ulm university universität
uulm

Fakultät für
Ingenieurwissenschaften
und Informatik
Institut für Datenbanken und
Informationssysteme

2014

Multi-Touch Gestures for Process Modeling

Masterarbeit an der Universität Ulm

Vorgelegt von:
Adam Just
adam.just@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert
Prof. Dr. Peter Dadam

Betreuer:
Dipl.-Inf. Jens Kolb

Fassung 25. Februar 2014

Satz: PDF-LaTeX 2ε

Abstract

Mithilfe sogenannter Prozesssichten, kann die Komplexität von Prozessmodellen reduziert werden. Die Erstellung sowie Bearbeitung von Prozesssichten wird durch das proView-Framework unterstützt. Dieses Framework kann aufgrund seiner web-basierten Bedienoberfläche auch auf mobilen Geräten, wie Smartphones oder Tablets, eingesetzt werden. Da mobile Geräte jedoch heutzutage ausschließlich mittels Fingern bedient werden, ist eine Anpassung des Desktop-orientierten Bedienkonzepts von proView notwendig. In dieser Arbeit werden hierzu Multi-Touch-Gesten ermittelt, implementiert und mit den entsprechenden Modellierungsfunktionen von proView verknüpft, um eine Gesten-basierte Prozessmodellierung auf Basis von Prozesssichten zu ermöglichen. Die Validierung der Implementierung wird mithilfe eines abschließenden Experiments durchgeführt.

Inhaltsverzeichnis

1 Einleitung

Die Komplexität von Prozessmodellen kann durch sogenannte Prozesssichten reduziert werden. Benutzer bekommen mithilfe einer Prozesssicht nur den Teil eines Prozesses zu sehen, der für sie relevant ist. Gleichzeitig können Veränderungen an einer Prozesssicht vorgenommen werden, die im komplexen Prozessmodell übernommen werden. Diese Technologie ist wesentlicher Bestandteil des proView-Frameworks[1], das an der Universität Ulm entwickelt wird. Es handelt sich hierbei um eine webbasierte Anwendung, die in Browsern unterschiedlicher Geräte ausgeführt werden kann. Dies gilt auch für mobile, internetfähige Geräte, wie Smartphones oder Tablets, die sich derzeit einer steigenden Beliebtheit erfreuen. Das derzeitige Problem von proView ist jedoch, dass es für den Einsatz auf Desktop-Computern optimiert ist und daher eine Menü-basierte Interaktion mittels Maus und Tastatur erzwingt [12]. Da mobile Geräte heutzutage ausschließlich mit den Fingern bedient werden und eine übermäßige Verwendung von Menüs für kleine Bildschirme ungeeignet ist, muss das proView-Framework an mobile Geräte angepasst werden.

Ziel dieser Arbeit ist es das proView-Framework um eine Gesten-basierte Prozessmodellierung zu erweitern, sowie die Verwendung von Menüs zu reduzieren und eine intuitive Bedienung von proView mittels Finger auf mobilen Geräten zu fördern. Hierzu sollen zunächst verschiedene Multi-Touch-Gesten ermittelt werden, die zu den von proView angebotenen Modellierungsfunktionen passen. Nach einer sorgfältigen Analyse der Umsetzungsmöglichkeiten sollen die Gesten implementiert und in das proView-Framework integriert bzw. mit den bestehenden Modellierungsfunktionen verknüpft werden. Abschließend soll eine experimentelle Untersuchung durchgeführt werden, um die implementierten Gesten zu validieren.

Kapitel 2 gibt einen kurzen Einblick in die Grundlagen, die für das Verständnis dieser Arbeit erforderlich sind. Eine ausführliche Analyse der Problemstellung thematisiert Kapitel 3, das unter anderem eine Ermittlung von Multi-Touch-Gesten enthält. Das darauffolgende Kapitel 4 beschäftigt sich mit unterschiedlichen Multi-Touch-Implementierungen für die ermittelten Gesten. Kapitel 5 beschreibt das in dieser Arbeit entwickelte Vaadin-Add-on TouchPanel. Dieses Add-on ist für die

[1] http://www.dbis.info/proView

in Kapitel 6 beschriebene Erweiterung des proView-Frameworks notwendig. Eine Evaluation in Form einer experimentellen Untersuchung der finalen Implementierung ist Gegenstand von Kapitel 7. Das abschließende Kapitel 8 beinhaltet eine Zusammenfassung dieser Arbeit sowie einen kurzen Ausblick auf zukünftige Arbeiten.

2 Grundlagen

Dieses Kapitel beschreibt grundlegende Aspekte, die für ein besseres Verständnis dieser Arbeit wichtig sind. Abschnitt 2.1 stellt verschiedene Bedienkonzepte vor, die auf Multi-Touch-Geräten eingesetzt werden können. Der anschließende Abschnitt 2.2 beinhaltet eine kurze Einführung in die Prozessmodellierung, indem die für diese Arbeit wichtigsten Prozesselemente vorgestellt werden. Das proView-Framework [13, 14] und das für dessen Implementierung eingesetzte Framework Vaadin [7] werden in Abschnitt 2.3 und 2.4 näher erläutert.

2.1 Varianten von Bedienkonzepten

Die Bedienung eines Multi-Touch-Gerätes kann je nach Benutzeroberfläche der Anwendung, die darauf ausgeführt wird, unterschiedlich aussehen. Bietet die Anwendung eine gewöhnliche Menüleiste im oberen Bereich des Bildschirms an, kann durch eine einfache Tippgeste ein Menüpunkt ausgewählt werden, woraufhin sich dieser aufgeklappt und dem Nutzer hierarchisch weitere Auswahlmöglichkeiten präsentiert. Beispielsweise ist bei vielen Anwendungen das Dateimenü vorhanden, das bei einem Antippen oder Anklicken dem Nutzer unter anderem die Möglichkeit bietet, die Anwendung zu beenden oder den aktuellen Programmzustand abzuspeichern. Bei diesem Verfahren handelt es sich um eine sogenannte *Menü-basierte Interaktion* mit dem System, die den meisten Computernutzern durchaus bekannt ist. Zu beachten ist, dass die Einträge eines Menüs eine Kantenlänge von mindesten 11,52 mm betragen sollten, um bei Eingaben per Finger eine Trefferwahrscheinlichkeit von 95% zu erzielen [31].

Auch ein Kontextmenü zählt zum Bereich Menü-basierte Interaktion. Bei diesem Verfahren klickt der Nutzer mit der rechten Maustaste typischerweise auf ein Bedienelement auf der Benutzeroberfläche und es öffnet sich das Kontextmenü mit verschiedenen Interaktionsfunktionen. Bei der Bedienung einer solchen Anwendung mit Fingern kann je nach Programmierung der Anwendung das Kontextmenü beispielsweise durch Antippen an einer bestimmten Stelle der Bedienoberfläche geöffnet werden. Abgesehen davon, dass Menü-basierte Interaktionen sehr

bekannt sind, haben sie den Vorteil, dass viele Funktionen der Anwendung sofort ersichtlich sind. Dies ist nämlich bei einer rein *Gesten-basierten Interaktion* nicht der Fall [11].

Bei der Gesten-basierten Interaktion weiß der Nutzer meistens erst nach einer Studie der Bedienungsanleitung oder einem Tutorial, mit welcher Geste er welche Funktion auslösen kann. Je mehr Funktionen eine Anwendung besitzt, desto mehr Gesten muss sich ein Nutzer merken, um mit der Anwendung problemlos interagieren zu können. Bei dieser Interaktionsart unterscheidet man weiter zwischen *physikalischen* und *symbolischen Gesten* [31]. Bei der *physikalischen Geste* werden Bedienelemente direkt manipuliert. Beispielsweise kann ein Element ausgewählt werden, und während sich der Finger noch darauf befindet, an eine andere Position verschoben werden. Diese Art der Berührung gibt einem Nutzer ein Gefühl der direkten Manipulation von Bildschirmelementen; im Gegensatz zu einer Maus oder Tastatur [23]. Bei *symbolischen Gesten* zeichnet der Nutzer verschiedene Symbole bzw. Metaphern auf den Bildschirm, die die Anwendung erkennt und daraus eine Funktion ableitet. Zeichnet der Nutzer beispielsweise ein Fragezeichen auf dem Bildschirm, so könnte die Hilfefunktion aufgerufen werden. An diesem Beispiel erkennt man sofort, dass es in manchen Fällen schwierig sein kann, ein geeignetes Symbol zu definieren, das sich ein Nutzer zudem auch merken kann.

Bei der Gesten-basierten Interaktion haben sich vor allem durch den Einsatz von mobilen Geräten, verschiedene Gesten für bestimmte Funktionen etabliert und sind mittlerweile bei den meisten Nutzern zur Gewohnheit geworden. Ein Beispiel hierfür ist die Zoom-Funktion, die mittels einer sogenannten *Pinch-Geste* ausgelöst wird. Der Nutzer berührt bei dieser Geste den Bildschirm mit zwei Fingern, die sich entweder aufeinander zubewegen (d.h. vergrößern) oder voneinander wegbewegen (d.h. verkleinern). Nach Ausführung dieser bekannten Geste könnte sich ein Kontextmenü öffnen, das weitere Funktionen zur Auswahl anbietet. In einem solchen Fall würde es sich um eine *hybride Interaktion* handeln, das heißt eine Kombination aus Menü- und Gesten-basierter Interaktion.

2.2 Prozessmodell

Die Prozessabläufe eines Unternehmens können mithilfe von Prozessmodellen textuell [15], grafisch [6] oder formal [1] dargestellt werden. Diese Prozessmodelle unterstützen unter anderem die Analyse von Geschäftsprozessen sowie die Kommunikation über Geschäftsprozesse. Des Weiteren können sie aufgrund von Optimierungsmaßnahmen zu einer Steigerung der Produktivität und des Umsatzes

eines Unternehmens beitragen.

Es existiert eine Vielzahl von Standards, die für Notationen von Prozessmodellen verwendet werden können. Eines davon ist die Business Process Model and Notation (BPMN)-Notation [19] mit der sich Prozessmodelle grafisch darstellen, modellieren und dokumentieren lassen. Die BPMN-Notation ist wichtiger Bestandteil von proView und daher werden die wichtigsten Elemente dieser Prozessnotation in Abschnitt 2.2.1 ausführlicher beschrieben.

Der gesamte Geschäftsprozess wird durch ein *Central Process Model* (CPM) repräsentiert. Allerdings ist nicht immer für jeden Benutzer das gesamte Prozessmodell von Bedeutung. Für einen Manager sind zum Beispiel bestimmte Aktivitäten wichtiger als für einen „gewöhnlichen“ Bearbeiter. Daher besteht die Möglichkeit aus einem komplexen Prozessmodell sogenannte *Prozesssichten* (engl. *Process Views*) zu abstrahieren, die den einzelnen Benutzern zugeordnet werden. Prozesssichten beinhalten dabei nur die wichtigsten Prozesselemente, die für einen Benutzer relevant sind. Alle anderen Prozesselemente sind nicht sichtbar. Dies verbessert zusätzlich die Pflege sowie Erweiterbarkeit eines Prozessmodells. Die von den Benutzern durchgeführten Änderungen an den Prozesssichten werden an das CPM weitergeleitet und auch in anderen Prozesssichten übernommen, die dieselben Prozesselemente beinhalten.

2.2.1 BPMN-Prozesselemente

Die BPMN-Notation [19] ist sehr umfangreich, aus diesem Grund sind im Folgenden nur die für diese Arbeit wichtigsten Prozesselemente beschrieben, die auch von proView unterstützt werden.

Start- und End-Ereignis Start- und End-Ereignis (engl. *Start/End Event*) werden in einem Prozessmodell als Kreis mit unterschiedlicher Linienstärke repräsentiert. Sie signalisieren den Anfang bzw. das Ende eines Prozesses und werden entsprechend im Prozessmodell positioniert (siehe Abb. 2.1).

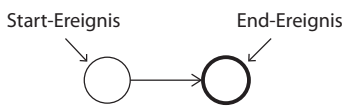

Abbildung 2.1: Start- und End-Ereignis.

Aktivität Eine Aktivität (engl. *Activity*) wird als Rechteck mit abgerundeten Ecken repräsentiert. Sie symbolisiert eine atomare Aufgabe (engl. *Task*) oder ein

Subprozess (engl. *Subprocess*) und verweist auf ein weiteres Prozessmodell, das das Verhalten der Subprozess-Aktivität näher beschreibt. Im kollabierten Zustand wird ein Subprozess mit einem Plus-Symbol am unteren Rand des Elements kenntlich gemacht. Im expandierten Zustand durch ein Minus-Symbol (siehe Abb. 2.2).

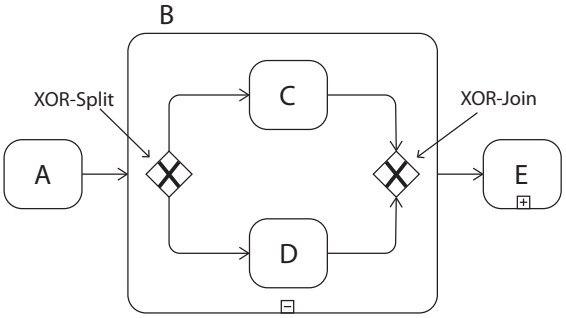

Abbildung 2.2: Aktivitäten (A, C, D) und Subprozesse (B, E).

Verzweigungsblock Ein Verzweigungsblock (engl. *Branching Block*) wird durch zwei Raute-ähnliche Symbole repräsentiert. Innerhalb der Symbole kann sich ein weiteres X- oder Plus-Symbol befinden (siehe Abb. 2.3). In diesen beiden Fällen würde es sich um ein XOR- bzw. AND-Verzweigungsblock handeln, die von proView unterstützt werden. Der XOR-Verzweigungsblock wird verwendet, um in einem Prozess alternative Prozessflüsse zu schaffen, von denen aufgrund einer bestimmten Bedingung nur einer ausgeführt wird. Die Parameter für diese Bedingung werden aus einem Datenelement ausgelesen. Bei einem AND-Verzweigungsblock werden zwei oder mehrere Prozessflüsse parallel ausgeführt. Ein AND-Split-Verzweigungsknoten sorgt am Anfang des Verzweigungsblocks für die Aufsplittung des Prozessflusses und ein AND-Join-Verzweigungsknoten am Ende des Verzweigungsblocks für die Zusammenführung. Bei einem XOR-Verzweigungsblock handelt es sich dabei immer um die Verzweigungsknoten XOR-Split und XOR-Join.

Die Prozessmodelle in proView besitzen generell eine Blockstruktur, wodurch die verschiedenen Blöcke geschachtelt und disjunkt auftreten. Überlappungen sind allerdings nicht möglich.

Schleife Einzelne Prozessschritte bzw. Aktivitäten können wiederholt werden, indem der Prozessfluss mit den XOR-Verzweigungsknoten als Schleife (engl. *Loop*) zurückgeleitet wird (siehe Abb. 2.4). Der XOR-Split-Verzweigungsknoten besitzt hierbei eine Abbruchbedingung, die bei Nichterfüllung eine Weiterleitung zum XOR-Join-Verzweigungsknoten einleitet. Innerhalb des Schlei-

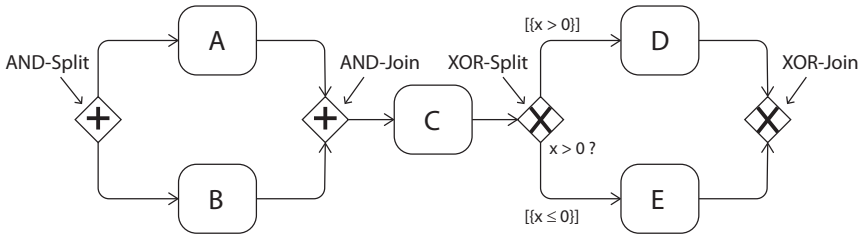

Abbildung 2.3: AND- und XOR-Verzweigungsblock.

fenblocks wird der Wert eines Datenelements modifiziert, das zur Prüfung der Abbruchbedingung vom XOR-Split-Verzweigungsknoten ausgelesen wird.

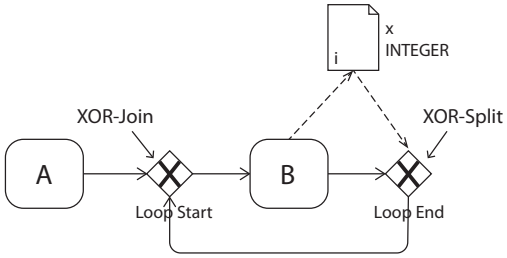

Abbildung 2.4: Eine Schleife.

Datenelement Ein Datenelement (engl. *Data Element*) wird durch eine Seite mit abgeknickter, oberer Ecke repräsentiert (siehe Abb. 2.5). Es hat keinen direkten Einfluss auf den Kontrollfluss, sondern repräsentiert bestimmte Daten bzw. Variablen, die während eines Prozesses relevant sind. Aktivitäten können Datenelemente auslesen oder beschreiben. Ein Datenelement hat einen definierten Datentyp, wie zum Beispiel String, Integer, Float oder Boolean.

Abbildung 2.5: Datenelemente und deren Typen.

Sequenzflusskante Eine Sequenzflusskante (engl. *Sequence Flow Edge*) wird durch einen gewöhnlichen, durchgehenden Pfeil repräsentiert (siehe Abb. 2.6). Sie verbindet Ereignisse, Aktivitäten und Verzweigungsblöcke miteinander und bestimmt deren Ausführungsreihenfolge.

Synchronisationskante Eine Synchronisationskante (engl. *Synchronization Ed-*

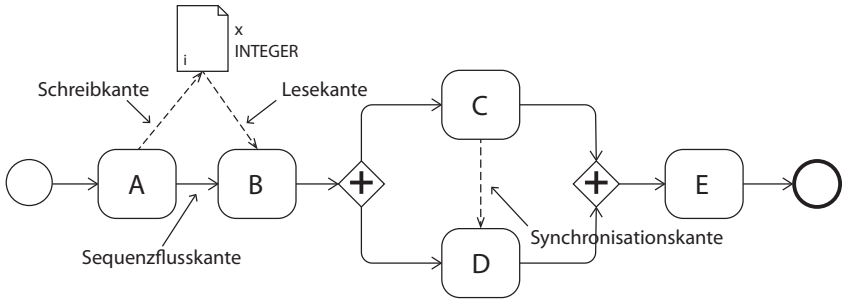

Abbildung 2.6: Sequenzfluss-, Synchronisations-, Lese- und Schreibkanten.

ge) wird durch einen gestrichelten Pfeil repräsentiert und dient dazu, Abhängigkeiten zwischen Aktivitäten innerhalb von AND-Verzweigungsblöcken zu definieren (siehe Abb. 2.6). Eine Aktivität D wird beispielsweise erst nach Beendigung von Aktivität C, die auf einem anderen Zweig liegt, ausgeführt.

Lese- und Schreibkanten Lese- und Schreibkanten (engl. *Read/Write Edge*) werden wie Synchronisationskanten durch einen gestrichelten Pfeil repräsentiert (siehe Abb. 2.6). Ihre Aufgabe ist es allerdings Lese- und Schreiboperationen zwischen Datenelementen und Aktivitäten bzw. Verzweigungsknoten zu symbolisieren. Wird beispielsweise ein Datenelement von einer Aktivität ausgelesen, kennzeichnet eine Lesekante vom Datenelement zur Aktivität diese Leseoperation. Schreiboperationen werden durch eine Schreibkante von Aktivität bzw. Verzweigungsknoten zu Datenelement symbolisiert.

2.3 proView-Framework

Das proView-Framework[1] ist ein Projekt des Instituts für Datenbanken und Informationssysteme an der Universität Ulm. Es bietet im Wesentlichen die Möglichkeit, aus komplexen Prozessmodellen personalisierte Prozesssichten zu abstrahieren, die von bestimmten Benutzern bzw. Rollen bearbeitet werden können [12, 13].

Als Web-Anwendung nutzt proView eine Client-Server-Architektur (siehe Abb. 2.7) [12]. Der proView-Server ist hierbei für die gesamte Programmlogik, sowie Verwaltung der einzelnen Prozessmodelle und -sichten verantwortlich. Der proView-Client übernimmt die Visualisierung von Prozessmodellen, die entweder als Formular, Text [15] oder in den beiden Notationen BPMN und ADEPT [3] dargestellt werden können. Die webbasierte Benutzeroberfläche von proView ist mithilfe des

[1] http://www.dbis.info/proView

Vaadin-Frameworks entwickelt worden [2], das in Abschnitt 2.4 genauer beschrieben ist. Dies hat den Vorteil, dass proView mit den meisten aktuellen Browsern kompatibel ist. In dieser Arbeit wird die Benutzeroberfläche erweitert, damit Multi-Touch-Ereignisse, die für eine Gesten-basierte Prozessmodellierung notwendig sind, entgegen genommen werden können.

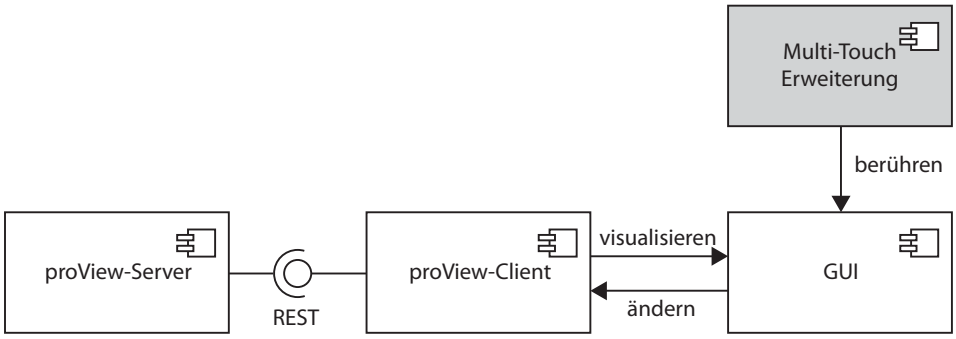

Abbildung 2.7: proView-Framework mit Multi-Touch-Erweiterung.

In Abbildung 2.8 ist die Benutzeroberfläche von proView mit den wesentlichen Bestandteilen zu sehen.

Menüleiste (1) Die Menüleiste bietet zusätzliche Einstellmöglichkeiten an. Zum Beispiel ist ein Wechsel zwischen den unterschiedlichen Notationen, wie BPMN oder ADEPT, durchführbar. Es stehen zwei unterschiedliche Layoutvarianten (Normal und Simulated) zur Verfügung, um zwischen einem Block- oder CPM-orientierten Layout zu wechseln [2].

TemplateTree (2) Der TemplateTree stellt alle zur Verfügung stehenden Prozessmodelle (d.h. CPM oder Prozesssichten) in einer baumartigen Struktur dar. Über den TemplateTree können auch neue Prozesssichten erstellt oder vorhandene gelöscht werden.

Bearbeitungsbereich (3) Das im TemplateTree ausgewählte Prozessmodell wird in einem Bearbeitungs- und Visualisierungsbereich angezeigt.

Kontextmenü (4) Die Modellierungsfunktionen von proView können mithilfe eines Kontextmenüs ausgeführt werden.

Properties-Panel (5) Wird im Bearbeitungsbereich ein Prozesselement ausgewählt, werden weitere Informationen zu diesem Element, wie beispielsweise die ID, im Properties-Panel angezeigt.

Create Operations-Panel (6) Mit dem Create Operations-Panel werden alle Operationen aufgelistet, die in Bezug auf das angezeigte Prozessmodell aus-

geführt worden sind.

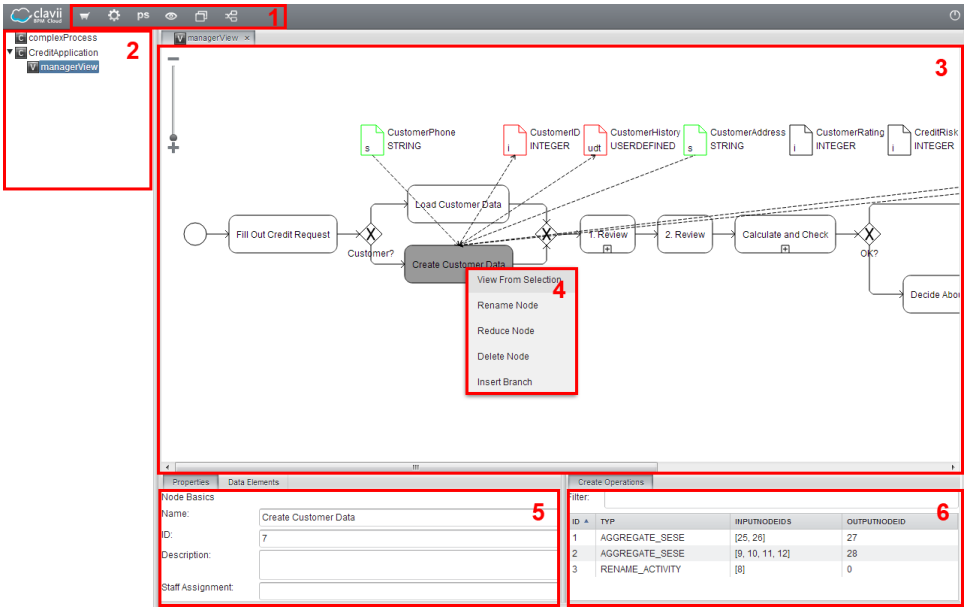

Abbildung 2.8: Webbasierte Benutzeroberfläche von proView.

2.4 Vaadin-Framework

Vaadin ist ein Framework, das der Entwicklung von umfangreichen Benutzerschnittstellen für Web-Anwendungen dient [7]. Eine gewöhnliche Web-Anwendung besteht bekanntlich aus einer Client- und einer Serverseite für deren Entwicklung Kenntnisse in unterschiedlichen Web-Technologien wie HTML oder JavaScript notwendig sind. Beide Seiten werden in Vaadin hauptsächlich mit der Programmiersprache Java entwickelt, sodass sich der Entwickler auf die wichtigere Programmlogik einer Web-Anwendung konzentrieren kann. Zusätzlich wird eine Vielzahl an vordefinierten UI-Komponenten angeboten, die auch in Java programmiert sind. Eine UI-Komponente, wie beispielsweise ein Button, besteht hierbei aus einer serverseitigen und einer clientseitigen Komponente, auch *Widget* genannt.

Das clientseitige Framework von Vaadin basiert auf dem Google Web Toolkit (GWT) und nutzt eine erweiterte Version des GWT-Compilers, um den Java-Code eines Widgets vor der Ausführung in JavaScript umzuwandeln. Die serverseitige Komponente wird als Java-Servlet in einem Anwendungsserver wie Apache Tomcat ausgeführt. Die Nutzung von GWT hat zusätzlich den Vorteil, dass die Web-Anwendung dadurch mit den meisten aktuellen Browsern kompatibel ist.

Clientseitige Widgets sind insbesondere für die Visualisierung der Benutzerschnittstelle zuständig, sowie für die Weiterleitung von Benutzereingaben an den Server. Die Durchführung dieser Kommunikation zwischen Client und Server wird ebenfalls vom Vaadin-Framework übernommen, das hierfür die Web-Technologie AJAX verwendet.

Das Erscheinungsbild der Benutzerschnittstelle bzw. der einzelnen UI-Komponenten kann durch sogenannte CSS-basierte Themes beeinflusst werden. Der Entwickler hat hier die Möglichkeit eigene Themes zu definieren oder Standard Themes von Vaadin zu verwenden.

Die Auswahl an vorgefertigten UI-Komponenten kann durch weitere Komponenten, die als Add-ons auf der Vaadin-Webseite zu finden sind, ergänzt werden. Darüber hinaus ist es ebenfalls möglich, eigene Add-ons zu entwickeln und in eine Web-Anwendung zu integrieren.

3 Analyse der Problemstellung

Bevor mit der eigentlichen Implementierung einer Multi-Touch-Erweiterung für das proView-Framework begonnen werden kann, wird in diesem Kapitel analysiert, welche Gesten sich am besten für diese Erweiterung eignen. Abschnitt 3.1 ermittelt zunächst einige Gesten, die eventuell zum Einsatz kommen könnten. Abschnitt 3.2 beschäftigt sich anschließend mit der Zuordnung der Gesten zu passenden Modellierungsfunktionen in proView.

3.1 Ermittlung von Multi-Touch-Gesten

Die folgenden Abschnitte stellen drei Arbeiten vor, die als Ergebnis jeweils ein Gesten-Set für den Einsatz in Multi-Touch-Anwendungen präsentieren. Abschnitt 3.1.1 und 3.1.3 beinhalten Gesten-Sets, die zum Zwecke einer besseren Multi-Touch-Bedienung experimentell ermittelt worden sind. Im zweiten Fall handelt es sich um ein Experiment, das im direkten Zusammenhang mit Prozessmodellierung steht. Welche Gesten sich bereits in bestehenden, interaktiven Systemen etabliert haben, ist Thema der Arbeit von Abschnitt 3.1.2.

3.1.1 Benutzerdefiniertes Gesten-Set

Für gewöhnlich entscheiden Softwaredesigner, welche Funktionen einer Anwendung mit welchen Gesten ausgelöst werden. Dies hat allerdings den Nachteil, dass Nutzer mit sehr wenig Technikkenntnissen Probleme bei der Bedienung haben, da sie eventuell eine andere Geste bevorzugen würden. Welche Geste für welche Funktion von den Nutzern bevorzugt wird, ermittelt ein Experiment in [31], um darauf aufbauend ein benutzerdefiniertes Gesten-Set zu präsentieren. Dieses Gesten-Set bietet Softwaredesignern die Möglichkeit, bessere Gesten für ihre eigenen Implementierungen zu entwickeln.

Den einzelnen Testpersonen dieses Experiments werden auf einem Touch-Table von Microsoft verschiedene Grundobjekte gezeigt. Die Software des Touch-Tables

führt automatisierte Aktionen aus, die die gezeigten Grundobjekte für die Testpersonen sichtbar verändern. Zum Beispiel zeigt die Software zunächst ein Rechteck an und führt anschließend die Aktion zum Vergrößern des Rechtecks aus. Jede Testperson muss sich diese Aktionen anschauen und daraufhin versuchen dieselbe Aktion durch eigene Gesten herbeizuführen (siehe Abb. 3.1). Wie viele Finger bzw. Hände sie dafür benutzt, entscheidet die Testperson selbst. Gleiches gilt für die Verwendung einer symbolischen oder einer physikalischen Geste.

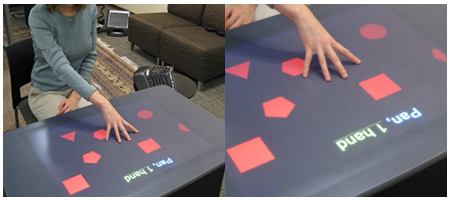

Abbildung 3.1: Teilnehmer des Experiments führt eine Pan-Geste aus [31].

Nach einer sorgfältigen Analyse der ermittelten Daten hat sich gezeigt, dass 43,9% aller verwendeten Gesten, physikalische Gesten sind und damit häufiger verwendet werden als symbolische Gesten mit 28,9%. Bei den verbleibenden 27,2% handelt es sich um abstrakte Gesten, wie beispielsweise das Ausführen einer Lösch-Geste durch dreimaliges Antippen eines Bedienelements. Des Weiteren ist die Komplexität einer jeden Geste ermittelt worden. Die Geste zum Rückgängig machen von Eingaben stellt sich dabei als komplizierteste Geste heraus, denn hier müssen die Testpersonen am längsten überlegen, welche Bewegung sich am besten für diese Funktion eignet. Im Gegensatz dazu ist die Verschieben-Geste die einfachste Geste. Hier müssen die Testpersonen im Durchschnitt nur acht Sekunden vor Gestenausführung nachdenken. Bei den komplizierten Gesten sind es im Durchschnitt 15 Sekunden. Für die 27 Aktionen wird für 25 Aktionen nur eine Hand verwendet. Die zweite Hand kommt lediglich bei den Aktionen „Einfügen“ und „Maximieren“ zum Einsatz. 72,0% aller Gesten werden ähnlich wie mit einer Maus ausgeführt. Das heißt, die Testpersonen wählen die Bedienelemente zunächst mit einem Antippen aus und führen anschließend eine Veränderung aus. Die Anzahl der verwendeten Finger ist in den meisten Fällen unbedeutend. Mehrere Finger sind beispielsweise bei größeren Bedienelementen von Bedeutung, da diese Bedienelemente den Anschein erwecken, dass man hier zur Manipulation mehr „Kraft“ aufwenden muss. Bei Vergrößerung von Elementen werden ebenfalls mehrere Finger verwendet, sowie für Aktionen bei denen man sicherstellen will, dass auch wirklich gedrückt wird. In einigen Fällen verwenden Testpersonen für verschiedene Aktionen dieselben Gesten. Zum Beispiel ist die Pinch-Geste einmal für die

Vergrößerung eines Bedienelementes nützlich und ein anderes Mal zum Zoomen, indem die Geste auf dem Hintergrund angewendet wird.

Als Ergebnis des Experiments ist ein benutzerdefiniertes Gesten-Set ermittelt worden, das sich im Anhang A.1 dieser Arbeit befindet.

3.1.2 Symbolische Gesten interaktiver Systeme

Vor der Einführung von Smartphones und Tablets mit Multi-Touch-Funktionalität existierten bereits Geräte, wie Tablet-PCs und Pocket-PCs, die mit einem Stift bedient werden können. Für diese Geräte sind bereits interaktive Systeme inklusive Gestenerkennern und Bibliotheken entwickelt worden, die eine Vielzahl an unterschiedlichen symbolischen Gesten unterstützen [9, 16, 17, 25]. Einige dieser Gesten haben sich bewährt und können deshalb auch auf aktuelle Multi-Touch-Entwicklungen angewendet werden. Eine Zusammenstellung der 16 nützlichsten symbolischen Gesten, die im Kontext der Entwicklung des *$1 Gesture Recognizers* [30] (siehe Kapitel 5.3.1) erstellt worden ist, ist in Abbildung 3.2 zu sehen.

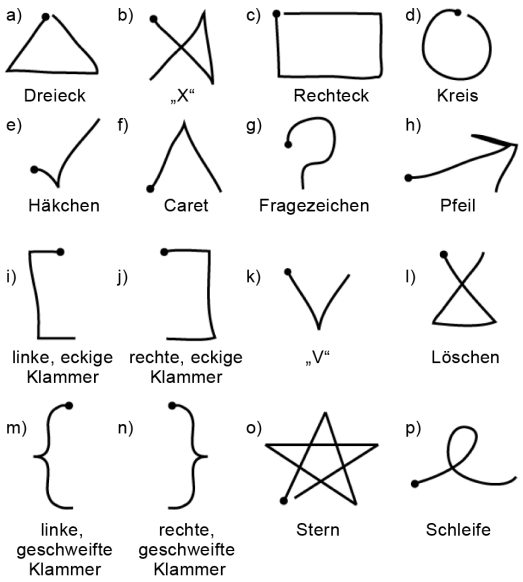

Abbildung 3.2: 16 verschiedene, symbolische Gesten [30].

3.1.3 Multi-Touch-Gesten für die Prozessmodellierung

In [23] ist ein Experiment zum Thema Gesten-basierte Prozessmodellierung beschrieben. Bei diesem Experiment sind zunächst acht elementare Modellierungs-

funktionen festgelegt worden (siehe Tabelle 3.1), zu denen sich Testpersonen geeignete Multi-Touch-Gesten überlegen müssen. Dabei beachtet das Experiment unterschiedliche Benutzertypen (erfahrene und unerfahrene Benutzer) und Bedienkonzepte, sowie spezifische Problemstellungen wie das Verdecken der Bedienoberfläche durch Hände und Finger, Präzision beim Tippen, Ermüdung von Gliedmaßen, Anzahl verwendeter Finger, beidhändige Bedienung und gemeinsames Arbeiten.

Es sind zunächst drei unterschiedliche Gesten-Sets definiert worden, bestehend aus einem Set für rein Gesten-basierte Modellierung, für Menü-basierte Modellierung mittels Menüleisten und schließlich eine hybride Lösung der ersten beiden Ansätze, die ein gutes Verhältnis zwischen einfach zu erlernenden und schnell auszuführenden Gesten beinhaltet. Das Experiment evaluiert diese Hybridlösung und liefert als Resultat ein finales Gesten-Set.

Insgesamt lösen 26 Testpersonen auf einem Apple iPad acht aufeinander aufbauende Aufgaben, um einen bestehenden Prozess eines Bestellvorgangs zu modellieren. Jede Aufgabe beinhaltet die Anwendung eines der Modellierungsfunktionen (siehe Tabelle 3.1). Die Reihenfolge der Aufgaben ist dabei bei jeder Testperson gleich. Die Testpersonen können jedoch frei entscheiden, welche Geste sie für welche Modellierungsfunktion anwenden.

Elementare Modellierungsfunktionen			
MF1	Aktivität einfügen	MF5	Datenelement einfügen
MF2	Element benennen	MF6	Lese- und Schreibkanten setzen
MF3	Verzweigungsblock einfügen	MF7	Subprozess bilden und auflösen
MF4	Verzweigung einfügen	MF8	Element löschen

Tabelle 3.1: Elementare Modellierungsfunktionen.

Dieses Experiment beinhaltet eine Unterscheidung zwischen symbolischen, physikalischen und Menü-basierten Gesten. Zum Beispiel ist eine gestrichelte Linie mit einem Pfeil, die auf diese Weise eine Lese- oder Schreibkante repräsentiert, eine symbolische Geste. Physikalische Gesten sind zum Beispiel Drag & Drop-Bewegungen beim Erstellen einer gewöhnlichen Kante. Menü-basierte Gesten beinhalten die Verwendung von Menüleisten und Kontextmenüs, die von den Testpersonen zeichnerisch dargestellt werden. Im Gegensatz zu Menü-basierten Gesten bereitet die Zuordnung anderer ausgeführter Gesten zu einer bestimmten Kategorie allerdings Schwierigkeiten, da dies nicht immer ersichtlich ist.

Die Aufgaben werden den Testpersonen in Form von statischen Bildern präsentiert, die aus dem zu bearbeitenden Prozess und einem Aufgabentext bestehen.

Die Testpersonen zeichnen ihre Lösungen in diese Bilder ein, die anschließend als Testergebnis abgespeichert werden.

Die abschließende Analyse des Experiments ergibt, dass die insgesamt 208 ermittelten Gesten aus 57 symbolischen Gesten, 93 physikalischen Gesten und 58 Menü-basierten Gesten bestehen. Abbildung 3.3 stellt den prozentualen Anteil eines Gestentyps pro Modellierungsfunktion dar. Es hat sich herausgestellt, dass Testpersonen, die Erfahrung mit Multi-Touch-Geräten haben, physikalische sowie Menü-basierte Gesten bevorzugen, wohingegen unerfahrene Testpersonen zu symbolischen Lösungen tendieren. Die Verwendung von beiden Händen oder mehreren Fingern lehnen die Testpersonen größtenteils ab. Diese Möglichkeiten kommen lediglich bei der Pinch-Geste sowie beim Zoomen zum Einsatz.

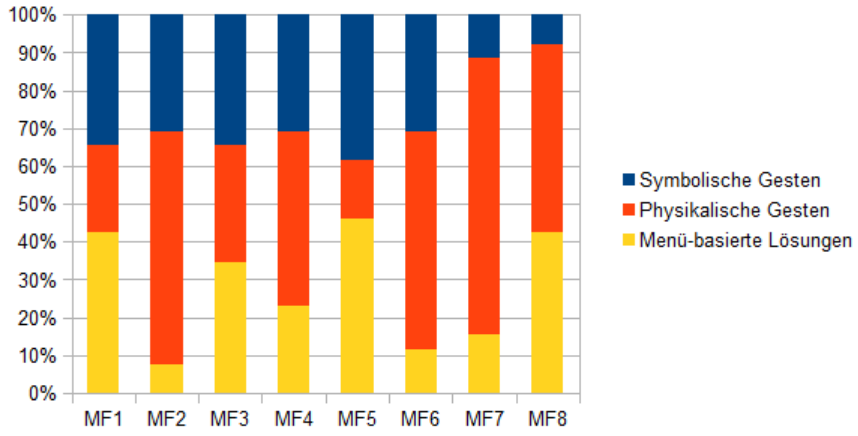

Abbildung 3.3: Anteil der unterschiedlichen Gesten in Bezug auf die einzelnen Modellierungsfunktionen.

3.2 Abbildung von Multi-Touch-Gesten auf Prozessmodellierungsfunktionen

Die Gesten aus den vorherigen Abschnitten sind genauer betrachtet worden, um passende Gesten für die in proView implementierten Modellierungsfunktionen zu definieren. Im Folgenden stellen wir die einzelnen Modellierungsfunktionen vor. Gleichzeitig findet die Zuweisung einer entsprechenden Geste zu einer Modellierungsfunktion statt.

F1 (Element auswählen) Die einzelnen Aktivitäten, Verzweigungsblöcke und Datenelemente eines Prozessmodells können selektiert werden. Diese Modellierungsfunktion nutzt ein Benutzer während der Bearbeitung eines Prozessmodells sehr häufig und sollte daher auf sehr einfache und schnelle Weise ausgelöst werden. Deshalb wird zum Auswählen eines einzelnen Prozesselements die Tap-Geste verwendet. Die Benutzer tippen dabei mit einem Finger auf ein Prozesselement und lösen damit die Auswahlfunktion aus (siehe Abb. 3.4). Ausgewählte Prozesselemente werden in proView farblich hervorgehoben.

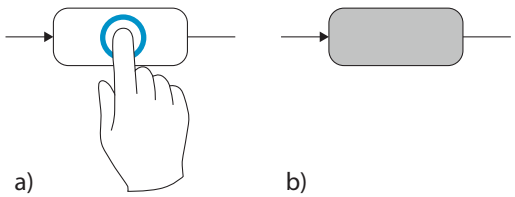

Abbildung 3.4: Aktivität selektieren a) und farblich hervorheben b).

Um mehrere Prozesselemente gleichzeitig auszuwählen, wird eine symbolische Geste in Form eines Rechtecks verwendet (siehe Abb. 3.5). Das Symbol „S“ kennzeichnet den Startpunkt der Geste. Der Benutzer kann anschließend selbst entscheiden, in welche Richtung er die Geste weiter zeichnet.

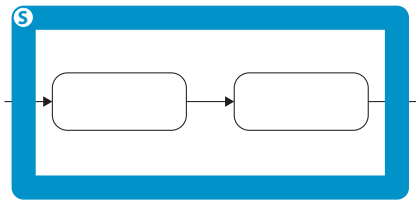

Abbildung 3.5: Mehrere Prozesselemente gleichzeitig auswählen.

Die Verwendung einer Tap-Geste und des Rechteck-Symbols ist auch ein Vorschlag in [23]. Eine Untersuchung der Auswahlfunktion ist in diesem Kontext nicht durchgeführt worden. Die Tap-Geste ist jedoch Bestandteil des resultierenden Gesten-Sets von [31] (siehe Abschnitt 3.1.1).

F2 (Aktivität einfügen) In ein bestehendes Prozessmodell können weitere Aktivitäten eingefügt werden. Diese Modellierungsfunktion sollte ebenfalls schnell und einfach auslösbar sein, da diese bei der Modellierung häufig verwendet wird.

Eine intuitive Lösung ist das Zeichnen eines Rechtecks oder eines Kreises beispielsweise auf der Sequenzflusskante zwischen zwei bestehenden Aktivitäten. Dieser Ansatz benötigt jedoch viel Ausführungszeit und Platz im Prozessmodell. [23] stellt eine physikalische Geste vor, die die Ausführungszeit, sowie den benötigten Platz reduziert. Es handelt sich hierbei um eine vertikale Drag & Drop-Bewegung, die der Benutzer auf der Sequenzflusskante zwischen zwei Aktivitäten ausführt (siehe Abb. 3.6). Zeichnerisch ähnelt die Geste einem vertikalen Strich und wird daher auch *Strich-Geste* genannt.

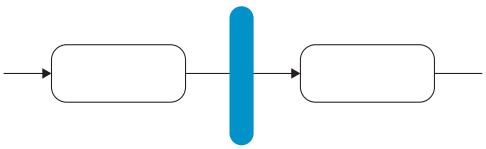

Abbildung 3.6: Aktivität einfügen.

Eine alternative Lösung ist die Verwendung einer Tap-Geste, die ein Kontextmenü einblendet. Mit diesem Menü kann anschließend eine neue Aktivität eingesetzt werden. Angetippt wird die Sequenzflusskante zwischen zwei Aktivitäten. Bei dieser Lösung ist jedoch aufgrund des Kontextmenüs ein Schritt mehr notwendig als für die Strich-Geste. Darüber hinaus ist ein präzises Antippen der Sequenzflusskante Voraussetzung, da eventuell benachbarte Prozesselemente ungewollt ausgewählt werden.

Der Name der neuen Aktivität wird mithilfe eines Dialogfensters und einer virtuellen Tastatur eingegeben.

F3 (Element umbenennen) In einigen Fällen ist es notwendig, den Namen bestehender Prozesselemente zu ändern. [23] schlägt hierzu eine Tap-Geste oder eine Double-Tap-Geste vor, die der Benutzer auf die entsprechende Aktivität anwendet. Mit der Tap-Geste werden in dieser Arbeit bereits Prozesselemente ausgewählt, daher wird für die Umbenennen-Funktion die Double-Tap-Geste eingesetzt (siehe Abb. 3.7).

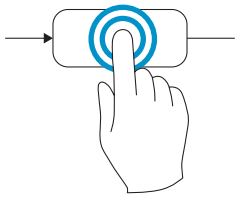

Abbildung 3.7: Element umbenennen.

Nach Auslösen dieser Funktion öffnet sich ein neues Dialogfenster mit einem Eingabefeld, in dem der neue Name des Prozesselements mithilfe einer virtuellen Tastatur eingegeben werden kann.

F4 (Element löschen) Einzelne Prozesselemente, wie Verzweigungsknoten oder Aktivitäten, können aus einem Prozessmodell entfernt werden. Eine intuitive Geste für diese Funktion ist das Durchstreichen des zu löschenden Elements. Dabei werden verschiedene Symbole wie ein diagonaler Strich, X- oder ein Zickzack-Symbol verwendet.

Bei der Wahl der Geste muss darauf geachtet werden, dass sie nicht zu einfach, aber auch nicht zu kompliziert ist. Bei einer einfachen Geste, wie zum Beispiel dem diagonalen Strich, kann es schnell passieren, dass ein Benutzer Prozesselemente ungewollt löscht. An dieser Stelle ist deshalb der Einsatz einer Sicherheitsabfrage notwendig. Komplexe Gesten, wie zum Beispiel ein Zickzack-Symbol, benötigen mehr Ausführungszeit, schließen den Einsatz einer Sicherheitsabfrage allerdings aus, da ungewollte Löschvorgänge nicht so schnell auftreten können.

Die Löschfunktion wird ebenfalls in [23] untersucht. Das finale Gesten-Set des Experiments beinhaltet hierfür eine Hold-Geste, die ein Kontextmenü aufruft. Der Benutzer verweilt mit seinem Finger für eine gewisse Zeit auf dem zu löschenden Prozesselement, bis sich das Kontextmenü öffnet. Anschließend kann er mit diesem Kontextmenü die Löschfunktion auslösen.

Einige Testpersonen bevorzugen bei diesem Experiment allerdings auch eine zeichnerische Lösung in Form eines X- oder Zickzack-Symbols. Eine Vielzahl an interaktiven Systemen auf Tablet-PCs verwenden ebenfalls ein solches Symbol, wie das Gesten-Set in Abschnitt 3.1.2 beweist. Ein Vorteil symbolischer Gesten ist, dass nur ein Schritt zum Löschen notwendig ist und nicht zwei Schritte, wie es bei dem Kontextmenü der Fall ist.

Aufgrund dieser Ergebnisse wird für die Löschfunktion eine Geste in Form eines X-Symbols bevorzugt (siehe Abb 3.8). Dabei ist zu beachten, dass der Finger während der Gestenausführung nicht abgesetzt werden darf.

Abbildung 3.8: Element löschen.

F5 (Elemente aggregieren) Mit dieser Modellierungsfunktion ist es möglich ein oder mehrere Prozesselemente zu einem Subprozess zusammenzufassen, wodurch eine Steigerung der Übersichtlichkeit erreicht wird.

Eine intuitive Lösung ist das Zeichnen eines Kreises oder eines Rechtecks um die zu aggregierenden Prozesselemente (siehe Abb. 3.9). Da das Rechteck-Symbol nicht nur für diese Modellierungsfunktion, sondern auch für die Funktionen F7 (Verzweigungsblock einfügen) und F14 (Prozesssicht erstellen) verwendet wird, ist der zusätzliche Einsatz eines Kontextmenüs notwendig, um die Modellierungsfunktion auszulösen.

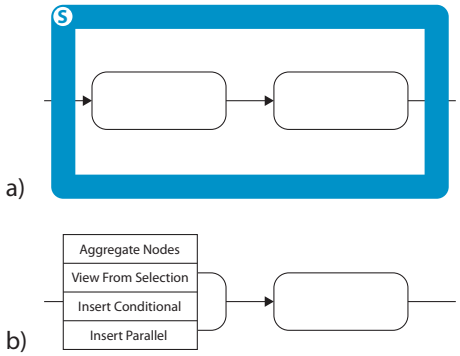

Abbildung 3.9: Prozesselemente auswählen a) und entsprechende Modellierungsfunktion per Kontextmenü ausführen b).

F6 (Element reduzieren) Prozesselemente können aus einer Prozesssicht entfernt werden, sind aber im CPM noch vorhanden, d.h. sie werden nicht vollständig gelöscht. Auf diese Weise ist es möglich, die Übersichtlichkeit eines Prozessmodells zu verbessern.

Da diese Funktion ähnlich zur Funktion F4 (Element löschen) ist, stellt das Durchstreichen der Prozesselemente mittels Kreuz oder Zickzack-Symbol auch hier die intuitive Lösung dar. Diese Gesten eignen sich allerdings nicht besonders gut, um mehrere Prozesselemente gleichzeitig zu reduzieren. [23] stellt zum Löschen von Prozesselementen noch eine weitere Geste vor, die von der Form her einem Verbotsschild ähnelt. Eine ähnliche Geste wird auch an dieser Stelle für das Reduzieren von Prozesselementen verwendet (siehe Abb. 3.10). Diese Geste ist aber nicht wie in [23] kreisförmig, sondern rechteckig.

Der Benutzer zeichnet wie beim Aggregieren ein Rechteck um die zu reduzierenden Prozesselemente. Das Symbol „S“ kennzeichnet hierbei den Startpunkt der Geste. Zusätzlich zeichnet er einen diagonalen Strich von der obe-

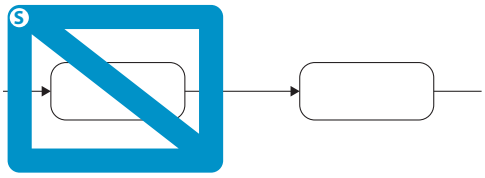

Abbildung 3.10: Element reduzieren.

ren, linken Ecke zur unteren, rechten Ecke des Rechtecks ein. Diese symbolische Geste wird ohne Absetzen des Fingers ausgeführt. In welche Richtung der Benutzer seinen Finger bei Gestenausführung vom Startpunkt aus weiter bewegt, kann er selbst entscheiden.

Der Vorteil dieser Geste ist, dass kein zusätzliches Kontextmenü, wie zum Beispiel beim Aggregieren von Prozesselementen, notwendig ist.

F7 (Verzweigungsblock einfügen) Mit dieser Modellierungsfunktion wird ein neuer Verzweigungsblock in das Prozessmodell eingefügt. Dabei hat der Benutzer die Wahl zwischen einem AND- oder einem XOR-Verzweigungsblock.

Fast ein Drittel der Testpersonen in [23] entscheiden sich für eine symbolische Geste, indem sie zum Beispiel den Start- sowie den Endpunkt des Verzweigungsblocks mit einem rautenförmigen Symbol kennzeichnen. Innerhalb des Rauten-Symbols zeichnen sie ein Plus- oder ein X-Symbol ein, um zwischen einem AND- und XOR-Verzweigungsblock zu unterscheiden. Das Experiment ergibt allerdings im Wesentlichen ein recht ausgeglichenes Ergebnis, da genauso viele Testpersonen Menü-basierte Lösungen als auch physikalische Gesten bevorzugen.

Das finale Gesten-Set in [23] beinhaltet eine Menü-basierte Lösung für diese Modellierungsfunktion. Der Benutzer führt zunächst eine Strich-Geste aus. Der Startpunkt der Geste markiert den öffnenden Verzweigungsknoten und der Endpunkt entsprechend den schließenden Verzweigungsknoten. Alle Prozesselemente, die sich innerhalb der beiden Gestenpunkte befinden, sind Bestandteil des neuen Verzweigungsblocks. Nach Gestenausführung öffnet sich ein Kontextmenü, mit dem ein Verzweigungstyp ausgewählt werden kann.

Einige Testpersonen in [23] haben sich aber auch für ein Rechteck-Symbol entschieden, um die entsprechenden Prozesselemente zu selektieren. Da die Funktion F1 (Element auswählen) ebenfalls dieses Symbol nutzt, ist es naheliegend, dieses Symbol für das Einfügen neuer Verzweigungsblöcke anstelle der Strich-Geste zu verwenden. In einem Kontextmenü, das nach

dem Ausführen der symbolischen Geste erscheint, kann der Benutzer den gewünschten Verzweigungstyp auswählen (siehe Abb. 3.9).

F8 (Verzweigung einfügen) In einen Verzweigungsblock können weitere Verzweigungen eingefügt werden. Intuitiv wird von den meisten Benutzern eine Strich-Geste zwischen den beiden Verzweigungsknoten eines Blocks gezeichnet [23]. Das Problem dieser Geste ist jedoch, dass beide Verzweigungsknoten auf dem Bildschirm sichtbar sein müssen, was sich bei einem sehr großen Prozessmodell als schwierig erweist. Beim Verkleinern eines Prozessmodells besteht wiederum das Problem, dass die Verzweigungsknoten zu klein sind, um sie präzise auszuwählen. Für diese Modellierungsfunktion wird daher eine Geste gewählt, die nur den öffnenden Knoten eines Verzweigungsblocks benötigt. Es handelt sich hierbei um eine symbolische Geste in Form eines nach oben gerichteten Pfeils (siehe Abb. 3.11). Der Startpunkt „S“ dieses Pfeils liegt auf dem öffnenden Knoten des Verzweigungsblocks. Der schließende Verzweigungsknoten wird nach Gestenausführung automatisch von der Anwendung ermittelt.

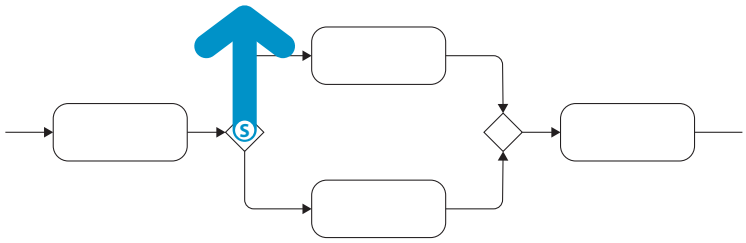

Abbildung 3.11: Verzweigung einfügen.

F9 (Schleife einfügen) Mit dieser Modellierungsfunktion kann in das Prozessmodell eine neue Schleife eingefügt werden, um bestimmte Prozesselemente wiederholt auszuführen. [23] besitzt keine experimentelle Untersuchung zu dieser Modellierungsfunktion, dennoch wird eine Geste vorgeschlagen. Es handelt sich dabei um ein wellenförmiges Symbol, das an drei Stellen den Sequenzfluss schneidet. Der erste und der letzte Schnittpunkt definieren den Start- bzw. Endpunkt der Schleife. Zwischen diesen beiden Punkten befinden sich die Prozesselemente, die wiederholt werden sollen. Da diese vorgeschlagene Geste nicht intuitiv genug ist, wird für diese Modellierungsfunktion eine andere Geste bevorzugt.

Der Benutzer zeichnet die Schleife direkt in das Prozessmodell ein (siehe Abb. 3.12). Die rautenförmigen Symbole für die Verzweigungsknoten werden jedoch weggelassen. Die Geste ähnelt auf diese Weise einem halben

Rechteck. Die Endpunkte dieser Geste starten und enden direkt auf den Prozesselementen, die noch in die Schleife miteinbezogen werden sollen. Nach Gestenausführung befinden sich die entsprechenden Prozesselemente innerhalb des Schleifenblocks.

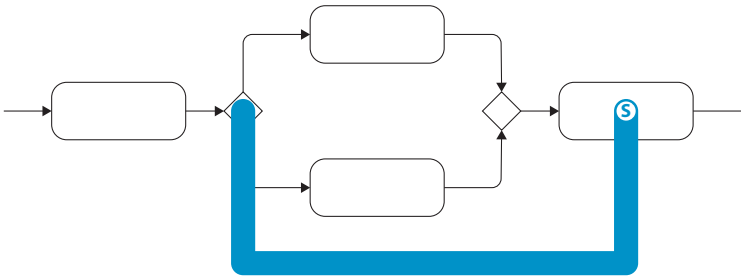

Abbildung 3.12: Schleife einfügen.

F10 (Synchronisationskante einfügen) Um die Abhängigkeit zwischen zwei Aktivitäten innerhalb eines AND-Verzweigungsblocks zu bestimmen, kann eine Synchronisationskante zwischen diesen Prozesselementen eingefügt werden. Für diese Modellierungsfunktion existiert keine experimentelle Untersuchung. Sie ähnelt jedoch der Modellierungsfunktion zum Einfügen einer Lese- oder Schreibkante. In [23] hat sich herausgestellt, dass 57,69% der Testpersonen mithilfe einer Strich-Geste eine Linie zwischen Datenelement und Aktivität ziehen. Die Richtung der Linie bestimmt, ob es eine Lese- oder Schreibkante ist. Diese Geste lässt sich ebenfalls für die Synchronisationskante verwenden (siehe Abb. 3.13).

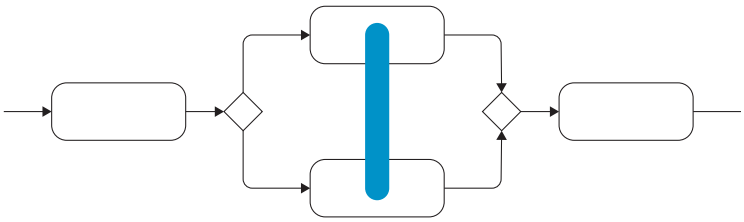

Abbildung 3.13: Synchronisationskante einfügen.

F11 (Subprozess anzeigen) Ein Subprozess in kollabierter Version kann in pro-View wieder vollständig angezeigt werden. Zu dieser Modellierungsfunktion fand bisher keine experimentelle Untersuchung statt. Die Überlegungen zu dieser Modelllierungsfunktion beinhalteten die Implementierung einer Swipe- oder einer Pinch-Geste, die auf dem Subprozess ausgeführt wird. Schlussendlich wird eine Hold-Geste verwendet, da der kollabierte Subprozess mit

dieser Geste präziser ausgewählt werden kann. Der Subprozess wird hierbei für eine bestimmte Zeitdauer berührt, um die Modellierungsfunktion auszulösen und den Subprozess aufzuklappen (siehe Abb. 3.14a).

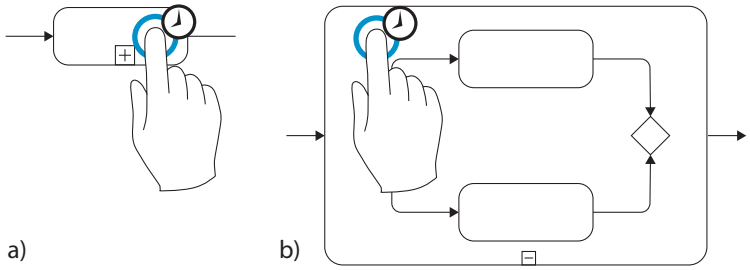

Abbildung 3.14: Subprozess anzeigen a) und verbergen b).

F12 (Subprozess verbergen) Diese Modellierungsfunktion ermöglicht die Überführung eines expandierten Subprozesses in seine kollabierte Version. Wie schon bei F11 (Subprozess anzeigen), wird auch an dieser Stelle wieder die Hold-Geste verwendet (siehe Abb. 3.14b).

F13 (Subprozess auflösen) Ein bestehender Subprozess kann in seiner expandierten Version mit dieser Modellierungsfunktion aufgelöst werden, sodass die enthaltenen Prozesselemente wieder fester Bestandteil des gesamten Prozessmodells sind.

[23] untersucht nur das Bilden eines Subprozesses, d.h. das Aggregieren von Prozesselementen. Im finalen Gesten-Set ist jedoch ein Gestenvorschlag für die Auflösen-Funktion enthalten. Der Benutzer zeichnet hierbei um den expandierten Subprozess ein Rechteck, um ihn auszuwählen. Im sich anschließend öffnenden Kontextmenü kann die entsprechende Modellierungsfunktion zum Auflösen des Subprozesses ausgewählt werden.

Eine weitere Alternative ist in [31] enthalten. Die Mehrheit der Testpersonen des Experiments zeichnen beim Rückgängig machen bestimmter Aktionen ein Zickzack-Symbol auf den Bildschirm (siehe Abb. A.1m). Da diese Geste experimentell ermittelt ist, wird sie für die Implementierung bevorzugt (siehe Abb. 3.15).

Zu beachten ist, dass sich der Startpunkt der Geste innerhalb des Subprozesses befinden muss, da das Zickzack-Symbol für weitere Modellierungsfunktionen eingesetzt werden könnte. Der restliche Teil der Geste kann sich auch außerhalb des Subprozesses befinden. Ein Vorteil dieser symbolischen Geste ist, dass die entsprechende Modellierungsfunktion sofort nach Gestenausführung ausgelöst wird und kein weiterer Schritt über ein Kontext-

menü notwendig ist.

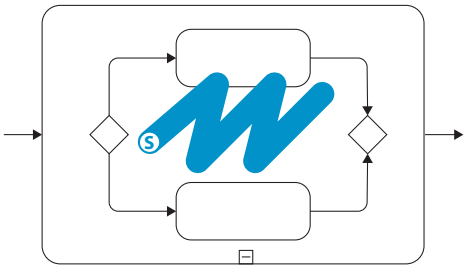

Abbildung 3.15: Subprozess auflösen.

F14 (Prozesssicht erstellen) Aus einem Prozessmodell können weitere Prozesssichten extrahiert werden, indem die entsprechenden Prozesselemente ausgewählt werden. Diese Modellierungsfunktion ist beinahe identisch zur Modellierungsfunktion F5 (Elemente aggregieren). Daher wird für das Erstellen von neuen Prozesssichten dieselbe Geste verwendet.

Der Benutzer wählt mit einem Rechteck die Prozesselemente aus, die in der neuen Prozesssicht enthalten sein sollen. In einem sich öffnenden Kontextmenü kann die Modellierungsfunktion zum Erstellen der Prozesssicht ausgewählt werden (siehe Abb. 3.9).

F15 (Datenelement einfügen) In ein Prozessmodell können nicht nur neue Aktivitäten, sondern auch neue Datenelemente eingefügt werden. [23] untersucht diese Modellierungsfunktion und liefert ein ausgeglichenes Ergebnis zwischen symbolischen und physikalischen Gesten. 38,46% der Testpersonen entscheiden sich bei dem Experiment für das Zeichnen eines Rechtecks oder Kreises auf eine freie Fläche des Hintergrunds. 46,15% der Testpersonen, bei denen es sich größtenteils um erfahrene Testpersonen handelt, favorisieren Menü-basierte Lösungen, d.h. sie setzen mittels Tap-Geste und Kontextmenü ein neues Datenelement ein.

Das finale Gesten-Set des Experiments schlägt das Zeichnen eines Rechtecks abseits des Prozessmodells, d.h. auf freiem Hintergrund, vor. Dieser Vorschlag wird übernommen (siehe Abb. 3.16). Die Geste erzeugt sofort nach ihrer Ausführung ein neues Datenelement, das der Benutzer unter Einsatz einer virtuellen Tastatur benennen kann.

F16 (Hilfe aufrufen) Ist eine Geste dem Benutzer nicht bekannt, kann er in einem Hilfe-Dialog nach dieser Geste suchen. Für den Aufruf dieses Hilfe-Dialogs ist ebenfalls eine Geste notwendig.

Benutzer verbinden mit einer Hilfe in der Regel das Fragezeichen-Symbol.

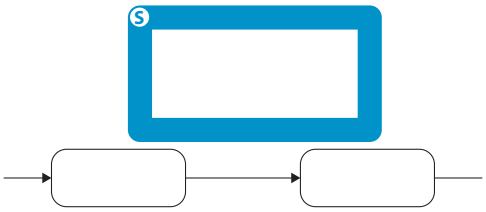

Abbildung 3.16: Datenelement einfügen.

Daher ist die naheliegende Lösung, eine symbolische Geste zum Aufrufen des Hilfe-Dialogs zu verwenden. Dies wird auch von [31] bestätigt, in dem die meisten Testpersonen ein Fragezeichen ohne unteren Punkt zeichnen (siehe Abb. A.1i). Das Fragezeichen-Symbol ist ebenfalls fester Bestandteil vieler interaktiver Systeme auf Tablet-PCs, wie Abb. 3.2g zeigt.

Verwendet wird somit für diese Funktion ebenfalls eine symbolische Geste in Form eines Fragezeichens ohne den unteren Punkt, da der Benutzer seinen Finger nicht absetzen darf (siehe Abb. 3.17). Die Geste kann an einer beliebigen Stelle im Bearbeitungsbereich ausgeführt werden.

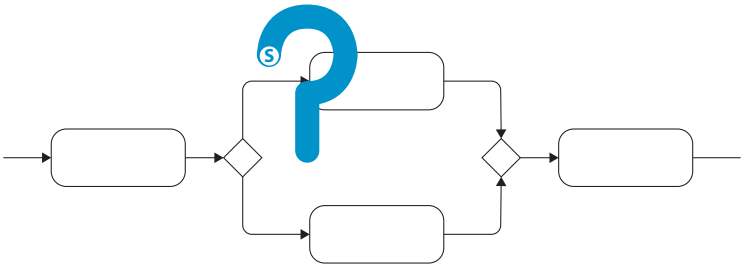

Abbildung 3.17: Hilfe aufrufen.

F17 (Rückgängig machen) Ausgeführte Bearbeitungsschritte können, wie in vielen anderen Anwendungen auch, wieder rückgängig gemacht werden. Eine experimentelle Untersuchung dieser Modellierungsfunktion ist in [31] enthalten. Die Mehrheit der Testpersonen dieses Experiments entscheidet sich die Modellierungsfunktion mit einem Zickzack-Symbol auszulösen (siehe Abb. A.1m). An dieser Stelle wird eine symbolische Geste in Form eines Zickzack-Symbols gewählt. Zu beachten ist, dass die Geste abseits des Prozessmodells auf freiem Hintergrund ausgeführt werden muss (siehe Abb. 3.18), da F13 (Subprozess auflösen) diese Geste bereits nutzt.

F18 (Lese- und Schreibkante einfügen) Zwischen einer Aktivität und einem Datenelement kann eine Lese- oder Schreibkante eingefügt werden, damit die

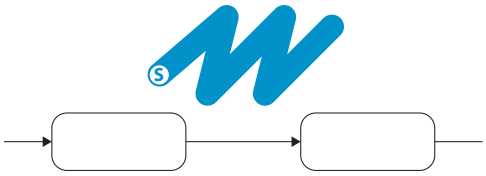

Abbildung 3.18: Rückgängig machen.

Aktivität lesenden bzw. schreibenden Zugriff auf das Datenelement hat.

Mit 57,69% und somit mehr als die Hälfte der Testpersonen in [23] entscheiden sich eine Strich-Geste zwischen Aktivität und Datenelement auszuführen. Die Richtung der Bewegung bestimmt dabei den Kantentyp. Für Lesekanten zum Beispiel führen die Testpersonen eine Bewegung vom Datenelement zur Aktivität aus. Aufgrund dieses Ergebnisses wird diese Geste übernommen (siehe Abb. 3.19).

Zu beachten ist, dass sich die Endpunkte des Strichs genau auf den entsprechenden Prozesselementen befinden, da die Modellierungsfunktion sonst nicht ausgelöst werden kann.

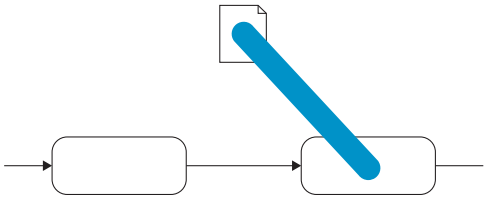

Abbildung 3.19: Lese- und Schreibkante einfügen.

4 Analyse von Multi-Touch-Implementierungen

Die zur Verfügung gestellten UI-Komponenten in Vaadin reagieren standardmäßig auf keine Touch-Events. Es können allerdings clientseitige Widgets erzeugt werden, die aufgrund der Verwendung von GWT auf Touch-Events reagieren können. Vaadin bietet auf der eigenen Webseite Add-ons an, von denen einige Touch-Events unterstützen. Diese Add-ons sind TouchScroll [26], TouchMenu [5], JSlider [20] und das Vaadin TouchKit [7].

Die ersten drei Add-ons bieten jeweils eine einzelne UI-Komponente an, die ausschließlich auf Scroll-Gesten reagiert. Eine von der Komponente losgelöste Nutzung der Geste ist nicht möglich. Darüber hinaus reicht die einzelne Scroll-Geste nicht für eine Erweiterung des proView-Frameworks aus. Einzelheiten zum TouchKit-Add-on von Vaadin werden in Abschnitt 4.1 diskutiert.

Vaadin unterstützt auch den Einsatz von JavaScript-Bibliotheken mit denen Multi-Touch-Funktionalität realisiert werden kann. Abschnitt 4.2 stellt diese Bibliotheken vor. Schließlich beinhaltet Abschnitt 4.3 ein Fazit zu den vorgestellten Multi-Touch-Implementierungen.

4.1 Vaadin TouchKit

Das Vaadin TouchKit-Add-on ist von Vaadin Ltd. entwickelt worden und dient der Erstellung von Web-Anwendungen mit dem Look and Feel nativer Anwendungen auf mobilen Geräten [7]. So beinhaltet es beispielsweise spezielle UI-Komponenten wie Buttons oder Navigationsleisten, die eine optische Ähnlichkeit mit Komponenten einer nativen iPhone-Anwendung haben. Zudem unterstützt die SwipeView-Komponente des TouchKits die Swipe-Geste. Mit dieser Geste kann zwischen verschiedenen Seiten einer Web-Anwendung gewechselt werden, indem diese nach links oder rechts „gewischt“ werden. Dies ist allerdings die einzige Geste, die das TouchKit-Add-on unterstützt.

Es werden verschiedene Browser-Eigenschaften unterstützt, wie Geolocation, Home Screen Launching, Splash Screen und Web App Mode. Zusätzlich bietet das Add-on einen Offline-Modus für Web-Anwendungen an, bei dem es sich um eine spezielle Vaadin-Benutzerschnittstelle handelt, die im Cache eines Browsers abgelegt wird und bei einer fehlenden Netzwerkverbindung zur Laufzeit oder beim Start der Web-Anwendung aktiv wird.

Ein wesentlicher Nachteil dieses Add-ons ist, dass es nur mit Browsern kompatibel ist, die für das Rendering von HTML und JavaScript die freie Browser-Engine WebKit verwenden. Dies ist beispielsweise bei den Standardbrowsern von iPhone und Android der Fall. Andere Browser, wie zum Beispiel Mozilla Firefox, der die Browser-Engine Gecko verwendet, werden vom TouchKit-Add-on nicht unterstützt.

4.2 Multi-Touch-Bibliotheken für JavaScript

Das W3C beschreibt eine Touch-Events Spezifikation [24], die das Auslösen von Low-Level-Ereignissen beim Berühren von berührungsempfindlichen Bildschirmen beinhaltet. Zu beachten ist, dass es sich bei dieser Spezifikation momentan nur um einen W3C-Empfehlungsvorschlag (engl. *Proposed Recommendation*) handelt, der im Kontext der HTML5-Entwicklung entstanden ist. Touch-Events sind demzufolge noch keine endgültige W3C-Empfehlung (d.h. W3C-Standard), sind aber wie HTML5 bereits in den meisten Browsern implementiert. Die Ereignisse, die von der Touch-Events Spezifikation definiert werden sind:

TouchStart Dieses Ereignis wird ausgelöst, wenn der Multi-Touch-Bildschirm bzw. ein berührungsempfindliches DOM-Element mit einem Finger berührt wird. Beim Einsatz mehrerer Finger löst jedes dieser Finger ein separates Touch-Start-Event aus.

TouchEnd Verlässt ein Finger den Multi-Touch-Bildschirm bzw. das DOM-Element, so wird ein TouchEnd-Event ausgelöst. Dies schließt auch den Fall ein, wenn ein Finger über den Rand eines Multi-Touch-Bildschirms gezogen wird.

TouchMove Berührt ein Finger den Bildschirm und wird währenddessen an eine andere Position auf dem Bildschirm bzw. DOM-Element verschoben, löst dies während der Verschiebung mehrere TouchMove-Events aus. Die Anzahl dieser Ereignisse hängt dabei von der Implementierung der Touch-Events Spezifikation ab, sowie von der Hardware des Multi-Touch-Gerätes.

TouchCancel Das TouchCancel-Event wird ausgelöst, wenn der Finger während eines Touch-Events die HTML-Seite verlässt oder ein Touch-Event aus be-

stimmten Gründen vom Browser unterbrochen wird. Es wird ebenfalls ein TouchCancel-Event ausgelöst, wenn die vorkonfigurierte Maximalanzahl an Berührungspunkten überschritten bzw. hardwareseitig nicht bewältigt werden kann.

Listing 4.1 demonstriert die Verwendung des TouchStart-Events innerhalb eines HTML5-Dokuments. Dem `DIV`-Container mit der ID `panel` in Zeile 1 wird ein TouchStart-Listener hinzugefügt (Zeile 3). Dieses DOM-Element ist nun berührungsempfindlich und löst bei Berührung die Handler-Methode `handleStart` aus (Zeile 5), die anschließend die Position des Berührungspunktes in der Browser-internen Konsole ausgibt (Zeile 7).

Listing 4.1: Code-Beispiel des TouchStart-Events

```
1 <div id="panel">touch</div>
2 <script type="text/javascript">
3    panel.addEventListener("touchstart", handleStart, false);
4
5    function handleStart(evt) {
6       var touch = evt.touches[0];
7       console.log(touch.pageX + " " + touch.pageY);
8    }
9 </script>
```

Es existieren bereits eine Vielzahl an JavaScript-Bibliotheken die diese Touch-Events nutzen, um damit verschiedene Gesten-Implementierungen anzubieten. Diese Bibliotheken sind zum Beispiel QuoJS [29], HammerJS [27], Touchy [4] und jGestures [8]. In einigen Fällen ist eine zusätzliche Einbindung der jQuery-Bibliothek [10] in das HTML-Dokument notwendig.

Abhängig von der verwendeten JavaScript-Bibliothek werden unter anderem die Gesten Single-Tap, Double-Tap, Hold, 2x Fingers Tap, 2x Double-Tap, Swipe, Drag, Rotate und Pinch unterstützt. Symbolische Gesten werden allerdings von keinen genannten Bibliotheken unterstützt.

Das Vaadin-Framework bietet standardmäßig die Möglichkeit, ein clientseitiges Widget in JavaScript zu implementieren. Dies wiederum ermöglicht die Verwendung eines der zuvor vorgestellten JavaScript-Bibliotheken, um die dort bereits implementierten Gesten für die Erweiterung von proView zu nutzen. Eine Implementierung mit JavaScript hat zudem den Vorteil, dass während der Entwicklung ein kompilieren bzw. umwandeln von Java nach JavaScript entfällt. Die Nachteile sind allerdings, dass zusätzliche Kenntnisse in JavaScript notwendig sind, eine

bewährte Entwicklungsumgebung für JavaScript fehlt und die GWT-Bibliothek mit eventuell nützlichen Klassen und Widgets nicht zur Verfügung steht.

4.3 Fazit

Um das proView-Framework zu erweitern und Gesten-basierte Prozessmodellierung zu ermöglichen, werden zunächst Gesten-Implementierungen benötigt. Es existieren einige Add-ons für das Vaadin-Framework, die Gesten-Implementierungen anbieten. Die Anzahl der Gesten ist hier allerdings hauptsächlich auf nur eine Geste beschränkt. Diese einzelne Geste ist jedoch eng mit einer UI-Komponente verknüpft, sodass eine Nutzung der Geste ohne die UI-Komponente nicht möglich ist. Eine Alternative zu Vaadin-Add-ons sind JavaScript-Bibliotheken, die oft eine große Anzahl an Gesten-Implementierungen besitzen. Allerdings fehlt, wie bei den Vaadin-Add-ons, eine Implementierung von symbolischen Gesten und clientseitige Widgets müssen in JavaScript implementiert werden, das einige Entwicklungsschritte erschweren könnte.

Aufgrund dieser Kritikpunkte ist es sinnvoll ein eigenes Vaadin-Add-on zu entwickeln, das auf Touch-Events reagiert und alle benötigten Gesten inklusive symbolischer Gesten anbietet. Dieses Add-on kann später sowohl für das proView-Framework als auch für andere Vaadin-Projekte verwendet werden. Im nächsten Kapitel 5 wird das in dieser Arbeit entwickelte Vaadin-Add-on TouchPanel ausführlicher beschrieben.

5 Entwicklung eines Multi-Touch-Add-ons

In diesem Kapitel werden die Entwicklung und der Aufbau des *TouchPanel-Add-ons* beschrieben. Hierbei handelt es sich um ein erweitertes Vaadin-Panel, das auf Gesten-basierte Interaktionen reagiert. Wie jede andere Vaadin-Komponente auch, besteht das TouchPanel-Add-on aus einem clientseitigen Widget (siehe Abschnitt 5.2) und einer serverseitigen Komponente (siehe Abschnitt 5.3). Das Zusammenspiel zwischen Client- und Serverseite ist Thema von Abschnitt 5.1. Abschnitt 5.4 beinhaltet die Emulation von Touch-Events, damit ein Testen des Add-ons auch auf Geräten ohne Multi-Touch-Funktion möglich ist.

5.1 Aufbau des TouchPanel-Add-ons

Das clientseitige Widget ist eine Instanz der Klasse `TouchPanelWidget` und erweitert das Vaadin-Widget `VPanel` (siehe Abb. 5.1). Die Klasse `TouchPanel` erweitert dagegen die Vaadin-Komponente `Panel` und repräsentiert auf diese Weise die serverseitige Komponente des TouchPanel-Add-ons (siehe Abschnitt 5.3). Zu beachten ist, dass sich laut Vaadin-Spezifikation alle für das Widget relevanten Klassen in einem separaten Paket mit dem Namen `client` befinden müssen. Im Gegensatz dazu dürfen die serverseitigen Klassen überall abgelegt sein. Statische Ressourcen wie Bilder und Stylesheets werden in einem Paket-unabhängigen, öffentlichen Ordner abgelegt.

Die Kommunikation zwischen `TouchPanelWidget` und `TouchPanel` wird durch die Klasse `TouchPanelConnector` geregelt. Dieser Connector befindet sich im `client`-Paket und wird mittels Annotation `@Connect` mit dem serverseitigen `TouchPanel` verbunden. Auf diese Weise ist es möglich, den Zustand des Widgets sowie Ereignisse vom und zum Server zu synchronisieren.

Im `client`-Paket befindet sich zusätzlich ein sogenanntes `shared state`-Objekt, das dazu dient, Zustandsänderungen der serverseitigen Komponente automatisch an das Widget weiterzuleiten. Dieses Objekt ist eine Instanz der `TouchPanelState`-Klasse und unterstützt ausschließlich primitive Datentypen wie `String`, `Array`, sowie die Datenstrukturen `List`, `Map` und `Set`.

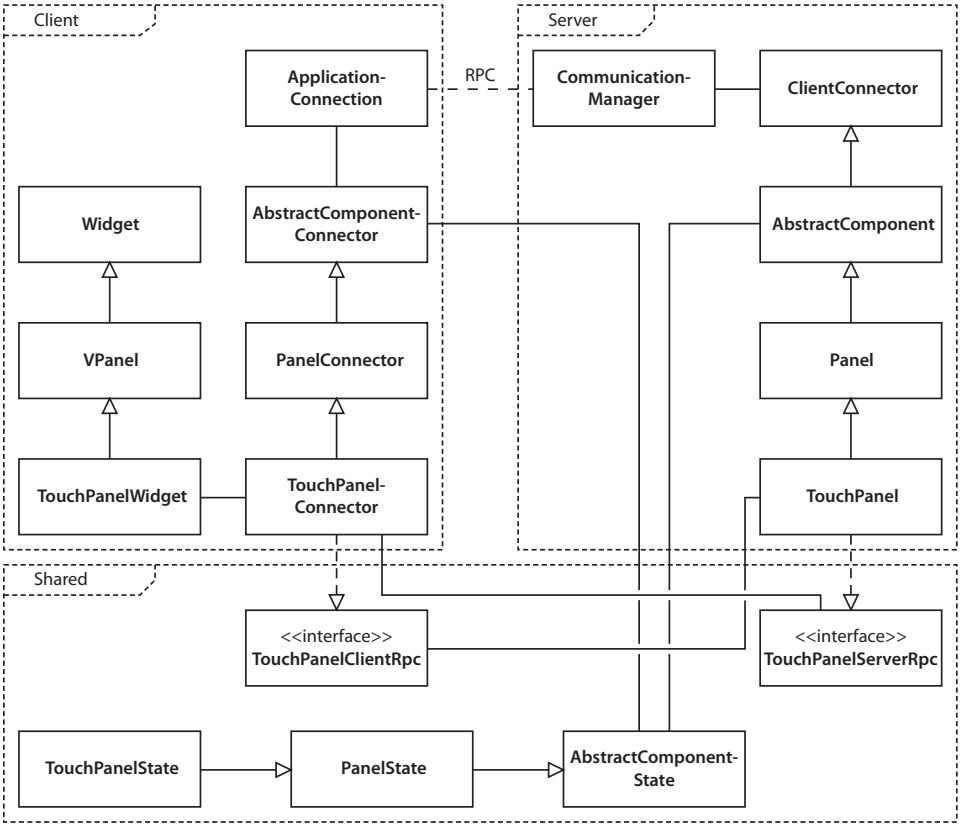

Abbildung 5.1: Klassendiagramm zum TouchPanel-Add-on.

Das `shared state`-Objekt wird ausschließlich vom Server an den Client gesendet. Das Senden von Zustandsänderungen vom Client an den Server ist mithilfe von RPC-Aufrufen möglich. Hierzu nutzt der `TouchPanelConnector` die Schnittstelle `TouchPanelServerRpc`, die sich mit der Methode `registerRpc` beim `TouchPanel-Connector` registriert. Diese Methode erbt der `TouchPanelConnector` vom `AbstractComponentConnector`. Implementiert wird diese Schnittstelle vom `TouchPanel`.

RPC-Aufrufe vom Server an den Client sind mit der Schnittstelle `TouchPanel-ClientRpc` möglich. Eine Implementierung dieser Schnittstelle ist in der Klasse `TouchPanelConnector` enthalten. Die Schnittstelle wird mit der `getRpcProxy`-Methode beim serverseitigen `TouchPanel` registriert, das die Methode vom `Client-Connector` erbt. Den Rest übernimmt der serverseitige `CommunicationManager`, den Vaadin zur Verfügung stellt.

Ebenfalls im TouchPanel-Add-on enthalten ist ein *Module Descriptor* mit dem Namen `TouchPanelWidgetSet.gwt.xml`. Dieser Descriptor beschreibt das vom Add-on verwendete Widget-Set, das bei einer Kompilierung von Java nach JavaScript

berücksichtigt werden soll. Andere Vaadin-Projekte, die das TouchPanel-Add-on verwenden, müssen eine Referenz zu diesem Module Descriptor in ihrem eigenen Module Descriptor angeben. Bereits vorkompilierte Widget-Sets befinden sich im Ordner `WebContent/VAADIN/widgetsets` des TouchPanel-Add-ons.

Das `client`-Paket enthält noch weitere Unterpakete, beispielsweise das `gestures`-Paket, das verschiedene Gesten-Implementierungen enthält.

5.2 Clientseitiges TouchPanel-Widget

Im Folgenden wird die Implementierung des TouchPanel-Widgets näher erläutert, das auf Clientseite zur Ausführung kommt. Im Mittelpunkt dieses Widgets steht die Klasse `TouchPanelWidget`, die Touch-Events mit der Methode `onBrowserEvent` entgegennimmt und an initialisierte Klassen zur Weiterverarbeitung delegiert. Bei diesen Klassen handelt es sich um Gesten-Implementierungen, die in den nächsten Abschnitten genauer beschrieben werden (siehe Abb. 5.2).

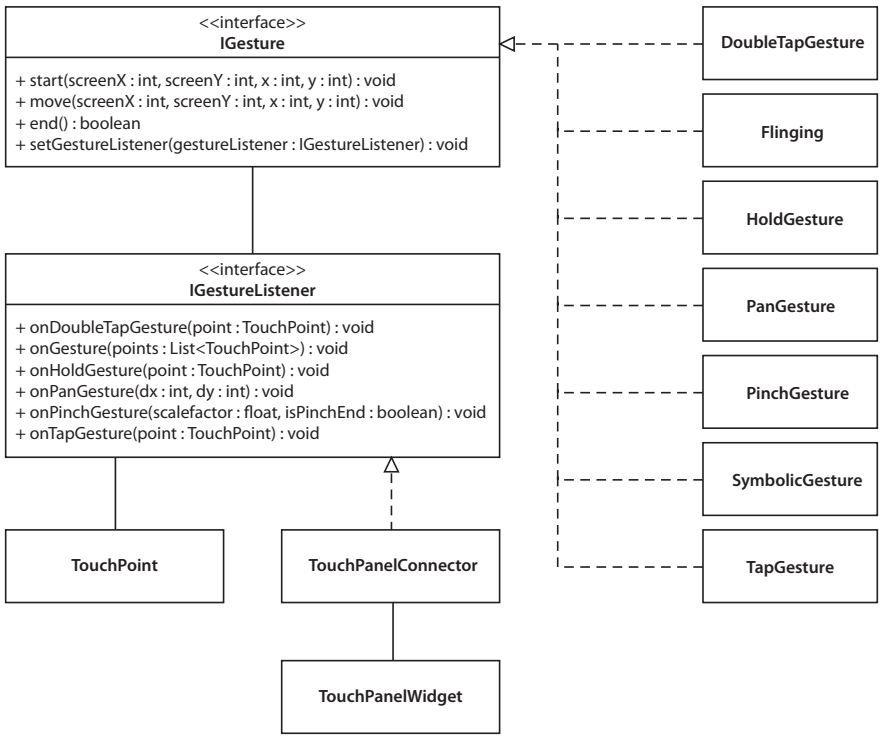

Abbildung 5.2: Klassendiagramm mit Gesten-Implementierungen.

Jede dieser Klassen (d.h. DoubleTapGesture, Flinging, HoldGesture, PanGesture, PinchGesture, SymbolicGesture und TapGesture) implementiert die Schnittstelle

`IGesture`, um auf diese Weise sicherzustellen, dass in jeder Klasse die Methoden `start`, `move`, `end` und `setGestureListener` vorhanden sind. Die Methoden `start` und `end` werden aufgerufen, sobald ein Finger den Bildschirm berührt bzw. diesen wieder verlässt. Bewegt sich der Finger während einer Berührung, so wird die Methode `move` aufgerufen. Die Position eines Fingers auf dem Bildschirm sowie auf dem Element, das dem TouchPanel untergeordnet ist, wird den Methoden `start` und `end` als Parameter übergeben. Mithilfe des GestureListeners, der mit der Methode `setGestureListener` initialisiert wird, können Ergebnisse der implementierten Methoden an den `TouchPanelConnector` weitergegeben werden, der für RPC-Aufrufe zwischen Client und Server zuständig ist.

Zu beachten ist, dass TouchMove-Events pro Sekunde sehr oft auftreten können und sie daher anders behandelt werden müssen als TouchStart- oder TouchEnd-Events, um das Multi-Touch-Gerät bzw. den Browser nicht zu überlasten. Der nächste Abschnitt 5.2.1 beinhaltet nähere Informationen hierzu.

5.2.1 Behandlung von TouchMove-Events

TouchMove-Events werden zum Beispiel bei einer Verschiebung eines Prozessmodells verwendet. Aufgrund einer Überlastung des Rechners, kann es passieren, dass einzelne Ereignisse verworfen werden, was zu einer ruckeligen Verschiebung führt. Auch spielen TouchMove-Events bei der Gestenerkennung eine wichtige Rolle, da bei einer erhöhten Auslöserate der Ereignisse mehr Berührungspunkte erzeugt werden, die für eine genaue Formung einer Geste wichtig sind. Es ist deshalb notwendig ein spezielles Verarbeitungsintervall zu nutzen, um die Überlastung eines Rechners zu reduzieren.

Jeder Browser führt in einem speziellen Intervall eine Repaint-Funktion aus. Dieses Zeichnungsintervall ist an die Bildwiederholfrequenz des eingesetzten Bildschirms angepasst. Unter Verwendung von RequestAnimationFrame (RAF) [21] ist es möglich TouchMove-Events an das Zeichnungsintervall zu koppeln, damit diese Browser-gesteuert mit einer angepassten Rate verarbeitet werden. Dies würde die Überlastung des Rechners reduzieren und beispielsweise zu einer weicheren Bewegung beim Verschieben führen. In Abbildung 5.3 ist zu erkennen, dass auf unterschiedlichen Geräten durch den Einsatz von RequestAnimationFrame sogar mehr TouchMove-Events verarbeitet werden.

Unterstützt wird die RequestAnimationFrame-Funktion durch den `AnimationScheduler` von GWT. Das `TouchPanelWidget` implementiert hierzu die Schnittstelle `AnimationCallback` und die damit verbundene Methode `execute`, die aufgerufen

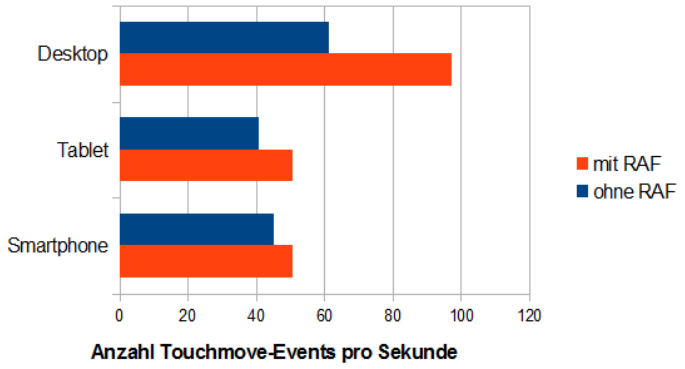

Abbildung 5.3: RequestAnimationFrame (RAF) im Einsatz.

wird, sobald der Browser durch einen Callback die Ausführung des nächsten Ereignisses freigibt. Die Anfrage nach der Freigabe des nächsten Ereignisses geschieht mit der Methode `requestAnimationFrame` des `AnimationSchedulers`. Innerhalb der Methode `execute` wird ein TouchMove-Event verarbeitet.

Der `AnimationScheduler` bzw. RequestAnimationFrame hat im Gegensatz zu einer gewöhnlichen Timer-Implementierung den Vorteil, dass nebenläufige Ereignisse durch den Browser optimiert werden und dass die Verarbeitungsschleife angehalten wird, sobald der Browser-Tab mit der Web-Anwendung nicht sichtbar ist. Dies resultiert in einer reduzierten Nutzung von CPU, GPU und Speicher. Gleichzeitig wird auf diese Weise der Akku von mobilen Geräten geschont.

5.2.2 Symbolische Gesten

Symbolische Gesten werden mithilfe der Klasse `SymbolicGesture` realisiert. Die Methode `start` erzeugt bei einem TouchStart-Event ein neues `TouchPoint`-Objekt (siehe Abb. 5.4). Weitere `TouchPoint`-Objekte erzeugt die Methode `move`, sobald der Benutzer TouchMove-Events auslöst. Alle `TouchPoint`-Objekte zusammen ergeben eine symbolische Geste. Ein ausgelöstes TouchEnd-Event führt dazu, dass die `TouchPoint`-Objekte unter Verwendung der Methode `onGesture` des `IGesture-Listeners` als Parameter an die Klasse `TouchPanelConnector` übergeben werden. Der `TouchPanelConnector` leitet die `TouchPoint`-Objekte anschließend per RPC-Methode `gestureExecuted` an den Server zur Gestenerkennung weiter.

Erzeugt wird eine symbolische Geste erst dann, wenn sich der Finger weiterbewegt und eine bestimmte Distanz zum Startpunkt besitzt. Dies verhindert, dass symbolische Gesten bereits bei Ausführung einer Tap- oder Double-Tap-Geste erzeugt werden. Wird der Finger über den Rand des Multi-Touch-Bildschirms gezogen,

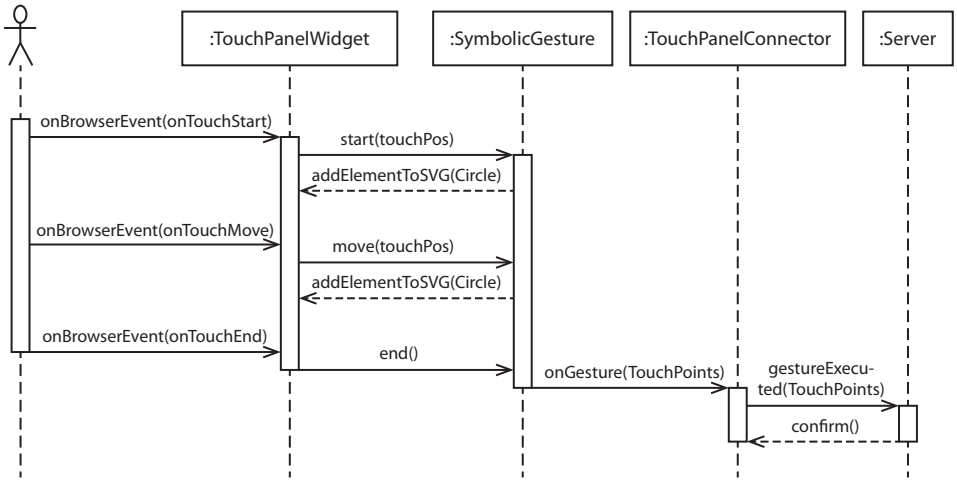

Abbildung 5.4: Übertragen einer symbolischen Geste an den Server.

löst dies ein TouchCancel-Event aus, das die symbolische Geste abbricht.

Die `TouchPoint`-Objekte einer symbolischen Geste werden auf einem dem Touch-Panel untergeordneten Canvas als cyanfarbige SVG-Kreise visualisiert. Dies hat den Vorteil, dass ein Nutzer Feedback über die korrekte Darstellung einer symbolischen Geste erhält und auf mögliche Fehleingaben besser reagieren kann. Die Klasse `SymbolicGesture` fügt dem Canvas die SVG-Kreise mithilfe der Methode `addElementToSVG` hinzu.

5.2.3 Tap-Geste

Die Tap-Geste ist ein einfaches Antippen des Multi-Touch-Bildschirms mit einem Finger. Durch das Antippen wird die Methode `onTapGesture` des `IGestureListeners` aufgerufen, der als Parameter ein `TouchPoint`-Objekt mit den Koordinaten des Berührungspunktes übergeben bekommt. Der `TouchPanelConnector`, der die Methode `onTapGesture` implementiert, leitet den Berührungspunkt per RPC-Methode `tapGestureExecuted` an den Server weiter. Zu beachten ist, dass die Berührung des Bildschirms nicht länger als eine Sekunde dauern darf, da anstelle einer Tap-Geste sonst eine Hold-Geste ausgeführt wird (siehe Abschnitt 5.2.7).

5.2.4 Double-Tap-Geste

Wird der Bildschirm an einer bestimmten Stelle zweimal hintereinander mit einem Finger angetippt, handelt es sich um eine Double-Tap-Geste. Das zweima-

lige Antippen muss innerhalb einer Sekunde erfolgen, damit es zu einer erfolgreichen Ausführung kommt. Anschließend wird die Methode `onDoubleTapGesture` des `IGestureListeners` ausgeführt, die den Berührungspunkt als `TouchPoint`-Objekt an den `TouchPanelConnector` weiterleitet. Dieser sendet die Informationen per RPC-Methode `doubleTapGestureExecuted` an den Server weiter. Zu beachten ist, dass bei einer Double-Tap-Geste ebenfalls eine Tap-Geste ausgeführt wird.

5.2.5 Pan-Geste

Mit der Pan-Geste ist es möglich den Bearbeitungsbereich in proView zu verschieben. Der Benutzer verschiebt hierzu lediglich seinen Finger auf dem Bildschirm. Anders als beim Scrolling wird der Bearbeitungsbereich beim Panning gleichzeitig in x- und y-Richtung verschoben. Der Einsatz mehrerer Finger beim Panning ist ebenfalls möglich. In diesem Fall wird ein Mittelwert zwischen den Berührungspunkten berechnet.

Die Klasse `PanGesture` ruft während der Gestenausführung die Methode `onPanGesture` des `IGestureListeners` auf. Als Parameter werden die zurückgelegten Distanzen (dx, dy) des Fingers in Pixel übergeben, die zwischen den einzelnen, ausgelösten TouchMove-Events ermittelt wurden. Der Connector leitet die Distanzen an den Server weiter. Gleichzeitig werden die Distanzen clientseitig direkt an die horizontale sowie vertikale Scrollleiste des Bearbeitungsbereichs übergeben, damit eine sofortige Verschiebung des Bearbeitungsbereichs stattfindet.

5.2.5.1 Flinging

Unter Flinging versteht man das Verschieben eines Elements mit abnehmender Geschwindigkeit, nachdem der Finger den Bildschirm nach Ausführen einer Pan-Geste bereits verlassen hat. So ist es möglich mit nur einem kurzen Wisch über den Bildschirm ein Prozessmodell bzw. den Bearbeitungsbereich von proView um eine größere Distanz zu verschieben. Je schneller der Benutzer die Pan-Geste ausführt, desto größer die Distanz beim Flinging, die innerhalb der vier Sekunden nach dem Start des Flinging zurückgelegt wird. Aus diesem Grund wird zunächst die Pan-Geschwindigkeit in Pixel pro Sekunde berechnet (siehe Algorithmus 1, Zeile 2 und 3). Hierzu werden die einzelnen vertikalen sowie horizontalen Abstände, die sich innerhalb zweier TouchMove-Events ergeben, ermittelt. Der durchschnittliche Wert dieser Abstände wird durch die durchschnittliche Ausführungszeit der Pan-Geste dividiert.

Algorithm 1 Flinging Algorithmus

```
1:  function START
2:      velocityX ← (avgDx * 1000)/avgT
3:      velocityY ← (avgDy * 1000)/avgT
4:  end function

5:  function FLING
6:      distX ← DISTANCE_TIME_FACTOR * velocityX
7:      distY ← DISTANCE_TIME_FACTOR * velocityY
8:      SCROLLBY(elem, distX, distY, duration)
9:  end function

10: function SCROLLBY(elem, distX, distY, duration)
11:     now ← curTime
12:     curMs ← MIN(duration, now − startTime)
13:     x ← EASEOUT(curMs, 0, distX, duration)
14:     y ← EASEOUT(curMs, 0, distY, duration)
15:     PANBY(elem, −(x − lastX), −(y − lastY))
16:     lastX ← x
17:     lastY ← y
18:     if curMs < duration then
19:         isFling ← 1
20:     else
21:         isFling ← 0
22:     end if
23: end function

24: function EASEOUT(t, start, end, duration)
25:     t ← t/duration − 1
26:     return end * (t^3 + 1) + start
27: end function
```

Die Geschwindigkeit der Pan-Geste wird anschließend mit dem Faktor `DISTANCE_TIME_FACTOR` multipliziert (Zeile 6 und 7), um größere Distanzen beim Flinging zu erreichen.

Eine Verlangsamung der Verschiebung beim Flinging kann mittels verschiedener Übergangsfunktionen (engl. *Easing Functions*) realisiert werden, die durch die Schnittstelle `Easing` im Paket `easing` zur Verfügung gestellt werden. Das Flinging verwendet eine kubische `EaseOut`-Funktion, die von der Klasse `Cubic` implementiert wird. Diese Funktion wird innerhalb der Methode `scrollBy` aufgerufen (Zeile 13, 14 und 24), die ähnlich zu den Touch-Events vom `AnimationScheduler` ausgeführt wird. Als ersten Parameter erhält die `EaseOut`-Funktion einen Differenzwert aus dem Startzeitpunkt des Flingings und dem aktuellen Zeitpunkt in Millisekunden. Sobald dieser Differenzwert größer als die eigentliche Dauer der `Ease Out`-Funktion ist, so ist das Flinging beendet. Das bedeutet, die Variable `isFling` (Zeile

19 und 21) erhält nach vier Sekunden den Wert 0, um das Flinging als beendet zu kennzeichnen. Der zweite und dritte Parameter der `Ease Out`-Funktion beinhaltet den Startpunkt und die Distanz der Verschiebung. Der letzte Parameter enthält die Dauer des Flinging. Als Ergebnis der `Ease Out`-Funktion werden die neuen Positionswerte ausgegeben, die anschließend an die Scrollleiste des Bearbeitungsbereichs übergeben werden.

5.2.6 Pinch-Geste

Der Benutzer führt eine Pinch-Geste aus, wenn er zwei Finger auf dem Multi-Touch-Bildschirm gleichzeitig aufeinander zubewegt oder voneinander weg bewegt. Diese Geste findet oft Anwendung in Bildbetrachtungsprogrammen auf mobilen Geräten, um Bilder zu vergrößern bzw. zu verkleinern. In der proView-Erweiterung wird sie daher auf gleiche Weise zur Vergrößerung und Verkleinerung von Prozessmodellen verwendet.

Befinden sich genau zwei Finger auf dem Multi-Touch-Bildschirm, löst dies die `start`-Methode der Klasse `PinchGesture` aus, die mittels dem Satz von Pythagoras die Distanz zwischen den beiden Fingern ermittelt. Diese Start-Distanz ist während eines TouchMove-Events für die Berechnung eines Skalierungsfaktors notwendig (siehe Listing 5.1), mit dessen Hilfe eine Unterscheidung zwischen Vergrößerung und Verkleinerung stattfinden kann.

Listing 5.1: Berechnung des Skalierungsfaktors

```
1 newScalefactor = distance / startDistance;
```

Besitzt der Skalierungsfaktor einen Wert kleiner als 1, so handelt es sich hierbei um eine Verkleinerung. Im Gegensatz dazu signalisieren Werte größer als 1 eine Vergrößerung. Der Wert 1 ist ein neutraler Wert, d.h. es findet keine Skalierung statt.

Die Methode `onPinchGesture` des `IGestureListeners` erhält den Skalisierungsfaktor als Parameter zugewiesen. Des Weiteren erhält sie als Parameter einen boolschen Wert `isPinchEnd`, der bei Auslösung eines TouchEnd-Events das Ende einer Pinch-Geste kennzeichnet. Dies ist später bei der Erweiterung von proView relevant (siehe Abschnitt 6.5). Die `onPinchGesture`-Methode kommt erst zur Ausführung, wenn während eines TouchMove-Events die Differenz zwischen altem und neuem Skalierungsfaktor größer als 0,1 ist. Dies reduziert die Anzahl an RPC-Aufrufen, die durch die `onPinchGesture`-Methode und dem `TouchPanelConnector` angestoßen werden.

5.2.7 Hold-Geste

Berührt der Benutzer länger als eine Sekunde den Multi-Touch-Bildschirm an einer Stelle, so handelt es sich hierbei um eine Hold-Geste. Eine Implementierung dieser Geste befindet sich in der Klasse `HoldGesture`. Der Finger darf sich dabei nicht weiter als 15 Pixel vom Startpunkt entfernen, da dies sonst zum Abbruch der Geste führt. Der Einsatz eines weiteren Fingers führt ebenfalls zum Abbruch. Bei einer erfolgreichen Ausführung der Hold-Geste wird die Methode `onHoldGesture` des `IGestureListeners` aufgerufen. Diese Methode erhält als Parameter den Berührungspunkt als `TouchPoint`-Objekt, der somit an die Klasse `TouchPanelConnector` übergeben wird.

5.3 Serverseitige TouchPanel-Komponente

Die serverseitige TouchPanel-Komponente nimmt vom Client gesendete, Gesten-basierte Daten entgegen und verarbeitet sie entsprechend ihrer Logik weiter. Hierzu wird ein Gestenerkenner (siehe Abschnitt 5.3.1) verwendet, der die Daten auswertet und ausgeführte Symbol- sowie Strich-Gesten identifiziert. Eine Implementierung des Gestenerkenners auf Clientseite ist ebenfalls möglich, ist allerdings für mobile Client-Geräte mit niedriger Rechenleistung ungeeignet. Deshalb muss der Gestenerkenner auf einem leistungsstärkeren Server ausgeführt werden. Nach einer erfolgreichen Erkennung der Geste wird ein entsprechendes Gesten-Event erzeugt und an die Klassen weitergereicht, die den in Abschnitt 5.3.2 vorgestellten Gesten-Listener implementiert haben.

5.3.1 $1 Gesture Recognizer

Bei dem $1 Gesture Recognizer handelt es sich um einen Algorithmus, der verschiedene symbolische Gesten erkennen kann [30]. In einem Experiment, das von den Entwicklern durchgeführt worden ist, hat sich herausgestellt, dass der $1 Gesture Recognizer eine nahezu identische Erkennungsgenauigkeit liefert, wie die beiden wesentlich komplexeren Gestenerkenner *Dynamic Time Warping* (DTW) [18] und *Rubine* [22], die häufig in Schrifterkennungssoftware eingesetzt werden. Für diese Gestenerkenner sind zur Implementierung spezielle Kenntnisse in den Bereichen Künstliche Intelligenz und Mustererkennung notwendig. Des Weiteren hat der $1 Gesture Recognizer im Vergleich zu vorhandenen Bibliotheken und Toolkits den Vorteil, dass er unabhängig von der verwendeten Programmiersprache, in jedes Programm integriert werden kann.

Eine symbolische Geste wird zunächst resampelt, rotiert, skaliert, verschoben und anschließend mit bereits vordefinierten Referenzmustern verglichen. Gleichzeitig werden die Wahrscheinlichkeiten ermittelt, mit denen die Referenzmuster mit der symbolischen Geste übereinstimmen. Als Ergebnis wird das Referenzmuster mit der höchsten Wahrscheinlichkeit zurückgeliefert. Die einzelnen Schritte sind in den folgenden Abschnitten genauer erläutert. Der entsprechende Pseudocode befindet sich im Anhang A.4.

5.3.1.1 Resampling einer Geste

Je nach Multi-Touch-Gerät werden pro Sekunde unterschiedlich viele Touch-Events ausgelöst, wenn ein Nutzer den Bildschirm mit seinen Fingern berührt. Besonders kleinere Geräte wie Smartphones lösen aufgrund ihrer schwächeren Hardware pro Sekunde weniger Touch-Events aus als leistungsstärkere Desktop-PCs (siehe Abb. 5.3). Die Anzahl an ausgelösten Touch-Events pro Sekunde, im Folgenden *Abtastpunkte* genannt, ist für die Erkennungswahrscheinlichkeit einer symbolischen Geste besonders kritisch, denn zu wenige Abtastpunkte können dazu führen, dass die symbolische Geste eventuell nicht erkannt wird. Dies ist besonders bei einer schnellen Ausführungsgeschwindigkeit einer Geste der Fall (siehe Abb. 5.5).

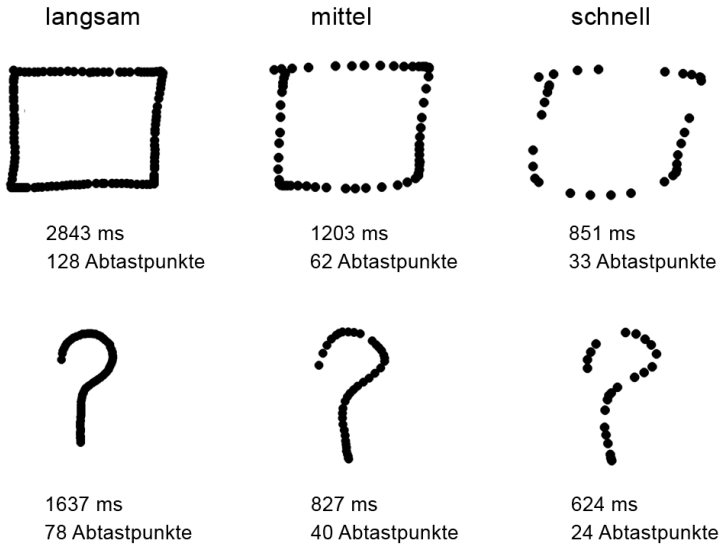

Abbildung 5.5: Ausführung eines Rechteck- und Fragezeichen-Symbols mit unterschiedlichen Ausführungsgeschwindigkeiten auf einem Tablet.

Erzeugt eine Geste zu viele Abtastpunkte, wird mehr Rechenleistung benötigt, um

die Geste mit den vordefinierten Referenzmustern zu vergleichen. Bei einer gleichzeitig langsamen Ausführungsgeschwindigkeit kann es sogar zu ungeraden Linien kommen, die eine Erkennung ebenfalls erschweren. Die optimale Anzahl an Abtastpunkten liegt daher bei $N = 64$ verbunden mit einer mittleren Ausführungsgeschwindigkeit, um eine Geste mit hoher Wahrscheinlichkeit und mit wenig Rechenleistung zu erkennen [30]. Beim Resampling wird die Anzahl der Abtastpunkte angepasst, indem Punkte entfernt oder mittels linearer Interpolation hinzugefügt werden (siehe Abb. 5.6). Der gleichbleibende Abstand zwischen den einzelnen 64 Abtastpunkten ergibt sich aus der Gesamtlänge des originalen Gesten-Pfades, der durch $(N - 1)$ dividiert wird.

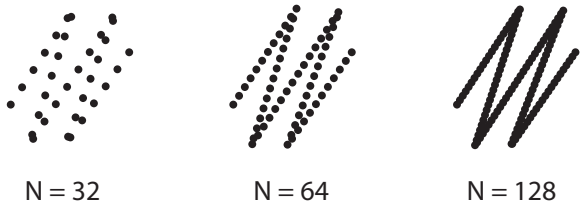

Abbildung 5.6: Zickzack-Symbol bestehend aus N=32, 64 und 128 Abtastpunkten.

5.3.1.2 Rotation

Im zweiten Schritt findet eine Rotation der symbolischen Geste statt. Hierzu wird der indikative Winkel der Geste bestimmt, der sich aus dem Startpunkt und dem Mittelpunkt der Geste ergibt. Die Rotation endet, sobald die Geste einen Winkel von $0°$ besitzt (siehe Abb. 5.7).

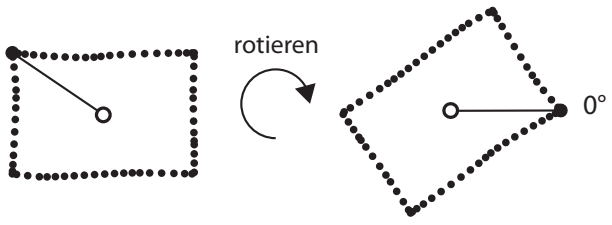

Abbildung 5.7: Rechteck-Symbol wird rotiert, sodass sein indikativer Winkel bei $0°$ liegt.

5.3.1.3 Skalierung und Verschiebung

Die symbolische Geste wird im dritten Schritt auf eine vordefinierte Größe skaliert. Dabei handelt es sich um eine nicht uniforme Skalierung. Die Bounding Box der Geste besitzt nach abschließender Skalierung eine quadratische Form, mit einer Kantenlänge von 200 Pixeln. Zu beachten ist, dass eine gezeichnete Gerade nicht konform skaliert werden kann. Die Erkennung einer gezeichneten Gerade erfolgt daher ohne Anwendung des $1 Gesture Recognizers nur mithilfe der Bounding Box der Geste. Hierzu werden die Kantenlängen der Bounding Box miteinander verglichen, bevor es zur Ausführung des Gestenerkenners kommt. Auch gibt es Unterscheidungsprobleme zwischen einem Quadrat und Rechteck, sowie einem Kreis und einer Ellipse, da diese Symbole nach der Skalierung identisch aussehen. Deshalb wird in dieser Arbeit darauf geachtet, dass nur symbolische Gesten verwendet werden, die sich eindeutig voneinander unterscheiden lassen.

Nachdem eine symbolische Geste skaliert ist, wird anschließend ihr Mittelpunkt verschoben, sodass dieser sich an der Position $(0, 0)$ des Bearbeitungsbereichs in proView befindet.

5.3.1.4 Gestenerkennung mittels Referenzmuster

Die bisher vorgestellten Verarbeitungsschritte werden ebenfalls auf die davor abgelegten Referenzmuster angewandt. Diese Referenzmuster sind für die optimale Erkennung einer symbolischen Geste wichtig und müssen daher dieselbe Anzahl an Abtastpunkten, denselben Winkel und dieselbe Größe sowie Position aufweisen, wie die symbolische Geste selbst.

Referenzmuster sind in der XML-Datei `templates.xml` im Ordner `WEB-INF` des TouchPanel-Add-ons abgelegt. Listing 5.2 zeigt einen beispielhaften Aufbau dieser XML-Datei.

Listing 5.2: Referenzmuster in `templates.xml`

```
1 <?xml version="1.0" encoding="utf-8" standalone="yes"?>
2 <templates>
3  <template name="RECTANGLE"><pt x="59" y="158" />...</template>
4  <template name="DELETE"><pt x="11" y="58" />...</template>
5 </templates>
```

In Zeile 3 und 4 sind zwei unterschiedliche Referenzmuster aufgeführt, die jeweils durch ein `template`-Tag repräsentiert sind. Innerhalb dieses Tags befinden sich

die zu den Referenzmustern gehörenden x- und y-Koordinaten der einzelnen Abtastpunkte. Pro Abtastpunkt wird ein `pt`-Tag verwendet, dessen Attribute die Koordinaten enthalten.

Die Klasse `TemplateList` im `recognizer`-Paket liest die XML-Datei mithilfe des SAX-Parsers aus und legt einzelne Referenzmuster als Instanz der Klasse `Template` in einer `LinkedList` ab. Da außer einem Lesevorgang keine weiteren Operationen an der XML-Datei durchgeführt werden, ist der SAX-Parser ausreichend. Die `TemplateList` nutzt das Singleton-Pattern, um eine wiederholte und rechenaufwendige Initialisierung der Liste zur Laufzeit zu vermeiden.

Ein neues Referenzmuster wird zunächst wie eine symbolische Geste gezeichnet und mit der Methode `printTouchPointsToConsole`, die sich in der Klasse `GestureRecognizer` im `recognizer`-Paket befindet, auf der Konsole ausgegeben. Der Entwickler kann die Ausgabe anschließend direkt in die XML-Datei `templates.xml` kopieren und das Referenzmuster benennen. Client-Nutzer von proView haben somit keine Möglichkeit eigene Referenzmuster zu erzeugen, die eventuell zu einer falsch-positiven Gestenerkennung innerhalb der Web-Anwendung führen könnten.

Pro symbolischer Geste können im System mehrere Referenzmuster vordefiniert sein, um die Trefferwahrscheinlichkeit bei der Erkennung zu erhöhen. Bereits bei einem geladenen Referenzmuster erhält man eine Trefferwahrscheinlichkeit von 97% (siehe Abb. 5.8). Ab drei geladenen Referenzmustern zu einer Geste erhöht sich die Trefferwahrscheinlichkeit auf bis zu 99,5% [30]. Auch bei den anderen Gestenerkennern Rubine und DTW hängt die Erkennungsrate von der Anzahl an vordefinierten Referenzmustern ab. Je mehr Referenzmuster definiert sind, desto mehr Rechenleistung ist allerdings für die Erkennung notwendig, da die symbolische Geste mit jedem Referenzmuster verglichen werden muss.

Während des Vergleichs wird zunächst eine durchschnittliche Distanz d_i zwischen den Abtastpunkten einer Geste C und den Punkten des Referenzmusters T_i ermittelt (siehe Gleichung 5.1).

$$d_i = \frac{\sum_{k=1}^{N} \sqrt{(C[k]_x - T_i[k]_x)^2 + (C[k]_y - T_i[k]_y)^2}}{N} \tag{5.1}$$

Das Referenzmuster mit der kleinsten Distanz zur Geste ist das Ergebnis der Gestenerkennung. Zur Ermittlung der kleinsten Distanz muss die Geste trotz eines indikativen Winkels von $0°$ weiter rotiert werden, sodass die Geste im optimalen Winkel zum Referenzmuster steht. Daher wird sie unter Verwendung des Bergsteigeralgorithmus (engl. *hill climbing*) iterativ um ± 1 Grad weiter rotiert, bis die

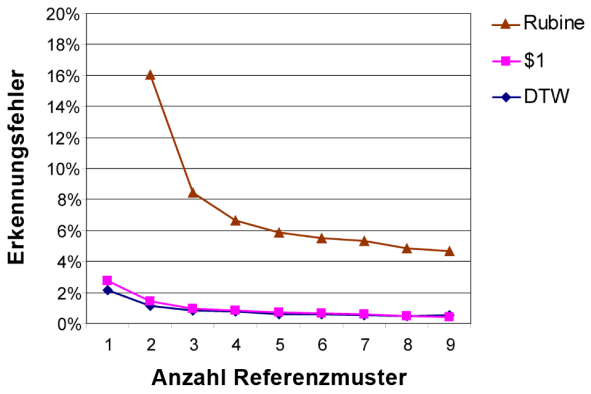

Abbildung 5.8: Erkennungsfehler sind abhängig von der Anzahl der Referenzmustern pro Geste.

optimale, kleinste Distanz erreicht ist, d.h. sich die Distanz bereits im nächsten Schritt wieder verschlechtert.

Die ermittelte Distanz wird anschließend in einen Ergebniswert `score` umgewandelt, dessen Wertebereich zwischen 0 und 1 liegt (siehe Gleichung 5.2).

$$score = 1 - \frac{d_i}{\frac{1}{2}\sqrt{size^2 + size^2}} \tag{5.2}$$

Überschreitet der `score` den Grenzwert 0,85 gilt die symbolische Geste als erkannt. Mehrere Gestenausführungen haben gezeigt, dass bei einer mittleren bis langsamen Ausführungsgeschwindigkeit, `score`-Werte über 0,9 erreichbar sind, daher ist im Kontext dieser Arbeit ein Grenzwert von 0,85 gewählt worden. Bei Werten, die unterhalb dieses Grenzwerts liegen, wird die eingegebene Geste ignoriert bzw. muss vom Benutzer erneut ausgeführt werden.

Die Variable `size` in der Gleichung entspricht der Kantenlänge einer Geste nach ihrer Skalierung.

5.3.2 Gesten-Listener und -Events

Das TouchPanel gibt Informationen über physikalische sowie symbolische Gesten mittels Ereignissen an andere Klassen weiter. Diese Klassen müssen die Schnittstelle `ITouchPanelGestureListener` implementiert haben, um auf Gesten-Ereignisse zu reagieren. In diesem Fall ist es die proView-Klasse `BPMN_Appearance`, die eine Implementierung der Schnittstelle enthält. Im Folgenden werden die einzelnen Ereignisse, die ausgelöst werden können, genauer beschrieben.

TouchPanelDoubleTapGestureEvent Dieses Ereignis wird ausgelöst, sobald der Nutzer eine Double-Tap-Geste auf dem Multi-Touch-Bildschirm ausführt. Die Methode des `ITouchPanelGestureListeners`, die hierfür implementiert werden muss, heißt `doubleTapGestureExecuted`. Das TouchPanelDoubleTapGestureEvent enthält die Quelle des Ereignisses und den Berührungspunkt der ausgeführten Double-Tap-Geste.

TouchPanelGestureEvent Symbolische Gesten lösen ein TouchPanelGestureEvent aus. Dieses Ereignis enthält die Ausführungsquelle, Abtastpunkte, Mittelpunkt und Bounding Box der Geste. Ebenfalls enthalten ist das Ergebnis des Gestenerkenners, das aus dem Gestentyp und Score besteht. Übergeben werden diese Informationen an die Implementierung der Methode `gestureExecuted` des `ITouchPanelGestureListeners`.

TouchPanelHoldGestureEvent Verweilt der Finger des Nutzers für mehr als eine Sekunde an einer Stelle auf dem Multi-Touch-Bildschirm, wird ein TouchPanelHoldGestureEvent ausgelöst. Die Informationen, die dieses Ereignis enthält, sind die Quelle des Ereignisses und der Berührungspunkt. Das Ereignis wird an die Methode `holdGestureExecuted` übergeben.

TouchPanelPanGestureEvent Das TouchPanelPanGestureEvent wird ausgelöst, wenn der Benutzer seinen Finger auf dem Multi-Touch-Bildschirm verschiebt. Die Abstände (dx, dy) zwischen den einzelnen Abtastpunkten während der Gestenausführung, sind in diesem Ereignis enthalten. Ebenfalls enthalten ist die Quelle des Ereignisses. Zur Weiterverarbeitung wird die Geste an die Methode `panGestureExecuted` des `ITouchPanelGestureListeners` übergeben. Für die serverseitige proView-Erweiterung ist dieses Ereignis nicht von großer Relevanz, da Pan-Gesten bereits clientseitig verarbeitet werden und direkt an den Bearbeitungsbereich mit den Prozessmodellen übergeben werden. Das TouchPanelPanGestureEvent wird hier lediglich für zukünftige Arbeiten zur Verfügung gestellt.

TouchPanelPinchGestureEvent Eine Pinch-Geste führt die Vergrößerung sowie Verkleinerung des Prozessmodells herbei. Hierzu wird das TouchPanelPinchGestureEvent ausgelöst, das einen Skalierungsfaktor und einen boolschen Wert `isPinchEnd` enthält. Der Skalierungsfaktor ist zur Unterscheidung zwischen Vergrößerung und Verkleinerung wichtig. Gleichzeitig gibt er an, wie stark skaliert werden soll. Der boolsche Wert `isPinchEnd` signalisiert lediglich das Ende einer Pinch-Geste. Das Ereignis wird an die Methode `pinchGestureExecuted` des `ITouchPanelGestureListeners` zur Weiterverarbeitung übergeben.

TouchPanelTapGestureEvent Ein kurzes Antippen des Bildschirms führt ebenfalls zu einem Ereignis. Dieses TouchPanelTapGestureEvent enthält Informationen zur Quelle des Ereignisses sowie Positionsangaben des Berührungspunktes, die an die Methode `tapGestureExecuted` weitergegeben werden.

5.4 Emulation von Touch-Events

Da das Testen des TouchPanel-Add-ons auf einem Multi-Touch-fähigen Gerät sehr zeitaufwendig sein kann, bietet es sich an einen Emulator zu verwenden, der Touch-Events auf dem Desktop-PC simulieren kann.

Es gibt Emulatoren für Betriebssysteme sowie für bestimmte Browser. Diese haben allerdings diverse Nachteile. Für den Firefox-Browser gibt es zum Beispiel ein Multi-Touch-Emulator-Add-on. Dieses ist allerdings seit 2010 nicht mehr weiterentwickelt worden und mit neueren Firefox-Versionen nicht mehr kompatibel. Der Chrome-Browser besitzt einen integrierten Touch-Event-Emulator, der über die Web-Entwickler Eigenschaften aktiviert werden kann. Dieser Emulator unterstützt jedoch keine Multi-Touch-Eingaben (außer auf Apple MacBook oder MagicPad), die beispielsweise zur Ausführung von Pinch-Gesten notwendig sind. Für das Betriebssystem Windows 7 sind keine Emulatoren vorhanden, sondern nur ein Emulator für die Vorgängerversion Windows Vista, den der Benutzer wie einen gewöhnlichen Treiber installiert.

Aufgrund dieser Umstände besitzt das TouchPanel einen integrierten Emulator, mit dem es möglich ist, einzelne Touch-Events sowie Multi-Touch-Events auszulösen. Die einzelnen Touch-Events werden mittels gewöhnlicher Mouse-Events simuliert. Für die Simulation von Multi-Touch-Events wird jedoch mindestens ein weiterer Berührungspunkt benötigt. Durch einen Doppelklick wird ein weiterer fester Punkt im Bearbeitungsbereich platziert, der den Einsatz eines zweiten Fingers simuliert. Bei gedrückter, linker Maustaste und einer Bewegung des Mauszeigers Richtung zweiten Berührungspunkt bzw. in entgegengesetzte Richtung, wird ein Pinch-Ereignis ausgelöst.

Diese Emulation von Touch-Events lässt sich in proView jederzeit über die obere Menüleiste der Anwendung aktivieren bzw. deaktivieren. Im Programm wird hierfür die Methode `setMouseMode(true)` bzw. `setMouseMode(false)` des TouchPanels verwendet.

6 Erweiterung des proView-Frameworks

Dieses Kapitel diskutiert die Erweiterung des bestehenden proView-Frameworks, um Gesten-basierte Prozessmodellierung zu ermöglichen. An dieser Stelle kommt das TouchPanel-Add-on zum Einsatz, das in Kapitel 5 bereits ausführlich beschrieben ist. Das TouchPanel wird in das proView-Projekt integriert (siehe Abschnitt 6.1) und die vom TouchPanel ausgelösten Gesten-Ereignisse mit den entsprechenden proView-Funktionen verknüpft. Abschnitt 6.2 behandelt hierbei die symbolischen Gesten und die Abschnitte 6.3 bis 6.6 beziehen sich auf physikalische Gesten.

6.1 Integration des TouchPanel-Add-ons

In Form einer JAR-Bibliothek wird das TouchPanel-Add-on in den `WEB-INF/lib`-Ordner des proView-Projekts kopiert. Dieses Projekt enthält die Klasse `BPMN_Appearance`, die im Wesentlichen für die Visualisierung von BPMN-Prozessmodellen zuständig ist. Sie besteht aus mehreren ineinander verschachtelten Vaadin-Layouts und -Panels. Eines davon ist das Panel `processVisPanel`, das durch das TouchPanel ersetzt wird (siehe Abb. 6.1).

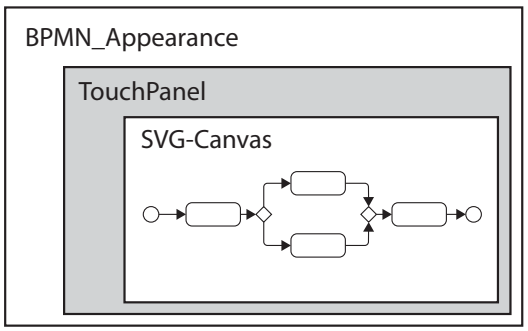

Abbildung 6.1: Integration des TouchPanels in die Klasse `BPMN_Appearance`.

Als Unterobjekt ist diesem Panel ein Canvas zugeordnet, das Prozessmodelle als SVG-Grafik darstellt. Dieses Canvas bekommt eine eindeutige ID mit der Bezeichnung „canvas“ zugewiesen. Dieselbe ID wird auch dem `TouchPanel`-Objekt als

Parameter im Konstruktor übergeben, um eine eindeutige Verknüpfung zwischen TouchPanel und Canvas zu erhalten. Zum Beispiel ist es wichtig, dass Pan-Gesten clientseitig direkt vom TouchPanel-Widget an das Canvas übergeben werden, um das im Canvas enthaltene Prozessmodell mit der Geste zu verschieben. Ein „Umweg“ über den Server führt zu einer zu hohen Latenzzeit bzw. einer ruckelnden Verschiebung.

Damit vom TouchPanel ausgelöste Ereignisse bei der Klasse `BPMN_Appearance` ankommen, implementiert diese das `ITouchPanelGestureListener`-Interface. Mit der Methode `addGestureListener` registriert sich die Klasse als Listener beim TouchPanel.

Um die bereits in Abschnitt 5.4 beschriebene Emulation von Touch-Events mittels Mauszeiger zu ermöglichen, wird die Menüleiste von proView modifiziert. Durch einen zusätzlichen Eintrag im Menü von proView kann der Benutzer zwischen Maus- und Touch-Modus wechseln (siehe Abb. 6.2a). Der entsprechende Menüeintrag hierzu lautet „Enable Touch Mode“. Die Aktivierung des Maus-Modus ruft die Methode `setMouseMode(true)` des `TouchPanel`-Objekts auf. Dem gegenüber ist mit derselben Methode und einem negativen Parameterwert wieder ein Wechsel zum Touch-Modus möglich. Standardmäßig ist der Maus-Modus jedoch deaktiviert.

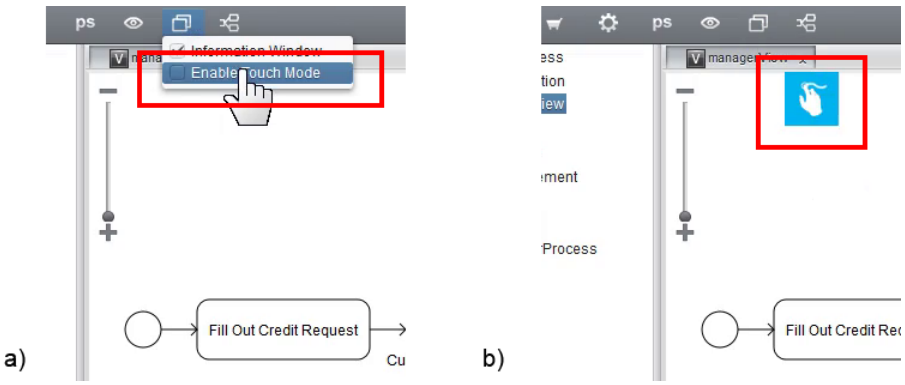

Abbildung 6.2: Den Touch-Modus aktivieren a) und zwischen Ansichts- und Bearbeitungsmodus wechseln b).

Das TouchPanel unterstützt außer der Emulation von Maus- und Touch-Events noch zwei weitere Modi: Den *Ansichts-* und *Bearbeitungsmodus*. Der Ansichtsmodus dient dazu, dass der Benutzer ein Prozessmodell problemlos verschieben sowie vergrößern kann, ohne dass es zu einer (unerwünschten) Manipulation von Prozesselementen kommt. Eine Manipulation des Prozessmodells ist nur im Bearbeitungsmodus möglich. Diese Trennung hat zudem den Vorteil, dass manche

Gesten, wie zum Beispiel die Pinch-Geste, zweimal mit einer Funktion belegt werden können.

Der Wechsel zwischen den beiden Modi wird durch einen Switch-Button realisiert, der sich auf dem Canvas in der oberen, linken Ecke befindet (siehe Abb. 6.2b). Das Antippen oder Anklicken dieses Switch-Buttons führt zu einer Ausführung der Methode `setMode` des `TouchPanel`-Objekts. Diese Methode erhält, je nachdem welcher Modus aktiviert wird, einen boolschen Wert, damit das TouchPanel die richtigen Ereignisse auslöst. Gleichzeitig ändert sich das Symbol des Switch-Buttons, um den aktivierten Modus zu kennzeichnen.

Die Größe des TouchPanels muss sich initial an die Größe des übergeordneten Layout-Objektes `processContent` anpassen, daher ist es notwendig zu Beginn der Ausführung die Methoden `setHeight` und `setWidth` des `TouchPanel`-Objektes aufzurufen. Diese Methoden erhalten als Parameter die entsprechenden Größen zugewiesen. Wenn sich das Browserfenster vergrößert oder verkleinert, löst dies ein Ereignis aus, auf das die Methode `resize` innerhalb der Klasse `BPMN_Appearance` reagiert. Diese Methode enthält ebenfalls eine Anweisung zur Anpassung der TouchPanel-Größe.

Aufgrund der Nutzung der `ITouchPanelGestureListener`-Schnittstelle seitens der `BPMN_Appearance`-Klasse sind weitere Methoden-Implementierungen notwendig, die auf die in Abschnitt 5.3.2 beschriebenen TouchPanel-Ereignisse reagieren. Diese Methoden sowie deren Implementierung werden im Folgenden erläutert.

6.2 Symbolische Gesten

Mithilfe der Methode `gestureExecuted` ist die Klasse `BPMN_Appearance` in der Lage im Bearbeitungsmodus auf symbolische Gesten zu reagieren. Die Methode besitzt als Parameter das `TouchPanelGesture`-Event, das alle notwendigen Informationen zur Geste enthält. Dies ist unter anderem ein Score-Wert sowie der Typ der Geste.

Zunächst findet eine Überprüfung des Score-Wertes statt, der einen Wert größer als 0,85 enthalten muss. Ist dies nicht der Fall, wird die Geste ignoriert. Ein passender Score-Wert leitet die anschließende Untersuchung des Gestentyps ein, damit die Geste entsprechend verarbeitet werden kann. Der Winkel einer Geste spielt übrigens für eine korrekte Erkennung keine Rolle und hat auch keinen großen Einfluss auf den Score-Wert. So können Benutzer Gesten beispielsweise auch in einem 90 Grad Winkel auf dem Multi-Touch-Bildschirm zeichnen.

Da das TouchPanel sehr viele unterschiedliche Symbol-Typen unterstützt, behan-

delt die Methode `gestureExecuted` auch entsprechend viele Fälle, die in den Abschnitten 6.2.1 bis 6.2.8 näher beschrieben sind.

6.2.1 Löschen-Symbol

Erkennt der Gestenerkenner des TouchPanels das Löschen-Symbol (siehe Abb. 3.8), ermittelt dieser daraufhin das zu löschende Prozesselement. Hierzu wird der Mittelpunkt der Geste aus dem `TouchPanelGesture`-Event ausgelesen und an die Methode `getElementFromPoint` übergeben. Diese Methode ermittelt anschließend das involvierte Prozesselement. Dabei durchläuft sie in einer Schleife alle Aktivitäten sowie Verzweigungsknoten des Prozessmodells und vergleicht gleichzeitig deren Position und Größe mit dem übergebenen Mittelpunkt. Ist eine Aktivität gefunden, deren Position mit der Position des Gesten-Mittelpunkts übereinstimmt, ist diese Aktivität das Resultat der Methode `getElementFromPoint`. Stimmen die Positionen allerdings nicht überein, so wird ein Null-Objekt zurückgeliefert und die symbolische Geste löst somit keine Reaktion aus.

Ein erfolgreich ermitteltes Prozesselement gelangt anschließend als Parameter zu der bereits in proView implementierten Methode `deleteNode`. Diese Methode ist Bestandteil der `ModellingService`-Schnittstelle in proView, die auch für weitere Modellierungsfunktionen verantwortlich ist. Die Methode `deleteNode` entfernt das Prozesselement endgültig aus dem Prozessmodell.

6.2.2 Rechteck-Symbol

Das Rechteck-Symbol ist für unterschiedliche Modellierungsfunktionen in proView relevant. Mit diesem Symbol lassen sich mehrere Prozesselemente zusammenfassen bzw. aggregieren und es dient unter anderem dazu, eine neue Prozesssicht oder einen neuen Verzweigungsblock zu erstellen. Diese Modellierungsfunktionen werden allerdings nicht direkt ausgeführt, sondern in einem sich öffnenden Kontextmenü angeboten. Der Benutzer kann mithilfe dieses Menüs die gewünschte Modellierungsfunktion ausführen.

Die Methode `getElementsFromRectangle` ermittelt zunächst die vom Rechteck umschlossenen Prozesselemente. Sie erhält als Parameter die Bounding Box der ausgeführten Geste. Eine Schleife durchläuft alle Aktivitäten sowie Verzweigungsknoten des Prozessmodells und prüft, ob sich ein komplettes Prozesselement innerhalb der Bounding Box befindet. Ragt beispielsweise eine Aktivität aufgrund ihrer Größe über den Rand der Bounding Box hinaus, so wird sie nicht mitgezählt.

Alle anderen Aktivitäten gelangen in einem `ArrayList`-Objekt als Resultat an die Methode `showTouchMenu`. Gleichzeitig erhält diese Methode als Parameter die Position des letzten Berührungspunktes, da an dieser Stelle das Kontextmenü erscheint.

Die `showTouchMenu`-Methode ermittelt die Anzahl der ausgewählten Prozesselemente und erzeugt anschließend das Kontextmenü. Abhängig von der Anzahl an Prozesselementen, variieren die Einträge im Kontextmenü. Die beiden Modellierungsfunktionen zur Erzeugung eines XOR- oder AND-Verzweigungsblocks zum Beispiel, setzen eine Anzahl von genau zwei Prozesselementen voraus. Ist diese Anzahl nicht gegeben, sind diese Modellierungsfunktionen im Kontextmenü auch nicht vorhanden.

Das Antippen eines Menü-Eintrags löst ein `ContextMenuItemClick`-Event aus, auf das die Methode `contextMenuItemClicked` reagiert. Diese Methode ermittelt den angetippten Menüeintrag und führt mit der `ModellingService`-Schnittstelle die passenden Modellierungsfunktionen aus. So führt sie zum Beispiel die Methode `createViewFromSubprocessAndNodes` aus, wenn aus den ausgewählten Prozesselementen eine neue Prozesssicht erstellt werden soll. Die Methode `aggregateNodeIds` des Modelling-Services ist zuständig, Prozesselemente zu aggregieren und die Methoden `insertConditional` sowie `insertParallel` sind für die Erstellung neuer Verzweigungsblöcke verantwortlich.

6.2.3 Rechteck-Symbol mit Diagonale

Prozesselemente lassen sich mit einem Rechteck-Symbol, das zusätzlich einen diagonalen Strich von der oberen, linken Ecke zur unteren, rechten Ecke besitzt, reduzieren. Die Ermittlung der Prozesselemente, die von der Geste betroffen sind, geschieht wie in Abschnitt 6.2.2 mit der Methode `getElementsFromRectangle`. Auch in diesem Fall erhält diese Methode die Bounding Box der symbolischen Geste als Parameter.

Im Gegensatz zum vorherigen Rechteck-Symbol erscheint hier nach Gestenausführung kein Kontextmenü, sondern die entsprechenden Prozesselemente werden sofort reduziert. Zu diesem Zweck gibt es die Methode `reduceNodeIds` der `ModellingService`-Schnittstelle, die als Parameter ein `Set`-Objekt mit allen ausgewählten Prozesselementen erhält.

6.2.4 Strich-Symbol

Strich-Symbole sind sowohl für Synchronisations-, Lese- und Schreibkanten, als auch zum Einfügen neuer Aktivitäten relevant. Als erstes ermittelt die Methode `distance` nach Gestenausführung die Strichlänge. Diese Methode erhält als Parameter den Start- sowie den Endpunkt des Strichs. Strich-Symbole, die kleiner als 40 Pixel sind, werden ignoriert. Der Grund hierfür ist, dass es sich bei einer ausgeführten Tap-Geste ebenfalls um ein Strich-Symbol handeln könnte.

Es wird ebenfalls geprüft, ob die Endpunkte des Strich-Symbols auf ein und demselben Prozesselement liegen. Die Methode `getElementFromPoint` bekommt die Endpunkte zugewiesen und ermittelt die Prozesselemente auf denen sie liegen. Bei ein und demselben Prozesselement sind keine weiteren Aktionen definiert. Liegt der Startpunkt des Strich-Symbols auf einem Datenelement und der Endpunkt auf einer Aktivität, so handelt es sich hierbei um eine Lesekante und im umgekehrten Fall um eine Schreibkante. Bei diesen Kanten finden keine weiteren Aktionen statt, da die Funktionen zum Einfügen einer neuen Lese- sowie Schreibkante noch nicht Bestandteil des proView-Frameworks sind.

Liegen beide Endpunkte auf zwei unterschiedlichen Aktivitäten innerhalb eines AND-Verzweigungsblocks, wird eine neue Synchronisationskante zwischen diesen beiden Aktivitäten eingesetzt. Dies geschieht mithilfe der Methode `insertSyncEdge` der `ModellingService`-Schnittstelle. Sie erhält die beiden ermittelten Aktivitäten als Parameter. Die Richtung der Synchronisationskante hängt von den Positionen der Endpunkte des Strich-Symbols ab, die auf den Aktivitäten liegen.

Ermittelt die Methode `getElementFromPoint` für beide Endpunkte keine Prozesselemente, geht die Anwendung davon aus, dass sie eine neue Aktivität einfügen soll. Deshalb kommt es zum Aufruf der Methode `insertActivity`, die prüft, ob es sich um ein vertikales oder horizontales Strich-Symbol handelt. Je nachdem wird anschließend die Methode `insertActivityWithVerticalStroke` oder `insertActivityWithHorizontalStroke` ausgeführt. Bei beiden Methoden findet zunächst eine Erweiterung der Bounding Box der Geste statt. Ein vertikaler Strich führt dazu, dass sich die Höhe der Bounding Box um 20 Pixel und die Breite um 80 Pixel vergrößert (siehe Abb. 6.3a). Bei horizontalen Strichen vergrößert sich die Breite um 20 Pixel und die Höhe um 80 Pixel. Diese Erweiterung der Bounding Box dient dazu, Prozesselemente in der Nähe der Geste zu ermitteln. Daher muss der Benutzer beim Zeichnen des Strich-Symbols die Entfernung zu anderen Prozesselementen beachten.

Alle Prozesselemente werden in einer Schleife durchlaufen, um Überschneidungen

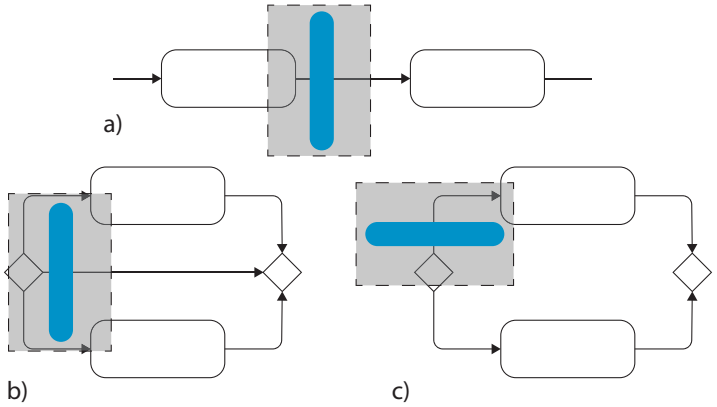

Abbildung 6.3: Erweiterung der Bounding Box.

der Prozesselemente mit der erweiterten Bounding Box des Strich-Symbols zu finden. Handelt es sich bei einem der entdeckten Prozesselemente um ein XOR- oder AND-Split und besitzt dieser Verzweigungsknoten drei Zweige, fügt die Methode `insertActivityWithVerticalStroke` auf dem mittleren Zweig eine neue Aktivität ein (siehe Abb. 6.3b). Die Methode `insertActivityWithHorizontalStroke` prüft dahingegen noch zusätzlich, ob der Verzweigungsknoten oberhalb oder unterhalb des Strich-Symbols liegt. Abhängig davon wird eine neue Aktivität auf dem ersten oder auf dem zweiten bzw. dritten Zweig eingefügt, wenn der Verzweigungsblock aus drei Zweigen besteht (siehe Abb. 6.3c).

Ermittelt die Methode `insertActivityWithVerticalStroke` zwei Aktivitäten bzw. ein Start- oder End-Ereignis, prüft sie anschließend, ob diese mittels einer Sequenzflusskante miteinander verbunden sind. Erst dann setzt sie zwischen diesen Prozesselementen eine neue Aktivität ein. Bei nur einem gefundenen Prozesselement prüft die Methode ebenfalls, ob dieses mit einem anderen per Sequenzflusskante verbunden ist. Auf diese Weise ist es möglich Datenelemente oder sonstige unerwünschte Prozesselemente auszuschließen.

Neue Aktivitäten werden mit der Methode `insertSerial` eingefügt, die Bestandteil der `ModellingService`-Schnittstelle ist.

6.2.5 Pfeil-Symbol

Das Pfeil-Symbol ist zum Einfügen eines neuen Zweiges innerhalb eines Verzweigungsblocks notwendig. Der Startpunkt der Geste muss sich auf einem XOR- oder AND-Split befinden. Zur Ermittlung des darunterliegenden Prozesselements ist die Methode `getElementFromPoint` zuständig. Die Pfeilspitze muss sich dahingegen

auf keinem speziellen Prozesselement befinden. Es muss jedoch darauf geachtet werden, dass der Pfeil nach oben ausgerichtet ist.

Neue Verzweigungen setzt die Methode `insertBranch` der `ModellingService`-Schnittstelle in das Prozessmodell ein. Diese Methode benötigt allerdings zu einem Split-Gateway auch das dazugehörige Join-Gateway. Beide Gateways, sowie andere Prozesselemente auch, besitzen in proView eine eindeutige ID. Wenn die ID eines Split-Gateways n ist, dann ist die ID eines Join-Gateways immer $n+1$, da das System beide Prozesselemente beinahe zeitgleich in ein Prozessmodell eingefügt. Aus diesem Grund wird das zu einem Split-Gateway zugehörige Join-Gateway über dessen ID, sowie mithilfe der Methode `getNode` der Klasse `BPMN_ProcessVisualisationComponent` ermittelt und an die `insertBranch`-Methode übergeben.

6.2.6 Halbes Rechteck-Symbol

Halbe Rechtecke dienen dazu, Schleifen in das Prozessmodell einzuzeichnen. Die beiden Endpunkte der Geste erhält die Methode `getElementFromPoint` als Parameter, um die Prozesselemente zu ermitteln, auf denen die Endpunkte liegen. Handelt es sich bei den ermittelten Prozesselementen um Aktivitäten oder Verzweigungsknoten, kann die Schleife eingefügt werden. Die Schnittstelle `ModellingService` stellt die Methode `insertLoop` zur Verfügung, die als Parameter ein `ArrayList`-Objekt mit den beiden Prozesselementen zugewiesen bekommt. Bei einer erfolgreichen Ausführung der Methode, wird die Schleife in das Prozessmodell eingefügt, ansonsten löst der proView-Server eine Exception aus.

6.2.7 Zickzack-Symbol

Das Zickzack-Symbol dient dazu, einen bestehenden Subprozess aufzulösen oder eine zuletzt ausgeführte Aktion wieder rückgängig zu machen. Die Auflösung eines expandierten Subprozesses findet nur dann statt, wenn sich der Startpunkt der Geste innerhalb des Subprozesses befindet. Um diese Bedingung zu überprüfen, kommt an dieser Stelle die Methode `subprocessOperations` zum Einsatz. Sie überprüft mit der Methode `getElementFromPoint`, ob der Startpunkt auf einem Subprozess liegt oder nicht. Der Startpunkt, den die Methode als Parameter erhält, ist im `TouchPanelGesture`-Event enthalten. Dieses Ereignis besitzt alle Berührungspunkte der ausgeführten Geste.

Befindet sich der Startpunkt auf einem Subprozess, der sich im expandierten Zu-

stand befindet, löst dies die Methode `undoOperations` der Schnittstelle `Modelling-Service` aus. Diese Methode erhält als Parameter den Subprozess, der aufgelöst werden soll. Führt der Benutzer dieselbe Geste auf einer freien Fläche des Hintergrunds aus, löst dies eine Funktion zum Rückgängig machen der letzten Aktion aus. Diese Funktion ist allerdings noch nicht Bestandteil des proView-Frameworks.

6.2.8 Fragezeichen-Symbol

Mithilfe des Fragezeichen-Symbols kann der Benutzer ein Hilfe-Dialog öffnen, der alle Gesten samt zugehöriger Modellierungsfunktionen beschreibt (siehe Abb. 6.4). Die Gestenausführung löst die Methode `openGesturesWindow` aus. In dieser Methode wird ein neues `GesturesWindow`-Objekt initialisiert, das für die Erzeugung des Dialog-Layouts zuständig ist.

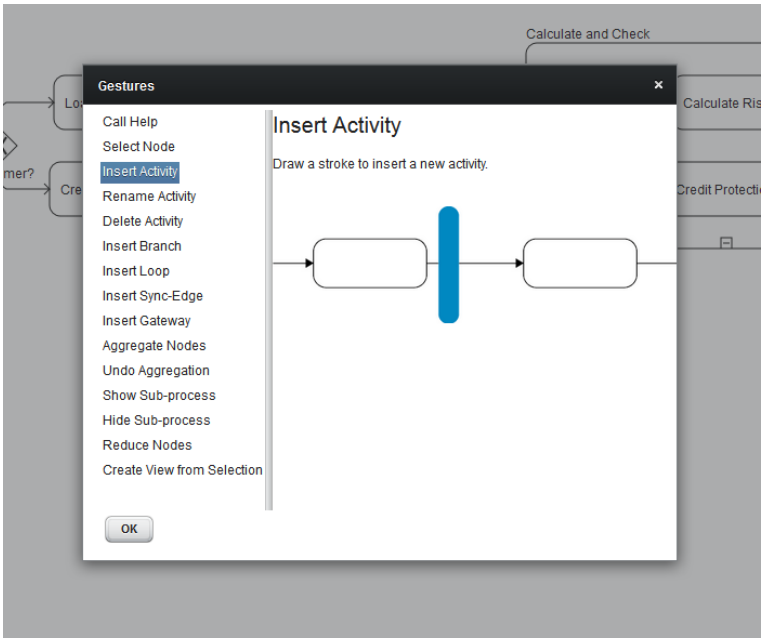

Abbildung 6.4: Hilfe-Dialog.

6.3 Tap-Geste

Unter Verwendung der Tap-Geste im Ansichts- oder Bearbeitungsmodus, kann der Benutzer ein einzelnes Prozesselement selektieren, das sich in proView anschließend farblich von den anderen Prozesselementen hervorhebt. Das Properties-

Panel zeigt weitere Informationen zu diesem selektierten Prozesselement an, wie zum Beispiel dessen ID.

Tap-Ereignisse nimmt die Methode `tapGestureExecuted` der Klasse `BPMN_Appearance` entgegen, die die Position des Berührungspunktes aus dem `TouchPanelTapGesture`-Event ausliest und an die Methode `setSelectedElement` des Objekts `BPMN_ProcessVisualisationComponent` übergibt.

6.4 Double-Tap-Geste

Die Double-Tap-Geste ist im Bearbeitungsmodus zum Umbenennen von Prozesselementen relevant. Durch doppeltes Antippen einer Aktivität, öffnet sich zum Beispiel ein neues Dialog-Fenster, in das der Benutzer den neuen Name für die Aktivität eingeben kann.

Eine Double-Tap-Geste löst ein `TouchPanelDoubleTapGesture`-Event aus, das die Methode `doubleTapGestureExecuted` der Klasse `BPMN_Appearance` abfängt. Dieses Ereignis beinhaltet die genaue Position des Berührungspunktes, die notwendig ist, um das ausgewählte Prozesselement zu ermitteln. Für diese Aufgabe ist die Methode `getElementFromPoint` zuständig, die als Ergebnis entweder ein Prozesselement oder ein Null-Objekt zurückliefert. Bei einem erfolgreich ermittelten Prozesselement wird dieses an die Methode `renameNode` übergeben, die Bestandteil der `ModellingService`-Schnittstelle ist.

6.5 Pinch-Geste

Die Pinch-Geste ist im Ansichtsmodus zum Vergrößern oder Verkleinern des Prozessmodells relevant. Eine Implementierung der Methode `pinchGestureExecuted` in der Klasse `BPMN_Appearance` ermöglicht das Abfangen des `TouchPanelPinchGesture`-Events, das weitere Informationen, wie beispielsweise den Skalierungsfaktor, enthält.

Der Skalierungsfaktor wird nicht direkt an eine Methode zur Visualisierung des Prozessmodells übergeben, sondern an eine vorhandene Slider-Komponente. Diese Slider-Komponente gibt dann die Skalierungsdaten an die Visualisierungskomponente weiter.

Damit die Skalierung nach der Gestenausführung bestehen bleibt, ist die im `TouchPanelPinchGesture`-Event enthaltene boolsche Variable `isPinchEnd` notwendig.

Diese signalisiert das Ausführungsende der Geste, sodass die Skalierung erhalten bleibt.

6.6 Hold-Geste

Der Benutzer kann Subprozesse mit der Hold-Geste entweder aufklappen oder zuklappen. Das `TouchPanelHoldGesture`-Event wird mit der Methode `holdGestureExecuted` abgefangen. Dieses Ereignis enthält die Startkoordinaten des Berührungspunktes, die die Methode `subprocessOperations` weiter verarbeitet. Sie ermittelt, ob der Benutzer die Hold-Geste auf einem Subprozess angewendet hat. Ist dies der Fall, prüft die Methode anschließend, ob sich der Subprozess im expandierten oder im kollabierten Zustand befindet.

Ein expandierter Subprozess löst die Methode `hideSubprocess` aus, die den Subprozess zuklappt. Hierzu ist die Methode `removeExpandedSubProcess` notwendig, die von der Klasse `BPMN_ProcessVisualisationComponent` bereitgestellt wird.

Zum Aufklappen eines Subprozesses ist die Methode `showSubprocess` zuständig, die auf die Methode `addExpandedSubProcess` zugreift, um einen Subprozess in den expandierten Zustand zu überführen.

7 Evaluation der Implementierung

Im Rahmen dieser Arbeit ist eine experimentelle Untersuchung der proView-Erweiterung durchgeführt worden. An diesem Experiment nehmen mehrere unterschiedliche Testpersonen teil, die mithilfe der implementierten Multi-Touch-Gesten ein bestehendes Prozessmodell bearbeiten müssen. Zuvor müssen sie sich allerdings in einem ersten Teil eigene Gesten für die in dieser Arbeit eingesetzten Modellierungsfunktionen einfallen lassen, ohne die bereits implementierten Gesten für diese Modellierungsfunktionen zu kennen.

Das Ziel ist eine Validierung der implementierten Gesten, sowie eine Evaluation, ob die richtigen Gesten für die Implementierung verwendet werden, sodass zukünftige Benutzer von proView in der Lage sind Prozesse mit den bereits implementierten Gesten intuitiv zu modellieren bzw. zu bearbeiten.

Ein solches Experiment bietet zudem den Vorteil, die Weisheit von Vielen bei der Implementierung von Benutzeroberflächen zu nutzen und sich nicht nur nach gewissen Gestaltungsrichtlinien und Erfahrungen des Usability-Designers zu richten. Das bedeutet, je mehr Testpersonen an einem solchen Experiment teilnehmen, desto aussagekräftiger ist das Resultat dieses Experiments und desto besser und robuster ist das Bedienkonzept.

Im weiteren Verlauf dieses Kapitels werden in Abschnitt 7.1 zunächst die Vorbereitungen beschrieben, die zur Durchführung eines solchen Experiments notwendig sind. Die Beschreibung der Experimentdurchführung ist ebenfalls in dem Abschnitt enthalten. In Abschnitt 7.2 des Kapitels werden die ermittelten Daten der einzelnen Testpersonen ausgewertet und genau analysiert. Dabei wird jede ausgeführte Modellierungsfunktion separat betrachtet. Abschnitt 7.3 beinhaltet eine Zusammenfassung zu den Ergebnissen des Experiments.

7.1 Vorbereitung und Durchführung der experimentellen Untersuchung

Das Experiment besteht aus 18 unterschiedlichen Aufgaben, die alle relevanten Modellierungsfunktionen der proView-Erweiterung abdecken. Die einzelnen Testpersonen bearbeiten diese Aufgaben jeweils in drei Teilen. Dabei sind die Reihenfolgen der 18 Aufgaben in jedem Teil unterschiedlich (siehe Tabelle 7.1). Die Aufgabenreihenfolge eines Teils ist allerdings für jede Testperson gleich.

Teil 1: Eigene Gesten ausdenken
F16, F1, F2, F3, F4, F8, F9, F10, F7, F5, F11, F12, F13, F6, F14, F15, F17, F18
Teil 2: Tutorial
F2, F4, F1, F16, F3, F5, F8, F7, F10, F9, F11, F12, F13, F17, F15, F6, F18, F14
Teil 3: Implementierte Gesten anwenden
F11, F12, F13, F10, F5, F8, F9, F7, F16, F2, F3, F4, F1, F6, F17, F15, F18, F14

Tabelle 7.1: Reihenfolge der Aufgaben von Teil 1, 2 und 3 des Experiments.

In jedem Teil kommt eine spezielle Anwendung zum Einsatz, deren Bedienoberfläche den Testpersonen vor Beginn eines Teils vom Versuchsleiter erläutert wird. Die Anwendungen sind in den Abschnitten 7.1.1 bis 7.1.3 beschrieben. In Teil 1 und Teil 3 des Experiments besteht die Möglichkeit zwischen dem Ansichts- und Bearbeitungsmodus zu wechseln. Eine experimentelle Untersuchung der Gesten für den Ansichtsmodus findet allerdings nicht statt, da es sich hierbei um Gesten handelt, die bereits fester Bestandteil vieler berührungsempfindlicher Bedienoberflächen sind.

Der von den Testpersonen zu bearbeitende Prozess in Teil 1 und Teil 3, beinhaltet die Prüfung der Kreditwürdigkeit eines Kreditnehmers. Jeder dieser beiden Teile beginnt mit dem initialen Zustand des Prozesses, wie er in Abbildung A.2 zu sehen ist.

Vor Beginn des Experiments findet eine Aufklärung der Testperson statt. Dies dient dazu, der Testperson das Thema dieser Arbeit, sowie den Ablauf und das Ziel des Experiments näher zu bringen. Daneben werden wichtige demografische Daten zur Person erfasst (d.h. Geschlecht, Alter, Tätigkeit, Erfahrungen mit Multi-Touch-Geräten sowie Prozess-Management). Testpersonen, die keine oder nur wenig Erfahrung mit Prozess-Management haben, bekommen eine kurze Einführung in dieses Thema, indem ihnen einige Prozesselemente vorgestellt werden. Die Testpersonen haben zudem die Möglichkeit selbst Fragen zum Experiment, sowie zur

Prozessmodellierung zu stellen. Schließlich gibt es eine kurze Erläuterung zu den Unterschieden zwischen symbolischen und physikalischen Gesten sowie zu Menübasierten Lösungen. Die Testpersonen haben durch diese Erläuterungen somit einen besseren Überblick über ihre Interaktionsmöglichkeiten.

Um Störungen bzw. Beeinflussungen der Testpersonen auszuschließen, findet das Experiment in einem freien Seminarraum der Universität Ulm statt. Zur Anwendung kommt ein Google Nexus 7 Tablet mit dem Betriebssystem Android, auf dem die Testpersonen die Aufgaben bearbeiten müssen. Eingegebene Resultate übertragen die eingesetzten Anwendungen per WLAN an ein Notebook für spätere Analysezwecke (siehe Abb. 7.1). Das Notebook stellt den proView-Server für das Experiment bereit, der für die Ausführung von proView notwendig ist. Den Testpersonen ist es nicht möglich den Bildschirm des Notebooks zu sehen und können somit nicht abgelenkt werden. Eine Kamera, die sich schräg oberhalb der Testpersonen befindet, zeichnet die Eingaben der Testpersonen auf. Gleichzeitig macht sich der Versuchsleiter, der rechts neben den Testpersonen sitzt, Notizen zu den Eingaben und Kommentaren der Testpersonen.

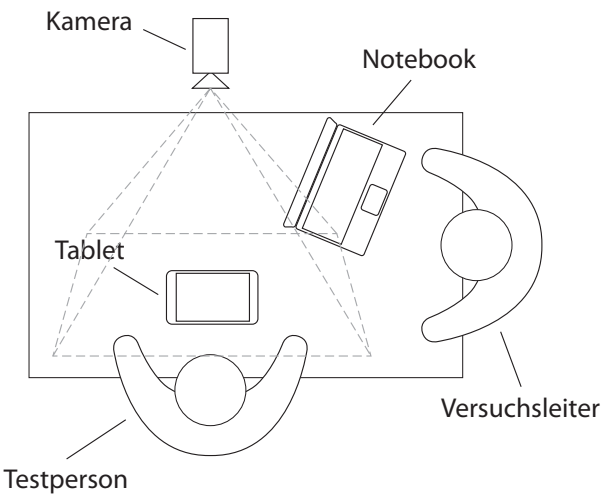

Abbildung 7.1: Aufbau des Experiments.

Es finden drei Probeläufe des Experiments statt, um die Machbarkeit des Experiments sicherzustellen [32]. Nach jedem Probelauf finden Optimierungen an den Aufgaben des Experiments statt.

7.1.1 Teil 1: Eigene Gesten ausdenken

In Teil 1 haben die Testpersonen die Aufgabe, sich zu überlegen, welche Geste sie für welche Modellierungsfunktion verwenden würden, ohne die bereits implementierten Gesten zu kennen. Hierfür ist eine spezielle Anwendung mit dem Namen ProViewTouch entwickelt worden, die den Testpersonen nacheinander die 18 Aufgaben auf dem Bildschirm des Tablets einblendet (siehe Abb. 7.2).

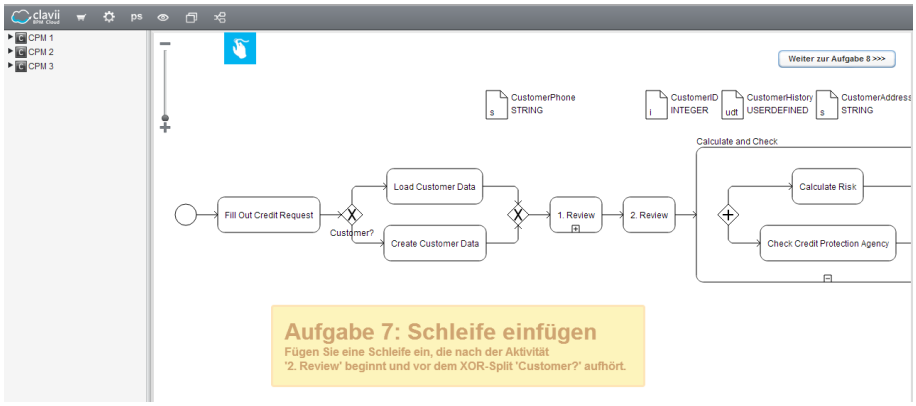

Abbildung 7.2: Benutzeroberfläche von ProViewTouch.

Mit einem Start-Button, der sich in der rechten, oberen Ecke des Bildschirms befindet, können die Testpersonen den Versuch starten und sich die erste Aufgabe anzeigen lassen. Ein Switch-Button in der linken, oberen Ecke des Bildschirms, ermöglicht den Wechsel zwischen Ansichts- und Bearbeitungsmodus. Die Anwendung überträgt eingegebene Gesten zusammen mit dem Startzeitpunkt einer Aufgabe an den proView-Server, der die Daten zur späteren Auswertung in einer XML-Datei ablegt. Zur nächsten Aufgabe gelangen die Testpersonen ebenfalls über den Button in der rechten, oberen Ecke des Bildschirms.

Benutzereingaben führen zu keinen Veränderungen des Prozessmodells, da die eingesetzte Anwendung einerseits keine Prozessänderungen unterstützt und die eingegebenen Gesten andererseits keine Veränderungen herbeiführen können, wenn sie nicht zufällig implementiert sind.

Der Versuchsleiter fordert die Testpersonen vor diesem Teil zu lautem Denken auf, damit Notizen und Skizzen zu ihren Ideen gemacht werden können [28]. Zudem haben Testpersonen die Möglichkeit, während des Experiments Fragen an den Versuchsleiter zu stellen, falls einige der Aufgaben für sie unklar sind.

7.1.2 Teil 2: Tutorial

Im zweiten Teil des Experiments müssen die Testpersonen ein Tutorial bearbeiten, um die implementierten Gesten der proView-Erweiterung zu erlernen. Dies ist für die Durchführung des dritten Teils notwendig. Mit diesem Tutorial können sich die Testpersonen nicht nur an die Gesten gewöhnen, sondern auch an die richtige Ausführungsgeschwindigkeit einer Geste, die für eine erfolgreiche Gestenerkennung im dritten Teil wichtig ist.

Wie in Teil 1 ist auch für dieses Tutorial eine spezielle Anwendung mit dem Namen ProViewTutorial unter Verwendung des Vaadin-Frameworks und des TouchPanel-Add-ons entwickelt worden (siehe Abb. 7.3). Im oberen Bereich der Benutzerschnittstelle blendet die Anwendung den Testpersonen die 18 unterschiedlichen Aufgaben textuell ein. Die Reihenfolge der Aufgaben ist allerdings im Vergleich zum vorherigen Teil anders (siehe Tabelle 7.1). Mithilfe eines Start-Buttons in der rechten, oberen Ecke des Bildschirms können die Testpersonen das Tutorial starten, sowie zur nächsten Aufgabe gelangen.

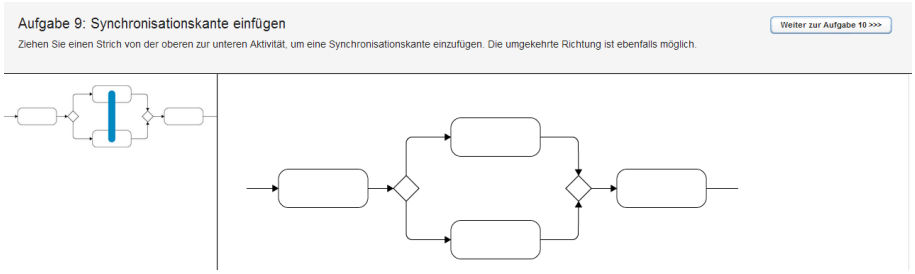

Abbildung 7.3: Benutzeroberfläche von ProViewTutorial.

Pro Aufgabe blendet die Anwendung im linken Bereich der Benutzerschnittstelle die implementierten Gesten inklusive Prozesselemente in bildhafter Form ein. Rechts neben dieser Darstellung erscheint eine Abbildung derselben Prozesselemente ohne eingezeichneter Geste. Die Testpersonen haben auf diese Weise die Möglichkeit, die Geste nachzuzeichnen und sie zu erlernen. Sowohl bei erfolgreicher als auch bei nicht erfolgreicher Eingabe erscheint auf dem Bildschirm eine entsprechende Meldung.

7.1.3 Teil 3: Implementierte Gesten anwenden

Nachdem die Testpersonen mithilfe des Tutorials die Gesten erlernt haben, müssen sie in Teil 3 des Experiments ein erneutes Mal alle 18 Modellierungsfunktionen

bzw. Aufgaben in unterschiedlicher Reihenfolge ausführen, diesmal allerdings unter Verwendung der implementierten Gesten der proView-Erweiterung. Auf diese Weise kann untersucht werden, ob die verwendeten Gesten einfach zu erlernen sind.

Für diesen Teil des Experiments kommt die proView-Anwendung zum Einsatz, die nach erfolgreicher Gestenausführung das Prozessmodell nun auch visuell verändert (siehe Abb. 7.4). Mit dem Start-Button im rechten, oberen Bereich der Benutzerschnittstelle, können die Testpersonen den Versuch starten und zur nächsten Aufgabe gelangen. Die Anwendung blendet jede einzelne Aufgabenstellung textuell ein und blendet sie wieder aus, sobald die Testperson die Aufgabe nach dem Lesen antippt.

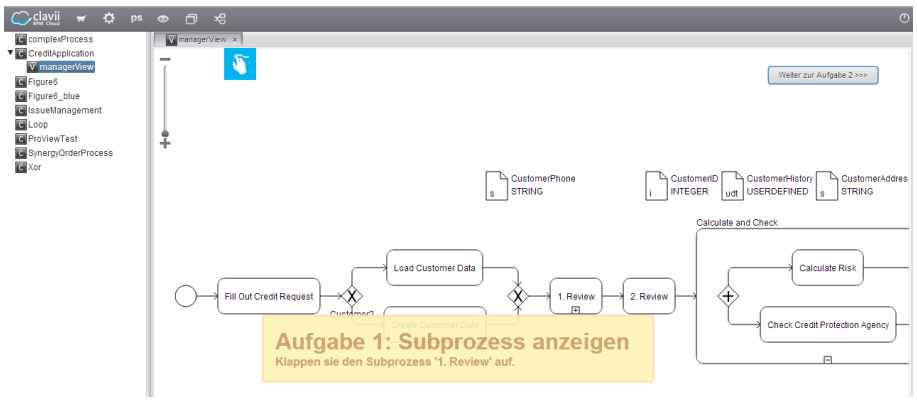

Abbildung 7.4: Benutzeroberfläche von proView.

Startzeitpunkt einer Aufgabe, sowie ausgeführte Gesten sendet die Anwendung per WLAN an den proView-Server, der die Daten in einer XML-Datei für die spätere Auswertung speichert. Die Testpersonen haben bei unklaren Aufgaben die Möglichkeit Fragen an den Versuchsleiter zu stellen.

Implementierte Gesten, an die sich eine Testperson nicht mehr erinnern kann, werden ebenfalls vom Versuchsleiter für die spätere Auswertung notiert. Anschließend werden die Testpersonen auf die richtige Lösung hingewiesen.

Am Ende des Experiments werden die Testpersonen aufgefordert, ein Feedback zu den implementierten Gesten abzugeben. Hier sind insbesondere die Gesten wichtig, an die sich eine Testperson im Laufe des dritten Teils nicht mehr erinnert. Feedback und Verbesserungsvorschläge werden schriftlich festgehalten.

7.2 Auswertung der experimentellen Untersuchung

An dem Experiment nehmen insgesamt elf Testpersonen teil, von denen eine Testperson weiblich ist. Neun Testpersonen sind Studenten, eine Testperson ist Mitarbeiter des Instituts für Datenbanken und Informationssysteme der Universität Ulm und eine weitere Testperson ist Mitarbeiter des Kommunikations- und Informationszentrums (KIZ) der Universität Ulm. Das Durchschnittsalter aller Testpersonen liegt bei 26,27 Jahren (siehe Abb. 7.5). An diesem Experiment nimmt ein Linkshänder teil. Alle restlichen Testpersonen sind Rechtshänder.

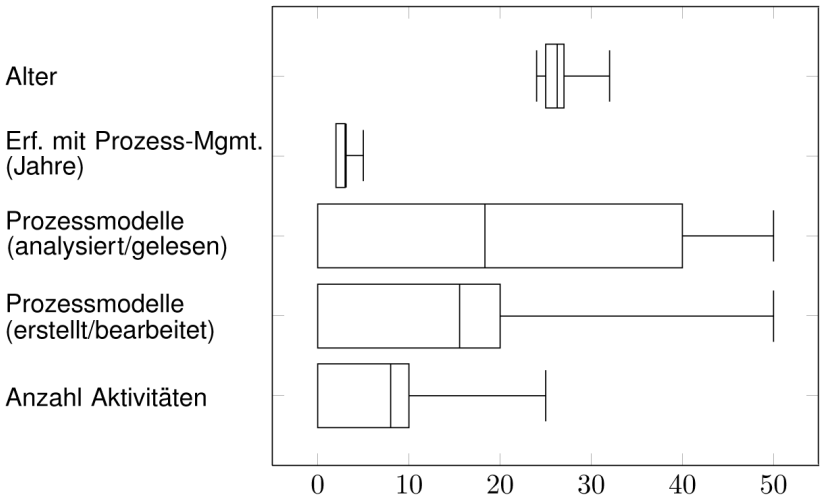

Abbildung 7.5: Demografische Daten der Testpersonen.

Alle Testpersonen haben bereits Erfahrungen mit Multi-Touch-Geräten gesammelt. Neun Testpersonen besitzen ein Smartphone und sechs Testpersonen ein Tablet. Als Betriebssystem kommt auf diesen Geräten überwiegend Android zum Einsatz. Abbildung 7.6 zeigt eine Verteilung der eingesetzten Multi-Touch-Geräte und Betriebssysteme.

Neun Testpersonen kennen sich bereits mit Prozess-Management und Prozessmodellierung aus. Davon halten sich fünf Testpersonen für Experten in diesem Bereich. Im Durchschnitt haben die Testpersonen 3,11 Jahre Erfahrung mit Prozess-Management. In den letzten zwölf Monaten vor dem Experiment sind von den Testpersonen im Durchschnitt 18,33 Prozessmodelle analysiert und gelesen worden. In derselben Zeit bearbeiteten und erstellten die Testpersonen durchschnittlich 15,56 Prozessmodelle. Diese Prozessmodelle besitzen eine durchschnittliche Anzahl von acht Aktivitäten.

Insgesamt führen die Testpersonen während der drei Teile $11 * 3 * 18 = 594$ Ge-

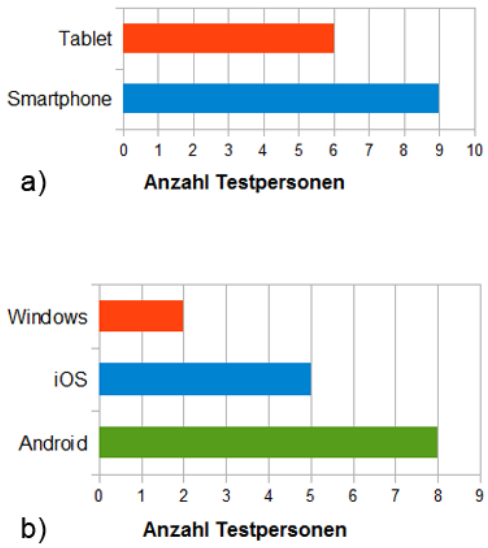

Abbildung 7.6: Verwendete Multi-Touch-Geräte a) und Betriebssysteme b).

sten aus. Ungültige Eingaben werden nicht mitgezählt. Der erste Teil, bei dem sich die Testpersonen eigene Gesten haben einfallen lassen, beinhaltet insgesamt $11 * 18 = 198$ Gesten, von denen 54 (d.h. 27,27%) symbolische Gesten sind, 100 (d.h. 50,51%) sind physikalische Gesten und bei 44 (d.h. 22,22%) handelt es sich um Menü-basierte Lösungen. Tabelle 7.2 zeigt eine Verteilung der verwendeten Gesten bezogen auf die acht Modellierungsfunktionen, die auch in [23] Bestandteil sind. In einem direkten Vergleich der Ergebnisse beider Experimente und unter Beachtung der entsprechenden Modellierungsfunktionen, ist deutlich zu erkennen, dass die Verteilung der Gesten nahezu identisch ist.

	Ermittelte Werte	[23]
Symbolische Gesten	23,87%	27,40%
Physikalische Gesten	45,45%	44,72%
Menü-basierte Lösungen	30,68%	27,88%

Tabelle 7.2: Gesten-Verteilung im Vergleich.

Für die Bearbeitung von Teil 1 benötigen die Testpersonen im Durchschnitt 10,16 Minuten. Im dritten Teil sind es 9,98 Minuten im Durchschnitt. Im Anhang A.3 sind weitere Tabellen und Diagramme zu den Ergebnissen dieses Experiments zu finden.

7.2.1 Detaillierte Auswertung der einzelnen Aufgaben

Im Folgenden findet eine detaillierte Auswertung jeder einzelnen Aufgabe statt. Es handelt sich hierbei um die Aufgaben bzw. Modellierungsfunktionen F1 bis F18, die in den drei Teilen des Experiments in unterschiedlichen Reihenfolgen gestellt werden (siehe Tabelle 7.1). Teil 2 des Experiments, das ausschließlich das Tutorial beinhaltet, wird aus der Analyse ausgeschlossen, da sich hier keine besonders aussagekräftigen Daten ermitteln lassen.

Pro ausgewerteter Aufgabe stellt eine Tabelle die Verteilung der Bedienkonzepte dar, die in Teil 1 des Experiments ermittelt worden sind. Weitere relevante Tabellen befinden sich im Anhang dieser Arbeit, wie die Bearbeitungsdauern von Teil 1 und Teil 2 des Experiments (siehe Tabelle A.6 und A.7) oder eine tabellarische Darstellung, wie häufig eine bestimmte Aufgabe in Teil 3 gelöst worden ist (siehe Tabelle A.5). Auf diese Tabellen wird des Öfteren während der detaillierten Auswertung verwiesen.

7.2.1.1 F1 (Element auswählen)

Bei dieser Aufgabe müssen die einzelnen Testpersonen die Aktivität „Load Customer Data“ des Prozessmodells auswählen. Es gibt so gut wie keine Verständnisprobleme dieser Aufgabe, daher benötigen alle elf Testpersonen durchschnittlich 16,64 Sekunden, um diese Aufgabe im ersten Teil zu bearbeiten (siehe Abb. 7.7). In dieser Zeit ist das Lesen und Verstehen der Aufgabe, sowie das Ausführen der Geste enthalten. Im Vergleich zu den anderen Aufgaben müssen die Testpersonen nicht lange überlegen, um sich eine Geste einfallen zu lassen. Dies wird nochmals mit Tabelle A.6 verdeutlicht.

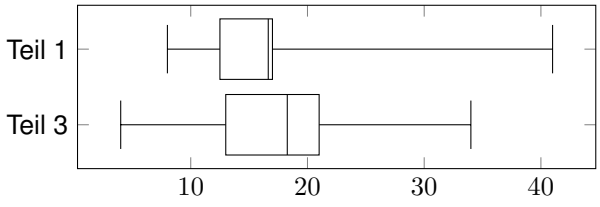

Abbildung 7.7: Bearbeitungszeiten (in Sek.) von Aufgabe F1 (Element auswählen).

Alle elf Testpersonen bevorzugen bei dieser Aufgabe in Teil 1 des Experiments, die Verwendung einer physikalischen Geste (siehe Tabelle 7.3). Dabei handelt es sich in allen elf Fällen um die Tap-Geste, die auch in der Implementierung eingesetzt wird.

	Ermittelte Werte
Symbolische Gesten	0%
Physikalische Gesten	100%
Menü-basierte Lösungen	0%

Tabelle 7.3: Gesten-Verteilung beim Auswählen von Prozesselementen.

Aufgrund dieses Resultats können sich in Teil 3 des Experiments 10 Testpersonen an die implementierte Tap-Geste erinnern. Sie lösen die Aufgabe in durchschnittlich 18,27 Sekunden (siehe Tabelle A.7). Die Geste hat sich somit für das Auswählen von Elementen bewährt.

7.2.1.2 F2 (Aktivität einfügen)

Die Testpersonen werden bei dieser Aufgabe aufgefordert, den gegebenen Prozess um eine weitere Aktivität zu ergänzen. Diese Aktivität muss zwischen dem Subprozess „1. Review“ und der Aktivität „2. Review“ eingefügt werden. Der Name für die neue Aktivität ist „A“.

Bei dieser Aufgabe gibt es keine Verständnisprobleme. Die durchschnittliche Bearbeitungszeit beträgt 29,64 Sekunden in Teil 1, was sehr nahe am durchschnittlichen Wert von 33,87 Sekunden aller Aufgaben liegt (siehe Abb. 7.8).

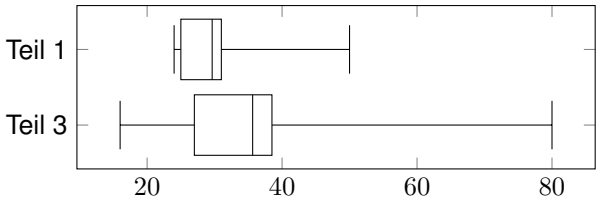

Abbildung 7.8: Bearbeitungszeiten (in Sek.) von Aufgabe F2 (Aktivität einfügen).

Zwei Testpersonen setzen die neue Aktivität mithilfe einer symbolischen Geste um. Sie zeichnen entweder ein X- oder ein Kreuz-Symbol auf die Sequenzflusskante zwischen Subprozess und Aktivität.

Vier Testpersonen bevorzugen bei dieser Aufgabe eine Menü-basierte Lösung. Hierzu tippen sie zunächst die Sequenzflusskante zwischen dem Subprozess und der Aktivität an, um ein Kontextmenü zu öffnen. Anschließend setzen sie die neue Aktivität mithilfe des Kontextmenüs ein.

Die Mehrheit, bestehend aus fünf Testpersonen, entscheidet sich für eine physikalische Geste. Diese besteht zum einen aus einer Hold-Geste, die auf der Sequenz-

flusskante ausgeführt wird. Zum anderen aus einer einfachen Strich-Geste, die die Testpersonen mit einer Bewegung von oben nach unten zwischen den zwei bestehenden Prozesselementen ausführen. Diese Lösung ist mit der implementierten Geste identisch.

In [23] ist ebenfalls eine experimentelle Untersuchung dieser Modellierungsfunktion enthalten. Hier entscheidet sich die Mehrheit der Testpersonen allerdings für eine Menü-basierte Lösung und die wenigsten Testpersonen bevorzugen physikalische Gesten, wie Tabelle 7.4 zeigt. Auch symbolische Gesten überwiegen, wie das Einzeichnen eines Rechtecks auf oder über die Sequenzflusskante.

	Ermittelte Werte	[23]
Symbolische Gesten	18,18%	34,62%
Physikalische Gesten	45,46%	23,08%
Menü-basierte Lösungen	36,36%	42,31%

Tabelle 7.4: Gesten-Verteilung beim Einfügen einer neuen Aktivität.

Alle Testpersonen erkennen die implementierte Strich-Geste dennoch als sinnvoll an und können sich daher problemlos an diese Geste im dritten Teil des Experiments erinnern. Zum Lösen der Aufgabe benötigen die Testpersonen bei diesem Teil im Durchschnitt 35,64 Sekunden und somit überdurchschnittlich viel Zeit. Im Wesentlichen ist eine lange Überlegungsdauer bei dieser Aufgabe für die lange Bearbeitungsdauer der Aufgabe verantwortlich.

Trotz der Unterschiede zwischen den beiden Experiment-Teilen, akzeptieren die Testpersonen die implementierte Geste. Intuitiv bevorzugen sie allerdings eine andere Geste.

7.2.1.3 F3 (Element umbenennen)

Im ersten Teil des Experiments haben die Testpersonen an dieser Stelle die Aufgabe, die Aktivität „2. Review“ umzubenennen. Teil 3 beinhaltet die Umbenennung der in Aufgabe F2 erstellten Aktivität.

Die Testpersonen benötigen im ersten Teil im Durchschnitt 25,45 Sekunden für diese Aufgabe (siehe Abb. 7.9). Zwei Testpersonen bevorzugen bei dieser Aufgabe den Einsatz eines Kontextmenüs. Sie tippen dabei die bestehende Aktivität einmal an und erwarten daraufhin ein Menü. Die restlichen neun Testpersonen entscheiden sich für eine physikalische Geste. Dabei lösen sie die Umbenennen-Funktion entweder mit einer einfachen Tap-Geste oder mit einer horizontalen Strich-Geste

auf der Aktivität aus. Sechs der neun Testpersonen wählen die Double-Tap-Geste, die auch für die proView-Erweiterung genutzt wird.

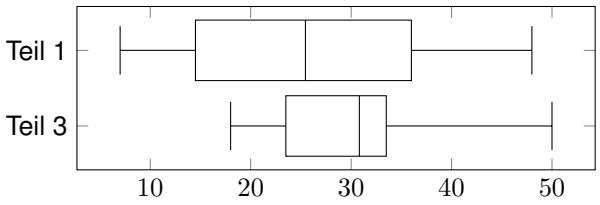

Abbildung 7.9: Bearbeitungszeiten (in Sek.) von Aufgabe F3 (Element umbenennen).

Der Vergleich mit [23] zeigt, dass sich die Testpersonen in diesem Experiment ebenfalls zum größten Teil für eine physikalische Geste entscheiden, die unter anderem eine Tap-, sowie eine Double-Tap-Geste beinhaltet (siehe Tabelle 7.5).

	Ermittelte Werte	[23]
Symbolische Gesten	0%	30,77%
Physikalische Gesten	81,82%	61,54%
Menü-basierte Lösungen	18,18%	7,69%

Tabelle 7.5: Gesten-Verteilung beim Umbenennen einer Aktivität.

Im dritten Teil des Experiments können sich vier von elf Testpersonen nicht an die implementierte Double-Tap-Geste erinnern. Drei dieser Testpersonen bevorzugen aber bereits im ersten Teil eine andere Geste. Das Feedback der Testpersonen, die sich an die implementierte Geste nicht erinnern können, fällt dennoch positiv aus.

Viel Zeit wird zum Lösen der Aufgabe in Teil 3 ebenfalls nicht benötigt. Wie in Tabelle A.7 zu sehen ist, brauchen die Testpersonen im Durchschnitt 30,82 Sekunden und daher weniger Zeit als für die meisten anderen Aufgaben.

Die Wahl der Double-Tap-Geste für das Umbenennen einer Aktivität hat sich mit diesem Experiment bestätigt.

7.2.1.4 F4 (Element löschen)

Nachdem in den vorherigen Aufgaben eine neue Aktivität erstellt und umbenannt worden ist, muss sie in Teil 3 des Experiments mit dieser Aufgabe wieder gelöscht

werden. Im ersten Teil des Experiments müssen die Testpersonen die Aktivität „2. Review“ löschen.

Zum Lösen dieser Aufgaben brauchen die Testpersonen 28,73 Sekunden im Durchschnitt (siehe Abb. 7.10).

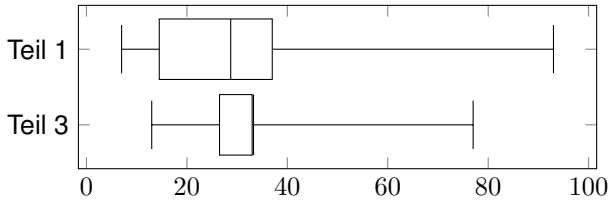

Abbildung 7.10: Bearbeitungszeiten (in Sek.) von Aufgabe F4 (Element löschen).

Eine Testperson entscheidet sich ein Double-Tap auf die Aktivität anzuwenden, woraufhin sich ein Kontextmenü öffnet, mit dem die Aktivität gelöscht werden kann.

Sechs Testpersonen und damit die Mehrheit bevorzugen es, die Aktivität aus dem Bearbeitungsbereich zu ziehen. Sie platzieren dabei ihren Finger auf die Aktivität und führen eine Drag & Drop-Bewegung Richtung oberen oder unteren Bildschirmrand aus.

Die restlichen vier Testpersonen entscheiden sich für eine symbolische Geste in Form eines X-Symbols und somit für die implementierte Geste. Die Testpersonen setzen allerdings alle beim Zeichnen ihren Finger ab, was der einzige Unterschied zur implementierten Geste ist.

In [23] entscheidet sich die Mehrheit ebenfalls für eine physikalische Geste (siehe Tabelle 7.6). Auch bei diesem Experiment kommt die Drag & Drop-Bewegung Richtung Bildschirmrand zum Einsatz. Zu diesen physikalischen Gesten zählen aber auch diagonale Drag & Drop-Bewegungen, die das Durchstreichen einer Aktivität symbolisieren sollen. Das X-Symbol kommt ebenfalls in einigen Fällen vor, unter anderem auch in Kombination mit einem Kontextmenü.

	Ermittelte Werte	[23]
Symbolische Gesten	36,36%	7,69%
Physikalische Gesten	54,55%	50,00%
Menü-basierte Lösungen	9,09%	42,31%

Tabelle 7.6: Gesten-Verteilung beim Löschen einer Aktivität.

Die Testpersonen tun sich mit der implementierten Geste schwer. Nur acht Testpersonen können sich im dritten Teil an die Geste erinnern, was sie mit der et-

was gewöhnungsbedürftigen, durchgehenden Form des X-Symbols begründen. Es wird deshalb auch im Vergleich zu den anderen Aufgaben mit Durchschnittlich 33,27 Sekunden mehr Zeit benötigt, um die Aufgabe im dritten Teil zu lösen. Dennoch äußern viele der Testpersonen auch positive Kritik zu der eingesetzten Geste.

Eine Implementierung der bevorzugten Drag & Drop-Bewegung Richtung unteren Bildschirmrand ist in proView möglich. Allerdings muss darauf geachtet werden, dass man beispielsweise beim schlechten Einfügen einer Synchronisationskante, unbeabsichtigt eine Aktivität löschen könnte, da die Gesten ähnlich zueinander sind. Dann ist der Einsatz einer vorherigen Sicherheitsabfrage von Vorteil.

7.2.1.5 F5 (Elemente aggregieren)

Die Aufgabe im ersten Teil des Experiments ist es, die Aktivität „Fill Out Credit Request“ und den ersten XOR-Verzweigungsblock, der aus der Aktivität „Load Customer Data“ und „Create Customer Data“ besteht, zu einem Subprozess zusammenzufassen. Im dritten Teil muss der einzige AND-Verzweigungsblock aggregiert werden.

Es gibt keine Verständnisprobleme mit dieser Aufgabe, dennoch haben viele Testpersonen im ersten Teil Probleme diese Aufgabe zu lösen, da ihnen nicht sofort eine Geste einfällt. Aus diesem Grund beträgt die durchschnittlich benötigte Zeit 39,00 Sekunden und liegt damit über der durchschnittlichen Zeit aller Aufgaben (siehe Abb. 7.11).

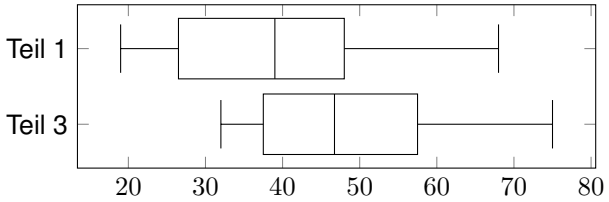

Abbildung 7.11: Bearbeitungszeiten (in Sek.) von Aufgabe F5 (Elemente aggregieren).

Für die Aggregation der entsprechenden Prozesselemente verwendet eine Testperson eine obere Menüleiste, um die Modellierungsfunktion zunächst zu aktivieren. Anschließend wählt sie mittels Tap-Geste die Prozesselemente aus, die aggregiert werden sollen.

Vier Testpersonen verwenden eine physikalische Geste. In drei Fällen wird die

Pinch-Geste verwendet, indem die Testpersonen zwei Finger auf den zu aggregierenden Prozesselementen zusammenziehen. Bei dieser Ausführung fehlt allerdings eine gewisse Genauigkeit bei der Auswahl der entsprechenden Prozesselemente.

Die Mehrheit, bestehend aus sechs Testpersonen, führt eine symbolische Geste aus. Dabei zeichnen zwei Testpersonen einen Kreis und vier Testpersonen einen Rechteck um die zu aggregierenden Prozesselemente.

In [23] werden ebenfalls mit 73,08% hauptsächlich Kreise und Rechtecke zum Aggregieren verwendet (siehe Tabelle 7.7). Zu beachten ist, dass [23] diese Symbole zu einer physikalischen Geste zählt, wohingegen es sich in diesem Experiment um symbolische Gesten handelt. Wenn dies berücksichtigt wird, sind die Ergebnisse nahezu identisch.

	Ermittelte Werte	[23]
Symbolische Gesten	54,55%	11,54%
Physikalische Gesten	36,36%	73,08%
Menü-basierte Lösungen	9,09%	15,38%

Tabelle 7.7: Gesten-Verteilung beim Aggregieren von Elementen.

Die Testpersonen benötigen in Teil 3 des Experiments im Durchschnitt 46,73 Sekunden, um die Aufgabe zu lösen. Dies ist ein schlechter Wert (siehe Tabelle A.7). Erinnern können sich allerdings alle Testpersonen an die implementierte Geste. Der Grund für die schlechte Zeit liegt darin, dass einige Testpersonen über den mehrmaligen Einsatz des Rechteck-Symbols für die Modellierungsfunktionen verwundert sind und bei dieser Aufgabe zunächst an eine andere Geste denken.

7.2.1.6 F6 (Element reduzieren)

Diese Aufgabe beinhaltet das Reduzieren von Prozesselementen. Dabei muss in Teil 1 der expandierte Subprozess „Calculate and Check“ und in Teil 3 die Aktivität „Fill Out Credit Request“ reduziert werden.

Für Teil 1 benötigen die Testpersonen im Durchschnitt 40,91 Sekunden (siehe Abb. 7.12). Das liegt unter anderem daran, dass vielen Testpersonen trotz Erfahrung mit Prozess-Management zu dem Zeitpunkt der Begriff „Reduzieren“ nicht geläufig ist.

Drei Testpersonen bevorzugen den Einsatz eines Kontextmenüs. Sie wählen die entsprechenden Prozesselemente aus und rufen die Funktion zum Reduzieren in einem Kontextmenü auf.

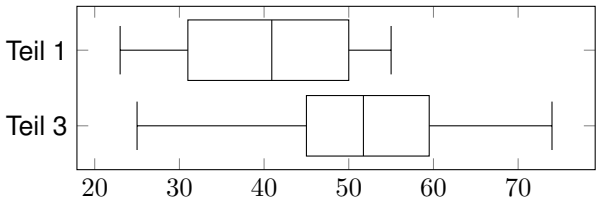

Abbildung 7.12: Bearbeitungszeiten (in Sek.) von Aufgabe F6 (Element reduzieren).

Symbolische Lösungen kommen überhaupt nicht zum Einsatz, sondern im Wesentlichen physikalische Gesten. Die am meisten ausgeführte Geste ist dabei die Pinch-Geste. Die Testpersonen ziehen auf dem zu reduzierenden Prozesselement ihre Finger zusammen, um auf diese Weise die Reduktions-Funktion auszulösen. Hierbei beachten sie allerdings nicht, wie die Geste auf mehrere Prozesselemente gleichzeitig angewendet werden könnte. In so einem Fall gibt es mit der Pinch-Geste Probleme, da eine gewisse Auswahlgenauigkeit fehlt. Eine Verteilung der Gesten für diese Funktion ist in Tabelle 7.8 zu sehen.

	Ermittelte Werte
Symbolische Gesten	0%
Physikalische Gesten	72,73%
Menü-basierte Lösungen	27,27%

Tabelle 7.8: Gesten-Verteilung bei der Reduktion.

In Teil 3 des Experiments benötigen die Testpersonen 51,73 Sekunden. Auch hier gibt es erhebliche Schwierigkeiten sich an die Geste zu erinnern. Viele können sich teilweise an das Rechteck erinnern, vergessen aber des Öfteren die zusätzliche Diagonale. Nur eine Testperson kann sich an die implementierte, symbolische Geste erinnern.

Die Testpersonen geben überwiegend an, dass die Geste schlecht zu merken und nicht intuitiv ist. Andere kritisieren die lange Ausführungsdauer der Geste. Um eine Verbesserung zu erreichen, kann auf die zusätzliche Diagonale verzichtet werden, sodass nur noch ein Rechteck um die zu reduzierenden Prozesselemente gezeichnet werden muss. Dies bedeutet allerdings auch, dass sich in dem anschließend öffnenden Kontextmenü ein weiterer Eintrag für die Reduktions-Funktion befinden muss.

7.2.1.7 F7 (Verzweigungsblock einfügen)

Die Testpersonen haben die Aufgabe einen neuen XOR-Verzweigungsblock zu erstellen. In Teil 1 des Experiments soll die Aktivität „Fill Out Credit Request“ Bestandteil des Verzweigungsblocks sein, d.h. sich auf einem XOR-Zweig befinden. In Teil 3 soll sich der Subprozess „1. Review“ und die Aktivität „2. Review“ im Verzweigungsblock befinden.

Für diese Aufgabe wird am meisten Zeit benötigt. Im Durchschnitt vergehen 60,91 Sekunden, bis die Testpersonen die Aufgabe lösen (siehe Abb. 7.13). Dabei haben sie keine Probleme die Aufgabe zu verstehen, sondern sich eine passende Geste für diese Modellierungsfunktion zu überlegen.

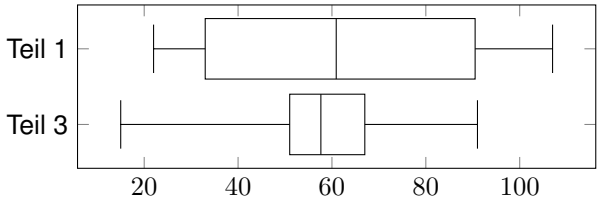

Abbildung 7.13: Bearbeitungszeiten (in Sek.) von Aufgabe F7 (Verzweigungsblock einfügen).

Vier Testpersonen zeichnen den öffnenden und den schließenden XOR-Verzweigungsknoten auf die Sequenzflusskante. Die verwendeten Symbole sind Kreise, X-Symbole sowie ein horizontales, halbes Rechteck.

Zwei Testpersonen setzen die Verzweigungsknoten mithilfe einer Hold-Geste ein, indem sie die entsprechenden Sequenzflusskanten für eine gewisse Dauer berühren.

Die Mehrheit der Testpersonen entscheidet sich für eine Menü-basierte Lösung. Dabei tippen sie die Sequenzflusskanten an, um ein Kontextmenü zu öffnen, mit dem sie die Verzweigungsknoten einsetzen können.

Das Ergebnis in [23] ist für diese Modellierungsfunktion ziemlich ausgeglichen (siehe Tabelle 7.9). Es entscheiden sich genauso viele Testpersonen für eine symbolische wie auch für eine Menü-basierte Lösung. Die Lösungen enthalten eine große Vielzahl an unterschiedlichen Symbolen und Gesten, sodass sich keine eindeutige Lösung herauskristallisiert. Lediglich die physikalischen Gesten werden in beiden Experimenten etwas weniger eingesetzt.

Wie in Teil 1 des Experiments, benötigen die Testpersonen auch in Teil 3 mit im Durchschnitt 57,64 Sekunden viel Zeit, um die Aufgabe zu lösen. In fünf Fällen

	Ermittelte Werte	[23]
Symbolische Gesten	36,36%	34,62%
Physikalische Gesten	18,19%	30,76%
Menü-basierte Lösungen	45,45%	34,62%

Tabelle 7.9: Gesten-Verteilung beim Einfügen eines Verzweigungsblocks.

scheitern die Testpersonen hierbei. Als Grund geben sie an, dass sie das Rechteck-Symbol zum Auswählen der entsprechenden Prozesselemente zwar vermuten, es jedoch nicht anwenden, da es ebenfalls für andere Modellierungsfunktionen zum Einsatz kommt. Verbesserungsvorschläge gibt es jedoch keine. Schlussendlich wird das Rechteck-Symbol zum Öffnen eines Kontextmenüs, das die entsprechende Modellierungsfunktion enthält, von den Testpersonen akzeptiert und als sinnvoll befunden.

7.2.1.8 F8 (Verzweigung einfügen)

In den ersten XOR-Verzweigungsblock, der aus zwei XOR-Zweigen und den Aktivitäten „Load Customer Data“ und „Create Customer Data“ besteht, sollen die Testpersonen einen neuen XOR-Zweig einsetzen. Hierfür benötigen sie im Durchschnitt 32,64 Sekunden (siehe Abb. 7.14).

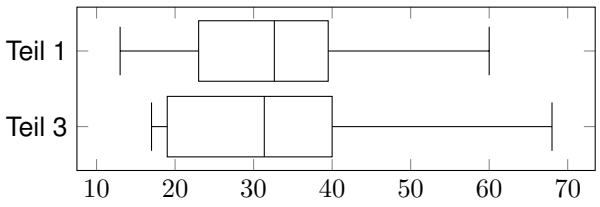

Abbildung 7.14: Bearbeitungszeiten (in Sek.) von Aufgabe F8 (Verzweigung einfügen).

Eine Testperson führt eine symbolische Geste aus, indem sie einen Bogen vom öffnenden Verzweigungsknoten zum schließenden Verzweigungsknoten oberhalb des XOR-Verzweigungsblocks zeichnet.

Die Mehrheit bevorzugt den Einsatz eines Kontextmenüs, das nach dem Antippen des öffnenden Verzweigungsknotens erscheint und die Modellierungsfunktion zum Einfügen eines neuen Zweigs anbietet.

Vier Testpersonen entscheiden sich eine Strich-Geste von links nach rechts auszuführen, deren Startposition der öffnende Verzweigungsknoten ist und in zwei

Fällen der schließende Verzweigungsknoten als Endposition dient. Die anderen zwei Testpersonen machen sich Gedanken darüber, dass in manchen Prozessmodellen der schließende Verzweigungsknoten aufgrund eines kleinen Bildschirms eventuell nicht sichtbar ist und beenden die Strich-Geste deshalb nur wenige Zentimeter nach dem öffnenden Knoten, um auf diese Weise einen Zweig anzudeuten. Diese Lösung ist ähnlich zu der implementierten Geste, die die Form eines nach oben gerichteten Pfeils besitzt.

Werden die hier ermittelten Werte mit denen in [23] verglichen, ist deutlich ein Unterschied erkennbar (siehe Tabelle 7.10). In [23] entscheiden sich 46,15% für eine physikalische Geste, die im Wesentlichen aus dieser bereits erläuterten Strich-Geste besteht.

	Ermittelte Werte	[23]
Symbolische Gesten	9,09%	30,77%
Physikalische Gesten	36,36%	46,15%
Menü-basierte Lösungen	54,55%	23,08%

Tabelle 7.10: Gesten-Verteilung beim Einfügen einer neuen Verzweigung.

Trotz der ungewöhnlichen Form können sich dennoch neun Testpersonen an das implementierte Pfeil-Symbol erinnern. Zeitlich wird die Aufgabe in Teil 3 des Experiments in 31,36 Sekunden gelöst, wie Tabelle A.7 beweist.

Besonders negative Kritik gibt es in Bezug auf die Pfeilspitze. Fast alle Testpersonen empfehlen die Pfeilspitze wegzulassen, da sie schwierig zu zeichnen ist, ohne den Finger abzusetzen. Ohne Pfeilspitze verbleibt eine einfache Strich-Geste, die wie bereits erwähnt, vier Testpersonen in Teil 1 des Experiments verwenden.

7.2.1.9 F9 (Schleife einfügen)

Die Testpersonen haben die Aufgabe in den Prozess eine neue Schleife einzusetzen. Diese Schleife soll nach der Aktivität „2. Review“ beginnen und vor dem XOR-Split des ersten XOR-Verzweigungsblocks im Prozessmodell enden.

Im Durchschnitt benötigen die Testpersonen 42,36 Sekunden für diese Aufgabe (siehe Abb. 7.15). Einige Sekunden sind notwendig, um die richtigen Stellen, an denen die Schleife eingesetzt werden soll, zu finden. Die restliche Zeit benötigen die Testpersonen für das Ausdenken einer passenden Geste.

Zwei Testpersonen entscheiden sich, die Schleife über eine Menü-basierte Lösung zu realisieren. Dabei verwenden sie ein Kontextmenü, das sie über ein einfaches

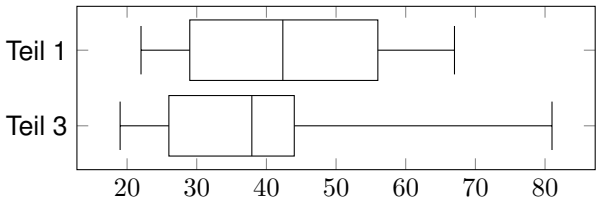

Abbildung 7.15: Bearbeitungszeiten (in Sek.) von Aufgabe F9 (Schleife einfügen).

Antippen oder langes Berühren der entsprechenden Sequenzflusskanten öffnen. Dieses Kontextmenü enthält eine Modellierungsfunktion zum Einsetzen des Startknotens der Schleife, sowie eine Modellierungsfunktion zum Einsetzen des Endknotens der Schleife.

Physikalische Gesten kommen nicht zum Einsatz, sondern im Wesentlichen symbolische Gesten (siehe Tabelle 7.11). Die restlichen Testpersonen zeichnen hierbei die Schleife in das Prozessmodell ein. Diese Lösung ist mit der implementierten Geste identisch. Allerdings gibt es bei der Gestendurchführung zwei unterschiedliche Ansätze. Fünf der neun Testpersonen bevorzugen es, dass die Endpunkte der zu zeichnenden Schleife auf der Aktivität „2. Review“ und dem XOR-Split liegen, wohingegen die anderen Testpersonen nach der Aktivität beginnen zu zeichnen bzw. vor dem XOR-Split mit dem Zeichnen aufhören. Dennoch können sich alle Testpersonen in Teil 3 des Experiments an die Geste erinnern und wählen den richtigen Ansatz zum Zeichnen, d.h. platzieren Start- und Endpunkt der Schleife auf den entsprechenden Prozesselementen. Die hierfür benötigte Durchschnittszeit liegt bei 37,91 Sekunden und ist aufgrund der sehr guten Erinnerungsrate ein akzeptabler Wert.

	Ermittelte Werte
Symbolische Gesten	81,82%
Physikalische Gesten	0%
Menü-basierte Lösungen	18,18%

Tabelle 7.11: Gesten-Verteilung beim Einfügen einer Schleife.

Das Ergebnis dieser Aufgabe zeigt, dass die implementierte Geste zur Modellierungsfunktion passt.

7.2.1.10 F10 (Synchronisationskante einfügen)

Die Testpersonen müssen bei dieser Aufgabe in den AND-Verzweigungsblock des Prozessmodells eine neue Synchronisationskante zwischen der Aktivität „Calculate Risk“ und „Check Credit Protection Agency“ einsetzen. Die Richtung der Kante spielt dabei keine Rolle. Die Aufgabe lösen die Testpersonen in durchschnittlich 37,45 Sekunden (siehe Abb. 7.16).

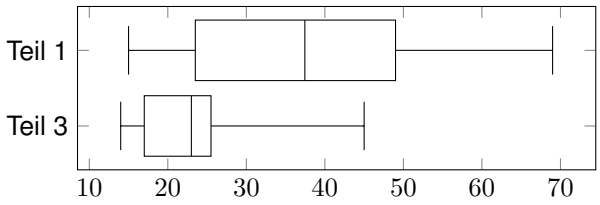

Abbildung 7.16: Bearbeitungszeiten (in Sek.) von Aufgabe F10 (Synchronisationskante einfügen).

Eine Testperson bevorzugt bei dieser Aufgabe eine Menü-basierte Lösung (siehe Tabelle 7.12). Sie tippt dabei zunächst auf die obere Aktivität, um ein Kontextmenü aufzurufen, wählt die Synchronisationskante aus und zieht sie per Drag & Drop auf die untere Aktivität.

	Ermittelte Werte
Symbolische Gesten	27,27%
Physikalische Gesten	63,64%
Menü-basierte Lösungen	9,09%

Tabelle 7.12: Gesten-Verteilung beim Einfügen einer Synchronisationskante.

Drei Testpersonen bevorzugen eine symbolische Geste, indem sie eine gestrichelte Synchronisationskante einzeichnen. In einem Fall wird die Kante durch eine Pfeilspitze ergänzt.

Die restlichen sieben Testpersonen entscheiden sich bei dieser Aufgabe für eine physikalische Geste. Sie führen eine Strich-Geste von der oberen zur unteren Aktivität aus, um die Synchronisationskante auf diese Weise einzusetzen. Diese Geste wird auch in der proView-Erweiterung eingesetzt. Daher lösen alle Testpersonen dieselbe Aufgabe in Teil 3 des Experiments problemlos. Sie benötigen hierfür durchschnittlich 23,00 Sekunden.

7.2.1.11 F11 (Subprozess anzeigen)

Das Prozessmodell enthält einen Subprozess „1. Review“, der sich im kollabierten bzw. zugeklappten Zustand befindet. Bei dieser Aufgabe gilt es nun, diesen Subprozess mittels einer Geste in den expandierten Zustand zu überführen.

Die Testpersonen lösen diese Aufgabe in Teil 1 des Experiments mit durchschnittlich 14,36 Sekunden (siehe Abb. 7.17). Eine Testperson führt dabei zum Aufklappen des Subprozesses eine symbolische Geste in Form eines Häkchens aus. Die restlichen zehn Testpersonen entscheiden sich, eine physikalische Geste zu verwenden (siehe Tabelle 7.13). Es handelt sich hierbei um eine Pinch-, Double-Tap und eine Hold-Geste. Die Mehrheit führt allerdings eine gewöhnliche Tap-Geste auf dem Plus-Symbol des Subprozesses aus, um diesen aufzuklappen.

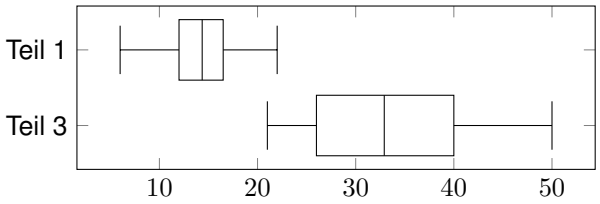

Abbildung 7.17: Bearbeitungszeiten (in Sek.) von Aufgabe F11 (Subprozess anzeigen).

	Ermittelte Werte
Symbolische Gesten	9,09%
Physikalische Gesten	90,91%
Menü-basierte Lösungen	0%

Tabelle 7.13: Gesten-Verteilung beim Anzeigen eines Subprozesses.

In Teil 3 des Experiments benötigen die Testpersonen für dieselbe Aufgabe Durchschnittlich 32,91 Sekunden. Neun von insgesamt elf Testpersonen führen die implementierte Geste richtig aus.

Beinahe alle Testpersonen sind mit der Wahl der implementierten Geste zufrieden. Es gibt eine Kritik zur langen Ausführungsdauer der Geste, die bei einer häufigeren Anwendung störend sein kann.

7.2.1.12 F12 (Subprozess verbergen)

In Teil 1 des Experiments müssen die Testpersonen bei dieser Aufgabe den Subprozess „Calculate and Check“ zuklappen, d.h. in den kollabierten Zustand bringen. In Teil 3 handelt es sich um den Subprozess „1. Review“.

Wie bei der Aufgabe „Subprozess anzeigen“ können die Testpersonen auch diese Aufgabe mit durchschnittlich 20,09 Sekunden schnell lösen (siehe Abb. 7.18). Alle Testpersonen wählen hierbei eine physikalische Geste aus.

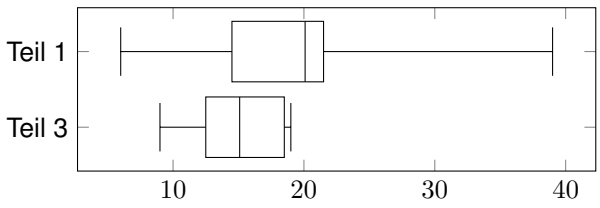

Abbildung 7.18: Bearbeitungszeiten (in Sek.) von Aufgabe F12 (Subprozess verbergen).

Vier Testpersonen klappen den Subprozess mit einer Pinch-Geste zu, d.h. sie berühren den Bildschirm mit zwei Fingern, die sie anschließend auf dem Subprozess zusammenziehen.

Sechs Testpersonen tippen mit ihrem Finger auf das Minus-Symbol des Subprozesses, das sich zentriert am unteren Rand des Prozesselements befindet.

Eine Verteilung der verwendeten Bedienkonzepte ist in Tabelle 7.14 zu sehen.

	Ermittelte Werte
Symbolische Gesten	0%
Physikalische Gesten	100%
Menü-basierte Lösungen	0%

Tabelle 7.14: Gesten-Verteilung beim Verbergen eines Subprozesses.

In Teil 3 des Experiments, wo es darum geht sich an die implementierte Hold-Geste zu erinnern, haben die Testpersonen keine großen Schwierigkeiten. Diese Aufgabe lösen alle Testpersonen mit einer durchschnittlichen Zeit von 15,09 Sekunden. Eine negative Kritik zu dieser Geste gibt es nicht.

7.2.1.13 F13 (Subprozess auflösen)

Der bestehende Subprozess „Calculate and Check“ muss von den Testpersonen bei dieser Aufgabe aufgelöst werden, sodass die im Subprozess enthaltenen Prozesselemente wieder fester Bestandteil des gesamten Prozessmodells sind.

Es gibt keine Verständnisprobleme bei dieser Aufgabe. Eine passende Geste zu finden, gestaltet sich für manche Testpersonen dennoch als schwierig. Im Durchschnitt benötigen alle Testpersonen 37,36 Sekunden für diese Aufgabe (siehe Abb. 7.19).

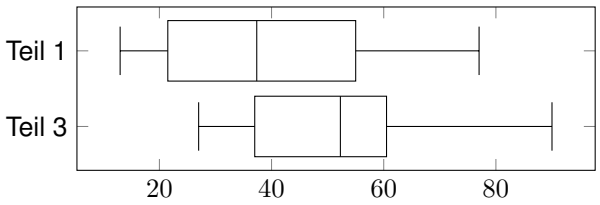

Abbildung 7.19: Bearbeitungszeiten (in Sek.) von Aufgabe F13 (Subprozess auflösen).

Drei Testpersonen führen eine symbolische Geste aus (siehe Tabelle 7.15). In einem Fall wird der Subprozess mit einem Kreuz durchgestrichen. Die anderen beiden Ansätze sind sich sehr ähnlich. Die Testpersonen zeichnen dabei die Umrandung des Subprozesses mit einer durchgehenden bzw. einer zickzackförmigen Linie nach, um diese praktisch aufzulösen.

	Ermittelte Werte
Symbolische Gesten	27,27%
Physikalische Gesten	45,46%
Menü-basierte Lösungen	27,27%

Tabelle 7.15: Gesten-Verteilung beim Auflösen eines Subprozesses.

Weitere drei Testpersonen bevorzugen den Einsatz eines Menüs. Auch hier gibt es drei unterschiedliche Ansätze. Zum Öffnen eines Kontextmenüs tippt einer der Testpersonen den Namen „Calculate and Check“ oberhalb des Subprozesses an. Ein anderer tippt den Subprozess direkt an. Im dritten Ansatz aktiviert die Testperson in einer Menüleiste die Modellierungsfunktion zum Auflösen des Subprozesses und wählt anschließend den entsprechenden Subprozess aus.

Die Mehrheit, bestehend aus fünf Testpersonen, führt eine physikalische Geste aus. Hier ergibt sich allerdings ebenfalls keine eindeutige Lösung. Drei Testperso-

nen lösen den Subprozess mit einer Hold- bzw. einer Pinch-Geste auf. Die restlichen beiden Testpersonen führen eine Drag & Drop-Bewegung aus. Der Startpunkt der Bewegung ist in einem Fall die obere, rechte Ecke des Subprozesses. Die anschließende Bewegung ist sehr kurz. Im zweiten Fall wird die Drag & Drop-Bewegung quer auf dem Subprozess ausgeführt.

Nur sieben Testpersonen können sich in Teil 3 an die implementierte Zickzack-Geste erinnern. Die durchschnittliche Bearbeitungszeit der Aufgabe liegt daher bei 52,27 Sekunden.

Einige Testpersonen finden die Geste zu kompliziert und nicht intuitiv genug. Außerdem ist die Ausführungsdauer für viele Testpersonen zu lang. Die Testpersonen bevorzugen größtenteils ihre ausgedachten Gesten, die allerdings sehr unterschiedlich ausgefallen sind, sodass es nicht möglich ist, eine eindeutige Geste für diese Modellierungsfunktion zu finden. Die implementierte Geste könnte zur Verbesserung auf einen zu zeichnenden Zacken reduziert werden, um die Ausführungsdauer und die Komplexität etwas zu reduzieren.

7.2.1.14 F14 (Prozesssicht erstellen)

Bei dieser Aufgabe müssen die Testpersonen aus dem ersten XOR-Verzweigungsblock, der aus den zwei Aktivitäten „Load Customer Data“ und „Create Customer Data“ besteht, eine neue Prozesssicht erstellen. Eine Benennung der neuen Prozesssicht wird nicht verlangt.

Zum Lösen dieser Aufgabe benötigen die Testpersonen in Teil 1 durchschnittlich 56,73 Sekunden (siehe Abb. 7.20). Dies liegt vor allem daran, dass die Testpersonen versuchen unbedingt eine Geste zu finden, die sie in den letzten Aufgaben noch nicht verwendet haben.

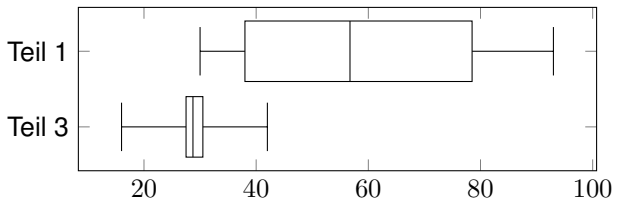

Abbildung 7.20: Bearbeitungszeiten (in Sek.) von Aufgabe F14 (Prozesssicht erstellen).

Eine Testperson erstellt eine neue Prozesssicht, indem sie eine Strich-Geste von links nach rechts auf dem Verzweigungsblock ausführt.

Menü-basierte Lösungen werden von drei Testpersonen bevorzugt (siehe Tabelle 7.16). Sie wählen zunächst die entsprechenden Prozesselemente mit einer Tap-Geste aus und lösen die Modellierungsfunktion zum Erstellen einer neuen Prozesssicht über ein Kontextmenü aus.

	Ermittelte Werte
Symbolische Gesten	63,64%
Physikalische Gesten	9,09%
Menü-basierte Lösungen	27,27%

Tabelle 7.16: Gesten-Verteilung beim Erstellen einer Prozesssicht.

Sieben Testpersonen zeichnen einen Rechteck um den XOR-Verzweigungsblock, aus dem eine neue Prozesssicht erstellt werden soll.

Im Gegensatz zu Teil 1 lösen die Testpersonen dieselbe Aufgabe in Teil 3 schneller. Im Durchschnitt benötigen sie 28,73 Sekunden, um sich an die implementierte Geste zu erinnern und diese anschließend anzuwenden. Die Aufgabe wird von allen Testpersonen gelöst. Eine negative Kritik zu der implementierten Lösung gibt es nicht. Daher sind keine Änderungen der implementierten Geste notwendig.

7.2.1.15 F15 (Datenelement einfügen)

Diese Aufgabe beinhaltet das Einfügen eines neuen Datenelements in das bestehende Prozessmodell. Die Position des neuen Datenelements wird nicht vorgegeben. Die Testpersonen benötigen nicht viel Zeit, um für diese Modellierungsfunktion eine Geste zu finden. Die durchschnittliche Zeit liegt in Teil 1 bei 28,09 Sekunden (siehe Abb. 7.21).

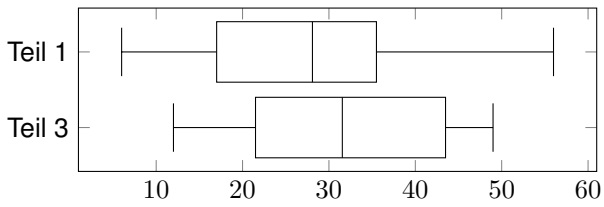

Abbildung 7.21: Bearbeitungszeiten (in Sek.) von Aufgabe F15 (Datenelement einfügen).

Zwei Testpersonen entscheiden sich bei dieser Aufgabe für eine symbolische Geste. Sie zeichnen dabei eine horizontale S-Kurve oder ein Plus-Symbol auf den freien Hintergrund.

Zwei weitere Testpersonen platzieren das neue Datenelement mit einer Hold-Geste auf den freien Hintergrund.

Die Mehrheit entscheidet sich ein Menü einzusetzen. Hier kommen ebenfalls sehr unterschiedliche Ansätze zum Einsatz. Unter Verwendung einer Tap-, Hold oder einer symbolischen Geste in Form eines X-Symbols, rufen die Testpersonen ein Kontextmenü auf, mit dem sich das neue Datenelement einsetzen lässt. Die restlichen vier Testpersonen bevorzugen die Verwendung einer oberen bzw. seitlichen Menüleiste, die ein neues Datenelement zur Verfügung stellt. Per Drag & Drop kann so das neue Datenelement eingefügt werden.

In [23] entscheidet sich die Mehrheit ebenfalls für eine Menü-basierte Lösung, um das Datenelement einzusetzen (siehe Tabelle 7.17). Es handelt sich hierbei um dieselben Ansätze, wie oben beschrieben. Symbolische Gesten, wie Kreise, Dreiecke und Rechtecke kommen ebenfalls sehr häufig vor.

	Ermittelte Werte	[23]
Symbolische Gesten	18,18%	38,46%
Physikalische Gesten	18,18%	15,38%
Menü-basierte Lösungen	63,64%	46,16%

Tabelle 7.17: Gesten-Verteilung beim Einfügen eines neuen Datenelements.

In Teil 3 können sich nur wenige Testpersonen an die implementierte Geste erinnern. Es sind insgesamt fünf Testpersonen, die die Geste korrekt ausführen. Die Testpersonen geben an, dass die symbolische Geste nicht intuitiv und zu kompliziert ist. Favorisiert werden physikalische Gesten, wie die Hold- oder Double-Tap-Geste. Aufgrund dieses Ergebnisses ist es sinnvoll für das Einsetzen eines Datenelements die Double-Tap-Geste zu verwenden, die auf dem freien Hintergrund ausgeführt wird. Aktivitäten dürfen während dieser Gestendurchführung nicht berührt werden, da dies sonst die Umbenennen-Funktion auslöst.

7.2.1.16 F16 (Hilfe aufrufen)

Die Testpersonen haben die Aufgabe einen Hilfe-Dialog aufzurufen, der alle implementierten Gesten inklusive Beschreibung aufführt.

Es werden bei dieser Aufgabe in Teil 1 sehr viele unterschiedliche Lösungen in durchschnittlich 48,64 Sekunden ausgeführt (siehe Abb. 7.22). Es handelt sich hierbei um die erste Aufgabe des Experiments, bei der sich die Testpersonen zunächst auf das Experiment einstellen müssen.

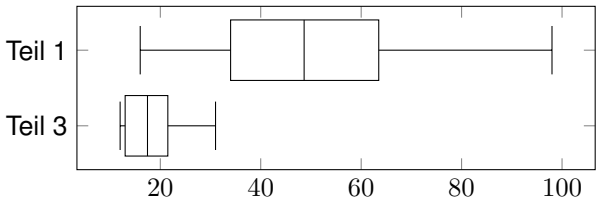

Abbildung 7.22: Bearbeitungszeiten (in Sek.) von Aufgabe F16 (Hilfe aufrufen).

Drei Testpersonen rufen den Hilfe-Dialog mithilfe einer oberen Menüleiste auf. Dabei tippen sie auf einen Button, der ein Fragezeichen-Symbol besitzt.

Physikalische Gesten werden von drei Testpersonen bevorzugt (siehe Tabelle 7.18). Sie führen eine Swipe-Geste aus, um den Hilfe-Dialog von oben nach unten oder von links nach rechts auf die Bearbeitungsfläche zu ziehen. In einem Fall werden für die Swipe-Geste drei Finger gleichzeitig benutzt.

	Ermittelte Werte
Symbolische Gesten	45,46%
Physikalische Gesten	27,27%
Menü-basierte Lösungen	27,27%

Tabelle 7.18: Gesten-Verteilung beim Aufrufen der Hilfe.

Die Mehrheit der Testpersonen entscheidet sich eine symbolische Geste zu verwenden. Eine Testperson zeichnet ein H-Symbol und die restlichen Testpersonen ein Fragezeichen-Symbol in den Bearbeitungsbereich. Das Fragezeichen-Symbol wird ebenfalls in der Implementierung eingesetzt. Die Wahl des Symbols für die Implementierung hat sich mit Teil 3 des Experiments bestätigt, da sich alle Testpersonen in diesem Teil an die Geste erinnern können und die Aufgabe in durchschnittlich 17,45 Sekunden lösen.

Einige Testpersonen machen nach dem Experiment den Vorschlag eine kontextsensitive Hilfe-Funktion zu implementieren. Wird beispielsweise das Fragezeichen auf einer Aktivität gezeichnet, zeigt der sich öffnende Hilfe-Dialog nur Modellierungsfunktionen an, die sich auf die entsprechende Aktivität anwenden lassen.

7.2.1.17 F17 (Rückgängig machen)

Bereits ausgeführte Aktionen können wieder rückgängig gemacht werden. Die Testpersonen müssen sich für diese Funktion in Teil 1 eine Geste überlegen. Aufgrund ihrer Erfahrung mit Multi-Touch-Geräten fällt den meisten Testpersonen die-

se Aufgabe nicht besonders schwer, weshalb sie im Durchschnitt nur 22,09 Sekunden benötigen (siehe Abb. 7.23). Dabei kommen allerdings sehr viele unterschiedliche Gesten zum Einsatz.

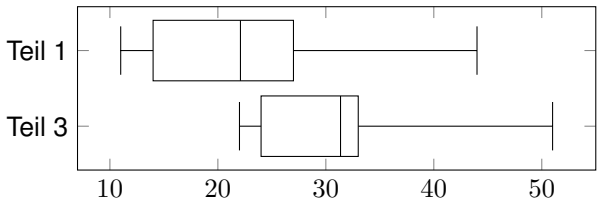

Abbildung 7.23: Bearbeitungszeiten (in Sek.) von Aufgabe F17 (Rückgängig machen).

Zwei Testpersonen bevorzugen einen einfachen Button in einer oberen Menüleiste, sowie es von aktuellen Browsern her bekannt ist.

Vier Testpersonen entscheiden sich für eine physikalische Geste (siehe Tabelle 7.19). Dabei handelt es sich in drei Fällen um eine Swipe-Geste, die entweder von links nach rechts oder umgekehrt ausgeführt wird. Eine Double-Tap-Geste kommt ebenfalls zum Einsatz.

	Ermittelte Werte
Symbolische Gesten	45,46%
Physikalische Gesten	36,36%
Menü-basierte Lösungen	18,18%

Tabelle 7.19: Gesten-Verteilung für Rückgängig machen.

Fünf symbolische Gesten werden bei dieser Aufgabe verwendet. In zwei Fällen handelt es sich um einen Kreis, den die Testpersonen gegen den Uhrzeigersinn in den Bearbeitungsbereich zeichnen. Zwei Testpersonen zeichnen ein nach links ausgerichteten Pfeil. Bei einem Pfeil ist allerdings nur die obere Hälfte der Pfeilspitze sichtbar. Eine Testperson zeichnet eine Spirale, um die Rückgängig-Funktion auszulösen.

Nachdem die Testpersonen das Tutorial in Teil 2 des Experiments absolvieren, müssen sie sich in Teil 3 an die implementierte Geste für die Funktion erinnern. Dies gelingt nur acht Testpersonen des Experiments. Die durchschnittliche Zeit zum Lösen der Aufgabe beträgt 31,36 Sekunden, was im Vergleich zu den anderen Aufgaben ein akzeptabler Wert ist.

Mit der implementierten Geste sind viele Testpersonen unzufrieden. Kritisiert wird

vor allem die zu lange Ausführungsdauer der Geste, sowie die zu hohe Komplexität. Es gibt Vorschläge die Geste durch eine Swipe-Geste nach links oder eine symbolische Geste in Form eines Kreises zu ersetzen. Dies sollte bei einer erneuten Gesten-Implementierung beachtet werden.

7.2.1.18 F18 (Lese- und Schreibkante einfügen)

Die Testpersonen müssen bei dieser Aufgabe eine Lesekante zwischen dem Datenelement „CustomerPhone“ und der Aktivität „Load Customer Data“ einfügen. In Teil 1 des Experiments benötigen sie hierfür durchschnittlich 28,64 Sekunden (siehe Abb. 7.24).

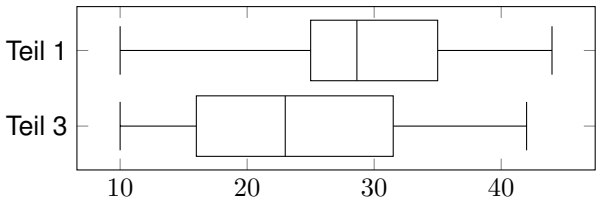

Abbildung 7.24: Bearbeitungszeiten (in Sek.) von Aufgabe F18 (Lese- und Schreibkante einfügen).

Es gibt eine Testperson, die für diese Aufgabe ein Kontextmenü verwendet. Zuvor tippt sie die Aktivität an, um die Lesekante im Kontextmenü auszuwählen. Anschließend tippt sie auf das Datenelement, um dieses mit der Aktivität zu verbinden.

Zwei Testpersonen zeichnen die Lesekante unter Verwendung eines Pfeil-Symbols ein. Die restlichen acht Testpersonen verwenden eine physikalische Geste. Sie führen eine Strich-Geste aus, die beim Datenelement startet und bei der Aktivität endet.

Der Vergleich mit [23] in Tabelle 7.20 zeigt, dass die Mehrheiten bei beiden Experimenten übereinstimmen.

	Ermittelte Werte	[23]
Symbolische Gesten	18,18%	30,77%
Physikalische Gesten	72,73%	57,69%
Menü-basierte Lösungen	9,09%	11,54%

Tabelle 7.20: Gesten-Verteilung beim Einfügen einer Lesekante.

Für dieselbe Aufgabe in Teil 3 benötigen die Testpersonen durchschnittlich 23,00 Sekunden. Alle Testpersonen können sich ohne Weiteres an die implementierte Geste erinnern, da die Mehrheit sich ebenfalls in Teil 1 für die Geste entschieden hat. Die Strich-Geste hat sich somit für das Erstellen von Lese- sowie Schreibkanten etabliert.

7.3 Abschließende Bemerkungen

Die Hälfte aller Aufgaben werden im dritten Teil des Experiments von allen Testpersonen problemlos gelöst. Im Durchschnitt können sich die Testpersonen an 3,64 Gesten nicht mehr erinnern, finden die Wahl einiger Gesten dennoch als sinnvoll (siehe Abb. 7.25). Eine Änderung dieser Gesten ist deshalb nicht notwendig. Dies sieht bei den Gesten für die Modellierungsfunktionen F6 (Element reduzieren), F13 (Subprozess auflösen), F15 (Datenelement einfügen) und F17 (Rückgängig machen) allerdings anders aus. Aufgrund ihrer Komplexität und langen Durchführungsdauer sollten sie in zukünftigen Verbesserungsarbeiten ersetzt werden, sofern Kollisionen mit anderen Gesten ausgeschlossen werden können. Alternative Gesten-Implementierungen sind in den Abschnitten 7.2.1.6, 7.2.1.13, 7.2.1.15 und 7.2.1.17 beschrieben.

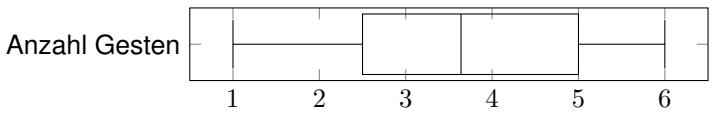

Abbildung 7.25: Anzahl Gesten, an die sich die Testpersonen in Teil 3 nicht erinnern können.

In zwei von acht Vergleichen mit [23] stimmen die Mehrheiten der beiden Experimente nicht überein. Die Unterschiede sind hier allerdings sehr gering und würden sich womöglich durch eine Erhöhung der Teilnehmerzahl angleichen. In einigen Fällen ist es auch nicht immer eindeutig, die ausgeführten Gesten einem Bedienkonzept zuzuordnen, weshalb dies ebenfalls zu einigen Abweichungen führt.

Wird zwischen Testpersonen mit und ohne Erfahrung mit Prozess-Management unterschieden, ergibt sich folgende Verteilung (siehe Abb. 7.26 und Abb. 7.28): An dem Experiment nehmen zwei Testpersonen ohne und neun Testpersonen mit Erfahrung mit Prozess-Management teil. Die Testpersonen ohne Erfahrung brauchen pro Aufgabe im ersten Teil des Experiments durchschnittlich 27,58 Sekunden. Für die Bearbeitung aller Aufgaben des ersten Teils benötigen sie im Durchschnitt 8,28

Minuten. Im Gegensatz dazu benötigen die Testpersonen mit Erfahrung mehr Zeit, obwohl hier ein besserer Wert zu erwarten wäre. Pro Aufgabe benötigen die Testpersonen mit Erfahrung durchschnittlich 35,27 Sekunden und für die Lösung aller Aufgaben im ersten Teil insgesamt 10,58 Minuten.

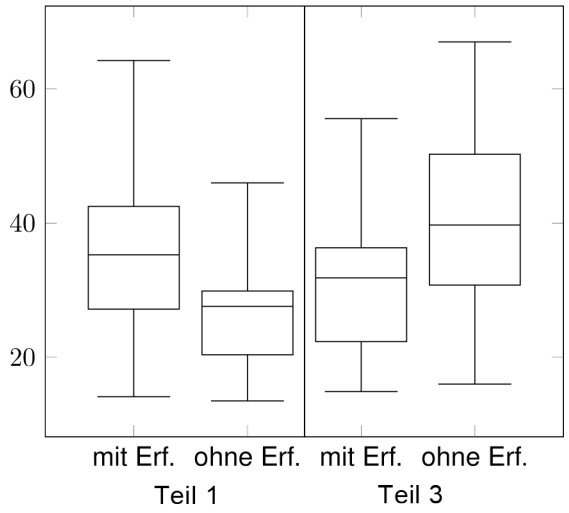

Abbildung 7.26: Durchschnittliche Bearbeitungszeiten (in Sek.) pro Aufgabe von Testpersonen mit und ohne Erfahrung mit Prozess-Management.

In Teil 3 des Experiments ist das Ergebnis umgekehrt. Bei diesem Teil benötigen die Testpersonen mit Erfahrung weniger Zeit. Sie lösen eine Aufgabe in durchschnittlich 31,83 Sekunden und brauchen für alle Aufgaben im dritten Teil 9,55 Minuten im Durchschnitt. Dem gegenüber benötigen Testpersonen ohne Erfahrung durchschnittlich 39,72 Sekunden pro Aufgabe und für das Lösen aller Aufgaben im Durchschnitt 11,92 Minuten. In diesem Teil sind es 4,5 Aufgaben, die von den Testpersonen ohne Erfahrung nicht gelöst werden. Testpersonen mit Erfahrung können sich nur an 3,44 Aufgaben nicht erinnern.

Tabelle 7.21 stellt die Verteilung der Gesten dar, die von Testpersonen mit und ohne Erfahrung bezüglich Prozess-Management bevorzugt werden. Die Mehrheit der Testpersonen auf beiden Seiten entscheidet sich hierbei hauptsächlich für physikalische Gesten. Der Unterschied ist, dass sich Testpersonen ohne Erfahrung öfters für Menü-basierte Lösungen entscheiden als Testpersonen mit Erfahrung bezüglich Prozess-Management.

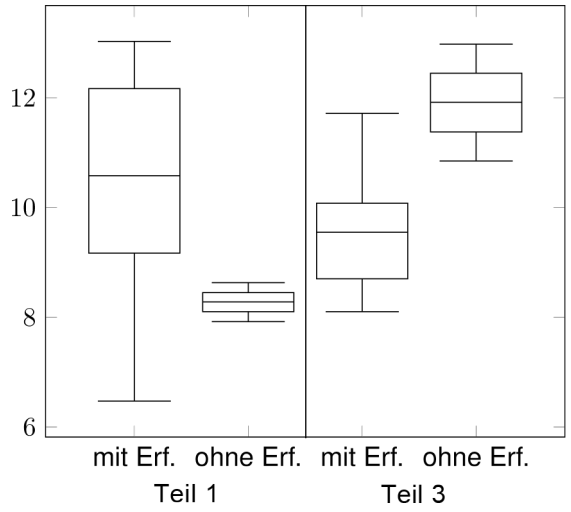

Abbildung 7.27: Gesamtbearbeitungszeiten (in Min.) von Testpersonen mit und ohne Erfahrung mit Prozess-Management.

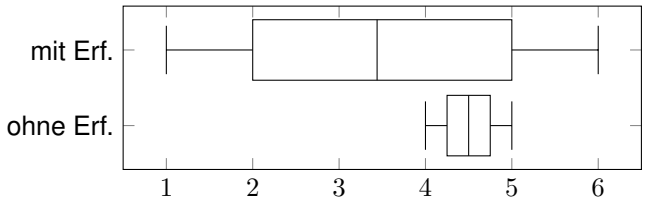

Abbildung 7.28: Anzahl Gesten, an die sich Testpersonen mit und ohne Erfahrung mit Prozess-Management nicht erinnern können.

	Erfahrung	Ohne Erfahrung
Symbolische Gesten	30,86%	11,11%
Physikalische Gesten	50,00%	52,78%
Menü-basierte Lösungen	19,14%	36,11%

Tabelle 7.21: Bevorzugte Gesten von Testpersonen mit und ohne Erfahrung bezüglich Prozess-Management.

8 Zusammenfassung und Ausblick

Das proView-Framework unterstützt die Erstellung und Modifikation von Prozesssichten. Hierfür wird eine Vielzahl an Modellierungsfunktionen angeboten, die in einer webbasierten Umgebung ausgeführt werden. Diese Arbeit beinhaltet die Implementierung einer Erweiterung von proView, mit der die zur Verfügung stehenden Modellierungsfunktionen per Multi-Touch-Gesten ausgelöst werden können. Dies soll den Einsatz von proView auf Multi-Touch-Geräten, wie Smartphones oder Tablets, verbessern. Nach einer ausführlichen Beschreibung des proView-Projekts und unterschiedlicher Bedienkonzepte wird die Problemstellung dieser Arbeit analysiert. Es werden verschiedene Experimente, die sich mit dem Thema Multi-Touch beschäftigen, durchleuchtet, um passende Multi-Touch-Gesten für die Modellierungsfunktionen zu finden. Da die Visualisierungskomponente von proView mit dem Vaadin-Framework entwickelt worden ist und für dieses Framework keine geeigneten Multi-Touch-Add-ons existieren, ist die Implementierung eines eigenen Add-ons notwendig, das alle für diese Arbeit relevanten Multi-Touch-Gesten unterstützt. Dieses TouchPanel-Add-on wird in proView integriert und mit den bestehenden Modellierungsfunktionen verknüpft.

Der abschließende Teil der Arbeit beinhaltet eine experimentelle Untersuchung der Implementierung. In einem aus drei Teilen bestehenden Experiment hatten die Testpersonen die Aufgabe, sich eigene Gesten einfallen zu lassen, sowie die implementierten Gesten nach einem Tutorial anzuwenden bzw. zu testen. Ein Großteil der für die proView-Funktionen gewählten Multi-Touch-Gesten ist von den Testpersonen akzeptiert worden.

In zukünftigen Verbesserungsarbeiten können die von den Testpersonen des Experiments als negativ bewerteten Gesten durch bessere Lösungen ersetzt werden. Zudem ist der Einsatz von Responsive Webdesign für die webbasierte Visualisierungskomponente des proView-Frameworks sinnvoll, da einige Bedienelemente der Benutzeroberfläche für mobile Geräte mit kleinen Bildschirmen, die per Fingereingaben bedient werden, ungeeignet sind. Je nach Größe des Geräts, Auflösung, Orientierung sowie Eingabemöglichkeiten, sollten die Bedienelemente automatisch ihre Größe anpassen.

A Anhang

A.1 Benutzerdefiniertes Gesten-Set

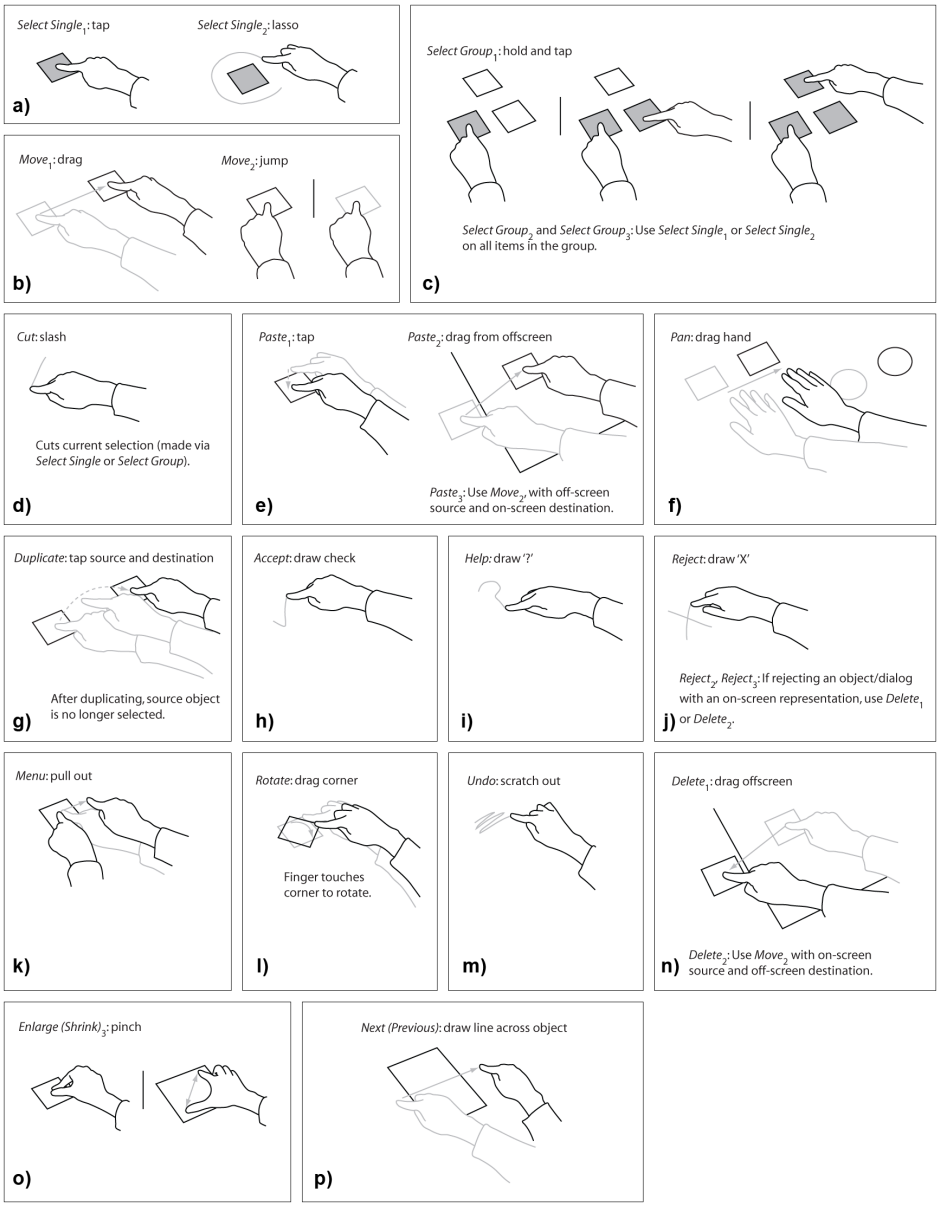

Abbildung A.1: Benutzerdefiniertes Gesten-Set.

A.2 Prozessmodell

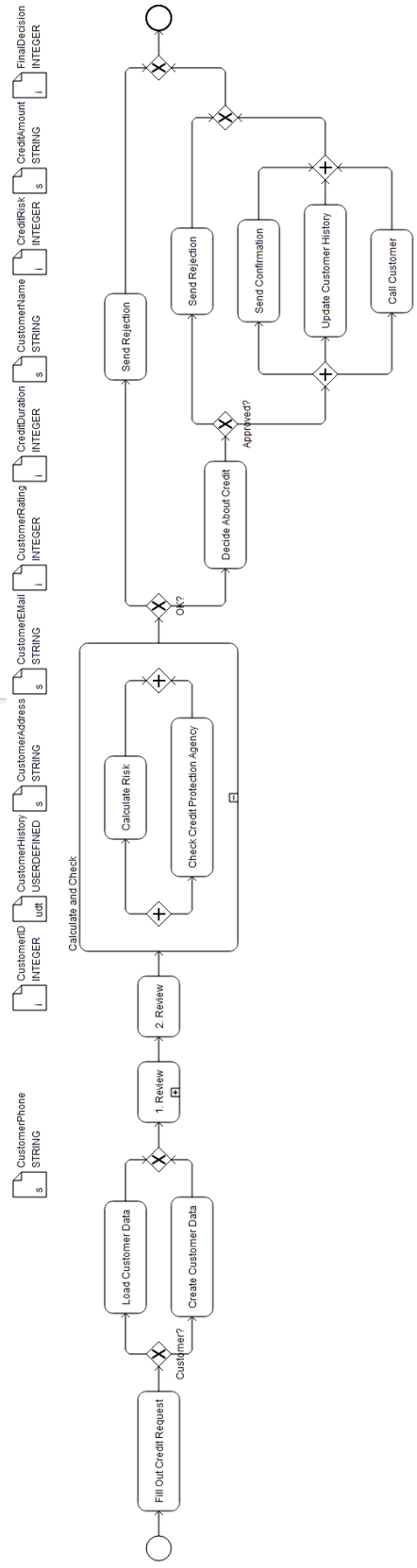

Abbildung A.2: Das im Experiment verwendete Prozessmodell.

A.3 Ergebnisse der experimentellen Untersuchung

Legende

F1: Element auswählen
F2: Aktivität einfügen
F3: Element umbenennen
F4: Element löschen
F5: Elemente aggregieren
F6: Element reduzieren
F7: Verzweigungsblock einfügen
F8: Verzweigung einfügen
F9: Schleife einfügen
F10: Synchronisationskante einfügen
F11: Subprozess anzeigen
F12: Subprozess verbergen
F13: Subprozess auflösen
F14: Prozesssicht erstellen
F15: Datenelement einfügen
F16: Hilfe aufrufen
F17: Rückgängig machen
F18: Lese- und Schreibkante einfügen

S: Symbolische Geste
G: Physikalische Geste
M: Menü-basierte Lösung

TNr	Testperson-Nummer
FNr	Funktions-Nummer
Geschl.:	Geschlecht
m:	männlich
w:	weiblich
L/R:	Links- / Rechtshänder
Tätigk.:	Tätigkeit
ST:	Student
MA:	Mitarbeiter des Instituts für Datenbanken und Informationssysteme der Universität Ulm
SO:	Sonstige
Erfahr. MT:	Erfahrung im Umgang mit Multi-Touch-Geräten
Erfahr. PM:	Erfahrung mit Prozess-Management

TNr	Geschl.	Alter	L/R	Tätigk.	Erfahr. MT	Erfahr. PM	PM Experte
1	m	24	R	ST	ja	ja	ja
2	m	27	R	ST	ja	ja	ja
3	m	25	R	ST	ja	ja	ja
4	m	32	R	SO	ja	nein	nein
5	m	29	R	MA	ja	ja	nein
6	m	26	R	ST	ja	ja	ja
7	m	27	L	ST	ja	ja	nein
8	m	25	R	ST	ja	ja	nein
9	m	25	R	ST	ja	nein	nein
10	w	24	R	ST	ja	ja	ja
11	m	25	R	ST	ja	ja	nein

Tabelle A.1: Demografische Daten der Testpersonen des Experiments.

TNr	Smartphone	Tablet	Android	iOS	Windows
1	ja	nein	ja	nein	nein
2	ja	ja	ja	nein	ja
3	nein	ja	nein	nein	ja
4	ja	ja	nein	ja	nein
5	ja	nein	ja	nein	nein
6	ja	ja	ja	ja	nein
7	ja	nein	ja	nein	nein
8	ja	nein	ja	ja	nein
9	ja	ja	ja	ja	nein
10	nein	ja	nein	ja	nein
11	ja	nein	ja	nein	nein

Tabelle A.2: Von den Testpersonen verwendete Multi-Touch-Geräte und Betriebssysteme.

TNr	Erfahr. PM (in Jahren)	analysiert / gelesen (Letzten 12 Monate)	erstellt / bearbeitet (Letzten 12 Monate)	øAnzahl Aktivitäten
1	2	40	20	7
2	3	50	50	25
3	5	10	5	10
5	5	0	0	0
6	3	50	50	20
7	3	0	0	0
8	2	0	0	0
10	3	15	15	10
11	2	0	0	0
ø	3,11	18,33	15,56	8

Tabelle A.3: Erfahrungen der Testpersonen mit Prozess-Management.

FNr	1	2	3	4	5	6	7	8	9	10	11
F1:	G	G	G	G	G	G	G	G	G	G	G
F2:	M	S	G	M	G	M	G	G	G	S	M
F3:	G	G	G	M	G	M	G	G	G	G	G
F4:	M	G	S	G	S	S	G	G	G	S	G
F5:	S	S	G	S	G	G	S	S	M	S	G
F6:	G	G	G	M	G	G	M	G	G	G	M
F7:	M	M	S	M	S	S	S	G	G	M	M
F8:	M	G	G	M	M	G	M	M	G	S	M
F9:	M	S	S	S	S	S	S	S	S	S	M
F10:	M	S	S	G	G	G	G	G	G	S	G
F11:	G	G	G	G	S	G	G	G	G	G	G
F12:	G	G	G	G	G	G	G	G	G	G	G
F13:	G	S	G	M	G	G	S	G	M	S	M
F14:	S	M	S	M	S	S	S	S	M	S	G
F15:	M	M	S	M	S	G	M	M	M	M	G
F16:	M	S	S	G	S	M	S	G	S	M	G
F17:	M	S	S	M	S	G	S	G	G	S	G
F18:	M	S	G	G	S	G	G	G	G	G	G

Tabelle A.4: Von den Testpersonen gewählte Bedienkonzepte.

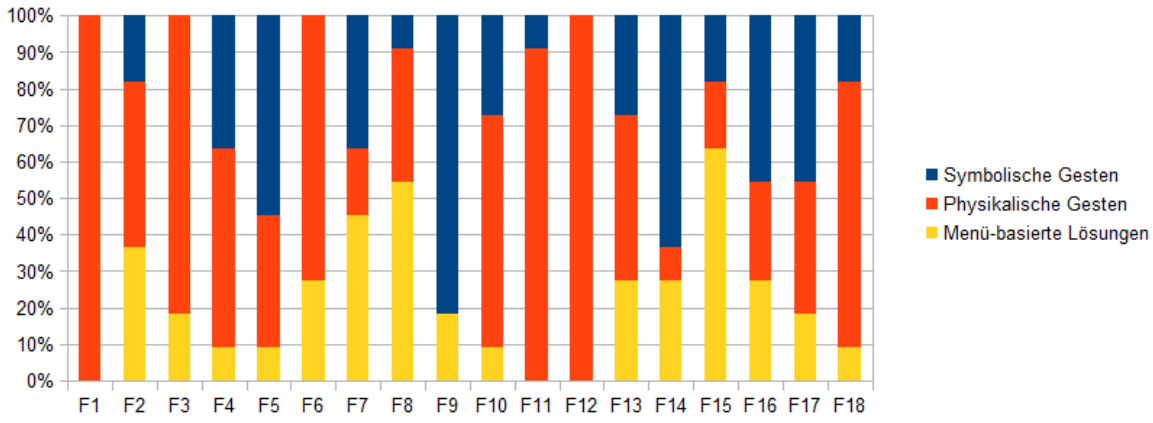

Abbildung A.3: Gewählte Bedienkonzepte im Überblick.

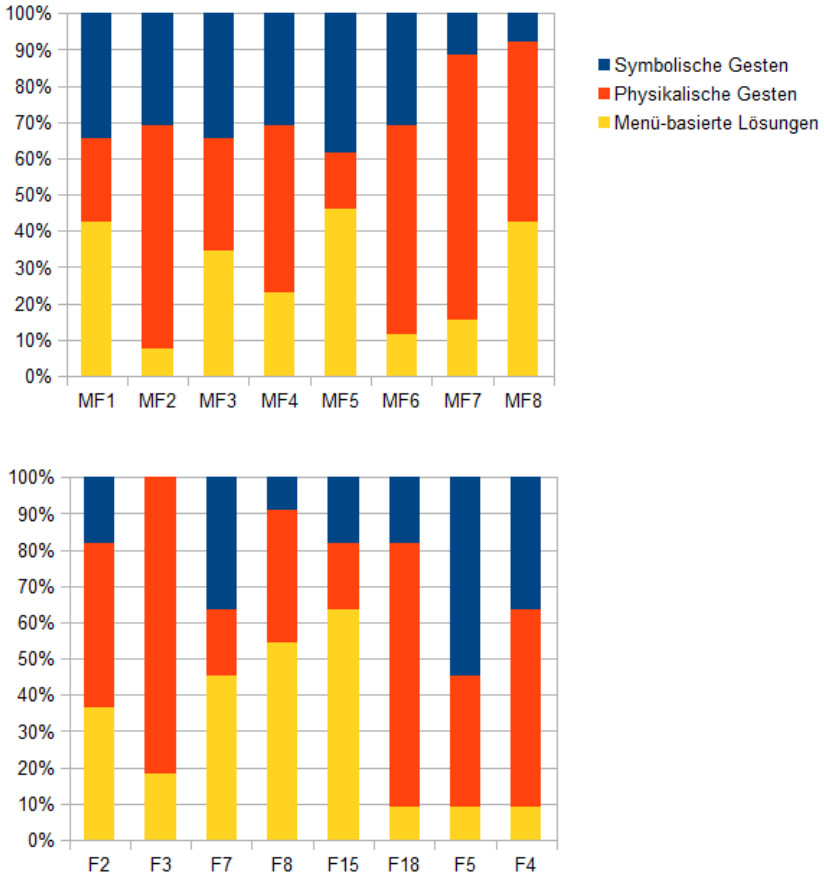

Abbildung A.4: Verwendete Bedienkonzepte in [23] (oben) und des Experiments dieser Arbeit (unten).

FNr	Aufgabe	Häufigkeit (gelöst)
F2	Aktivität einfügen	11
F5	Elemente aggregieren	11
F9	Schleife einfügen	11
F10	Synchronisationskante einfügen	11
F12	Subprozess verbergen	11
F14	Prozesssicht erstellen	11
F16	Hilfe aufrufen	11
F18	Lese- und Schreibkante einfügen	11
F1	Element auswählen	10
F8	Verzweigung einfügen	9
F11	Subprozess anzeigen	9
F4	Element löschen	8
F17	Rückgängig machen	8
F3	Element umbenennen	7
F13	Subprozess auflösen	7
F7	Verzweigungsblock einfügen	6
F15	Datenelement einfügen	5
F6	Element reduzieren	1

Tabelle A.5: Anzahl gelöster Aufgaben des dritten Teils des Experiments in absteigender Reihenfolge.

FNr	Aufgabe	øBearbeitungsdauer (in Sek.)
F11	Subprozess anzeigen	14,36
F1	Element auswählen	16,64
F12	Subprozess verbergen	20,09
F17	Rückgängig machen	22,09
F3	Element umbenennen	25,45
F15	Datenelement einfügen	28,09
F18	Lese- und Schreibkante einfügen	28,64
F4	Element löschen	28,73
F2	Aktivität einfügen	29,64
F8	Verzweigung einfügen	32,64
F13	Subprozess auflösen	37,36
F10	Synchronisationskante einfügen	37,45
F5	Elemente aggregieren	39,00
F6	Element reduzieren	40,91
F9	Schleife einfügen	42,36
F16	Hilfe aufrufen	48,64
F14	Prozesssicht erstellen	56,73
F7	Verzweigungsblock einfügen	60,91
ø		33,87

Tabelle A.6: Durchschnittliche Bearbeitungszeit der einzelnen Aufgaben im ersten Teil des Experiments in aufsteigender Reihenfolge.

FNr	Aufgabe	øBearbeitungsdauer (in Sek.)
F12	Subprozess verbergen	15,09
F16	Hilfe aufrufen	17,45
F1	Element auswählen	18,27
F10	Synchronisationskante einfügen	23,00
F18	Lese- und Schreibkante einfügen	23,00
F14	Prozesssicht erstellen	28,73
F3	Element umbenennen	30,82
F8	Verzweigung einfügen	31,36
F17	Rückgängig machen	31,36
F15	Datenelement einfügen	31,55
F11	Subprozess anzeigen	32,91
F4	Element löschen	33,27
F2	Aktivität einfügen	35,64
F9	Schleife einfügen	37,91
F5	Elemente aggregieren	46,73
F6	Element reduzieren	51,73
F13	Subprozess auflösen	52,27
F7	Verzweigungsblock einfügen	57,64
ø		33,26

Tabelle A.7: Durchschnittliche Bearbeitungszeit der einzelnen Aufgaben im dritten Teil des Experiments in aufsteigender Reihenfolge.

A.4 $1 Gesture Recognizer

Algorithm 2 Schritt 1. Resample den $points$ Pfad, sodass n Punkte mit gleichmäßigen Abstand zueinander bestehen bleiben.

```
1:  function RESAMPLE(points, n)
2:      I ← PATH-LENGTH(points)/(n–1)
3:      D ← 0
4:      newPoints ← points_0
5:      for all point p_i for i ≥ 1 in points do
6:          d ← DISTANCE(p_{i−1}, p_i)
7:          if (D + d) ≥ I then
8:              q_x ← p_{i−1_x} + ((I–D)/d) ∗ (p_{i_x}–p_{i−1_x})
9:              q_y ← p_{i−1_y} + ((I–D)/d) ∗ (p_{i_y}–p_{i−1_y})
10:             APPEND(newPoints, q)
11:             INSERT(points, i, q)                    ▷ q ist das nächste p_i
12:             D ← 0
13:         else
14:             D ← D + d
15:         end if
16:     end for
17:     return newPoints
18: end function

19: function PATH-LENGTH(A)
20:     d ← 0
21:     for i from 1 to |A| step 1 do
22:         d ← d+ DISTANCE(A_{i−1}, A_i)
23:     end for
24:     return d
25: end function
```

Algorithm 3 Schritt 2. Rotiere $points$, sodass ihr indikativer Winkel bei $0°$ liegt.

function ROTATE-TO-ZERO($points$)
  $c \leftarrow$ CENTROID($points$) ▷ berechnet $(\overline{x}, \overline{y})$
  $\theta \leftarrow$ ATAN($c_y – points_{0_y}, c_x – points_{0_x}$) ▷ für $-\pi \leq \theta \leq \pi$
  $newPoints \leftarrow$ ROTATE-BY($points, -\theta$)
  return $newPoints$
end function

function ROTATE-BY($points, \theta$)
  $c \leftarrow$ CENTROID($points$)
  for all point p in $points$ **do**
    $q_x \leftarrow (p_x – c_x)$ COS(θ) $–(p_y – c_y)$ SIN(θ) $+c_x$
    $q_y \leftarrow (p_x – c_x)$ SIN(θ) $–(p_y – c_y)$ COS(θ) $+c_y$
    APPEND($newPoints, q$)
  end for
  return $newPoints$
end function

Algorithm 4 Schritt 3. Skaliere $points$, sodass die Bounding Box eine Größe von $size^2$ besitzt. Verschiebe $points$ anschließend zum Ursprung. BOUNDING-BOX liefert ein Rechteck mit den Werten (min_x, min_y), (max_x, max_y) zurück. Für Gesten, die als Referenzmuster abgelegt werden sollen, sollten die Schritte 1-3 auf die eingegebenen Punkte angewandt werden. Für ausgeführte Gesten, sollten die Schritte 1-4 verwendet werden.

function SCALE-TO-SQUARE($points, size$)
  $B \leftarrow$ BOUNDING-BOX($points$)
  for all point p in $points$ **do**
    $q_x \leftarrow p_x * (size / B_{width})$
    $q_y \leftarrow p_y * (size / B_{height})$
    APPEND($newPoints, q$)
  end for
  return $newPoints$
end function

function TRANSLATE-TO-ORIGIN($points$)
  $c \leftarrow$ CENTROID($points$)
  for all point p in $points$ **do**
    $q_x \leftarrow p_x – c_x$
    $q_y \leftarrow p_y – c_y$
    APPEND($newPoints, q$)
  end for
  return $newPoints$
end function

Algorithm 5 Schritt 4. Vergleiche $points$ mit den abgelegten Referenzmustern. Die $size$ Variable von RECOGNIZE bezieht sich auf die Variable $size$ von SCALE-TOSQUARE in Schritt 3. Das Symbol φ gleicht dem Ausdruck $\frac{1}{2}(-1+\sqrt{5})$. Es wird $\theta = \pm 45°$ und $\theta_\Delta = 2°$ von RECOGNIZE verwendet. Aufgrund der Funktion RESAMPLE, kann angenommen werden, dass A und B dieselbe Anzahl an Punkten besitzen, d.h. $|A| = |B|$.

1: **function** RECOGNIZE($points, templates$)
2: $b \leftarrow +\infty$
3: **for all** template T in $templates$ **do**
4: $d \leftarrow$ DISTANCE-AT-BEST-ANGLE($points, T, -\theta, \theta, \theta_\Delta$)
5: **if** $d < b$ **then**
6: $b \leftarrow d$
7: $T' \leftarrow T$
8: **end if**
9: **end for**
10: $score \leftarrow 1–b/0.5\sqrt{(size^2 + size^2)}$
11: **return** $\langle T', score \rangle$
12: **end function**

13: **function** DISTANCE-AT-BEST-ANGLE($points, T, \theta_a, \theta_b, \theta_\Delta$)
14: $x_1 \leftarrow \varphi\theta_a + (1–\varphi)\theta_b$
15: $f_1 \leftarrow$ DISTANCE-AT-ANGLE($points, T, x_1$)
16: $x_2 \leftarrow (1–\varphi)\theta_a + \varphi\theta_b$
17: $f_2 \leftarrow$ DISTANCE-AT-ANGLE($points, T, x_2$)
18: **while** $|\theta_b–\theta_a| > \theta_\Delta$ **do**
19: **if** $f_1 < f_2$ **then**
20: $\theta_b \leftarrow x_2$
21: $x_2 \leftarrow x_1$
22: $f_2 \leftarrow f_1$
23: $x_1 \leftarrow \varphi\theta_a + (1–\varphi)\theta_b$
24: $f_1 \leftarrow$ DISTANCE-AT-ANGLE($points, T, x_1$)
25: **else**
26: $\theta_a \leftarrow x_1$
27: $x_1 \leftarrow x_2$
28: $f_1 \leftarrow f_2$
29: $x_2 \leftarrow (1–\varphi)\theta_a + \varphi\theta_b$
30: $f_2 \leftarrow$ DISTANCE-AT-ANGLE($points, T, x_2$)
31: **end if**
32: **end while**
33: **return** MIN(f_1, f_2)
34: **end function**

35: **function** DISTANCE-AT-ANGLE($points, T, \theta$)
36: $newPoints \leftarrow$ ROTATE-BY($points, \theta$)
37: $d \leftarrow$ PATH-DISTANCE($newPoints, T_{points}$)
38: **return** d
39: **end function**

40: **function** PATH-DISTANCE(A, B)
41: $d \leftarrow 0$
42: **for** i **from** 0 **to** $|A|$ **step** 1 **do**
43: $d \leftarrow d+$DISTANCE(A_i, B_i)
44: **end for**
45: **return** $d/|A|$
46: **end function**

Literaturverzeichnis

[1] BAUMGARTEN, Bernd (1996): *Petri-Netze - Grundlagen und Anwendungen*. Spektrum-Akademischer Verlag.

[2] BÜRINGER, Stefan (2012): *Development of a Business Process Abstraction Component based on Process Views*. Bachelorarbeit, Fakultät für Ingenieurwissenschaften und Informatik, Universität Ulm.

[3] DADAM, Peter; REICHERT, Manfred (2009): *The ADEPT Project: A Decade of Research and Development for Robust and Flexible Process Support*. Technical Report, Fakultät für Ingenieurwissenschaften und Informatik, Universität Ulm.

[4] FISHER, Bill (2013): *Touchy*. `https://github.com/HotStudio/touchy` (besucht am 6. Aug. 2013).

[5] GRANKVIST, Michael (2011): *TouchMenu*. `http://vaadin.com/addon/touchmenu` (besucht am 3. Aug. 2013).

[6] GROSSKOPF, Alexander; DECKER, Gero; WESKE, Mathias (2009): *The Process: Business Process Modeling Using BPMN*. Meghan-Kiffer Press.

[7] GRÖNROOS, Marko (2013): *Book of Vaadin: Vaadin 7*. 2. Auflage. Vaadin Ltd.

[8] JGESTURES (n.d.): *jGestures: A jQuery Plugin for Gesture Events*. `http://jgestures.codeplex.com/` (besucht am 6. Aug. 2013).

[9] HONG, Jason I.; LANDAY, James A. (2010): *SATIN: A Toolkit for Informal Ink-based Applications*. In: Proc. UIST '00. New York: ACM Press, 63-72.

[10] JQUERY FOUNDATION (2013): *jQuery*. `http://jquery.com/` (besucht am 6. Aug. 2013).

[11] KOLB, Jens; RUDNER, Benjamin; REICHERT, Manfred (2012): *Towards Gesture-based Process Modeling on Multi-Touch Devices*. In: 1st Int'l Workshop on Human-Centric Process-Aware Information Systems (HC-PAIS'12), Gdansk, Poland.

[12] KOLB, Jens; REICHERT, Manfred (2013): *Supporting Business and IT through Updatable Process Views: The proView Demonstrator*. In: ICSOC'12, Demo

Track of the 10th Int'l Conference on Service Oriented Computing, November 12-15, 2012, Shanghai, China.

[13] KOLB, Jens; REICHERT, Manfred (2013): *Data Flow Abstractions and Adaptations through Updatable Process Views*. In: 28th Symposium on Applied Computing (SAC'13), 10th Enterprise Engineering Track (EE'13), Coimbra, Portugal, March 2013, ACM Press, pp. 1447-1453.

[14] KOLB, Jens; REICHERT, Manfred (2013): *A Flexible Approach for Abstracting and Personalizing Large Business Process Models*. In: Applied Computing Review, 13(1): 6-17, ACM SIGAPP.

[15] KOLB, Jens; LEOPOLD, Henrik; MENDLING, Jan; REICHERT, Manfred (2013): *Creating and Updating Personalized and Verbalized Business Process Descriptions*. In: 6th IFIP WG 8.1 Working Conference on the Practice of Enterprise Modeling (PoEM'13), Riga, Latvia, November 6-7, 2013, LNBIB, Springer.

[16] LANDAY, James A.; MYERS, Brad A. (1993): *Extending an Existing User Interface Toolkit to Support Gesture Recognition*. In: Proc. CHI '93. New York: ACM Press, 91-92.

[17] LIN, James; NEWMAN, Mark W.; HONG, Jason I.; LANDAY, James A. (2000): *DENIM: Finding a Tighter Fit Between Tools and Practice for Website Design*. In: Proc. CHI '00. New York: ACM Press, 510-517.

[18] MYERS, Cory S.; RABINER, Lawrence R. (1981): *A Comparative Study of Several Dynamic Time-Warping Algorithms for Connected Word Recognition*. In: The Bell System Technical J. 60 (7), 1389-1409.

[19] OMG (2011): *Business Process Model and Notation (BPMN)*. `http://www.omg.org/spec/BPMN/2.0/PDF` (besucht am 2. Jan. 2014).

[20] PETERS, Andre (2013): *JSlider*. `http://vaadin.com/addon/jslider-add-on` (besucht am 3. Aug. 2013).

[21] ROBINSON, James; MCCORMACK, Cameron (2013): *Timing control for script-based animations: W3C Candidate Recommendation 31 October 2013*. `http://www.w3.org/TR/animation-timing/` (besucht am 12. Sep. 2013).

[22] RUBINE, Dean (1991): *Specifying Gestures by Example*. In: Proc. SIGGRAPH '91. New York: ACM Press, 329-337.

[23] RUDNER, Benjamin (2011): *Fortgeschrittene Konzepte der Prozessmodellierung durch den Einsatz von Multi-Touch-Gesten*. Bachelorarbeit, Fakultät für Ingenieurwissenschaften und Informatik, Universität Ulm.

[24] SCHEPERS, Doug; SANGWHAN, Moon; BRUBECK, Matt; et al. (2013): *Touch Events version 1: W3C Proposed Recommendation 09 May 2013*. `http://www.w3.org/TR/2013/PR-touch-events-20130509/` (besucht am 12. Sep. 2013).

[25] SWIGART, Scott (2005): *Easily Write Custom Gesture Recognizers for Your Tablet PC Applications*. `http://msdn.microsoft.com/en-us/library/aa480673.aspx` (besucht am 21. Sep. 2013).

[26] TAHVONEN, Matti (2011): *TouchScroll*. `http://vaadin.com/addon/touchscroll` (besucht am 3. Aug. 2013).

[27] TANGELDER, Jorik (n.d.): *Hammer.js - A JavaScript Library for Multi-Touch Gestures*. `http://eightmedia.github.io/hammer.js/` (besucht am 6. Aug. 2013).

[28] VAN SOMEREN, Maarten W.; BARNARD, Yvonne F.; SANDBERG, Jacobijn A.C. (1994): *The Think Aloud Method: A Practical Guide to Modelling Cognitive Processes*. Academic Press, London.

[29] VILLAR Javi J. (n.d.): *QuoJS - Micro JavaScript Library*. `http://quojs.tapquo.com` (besucht am 3. Aug. 2013).

[30] WOBBROCK, Jacob O.; WILSON, Andrew D.; LI, Yang (2007): *Gestures without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes*. In: Proc. of the 20th Annual ACM Symposium on User Interface Software and Technology, New York, USA, S. 159-168.

[31] WOBBROCK, Jacob O.; MORRIS, Meredith R.; WILSON, Andrew D. (2009): *User-defined Gestures for Surface Computing*. In: Proc. 27th Int'l Conf on Human Factors in Computing Systems (CHI '09), New York, USA, S. 1083-1092.

[32] WOHLIN, Claes; RUNESON, Per; HST, Martin; OHLSSON, Magnus C. (2012): *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated.

Name: Adam Just Matrikelnummer: 673424

Erklärung

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Adam Just