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

30. Juli 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...

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Symbols

W	Prozessschema
W_i	Prozessschema i
W_i^k	k-te Instanz des Prozessschema i
$T = \{t_1, t_2, ..., t_n\}, n \in \mathbb{N}$	Menge von Aktivitäten
D	Menge der Abhängigkeiten der Aktivitäten
t_{ij}	Aktivität j aus Prozessschema i
t_{ij}^k	Aktivität j aus der k-ten Instanz des Prozessschema i
R	Menge der Rollen
U	Menge der Nutzer
$\mathcal{T}$	Menge aller Zeitpunkte

Kapitel 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:

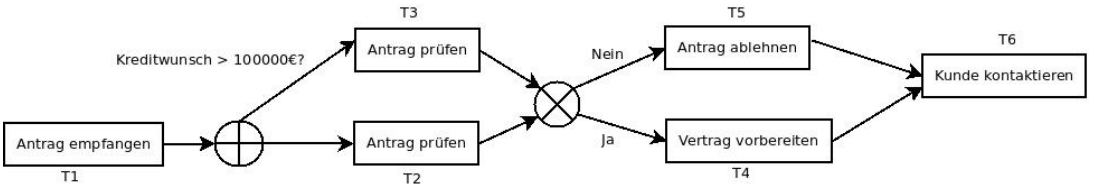

Abbildung 1.1: Workflow

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.

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.

Kapitel 2

Theorie

2.1 Grundlagen und Definitionen

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

2.1.1 Prozess Schemata und Instanzen

Ein Prozessschema $\mathbf{W} = \{T, D\}$ mit n $\in \mathbb{N}$ ist eine Menge von Aktivitäten $\mathbf{T} = \{t_1, t_2, ..., t_n\}$ und einer Menge **D** von Abhängigkeiten, welche bestimmen, in welcher Reihenfolge die einzelnen Aktivitäten ausgeführt werden bzw von welchen Parametern abhängt, ob sie ausgeführt werden müssen, oder nicht. Die Menge der Aktivitäten muss mindestens eine Startaktivität haben, kann aber mehrere terminierende Aktivitäten beinhalten, die gleichberechtigt den Prozess abschließen. t_{ij} bezeichnet hierbei die Aktivität t_j aus dem Prozessschema W_i.

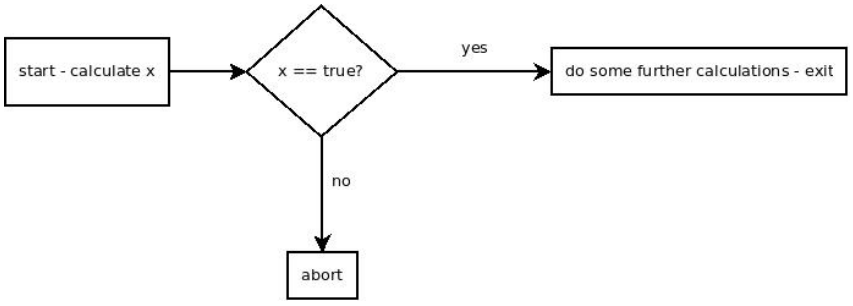

Abbildung 2.1: einfaches Beispiel eines Prozessschemas
Ein einfaches Beispiel für das Schema eines Prozesses mit einem Start und zwei finalen Aktivitäten.In Abhängigkeit davon, welcher Wert für x berechnet wird, wird der Prozess entweder sofort abgebrochen, oder es werden weitere Aktivitäten ausgeführt.

Ein Prozessschema kann mehrere Instanzen besitzen, die mit $\mathbf{W}_i^k$ gekennzeichnet werden. Eine Prozessinstanz W_i^k ist eine Menge von Instanzen T_i^k der zugehörigen Aktivitäten. Dabei ist

$|T_i| \subseteq |T|$ eine Teilmenge aller möglichen Aktivitäten des zugehörigen Schemas, da einzelne Aktivitäten unter Umständen aufgrund der Abhängigkeiten nicht ausgeführt werden müssen.

2.1.2 Aktivitäten

Eine Aktivität ist ein atomares Event, dh. eine in sich abgeschlossene Aufgabe im Kontext eines Prozesses. Sie haben eine definierte Menge von potentiellen Rollen und Nutzern, die die Erlaubnis besitzen, diese Aufgabe auszuführen. Sobald sie einem Nutzer zugewiesen wurde, kann niemand anderes mehr die Aufgabe annehmen. Das bedeutet, dass jede Aktivität einen eindeutigen Nutzer und eine eindeutige Rolle hat, welcher sie ausgeführt hat.

Des weiteren besitzen die Aktivitäten einen eindeutigen Zeitstempel τ, zu dessen Zeitpunkt sie ausgeführt wurden. Diese Zeitstempel bestimmen eine Ordnung $< T, \leq >$. Es gilt nämlich $t_1 < t_2$, wenn t_1 vor t_2 ausgeführt wurde, dh. timestamp$(t_1) <$ timestamp(t_2).

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 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, ...

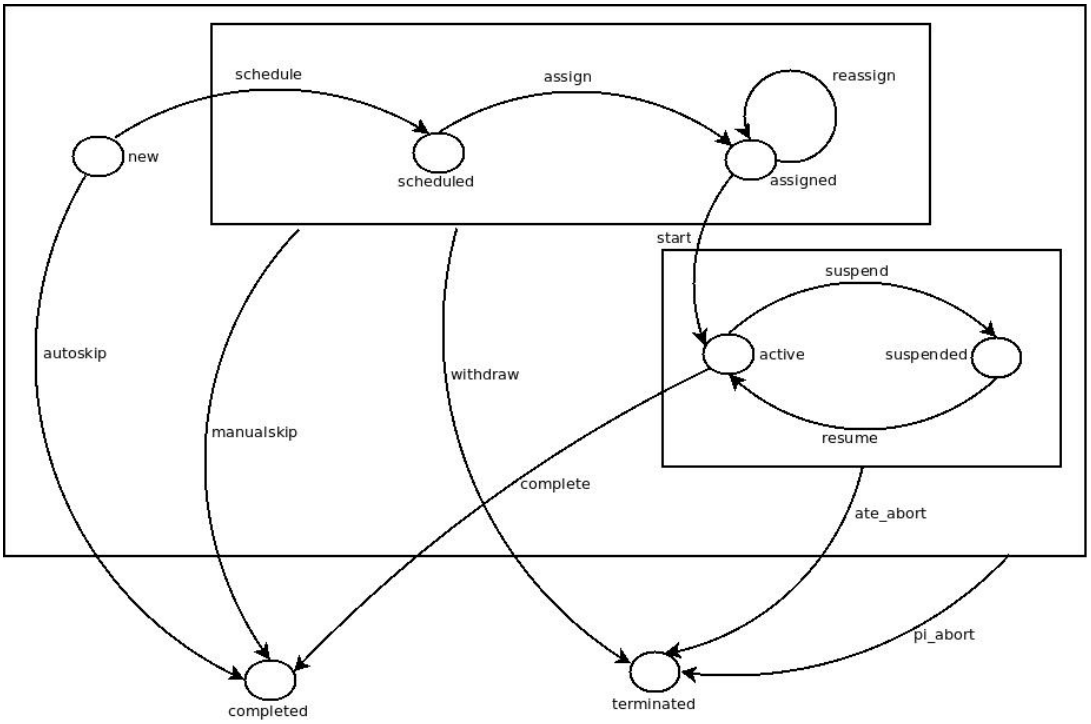

Abbildung 2.2: Tasks und Events

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.

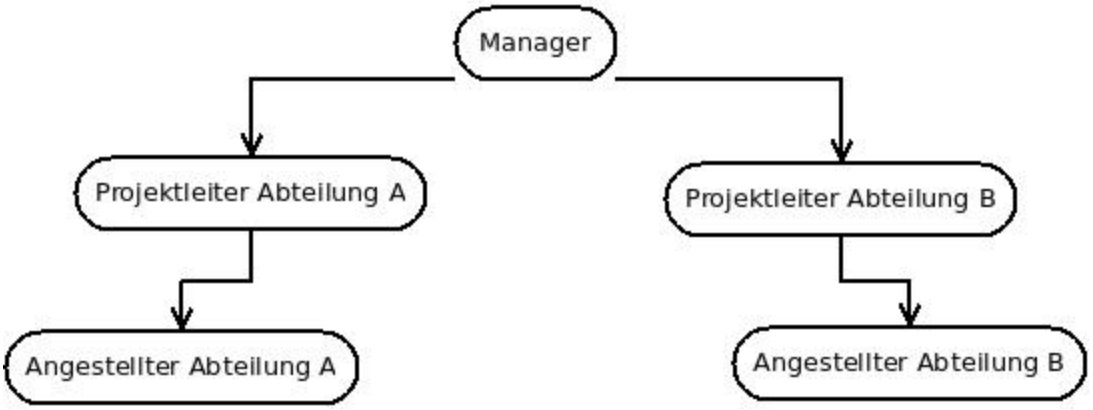

Abbildung 2.3: Beispiel Rollenmodell

Eine **Authorisierung** ist eine Menge von potentiellen Nutzer 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. 2.3 dominiert die Rolle Projektleiter"die Rolle Ängestellter", das bedeutet, dass der Projektleiterälle Tasks ausführen darf, die der Rolle Ängestellterßugeordnet wurde. Die Rolle und all ihre Elternrollen bis zur Wurzel können einem Task zugewiesen werden.

[3]

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. [1]

2.1.5 Event Logs

EventLogs sind Abbildungen von Workflows. Ein EventLog kann durchaus unvollständig sein bzw Fehler beinhalten. Jedoch wird in dieser Arbeit von einer Closed World"davon ausgegangen.

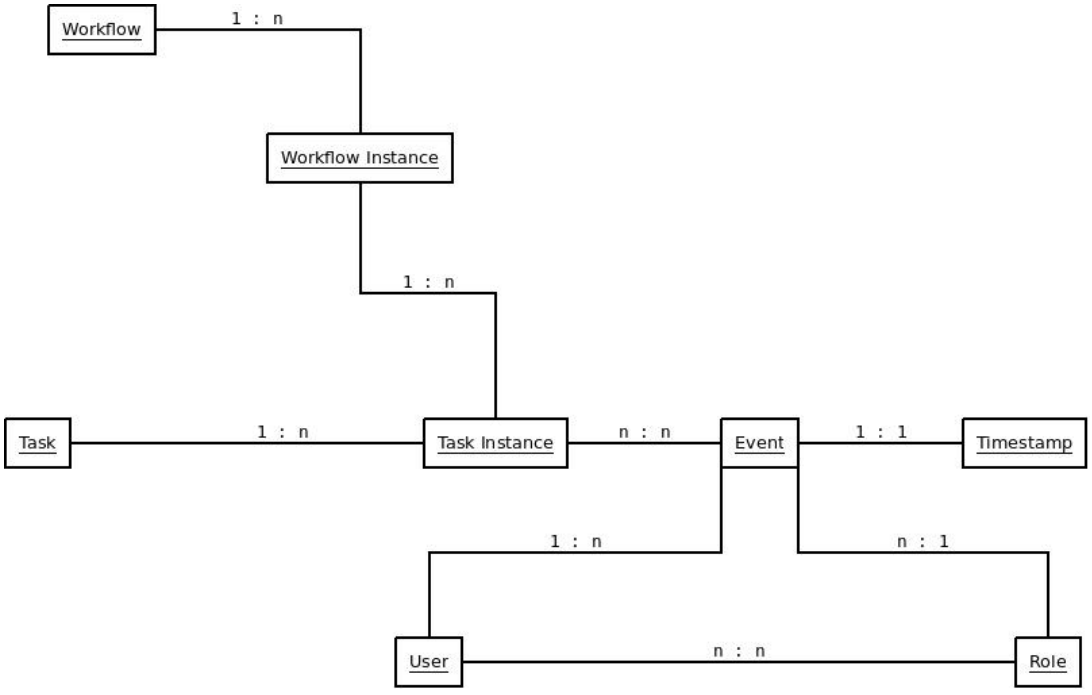

Abbildung 2.4: Ontologie eines Workflows

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.

Tabelle 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

Autorisierung
Welche Subjekte bzw Rollen dürfen auf welche Ressourcen zugreifen? Da es in größeren Unternehmen komplex werden kann, wurde das Rollenmodell zur Vereinfachung entwickelt
Nutzungskontrolle
Regelt die Art der Nutzung. RWX, begrenzte Anzahl an Zugriffen, bzw Verpflichutung einer Löschoperation (weiteren, resultierenden Handlungen) nach Zugriff(Kontrollfluss)
Interessenskonflikt
Unterbindung von unzulässiger Ausnutzung von insider-Wissen. Chinese Wall Modell.
Funktionstrennung
Bestimmte Aufgaben dürfen nicht vom selben Subjekt, Rolle, Abteilung, ausgeführt werden. Unterbindung von kriminellen Handlungen und Betrug.
Aufgabenbindung
Gegenteil von Funktionstrennung
Mehr-Augen-Kontrolle
kritische Aktivität im Prozess darf nicht von einer einzelnen Person ausgeführt werden. Das wäre bei mir Cardinality Constraints, wenn eine Aktivität aus mehreren Tasks besteht.
Isolation
keine Datenlecks innerhalb eines Prozesses bzw zwischen mehreren Prozessen.

[4]

2.2 Herleitung Constraints

2.2.1 Ein praktisches Beispiel mit vielen Einschränkungen

Das folgende Beispiel wurde in leicht veränderter Form [3] entnommen. Blabla Erklärung des Workflows Folgende Einschränkungen und Regeln sind gewünscht:

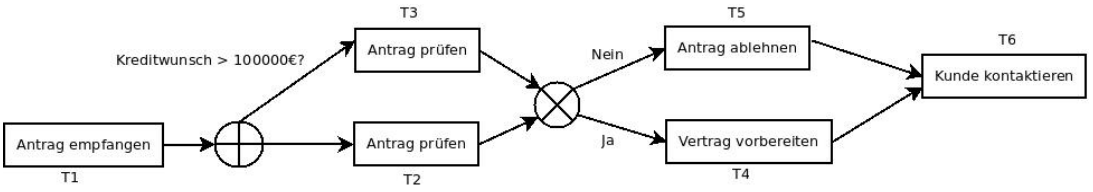

Abbildung 2.5: Workflow

1. (Der Kontakt mit Kunden) T1 und T6 muss vom Kundenberater erledigt werden
2. Um den Kunden nicht zu lange warten zu lassen, sollte der Kunde spätestens 3 Tage nach erstem Kontakt über das Ergebnis informiert werden (timestamp(T6) < timestamp(T1) + 3D)
3. Um zu verhindern, dass Fehler durch Überlastung passieren, darf jeder Mitarbeiter am Tag höchstens 100 Tasks beartbeiten.
4. Den Antrag annehmen (T1) und den Antrag prüfen(T2, T3) sollten von verschiedenen Personen erledigt werden (4 Augen Prinzip).
5. Ferner sollten auch die zwei Prüfungen von verschiedenen Mitarbeitern vollzogen werden. T3 muss durch den Bank Manager erfolgen.
6. Wenn ein Mitarbeiter 5x einem Task zugewiesen wird und ihn dann abbricht, darf er nicht mehr an dem Task arbeiten.
7. Es dürfen keine Anträge von Verwandten geprüft werden.
8. Es dürfen auch höchstens 3 mal Mitarbeiter an den Anträgen des jeweils anderen Verwandten arbeiten.
9. Ein Mitarbeiter darf bei dem selben Kunden höchstens Kredite bis 100000 Euro prüfen.
10. Es dürfen hochstens 3 mal die selben Personen an T2 und T3 arbeiten

2.2.2 Arten von Constraints - Herleitung

Generell kann man verschiedene Arten von Regeln erkennen.

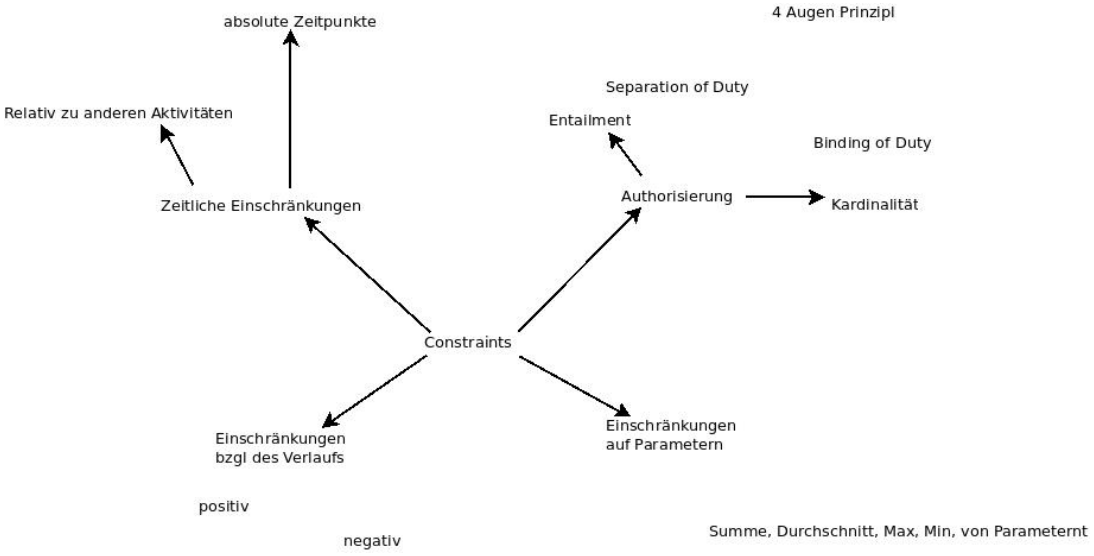

Abbildung 2.6: Typen von Einschränkungen und regeln

2.2.2.1 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 $t_m < t_n$. [3]

2.2.2.2 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. [3]

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

2.2.3 Gültigkeitsbereich von Constraints

Intra-Instanz
Die meisten Ansätze beziehen sich auf Regeln, die in Bezug zu einer Prozessinstanz stehen. Hier gilt, dass die Prozessinstanz für alle Aktivitäten die selbe ist. **Inter Instanz**
Geht man davon aus, dass gleiche Arbeitsabläufe zum selben Prozessschema gehören, dann ist dieser konstant. Die Aktivitäten können jedoch zu verschiedenen Prozessinstanzen gehören.
Inter-Prozess
Die Regeln beziehen sich auf alle Aktivitäten, unabhängig davon, zu welcher Prozessinstanz oder welchem Prozessschema sie gehören.

2.3 Vorüberlegungen zur Grammatik

2.3.1 Was muss sie können?

- Sie sollte Regeln auf Basis der verschiedenen Aktivitäten aufstellen können
- Rollenmodell berücksichtigen, dh Möglichkeit, es anzugeben
- Ziel ist eine möglichst intuitive Notation
- arithmetische Operationen auf Grundbiveau sollten möglich sein, um Bedingungen in Bezug auf Zeiten und andere numerische Werte aufstellen zu können.

2.3.2 Weiterführende Fragen

-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?
-Soll auf der Event Basis oder auf der Task Basis gearbeitet werden?

2.4 Definition der Grammatik

2.4.1 Basis von Werten, Aussagen...

Zuerst Herleitung, was wir genau an Aussagen brauchen

2.4.1.1 Argumenttypen - Variablen und Konstanten

Man kann einem Prädikat entweder eine Variable oder eine Konstante des jeweiligen Typs übergeben. Variablen müssen mit einem Großbuchstaben beginnen und können die Zeichen [A-Z][a-z][0-9] enthalten. Konstanten können aus beliebigen Zeichen bestehen, müssen aber innerhlab zwei einfacher Anführungsstriche stehen. Die Grammatik ist dynamisch typisiert, das heißt der jeweilige Typ wird aus dem Kontext bestimmt.

1. Originator: UT = UV + UC, mit UV Variablen ueber Originator, UC Konstanten
2. Rollen: RT = RV + RC
3. zeitliche Parameter
4. Eventtypen: ET = EV + EC, mit EC $\in \{started, completed, ...\}$

2.4.1.2 Prädikate

Prädikate sind Aussagen über bestimmte Zustände. Es gibt verschiedene Typen. Externe Informationen, Spezifikation, Status, Enforcement und Konditionell.

Externe Informationen (Tabelle 2.2) sind Aussagen, die nicht direkt mit der Workflow Spezifikation zu tun haben aber dennoch relevant für den Ablauf sein könnten. Diese Prädikate müssen explizit in den Regeln gesetzt werden, da es sonst zu keinem Ergebnis führt.

Prädikat	**Beschreibung**
UT is related to UT	Beide User sind verwandt
UT is partner fo UT	Beide Akteure sind Partner
UT is in same group as UT	Beide Akteure sind in der selben Gruppe, Abteilung

Tabelle 2.2: Prädikate für externe Informationen. Das sind nur drei Vorlagen. Dem Programmierer ist selbst überlassen, wie er diese Prädikate interpretieren will.

Spezifikationsprädikate (Tabelle 2.3) bestimmen die Beziehungen und Zugehörigkeit zwischen Nutzern, Rollen und Tasks. Sie Sollten genauso gesetzt werden, wie die Spezifikation war, als der Workflow ausgeführt wurde.

Prädikat	**Beschreibung**
'role' RT 'can execute' TT	RT ist in R(TT)
'user' UT 'can execute' TT	UT ist in U(TT)
'user' UT 'belongs to role' RT	(UT,RT) ist in UR
RT 'is glb of' TT	greatest lower bound. TT muss mindestens mit Rolle RT ausgeführt werden
RT 'is lub' TT	lowest upper bound. TT darf höchstens mit Rolle RT ausgeführt werden
RT 'dominates' RT	Rolle 1 dominiert Rolle 2
'critical_task_pair(' TT ',' TT ')'	Die beiden Tasks sind ein kritisches Paar. Die User werden markiert, die dieses Paar ausführen

In Bezug auf die jeweilige Rollenhierarchie

Tabelle 2.3: Prädikate für die Spezifikation von ...

Statusprädikate (Tabelle 2.4) sind Aussagen über Aktivitäten. Diese werden später mit den Informationen aus den Logs verglichen.

Konditionelle Prädikate (Tabelle 2.5)

Vergleiche (Tabelle 2.6)

Operationen (Tabelle 2.7)

Kopfprädikate. Diese müssen im Kopf einer Regel stehen, und dürfen nicht im Körper vorkommen. (Tabelle 2.8)

Prädikat	**Beschreibung**
('user')? UT 'executed' TT	Ut hat TT so ausgeführt, dass TT completed ??
'role' RT 'executed' TT	RT hat TT ausgeführt
UT 'is assigned to' TT	UT wurde TT zugewiesen
TT 'aborted'	TT ist abgebrochen
TT 'succeeded'	TT hat geklappt
UT 'is collaborator of' UT	UT sind alle Akteure, die an criticalTaskPair gearbeitet haben

Tabelle 2.4: Prädikate, um Aussgen über den Status in die Regeln mit einbeziehen zu können

Prädikat	**Beschreibung**
number where body is RES	Zählt die Anzahl der verschiedenen Lösungen für body
number of VAR where body is RES	Zählt die Anzahl der verschiedenen Lösungen für VAR, die body erfüllen. VAR muss mindestens einmal in body vorkommen.
sum of NT where body is RES	Gibt die Summe von NT zurück. NT muss im body enthalten sein und zählt alle Lösungen mit. NT muss
avg of NT ?TauT? where body is RES	Gibt den Durchschnitt von NT zurück.
min of NT ?TauT? where body is RES	Gibt das Minimum von NT zurück.
max of NT ?TauT? where body is RES	Gibt das Maximum von NT zurück.

Das Resultat wird in der Variable gespeichtert, die anstelle von RES definiert wurde. Body ist eine Konjunktion von Status- , Externen und Spezifikationsprädikaten.

Tabelle 2.5: Prädikate für Aussagen über die Akkumulation von Werten

Prädikat	**Beschreibung**
$= \mid != $	ww
$< \mid <= \mid > \mid >=$	dd
...	

Tabelle 2.6: Vergleiche

Prädikat	**Beschreibung**
$+ \mid -$	bb
$* \mid /$	bb
...	

Tabelle 2.7: Operationsn

2.4.1.3 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,...
Die Köper der Regel werden als Konjunktion (Disjunktion ist ebenfalls möglich, jedoch nicht zwingend notwendig - siehe Kapitel 2.5.1) von Prädikaten gebildet.

Prädikat	**Beschreibung**
UT cannot execute TT	ww
UT must execute TT	dd
RT cannot execute TT	
RT must execute TT	
illegal execution	

...

Tabelle 2.8: Prädikate für den Kopf einer Regel

```
IF body THEN head
```

Der Körper - body ...
Der Kopf der Regel - head darf nur aus

2.4.2 Grammatik - Syntax

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'
```

Abbildung 2.7: 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 ...

2.4.2.1 Definition eigener Prädikate

Es kann nötig erscheinen, eigene Prädikate zu definieren. Zum Beispiel um weitere Personengruppen anzulegen oder um Abkürzungen für Zusammenhänge zu erstellen. Dafür gibt es die Möglichkeit, ein Prädikat mittels des Schlüsselwortes "Deßu Beginn der Regeldefinitionen zu setzen. In der Definition wird der Typ der Argumente gesetzt. Man hat die Wahl zwischen UT,RT,TT,... Um die eigenen Prädikate nutzen zu können, muss man jedoch beachten, dass sie bei der Definition noch keinen "Wert"haben. Entweder muss man sie dann mit SSETßetzen

oder mann kann ihnen eine Regel erstellen. Um die Prädikate für den Parser und die weitere Verarbeitung eindeutig sind, sind sie an eine spezielle Form gebunden: predicate_name(ARG-Type,...).

```
DEF suspicious(UT)
DEF skipped(TT)
DEF aborted(UT,TT)

SET suspicous('Max Neuer')
SET suspicous('Tom Weisser')

IF EventType(ACTIVITY).'skipped' THEN skipped(ACTIVITY)
IF ACTOR executed ACTIVITY AND EventType(ACTIVITY).'aborted' THEN aborted(ACTOR, ACTIVITY)
```

Abbildung 2.8: Definition eigener Prädikate. .. wurde gesetzt, .. hat eine eigene Regel

2.5 Verwendung der Grammatik, Erklärungen

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.

2.5.1 Disjunktion

Es wird wenige Fälle geben, in denen eine Disjunktion von Klauseln notwendig ist. Um besser nachvollziehen zu können, zu welchem Fall eine Regelverletzung gehört, ist es oft sogar sinnvoller, eine Regel, die mehrere Fälle erlaubt, und mehrere Regeln aufzuteilen. Jedoch kann es verwendet werden, um Kardinalitätsaussagen einfacher darstelen zu können (TODO: mein Beispiel auf einem Zettel suchen), deswegen wird Disjunktion in der Grammatik erlaubt. Um die ??Assoziativität?? deutlich zumachen, müssen Disjunktionen in einer Klammer stehen. Disjunktion ohne umgebende Klammern ist nicht erlaubt. Disjunktionen von Konjunktionstermen ist nicht erlaubt.

```
IF (User executed 'task1' OR USER executed 'task2')
THEN User cannot execute 'task3'
```

Im Kopf einer Regel ist Disjunktion nicht erlaubt. Die Regel muss in zwei getrennte Regeln gespalten werden.

Abbildung 2.9: Disjunktion im Körper einer Regel. Man beachte, dass jede Disjunktion von umschließenden Klammern umgeben sein muss.

```
IF User executed 'task1'
THEN User cannot execute 'task2'

IF User executed 'task1'
THEN User cannot execute 'task3'
```

Abbildung 2.10: Disjunktion im Kopfbereich ist nicht erlaubt. Es müssen zwei getrennte Regeln erstellt werden.

2.5.2 Umgang mit verschiedenen Rollenmodellen

In den meisten Systemen wird ein hierarchisches Rollenmodell eingesetzt. Man ist bei der Analyse mit diesem Modelchecker nicht daran gebunden.

2.5.3 Negation

Negation wird ebenfalls für die Bildung der meisten Regeln nicht benötigt, wird hier aber der Vollständigkeit halber erlaubt. Dabei gilt hier das Prinzip der **negation as failure**. Da nicht garantiert werden kann, dass ein Arbeitsablauf vollständig und korrekt geloggt wurde, muss man sich hier darauf einigen, dass das nicht-vorhandenseinëiner Klausel trotzdem mit der Negation dieser Klausel gleichzusetzen ist. Sollte das ein Fehler sein, entsteht ein False-positive Eintrag, dh. es wird ein Fehler zuviel angezeigt.

2.6 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
```

Abbildung 2.11: Regeln für unsere gefundenen Beispiele

Kapitel 3

Praxis

3.1 Implementierung

3.1.1 Aufbau mein Programm

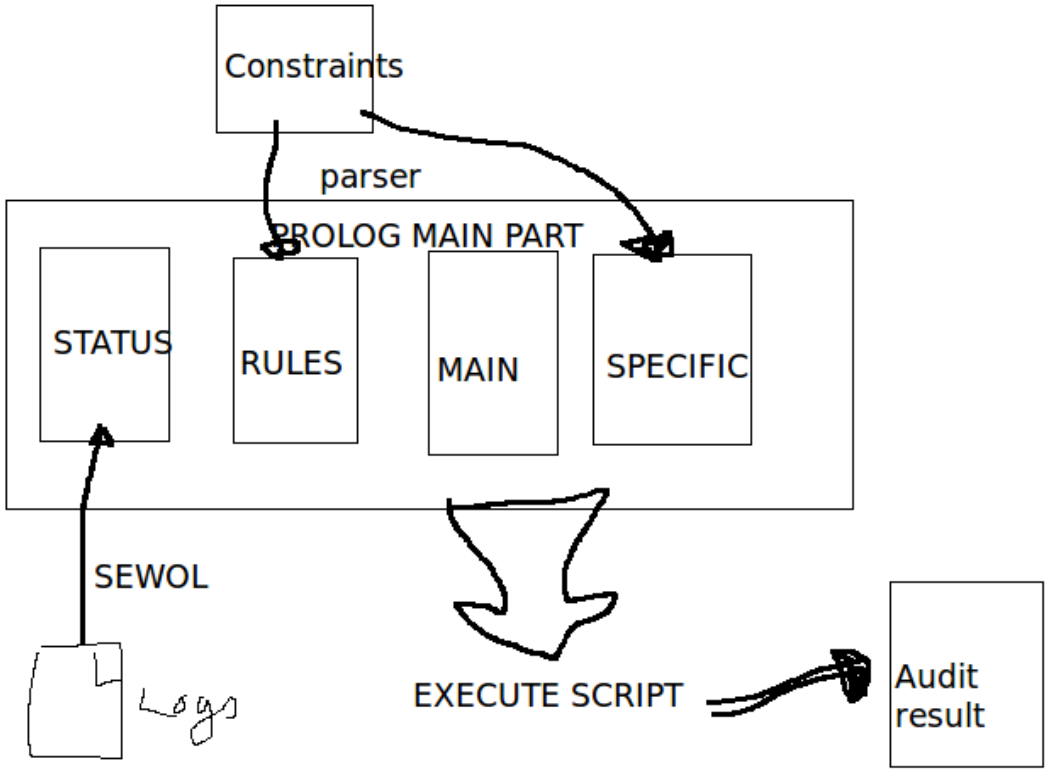

Abbildung 3.1: Aufbau mein Programm

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

3.1.2 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:

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

Tabelle 3.1: fd

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

3.1.3 Übersetzung der Regeln

Die Regeln werden in einer oder mehreren Dateien definiert und von dem ConstraintReader zuerst in speziellen Containern gespeichert, dort sortiert und angepasst und anschließend in zwei Prologdateien geschrieben, die später vom eigentlichen modelchecker verwendet werden.

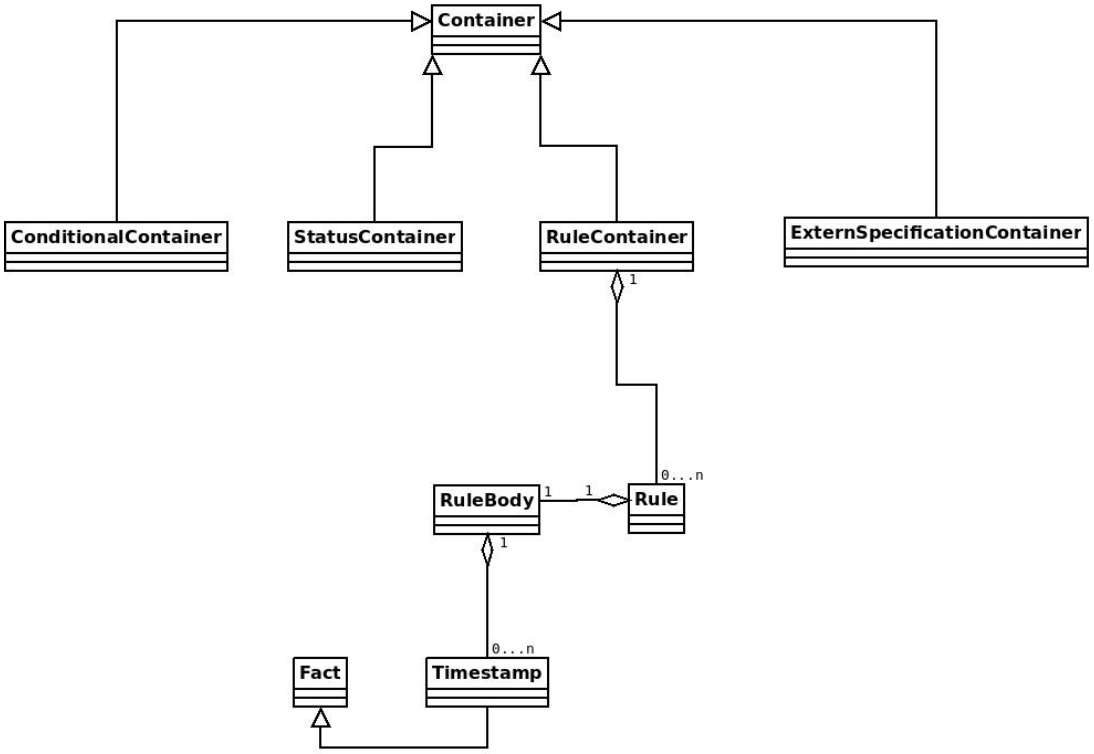

Abbildung 3.2: Interne Datenstruktur

3.1.4 Algorithmus Modelchecker

```
3. Add Collaborators
4. Adddominates
   Addrelated
5. For each cannot_execute_user(Actor, Activity):
6. If exists executed_user(Actor, Activity)
7.  then write trace
8. For each cannot_execute_role(Role, Activity):
9.  If exists executed_role(Role, Activity)
10. then write trace
11. For each must_execute_user(Actor, Activity):
12. If not exists executed_user(Actor, Activity)
13.  then write trace
14. For each must_execute_role(Role, Activity):
15.  If not exists executed_role(Role, Activity)
16. then write trace
```

```
17. For each panic write trace
```

Abbildung 3.3: Pseudocode des Modelchecker

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

Kapitel 4

Ergebnisse und Diskussion

4.1 Evaluation

4.1.1 Mein Beispiel mit ein paar Logs untersuchen

Zur Auswertung wurden für den in ... beschrieben Workflow wurden für jede Regel jeweils 3 Logs manuell erstellt, die zusammen genau eine Regelverletzung beinhalten. Des weiteren gibt einen Log für 3 Cases, in denen keine Regel verletzt wurde. Um den Reibungslosen Ablauf auch bei mehreren Regelbrüchen zu prüfen, werden zum Schluss alle Logs gemeinsam eingelesen. Es wird erwartet, dass die selben Fehler gefunden werden, die bereits in den einzelnen Logs auftraten (+ natürlich ein paar weitere, wenn es um Akkumulation geht).

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? Laufzeit? Bild vom Output

Kapitel 5

Verwandte Arbeit

5.1 Related work

5.1.1 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.

5.1.2 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.

5.1.3 warner inter-Instance

Diese Arbeit bezieht sich hauptsächlich auf die Arbeit von Warner und Atluri [1]. 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.

5.1.4 leitner instance-spanning

Leitner et al. [2] stellen in ihrer Arbeit das Identification and Unification of Process Constraints (IUPC) compliance framework vor, dessen Fokus ebenfalls nicht nur auf Regeln liegt, deren Aktivitäten zur selben Prozessinstanz gehören. In der Arbeit beschäftigen sie sich mit der Frage, auf welche Aktivitäten sich eine Einschränkung bezieht (sind alle Aktivitäten involviert oder nur diejenigen aus einer bestimmten Prozessinstanz) und welche Spannweite eine Einschränkung hat. Sie unterscheiden dabei zwischen *intra-instance*, *inter-instance*, *inter-process*, *inter-organizational* und *trans-organizational*. Auf eine explizite Unterscheiden zu Einschränkungen zwischen Organisationen wird bei meiner Arbeit verzichtet, da der Organisationsname im verwendeten Logmodell als Metaattribut gespeichtert werden würde und somit einfach über die bereits vorhandenen Ausdrücke definiert werden könnte.

5.1.5 tan consistency

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

5.1.6 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.

Kapitel 6

Ausblick

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

Anhang A

Weitere Beispiele für die Benutzung der Grammatik

Als Orientierungshilfe zum Erstellen eigener Regeln werden hier ein paar weitere Beispiele für Regeln gezeigt, die während der Thesis nicht behandelt wurden.

Intra Instance

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

```
DESC "Wenn etwas bestellt wurde, sollte es innerhalb von 3 Tagen verschickt werden."
IF 'receive order' succeeded AND ...

DESC "30 Tage, nachdem das Gewinnspiel begonnen hat,
darf kein neuer Teilnehmer mehr angenommen werden."

SET 'asd' is related to 'asdas'
DESC "Mitarbeiter aus dem Unternehmen dürfen nicht an dem Gewinnspiel teilnehmen"
IF P1 executed...

DESC "task1 und task2 müssen von verschiedenen Gruppen erledigt werden."
IF role R executed 'task1'
THEN role R cannot execute 'task2'
```

Inter Instance

Inter Process

Anhang 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

Tabelle B.1: Kommandozeilenparameter

Anhang C

User Manual

Benötigte Pakete

1. ANTLR 4.5 ...
2. SEWOL

Es wurde auf folgendem System getestet

1. java7
2. SWI-Prolog 6
3. Linux Ubuntu 12....

Wie startet man das programm?

Anhang D

Grammatik

<file>	::=	(<define>)* (<explicitSetting>)* (<assignment>)* <EOF>
<define>	::=	<def-symbols> <clause> ”(”<arg-type> (”,”<arg-type>)* ”)”
<explicitSetting>	::=	<set-symbols> <settableClauses> (<konj> <settableClauses>)*
<settableClauses>	::=	<extern> | <specification> | <definedClause>
<assignment>	::=	<if> <assignmentBody> <then> <assignmentHead>
<description>	::=	<desc> <constant>
<assignmentBody>	::=	(<neg>)? <clauses> (<konj> (<neg>)? <clauses>)*
<assignmentHead>	::=	<enforcement> | <definedClause>
<clauses>	::=	<atoms> | ”(”<atoms> (<disj> <atoms>)* ”)”
<atoms>	::=	<specification>
		| <status>
		| <comparison>
		| <conditional>
		| <extern>
		| <definedClause>
<definedClause>	::=	<clause> ”(”(<const> | <var>) (”,”(<const> | <var>))* ”)”
<var>	::=	<uppercase letter> | <variable><character>
<lowercase letter>	::=	a | b | c | ... | x | y | z
<uppercase letter>	::=	A | B | C | ... | X | Y | Z
<numeral>	::=	<digit> | <numeral><digit>
<digit>	::=	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<character>	::=	<lowercase letter> | <uppercase letter> | <digit> | <special>
<special>	::=	+ | - | * | / | || | : | . | ? | |#| $ | &
<string>	::=	<character> | <string><character>

Literaturverzeichnis

[1] 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.

[2] Maria Leitner, Juergen Mangler, and Stefanie Rinderle-Ma. Definition and Enactment of Instance-Spanning Process Constraints. In X.Sean Wang, Isabel Cruz, Alex Delis, and Guangyan Huang, editors, *Web Information Systems Engineering - WISE 2012*, volume 7651 of *Lecture Notes in Computer Science*, pages 652–658. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-35062-7.

[3] Christian Wolter and Andreas Schaad. Modeling of task-based authorization constraints in bpmn. In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, *Business Process Management*, volume 4714 of *Lecture Notes in Computer Science*, pages 64–79. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-75182-3.

[4] Günter Müller and Rafael Accorsi. Why are business processes not secure? In Marc Fischlin and Stefan Katzenbeisser, editors, *Number Theory and Cryptography*, volume 8260 of *Lecture Notes in Computer Science*, pages 240–254. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-42000-9.

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