Universität Konstanz
FB Informatik und Informationswissenschaft
Master-Studiengang Information Engineering

Masterarbeit

„Improving data-gathering in field studies by using electronic devices“

zur Erlangung des akademischen Grades eines
Master of Science (M.Sc.)

Studienfach: Information Engineering
Schwerpunkt: Mensch-Computer Interaktion
Themengebiet: Angewandte Informatik

von

Patric Schmid

(01/717761)

Erstgutachter: Prof. Dr. Harald Reiterer
Zweitgutachter: Prof. Dr. Marc Scholl
Einreichung: 30.03.2011

Kurzfassung

Je mehr die Produktnutzung von Konsumenten in den mobilen Kontext ihres Alltags rückt, desto wichtiger wird es auch für Forscher die Evaluation dieser Produkte in ihrem natürlichen Nutzungskontext durchzuführen. Tagebuchstudien, experience sampling und Logging sind prominente Methoden um Nutzungsinformationen zu erheben ohne als Forscher zum Erhebungszeitpunk vor Ort sein zu müssen. Durch moderne Smartphones ist es möglich, all diese Daten komprimiert über das Mobiltelefon eines Probanden zu erheben.

Diese Masterthesis beschreibt die Ursprünge und Entwicklung von „PocketBee“ - einem System zur methodischen Kombination der experience sampling method, Tagebuchstudien und Logging [9]. Im Folgenden wird der bestehende Ansatz überarbeitet und sowohl ein neues Trigger-Framework zur Verarbeitung der Bedingungen und Aktionen vorgestellt, als auch eine grafische Benutzeroberfläche präsentiert, die es den Forschern ermöglicht komplexe Studiendesigns zu erstellen.

Im Vorlauf dieser Thesis wurde Systemanforderungen erhoben, die durch eine State-of-the-Art Analyse von bereits existierenden ESM- und Tagebuchwerkzeugen sowie einer ausgiebigen Analyse der wissenschaftlichen Vorarbeit gestützt werden. Anhand dieser Anforderungen wurde ein Pilotsystem entwickelt, welches es aktuell ermöglicht ESM- und Tagebuchstudien zu planen und durchzuführen. Das bestehende System basiert auf einer Anwendung für Android-Smartphones sowie einem Servers mit einer webbasierten Oberfläche für den Forscher. Durch die praktische Durchführung zweier Pilotstudien mit diesem System konnten sowohl interessante Informationen über die Forschungsobjekte als auch wichtiges Systemfeedback zu PocketBee gewonnen werden. Während die erste der Studien innerhalb der Universität zum Thema „Behindertenfreundlichkeit“ durchgeführt wurde, handelte es sich bei der zweiten Studie um ein Projekt des Kooperationspartners Daimler AG , dessen Customer Research Center am Thema „Wohlbefinden im Fahrzeug“ interessiert war. Während diesen Studien zeigte sich das Potential des PocketBee Systems welches noch weiter ausgebaut werden soll.

Der Fokus dieser Thesis liegt in zwei großen Bereichen:

- Zum Einen die Architektur und Umsetzung eines neuen modulareren Trigger-Frameworks, welches sicherstellen soll, dass alle Systemfehler eliminiert werden die durch die bisherige Implementierung ausgelöst wurden. Die Herausforderung dabei war die Erstellung komplexer Condition-Chains[1] zu ermöglichen und dem Forscher somit die Möglichkeit zu geben eine Mischung aus Tagebuch- sowie ESM und Loggingstudien konfigurieren und durchführen zu können. Die Kombination und Verkettung von Bedingungen und Aktionen ermöglicht es

[1] Die Verkettung von Bedingungen mit Aktionen die aktiviert werden wenn die Bedingungen erfüllt sind.

dem Forscher jede denkbare Situation in einer Studie zu erkennen (z.B. es ist 13:30 Uhr, der Proband ist bei der Arbeit) und durch die Bedingungsevaluation des Systems auf diese Situation reagieren zu können (z.B. zeige neuen Fragebogen). Durch die Bereitstellung klarer Schnittstellen soll zusätzlich eine einfache Erweiterbarkeit mit neuen Sensoren, Bedingungen und Aktionen möglich sein.

- Zum Anderen wird das Konzept und prototypische Umsetzen einer grafischen Benutzeroberfläche beschrieben mit der ein Forscher möglichst einfach die zugrundeliegende Triggerkonfiguration vornehmen kann. Die Herausforderung hierbei war es, das Interface so zu gestalten, dass ein Forscher das Konzept der verketteten Bedingungen und Aktionen verstehen, erlernen und anwenden kann. Auf der Anderen Seite sollte das Interface die Modularität des Trigger-Frameworks wiederspiegeln um auch hier die Pflege neuer Sensoren, Bedingungen und Aktionen zu ermöglichen.

Das Trigger-Framework wird anhand den Anforderungen und Praxiserfahrungen so entworfen, dass eine einfache modulare Erweiterbarkeit sowie die methodische Verschmelzung von Tagebuchstudien, ESM und Logging gewährleistet werden kann. Dieses Framework kann in seinem Konzept auch für andere Plattformen (z.B. iOS oder Windows Mobile 7) umgesetzt werden. Während das Trigger-Framework die Grundlage des PocketBee Systems bildet, wird ein simples Interfacekonzept vorgestellt, welches dem Forscher die Konfiguration einer Studie erleichtern soll. Dieses Interfacekonzept verbindet ein Pipe/Filter Prinzip mit semantischem Zoom und weiteren unterstützenden Elementen. Durch das Pipe/Filter Prinzip können nicht nur die logischen Zusammenhänge von Bedingungen und Aktionen in einer Art Datenflussdiagramm dargestellt und verändert werden, das semantische Zoomen ermöglicht zusätzlich einen fließenden Wechsel zwischen Überblick und Detailansicht der einzelnen Bedingungen bzw. Aktionen. Dieses Interfacekonzept könnte auch als Grundstein für viele Andere ESM und Tagebuchwerkzeuge dienen, die häufig vor ähnlichen Herausforderungen stehen [19].

Abstract

In the current years consumers spend more and more time using their electronic devices within the mobile context of their life where they are “on the go”. For this matter the researchers who try to observe the usage behaviours of those consumers also need to move from their laboratories into the field. To save resources and manpower during an observation researchers are now able to use modern Smartphone technology for context-logging and querying experience samples. Those phones can also be used as multi modal diaries to gather more information on the subject’s experiences when they happen.

This thesis describes the origin and first development of PocketBee - a multi modal ESM and diary tool for field research [9]. While the first prototype of the system was already used in two pilot studies a re-design is proposed, which introduces a new generation of triggering framework as well as a user friendly graphical interface for the researcher. Both parts enable the researcher to configure studies using conditions and actions by putting them together into condition-chains.

Prior to this thesis a state-of-the-art analysis was performed which led to multiple findings about existing ESM and diary tools which were already developed by researchers [26]. In addition to this analysis and further theoretical research two field-studies were conducted using the pilot system of PocketBee which was already developed as a prototype for this purpose [25].

As already mentioned this thesis provides two main contributions to the field:

- The first contribution is the new architecture of the trigger-framework which represents the backbone of the system. It was designed to match requirements like “combination of ESM, diary and logging studies” and “easy to extend with future sensors”. The challenge here was to give researchers a tool for building complex concatenation chains of conditions and actions which then represent the aspired study design within the system. These condition-chains are as modular as possible, while simultaneously being as flexible as possible to combine several study designs.
- The second contribution is a user interface including an interaction design which enables the researcher to create this complex condition-chains within an easy graphical context. Here the user can arrange and concatenate conditions and actions using the simple drag and drop technique. To furthermore extend the used Pipe/Filter concept the researcher is able to view details of a single condition-element on demand by using semantic zooming which only shows the needed details for the current zoom level. The challenge here was to provide a concept and interaction technique which can also be used by non-computer experts from different backgrounds like psychology, sociology and sports.

Inhaltsverzeichnis

Kurzfassung **2**

Abstract **4**

Inhaltsverzeichnis **5**

Abbildungsverzeichnis **7**

1 Einleitung und Motivation **8**

2 Anforderungsanalyse **11**

2.1 State of the Art 11

2.2 Requirements 13

2.2.1 Datenerhebung an Events im Kontext des Probanden koppeln 13

2.2.2 Umwelt und Situation des Probanden sind Kontextinformationen 13

2.2.3 Qualität und Tiefe der gesammelten Daten erhöhen 13

2.2.4 Erhöhen der Bindung zwischen Proband und Forscher 14

2.2.5 Verschmelzung der Kategorien ESM- und Tagebuchstudie 14

2.2.6 Erleichterung bei Planung, Durchführung und Analyse einer Studie 14

2.2.7 Grafische Definition von Bedingungen und Events 15

2.2.8 Modularer Systemaufbau für einfache Erweiterbarkeit 15

3 Implementierung des Prototyps **15**

3.1 Architektur 15

3.2 Synchronisation 16

3.3 Bedingungsbasierte Events 17

3.4 Interface 18

3.4.1 Client 18

3.4.2 Server 19

4 Studien **20**

4.1 Studie 1: Behindertenfreundlichkeit der UNI 21

4.2 Studie 2: Wellness im Auto 22

4.3 Interview mit einer Forscherin 23

4.4 Probleme & Erkenntnisse 25

4.4.1 Beschränktes Erstellen und Verwalten von Triggern 25

4.4.2 Schwieriges Erstellen von Fragebögen 25

4.4.3 Fehlerhafte Anzeige von Fragebögen und Aufgaben 26

4.4.4 Instabile Synchronisation nach Geräte-Neustart 26

4.4.5 Hoher Stromverbrauch 26

4.4.6 Gesammelte Erkenntnisse 27

5 Konzeption eines modularen Trigger-Frameworks **28**

5.1 Motivation & Anforderungen 28

5.1.1 Höchste Modularität auf allen Ebenen des Frameworks 29
5.1.2 Klar definierte Schnittstellen an den Modulübergängen 29
5.1.3 Erhöhte Zuverlässigkeit des gesamten Systems 29
5.1.4 Erhöhte Flexibilität bei der Gestaltung von Studien....30
5.2 Architektur....30
5.2.1 Sensor 33
5.2.2 Condition 35
5.2.3 Condition-Implementation....35
5.2.4 Condition-Bundle 36
5.2.5 Action 38
5.2.6 Action-Bundle....38
5.2.7 Connector-Element....40
5.3 Verarbeitung 41
5.3.1 Konfiguration und Datenhaltung auf dem Server....41
5.3.2 Synchronisation zwischen Client und Server 41
5.3.3 regelmäßige Prüfung aller Bedingungen auf dem Client....42
5.3.4 Verarbeitung von Statusänderungen einer Bedingung 42
5.3.5 Aktivieren und deaktivieren von Actions 43
5.4 Beispielhafte Implementierung: GPS 43
5.4.1 GPSSensor....44
5.4.2 GPSPosition....45
5.4.3 GPSCondition....45
5.4.4 GPSConditionStandardImplementation....46
5.4.5 QuestionnaireStandardAction....46
6 Konzeption eines Interfaces zur Trigger-Definition....48
6.1 Anforderungen....48
6.2 Visualisierungen 49
6.2.1 Pipe & Filter Prinzip....49
6.2.2 Semantischer Zoom 51
6.2.3 Vorteile und Synergien....51
6.3 Interfacekonzept 53
6.3.1 GUI....54
6.3.2 Interaktion....57
7 Ausblick 66
7.1 Analyse der Daten....66
7.2 Funktionsumfang 67
7.3 Verfügbarkeit....67
8 Zusammenfassung und Fazit 68
Literaturverzeichnis....72

Abbildungsverzeichnis

Abbildung 1 - Client/Server Architektur des PocketBee Systems 16
Abbildung 2 - Ping-Request mit Server-Antwort 200OK 17
Abbildung 3 - ERM des Triggerprinzips 18
Abbildung 4 - neuer Tagebucheintrag 19
Abbildung 5 - Widget auf Homescreen 19
Abbildung 6 - Element eines Fragebogens 19
Abbildung 7 - Interface des ControlCenters 20
Abbildung 8 - Der logische Aufbau einer Condition-Chain 30
Abbildung 9 - Klassendiagramm der Trigger-Architektur 32
Abbildung 10 - Beispiel einer Condition-Chain 33
Abbildung 11 - Die abstrakte Klasse ASensor<T> 34
Abbildung 12 - Schematische Struktur eines Condition-Bundles 36
Abbildung 13 – Die Klassen ConditionBundle und ConditionImplementation 36
Abbildung 14 - Schematische Struktur eines Action-Bundles 39
Abbildung 15 - visuelle Darstellung einer Condition-Chain 42
Abbildung 16 - Klassendiagramm der Beispielimplementierung GPS 44
Abbildung 17 – Eine visuelle Programmiersprache als Basis von LABView [22] 50
Abbildung 18 - Illustration des semantischen Zooms in drei Schritten 51
Abbildung 19 - Pipe/Filter Interface von Squidy 52
Abbildung 20 - Eine Condition-Chain mit Bedingung, Connector und Aktion 53
Abbildung 21 - GUI mit Header (1), Canvas (2) und Toolbar (3) 54
Abbildung 22 - Detailansicht einer GPS-Bedingung 56
Abbildung 23 - Verzweigung einer GPS-Bedingung 57
Abbildung 24 - Verkettung von Objekten mit Connector-Element und Drop-Targets 59
Abbildung 25 - Verzweigungen einer Bedingungskette 60
Abbildung 26 - Zeit-Bedingungsknoten a) mit Detailansicht b) 61
Abbildung 27 - Fragebogen-Action als Knoten a) im Detail b) mit Probandenauswahl c) 62
Abbildung 28 - Spezifikation eines Fragebogens mit Gabelfrage 62
Abbildung 29 - Drop-Targets in einer Condition-Chain 64
Abbildung 30 - Drop-Targets in einer Condition-Chain mit Snap-to-Grid Effekt 64
Abbildung 31 - farbliche Hinweise auf Fehler bei der Validierung 65
Abbildung 32 - Schematische Struktur des Trigger-Frameworks 70
Abbildung 33 - grafisches Interface zur Trigger-Erstellung 71

1 Einleitung und Motivation

Durch die weite Verbreitung und den allgegenwärtigen Gebrauch von Smartphones, mobilen Computern und digitalen Informationen in unserem täglichen Leben findet auch die Nutzung dieser modernen Technologien immer mehr im mobilen Kontext unserer Umgebung statt [28]. Dementsprechend sind auch die Forscher dieser Bereiche gezwungen sich aus ihrem Labor hinaus zu wagen, und sich zusammen mit den Probanden ins tägliche digitale Leben zu mischen. Denn nur durch die direkte Observation in Feldstudien lassen sich realistische Daten über die tatsächliche Nutzung moderner Technologieprodukte erheben. Aus diesem Grund wird die Durchführung von Feldstudien „in the wild“ nicht nur für die Mensch-Computer Interaktion immer wichtiger, sondern auch Psychologen und Soziologen setzen sich mit den sozialen Verhaltensweisen und der Technologienutzung einer neuen vernetzten „always-on“ Generation auseinander [16].

Die dadurch geforderte allgegenwärtige Präsenz eines Forschers kann natürlich in den meisten Forschungsgebieten nicht gewährleistet werden. Aus diesem Grund bietet es sich an, ebendiese mobile Technologie zu verwenden, um trotzdem direkt am Leben des Probanden teilnehmen zu können – ohne jedoch physikalisch präsent sein zu müssen. Moderne Mobiltelefone, so genannte Smartphones, bieten nicht nur ein breites Spektrum an Sensoren und Kommunikationsmöglichkeiten, sondern stellen gleichzeitig eine leistungsfähige Basis zur Verfügung, die durch entsprechende Soft- und Hardware erweitert und so zu Studienzwecken nutzbar gemacht werden kann. Bereits seit mehreren Jahren entwickeln Forscher spezielle Softwareprogramme die zur Durchführung von Feldstudien auf PDAs, Handhelds und Smartphones installiert werden können [26]. Jedes dieser Werkzeuge spezialisiert sich meist auf eine der drei folgenden Methoden zur Datenerhebung im Feld. Die experience sampling method (ESM) nach Hektner [14], die Tagebuchmethode nach Bolger [2] und das Sensor- bzw. Kontext-Logging wie es von Hofte praktiziert wird [16].

Die Methoden lassen sich außerdem durch die Kategorien condition-based design und human-recognition-based design klassifizieren. Während beim condition-based design verschiedene äußere Bedingungen (z.B. Sensoren wie Zeit oder GPS) als Auslöser für Reaktionen dienen, basiert das human-recognition-based design auf dem Menschen als auslösenden Faktor. Der Proband selbst entscheidet, ob die jeweilige Situation dokumentationswürdig ist oder nicht. Parallel zu diesen Bezeichnungen können Reaktionen des Systems ohne Nutzereinwirkung als „system-action“ bezeichnet werden, während alle Aktionen die durch einen Nutzer ausgelöst werden müssen als „user-actions“ bezeichnet werden.

Die ESM (experience sampling method) wurde von Hektner et. al. vorgestellt und dient zur Sammlung von Daten über den Inhalt und den Kontext des täglichen Lebens einzel-

ner Personen. Der Vorteil dieser Methode ist dabei, dass sie das tägliche Leben so aufzeichnet wie es die betroffene Person selbst von Situation zu Situation erlebt [14]. Diese einzelnen Erlebnisse werden meist dadurch erhoben, dass ein Proband über den Tag verteilt eine oder mehrere Fragen gestellt bekommt die seine aktuellen Erfahrungen wiederspiegeln können. Beispielsweise wird ein Proband in zufälligen Abständen (sampling intervall) immer wieder mit derselben Frage konfrontiert (z.B. seine aktuelle Gefühlslage auf einer Ratingskala zwischen 1 und 5). Die Aggregation all dieser Datenpunkte ergibt schlussendlich den Gefühlsverlauf dieses Probanden über einen oder mehrere Tage. Eine Variation der Methode nimmt statt zufällig verteilten Zeitpunkten vorbestimmte Ereignisse, z.B. das tägliche Abendessen, als Auslöser für eine neue Datenerhebung.

Wie der Name schon verrät, basiert die Tagebuchmethode darauf, dass ein Proband über einen längeren Zeitraum ein Tagebuch mit sich führt und dieses entsprechend seiner Forschungsaufgabe mit Einträgen füllt. Während vor einigen Jahren tatsächlich noch papierbasierte Tagebücher an die Probanden verteilt wurden, nutzt man heutzutage meist die digitale Alternative (Digitalkameras oder Diktiergeräte) [2]. Dies hat zur Folge, dass die Probanden durch das anfertigen eines Tagebucheintrags weniger belastet werden und zusätzlich informationsreicheren Inhalt (Videos, Bilder und Sprachaufnahmen) produzieren. Die Tagebuchmethode ermöglicht eine Datenerhebung in realem Kontext und reduziert die Realitätsverzerrung der Information, da die Dokumentation zeitlich näher am tatsächlichen Ereignis stattfindet.

Beim Sensor- oder Kontext-Logging speichert das System zu beliebigen vordefinierten Zeitpunkten im Programmablauf Informationen über Sensorwerte oder den Status bzw. die Umgebung des Systems ab. Diese gespeicherten Informationen können später über eine Logdatei zur Analyse bereitgestellt werden. Durch ein regelmäßiges speichern solcher Informationen kann z.B. das Bewegungsverhalten des Probanden (GPS-Logging), seine Kommunikationsvorlieben (Telefon-Logging) oder seine sozialen Interaktionen (Bluetooth- / Sprach-Logging) analysiert werden. Durch eine breitere Anzahl an Sensoren kann diese Methode viele zusätzliche Informationen über einen Probanden bereitstellen die die Analyse des Studienergebnisses erleichtern [16].

Viele der verfügbaren Werkzeuge spezialisieren sich auf eine oder maximal zwei der oben genannten Methoden. Es bietet sich jedoch für die meisten Studienziele an, mehrere Methoden zu kombinieren um ein breiteres Spektrum an Informationen zu erhalten. Beispielsweise könnte ein Proband regelmäßig zu seiner aktuellen Gefühlslage bzw. zu seinem Wohlbefinden befragt werden während er seinem täglichen Leben nachgeht. Während dieser Zeit hat er die Möglichkeit zu beliebigen Situationen manuelle Einträge zu erstellen und so einen Umstand zu dokumentieren, den er persönlich als relevant empfindet. Zur zusätzlichen Anreicherung der Daten könnte das System laufend Kontextinformationen über die Position, den sozialen Kontext, das Kommunikationsverhal-

ten und die körperliche Verfassung des Probanden sammeln. In diesem Szenario wird klar, dass es hilfreich sein kann verschiedene Erhebungsmethoden zu kombinieren um ein besseres Gesamtbild des Probanden und seiner aktuellen Situation zu erhalten. Während es schlussendlich vom jeweiligen Studiendesign des Forschers abhängt wie die Methoden kombiniert werden, versucht PocketBee für jede dieser drei Methoden die benötigten Funktionen bereitzustellen. So kann der Forscher sein Studiendesign unabhängig von speziellen Methoden entwickeln und hat dadurch größere Freiräume in der Studiengestaltung.

Das Ziel des PocketBee Systems ist es, die freie Wahl der Erhebungsmethoden zuzulassen und so Studien aus Kombinationen von ESM, Tagebuch- und Loggingmethode einfach planen, durchführen und analysieren zu können. In der vorliegenden Masterthesis wird zum Einen die Frage behandelt wie das zugrundeliegende Trigger-Framework umgesetzt werden muss um jede der genannten Methoden unterstützen zu können und zum Anderen wird ein Konzept vorgeschlagen, wie eine grafische Benutzerschnittstelle aussehen kann die auf diesem Trigger-Framework aufbaut und vom Forscher einfach zu bedienen ist. Die Herausforderung besteht darin, im Voraus jede denkbare Studiensituation im System abbilden zu können und später während des Betriebs genau auf diese Situationen reagieren zu können. Beispielsweise ist es denkbar, dass der Proband genau dann nach seiner Gefühlslage befragt werden soll wenn er zu einer bestimmten Uhrzeit alleine ist, sich an einem vordefinierten Ort befindet und seine Herzfrequenz über 120 steigt. Dieses Szenario enthält vier Bedingungen (Zeit, Ort, sozialer Kontext und Herzfrequenz) die geprüft werden müssen bevor das System die entsprechende Aktion (Fragebogen) präsentiert. Natürlich sind beliebige Erweiterungen und Verschachtelungen zu diesem recht linearen Szenario denkbar, die die Komplexität der Konfiguration erhöhen. Damit der Forscher auch bei einem komplexeren Szenario den Überblick über Bedingungen und Aktionen behält ist es wichtig, dass auch die grafische Benutzerschnittstelle diese Aufgabe unterstützt.

In Kapitel 0 werden zunächst die Anforderungen an das PocketBee System anhand einer State-of-the-Art Analyse sowie verschiedenen wissenschaftlichen Quellen hergeleitet. Nachfolgend wird in Kapitel 3 kurz auf den bereits implementierten und in [25] dokumentierten Prototypen eingegangen. Kapitel 4 beschreibt die beiden durchgeführten Studien und dokumentiert anschließend die dadurch erkannten technischen Probleme des Systems. Die gewonnenen Erkenntnisse führen schließlich zur Konzeption eines modulareren Triggerkonzepts welches in Kapitel 5 detailliert dargestellt wird. Neben der Architekturbeschreibung finden sich dort auch beispielhafte Implementierungen und die Dokumentation einzelner Elemente des neuen Frameworks. In Kapitel 6 wird schließlich der Anforderung an ein visuelles Interface Rechnung getragen, welches die betroffenen Forscher einfacher erlernen und bedienen können als die bisherige Implementierung der ersten Version. Das dort vorgestellte Konzept ist mit einer detaillierten

Beschreibung der Interfaceelemente, verschiedenen Interaktionstechniken sowie einem beispielhaft implementierten Prototyp bestens für die Verwendung im PocketBee System geeignet. Die Kapitel 7 und 8 schließen mit möglichen Anknüpfungspunkten und einer Zusammenfassung dieser Arbeit ab.

2 Anforderungsanalyse

Die Anforderungen an das PocketBee System teilen sich in mehrere Rubriken auf. Es existieren wissenschaftliche Anforderungen aus der Methodensicht, technische Anforderungen aus Entwicklungssicht sowie praxisnahe Anforderungen durch die Erfahrung unseres Kooperationspartners. Es wurde eine State-of-the-Art Analyse auf der Basis bereits existierenden ESM- und Tagebuchwerkzeugen durchgeführt die viele wissenschaftlichen Erkenntnisse und Anforderungen zu Tage förderte. Diese State-of-the-Art Analyse bestätigte viele der bereits von Khan et. al. geforderten Eigenschaften für zukünftige ESM-Werkzeuge [19].

2.1 State of the Art

Im Seminar zu dieser Masterthesis „State of the Art – mobile Werkzeuge für die ESM- und Tagebuchmethode“ [26] wurden bereits die aktuell existierenden Werkzeuge für ESM- und Tagebuchstudien analysiert und entsprechend ihrer Eigenschaften verglichen. Dieses Kapitel beschränkt sich daher auf die Resultate dieser Seminararbeit sowie die sich daraus ergebenden Anforderungen.

Es stellte sich die Frage ob eine Vereinigung von mehreren Methoden (ESM, Tagebuch und Logging) in einem Werkzeug technisch machbar und methodisch sinnvoll wäre. Außerdem wurde analysiert welche Werkzeuge auf diesem Gebiet bereits existieren, wie diese Umgesetzt wurden und welche Erkenntnisse sich daraus ergaben.

Es zeigte sich, dass ein methodenübergreifendes Werkzeug nicht nur der nächste logische Schritt in der Entwicklung von ESM- und Tagebuchwerkzeugen ist, sondern dass bereits einige Forschungsgruppen in diese Richtung vorgestoßen sind. Einige der analysierten Werkzeuge haben den Inhalt der vorliegenden Arbeit besonders geprägt. Während das Tool EmotionSense [23] ESM mit Logging kombiniert, fokussiert sich Momento [4] auf die Vereinigung der ESM mit der Tagebuchmethode. Die beiden Werkzeuge MyExperience [8] und XENSOR [16] bieten bereits eine breitere Palette an Funktionalität und decken sowohl die ESM-, die Logging- als auch die Tagebuchmethode ab.

Alle Werkzeuge wurden anhand der folgenden Eigenschaften analysiert und bewertet:

- Erhebungsmethode und Erhebungsdaten
- Ausmaß der Konfigurierbarkeit

- Erweiterbarkeit durch den Forscher
- Öffentliche Verfügbarkeit
- Bisherige Anwendung in Studien
- Synchronisationsmethode
- Möglichkeit zur Studienüberwachung
- Möglichkeit zur Datenanalyse

Die *Erhebungsmethode*, die *Erhebungsdaten* sowie das *Ausmaß der Konfigurierbarkeit* haben Einfluss auf die Studien die mit diesem Werkzeug durchgeführt werden können. Studiendesigns die nicht vom Grundsystem abgedeckt werden können verlangen nach einer einfachen *Erweiterbarkeit durch den Forscher*. So kann sichergestellt werden dass möglichst viele verschiedene Studien mit dem System durchgeführt werden können. Eine *öffentliche Verfügbarkeit*, z.B. unter OpenSource Lizenzen, bietet auch anderen Forschern die Möglichkeit von dem implementierten System zu profitieren. In der bisherigen *Anwendung in Studien* zeigt sich wie praxistauglich und komfortabel das System tatsächlich ist. Außerdem wird deutlich welche Vielfalt an Studiendesigns mit diesem Werkzeug abgedeckt werden kann. Die *Synchronisationsmethode* gibt häufig Aufschluss über *Möglichkeiten zur Studienüberwachung* vom Forscher, da generierte Daten idealerweise in „Echtzeit“ auf den Server des Forschers übertragen werden. Sobald die Daten verfügbar sind ist es die Aufgabe des Forschers diese zu analysieren und die Ergebnisse seiner Forschungsfrage sichtbar zu machen. Viele der Werkzeuge verfügen jedoch über keine bzw. nur minimale *Möglichkeiten zur Datenanalyse*.

In der State of the Art Analyse wurde klar, dass die meisten der Werkzeuge auf spezielle Anwendungsfälle hin entwickelt und eingesetzt wurden. Daraus ergibt sich eine starke Fragmentierung, und für Forscher in diesem Bereich die Qual der Wahl welches Werkzeug für die eigene Studie am besten geeignet ist. Als eines der größten Probleme lässt sich hierbei die Konfiguration für die eigenen Bedürfnisse identifizieren. Die Werkzeuge können teilweise nur durch eine Programmier- oder Scriptsprache an die geplante Studie angepasst werden. Zudem existieren kaum mächtige Verwaltungswerkzeuge die den gesamten Ablauf der Studie unterstützen. Die Forscher haben zwar teilweise die Möglichkeit in Echtzeit auf die Ergebnisse der Probanden zuzugreifen jedoch fehlt überall eine effektive Möglichkeit die Daten und Ergebnisse am Ende der Studie auszuwerten.

Aus den hier erwähnten Problemen und Erkenntnissen ergeben sich Anforderungen an ein zukünftiges Werkzeug zur Kombination von ESM und Tagebuchstudien. Diese Anforderungen werden im folgenden Kapitel genauer beschrieben.

2.2 Requirements

Nachfolgend werden die wichtigsten erhobenen Anforderungen kurz beschrieben und begründet. Teilweise ergeben sich dadurch auch konkrete Anforderungen an Systemfeatures. Während in Kapitel 3 illustriert wird welche der Anforderungen bereits im Prototyp realisiert werden konnten, beschreibt das Kapitel 5 eine Möglichkeit zur Umsetzung der weiteren Anforderungen.

2.2.1 Datenerhebung an Events im Kontext des Probanden koppeln

Um die Basisfunktionalität eines ESM-Werkzeugs zu erfüllen, sollte das System die Möglichkeit bieten zeitnah zu vorbestimmten Ereignissen interessante Daten vom Probanden zu erheben. (z.B. Das Abfragen der Gefühlslage nach einem Telefonat oder zu bestimmten Tageszeiten)

➔ Der Forscher muss am System sowohl die zugrundeliegenden Bedingungen als auch die resultierenden Events definieren können um eine ESM-Studie durchführen zu können. Diese Funktionalität sollte nicht nur bisherige Studiendesigns abdecken sondern auch für die Zukunft erweiterbar sein.

2.2.2 Umwelt und Situation des Probanden sind Kontextinformationen

Oft bergen die aktuelle Situation sowie die direkte Umwelt eines Probanden interessante Informationen über die Auslöser seiner Aktionen und Reaktionen. Daher sollte das System die Möglichkeit bieten, diese Kontextinformationen ebenfalls zu erfassen.

➔ Während der Proband einen Großteil der Informationen durch manuelle Aktionen erheben muss (z.B. Fragebogen, Tagebucheintrag, Foto,…) können die Kontextinformationen sehr oft über automatische Mechanismen (z.B. Logging) erhoben werden. Das System sollte die automatische Erhebung von Kontextinformationen während einer Studie unterstützen und diese mit ins Ergebnis einfließen lassen.

2.2.3 Qualität und Tiefe der gesammelten Daten erhöhen

Das System sollte sowohl manuelle als auch automatische Datenerfassung unterstützen um qualitative und quantitative Daten sammeln zu können. Um in jeder Situation die passenden Daten zu sammeln müssen sowohl der Forscher als auch der Proband Einfluss auf die erhobenen Datentypen haben.

➔ Je nach gegebenem Studiendesign muss entweder der Proband ein der Situation angepasstes Datenformat wählen können oder der Forscher die Möglichkeit haben die verfügbaren Datenformate des Probanden einzuschränken. Die verfügba-

ren Datenformate sollten sehr breit gestreut sein (z.B. Text, Video, Foto, Audio sowie sonstige Sensordaten).

2.2.4 Erhöhen der Bindung zwischen Proband und Forscher

Eine enge Zusammenarbeit und persönliche Bindung zwischen dem Forscher und dem Probanden könnte die Qualität der Ergebnisse erhöhen. Daher sollte das System sowohl direkte Kommunikation als auch einen Informationsaustausch in Echtzeit unterstützen. Dies ermöglicht dem Forscher zusätzlich eine angemessene Vorbereitung auf persönliche Interviews mit dem Probanden.

➔ Das System muss dem Forscher sowohl eine Möglichkeit bieten direkt mit dem Probanden zu kommunizieren (z.B. über Text- oder Sprachnachrichten) als auch eine Möglichkeit die generierten Daten des Probanden in einem geeigneten Format anzuzeigen. Das System sollte hierbei auch die Vorbereitung des Forschers auf ein Interview unterstützen.

2.2.5 Verschmelzung der Kategorien ESM- und Tagebuchstudie

Das System soll sowohl für klassische ESM-Studien als auch für Tagebuchstudien eingesetzt werden können. Da das Design dieser Studien oft nicht eindeutig ist, sollte das System auch eine Kombination beider Techniken ermöglichen.

➔ Im System müssen alle Designelemente von ESM- und Tagebuchstudien modular und parallel aktiviert werden können (zeitbasiertes Sampling und manuelle Einträge). Durch die zugrundeliegende Systemlogik müssen alle kombinatorischen Varianten realisierbar sein. Nicht existierende Funktionalitäten sollten vom Forscher mit minimalem Entwicklungsaufwand erstellt werden können.

2.2.6 Erleichterung bei Planung, Durchführung und Analyse einer Studie

Die Belastung des Forschers im Vorfeld und Verlauf einer Studie sollte reduziert werden. So kann sich ein Forscher komplett auf die wissenschaftlichen Fragestellungen und Analysen fokussieren, statt sich mit technischen und methodischen Problemen zu beschäftigen.

➔ Das System sollte für alle Phasen einer Studie die passende Unterstützung anbieten. Die Planung sollte durch bereits bewährte Studiendesigns (z.B. Trigger-Templates) beschleunigt werden. Das System sollte den Forscher während der Durchführung der Studie entlasten und möglichst viele Aufgaben (z.B. Überwachung und Alarme) automatisieren. Außerdem sollten die Ergebnisse bereits während der Studie entsprechend ihres Datentyps aufbereitet und in Relation gesetzt werden (z.B. Transkription von Sprachaufnahmen und Videos).

2.2.7 Grafische Definition von Bedingungen und Events

Die Verschachtelung von Sensoren, Bedingungen und auszulösenden Events kann in verschiedenen Studien sehr komplex werden. Forscher die nicht technikaffin sind haben häufig Probleme mit Werkzeugen die eine Scriptsprache für die Definition verwenden. Daher sollte das System eine leicht erlernbare Schnittstelle zur Verfügung stellen mit welcher ein Forscher alle nötigen Konfigurationseinstellungen vornehmen kann.

➔ Das System sollte sich komplett über eine grafische Benutzerschnittstelle bedienen lassen. Diese Schnittstelle sollte einfach zu bedienen und von einem Forscher innerhalb weniger Stunden erlernbar sein.

2.2.8 Modularer Systemaufbau für einfache Erweiterbarkeit

Da im Voraus nicht alle möglichen Anwendungsfälle von ESM-, Tagebuch und Loggingstudien bedacht werden können sollte das System so modular wie möglich aufgebaut werden.

➔ Durch eine modulare Architektur und klar definierte Schnittstellen wird externen Forschern aus anderen Interessensgebieten ermöglicht das System entsprechend ihren Bedürfnissen zu erweitern.

3 Implementierung des Prototyps

Im Vorlauf dieser Thesis wurde bereits ein Prototyp des PocketBee Systems entsprechend einigen Anforderungen aus Kapitel 2.2 implementiert und durch einen technischen Projektbericht dokumentiert [25]. Mit diesem Prototyp konnte sowohl das Prinzip des multimodalen Tagebuchs positiv evaluiert als auch einige Tagebuchstudien durchgeführt werden (siehe Kapitel 4). Diese erste Version des PocketBee Systems dient als Grundlage für weitere Entwicklungen und wird daher im folgenden Kapitel kurz vorgestellt.

3.1 Architektur

Wie in Abbildung 1 zu sehen ist, setzt sich das PocketBee System aus zwei Teilen zusammen. Die eine Seite besteht aus der Android-basierten [11] Client-Anwendung, die Serverseite aus der Synchronisationsschnittstelle sowie einem webbasiertem Control-Center für den Forscher.

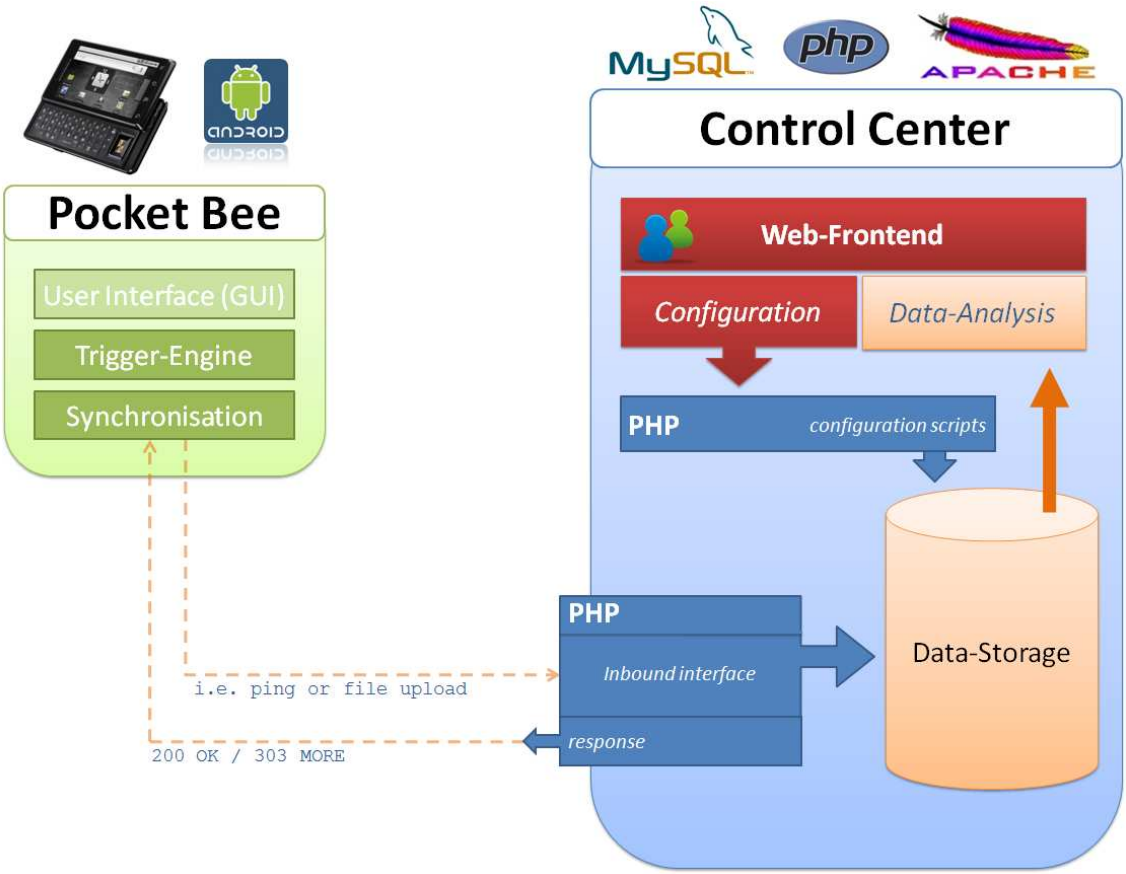

Abbildung 1 - Client/Server Architektur des PocketBee Systems

Auf der Client-Anwendung kann ein Proband Tagebucheinträge verfassen (Text, Foto, Video und Sprache), Fragebogen ausfüllen und seine Aufgaben einsehen. Das Control-Center bietet dem Forscher eine webbasierte Oberfläche um seine Studien zu planen, durchzuführen und auszuwerten. Alle angemeldeten Clients kommunizieren regelmäßig mit dem Server um über jegliche Veränderungen informiert zu sein.

3.2 Synchronisation

Damit ein Client immer die neuesten Fragebögen und Aufgaben zur Verfügung hat versucht dieser regelmäßig den Server zu „pingen“. Ein solcher „ping“ wird alle 30 Minuten vom Client initiiert und überträgt den Zeitstempel der letzten erfolgreichen Aktualisierung. Empfängt der Server einen solchen Ping wird der Client zunächst über einen Benutzernamen und Identifikator authentifiziert. Nun prüft der Server ob Datensätze vorliegen die aktueller sind als der vom Client übertragene Zeitstempel. Ist dies der Fall erhält der Client in seiner Antwort („pong“) die Identifikationsnummern der aktualisierten Datensätze. Diese kann er nun über eine ähnliche Schnittstelle abrufen. Sollten keine neuen Datensätze vorliegen antwortet der Server mit der Antwort „200 OK“ und überträgt den aktuellen Zeitstempel für das nächste Ping-Intervall (siehe Abbildung 2).

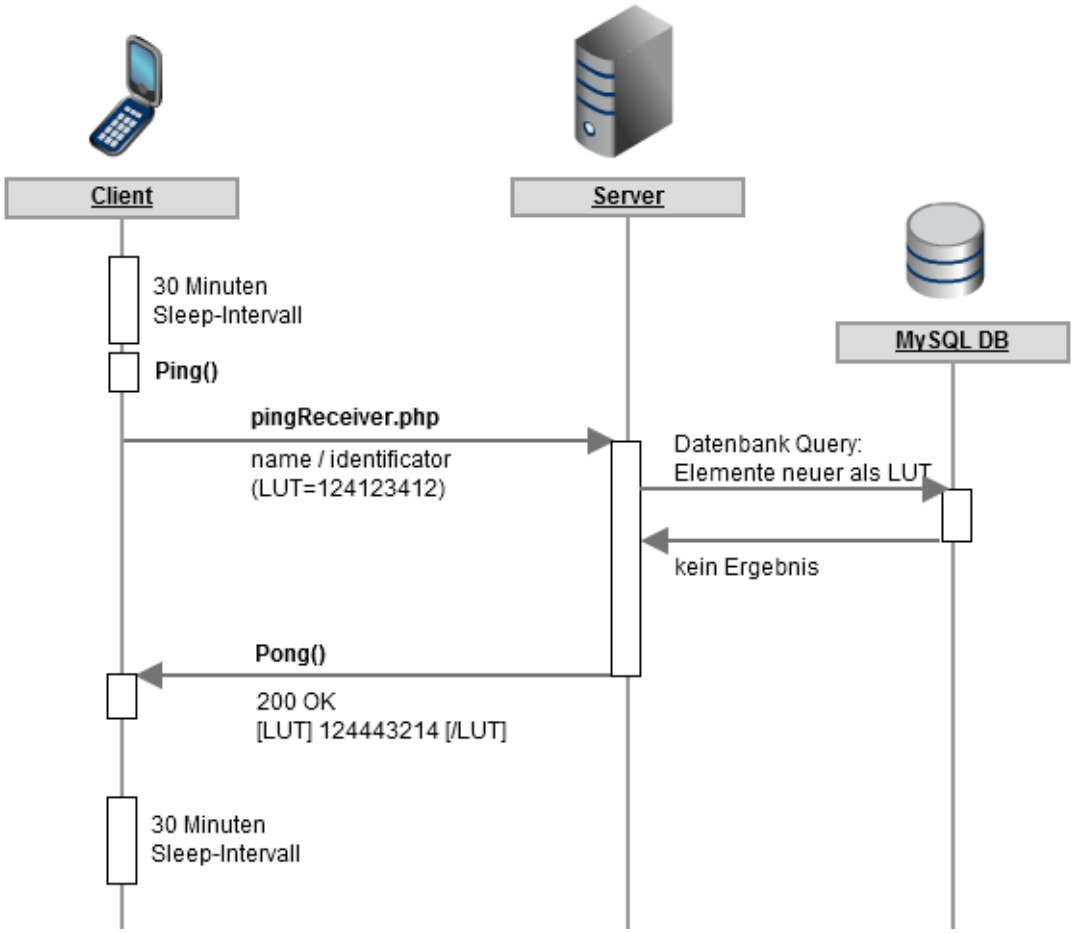

Abbildung 2 - Ping-Request mit Server-Antwort 200OK

Die Datensätze die zur Synchronisation verwendet werden heißen „Trigger“ und repräsentieren als bedingungsbasierte Events im PocketBee System sowohl Aufgaben, Kernfragen als auch Fragebögen.

3.3 Bedingungsbasierte Events

Die bedingungsbasierten Events (Trigger) dienen im System als der universelle Datentyp. Sie können Fragebögen, Aufgaben oder Kernfragen repräsentieren. In Abbildung 3 ist die formale Struktur eines Triggers abgebildet: Bedingungen (conditions) und Ereignisse (events) werden unter einem Element, dem Trigger, zusammengefasst. Jeder Trigger kann auf mehreren verschachtelten Bedingungen basieren, während ebenfalls beliebig viele Events ausgelöst werden können sobald diese Bedingungen erfüllt sind. Ereignisse werden nur ausgelöst, wenn alle Bedingungen gleichzeitig erfüllt sind.

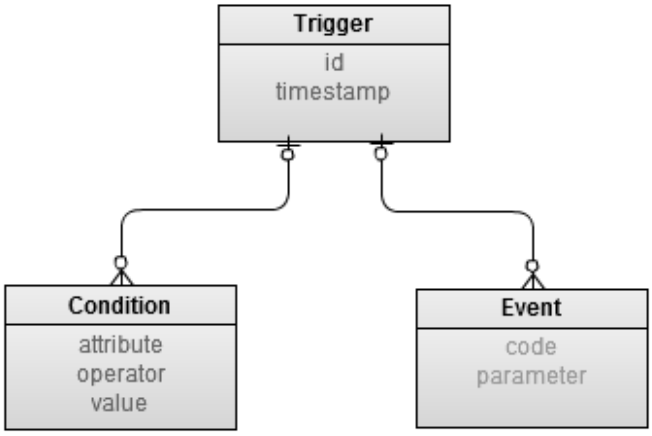

Abbildung 3 - ERM des Triggerprinzips

Ein **Trigger** kann durch seine *id* eindeutig einem einzelnen Probanden zugeordnet werden. So ist es z.B. möglich durch duplizierte Trigger mehrere Probanden mit dem gleichen Fragebogen auszustatten und trotzdem Einzelne davon auszuschließen. Die **Bedingungen** eines Triggers bestimmen wann dieser aktiv wird und wann dessen Events ausgelöst werden. Eine solche Condition kann ein bestimmtes Attribut oder einen Sensor über den Wert *attribute* definieren welcher dann über den *operator* mit dem *value* verglichen wird. Sollte diese Gleichung übereinstimmen ist die gesamte Bedingung erfüllt. Die **Events** eines Triggers definieren die Aktionen welche auf dem Gerät des Probanden ausgeführt werden sobald alle Bedingungen erfüllt sind. Beispielsweise kann die Benachrichtigung für einen Fragebogen oder eine neue Tagebuchaufgabe angezeigt werden. Welcher Typ von Event (Fragebogen oder Aufgabe) auslöst wird, ist im Wert *code* definiert. Weitere Informationen, z.B. die ID des Fragebogens, werden über den Wert *parameter* spezifiziert.

3.4 Interface

Der Fokus der Interfaceentwicklung, deren Resultat hier vorgestellt wird, lag sowohl auf Clientseite als auch auf Serverseite klar auf Einfachheit und Funktionalität.

3.4.1 Client

Die Clientanwendung läuft auf dem Android-Smartphone des Probanden und bietet mehrere Interaktionsmöglichkeiten. Ein Tagebuch-Widget wird direkt auf dem Homescreen des Gerätes aktiviert. Auf diesem dauerhaft sichtbaren Widget kann der Proband auf einen Blick sehen, welche Aufgaben er aktuell hat (Abbildung 5). Vom Widget aus kann der Proband neue Tagebucheinträge anlegen, Fragebögen ausfüllen und Aufgaben starten. Ein Fragebogen führt den Probanden Schritt für Schritt durch alle Fragen und ermöglicht durch Gabelfragen und 8 unterschiedliche Fragetypen eine komplexe Struk-

tur (Abbildung 6). Um einen neuen Tagebucheintrag anzulegen kann der Proband über eine Kernfrage[2] in den Eintragsmodus wechseln. Dort stehen ihm verschiedene Modalitäten zur Verfügung um eine Situation zu dokumentieren (Abbildung 4). Ein Tagebucheintrag kann aus einer beliebigen Kombination an Text, Foto, Video, Sprachaufnahmen und Zeichnungen bestehen. Sobald der Eintrag vom Probanden abgeschlossen wurde werden die Daten im Hintergrund an den Server übermittelt.

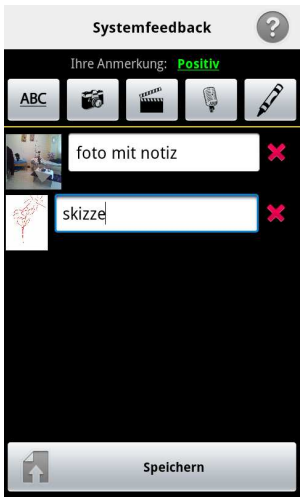

Abbildung 4 - neuer Tagebucheintrag

Abbildung 5 - Widget auf Homescreen

Abbildung 6 - Element eines Fragebogens

3.4.2 Server

Auf dem Server steht dem Forscher eine webbasierte Oberfläche zur Verfügung mit welcher er komplette Studien planen, durchführen und analysieren kann..

Abbildung 7 zeigt die Einzelprojektansicht des ControlCenters. Hier kann der Forscher alle Einstellungen vornehmen die nötig sind um eine Studie durchzuführen. *Bereich (1)* bildet den Header und bietet die Möglichkeit für den Forscher sich abzumelden. Im *Bereich (2)* kann der Titel und die Beschreibung des Projekts geändert werden. Außerdem können alle Ergebnisse und Daten dieses Projekts komprimiert und heruntergeladen werden. Der *Bereich (3)* repräsentiert die Probandenverwaltung. Hier lassen sich neue Probanden anlegen, Alte bearbeiten oder löschen. Außerdem kann der Forscher weitere Details zu einem Probanden einblenden und so einsehen, welche Kernfragen, Aufgaben und Fragebögen diesem Probanden zugeordnet wurden. Über die Funktion „Uploads des Probanden“ lassen sich alle Informationen anzeigen die dieser Proband bisher produziert hat. Dort finden sich alle hochgeladenen Tagebucheinträge, Fotos, Videos und ausgefüllte Fragebögen. Im *Bereich (4)* können Kernfragen und Aufgaben erstellt, bear-

[2] Eine Kernfrage ist eine als mentaler Auslöser formulierte Forschungsfrage. Das Widget bietet alle Kernfragen als einen direkten Einstieg an. So werden Tagebucheinträge direkt einer Kernfrage zugeordnet.

beitet und gelöscht werden. Die Reihenfolge der Kernfragen lässt sich hier verändern und wird entsprechend auf den Geräten der Probanden angezeigt. Die Verwaltung der Fragebögen befindet sich im *Bereich (5).* Für alle Elemente die erstellt, bearbeitet und gelöscht werden können gilt folgendes Prinzip: Ein Element kann einem oder mehreren Probanden zugeordnet werden und muss eine oder mehrere Bedingungen erfüllen um auf dem Clientgerät angezeigt zu werden. Zum Beispiel kann ein Fragebogen nur für Proband 1 und 3 aktiviert werden, wenn die Bedingung 16.02.2011 zwischen 13:00 – 18:00 Uhr erfüllt ist. Neue Elemente werden in regelmäßigen Intervallen vom Client abgerufen (siehe Kapitel 3.2 Synchronisation), und dort aktiv sobald alle Bedingungen erfüllt sind.

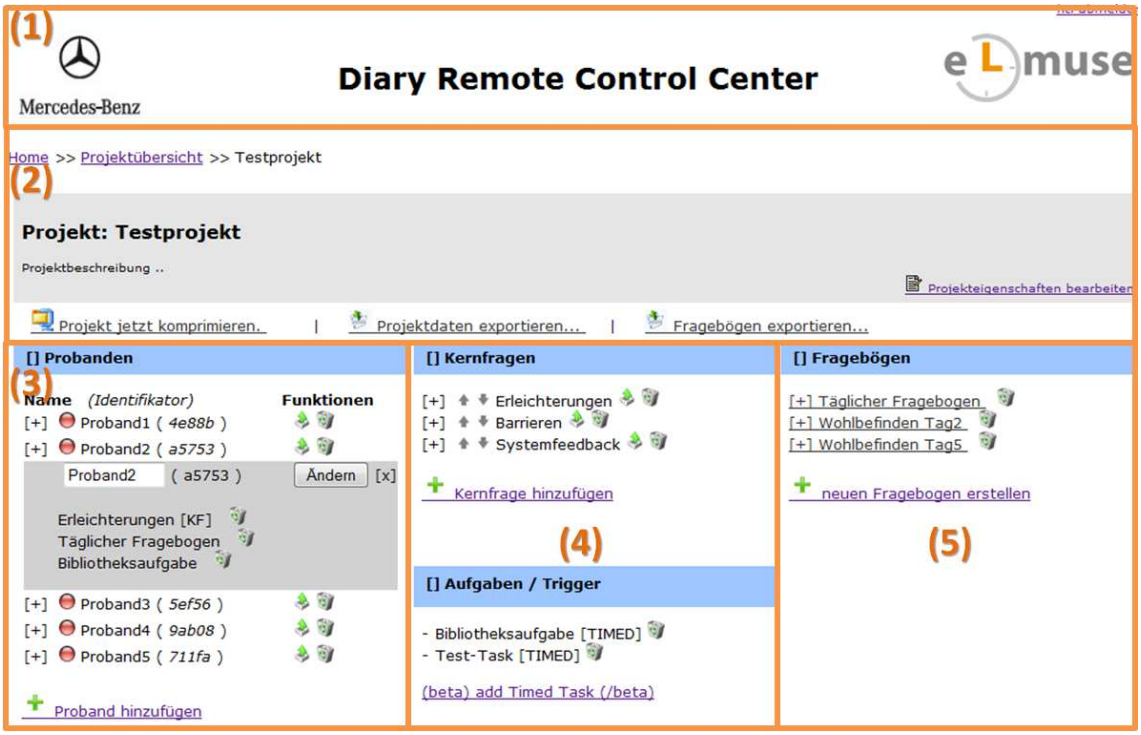

Abbildung 7 - Interface des ControlCenters

4 Studien

Mit dem ersten Prototyp des Systems wurden zwei Feldstudien durchgeführt die das Werkzeug und die Methoden auf die Probe stellten. Während in der ersten Studie 10 Studenten die Gebäude der Universität Konstanz auf Behindertenfreundlichkeit hin dokumentierten, wurde die zweite Studie in Kooperation mit dem Customer Research Center der Daimler AG durchgeführt. Dort wurden 10 Kunden der Daimler AG eine Woche lang zum Thema „Wellness im Auto“ befragt und zur Dokumentation ihrer Eindrücke per Tagebucheintrag angehalten.

In diesem Kapitel werden sowohl die beiden Studien beschrieben, als auch die aufgetretenen Probleme und daraus resultierenden Erkenntnisse aufgezeigt.

4.1 Studie 1: Behindertenfreundlichkeit der UNI

Das Ziel dieser ersten Studie war vor allem die Funktionsfähigkeit des PocketBee Systems sicherzustellen und gleichzeitig das System und den Studienablauf unter realen Bedingungen zu testen. Inhaltlich behandelte die Studie die Behindertenfreundlichkeit der Universität Konstanz und wie diese von nicht behinderten Personen wahrgenommen wird. Über den Verlauf einer Woche wurden 10 Probanden (4 weiblich im Alter zwischen 21 und 30 Jahren) mit einem GalaxyS Smartphone ausgestattet, auf welchem die PocketBee Anwendung installiert war. Die Aufgabe der Teilnehmer war es jegliche Barrieren und Erleichterungen in der Umgebung und den Gebäuden der Universität Konstanz zu dokumentieren (Tagebuchmethode). Zusätzlich wurden zeitlich definierte Fragebögen als auch eine zeitlich spezifizierte Aufgabe eingeblendet (ESM). Während die Fragebögen täglich abfragten wie bewusst die Barrieren und Erleichterungen wahrgenommen wurden, erforderte die Aufgabe eine spezielle Aktion von den Probanden. Ziel war es ein bestimmtes Buch in der Bibliothek zu finden und ein Foto dieses Buchen zu machen. Dadurch sollten alle Probanden denselben Weg durch die Bibliothek gehen um die dokumentierten Barrieren vergleichbar zu machen.

Keiner der Teilnehmer hatte Probleme mit der Nutzung der PocketBee Anwendung und die meisten genossen den Vorteil eine Woche lang ein neues Smartphone testen zu können. Die meistgenutzte Modalität war, wie für die physikalischen Barrieren vorauszusehen, die Kamera. Dies bestätigt die Annahme, dass die gewählte Modalität jeweils vom dokumentierten Gegenstand als auch von der Umgebungssituation abhängt. Auf die Frage welche Modalität nur sehr ungern verwendet wurde antworteten die meisten Teilnehmer mit der Modalität Video sowie der Sprachaufnahme. Nach weiterem Nachhaken stellte sich heraus, dass es den meisten Teilnehmern unangenehm war in der Öffentlichkeit Video- oder Sprachdokumentation ihrer Umgebung vorzunehmen.

Durch die Durchführung dieser Studie konnten sowohl einige technische Probleme des Systems aufgedeckt werden, aber auch die prinzipielle Praxistauglichkeit bewiesen werden. Die gesammelten Barrieren und Erleichterungen wurden an die Universitätsleitung weitergegeben und führen hoffentlich zu Verbesserungen in der Benutzerfreundlichkeit der Universität.

4.2 Studie 2: Wellness im Auto

Während das allgemeine Ziel des „Customer Research Centers“ der Daimler AG die kontinuierliche Verbesserung der Fahrzeuge darstellt, beschäftigte sich die durchgeführte Studie mit einem spezifischen Punkt der Kundenzufriedenheit: dem Wohlbefinden im Fahrzeug. Dieser Forschungsaspekt lässt sich meist nicht vollständig über herkömmliche Methoden wie Usabilitytests, Fokusgruppen oder Kundenbefragungen abdecken. Die komplexe Kombination zwischen Funktionserfüllung, Komfort und Wohlbefinden bei einem Produkt kann von den Probanden nur in ihrer natürlichen Umgebung und in spontanem Kontext abgefragt werden. Daher bietet sich für diese Art des Studiendesigns sowohl die experience sampling method als auch die Tagebuchmethode und damit der Einsatz des PocketBee Systems an.

Die 10 Teilnehmer der Studie waren zwischen 28 und 55 Jahre alt. Die 5 weiblichen und 5 männlichen Teilnehmer waren alle Eigentümer eines Mercedes. Für den Zeitraum einer Woche wurde ihnen PocketBee auf einem Motorola Milestone mit der Android Version 2.0.1 ausgehändigt. Während das hauptsächliche Studienziel war, herauszufinden wann und aus welchem Grund die Probanden während dieser Woche positive bzw. negative Erfahrungen in ihrem Fahrzeug gemacht hatten, wurden sie zusätzlich nach Situationen gefragt in welchen sie sich generell wohlfühlten. Diese allgemeinen Informationen zum Wohlfühlen sollten Inspiration und Anregungen für die Fahrzeugdesigner liefern um das Produkterlebnis noch verbessern zu können.

Beim Studiendesign wurde eine Kombination aus „human-recognition based design“ (manuelle Datensammlung mit Tagebuchtechnik) und „condition based design“ (Datensammlung nach Bedingungen mit experience sampling method) verwendet. Während die Probanden die Aufgabe bekamen alle Situationen innerhalb und außerhalb des Fahrzeugs zu dokumentieren in denen sie sich wohlfühlten (Tagebuchmethode), wurden außerdem Fragebögen eingeblendet in denen der Ablauf des Tages bzw. allgemeine Fragen zum Wohlbefinden gestellt wurden. Diese Fragebögen wurden täglich zwischen 19:00 und 24:00 Uhr eingeblendet (experience sampling method).

Nach Ablauf der Woche wurden die Probanden in einem Abschlussinterview über ihre Erfahrungen und Dokumentationen befragt. Mit Hilfe des ControlCenters des PocketBee Systems konnten die Forscher einzelne Einträge mit dem Probanden besprechen und gegebenenfalls kommentieren, so ließen sich die interessanten Erkenntnisse noch weiter klären. Das Feedback der Probanden gegenüber der PocketBee Anwendung war durchweg positiv. Als hervorstechende Punkte wurden die einfache Bedienung sowie die vielen Modalitäten und deren Kombination zur Dokumentation genannt. Während die meisten Teilnehmer die Texteingabe zur Dokumentation favorisierten folgten dicht darauf Fotos und Sprachaufnahmen. Von der Möglichkeit zur Kombination mehrerer Modalitäten wurde vor allem bei Fotos und Text oft Gebrauch gemacht. Acht der zehn

Teilnehmer wurden meistens durch die visuelle Repräsentation auf dem Gerät auf neue Fragebögen aufmerksam während nur zwei der Teilnehmer durch die Notifikationen (Vibration und Audio) auf neue Aufgaben aufmerksam wurden. Dies zeigt wie wichtig eine gut sichtbare visuelle Repräsentation der Aufgaben auf dem Gerät ist, bzw. dass die Audionotifikationen verbessert werden müssen. Die Aufteilung der Teilnehmer in zwei Gruppen (beschreibende Kernfragen als Titel der Buttons vs. „neuen Eintrag“ Button) führte zu einem unerwarteten Resultat: Beide der Gruppen bevorzugten in der abschließenden Befragung genau die Variante mit der sie selbst konfrontiert wurden. Dies lässt somit keine Schlüsse auf eine „bessere“ Variante zu und muss in Zukunft weiter evaluiert werden. Kritikpunkte der Probanden bezogen sich hauptsächlich auf die Batterielaufzeit des Geräts und den dadurch verbundene tägliche Ladevorgang.

Abschließend lässt sich sagen, dass unser Kooperationspartner durch diese Studie sowohl viele interessante Ideen aber auch Kritik an spezifischen Eigenschaften des Fahrzeugs erhalten hat und dies sehr gut als Ausgangsmaterial für weitere Verbesserungen verwendet werden kann. Während im folgenden Kapitel darauf eingegangen wird wie die beteiligten Forscher das ControlCenter genutzt haben und wie dieses verbessert werden kann, steht die zukünftige Verwendung von PocketBee im Customer Research der Daimler AG bereits fest.

4.3 Interview mit einer Forscherin

Wie bereits beschrieben wurde die zweite Studie „Wohlfühlen im Fahrzeug“ außerhalb der Universität Konstanz gemeinsam mit dem Customer Research Center der Daimler AG durchgeführt. Die dort beteiligten Personen wurden zwar in der Design- und Iterationsphase mit in die Entwicklung des PocketBee-Prototyps miteinbezogen und sind demnach vorbelastet, sie können aber generell als typische Endnutzer des ControlCenter-Systems gesehen werden. Während der Wohlfühlen-Studie war nur eine einzige Forscherin aktiv im ControlCenter tätig. Diese wurde für qualitative Evaluationszwecke zu ihrer Erfahrung mit dem System befragt. Im Folgenden werden die Erkenntnisse aus diesem Interview erläutert.

Die befragte Forscherin ist beim Customer Research Center der Daimler AG (CRC) direkt mit der Planung, Durchführung und Auswertung verschiedener Studien betraut. Zu ihren Tätigkeiten zählen auch die Abstimmung der Untersuchungsziele mit dem Auftraggeber sowie die entsprechende Auswahl an geeigneten Methoden zur Beantwortung dieser Fragestellungen, die Vorbereitung der Untersuchungsmaterealien (Fragebögen, Interviewleitfäden, usw.), die Datenerhebung von Probanden in Interviews, Testlabors oder im Feld und schlussendlich die Aufbereitung, Auswertung und Interpretation der gewonnenen Daten.

Jeder Forscher des CRC ist in mehrere thematisch unterschiedliche Projekte involviert. Bei der befragten Forscherin handelte es sich im Jahr 2010 um fünf unterschiedliche Projekte (davon drei als Projektleiterin). Im Rahmen dieser Projekte wurden insgesamt 11 verschiedene Evaluationen durchgeführt. Dabei wurden verschiedene Methoden eingesetzt, darunter Interviews, Online-Befragungen, Fokusgruppen, Beobachtungsstudien bis hin zu Fahrversuchen mit „Thinking Aloud"-Technik. Zur Vorbereitung dieser Arbeit verwenden die Forscher Softwareprodukte wie Microsoft Excel, PowerPoint und Word [21] als auch EFS von Globalpark zur Durchführung von Online-Fragebögen [10] und SPSS zur Auswertung der Daten [17].

Bei der direkten Befragung zu den Erfahrungen mit dem ControlCenter stellte sich heraus, dass die Projektübersicht ein wichtiges Element des Interfaces ist, da dort alle Informationen auf einen Blick erfasst werden können. Das grafische Design lässt zwar „zu wünschen übrig", jedoch wurde die Umsetzung der Funktionen als effektiv eingeschätzt. Als einen weiteren großen Vorteil wurde der direkte Zugriff auf die generierten Daten der Probanden genannt, dies bietet zusammen mit der Kommentarfunktion für den Forscher eine komfortable Basis für die Vorbereitung und Durchführung der Post-Interviews. Die Erstellung von Fragebögen und deren zeitliche Terminierung wurden als eine „nicht triviale Aufgabe" bezeichnet, da die Fragebogenerstellung durch die Definition als XML sehr fehleranfällig ist und die Auswahl der Zeitbedingung durch viele einzelne Formularobjekte schnell unübersichtlich werden kann. Neben einzelnen Systemausfällen während der Studie (z.B. falsch terminierte Fragebögen) wurde zusätzlich die aufwändige Analyse der generierten Daten als Manko genannt. Das ControlCenter bietet hierbei nicht ausreichend automatisierte Unterstützung und belastet den Forscher so mit der manuellen Auswertung der Fragebögen sowie der manueller Transkription von Sprachaufzeichnungen. Abschließend wurden jedoch folgende Punkte als positive Verbesserung gegenüber der herkömmlichen Methode genannt: Die Geräte können schnell konfiguriert werden, die Vorbereitung einer „Real-Life-Analyse"[3] ist auf der Basis eines Smartphones um einiges komfortabler, es muss kein Callcenter besetzt werden und die Probanden können ihre Dokumentation durch verschiedene Modalitäten genau auf das Thema abstimmen. Als Verbesserungsvorschläge für das ControlCenter wurde eine vereinfachte, visuell unterstützte, Erstellung von Fragebögen sowie eine anschauliche Terminierung von Zeitbedingungen genannt. Außerdem wurden Erleichterungen in der Analysephase wie z.B. ein automatisch erstelltes Transkript einer Sprachaufnahme gefordert.

Dieses Interview hilft zu verstehen, was die tatsächlichen Probleme und Herausforderungen des PocketBee Systems aus der Sicht einer Endnutzerin sind. Nachfolgend werden weitere Herausforderungen beschrieben, die im Laufe des Praxiseinsatzes auftraten.

[3] Bisherige automatisierte Methode des CRC zur Evaluation im Fahrzeug

4.4 Probleme & Erkenntnisse

Durch die beiden Pilotstudien entstanden sowohl viele Erkenntnisse über das System aus technischer Sicht, als auch Erkenntnisse über die Erhebungsmethoden „Tagebuch“ [2] und „ESM“ [14] in einem mobilen Kontext. Im Folgenden Kapitel werden ausschließlich die technischen Herausforderungen und Erkenntnisse erläutert. Die methodischen Erkenntnisse finden sich in der Masterarbeit von Stefan Dierdorf [6].

4.4.1 Beschränktes Erstellen und Verwalten von Triggern

In der ersten Implementierung des Systems beschränkte sich die einzige verfügbare Bedingung auf den „Sensor“ Zeit. Außerdem waren auch die Events auf Fragebögen und einzelne Aufgaben limitiert. So konnten zwar die ersten Studien durchgeführt werden, eine spontane Erweiterung des Studiendesigns wäre aber nicht möglich gewesen. Im Allgemeinen ist eine Erweiterung der Sensoren, z.B. den CAN-Bus eines Fahrzeugs, mit erheblichem Implementierungsaufwand auf beiden Seiten (Client und Server) verbunden. Zusätzlich könnte nicht garantiert werden, ob das System nach weiterem Hinzufügen von Sensoren und Events weiterhin robust arbeiten würde.

Daher ergibt sich die Anforderung an ein noch modulareres und robusteres Trigger-Framework, welches es durch klar definierte Schnittstellen erlaubt, beliebige weitere Sensoren oder Events anzubinden. Außerdem sollte ein Mechanismus existieren, der überprüft ob Definitionen sowohl auf Client als auch auf Serverseite korrekt implementiert wurden.

Ein weiteres Problem in diesem Kontext ist die Benutzerschnittstelle für den Forscher. Aktuell wurden die Konfigurationsmöglichkeiten als vordefinierte Optionen in der GUI des ControlCenters implementiert. Mit Blick auf ein modulares Framework und der Möglichkeit viele neue Sensoren, Bedingungen und Events einzubinden ist es von Nöten, auch dem Forscher ein modulares grafisches Nutzerinterface bereit zu stellen.

4.4.2 Schwieriges Erstellen von Fragebögen

Im aktuellen Prototyp des Systems kann der Forscher einen neuen Fragebogen nur direkt als XML-Code definieren. Für nicht technikaffine Menschen ist dies recht ungewohnt, da sie „Programmcode“ schreiben müssen um einen Fragebogen zu definieren (siehe Kapitel 4.3). Dies führt zwangsläufig zu einer hohen Fehleranzahl beim Erstellen von Fragebögen und steigert die Einstiegshürde für neue Forscher.

Aus dieser Erkenntnis ergibt sich die Anforderung an eine grafische Möglichkeit Fragebögen zu definieren und idealerweise direkt im System testen zu können. Eine solche Lösung sollte intuitiv sein und direkt in das Gesamtsystem integriert werden.

4.4.3 Fehlerhafte Anzeige von Fragebögen und Aufgaben

In den Pilotstudien traten teilweise Probleme auf, die dazu führten, dass Fragebögen oder Aufgaben entweder zur falschen Uhrzeit aktiviert wurden, teilweise doppelt auf dem Gerät erschienen oder gar nicht angezeigt wurden. Diese Probleme lassen sich darauf zurückführen, dass sowohl die Trigger-Interpretation auf Clientseite als auch Serverseite manuell in unterschiedlichen Programmiersprachen implementiert wurde. Dies führt dazu, dass keine automatische Überprüfung der Namensräume oder Variablennamen stattfinden kann. Durch Inkonsistenzen bei der Triggerinterprätation entstanden Situationen die zu fehlerhafter Anzeige von Fragebögen und Aufgaben führten.

Um diese Probleme zu beheben wird vorgeschlagen das bestehende Triggerkonzept durch ein modulareres Trigger-Framework zu ersetzen. Dadurch sollte sichergestellt werden, dass nicht nur die Erweiterung um neue Sensoren und Bedingungen vereinfacht wird, sondern auch die Fehleranfälligkeit durch die klare Struktur und definierte Schnittstellen abnimmt.

4.4.4 Instabile Synchronisation nach Geräte-Neustart

Um dem Forscher einen zeitnahen Zugriff auf den Studienverlauf zu geben ist es nötig, dass die Client-Geräte über jegliche Änderungen der Konfiguration benachrichtigt werden. Dies wurde über einen Mechanismus realisiert, der vom Client aus in regelmäßigen Abständen beim Server nach Neuerungen anfragt. Dieser Synchronisationszyklus wurde bei einem Neustart des Geräts unterbrochen und nicht wieder aufgenommen. Daher konnte es passieren, dass Geräte die aufgrund eines niedrigen Batteriestandes automatisch ausgeschalten wurden nach dem Neustart nicht wieder im System verfügbar waren. Das System ermöglichte zwar weiterhin die manuelle Erstellung von Tagebucheinträgen am Client, jedoch wurde keine der Änderungen des Forschers auf das Client-Gerät übertragen.

Aus diesen Erkenntnissen ergeben sich zwei Anforderungen für zukünftige Versionen von PocketBee. Zum Einen muss das Client-System über eine „wasserdichte“ Self-Recovery Funktion verfügen, um sich nach unvorhergesehenen Ausfällen ohne manuellen Einfluss des Probanden wieder in einen Normalbetrieb begeben zu können. Und zum Anderen sollte die Synchronisation in Zukunft über das ab Android 2.2 angebotene Cloud-to-Device Messaging statt über regelmäßiges Polling abgewickelt werden [12]. Dies ermöglicht einen direkten Zugriff des ControlCenters auf das Gerät.

4.4.5 Hoher Stromverbrauch

Wie bereits im vorigen Kapitel 4.4.4 erläutert, war es in regelmäßigen Abständen nötig eine Verbindung zwischen Client und Server aufzubauen, da die verwendete Version 2.0.1 von Android noch nicht ermöglichte die Geräte direkt vom Server aus zu benach-

richtigen. Daher wurde für den Prototyp ein Update-Mechanismus implementiert der in regelmäßigen Abständen beim Server anfragt ob Änderungen für diesen Client vorliegen (siehe Kapitel 3.2). Dies führt zwangsläufig dazu, dass die Akku-Betriebsdauer dieser Clients zeitlich eingeschränkt ist und die Geräte regelmäßig geladen werden müssen.

Eine Lösung für dieses Problem bietet der von Android ab der Version 2.2 angebotene Cloud-to-Device Service [12]. Durch diesen Mechanismus wäre es möglich Geräte nur zu benachrichtigen wenn auch tatsächlich eine Aktualisierung auf dem Server vorliegt. Daher sollte eine Zukünftige Version von PocketBee diesen Mechanismus zur Synchronisation nutzen.

4.4.6 Gesammelte Erkenntnisse

Nachfolgend werden noch einmal alle Erkenntnisse aus den Pilotstudien aufgeführt und daraus die entsprechende Anforderung definiert. Diese Anforderungen entstanden durch den Praxistest des Systems und haben somit eine sehr hohe Priorität für zukünftige Versionen.

Problem: Es existieren Beschränkungen bei der Erstellung und Verwaltung von Sensoren, Bedingungen und Events sowie eine teilweise fehlerhafte Interpretation dieser Bedingungen und Events auf dem Clientgerät.

Anforderung: Stabiles und gleichzeitig modulares Framework zur Definition von Sensoren, Bedingungen und Events. Das Framework soll sowohl die Verwendung existierender Bedingungen durch den Forscher vereinfachen als auch die Integration von komplett neuen Sensoren und Events leicht ermöglichen.

Problem: Fragebögen müssen vom Forscher direkt als XML-Datei definiert werden. Dies führt zu einer hohen Einstiegshürde und einer hohen Fehleranfälligkeit.

Anforderung: Ein grafisches Benutzerinterface zur Erstellung von Fragebögen.

Problem: Durch den bisherigen Mechanismus zur Synchronisation existieren Probleme mit einem hohen Energieverbrauch und einer mittelmäßigen Zuverlässigkeit.

Anforderung: Verwendung des Cloud-to-Device Messaging zur Synchronisation sowie Implementierung eines Failover-Mechanismus der auch nach unvorhergesehenen Ausfällen für den Normalbetrieb sorgt.

Problem: Durch das bisherige zu komplexe Triggerframework entstanden Fehler in der Interpretation von Triggern.

Anforderung: Ein modulareres Triggerframework welches problemlos Erweiterungen zulässt und durch klar definierte Schnittstellen Komplexität reduziert.

Die hier beschriebenen Probleme, Herausforderungen und Verbesserungsmöglichkeiten führten zum Re-Design des Systems welches der Inhalt der folgenden zwei Kapitel ist. Es wird sowohl ein neues Trigger-Framework als auch eine grafische Benutzerschnittstelle für den Forscher vorgestellt.

5 Konzeption eines modularen Trigger-Frameworks

Wie bereits klar geworden ist, bietet das PocketBee System viel Spielraum für Erweiterungen und Verbesserungen. Im folgenden Kapitel wird ein Ansatz vorgestellt, der das bestehende Trigger-Framework von Grund auf erneuern und verbessern soll. Unter dem Trigger-Framework wird die systeminterne Logik verstanden, die alle Sensoren, Bedingungen und Events vereint und die das Rückgrat des PocketBee Systems darstellt.

5.1 Motivation & Anforderungen

Es stellt sich die Frage warum bei einem laufenden System grundlegende Bestandteile verändert oder gar neu entwickelt werden müssen obwohl die gewünschte Funktionalität bereits gegeben ist. Die Antwort darauf geben sowohl die wissenschaftlichen Anforderungen aus Kapitel 0 als auch die Erkenntnisse der durchgeführten Praxisstudien aus Kapitel 4. Das PocketBee System erfüllt zwar die Basisfunktionalität die es benötigt um normale Tagebuchstudien durchzuführen, jedoch fehlt es an der nötigen Erweiterbarkeit um mit dem System auch Studien aus anderen Fachbereichen oder komplexere Szenarien durchführen zu können [19]. Um tatsächlich ein System zu erhalten welches sowohl die komplexesten Szenarien der electronic sampling method als auch die der Tagebuchmethode beherrscht bedarf es etwas Veränderung.

Die zwei wichtigsten Punkte sind zum Einen die Kombination von Methoden (Tagebuch, ESM und Logging) in einer einzelnen Studie mit einem einzigen Werkzeug (2.2.5) und zum Anderen die Erleichterung der Konfiguration für den Forscher durch eine grafische Oberfläche (2.2.7). Während sich der Inhalt dieses Kapitels ausschließlich um das zugrundeliegende Trigger-Framework dreht und wie dieses eine Methodenverschmelzung realisieren kann, wird später in Kapitel 6 das darauf abgestimmte grafische Interface vorgestellt.

Die größte Herausforderung stellt die Definition der Trigger dar, die aus verketteten Bedingungen und Aktionen bestehen. Mit Hilfe der Trigger sollte es möglich sein jede beliebige Situation während einer Studie abzubilden, und im Voraus die gewünschte Systemreaktion festlegen zu können. Beispielsweise kann es nötig sein, dass der Proband jedes Mal einen Fragebogen zu seiner Gefühlslage präsentiert bekommt, wenn folgende Bedingungen zutreffen: Der Proband ist zu einer bestimmten Uhrzeit an einem

Bestimmten Ort während seine Herzfrequenz über 120 steigt. Zusätzlich soll der Fragebogen davon abhängig sein, ob der Proband aktuell alleine oder in einer Gruppe unterwegs ist. Während dieses Szenario bereits 4 Bedingungen (Zeit, Ort, Herzfrequenz und sozialer Kontext) enthält kann es durchaus möglich sein, dass der Forscher auch an einigen Variationen dieser Situation interessiert ist. Beispielsweise soll die Herzfrequenz des Probanden dauerhaft mit geloggt werden wenn er sich nicht an dem Ort X aufhält. Dies führt zu einer Verschachtelung der Bedingungen und einer zusätzlichen Aktion (Sensor-Logging). Denkt man diese Variationen und mögliche Forschungsfragen weiter kommt man schnell zu dem Schluss, dass die Kombination der Bedingungen und Aktionen keine triviale Aufgabe ist und das System zusätzlich mit weiteren interessanten Sensoren ausgestattet werden sollte. Aus diesem Grund muss das Trigger-Framework nicht nur offen für neue Sensoren und Bedingungen sein, sondern auch die Erstellung und Verarbeitung komplexer Verkettungen unterstützen.

Die Folgenden Anforderungen basieren sowohl auf der wissenschaftlichen Literatur (Kapitel 2.2), der durchgeführten State-of-the-Art Analyse (Kapitel 2.1) als auch auf Erkenntnissen aus den Pilotstudien (Kapitel 4.4).

5.1.1 Höchste Modularität auf allen Ebenen des Frameworks

Das Ziel dieser Anforderung ist es alle Elemente des Systems so modular wie möglich zu gestalten um Erweiterungen (z.B. Sensoren oder Actions) möglichst einfach umsetzen zu können. Die Integration eines neuen Sensors sollte unabhängig von anderen Sensoren und unabhängig vom restlichen System erfolgen können. Außerdem sollten alle bereits existierenden Bedingungen und Actions durch klar definierte Schnittstellen problemlos miteinander kommunizieren können.

5.1.2 Klar definierte Schnittstellen an den Modulübergängen

Wie bereits erwähnt ist das Ziel der Modularisierung nicht nur die einfache Integration neuer Elemente sondern auch die Absicherung der Kommunikation zwischen den Elementen durch klar definierte Schnittstellen. Dies reduziert die Fehleranfälligkeit des gesamten Systems und sorgt so für einen stabilen Betrieb. Einzelne Module können durch die klare Schnittstellendefinition je nach Bedarf ausgetauscht oder verbessert werden.

5.1.3 Erhöhte Zuverlässigkeit des gesamten Systems

Durch die bisherigen Schnittstellen und Verarbeitungsmethoden im Pilotsystem entstanden einige Fehler [25], die in Zukunft vermieden werden sollen. Beispielsweise wurden zeitliche Trigger nicht ausgelöst oder durch falsch interpretierte Schnittstellen

fehlerhaft terminiert. Durch den Fokus auf eine klare Modularisierung soll sichergestellt werden, dass Fehlinterpretationen nicht mehr auftreten.

5.1.4 Erhöhte Flexibilität bei der Gestaltung von Studien

Während das Pilotsystem auf zeitliche Bedingungen, Fragebögen und Kernfragen beschränkt ist soll durch das neue Trigger-Framework die Erweiterung stark vereinfacht werden. Es soll möglich sein weitere Sensoren und Actions problemlos anzubinden als auch bereits bestehende Konfigurationselemente in beliebig komplexen Szenarien zusammenzustellen ohne die Stabilität des Systems zu gefährden.

Diese Erkenntnisse und Anforderungen wurden aus den bisherigen Aktivitäten und Nachforschungen hergeleitet und sind die Grundlage für die vorgestellte Architektur.

5.2 Architektur

Um der Motivation und den Anforderungen aus Kapitel 5.1 gerecht zu werden wird im Folgenden die Architektur des neuen Trigger-Frameworks vorgestellt. Diese Architektur kann aus zwei Blickwinkeln betrachtet werden. Die Eine Seite ist der logische Blickwinkel, der auf die einzelnen Framework-Elemente und ihre Funktionen fokussiert. Die andere Seite ist der Blickwinkel der UML[4], welcher die Softwarearchitektur als Konstrukt von Klassen und Relationen abbildet, die später als logische Elemente zusammenspielen. Diese beiden Blickwinkel spiegeln sich in Abbildung 8 (Struktur und Funktion der Elemente) und Abbildung 9 (Klassenrelationen in UML) wieder.

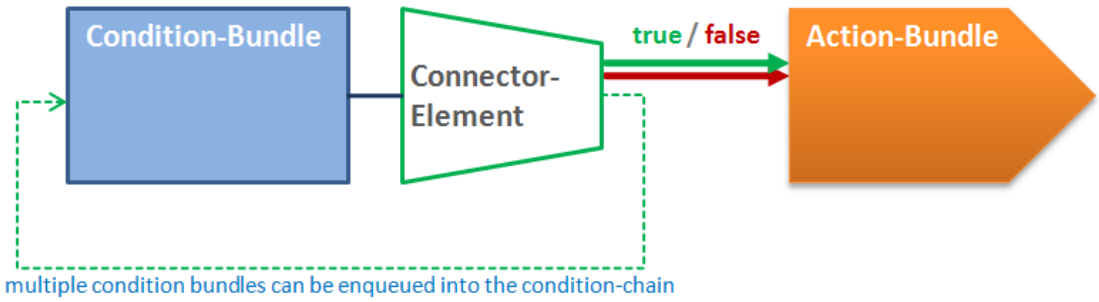

Abbildung 8 - Der logische Aufbau einer Condition-Chain

In Abbildung 8 zeigt sich die logische Struktur der Elemente als verkettete Condition-Chain sowie ihre spätere Funktionen. Eine Condition-Chain bildet die Grundlage für einen Teilaspekt einer Studie der durch Bedingungen und darauffolgende Aktionen abgebildet wird. Sie basiert auf dem Grundprinzip, dass alle enthaltenen Elemente, z.B. Bedingungen als Condition-Bundle und Aktivitäten als Action-Bundle beliebig oft mi-

[4] Unified Markup Language – wird zur Beschreibung von Klassenrelationen und Hierarchien verwendet.

teinander verkettet werden können. Die einzige Bedingung in der Kopplung besteht darin, dass sowohl zwischen zwei Condition-Bundles als auch zwischen einem Condition-Bundle und einem Action-Bundle ein Connector-Element platziert werden muss. Diese Connector-Elemente fungieren ausschließlich als Verbindungs- und Filterelemente zwischen den Bundles. Während ein Condition-Bundle jeweils zur Prüfung einer Bedingung herangezogen wird, dient ein Action-Bundle zum starten einer System- oder User-Action[5]. Für den einfachsten Fall einer Condition-Chain wird angenommen, dass mehrere Bedingungen als Condition-Bundles hintereinander verkettet werden und mit einem Action-Bundle abschließen. Das entsprechende Action-Bundle am Ende der Kette wird nur dann aktiviert, wenn alle vorhergehenden Bedingungen als wahr evaluiert werden bzw. die Connector-Elemente in ihrer Filterfunktion nicht blockieren.

Die Abbildung 9 präsentiert die klassenorientierte Sichtweise als UML Diagramm. Hier zeigen sich nicht nur die Relationen zwischen den einzelnen Klassen sondern auch die technischen Implementierungen die dafür sorgen, dass eine Condition-Chain als schnell zugreifbare Datenstruktur (n-ary Tree [3]) vorgehalten werden kann. Diese Baumstruktur wird im Klassendiagramm durch die Klassen FlowTree und FlowNode repräsentiert. Während FlowTree die Ausgangsbasis mit der Wurzel des Baumens bildet, können beliebig viele FlowNodes hintereinander und nebeneinander als Äste und Blätter angeordnet werden. Jedes dieser FlowNode Elemente besitzt sowohl eine Liste mit weiteren Kind-Elementen als auch ein „Payload-Datenelement“ vom Typ AFlowElement. Somit wird sichergestellt, dass jedes Datenelement im Baum (bzw. in der Condition-Chain) vom abstrakten Typ AFlowElement erbt.

Durch diese Baumstruktur mit der abstrakten Klasse AFlowElement als Datenknoten wird dem System ermöglicht zur Evaluierung einer Condition-Chain schrittweise durch den kompletten Baum zu gehen und einzelne Bedingungen zu prüfen, bzw. Aktionen auszulösen bei denen alle vorangehenden Bedingungen erfüllt sind.

[5] Als User- / System-Action werden Reaktionen auf erfüllte Bedingungen bezeichnet, die entweder die Eingabe eines Nutzers benötigen oder unabhängig vom Nutzer Systeminterne Funktionen ausführen.

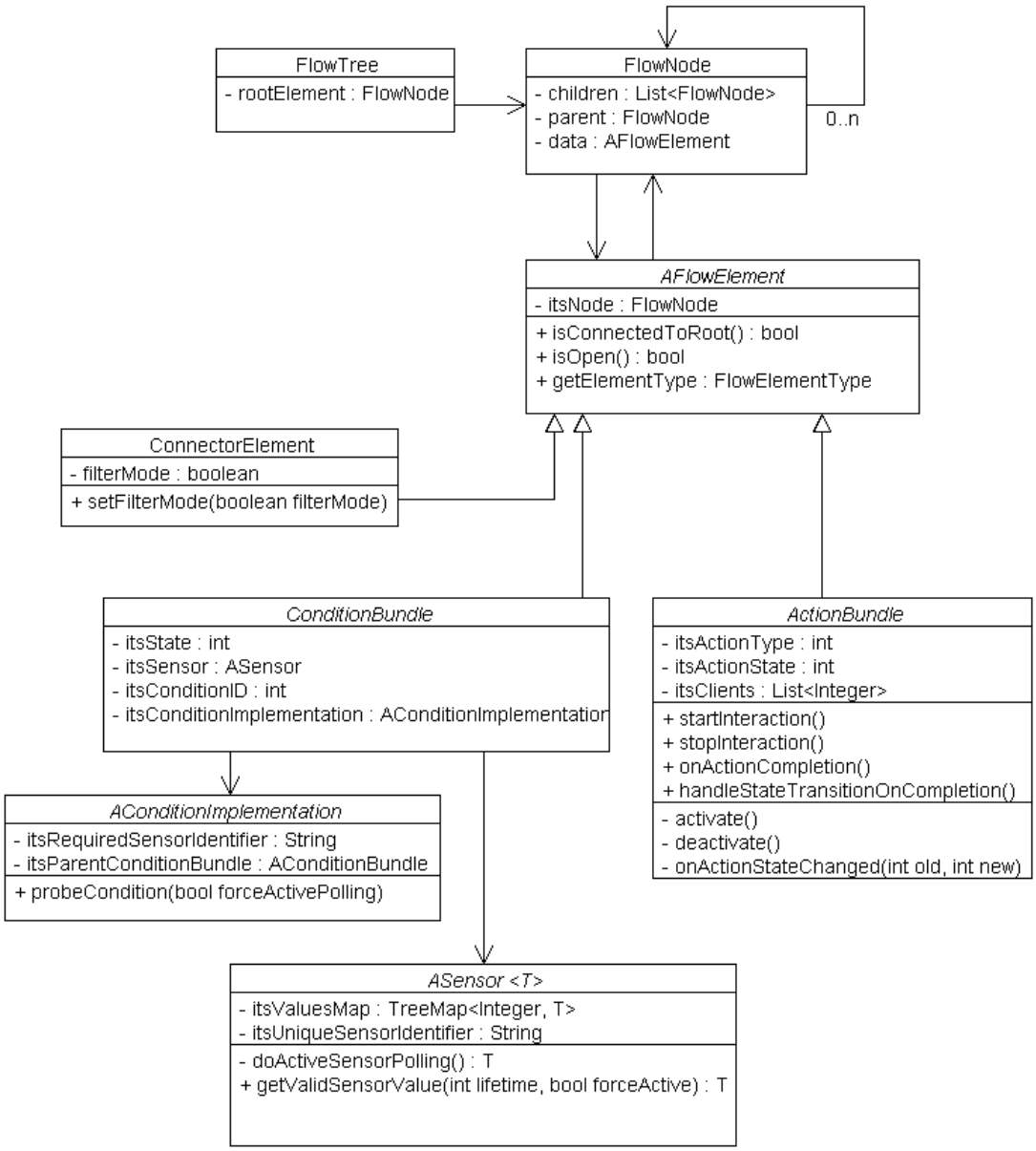

Abbildung 9 - Klassendiagramm der Trigger-Architektur

Die abstrakte Klasse AFlowElement stellt die Funktionen bereit, die von allen Elementen in einer Condition-Chain benötigt werden. Diese Elemente sind wie bereits in Abbildung 8 vorgestellt das ConditionBundle, das ActionBundle und das Connector-Element mit seiner Filterfunktion. Für jedes dieser Elemente muss geprüft werden können ob es über eine „offene“ Verbindung zum root-Element verfügt bzw. ob es selbst als „offen“ gilt. Ein einzelnes Element gilt je nach Element-Typ und evaluiertem Kontext als offen oder geschlossen. Die genauen Kontextbedingungen sind in den jeweiligen Kapiteln 5.2.3, 5.2.6 und 5.2.7 spezifiziert. Ein Pfad zum Root-Element gilt als „offen“ wenn alle vorhergehenden Bedingungen und Connector-Elemente korrekt erfüllt und entlang des passenden Kettenpfads gefiltert werden (siehe Abbildung 10). Die Con-

dition-Elemente selbst gelten zu jeder Zeit als „open“, da erst die nachfolgenden Connector-Elemente als Filter fungieren.

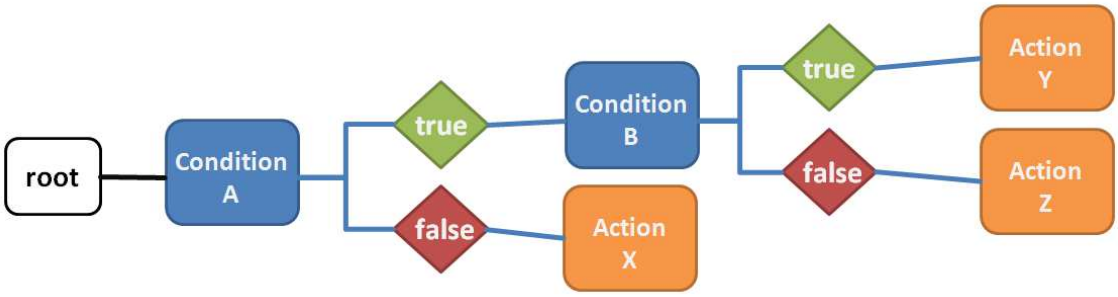

Abbildung 10 - Beispiel einer Condition-Chain

Anhand des Condition-Chain Beispiels von Abbildung 10 kann erläutert werden welche Elemente als „offen“ oder „geschlossen“ gelten und welche Auswirkungen dies auf nachfolgende Elemente hat. Angenommen die **Condition A** würde als **true** und die **Condition B** als **false** evaluiert. Dadurch ergeben sich für die Connector-Filter-Elemente folgende open-Stati:

- **true**-Connector nach **Condition A** : open
- **false**-Connector nach **Condition A** : closed
- **true**-Connector nach **Condition B** : closed
- **false**-Connector nach **Condition B** : open

In dieser Konstellation hätte nur die **Action Z** eine durchgehend offene Verbindung zum Root-Element (über Root, Condition A, True, Condition B, False, Action Z) und würde somit aktiviert werden. Alle anderen Actions haben keine durchgehend offene Verbindung zum Root-Element und werden dementsprechend deaktiviert.

Die einzelnen Elemente und zugehörige Klassen der Trigger-Architektur werden in den folgenden Kapiteln detailliert beschreiben.

5.2.1 Sensor

Das Sensor-Element ermöglicht entweder den Zugriff auf einen real existierenden Hardwarebasierten Sensor (z.B. GPS oder Herzfrequenz) oder auf die Daten eines „virtuellen“ Sensors (z.B. Datum/Zeit oder das Ergebnis einer User-Action). Während nicht nur viele externe Hardware-Sensoren denkbar sind, ermöglicht auch ein „virtueller“ Sensor eine breite Palette von Funktionen. Beispielsweise wäre es denkbar beliebige externe Webservices als virtuellen „Sensor“ anzubinden welche dann, auf ein Systemereignis oder eine Nutzereingabe reagieren und so als Input dienen. Alternativ könnte solch ein Webservice auch aufbereitete Log-Daten eines nicht direkt zugreifbaren exter-

nen Sensors (z.B. Wetterstation oder Seismograph) bereitstellen, und auf diesem Weg für das PocketBee System zugreifbar machen.

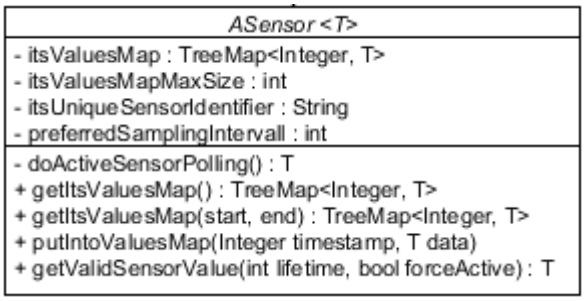

Abbildung 11 - Die abstrakte Klasse ASensor<T>

Die abstrakte Klasse *ASensor<T>* (siehe Abbildung 11) besitzt eine TreeMap-Datenstruktur [3] welche alle bisher gemessenen Datensätze vom generischen Typ <T> zusammen mit ihren Messzeitpunkten als Timestamp vorhält. Diese Datenstruktur kann als „Gedächtnis“ des Sensors verstanden werden und sorgt dafür, dass nicht nur der jüngste Messwert zugreifbar ist, sondern der Forscher auch mit dem bisherigen Verlauf der Sensorwerte arbeiten kann. Um ein unkontrolliertes wachsen dieser Datenstruktur zu verhindern, existiert für jeden Sensor der Wert *itsValuesMapMaxSize* welcher die maximale Größe der Datenstruktur definiert. Überschreitet die Datenstruktur diese Größe z.B. durch das hinzufügen eines Messwertes mit *putIntoValuesMap(ts, value)* werden automatisch die ältesten Messwerte zugunsten des neuen Messwerts entfernt. Durch den generischen Datentyp <T> ist es möglich verschiedene Unterklassen zu implementieren welche zwar alle von der abstrakten Klasse ASensor erben, jedoch unterschiedliche Datentypen als Sensorwerte haben können. Beispielsweise wird der Herzfrequenz-Sensor einen primitiven Datentyp double verwenden, während der GPS-Sensor den komplexen Datentyp GPSPosition als Messwert speichert. Vor allem in Hinblick auf die Hardware-basierten Sensoren wird für jeden implementierten Sensor ein *preferredSamplingIntervall* in Sekunden angegeben, welches spezifiziert in welchem Takt dieser Sensor idealerweise abgefragt wird. Dies ist nur eine Richtlinie und kann im jeweiligen Condition-Bundle abweichend spezifiziert werden. Jedoch muss sich der Ersteller eines Condition-Bundles im Klaren sein, welchen Einfluss eine erhöhte Sampling-Rate auf den Batterieverbrauch des Geräts hat. Ein einzelner Sensor kann von mehreren Bedingungen (Conditions) implementiert und verwendet werden. Aus diesem Grund werden die Sensoren als einzelne globale Singleton-Klassen [29] realisiert. So existiert in der Anwendung nur eine einzige Instanz eines Sensors von der mehrere Bedingungen zu jeder Zeit den aktuellen Wert holen können was die Gesamtlaufzeit des Geräteakkus erhöhen kann. Mit dem Attribut *itsUniqueSensorIdentifier* kann ein Sensor eindeutig identifiziert werden.

5.2.2 Condition

Eine Bedingung (Condition) ist eine Implementierung der abstrakten Klasse Condition-Bundle (Kapitel 5.2.4) und stellt dementsprechend alle Methoden dieser Abstrakten Klasse bereit. Das Ziel einer neuen Condition ist die modulare Generierung einer neuen Bedingungsvariation die auf einen bestimmten Sensor zugreift und deren Validierungsmechanismus mit verschiedenen Condition-Implementations (Kapitel 5.2.3) ausgetauscht werden kann. Dementsprechend verbirgt sich die tatsächliche Logik einer Bedingung in einer ConditionImplementation während die Instanz der Condition selbst nur als Objekt-Container und für den Methodenzugriff dient.

5.2.3 Condition-Implementation

Eine ConditionImplementation dient zur Realisierung verschiedener Evaluationsmethoden die in einer einzelnen Bedingung über die Methode *probeCondition(forceActive)* angestoßen werden. Jede Condition referenziert zu jeder Zeit auf genau eine Condition-Implementation (siehe Abbildung 13). Es soll dem Forscher einfach möglich sein eine neue Bedingung für einen bestimmten Sensor zu spezifizieren, z.B. GPS oder Zeit. In dieser Bedingung sollte dann die passende Evaluationsmethode in Form einer Condition-Implementation ausgewählt werden können. Ein typisches Beispiel sind die Zeit- und Datumsbedingungen, welche unter Anderem die unterschiedlichen Implementierungen „regelmäßig wiederholend“ bzw. „einmaliges Ereignis“ besitzen. Die Methode *probeCondition(forceActive)* leitet den Aufruf an die aktuell initialisierte Condition-Implementation weiter. Somit kann ein Forscher zu einzelnen Bedingungen sehr unterschiedliche Evaluationmethoden als Condition-Implementation realisieren.

Jede Condition-Implementation legt über den Parameter *itsRequiredSensorIdentifier* fest, welchen Sensor diese Implementierung zur Evaluation ihrer Bedingung braucht. Ob der benötigte Sensor mit dem tatsächlich verwendeten übereinstimmt wird beim setzen einer Condition-Implementation überprüft.

5.2.4 Condition-Bundle

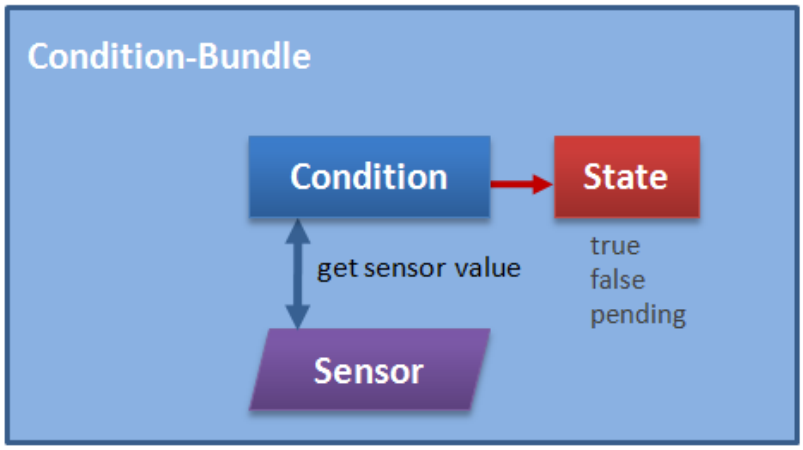

Abbildung 12 - Schematische Struktur eines Condition-Bundles

Ein Condition-Bundle enthält bei detaillierter Betrachtung (siehe Abbildung 12 und Abbildung 13) eine Sensorreferenz (Kapitel 5.2.1), eine Bedingungsimplementierung (siehe Kapitel 5.2.3 Condition-Implementation), einen Bedingungsstatus sowie eine systemweit eindeutige Bedingungs-ID. Während der Sensor und die Bedingungsimplementierung bereits beschrieben wurden fokussiert sich dieses Kapitel auf die weiteren Attribute und Methoden eines Condition-Bundles.

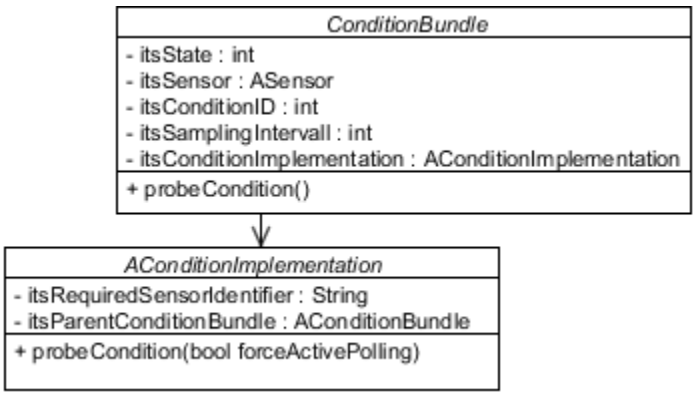

Abbildung 13 – Die Klassen ConditionBundle und ConditionImplementation

Im Konstruktor jedes Condition-Bundles werden der benötigte Sensor, die verwendete Condition-Implementation und die eindeutige Condition-ID gesetzt. Damit ist die Bedingung ein modulares Element welches auch für sich allein stehend evaluiert werden kann. Außerdem verfügt ein Condition-Bundle über den internen Bedingungsstatus *itsState*. Dieser kann nach der Evaluierung durch die Methode *probeCondition()* die drei verschiedene Zustände FALSE, TRUE und PENDING annehmen.

Die Bedeutung dieser Zustände wird im Folgenden erläutert:

- STATE_FALSE (*itsState* = 0)
  Die Bedingung ist seit der letzten Evaluierung im Zustand „nicht Erfüllt“.

- STATE_TRUE (*itsState* = 1)
  Die Bedingung ist seit der letzten Evaluierung im Zustand „Erfüllt“

- STATE_PENDING (*itsState* = 2)
  Die Bedingung ist seit der letzten Evaluierung im logischen Zustand „nicht Erfüllt“ wartet jedoch auf ein externes Ereignis um ihren Status zu ändern. Dieser Status kann zur Realisierung von Bedingungen verwendet werden, die nicht durch das klassische polling eines Sensors abgefragt werden können (z.B. Zeitliche Aktivierung durch den Android Alarm-Manager).

Diese Status-Information der Bedingung wird von den in der Bedingungskette nachfolgenden Connector-Elementen dazu verwendet, bestimmte Zweige der Condition-Chain zu filtern. Während ein Connector-Element mit True-Filter logischerweise nur durch schält wenn die vorhergehende Bedingung den Status „STATE_TRUE“ hat, wird der Status „STATE_PENDING“ ebenfalls wie der Status „STATE_FALSE“ behandelt.

Als Besonderheit des Status „STATE_PENDING“ in einer Condition-Chain gilt, dass nachdem ein Condition-Bundle den Status von „STATE_PENDING“ auf „STATE_TRUE“ gewechselt hat alle betroffenen Elemente der Kette aktiv auf ihren Status geprüft werden müssen. Dies hat den Zweck, dass präzise getimte Events wie z.B. ein zeitlicher Alarm des Alarm-Managers auch tatsächlich im passenden Augenblick eine gewünschte Aktion auslösen können und nicht durch die verzögerte Anfrage anderer Bedingungen in der Kette behindert werden.

Um festzulegen in welchen Abständen eine Bedingung normalerweise geprüft werden sollte kann der Forscher den Parameter *itsSamplingIntervall* in Sekunden definieren. Durch diesen Wert wird bestimmt wie oft eine Bedingung aktiv den Sensorwert ausliest, bzw. nach welcher Zeitperiode ein Sensorwert seine Gültigkeit verliert. Das SamplingIntervall kann für jede Instanz einer Bedingung in der Studie angepasst werden und orientiert sich meist am Intervall des genutzten Sensors. Das System sorgt automatisch dafür, dass alle Bedingungen in ihrem spezifischen Intervall abgefragt werden und gruppiert überschneidende Intervalle um Energie zu sparen.

Ein Condition-Bundle wird von der Funktion *isOpen()* grundsätzlich den Wert „true“ zurückgeben, egal welchen Wert der aktuelle Zustand hat, da die nachfolgenden Connector-Elemente für die Filterung des Ergebnisses zuständig sind.

5.2.5 Action

Eine Action erbt von der abstrakten Klasse Action-Bundle (5.2.6) und stellt neben den abstrakten Methoden auch weitere Methoden bereit die jedoch für jede Instanz getrennt Implementiert werden müssen. Diese Methoden können beliebig vom Entwickler gestaltet werden und beinhalten den jeweiligen Programmcode der ausgeführt werden soll, wenn das Action-Bundle seinen Status ändert.

- activate()
  Wird ausgeführt wenn der Zustand des Action-Bundles von *inactive* auf *active* wechselt. Im Falle einer User-Action (ref) wird diese für den Nutzer initialisiert (z.B. einblenden eines Fragebogens). Im Falle einer System-Action wird diese direkt gestartet (z.B. Logging des GPS Sensor).

- deactivate()
  Wird ausgeführt wenn der Zustand des Action-Bundles von *active* auf *inactive* bzw. *disabled* wechselt. Im Falle einer User-Action wird die eingeblendete Anzeige für den Nutzer wieder ausgeblendet und ggf. eine Timeout-Information gespeichert (z.B. Fragebogen wurde in aktivem Zeitraum nicht ausgefüllt). Eine System-Action stoppt alle ausgelösten Systemaktivitäten (z.B. Sensor-Logging).

- startInteraction()
  Wird nur bei User-Actions ausgeführt wenn der Nutzer mit der Interaktion beginnt (z.B. starten eines eben auf dem Widget eingeblendeten Fragebogens).

- stopInteraction()
  Wird nur bei User-Actions ausgeführt wenn der Nutzer die Interaktion mit der Action beendet (z.B. Fragebogen wurde ausgefüllt und beendet).

- onActionCompletion()
  Muss ausgeführt werden nachdem die Action manuell vom Nutzer abgeschlossen wurde bzw. nachdem eine System-Action beendet wurde. Diese Methode sorgt für den korrekten Abschluss einer Action und behandelt die entstandenen Ergebnisse.

5.2.6 Action-Bundle

Ein Action-Bundle repräsentiert ein Paket das sich um eine einzelne Action herum aufbaut (siehe Abbildung 14). Es enthält zusätzlich einen Action-State, einen Action-Type sowie mehreren Probanden die diesem Action-Bundle zugeordnet sind. Während die Action verschiedene Reaktionsmethoden implementiert (siehe 5.2.5) stellt das Action-Bundle die umgebenden Parameter bereit.

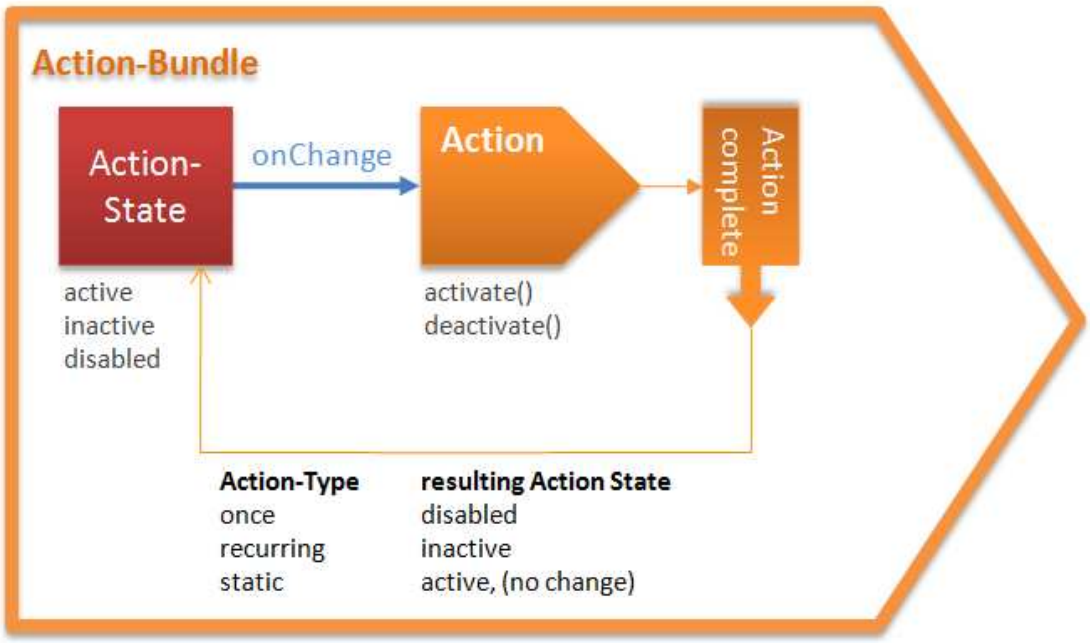

Abbildung 14 - Schematische Struktur eines Action-Bundles

Der Action-State kann drei verschiedene Zustände annehmen: *active*, *inactive* und *disabled*. Diese Zustände werden vom System nur verändert, wenn sich die vorhergehenden Bedingungen in der Condition-Chain ebenfalls in passender Kombination verändern. Bei einem Zustandswechsel auf *active* wird die Methode *activate()* aufgerufen. Bei einem Zustandswechsel zu *inactive* oder *disabled* wird die Methode *deactivate()* ausgeführt. Als Besonderheit gilt, dass wenn das Action-Bundle einmal den Zustand *disabled* hat kann es nicht wieder aktiviert werden. So können Aktionen realisiert werden die nur ein einziges Mal ausgeführt werden sollen.

Wie bereits beschrieben hat ein Zustandswechsel Einfluss auf die ausgeführten Methoden der Action. Dementsprechend werden die Methoden *activate()* und *deactivate()* nur bei einem tatsächlichen Zustandswechsel ausgeführt.

Der Parameter *itsClients* enthält eine Liste mit den IDs aller betroffenen Probanden. Die Action kann somit dediziert für verschiedene Probanden aktiviert oder deaktiviert werden.

Der Typ eines Action-Bundles bestimmt die Reaktion und die Veränderung des Status nachdem die Action beendet wurde, unabhängig davon ob ein Nutzer oder das System selbst die Action beendet. Die Tabelle 1 beschreibt hierbei die verschiedenen Typen und zeigt die Zustandsveränderungen die nach Beendigung der Action auftreten.

Action-Type	**Beschreibung & Zustandswechsel**
once	Wird für Actions verwendet die nur ein einziges Mal aktiviert werden dürfen, z.B. einmalige Fragebögen. → resultierender Zustand nach Beendigung: *disabled*
recurring	Wird für Actions verwendet die wiederholend auftreten können, z.B. täglich wiederholende Fragebögen. → resultierender Zustand nach Beendigung: *inactive*
static	Wird für Actions verwendet die während eines längeren Zeitraums ständig aktiviert bleiben und ihren Zustand nicht wechseln, z.B. ständig aktive Kernfragen. → resultierender Zustand nach Beendigung: *active*

Tabelle 1 - Der Parameter Action-Type mit resultierenden Zustandswechseln

Eine Action kann wiederrum als Bedingungselement dienen da es nötig sein kann, das Ergebnis einer Action weiterzuverwenden oder weitere Actions von der Beendigung einer vorhergehenden Action abhängig zu machen. Zu diesem Zweck kann ein spezielles Condition-Bundle hinter einer Action angeordnet werden. Diese Bedingung analysiert statt einem Sensorwert die Ergebnisse der vorhergehenden Action (z.B. die manuelle Eingabe eines Wertes vom Nutzer).

5.2.7 Connector-Element

Ein Connector-Element dient zum Filtern der Bedingungsergebnisse in einer Condition-Chain (Abbildung 8). Es verfügt über den Parameter *filterMode* der definiert welches Ergebnis die vorhergehende Bedingung haben muss, damit dieses Connector-Element als „open“ gilt. Es folgen maximal zwei Connector-Elemente auf ein Condition-Bundle in der Condition-Chain. Der jeweilige *filterMode* kann entweder *true* oder *false* sein, jedoch dürfen keine zwei Connector-Elemente mit demselben *filterMode* auf der gleichen Ebene existieren. Wie bereits beschrieben evaluiert die Methode *isOpen()* das Ergebnis der vorhergehenden Bedingung und gibt den entsprechenden Wert zurück. Wenn der *filterMode true* ist muss die vorhergehende Bedingung ebenfalls erfüllt sein damit das Connector-Element als *true* gilt. Wenn das Ergebnis der vorhergehenden Bedingung nicht mit dem *filterMode* übereinstimmt wird das Connector-Element als nicht *open* behandelt.

5.3 Verarbeitung

Während im vorhergehenden Kapitel 5.2 auf die Architektur und deren Elemente eingegangen wurde beschäftigt sich dieses Kapitel mit der Trigger-Verarbeitung während der Konfiguration sowie der Laufzeit des Systems. Es muss sichergestellt werden, dass alle vom Forscher definierten Trigger an die Client-Geräte übertragen werden, diese dort regelmäßig geprüft und bei Bedarf abgearbeitet werden. Der Verarbeitungsmechanismus teilt sich in folgende Teilbereiche auf: Die Konfiguration und Datenhaltung auf dem Server, die Synchronisation zwischen Client und Server, die regelmäßige Prüfung aller Bedingungen auf dem Client, die Verarbeitung von Statusänderungen einer Bedingung und die Systemreaktion auf aktivierte Actions. Im Folgenden werden diese Teilbereiche einzeln erläutert.

5.3.1 Konfiguration und Datenhaltung auf dem Server

Die später in Kapitel 6 vorgestellte grafische Benutzeroberfläche ermöglicht es dem Forscher Condition-Chains für komplexe Studienszenarien zu definieren und abzuspeichern. Solche Konfigurationen in Form von Bedingungen, Aktionen und Connectoren können direkt vom Server aus an die Probanden zugewiesen werden. Eine einzelne Condition-Chain wird in Form einer XML-Datei gespeichert und auf das Gerät des Probanden übertragen. Eine alternative Datenhaltung auf dem Server könnte durch die Verwendung einer speziellen XML-Datenbank realisiert werden. Für die Übertragung zum Client ist es jedoch sinnvoll XML-Dateien nur einzeln zu senden um die mobile Datenverbindung zu entlasten und bei Fehlern die Übertragung erneut zu starten.

5.3.2 Synchronisation zwischen Client und Server

Die Synchronisation zwischen Client und Server sorgt dafür, dass der Client zu jeder Zeit auf dem neuesten Stand der Studienkonfiguration ist und dass alle Bedingungen und Aktionen übertragen werden. Im Gegensatz zur „Polling-Methode“ des Pilotsystems welche sehr energiehungrig war, kann seit der Android Version 2.2 die „Cloud-to-Device“ Technik verwendet werden [12]. Dies sorgt dafür, dass die Clientgeräte nur benachrichtigt werden, wenn tatsächlich neue Informationen zur Verfügung stehen. Beispielsweise wird das Gerät des Probanden benachrichtigt, sobald der Forscher einen neuen Fragebogen erstellt, zugewiesen und aktiviert hat. Daraufhin verbindet sich das Gerät des Probanden mit dem Server und lädt die neueste Konfiguration und den entsprechenden Fragebogen herunter. Außerdem kann der Forscher vom ControlCenter aus sehen, welche Actions (z.B. Fragebögen) bereits beim Probanden aktiviert und ausgefüllt wurden bzw. welche Probanden ihre Aufgaben nicht erfüllt haben.

5.3.3 regelmäßige Prüfung aller Bedingungen auf dem Client

Um wichtige Situationen nicht zu verpassen kann der Forscher für jede Bedingung eine bestimmte „sampling rate“ einstellen. Diese Rate gibt an nach welcher Zeitspanne eine Bedingung wieder aktiv geprüft wird. Sollte eine Bedingung geprüft werden deren valide Zeitspanne abgelaufen ist wird diese Bedingung den entsprechenden Sensor aktiv auslesen statt den passiv gespeicherten Sensorwert der letzten Abfrage zu verwenden.

Um diese Methode zu realisieren werden nach dem Laden einer neuen Condition-Chain alle benötigten Bedingungen vom Client in einer speziellen Verarbeitungsqueue gespeichert. Um Energie zu sparen werden Bedingungen, deren sampling-rate gleich ist, in Zyklen zusammengefasst und entsprechend an gemeinsamen Zeitpunkten geprüft. Nach der Evaluation einer Bedingung wird geprüft ob sich deren Status geändert hat.

5.3.4 Verarbeitung von Statusänderungen einer Bedingung

Wenn sich der Status einer Bedingung durch die Prüfung geändert hat wird eine Referenz auf diese Bedingung in die „recently-changed-queue“ gespeichert. Nachdem alle Bedingungen dieses Zyklus geprüft wurden startet die Verarbeitung dieser Queue. Die Elemente der „recently-changed-queue“ werden entsprechend ihrer Position in der Condition-Chain sortiert. Je weiter „vorne“ eine Bedingung in einer Condition-Chain positioniert ist, desto höher ist ihre Relevanz in der Bearbeitung

Die Änderung eines Bedingungsergebnisses ist nur Relevant, wenn die Bedingung selbst eine direkte Verbindung zur Wurzel der Condition-Chain besitzt. D.h. alle vorhergehenden Elemente müssen aktiviert sein und ebenfalls eine Verbindung zur Wurzel besitzen (siehe Abbildung 15).

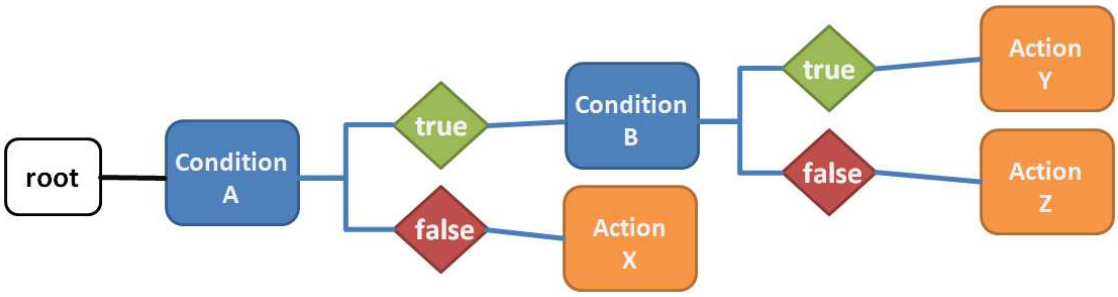

Abbildung 15 - visuelle Darstellung einer Condition-Chain

Ist eine Bedingung nicht „aktiv“ mit der Wurzel der Condition-Chain verbunden ist ihr Ergebniswechsel irrelevant für den aktuellen Status des Systems. Wenn die Bedingung jedoch eine aktive Verbindung zur Wurzel (links) besitzt ist ihr Ergebniswechsel relevant und muss an alle tiefer liegenden Elemente (rechts) publiziert werden. Das System arbeitet sich nun von der Bedingung aus tiefer in die Condition-Chain vor, bis das Ende

erreicht ist. Statt alle auf diesem Weg gefundenen Actions zu aktivieren oder zu deaktivieren werden diese mit dem jeweils neuesten Aktionswert in eine weitere Queue gespeichert und später verarbeitet. Dies ist nötig, damit Actions nicht mehrfach aktiviert und deaktiviert werden wenn sich mehr als eine Bedingung in der Kette ändert.

5.3.5 Aktivieren und deaktivieren von Actions

Wenn die Statusänderung einer Action nicht zuerst alle Bedingungsänderungen abwarten würde, könnte im System ein „flackern“ entstehen welches sich dadurch auszeichnet, dass einzelne Actions mehrere male hintereinander aktiviert und wieder deaktiviert würden. Durch die Reihenfolge der Bedingungsverarbeitung und die Zwischenspeicherung des jeweils aktuellsten Änderungswertes für eine Action wird dieses flackern abgefangen. Nachdem alle Bedingungen eines Zyklus evaluiert wurden und die letzte Action in die „activation-queue“ gespeichert ist können die aktuellsten Elemente der activation-queue verarbeitet werden. Die einzelnen Actions werden nun, entsprechend des letzten Status, entweder aktiviert oder deaktiviert. Um eine Action zu aktivieren bzw. zu deaktivieren werden die jeweiligen Methoden aufgerufen und ausgeführt (Kapitel 5.2.5).

5.4 Beispielhafte Implementierung: GPS

Nachdem die theoretische Struktur des Trigger-Frameworks erklärt wurde folgt in diesem Kapitel eine beispielhafte Implementierung anhand des GPS Sensors. Es werden alle Klassen beschrieben die nötig sind um eine GPS-Bedingung zu konfigurieren und eine entsprechende Aktion (Fragebogen) dadurch auszulösen. Wie von der Architektur vorgeschrieben müssen für dieses Beispiel sowohl ein eigener Sensor (*GPSSensor*), ein Datentyp für den Sensor (*GPSPosition*), eine Bedingung (*GPSCondition*), eine Bedingungsimplementierung (*GPSConditionStandardImplementation*) sowie eine Action (*QuestionnaireStandardAction*) mit entsprechenden Funktionen erstellt werden. Die finale Konstruktion der Klassen wird in Abbildung 16 deutlich.

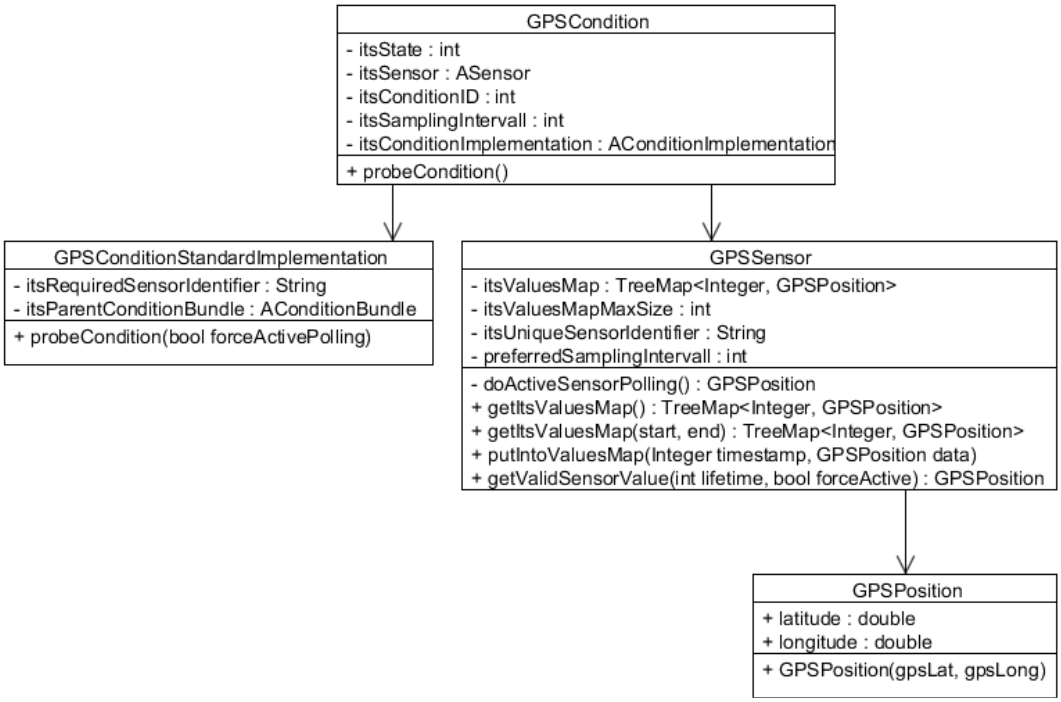

Abbildung 16 - Klassendiagramm der Beispielimplementierung GPS

Nachfolgend werden die Implementierungen der einzelnen Klassen beschrieben.

5.4.1 GPSSensor

GPSSensor ist die Implementierung der Abstrakten Klasse ASensor<T> wobei T in diesem Fall vom Datentyp GPSPosition besetzt ist. Dieser Datentyp wird in der Klasse als TreeMap vorgehalten um mehrere Messwerte mit entsprechendem Zeitstempel speichern zu können. Nachfolgend findet sich die Implementierung der Klasse GPSSensor ohne die bereits in der abstrakten Klasse implementierten Methoden.

```
public class GPSSensor extends ASensor<GPSPosition> {

    private String TAG = this.getClass().getName().toString();

    public GPSSensor() {
        //preferred Sampling intervall (60 min) and unique sensor identifier
        super(3600, "de.ukn.hci.android.pocketbee.trigger.sensors.GPSSensor");
    }

    @Override
    protected GPSPosition doActiveSensorPolling() {

        GPSPosition currentPosition = new GPSPosition(0, 0);

        Context gac = (Context)StartTest.itsContext;

        try {
            LocationManager lm = gac.getSystemService(Context.LOCATION_SERVICE);

            LocationListener myLocationListener = new GpsLocationListener();
            if( myLocationListener != null ) {
                lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0);
```

```
                if (lm != null) {
                    currentPosition.latitude=lm.getLastLocation().getLatitude();
                    currentPosition.longitude=lm.getLastLocation().getLongitude();
                } else {
                    Log.e(TAG,"GPS Sensor failed. LocationManager lm was null.");
                }
            } else {
                Log.e(TAG,"GPS Sensor failed. myLocationListener was null.");
            }

        }catch ( Exception e ) {
            Log.e(TAG,"GPS Sensor failed to receive position data.");
        }

        //get timestamp from android
        int timestamp = (int)System.currentTimeMillis();

        //put location into log-map
        this.putIntoValuesMap(timestamp, currentPosition);

        //return obtained gps location
        return currentPosition;
    }
}
```

Codebeispiel 1 - GPSSensor

5.4.2 GPSPosition

Der Datentyp GPSPosition wird vom GPSSensor verwendet um sowohl latitude als auch longitude in einem Wert zu vereinen. Dies ermöglicht die Speicherung in der generischen Map

```
public class GPSPosition {

    public double latitude;
    public double longitude;

    public GPSPosition(double gpsLat, double gpsLong) {
        super();
        this.latitude = gpsLat;
        this.longitude = gpsLong;
    }
}
```

Codebeispiel 2- Der Datentyp GPSPosition

5.4.3 GPSCondition

Die Klasse GPSCondition repräsentiert ein abstraktes Condition-Bundle und dient als Container für den benutzten Sensor und die entsprechende Bedingungsimplementierung. Daher muss in dieser Klasse nur der Konstruktor überschrieben werden der später vom System aufgerufen wird. Dieser übergibt sowohl die Singleton-Klasse des Sensors als auch die Bedingungsimplementierung die zur Evaluation genutzt werden soll an den Konstruktor der Elternklasse. Die Bedingungs-ID dient im gesamten System zur Identifizierung dieser Bedingung.

```
public class GPSCondition extends AConditionBundle {

    private String TAG = this.getClass().getName().toString();

    public GPSCondition( AConditionImplementation implementationToBeUsed,
                         int conditionIDToBeUsed) {
        /**
         * @developer specify your used sensor here, use a singleton class
         */
        super( GPSSensor.getInstance(),
               implementationToBeUsed,
               conditionIDToBeUsed);

    }
}
```

Codebeispiel 3 - Das Condition-Bundle GPSCondition

5.4.4 GPSConditionStandardImplementation

Die Bedingungsimplementierung welche dem Condition-Bundle *GPSCondition* übergeben wird muss im Voraus definiert und implementiert werden. Über diese Klasse wird später die Evaluation der Bedingung durchgeführt. In der Methode probeCondition() wird jeweils die spezielle Auswertung der Sensordaten vorgenommen. In diesem Beispiel wurde die tatsächliche Berechnung aus Platzgründen ausgelagert.

```
public class GPSConditionStandardImplementation extends
        AConditionImplementation {

    //parameters which can be specified within the user interface
    private GPSPosition targetPos;
    private long targetRadius;

    public GPSConditionStandardImplementation() {
        //specify sensor to be used by identifier
        super(new GPSSensor().getItsUniqueSensorIdentifier() );
    }

    @Override
    public boolean probeCondition(boolean forceActivePolling) {

        //obtain the current value of the sensor
        GPSPosition gpsPos = (GPSPosition) itsParentConditionBund-
le.getItsSensor().getValidSensorValue(1, true);

        //calculate if we are within the range of the targetArea and return
        //  _____
        // |     |
        // |  x  |
        // |_____|
        //
        return positionIsWithinTarget();
    }
}
```

Codebeispiel 4 - GPSConditionStandardImplementation

5.4.5 QuestionnaireStandardAction

Nachdem die Bedingungen implementiert wurden muss für die abstrakte Klasse AActionBundle ebenfalls eine Implementierung erstellt werden. Im aktuellen Beispiel wurde

eine Action zur Einblendung eines Fragebogens verwendet. Die vorliegende Implementierung ist aus Platzgründen verkürzt und geht von einer QuestionnaireFactory Klasse aus die die Erstellung eines Fragebogens auf der XML-Basis übernimmt.

```
public class QuestionnaireStandardAction extends AActionBundle {

    //parameter which can be set with the user interface
    private int questionnaireID;

    @Override
    public void activate() {

        //get questionnaire object
        QuestionnaireFactory qnf = new QuestionnaireFactory();
        Questionnaire myQuestionnaire = qnf.serializeQNFromXML(questionnaireID);

        //show questionnaire on widget
        WidgetDB wdb = new WidgetDB();
        wdb.insertIntoWidgetDB(myQuestionnaire);
    }

    @Override
    public void deactivate() {

        //hide questionnaire from widget
        WidgetDB wdb = new WidgetDB();
        wdb.deleteQuestionnaireFromWidget(questionnaireID);
    }




    @Override
    public void onActionCompletion() {

       //get questionnaire result
        QuestionnaireFactory qnf = new QuestionnaireFactory();
        QuestionnaireResult result = qnf.getQNResult(questionnaireID);

        //transfer result file to server
        DataTransferHelper dth = new DataTransferHelper();
        dth.sendResultToServer(result);

        /* INFO:
         * always let the the state transition be handled after a manual or
         * automated completion this is essential in every implementation
         * of any action, don't forget!*/
        this.handleStateTransitionOnCompletion();
    }

    @Override
    public void startInteraction() {

        //start questionnaire acitivity instance
        Intent intent = new Intent(this, QuestionnaireActivity.class);
        intent.putExtra("ID", questionnaireID);
        context.startActivity(intent);
    }

    @Override
    public void stopInteraction() {

        //end questionnaire activity instance, save state of qn
        QuestionnaireFactory qnf = new QuestionnaireFactory();
        qnf.saveCurrentQuestionnaireState(questionnaireID);
    }
}
```

Codebeispiel 5 - QuestionnaireStandardAction

6 Konzeption eines Interfaces zur Trigger-Definition

Wie bereits erwähnt existiert im aktuellen System mit dem ControlCenter zwar eine grafische webbasierte Oberfläche für den Forscher, jedoch beschränkt sich diese auf die Erstellung und Verwaltung von einigen Grundelementen wie Fragebögen, Kernfragen und Aufgaben. Sowohl die Erkenntnisse aus Kapitel 4.4 als auch die Anforderungen an zukünftige ESM-Tools von Khan et. al. [19] sehen einen Bedarf einer visueller Definitionssprache um Trigger und Bedingungen einfach erstellen zu können. Auch um den Fähigkeiten des neuen Trigger-Frameworks Rechnung zu tragen, wird in diesem Kapitel ein Konzept vorgestellt, dass die Modularität und die Vielfalt der neuen Sensoren und Events auch auf der Benutzeroberfläche widerspiegelt. Das Interfacekonzept basiert auf dem Pipe/Filter-Prinzip (siehe Kapitel 6.2.1) welches in einer semantisch zoombaren Oberfläche (siehe Kapitel 6.2.2) verankert ist. Die Inspiration und Codebasis zu dieser visuellen Definitionsprache stammt vom Projekt „Squidy“ [24] das bisher zur Kontrolle von verschiedenen Eingabegeräten verwendet wird.

Wie bereits in [1] und [18] gezeigt ist eine visuelle Spezifikationssprache ein leicht zu erlernendes Werkzeug um boolesche Operatoren zu kombinieren. Dies ist vor allem für Nutzer wichtig, die in ihrer täglichen Arbeit nicht mit dem mathematischen bzw. informationstechnischen Konzept der booleschen Logik konfrontiert sind.

In diesem Kapitel werden zunächst die Anforderungen an eine grafische Oberfläche zur Triggererstellung festgelegt. Anschließend werden die verwendeten Visualisierungskonzepte vorgestellt und auf gemeinsame Synergien geprüft. Später wird sowohl das Interfacekonzept als auch das Interaktionsdesign der neuen Oberfläche detailliert vorgestellt.

6.1 Anforderungen

Um seine Forschungsfragen zu beantworten muss der Forscher eine Studie planen, durchführen und analysieren können. Während der Planung muss der Forscher definieren welche Probanden welches Studiendesign präsentiert bekommen, die Geräte entsprechend präparieren und spezielle Mechanismen zur Datensammlung vorbereiten. Es muss klar sein, welche Daten gesammelt werden und nach welchen Ereignissen welche Aktivitäten vom Probanden gefordert werden. Wie bereits Khan et. al. [19] fordern sollten all diese Einstellungen idealerweise an einem zentralen grafischen Interface vorgenommen werden. Die Logik und Funktionalität der Trigger sollte an diesem Interface auch für nicht technikaffine Forscher durchschaubar sein und es ihnen so ermöglichen komplexe Konstrukte ohne fremde Hilfe zu erstellen. Die folgenden Anforderungen werden zusätzlich durch die Grundsätze der Dialoggestaltung der ISO 9241-110 [7] gestützt:

- Das Interface soll den Nutzer bei der Erstellung von Triggern, Bedingung und Actions unterstützen. (klare Benutzerführung)
- Fehler in der Logik müssen vom System sofort erkannt werden. (Fehlertoleranz)
- Das Interface muss alle Varianten und logischen Möglichkeiten des zugrundeliegenden Trigger-Frameworks unterstützen. (Steuerbarkeit)
- Das Interface soll die zugrundeliegende Logik des Trigger-Frameworks durch ein leicht erlernbares mentales Modell darstellen. (Lernförderlichkeit)
- Das Interface sollte leicht an mögliche Erweiterungen des Trigger-Frameworks angepasst werden können. (Individualisierbarkeit an Erweiterungen des Nutzers)
- Das Prinzip und die Funktionen des Interfaces müssen von einem Forscher innerhalb eines Arbeitstages erlernbar sein. (Selbstbeschreibungsfähigkeit)

6.2 Visualisierungen

Um die Anforderungen erfüllen zu können stützt sich das neue Interface auf das Filter/Flow Prinzip [27] welches zusammen mit einer zoombaren Oberfläche eine leistungsstarke Kombination darstellt. Die Anregung zu dieser Kombination bot das Framework „Squidy“ [24]. Während das Pipe/Filter Prinzip die zugrundeliegende Funktionalität der Bedingungsketten und Actions widerspiegelt (Kapitel 5.2), bietet die zoombare Oberfläche eine Möglichkeit zwischen Detailansicht und Überblick einzelner Objekte zu wechseln. So kann der Inhalt eines Objekts genauso einfach spezifiziert werden wie die Relation von Objekten zueinander. Zusätzlich besteht die Möglichkeit mehrere zoombare Oberflächen ineinander zu verschachteln. Beispielsweise kann die Detailkonfiguration eines Fragebogens ebenso über das Pipe/Filter Prinzip realisiert werden wie die Anordnung des Fragebogenknotens selbst. Im Folgenden wird sowohl das Pipe/Filter Prinzip als auch die Idee des semantische Zooms kurz beschrieben. Es wird geprüft ob sich durch diese Kombination Synergien ergeben können.

6.2.1 Pipe & Filter Prinzip

Wie bereits von Lewis und Olson [20] festgestellt wurde fällt es Menschen ohne programmiertechnischen Hintergrund sehr schwer eine neue Programmiersprache zu erlernen. Dies gilt vor allem für die klassische Variante einer textuellen Programmiersprache. Da angenommen werden muss, dass PocketBee nicht nur von Informatikern sondern auch von Psychologen, Soziologen und Sportwissenschaftlern verwendet werden wird, ist es daher angebracht eine Art visuelle Programmiersprache zur Konfiguration des Systems bereit zu stellen.

Die Pipe/Filter Visualisierung von Shneiderman et. al. [27] entspricht einer solchen visuellen Spezifikationssprache mit der einzelne Elemente, deren Funktion und deren Verbindungen untereinander dargestellt werden können. Das Konzept basiert auf der

Annahme, dass einzelne Elemente (Filter) untereinander über so genannte Pipes verbunden sind. Der Datenfluss welcher durch das System geleitet wird fließt über diese Pipes und kann von den Filter-Elementen in seinen Eigenschaften und Inhalt verändert werden. Solche visuellen Programmiersprachen wurden bereits 1986 verwendet um digitale Schaltungen darzustellen (siehe Abbildung 17).

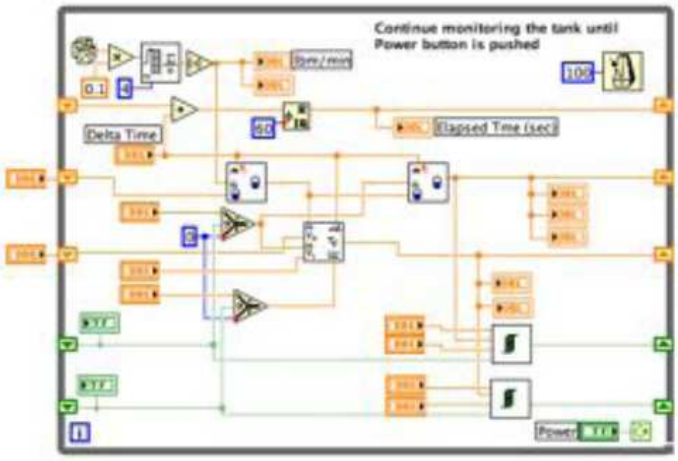

Abbildung 17 – Eine visuelle Programmiersprache als Basis von LABView [22]

Ein sehr ähnliches Prinzip, welches durch den binären Charakter des Datenflusses noch eher für die Anwendung im PocketBee System in Frage kommt ist die Filter/Flow Metapher von Young et. al. [27]. Diese von Young beschriebene Metapher versucht die boolesche Logik (AND, OR) auch für Nutzer verständlich zu machen die diese in ihrer täglichen Arbeit nicht verwenden. Dass dieses Vorgehen Erfolg hat zeigt die kürzlich durchgeführte Nutzerstudie der Anwendung Facet-Streams [18]. Dort wurden Probanden ohne mathematischen und informationstechnischen Hintergrund mit einer visuellen Spezifikationssprache konfrontiert die verwendet werden sollte um einen Datensatz nach bestimmten Kriterien zu durchsuchen. Die Probanden konnten diese Kriterien innerhalb weniger Minuten zu komplexen boolesche Ausdrücken kombinieren ohne vorher über die zugrundeliegende boolesche Logik aufgeklärt worden zu sein. Dies lässt zusammen mit den Erkenntnissen von Blackwell et. al. [1] darauf schließen, dass eine derartige Filter/Flow Methapher sehr gut dafür geeignet ist boolesche Logik in eine natürliche visuelle Sprache zu verpacken.

Von Greene und Petre wurde zu diesem Feld das so genannte „Cognitive Dimension Framework“ entwickelt, welches verschiedene Charakteristiken einer visuellen Programmiersprache beleuchten soll [13]. Unter Anderem sollte bei der Entwicklung einer solchen Sprache auf den dargestellten Abstraktionsgrad, die Fehleranfälligkeit und die Performance geachtet werden.

6.2.2 Semantischer Zoom

Durch die große Anzahl an Funktionen sind viele Benutzerschnittstellen überfüllt und müssen daher bestimmte Techniken verwenden um Inhalte zu verstecken und nur bei Bedarf anzuzeigen. Typische Beispiele hierfür sind Schnittstellen die in einzelne Seiten oder einzelne Registerkarten aufgeteilt sind. So lassen sich mit demselben Platzbedarf verschiedene Inhalte darstellen. Als Nachteil solcher Techniken wird oft der Verlust des Arbeitskontextes genannt [15] der sehr oft essentielle Information bereitstellt. Als Alternative bietet sich daher ein Interface an, bei dem der Nutzer durch selbständiges justieren der Ansicht (zoomen) entscheiden kann, ob er eine Detailansicht oder eine Kontextübersicht angezeigt haben will [5].

Während freies zoomen es einem Nutzer ermöglicht mit sehr vielen Objekten zu interagieren und gleichzeitig deren Kontext im Blick zu haben, sorgt genau diese Möglichkeit auch für eine zusätzliche mentale Belastung. Der Nutzer muss permanent entscheiden welche Zoomstufe für die aktuelle Aufgabe angemessen ist und ob alle relevanten Informationen auch tatsächlich angezeigt werden. Dem entgegen wirkt eine Erweiterung dieser Technik: der semantische Zoom. Semantisches Zooming kann als Unterkategorie des „goal-directed“ Zooms von Woodruff et. al. gesehen werden [30]. Hier wird als Reaktion auf eine Nutzeraktion, z.B. den Klick auf ein Objekt (Abbildung 18 a), eine konstante „zoombewegung“ ausgeführt, welche das gewünschte Objekt so in der visuellen Anzeige platziert, dass der Nutzer eine ideale Sicht auf dieses Objekt und einen Teil des Kontextes besitzt (Abbildung 18 b). Zusätzlich verändert sich dabei die tatsächlich angezeigte Repräsentation des Objekts. Durch den zusätzlich erlangten Platz ist es nun möglich sowohl feine Details des Objekts als auch den umgebenden Kontext auf einem einzigen Bildschirmausschnitt darzustellen (Abbildung 18 c).

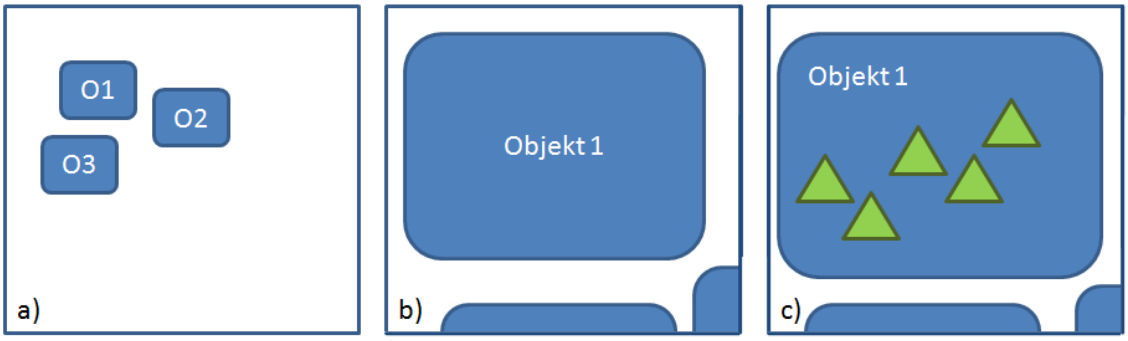

Abbildung 18 - Illustration des semantischen Zooms in drei Schritten

6.2.3 Vorteile und Synergien

Die vorgestellten Visualisierungen bieten einige Vorteile für den vorliegenden Anwendungsfall. Zum Einen bildet die *Pipe/Filter Visualisierung* die modularen Trigger logisch korrekt ab und vermittelt dem Nutzer ein Gefühl für das zugrundeliegende Sys-

tem. Dies führt zu einem klaren mentalen Modell welches die Erlernbarkeit fördert und den Umgang mit Sensoren, Bedingungen und Actions verständlich macht. Zum Anderen kann der Benutzer durch diese Visualisierungen das zugrundeliegende System quasi direkt manipulieren und bekommt simultan Feedback zu seinen Konstruktionen. Außerdem ermöglicht das Pipe/Filter Prinzip eine beliebig komplexe Verkettung von Bedingungen und Actions mit dem Vorteil, dass die Übersicht nicht so schnell verloren geht wie bei anderen Visualisierungsmöglichkeiten. Der Nutzer kann auf einen Blick erkennen welche Elemente wie in Relation zueinander stehen. Die *zoombare Oberfläche* ermöglicht dem Nutzer zudem den nahtlosen Wechsel zwischen dem Überblick über eine Bedingungskette und der Detailansicht einer einzelnen Action. So kann ein einzelnes Element immer mit dem Gedanken an dessen Kontext bearbeitet werden um die mentale Belastung des Nutzers zu verringern.

Die für PocketBee angepasste Implementierung einer visuellen Spezifikationssprache stammt mit freundlicher Unterstützung aus dem Open-Source Projekt Squidy [24] welches sowohl das semantische Zooming als auch die Pipe/Filter Metapher in seinem Interface verwendet (siehe Abbildung 19). Squidy bietet nicht nur die passenden Interaktionstechniken für die zukünftige PocketBee Konfiguration (Kapitel 6.3) sondern stellt auch eine robuste und umfangreiche Codebasis zur Verfügung, die an spezielle Anforderungen angepasst werden kann. Viele der im folgenden Kapitel verwendeten Screenshots stammen aus dem bisher nur leicht modifizierten *Squidy-PocketBee-Prototyp* welcher in Zukunft die Basis der PocketBee-Konfiguration bilden wird.

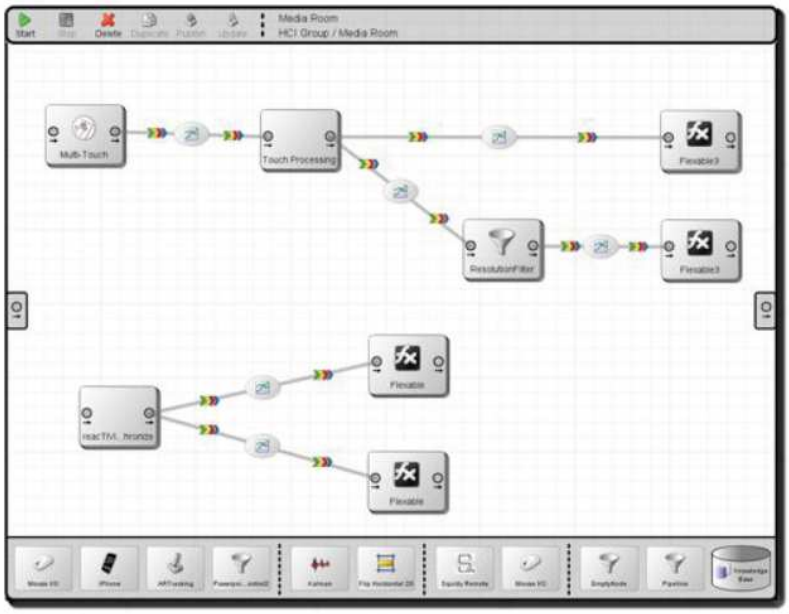

Abbildung 19 - Pipe/Filter Interface von Squidy

6.3 Interfacekonzept

Im Folgenden wird das entwickelte Interfacekonzept vorgestellt, die einzelnen GUI-Elemente beschrieben sowie die verwendeten Interaktionstechniken aufgezeigt. Im Gegensatz zu dem in Kapitel 3 vorgestellten Prototypen fokussiert sich dieses Interface ausschließlich auf die Erstellung von Triggern bzw. Condition-Chains für einen oder mehrere Probanden. Eine solche Condition-Chain besteht aus mehreren miteinander verketteten Elementen, und enthält sowohl Bedingungen, Connectoren als auch Actions (siehe Abbildung 20). Diese Elemente können per Drag and Drop auf den Canvas gezogen und dort miteinander verbunden werden. So kann der Forscher die bereits in Kapitel 5.2 beschriebene Condition-Chain visuell nachbilden. Während durch die Verbindung eine Kette entsteht kann in einzelne Elemente durch einen Doppelklick hinein gezoomt werden um detailliertere Konfigurationseinstellungen vorzunehmen.

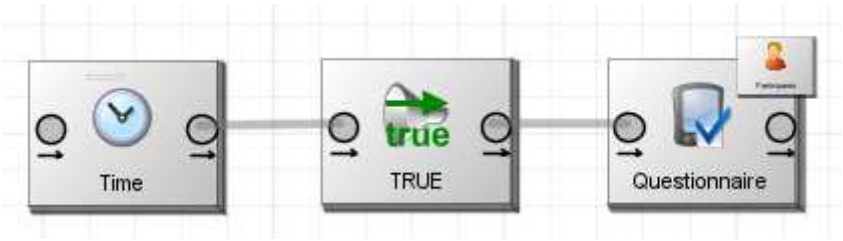

Abbildung 20 - Eine Condition-Chain mit Bedingung, Connector und Aktion

Um den Anforderungen aus Kapitel 6.1 gerecht zu werden versucht das Interfacekonzept (siehe Abbildung 20) die zugrundeliegende Architektur aus Sensoren, Conditions und Actions so abzubilden, dass es dem Nutzer möglichst leicht fällt das System zu verstehen. Jedem Action-Element (5.2.6) geht mindestens ein Condition-Element (5.2.2) voraus, welches über ein Connector-Element (5.2.7) verbunden ist. Ein Connector-Element kann entweder der Wert „true“ oder der Wert „false“ haben, wobei die nachfolgende Aktion nur dann Aktiv wird, wenn das Ergebnis der evaluierten Bedingung dem Wert des Connector Elements entspricht. Dies repräsentiert die Logik des bereits in Kapitel 5 vorgestellten modularen Triggerframeworks und soll das mentale Modell des Forschers unterstützen.

Durch Verkettung können auch mehrere Bedingungen logisch miteinander verknüpft werden. Durch eine so entstandene „Condition-Chain“ wird ermöglicht, dass mehrere Bedingungen wahr sein müssen um eine bestimmte Aktion auszuführen. Außerdem kann der Forscher durch die umschaltbare Funktionalität der Connector-Elemente (true/false) zusätzlich definieren, ob eine einzelne Bedingung der Condition-Chain wahr oder falsch sein muss um eine folgende Aktion auszuführen.

Der Forscher kann während der Erstellung einer solchen Condition-Chain aus einem Pool an Bedingungen und Aktionen wählen die ihm in der Toolbar, einer so genannten Knowledge Base, zur Verfügung stehen. Die dort existierenden Elemente wurden be-

reits implementiert und können vom Forscher per Drag-and-Drop in die bestehende Condition-Chain eingefügt werden. Einzelne Elemente können untereinander verbunden werden indem eine Pipe-Verbindung vom Einen zum Andern gezogen wird.

6.3.1 GUI

Das grafische Interface zur Triggererstellung (siehe Abbildung 21) setzt sich aus drei Elementen zusammen. Im Header (1) finden sich Informationen über die aktuell sichtbare Condition-Chain sowie Möglichkeiten zum Export und Import. Der Canvas (2) bildet den Hauptteil des Interfaces und dient als Arbeitsfläche für den Forscher. Hier werden die Bedingungen und Aktionen zu Condition-Chains verkettet. Die dafür nötigen Elemente (Conditions und Actions) können aus der Toolbar (3) per Drag-and-Drop auf den Canvas gezogen werden.

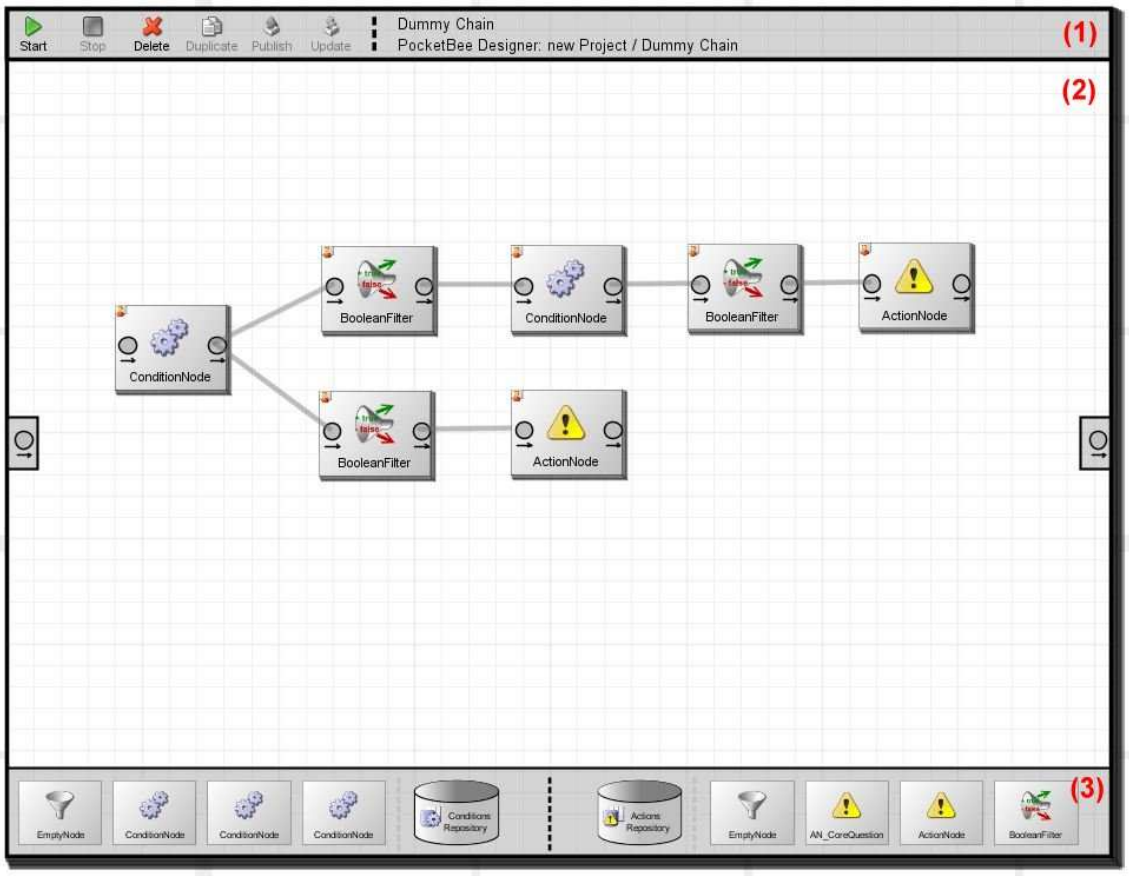

Abbildung 21 - GUI mit Header (1), Canvas (2) und Toolbar (3)

Im Folgenden werden die GUI-Elemente Canvas und Toolbar sowie ihre Funktionalität detailliert beschrieben. Außerdem werden wichtige Aktivitäten wie die Verkettung und das Verschachteln von Objekten vorgestellt.

6.3.1.1 Canvas

Der Canvas (siehe Abbildung 21 Element 2) stellt die Arbeitsoberfläche dar, auf der alle Objekte angeordnet und miteinander verbunden werden. Hier können Bedingungen und Actions abgelegt werden die per Drag & Drop aus der Toolbar gezogen wurden. Die Elemente auf dem Canvas müssen alle Teile einer einzelnen „Condition-Chain“ sein um als valide anerkannt zu werden (siehe Kapitel 6.3.2.7). Das Konzept sieht vor, dass alle Elemente auf dem Canvas durch einen speziellen Layout-Algorithmus angeordnet werden. Dies bedeutet, dass beim Hinzufügen von neuen Elementen das bestehende Layout angepasst wird. Ziel ist es, zu jeder Zeit ein ausgeglichenes und übersichtliches Layout zu bewahren. Da der Canvas eine Condition-Chain beinhaltet die auf einer höheren Ebene im Projekt als eigener modularer Objektknoten gesehen wird besteht die Möglichkeit die gesamte Condition-Chain umzubenennen.

6.3.1.2 Toolbar

Die Toolbar (siehe Abbildung 21 Element 3) enthält sowohl die meistbenutzten Actions und Bedingungen als auch die so genannte Knowledge-Base der jeweiligen Elemente. Während Elemente die direkt auf der Toolbar verfügbar sind mit einem einfachen Drag & Drop auf den Canvas gezogen werden können muss der Nutzer für weniger frequentierte Objekte in die Knowledge-Base hineinzoomen und ein Objekt von dort aus auf den Canvas ziehen. Während die linke Seite der Toolbar alle Bedingungen beherbergt, enthält die rechte Seite alle Actions.

6.3.1.3 Logik-Objekte

Es existieren verschiedene Objekte die von der Toolbar auf den Canvas gezogen werden können und somit Teile der Condition-Chain sind. Im Folgenden werden die Elemente Conditions, Actions und Connectoren sowie deren Funktion beschrieben.

6.3.1.3.1 Condition

Ein Bedingungsobjekt repräsentiert sowohl einen bestimmten Sensor (siehe Kapitel 5.2.1) als auch die Bedingungen die die Werte des Sensors erfüllen müssen, um die gesamte Bedingung zu bewahrheiten. Mehrere Bedingungsobjekte können beliebig hintereinander verkettet werden, gehen jedoch immer einer Action voraus, die erst aktiviert wird sobald alle Bedingungen erfüllt sind (siehe Kapitel 5.2). Ein Bedingungsobjekt repräsentiert immer nur einen einzelnen Sensor. Da jedoch mehrere Bedingungsimplementierungen zu einem einzelnen Sensor existieren können, ermöglicht es das Condition-Objekt in der Detailansicht zwischen diesen Implementierungen zu wechseln. Der Wechsel zwischen Implementierungen wirkt sich sowohl auf die verfügbaren Parameter als auch auf die

interne Evaluierung der Bedingung aus (siehe Abbildung 22). Ein Forscher kann in dem modularen Trigger-Framework zu jeder Zeit eine eigene Implementierung hinzufügen.

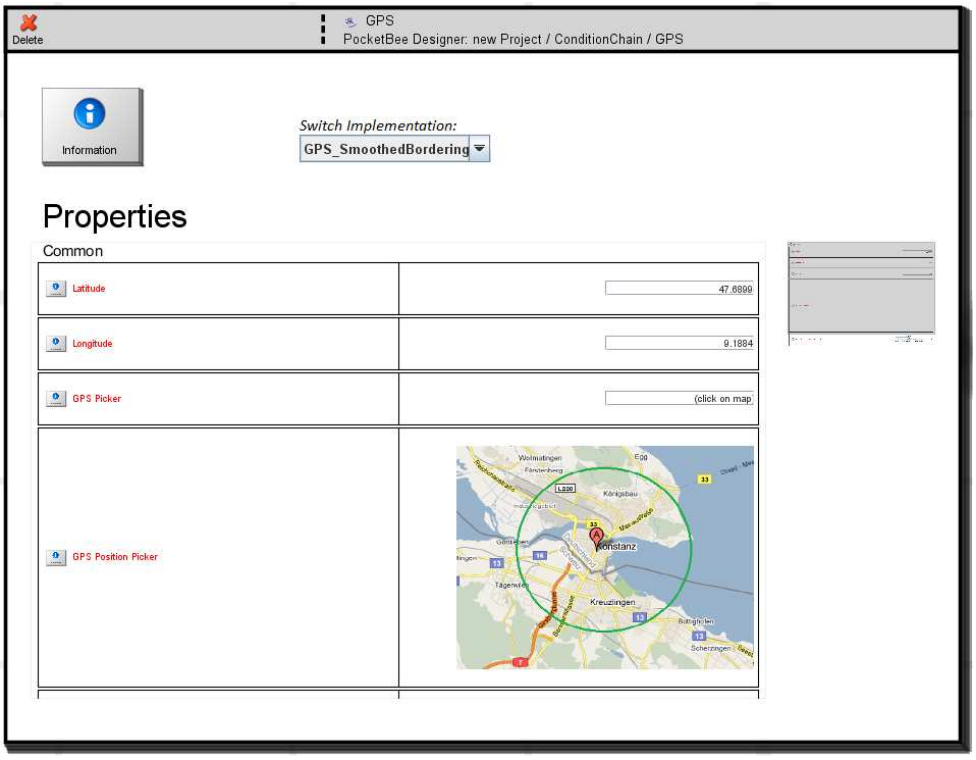

Abbildung 22 - Detailansicht einer GPS-Bedingung

6.3.1.3.2 Action

Ein Actionobjekt steht für eine Systemaction die ausgeführt wird, sobald alle vorhergehenden Bedingungen der Condition-Chain erfüllt sind. Neben dem Fragebogen, der Kernfrage und einer Aufgabe existieren viele weitere Actions. Zusätzliche Actions können vom Forscher selbst implementiert und in das System eingebunden werden. Die Parameter in der Detailansicht hängen von der jeweiligen Implementierung und dem Anwendungsfall ab. Ein festes Unterelement einer Action ist jedoch die Probandenauswahl. Dort kann der Nutzer festlegen welche Probanden von dieser Action betroffen sind, sobald sie aktiviert wird. Eine Action darf nie ohne eine vorhergehende Bedingung gespeichert und an die Probanden verteilt werden (siehe Kapitel 6.3.2.7). Für dauerhaft benötigte Actions wie z.B. einer Kernfrage wird eine zeitliche Bedingung vorangestellt die sich über den gesamten Zeitraum der Studie erstreckt. Ein weiterer Parameter der bei jeder Action eingestellt werden kann ist der in Kapitel 5.2.6 vorgestellte ActionType. Dieser definiert ob die Action einmalig, wiederholend oder dauerhaft aktiviert werden kann.

6.3.1.3.3 Connector

Connector-Elemente befinden sich immer zwischen einer Bedingung und einer Action. Sie dienen im System als Verzweigungspunkte- Durch sie kann der Forscher entscheiden mit welchem Ergebnis einer Bedingung, *erfüllt* (true) oder *nicht erfüllt* (false), er weiter arbeiten will. Beispielsweise kann er durch die Verzweigung nach einer GPS-Bedingung verschiedene Bedingungsketten weiterführen (siehe Abbildung 23). Wenn die GPS-Bedingung erfüllt ist wird Kernfrage A angezeigt, wenn sie nicht erfüllt ist bekommt der Nutzer die Kernfrage B präsentiert. So stellt er sicher dass zwar zu jeder Zeit eine Kernfrage angezeigt wird, je nach Position des Probanden wird diese jedoch variiert.

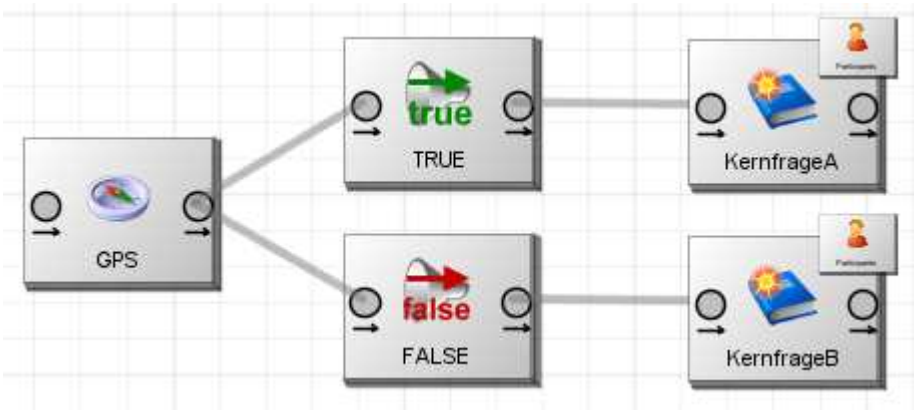

Abbildung 23 - Verzweigung einer GPS-Bedingung

Connector-Elemente werden vom System automatisch eingefügt sobald der Nutzer eine Bedingung mit einer Action verbindet. So wird sichergestellt, dass die logische Struktur des Systems beibehalten wird. Im Normalfall wird automatisch ein Connector-Element „TRUE“ erstellt. Der Nutzer kann diese Eigenschaft jedoch in der Detailansicht des Connector-Elements verändern. Über den Validierungsmechanismus (siehe Kapitel 6.3.2.7) wird sichergestellt dass eine Bedingung jeweils nur einen TRUE und einen FALSE Ausgang besitzt. An die Connector-Elemente selbst können jedoch beliebig viele weitere Elemente angehängt werden um beispielsweise mehrere Actions gleichzeitig auszulösen (siehe Kapitel 6.3.2.3).

6.3.2 Interaktion

Neben der visuellen Darstellung spielen natürlich auch die Interaktionsmöglichkeiten des Nutzers eine wichtige Rolle. Im folgenden Kapitel werden alle Interaktionstechniken vorgestellt die ein Nutzer am System anwenden kann. Neben den Grundfunktionen Drag & Drop, Verkettung, Verzweigung und Verschachtelung von Elementen bietet das Interface auch einige Hilfestellungen beim Erstellen einer Condition-Chain an. Es werden dem Nutzer z.B. so genannte Drop-Targets angeboten auf welchen er sein aktuell

ausgewähltes Objekt ablegen kann. Dies und die dynamische Validierung der gesamten Kette erhöhen die Lernförderlichkeit und Fehlertoleranz des Systems.

6.3.2.1 Drag and Drop

Die altbekannte Interaktionstechnik Drag & Drop wird auch in diesem System angewendet. Der Nutzer kann bestimmte Objekte (Bedingungen und Actions aus der Toolbar) mit einem Klick auswählen und diese durch gedrückt halten der Maustaste und simultanes Bewegen der Maus auf den Canvas ziehen. Das Objekt bewegt sich während dieser Bewegung unterhalb des Mauszeigers mit. Sobald das Objekt an der gewünschten Stelle ist, lässt der Nutzer die Maustaste los und platziert das Objekt somit an der aktuellen Stelle. Eine Besonderheit an diesem System ist, dass wenn der Nutzer ein Objekt innerhalb der Knowledge-Base selektiert und bewegt zoomt die Ansicht automatisch aus der Knowledge-Base heraus und zeigt wieder die gesamte Condition-Chain. So kann der Nutzer auch Elemente aus innerhalb der Knowledge-Base per Drag & Drop in die Condition-Chain einfügen.

6.3.2.2 Verkettung von Objekten

Alle Elemente auf dem Canvas müssen miteinander verkettet sein. Der Normalfall für die Reihenfolge ist: Condition-Connector-Action. Wie jedoch bereits in Kapitel 5.2 erwähnt können beliebig viele Bedingungen vor eine Action geschalten werden. Die anhängende Action wird nur aktiv wenn alle vorhergehenden Bedingungen gleichzeitig erfüllt sind. Wenn der Nutzer eine Verbindung zwischen zwei Elementen zieht generiert das System automatisch ein Connector-Element mit dem default-Wert „TRUE“ welches sich zwischen den beiden Elementen anordnet (siehe Abbildung 24). Durch diese Automatische Erstellung von Connector-Elementen wird zu jeder Zeit die Konsistenz der Condition-Chain aufrecht erhalten. Als weitere Hilfestellung werden dem Nutzer beim Einfügen von Elementen so genannte Drop-Targets eingeblendet. Diese Drop-Targets zeigen an, an welchen Stellen das aktuell selektierte Element platziert werden könnte (siehe Abbildung 24). Drop-Targets werden in Kapitel 6.3.2.5 detailliert beschrieben.

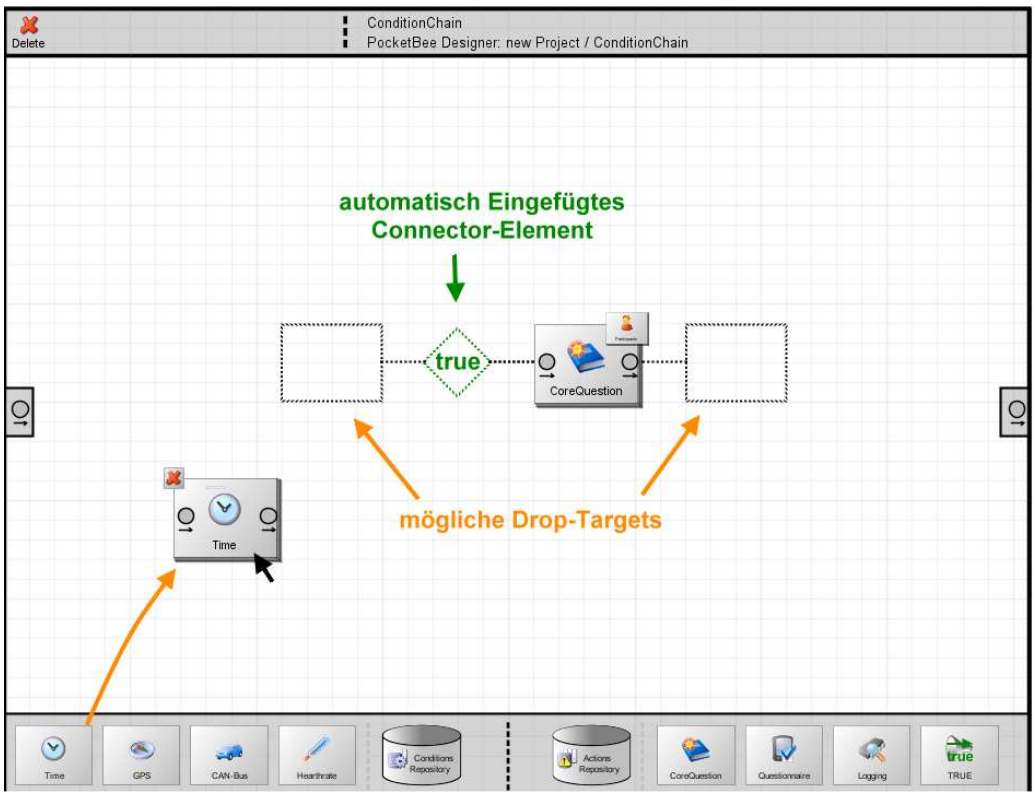

Abbildung 24 - Verkettung von Objekten mit Connector-Element und Drop-Targets

6.3.2.3 Verzweigung von Bedingungsketten

In einer Condition-Chain können Bedingungen nicht nur beliebig oft miteinander verkettet sonder auch anhand ihres Ergebnis in mehrere Zweige abgespalten werden. Um dies zu realisieren werden die bereits in Kapitel 6.3.1.3.3 vorgestellten Connector-Elemente benötigt. Die Connector-Elemente befinden sich immer direkt nach einer Bedingung und können zwei Werte annehmen: TRUE und FALSE. Die Werte TRUE und FALSE beziehen sich jeweils auf das Ergebnis der vorhergehenden Bedingung und sorgen dafür, dass nur jeweils der Zweig weiter evaluiert werden kann mit dessen Wert das Ergebnis der Bedingung übereinstimmt. D.h. im Beispiel von Abbildung 25 entscheidet sich bei der ersten Bedingung „Time“ welcher Pfad eingeschlagen werden kann. Sollte die Bedingung „Time“ erfüllt sein (true) muss zusätzlich die Bedingung „GPS“ erfüllt sein (true) um den Fragebogen „Questionnaire“ zu aktivieren. Wenn aber die Bedingung „Time“ nicht erfüllt ist (false) wird der alternative Pfad evaluiert und es werden stattdessen die beiden Actions „CoreQuestion“ und „Logging“ aktiviert. In diesem Szenario würde dem Probanden also kontinuierlich die Kernfrage „CoreQuestion“ präsentiert während simultan ein Sensor geloggt wird oder, sollte der gewünschte Zeitpunkt am richtigen Ort eingetreten sein, der Fragebogen „Questionnaire“ vorgelegt.

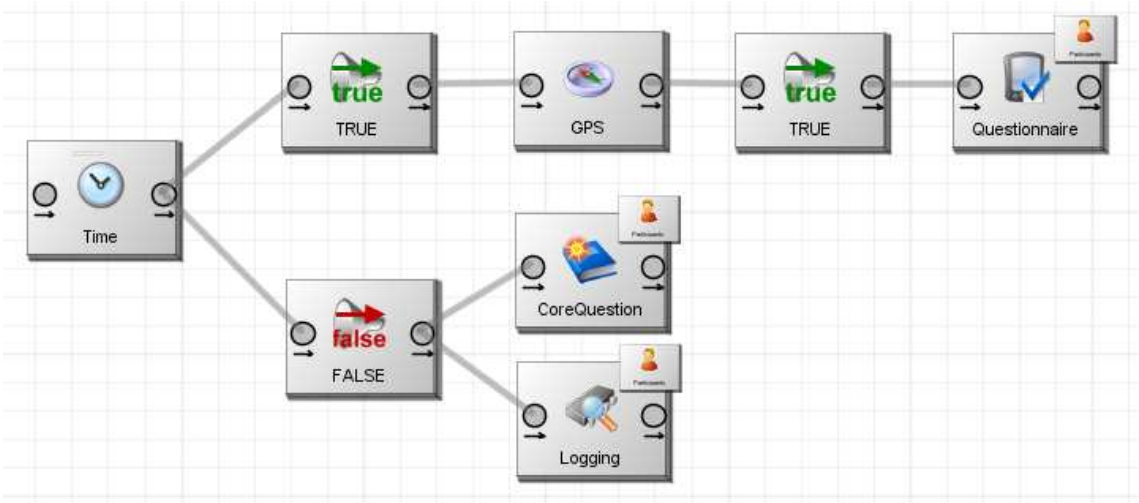

Abbildung 25 - Verzweigungen einer Bedingungskette

Während immer nur eine Verbindung in ein Connector-Element hineinführen kann ist es möglich den Ausgang eines Connector-Elements mit mehreren Elemente zu verbinden. So lässt sich z.B. realisieren, dass mehrere Actions ausgeführt werden nachdem eine einzelne Bedingung erfüllt ist (siehe ebenfalls Abbildung 25).

6.3.2.4 Verschachteln von Objekten

Alle in Kapitel 6.3.1.3 beschriebenen Objekte sind nicht nur Knoten in einer Bedingungskette, sondern gleichzeitig auch die abstrakte Repräsentation eines Konfigurationselements. Durch die Technik des semantischen Zooms werden diese Elemente auf verschiedenen Zoomstufen mit anderem Inhalt angezeigt. Beispielsweise enthält der Bedingungsknoten „Time“ in seinem Innern mehrere verschiedene Implementierungen die den Sensor „Zeit“ anhand unterschiedlicher Parameter vergleichen können. Um eine dieser Implementierungen auszuwählen kann der Nutzer durch einen Doppelklick auf eine Zeit-Bedingungsknoten a) in die Detailansicht b) dieses Knotens wechseln (siehe Abbildung 26). Dort kann er nun nicht nur zwischen den verschiedenen Implementierungen wechseln, sondern auch die passenden Parameter für jede dieser Implementierungen einstellen. Die Konfiguration eines Bedingungsknotens passiert also direkt im Kontext seiner Condition-Chain. Die Properties der Tabelle in Abbildung 26 b) können vom Ersteller der Implementierung definiert werden. Dort kann der Forscher alle nötigen Parameter für die Bedingung einstellen. Hier können beliebige Datentypen von primitiven String und Integer bis hin zu selbst entwickelten komplexen Datentypen wie z.B. eine GPS-Position mit Radius alles Mögliche definiert werden.

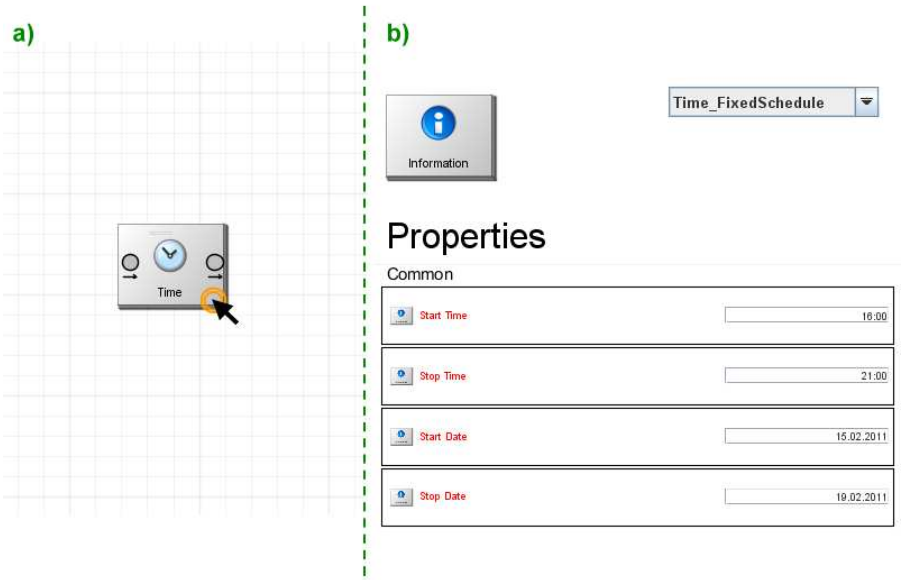

Abbildung 26 - Zeit-Bedingungsknoten a) mit Detailansicht b)

Ein weiteres Beispiel der Verschachtelung sind die Probanden, welche zu jeder Action zugeordnet werden können. Diese Zuordnung entscheidet beim aktivieren wer die Action präsentiert bekommt und wer nicht (siehe Kapitel 5.2.6). Die Probandenauswahl muss dementsprechend für jede Action verfügbar sein. Durch die Technik der Verschachtelung wird die Probandenauswahl jedoch einfach als Sub-Knoten jeder Action eingebunden (siehe Abbildung 27). So steht sie jederzeit zur Verfügung, verschwendet jedoch keinen wertvollen Interfaceplatz. Durch hinein zoomen (Doppelklick) kann der Nutzer hier ebenfalls die Detailansicht der Probandenauswahl anzeigen und spezifizieren welche Probanden diese Action erhalten sollen und welche nicht.

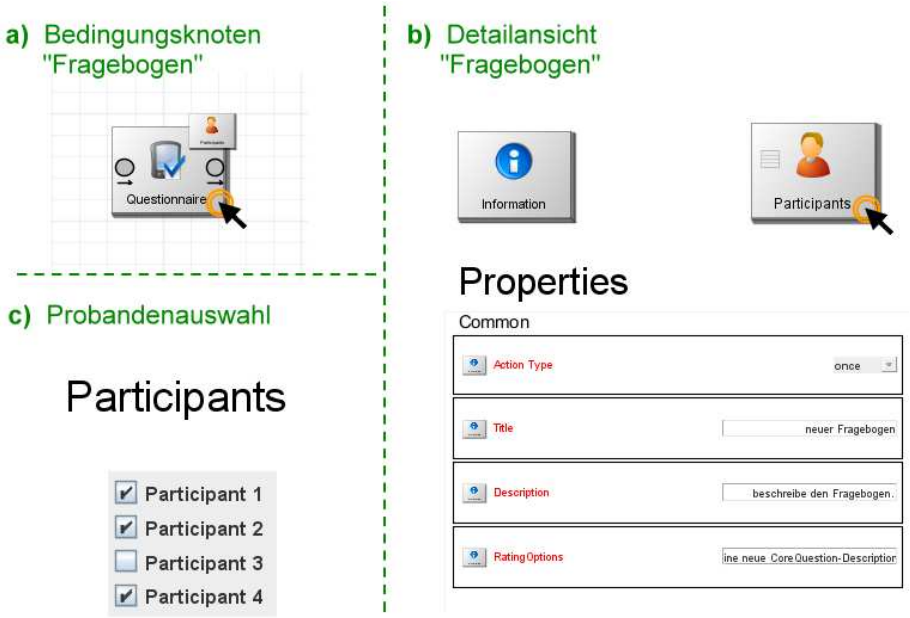

Abbildung 27 - Fragebogen-Action als Knoten a) im Detail b) mit Probandenauswahl c)

Im Falle der Fragebogen-Action geht das Verschachtelungsprinzip sogar soweit innerhalb des Fragebogen-Knotens eine weitere, der ursprünglichen Canvas-GUI ähnliche, Oberfläche zu präsentieren. Dort kann der Nutzer nach demselben Interaktionsprinzip die Logik eines Fragebogens definieren. Der Forscher muss somit um einen Fragebogen zu erstellen weder das Programm wechseln, noch eine neue Definitionsmethode lernen. In Abbildung 28 findet sich eine beispielhafte Implementierung eines Fragebogens mit Gabelfragen und verschiedenen Fragetypen.

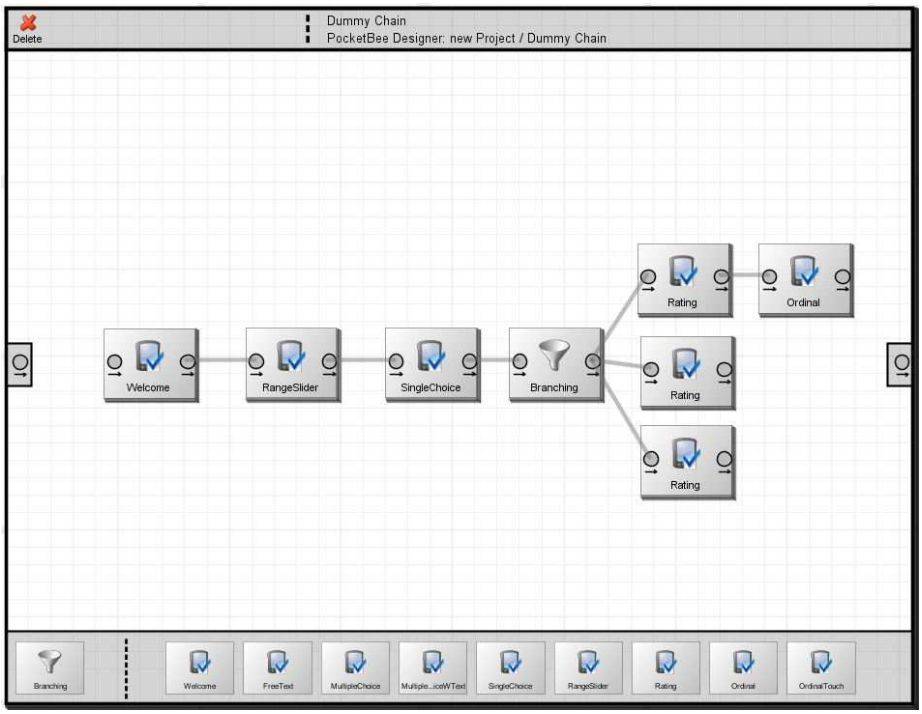

Abbildung 28 - Spezifikation eines Fragebogens mit Gabelfrage

6.3.2.5 Drop Targets

Wie bereits in Kapitel 6.3.2.2 kurz angeschnitten, werden zur Unterstützung der Nutzer sogenannte Drop-Targets auf dem Interface eingeblendet wenn ein Objekt von der Toolbar auf den Canvas gezogen wird. Diese Drop-Targets zeigen dem Nutzer alle möglichen Positionen an, auf denen er sein aktuelles Objekt ablegen kann. Durch diese klare Benutzerführung hat der Nutzer weniger mentale Belastung und kann sich so komplett auf die Logik der Bedingungskette konzentrieren. Ein weiterer Vorteil der Drop-Targets ist die automatische Verknüpfung mit bereits platzierten Elementen auf dem Canvas. Wie bereits in Abbildung 24 des Kapitels 6.3.2.2 zu sehen werden nur Positionen vorgeschlagen die im Kontext der Bedingungskette einen Sinn ergeben. Dementsprechend werden beim Einfügen des Elements dann auch direkt die nötigen Connector-Elemente und Verbindungen eingefügt um die Kette zu vervollständigen. Die möglichen Drop-Targets werden auf dem Canvas als gestrichelte, etwas schwächere Linien gezeichnet. Sobald sich die Maus mit dem aktuell selektierte Objekt in die Nähe eines Drop-Targets bewegt greift ein sogenannter „Snap-To-Grid“ Mechanismus welcher dafür sorgt, dass das neue Objekt direkt über dem Drop-Target platziert wird. Das gesamte Layout der Bedingungskette wird automatisch an die neue Situation angepasst indem angenommen wird das neue Element befindet sich an der richtigen Position und würde hier abgelegt.

Wie in Abbildung 29 zu sehen lässt sich dieser Mechanismus auch bei einer komplexen Bedingungskette anwenden: Während der Nutzer ein „CAN-Bus“ Bedingungselement von der Toolbar zieht erscheinen in der Condition-Chain die möglichen Drop-Targets (orange gestrichelt). Diese geben die möglichen Ablagepositionen für das Objekt an. Wenn sich nun der Nutzer mit dem Objekt einem dieser Drop-Targets annähert wächst das Target auf die benötigte Größe an und zieht das Objekt des Nutzers „magnetisch“ an. Zusätzlich wird zu dem magnetischen Drop-Target das Connector-Element angedeutet welches nach dem Einfügen automatisch erstellt würde (siehe Abbildung 30).

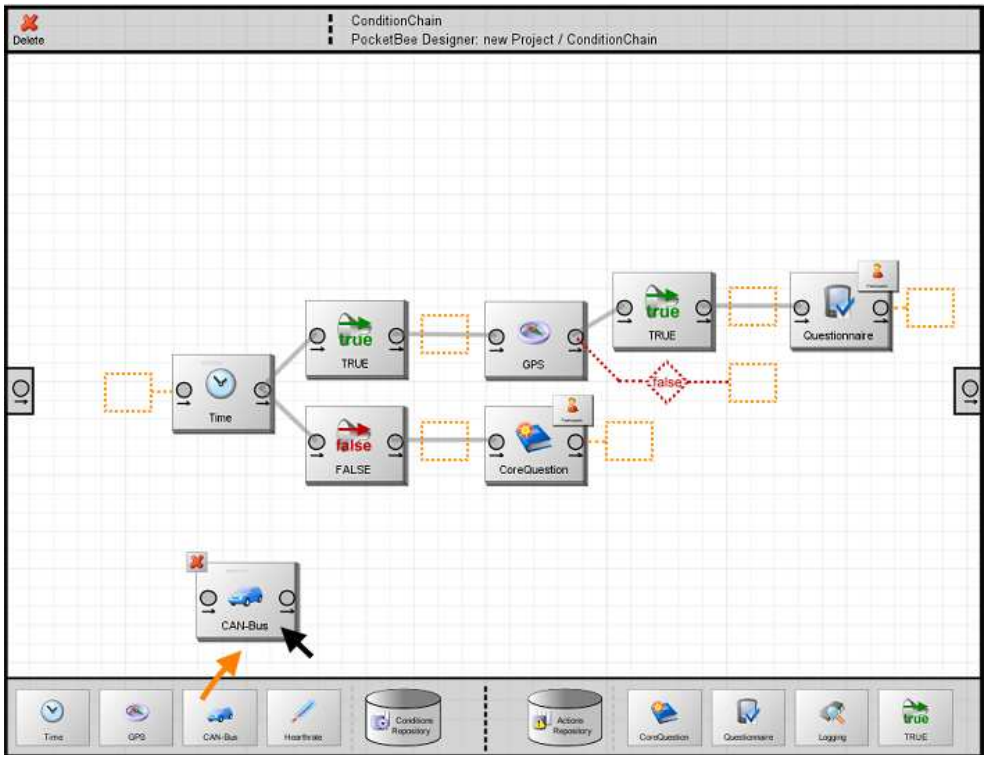

Abbildung 29 - Drop-Targets in einer Condition-Chain

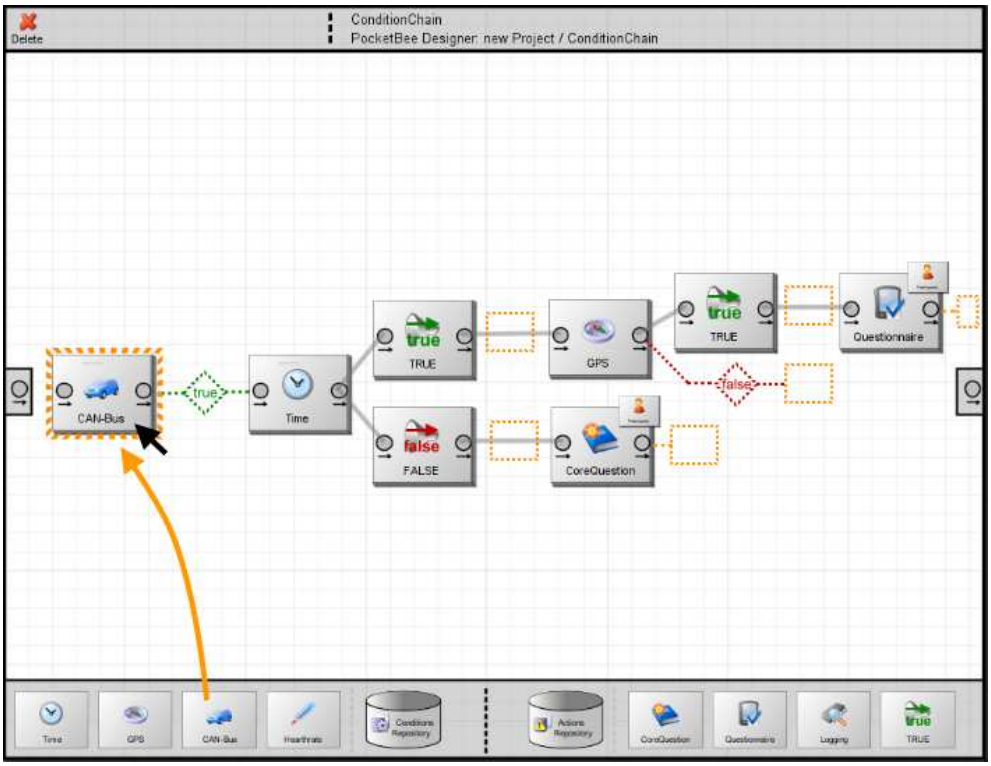

Abbildung 30 - Drop-Targets in einer Condition-Chain mit Snap-to-Grid Effekt

6.3.2.6 Automatisches Layout des Canvas

Wie bereits im vorherigen Kapitel erwähnt werden die Elemente einer Condition-Chain automatisch auf dem Canvas angeordnet. Dies hat den Vorteil dass der Platz ideal ausgenutzt wird und eine übersichtliche Struktur entsteht. Außerdem ist ein automatisch generiertes Layout für das automatische Einfügen von Elementen (Connectoren) essentiell. Der hier verwendete Layout-Algorithmus (Balancing Visual Tree) sorgt dafür, dass alle Elemente die zu einer Condition-Chain gehören in passendem relativem Abstand zueinander angeordnet werden und ihre Größe gegebenenfalls angepasst wird. Derselbe Algorithmus wird verwendet wenn sich Elemente der Kette dynamisch verändern (siehe Kapitel 6.3.2.5). Unabhängig von dem automatischen Layout-Algorithmus ist es möglich einzelne Objekte außerhalb einer Condition-Chain abzulegen um sie später mit der Kette zu verbinden. Diese Elemente werden nicht vom Layout-Algorithmus beeinflusst.

6.3.2.7 Validierung

Um die Fehlertoleranz zu erhöhen und dem Nutzer direktes Feedback zu seiner Konstruktion zu geben werden die Elemente und die Condition-Chain nach jeder Änderung evaluiert. Die Regeln für diese Evaluierung lauten wie folgt:

- Auf einem Canvas darf jeweils nur eine vollständige Condition-Chain existieren.
- Alle Elemente müssen Teil einer Condition-Chain sein. (siehe Abbildung 31)
- Endlosschleifen und Kreisschlüsse müssen aufgelöst werden.
- Jede Bedingung besitzt jeweils nur einen TRUE und einen FALSE „Ausgang“.

Regeln die Knotenspezifisch sind werden im jeweiligen Knoten selbst definiert.

Sollten in der Condition-Chain oder innerhalb eines einzelnen Knotens Fehler auftreten werden die fehlerhaften Elemente mit einer roten Aura markiert (siehe Abbildung 31). So sieht der Nutzer auf einen Blick welche Elemente verändert werden müssen. Außerdem lässt sich die gesamte Condition-Chain nicht im fehlerhaften Zustand aktivieren.

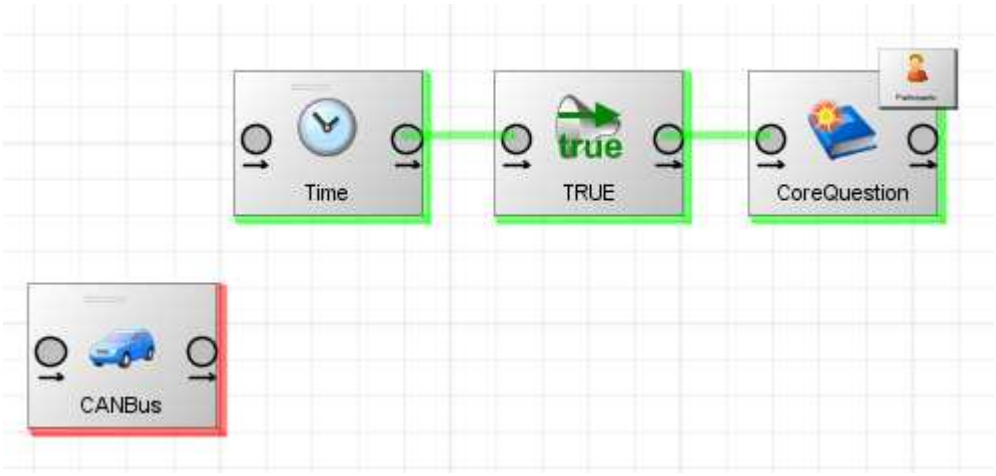

Abbildung 31 - farbliche Hinweise auf Fehler bei der Validierung

7 Ausblick

Auch nach Abschluss dieser Arbeit bleiben einige Punkte offen, die im Verlauf der Entwicklung und durch die Studien zu Tage gefördert wurden. Hier werden diese Anknüpfungspunkte vorgestellt und entsprechende Lösungsvorschläge aufgezeigt.

7.1 Analyse der Daten

Während der Durchführung der Pilotstudien bestätigte sich die Feststellung von Khan et. al. [19], dass ein automatisiertes aber konfigurierbares Tool zur Informationsvisualisierung und Datenanalyse ein essentieller Bestandteil eines solchen Systems sein muss. Sowohl mit der Tagebuchmethode, beim Logging als auch durch die experience sampling method wird von den Probanden eine Unmenge an Daten produziert. All diese Videos, Fotos, Zeichnungen, Texte und Sprachaufnahmen sowie Sensorlogs und ausgefüllte Fragebögen müssen von Forschern analysiert und interpretiert werden. Dass diese Aufgabe alles andere als trivial ist zeigte auch das Interview mit einer Forscherin (siehe Kapitel 4.3) worin deutlich wird, welche Probleme bei der Analyse solcher Daten auftreten. Der Forscher muss manuell all diese Mediendaten durchgehen und transkribieren, interpretieren oder Statistiken von Fragebögen selbst erstellen. Diese Arbeit ist sehr zeitintensiv und sollte daher vom System unterstützt werden. Demnach ist es von höchster Relevanz für die praktische Anwendbarkeit des PocketBee Systems auch für die Analyseaufgaben des Forschers eine unterstützende Lösung zu bieten. Da dieses Problem bereits länger existiert und sich auch andere Fachbereiche damit beschäftigen beinhaltet die hierzu vorgeschlagene Herangehensweise eine ausgiebige Analyse von existierenden Analyseverfahren und Werkzeugen aus verschiedenen fachlichen Kontexten (z.B. Psychologie, Soziologie und Verhaltensforschung). Im Falle einer geplanten Implementierung der Analysefunktionalität sollten in jedem Falle Experten aus dem Bereich der Informationsvisualisierung sowie anwendende Personen aus den bereits erwähnten Fachbereichen Psychologie und Soziologie zu Rate gezogen werden. Außerdem ist es fraglich, ob eine direkte Zusammenführung in ein Gesamtsystem Entwicklungstechnisch vertretbar ist. Da die Komplexität des bisherigen Systems bereits sehr hoch ist sollte diese Option genauestens geprüft werden. Eine mögliche Alternative mit zwei unterschiedlichen Systemen könnte durch klar definierte Schnittstellen die Gesamtkomplexität überschaubar halten.

7.2 Funktionsumfang

Um das PocketBee System auch in anderen Fachbereichen und mit unterschiedlichen Fragestellungen verwenden zu können müssen noch einige Erweiterungen oder Fachspezifische Funktionen implementiert werden. Inspiration für diese Erweiterungen ergeben sich idealerweise durch direkte Kooperationen und Pilotstudien in den gewünschten Gebieten (Psychologie, Sportwissenschaften, usw.) oder durch die Analyse bereits durchgeführter ESM und Tagebuchstudien in den entsprechenden Gebieten.

Der Funktionsumfang des Systems könnte in jedem der drei Hauptbereiche erweitert werden. Das ControlCenter birgt reichlich Verbesserungspotential für Hilfestellungen die einem Forscher die Durchführung und Analyse einer Studie erleichtern (z.B. automatische Sprachtranskription oder Benachrichtigung in vordefinierten Situationen). Aus Methodensicht kann das gesamte System mit weiteren Sensoren (Hertzrate, CAN-Bus, usw.) und Actions (Moodmaps, Reaktionstests, Erinnerungstests, usw.) ausgestattet werden damit Studiendesigns aus den unterschiedlichen Fachbereichen möglich werden. Aus der Sicht der Probanden gibt es sicher ebenfalls noch Verbesserungspotential das über die pure Akkulaufzeit der Geräte hinausgeht.

7.3 Verfügbarkeit

Aus der regen Nachfrage nach dem PocketBee System lässt sich schließen, dass auch andere Forscher Bedarf an einem solchen Werkzeug haben. Im Sinne der öffentlich geförderten Forschung sollte daher weiterhin eine Veröffentlichung unter einer OpenSource Lizenz verfolgt werden. Natürlich muss bei Aktivitäten in dieser Richtung auf die Wettbewerbssituation des Kooperationspartners eingegangen werden, jedoch sollte eine klare Trennung zwischen Industrieller Dienstleistung und öffentlicher Forschung angestrebt werden. Im Bereich Open Source existieren die unterschiedlichsten Kombinationsmöglichkeiten welche auch für diesen Fall angepasst werden können.

Die Vorteile einer Veröffentlichung liegen auf der Hand. Es könnten dadurch nicht nur eine breitere Wahrnehmung und daraus folgende interessante Anwendungsfälle generiert werden, diese Anwendungsfälle und Anpassungen für andere Fachbereiche führen auch zwangsläufig zu einer Erweiterung der Funktionalität und damit zu einer Wertsteigerung des gesamten Systems.

8 Zusammenfassung und Fazit

In Zuge dieser Masterarbeit wurde eine modulare Systemarchitektur vorgestellt, die es einem Forscher ermöglicht beliebige Studien aus einer Kombination von ESM, Tagebuchstudien und Logging einfach planen und durchführen zu können. Zusätzlich wurde ein Interfacekonzept vorgestellt, welches mit Hilfe des Pipe/Filter Prinzips und semantischem Zoom die grafische Erstellung und Konfiguration einer Studie ermöglicht. Das Interfacekonzept unterstützt die Modularität der zugrundeliegenden Architektur und soll auch für Forscher einfach erlernbar sein die wenig technisches Fachwissen haben.

Im Kapitel 1 wurde in das Thema „Datensammlung im mobilen Kontext“ eingeführt und gezeigt, dass verschiedene Erhebungsmethoden wie ESM [14], Tagebuch [2] und Sensor-Logging [16] in der Praxis einen hohen Überschneidungsgrad haben und oft gemeinsam in Studien eingesetzt werden. Die Motivation für mobile Datensammlung ergibt sich aus der ubiquitären und immer mobiler werdenden Produktnutzung der Konsumenten und der Tatsache, dass diese Produkte nur sinnvoll in ihrem natürlichen Nutzungskontext evaluiert werden können. Bereits existierende Werkzeuge sind meist nur auf einzelne Methoden und Anwendungsszenarien spezialisiert und unterstützen weder eine Methodenkombination noch die technischen Möglichkeiten aktueller Smartphones. Aus diesem Grund zielt das PocketBee System darauf, es dem Forscher zu ermöglichen beliebige Studiendesigns aus Kombinationen von ESM, Tagebuch- und Loggingmethode einfach planen, durchführen und analysieren zu können.

In Kapitel 0 wurden die Anforderungen an ein solches System zusammengefasst. Diese Anforderungen stammen sowohl aus einer durchgeführten State-of-the-Art Analyse von existierenden ESM- und Tagebuchwerkzeugen [26], aus den praktischen Erfahrungen in Pilotstudien (Kapitel 4.4) als auch aus diversen wissenschaftlichen Vorarbeiten (z.B. von Khan et. al. [19]). Einige dieser Anforderungen sind zum Beispiel die „Ermöglichung der Datenerhebung im Kontext der tatsächlichen Anwendung“, die „Verschmelzung der Kategorien ESM- und Tagebuchstudie“ als auch die „erleichterte Planung, Durchführung und Analyse solcher Studien“. Die State-of-the-Art Analyse zeigte zudem, dass nur wenige Werkzeuge existieren die eine Kombination aus ESM, Tagebuch und Logging zulassen, und die meisten der analysierten Werkzeuge in ihrer Funktion sehr speziell auf den gegebenen Anwendungsfall zugeschnitten wurden.

Das Kapitel 3 beschreibt in Kürze die bereits im Projektbericht [25] ausführlich dokumentierte bisherige Pilotversion des PocketBee Systems. Zu frühen Evaluationszwecken wurde diese Pilotversion des Systems entwickelt welche zum Einen auf dem Android-Betriebssystem für Smartphones basiert und zum Anderen durch einen Server unterstützt wird der den zentralen Sammelpunkt für alle dokumentierten Daten bildet. Dieser Server verfügt zusätzlich über ein webbasiertes Benutzerinterface über das der Forscher eine komplette Tagebuch- und ESM-Studie planen und durchführen kann. Zur Auswer-

tung steht dem Forscher dort eine tabellarische Auflistung der übertragenen Mediendateien und ausgefüllten Fragebögen zur Verfügung. Mit dieser Pilotversion des Systems kann der Forscher sowohl zeitlich bedingte Fragebögen und Aufgaben an die Probanden verteilen, als auch Forschungsfragen im Sinne eines human-recognition-based[6] Designs auf dem Gerät einblenden.

Kapitel 4 beschreib die bisherige Praxiserfahrung mit dem aktuellen Pilotsystem anhand zweier damit durchgeführter Studien und einer qualitativen Einschätzung einer Nutzerin/Forscherin. Die erste der beiden Studien „Behindertenfreundlichkeit in der UNI“ wurde in der Universität Konstanz mit 10 Studenten durchgeführt und war vor allem ein erster Praxistest um die Funktionsfähigkeit des Gesamtsystems zu erproben. Dabei hatten die Teilnehmer, welche die PocketBee Anwendung auf GalaxyS Smartphones bereitgestellt bekamen, die Aufgabe eine Woche lang jegliche Rollstuhl-Barrieren in der UNI zu dokumentieren. Als weiterer Praxistest zählte die Studie „Wellness im Auto“, welche von unserem Kooperationspartner Daimler mit eigenen Kunden durchgeführt wurde. Dort wurden ebenfalls 10 Teilnehmer eine Woche lang mit einem Motorola Milestone ausgestattet und gebeten jegliche Gedanken über Komfort und Wohlfühlen in ihrem Fahrzeug zu dokumentieren. In beiden Studien wurden die Teilnehmer mit regelmäßigen Fragebögen konfrontiert die sie direkt auf ihrem Smartphone ausfüllen konnten. Trotz einigen technischen Problemen konnten beide Studien erfolgreich abgeschlossen werden und produzierten viele Informationen die sowohl zur Verbesserung des PocketBee Systems als auch für den eigentlichen Studienzweck dienlich waren. Abschließend wurden in Kapitel 4.4 sowohl die Erkenntnisse und Verbesserungsvorschläge der verantwortlichen Forscherin als auch die technischen Probleme zusammengefasst.

Sowohl die State-of-the-Art Analyse und die technischen Erfahrungen mit der Pilotversion, die durchgeführten Studien als auch die Einschätzung der verantwortlichen Forscherin führten zur Motivation eines neuen Trigger-Frameworks welches in Kapitel 5 vorgestellt wurde. Einer der Hauptgründe war die nötige die Kombination von Methoden (Tagebuch, ESM und Logging) in einer einzelnen Studie mit einem einzigen Werkzeug. Die daraus entstandene Anforderung an beliebige Kombination von Sensoren, Bedingungen und Aktionen konnte mit dem bisherigen System nicht realisiert werden. Außerdem sollte die Fehleranfälligkeit durch eine klare Definition der Schnittstellen reduziert werden.

Das neue Framework (siehe Abbildung 32) setzt sich aus drei Hauptelementen zusammen: Das Condition-Bundle ist eine Bedingung mit zugehörigem Sensor, das Action-Bundle repräsentiert eine Aktivität die gestartet wird wenn alle vorhergehenden Bedin-

[6] Studiendesign bei dem die Probanden selbst den Zeitpunkt und das Objekt der Dokumentation bestimmen. Beispielsweise eine Tagebuchstudie mit der Frage „Dokumentiere deine Essgewohnheiten“

gungen erfüllt sind und zuletzt das Connector-Element welches die Verbindung zwischen zwei Bundles herstellt. Durch die Kombination und Verkettung dieser Elemente kann ein Forscher beliebig komplexe Bedingungskonstellationen bilden und so den Studienablauf bestimmen. Dieses Framework prüft und verarbeitet alle geladenen Condition-Chains mit einer Systeminternen Logik. So werden z.B. alle Bedingungen entsprechend ihrer „sampling rate“ gruppiert und in gemeinsamen Intervallen geprüft (5.3).

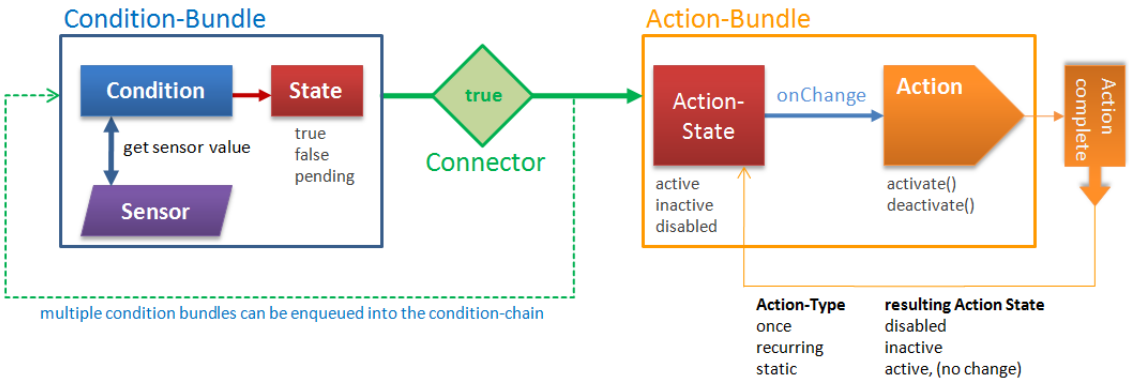

Abbildung 32 - Schematische Struktur des Trigger-Frameworks

Aufbauend auf das Trigger-Framework wurde das grafische Interface zur Trigger-Erstellung konzipiert, welches dem Forscher die Konfiguration von Studien erleichtern soll (siehe Kapitel 6). Der logische Aufbau des Interfaces spiegelt das zugrundeliegende Trigger-Framework wieder und basiert ebenfalls auf einer Verkettung von Bedingungen und Aktionen. Es werden Bedingungsobjekte, Aktionsobjekte und Verbindungselemente verwendet um eine so genannte Condition-Chain aufzubauen. Eine solche Bedingungskette stellt die Konfiguration einer Studie visuell dar und veranschaulicht durch die Logik eines Datenflusses von Objekt zu Objekt welche Aktionen von welchen Bedingungen abhängig sind (siehe Abbildung 33). Das gesamte Interface basiert auf einer Filter/Flow Visualisierung sowie der Möglichkeit durch semantisches Zooming dynamisch zwischen Detailkonfiguration und Übersicht zu wechseln. Objekte wie Bedingungen und Aktionen können per Drag and Drop zur Konfiguration hinzugefügt werden. Das Interface unterstützt den Forscher bei der Erstellung einer Konfiguration durch visuelle Hinweise auf erlaubte „Dropzones“ und einem automatischen Layouten der Landschaft. So sollen Fehler bei der Studienkonfiguration vermieden werden und das System für neue Forscher einfacher zu erlernen sein.

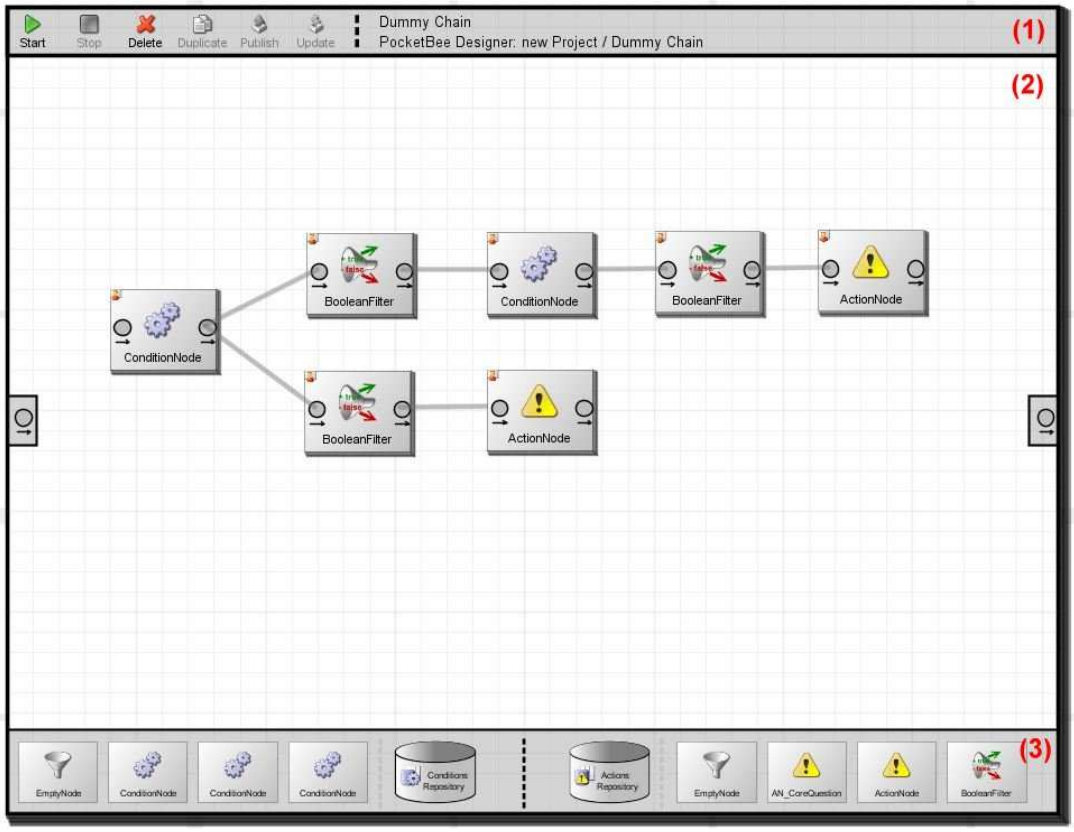

Abbildung 33 - grafisches Interface zur Trigger-Erstellung

Im Ausblick (Kapitel 7) werden die offenen Punkte und Erweiterungsmöglichkeiten angesprochen die in dieser Masterarbeit nicht behandelt werden konnten. Unter Anderem fehlt weiterhin eine systematische Unterstützung zur Analyse der Studiendaten welche noch manuell vom Forscher durchgeführt werden muss. Außerdem wird empfohlen das System (möglichst unter OpenSource Lizenz) externen Forschern zur Verfügung zu stellen, um die Leistungsfähigkeit zu testen und auch Anregungen aus anderen Fachbereichen zu bekommen. Die inhaltliche Erweiterung mit neuen Sensoren, Bedingungen und Aktionen kann im selben Zug durchgeführt werden wie deren Bedarf durch weitere Teststudien und Anwendungsfälle entsteht. Durch die modulare Erweiterbarkeit könnte jeder Forscher die von ihm benötigten Sensoren und Aktionen selbst anbinden und so in eigenen Studien verwenden. Dies würde die breite Anwendbarkeit von PocketBee stärken und so zu einer weitern Verbreitung beitragen.

Literaturverzeichnis

1. Blackwell, A.F., Whitley, K.N., Good, J., and Petre, M. Cognitive Factors in Programming with Diagrams. *Psychology*, , 1-12.

2. Bolger, N., Davis, A., and Rafaeli, E. Diary methods: capturing life as it is lived. *Annual review of psychology 54*, (2003), 579-616.

3. Bruno R. Preiss. *Data Structures and Algorithms with Object-Oriented Design Patterns in Java*. University of Waterloo, Waterloo, 1999.

4. Carter, S. and Heer, J. Momento : Support for Situated Ubicomp Experimentation. (2007).

5. Cockburn, A., Karlson, A., and Bederson, B.B. A review of overview+detail, zooming, and focus+context interfaces. *ACM Computing Surveys 41*, 1 (2008), 1-31.

6. Dierdorf, S. Spezifikation von Anforderungen und Interaktionsdesign einer multimodalen Tagebuch-Anwendung für Feld-Studien. 2011.

7. DIN. DIN-EN ISO 9241-110. http://de.wikipedia.org/wiki/EN_ISO_9241.

8. Froehlich, J., Chen, M.Y., Consolvo, S., et al. MyExperience : A System for In situ Tracing and Capturing of User Feedback on Mobile Phones. *Design*, (2007).

9. Gerken, J., Dierdorf, S., Schmid, P., Sautner, A., and Reiterer, H. Pocket Bee - a multi-modal diary for field research. *Information Systems*, (2010), 7-10.

10. Global Park. EFS Survey. 2011. http://www.globalpark.de/efs-uebersicht/efs-survey.html.

11. Google. Android Developers. 2011. http://developer.android.com/.

12. Google. Android Cloud-to-Device Messaging Framework. 2011. http://code.google.com/intl/de-DE/android/c2dm/index.html.

13. Green, T. Usability Analysis of Visual Programming Environments: A ‘Cognitive Dimensions’ Framework. *Journal of Visual Languages & Computing 7*, 2 (1996), 131-174.

14. Hektner, J.M., Schmidt, J.A., and Csikszentmihalyi, M. *Experience Sampling Method: Measuring the Quality of Everyday Life*. Sage Publications, 2006.

15. Hinckley, K. and Igarashi, T. Speed-dependent automatic zooming for browsing large documents. *UIST ’00 Proceedings of the 13th annual ACM symposium on User interface software and technology*, (2000).

16. Hofte, G.H.H. What ’ s that hot thing in my pocket ? SocioXensor , a smartphone data collector. *In Proceedings of e-Social Science.*, (2007).

17. IBM. SPSS. 2011. http://www.spss.com/.

18. Jetter, H.-christian, Gerken, J., Zöllner, M., Reiterer, H., and Milic-frayling, N. Materializing the Query with Facet-Streams – A Hybrid Surface for Collaborative Search on Tabletops. *Proceedings of the 29th international conference on Human factors in computing systems*, (2011).

19. Khan, V.-javed and Eggen, B. Features for the future Experience Sampling Tool. *Human Factors*, (2009), 2-5.

20. Lewis, C. and Olson, G. *Can principles of cognition lower the barriers to programming?* 1987.

21. Microsoft. Microsoft Office. 2011. http://office.microsoft.com/.

22. National Instruments. LABView. 1986. http://www.ni.com/labview/d/.

23. Rachuri, K.K., Mascolo, C., Rentfrow, P.J., and Longworth, C. EmotionSense : A Mobile Phones based Adaptive Platform for Experimental Social Psychology Research. *International Studies*, (2010), 281-290.

24. Rädle, R. Squidy - A Zoomable Design Environment for Natural User Interfaces. *Master Thesis HCI Universität Konstanz*, 2010.

25. Schmid, P. *PocketBee : Technischer Projektbericht*. 2010.

26. Schmid, P. *„ State of the Art “ mobile Werkzeuge für die ESM und Tagebuchmethode*. Konstanz, 2011.

27. Shneiderman, B. and Young, D. A Graphical Filter/Flow Representation of Boolean Queries: A Prototype Implementation and Evaluation. *The Carlyle Letters Online 20*, 1 (1993), e2-e2.

28. Weiser, M. The Computer of the 21st Century. *SIGMOBILE Mob. Comput*, (1999).

29. Wikipedia. Singleton Pattern. 2011. http://de.wikipedia.org/wiki/Singleton_(Entwurfsmuster).

30. Woodruff, A., Landay, J., and Stonebraker, M. Goal-directed zoom. *CHI 98 conference summary on Human factors in computing systems*, (1998).

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Masterthesis selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Ort, Datum	Unterschrift