



ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

**Fakultät für
Ingenieurwissenschaften
und Informatik**
Institut für Datenbanken und Infor-
mationssysteme

Entwicklung eines mobilen und Service getriebenen Workflow-Clients zur Unter- stützung von evaluierten Studien der kli- nischen Psychologie am Beispiel der **AristaFlow BPM Suite und Android**

Bachelorarbeit an der Universität Ulm

Vorgelegt von:

Fabian Maier
fabian.maier@uni-ulm.de

Gutachter:

Prof. Dr. Manfred Reichert

Betreuer:

Rüdiger Pryss

2012

„ Entwicklung eines mobilen und Service getriebenen Workflow-Clients zur Unterstützung von evaluierten Studien der klinischen Psychologie am Beispiel der AristaFlow BPM Suite und Android“
Fassung vom 13. Dezember 2012

DANKSAGUNGEN: Vielen Dank an meinen Gutachter Prof. Dr. Manfred Reichert für die Unterstützung bei meiner Bachelorarbeit. Vielen Dank auch meinem Betreuer Rüdiger Pryss, der mir bei jedem Problem hilfreich zur Seite stand. Auch bei Andreas Lanz und Ulrich Kreher möchte ich mich bedanken, konnte ich doch bei Problemen jederzeit Hilfe erwarten.

© 2012 Fabian Maier

Inhaltsverzeichnis

1 Motivation	1
2 Aufbau der Arbeit	3
3 Related Work	5
4 Anwendungsszenario	7
4.1 Konstanzer Index zur Erhebung von psychosozialen Belastungen während der Schwangerschaft (KINDEX)	7
4.2 Realisierung in Aristaflow	8
4.2.1 Linear modellierter Prozess	9
4.2.2 Nichtlinear modellierter Prozess	11
5 Anforderungen	15
5.1 Funktionale Anforderungen	15
5.1.1 Mobiler Klient: Funktionalität - Prozesse	15
5.1.2 Mobiler Klient: Funktionalität - Aktivitäten	16
5.1.3 Mobiler Klient: Funktionalität - Psychologische Anforderungen	16
5.2 Nichtfunktionale Anforderungen	16
5.3 Zusammenfassung	17
6 Vorstellung der App	19
6.1 Loginbildschirm	19
6.2 Hauptbildschirm	19
6.3 Aktivitätenbildschirm	20
7 Architektur	25
8 Implementierung	27
8.1 Android	27

Inhaltsverzeichnis

8.2	Ausgewählte Implementierungsaspekte	27
8.2.1	Package de.ulm.uni.maier.bachelor	27
8.2.1.1	ActivityTemplate	27
8.2.1.2	LoginView	28
8.2.1.3	MainView	29
8.2.1.4	ActivityView	30
8.2.2	Package aristaFlowCommunication	30
8.2.2.1	Proxy	31
8.2.2.2	Com	33
8.2.2.3	Configuration	33
8.2.2.4	Namespaces	33
8.2.2.5	XmlGen	34
8.2.2.6	String2Xml	39
8.2.2.7	Von AristaFlow verwendete Manager	39
8.2.3	Package aristaFlowCommunication.responses	39
8.2.3.1	UpdMgrCreateClientWorklistNoTLResponse	40
8.2.3.2	WorklistItem	42
8.2.3.3	GetInstanceResponse	45
8.2.3.4	InstContCreateAndStartInstanceResponse	45
8.2.3.5	InstContGetInstantiableTemplatesResponse	45
8.2.3.6	InstContGetTemplateReferenceResponse	46
8.2.3.7	LogonResponse	46
8.2.3.8	QualifiedAgent	46
8.2.3.9	RemActStartStartActivityResponse und -Parameter	47
8.2.3.10	SecMgrAuthenticateAgentNameResponse	48
8.2.3.11	SecMgrAuthenticateNameResponse	49
8.2.4	Package tools	49
8.2.4.1	ImageDialogTools	49
8.2.4.2	ProzessItemAdapter	49
8.2.4.3	Input	50
8.2.4.4	InputBool	51
8.2.4.5	InputDate	51
8.2.4.6	InputFloat und InputInt	52
8.2.4.7	InputScale	52

8.2.4.8	InputString	52
8.2.4.9	InputUri	52
8.3	Probleme	52
9	Abgleich der Anforderungen	55
10	Zusammenfassung	57
	Literaturverzeichnis	59

1 Motivation

Es gibt verschiedene Möglichkeiten, klinisch psychologische Studien durchzuführen. Die gebräuchlichste Art ist die Papierform. Mit dieser wollen wir uns im Weiteren nicht beschäftigen.

Eine neuere Idee ist die Durchführung in digitaler Form mithilfe eines Business Process Management Systems (BPMS). Dieser Ansatz verspricht einige Vorteile:

- Man spart sich das Übertragen der Antworten vom Papier in eine digitale Form, und kann so deutlich schneller die Ergebnisse auswerten.
- Eine flexible Änderung der Studie ist leicht zu bewerkstelligen.
- Die Umsetzungsmöglichkeiten für Logikanforderungen von Studien während einer Befragung unterstützt ein BPMS durch vielfältige Modellierungstechniken adäquat.

Auch hier können wir wieder zwischen verschiedenen Möglichkeiten unterscheiden. Je nach Art der Umfrage kann es von Vorteil sein, wenn diese an einem festen Platz (zum Beispiel einem Desktop-PC), oder mobil durchgeführt wird.

Wenn man sich für die mobile Durchführung entscheidet, kommen noch weitere Vorteile zum Vorschein:

- Ein Tablet oder ein Smartphone ist deutlich praktikabler zu transportieren als ein großer Stapel Papier.
- Durch den Einsatz von mobilen Computern wie Smartphones oder Tablets lassen sich auch mobil schnell und unkompliziert Befragungen durchführen.

Wie lässt sich dies am Besten umsetzen?

In dieser Bachelorarbeit wurde ein Prototyp entwickelt, um diese Frage näher zu beleuchten und Probleme bei dieser Art psychologische Studien zu realisieren, zu erkennen. Interessant schien auch die Frage, welche Probleme mit einem mobilen Client in Zusammenarbeit mit einem BPMS auftauchen und ob- bzw., wie man diese behandeln kann. Mehr dazu im Kapitel Implementierung (Seite 27ff.).

1 Motivation

Am Beispiel des "Konstanzer-Index zur Erhebung von psychosozialen Belastungen während der Schwangerschaft" (KINDEX Mum Screen), wird der Vorteil einer digitalen und mobilen Durchführung von klinisch psychologischen Studien mithilfe eines BPMS gezeigt. Der "KINDEX Mum Screen" wurde von dem Fachbereich Psychologie der Universität Konstanz [5], dem Babyforum im Landkreis Konstanz e.V [3] und vivo e.V. [6] entwickelt. Als BPMS wird AristaFlow [2] verwendet.

Als mobiles Betriebssystem wird Android [1] verwendet. Im Gegensatz zu iOS erlaubt uns Android, Apps, sowohl für die Darstellung auf Tablets, als auch auf Smartphones, zu entwickeln. Da für Android mit Java [4] entwickelt wird, können wir zum Beispiel unsere Kommunikationsbibliothek einfach auf andere Betriebssysteme portieren.

2 Aufbau der Arbeit

Nachdem in Kapitel 1 die Motivation für diese Arbeit sowie die benutzten Systeme geschildert wurden, werde ich in diesem Kapitel den restlichen Aufbau der Arbeit aufzeigen (siehe auch Abbildung 2.1). In Kapitel 3 werden bisherige Forschungsergebnisse vorgestellt sowie schon vorhandene Anwendungen betrachtet. In Kapitel 4 wird näher auf den “Konstanzer Index zur Erhebung von psychosozialen Belastungen während der Schwangerschaft” eingegangen und die dazu modellierten Prozesse in AristaFlow gezeigt. Im fünften Kapitel werden funktionale und nichtfunktionale Anforderungen betrachtet und beschrieben. Screenshots der App und eine Beschreibung der Funktionsweise im Detail kommen in Kapitel 6. Um die Architektur besser verständlich zu machen, wird in Kapitel 7 mit Abbildungen der Aufbau des Prototypen erläutert. Auf Aspekte und Probleme bei der Implementierung wird in Kapitel 8 näher eingegangen. Zu guter Letzt wird in Kapitel 9 ein Abgleich der Anforderungen gegeben und in Kapitel 10 die Arbeit zusammengefasst.

2 Aufbau der Arbeit

1	Motivation			
2	Aufbau der Arbeit			
3	Related Work			
4	Anwendungsszenario	KINDEX 4.1	Realisierung in AristaFlow 4.2	
5	Anforderungen	Funktionale Anforderungen 5.1	Nichtfunktionale Anforderungen 5.2	Zusammenfassung 5.3
6	Vorstellung der App	Loginbildschirm 6.1	Hauptbildschirm 6.2	Aktivitätenbildschirm 6.3
7	Architektur			
8	Implementierung	Android 8.1	Ausgewählte Implementierungsaspekte 8.2	Probleme 8.3
9	Ableich der Anforderungen			
10	Zusammenfassung			

3 Related Work

Einige Forschungsarbeiten mit gleichem Fokus werde ich kurz vorstellen.

Martin Liebrecht hat in seiner Diplomarbeit "Technische Konzeption und Realisierung einer mobilen Anwendung für den Konstanzer-Index zur Erhebung von psychosozialen Belastungen während der Schwangerschaft" [19] einen Prototyp für eine Umsetzung des KINDEX auf einem iPad mit iOS entwickelt (siehe Abbildung 3.1). Durch die Umsetzung für genau diesen Anwendungsfall ist der Prototyp hier sehr gut angepasst. Allerdings sind Änderungen nur schwer umsetzbar, da alles fest codiert wurde. Der Fokus der Arbeit lag jedoch auch auf Rapid Prototyping und einer generellen User-Akzeptanz für eine solche Durchführungsart von klinisch, psychologischen Studien.



Abbildung 3.1: Screenshot der iOS Anwendung KINDEX

3 Related Work

Einen anderen Weg ging Maximilian Schmid mit seiner Arbeit "Technische Konzeption und Realisierung der Anwenderumgebung für ein generisches Fragebogensystem zur IT-gestützten Durchführung von evaluierten Studien der Klinischen Psychologie" [21]. Er baut auf einer Middleware-Schicht von Jörg Grüning [9] und Sebastians Jehles Konfigurator "Questionnaire Creator" (siehe Abbildung 3.2) auf. Maximilian Schmid's Prototyp kann variable Fragestellungen bearbeiten und ist nicht gezwungen, bei der Durchführung einer klinischen Studie online zu sein. Da er nicht auf einem BPMS oder einem anderen System mit integrierter Logik aufbaut, wurde ein Fortschrittsbalken implementiert, welcher dem User anzeigt, an welcher Stelle er in dem Prozess gerade ist. Diese Funktion wird jedoch in einem BPMS frei Haus mitgeliefert, was dieses besonders geeignet erscheinen lässt. Diese Funktionalität eines BPMS, so wie die anderen Vorteile, welche man durch den Einsatz erhalten würde, könnten aber sehr nützlich sein.



Abbildung 3.2: Screenshot des Questionnaire Creator

Hier setzt der Prototyp an, der in dieser Arbeit realisiert werden soll. Da ein BPMS eingesetzt wird, muss man auf Offline Durchführungen und einen Fortschrittsbalken ohne weitere Implementierung wiederum verzichten. Dafür hat man aber eine mächtige Logikunterstützung, mit der man den Prozess anpassen kann. Außerdem kann man unterschiedliche Prozesse modellieren und ist nicht auf einen fest codierten Prozess beschränkt.

4 Anwendungsszenario

4.1 Konstanzer Index zur Erhebung von psychosozialen Belastungen während der Schwangerschaft (KINDEX)

Häufig lassen sich Probleme, welche während der Schwangerschaft und nach der Geburt auftreten, durch bestimmte Indikatoren schon vor der Geburt des Kindes entdecken. Durch geeignete Befragung der Mütter zu ihren Lebensumständen kann man diese Indikatoren erkennen.

Hierfür wurde von Prof. Dr. Thomas Elbert, Dr. Dipl.-Psych. Martina Ruf und Dr. Maggie Schauer [20] [10] an der Klinischen Psychologie und der Klinischen Neuropsychologie der Universität Konstanz [5] der Konstanzer Index zur Erhebung von psychosozialen Belastungen während der Schwangerschaft (KINDEX) in Zusammenarbeit mit dem Babyforum Landkreis Konstanz e.V. [3] und vivo e.V. [6] entwickelt.

Der KINDEX Mum Screen soll von Menschen und Institutionen, welche mit Mütter in Kontakt kommen (wie zum Beispiel Hebammen, Beratungsstellen, Kinderärzte, Kliniken), eingesetzt werden können.

Martin Liebrecht weist in seiner Diplomarbeit mit dem Titel "Technische Konzeption und Realisierung einer mobilen Anwendung für den Konstanzer-Index zur Erhebung von psychosozialen Belastungen während der Schwangerschaft" [19] auf den Seiten 20ff. daraufhin, dass durch klinisch diagnostische Experteninterviews (das heißt, von Hebammen, Gynäkologinnen und Mitarbeiterinnen von Beratungsstellen durchgeführte Befragungen) in 119 Fällen die hohe Validität des KINDEX gezeigt werden konnte.

Nach der Studie fiel jedoch schnell auf, dass der Aufwand für die Durchführung zu hoch war. Nicht nur die zeitliche Belastung, sondern auch das Fehlen einer schnellen Auswertung, welche erst durch die Klinische Psychologie der Universität Konstanz, nach dem Zusenden des KINDEX per Post, durchgeführt wurde, machte den Mum Screen für die Durchführenden zu einem schwierig zu nutzenden Werkzeug.

4 Anwendungsszenario

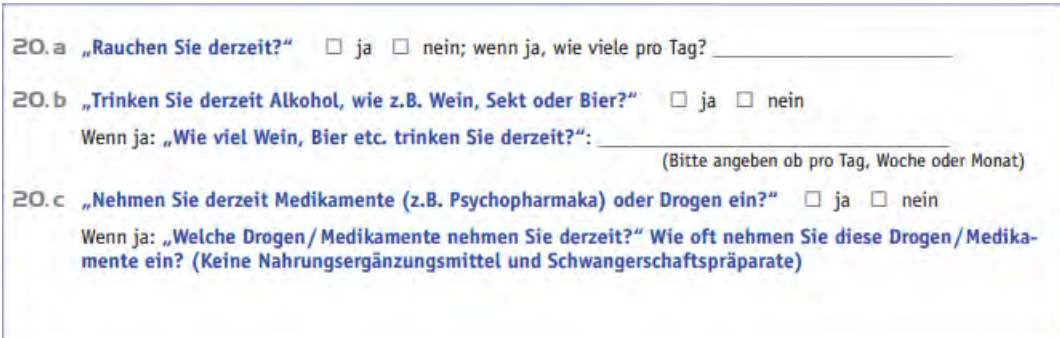
Ein digital von den Schwangeren selbst durchgeführter KINDEX weist diese Problematiken nicht auf.

4.2 Realisierung in Aristaflow

Durch das Adept-Prozessmodell [14] [15] und Konzepte der Prozessflexibilität [17] lassen sich Prozesse der unterschiedlichsten Formen abbilden. Auf diese Möglichkeiten wollen wir im Detail nicht eingehen, diese können mittels Selbststudium in [7] und [16] nachgelesen werden.

Der KINDEX Mum Screen lässt sich zufriedenstellend linear in AristaFlow modellieren. Wollen wir jedoch später noch andere Prozesse modellieren, welche unbedingt nichtlinear sein sollen, sollten wir dies vorher untersucht haben. Als Beispiel könnte man sich eine klinische Studie vorstellen, welche bei Fragen, die einem männlichen Probanden gestellt werden sollen, anders verläuft als bei einem weiblichen Probanden.

Am Beispiel der Frage 20 des KINDEX (siehe Abbildung 4.1) werde ich nun den Unterschied zwischen linearer und nichtlinearer Modellierung zeigen.



20. a „Rauchen Sie derzeit?“ ja nein; wenn ja, wie viele pro Tag? _____

20. b „Trinken Sie derzeit Alkohol, wie z.B. Wein, Sekt oder Bier?“ ja nein
Wenn ja: „Wie viel Wein, Bier etc. trinken Sie derzeit?“: _____
(Bitte angeben ob pro Tag, Woche oder Monat)

20. c „Nehmen Sie derzeit Medikamente (z.B. Psychopharmaka) oder Drogen ein?“ ja nein
Wenn ja: „Welche Drogen/Medikamente nehmen Sie derzeit?“ Wie oft nehmen Sie diese Drogen/Medikamente ein? (Keine Nahrungsergänzungsmittel und Schwangerschaftspräparate)

Abbildung 4.1: Frage 20 des KINDEX-Papierbogens

In Frage 20 geht es darum, dass die Mutter gefragt wird, ob sie raucht, trinkt oder Drogen nimmt, und wenn ja, wieviel. Es ist gut ersichtlich, dass der linear modellierte Prozess (siehe dazu Abbildung 4.8) bedeutend kürzer als der nichtlinear modellierte Prozess (siehe dazu Abbildung 4.9) ist. Durch das Fehlen der XOR und AND Knoten, sowie das Aufspalten einer Frage in mehrere Knoten, war dies jedoch auch zu erwarten. Im Anhang sind beide

Modellierungen in einer lesbaren Größe angefügt. Bei nichtlinearen Prozessen kann es zum Beispiel vorkommen, dass es mehrere aktive Knoten gibt. Hier musste eine Lösung gefunden werden, wie man mit einer solchen Situation umgehen kann.

4.2.1 Linear modellierter Prozess

Wenn wir uns die lineare Modellierung (siehe Abbildung 4.2) genauer anschauen, sehen wir, dass genau ein Knoten für die Frage gebraucht wird. Ein Blick auf die Einstellungen des Knoten (siehe Abbildung 4.3) zeigt uns, dass sechs Parameter in AristaFlow beschrieben werden. Da AristaFlow uns keine Reihenfolge garantiert, müssen wir jeder Variablen, die wir beschreiben, als Attribut noch die Position der Variable auf dem Fragebogen mitgeben (siehe Abbildung 4.5).

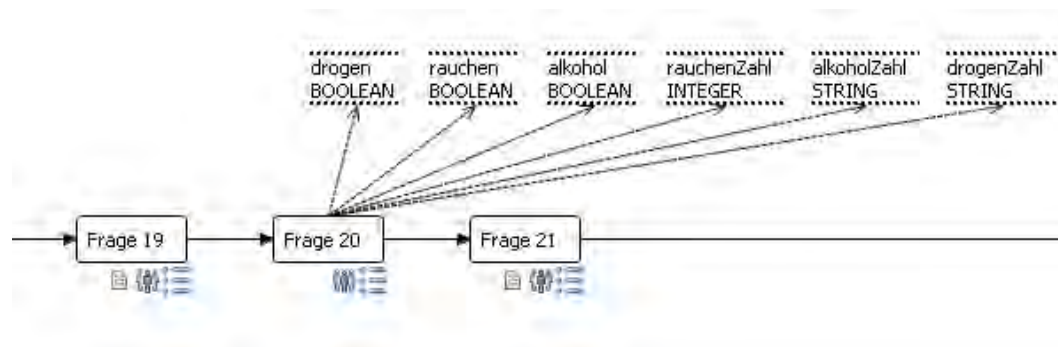


Abbildung 4.2: Frage 20 des KINDEX - linear modelliert in AristaFlow

4 Anwendungsszenario

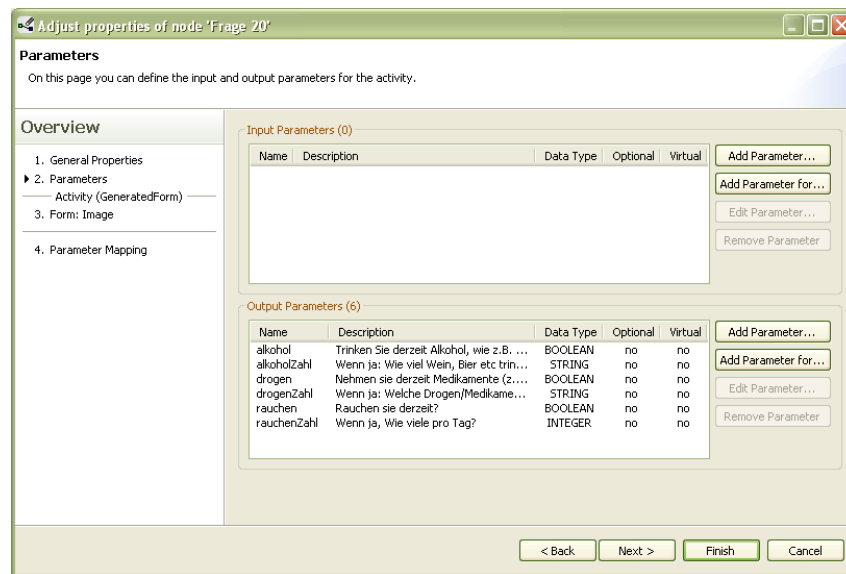


Abbildung 4.3: Frage 20 des KINDEX - linear modelliert - Einstellungen 1

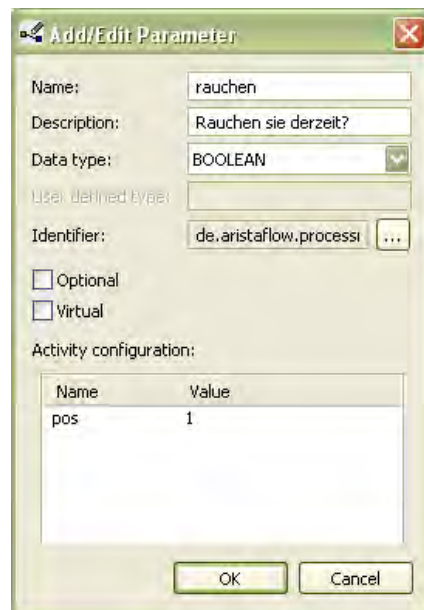


Abbildung 4.4: Frage 20 (KINDEX) linear - Einstellungen



Abbildung 4.5: Frage 20 (KINDEX) linear - Einstellungen

4.2.2 Nichtlinear modellierter Prozess

Ganz anders sieht es bei der nichtlinearen Variante aus (siehe Abbildung 4.6). Wir sehen, dass der Knoten "Frage 20" nur die Wahr oder Falsch Fragen abfragt (siehe Abbildung 6.9). Danach kommt ein AND, welches 3 Pfade gleichzeitig aktiviert. Daraufhin wird jeweils ein XOR benutzt, welches sicherstellt, dass nur weitere Fragen zu rauchen, Alkohol trinken oder Drogen nehmen gestellt werden, wenn dies auch nötig ist. Dies haben wir mit dem Knoten "Frage 20" sichergestellt. Bevor nicht alle Fragen (das heißt Aktivitäten) einzeln beantwortet wurden (siehe Abbildung 6.11), wird der Prozess nicht fortfahren (siehe Abbildung 6.10).

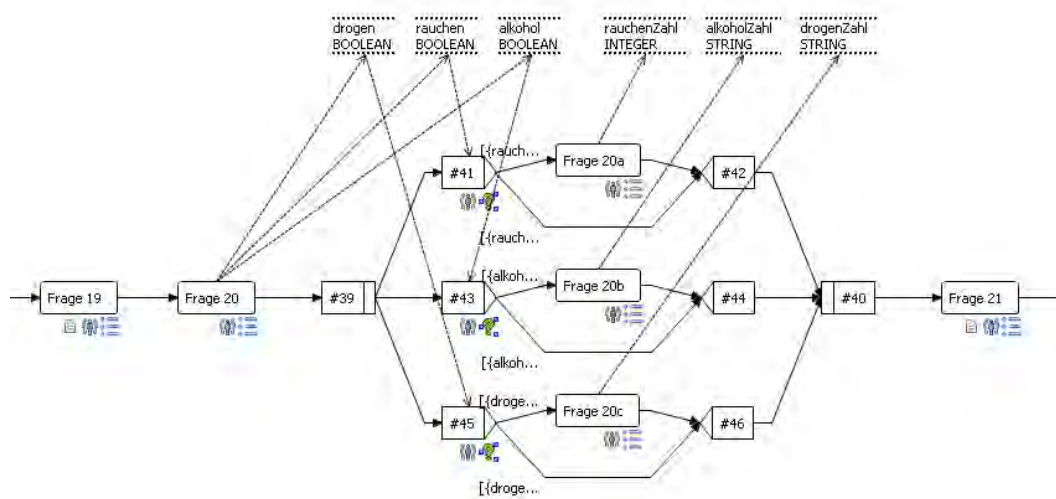


Abbildung 4.6: Frage 20 des KINDEX - nichtlinear modelliert in AristaFlow

Der linear modellierte Prozess ist deutlich kürzer und leichter zu modellieren. Wenn in der klinisch psychologischen Studie jedoch gewünscht ist, dass der Proband nur ihn betreffende Fragen sieht, ist die nichtlineare Modellierung hier von Vorteil.

Der gesamte KINDEX in Papierform ist in Abbildung 4.7 zu sehen. Auch der gesamte linear modellierte Prozess wird in Abbildung 4.8 gezeigt, in Abbildung 4.9 dementsprechend der nichtlinear modellierte Prozess.

4 Anwendungsszenario

KINDEX – Mum Screen

Konstanzer Index zur Erhebung von psychosozialen Belastungen während der Schwangerschaft

1. Datum des Interviews: _____

2. Name der Interviewerin / des Interviewers und Telefonnummer / Postadresse / Praxisstempel: _____

3. Code (1. Buchstabe Nachname + vollständiges Geburtsdatum): _____

4. „Wie alt sind Sie?“ _____

5. „Wo wurden Sie geboren?“
 Herkunftsland Deutschland ja nein
 Herkunftsland anderes: _____

6. „Wo wurde der Vater Ihres Kindes geboren?“
 Herkunftsland Deutschland ja nein
 Herkunftsland anderes: _____

7. „Leben Sie mit dem Vater Ihres Kindes zusammen?“ ja nein

8. „Wie wohnen Sie? Wieviele Personen werden in Ihrer Wohnung in wie vielen Zimmern mit dem Baby zusammen leben?“
 Anzahl Zimmer: _____
 Anzahl Personen (inkl. Baby): _____

9. „In welcher Schwangerschaftswoche sind Sie momentan?“ _____

10. „Leiden Sie unter Schmerzen, körperlichen Beschwerden oder Unwohlsein?“
 Frei vorgebrachte Nennungen: _____

11. „Wären Sie schon einmal schwanger?“ ja nein
 „Wenn ja, gab es bei zurückliegenden Schwangerschaften Komplikationen?“ ja nein
 Wenn ja, welche? _____

12. „Kam es während der aktuellen Schwangerschaft zu Komplikationen?“ ja nein
 Wenn ja, zu welchen? _____

In Kooperation: Universität Konstanz, Babyforum im Landkreis Konstanz e.V., vivo e.V., Dr. Martina Ruf, Dr. Maggie Schauer, Prof. Dr. Thomas Elbert & Dr. Wilfried Kotzer
 Kontakt: +49 (0)7531/88-4623 oder marina.ruf@uni-konstanz.de

13. „Liegen bei Ihnen weitere medizinische Risikofaktoren vor?“ ja nein
 Wenn ja, welche: _____

14. a. „Schwangerschaften können geplant und ungeplant sein.“
 „War Ihre jetzige Schwangerschaft geplant?“ ja nein

14. b. „Unabhängig von der Planung, wie sehr freuen Sie sich momentan auf Ihr Kind?“
 Freude: wenig 0_1_2_3_4_5_6_7_8_9_10 sehr

14. c. „Neben der Freude auf ein Kind, kann man natürlich auch besorgt oder bedrückt sein, was die Zukunft mit einem Kind bringt? Wie sehr sorgen Sie sich momentan?“
 Sorge: wenig 0_1_2_3_4_5_6_7_8_9_10 sehr

14. d. „Wie steht Ihr Partner aktuell zu der Schwangerschaft und der Vorstellung Vater zu werden. Auch Männer können sich freuen und / oder sich Sorgen machen. Wie schützen Sie das bei Ihrem Partner aktuell ein?“
 Freude des Partners: wenig 0_1_2_3_4_5_6_7_8_9_10 sehr
 Sorge des Partners: wenig 0_1_2_3_4_5_6_7_8_9_10 sehr

15. „Glauben Sie, dass Sie durch die Geburt des Kindes in finanzielle Schwierigkeiten kommen werden?“
 ja nein

16. a. „Wie häufig haben Sie sich im letzten Monat nicht in der Lage gefühlt, wichtige Dinge in Ihrem Leben zu kontrollieren?“
 nie selten manchmal häufig sehr häufig

16. b. „Wie häufig waren Sie im letzten Monat davon überzeugt, dass Sie die Fähigkeit besitzen, mit Ihren persönlichen Problemen fertig werden zu können?“
 nie selten manchmal häufig sehr häufig

16. c. „Wie häufig haben Sie im letzten Monat das Gefühl gehabt, dass die Dinge so laufen, wie Sie es wollen?“
 nie selten manchmal häufig sehr häufig

16. d. „Wie häufig haben Sie im letzten Monat das Gefühl gehabt, dass Ihre Probleme Ihnen so über den Kopf wachsen, dass Sie sie nicht mehr bewältigen können?“
 nie selten manchmal häufig sehr häufig

„Im Folgenden möchten wir Sie einige Dinge fragen, die Sie möglicherweise in der Vergangenheit oder gegenwärtig erleben. Wie werden die folgenden Dinge nur kurz ansprechen, sollten Sie im Anschluss das Bedürfnis haben ausführlicher darüber zu sprechen oder sich beraten zu lassen, so kann ich Ihnen gerne bei der Vermittlung einer Gesprächspartnerin behilflich sein.“

17. „Manche Menschen erleben während ihrer Kindheit oder Jugend sehr viel Stress. Das ist nicht ungewöhnlich. Haben auch Sie jemals in Ihrer eigenen Kindheit Erfahrungen von körperlicher Gewalt gemacht? Sind Sie zum Beispiel von Ihren Eltern oder anderen Erwachsenen in Ihrem näheren Umfeld geschlagen worden?“ ja nein

18. „Haben Sie jemals in Ihrer eigenen Kindheit oder Jugend Erfahrungen von sexueller Gewalt gemacht? Gab es zum Beispiel von Ihren Eltern oder anderen Erwachsenen in Ihrem näheren Umfeld sexuelle Übergriffe auf Sie?“ ja nein

In Kooperation: Universität Konstanz, Babyforum im Landkreis Konstanz e.V., vivo e.V., Dr. Martina Ruf, Dr. Maggie Schauer, Prof. Dr. Thomas Elbert & Dr. Wilfried Kotzer
 Kontakt: +49 (0)7531/88-4623 oder marina.ruf@uni-konstanz.de

19. a. „Eine Schwangerschaft ist eine große Veränderung. Dies wirkt sich auch auf die Partnerschaft aus. Haben Sie momentan das Gefühl, dass die Stilleisigkeiten mit Ihrem Partner zunehmen?“
 ja nein

19. b. „Kam es in den letzten 8 Wochen zu lautstarken Auseinandersetzungen zwischen Ihnen und Ihrem Partner?“
 ja nein

19. c. „Kam es in den letzten 8 Wochen zu Handgreiflichkeiten zwischen Ihnen und Ihrem Partner?“
 ja nein

19. d. „Hatten Sie jemals in Ihrem Leben eine Partnerschaft in der es zu gewalttätigen Übergriffen kam?“
 ja nein

20. a. „Rauchen Sie derzeit?“ ja nein; wenn ja, wie viele pro Tag? _____

20. b. „Trinken Sie derzeit Alkohol, wie z.B. Wein, Sekt oder Bier?“ ja nein
 Wenn ja: „Wie viel Wein, Bier etc. trinken Sie derzeit?“ _____
(Bitte angeben ob pro Tag, Woche oder Monat)

20. c. „Nehmen Sie derzeit Medikamente (z.B. Psychopharmaka) oder Drogen ein?“ ja nein
 Wenn ja: „Welche Drogen / Medikamente nehmen Sie derzeit?“ Wie oft nehmen Sie diese Drogen / Medikamente ein? (Keine Nahrungsergänzungsmittel und Schwangerschaftspräparate)

„Wie verhält sich das bei Ihrem Partner?“

21. a. „Raucht Ihr Partner?“ ja nein

21. b. „Trinkt Ihr Partner regelmäßig Alkohol?“ ja nein
 Wenn ja: „Wie viel Wein, Bier etc. trinkt er derzeit?“ _____
(Bitte angeben ob pro Tag, Woche oder Monat)

21. c. „Machen Sie sich Sorgen bezüglich des Alkoholkonsums Ihres Partners?“ ja nein

21. d. „Nimmt Ihr Partner Drogen?“ ja nein

22. a. „Viele Menschen leiden im Laufe ihres Lebens unter unterschiedlichen psychischen Erkrankungen wie Depressionen oder Ängsten. Die Häufigkeit dieser Erkrankungen ist viel höher als die meisten annehmen. Hat bei Ihnen jemals ein Arzt oder ein Psychologe eine psychische Erkrankung, wie z.B. eine Depression, eine Angststörung oder eine andere psychische Erkrankung diagnostiziert?“
 ja nein

Wenn ja:
 22. b. Depression: ja nein
 22. c. Angststörung: ja nein
 22. d. Andere psychiatrische Diagnose: ja nein;
 Wenn ja welche: _____

22. e. „Hatten Sie selbst jemals das Gefühl unter einer psychischen Erkrankung zu leiden – auch wenn diese nicht von einem Psychologen oder Arzt diagnostiziert wurde?“
 ja nein
 Wenn ja welche Erkrankung / welche Probleme lagen vor: _____

In Kooperation: Universität Konstanz, Babyforum im Landkreis Konstanz e.V., vivo e.V., Dr. Martina Ruf, Dr. Maggie Schauer, Prof. Dr. Thomas Elbert & Dr. Wilfried Kotzer
 Kontakt: +49 (0)7531/88-4623 oder marina.ruf@uni-konstanz.de

23. „Hat Ihnen jemals in einer schwierigen Lebensphase ein Arzt Medikamente verschrieben, damit es Ihnen leichter besser geht? Häufig werden in kritischen Lebensphasen Medikamente verschrieben, die die Stimmung heben, einen beruhigen oder die einem beim Schlafen helfen. Haben Sie jemals solche Medikamente bekommen?“
 Psychopharmaka: ja nein
 Falls bekannt, Name oder Anwendungsbereich der Medikamente: _____

24. „Wir haben gerade über kritische Lebensphasen gesprochen. Haben Sie in einer solchen Phase jemals Hilfe gesucht bei einem Arzt, einer Psychotherapeutin oder einer Beratungsstelle?“
 ja nein

25. „Kam es in einer solchen schwierigen Lebensphase jemals zu einer Aufnahme in eine Klinik, eine psychosomatische Klinik oder eine Psychiatrie?“
 ja nein

Zum Abschluss:

26. „Welche Art von Unterstützung und Hilfe würden Sie sich für sich und Ihr Baby in Zukunft wünschen?“ _____

Frage an die Interviewerin / den Interviewer:
 Haben Sie im Anschluss an die Durchführung dieses Fragebogens bereits eine Empfehlung an die Schwangere gegeben, z.B. Hinweis auf Beratungsstelle etc.?
 ja nein
 wenn ja, welche Empfehlung? _____

Bemerkungen der Interviewerin / des Interviewers:

Bitte senden Sie den ausgefüllten Fragebogen an folgende Adresse zurück:
 Dr. Martina Ruf, Dr. Maggie Schauer
 Universität Konstanz
 Kompetenzzentrum Psychotraumatologie
 Klinische Psychologie
 D-78457 Konstanz, Fach 23

Ihre Bankverbindung: KontoinhaberIn: _____
 Bank: _____ BLZ _____ Konto-Nr. _____
Die Überweisungen erfolgen aus organisatorischen Gründen kumulativ alle 3 Monate.

www.babyforum-landkreis.konstanz.de
www.vivo.org
www.psychologie.uni-konstanz.de/abteilungen/clinicalpsychology/

Abbildung 4.7: Der komplette KINDEX Mum Screen in Papierform

4.2 Realisierung in AristaFlow



Abbildung 4.8: KINDEX linear modelliert in AristaFlow



Abbildung 4.9: KINDEX nichtlinear modelliert in AristaFlow

5 Anforderungen

5.1 Funktionale Anforderungen

Die realisierte Android App hat einige funktionale Anforderungen, um clientseitig den modellierten AristaFlow Prozess auszuführen.

Da AristaFlow uns die Möglichkeit gibt, verschiedenen Usern, verschiedene Rollen zuzuweisen, wollen wir das nutzen. Das bedeutet, die App sollte dem User die Möglichkeit geben, sich anzumelden und mit AristaFlow zu verbinden.

Die Kommunikation wollen wir ferner mittels Web Services realisieren. Dies geht, da AristaFlow eine Web Service API hat, mit welcher wir über SOAP kommunizieren können.

Außerdem sollte es die Möglichkeit geben, zu bestimmen, mit welchem Serverendpunkt wir kommunizieren wollen.

Da wir einen mobilen Client (siehe dazu auch [18], [11] und [8]) entwickeln, müssen wir uns besonders darum kümmern, wie der Client reagieren soll, wenn es zu Verbindungsabbrüchen kommt.

5.1.1 Mobiler Klient: Funktionalität - Prozesse

Wir sollten in der Lage sein, jeden formularbasierten Prozess (was für Kindex und andere klinisch psychologischen Studien ausreichend ist) der in AristaFlow modelliert wurde, ohne weitere Änderungen in dem mobilen Client ausführen zu können (siehe dazu auch [13] und [12]).

Ebenso wollen wir die Möglichkeit realisieren, einen Prozess, den unser User starten darf, auch starten zu können.

Wollen wir einen Prozess abbrechen, sollte dies auch aus der Android App machbar sein.

5 Anforderungen

5.1.2 Mobiler Klient: Funktionalität - Aktivitäten

Nachdem der Server uns authentifiziert hat, wollen wir eine Übersicht über gestartete Prozesse, sowie Aktivitäten, auf welche wir mit unseren Userrechten Zugriff haben, zu sehen bekommen.

Ist eine Aktivität von unserem User gestartet, soll die Möglichkeit bestehen, diese Aktivität zu bearbeiten und zurück an den Server zu schicken. Sollten wir dies nicht wollen, können wir die Aktivität "Resetten" : Ein anderer (oder unser) User bekommt die Möglichkeit, diese Aktivität auszuführen, oder zu suspendieren. Nur unserer User hat die Möglichkeit, diese Aktivität fortzuführen. Eine Aktivität welche den Status "Suspended" hat, sollte in der Übersicht erkennbar sein.

Um die Orientierung in komplexeren Prozessen zu erleichtern, wollen wir zudem die Möglichkeit haben, grafisch zu sehen, an welcher Stelle im Prozess wir uns momentan befinden.

5.1.3 Mobiler Klient: Funktionalität - Psychologische Anforderungen

Als Beispielszenario verwenden wir den Kindex Mum Screen. Dieser soll jedoch nicht der einzige Anwendungsfall bleiben. Daher soll der Client flexibel für unterschiedlichste Szenarien einsetzbar sein.

Spezielle, für klinisch psychologische Studien erforderliche Elemente, sollen eingearbeitet und über AristaFlow einsetzbar sein.

5.2 Nichtfunktionale Anforderungen

Der User soll ein ansprechendes und leicht zu überblickendes Graphical User Interface (GUI) haben. Hierzu gehören auch annehmbare Reaktionszeiten der App.

Für zukünftige Weiterentwicklungen, sowie Fehlerbeseitigung, ist es unerlässlich, eine geordnete Struktur im Quellcode zu haben. Über eine modulare Struktur lassen sich solche Veränderungen schnell und einfach realisieren. Hinzu kommt die Möglichkeit, Programmteile, wie zum Beispiel das Kommunikationsmodul, in anderen Projekten wiederzuverwenden. Auch unerfahrene Nutzer sollen nach einer möglichst kurzen Einarbeitungsphase in der La-

ge sein, die Bedienung der App zu beherrschen und alle wichtigen Funktionen zu kennen und einsetzen können.

5.3 Zusammenfassung

Zur besseren Übersicht wurde in Tabelle 5.1 die Anforderungen noch einmal tabellarisch zusammengefasst.

Anforderung	Anforderungsart
Multi-User-Unterstützung	funktional
Web Service orientierte Kommunikation mit AristaFlow Servern	funktional
Verschiedene Serverendpunkte sollen verwendet werden können	funktional
Verbindungsabbrüche müssen verarbeitet werden können	funktional
<i>Mobiler Klient: Funktionalität - Prozesse</i>	
Ein in Aristaflow modellierter formularbasierter Prozess soll ohne weitere Änderungen von der App ausgeführt werden können	funktional
Neue Prozesse müssen gestartet werden können	funktional
Laufende Prozesse müssen abgebrochen werden können	funktional
<i>Mobiler Klient: Funktionalität - Aktivitäten</i>	
Übersicht über von diesem User bearbeitbare Aktivitäten	funktional
Aktivität soll gestartet, pausiert sowie resettet werden können	funktional
Grafische Anzeige über Fortschritt und gerade ausführbare Aktivitäten im Prozess	funktional
<i>Psychologische Anforderungen</i>	
Spezielle Formularanpassung an die Bedürfnisse für eine klinisch psychologische Studie	funktional
Soll flexibel für unterschiedlichste Szenarien einsetzbar sein	funktional
Geringe Zeiträume zwischen Eingaben	nicht funktional
App soll leicht erweiterbar sein	nicht funktional
App soll gut wartbar sein	nicht funktional
Es soll ein ansprechende GUI umgesetzt werden	nicht funktional
Leicht verständliche Eingabemöglichkeiten ohne lange Einlernzeiten	nicht funktional

Tabelle 5.1: Anforderungen

6 Vorstellung der App

6.1 Loginbildschirm

Nach dem Starten der Anwendung öffnet sich die Startactivity (siehe Abbildung 6.1). In dieser sehen wir die voreingestellte Serveradresse, den User, sowie ein Feld mit dem Passwort. Nach dem Klick auf den Button "ändern" öffnet sich ein Pop-Up, in dem wir die neuen Daten eingeben können (siehe Abbildung 6.2). Nachdem gültige Daten eingegeben worden sind und der Anwender auf den Button "Verbinden" klickt, öffnet sich wieder ein Pop-Up (siehe Abbildung 6.3), in dem die zur Verfügung stehenden Rollen angezeigt werden, und eine dieser Rollen ausgewählt werden kann.

6.2 Hauptbildschirm

In der Hauptactivity (siehe Abbildung 6.4) kann der Benutzer über den Button "Neuer Prozess" einen neuen Prozess starten. Es öffnet sich ein Pop-Up (siehe Abbildung 6.6), in dem er eine Übersicht der Prozesse sieht, die mit unserem gegenwärtigen User und der ausgewählten Rolle gestartet werden dürfen. Die darunter stehende Übersichtsliste über für ihn verfügbare Aktivitäten kann er mit dem Button "Update Liste" neu laden. Der "Logout" Button sendet an den Server seinen Logout und er kehrt zur Startactivity (siehe Abbildung 6.1) zurück.

In der Liste der verfügbaren Aktivitäten sieht er beim ersten Prozess ein Pausezeichen. Dies bedeutet, die Aktivität ist schon an unseren User vergeben und im Moment im Zustand "Suspended". Durch einen langen Klick auf die Aktivität öffnet sich ein Pop-Up (siehe Abbildung 6.6), in dem er den Prozess abbrechen kann. Optional ist es hier auch noch möglich, einen Grund hierfür anzugeben. Durch einen kurzen Klick startet er die Aktivität und es öffnet sich die Prozessschrittactivity (siehe Abbildung 6.7).

6.3 Aktivitätenbildschirm

Hier sieht der Benutzer oben in der Titelzeile der Activity den Titel der Aktivität, direkt darunter schwarz die Beschreibung der Aktivität.

In diesem Beispiel sieht der User eine Aktivität mit einer Booleschen- sowie vier Integer Variablen. Die Integer Variablen haben jeweils das Minimum sowie das Maximum angegeben, deswegen werden sie als Skalen angezeigt. In Blau über dem Feld wird die Variablenbeschreibung angezeigt, darunter kann der Wert bestimmt werden. Der Variablenname wird nicht angezeigt so dass man beim Modellieren des Prozesses mehr Freiheiten bei den Namen hat, ohne dass der Benutzer diese zu Gesicht bekommt.

Mit dem Klick auf den "Ok" Button schickt der Benutzer die ausgefüllten Felder an den Server. Mit einem Klick auf "abbrechen" wird eine Resetnachricht an den Server geschickt und andere Benutzer (oder unser Benutzer) können diese Aktivität übernehmen. "Suspend" schickt eine Suspendnachricht an den Server, woraufhin die Aktivität suspendiert wird. Im Unterschied zu "abbrechen" kann später **nur** unser Nutzer diese Aktivität fortsetzen.

6.3 Aktivitätenbildschirm



Abbildung 6.1: Startactivity

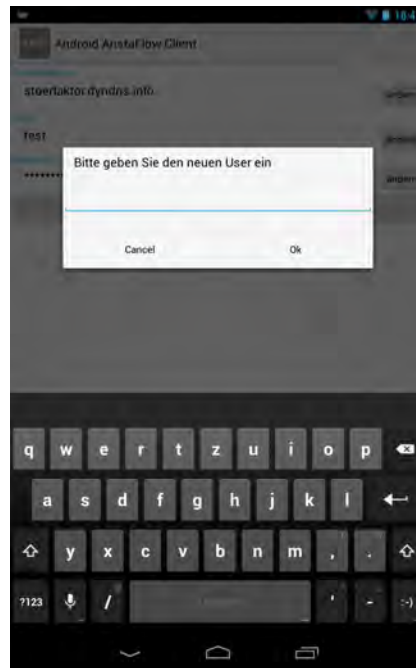


Abbildung 6.2: Beispiel einer Dateneingabe

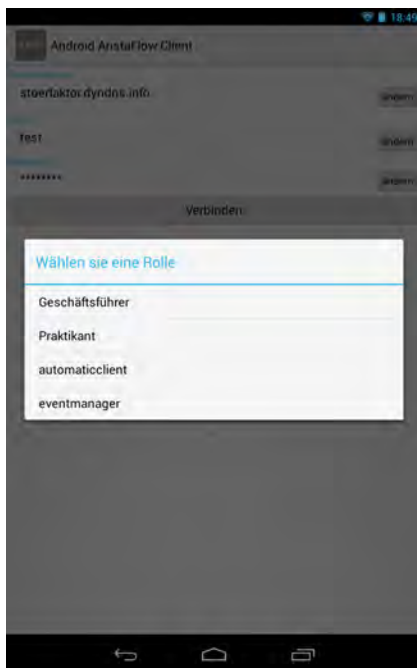


Abbildung 6.3: Rollenauswahl



Abbildung 6.4: Hauptactivity

6 Vorstellung der App

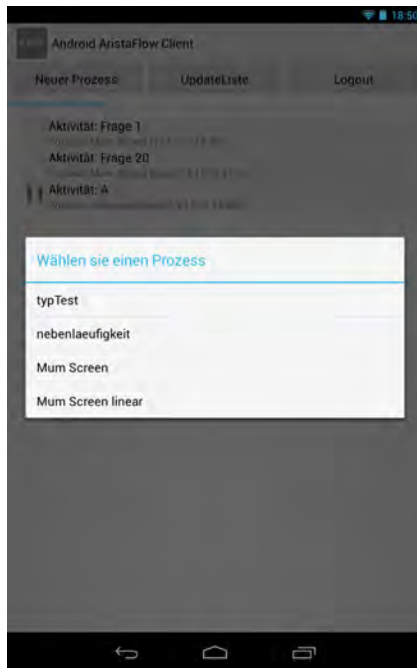


Abbildung 6.5: Prozessauswahl

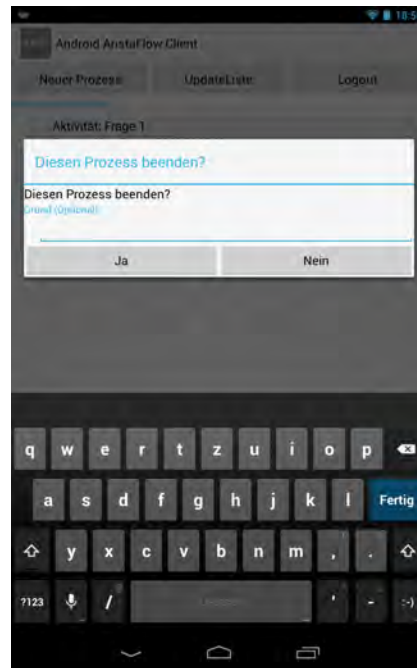


Abbildung 6.6: Beenden eines Prozesses



Abbildung 6.7: Screenactivity



Abbildung 6.8: Frage 20 linear

6.3 Aktivitätenbildschirm

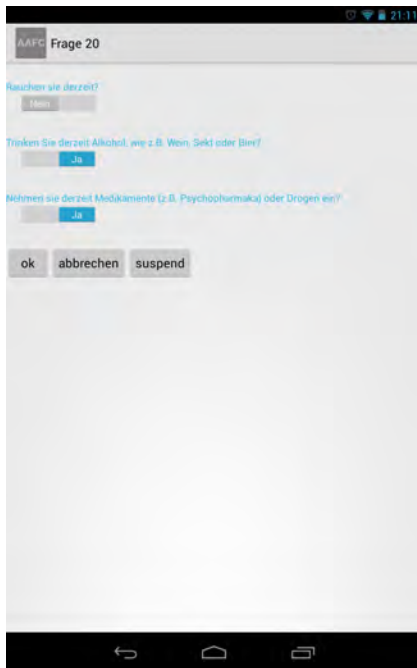


Abbildung 6.9: Frage 20 nichtlinear I

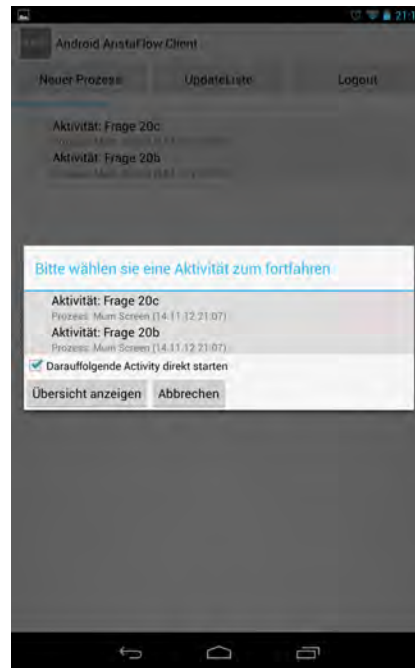


Abbildung 6.10: Frage 20 nichtlinear II

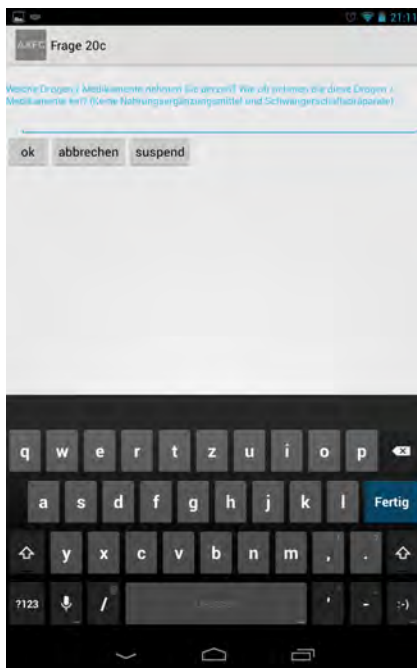


Abbildung 6.11: Frage 20 nichtlinear III



Abbildung 6.12: Auswahl eines Datums

7 Architektur

In Abbildung 2.1 ist eine Übersicht über den Aufbau der realisierten Android-App abgebildet. Das Client Package kann über das Communicationpackage SOAP Nachrichten an den AristaFlow Web Service schicken. Zusätzlich hat es im *Tools Package* nützliche Methoden, welche ihm die Arbeit erleichtern.

In Abbildung 7.2 ist eine detailliertere Ansicht gegeben, was clientseitig geschieht. Eine genauere Beschreibung über das Zusammenspiel der Klassen findet sich in Kapitel 8 - Implementierung.

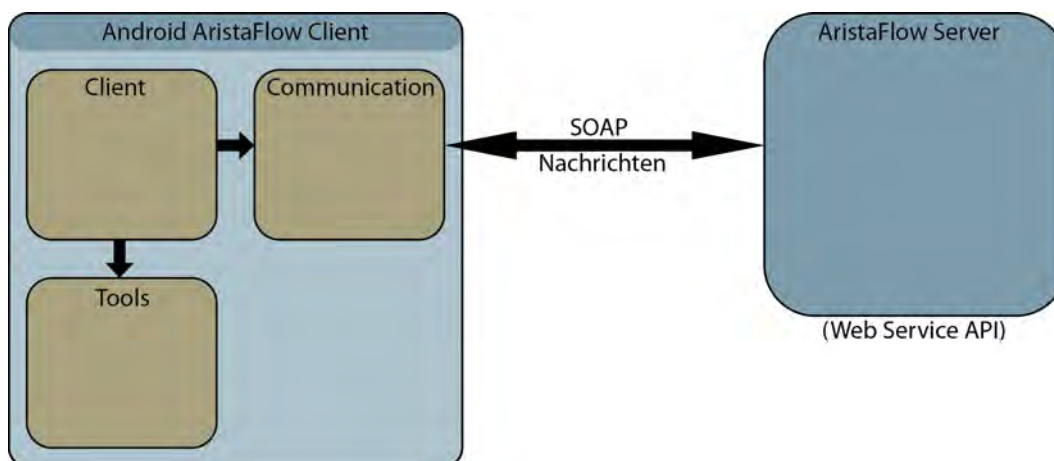


Abbildung 7.1: Übersicht der Architektur

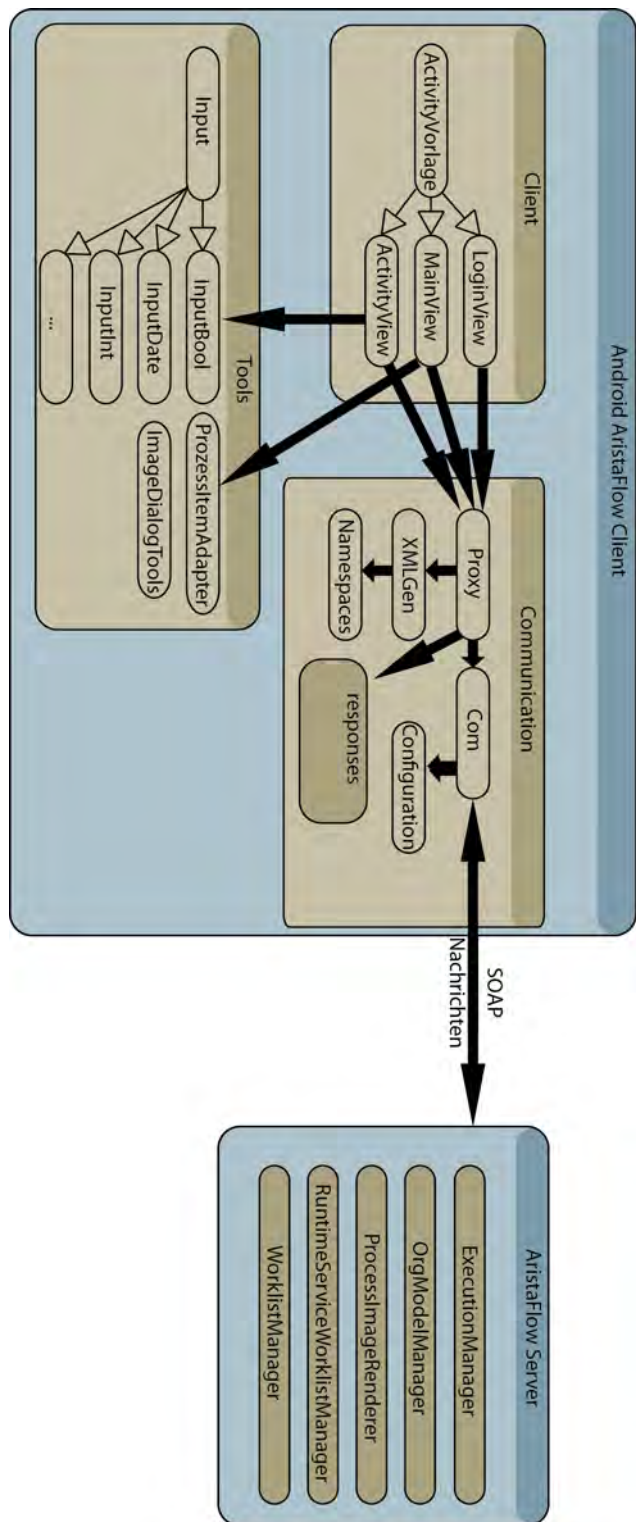


Abbildung 7.2: Detailliertere Ansicht der Architektur der App

8 Implementierung

8.1 Android

Zur Entwicklung der App wurde als Testgerät ein Nexus 7 Tablet benutzt. Das Betriebssystem war Android 4.1 Jelly Bean.

Dies wird auch als Mindestanforderung für die App vorausgesetzt. Das hat jedoch hauptsächlich kosmetische Gründe durch spezielle grafische Vorteile von Android 4.1. Mit wenigen Anpassungen kann die App auch auf Smartphones mit einem kleineren Display, sowie älteren Android Versionen laufen.

8.2 Ausgewählte Implementierungsaspekte

Die gesamten Implementierungsaspekte in diesem Kapitel zu beschreiben würden den Rahmen dieser Arbeit deutlich sprengen. Deswegen werde ich mich auf einige ausgewählte Implementierungsaspekte beschränken.

8.2.1 Package `de.ulm.uni.maier.bachelor`

8.2.1.1 `ActivityTemplate`

Das *ActivityTemplate* dient als Vorlage für alle anderen Klassen in diesem Package. Sie erbt von *android.app.Activity*. Diese Klasse ist nötig, damit Android unsere Klassen als Android Activities erkennt. Wenn von einer Android Activity zur nächsten gewechselt werden soll, wollen wir unser *Proxy* Objekt natürlich auch in der nächsten Android Activity noch benutzen können.

8 Implementierung

Um von einer Android Activity zur nächsten wechseln zu können, müssen wir als erstes ein *Intent* Objekt, welches den Namen der neuen Android Activity enthält, erzeugen. Dieses übergeben wir dann an die Methode *startActivity*. Um nun unser *Proxy* Objekt mitnehmen zu können, haben wir in dieser Vorlage die Methode *startActivity* überschrieben (siehe Listing 8.1).

Listing 8.1: Die Methode *startActivity*

```
1  
2 public void startActivity(Intent intent) {  
3     intent.putExtra("vertreter", proxy);  
4     super.startActivity(intent);  
5 }
```

Wenn die neue Activity gestartet wird, ruft Android die Methode *onCreate* auf. In dieser holen wir unser *Proxy* Objekt wieder zurück (siehe Listing 8.2) und können es auf Gültigkeit prüfen.

Listing 8.2: Wie der *Proxy* zurückgeholt wird

```
1  
2 proxy = (Proxy)b.get("vertreter");
```

8.2.1.2 LoginView

LoginView kümmert sich um die Logik des Login Bildschirms, wenn der User das App startet. Sie erbt von *ActivityTemplate* und die GUI ist durch die XML Datei *login.xml* vorgegeben. Buttons werden mit *OnClickListnern* versorgt. Drei Methoden kümmern sich um die Reaktionen auf ein Klick Event.

config Die Methode *config* überprüft als erstes, ob der User, das Passwort oder der Server geändert werden soll. Dann überschreibt es den vorgegeben Wert, in dem entsprechenden String.

login Diese Methode holt sich von dem *Proxy* die zur Verfügung stehenden Rollen und lässt den User in einem Dialog die gewünschte Rolle auswählen.

chosen Klickt der User in dem von *login* erstellten Dialog auf die Rolle, wird diese an die *chosen* Methode übergeben. Nun können wir uns mit dieser Rolle anmelden und zur *MainView* wechseln.

8.2.1.3 MainView

Die GUI für die *MainView* Klasse ist in der XML Datei *main.xml* definiert. Sie erbt ebenfalls von der Vorlage *ActivityTemplate*.

Wenn das Objekt erstellt wird, werden in der *onCreate* Methode *OnClickListener* definiert und die Liste mit den verfügbaren *Activity*s aus *AristaFlow* generiert.

checkNext Diese Methode überprüft, wenn wir von einer *Activity* kommen, ob eine weitere *Activity* des gleichen Prozesses vorhanden ist.

Findet er keine passende Aktivität passiert nichts und die App erwartet weitere Eingaben des Benutzers. Findet er genau eine passende *Activity*, startet er diese. Bei mehreren *Activity*s kann der User in einem Dialog auswählen, mit welcher *Activity* er fortfahren möchte.

updateList Die Methode *updateList* holt sich vom *Proxy* Objekt eine Liste mit startbaren *Activity*s und stellt diese dar.

neuerProzess Wird diese Methode ohne Parameter gestartet, holt sie eine Liste mit Prozessen die der User starten darf. Andernfalls kann man dieser Methode auch eine Zahl zur Identifizierung des Prozesses übergeben. Nun wird der Prozess gestartet und die Liste mit den verfügbaren *Activity*s aktualisiert.

showProcessImage Als Parameter erwartet *showProcessImage* ein *WorklistItem*. Von diesem Prozess wird dann vom *Proxy* ein Bild angefordert und in einem Dialog dargestellt.

prozessAbbrechen Diese Funktion erhält, als Parameter die Position eines Prozesses in der Aktivitätenliste übergeben und einen Grund, wieso der Prozess abgebrochen werden soll. Dann ruft sie bei dem *Proxy* die Methode zum Abbrechen von Prozessen auf.

8 Implementierung

logOutGeklickt Diese Methode meldet sich über den *Proxy* beim AristaFlow Server ab.

8.2.1.4 ActivityView

Die GUI für die *ActivityView* Klasse ist in der XML Datei *activityview.xml* definiert. Sie erbt ebenfalls von der Vorlage *ActivityTemplate*.

In dieser Android Activity wird eine Activity dargestellt. Als erstes werden alle Felder der Activity überprüft, ob sie von der App darstellbar sind. Wenn nicht, erscheint eine Fehlermeldung, die uns zurück in die *MainView* bringt. Wenn ja, werden die Felder dynamisch generiert.

Dementsprechend bietet diese Klasse Methoden zum Abbrechen, Zurücksetzen, Pausieren und Beantworten der Activity.

Listing 8.3: Die Methode *onDestroy*

```
1
2 public void onDestroy() {
3     abort = true;
4     resetAbschicken();
5     super.onDestroy();
6 }
```

Sollte die App geschlossen werden, während wir in dieser Android Activity sind, ruft Android für uns die Methode *onDestroy* auf. Diese haben wir überschrieben (siehe Listing 8.3) und leiten vor der Zerstörung noch ein Pausieren der Activity ein. So können wir verhindern, dass ein Prozess durch das Schließen der App nicht mehr weiter ausführbar ist.

8.2.2 Package aristaFlowCommunication

Einen großen Teil der Arbeit nimmt das für die Kommunikation zuständige Package ein. Sehen wir uns die Klassen in diesem Package etwas genauer an.

8.2.2.1 Proxy

Die wichtigste Klasse in diesem Package stellt die *Proxy.java* dar. Über diese Stellvertreter Klasse wird die Kommunikation von außen aufgerufen, daher bestimmt sie die ganze Kommunikation. Wichtige Attribute sind die *Com* Klasse (8.2.2.2 auf Seite 33), die *XmlGen* Klasse (8.2.2.5 auf Seite 34), eine *LinkedList* mit *WorklistItem*'s und eine *LinkedList* mit *InstContGetTemplateReferenceResponse* Objekten.

availableRoles Bevor man sich einloggen und dann mit einem *Token* identifiziert werden kann, muss man erst einmal erfahren, welche Rollen für die User Passwort Kombination, die wir haben, verfügbar sind. Diese Aufgabe übernimmt die Funktion *availableRoles*. Der Server antwortet uns mit einer Reihe von qualifizierten Agenten, welche in der Funktion in einen String geschrieben werden und an den Aufrufer zurückgegeben wird, der nun die passende Rolle auswählen kann.

loginWithRole Sobald der User sich für eine Rolle entschieden hat, wird er mit dieser Funktion authentifiziert. Der Server gibt uns ein *Sessiontoken* zurück, mit dem die Autorisierung in der restlichen Sitzung vonstatten gehen wird.

listOfExecutableActivities Sobald man mit dem Server verbunden ist, können wir uns über diese Funktion, die gerade laufenden Aktivitäten, die wir mit dieser Rolle ausführen dürfen, holen. Diese Antwort spalten wir in mehrere *WorklistItems* auf, welche in einer *LinkedList* zurückgegeben werden.

getInstantiableTemplates Diese Funktion gibt uns alle Prozesse, die wir starten können, als *LinkedList* mit *InstContGetTemplateReferenceResponse* Objekten zurück.

createNewProcess Diese Funktion erlaubt uns, einen neuen Prozess zu starten. Hierzu müssen wir ihr übergeben, an welcher Stelle der *LinkedList* aus *getInstantiableTemplates* der gewünschte Prozess ist. Nun stellt sie die passende XML zusammen, schickt diese an den Server und gibt uns ein *Boolean* zurück, welches uns darüber informiert, ob auf Serverseite eine *Exception* stattfand.

8 Implementierung

startActivity, cancelProcess, answerActivity, suspendActivity und resetActivity Wenn wir eine Aktivität bestimmt haben, mit der wir etwas durchführen wollen, haben wir hierfür mehrere Möglichkeiten.

- Eine Aktivität starten: Um eine Aktivität zu starten, können wir *startActivity* entweder ein *WorkListItem* übergeben oder aber wir übergeben die Stelle an der das *WorkListItem* in der *LinkedList* steht. Nun kann die Aktivität nur noch von uns ausgeführt werden. Als Antwort bekommen wir die Details des Formulars, die wir zur Beantwortung benötigen.
- Eine Aktivität beantworten: Sobald der User das Formular beantwortet hat, können wir mit *answerActivity* diese Daten an den Server schicken.
- Eine Aktivität abbrechen: Wenn wir mit *resetActivity* eine Aktivität abbrechen, kann jeder andere berechtigte User diese Aktivität wieder ausführen. Der Prozess wird nicht abgebrochen!
- Einen zugehörigen Prozess abbrechen: Mit *cancelProcess* wird der gesamte Prozess abgebrochen. Dieser Funktion können wir optional auch noch einen Grund für den Abbruch übergeben, der nun in AristaFlow gespeichert ist.
- Eine Aktivität pausieren: Sollten wir uns entscheiden, die Aktivität später erst ausführen zu wollen, können wir dies mit *suspendActivity* erreichen. Für andere User ist diese Aktivität nun nicht startbar, wollen wir die Bearbeitung fortsetzen, können wir dies mit *startActivity* erreichen.

getImageOfProcess Dieser Funktion übergeben wir die *UUID* des Prozess, woraufhin wir eine *GetInstanceResponse* zurückbekommen. Diese beinhaltet nun ein Bild des Prozesses.

logOff Mit *logOff* kann man sich vom Server abmelden. Nun sollte das *Token* nicht mehr funktionieren.

Andere Funktionen

- Die Funktion *readObject*: Diese Funktion wird benötigt, um das *ProxyObjekt* beim Deserialisablen wieder komplett herstellen zu können. Das *Com* Objekt hat als Attribut eine spezielle Verbindung, die nach dem Speichern natürlich nicht mehr gültig ist. Dies können wir mit dieser Methode abfangen und ein neues Objekt erstellen.
- Die Funktion *log*: Einfache Log Funktion für die Debugausgabe.
- Die Funktion *valid*: Hier wird überprüft, ob das *Proxy* Objekt alle wichtigen Variablen instanziiert hat und benutzbar ist.
- Die Funktion *isFail*: Diese Funktion wird bei jeder Antwort benutzt, um vor der Verarbeitung sicherzustellen, dass Serverseitig keine Exception stattfand.

8.2.2.2 Com

In dieser Klasse findet die eigentliche Kommunikation statt.

sendMessage In dieser Funktion erstellen wir eine *HttpURLConnection* zu unserem Server. Wir übergeben eine IP, eine Pfadangabe, die SOAPAction sowie eine XML in Textform. Dies wird an den Server geschickt und die Antwort an den Aufrufenden zurückgegeben.

8.2.2.3 Configuration

Die statische *Configuration* Klasse hat keine Funktionen, lediglich einige Voreinstellungen sind hier als Attribute hinterlegt.

Neben den IPs zum AristaFlow Server sowie dem AristaFlow *ProcessImageRenderer* Server, liegen hier die Pfade zu den einzelnen benutzten Managern von AristaFlow sowie eine voreingestellte User/ Passwort Kombination.

8.2.2.4 Namespaces

Durch die relativ komplexe Namespace Situation in *XmlGen.java* wurden alle Namespacenamen sowie *NamespaceUris* in diese Klasse ausgelagert (siehe Listing 8.4 als Beispiel).

Listing 8.4: Eintrag für *WorklistManager*

```
1
2     private static String wlm = "wlm:";
3     private static String wlmUrl =
4     "http://aristaflow.de/adept2/xml1/core/worklistmanager";
5
6     public static String getWlm() {
7         return wlm;
8     }
9     public static String getWlmUrl() {
10        return wlmUrl;
11    }
```

8.2.2.5 XmlGen

In dieser Klasse werden die erforderlichen XML Strukturen erstellt.

createBackbone Die Funktion *createBackbone* erstellt das Grundgerüst (siehe Listing 8.5) für alle anderen XML generierenden Methoden in dieser Klasse. Alle Namespaces werden bei jede XML als Attribut dem *Envelope Element* mitgegeben. Dies hat den Vorteil, dass diese konsistent sind. Außerdem erleichtert es die Anpassung bei Änderungen erheblich.

Listing 8.5: Grundgerüst der XML Dateien

```
1
2 <env:Envelope
3     xmlns:env=" http://schemas.xmlsoap.org/soap/envelope/"
4     xmlns:omm=" http://aristaflow.de/adept2/xml1/core/
5         orgmodelmanager"
6     xmlns:ssm=" http://aristaflow.de/adept2/xml1/base/
7         sessionmanagement "
8     xmlns:bas=" http://aristaflow.de/adept2/xml1/basic"
```


8.2 Ausgewählte Implementierungsaspekte

```
7   xmlns:wlm=" http://aristaflow.de/adept2/xml1/core/
      worklistmanager"
8   xmlns:exe=" http://aristaflow.de/adept2/xml1/core/
      executionmanager"
9   xmlns:rts=" http://aristaflow.de/adept2/xml1/core/
      runtimeservice"
10  xmlns:psm=" http://aristaflow.de/adept2/xml1/model/
      processmodel"
11  xmlns:rte=" http://aristaflow.de/adept2/xml1/model/
      runtimeenvironment"
12  xmlns:com=" http://aristaflow.de/adept2/xml1/model/common"
13  xmlns:grp=" http://aristaflow.de/adept2/xml1/model/graphical"
14  xmlns:pir=" http://aristaflow.de/adept2/xml1/core/
      processimagerenderer"
15 >
16 <env:Body>
17 </env:Body>
18 </env:Envelope>
```

createAuth Wenn dieser Funktion einen Username und ein Passwort übergeben wird, erstellt sie innerhalb des Grundgerüsts ein *secMgrAuthenticateName* Element. Der Server antwortet auf diese Nachricht mit den verfügbaren *orgPositionIDs*. Übergibt man als dritte Variable nun noch diese *orgPositionID*, wird eine XML erzeugt, mit welcher man sich bei dem AristaFlow Server anmelden kann.

getSession *getSession* erzeugt ein *session*-Element (siehe Listing 8.6). Dies wird nach der Anmeldung in jedem XML benötigt.

Listing 8.6: Das *Session* Objekt

```
1 <exe:session>
2 <ssm:sessionId>18b33a86-8d9c-4df4-a1f6-3720726185e4</ssm:
3   sessionId>
```

8 Implementierung

```
4 <ssm:token><!-- Zur besseren Betrachtbarkeit wurde hier der
   Token entfernt --></ssm:token>
5 <ssm:callingComponent>
6   <bas:uri>http://pubweb/client</bas:uri>
7 </ssm:callingComponent>
8 </exe:session>
```

Das *Token* ist eine 1340 Zeichen lange Zeichenkette, mit welcher wir die Session verifizieren.

Der Funktionskopf wird in Listing 8.7 gezeigt.

Listing 8.7: Funktionskopf von *getSession*

```
1 private Element getSession(Document doc, String sessionId, String
   token, String ns, String name)
```

AristaFlow verlangt hier von uns unterschiedlichste Namespaces für das umschließende *Session* Element. Deswegen wird der Funktion mit übergeben, welcher Namespace benutzt werden soll. Da es zusätzlich auch noch ein *Token* Element gibt, welches bis auf den Namen des äußersten Elementes gleich ist, übergeben wir zusätzlich auch noch, wie das äußerste Element heißen soll.

Wenn wir eine neue Session aufbauen, müssen wir zusätzlich noch eine *UUID* für die *sessionId* generieren. Dies führt dazu, dass wir drei Helferfunktionen benötigen (siehe Listing 8.8).

Listing 8.8: Drei Helferfunktionen

```
1
2 private Element createSession(Document doc, String token) {
3   UUID t = UUID.randomUUID();
4   return getSession(doc, t.toString(), token, Namespaces.getWlm());
5 }
6 private Element getToken(Document doc, String token, String
   sessionId, String ns) {
7   return getSession(doc, sessionId, token, ns, "token");
8 }
```

8.2 Ausgewählte Implementierungsaspekte

```
9 private Element getSession(Document doc, String sessionId, String  
    token, String ns) {  
10     return getSession(doc, sessionId, token, ns, "session");  
11 }
```

createCreateClientWorklistNoTL Mit dieser Funktion kann man am Server abfragen, welche Aktivitäten momentan von dem angemeldeten User ausgeführt werden können.

createGetInstancelImage Die *createGetInstancelImage* Funktion sendet eine Anfrage an den *ProcessImageRenderer*, um ein Bild einer Aktivität zu bekommen.

createInstContCreateAndStartInstance In *createInstContCreateAndStartInstance* wird ein XML erstellt, mit welcher wir einen neuen Prozess starten können.

createInstContGetInstantiableTemplates Mithilfe von *createInstContGetInstantiableTemplates* erfahren wir die *templateIDs* der Templates, welche wir starten können.

createInstContGetTemplateReference Durch die in *createInstContGetInstantiableTemplates* erfahrene *templateID* kann die Funktion *createInstContGetTemplateReference* ein XML erzeugen welches mehr Informationen (zum Beispiel die für Menschen lesbaren Namen des Templates) erfahren.

createInstContStopAndAbortInstance Wollen wir einen Prozess beenden, müssen wir zur Generierung der XML diese Funktion aufrufen. Übergeben wird unter anderem ein Grund für den Abbruch. Als Grund ist auch ein leerer String möglich.

createLogon Erstellt eine XML, mit der man sich einloggen kann.

createLogoff Erstellt eine XML, um sich abzumelden.

8 Implementierung

createRemActWhateverSomeActivity Wenn wir eine Aktivität starten oder wiederaufnehmen wollen, haben wir bis auf den Namen des äußersten Elements die gleiche XML. Deswegen nutzen *createRemActStartResumeActivity* und *createRemActStartStartActivity* die *createRemActWhateverSomeActivity*, um die in Listing 8.9 gezeigte XML zu erstellen.

Listing 8.9: Von *createRemActWhateverSomeActivity* erstellte XML

```
1
2 <rts:remActStartStartActivity>
3   <rts:session>
4     <ssm:sessionID>092a66b4-644d-45ab-9858-b3b09d364d6a</ssm:
5       sessionID>
6     <ssm:token><!-- Zur besseren Betrachtbarkeit wurde hier der
7       Token entfernt --></ssm:token>
8     <ssm:callingComponent>
9       <bas:uri>http://pubweb/client</bas:uri>
10    </ssm:callingComponent>
11  </rts:session>
12 <rts:activity>
13   <psm:ebpType>ACTIVITY</psm:ebpType>
14   <psm:instanceID>35d522e0-35f4-4398-82a3-a5484e21dae5</psm:
15     instanceID>
16   <psm:baseTemplateID>091775e1-f9df-4720-9d98-06c7dac11be6</psm:
17     baseTemplateID>
18   <psm:nodeID>2</psm:nodeID>
19   <psm:iteration>0</psm:iteration>
20   <psm:activity>true</psm:activity>
21   <psm:emURIs>
22     <bas:uri>mina-xs://127.0.0.1:47999/ExecutionManager/
23       ExecutionManager</bas:uri>
24     <bas:uri>http://192.168.178.174:8088/AristaFlowWebService/
25       ExecutionManager/ExecutionManager</bas:uri>
26   </psm:emURIs>
27 </rts:activity>
28 <rts:executionMode>PRODUCTION</rts:executionMode>
29 <rts:timeout>30000</rts:timeout>
```

24 `</rts:remActStartStartActivity>`

createRemRunEnvApplicationWhatever Wenn wir eine Aktivität gestartet haben, gibt es drei Möglichkeiten, was wir weiter ausführen können. Wir können sie Schließen, Zurücksetzen oder Pausieren. Auch hier kommt bei diesen drei Fällen exakt die gleiche Struktur zum Vorschein. Wieder ist lediglich das äußerste Element in seinem Namen unterschiedlich. Deswegen benutzen *createRemRunEnvApplicationClosed*, *createRemRunEnvApplicationReset* und *createRemRunEnvApplicationSuspended* die Funktion *createRemRunEnvApplicationWhatever*, um die Antworten zu erstellen und übergeben hierbei den Namen des äußersten Elements.

8.2.2.6 String2Xml

String2Xml ist dafür zuständig, Antwortstrings in leichter bearbeitbare *Document* Objekte umzuwandeln.

8.2.2.7 Von AristaFlow verwendete Manager

Je nachdem, welche Aktion wir durchführen wollen, benutzen wir unterschiedliche Endpunkte des AristaFlow Web Services. In Tabelle 8.1 sehen wir eine Übersicht der verwendeten Manager.

8.2.3 Package `aristaFlowCommunication.responses`

Nachdem sich im letzten Package die *XmlGen* Klasse ausschließlich um die Erstellung von Xml Dokumenten gekümmert hat, kümmern sich alle Klasse in diesem Package um die Verarbeitung der Antworten des AristaFlow Servers.

Die erste Klasse, *UpdMgrCreateClientWorklistNoTLResponse*, wird ausführlicher behandelt, um das Prinzip in den restlichen Klassen zu veranschaulichen.

8 Implementierung

Manager in AristaFlow	Verwendung
ExecutionManager	Wird benutzt, um eine Liste der startbereiten Prozesse zu bekommen, einen neuen Prozess zu starten oder zu beenden.
OrgModelManager	Wird benutzt, um den Login Vorgang auszuführen.
ProcessImageRenderer	Wird benutzt, um ein Bild des Prozesses vom Web Service zu holen.
RuntimeServiceWorklistManager	Wird benutzt, um eine Aktivität zu starten, zu beantworten, zu pausieren oder abubrechen.
WorklistManager	Wird zum Ein- und Ausloggen in die Worklist und zum Holen einer Liste aller ausführbarer Aktivitäten benutzt.

Tabelle 8.1: Übersicht über die verwendeten Manager

8.2.3.1 UpdMgrCreateClientWorklistNoTLResponse

In Listing 8.10 sehen wir Ausschnitt der XML, die uns der Server als Antwort auf *UpdMgrCreateClientWorklistNoTL* schickt. Diese Informationen bringen wir nun durch diese Klasse in eine Form mit der wir besser arbeiten können.

Listing 8.10: Ausschnitt aus der *UpdMgrCreateClientWorklistNoTL* Antwort

```
1 <updMgrCreateClientWorklistNoTLResponse xmlns="http://aristaflow.
  de/adept2/xml1/core/worklistmanager">
2   <return>
3     <worklistID xmlns="http://aristaflow.de/adept2/xml1/model/
      worklistmodel">a9723878-b753-40b6-95fd-37fcab4e2e79</
      worklistID>
4     <agent xmlns="http://aristaflow.de/adept2/xml1/model/
      worklistmodel">
5       <sm:agentID xmlns:sm="http://aristaflow.de/adept2/xml1/base/
        sessionmanagement">20</sm:agentID>
6       <sm:agentUserName xmlns:sm="http://aristaflow.de/adept2/xml1
        /base/sessionmanagement">test</sm:agentUserName>
7       <sm:orgPositionID xmlns:sm="http://aristaflow.de/adept2/xml1
        /base/sessionmanagement">1</sm:orgPositionID>
```

8.2 Ausgewählte Implementierungsaspekte

```
8      <sm:orgPositionName xmlns:sm="http://aristaflow.de/adept2/
      xml1/base/sessionmanagement">Praktikant</sm:
      orgPositionName>
9
10     </agent>
11     <revision xmlns="http://aristaflow.de/adept2/xml1/model/
      worklistmodel">613</revision>
12     <worklistItems xmlns="http://aristaflow.de/adept2/xml1/model/
      worklistmodel" xmlns:wlm=" http://aristaflow.de/adept2/
      xml1/model/worklistmodel">
13         <!-- Hier folgen die workListItems -->
14     </worklistItems>
15     <worklistUpdateConfiguration xmlns="http://aristaflow.de/
      adept2/xml1/model/worklistmodel">
16         <updateModeThreshold>2147483647</updateModeThreshold>
17         <incrementalUpdates>>false</incrementalUpdates>
18     </worklistUpdateConfiguration>
19 </return>
</updMgrCreateClientWorklistNoTLResponse>
```

In der Klasse gibt es nun Attribute für alle Elemente in dieser XML. Es werden auch Informationen gespeichert, die wir nicht zum Arbeiten brauchen, dies erleichtert Erweiterungen später sehr gut.

Listing 8.11: Ausschnitt aus *UpdMgrCreateClientWorklistNoTLResponse.java*

```
1 public UpdMgrCreateClientWorklistNoTLResponse(Document doc) {
2     XmlGen xg = new XmlGen();
3     String test = xg.toString(doc);
4     try {
5         worklistID = doc.getElementsByTagName("worklistID").item(0).
            getTextContent();
6         revision = doc.getElementsByTagName("revision").item(0).
            getTextContent();
7         updateModeThreshold = doc.getElementsByTagName("
            updateModeThreshold").item(0).getTextContent();
```

8 Implementierung

```
8     incrementalUpdates = doc.getElementsByTagName ("
          incrementalUpdates").item(0).getTextContent();
9     qA = new QualifizierteAgenten((Element) doc.
          getElementsByTagName("agent").item(0));
10    NodeList worklistItems = doc.getElementsByTagName ("
          worklistItem");
11    for (int i = 0; i < worklistItems.getLength(); i++) {
12        Element temp = (Element) worklistItems.item(i);
13        WorklistItem wli = new WorklistItem(temp);
14        ll.add(wli);
15    }
16 } catch (Exception e) {
17     System.out.println(e.getMessage());
18     System.out.println("Exception");
19 }
20 }
```

Wir fragen jedes Element der XML ab und speichern es als String (siehe Listing 8.11). Der qualifizierte Agent und die *WorklistItems* bekommen eigene Klassen, da wir die Strukturen öfters brauchen werden. Da die Liste der ausführbaren Aktivitäten mehr als ein Element enthalten kann, speichern wir diese in einer *LinkedList*.

Alle Parameter sind über Getter Funktionen aufrufbar.

8.2.3.2 WorklistItem

Ein *WorklistItem* (siehe Listing 8.12) repräsentiert eine Aktivität eines gestarteten Prozesses, die wir starten können.

Listing 8.12: Ein *WorklistItem* Element

```
1 <worklistItem xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
   " xsi:type="wlm:ClientWorklistItem">
2   <id>6d7d3fc5-3813-4a1c-a0a7-68ee4c0a09c6</id>
3   <title>#2</title>
4   <description/>
```


8.2 Ausgewählte Implementierungsaspekte

```
5 <activationDate>1350569082682</activationDate>
6 <activityReference>
7   <type>de.aristaflow.ADEPT2ActivityReference</type>
8   <afEBPReference xmlns:wlm="http://aristaflow.de/adept2/xml1/
9     model/worklistmodel" xsi:type="wlm:AFActReference">
10    <ebpType xmlns="http://aristaflow.de/adept2/xml1/model/
11      processmodel">ACTIVITY</ebpType>
12    <instanceID xmlns="http://aristaflow.de/adept2/xml1/model/
13      processmodel">35d522e0-35f4-4398-82a3-a5484e21dae5</
14      instanceID>
15    <baseTemplateID xmlns="http://aristaflow.de/adept2/xml1/
16      model/processmodel">091775e1-f9df-4720-9d98-06c7dac11be6<
17      /baseTemplateID>
18    <nodeID xmlns="http://aristaflow.de/adept2/xml1/model/
19      processmodel">2</nodeID>
20    <iteration xmlns="http://aristaflow.de/adept2/xml1/model/
21      processmodel">0</iteration>
22    <activity xmlns="http://aristaflow.de/adept2/xml1/model/
23      processmodel">true</activity>
24    <emURIs xmlns="http://aristaflow.de/adept2/xml1/model/
25      processmodel">
26      <bsc:uri xmlns:bsc="http://aristaflow.de/adept2/xml1/basic
27        ">mina-xs://127.0.0.1:47999/ExecutionManager/
28        ExecutionManager</bsc:uri>
29      <bsc:uri xmlns:bsc="http://aristaflow.de/adept2/xml1/basic
30        ">http://192.168.178.174:8088/AristaFlowWebService/
31        ExecutionManager/ExecutionManager</bsc:uri>
32    </emURIs>
33    <rmURIs xmlns="http://aristaflow.de/adept2/xml1/model/
34      processmodel">
35      <bsc:uri xmlns:bsc="http://aristaflow.de/adept2/xml1/basic
36        ">mina-xs://127.0.0.1:47999/RuntimeService/
37        RuntimeService</bsc:uri>
```

8 Implementierung

```
21     <bsc:uri xmlns:bsc="http://aristaflow.de/adept2/xml1/basic
      ">http://192.168.178.174:8088/AristaFlowWebService/
      RuntimeService/RuntimeService</bsc:uri>
22 </rmURIs>
23 <execCtrlProp>
24     <cmn:suspendable xmlns:cmn="http://aristaflow.de/adept2/
      xml1/model/common">true</cmn:suspendable>
25     <cmn:resettable xmlns:cmn="http://aristaflow.de/adept2/
      xml1/model/common">true</cmn:resettable>
26     <cmn:closable xmlns:cmn="http://aristaflow.de/adept2/xml1/
      model/common">true</cmn:closable>
27 </execCtrlProp>
28 <ecName>de.aristaflow.form.GeneratedForm</ecName>
29 <guiContextID>SWTContext</guiContextID>
30 <viewOnly>true</viewOnly>
31 </afEBPReference>
32 </activityReference>
33 <complexity>0</complexity>
34 <priority>4000</priority>
35 <processInstanceName>typTest (18.10.12 16:04)</
      processInstanceName>
36 <processTemplateName>typTest</processTemplateName>
37 <state>AVAILABLE</state>
38 <worklistID>a9723878-b753-40b6-95fd-37fcab4e2e79</worklistID>
39 <individualPriority>0</individualPriority>
40 </worklistItem>
```

Besonders Interessant sind folgende Elemente:

id Die *id* ermöglicht uns eine eindeutige Identifizierung der Aktivität.

title und description Diese beiden Elementen helfen dem User zu erklären, was für eine Aktivität sich hinter der *id* verbirgt.

processInstanceName und processTemplateName Dies ermöglicht den Rückschluss auf den Prozess, zu dem diese Aktivität gehört. Der *processInstanceName* gibt uns die Möglichkeit, diese Instanz des Prozesses zu identifizieren, der *processTemplateName* zeigt uns den Namen der Vorlage.

state Je nachdem in welchem Status die Aktivität sich befindet, müssen wir unterschiedlich auf sie reagieren. *AVAILABLE* und *SUSPENDED* können wir mit den entsprechenden Methoden starten, einen anderen Status können wir nicht verarbeiten.

8.2.3.3 GetInstanceResponse

Damit der User sich besser im Prozess zurechtzufinden, hat er die Möglichkeit, sich ein Bild des Prozesses anzuschauen. In *GetInstanceResponse* verarbeiten wir die Antwort auf eine solche Anfrage (siehe Listing 8.13).

Listing 8.13: Variablen aus der Klasse *GetInstanceResponse.java*

```
1 private String base64Png;  
2 private int x, y, height, width;  
3 private boolean relative;
```

Wir bekommen von dem *ProcessImageRenderer* eine (von uns so angefragte) PNG in base64 codiert. Für unsere Anwendung sind sonst nur noch die Höhe und Breite interessant, die Variablen *x*, *y* und *relative* nutzen wir nicht.

8.2.3.4 InstContCreateAndStartInstanceResponse

In dieser Methode holen wir uns aus der Antwort-XML nur die für uns interessante *instanceID*.

8.2.3.5 InstContGetInstantiableTemplatesResponse

Will der User einen neuen Prozess starten, schicken wir eine *InstContGetInstantiableTemplates* Anfrage an den Server. In der Antwort bekommen wir eine Reihe von *UUIDs* der verschiedenen startbaren Prozesse. Was wir mit diesen weiteres durchführen können, sehen wir im nächsten Punkt, *InstContGetTemplateReferenceResponse*.

8 Implementierung

8.2.3.6 InstContGetTemplateReferenceResponse

Da wir von der vorherigen Anfrage die *UUID* eines vom User startbaren Prozesses bekommen haben, können wir nun detailliertere Infos zu diesem Prozess-Template bekommen.

Listing 8.14: Gespeicherte Variablen in *InstContGetTemplateReferenceResponse*

```
1  private String id, processType, version, name, description,  
   topLevelUsable,  
2      usageAsSubprocess, buildtimeState, derivable,  
   tlOutdated,  
3      tlInstantiable, tlMigratableTo, tlChangeable;
```

Wir sehen nun die Version, den Namen, eine für den Menschen lesbare Beschreibung und viele andere nützliche Informationen (siehe Listing 8.14).

8.2.3.7 LogonResponse

Nach dem *Logon* bekommen wir durch die *LogonResponse* die *worklistID* übermittelt. Diese brauchen wir zum Abrufen der verfügbaren Aktivitäten.

8.2.3.8 QualifiedAgent

Da unser User verschiedene Rollen besitzen kann, läuft der Login in zwei Schritten ab. Im ersten Schritt übermitteln wir Nutzernamen und Passwort. Als Antwort erhalten wir eine Liste der qualifizierten Agenten (siehe Listing 8.15). Jeder dieser qualifizierten Agenten repräsentiert eine Rolle, welche wir einnehmen können. Nachdem wir uns eine oder mehrere dieser Rollen ausgesucht haben, können wir den Login beenden.

Listing 8.15: Variablen in der *QualifiedAgent.java*

```
1  private String id = "";  
2  private String agentUserName = "";  
3  private String orgPositionID = "";  
4  private String orgPositionName = "";
```

8.2.3.9 RemActStartStartActivityResponse und -Parameter

Wenn der User eine Aktivität in dem Android Client gestartet hat, bekommt er als Antwort eine *RemActStartStartActivityResponse*. Diese bietet uns nun alle nötigen Informationen (siehe Listing 8.16), um diese Aktivität zu bearbeiten.

Listing 8.16: Variablen der *RemActStartStartActivityResponse* Klasse

```

1  private String sessionID, ebpType, instanceID, baseTemplateID,
    nodeID, iteration,
2    dataContext2, activityInstance, templateName, instanceName,
3    instanceLogID, name, description, execCtrlProp, dataContext1
    , id,
4    ecName, opName, confName, actName, implClass,
    systemComponent,
5    guiContextID, supportsViewOnly;
6  private boolean activity, readOnly, suspensible, resettable,
    closable,
7    shareClassLoader, testMode, viewOnly;
8  private LinkedList<RemActStartStartActivityResponseParameter>
    llParameter = new LinkedList<
    RemActStartStartActivityResponseParameter> ();
9  private Element dataContext;
```

Besonders interessant sind für uns die Informationen, die wir dem User zeigen müssen. Neben den Variablen *name* und *description* ist hier auch die *LinkedList* mit *RemActStartStartActivityResponseParameter* Objekten sehr interessant.

Diese *RemActStartStartActivityResponseParameter* Objekte enthalten die einzelnen Felder der Aktivität (siehe Listing 8.17).

Listing 8.17: Variablen der *RemActStartStartActivityResponseParameter* Klasse

```

1  private String name;
2  private String description;
3  private String type;
4  private String identifierID;
5  private String instanceID;
```

8 Implementierung

```
6  private boolean optional;
7  private boolean virtual;
8  private boolean failure;
9  private String parent, inhalt;
10 private String tag = " RASSARparameter";
11 private boolean output;
12 private Integer min = null;
13 private Integer max = null;
14 private Integer pos = null;
```

Neben dem Namen und der Beschreibung erfahren wir wichtige Parameter, wie den Typ der Variable (*type*), den Namen der Eltern-Node (*parent*), unter Umständen den Inhalt (*inhalt*), und ob wir überhaupt etwas beantworten sollen (*output*). Als letztes folgen noch drei Attribute die wir in AristaFlow unter Umständen mitgeben wollen (*min*, *max*, *pos*).

Den Typ der Variable benötigen wir für die Eingabe. Da wir einen String anders eingeben wollen als ein Datum, müssen wir auf diese Variable speziell aufpassen.

Über den Namen der Eltern-Node erfahren wir, ob es ein *inputParameter* oder ein *outputParameter* ist. *InputParameter* bieten uns in AristaFlow die Möglichkeit, Daten vom Server dem User zu zeigen. Dies könnte zum Beispiel nach einem Fragebogen eine Zusammenfassung aller Fragen sein. Wenn es ein *inputParameter* ist, setzen wir *output* auf *true*, die GUI erkennt daran, dass das Feld für diese Variable nicht als Eingabe fungieren soll und wir es nicht aktivieren, sondern den Inhalt der Variable *inhalt* ausgeben.

Speziell für klinisch psychologische Studien haben wir die Parameter *min* und *max*. Diese ermöglichen uns dem User eine Frage zu stellen, in der er auf einer Skala zwischen *min* und *max* einen Wert auswählen muss. Die Variable *pos* gibt an, an welcher Position dieses Feld erscheinen soll. Hierfür gibt es auch noch eine Funktion *compareTo*, welche diesen *RemActStartStartActivityResponseParameter* mit einem anderen vergleicht.

8.2.3.10 SecMgrAuthenticateAgentNameResponse

Hier bekommen wir Infos über den User, mit dem wir uns angemeldet haben. Außerdem bekommen wir das Token für die Session.

8.2.3.11 SecMgrAuthenticateNameResponse

In dieser Klasse wird eine Liste von qualifizierten Agenten gespeichert. Aus diesen kann der User sich dann seine gewünschte Rolle aussuchen.

8.2.4 Package tools

Im *Tools* Package sind einige Helfer-Klassen, welche uns die Arbeit erleichtern.

8.2.4.1 ImageDialogTools

Diese Klasse stellt Hilfe für die Anzeige des Übersichtsbildes des Prozesses zur Verfügung (siehe Listing 8.18).

Listing 8.18: Variablen in der *ImageDialogTools* Klasse

```
1 private View.OnClickListener oclzi;  
2 private View.OnClickListener oclzo;  
3 private View.OnTouchListener otl;  
4 private ImageView iv;  
5 private int zoomlevel = 1;  
6 private int scrollX = 0;  
7 private int scrollY = 0;  
8 private ZoomControls zC;
```

Er stellt für die Zoom Buttons zwei *OnClickListener* (*oclzi* und *oclzo*) zur Verfügung, einen *OnTouchListener*, um das Bild verschieben zu können, *ZoomControls* und ein *ImageView* Objekt, in dem das Bild dargestellt wird. Um zu wissen, auf welcher Zoom Stufe wir sind, haben wir die Variable *zoomlevel*, die Position des Bildes wird durch *scrollX* und *scrollY* bestimmt.

Die Logik der *Listener* wird im Konstruktor festgelegt.

8.2.4.2 ProzessItemAdapter

ProzessItemAdapter erbt von *ArrayAdapter<WorklistItem>*. Ein *ArrayAdapter* ermöglicht uns die Darstellung und das Verhalten einer Liste in Android an unsere Bedürfnisse an-

8 Implementierung

zupassen. In unserem Fall wollen wir eine *LinkedList* mit *WorklistItems* (welche unsere Aktivitäten darstellen) übergeben und anzeigen lassen.

Listing 8.19: Entscheidung, welches Icon angezeigt wird

```
1 if (wli.getState().equals("AVAILABLE")) {
2     icon.setImageResource(R.drawable.notsuspended);
3 }else{
4     icon.setImageResource(R.drawable.suspended);
5 }
```

Es wird überprüft, ob die Aktivität gerade pausiert wurde und je nachdem wird das passende Icon angezeigt(siehe Listing 8.19). Android hat nun die Möglichkeit, mit der Methode *getView*, die Darstellung für jede Spalte abzurufen. Mittels eines *LayoutInflater* müssen wir nur noch die schon vorhandene Vorlage *listitem.xml* an das jeweilige *WorklistItem* anpassen.

8.2.4.3 Input

Ein *Input* Objekt stellt ein Feld einer Aktivität dar (siehe Listing 8.20).

Listing 8.20: Die *Input* Klasse

```
1
2 protected String getIdentifierID, vName, description;
3 protected EditText edit;
4 protected TextView tv;
5 protected LinearLayout lla;
6
7 public Input (RemActStartStartActivityResultResponseParameter para) {
8     this.getIdentifierID = para.getIdentifierID();
9     this.vName = para.getName();
10    this.description = para.getDescription();
11 }
12 public String getIdentifierID() {
13     return getIdentifierID;
14 }
```



```
15 public String getvName() {
16     return vName;
17 }
18 public LinearLayout getLayout() {
19     return lla;
20 }
21 public String getText() {
22     return edit.getText().toString();
23 }
24 protected String getAnzeigeName() {
25     return description;
26 }
```

Die *getIdentfierID* brauchen wir später, um dieses Objekt eindeutig einem Feld zuordnen zu können. Der Feldname (*vName*) und die Beschreibung (*description*) werden im *TextView* (*tv*) für den User angezeigt. Die *Textview* wird zusammen mit dem Eingabebereich (*edit*) in einem *LinearLayout* (*lla*) gespeichert. Dies kann nun von Android direkt angezeigt werden.

Diese Klasse wird jedoch nicht direkt eingebunden, sondern dient als Eltern Element für die speziellen Eingabe Klassen, die nun folgen.

8.2.4.4 InputBool

InputBool erbt von *Input* und erweitert *Input* zur Darstellung und Verarbeitung von *Booleans*. Die Darstellung erfolgt über ein *android.widget.Switch* Objekt. Die Methode *getText* der *Input* Klasse wird so überschrieben, dass der Status des *Switch* als *TRUE* oder *FALSE* zurückgegeben wird. Über einen *LayoutInflater* wird die XML Datei *eingabebool.xml* als Layout verwendet.

8.2.4.5 InputDate

InputDate erbt von *Input* und erweitert *Input* zur Darstellung und Verarbeitung von einem Datum. Wenn der User auf eines der Felder klickt, bekommt er entweder einen *DatePickerDialog* oder einen *TimePickerDialog* angezeigt, um die Eingabe zu erleichtern. Über einen *LayoutInflater* wird die XML Datei *eingabedate.xml* als Layout verwendet.

8 Implementierung

8.2.4.6 InputFloat und InputInt

Auch diese beiden Klassen erben von *Input*. Sobald auf das Feld geklickt wird, erscheint die Tastatur. Diese erlaubt aber nur die Eingabe eines *Float* / *Int*. Über einen *LayoutInflater* wird entweder die XML Datei *eingabeint.xml* oder *eingabefloat.xml* als Layout verwendet.

8.2.4.7 InputScale

InputScale erbt von *Input*. Zur Darstellung einer Skala zwischen zwei Werten werden *RadioButtons* mit allen Werten dazwischen erstellt. Zur Darstellung wird *inputscale.xml* verwendet.

8.2.4.8 InputString

Der einfachste Fall, die Eingabe eines *Strings*, wird auch von der *Input* Klasse abgeleitet.

8.2.4.9 InputUri

InputUri erbt von *InputString*. Der einzige Unterschied ist das *InputUri* sicherstellt, dass keine Leerzeichen in dem *String* vorkommen.

8.3 Probleme

Gerade bei der Arbeit mit dem AristaFlow Web Service kamen einige Probleme zum Vorschein.

Am Beispiel der *Session* Objekte (siehe 8.2.2.5) lässt sich das Problem der verschiedenen Namespaces verdeutlichen. Vom Aufbau komplett identische Objekte werden, je nachdem wo sie eingesetzt werden, mit verschiedenen Namespaces ausgestattet. Dies widerspricht der Idee von Namespaces in XML.

Auch gewöhnungsbedürftig war, dass der Web Service auf eine bestimmte Reihenfolge von Elementen in XML besteht. Ein *Session* Objekt, in welchem als erstes das *Token* und dann erst die *sessionID* steht, wird von dem Web Service nicht angenommen.

8.3 Probleme

Wenn eine Aktivität pausiert wird, übergeben wir alle schon ausgefüllten Parameter (siehe 8.2.2.5). Mit diesen Informationen scheint der Web Service aber nichts anfangen zu können. Wenn wir die Aktivität wieder aufnehmen, erhalten wir wieder die leeren Felder, wie nach dem ersten Start.

Die restliche Implementierung gelang ohne weitere Probleme.

9 Abgleich der Anforderungen

Wie wir in Tabelle 9.1 sehen können, wurden fast alle Anforderungen erfüllt. Zwei Anforderungen wurden jedoch nur teilweise erfüllt, auf diese wird nun näher eingegangen.

Verbindungsabbrüche Wenn während der Ausführung der App ein Verbindungsabbruch erfolgt, wird normalerweise eine für sich sprechende Fehlerbehandlung vorgenommen. Es erscheint eine Fehlermeldung, und sobald die Verbindung wieder hergestellt werden konnte, wird die App weitergeführt.

Problematisch wird es aber, wenn wir eine Activity gestartet haben, einen länger dauernden Verbindungsabbruch haben und Android die App aus dem Arbeitsspeicher entfernt. Die Activity kann jedoch nicht beendet werden und ist nun nicht mehr ausführbar, ohne das Eingreifen einer berechtigten Person auf dem AristaFlow Server.

Dies lässt sich umgehen, indem die App eine Funktion bekommt, welche sich merkt, welche Activity gerade ausgeführt wird. Bricht nun die Verbindung ab, so kann die Activity pausiert werden.

Grafische Anzeige Android limitiert die Größe und Breite von Bildern auf 2048x2048 Pixeln. Hieraus ergibt sich bei sehr langen Prozessen (wie zum Beispiel auch dem KINDEX Mum Screen) die Notwendigkeit, die Bilder, welche wir vom *ProcessImageRenderer* bekommen, auf weniger Pixel zu skalieren. Diese klein skalierten Bildern sind für den User unter Umständen nicht mehr benutzbar.

Dieses Problem wäre lösbar, indem man die Bilder sinnvoll für den Kontext beschneidet.

Beide Probleme sind glücklicherweise nicht im Kontext dieser Arbeit störend, da die Umsetzung des KINDEX deswegen trotzdem möglich ist und eine Evaluierung ferner auch möglich bleibt. In zukünftigen Arbeiten müssen diese Probleme aber speziell behandelt werden.

9 Abgleich der Anforderungen

Anforderung	Anforderungsart	
Multi-User-Unterstützung	funktional	■
Web Service orientierte Kommunikation mit AristaFlow Servern	funktional	
Verschiedene Serverendpunkte sollen verwendet werden können	funktional	
Verbindungsabbrüche müssen verarbeitet werden können	funktional	
<i>Mobiler Klient: Funktionalität - Prozesse</i>		
Ein in Aristaflow modellierter formularbasierter Prozess soll ohne weitere Änderungen von der App ausgeführt werden können	funktional	■
Neue Prozesse müssen gestartet werden können	funktional	
Laufende Prozesse müssen abgebrochen werden können	funktional	
<i>Mobiler Klient: Funktionalität - Aktivitäten</i>		
Übersicht über von diesem User bearbeitbare Aktivitäten	funktional	■
Aktivität soll gestartet, pausiert sowie resettet werden können	funktional	
Grafische Anzeige über Fortschritt und gerade ausführbare Aktivitäten im Prozess	funktional	
<i>Psychologische Anforderungen</i>		
Spezielle Formularanpassung an die Bedürfnisse für eine klinisch psychologische Studie	funktional	■
Soll flexibel für unterschiedlichste Szenarien einsetzbar sein	funktional	
Geringe Zeiträume zwischen Eingaben	nicht funktional	■
App soll leicht erweiterbar sein	nicht funktional	
App soll gut wartbar sein	nicht funktional	
Es soll ein ansprechende GUI umgesetzt werden	nicht funktional	
Leicht verständliche Eingabemöglichkeiten ohne lange Einlernzeiten	nicht funktional	

Tabelle 9.1: Abgleich der Anforderungen

10 Zusammenfassung

Die Nachteile von Martin Liebrechts Prototyp (keine Logik, nur KINDEX) konnten gut mit der hier vorgestellten Lösung adressiert werden. Die konkrete Umsetzung des KINDEX ist an manchen Stellen noch mit großem Verbesserungspotential zu sehen, so ist die Beschriftung der Skalen nur mit Zahlen nicht ideal gelöst. Die Bedeutung der Zahlen muss hier noch im Fragetext untergebracht werden und kann nicht, wie in Papierform, die Auswahl zwischen "nie" "selten" "manchmal" "häufig" und "sehr häufig" bieten. Dies wird durch die anderen Vorteile jedoch aufgewogen, wie zum Beispiel das nachträgliche Änderungen gut umsetzbar sind.

Im Vergleich zu Maximilian Schmidts Masterarbeit ist die Erstellung der Fragebögen etwas langwieriger. Auch hier sind aber die Vorteile der Arbeit mit dem AristaFlow BPMS gut zu sehen, da Abbildung von Logik möglich ist.

Die Zusammenarbeit mit Web Services bringt leider auch einige Nachteile mit sich.

Große Nachteile offenbaren sich bei der Offline-Nutzung. Nicht überall steht eine Verbindung zum Internet zur Verfügung, gerade in Entwicklungsländern, wo solche Studien oftmals durchgeführt werden, sollte man sich nicht auf diese verlassen müssen. Bei entsprechenden Szenarien müsste man sich hier entsprechende Lösungen erarbeiten.

Die Ansteuerung von AristaFlow würde auch über Java Remote Method Invocation gehen. Hier würde man sich das Verpacken, Verschicken und Entpacken der Anfragen in XML sparen und könnte direkter kommunizieren.

Unter Android würde das Sinn ergeben, sobald man jedoch ein anderes Betriebssystem (wie zum Beispiel iOS) nutzen möchte, müsste man wieder auf Web Services zurückgreifen. So wurde durch diese Arbeit eine breitere Grundlage geschaffen, welche man nun auch auf andere Systeme anwenden kann.

Auch die Umsetzung auf andere Workflow Systeme wäre machbar. Das Einsetzen von Business Process Management Systemen zur Unterstützung von klinisch psychologischen Studien hat sich als sehr sinnvoll erwiesen. Gerade durch das Zurückgreifen auf Logik können einige Probleme sehr schön durch eine gute Modellierung gelöst werden.

Literaturverzeichnis

- [1] *Android*. <http://www.android.com/>, 2012. – zuletzt besucht: 12.11.2012
- [2] *AristaFlow*. <http://www.aristaflow.com/>, 2012. – zuletzt besucht: 12.11.2012
- [3] *Babyforum Landkreis Konstanz*. <http://www.babyforum-landkreis-konstanz.de/>, 2012. – zuletzt besucht: 12.11.2012
- [4] *Java*. <http://www.java.com/>, 2012. – zuletzt besucht: 12.11.2012
- [5] *Universität Konstanz*. <http://www.psychologie.uni-konstanz.de/>, 2012. – zuletzt besucht: 12.11.2012
- [6] *vivo e.V.* <http://www.vivo.org/>, 2012. – zuletzt besucht: 12.11.2012
- [7] DADAM, P. ; REICHERT, M.: The ADEPT Project: A Decade of Research and Development for Robust and Flexible Process Support - Challenges and Achievements. In: *Computer Science - Research and Development* 23 (2009), Nr. 2, S. 81–97
- [8] GEIGER, P.: *Entwicklung einer Augmented Reality Engine am Beispiel des iOS*. September 2012
- [9] GRÜNING, J.: *Technische Konzeption und Realisierung der Laufzeitumgebung für ein generisches Fragebogensystem zur IT-gestützten Durchführung von evaluierten Studien der Klinischen Psychologie*. September 2012
- [10] LIEBRECHT, M.: *Technische Konzeption und Realisierung einer mobilen Anwendung für den Konstanzer-Index zur Erhebung von psychosozialen Belastungen während der Schwangerschaft*. Deutschland, Universität Ulm, Diplomarbeit, 2012
- [11] PRYSS, R. ; LANGER, D. ; REICHERT, M. ; HALLERBACH, A.: Mobile Task Management for Medical Ward Rounds - The MEDo Approach. In: *1st Int'l Workshop on Adaptive Case Management (ACM'12), BPM'12 Workshops*, Springer, September 2012 (LN-BIP)

Literaturverzeichnis

- [12] PRYSS, R. ; TIEDEKEN, J. ; KREHER, U. ; REICHERT, M.: Towards Flexible Process Support on Mobile Devices. In: *Proc. CAiSE'10 Forum - Information Systems Evolution*, Springer, 2010 (LNBIP 72), S. 150–165
- [13] PRYSS, R. ; TIEDEKEN, J. ; REICHERT, M.: Managing Processes on Mobile Devices: The MARPLE Approach. In: *CAiSE'10 Demos*, 2010
- [14] REICHERT, M.: *Dynamische Ablaufänderungen in Workflow-Management-Systemen*, Universität Ulm, Dissertation, 2000
- [15] REICHERT, M. ; DADAM, P.: Adeptflex—Supporting Dynamic Changes of Workflows Without Losing Control. In: *Journal of Intelligent Information Systems* 10 (1998), S. 93–129. – ISSN 0925–9902
- [16] REICHERT, M. ; RINDERLE-MA, S. ; DADAM, P.: Flexibility in Process-aware Information Systems. In: *LNCS Transactions on Petri Nets and Other Models of Concurrency (ToPNoC), Special Issue on Concurrency in Process-aware Information Systems. 2* (2009), March, S. 115–135
- [17] REICHERT, M. ; WEBER, B.: *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies*. Berlin-Heidelberg : Springer, 2012
- [18] ROBECKE, A. ; PRYSS, R. ; REICHERT, M.: DBIScholar: An iPhone Application for Performing Citation Analyses. In: *CAiSE Forum-2011*, CEUR Workshop Proceedings, June 2011 (Proceedings of the CAiSE'11 Forum at the 23rd International Conference on Advanced Information Systems Engineering Vol-73)
- [19] RUF, M. ; SCHAUER, M.: *Erfassung von psychosozialen Risikofaktoren während der Schwangerschaft als Prädiktoren für Entwicklungsprobleme anhand des 'KINDEX' – ein validiertes Instrument für die alltägliche Praxis*. DGPPN Kongress, 2010
- [20] RUF, M. ; SCHAUER, M.: *External and prospective validity of the KINDEX Mum Screen – a short instrument for midwives and gynaecologists to assess psycho-social risk factors during pregnancy (Externale und Prospektive Validität des KINDEX Mum Screen – ein Instrument für die Erfassung psychosozialer Risiko-faktoren bei Schwangeren)*. 2012
- [21] SCHMID, M.: *Technische Konzeption und Realisierung der Anwenderumgebung für ein generisches Fragebogensystem zur IT-gestützten Durchführung von evaluierten Studien der Klinischen Psychologie*. Deutschland, Universität Ulm, Masterarbeit, 2012