Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Ein Werkzeug zur Erfassung von Geschäftsprozessen durch szenariobasierte Interviews

Bachelorarbeit

im Studiengang Informatik

von

Nils Prenner

Prüfer: Prof. Dr. Joel Greenyer
Zweitprüfer: Prof. Dr. Kurt Schneider
Betreuer: Prof. Dr. Joel Greenyer

Hannover, 19.03.2015

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 19.03.2015

Nils Prenner

Zusammenfassung

Firmen wollen ihre Geschäftsprozesse ständig verbessern, um ihre Produktivität zu steigern. Um diese Geschäftsprozesse analysieren zu können, muss erfasst werden, wie diese Prozesse zur Zeit ablaufen. Dafür werden Interviews geführt, in denen die Prozesse von Mitarbeitern der Firma beschrieben werden. Bei diesen Interviews kann es vorkommen, dass sich die einzelnen Personen widersprechen. Das Finden dieser Widersprüche ist aufwendig und kostet Zeit und Geld. Dieser Arbeitsschritt wird durch ein Tool erleichtert. In diesem Tool protokolliert der Interviewer die Prozesse mit einer formalen textuellen Sprache und das Tool verweist automatisch auf Widersprüche zu den anderen Interviews. Der Interviewer kann daraufhin diese Widersprüche sofort klären. Dieses Tool spart so viel Zeit und entlastet den Interviewer.

Abstract

Companies strive to improve their business processes in order to increase their productivity. In order to analyze these business processes, it must be recorded how these processes take place currently. Therefore interviews are conducted to show the processes of a company's employees. In such interviews it may happen that one person's statement is contradictory to another person's. Finding these contradictions is complex and costs time and money. This working step is facilitated by a tool. In this tool the interviewer records the processes with a formal and textual language, whereat the tool automatically points out inconsistencies regarding other interviews. Due to that, the interviewer can resolve contradictions instantly. This tool saves much time and simplifies the procedure of an interview.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Firmen wollen ihre Geschäftsprozesse möglichst effizient gestalten und immer weiter verbessern. Eine Firma kann so Kosten sparen und ihre Produktivität steigern. Um die Angestellten einer Firma bei ihrer Arbeit zu unterstützen, soll meist auch ein IT-System installiert werden. In vielen Fällen gibt es bereits ein solches IT-System, doch es gibt Stellen an denen es noch verbessert werden muss.

Um die Abläufe in einer Firma zu analysieren und dann zu verbessern, muss zuerst der Ist-Zustand dieser Geschäftsprozesse ermittelt werden. Eine Möglichkeit, dies durchzuführen, ist, Interviews mit Personen zu führen, die an diesen Prozessen beteiligt sind. So weiß der Interviewer, wie diese Prozesse zurzeit ablaufen und kann auf Grundlage dieser Erkenntnisse die Analyse durchführen und Verbesserungen vornehmen. Das Problem ist, dass sich die einzelnen interviewten Personen teilweise widersprechen oder etwas Wichtiges im Interview vergessen und so im Ist-Zustand Widersprüche oder Unklarheiten auftreten können. Das zeigt sich schon an dem folgenden Beispiel. Hier berichten zwei Mitarbeiter, die in derselben Abteilung arbeiten und dabei dieselbe Rolle (Job) verkörpern, wie sie einen Brief verschicken:

- **Mitarbeiter A:** „Wenn ich einen Brief schreibe, dann gebe ich den immer an die Poststelle und hefte immer eine Kopie davon ab. Danach lege ich immer die Datei auf Laufwerk X ab.
- **Mitarbeiter B:** „Wenn ich einen Brief schreibe, dann gebe ich den immer an die Poststelle und ich lege immer die Datei auf Laufwerk X ab.

Mitarbeiter B heftet anscheinend die Kopie des Briefes nicht ab. Da beide angeben, dass sie immer so vorgeben und dabei die gleiche Rolle verkörpern, liegt hier ein Widerspruch vor. Um diesen Widerspruch zu lösen gibt es nun vier Möglichkeiten. Entweder Mitarbeiter B hat das Abheften des Briefes nur vergessen oder aber das Abheften ist vom Management vorgeschrieben und er muss es in Zukunft auch machen. Eine andere Möglichkeit ist, dass

Mitarbeiter A sich geirrt hat und bei den Prozessen etwas vertauscht hat. Es kann aber auch sein, dass das Management auf Grund dieses Widerspruchs feststellt, dass das Abheften des Briefes nicht notwendig ist. So können die Geschäftsprozesse analysiert und verbessert werden.

Das Aufdecken von Widersprüchen ist sehr mühsam und einige Interviews müssen daher wiederholt werden. Das wiederum kostet viel Zeit und Geld. Ein zusätzliches Problem ist die Frage, in wieweit der Interviewer überhaupt alle Widersprüche vollständig aufdecken kann.

1.1 Lösungsansatz

Die Vision, um diese beiden Probleme zu lösen, ist, dass Widersprüche direkt beim Führen der Interviews aufgedeckt werden und der Interviewer so gezielt nach den fehlenden Informationen fragen kann. Diese Aufdeckung erfolgt durch ein Tool, in dem der Interviewer die Gespräche protokollieren kann. Damit wird Zeit gespart, da der Interviewer die Interviews nicht selber analysieren muss und eine computergestützte Analyse ist in diesem Fall gründlicher. Da für eine computergestützte Analyse die Interviews nicht in natürlicher Sprache protokolliert werden können, muss die Grundlage des Tools eine formale textuelle Sprache sein. Eine visuelle Sprache wäre auch möglich, aber in einem Interview muss das Protokollieren schnell gehen. Eine visuelle Sprache ist dafür zu klickintensiv und das Protokollieren würde zu lange dauern.

Während des Protokollierens gleicht das Tool die beschriebenen Prozesse mit den vorangegangenen Interviews ab und meldet dem Interviewer die gefundenen Widersprüche. Der Interviewer kann diese Punkte noch einmal klären. Vielleicht hat der Interviewte dort nur etwas vergessen. Es kann aber auch sein, dass die Angestellten die Prozesse grundsätzlich anders handhaben. Diese Umstände können dann gleich während des Interviews geklärt werden.

1.2 Zielsetzung der Arbeit

In meiner Arbeit leiste ich die Vorarbeit zu einer solchen computergestützten Analyse von Interviews. Dafür habe ich eine formale Protokollsprache entwickelt. Um eine Grundlage zu haben, auf der ich diese Sprache entwickeln kann, habe ich zwei Beispielinterviews geführt. Daneben habe ich eine umfangreiche Recherche zu den verschiedenen Modellierungstechniken für Geschäftsprozesse durchgeführt. Dabei habe ich untersucht, welche dieser Modellierungstechniken sich für meine Arbeit eignen. Zusätzlich habe ich die Beispielinterviews auf Probleme beim Führen von Interviews untersucht und daraus Regeln für das Führen von Interviews entwickelt.

Auf der Grundlage meiner Analyse habe ich meine Protokollsprache entwickelt und habe diese durch eine Validierung mit Hilfe der Beispielinterviews beständig verbessert. Die reine Syntax der Sprache reicht jedoch noch nicht aus, um meine Sprache nutzen zu können, daher habe ich mir auch bereits Gedanken über die Semantik meiner Sprache gemacht und dafür ein Konzept entwickelt.

Um den Leser meiner Arbeit einen Eindruck zu vermitteln, wie Widersprüche aussehen können, habe ich mir Gedanken über das Aussehen und Auftreten von Widersprüchen gemacht. Dafür habe ich jeweils kleine Beispiele entwickelt. Zum Abschluss meiner Arbeit habe ich noch ein drittes Interview durchgeführt, um meine Sprache in der Praxis zu testen. Ausgehend von diesem Interview führe ich eine Validierung meiner Sprache durch und gehe dabei auch auf mögliche Verbesserungen in der Zukunft ein. Für den Praxistest habe ich einen Editor-Prototypen entwickelt, mit dem ich dies durchgeführt habe.

Den Inhalt meiner Arbeit habe ich hier noch mal zur Verdeutlichung zusammengefasst.

1. Recherche über Modellierungstechniken und die Untersuchung dieser
2. Führen zweier Interviews und die Analyse der beiden
3. Die Erstellung einer Syntax für meine Protokollsprache
4. Die Erstellung einer Semantik für meine Protokollsprache
5. Die Untersuchung von Widersprüchen
6. Entwickelung einer Methodik für das Führen szenariobasierter Interviews
7. Entwicklung eines Editor-Prototypen
8. Führen eines dritten Interviews und Validierung der Protokollsprache

Hier möchte ich noch auf die Grenzen meiner Arbeit hinweisen.

Meine Arbeit öffnet ein neues Forschungsgebiet und ist daher eher als Wegweiser zu verstehen. Ich stelle in meiner Arbeit Konzepte vor und setzte diese zum Teil auch schon um. Trotzdem ist dieses Thema damit nicht abgeschlossen, sondern muss noch weitergedacht werden.

1.3 Struktur der Arbeit

In Kapitel 2 der Arbeit erkläre ich die Grundlagen, die für meine Arbeit wichtig sind. Dabei erkläre ich primär die visuelle Sprache Life Sequence

Charts. Hier verweise ich auch verwandte Themen und andere Modellierungstechniken für Geschäftsprozesse.

In Kapitel 3 führe ich eine Analyse von Beispielinterviews durch und erläutere dabei zunächst die Bedeutung von szenariobasierten Interviews. Dort wähle ich auch die passende Modellierungstechnik für meine Protokollsprache aus. Danach gehe ich auf Schwierigkeiten ein, die beim Führen von szenariobasierten Interviews auftreten.

In Kapitel 4 stelle ich die Syntax meiner Protokollsprache mit Hilfe von Beispielen dar. Im Anschluss gehe ich in Kapitel 5 auf die Semantik meiner Sprache ein. In Kapitel 6 erläutere ich an Hand der Semantik meiner Sprache, was Widersprüche sind. In Kapitel 7 gehe ich auf die Methodik zum Führen von szenariobasierten Interviews ein. In Kapitel 8 stelle ich meinen Editor-Prototypen, der zum Testen der Sprache dient, vor.

In Kapitel 9 führe ich eine Validierung meiner Protokollsprache durch und gehe auf mögliche Verbesserungen ein. Zum Abschluss erstelle ich in Kapitel 10 eine Zusammenfassung mit einem Ausblick in die Zukunft meines Themas.

Kapitel 2

Grundlagen

In diesem Kapitel stelle ich zuerst vor, was im allgemeinen Geschäftsprozesse sind. Danach gehe ich auf die allgemeine Bedeutung von Szenarien ein. Die Sprache Life Sequence Charts, die ich als Grundlage meiner Protokollsprache nutze, stelle ich im Anschluss vor. Zuletzt erkläre ich noch, was eine Domain Specific Language und das Xtext-Framework ist. Am Ende gehe ich noch auf verwandte Arbeiten ein.

2.1 Geschäftsprozesse

Bei einem Geschäftsprozess handelt es sich um eine Reihe miteinander verknüpfter Tätigkeiten, die ausgeführt werden, um ein unternehmerisches Ziel zu erreichen [5]. Die Tätigkeiten befinden sich in einem logischen Zusammenhang und werden von Menschen (Rollen) oder Maschinen, durch die Hilfe von Ressourcen und Informationen durchgeführt [4]. Dabei kann ein Geschäftsprozess Teil eines anderen Geschäftsprozesses sein. Geschäftsprozesse können voneinander abhängig sein und sich gegenseitig anstoßen. Ein Geschäftsprozess ist zudem wiederholbar und zeichnet sich durch einen fest definierten Anfang und ein fest definiertes Ende aus.

Bei Geschäftsprozessen gibt es die Kernprozesse. Diese tragen direkt zum Erreichen des unternehmerischen Ziels bei. Auf der anderen Seite gibt es die unterstützenden Prozesse, die für die Kernprozesse wichtig sind.

Geschäftsprozesse werden generell modelliert, um eine Analyse und eine Optimierung dieser Geschäftsprozesse durchzuführen. Dadurch können Kosten gespart werden, möglicherweise Personal abgebaut und Ressourcen besser genutzt werden.

Um eine Analyse durchzuführen, muss aus den bestehenden Geschäftsprozessen ein Modell gewonnen werden. Der Vorteil davon ist, dass die Komplexität der Prozesse reduziert wird und diese so besser analysiert werden können. Der Nachteil ist, dass durch ein Modell immer Informationen verloren gehen. Die grundlegende Aufgabe der Prozessmodellierung ist es

die einzelnen Arbeitsschritte, zu erfassen, zu strukturieren und darzustellen.

2.2 Szenarien

Werden Szenarien auf der untersten Ebene betrachtet, sind diese nur Erzählungen, sogenannte Storyboards. Diese Storyboards dienen dazu, das Verhalten von Systemen festzuhalten. Dabei wird entweder geklärt, wie sich ein System verhalten soll oder wie sich ein bereits existierendes System momentan verhält. In der frühen Entwicklungsphase eines Systems muss das Verhalten, das dieses System haben soll, erfasst werden. Dafür werden Stakeholder befragt, die das Verhalten erläutern. Dabei wird das Verhalten durch möglichst kurze und einfache Sätze spezifiziert [15]. Storyboards beschreiben das Verhalten eines Systems noch unpräzise, daher muss man verschiedene Aspekte klären, um ein klares strukturiertes Szenario zu erstellen. Es ist wichtig zu wissen, welche Akteure an einem Scenario beteiligt sind und wie diese genau untereinander agieren. In dieser Überlegung wird auch geklärt, was passiert, wenn ein bestimmtes Event auftritt und an welchen Stellen es im System zu einem Fehlverhalten kommen kann. Nach Lamsverde ist ein Szenario „eine typische Folge von Interaktionen zwischen Systemkomponenten, die ein implizites Ziel erreichen“ [15, S. 67]

2.3 Life Sequence Charts

Life Sequence Charts (LSC) ist eine visuelle Sprache, mit der sich Szenarien darstellen lassen. Sie ist eine Weiterentwicklung der Message Sequence Charts und wird von David Harel und Werner Damm in ihrem Buch „Come Let's Play“ [12] beschrieben. LSCs können in dem Eclipse Plugin PlayGo [7, 9] dargestellt und genutzt werden. Ich stütze mich bei meinen Erklärungen auf die dort gewählte Darstellungsweise.

Bei Life Sequence Charts gibt es ein auslösendes Szenario und ein Szenario, das daraufhin folgt. Um den Gebrauch von LSC besser zu verstehen, habe ich ein kleines Beispiel (siehe Abb. 2.1) erstellt. Dabei soll sich, wenn irgendein Button gedrückt wird, der Displaytext zu „Button pressed“ ändern.

2.3.1 Die Syntax

LSC ist eine visuelle Sprache, die Sequence Diagramme enthält. Diese Diagramme werden Life Sequence Charts genannt. Oben links in der Ecke steht der Namen des Charts und das Szenario, das dieses Diagramm darstellt. Ein LSC Diagramm besteht aus einem Prechart und einen Mainchart. Im Prechart wird ein auslösendes Szenario beschrieben. Das System ist daraufhin gezwungen, das Szenario, das im Mainchart beschreiben ist

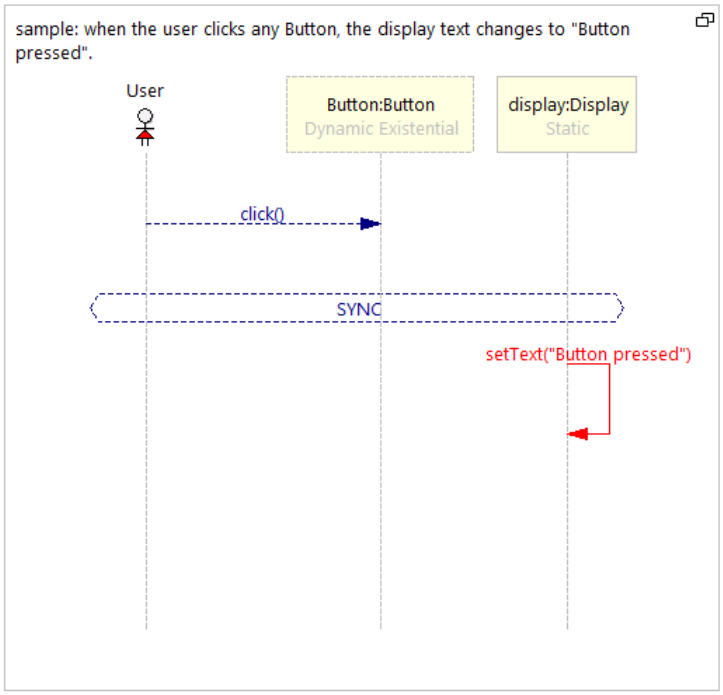

Abbildung 2.1: Beispiel für eine LSC-Diagramm

auszuführen. In diesem Beispiel ist das auslösende Ereignis, das der User auf irgendeinen Button drückt. Das System muss danach den Displaytext ändern. Prechart und Mainchart werden durch eine Synchronisation (sieh Abschnitt 2.3.1) getrennt.

Bei LSCs gibt es universelle und existenzielle Charts. Das universelle Chart wird, wie in der Abbildung 2.1 zu sehen, durch eine durchgezogene Umrandung dargestellt. Existenzielle Charts werden dafür genutzt, um ein beispielhaftes Verhalten eines Systems darzustellen, das von einem Systemdurchlauf erfüllt wird. Im Gegensatz dazu werden bei einem universellen Chart alle Systemdurchläufe erfüllt. Da sich meine Arbeit auf das Konzept der universellen Charts stützt, werde ich die existenziellen hier nicht weiter ausführen.

Des Weiteren gibt es in Charts Objekte, die durch vertikale Linien (Lifelines) dargestellt werden. Diese Objekte können untereinander Nachrichten austauschen, sogenannte Messages.

Lifelines

Objekte werde in LSCs durch gestrichelte vertikale Linien dargestellt. In einem LSC können beliebig viele Objekte miteinander interagieren, doch eine Lifeline wird immer genau einem Objekt zugeordnet. Über den

Lifelines sind Rechtecke zu sehen, die entweder eine durchgezogene oder eine gestrichelte Umrandung haben. Eine gestrichelte Linie bedeutet, dass ein Objekt dynamisch ist und es erst zur Laufzeit an ein bestimmtes Objekt gebunden wird. In diesem Beispiel (siehe Abb. 2.1) drückt der User auf irgendeinen Button, daher ist der tatsächliche Button erst zur Laufzeit bekannt. Damit ist der Button ein dynamisches Objekt. Eine durchgezogene Umrandung bedeutet, dass das Objekt statisch ist und damit schon während der Erstellung des LSCs an ein bestimmtes Objekt gebunden wird. Das Display steht schon bei der Festlegung des Charts als bestimmtes Objekt fest und ist daher statisch.

Im Rechteck steht zuerst der Name des Objekts und dann, getrennt durch einen Doppelpunkt, dessen Klasse. Zwischen den Lifelines können Messages ausgetauscht werden. Sie müssen als Pfeile in das Chart eingetragen werden. Dabei ist eine Lifeline wie ein Zeitstrahl, bei dem die Zeit von oben nach unten verläuft. Messages, die oben im Diagramm auftreten, passieren also vor denen, die weiter unten eingetragen wurden.

Messages

Objekte können durch Messages Methoden anderer Objekte aufrufen. Eine Message besteht aus einem Pfeil und einem Methodenaufruf. Der Pfeil beginnt an der Lifeline des Objekts, das sendet, und endet an der Lifeline des Objekts, das empfängt. Der Methodenaufruf besteht aus dem Namen der Methode und optionalen Parametern, die mit Klammern umschlossen werden.

Die aufgerufene Methode muss eine Methode der Klasse des Empfängerobjekts sein. Wenn der Pfeil der Message an der gleichen Lifeline endet, an der sie auch begonnen hat, ist es eine Self-Message. Bei Messages wird zudem noch die *temperature* und die *execution kind* unterschieden. Die *temperature* kann entweder *hot* oder *cold* sein. Eine *hot* Message wird als roter Pfeil dargestellt und eine *cold* Message als blauer Pfeil. Eine *hot* Message muss, wenn sie erwartet wird, auch auftreten. Eine *cold* Message muss dagegen nicht auftreten Bei der *execution kind* gibt es *monitored* und *executed*. Eine *monitored* Message wird durch einen gestrichelten Pfeil dargestellt und eine *executed* Message durch einen durchgezogenen Pfeil. Bei einer *executed* Message, muss diese ausgeführt werden, bei einer *monitored* Message, wird lediglich auf die Ausführung gewartet [7]. Der Klick des User (siehe Abb. 2.1) wird daher mit einer *cold monitored* Message dargestellt. Das Display ist im Mainchart gezwungen, eine Self-Message auszuführen, die *hot* und *executed* ist.

Conditions

Ein weiteres Element der LSCs sind Conditions, die durch gestrichelte Sechsecke dargestellt werden. In ihnen kann ein Ausdruck zu wahr oder falsch ausgewertet werden. Wenn dieser Ausdruck zu falsch ausgewertet wird, wird in diesem Chart nicht weitergegangen. Genau wie bei den Messages kann eine Condition *hot* oder *cold* sein, dargestellt durch eine rote oder blaue Linie. Wenn eine *cold* Condition zu falsch ausgewertet wird, ist das aktuelle LSC-Diagramm verletzt und wird abgebrochen. Eine *hot* Condition muss zu wahr ausgewertet werden, denn sonst liegt eine Verletzung der Requirements vor und der Systemablauf wird abgebrochen. Eine besondere Art der Condition ist eine Synchronisation. Eine Synchronisation ist eine Condition, die sich über beliebig viele Lifelines erstreckt. Eine Synchronisation dient dazu, eine bestimmte Reihenfolge im Chart einzuhalten. Dafür wird eine cold Synchronisation verwendet, in der „Sync“ steht. Diese Condition wird immer zu wahr ausgewertet.

Subchart

Ein Subchart wird durch eine durchgezogene schwarze Linie repräsentiert. Es kann entweder eine Alternative oder eine Schleife sein. Eine Alternative wird dadurch gekennzeichnet, dass oben links im Subchart „Alternatives“ steht. Darunter folgt ein Ausdruck, der entweder zu wahr oder falsch ausgewertet werden kann. Wenn der Ausdruck zu wahr ausgewertet wird, wird das im Subchart beschriebene Szenario ausgeführt. Diese Art von Subchart kann man auch als If-Anweisung interpretieren. Ein Subchart kann dabei auch eine if-else-Anweisung sein.

Ein weiteres Subchart Element ist die Loop. Dabei kann ein Szenario beliebig oft wiederholt werden. Wenn im Vorfeld genau feststeht, wie oft ein Szenario wiederholt werden soll, wird dafür bei der Loop eine Zahl eingetragen. Wenn die Anzahl der Durchläufe bei der Erstellung des Charts noch nicht bekannt ist, kann auch eine undefinierte Anzahl an Durchläufen angenommen werden.

2.3.2 Play-Out

Das Erstellen von verschiedenen LSC-Diagrammen wird generell Play-In genannt. Dabei wird das Verhalten eines Systems mit Hilfe der LSC-Diagramme beschrieben. Beim Play-Out handelt es sich um eine Methode, bei der diese Diagramme ausgeführt werden. Dabei führt das System alle Steps, die *hot* und *executed* sind, aus, bis das System wieder eine Usereingabe oder eine Veränderung der Umwelt erwartet [7].

Der Zustand, in dem sich ein LSC-Diagramm beim Ausführen befindet, wird durch einen Cut dargestellt. Ein Cut ist eine horizontale Linie, die sich über die Lifelines erstreckt. Ein Cut kann jeweils *hot* oder *cold* sein, je

nachdem, ob das nächste erwartete Event *hot* oder *cold* ist. Der aktuelle Cut zeigt an, welches Event im Diagramm aktiviert ist und damit als nächstes passiert.

Wenn mehrere LSC-Diagramme nebeneinander ausgeführt werden, kann es an einigen Stellen passieren, dass mehrere Events aktiviert sind. Der Play-Out-Algorithmus führt die Events dabei in einer zufälligen Reihenfolge aus. Dieses Vorgehen kann jedoch zu einer Verletzung (Violation) des LSC-Diagramms führen. Ein LSC-Diagramm wird verletzt, wenn eine Message passiert, die im Diagramm erwartet wird, jedoch durch den aktuellen Cut nicht aktiviert ist. Ist der Cut *cold*, so führt dies zu einem Abbruch des Diagramms. Ist der Cut jedoch *hot*, so führt das zu einem generellen Abbruch des Systemablaufs. Um festzustellen, wann es zu einer Violation kommt, ist es wichtig, dass die Messages auf ihre Gleichheit hin überprüft werden. Zwei Messages sind gleich, wenn sowohl das Sendeobjekt und das Empfängerobjekt gleich sind, jeweils die gleiche Message gesendet wird und die übergebenen Parameter gleich sind.

Wenn das erste Event aus einem Prechart passiert, wird eine Kopie des LSC-Diagramms erstellt. Der Lebenszyklus dieser Kopie wird in der Abbildung 2.2 dargestellt. Die Abbildung habe ich aus Come Let's Play [12, S.68] übernommen und leicht modifiziert, um sie verständlicher zu machen.

Wenn das erste Event aus dem Prechart passiert, wird eine Kopie des Charts erstellt und das Prechart aktiviert. Wenn das Prechart durch ein falsches Event verletzt wird, bricht das Chart ab und geht zu Exit. Die Kopie wird daraufhin wieder gelöscht. Wird das Prechart jedoch erfolgreich ausgeführt, wird das Mainchart aktiviert. Wird das Chart dort durch ein falsches Event in einem *hot* Cut verletzt, geht es in einen Fehlerzustand, aus dem es nicht mehr herauskommt. Wird das Chart dort durch ein falsches Event in einem *cold* Cut verletzt, bricht das Chart ab und geht in Exit. Der Ablauf aller LSCs stoppt dadurch jedoch nicht, sondern nur die Kopie dieses einen Charts. Die Kopie wird auch hier wieder gelöscht.

Wenn das Mainchart ohne Violation ausgeführt wird, geht es ebenfalls in Exit und die Kopie wird auch hier im Anschluss gelöscht.

2.4 Domain-Specific Language

Eine Domain-Specific Language (DSl) [8] wird für einen bestimmten Themenbereich (Domain) erschaffen. Sie ist für diese Domain zugeschnitten und soll sich möglichst nahe an ihr orientieren. Die Idee dahinter ist, dass durch eine DSL Sachzusammenhänge, die die zugehörige Domain betreffen, leichter dargestellt werden können. Die Sprache ist so auch einfacher zu erlernen. Ein Beispiel für eine solche Sprache ist SQL. Im Gegensatz dazu ist zum Beispiel C eine Sprache, die universell einsetzbar ist. Sie ist damit jedoch sehr komplex und schwer zu erlernen.

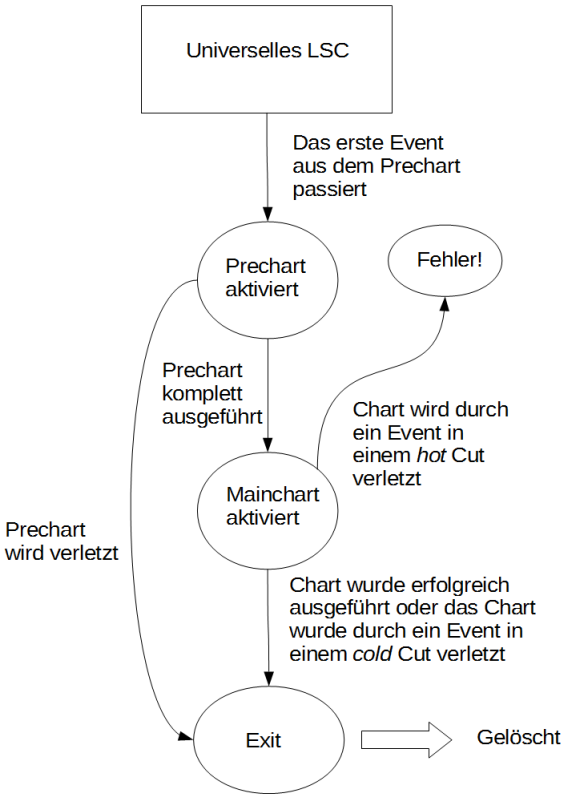

Abbildung 2.2: Der Lebenszyklus einer Kopie eines universellen Charts

Xtext [8] ist ein Eclipse Plugin mit dessen Hilfe eine DSL erstellt werden kann. Der Kern von Xtext ist, dass für eine entwickelte DSL eine Grammatik erstellt werden kann. Xtext stellt für die erstellte Grammatik einen Parser , eine Syntax-Färbung, eine Autovervollständigung und eine Validierung des Codes zur Verfügung. Xtext stellt zudem einen Editor zur Verfügung, in dem in Eclipse die Sprache genutzt werden kann. Der Compiler der Sprache ist jedoch unabhängig von Eclipse und kann in jeder Java Umgebung genutzt werden.

Da es sich bei einer DSL oft um eine Programmiersprache handelt, soll diese ausführbar sein. Daher bietet Xtext einen Code-Generator für die DSL.

Da ich in meiner Arbeit nicht zum Erstellen eines Tools komme und nur einen Prototypen meines Editors entwickle, gehe ich auf Xtext hier nicht weiter ein.

2.5 Verwandte Arbeiten

In meiner Arbeit betrete ich kein neues Feld, da es bereits eine Vielzahl an Modellierungstechniken für die Darstellung von Geschäftsprozessen gibt. Bei meinen Recherchen habe ich mir einige Techniken angesehen und möchte die wichtigsten hier kurz vorstellen.

Als Modellierungstechnik gibt es zum Beispiel bereits die ereignisgesteuerten Prozessketten (EPKs) [2], diese bestehen aus einem gerichteten Graphen. In diesem Graph gibt es Tätigkeiten, Ereignisse und Verknüpfungen. Dabei wechseln sich Ereignisse und Tätigkeiten ständig ab und werden durch logische Operatoren miteinander verknüpft.

Neben den EPKs gibt es eine ähnliche Modellierungstechnik, die Business Process Model and Notation (BPMN) [1]. Diese hat den Vorteil, dass sie bekannter ist als die EPKs und standardisiert ist. Zudem bietet sie eine deutlich größere Anzahl an Funktionen.

Zusätzlich gibt es noch die Modellierungstechnik FLOW [3]. Dabei geht es darum, Informationsflüsse in einer Firma darzustellen. Dafür wird dargestellt, wie Personen miteinander kommunizieren und in welcher Form Informationen auftreten.

Eine weitere Technik, die den EPKs ähnlich ist, ist das Modellieren von Geschäftsprozessen durch Petri-Netze [14, 6]. Ein Petri-Netz ist ein Graph, in dem es zwei Arten von Knoten gibt, die Stellen und die Transitionen. Eine Transition schaltet, wenn die vorherigen Stellen aktiv sind. Zudem wird in einem Kapitel des Buches EMISA [10] erläutert, wie Rollen in einem Petri-Netz eingebaut werden können.

Bei meinen Recherchen stieß ich auf einen weiteren Artikel [13]. In diesem, wird beschrieben, wie das Vorgehen bei der Softwareentwicklung genutzt werden kann, um Geschäftsprozesse zu erfassen.

Bei der Entwicklung eines Systems muss in der Softwareentwicklung festgehalten werden, wie sich ein System verhalten muss. Dabei wird festgehalten, welches Verhalten ein System haben darf, aber auch welches es nicht haben darf. Dieses Verhalten geben Stakeholder an. Stakeholder sind Personen, die von diesem System betroffen sind. Um das Verhalten des Systems von Stakeholdern zu erfahren, müssen Interviews mit diesen Personen durchgeführt werden. Diese Interviews laufen szenariobasiert ab. Was ein Szenario in diesem Zusammenhang ist, habe ich bereits erklärt (siehe Abschnitt 2.2).

Eine weitere Arbeit, die ich hier noch erwähnen möchte ist PlayGo [7, 9]. PlayGo habe ich bereits bei den Life Sequence Charts verwendet. Neben der reinen Darstellung von LSC-Diagrammen bietet PlayGo eine Übersetzung von natürlicher Sprache in die visuellen Life Sequence Charts [11]. Da ich mich in meiner Arbeit auch mit dem Übersetzen von in natürlicher Sprache geführten Interviews in eine formale Sprache beschäftige, ist PlayGo eine interessante Arbeit.

Kapitel 3

Analyse

Zu Beginn dieser Arbeit habe ich zwei Interviews geführt, die ich hier vorstellen werde. Ausgehend von den Erfahrungen, die ich in diesen Interviews gesammelt habe, werde ich auf Schwierigkeiten beim Führen der Interviews hinweisen. Vorher untersuche ich jedoch, welche der vorgestellten Modellierungstechniken (siehe Abschnitt 2.5) sich als Grundlage für meine Protokollsprache eignen.

3.1 Analyse der möglichen Modellierungstechniken

In diesem Abschnitt untersuche ich, welche der vorgestellten Modellierungstechniken sich als Grundlage für meine Sprache eignen. Bei meinen Recherchen habe ich mir einige Techniken angesehen und habe diese bereits in den Grundlagen 2.5 kurz vorgestellt. Hier untersuche ich welche dieser Methoden sich als Grundlage für meine Sprache eignen.

Als Modellierungstechnik habe ich die ereignisgesteuerten Prozessketten (EPKs) [2] und die Business Process Model and Notation vorgestellt. Diese beiden Techniken eignen sich jedoch eher, um die grundlegenden Prozesse in einer Firma darzustellen, losgelöst von bestimmten Personen beziehungsweise Rollen. Da ich mit meiner Sprache jedoch Widersprüche zwischen den einzelnen interviewten Personen aufdecken möchte, muss in meiner Sprache das Konzept von Rollen enthalten sein.

Die Modellierungstechnik FLOW [3] bietet zwar das Konzept von Rollen und das Austauschen von Nachrichten, jedoch führt sie weg von der Vorstellung einzelner Prozesse. Denn auch hier werden eher die grundlegenden Abläufe beziehungsweise Informationsflüsse modelliert.

Die nächste Modellierungstechnik sind die Petri-Netze. Sie haben den Vorteil, dass sie sehr gut an verschiedene Bedürfnisse angepasst werden können. Jedoch ähneln sie zu sehr den EPKs, sodass sie für meine Sprache nicht das passende Konzept sind.

In den Grundlagen habe ich bereits die Arbeit vorgestellt, in der es

darum geht, dass das Vorgehen aus der Softwareentwicklung genutzt wird, um Geschäftsprozesse darzustellen. Dabei habe ich auch schon das Führen von Interviews in der Softwareentwicklung vorgestellt.

In diesen Interviews wird oft deutlich, dass es bei dem Verhalten eines Systems ein auslösendes Ereignis gibt und Abläufe, die daraufhin folgen müssen. Genau diesen Zusammenhang möchte ich auch in meiner Sprache ausdrücken, nur, dass dabei nicht das Verhalten eines Systems, sondern die Prozesse in einer Firma festgehalten werden.

Eine Technik, die genau dieses Modellieren bietet, sind die Life Sequence Charts, die ich bereits in den Grundlagen vorgestellt habe. Sie bieten einige Vorteile. Zum einen besitzt die Sprache LSC ein Klassen-und-Objekt-Konzept, das sich auf Rollen einer Firma übertragen lässt. Zum anderen gibt es Nachrichten, die zwischen den Objekten ausgetauscht werden können. Im Hinblick auf die Modellierung von Geschäftsprozessen ist das eine interessante Eigenschaft. Dabei lässt sich genau ausdrücken, wie ein Prozess startet, wo er anfängt und wo er endet.

Mit Hilfe meiner Sprache möchte ich ausdrücken, was in einem Geschäftsprozess passieren kann beziehungsweise muss. Da das eine szenariobasierte Vorstellung eines Geschäftsprozesses ist, bietet sich als Grundlage für meine Protokollsprache die LSCs an. PlayGo, das ich bereits in den Grundlagen erwähnt habe, ist ein weiterer Grund auf die LSCs zu setzen.

3.2 Die Interviews

Zu Beginn dieser Arbeit geht es darum, Beispiele für Geschäftsprozesse zu finden. Daher habe ich zwei Interviews geführt, um authentisch Beispiele zu haben (siehe Anhang A.1 A.2). Mein erster Interviewkandidat ist ein Beratungslehrer für Sonderpädagogik. Er ist Lehrer für Sonderpädagogik, doch die meiste Zeit arbeitet er in einer Behörde, die Beratung für Lehrer, Eltern und Schüler zum Thema Sonderpädagogik anbietet. Meine zweite Interviewkandidatin ist bei einer Computerzeitschrift angestellt, die alle zwei Wochen erscheint. Die Zeitschrift kann in einer App über ein Tablet oder Smartphone abgerufen werden. Ihre Aufgabe ist es, die Artikel der gedruckten Zeitung für die App-Ansicht anzupassen.

3.3 Probleme beim Führen szenariobasierter Interviews

Hier werde ich nun die Erfahrungen, die ich beim Führen der Interviews gemacht habe darstellen und auf Probleme hinweisen, die beim Führen von szenariobasierten Interviews auftreten.

3.3.1 Die Unwissenheit bei den Beteiligten eines Interviews

Der Interviewer kennt das Betätigungsfeld der Firma nicht. In den einzelnen Abteilungen werden viele Fachbegriffe und Abkürzungen genutzt, die der Interviewer nicht kennt. Daraus ergibt sich, dass der Interviewer sich nicht nur auf das Interview selbst und den szenariobasierten Ansatz konzentrieren muss, sondern auch auf die Materie, mit der sich die Firma oder der jeweilige Mitarbeiter beschäftigt. Zudem weiß der Interviewte in der Regel nicht, was mit szenariobasierten Interviews gemeint ist.

3.3.2 Überblick über das Interview

Bei einem Interview ist es schwierig den Überblick zu behalten. Wichtige Informationen können schnell verloren gehen, da sie vielleicht in einem Nebensatz genannt werden oder der Interviewte weiterredet, während der Interviewer das Beschriebene protokolliert. Ein Beispiel für so einen möglichen Informationsverlust ist zum Beispiel die Stelle 10 des Beratungslehrer-Interviews (BL-Interview). Dort gibt der Beratungslehrer an, dass es mehrere Möglichkeiten gibt, wie ein Prozess abläuft. Dabei habe ich mich jedoch erst einmal auf den Regelfall konzentriert (Stelle 11). Und erst an Stelle 27 komme ich wieder auf die anderen Fälle zu sprechen. Manchmal können Prozesse auch etwas länger sein und der Interviewer vergisst schnell, dass für den gleichen Prozess noch andere Ablaufmöglichkeiten existieren. Der Interviewte kann auch plötzlich zu viel auf einmal sagen. Das ist problematisch, da dort meist Informationen verloren gehen.

3.3.3 Die Fragen bei einem Interview

Interviews, wie sie hier durchgeführt werden, bieten wenige Möglichkeiten sich im Vorfeld Fragen zu überlegen. Die Fragen kommen im Laufe des Interviews automatisch. Trotzdem zeigt sich auch hier, dass es Schwierigkeiten gibt. Dabei ist zu beachten, welche Fragen gestellt und wie sie gestellt werden. Am Anfang beider Interviews habe ich erst einmal gefragt, welchen Beruf die beiden Interviewten ausführen. Diese Frage eignet sich als Einstieg gut. Zuerst bekommt der Interviewer einen Eindruck darüber, was sein Gegenüber ungefähr in der Firma macht, in welchem Bereich der Firma er tätig ist und welche Position er in der Firma innehat. Jedoch liegt in so relativ offenen Fragen auch die Gefahr, dass der Interviewte zu viel auf einmal sagt. An Stelle 7 des Tablet-Producer-Interviews (TP-Interview) frage ich einfach nur: „Wie läuft das dann ab?“ Das ist eine sehr offene Frage und gibt sehr viel Raum zum Antworten und damit auch für lange Antworten. Doch, wie ich im vorherigen Abschnitt gezeigt habe, ist das eher schlecht. Andererseits hat sich gezeigt, dass der Interviewte nicht immer weiß, wie er anfangen soll. Also ist eine solche Frage am Anfang gut, um den Einstieg für den Interviewten zu erleichtern.

3.3.4 Unterschied zwischen geplanten und tatsächlichen Prozessen

Bei Geschäftsprozessen ist es oftmals so, dass ein bestimmter Ablauf durch das Management einer Firma vorgeschrieben wird. Aber es halten sich nicht immer alle Mitarbeiter einer Firma daran. Das kann verschiedene Gründe haben. Zum Beispiel kann der vorgeschriebene Ablauf zu umständlich oder zu zeitraubend sein. Ein Beispiel dafür ist an Stelle 44 des TP-Interviews zu finden.

Dort wird berichtet, dass ein Ticketsystem zur Qualitätssicherung genutzt werden soll. Doch einige Mitarbeiter greifen nicht darauf zurück und nutzen stattdessen E-Mails für den gleichen Prozess. Hier stellt sich nun die Frage, ob das Ticketsystem überflüssig ist oder es nur verbessert werden muss. In den Interviews geht es nicht darum zu erfassen, wie ein Prozess ablaufen sollte, sondern wie er tatsächlich abläuft. Dem Mitarbeiter einer Firma ist es natürlich peinlich, wenn er einen Prozess ganz anders handhabt, als es vorgeschrieben ist. Deshalb wird er in einem Interview eher von dem vorgeschriebenen Weg erzählen. Diese Abweichungen haben das Potenzial, besser zu sein als der vorgeschriebene Weg oder aber sie zeigen ein Fehlverhalten innerhalb der Firma, das so offengelegt wird. Daher ist für die Analyse wichtig, dass die Interviewten von den tatsächlichen Abläufen der Prozesse berichten.

Kapitel 4

Die Protokollsprache

In diesem Kapitel werde ich meine Protokollsprache an Hand von Beispielen vorstellen. Meine Beispiele stützen sich auf das BL-Interview. Ich stelle die Regeln vor und erkläre die Handhabung meiner Sprache. An den passenden Stellen habe ich zusätzlich Klassendiagramme hinzugefügt, um die Struktur meiner Sprache verständlicher zu machen.

Mein Sprache ist eine Domain-Specific Language, mit der sich Geschäftsprozesse darstellen lassen.

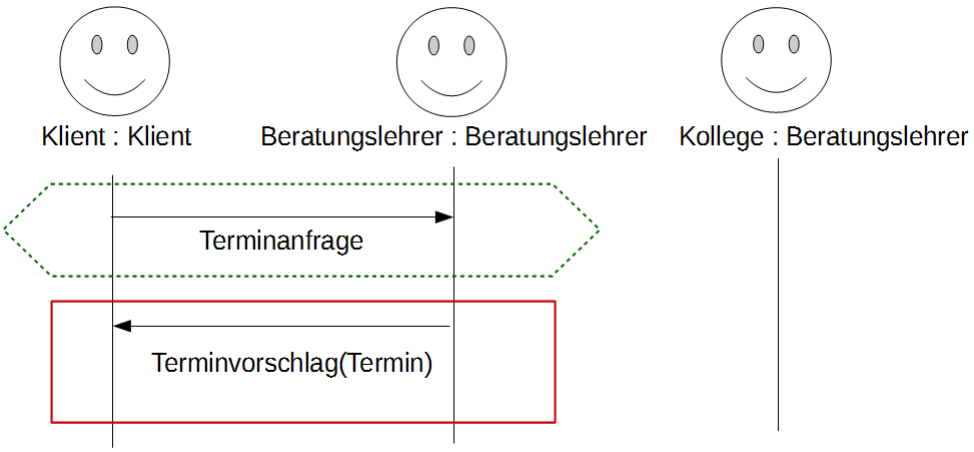

Abbildung 4.1: Das Konzept meiner Sprache

Mein Sprache basiert lose auf dem Konzept der LSC. Daher habe ich für die Abbildung 4.1 ein ähnliches Aussehen gewählt. Jedoch haben die Pfeile hier keine Farbe, da ich nicht alle Konzepte aus den LSCs übernehme. Meine Sprache ist eine textuelle Sprache, daher dient diese Abbildung lediglich zur Erklärung und hat keine syntaktische Bedeutung.

Die Abbildung zeigt einen beispielhaften Prozess. Wichtig in meiner Sprache ist, dass es, wie bei den LSCs, ein auslösendes Szenario (grünes

Sechseck) und ein Folgeszenario (rotes Rechteck) gibt. In diesem Beispiel bittet der Klient um einen Termin und der Beratungslehrer muss danach einen Terminvorschlag machen.

Ein wichtiger Bestandteil meiner Sprache sind die Rollen. Rollen sind Platzhalter für Personen, die diese erst zur Laufzeit eines Prozesses einnehmen. Rollen haben einen Typ (Wort hinter dem Doppelpunkt) und einen Namen (Wort vor dem Doppelpunkt). Ich folge damit dem Konzept der dynamischen Objekte aus den LSCs. In diesem Fall gibt es zum Beispiel die Rolle des Beratungslehrers. In der Institution des Beratungslehrers gibt es jedoch mehrere Beratungslehrer. Diese nehmen aber unterschiedliche Rollen ein. Die unterschiedlichen Rollen können, wie hier, zum Beispiel der Kollege oder der Chef sein. Beide Rollen haben zwar den gleichen Typ, sind aber im Prozess unterschiedliche Personen und damit unterschiedliche Rollen.

Weiterhin ist in meiner Sprache die Kommunikation zwischen den Rollen wichtig. Daher sind die Pfeile Kommunikationskanäle, über die die Rollen kommunizieren. Die Beschriftung der Pfeile ist die Nachricht, die gesendet wird. Neben den Nachrichten wird in meiner Sprache auch der Austausch von Informationen modelliert. Das ist an dem Termin zu sehen, der in der zweiten Nachricht übermittelt wird.

An diesem Beispiel wird bereits die grundlegende Struktur meiner Sprache deutlich. In den nächsten Abschnitten stelle ich meine Sprache im einzelnen und ausführlich vor.

4.1 Die Packagestruktur

Zuerst muss ein neues Projekt erstellt werden. Das Projekt trägt am besten den Namen der Institution, in der die Analyse durchgeführt werden soll. In jedem neuen Projekt gibt es das InterviewPackage, das RolePackage, das SystemPackage, das DataPackage und das ChannelPackage. Im InterviewPackage werden alle Interviews, die geführt werden, abgelegt. Das RolePackage beinhaltet alle Rollentypen, die in den Interviews vorkommen. Das SystemPackage enthält alle Systemtypen, mit denen gearbeitet wird. Im DataPackage werden alle benötigten Datentypen abgelegt. Im ChannelPackage befinden sich alle Kommunikationsmittel (Channel), mit denen die Rollen in den Interviews kommunizieren.

In diesem Diagramm (siehe Abb. 4.2) wird deutlich, in welcher Beziehung die Packages zueinander stehen. Das InterviewPackage kennt alle anderen Packages. System- und RolePackage kennen sich gegenseitig und auch sie kennen das DataPackage. Das Channel- und Datapackage kennt sich jeweils nur selbst.

Diese Aufteilung sorgt dafür, dass die Interviews und die Bestandteile der Interviews nicht vermischt werden. Diese Objekttypen werden für alle Interviews erstellt, da in verschiedenen Interviews mehrere Rollen des

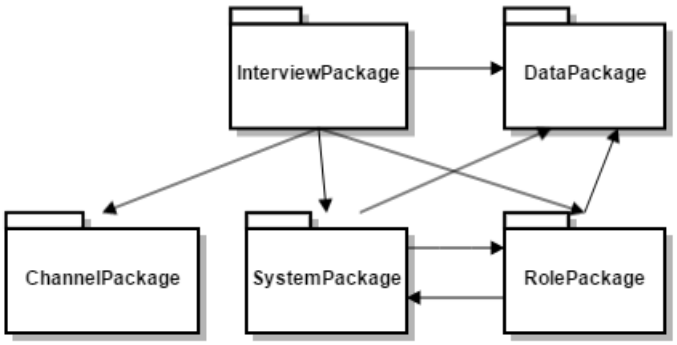

Abbildung 4.2: Die Packagestruktur

gleichen Typs auftreten können. Für die Datentypen und Systemtypen gilt das gleiche.

Eine neue Datei wird in einem der vier Packages abgelegt. Zu Beginn muss oben in der Datei angegeben werden, in welchem Package sich diese Datei befindet. Die Package-Deklaration wird mit einem Semikolon beendet. Je nachdem in welchem Package sich diese Datei befindet, kann in dieser Datei ein Interview, ein Rollentyp (Roletype), ein Systemtyp (Systemtype), ein Datentyp (Datatype) oder ein Channel beschrieben sein. In jeder Datei kann jeweils immer nur eins dieser Elemente beschrieben werden.

Kommentare können an jeder Stelle eingesetzt werden. Ein Kommentar beginnt mit den Zeichen „//“. Die gesamte Zeile, die hinter diesen beiden Zeichen folgt, ist nun als Kommentar gekennzeichnet.

4.2 Das Interview

Der Hauptbestandteil meiner Protokollsprache ist das Interview. Das Schlüsselwort „Interview“ zeigt an, dass hier ein neues Interview erstellt wird. Dahinter folgt der Titel des Interviews, der als String angegeben wird, da ein Interview nicht immer mit einem Wort beschrieben werden kann. Geschweifte Klammern kennzeichnen den Beginn und das Ende des Interviews.

Am Anfang werden zuerst die Objekte, die in dem Interview wichtig sind, deklariert. Im Laufe des Interview kann es sein, dass Objekte dazu kommen. Die Objekte können vom Typ Rolle (Role), Daten (Data), System (System) oder Boolean sein. Jedes Objekt besteht daher aus dem Objekttyp und einem Namen. Die Definition eines Objektes wird jeweils mit einem Semikolon beendet. Diese Zusammenhang habe ich hier nochmal als Klassendiagramm (siehe Abb. 4.4) dargestellt.

Bei der Angabe einer Role gibt es einige Dinge zu beachten. Eine Role

```
package InterviewPackage;

Interview "Beratungslehrer" {

    main insideRole Beratungslehrer beratungslehrer;
    outsideRole Klient klient;
    data Termin termin;
    system Outlook outlook;
    boolean klientIstBekannt;

    process "Klient benötigt Beratung"{

    }
}
```

Abbildung 4.3: Definition eines Interviews

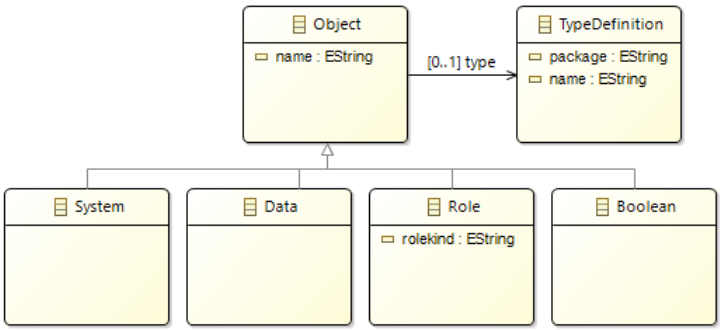

Abbildung 4.4: Ein Objekt in meiner Sprache

bezeichnet eine unbestimmte Person, die erst, wenn der Prozess stattfindet, durch eine bestimmte Person eingenommen wird. Die interviewte Person berichtet in einem Interview von seiner Role. Zu Beginn des Interviews wird daher die Role der interviewten Person mit einem „main“ als Mainrole gekennzeichnet. Diese Mainrole muss bei jedem Interview angegeben werden. In diesem Beispiel (siehe Abb. 4.3) ist die Mainrole der Beratungslehrer.

Bei der Angabe einer Role muss angegeben werden, ob diese Role eine Insiderole oder Outsiderole ist. Eine Insiderole kommt von innerhalb der Institution und eine Outsiderole von außerhalb. Der Beratungslehrer ist Teil der Institution und ist damit eine Insiderole. Im Gegensatz dazu tritt der Klient von außerhalb an die Institution heran und ist damit eine Outsiderole. Die Eigenschaft wird im Klassendiagramm (siehe Abb. 4.5) mit der „rolekind“ gekennzeichnet.

Dahinter muss der Roletype der Role folgen. In einem Interview können mehrere Roles auftreten, die den gleichen Roletype haben. Am Schluss folgt noch der Name der Role.

Neben Roles können auch Datas und Systems jeweils mit dem Schlüssel-

wort „data“ und „system“ erstellt werden.

Zuletzt sind noch die Angaben von Booleans möglich. Das sind Objekte, die nur den Wert „wahr“ oder „falsch“ annehmen können. Sie werden durch das Schlüsselwort „boolean“ und durch die Angabe eines Namens erschaffen.

Hinter der Deklaration der Objekte können noch beliebig viele Prozesse folgen (siehe 4.5). Das Interview habe ich auch nochmal als Klassendigramm (siehe Abb. 4.5) dargestellt. Daran wird deutlich, dass ein Interview aus einer Reihe von Objekten, Prozessen und einer Mainrole besteht.

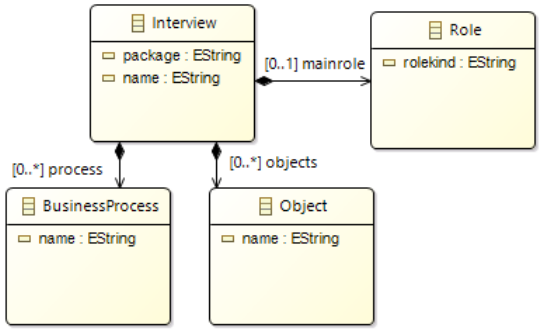

Abbildung 4.5: Klassendiagramm eines Interviews

4.3 Die Definition eines Objekttyps

In meiner Sprache können verschiedene Objekttypen erstellt werden. Erstellt werden kann der Roletype, der Systemtype und der Datatype (siehe Abb. 4.5).

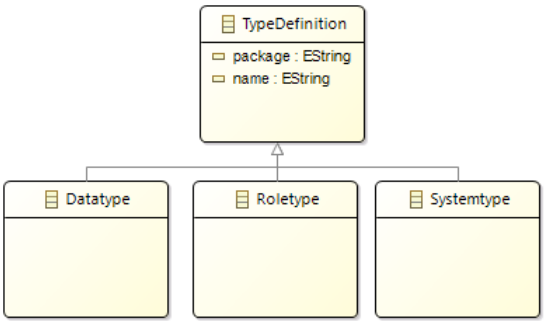

Abbildung 4.6: Das Klassendiagramm der Typedefnition

Ein neuer Typ wird jeweils durch das Schlüsselwort „Roletype“, „Systemtype“ oder „Datatype“ erschaffen. Dahinter folgt der Name des jeweiligen

Typs. Dieser muss immer großgeschrieben werden. Mit geschweiften Klammern beginnt und endet die Definition eines Typs. Die drei Typen stelle ich hier in den folgenden Abschnitten noch eingehender vor.

4.3.1 Die Definition eines Roletypes

Eine Role kann sich in verschiedenen Zuständen (States) befinden, daher werden diese bei der Definition eines Roletypes zuerst deklariert. In diesem Fall kann es sein, dass der Beratungslehrer zu beschäftigt ist, um einen neuen Klienten anzunehmen, daher hat er den State „istBeschäftigt“(siehe Abb. 4.7). Die Schaffung eines States erfolgt durch das Schlüsselwort „state“ und den Namen des States. Ein State kann entweder den Wert „wahr“ oder „falsch“annehmen.

```
package RolePackage;

Roletype Beratungslehrer{

    state istBeschäftigt;

    task GutachtenErstellen(Klient, Termin);

    message benötigeBeratung(Klient);
    message Terminvorschlag(Termin);
}
```

Abbildung 4.7: Definition des Roletypes Beratungslehrer

Eine Role kann, in Abhängigkeit des Roletypes, verschiedene Tätigkeiten (Tasks) ausführen (siehe 4.7.2). Bei einer Task führt die Role etwas aus, ohne dabei andere Roles zu kontaktieren. Oder aber die Kommunikation, die während einer Task abläuft, ist für die Modellierung des Geschäftsprozesses nicht relevant. Diese Tasks werden nach den States aufgeführt und werden durch das Schlüsselwort „task“ und den Namen der Tätigkeit erschaffen. Der Name der Task sollte kurz aussagen, worin die Tätigkeit besteht. Hinter der Tätigkeit können Parametertypen in Klammer angegeben werden. Dabei wird jeweils angegeben, von welchem Typ das Objekt, das bei einer Task übergeben wird, sein muss. Diese Objekte sind Informationen oder Objekte, die die Role für seine Task benötigt. Die Parametertypen werden durch ein Komma voneinander getrennt.

Als nächstes werden die Nachrichten aufgeführt, die eine Role bekommen kann. Diese Nachrichten werden später für die Kommunikation zwischen den einzelnen Roles benötigt (siehe 4.7.1). Eine Nachricht wird durch das Schlüsselwort „message“ und den anschließenden Namen der Nachricht erstellt. Der Name der Message sollte wie der Betreff einer E-Mail sein und möglichst kurz den Kern der Nachricht aussagen. Wenn in einer Nachricht

irgendwelche sonstigen Informationen mitgeteilt werden, können diese in Klammern hinter dem Namen als Parameter mitgegeben werden. So folgt hinter dem Namen, wie bei der Angabe einer Task, welche Parametertypen bei einer Message als Informationen mitgegeben werden können. Im Beispiel (siehe Abb. 4.7) kann der Role Beratungslehrer ein Terminvorschlag gemacht werden. Dabei werden Daten, die vom Typ Termin sind, bei der Nachricht mitgegeben. Zudem kann die Role Beratungslehrer ein Gutachten zu einem Klienten erstellen.

Da ich mich bei meinen weiteren Erklärungen, auch auf den Klienten beziehe, gibt es auch diesen Roletype.

Die Definition eines Roletypes habe ich auch nochmal als Klassendiagramm (siehe Abb. 4.8) dargestellt. Daran wird deutlich, dass ein Roletype aus States, Tasks und Messages besteht und dass bei Messages und Tasks Parameter angeben werden können.

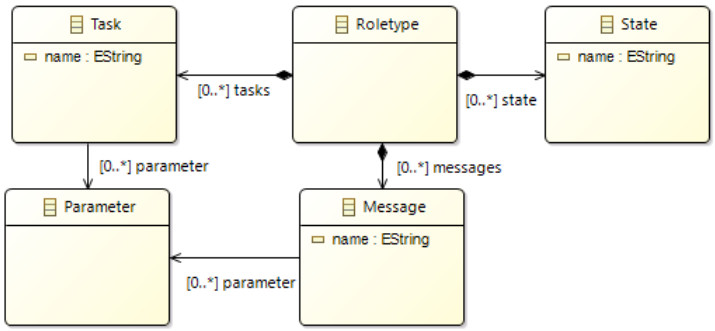

Abbildung 4.8: Klassendiagramm zur Definition eines Roletypes

4.3.2 Die Definition eines Systemtypes

Ein Systemtype kann beliebig viele Funktionen (Function) haben. Eine neue Function wird mit dem Schlüsselwort „function“ erzeugt. Darauf folgt der Name der Function. Dieser soll auch hier möglichst kurz und präzise aussagen, welche Tätigkeit an dem System durchgeführt wird. In Klammern können auch bei den Functions Parametertypen angegeben werden.

```
package SystemPackage;

Systemtype Outlook{

    function FreierTermin(Beratungslehrer, Beratungslehrer) returns Termin;
    function terminEingeben(Termin);
}
```

Abbildung 4.9: Definition des Systemtypes Outlook

Ein System wird in meiner Protokollsprache als eine Blackbox betrachtet. Daher werden die möglichen Eingaben in ein System in zwei Typen aufgeteilt. Beim ersten Typ handelt es sich um Eingaben, die einen Rückgabewert haben. Das ist immer dann der Fall, wenn aus einem System Informationen entnommen werden sollen. Zu diesen Functions gehört auch, wenn das System eine Eingabe bearbeitet und daraufhin ein Ergebnis oder Informationen ausgibt. Functions dieser Art werden durch ein „returns“ hinter der Parameterangabe gekennzeichnet. Dieser Rückgabewert wird Return-Statement genannt und dabei wird der Typ dieses Rückgabewertes angegeben. Ein besonderer Rückgabetyp ist ein Boolean. Bei diesem Rückgabetyp gibt die Function nur „wahr “ oder „falsch “ zurück.

In diesem Beispiel (siehe Abb. 4.9) wurde der Systemtype Outlook erstellt. Bei Outlook kann nach einem freien Termin gesucht werden und es können Termine eingegeben werden. Bei der Function „FreierTermin“ werden zwei Roles vom Typ Beratungslehrer übergeben und für beide nach einem freien Termin gesucht. Den freien Termin gibt die Function anschließend zurück.

Bei der zweiten Art von Functions handelt es sich um das reine Eingeben von Informationen in ein System. Bei diesen Eingaben wird kein Rückgabewert benötigt, daher gibt es bei der Function „terminEingegeben“ kein Return-Statement.

Bei beiden Function-Arten können Parameter mitgegeben werden, daher werden bei der Definition einer Function zusätzlich Parametertypen angegeben. Diese Parameter sind Informationen, die für die Systemeingabe nötig sind.

Das Klassendiagramm der Definition eines Systemtypes ist ähnlich zu dem der Roletype (siehe Abb. 4.8), daher lasse ich das hier weg.

4.3.3 Die Definition eines Datatypes

Wie ein Datentyp genau beschaffen ist, ist nicht relevant. In diesem Fall (siehe Abb. 4.10) handelt es sich bei dem Datentyp um einen Termin.

```
package DataPackage;

Datatype Termin{

}
```

Abbildung 4.10: Definition des Datatypes Termin

4.4 Die Definition eines Channels

Mit dem Schlüsselwort „Channel“ beginnt die Definition eines neuen Channels. Dahinter folgt der Name des Channels, der auch hier mit einem Großbuchstaben beginnen muss. Mit geschweiften Klammern beginnt und endet die Definition. Vor dem Schlüsselwort kann noch optional festgelegt werden, ob ein Channel synchron oder asynchron sein soll. Das bedeutet, bei einem synchronen Channel erfolgt das Senden und Empfangen zur gleichen Zeit. Das ist zum Beispiel bei Telefonen oder Videochats der Fall. Bei einem asynchronen Channel erfolgt das Senden und Empfangen nicht zu gleichen Zeit. Empfangen bedeutet in diesem Zusammenhang, dass der Empfänger die Nachricht wahrnimmt und verarbeitet hat. Beispiel für asynchrone Channels sind Briefe und E-Mails. Daneben gibt es keine weiteren Attribute, die ein Channel haben kann.

```
package ChannelPackage;

Channel Telefon{

}
```

Abbildung 4.11: Definition des Channels Telefon

In diesem Beispiel (siehe Abb. 4.11), wurde das Telefon als einziges Kommunikationsmittel erstellt. Das Telefon ist, wie bereits erwähnt, ein synchroner Channel.

4.5 Der Businessprocess

Wie im Abschnitt Interviews 4.2 bereits erwähnt, sind die Prozesse der eigentliche Kern meiner Protokollsprache. Ein Interview kann beliebig viele Prozesse enthalten. Ein neuer Prozess wird durch das Schlüsselwort „process“ gekennzeichnet. Dahinter folgt der Name des Prozesses als String. Der Prozess endet und beginnt ebenfalls mit geschweiften Klammern.

Meine Sprache beruht auf dem Konzept von LSCs (siehe Kap. 2), daher muss es in jedem Prozess ein Pre-und Mainchart geben. Im Prechart wird ein Szenario beschrieben, das den Prozess auslöst und im Mainchart, wird das Szenario beschrieben, das daraufhin passiert. Um diese Unterscheidung etwas einfacher zu machen, habe ich mich entschieden, das Prechart „Start“ und das Mainchart „Followed“ zu nennen. Die beiden Abschnitte Start und Followed müssen für jeden Prozess mindestens einmal definiert werden (siehe 4.6).

Wie in der Abbildung 4.12 zu sehen ist, wird Start mit dem Schlüsselwort „start“ begonnen und Followed mit dem Schlüsselwort „followed“. Die Blöcke

```
process "Klient benötigt Beratung"{

    start{
        // Start-Szenario
    } followed{
        // Folge-Szenario
    }

}
```

Abbildung 4.12: Der Prozess mit Start und Followed

von Start und Followed beginnen und enden jeweils mit einer geschweiften Klammer.

In den nächsten Abschnitten stelle ich die weiteren Elemente vor, die in einem Prozess verwendet werden können. Dabei können in Start nur Worksteps und alternative Abläufe verwendet werden.

4.6 Alternative Abläufe

In diesem Abschnitt stelle ich dar, wie die alternativen Abläufe in meiner Protokollsprache dargestellt werden. Diese sind wichtig, da es in einem Geschäftsprozess nicht immer nur genau einen möglichen Ablauf gibt.

Bei den alternativen Abläufen ist zu beachten, dass es grundsätzlich zwei Arten gibt. Zum einen gibt es die durch eine Auftrittswahrscheinlichkeit gewichtete Alternative und die ungewichtete Alternative. Zweitens ist es wichtig, darauf zu achten, wann diese Alternativen eingesetzt werden. Daher werde ich zuerst die beiden Arten der Alternativen allgemein vorstellen und danach auf die Benutzung der beiden Konstrukte eingehen.

4.6.1 Ungewichtete alternative Abläufe

Bei ungewichteten alternativen Abläufen sind die einzelnen Ablaufmöglichkeiten gleich wahrscheinlich. Sind die Ablaufmöglichkeiten jedoch nicht gleich wahrscheinlich, werden ungewichtete alternative Abläufe genutzt.

Um anzugeben, dass an einer Stelle ungewichtete Alternativen folgen, wird das Schlüsselwort „alt“ genutzt. Ein möglicher Ablauf wird dann von zwei geschweiften Klammern eingegrenzt. Dahinter muss mindestens eine zweite Alternative folgen, die durch das Schlüsselwort „or“ erstellt wird. Auch dieser Abschnitt beginnt und endet wieder mit geschweiften Klammern. Gibt es noch weitere Alternativen, werden diese, wie im Beispiel (siehe Abb. 4.13), angehängt.

```
alt{
  // Szenario 1
}
or{
  // Szenario 2
}
or{
  // Szenario 3
}
```

Abbildung 4.13: Ein ungewichteter alternativer Ablauf

4.6.2 Gewichtete alternative Abläufe

Ein gewichteter alternativer Ablauf zeichnet sich dadurch aus, dass der Interviewte angibt, dass es sich bei einem Ablauf um den Regelfall handelt oder etwas meistens passiert. Das bedeutet, dass hier ein Ablauf wahrscheinlicher ist als der andere.

```
mostly{
    // Szenario 1
}
sometimes{
    // Szenario 2
}
sometimes{
    // Szenario 3
}
```

Abbildung 4.14: Ein gewichteter alternativer Ablauf mit *Mostly* und *Sometimes*

Eine Verzweigung in gewichtete alternative Abläufe beginnt mit dem Schlüsselwort „mostly“. Hier wird, wie der Name schon sagt, angegeben, was meistens passiert, beziehungsweise wie der Regelfall aussieht. Dieser Abschnitt wird, wie bei den ungewichteten Abläufen mit geschweiften Klammern eingefasst. Danach muss mindestens eine zweite Alternative folgen, die durch das Schlüsselwort „sometimes“ begonnen wird und ebenfalls durch geschweifte Klammern eingefasst wird.

Da es meistens einen Regelfall und mehrere weitere Alternativen gibt, die jeweils manchmal passieren, können auch hier noch weitere mögliche Abläufe folgen, die, wie im Beispiel(siehe Abb. 4.14) zu sehen ist , jeweils die Wahrscheinlichkeit „manchmal“ haben. Um die Sprache nicht zu kompliziert zu gestalten, gibt es nur die Unterscheidungsmöglichkeit in *Mostly* und *Sometimes*. Bei gewichteten alternativen Abläufen kann es keine zwei Mostly-

Pfade geben.

4.6.3 Zusammenspiel von gewichteten und ungewichteten alternativen Abläufen

Obwohl es bei gewichteten alternativen Abläufen keine zwei Mostly-Pfade gibt, kann trotzdem folgender Fall eintreten, in dem der Interviewte sagt:

- „Meistens gibt es Szenario1 und Szenario2, aber manchmal passiert auch Szenario3."

Um dieses Beispiel modellieren zu können, müssen ungewichtete und gewichtete Alternativen ineinander geschachtelt werden.

```
mostly{
    alt{
      // Szenario1
    }
    or{
      // Szenario2
    }
}
sometimes{
    // Szenario3
}
```

Abbildung 4.15: Verkettung von gewichteten und ungewichteten alternativen Abläufen

Zuerst wird, wie in der Abbildung 4.15 zu sehen ist, ein gewichteter alternativer Ablauf erstellt. Da Szenario3 nur manchmal passiert, steht es im Sometimes-Block. Um nun zu zeigen, dass Szenario1 und Szenario2 beide meistens passieren, wird im Mostly-Block ein ungewichteter alternativer Ablauf erstellt.

Beide Konstrukte können syntaktisch frei ineinander geschachtelt werden, trotzdem gibt es Fälle, die durch die Semantik ausgeschlossen werden. Die Semantik meiner Sprache behandele ich in einem späteren Abschnitt. Dort behandele ich auch die Verbindung von alternativen Abläufen mit *Start*.

4.7 Worksteps

Worksteps sind in meiner Protokollsprache Activities, Communications und Systemworks. Diese Ereignisse haben zwei Dinge gemeinsam, die *execution kind* und die *importance*. Die *execution kind* sagt aus, ob die Role den Workstep ausführen muss oder nicht. Die *execution kind* kann entweder *must*

oder *can* sein. Wenn die Person die Tätigkeit ausführen muss, dann wird die *execution kind* auf *must* gesetzt, sonst auf *can*

Mit der *importance* wird angegeben, ob eine Tätigkeit für die Weiterführung eines Prozesses nötig ist. Eine Tätigkeit kann *important* oder *unimportant* sein. Ist die Tätigkeit nicht wichtig für die Weiterführung eines Prozesses, also *unimportant*, wird das durch das Schlüsselwort „unim“ ausgedrückt. Wenn der Workstep ausgeführt werden muss, damit der Prozess fortgesetzt werden kann, ist die *importance important*. Ein *important* Workstep, wird mit dem Schlüsselwort „im“ ausgedrückt. Bei einem Workstep, der die *execution kind must* hat, ist die *importance* automatisch *important*. Die Angabe der *importance* kann daher dort weggelassen werden. Die genauere Bedeutung der *execution kind* und der *importance* behandele ich im Kapitel 5, Semantik.

Jeder Workstep wird mit einem Semikolon beendet.

4.7.1 Communication

Bei einer Communication geht es um eine Nachricht, die von einer Role zu einer anderen geschickt wird. Die wichtigen Bestandteile einer Communication sind der Sender (Seder), der Empfänger (Receiver) und die Nachricht (Message).

```
im klient can send to beratungslehrer.Terminvorschlag(termin) via Telefon;

beratungslehrer must send to klient.terminBestätigt() via Telefon;
```

Abbildung 4.16: Beispiel für zwei Communications

In diesem Beispiel (siehe Abb. 4.16) handelt es sich um zwei Communications. Bei der ersten kann die Role „klient“ der Role „beratungslehrer“ einen Terminvorschlag machen und schickt dabei als Information einen „termin“ mit. Diese Communication ist wichtig, um den Prozess fortzusetzen. Die Role „beratungslehrer“ muss danach den Termin bestätigen und schickt diese Nachricht an die Role „klient“ zurück. Beide Communications laufen über den Channel Telefon.

Zuerst wird bei einer Communication die *importance* angegeben. Daraufhin folgt der Sender der Message, der eine Role oder ein System sein kann. Heutzutage werden oft vorgefertigte E-Mails abgeschickt. Hinter der Angabe des Senders folgt die *execution kind*. Um anzuzeigen, dass es sich bei diesem Workstep um eine Communication handelt, folgen nach der Angabe der *execution kind* die beiden Schlüsselwörter „send“ und „to“

Anschließend folgt der Receiver der Message, der nur eine Role sein darf. Dabei dürfen Sender und Receiver nicht gleich sein. Hinter dem Receiver wird, durch einen Punkt abgetrennt, die Message der Communication angegeben. Damit diese Message im Interview auftreten kann, muss sie zuvor

beim Roletype des Receivers als Message definiert werden. Wenn bei der Definition dieser Message Parameter angegeben wurden, müssen diese auch bei der Communication angegeben werden. Hinter der Message kann optional der Channel folgen, über den die Communication läuft. Dafür wird das Schlüsselwort „via“ angegeben, hinter dem der Name des genutzten Channel steht.

Eine Communication wird eindeutig über die Role des Senders, die Role des Receivers, die Message und die mitgegebenen Parameter identifiziert.

Dieser Zusammenhang wird auch durch das Klassendiagramm (siehe Abb. 4.17) deutlich.

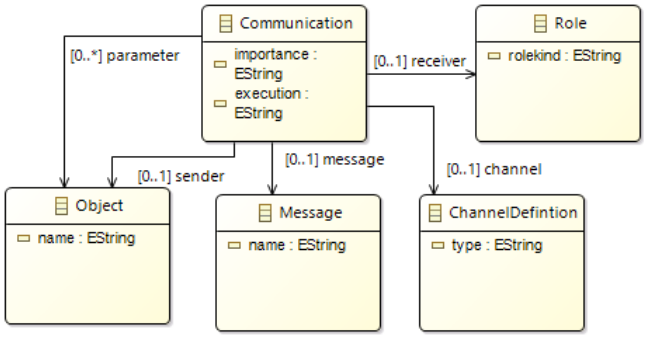

Abbildung 4.17: Klassendiagramm der Regel Communication

4.7.2 Activity

Bei einer Activity führt die Person eine Tätigkeit (Task) aus, bei der sie nicht mit einer anderen Role kommuniziert oder diese Kommunikation ist für die Modellierung des Prozesses unwichtig. Um ein Gutachten über einen Klienten zu erstellen, muss der Beratungslehrer mit dem Klienten reden, aber für die Modellierung der Geschäftsprozesse in dieser Institution ist der Inhalt dieser Gespräche nicht wichtig.

```
unim beratungslehrer can do GutachtenErstellen(klient,abgabetermin);
```

Abbildung 4.18: Beispiel für eine Activity

In diesem Beispiel (siehe Abb.4.18) kann die Role „beratungslehrer “ ein Gutachten über einem Klienten erstellen, wenn er dies zum Beispiel als nötig erachtet. Der Prozess kann aber auch ohne dieses Gutachten fortgesetzt werden. Dieses Gutachten soll zu einem bestimmten Termin fertig sein.

Zu Beginn einer Activity wird zuerst die *importance* der Activity angegeben. Dahinter folgt die Role, die die Task ausführt. Dahinter folgt die *execution kind*. Das folgende Schlüsselwort „do“ zeigt, dass es sich hier nicht

um eine Communication sondern um eine Activity handelt. Am Ende muss noch die Task folgen, die die Role ausführen soll. Diese Task muss vorher beim Roletype der ausführenden Role definiert werden. Sind dort Parameter angegeben, so müssen diese auch bei der Activity übergeben werden.

Eine Activity wird eindeutig über die ausführende Role, die Task und die Parameter, die bei der Task übergeben werden, identifiziert.

Dieser Zusammenhang wird auch durch das Klassendiagramm (siehe Abb. 4.19) deutlich.

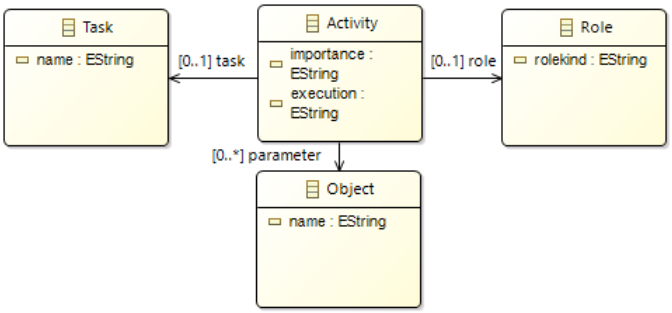

Abbildung 4.19: Klassendiagramm der Regel Activity

4.7.3 Systemwork

Bei Systemworks handelt es sich um Aktionen, die eine Person an einem System durchführen kann. Im Abschnitt 4.3.2 habe ich bereits die beiden Arten von Functions vorgestellt.

```
termin = beratungslehrer must systemwork
outlook.FreierTermin(beratungslehrer, kollege);

beratungslehrer must systemwork outlook.terminEingeben(termin);
```

Abbildung 4.20: Beispiel für zwei Systemworks

Im Beispiel (siehe Abb. 4.20) kommen beide Arten von Functions zum Einsatz. Bei der ersten Systemwork sucht die Role „beratungslehrer“ in Outlook einen freien Termin für sich und seinen Kollegen. Die Function „FreierTermin“ gibt einen möglichen freien Termin zurück. Dieser wird in der Data „termin “ abgelegt. Bei der zweiten Systemwork wird dieser „termin“ von der Role „beratungslehrer“ bei Outlook eingetragen.

Je nachdem, ob eine Function einen Rückgabewert besitzt, kann dieser in einem Objekt passenden Typs abgelegt werden. Ein Rückgabewert muss aber nicht abgelegt werden, falls dieser nicht weiter verwendet wird. Vor der Role, die die Systemwork ausführt, wird die *importance* angegeben. Hinter

der Role folgt die *execution kind* und das Schlüsselwort „systemwork“. Hinter dem Schlüsselwort muss das System und, abgetrennt durch einen Punkt, die Function, die die Role nutzen möchte, folgen. Wenn die Definition der Function Parameter vorsieht, müssen diese hier angegeben werden.

Eine Systemwork wird eindeutig durch die ausführende Role, das verwendete System, die Function und die benutzen Parameter identifiziert.

Dieser Zusammenhang wird auch nochmal deutlich durch das Klassendiagramm (siehe Abb. 4.21).

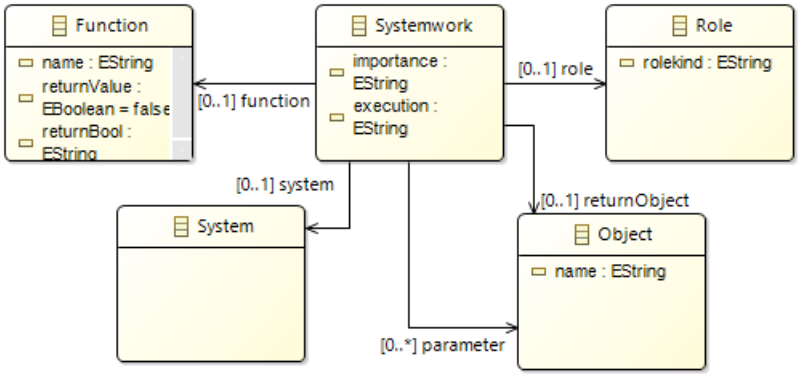

Abbildung 4.21: Klassendiagramm der Regel Systemwork

4.8 IF-Anweisung

Der Ablauf eines Prozesses ist manchmal von Bedingungen abhängig. Bei einer IF-Anweisung wird eine Bedingung angegeben, hinter der ein Szenario folgt. Die Bedingung ist vom Typ Boolean, kann also zu wahr oder falsch aufgelöst werden. Wenn die Bedingung zu wahr gelöst wird, folgt als nächstes das Szenario, das als Block hinter der Bedingung angegeben ist. Dieser Block wird durch geschweifte Klammern eingefasst. Wenn die Bedingung zu falsch gelöst wird, dann wird das angegebene Szenario übersprungen und der Prozess wird hinter der IF-Anweisung fortgesetzt.

```
if(Bedingung){
    // Bedingtes Szenario
}
```

Abbildung 4.22: Beispiel für eine allgemeine IF-Anweisung

Eine IF-Anweisung erfolgt durch das Schlüsselwort „if“. Dahinter wird die Bedingung in Klammer angegeben, gefolgt von dem Szenario in geschweiften Klammern.

Es gibt drei Arten von Bedingungen, die ich hier nacheinander vorstellen werde. Dazu gehört auch, wie die verschiedenen Bedingungen miteinander verknüpft werden können.

4.8.1 Zustandsbedingung

Der Zustand einer Person kann eine Bedingung sein. Im Beispiel (siehe Abb. 4.23) wird abgefragt, ob die Role „beratungslehrer“ beschäftigt ist.

```
if(beratungslehrer.istBeschäftigt is true){
    // Bedingtes Szenario
}
```

Abbildung 4.23: Beispiel für eine IF-Anweisung mit der Abfrage eines Zustandes

In den Klammern wird zuerst die Person angegeben. Abgetrennt durch einen Punkt wird auf den Zustand, der in der Role definiert wurde, zugegriffen. Das „is true“ kann weggelassen werden, da in der Bedingung immer Ausdrücke vom Typ Boolean verwendet werden. Der Zustand „ist-Beschäftigt“ hat bereits den Wert wahr oder falsch, daher ist es unnötig abzufragen, ob er wahr ist. Soll das Szenario folgen, wenn der Beratungslehrer nicht in dem Zustand ist, dann muss hinter dem „istBeschäfigt“ jedoch noch ein „is false“ folgen, da so die Bedingung wahr wird. Alternativ kann, wie in der Abbildung 4.24 zu sehen ist, vor den Ausdruck auch ein Ausrufezeichen gesetzt werden. Dadurch wird der Wert negiert, sodass aus wahr falsch und aus falsch wahr wird.

```
if(!beratungslehrer.istBeschäftigt){
    // Bedingtes Szenario
}
```

Abbildung 4.24: Beispiel für eine IF-Anweisung mit der Abfrage eines Zustandes

4.8.2 Vergleich zweier Objekte

Zwei Objekte, die in einem Interview vorkommen, können in einer Bedingung miteinander verglichen werden. Hier können immer zwei Objekte des gleichen Typs verglichen werden. Also kann zum Beispiel eine Data nur mit einer Data verglichen werden aber nicht eine Data mit einer Role. Im Beispiel (siehe Abb. 4.25) wird überprüft, ob der Terminvorschlag des Klienten gleich dem des Beratungslehrers ist.

Bei dem Vergleich zweier Objekte, wird zuerst das eine Objekt genannt und danach folgt ein Vergleichszeichen. Ein doppeltes Gleichheitszeichen sagt

```
if(terminvorschlag_Klient == terminvorschlag_Beratungslehrer){
    // Bedingtes Szenario
}
```

Abbildung 4.25: Beispiel für eine IF-Anweisung mit dem Vergleich zweier Objekte

aus, dass der Ausdruck wahr wird, wenn beide Objekte gleich sind. Mit einem Ausrufezeichen vor einem Gleichheitszeichen (!=) wird ausgesagt, dass der Ausdruck wahr wird, wenn beide Objekte nicht gleich sind.

4.8.3 Boolean-Bedingung

Bei einer Bedingung kann auch nur der Wert eines Objektes vom Typ Booelan abgefragt werden. Das funktioniert genauso, wie bei der Zustandsbedingung, daher gehe ich hier nicht weiter darauf ein.

```
if(klientIstBekannt){
    // Bedingtes Szenario
}
```

Abbildung 4.26: Beispiel für eine IF-Anweisung mit der Abfrage eines Boolean

Im Beispiel (siehe Abb. 4.26) wird abgefragt ob ein Klient bereits bekannt ist, oder nicht.

4.8.4 Verkettung von Bedingungen

Natürlich ist es ineffizient, wenn bei einer IF-Anweisung immer nur eine der drei oben vorgestellten Bedingungen geprüft werden kann. Daher können diese auch verkettet werden. Dafür gibt es die Und– und Oder-Verknüpfung. Wenn zwei Bedingungen mit Und verknüpft sind, müssen beide Bedingungen wahr sein, damit der gesamte Ausdruck wahr ist. Bei einer Oder-Verknüpfung muss nur einer der beiden Bedingungen wahr sein, damit der gesamte Ausdruck wahr wird. Eine Und-Verknüpfung wird durch ein doppeltes Und-Zeichen (&&) dargestellt und eine Oder-Verknüpfung durch zwei senkrechte Striche (||) (siehe Abb. 4.27).

Natürlich können auch mehr als zwei Bedingungen miteinander verknüpft werden. Hierbei ist es wichtig darauf zu achten, dass eine Negation am stärksten bindet. Das Und bindet am zweitstärksten und das Oder am drittstärksten. Das lässt sich mit der Rechnung von Plus und Minus vergleichen, denn dort erfolgt Punkt- vor Strich-Rechnung. Bei der Rechnung von 1 + 2 * 3 wird zuerst 2 * 3 zu 6 multipliziert und danach zu 6 die 1

```
boolean a;
boolean b;
boolean c;


if(a && b){
    // Bedingtes Szenario
}

if(a || b){
    // Bedingtes Szenario
}
```

Abbildung 4.27: Beispiel für eine IF-Anweisung mit einer Und- und einer Oder-Verknüpfung

addiert. Daher werden zuerst die Negationen, dann die Und-Verknüpfungen und zuletzt die Oder-Verknüpfungen ausgewertet. Natürlich ist es schwierig dabei den Überblick zu behalten. Daher können die einzelnen Ausdrücke wie bei der Rechnung von Plus und Minus in Klammern gefasst werden. Die beiden IF-Anweisungen (siehe Abb.4.28 sagen daher das gleiche aus.

```
if((a && b) || c){
    // Bedingtes Szenario
}

if(a && b || c){
    // Bedingtes Szenario
}
```

Abbildung 4.28: Beispiel für das Zusammenspiel von Und und Oder

```
if(!a && (b || c)){
    // Bedingtes Szenario
}
```

Abbildung 4.29: Zusätzliches Beispiel für das Zusammenspiel von Und und Oder mit einer Negation

Die letzte Anweisung (siehe Abb. 4.29) sagt aus, dass zuerst die Bedingung (b || c) ausgewertet wird und danach diese Bedingung mit nicht a und-verknüpft wird.

4.8.5 Else-Anweisung

Bei einer IF-Anweisung gibt es die Möglichkeit neben dem Szenario, das eintritt, wenn die Bedingung zu wahr ausgewertet wird, ein zweites Szenario anzugeben. Dieses Szenario wird ausgeführt, wenn die Bedingung zu falsch ausgewertet wird (siehe Abb. 4.30).

```
if(a is true){
    // Szenario1, wenn a wahr ist
}else{
    // Szenario, wenn a falsch ist
}
```

Abbildung 4.30: Ein Beispiel für eine IF-Anweisung mit Else-Block

Dieses zweite Szenario wird durch das Schlüsselwort „else“ angezeigt und wird ebenfalls durch geschweifte Klammern eingefasst.

4.9 Loop

Neben den IF-Anweisungen und alternativen Abläufen können Schleifen eingesetzt werden. Diese zeigen an, dass ein gewisser Ablauf im Prozess mehrfach wiederholt wird. Dabei gibt es zwei Arten von Schleifen, die bestimmte und die unbestimmte. Bei einer bestimmten Schleife ist die Anzahl der Durchläufe festgelegt und bei einer unbestimmten nicht.

```
loop[3]{
    // dieses Szenario wird genau dreimal durchlaufen
}
```

Abbildung 4.31: Beispiel für eine Loop

Eine Schleife wird mit dem Schlüsselwort „loop“ begonnen. Dahinter folgt die Anzahl der Schleifendurchläufe in eckigen Klammern. Eine fest definierte Anzahl an Durchläufen wird durch eine natürliche Zahl, größer Null, ausgedrückt. Bei einem Sternsymbol [*] in den eckigen Klammern, ist die Anzahl der Durchläufe unbestimmt.

Eine Schleife kann an jeder Stelle mit dem Schlüsselwort „stop “abgebrochen werden.

Kapitel 5

Die Semantik meiner Protokollsprache

Dieses Kapitel hat nicht den Anspruch, die komplette Semantik meiner Sprache zu beschreiben. Trotzdem möchte ich hier einige grundsätzliche Ideen vorstellen und Stellen aufzeigen, die einer tieferen Untersuchung und Definition bedürfen.

Einer dieser Stellen ist das Verständnis einer Role in meiner Sprache. Bei einer Role handelt es sich um ein dynamisches Objekt. Das bedeutet, dass eine Role ein Platzhalter für eine tatsächliche Person ist. Diese Person nimmt die Role aber erst während der Laufzeit des Prozesses ein. Zum Beispiel gibt es in dem BL-Interview den Klienten. Das ist natürlich keine festgelegte Person und es bedeutet auch nicht, dass es nur einen Klienten gibt. Es können sich bei einem Beratungslehrer mehrere Klienten melden. Um die Semantik meiner Sprache besser erklären zu können, gehe ich davon aus, dass, ähnlich zu den LSCs, für jeden Klienten, der sich meldet eine Kopie des bestehenden Prozesses erstellt wird. In dieser Kopie ist die Role des Klienten dann direkt an die Person gebunden und die Role des Klienten heißt in jeder Kopie anders. Dieses Problem existiert natürlich nicht nur für die Roles, sondern auch für die Systems und Datas. Daher werden auch diese Objekte direkt mit real existierenden Objekten verknüpft.

Wenn eine Role schon zu Beginn und nicht erst zur Laufzeit als eine feste Person betrachtet wird, können die *execution kind* und die *importance* gut mit endlichen Automaten dargestellt werden. Für die Darstellung von dynamischen Objects muss noch ein besserer Formalismus gefunden werden. Eine Möglichkeit dafür wären vielleicht Petri-Netze. Dort können sich in einem Zustand mehrere Markierungen gleichzeitig befinden. Jeder dieser Markierungen wäre ein Klient, der sich meldet. Dieser Frage gehe ich hier jedoch nicht weiter nach.

Ein endlicher Automat besteht aus einer Menge von Zuständen, einem Anfangszustand, einem oder mehreren Endzuständen, einer Übergangsfunk-

tion und einem Eingabealphabet. Die Menge an Zuständen sind in meiner Protokollsprache die Stufen zwischen zwei Worksteps. Diese Stufen werden Cuts genannt. Diese Cuts werden von mir mit c_0 bis c_n benannt. Die Worksteps, die in einem Prozess vorkommen, sind daher das Eingabealphabet für den zum Prozess gehörigen Automaten. Die Übergangsfunktion werde ich immer zeichnen. Die Endzustände entsprechen Stellen im Prozess, bei denen dieser frühzeitig beendet sein kann. Um das Eingabealphabet zu vereinfachen, habe ich die einzelnen Worksteps mit den Kleinbuchstaben a bis z benannt. Wie die einzelnen Worksteps eindeutig identifiziert werden, habe ich bereits bei meiner Protokollsprache dargelegt (siehe 4.7).

Ich nutze zur Erklärung Communications, jedoch lässt sich die Semantik auch auf die beiden anderen Worksteps übertragen.

5.1 Die Semantik der *execution kind must*

Wenn ein Workstep die *execution kind must* hat, muss dieser Workstep auch durchgeführt werden. Daher ist ein Workstep mit der *execution kind must* auch automatisch *important*. An diesem kurzen Beispiel (siehe Abb. 5.1) wird die Semantik von *must* deutlich. Diese Abbildung beinhaltet auch noch andere wichtige Aspekte meiner Semantik, jedoch gehe ich darauf erst später ein. Bei diesem Beispiel konzentriere ich mich auf die Cuts c_1 und c_2

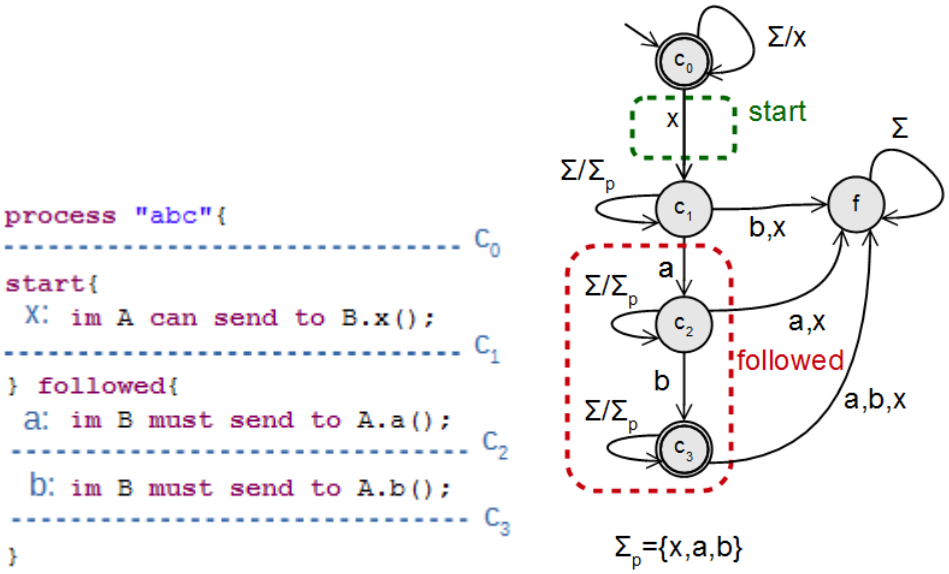

Abbildung 5.1: Die Semantik von *must*

In diesem Beispiel (siehe Abb. 5.1) muss, nachdem x passiert ist, auch a und b passieren. Am Anfang befindet sich der Prozess in Cut c_0. Das ist

sowohl der Startzustand des Prozesses, sowie auch ein Endzustand, da hier der Prozess noch nicht gestartet ist. Gestartet wird der Prozess erst durch den Workstep x. Da ich die genauere Semantik von Start in Abschnitt 5.4 noch erkläre, vernachlässige ich sie an dieser Stelle.

In einem Cut, auf den ein Workstep mit der *execution kind must* folgt, sind alle anderen Worksteps, die an anderer Stelle im Prozess erwartet werden, verboten. Alle anderen möglichen Worksteps, die in anderen Prozessen vorkommen, werden ignoriert und der Prozess verbleibt in dem gleichen Zustand. Wenn in einem Cut ein Workstep mit der *execution kind must* folgt, dann hat dieser Cut nur einen Folgezustand. Eine Ausnahme bilden hier die alternativen Abläufe, die Semantik dafür behandele ich in Abschnitt 5.3.

In c_1 wird a erwartet, jedoch können auch b oder x passieren, da diese aber verboten sind, führt das zum Fehlerzustand f, aus dem der Prozess nicht mehr herauskommt, egal welche Worksteps $\sum$ dort noch folgen. Die genaue Bedeutung des Fehlerzustandes habe ich noch nicht eingehender untersucht. Zur Zeit dient er eher dazu, Widersprüche aufzudecken (siehe Kap. 6). Bei einem Fehler kann nicht davon ausgegangen werden, dass die Geschäftsprozesse einer gesamten Firma zusammenbrechen. Durch einen Fehler ist die aktuelle Kopie des Prozesses verletzt und wird nicht weitergeführt. Wie mit einem Fehler umgegangen wird, muss noch geklärt werden.

Dadurch, dass a und b jeweils die *execution kind must* haben, ist der Prozess erst beendet, wenn sich der Prozess in c_3 befindet.

5.2 Die Semantik der *execution kind can*

Im Gegengensatz zu *must*, kann ein Workstep, der die *execution kid can* hat, durchgeführt werden, muss aber nicht. Bei der Semantik von *can* gibt es zwei Unterschiede, die durch die *importance* verursacht werden. Diese beiden Unterschiede erkläre ich in den folgenden beiden Abschnitten.

5.2.1 Die *importance unimportant*

Bei der *importance unimportant* ist der Workstep nicht wichtig, um den Prozess fortzusetzen. Die *importance* von Communication b ist auf *unimportant* gesetzt.

Wenn in einem Cut ein Workstep mit der *execution kind can* und der *importance unimportant* folgt, bedeutet es, dass hier mehrere Worksteps möglich sind. Das hängt davon ab, wie viele Worksteps mit der *importance unimportant* folgen. Alle anderen Worksteps, die in diesem Cut nicht möglich sind und an anderer Stelle im Prozess erwartet werden, sind verboten, beziehungsweise führen wieder zum Fehlerzustand f.

Im obigen Beispiel (siehe Abb.5.2) wird die Communication b nicht benötigt, um den Prozess fortzusetzen, daher kann dieser übersprungen

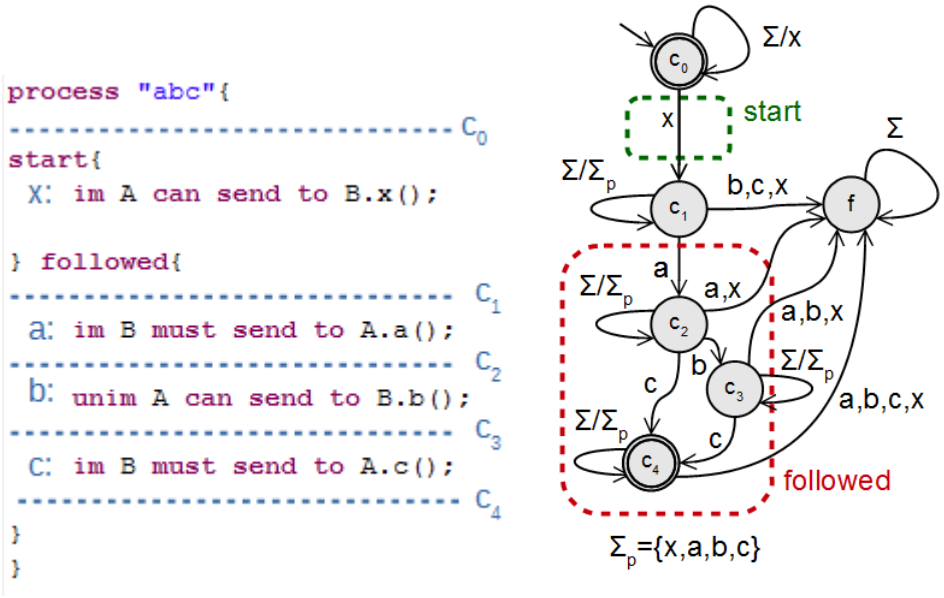

Abbildung 5.2: Die Semantik der *importance unimportant*

werden. Das bedeutet, dass im Zustand c_2 entweder b oder c passieren kann. Die Worksteps a und x führen jedoch zum Fehlerzustand f. Wenn c passiert, geht der Prozess direkt in den Zustand c_4, der ein akzeptierender Zustand ist. Falls in c_2 b passiert, geht der Prozess erst in den Zustand c_3. Da die Communication c *must* ist, muss in c_3 anschließend noch c folgen. Da nach dem *unimportant* Workstep b im Prozess der *must* Workstep c folgt, ist c_2 noch kein akzeptierender Zustand.

5.2.2 Die Semantik der *importance important*

Bei der *importance important* ist der Workstep wichtig, damit der Prozess fortgesetzt werden kann. Im Beispiel (siehe Abb. 5.3) ist die Communication b *can* und *important*.

Folgt in einem Zustand ein Workstep mit der *execution kind can* und der *importance important*, ist dieser bereits ein Endzustand, da der Prozess ohne diesen Workstep nicht fortgesetzt werden kann. Andere Worksteps, die im Prozess erwartet werden, aber an dieser Stelle nicht passieren können, sind verboten und führen zum Fehlerzustand f.

In diesem Beispiel 5.3 ist die Communication b *important*. Diese Communication könnte die Antwort eines Klienten sein und ohne diese Antwort kann der Prozess nicht durchgeführt werden. Der Klient ist jedoch nicht gezwungen zu antworten und kann zum Beispiel das Telefonat einfach beenden. Daher befindet sich der Prozess bereits bei c_2 in einem Endzustand. Wenn B jedoch antwortet, muss auch A mit c antworten. Der Prozess befindet sich dann bei

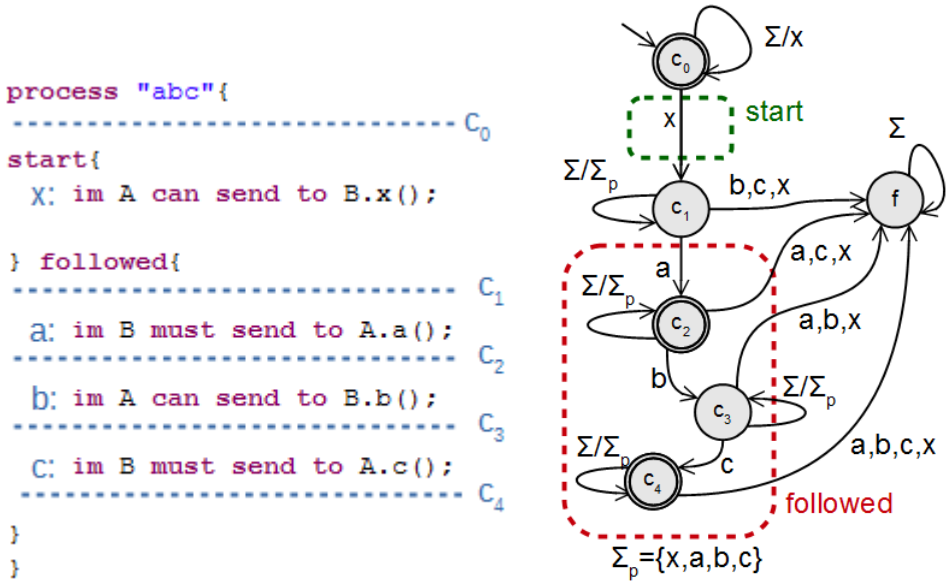

Abbildung 5.3: Die Semantik der *importance important*

c_4 in einem Endzustand. Da die Communication b wichtig für den Prozess ist, kann dieser nicht wie bei der *importance unimportant* übersprungen werden. Auch hier gilt, dass alle anderen Worksteps, die im Prozess vorkommen und hier nicht möglich sind, verboten sind und sonst zu einem Fehlerzustand führen.

An dieser Stelle möchte ich näher auf die Bedeutung eines Endzustandes für den Prozess eingehen. Ein Endzustand bedeutet grundsätzlich, dass dort ein Prozess beendet sein kann. Praktisch bedeutet es, dass durch das Weglassen der Communication b der Prozess in c_2 endet. Das Problem an dieser Stelle ist, dass in meiner Sprache nicht modelliert wird, ob absichtlich eine Kommunikation abgebrochen wird. Bei einer Überarbeitung meiner Sprache, die aber nicht Teil dieser Arbeit ist, müsste dafür ein Konzept entwickelt werden.

Um dieses Problem zu umgehen und trotzdem eine Grundsemantik zu bekommen, gehe ich davon aus, dass der Prozess in c_2 auf unbestimmte Zeit verharrt und auf das Eintreffen von b wartet. Da c_2 ein Endzustand ist, ist dieses Verharren möglich. Diese Definition behindert auch keine anderen Prozesse, da ich bereits am Anfang dieses Kapitels definiert habe, dass ich zum Beispiel eine Role nicht als dynamisches Objekt betrachte, sondern jede Role direkt an eine Person gebunden ist.

Diese Definition führt dazu, dass die Kopie eines Prozesses aus dem letzten möglichen Endzustand nicht mehr herauskommt. Es sei denn, es passiert etwas, was weiter oben im Prozess erwartet wird. Das ist jedoch unlogisch, da der gesamte Prozess bereits beendet ist. Daher führt das

ebenfalls zum Fehlerzustand f.

5.3 Die Semantik von alternativen Abläufen

Alternative Abläufe werden in den Automaten als eine Verzweigung dargestellt. Dabei ist es unwichtig, ob es sich bei dem alternativen Ablauf um einen gewichteten oder ungewichteten oder um eine IF-Anweisung handelt. Für einen Zustand, in dem als nächstes eine dieser Alternativen folgt, bedeutet dies, dass es dort mindestens zwei Folgezustände gibt.

Im Automaten lässt sich nicht aussagen, um welche Art der Alternative es sich handelt. Daher ist das Konzept des endlichen Automaten bei der Modellierung von alternativen Abläufen unzureichend. Eine Besonderheit meiner Sprache ist, dass in einem Prozess ein Regelfall angegeben werden kann, daher muss die Semantik von *mostly* und *sometimes* auch ausreichend dargestellt werden.

Ich habe bereits nach Modellierungstechniken recherchiert, habe aber hier noch keine zufriedenstellende Lösung gefunden.

Durch endliche Automaten können zudem auch keine Schleifen mit einer festen Durchlaufanzahl modelliert werden. Hinzu kommt, dass Schleifen an beliebigen Stellen abgebrochen werden können. Dies lässt sich ebenfalls nicht mit dem Konzept der endlichen Automaten darstellen. Aber ein Schleifenabbruch könnte ähnlich wie der Abbruch eines Prozesses funktionieren.

5.4 Die Semantik von Start und Followed

Nachdem ich die Semantik der *excution kind* und der *importance* beschrieben habe, ist es wichtig noch kurz die Semantik von Start und Followed zu erklären. Welche Worksteps zu Start und Followed gehören, ist in der Darstellung der Automaten durch die farblichen gestrichelten Linien gekennzeichnet.

Die grundlegende Bedeutung von Start und Followed ist, dass es ein auslösendes Szenario gibt, das aus einem oder mehreren Worksteps besteht. Dieses auslösende Szenario startet den Prozess. In Followed ist angegeben, was auf dieses auslösende Ereignis folgt. Jeder Workstep in Start und Followed muss die *execution kind can* und die *importance important* haben. *Can* müssen sie sein, da nicht automatisch davon ausgegangen werden kann, dass ein auslösendes Ereignis passiert. Die *importance* ist wichtig, da das Szenario genau so passieren muss und nicht anders. Die Worksteps, die zu Start gehören, sind bei der Darstellung der Automaten grün umrandet.

Warum die Worksteps in Start alle *important* sein müssen, wird am Beispiel (siehe Abb. 5.4) klar. Hier ist zu sehen, dass c_1 ein akzeptierender Zustand ist, da A zwar in c_1 bereits x gesendet hat aber y noch nicht. In c_1 führen auch alle Worksteps, außer y, wieder zu c_1. Das bedeutet,

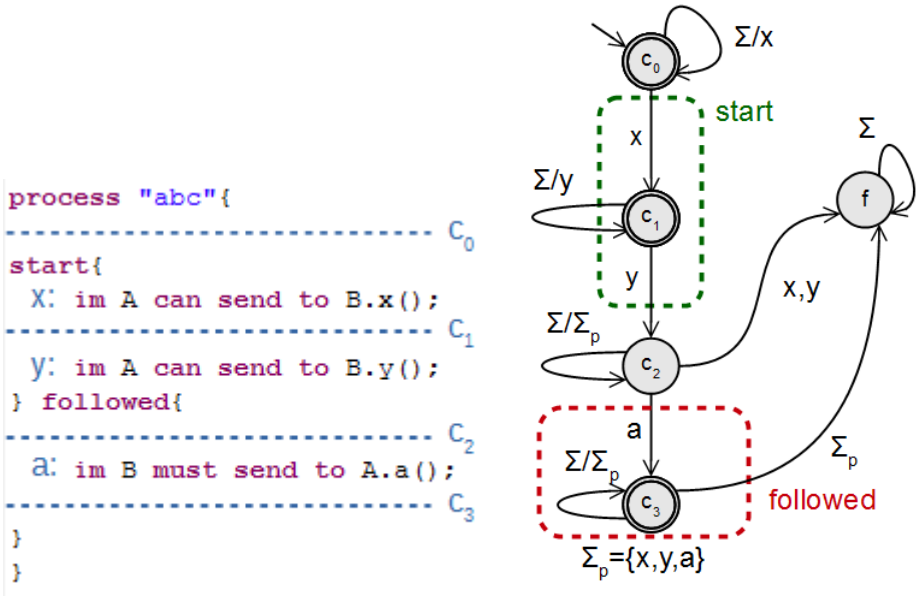

Abbildung 5.4: Die Semantik von Start und Followed

dass Start zur Zeit nicht verletzt werden kann. Ob und wie Start bei einem Geschäftsprozess verletzt werden kann, muss noch eingehender geprüft werden.

Der Prozess startet erst, wenn das gesamte Szenario passiert ist. In diesem Beispiel bedeutet das, erst wenn x und y passiert sind, startet der Prozess „abc “.

In Start können auch alternative Abläufe angegeben werden. Zum Beispiel um anzugeben, dass derselbe Prozess auf zwei Arten gestartet werden kann. Diese alternativen Abläufe haben in Start die gleiche Semantik, wie im vorherigen Abschnitt. Alle Worksteps müssen aber in den Alternativen *important* sein.

Kapitel 6

Widersprüche

Das übergeordnete Ziel meiner Arbeit ist, dass Widersprüche beim Führen der Interviews automatisch aufgedeckt werden können. Die beiden Grundlagen dafür sind eine formale Protokollsprache, die ich in den beiden vorherigen Kapiteln beschrieben habe und die Definition von Widersprüchen. In diesem Kapitel werde ich daher das Auftreten von Widersprüchen untersuchen. Dieses Kapitel hat, wie das vorherige, nicht den Anspruch das vollständige Gebiet der Widersprüche zu untersuchen. Stattdessen möchte ich einen Einblick geben, wie die Weiterarbeit an diesem Thema aussehen könnte und schon einige grundlegende Ideen dazu sammeln.

6.1 Widersprüche beim Durchlaufen der Prozesse

Als erstes können Widersprüche beim Durchlaufen der Prozesse auftreten. Mit Durchlauf ist gemeint, dass diese Prozesse auf dem Papier durchgespielt werden. Dabei kann es passieren, dass sich zwei oder mehrere Darstellungen eines Prozesses gegenseitig blockieren. Wie zwei Darstellungen sich blockieren, wird im folgenden Beispiel (siehe Abb. 6.1) deutlich.

```
process "abc"{

start{
    im A can send to B.x();

} followed{
    im B must send to A.a();
    im B must send to A.b();
}
}
```

```
process "abc"{

start{
    im A can send to B.x();

} followed{
    im B must send to A.b();
    im B must send to A.a();
}
}
```

Abbildung 6.1: Ein Beispiel für einen Widerspruch beim Durchlauf der Prozesse

Hierbei geht es um zwei interviewte Personen, die die gleiche Role B verkörpern. Im linken Interview hat der Interviewte im Prozess „abc“ angegeben, dass auf x immer erst a und dann b folgt. Im rechten Interview hat der Interviewte eine andere Reihenfolge angegeben.

In den endlichen Automaten wird angegeben, welche Worksteps in einem bestimmten Zustand folgen dürfen. Alle anderen Worksteps, die noch im Prozess erwartet werden, aber bei diesem Zustand nicht möglich sind, sind verboten. Für dieses Beispiel bedeutet dies, dass im linken Interview nach der Message x nur die Message a folgen kann und b an dieser Stelle verboten ist. Wenn die Message b doch gesendet wird, führt dies zu einem Fehler. Genau anders herum ist es beim rechten Interview, da darf nach x nur b folgen und a ist verboten. Beide Prozesse blockieren sich daher gegenseitig und es gibt kein weiteres mögliches Fortschreiten.

Es muss nicht immer sein, dass sich Prozesse so blockieren, dass es keine weiteren möglichen Worksteps gibt.Es kann in einem Zustand auch mehrere mögliche Worksteps geben. Ein Widerspruch liegt immer dann vor, wenn mindestens eine der möglichen Worksteps in einem Zustand durch einen anderen Prozess blockiert wird. Grund für ein solches Blockieren kann zum einen, wie oben im Beispiel, ein unterschiedlicher Ablauf der Worksteps sein. Ein anderer Grund dafür sind die Angabe unterschiedlicher *execution kinds* oder *importance*. Ob es sich bei zwei Worksteps um den gleichen Workstep handelt, hängt jedoch nicht von der *execution kind* oder der *importance* ab.

6.2 Sonstige Widersprüche

Im vorherigen Abschnitt ging es um Widersprüche, die direkt beim Durchspielen der Prozesse auftreten können. In diesem Kapitel geht es um Widersprüche, die keinen Fehler beim Durchspielen der Prozesse verursachen, aber trotzdem instinktiv als ein Widerspruch betrachtet werden. Um das deutlich zu machen, verwende ich auch hier wieder ein kleines Beispiel (siehe Abb. 6.2).

Auch hier geht es um zwei interviewte Personen. In den Interviews gibt es die Roles A, B und C. B ist in beiden Interviews die Mainrole und nimmt damit den Platz des Interviewten ein.

Im linken Interview berichtet B, dass sie nach Erhalt der Message x von A die Message a an A schickt. Im zweiten Interview berichtet B jedoch, dass sie nach der Message x von A, die Message a an C schickt. Beim Durchspielen der beiden Prozesse würde kein Widerspruch auftreten, da die beiden Communications unterschiedlich sind und es sich damit nicht um einen identischen Workstep handelt. Beide Communications können daher ohne Probleme ausgeführt werden. Trotzdem erscheint dieser Sachverhalt instinktiv als Widerspruch, da beim selben auslösenden Ereignis zwei unterschiedliche Worksteps passieren. Ein ähnlicher Widerspruch tritt bei

```
process "abc"{

start{
    im A can send to B.x();

} followed{
    im B must send to A.a();
}
}
```

```
process "abc"{

start{
    im A can send to B.x();

} followed{
    im B must send to C.a();
}
}
```

Abbildung 6.2: Ein Beispiel für einen Widerspruch

einer Communication auf, wenn zwar Sender und Receiver gleich sind, aber die Nachricht eine andere ist.

Bei den Worksteps in Abschnitt 4.7 bin ich bereits darauf eingegangen, wie diese eindeutig identifiziert werden. Für das Aufdecken von Widersprüchen müssem diese Regeln noch einmal verändert werden. Das liegt daran, dass in zwei unterschiedlichen Interviews nicht automatisch die gleichen Namen für eine Role verwendet werden. Natürlich können dabei immer nur zwei Worksteps gleichen Typs verglichen werden. Daher liste ich die Worksteps hier nochmal auf und erkläre, wie sie bei der Suche von Widersprüchen verglichen werden.

- **Communication:** Damit eine Communication gleich einer anderen Commuication ist, müssen die Objekttypen jeweils von beiden Sendern und beiden Receivern übereinstimmen. Außerdem müssen beide Messages der Communications übereinstimmen.

- **Activity:** Damit zwei Actitvties gleich sind, muss der Roletype beider ausführender Personen und die Task, die in beiden Activities ausgeführt wird, gleich sein.

- **Systemwork** Bei zwei Systemworks kommt es auch wieder auf den Roletype der ausführenden Person an. Auch müssen beide Systemtypes und die beiden Functions übereinstimmen. Bei einer Systemwork mit einem Returnargument ist es unwichtig, ob dieses verwendet wird und in welchem Object es abgelegt wird.

Neben den Widersprüchen, die durch einen falschen Workstep entstehen, können, gibt es noch weitere Widersprüche, die nicht durch eine Blockierung eines Worksteps verursacht werden. Auch das möchte ich an einem kurzen Beispiel (siehe Abb. 6.3) zeigen.

In diesem Fall geht es darum, dass beide Interviewte zwar die gleichen Worksteps beschreiben, aber sie zwei unterschiedliche Gewichtungen der Abläufe angeben. Im linken Interview wird berichtet, dass nach x meistens

```
process "abc"{

start{
    im A can send to B.x();

} followed{
    mostly{
     im B must send to A.a();
    } sometimes{
     im B must send to A.b();
    }
}
}
```

```
process "abc"{

start{
    im A can send to B.x();

} followed{
    mostly{
     im B must send to A.b();
    } sometimes{
     im B must send to A.a();
    }
}
}
```

Abbildung 6.3: Ein Beispiel für einen Widerspruch durch eine falsche Gewichtung

a und manchmal b passiert. Im rechten Interview wird jedoch beschrieben, dass nach x meistens b und manchmal a folgt. Nach x ist in beiden Fällen a oder b möglich, daher blockieren sich beide Prozesse nicht gegenseitig. Trotzdem ist auch hier instinktiv ein Widerspruch gegeben, da durch beide Interviews unklar ist, was von beiden denn nun wirklich meistens und was nur manchmal passiert.

In den vorangegangenen Beispielen, ging es immer um Widersprüche in *Followed*. Doch es können auch Fehler in *Start* auftreten. An diesem Beispiel (sishe Abb. 6.4) wird das deutlich:

```
process "abc"{

start{
    im A can send to B.x();

} followed{
     im B must send to A.a();
}
}
```

```
process "abc"{

start{
    im A can send to B.y();

} followed{
     im B must send to A.a();
}
}
```

Abbildung 6.4: Ein Beispiel für einen Widerspruch bei unterschiedlich auslösenden Szenarien

In *Followed* passieren in diesem Beispiel die gleichen Worksteps. Doch ist das auslösende Ereignis ein anderes. Bei Durchspielen dieser beiden Prozesse kommt es zu keinem Widerspruch, trotzdem muss hier davon ausgegangen werden, dass die beiden interviewten Personen sich hier widersprechen.

6.3 Das Aufdecken von Widersprüchen

An den beiden vorherigen Abschnitten wird deutlich, dass der Bereich der Widersprüche bei szenariobasierten Interviews sehr umfangreich ist und es eine Vielzahl von möglichen Widersprüchen gibt.

Eine grundlegende Idee, um Widersprüche aufzudecken, wäre die Darstellung eins zu eins miteinander abzugleichen. Doch das würde zu keinem befriedigenden Ergebnis führen, da diese Methode zwar viele Fehler finden würde, aber nicht alle. Zudem geht der Ansatz davon aus, dass zwei interviewte Personen genau das gleiche erzählen müssten, damit es keine Widersprüche gibt. Außerdem deckt diese Methode nicht auf, wenn sich zum Beispiel Personen widersprechen, die nicht die gleiche Role verkörpern und in unterschiedlichen Abteilungen arbeiten.

Daher wird hier eine andere Methode benötigt. Da ich ein Teil der Semantik meiner Sprache durch endliche Automaten darstelle, bietet sich die Methode der Produktkonstruktion von Automaten an. Am Beispiel (siehe Abb. 6.1) wird sehr gut deutlich, wie sich zwei Darstellungen blockieren können. Bei der Produktkonstruktion der beiden zugehörigen Automaten landet der Prozess insgesamt in einem Fehlerzustand.

Auch die Produktkonstruktion ist noch nicht die Lösung dieses Problems. Vielleicht könnte die Produktkonstruktion in Verbindung mit einem direkten Vergleich ein Teil der Lösung sein. Hier tut sich deutlich ein weiteres Forschungsgebiet auf, bei dem es noch sehr viel zu untersuchen gibt und andere Methoden entwickelt werden müssen.

Kapitel 7

Eine Methodik zum Führen szenariobasierter Interviews

In diesem Kapitel greife ich auf die Analyse aus Abschnitt 3.3 der beiden Interviews und die aufgelisteten Probleme zurück. Aus diesen Problemen habe ich ein Vorgehen beim Führen von szenariobasierten Interviews entwickelt.

7.1 Die Vorarbeit und der Beginn eines Interviews

Bevor Sie mit den Interviews in einer Firma beginnen, sollten Sie sich intensiv mit dieser auseinander setzen. Das ist wichtig, da in einer Firma viele Fachbegriffe und Abkürzungen genutzt werden. Da es bereits schwierig ist, sich auf das Interview zu konzentrieren, versuchen Sie schon im Vorfeld diese Begriffe zu lernen. Das vereinfacht das Führen der Interviews erheblich.

Vor dem Beginn des Interviews besprechen Sie kurz allgemein einige Dinge mit dem Interviewten. Dabei brauchen Sie ihm jedoch nicht zu erklären, was Sie genau für Interviews führen. In den meisten Fällen weiß er so wie so nicht, was damit gemeint ist. Wichtiger ist, dass Sie ihn durch geschickte Fragen in die richtige Richtung lenken. Sagen Sie dem Interviewten, dass er versuchen soll, in kurzen Sätzen zu antworten. Erklären Sie ihm auch, dass es Ihnen nicht um die geplante Ausführung von Prozessen, sondern um die tatsächlichen Abläufe geht. Nehmen Sie dem Interviewten die Angst vor Konsequenzen, wenn er erzählt, dass er etwas anders macht als vorgeschrieben ist. Sagen Sie ihm, dass dort das Potenzial für Verbesserungen liegt.

Fragen Sie den Interviewten zu Beginn, welchen Beruf er ausübt. Damit wissen Sie dann, welche Rolle der Interviewer einnimmt, sein ungefähres Betätigungsfeld und welche Position er in der Firma innehat. Diese Frage ist auch gut, um das Eis zu brechen, da der Interviewte nicht immer, weiß wie er anfangen soll. Aus dieser ersten Frage entwickeln sich dann meist schon die ersten Prozesse.

7.2 Der Verlauf des Interviews

Beim Verlauf eines Interviews ist es wichtig auf zwei Dinge zu achten. Welche das sind, erläutere ich in den nächsten beiden Abschnitten.

7.2.1 Überblick über das Interview

Das Problem beim Führen von Szenariobasierten Interviews ist, dass der Interviewer den Überblick über das Interview nicht verlieren darf und darauf achten muss, dass keine Informationen verloren gehen.

Um einen Informationsverlust zu vermeiden, lassen Sie den Interviewten nicht ins Erzählen geraten und unterbrechen Sie ihn zur Not. Wenn der Interviewte bereits eine längere Antwort gegeben hat, dann gehen Sie sie mit ihm noch einmal durch. Hier kann es helfen, wenn Sie das Gespräch aufnehmen und die entsprechende Textpassage abspielen. Es ist wichtig möglichst kleinschrittig durch die Prozesse zu gehen.

Wenn der Interviewte bei der Beschreibung eines Prozesses andere Prozesse oder alternative Abläufe erwähnt, unterbrechen Sie ihn auch dort und notieren Sie sich diese Informationen. Wenn die Beschreibung des Prozesses abgeschlossen ist, kommen Sie auf ihre Notizen zurück. Um das Interview nicht dauernd zu unterbrechen, sprechen Sie den Interviewten schon zu Beginn eines neuen Prozesses auf mögliche alternative Abläufe an. Achten Sie dabei auf eine Gewichtung der Alternativen und fragen Sie den Interviewten danach. Dabei kann es sein, dass zwei alternative Abläufe ohne Gewichtung nebeneinander existieren. Das bedeutet, dass beide Fälle gleich oft vorkommen. Möglich ist aber auch, dass die Alternativen durch eine Auftrittswahrscheinlichkeit gewichtet werden. Achten Sie daher auf Wörter, die eine Unterscheidung in Regelfälle und Nebenfälle deutlich macht.

Neben dieser Gewichtung ist es wichtig, wie stark die einzelnen Alternativen voneinander abweichen. Wenn Alternativen sich stark voneinander unterscheiden, ist es besser, erst die eine Alternative komplett zu erfassen und danach die nächste. Gibt es immer nur kleine Unterschiede, können beide Alternativen zusammen betrachtet werden und nur an den entsprechenden Stellen wird eine Unterscheidung vorgenommen. Natürlich kann es in einer Alternative zu weiteren Alternativen kommen, doch in diesem Fall wird die gleiche Herangehensweise genutzt.

Gehen Sie in Ihrem Protokoll gelegentlich zurück und erläutern Sie dieses dem Interviewten. So können Sie feststellen, ob Sie alles richtig protokolliert und verstanden haben. Manchmal fällt dem Interviewten dann noch etwas ein, das er oder Sie vergessen haben.

7.2.2 Die richtigen Fragen

Zu Beginn habe ich schon angesprochen, dass Sie den Interviewten mit Ihren Fragen auf dem richtigen Kurs halten müssen. Vermeiden Sie offene Fragen, die einen zu großen Raum für eine Antwort lassen. Nutzen Sie sie höchstens, um den Einstieg für den Interviewten zu erleichtern.

Formulieren Sie ihre Fragen so, dass nur eine kurze Antwort möglich ist. Gehen Sie möglichst kleinschrittig durch die Prozesse, sodass immer nur der nächste Schritt erfasst wird. Zu Beginn eines neuen Prozesse, müssen Sie natürlich darauf eingehen, wie dieser Prozess startet. Die folgenden Fragen helfen dabei, nur kurze Antworten zu bekommen:

- Welcher ist der nächste Schritt?
- Was genau machen Sie danach?
- Und dann?
- Wie fängt der Prozess an? Bitte Schritt für Schritt.

Achten Sie dabei immer darauf, ob es irgendwo alternative Abläufe gibt. Sie müssen mehrfach danach fragen, denn der Interviewte vergisst es schnell und achtet dann nicht mehr von selbst darauf. Daneben müssen Sie auch bei den Worksteps darauf achten welche *execution kind* und *importance* diese besitzen und jeweils danach fragen.

Kapitel 8

Der Prototyp meines Editors

In diesem Kapitel möchte ich meinen Editor vorstellen, den ich mit Hilfe des Eclipse Plugins Xtext erstellt habe. Dieser Editor ist in diesem Stadium lediglich ein Prototype und dient hauptsächlich dazu, meine Sprache zu testen. Das Xtext-Framework habe ich bereits in den Grundlagen vorgestellt.

Im Anhang A.3 findet sich meine Grammatik, die ich für meine Sprache bei Xtext erstellt habe. Da sich meine Arbeit eher mit dem theoretischen Konzept meiner Sprache beschäftigt gehe ich hier auf meinen Prototypen nicht ausführlich ein und stelle nur die Validation und den Proposal Provider kurz vor.

8.1 Validation

Xtext bietet eine Validation des vom Interviewer geschriebenen Textes. Von Xtext werden bereits Syntaxfehler automatisch erkannt. Jedoch gibt es in meiner Sprache weitere Regeln, die zusätzlich überprüft werden müssen. Im Validator können weitere Regeln angegeben werden. Dabei kann zusätzlich angegeben werden, ob im Editor ein Fehler oder eine Warnung angezeigt werden soll. Hauptsächlich habe ich mich bei meinem Prototype um die Validation gekümmert, daher möchte ich hier die wichtigsten Regeln vorstellen, die ich bereits implementiert habe. Alle Regeln habe ich mit Xtend programmiert. Das ist eine Programmiersprache, die ähnlich zu Java ist.

1. **Error: Überprüfung der Parameter bei einem Workstep**

   Bei der Eingabe von Parametern bei einem Workstep, muss sichergestellt werden, dass die Eingabe zu der Definition im Roletype oder im Systemtype passt. Bei einem Roletype kann bei einer Message oder Task eine Anzahl und die Typen von zu übergebenen Parametern definiert werden. Dasselbe gilt für den Systemtype und die Functions. Bei einem Workstep muss darauf geachtet werden, dass nicht zu viele

und nicht zu wenige Parameter angegeben werden. Zudem müssen die Typen der Objekte, die als Parameter angegeben werden überprüft werden. Diese müssen zu den angebenden Typen in der jeweiligen Message, Task und Function passen.

2. **Überprüfung von Sender und Receiver:**

   Bei einer Commuication ist es nicht vorgesehen, dass eine Role sich selbst eine Nachricht schickt, daher muss sichergestellt sein, das Sender und Receiver nicht die gleiche Role sind.

3. **Error: Fehlende oder falsche *importance***

   Ein Workstep mit der *execution kind must* muss automatisch die *importance important* sein. Hier muss überprüft werden, ob nicht die *importance unimportant* angegeben ist. Bei der *execution kind can* muss eine *importance* angegeben werden, daher darf diese dort nicht fehlen.

4. **Error: Überprüfung der Namen von einem Interview und einem Process**

   Hier wird überprüft, dass die Namen dieser beiden Konstrukte nicht leer sind.

5. **Error: Überprüfung der Systemwork**

   Wenn eine Systemwork ein Objekt zurückgeben soll und dieses in einem Objekt abgelegt werden soll, dann muss die verwendete Function ein Returnargument haben und ein Objekt vom passenden Typ zurückgeben.

6. **Error: IF-Anweisung** Wenn bei einer IF-Anweisung zwei Objekte verglichen werden, muss sichergestellt werden, dass diese von gleichem Typ sind.

7. **Warning: Überprüfung der Typ Namen**

   Der Name eines Roletype, eines Systemtype, eines Datatype und eines Channel, muss jeweils mit einem Großbuchstaben beginnen.

8. **Warning: Fehlender Channel**

   Wenn bei einer Communication der Channel nicht angegeben wird, zeigt der Editor eine Warnung an.

8.2 Der Proposal Provider

Neben der Untersuchung des Geschriebenen bietet Xtext einen Proposal Provider. Mit Steuerung und Leertaste kann sich der Interviewer Vorschläge anschauen, die er als nächstes tippen kann. Wählt er einen dieser Vorschläge, dann wird das Hinschreiben für ihn übernommen. Da bei einem Protokoll das Tippen schnell gehen muss und das Tippen von Start und Followed und der alternativen Abläufe viel Zeit verbraucht, habe ich dafür Funktionen geschrieben. Bei einem Workstep lasse ich beim Vorschlagen zusätzlich zu den zu verwendenden Messages, Tasks oder Functions die Parametertypen anzeigen, damit der Interviewer weiß, welche Objekte er als Parameter mitgeben kann.

Kapitel 9

Validierung meiner Protokollsprache

Am Ende meiner Arbeit habe ich noch ein drittes Interview geführt (siehe Anhang A.4). Dieses Interview dient dazu, meine Sprache, die ich im Laufe der Zeit entwickelt habe, zu testen. Da meine Sprache in diesem Stadium noch nicht perfekt ist, möchte ich an Hand dieses Interviews noch vorhanden Schwächen aufzeigen. Dieses Kapitel zeigt ähnlich zu den Widersprüchen und der Semantik meiner Protokollsprache, dass diese Arbeit kein abgeschlossenes Thema ist, sondern noch viel Potenzial enthält.

Für dieses Interview habe ich noch einmal den Beratungslehrer interviewt. Dabei habe ich meinen Editor genutzt und direkt das Gespräch mit meiner Sprache protokolliert. In diesem Kapitel gehe ich auf mögliche Verbesserungen ein. Trotzdem hat die Sprache in diesem Interview gut funktioniert und auch das LSC Konzept hat sich bewährt. Die drei wichtigsten Verbesserungen zähle ich hier auf:

1. **Abbruch der Prozesse**

   In meiner Protokollsprache ist es bis jetzt nicht vorgesehen, dass Prozesse in irgendeiner Weise abgebrochen werden. Jedoch hat sich schon bei der Arbeit an der Semantik gezeigt, dass das nicht der Realität entspricht. Daher muss in meiner Sprache ein zusätzliches Konzept entwickelt werden, um den Abbruch einer Kommunikation zu modellieren.

   In diese Überlegung muss einfließen, ob und wie das Auftreten von Ereignissen generell modelliert werden soll. Bei dieser Überlegung bieten vielleicht EPKs eine gute Grundlage

2. **Die Zeit**

   Ein Problem, das mit dem vorherigen eng verknüpft ist, ist die

Zeit. Die zeitliche Dauer der einzelnen Worksteps und die Dauer, die zwischen den Worksteps liegt, spielt bis jetzt keine Rolle. Bei einem E-Mail-Verkehr ist das Abbrechen einer Kommunikation jedoch schwieriger festzustellen als bei einem Telefonat. Bei der Weiterentwicklung meines Themas muss daher darüber nachgedacht werden, ob und wie die Zeit in meine Sprache eingebaut wird. Auch ist es interessant, wie viel Zeit zwischen zwei Worksteps vergehen darf und ob dazwischen noch Worksteps aus anderen Prozessen erfolgen dürfen.

3. **Die Modellierungsmöglichkeiten**

   Meine Sprache ist sehr offen und ein Prozess kann auf unterschiedliche Weise protokolliert werden. Hier müssen in Zukunft deutlichere Richtlinien gefunden werden, um das Modellieren eines Prozesses zu vereinheitlichen. Dabei ist es auch die Frage, in welcher Tiefe ein Prozess modelliert werden soll. Zum Beispiel, wenn ein Workstep eigentlich aus drei Worksteps besteht oder ein Prozess zu groß wird und er in zwei getrennte Prozesse geteilt werden muss. Diese Vereinheitlichung ist auch wichtig in Hinblick auf das Finden von Widersprüchen.

   Meiner Ansicht nach ist es wichtig, dass die Prozesse auf möglichst tiefer Ebene betrachtet werden sollten, da sich erst dort die meisten Widersprüche zeigen.

Kapitel 10

Zusammenfassung und Ausblick

In dieser Arbeit habe ich mich mit dem Erfassen von Geschäftsprozessen befasst. Um Geschäftsprozesse in szenariobasierten Interviews zu erfassen, habe ich eine formale Protokollsprache entwickelt.

10.1 Zusammenfassung

Um die Geschäftsprozesse einer Firma zu verbessern, müssen sie erfasst und analysiert werden. Beim Erfassen von Geschäftsprozessen durch Interviews kann es immer dazu kommen, dass sich einzelne Personen widersprechen. Diese Widersprüche zu finden und die Interviews einzeln abzugleichen ist sehr aufwendig und kostet Zeit um Geld.

Meine Lösung ist ein Tool, in dem die beschriebenen Prozesse mit Hilfe einer formalen textuellen Protokollsprache protokolliert werden. Dieses Tool untersucht die Interviews beim Führen automatisch auf Widersprüche und meldet diese dem Interviewer. Durch eine textuelle Sprache geht das Protokollieren schneller als mit einer visuellen Sprache. Zudem hat sich gezeigt, dass solche Prozesse sehr lang werden können. Bei einer visuellen Sprache würden die Prozesse schnell unübersichtlich werden. Der Interviewer kann so gezielt nach diesen Widersprüchen fragen. Das bedeutet weniger Aufwand für den Interviewer und die computergestützte Untersuchung der Interviews ist gründlicher. Dieses Tool spart damit viel Zeit und Geld.

In meiner Arbeit präsentiere ich noch nicht das fertige Tool. Stattdessen eröffne ich durch meine Arbeit ein neues Forschungsgebiet. In meine Arbeit habe ich bereits Modelle untersucht, die als Grundlage für eine Protokollsprache geeignet sind. Meine Wahl fiel dabei auf die Life Sequence Charts. Diese bieten den Vorteil, dass sie beim szenariobasierten Erfassen von Systemverhalten eingesetzt werden. Aus der visuellen Sprache LSC habe ich meine textuelle formale Protokollsprache entwickelt. Das LSC-Konzept

habe ich dabei auf meine Bedürfnisse angepasst und erweitert. Zusätzlich zu dem LSC-Konzept haben mir meine beiden geführten Interviews geholfen die Sprache beständig weiter zu verbessern. Zusätzlich zu der Syntax meiner Sprache habe ich mir auch Gedanken über die Semantik meiner Sprache gemacht. Meine Sprache ist zwar noch nicht perfekt, jedoch bietet sie eine gute Grundlage, auf der aufgebaut werden kann. Für meine Sprache habe ich auch einen Editor-Prototyps erstellt, um darin die Sprache zu testen.

Daneben habe ich mir bereits Gedanken über Widersprüche gemacht. Während ich daran gearbeitet habe, habe ich festgestellt, dass dieses Thema sehr komplex ist. Daher konnte ich meine Arbeit daran nicht beenden und habe nur den Weg aufgezeigt, der bei einer Weiterführung meiner Arbeit beschritten werden muss.

Neben den technischen Aspekten meiner Sprache habe ich mir auch Gedanken darüber gemacht, wie so ein szenariobasiertes Interview aussehen muss und welche Dinge dabei beachtet werden müssen.

Zuletzt habe ich noch ein drittes Interview geführt. Dieses Interview habe ich direkt mit meiner Sprache protokolliert und auf Grundlage dieses Protokolls eine Validierung meiner Sprache durchgeführt.

10.2 Ausblick

Wie ich bereits in dem vorherigen Abschnitt erläutert habe, habe ich zunächst nur die Grundlagen meiner Idee geschaffen. In der Zukunft muss meine Sprache durch weitere Interviews verbessert werden und gegebenenfalls noch erweitert werden. Diese Verbesserungen und Erweiterungen habe ich bereits bei der Validierung meiner Sprache beschrieben.

Dazu gehört auch, dass eine genau definierte Semantik gefunden wird. Insbesondere muss die genaue Bedeutung von *mostly* und *sometimes* festgelegt werden. Dazu gehört auch, dass der Editor weiter ausgebaut wird. Bei meinem Prototypen habe ich den Bereich der Quickfixes ausgelassen. Quickfixes sind vom Editor vorgeschlagene Fehlerlösungen, die automatisch übernommen werden können. Die Technik verbessert die Tippgeschwindigkeit deutlich, daher ist sie für den Editor wichtig. Aber auch der Proposal Provider und der Validator müssen noch ausgebaut werden.

Das Auftreten der Widersprüche und wie diese aufgedeckt werden können, ist denke ich das größte offene Feld in meiner Arbeit. Hier muss deutlich definiert werden, was ein Widerspruch ist und wann er auftritt. Daneben ist es vielleicht hilfreich, sich Gedanken über Widersprüche im Allgemeinen zu machen. Denn Widersprüche spielen nicht nur in meiner Arbeit ein Rolle, sondern treten auch anderswo auf. Daneben müssen Algorithmen entwickelt werden, die einen Widerspruch erkennen.

Wichtig ist, denke ich, einen Blick auf PlayGo zu werfen und zu schauen, wie dort natürliche Sprache in Szenarien übersetzt wird. Ich denke im

Hinblick auf das Führen von Interviews ist das eine sehr interessante und hilfreiche Technik.

Anhang A

Anhang

A.1 Das Beratungslehrer-Interview

1. **Interviewer:** Was ist Ihr Beruf?
2. **Beratungslehrer:** Ich bin Beratungslehrer für Sonderpädagogik.
3. **Interviewer:** Was sind dabei Ihre Aufgaben?
4. **Beratungslehrer:** Schüler, Eltern oder Lehrer können sich an mich wenden, wenn sie Beratung zum Thema Verhaltenspsychologie brauchen.
5. **Interviewer:** Diese Personen kommen also von außerhalb Ihrer Institution?
6. **Beratungslehrer:** Genau.
7. **Interviewer:** Wenn Sie diesen Personen einen Sammelbegriff geben müssten, welcher wäre das?
8. **Beratungslehrer:** Bei uns sagen wir Klienten.
9. **Interviewer:** Wenn sich ein Klient bei Ihnen meldet. Wie läuft das ab?
10. **Beratungslehrer:** Da gibt es verschiedene Möglichkeiten.
11. **Interviewer:** Ok, dann beschränken Sie sich erst mal auf den Regelfall. Auf die anderen Fälle komme ich später noch einmal zurück.
12. **Beratungslehrer:** In der Regel rufen mich die Personen in meinem Büro an und bitten mich um einen Termin für ein Beratungsgespräch. Ich schaue dann, wann ich Zeit habe und mache mit dem Klienten den Termin ab.

13. **Interviewer:** Gibt es bei diesen Terminen Unterschiede?
14. **Beratungslehrer:** Manchmal ist es so, dass ich einen meiner Kollegen dazu bitte.
15. **Interviewer:** Bei Ihnen arbeiten also mehrere?
16. **Beratungslehrer:** Ja, insgesamt sind wir zu sechst.
17. **Interviewer:** Ah, ok. Und was machen Sie dann anders?
18. **Beratungslehrer:** Ich muss bei meinem Kollegen im Kalender nachsehen, wann er Zeit hat und schiebe dann den Termin bei ihm rein.
19. **Interviewer:** Wie haben Sie Zugriff auf den Kalender Ihres Kollegen?
20. **Beratungslehrer:** Auf der Arbeit nutzen wir Outlook und wenn ich im Büro bin, dann kann ich die Termine meiner Kollege einsehen.
21. **Interviewer:** Also, wenn Ihr Kollege an dem ausgesuchten Termin Zeit hat, dann tragen Sie den Termin bei ihm ein.
22. **Beratungslehrer:** Ja genau, nur brauche ich zum Eintragen zusätzlich noch eine Freigabe von meinem Kollegen.
23. **Interviewer:** Und wenn Sie die nicht haben?
24. **Beratungslehrer:** Dann schaue ich, ob ein anderer Kollege Zeit hat oder ich muss den Klienten zurückrufen, da ich erst mit meinem Kollegen reden muss.
25. **Interviewer:** Wie entscheiden Sie, mit welchem Kollegen Sie in den Termin gehen?
26. **Beratungslehrer:** Wenn der Klient etwas schwieriger ist, dann ist es einfach manchmal gut, wenn ein zweiter Berater dabei ist. Dann ist auch egal wer. Aber natürlich hat man Kollegen, mit denen man lieber arbeitet als mit anderen.*(kurze Pause, weil ich dachte er erzählt weiter)*
27. **Interviewer:** Und welchen Fall gibt es noch?
28. **Beratungslehrer:** Ach so ja. Manchmal ist Wissen auf einem Fachgebiet gefragt. Dann kann ich mir den Kollegen natürlich nicht aussuchen, sondern bin auf einen bestimmten angewiesen.
29. **Interviewer:** Was meinten Sie mit "Wenn der Klient etwas schwieriger ist ..." ?
30. **Beratungslehrer:** Bei mir geht es darum, dass ich oft Konflikte lösen muss. Da kochen die Emotionen schnell hoch. Manchmal ist es dann besser, wenn man einen zweiten Kollegen als Rückendeckung hat.

31. **Interviewer:** Gut, kommen wir zurück zum Telefonanruf Ihres Klienten. Also während Sie mit ihm telefonieren, schauen Sie in Outlook nach einem freien Termin. Entweder nur für sich und oder auch für Ihren Kollegen. Und wenn Sie eine Freigabe bei ihm haben, tragen Sie den Termin gleich ein. Richtig?

32. **Beratungslehrer:** Ja genau.

33. **Interviewer:** Machen Sie, nachdem Sie das Telefonat beendet haben, noch etwas?

34. **Beratungslehrer:** Ich gebe meinem Kollegen Bescheid, dass ich einen Termin für ihn abgemacht habe.

35. **Interviewer:** Wie machen Sie das?

36. **Beratungslehrer:** Entweder ich gehe kurz zu ihm und sage es ihm persönlich oder ich lege ihm einen Zettel auf seinen Schreibtisch. Hier würde ich mir wünschen, dass ich z.B. eine Nachricht auf mein Handy bekomme, wenn ein Kollege einen Termin für mich macht. Und, dass ich Zugriff über mein Handy auf unser System habe, denn wenn ich mal im Außendienst unterwegs bin und einen Termin machen will, muss ich immer erst ins Büro fahren.

37. **Interviewer:** Ok, behalten Sie das kurz im Hinterkopf, dann kommen wir gleich noch darauf zurück. *(kurze Notiz)* Sonst ist damit der Prozess erst einmal abgeschlossen?

38. **Beratungslehrer:** Ja.

39. **Interviewer:** Gibt es bei der Vergabe eines Termins sonst noch etwas zu beachten?

40. **Beratungslehrer:** Manchmal, wenn der Termin eine größere Gruppe umfasst, brauche ich einen von unseren Konferenzräumen.

41. **Interviewer:** Und was machen Sie dann?

42. **Beratungslehrer:** Wir haben zwei Konferenzräume. Wenn ich an einem Termin einen Konferenzraum brauche, dann muss ich den Termin möglichst so legen, das dort niemand sonst einen Termin hat, damit einer der Konferenzräume dann auch frei ist.

43. **Interviewer:** Können Sie sich denn nirgendwo dafür eintragen oder es bei Outloook vermerken?

44. **Beratungslehrer:** Nein, schön wärs.

45. **Interviewer:** Ok. Fällt Ihnen sonst noch etwas zur Terminvergabe ein?

46. **Beratungslehrer:** Mein einer Kollege vergisst mir eigentlich immer Bescheid zu geben, wenn er einen Termin gemacht hat. Das ist ziemlich nervig, da ich dann nur durch Zufall sehe, dass ich einen neuen Termin habe. Es gibt aber auch noch meinen anderen Kollege, der macht den Termin mit seinem Klienten ab und geht danach erst rum und fragt wer Zeit hat.

47. **Interviewer:** Und wenn er niemand findet?

48. **Beratungslehrer:** Dann muss er einen neuen Termin mit dem Klienten abmachen, aber bis jetzt hat er glaube ich immer jemanden gefunden. Wir sind ja immerhin sechs Leute.

49. **Interviewer:** Fällt Ihnen noch mehr ein?

50. **Beratungslehrer:** Nein, ich denke das wars.

51. **Interviewer:** Gut, dann kommen wir nochmal an den Anfang zurück. Da meinten Sie, dass es auch andere Wege gibt, wie sich Klienten bei ihnen melden.

52. **Beratungslehrer:** Ja, so wie ich es eben beschrieben habe, ist so zu sagen der offizielle Weg. Manchmal rufen die Leute aber auch bei meiner Chefin an. Sie gibt dann denen meine Kontaktdaten und sagt ihnen, sie sollen mich direkt anrufen.

53. **Interviewer:** Aber Sie melden sich nicht zurück?

54. **Beratungslehrer:** Nein, normalerweise nicht. Ich biete die Beratung nur an. Mein Job ist es nicht, auf die Leute zuzugehen. In der Regel müssen sie sich bei mir melden. Ich bin aber natürlich gegenüber meiner Chefin weisungsgebunden. Wenn sie also sagt: Melde dich da mal zurück. Dann tue ich das natürlich. Das kommt aber eher selten vor.

55. **Interviewer:** Gibt es noch weitere Fälle?

56. **Beratungslehrer:** Es gibt noch den Fall, dass mich z.B. aufgebrachte Eltern anrufen und meinen, dass z.B. ein Lehrer an der Schule ihres Kindes mal eine Beratung bräuchte. Es ist jetzt aber nicht mein Job diesen Lehrer anzurufen und mit ihm einen Beratungstermin abzumachen. Stattdessen sage ich den Eltern, dass sie dem Lehrer meine Kontaktdaten geben sollen und wenn der Lehrer will, kann er sich dann bei mir melden.

57. **Interviewer:** Und wenn er das nicht tut?

58. **Beratungslehrer:** Dann kommt es meist nicht zur Lösung des Konflikts.

59. **Interviewer:** Zwingen können Sie ihn aber nicht dazu?

60. **Beratungslehrer:** Nein, wie gesagt, biete ich die Beratung nur an und die Leute müssen von alleine zu mir kommen. Es sei denn die Schulleitung der Schule zwingt den Lehrer dazu. Aber trotzdem würde er sich ja dann bei mir melden und nicht umgekehrt.

61. **Interviewer:** Darauf haben Sie ja dann aber keinen Einfluss mehr.

62. **Beratungslehrer:** Ja, das stimmt.

63. **Interviewer:** Gibt es noch weitere Fälle?

64. **Beratungslehrer:** Nein, mir fällt keiner mehr ein.

65. **Interviewer:** Gut, sie erwähnten vorhin den Außendienst. Was pssiert da?

66. **Beratungslehrer:** Manchmal finden Termine nicht bei uns statt, sondern ich fahre zu dem Treffen in die betreffende Schule oder so.

67. **Interviewer:** Was gibt es da zu beachten?

68. **Beratungslehrer:** Ich hab ja vorhin schon erzählt, dass ich nur im Büro Zufriff auf Outlook habe. Das heißt, wenn ich einen neuen Termin abmachen möchte, sind eigentlich schon alle da, die dafür nötig sind. Trotzdem muss ich erst zurück ins Büro und muss dann wieder anrufen, um einen neuen Termin abzumachen. Das gleiche mit unserer Festplatte. Im Büro haben wir über unsere Computer Zugriff auf eine Festplatte. Auf der sind Dokumente gespeichert, die wir manchmal für diese Treffen brauchen. Wenn ich im Vorfeld schon weiß, dass ich einzelne Dokumente davon brauche, dann drucke ich sie mir vorher schon aus und nehme sie mit. Ergibt sich das aber erst im Laufe des Termins, dann habe ich von außerhalb keinen Zugriff auf diese Festplatte. Ich muss dann die Dokumenten später vom Büro aus mit der Post oder per E-Mail schicken. Praktischer wäre, ich könnte in der Schule sagen: Ich lade ihnen das kurz runter und dann können Sie das schnell ausdrucken.

69. **Interviewer:** Was sind das für Dokumente?

70. **Beratungslehrer:** Meistens sind das irgendweche Anträge für eine Unterstützung, z.B. durchs Jugendamt oder so.

71. **Interviewer:** Ok, ich sehe da ist noch viel Verbesserungspotenzial. Fällt Ihnen sonst noch etwas ein?

72. **Beratungslehrer:** Nein, soweit erstmal nicht.

73. **Interviewer:** Dann danke ich Ihnen für dieses Interview.

74. **Beratungslehrer:** Nichts zu danken. Wenn Sie noch weitere Fragen haben, dann stehe ich Ihnen gerne zur Verfügung.

A.2 Das Tablet-Producer-Interview

1. **Interviewer:** Was ist Ihr Beruf?
2. **Tablet-Producer:** Ich bin bei einem Verlag angestellt, der eine zweiwöchig erscheinende Zeitschrift herausgibt. Mein Job ist es die Heftinhalte und die Artikel, die in dieser Zeitschrift abgedruckt werden, so aufzubereiten, dass sie auch in der App-Ansicht auf dem Smartphone oder Tablet schön aussehen.
3. **Interviewer:** Wie würden Sie sich nennen?
4. **Tablet-Producer:** Mein genaue Jobbezeichnung?
5. **Interviewer:** Nein, die ist wahrscheinlich etwas lang. Fällt Ihnen ein kurzer Begriff ein?
6. **Tablet-Producer:** Was es vielleicht trifft ist Tablet-Producer.
7. **Interviewer:** Wie läuft das dann ab? Es muss ja erstmal einen Redakteur geben, der den Artikel schreibt.
8. **Tablet-Producer:** Ja genau. Mit den Prozessen habe ich aber nichts zu tun. Meine Arbeit fängt erst an, wenn der Artikel fertig gelayoutet ist und zur Druckerei gegeben wird.
9. **Interviewer:** Wie kommt der Artikel zu Ihnen?
10. **Tablet-Producer:** Der Redakteur schreibt seinen Artikel mit einem ganz normalen Textprogramm und gibt ihn dann ans Layout weiter. Die kümmern sich dann darum, dass der Text, die Bilder usw. in der Druckversion der Zeitschrift gut aussehen. Wenn die mit ihrer Arbeit fertig sind, dann kann ich anfangen.
11. **Interviewer:** Wie erfahren Sie davon, dass das Layout des Artikels fertig ist?
12. **Tablet-Producer:** Es gibt einen Heftplan, in dem der Termin hinterlegt ist, zu dem das Layout fertig sein muss. An den müssen sich alle halten. Zudem gibt es ein Laufwerk, in dem das Layout die Dateien für die Druckerei hinterlegt. Dann gibt es ein Skript, das quasi die ganze Zeit in diesen Ordner guckt und mir per E-Mail meldet, wenn dort neue Dateien auftauchen.
13. **Interviewer:** Wie erfahren Sie von der E-Mail?
14. **Tablet-Producer:** Bei meinem Computer popt dann ein Fenster hoch.

15. **Interviewer:** Anschließend holen Sie sich die Dateien und fangen an sie zu beabeiten?

16. **Tablet-Producer:** Genau.

17. **Interviewer:** Müssen Sie dabei noch irgendwelche Rücksprachen halten oder arbeiten Sie einfach, bis Sie fertig sind.

18. **Tablet-Producer:** Es gib den Fall, dass kleine Änderungen an den Layoutdateien vorgenommen werden müssen, damit sie auch in der App passend dargestellt werden können, z.B. bei Bildern oder irgendwelchen Tabellen.

19. **Interviewer:** Wie machen Sie das?

20. **Tablet-Producer:** Bei kleinen Änderungen rufe ich einfach kurz an und die ändern das dann. Bei größeren Änderungen spreche ich mich vorher kurz in meiner Abteilung ab.

21. **Interviewer:** Wie läuft das ab?

22. **Tablet-Producer:** Das mache ich kurz persönlich. Danach rufe ich dann beim Layout an oder ich schreib auch mal eine E-Mail.

23. **Interviewer:** Also gibt es da keine großen Unterschiede?

24. **Tablet-Producer:** Nein.

25. **Interviewer:** Haben Sie sonst noch mit der Layoutabteilung zu tun?

26. **Tablet-Producer:** Nein, sonst fällt mir nichts mehr ein.

27. **Interviewer:** Haben Sie sonst noch mit den Redakteuren zu tun?

28. **Tablet-Producer:** Mit denen habe ich eigentlich nur zu tun, wenn es um Zusatzmaterial für ihre Artikel geht. Das können z.B. noch weitere Bilder oder so sein, die in der App gezeigt werden soll.

29. **Interviewer:** Wie läuft das ab?

30. **Tablet-Producer:** Also im Normalfall hat der Redakteur während des Schreibens schon Ideen für Zusatzmaterialien. Dann gibt er die einfach mit zum Layout. In dem Ordner, in dem sich auch später die fertigen Dateien vom Layout befinden, ist dann noch ein Unterordner mit dem Zusatzmaterial dabei. Wie das zwischen den Redakteuren und dem Layout jetzt im einzelnen läuft weiß ich auch nicht genau. Kann sein, dass der Redakteur seinen Artikel auch nochmal gedruckt abgibt oder so.

31. **Interviewer:** Haben Sie dann noch weiter mit den Redakteuren zu tun?

32. **Tablet-Producer:** Manchmal fehlt etwas bei den Zusatzmaterialien oder ist missverständlich, dann muss ich den Redakteur anrufen oder ihm eine E-Mail schreiben. Und er schickt mir eine E-Mail mit Anhang zurück.

33. **Interviewer:** Fällt Ihnen noch etwas dazu ein?

34. **Tablet-Producer:** Manchmal vergessen die Redakteure das Zusatzmaterial beim Schreiben und es fällt ihnen erst später wieder ein.

35. **Interviewer:** Wie bekommen Sie das dann noch?

36. **Tablet-Producer:** Entweder er lädt das dann noch auf das Laufwerk oder er schickt es mir per E-Mail. Vorgeschrieben ist aber, dass er die Dateien auf das Laufwerk lädt. Wenn er das Material später hochlädt oder mir schickt, ist es auch nicht mehr durch die Korrektur gegangen. Aber mit diesen Prozessen habe ich, wie gesagt nichts zu tun.

37. **Interviewer:** Fällt Ihnen noch mehr ein?

38. **Tablet-Producer:** Nein, mit den Redakteuren habe ich sonst weiter nichts zu tun.

39. **Interviewer:** Wenn Sie mit Ihrer Arbeit fertig sind, was passiert dann?

40. **Tablet-Producer:** Wenn ich fertig bin, lade ich die Artikel in eine Datenbank und die Korrekteure können sich dann die Artikel in einer Vorabversion in der App anschauen.

41. **Interviewer:** Wie erfahren die von Ihnen, dass Sie fertig sind?

42. **Tablet-Producer:** Ich schicke denen eine E-Mail.

43. **Interviewer:** Was passiert dann weiter?

44. **Tablet-Producer:** Das ist unterschiedlich. Also in der Regel sollen die Korrekteure ihre gefundenen Fehler in ein Ticketsystem eintragen. Wenn ich den Fehler behoben habe, setze ich das Ticket auf „bearbeitet“. Manchmal ist das Ticket aber auch missverständlich oder unvollständig, dann trage ich das da ein.

45. **Interviewer:** Wie erfahren Sie, dass Sie ein neues Ticket haben?

46. **Tablet-Producer:** Das System schickt mir eine E-Mail.

47. **Interviewer:** Welche Fälle gibt es noch?

48. **Tablet-Producer:** Einige Korrekteure schicken mir auch einfach eine lange E-Mail mit den Fehlern, die sie gefunden haben. Wenn ich die Fehler behoben habe, schicke ich dann eine E-Mail zurück. Es gibt aber auch einen. Wenn ich den anrufe und frage, wie es mit der Korrektur aussieht, dann fängt er erstmal an mir alles mögliche zu erzählen und ich merke, wie er während des Gesprächs den Artikel durchsieht. Und dann sagt er: Ja hier und hier müssen Sie nochmal gucken. Aber das kommt eigentlich nur vor, wenn er viel zu tun hat. Manchmal sagt er aber auch nur: Ich mache das gleich und schreibe Ihnen dann eine E-Mail.

49. **Interviewer:** Fällt Ihnen sonst noch etwas ein?

50. **Tablet-Producer:** Nein, ich denke das wars.

51. **Interviewer:** Dann danke ich Ihnen für dieses Interview.

A.3 Die Grammatik meiner Protokollsprache

Der folgende Code zeigt die Xtext Grammtaik zu meiner Sprache. Das volle Projekt ist zu finden auf: https://github.com/NPrenner/Editor

```
grammar org.xtext.example.mydsl.MyDsl with
org.eclipse.xtext.common.Terminals

generate myDsl "http://www.xtext.org/example/mydsl/MyDsl"

Domainmodel:
        (element += (Interview | TypeDefinition))?
        ;

TypeDefinition:
         SystemDefinition | ChannelDefintion | DataDefinition
        | RoleDefinition
;

DataDefinition:
        'package' package = 'DataPackage' ';'
        'Datatype' name=ID '{'

        '}'
;

SystemDefinition:

        'package' package = 'SystemPackage' ';'
        'Systemtype' name=ID '{'
                functions += Function*

        '}'
;

ChannelDefintion:
        'package' package = 'ChannelPackage' ';'

        (type = ('asynchronous' | 'synchronous') )?
        'Channel' name =ID '{'


        '}'
;
```

```
 RoleDefinition:
        'package' package = 'RolePackage' ';'
        'Roletype' name=ID '{'
                state += State*
                tasks += Task*
                messages += Message*
        '}'
;

State:
        'state' name=ID ';'
;

Task:
        'task' name= ID '(' (parameter += [Parameter]
        (',' parameter += [Parameter] )*)? ')' ';'
        ;

        Message:
        'message' name= ID '(' (parameter += [Parameter]
        ("," parameter += [Parameter] )*)? ')' ';'
;

Function:
        'function' name=ID '(' (parameter += [Parameter]
        (',' parameter += [Parameter] )*)? ')'
(returnValue ?='returns'
        (returnArgument = [TypeDefinition])?
        (returnBool = 'boolean')?)? ';'
;


Interview:
        'package' package = 'InterviewPackage' ';'


        'Interview' name=STRING '{'
                'main' mainPlayer = Role
                objects += Object*

                process += BusinessProcess*
        '}'
;
```

```
Object:
        Data | Role | System | Boolean
;

Boolean:
        'boolean' (type=[TypeDefinition])? name=ID ';'
;


System:
        'system' type=[TypeDefinition] name=ID ';'
;


Data:
        'data' type = [TypeDefinition] name=ID ';'
;

Role:
        rolekind=("insideRole" | "outsideRole")
        type=[TypeDefinition] name=ID ";"
;


Parameter:
        TypeDefinition
;

BusinessProcess:
        'process' name= STRING '{'
                        (element = ProcessStartElements)?
        '}'
 ;

ProcessStartElements:
         StartTimeAlternative | Chart | StartAlternative
;

Chart:{Chart}
        'start' '{'
                startElements += ProcessChartElements*
        '}'
        'followed' '{'
```

```
                followedElements += ProcessElements*
        '}'
;

StartTimeAlternative:{StarTimeAlternative}
        'mostly' '{'
                        mostlyAlternative +=
                        ProcessStartElements?
        '}'
        'sometimes' '{'
                        sometimesAlternative +=
                        ProcessStartElements?
        '}'
        ('sometimes' '{' sometimesAlternative +=
        ProcessStartElements? '}')*

;

TimeAlternative:{TimeAlternative}
        'mostly' '{'
                         mostlyAlternative += ProcessElements*
        '}'
        'sometimes' '{'
                         sometimeAlternative+= ProcessElements*
        '}'
        ('sometimes' '{'  sometimeAlternative += ProcessElements* '}')
;


StartAlternative:{StartAlternative}
        'alt' '{'
                alternative += ProcessStartElements?
        '}'
        'or' '{'
                alternative += ProcessStartElements?
        '}'
        ('or' '{' alternative += ProcessStartElements? '}')*
;

Alternative:{Alternative}
        'alt' '{'
                alternative += ProcessElements*
        '}'
        'or' '{'
```

```
                alternative += ProcessElements*
        '}'
        ( 'or' '{' alternative += ProcessElements*'}')*
;

ProcessElements:
        Communication | Activity | Alternative |
        TimeAlternative | IFQuery | Systemwork | Loop
;

ProcessChartElements:
        Communication | Activity | Alternative |
        TimeAlternative | Systemwork
;

Loop:
        'loop''[' (uncounter = '*' | counter = INT) ']' '{'
                elements += ProcessElements*
        '}'
;

IFQuery:
        'if' '(' queries += QueryTypes ( ('||'| '&&' )
        queries += QueryTypes)* ')' '{'
                elements += ProcessElements*
        '}'
        ('else' '{'  elements += ProcessElements* '}')?
;

QueryTypes:
        Query | BraceQuery
;

Query:
        DataQuery | PlayerQuery | BooleanQuery
;

BooleanQuery:
        ('!')? booleanValue = [Boolean]
        ('is' boolean = ('true' | 'false'))?
;

DataQuery:
        (exclamation ?= '!')? objectOne= [Object]
```

```
        condition= ('!=' | '==') objectTwo=[Object]
;

BraceQuery:
  ('!')?          '(' queries +=  QueryTypes
        ( ('||'| '&&' ) queries += QueryTypes )* ')'
;

PlayerQuery:
  ('!')? player=[Role] '.' state = [State]
        ('is' boolean = ('true' | 'false'))?
;

 Communication:
        (importance = ('im' | 'unim'))? sender=[Object]
        execution= ("can" | "must") "send" "to"
        receiver = [Role] "." message = [Message]
        "(" (parameter += [Object] ("," parameter += [Object])*)?
        ")" ('via' channel=[ChannelDefintion])? ";"
 ;

Activity:
        (importance = ('im' | 'unim'))? role=[Role]
        execution = ('can' | 'must') 'do' task= [Task]
        '(' (parameter += [Object] (',' parameter += [Object])*)?
        ')' ';'
         ;

Systemwork:
        (returnObject = [Object] '=')?
        (importance = ('im' | 'unim'))? role =[Role]
        execution =('can' | 'must')  'systemwork'
        system = [System] '.' function = [Function] '('
        (parameter += [Object] ("," parameter += [Object])*)?
        ")"  ";"
;

terminal ID : ('a'..'z' | 'A'..'Z'| 'ä' | 'ö' | 'ü' | 'ß' | '_')*;
```

A.4 Das Interview mit dem Beratungslehrer in der Protokollsprache

```
package InterviewPackage;

Interview "Berater"{

      main insideRole Beratungslehrer berater;
         insideRole Beratungslehrer kollege;
      outsideRole Klient  klient;
      outsideRole Klient drittePartei;
      data Schule KlientSchule;
      data Schule beraterSchule;
      system Outlook outlook;
      data Termin termin;
      data Termin terminZwei;
      data Zeitdauer termindauer;
      data Ort ort;
      data Kontaktdaten kontaktdaten;
      boolean klientBekannt;


process "Klient meldet sich"{
   start{
         mostly{
                im klient can send to berater.
                möchteBeratung() via Telefon;
                } sometimes{
                im klient can send to berater.
                möchteBeratung() via EMail;
         }
         } followed{
         if(!klientBekannt){
        berater must send to klient.
        welcheSchule();

         im klient can send to berater.
        schule(KlientSchule);

         if(KlientSchule == beraterSchule){
         berater must send to klient.Termin();
                }else{
```

```
          berater must send to klient
         .nichtZuständig();
         berater must send to klient
         .kontaktiere(kollege, kontaktdaten);
          }

          }else{
                  berater must send to klient
                  .Termin();
          }

         }

  }

 process "Terminvereinbarung"{
 start{
         alt{
         im berater can send to klient
         .Termin();
          } or{
         im klient can send to berater
         .Termin();
                        }
          } followed{

          berater must send to klient
         .Rahmenbedingungen(termindauer, ort);


loop[*]{
termin = berater must systemwork
outlook.terminFrei(berater, kollege);
berater must send to klient
         .terminvorschlag(termin);

          alt{
          im klient can send to berater
.bestätigtTermin();
         } or{
          im klient can send to berater
.passtNicht();
         }
```

```
        }
}
 }

 process "Klient bekommt Beratung"{
  start{
        im berater can do beratungGeben
(klient);
 } followed{
        berater must send to klient.
        wollenSieweitereBeratung();

        mostly{
        im klient can send to berater.
        probiereRatschläge();


        }sometimes{
         im klient can send to berater.
                möchteZusätzlichenTermin();
                }

         sometimes{
         berater must send to klient.
        beratungInAnderemRahmen(drittePartei);
                alt{
                im klient can send to berater.ja();
                }or{
                        im klient can send to berater.nein();
                        }

                }
        sometimes{
                berater must send to klient.
                anderePerspektive();
                berater must send to klient.
        kontaktiereKlient(klient);
                }
         }
     }

     process "Termin in anderem Rahmen"{
        start{
                        im klient can send to berater.
```

```
                    beratungInAnderemRahmen();
        } followed{
                loop[*]{
                    berater must send to klient.
                    terminvorschlag(termin);

                    alt{
                            im klient can send to berater.
                            bestätigtTermin();
                    } or{
                            im klient can send to berater.
                            neuerTermin();
                     }

                     }

                }
    }
       }
```

Literaturverzeichnis

[1] Business Process Model and Notation. http://www.bpmn.org/. letzer Zugriff: März 2015.

[2] Ereignisgesteuerte Prozessketten. http://www.re-wissen.de/opencms/Wissen/Techniken/EPK-Modellierung.html. letzer Zugriff: März 2015.

[3] FLOW. http://www.se.uni-hannover.de/pages/de:projekte_flow. letzer Zugriff: März 2015.

[4] Geschäftsprozesse. http://www.informatik.uni-rostock.de/tpp/lehre/slides/mgs.pdf. letzer Zugriff: März 2015.

[5] Geschäftsprozesse Wikipedia. http://de.wikipedia.org/wiki/Gesch%C3%A4ftsprozess. letzer Zugriff: März 2015.

[6] Petri-Netze. http://de.wikipedia.org/wiki/Petri-Netz. letzer Zugriff: März 2015.

[7] PlayGo Language and Concepts. http://wiki.weizmann.ac.il/playgo/index.php/Language_%26_Concepts. letzer Zugriff: März 2015.

[8] Xtext Documentation. https://eclipse.org/Xtext/documentation/2.7.0/Xtext%20Documentation.pdf. letzer Zugriff: März 2015.

[9] S. S. Assaf Marron. Lsc language reference manual. 2014.

[10] R. Bergenthum, J. Desel, and S. Mauser. Workflow within roles. In *EMISA*, pages 65–78. GI, 2011.

[11] M. Gordon and D. Harel. Generating executable scenarios from natural language. In *Computational Linguistics and Intelligent Text Processing, 10th International Conference, CICLing 2009, Mexico City, Mexico, March 1-7, 2009. Proceedings*, pages 456–467, 2009.

[12] D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSC's and the Play-Engine.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[13] J. D. Sebastian Mauser, Robin Bergenthum. An approach to business process modeling emphasizing the early design phases.

[14] W. van der Aalst and K. van Hee. *Workflow Management: Models, Methods, and Systems.* Cooperative information systems. MIT Press, 2002.

[15] A. van Lamsweerde. *Requirements Engineering - From System Goals to UML Models to Software Specifications.* Wiley, 2009.