche Auflistung der Einschränkungen
  - Intra-Instance
  - Inter-Instance
    - User U darf Task1 nicht mehr als 5 mal im Monat ausführen
    - User U darf bei Task1 nicht mehr als 100000 Euro auszahlen
    - U1 und U2 dürfen nicht mehr als 3mal an T1 und T2 zusammen arbeiten
  - Inter-Process
    - Ein User darf nicht mehr als 100 Tasks pro Tag erledigen

Arten von Constraints

- zeitliche Beschränkungen
  - absolute Einschränkungen
  - relative Einschränkungen
- Akkumulationen
- Separation of Duty
- Binding of Duty Constraints
- Cardinality Constraints
- Workflow Soundness
  - Bild dazu

Vorüberlegungen

Woran erkennt man den Kontext der Constraints?

Welches Rollenmodell wird benutzt, und welche Auswirkungen hat es auf die Grammatik?

Formale Definition

Zuerst Herleitung, was wir genau an Aussagen brauchen

- Workflows
- Zeitstempel
- Constanten
- Variablen
- Literale
- Regeln (Constraints sind Schlussfolgerungen, die sich aus Vorbedingungen ergeben.
  - Fakten
  - Status (aus Event Logs gelesen)
  - Ableitung,...

- Zu Beispiel gefundene Einschränkungen definieren
-
- Kann ein Task mehrere Instanzen haben?
- Hierarchisches Rollenmodell
- Umgang mit Negation

Grammatik

- Konstanten sind als 'String' , Variablen ohne
  - Welche Zeichen darf man wo verwenden?
- Verfügbare Literale
- Syntax
  - Fakten: SET extern|workflow
  - Regeln:
```
IF (..|..) (AND (..|..)*)
THEN (..|..)
WHERE t.name = t2.name AND  ...
```

Erklärung

- Wenn sich etwas auf jede Instanz einzeln beziehen muss, muss Task1.workflow.instance = Task2.workflow.instance
- Eignet sich für hierarchisches und auch für normales Rollenmodell. Hängt nur davon ab, ob die Hierarchie als Fakt gesetzt wurde.
- User und ihre Rollenzuweisungen müssen nicht explizit angegeben werden. Nur wenn man das für die Vergleiche braucht.
- Beziehung zwischen critical_task_pair und collaborateur.

Konkretes Beispiel

Die zuvor gefundenen Beispiele wären mit der neuen Grammtik:

(C1): IF NUMBER OF USER executed TASK IS N AND N > 5 THEN USER cannot execute TASK

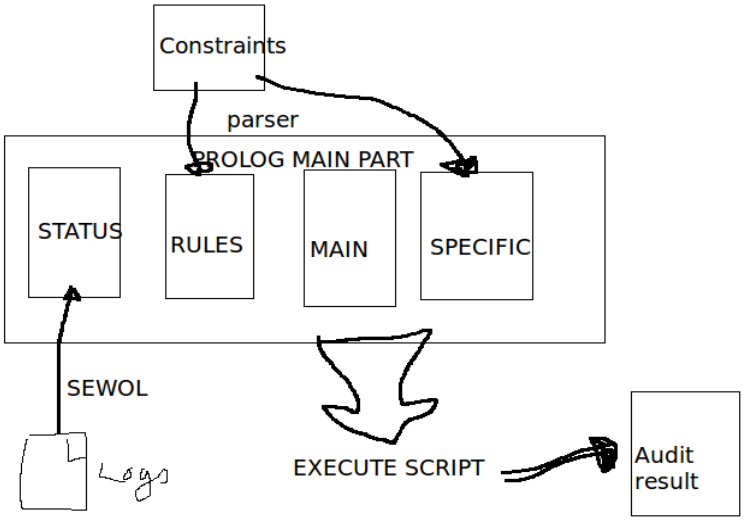
Praxis

Algorithmus

- Zuerst aus Logs Status-Prädikate auslesen.
- Für jedes critical_task_pair entsprechende collaborateurs setzen.
- Schleife über Regeln
  - Die resultierenden Heads bestimmen und schauen, ob es für cannot_do ein executed und für must_do kein executed gibt-> dann wurde es verletzt. Um das herauszufinden, wird die Regel in executed → blabla, oder not(executed) → blabla übersetzt.

- Einfaches Beispiel: Rule: executed(ui, t1)-> cannot_do(ui,t2) und status: executed('tom', t1). Daraus wird abgeleitet: cannot_do('tom', t2). Da es nicht in der DB ist, ist alles OK

Aufbau mein Programm



Das Programm liest mittels SEWOL MXML logs ein und übersetzt sie in Status Prädikate.
Die Constraints werden ebenfalls eingelesen und in Regelprädikate übersetzt. Der Parser für die Grammatik wurde mit ANTLR erzeugt.
Die eigentliche Untersuchung kann entweder durch ein Skript oder innerhalb des Javacodes gestartet werden.

Auswertung

Mein Beispiel mit ein paar Beispiel Logs untersuchen

Erklärung, woher diese Logs kommen und wie sie erzeugt wurden.

- Zeigen, wie der Output aussieht.
- Beweismethoden??
- Wann gibt es false Negatives, false Positives?

Aussicht

- Wie kann man die restlichen Constraints implementieren
- Compliance Checking:
  - Cannot_do und Must_execute vergleichen.
- Optimierung??

Quellenangaben

Anhang

kleine Doku Grammatik??

zusätzliche Beispiele zu Constraints und deren Darstellung

Intra Instance

Task1 und Task2 dürfen nicht vom selben User ausgeführt werden
→ IF USER executed 'Task1' THEN USER cannot execute 'Task2'

Inter Instance

Inter Process

Wie startet man das Programm

Voraussetzung sind Java 7
SWI-Prolog 6

Per Shell Skript: ./run.sh

Kommandozeilen optionen

- Input Dateien für die Constraints
- Input Dateien MXML

Fehlermeldungen, wenn man was falsches eingibt