Albert Ludwigs University - Freiburg

Bachelor Thesis

Inter-Instance Constraints

Author:
Regina König

Supervisor:
Adrian Lange

A thesis submitted in fulfilment of the requirements
for the degree of Bachelor of Science

in the

Research Group Name
Department or School Name

June 2015

Declaration of Authorship

I, Regina KÖNIG, declare that this thesis titled, 'Inter-Instance Constraints' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"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

ALBERT-LUDWIGS-UNIVERSITÄT FREIBURG

Abstract

Faculty Name
Department or School Name

Bachelor of Science

Inter-Instance Constraints

by Regina KÖNIG

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Contents

List of Figures

List of Tables

Chapter 1

Einleitung

1. Satz:

1.1 Motivation

TODO: Motivation, Beispiel, Konventionen in Arbeit, Gliederung der Arbeit
Wo werden alles EventLogs benutzt? Erklären, warum es wichtig wäre, genau dort Constraints zu benutzen? Außer dem Verhindern von Betrug ist auch das Vermeiden von Fehlern wichtig. -
Um internen 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 un der selben Person erledigt werden können. Dazu gibt es verschiedene Ansätze, die ermögliche, Einschränkungen auf die Authorisierung zu definieren, die je nach Art bereits in der Entwurfphase des Workflows oder dynamisch während der Laufzeit erzwungen wird, indem sich die Informationen dynamisch aktualisieren (Hier ein paas Quellen angeben). Allerdings erlauben die aktuellen Modelle nur Einschränkungen innerhalb eines Workflows, was aber nicht ausreicheng 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.

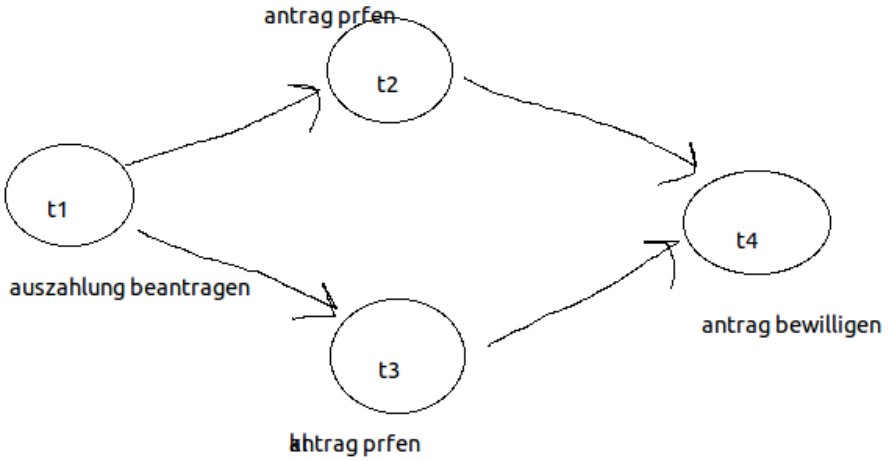

FIGURE 1.1: Beispiel Workflow

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.

1.2 Ziel der Arbeit

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.

1.3 Aufbau der Arbeit

Folgende Konventionen werden hier eingehalten: kursive Begriffe bezeichnen Fachbegriffe, Blockschrift kennzeichnet Pseudocode,... Begriffe, die zum ersten mal definiert werden, werden beim ersten Vorkommen **fett** geschrieben.

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.

Chapter 2

Theoretische Grundlagen

2.1 Erklärung der Begriffe

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

2.1.1 Workflow und Cases

Bild zu Workflow UML Ein Workflow Schema W = $\{t_1, t_2, ..., t_n\}$ mit n $\in \mathbb{N}$ ist eine Menge von Tasks. Ein Workflow Case ist eine konkrete Instanz eines Workflow Schemas...

TODO: suche nach einer offiziellen Definition von workflows, Beachte, dass es für mich keine Rolle spielt, ob es ein bereits definierter Workflow ist. Also nicht auf Workflow Models eingehen. Und vielleicht auch nicht so sehr den Begriff Workflow benutzen

2.1.2 Tasks und Activities

Ein Task ist ein atomares Event im KOntext eines Workflows. Tasks sind Aufgaben, die in sich abgeschlossen sind. Sie besitzen 3 verschiedene Zustände: Assigned, Started und Completed welche durch Activities/Events in den nächsten Zustand übergeleitet werden. Das MXML Modell unterstützt 13 verschiedene Events, die sich in ihrer Benennung 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, ... Eine Activity ist eine Aktion, die zu einem eindeutigen Task gehört und einen Eventtyp besitzt.

Tasks haben auch eine Ordnung $< T, \leq >$. Es gilt nämlich $t_1 < t_2$, wenn t_1 vor t_2 ausgeführt wurde.

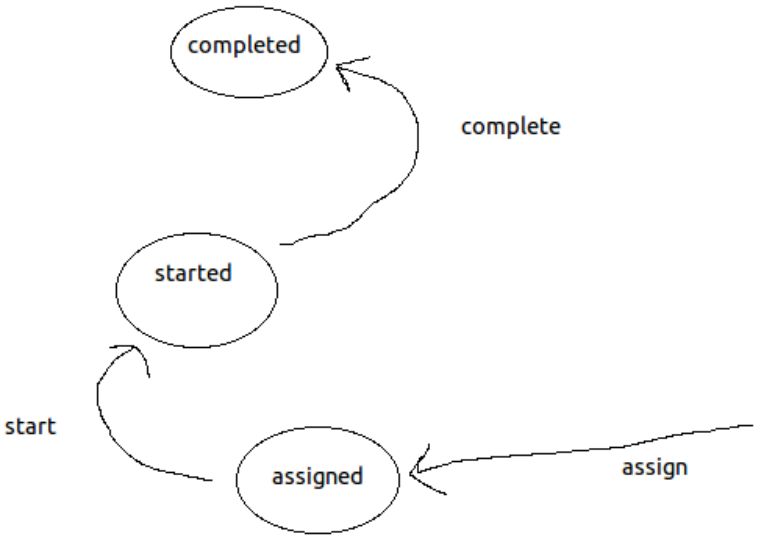

FIGURE 2.1: Tasks und Events

2.1.3 Rollenmodell und Authorisierung

Sei T = $\{t_1, t_2, ...t_m\}$, $m \in \mathbb{N}$ eine Menge von Tasks, R = $\{r_1, r_2, ...t_n\}$, $n \in \mathbb{N}$ eine Menge von Rollen, und U = $\{u_1, u_2, ...u_l\}$,$l \in \mathbb{N}$ eine Menge von Usern.

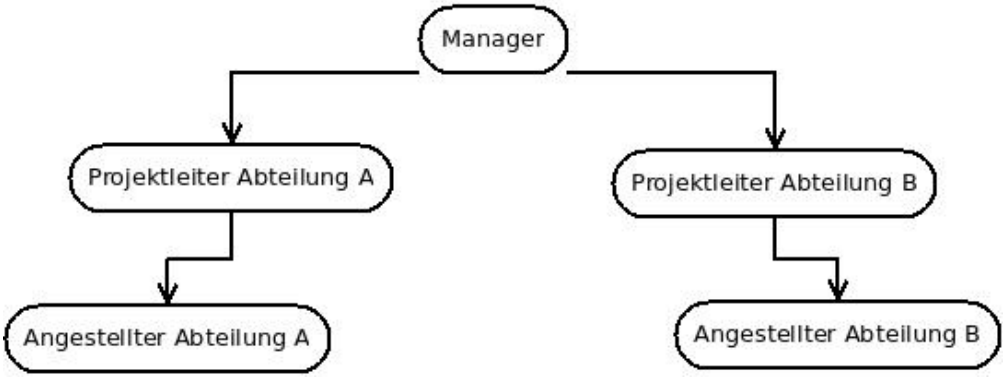

FIGURE 2.2: Beispiel Rollenmodell

Eine **Authorisierung** ist eine Menge von potentiellen Usern und Rollen, denen es erlaubt ist, einen Task auszuführen. Eine Authorisierung besteht aus den Tupeln

$$\mathbf{TR} = (T \times R)$$

und

$$\mathbf{UR} = (U \times R)$$

, welche eine n:m-Beziehung zwischen Tasks und Rollen, bzw zwischen Usern und Rollen kennzeichnen. Das bedeutet, dass User mit Rollen in der User-Rollen Beziehung assoziiert werden und Tasks mit Rollen in der Tast-Rollen Beziehung assoziiert werden.
Sei nun

$$\mathbf{R(t)} = \{r_m \in R : \exists(t_k, r_m) \in TR(t)\}$$

$$\mathbf{U(t)} = \{u_n \in U : \exists(u_n, r_m) \in UR, r_m \in R(t)\}$$

Mit anderen Worten ist R(t) die Menge aller Rollen, die authorisiert sind, einen Task auszuführen und U(t) die Menge alles User, die authorisiert sind, einem Task zugeteilt zu werden.
Eine **Zuweisung** ist die konkrete Ausführung eines Tasks durch einen User.

Ein **hierarchisches Rollenmodell** ist eine geordnete Menge von Beziehungen zwischen Rollen $< R, \leq >$. Wenn $r_1, r_2 \in R$ und $r_1 < r_2$, dann dominiert die Rolle r_2 die Rolle r_1 in Bezug auf die organisatorische Rollenhierarchie. In Abb. 2.2 dominiert die Rolle "Projektleiter" die Rolle "Angestellter", das bedeutet, dass der "Projektleiter" alle Tasks ausführen darf, die der Rolle "Angestellter" zugeordnet wurde. Die Rolle und all ihre Elternrollen bis zur Wurzel können einem Task zugewiesen werden.

[1]

2.1.4 Zeitmodell

Das Zeitmodell ist ein Tupel T = $(\mathcal{T}; \leq)$.
$\mathcal{T}$ ist eine Menge von *Zeitpunkten* τ und $\leq$ eine total Ordnung auf $\mathcal{T}$. Ein *Zeitinterval* $[\tau_a, \tau_b]$ ist eine Menge von Zeitpunkten $\tau \in \mathcal{T}$ mit $\tau_a \leq \tau \leq \tau_b$. Ein Zeitinterval $[\tau_a, \tau_b]$ wird als leer bezeichnet, falss $\tau_b \leq \tau_a$.

TODO darauf achten, dass hier Konsistenz herrscht mit ts, tp, ... in der späteren Grammatikdefinition. [2]

2.1.5 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

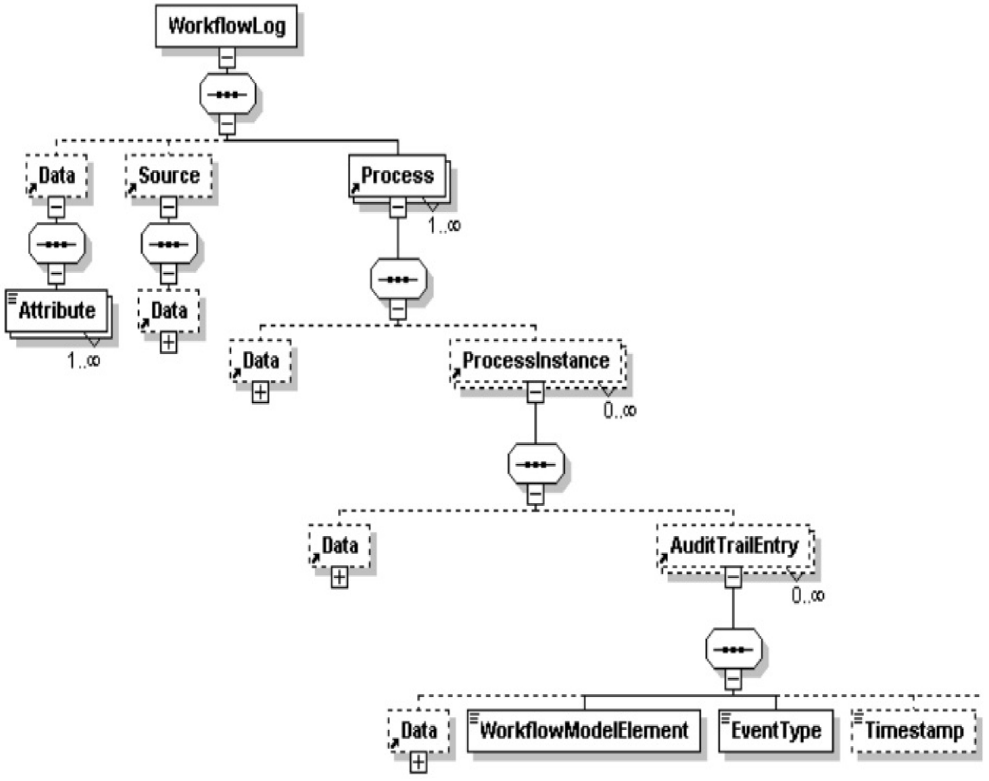

FIGURE 2.3: MXML Modell

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

Das ist nur ein Auszug und kein vollständiger Log. Es können auch weitere Daten vorhanden sein, die hier nicht dargestellt werden.

TABLE 2.1: Beispiel Log Einträge.

2.1.6 Definition Constraints

Constraints sind Regeln bzw Einschränkungen, die aus dem Verlauf von vorhergehenden Tasks resultieren.

2.1.7 Schutzziele

Die von Accorsi nehmen. Später darauf eingehen, was eingehalten wurde. Bzw schon bei Definition von Constraints darauf eingehen.

Chapter 3

Verwandte Arbeit

3.1 Related work

3.1.1 Geschichte

Chinese Wall Modell Bewer/ Nash
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.

3.1.2 Heute

Übergang zu Inter/instance Constraints Heute: verschiedene Forschungszweige: TBAC, Inter-Instance Constraints Allgemein eher den Inhalt auswerten, statt eine Liste von verwandten Arbeiten anzugeben Arten Unterscheiden, wie man Constraints spezifizieren kann Als logische Prädikate, GL4 Welche Constraint-Richtungen werden behandelt? ZB nur Entailment, Cardinality.

3.1.3 warner inter-Instance

Diese Arbeit bezieht sich hauptsächlich auf die Arbeit von Warner und Atluri [2]. In dieser Arbeit wird der theoretische Ansatz von Bertino et al. 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 und weitere Regeln, die zur Laufzeit nötig sind.
Das Paper ist allerdings nur ein theoretischer Ansatz und behandelt zB die verschiedenen Eventtypen nicht ausreichen. Des weiteren wird nicht darauf eingegangen, was passiert, wenn manche Informationen fehlen.

3.1.4 leitner instance-spanning

Hat ein Framework entwickelt.

3.1.5 tan consistency

Es geht eher um die Prüfung auf Konsistenz, es werden aber globale Constraints erwähnt.

3.1.6 ProM - SCIFFchecker

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.

Chapter 4

Material und Methoden

4.1 Theorie

4.1.1 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

4.1.2 Arten von Constraints - Herleitung

4.1.2.1 sich gegenseitig ausschließende Tasks (Conflicting Tasks)

TODO: kommt das zur Constraint Sammlung? In manchen Fällen kann man Tasks nicht verschiedenen Rollen zuweisen ohne die exisitierenden Rollen derart zu segmentieren, dass die schwer zu verwalten sind in Bezug auf das organsiatorische Modell. Außerdem können Rollenhierarchien dazu verwendet werden, in zwei verschiedenen Rollen zu agieren, die eigentlich getrennt waren. ZB könnte ein Manager als ein Clerk und gleichzeitig als sein eigenener Supervisor handeln.

Deswegen definieren wir **TC** $\subset$ T als eine Menge von **kollidierenden Tasks**. TC beinhaltet Tasks, deren Allokation von der Allokation von vorhergehend ausgeführten Tasks aus TC abhängt. Diese Abhängigkeit wird als Abhängigkeit zwischen Tasks aus TC beschrieben. $t_c \in TC$ gilt als entailed Task von $t_m \in TC$, wenn die Allokation von t_n von der Allokation von t_m eingeschränkt wird, mit $t_m < t_n$. [1]

4.1.2.2 Entailment Constraint

c=(TC, n_u,m_{th}) mit n_u als minimale Anzahl an verschiedenen Usern, die einem Task zugewiesen werden müssen. Wenn t_{ki} eine Instanz des Tasks t_k ist, dann ist m_th der Grenzwert von der Summe der Task Instanzen, denen ein User zugewiesen sein darf. [1]

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

4.1.3 Vorüberlegungen zur Grammatik

Woran erkennt man den Kontext der Constraints?
Welches Rollenmodell wird benutzt, und welche Auswirkungen hat es auf die Grammatik?
Was passiert bei fehlenden Einträgen, oder falschen Einträgen?
Soll Negation erlaubt sein?
Was definiert der Timestamp, wenn es sich doch nur auf verschiedene Events bezieht?

4.1.4 Basis von Werten, Aussagen...

Zuerst Herleitung, was wir genau an Aussagen brauchen

Prädikat	Aurgumente	Beschreibung

TABLE 4.1: sdf

4.1.4.1 Parameter - Variablen und Konstanten

User

Rollen

zeitliche Parameter

4.1.4.2 Prädikate

Prädikate sind Aussagen über bestimmte Zustände. Es gibt verschiedene Typen.

Statusprädikate sind Aussagen über Events

4.1.4.3 Regeln

Regeln (Constraints sind Schlussfolgerungen, die sich aus Vorbedingungen ergeben. Es gibt positive und negative. Die positiven sagen, dass etwas passieren muss, die negativen verbieten, dass etwas passiert)
Ableitung,...

4.1.5 Grammatik in BNF

Für ein besseres Verständnis der Definition erst einmal ein kurzes Beispiel:

```
SET 'Mark Maier' is related to 'Max Mueller'
IF USER_A executed 'Kredit beantragen' AND USER_A is related to USER_B
  THEN USER_B cannot execute 'Kredit prüfen'
```

FIGURE 4.1: Beispiel Spezifikation einer einfachen Regel

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 ...

4.1.6 Erklärungen zur Grammatik

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.

4.1.7 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
```

FIGURE 4.2: Regeln für unsere gefundenen Beispiele

Chapter 5

Praxis

5.1 Implementierung

5.1.1 Aufbau mein Programm

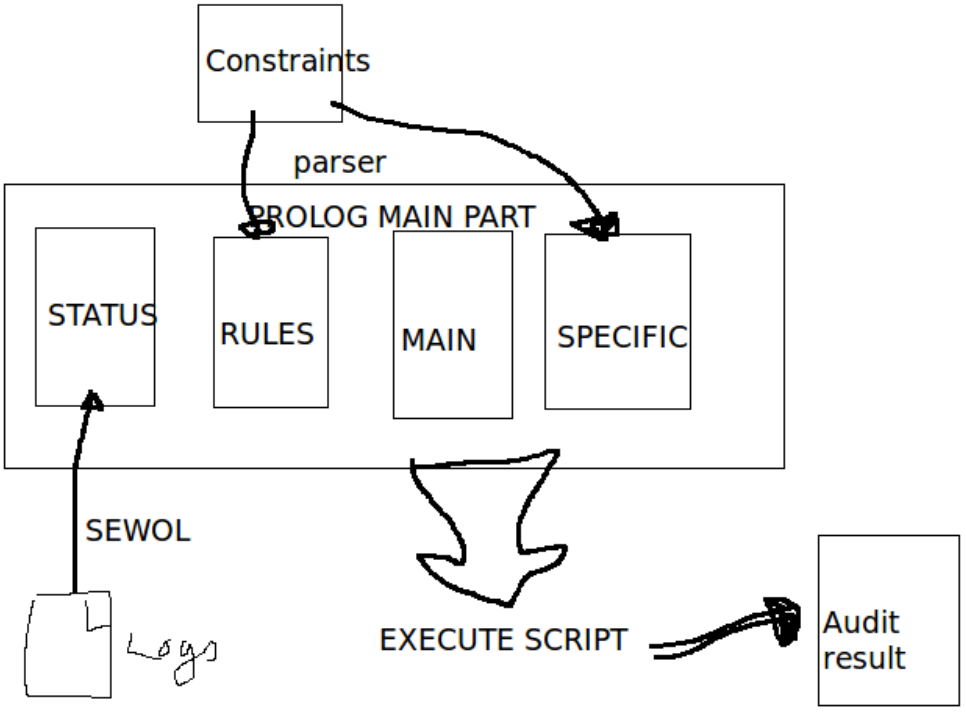

FIGURE 5.1: Aufbau mein Programm

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[3] erzeugt. Die eigentliche Untersuchung kann entweder durch ein Skript oder innerhalb des Javacodes gestartet werden.

5.1.2 Wissensbasis

Die Wissensbasis wird aus den Logs herausgelesen und in entsprechende Prädikate übersetzt. Auf Basis dieser Prädikate arbeitet der Modelchecker. Die Wissensbasis besteht aus genau 7 möglichen Prädikaten, welche ausreichen, um die Events vollständig wiederzugeben:

1. **workflow_name(caseID, workflowName)**: Weist einer Case ID einen eindeutigen Namen hinzu, welcher der Workflow Spezifikation entspricht. Die Case ID kennzeichnet die Instanz, konkrete Ausführung eines Workflows. Dieses Prädikat dient jedoch nur zur Vollständigkeit. Sollte es für den Arbeitsablauf keinen eindeutigen Namen geben, wird dieses Prädikat nicht gesetzt. Es kann im Modelchecker dann auch nicht darauf zugegriffen werden.
2. **activity_workflow(activityID, caseID)**: Setzt fest, zu welcher case ID die Activity gehört. TaskID wird intern vom Transformer gesetzt.
3. **activity_name(taskID, taskName)**: Legt den Namen der Activity fest. Dieser ist im MXML Modell als WorkflowModelElement zu finden.
4. **timestamp(taskID, timestamp in ms)**: Zeitpunkt der Aktivität in ms nach 1970. Sollte kein Zeitpunkt eingetragen sein, wird entweder kein Fakt gesetzt, oder es wird die Reihenfolge der Einträge genommen. Dazu wird der letzte Wert + 1 ms gesetzt.
5. **eventtype(taskID, eventtype)**: Der Eventtyp der Aktivität. Im Grundmodell sind es start, accept, abort, ... Kann jedoch varriieren.
6. **executed_user(user, taskID)**: Der Nutzer, der die Activity ausgeführt hat.
7. **executed_group(group, taskID)**: Die Rolle, in der die Activity ausgeführt wurde. Man beachte, dass ein Nutzer mehrere Rollen haben kann.
8. **task_attribute(taskID, attrName, attrType, attrValue)**: In dieser Wissensbasis wird zwischen zwei Typen unterschieden. String-Attributen, welche nur Vergleichsoperatoren kennen. Hat das Attribut im ürsprünglichen Event ein anderes Format, muss es für die Analyse in eine passende String Darstellung konvertiert werden.

Die erste Zeile aus dem Eventlog aus Tabelle 2.1 wird in folgende Wissensbasis konvertiert:

TODO: Closed World, was passiert mit nicht angegebenen Zeiten,.

activity_workflow(0,0)	activity_name(0,'Approach check')	timestamp(0, 123)
eventtype(0, start)	executed_user(0, 'Mark')	executed_role(0, 'Admin')

Table 5.1: fd

5.1.3 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

Chapter 6

Ergebnisse und Diskussion

6.1 Auswertung und Aussicht

6.1.1 Mein Beispiel mit ein paar 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?
Gibt es Fehler? Wieviele?

6.1.2 Aussicht

Wie kann man die restlichen Constraints implementieren
Compliance Checking:
Cannot do und Must execute vergleichen.
Optimierung??

Appendix A

Weitere Beispiele für die Benutzung der 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

Appendix B

Argumente und Konfiguration

Input Dateien für die Constraints: Wo sind sie? Wie kann man mehrere einlesen? (Dadurch sollen die Regeln wiederverwendbar werden)
Kann man vielleicht eine textdatei anlegen, wo die Inputfiles drinstehen? Input Dateien MXML - wo sind sie? Wird der ganze Ordner eingelesen oder gibt man ein bestimmtes File an?
Output Logger - Konsole oder File. Einstellung Level
Unterscheiden zwischen Parser Logger und Visitor Logger?
Konfigurationsdatei für die vorkommenden Events
Output Ordner

Parameter	Argument	Default	Beschreibung
–constraintfolder	abs oder rel Pfad zum Ordner	rulefiles	..
–logfolder	abs oder rel Pfad zum Ordner	logfiles	..
–outputfolder	abs oder rel Pfad zum Ordner	prologfiles	..

Tabelle mit allen möglichen Parametern

Table B.1: Kommandozeilenparameter

Appendix C

User Manual

Voraussetzung sind Java 7
SWI-Prolog 6

Per Shell Skript: ./run.sh

Appendix D

Fehlermeldungen

Write your Appendix content here.

Bibliography

[1] C Wolter and A Schaad. Modeling of task-based authorization constraints in bpmn. In *of Lecture Notes in Computer Science*, pages 64–79. Springer, 2007.

[2] Janice Warner and Vijayalakshmi Atluri. Inter-instance Authorization Constraints for Secure Workflow Management. In *Proceedings of the Eleventh ACM Symposium on Access Control Models and Technologies*, SACMAT '06, pages 190–199, New York, NY, USA, 2006. ACM. ISBN 1-59593-353-0.

[3] Terence Parr. *The Definitve ANTLR Reference - Building Domain-Specific Languages*. The Pragmatic Programmers, 2013.