UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Fakultät für Elektrotechnik, Informatik und Mathematik
Heinz Nixdorf Institut und Institut für Informatik
Fachgebiet Mensch-Maschine-Wechselwirkung
Warburger Straße 100
33098 Paderborn

Konzept und prototypische Umsetzung eines kollaborativen UML-Klassendiagramm-Editors für Multi-Touch-Tische

Bachelor-Arbeit

im Rahmen des Studiengangs Informatik
zur Erlangung des Grades
Bachelor of Science

von
Andreas Gehle
Ludwigstraße 81
33098 Paderborn

und
Alexei Quapp
Annette von Droste-Hülshoffstraße 31
32839 Steinheim

vorgelegt bei
Prof. Dr. Gerd Szwillus
und
Prof. Dr. Marco Platzner

Paderborn, 13. August 2013

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen worden ist. Alle Ausführungen, die wörtlich oder sinngemäß übernommen worden sind, sind als solche gekennzeichnet.

Ort, Datum	Unterschrift
Ort, Datum	Unterschrift

Inhalt

Abstract

Die gemeinsame Erstellung und Bearbeitung von Klassendiagrammen durch mehrere Personen, wie Projektteams gestaltet sich je nach genutzten Medien und Werkzeugen als schwierig, kompliziert oder wenig intuitiv. Insbesondere ist die fehlende Unterstützung eines Entwicklungsprozesses von einer Problemstellung bis hin zu einem ausgereiften Klassendiagramm, als auch die Kollaboration innerhalb eines Teams in vielen aktuell angebotenen Werkzeugen deutlich zu erkennen. Mit Hilfe eines Multi-Touch-Tisches kann eine Projektgruppe effizient, gleichzeitig und kollaborativ arbeiten. Außerdem wird dabei die Kommunikation innerhalb des Teams stark gesteigert.

Zu den Kernpunkten dieser Arbeit gehört eine benutzerorientierte Analyse dieses Entwicklungsprozesses, welche Probleme und Schwierigkeiten während einzelner Phasen der kollaborativen Diagrammerstellung aufzeigt sowie Grundfunktionalitäten für die Entwicklung am Tisch fordert. Um diese Probleme und Schwierigkeiten zu beheben bzw. zu mindern und geforderte Funktionalitäten bereit zu stellen, werden weiterhin Konzepte entwickelt, die Nutzer in ihrer Entwicklung unterstützen sollen. Eine prototypische Implementierung setzt einige der wichtigsten Konzepte um, die anschließend in einer Evaluation durch Versuchsgruppen, bestehend aus drei bis vier erfahrenen Informatik-Studenten, eingesetzt und geprüft werden. Die Arbeit wird mit dem Auswerten der Versuchsergebnisse und einem Ausblick für aufbauende Studien abgeschlossen.

1 Einleitung

In der Einleitung wird zunächst auf die Aufgabenstellung dieser Arbeit und unsere Motivation zu dieser eingegangen, bevor die Ziele der Arbeit festgelegt werden. Anschließend werden ähnliche, bereits bestehende Projekte vorgestellt, die als Grundlage dieser Arbeit dienen. Weiterhin werden innovative Ein- und Ausgabesysteme betrachtet, zu denen auch Multi-Touch-Tische gehören, um in die Eingabemöglichkeiten eines MTTs einzuleiten.

1.1 Aufgabenstellung

In dieser Bachelorarbeit soll der Entstehungsprozess von UML-Klassendiagrammen hinsichtlich der Kollaboration mehrerer Benutzer an einem Multi-Touch-Tisch untersucht werden. Dazu sollen in diesem Prozess Schwachstellen identifiziert werden, die eine gemeinsame Entwicklung durch mehrere Benutzer nicht hinreichend unterstützen oder sogar erschweren. Mit Hilfe von Konzepten soll der Entwicklungsprozess verbessert und die aufgezeigten Schwierigkeiten gelöst bzw. abgeschwächt werden. Ein Editor soll prototypisch konzipiert und implementiert werden, der nicht nur das kollaborative Erstellen von Klassendiagrammen ermöglicht, sondern auch Lösungen für einige aufgezeigte Schwachstellen bietet. Abschließend soll mit Hilfe des implementierten Prototypen eine Evaluierung der Konzeption durchgeführt werden. Dabei soll die konzipierte Anwendung unter anderem auf die sinnvolle Unterstützung des Entstehungsprozesses eines Diagramms untersucht werden.

1.2 Motivation

In Software-Projekten werden sowohl in der Analysephase als auch in der Entwurfsphase UML-Diagramme verwendet. UML steht für *Unified Modeling Language* und ist eine graphische Modellierungsspache mit deren Hilfe Software spezifiziert, konstruiert und auch dokumentiert werden kann. Ein oft verwendeter Diagrammtyp für Software-Projekte verschiedenster Größen ist das Klassendiagramm. Dieses beschreibt den Aufbau und die Verbindungen der Klassen untereinander. Der wesentliche Vorteil eines Klassendiagramms liegt laut den Autoren von „UML 2 Glasklar“[1] darin, dass Strukturen, Daten und Verhalten einer Anwendung leicht

[1] Rupp, Chris/Queins, Stefan/Zengler, Barbara: UML 2 glasklar: Praxiswissen für die UML-Modellierung. 3. Auflage. München und Wien: Hanser, 2007, ISBN 978–3446411180.

in diesem Modell überblickt bzw. verstanden werden können. Weiterhin wird das Klassendiagramm von den Autoren als „Kern der gesamten Modellierungssprache“ bezeichnet.[2] Da heutzutage solche Diagramme eher von Teams als von Einzelpersonen erstellt werden, müssen oftmals gravierende Schwierigkeiten in der Kommunikation, der Zusammenarbeit und der Arbeitsteilung beseitigt werden. Einerseits muss jedes Teammitglied effizient seiner Arbeit nachkommen, andererseits müssen alle Teammitglieder als Team agieren und miteinander kommunizieren, um auf diese Weise ein Diagramm fertigzustellen. Der Entstehungsprozess eines Diagramms beginnt zudem, entgegen der Erwartungen, nicht mit der Erstellung der ersten Klasse, sondern mit der Besprechung des Gesamtproblems, das kontinuierlich in kleinere Teilprobleme samt Assoziationen aufgeteilt wird, bis diese schließlich in Klassen modelliert werden. Dafür werden bislang mindestens zwei Diagrammtypen benötigt: Einerseits ein Diagramm mit freier Notation, um das Gesamtproblem zu erörtern, und andererseits eins mit Klassendiagramm-Notation. Zwischen diesen Notationen wird bei Besprechungen, je nach Entwicklungsstand, fortlaufend gewechselt. Ferner werden während der Arbeitsteilung einzelne Elemente, wie Teilprobleme und Klassen, Teammitgliedern zugeordnet, was wiederkehrend zu Missverständnissen und Problemen führt.

Mit Medien, wie dem Whiteboard, der Tafel oder herkömmlichen Papier können Teammitglieder zusammenarbeiten und kommunizieren. Weiterhin sind sie nicht gezwungen Standards, gegebenen durch die UML, einzuhalten und können daher Diagrammtypen miteinander vermischen. Klassen, Teilprobleme und Assoziationen lassen sich jedoch nicht ohne großen Aufwand verschieben und umstrukturieren. Auch der Platz auf einem Papierblatt, einer Tafel oder einem Whiteboard ist begrenzt, um nur einige Probleme dieser Medien zu nennen. Zudem werden die erstellten Diagramme im Anschluss digitalisiert, womit weiterer Arbeitsaufwand entsteht.

Mit herkömmlichen digitalen Medien lassen sich diese beiden Aspekte, Effizienz des Einzelnen und Zusammenarbeit des Teams, bislang nicht richtig vereinen. Wird ein einzelner PC genutzt, so kann das Team, wie an der Tafel, gemeinsam diskutieren, gestikulieren und durch Aufzeigen den Gesprächsgegenstand betonen. Jedoch kann nur ein einzelnes Teammitglied den PC bedienen, wodurch die restlichen Teammitglieder handlungsunfähig bleiben. Hat jedes Teammitglied seinen eigenen PC, so können alle Beteiligten zur selben Zeit arbeiten, worunter die Kommunikation und die Zusammenarbeit allerdings enorm leiden. Es kann beispielsweise nicht mehr mit einem Finger auf ein Element gedeutet werden, über das gerade gesprochen wird. Genauso können eigene Ideen nicht mehr mit Gesten verdeutlicht werden. Weiterhin können keine Kleingruppen kurzfristig und flexibel, wie an einem Arbeitstisch, gebildet werden. Zum Beispiel wenn einzelne Teammitglieder unabhängig vom Rest der Gruppe einem Gedanken nachgehen wollen, ohne die Gruppe zu stören. Entweder wird quer durch einen Raum gerufen oder es werden Dienste wie *Voice over IP* (VoIP) genutzt. Dies führt bekanntermaßen

[2] Vgl. Rupp, Chris/Queins, Stefan/Zengler, Barbara, a. a. O., S. 101.

zu Problemen, wenn mehrere Personen sich gleichzeitig unterhalten wollen. Eine Lösung mit verschiedenen VoIP-Channels ist zwar denkbar, jedoch offensichtlich unpraktisch, solange sich die Kleingruppen permanent neu bilden. Bei Betrachtung der UML-Klassendiagramm-Erstellung unter Einsatz eines Desktop-PCs, wird eine breite Auswahl an Editoren angeboten. Diese unterliegen einerseits den bereits erläuterten Problemen und bieten andererseits keine der benötigten Möglichkeiten, weswegen beispielsweise eine Vermischung von Diagrammtypen nicht möglich ist. Ein passendes digitales Medium ist ein Multi-Touch-Tisch. An diesem können mehrere Personen gleichzeitig, wie jeweils bei Einzelarbeiten an einem PC, arbeiten und trotzdem miteinander kommunizieren, wie über einem Blatt Papier.

Bislang existieren drei Prototypen von Anwendungen zum Erstellen von UML-Klassendiagrammen auf einem Multi-Touch-Tisch. Diese bieten unzureichende Möglichkeiten und sind, den Entstehungsprozess von Klassendiagrammen betreffend, wie ihre Ebensacher auf den PCs stark begrenzt. DTDiagram[3] ist ein Prototyp eines UML-Editors, welcher keine Möglichkeiten zur Zusammenarbeit bietet. Der Prototyp Collabee[4] dagegen bietet gute Möglichkeiten um Diagramme mit mehreren Personen zu erstellen. Die Funktionalität des Collabee-Prototyps ist allerdings unzureichend für die Erstellung von nicht-trivialen Klassendiagrammen. COSMOS[5] ist ein Framework zur Erstellung grafischer Editoren. Mit diesem soll es möglich sein, Editoren für einen Multi-Touch-Tisch mit wenig Aufwand zu entwickeln. Als Prototyp wurde ein Klassendiagramm-Editor entwickelt, welcher jedoch für Review-Szenarien und nicht zur Entwickelung von Klassendiagrammen gedacht ist.

Aus diesen Gründen möchten wir in unserer Bachelorarbeit den Prototypen eines UML-Klassendiagramm-Editors für den MTT entwickeln, mit dem das kollaborative Arbeiten an Klassendiagrammen generell möglich ist und der den gesamten Entstehungsprozess eines Klassendiagramms unterstützt. Dazu sollen die beiden vorhin genannten Aspekte, Effizienz des Einzelnen und Zusammenarbeit des Teams wie auch die Vorteile der existierenden Prototypen vereint werden.

[3] Selber, Thomas/Pongelli, Stefano/Valente, Giulio: DTDiagram - a Pen and Touch UML Class Diagram editor for DiamondTouch tabletops: Semester project for the Information Systems Lab 2012 at ETHZ (www.ethz.ch). Online im Internet: `http://www.youtube.com/watch?v=hsZOsjf5un4` – Zugriff am 04.01.2013.

[4] Totolici, Alex et al.: Collabee – Multi-touch Collaborative Diagramming. Online im Internet: `http://www.jrejre.net/2010/08/collabee-multi-touch-collaborative-diagramming` – Zugriff am 04.01.2013.

[5] Löffelholz, Daniel et al.: COSMOS: Multi-touch support for collaboration in user-centered projects. In Heiss, Hans-Ulrich (Hrsg.): Informatik 2011. Bonn: Ges. für Informatik. Online im Internet: `http://www.user.tu-berlin.de/komm/CD/paper/061521.pdf` – Zugriff am 04.01.2013, ISBN 978–3–88579–286–4.

1.3 Zielsetzung

Die Ziele der Arbeit lassen sich in vier wesentliche Punkte unterteilen. Zunächst soll der Entwicklungsprozess eines UML-Klassendiagramms von der Problemstellung bis hin zu einem ausgereiften Diagramm an einem MTT analysiert werden. Dabei sollen aus *Benutzersicht* Anforderungen und Probleme erkannt werden, die während des Entwicklungsprozesses auftreten sowie Schwierigkeiten, die die Kollaboration innerhalb eines Teams während der Bearbeitung eines Diagramms am Tisch erschweren. Für diese Schwierigkeiten und analysierten Anforderungen sollen anschließend Lösungskonzepte sowie effektive Umsetzungsmöglichkeiten für den Touch-Tisch entwickelt und beschrieben werden.

Des Weiteren soll ein Prototyp eines Klassendiagramm-Editors für einen Multi-Touch-Tisch entworfen werden, der eine Teilmenge der zuvor beschriebenen Konzepte unterstützt. Die prototypische Implementierung soll die Bearbeitung von Klassendiagrammen ermöglichen und die Kollaboration während des gesamten Entwicklungsprozesses unterstützen als auch vereinfachen. Dabei soll der Prototyp grundlegende Fähigkeiten zum Planen und Entwickeln begünstigen, die sich aus folgenden Funktionen zusammensetzen: Teammitglieder sollen sich am Tisch anmelden bzw. ein Nutzerprofil anlegen können. Dieses Profil soll genutzt werden, um Aufgaben und Verantwortlichkeiten den Benutzern zuzuweisen. Das Erstellen und Hinzufügen von Benutzerprofilen zum Softwareprojekt soll dabei zu jedem Entwicklungszeitpunkt möglich sein. Aufgaben bzw. Problembeschreibungen sollen als grafische Objekte am MTT erzeugt werden können, damit ein optischer Überblick über einen fortlaufenden Entwicklungsprozess und umfangreichen Aufgabenpool sowie dessen Aufteilung an Benutzer ermöglicht wird. Des Weiteren sollen Aufgaben mit einer intuitiven Geste in Teilaufgaben bzw. Teilprobleme aufgeteilt werden können, um umständliches Löschen und Neuerstellen bzw. Bearbeiten von Aufgaben zu umgehen. Für den Fortschrittsverlauf bzw. das Besprechen von Elementen innerhalb des Diagramms sollen Marker eingeführt werden. Diese sollen mit Symbolen und Farben den Bearbeitungsverlauf grafisch festhalten und somit weiteren Überblick schaffen. Erstellte (Teil-)Diagramme sollen für jeden Benutzer separat, aber auch als ein gemeinsames Diagramm digital gesichert werden können. Ebenso sollen überarbeitete Speicherzustände am Diagramm wieder geladen und leicht ins gemeinsame Klassendiagramm integrierbar sein, sodass erreichte Ergebnisse aus Kleingruppen oder Einzelarbeiten sich zusammentragen lassen. Zu den Standard-Gesten, wie dem Verschieben von Elementen oder Zoomen innerhalb eines Dokuments, sollen Teildiagramme bzw. mehrere Elemente gruppiert verschoben werden können.

Zu den entwicklungsprozess-unterstützenden Funktionen soll der Prototyp grundlegende Fähigkeiten eines Editors zum Erstellen von UML-Klassendiagrammen anbieten. Diese bestehen unter anderem aus dem Erstellen von UML-Elementen, wie Klassen mit zugehörigen Attributen und Methoden, aber auch dem Erzeugen von Assoziationen zwischen Klassen mit Angabemöglichkeiten der Kardinalitäten, Rollen, Assoziationsbeschriftung und Leserichtung. Zwischen den Assoziationen

selbst sollen Restriktionen mit logischen Verknüpfungen, wie {XOR}, möglich sein. Zusätzlich sollen Schlüsselwörter im Java-Kontext für Klassen, Methoden und Attribute ausgewählt, angezeigt und bearbeitet werden können. Für Klassen lassen sich dadurch zum Beispiel Interfaces und abstrakte Klassen sowie für Methoden abstrakte Methoden erstellen. Bei Attributen sind so zusätzlich zu Angaben wie Datentyp und Initialwert auch Informationen über „statische Attribute“ enthalten. Analog dazu soll der Typ einer Assoziation einstellbar sein, damit Generalisierungen, Aggregationen, Kompositionen, Uni- und Bidirektionale Verbindungen zusätzlich zu unspezifizierten Assoziationen realisiert werden können. Um Methoden mittels UML genau zu beschreiben, soll der Prototyp Möglichkeiten zum Visualisieren von Rückgabetyp, Bezeichner der Methode und Typen sowie Bezeichner der Übergabeparameter zur Verfügung stellen. Auf selten auftretende Spezialfälle, wie parametrisierte Klassen, qualifizierte Assoziationen und Eigenschaften zu Attributen, Methoden oder Assoziationen wird in diesem Prototypen verzichtet, um den Rahmen der Arbeit nicht zu sehr auszuweiten. In einigen Fällen, wie beispielsweise einer n-ären Beziehung, bietet es sich an, diese in alternativer Schreibweise durch einfache Assoziationen umzusetzen, die der Editor unterstützt. Zum Erstellen von UML- und Übergangselementen auf dem MTT sollen intuitive Gesten eingesetzt werden, die möglichst hohen Komfort und hohe Benutzerfreundlichkeit an die Benutzer weitergeben.

Eine Evaluierung des Prototypen am Multi-Touch-Tisch soll am Ende der Ausarbeitung die implementierten Konzepte durch Testpersonen prüfen. Daneben soll die Benutzerfreundlichkeit, Funktionalität aber auch die Einsetzbarkeit des Editor-Prototypen betrachtet werden. Benutzertests sollen durch Kleingruppen von drei bis fünf Personen am MTT durchgeführt werden, wobei eine vorgegebene Problembeschreibung analysiert und durch ein UML-Klassendiagramm kollaborativ am Tisch visualisiert werden soll. Gesammelte Testergebnisse werden anschließend ausgewertet, interpretiert und zusammengefasst.

Zu beachten gilt, dass das Ziel der Arbeit *nicht* darin besteht einen vollständig ausgereiften UML-Klassendiagramm-Editor für Multi-Touch-Tische zu konzipieren oder zu implementieren.

1.4 Ähnliche Projekte

Im Folgenden werden bestehende Arbeiten und Prototypen für MTTs zur Erstellung von UML-Klassendiagrammen beschrieben, auf ihre Vor- und Nachteile analysiert und als Grundlage für diese Arbeit verwendet. Besonderer Fokus liegt dabei auf der Unterstützung des Entwicklungsprozesses und der Kollaboration der Diagrammentwickler.

1.4.1 COSMOS

COSMOS (COlaborative Surface for MOdeling Software) ist ein Framework zur einfachen Erstellung von Modellierungs-Anwendungen für interaktive Tische, welche die Kollaboration zwischen System-Entwicklern bzw. -Designern und Kunden unterstützen. Dieses Framework wurde durch ein Team der *Hochschule für Angewandte Wissenschaft Hamburg* erstellt. In der dazugehörigen Ausarbeitung heißt es:

> This paper examines how software development can benefit from the possibilities of these new interface characteristics [like multi-touch technology, gestures and tangibles] in early design stages of a software development project.[6]

Hierunter fällt nach Meinung der Autoren, dass Tabletops nicht nur intuitive Eingabemöglichkeiten im Gegensatz zu den bisherigen digitalen Medien bieten, sondern auch die Kommunikation zwischen mehreren Gesprächspartnern positiv beeinflussen. Dafür spricht ihrer Ansicht nach, dass es einerseits keinen Superuser gibt, der die Kontrolle über ein Medium besitzt, wie beim Arbeiten an einen PC mit mehreren Personen, und andererseits, dass die Beteiligten sich anschauen und miteinander kommunizieren können, anstatt an eine Wand zu starren wie bei einer Beamer-Präsentation. Nicht zuletzt sehen sich diese Argumente dadurch bestätigt, dass Menschen sich trotz bisher bekannten technischen Möglichkeiten immer wieder an einem Tisch zu Besprechungen zusammensetzen. Ein MTT ist, der Meinung der Autoren nach, zusätzlich in der Lage Konsistenz zu bieten, indem nur *ein* Medium zur Bearbeitung *und* Präsentation von Artefakten notwendig ist. So muss ein Artefakt, beispielsweise ein Diagramm, nicht mehrfach konvertiert oder exportiert werden, um allen eingesetzten Medien und Situationen gerecht zu werden. Die Autoren geben zudem an, dass laut Studien die Leserichtung für die einzelnen um den Tisch stehenden Besprechungsteilnehmer nicht optimal sein muss, damit sie ihrer Arbeit nachgehen können und dass sie replizierte Menüs einer zentralen Steuerung vorziehen.

Die Projektmitglieder haben sich bewusst für die Erstellung eines Frameworks entschieden. Mit diesem können zahlreiche unterschiedliche graphische Editoren erstellt werden, die im wesentlichen aus Knoten und Kanten bestehen und die bereits, durch das Framework, Möglichkeiten für eine gemeinschaftliche Zusammenarbeit anbieten. Weiterhin bietet das Framework die Möglichkeit in den Editoren Informationen aus- und einblenden zu lassen. Auf diese Weise werden nur benötigte Informationen auf dem Tisch angezeigt und kein Bildschirmplatz verschwendet. Tangibles helfen zusätzlich den Bildschirm effizient auszunutzen, indem Menüs

[6] Löffelholz, Daniel et al.: COSMOS: Multi-touch support for collaboration in user-centered projects. In Heiss, Hans-Ulrich (Hrsg.): Informatik 2011. Bonn: Ges. für Informatik. Online im Internet: `http://www.user.tu-berlin.de/komm/CD/paper/061521.pdf` – Zugriff am 04.01.2013, ISBN 978–3–88579–286–4, [S. 2].

nur beim Aufsetzen der Tangibles auf den Tisch angezeigt werden. Durch mehrere gleiche Tangibles wird zudem die Anzeige von replizierten Menüs ermöglicht, sodass ein zentrales, für alle Benutzer zugängliches Menü nicht benötigt wird.[7] Insbesondere haben sich die Autoren darauf fokussiert, den Arbeitsplatz auf dem Tisch zu maximieren sowie die Interaktion mit dem Multi-Touch-Tisch intuitiver zu gestalten, um einen kollaborativen Arbeitsstil zu unterstützen.

Um die Tauglichkeit ihres Frameworks unter Beweis zu stellen, haben die Beteiligten in einer Machbarkeitsstudie eine Anwendung zur Besprechung und Verbesserung von UML-Klassendiagrammen auf einem Microsoft Surface Multi-Touch-Tisch entwickelt. Zu den Zielen der mit COSMOS erstellten Anwendungen gehört jedoch *nicht* die Erstellung von Diagrammen auf einem Multi-Touch-Tisch. Artefakte sollen mit Hilfe des Tabletops lediglich besprochen und überarbeitet werden.[8]

1.4.2 DTDiagram

Die Anwendung DTDiagram entstand während eines Semesterprojekts der *Eidgenössischen Technischen Hochschule Zürich* und ist ein Editor für UML-Klassendiagramme auf einem MTT, welcher für einen einzelnen Benutzer ausgelegt ist. Eine Dokumentation bzw. Ausarbeitung zu diesem Projekt ist nicht verfügbar, weswegen die folgende Beschreibung einem Vorführvideo[9] entnommen wurde. Gesteuert wird das Programm mit Hilfe eines Stiftes, diversen Gesten und eines einzelnen Touchpoints. Texteingaben können handschriftlich mit dem Stift geschrieben und mit Hilfe der Anwendung erkannt werden, sobald auf einer freien Fläche auf dem Tisch geschrieben wird. Zusätzlich steht eine virtuelle Tastatur zur Verfügung, welche jedoch viel Platz für sich beansprucht. Die Erstellung von Diagrammen sowie die Interaktion mit dem Programm wird dadurch effizient und intuitiv abgewickelt. Aufgrund von fehlendem haptischen Feedback liefern Ereignisse wie Touchpoints, Gesten und die Auswahl von angezeigten Elementen ein dezentes, optisches Feedback, wodurch der Benutzer die Reaktionen des Systems auf seine Eingaben wahrnehmen kann. Die sinnvolle Überlagerung von Gesten erlaubt den Einsatz weniger, intuitiver Gesten mit alternativem Verhalten. So lässt sich beispielsweise eine Linie zwischen zwei Klassen für eine Assoziation mit der zuletzt gewählten Assoziationsart zeichnen. Ist währenddessen ein Touchpoint aktiv, so erscheint nach dem Zeichnen der Linie ein Kreismenü, in dem die Assoziationsart gewählt werden kann. Die Auswahl der Eingabemethoden und Gesten macht diese Anwendung besonders interessant für unsere Arbeit.

[7] Vgl. Löffelholz, Daniel et al., a. a. O., [S. 8].
[8] Vgl. Löffelholz, Daniel et al., a. a. O., [S. 9].
[9] Selber, Thomas/Pongelli, Stefano/Valente, Giulio: DTDiagram - a Pen and Touch UML Class Diagram editor for DiamondTouch tabletops: Semester project for the Information Systems Lab 2012 at ETHZ (www.ethz.ch). Online im Internet: `http://www.youtube.com/watch?v=hsZOsjf5un4`⟩ – Zugriff am 04.01.2013.

1.4.3 Collabee

Im Rahmen einer einsemestrigen Veranstaltung an der *University of British Columbia* beschäftigte sich ein Projektteam aus fünf Mitgliedern mit der Entwicklung eines Systems und von Interaktionsmethoden, die die gemeinsame Erstellung von Diagrammen fördern. Durch Befragungen und Experimente mit Personen, die beruflich viel mit der Erstellung von Diagrammen zu tun haben, konnten die Projektmitglieder häufig genutzte bzw. bevorzugte Systeme und Methoden ermitteln. Es wurde festgestellt, dass ein Whiteboard (oder vergleichbare Systeme) zur Diagrammerstellung sehr beliebt ist, weil es eine einfache Kommunikation zwischen den Teammitgliedern selbst bei komplizierten Ideen ermöglicht. Zudem können Diagramme damit schnell und flexibel erstellt werden, da keine feste Notation vorgeschrieben ist.[10] Im Regelfall werden die Ergebnisse anschließend in eine handelsübliche PC-Software übertragen, um die Darstellung sowie Handhabung auch von großen und komplizierten Diagrammen zu ermöglichen. PC-Systeme erschweren jedoch die Kommunikation und Kollaboration erheblich, weswegen die Gruppe auf der Suche nach einem neuen Medium war, das die Vorteile beider bisher genutzten Systeme vereint.

Im Rahmen ihrer Forschung sollte gezeigt werden, dass ein Prototyp eines UML-Editors auf dem MTT gegenüber beiden bisherigen Systemen diverse Vorteile bietet und im schlimmsten Fall nicht schlechter abschneidet als die beiden Bisherigen. Dies gelang nur für den Vergleich mit dem PC. Das erheblich schlechtere Abschneiden des Prototypen für MTTs im Vergleich mit dem Whiteboard wurde auf die geringe Komplexität und Größe der Testaufgaben zurückgeführt, wonach der Unterschied bei schwierigeren Aufgaben zumindest minimal ausfallen sollte.[11] Der Prototyp dieses Projekts, wie in einem Vorführvideo[12] zu sehen, wurde mit dem Ziel entwickelt die Vorteile von primitiven Werkzeugen und PC-Software in einem System zu vereinigen, um Benutzern eine effiziente und angenehme kollaborative Erstellung von Diagrammen zu ermöglichen.[13] Dieser diente jedoch rein der Benutzerstudie, weshalb nur die für die Testaufgaben nötigen Funktionen, wie das Erstellen, Auswählen, Manipulieren und Löschen von Klassen und Assoziationen, implementiert wurden. Auf weitere elementare Funktionen, von Menü-Interaktionen bis hin zur Versionierung der Diagramme, wurde verzichtet, um den Programmieraufwand möglichst gering zu halten.[14]

[10] Vgl. Totolici, Alex et al.: Collabee – Multi-touch Collaborative Diagramming. Online im Internet: `http://www.jrejre.net/2010/08/collabee-multi-touch-collaborative-diagramming` – Zugriff am 04.01.2013, Meilenstein 2, S. 4.

[11] Vgl. Totolici, Alex et al., a. a. O., Meilenstein 4, S. 9.

[12] Totolici, Alex et al.: Collaborative UML Diagramming on a Multi-touch Surface. Online im Internet: `http://www.youtube.com/watch?v=K6NCj2czZrE` – Zugriff am 12.08.2013.

[13] Vgl. Totolici, Alex et al.: Collabee – Multi-touch Collaborative Diagramming, a. a. O., Meilenstein 4, S. 3.

[14] Vgl. Totolici, Alex et al., a. a. O., Meilenstein 3, S. 22.

1.4.4 Fazit der bisherigen Projekte

Die drei vorgestellten Projekte zeigen dass Multi-Touch-Tische mit ihren intuitiven Eingabe- und Steuerungsmöglichkeiten ein großes Potenzial besitzen. Aufgrund der guten Kommunikationsmöglichkeiten zwischen mehreren Benutzern am Tisch wird dieser bei Gruppenarbeiten einem PC vorgezogen. Dies wird wiederum durch Benutzerstudien innerhalb der vorgestellten Projekte mit Hilfe ihrer Prototypen belegt. Die vorgestellten Anwendungen gleichen bislang jedoch mehr einem System, welches vom PC auf einen MTT adaptiert und auf ein Mehrbenutzer-System erweitert wurde, wobei die Eingabemethoden und Visualisierungen zum Teil gut an das neue Medium angepasst sind.

In COSMOS wurden allgemeine Probleme bei der Arbeit mit mehreren Personen untersucht und der Nutzen der Anzeigefläche fokussiert, sodass beispielsweise Details und Menüs erst im Bedarfsfall angezeigt werden und nur dort wo der Benutzer sie wirklich benötigt. Jedoch begrenzt das Projekt ihren Prototypen nur auf bestimmte Szenarien, sodass es zur kollaborativen Diagrammerstellung nicht in Frage kommt. Zudem liegt der Fokus des Projekts auf der Erstellung des Frameworks, wodurch es sich nicht ausführlich mit einer einzelnen Diagrammart beschäftigt. DTDiagram überzeugt mit einer sehr intuitiven Umsetzung der Steuerung, die laut dem Dokumentationsvideo eine Erstellung und Bearbeitung von Klassendiagrammen in geringer Zeit ermöglicht. Das dezent wirkende optische Feedback wurde in diesem Prototypen beispielhaft umgesetzt, daher ist dieser Prototyp ein guter Ideenanreiz. Das Projekt Collabee ist ein Grundbaustein eines kollaborativen Diagramm-Editors auf einem Multi-Touch-Tisch. In diesem wurde das Nutzerverhalten bei der kollaborativen Diagrammerstellung untersucht, sich mit dem Auffinden eines geeigneten Mediums beschäftigt und aufgezeigt, dass ein MTT hervorragendes Potenzial in diesem Einsatzgebiet hat. Zudem liefert es wichtige Erkenntnisse, um den Arbeitsprozess während der Erstellung von Klassendiagrammen zu optimieren und stellt dafür wichtige Funktionen aus Benutzersicht heraus. Allerdings ist der Prototyp nur für die Lösung der Beispielaufgaben brauchbar, weswegen Konzepte zur Kollaboration nur minimal ausfallen.

Abschließend lässt sich sagen, dass die drei vorgestellten Prototypen diverse Ansätze von Konzepten zur Kollaboration aufweisen, obwohl sie als minimalistische UML-Editoren umgesetzt wurden. Was jedoch in keinem der bisherigen Projekte ersichtlich wird, ist ein Fokus auf die Kollaboration und die Unterstützung des Entwicklungsprozesses von Diagrammen durch die Software. So lassen sich die vorgestellten Prototypen erst nutzen, wenn ein Problem bereits bis zur Klassenebene besprochen wurde. Zudem müssen trotz einer Checkliste, wie bei COSMOS, zusätzliche Medien für beispielsweise Notizen, Aufgabenverteilung und Zuordnung von Verantwortlichkeiten innerhalb eines Diagramms genutzt werden, um das weitere Vorgehen zu organisieren. Somit ist die Konsistenz des Artefakts zwar gegeben, jedoch nicht die des Entstehungsprozesses des Diagramms, wodurch wiederum verschiedene Medien und Artefakte für ein und denselben Prozess benötigt werden.

1.5 Betrachtung des MTTs als multimodales System

Ein- und Ausgabesysteme stellen eine Kommunikation zwischen Mensch und Maschine dar. Im optimalen Fall ist diese Schnittstelle benutzerfreundlich und natürlich gestaltet, sodass dem Benutzer eine einfache, intuitive und schnelle Eingabe zur Verfügung steht. Ebenfalls sollte die Ausgabe vom Nutzer leicht aufgefasst werden können.

Neben den verbreiteten Desktop-PCs, Laptops, Netbooks und damit einhergehenden Ein- und Ausgabegräten in Form von Computermäusen, Tastaturen und Bildschirmen nimmt die Anzahl von Multi-Touch-Geräten zu. Diese vereinen Ein- und Ausgabegeräte in einer Touch-Oberfläche. Multi-Touch-Geräte, wie Tablets und Smartphones, bieten aufgrund ihrer Bildschirmgröße verschiedene Dimensionen der Touch-Eingaben.

Viele Endbenutzer sind erfahren im Umgang mit Eingaben durch Maus und Tastatur. Diese Eingabesysteme sind sowohl bei Laptops in Form von Mauspad und Tastatur als auch bei Multi-Touch-Geräten als virtuelle Maus bzw. Single-Touchpoint und virtuelle Tastatur zu finden. Durch die Verwendung von bekannten Eingabesystemen wird der Lernprozess im Umgang mit neuen Geräten verkürzt bzw. entfällt. Die Maus und Tastatur als Eingabemittel wurde für einen Personal Computer entwickelt und bietet dem Anwender eine gute, benutzerfreundliche Dateneingabe. Im Gegensatz dazu fehlt eine Eingabemöglichkeit, die speziell nach benutzerfreundlichen Kriterien für Multi-Touch-Flächen entwickelt wurde, um diese anstelle des virtuellen PC-Eingabesystems einzusetzen. Im Folgenden werden die daraus resultierenden Konsequenzen verdeutlicht.

Ein Nachteil von Tastaturen ist das Verdecken von Tasten, die sich unter der aufgelegten Handfläche befinden. Bei der physikalischen Tastatur erhält der Benutzer Orientierung durch fühlbare Tasten und gelerntes 10-Finger-System, aber auch durch abstehende, tastbare Erhöhungen an der F- und J-Taste, welche als Ausrichtungspunkte dienen. Ebenso sorgt ein haptisches Feedback für eine Rückmeldung der Tastenanschläge. Bei virtuellen Tastaturen kann ein haptisches Feedback lediglich durch Vibration erzeugt werden, wobei beim Drücken zwischen zwei Tasten nicht sichergestellt werden kann, welche der beiden Tasten erkannt wurde und ob eine oder zwei Tasten auf einmal gedrückt worden sind. Durch Verwendung von kleinen Bildschirmen und daraus folgenden kleinen Tasten tritt diese Situation häufig ein. Physikalische Tastaturen gibt es in verschiedenen Formen, Farben und Größen. Diese Merkmale sorgen für eine höhere Barrierefreiheit. So können beispielsweise Tastaturen mit extra großen Symbolen für sehschwache Menschen eingesetzt werden. Insbesondere Menschen mit Sehschwächen oder Farbblindheit können ohne großen Aufwand eine geeignete Tastatur wählen. Gleichgestellt mit virtuellen Tastaturen lassen sich auch dort Farben und Zoom-Verhältnisse einstellen, wobei die Größe des eingesetzten Bildschirms den Umfang der einstellbaren

Größen von Tastaturfeldern begrenzt. Dies erschwert besonders bei Geräten mit kleinen Touch-Bildschirmen, wie zum Beispiel Smartphones, die Eingabe. Außerdem beansprucht eine virtuelle Tastatur Anzeigeplatz, der einem Benutzer dadurch nicht weiter zur Verfügung steht. Abschließend lässt sich sagen, dass die aktuellen Eingabesysteme der Multi-Touch-Geräte für kleine und einfache Eingaben ausreichend sind. Für komplexe Aufgaben und längere Eingaben, wie beispielsweise das Programmieren einer Software, wirkt das Eingabesystem unnatürlich und ungeeignet. Dies ist überwiegend auf den mangelnden Platz der Bildschirmflächen zurückzuführen.

Ein- und Ausgabegeräte sollten ihrem Umfeld, Einsatzgebiet, Kontext und vor allem an den Benutzern angepasst werden, um native und intuitive Eingaben zu ermöglichen. Eine solche Voraussetzung bzw. Forderung an Benutzerschnittstellen gewährleistet eine hohe Benutzerfreundlichkeit. Eingaben und Ausgaben sollten anhand ihres Einsatzgebietes entwickelt werden und nicht zwangsweise auf alte Techniken zurückgreifen. Alternative Eingabetechniken stellen Sprachsteuerung, Videosteuerung, Gestensteuerung und Kombinationen aus verschiedenen Systemen dar.

Die Sprachsteuerung wird bereits in Smartphones aber auch PCs genutzt. Bei PCs schafft die Sprachsteuerung eine alternative Eingabemöglichkeit, welche insbesondere der Barrierefreiheit zu Gute kommt. Smartphones haben eine kleine Touch-Oberfläche und die Einblendung einer Tastatur benötigt weiteren Bildschirmplatz. Dadurch bietet die Sprachsteuerung bei Smartphones eine benutzerfreundliche Entlastung und somit unter anderem weiteren Anzeigeplatz für das Gerät. Nach Hahn[15], benötigt eine Sprachsteuerung zudem gute Algorithmen zur Spracherkennung. Diese müssen neben der Erkennung von Sprachmustern aus Geräuschen auch die Spracheingabe des Benutzers aus Tonsingnalen interpretieren können. Dabei müssen verschiedene Sprachen, Stimmlagen, Akzente und weitere Heuristiken beachtet werden.[16] Aus diesen Gründen ist die Spracheingabe im Gegensatz zu Maus und Tastatur unzuverlässiger. Auch das Navigieren mit Hilfe der Sprache erweist sich aus eigenen Erfahrungen als umständlicher.

Die Videoerkennung tritt in vielen verschiedenen Formen auf. Beim Eye-Tracking werden nur spezifische „Gesten“ des menschlichen Auges erkannt und interpretiert. Andere Systeme hingegen nehmen die gesamte menschliche Gestalt wahr und erkennen Gesten durch Körperhaltung, Mimik, Fingerzeichen und Bewegungen. Dabei kann das Umfeld, in dem sich der Benutzer befindet, wahrgenommen werden und eingegebene Gesten im Kontext des Umfeldes behandelt werden. Ebenso können Mimiken eine wichtige Rolle in der Interpretation von Gesten spielen. Häufig werden, Hahn zufolge, Zeigegesten zusammen mit Ausgabegeräten

[15] Hahn, Thomas: Future Human Computer Interaction with special focus on input and output techniques. Online im Internet: `http://www.olafurandri.com/nyti/papers2010/FutureHumanComputerInteraction.pdf` – Zugriff am 12.08.2013.

[16] Vgl. Hahn, Thomas, a. a. O., S. 6f..

verwendet. So kann beispielsweise eine Person einen Button auf einem Bildschirm durch einfaches Zeigen auslösen.[17] Im Gegensatz zu Gesten auf Eingabeoberflächen können bei der Videoerkennung 3D-Gesten verwendet werden. Beispiele hierzu sind Sport-Spiele an Konsolen mit Videoerkennung wie die *XBOX 360* Konsole mit *Kinect*. Durch Videoerkennung werden viele verschiedene Arten von Gesten unterstützt, so können Mehrbenutzereingaben beispielsweise durch Gesichtserkennung ermöglicht werden. Die Komplexität einer Umsetzung solcher 3D-Systeme besteht in den meisten Fällen in der Erkennung bzw. Interpretation der gewünschten Gesten aus Echtzeitaufnahmen.[18] Dabei müssen verschiedene Lichtquellen und Kontraste mit berücksichtigt werden, die Auswirkungen auf die Qualität der Aufnahmen haben. Hilfreiche Informationen können dabei (Infrarot-)Laser liefern, die als Abstandsmessgeräte bzw. zur Rastererkennung und -messung eingesetzt werden.[19]

Die Gestensteuerung auf Oberflächen wird überwiegend auf Multi-Touch-Geräten, wie Tablets, Smartphones und weiteren Touch-Bildschirmen in verschiedenen Größen und Lagen verwendet. Zusätzlich zum Single-Touchpoint gibt es nur wenige geräteübergreifende Gesten, wie Rotieren, Zoomen und Verschieben bzw. Drag & Drop. Dabei sind diese Gesten, abhängig vom verwendeten Gerät, unterschiedlich nutzbar. Kleine Touch-Oberflächen ermöglichen beispielsweise einen schlechteren Gebrauch dieser Gesten als größere. Im Gegensatz zu einer Computermaus lässt der Single-Touchpoint eine natürliche Eingabe des „Zeigens“ zu. Komplexere Eingaben sollten durch Multi-Touch unterstützt werden. Allerdings sind auch hier die Eingaben durch natürliche Gesten stark begrenzt.

Der Anwendungszweck sollte die Wahl des Eingabesystems bestimmen. Wird ein Szenario betrachtet, in dem ein Gerät zur Unterstützung der Zubereitung von Gerichten in einer Küche bereit gestellt wird, wie insbesondere eine Rezeptanzeige, so ist nach Hahn die Video- oder Spracherkennung ein sinnvolleres Eingabemittel als beispielsweise die Multi-Touch-Eingabe, da die Hände des Benutzers frei bleiben und die Arbeit nicht für Eingaben unterbrochen wird.[20] Für komplexe Aufgabenstellungen eignet sich in den meisten Fällen nur selten ein Eingabesysteme alleine. Für eine benutzerfreundliche Variante ist in der Regel ein Kompromiss bzw. eine Mischung von mehreren Eingabesystem nötig. Diese werden auch als multimodal betitelt. Ein Beispiel für die Verwendung eines multimodalen Eingabesystems stellen viele Smartphones dar. Diese bieten neben der Touch-Eingabe durch eine virtuelle Tastatur auch eine Spracheingabe für Texteingaben an. Ein großer Vorteil der multimodalen Systeme ist das Zurückgreifen auf einen großen Pool von möglichen Nutzerinteraktionen. Durch die Verwendung von mehreren Systemen werden dem Benutzer zusätzlich alternative Eingabemöglichkeiten angeboten, welche Barrierefreiheit und Benutzerfreundlichkeit steigern. Ebenso werden

[17] Vgl. Hahn, Thomas, a. a. O., S. 9f..
[18] Vgl. Hahn, Thomas, a. a. O., S. 4f..
[19] Vgl. Hahn, Thomas, a. a. O., S. 3.
[20] Vgl. Hahn, Thomas, a. a. O., S. 7f..

Nachteile von anderen Systemen teilweise gegenseitig ausgeglichen und erlauben eine gute Kommunikation zwischen Mensch und Maschine.

In dieser Arbeit wird der Entwicklungsprozess von Klassendiagrammen auf einem MTT konzipiert. Um die Arbeiten in einer Gruppe oder einer Einzelarbeit möglichst den Aufgaben entsprechend benutzerfreundlich zu gestalten, werden neben der reinen Multi-Touch-Eingabe zusätzlich multimodale Möglichkeiten beachtet. So sollen weitere Geräte verwendet werden, wenn sich ein MTT nicht für eine spezielle Aufgabe innerhalb des Entwicklungsprozesses eignet. Dabei wird die Verbreitung von Smartphones und Tablets ausgenutzt. Außerdem werden Tangibles, physikalische Gegenstände im Kontext des Multi-Touch-Tisches, verwendet, wenn diese die Eingabe erleichtern.

2 Analyse des Entwicklungsprozesses eines Klassendiagramms

In diesem Kapitel wird der Entwicklungsprozess am MTT, der beim Erstellen von Klassendiagrammen durchlaufen wird, benutzerorientiert analysiert. Es werden Schwierigkeiten, Missverständnisse, Bearbeitungsfehler und Probleme aufgezeigt und analysiert, die während der kollaborativen Entwicklung eines Klassendiagramms sowie durch die Nutzung des Multi-Touch-Tisches bei der Entwicklung auftreten können. Außerdem werden notwendige Aufgaben und Funktionen isoliert, die für Benutzer des MTTs unerlässlich zur Realisierung einer Problemstellung bis hin zu einem Klassendiagramm sind. Konzepte möglicher Lösungen dieser Anforderungen werden auf Grundlage der Analyse im nächsten Kapitel ausgearbeitet. Die hier beschriebene Analyse wird dabei aus Sicht der Nutzer geschildert, anstatt einer Software-Analyse und Konzipierung, die Benutzern einen vorgeschrieben system-optimierten Prozess aufzwingt. Dieses Vorgehen ermöglicht eine benutzerorientierte Optimierung der Software, der Arbeitsweise und der Auflösung von Benutzerproblemen.

Da in dieser Arbeit nicht sämtliche mögliche Entwicklungsprozesse, die unter anderem schon durch verschiedene Teamgrößen entstehen, am MTT untersucht werden können, wird die Analyse auf ein Szenario beschränkt. Dabei wird der Prozess in mehrere Entwicklungsstufen, die im folgenden Kontext als Phasen beschrieben werden, unterteilt.

2.1 Beschreibung des Szenarios

Folgende Vorüberlegungen beziehen sich auf eine Projektgruppe, bestehend aus drei bis fünf Teammitgliedern, die zum erfolgreichen Erstellen eines Klassendiagramms Meetings, Teamarbeiten aber auch selbstständig Einzelaufgaben durchführen möchten. Dabei soll ein Multi-Touch-Tisch zur Bewältigung der anfallenden Aufgaben und Besprechungen eingesetzt werden. Damit das Team gleichzeitig und kollaborativ Arbeiten kann, soll der MTT entsprechend der Anzahl der Teammitglieder groß genug sein.

Der Entwicklungsprozess wird dabei in mehrere Phasen unterteilt. In jeder Phase werden bestimmte Anforderungen bzw. Voraussetzungen an alle Entwickler gestellt, die erfüllt werden sollten, um ein optimales und fortschreitendes Arbeiten

am MTT zu ermöglichen. So soll, beispielsweise vor dem Beginn des Entwicklungsprozesses eines Klassendiagramms, das Thema grob verstanden und Teilkenntnisse diesbezüglich vorhanden sein, um ausreichende Besprechungen und Arbeiten am Multi-Touch-Tisch durchführen zu können. Zu den Anforderungen einer Phase werden auch anfallende Aufgaben und Probleme analysiert, die während diese innerhalb des Entwicklungsprozesses anfallen.
Der Verlauf eines möglichen Entwicklungsprozesses, welcher im Folgenden dargestellt wird, stützt sich auf eigene Erfahrungen mit Softwareteams und Probleme bei der Klassendiagramm-Erstellung, die zum Beispiel im Studium durch das Softwaretechnik-Praktikum gesammelt werden konnten.

2.2 Erste Phase: Aufgabenplanung und Verteilung

Der Entwicklungsprozess startet zunächst mit dem Kickoff-Meeting. Dabei wird vorausgesetzt, dass jedes Teammitglied benötigte Anforderungen der zu erstellenden Software teilweise verstanden, benötigte Kenntnisse erworben und sich in das Projektthema eingelesen hat, um ein erstes effektives Planen bzw. Arbeiten innerhalb der Gruppe zu ermöglichen. Im ersten Meeting wird meist das Ziel und die Problembeschreibung des zu entwickelnden Programms grob erläutert, wobei Kenntnisse und Vorstellung des endgültigen Diagramms noch sehr grob und undurchsichtig sind. Daher sollte die Problembeschreibung zu diesem Entwicklungsstand hauptsächlich Schritt für Schritt analysiert und in Teilprobleme unterteilt werden. Zunächst sollte auf UML-Elemente verzichtet und stattdessen Probleme modelliert werden. Dadurch wird das Verständnis der Aufgabe verdeutlicht und sich einer möglichen Lösung schrittweise angenähert, ohne diese durch eine zu hohe Detailschicht, wie die eines strikten Klassendiagramms, aus dem Blick zu verlieren. Das Modell des Problembereiches kann individuell gestaltet werden. Meist werden Aufgaben, Ideen und Gedankengänge zu diesem in einem mindmap-ähnlichen Diagramm visualisiert. Die grafisch abstrakte Darstellung ist dabei meist willkürlich aus ersten Ideen und Überlegungen eines Brainstormings entstanden. Nach der Ideensammlung sollte das Modell neu angeordnet und sortiert werden, um einen Überblick zu gewinnen. Im Collabee-Projekt wurde durch Umfragen ebenfalls erkannt, dass Entwickler von Diagrammen nicht gezwungen werden sollten eine bestimmte Menge von Symbolen zu benutzen bzw. vordefinierte Standards zu befolgen.[1] Um diesen Prozess und eine freie Notation am MTT zu unterstützen, sollte unseren Erfahrungen nach der Editor das Erstellen und Platzieren von Aufgaben und Teilaufgaben bzw. Problemstellungen unterstützen. Außerdem sollten Aufgaben leicht in Teilaufgaben zerlegt werden können, sodass Problemstellungen von groben bis hin zu detaillierten Formulierungen und im späterem Prozess

[1] Vgl. Totolici, Alex et al.: Collabee – Multi-touch Collaborative Diagramming. Online im Internet:
`http://www.jrejre.net/2010/08/collabee-multi-touch-collaborative-diagramming⟩` – Zugriff am 04.01.2013, Meilenstein 2 S. 5.

zu letztendlich strikten UML-Elementen benutzerfreundlich übergeleitet werden können. Verbindungen und gedankliche Verknüpfungen zwischen Aufgaben sollten sichtbar gebildet werden können, durch beispielsweise grafische Verbindungen. Zum allgemeinen Verdeutlichen und Vermitteln von Aufgaben- und Problemstellungen in Besprechungen reicht oftmals ein einfaches Zeigen durch Fingergesten nicht aus, daher sollte der Tisch den Benutzern Möglichkeiten zum Zeigen, Markieren und Zeichnen geben. Zusätzlich zum Formulieren und Zerlegen von Aufgaben- und Problemstellungen werden diese fortlaufend detaillierter analysiert und bearbeitet bzw. wenn nötig wieder verworfen. Folgend sollten erstellte Elemente am Multi-Touch-Tisch auch in verschiedenen Detailstufen bearbeitet und gelöscht werden können sowie Anzeigeoptionen zum Herauf- und Herabsetzen von Detailstufen existieren. Dadurch können unnötige Details durch das Setzen auf geringe Detailstufen ausgeblendet und mehr Details durch das Setzen auf hohe Detailstufen eingeblendet werden. Dies soll Anwendern zu jedem Zeitpunkt das Steuern der angezeigten Informationen ermöglichen, wodurch nicht nur die Übersicht gefördert werden soll, sondern auch das stufenweise Vertiefen von Elementen. Das Ein- und Ausblenden von Informationen wird auch im COSMOS-Framework berücksichtigt.[2] Zudem werden, unserer Erkenntnis nach, Aufgaben im Team verteilt, die außerhalb der festgelegten Meetings bearbeitet werden. Um eine Übersicht über die Verteilung der Aufgaben am MTT anzubieten, sollte der Editor einen einfachen Weg zum optionalen Zuweisen von Benutzern an Aufgaben und Probleme ermöglichen und assoziierte Zuweisungen optisch ansprechend visualisieren. Bei der Bearbeitung von Aufgaben, die außerhalb der Meetings in kleinen Gruppen oder sogar Einzelarbeiten durchgeführt werden, tritt häufig das Problem auf, dass Informationen aus den Besprechungen verloren gegangen sind oder eigene Notizen nicht mehr nachvollzogen werden können. Der Editor sollte dieses Problem mindern, indem Gedankengänge zum Beispiel durch Kommentare und Notizen, die an zugehörige Aufgabenstellungen oder Assoziationen gebunden sind, festgehalten werden können. Dabei sollte die Eingabe intuitiv sein, damit Benutzern weiterhin das typische, schnelle, stichpunktartige Schreiben von Notizen erhalten bleibt.

Auftretende Probleme am MTT während der ersten Konferenz sind unter anderem die Erstbedienung und ungewohnte Eingaben. Benutzer, die das erste Mal mit einem Multi-Touch-Tisch oder speziell mit der verwendeten UML-Editor-Software des Tisches arbeiten, haben keine Erfahrung im Umgang mit der Software und den vorhandenen Gesten. Mögliche Hilfestellungen könnten eine Hilfe-Funktion oder eine leitende Einführung über eingesetzte Gesten am Tisch sein. Zusätzlich sollte der Tabletop alternative Eingabesysteme oder Kombinationen solcher anbieten, um Nachteile der virtuellen Tastatur, wie in Kapitel 1.5 „Betrachtung des

[2] Vgl. Löffelholz, Daniel et al.: COSMOS: Multi-touch support for collaboration in user-centered projects. In Heiss, Hans-Ulrich (Hrsg.): Informatik 2011. Bonn: Ges. für Informatik. Online im Internet:
`http://www.user.tu-berlin.de/komm/CD/paper/061521.pdf` – Zugriff am 04.01.2013, ISBN 978–3–88579–286–4, [S. 6].

MTTs als multimodales System“ beschrieben, auszugleichen und Barrierefreiheit zu schaffen. Ein weiteres Problem stellt der Anzeigeplatz dar. Da sich das Arbeiten am Tisch gerade für umfangreiche Projekte, unter anderem durch die bereits vorhandene große Arbeitsfläche, anbietet, können die zu entwickelnden Diagramme während der Entwicklung an Größe gewinnen und benötigen weiteren Arbeitsplatz. Deswegen sollte die Anwendung geeignete Mittel zur Verfügung stellen, um weiteren Platz bereitzustellen. Dies könnte beispielsweise durch Scrollen oder Verschieben des gesamten Dokuments erreicht werden.

Am Ende eines Kickoff-Meetings sollte die Problemstellung der Software essentiell nachvollziehbar sein, erste Aufgaben geplant und verteilt sowie anfallende Probleme grob besprochen worden sein, um ein anschließendes Bearbeiten von Aufgaben außerhalb des Meetings zu ermöglichen. Damit in der Besprechung erreichte Arbeiten festgehalten und später fortgesetzt werden können, sollte der Editor diese absichern und für spätere Treffen wieder abrufen können.

2.3 Arbeiten in Kleingruppen oder Einzelarbeit

Häufig sollen in Teammeetings verteilte Aufgaben außerhalb der Besprechungen, von einer Person selbstständig oder in Kleingruppen, bearbeitet werden. Innerhalb eines Softwareprojekts werden zudem des öfteren Expertengruppen für größere Teilproblemstellungen gebildet, die ebenfalls eigenständig arbeiten. Sind Aufgaben und Probleme zur Lösungsfindung an Benutzer oder Kleingruppen verteilt, können diese eigenständig an weiteren digitalen Medien bearbeitet werden. Dabei sollten zuvor erstellte Aufgaben, Problemstellungen und Ergebnisse vom Multi-Touch-Tisch ausgelagert werden können, um ein externes und unabhängiges Arbeiten zu ermöglichen. Kleingruppen, die in diesem Szenario aus zwei bis drei Personen bestehen, können so unabhängig interne Probleme lösen und Lösungen zum zugewiesenen Teilproblem weiterentwickeln. Die Arbeiten können hierbei ebenfalls am MTT erfolgen, um von den kollaborativen Vorteilen des Tisches zu profitieren. Dabei ist ein gleichzeitiges, aber separates, arbeiten von Kleingruppen am Tisch denkbar. Aufgrund der Arbeitsfläche sollten allerdings nur maximal zwei Gruppen gleichzeitig am Tabletop agieren. Diesbezüglich sollte die Software entsprechende Unterstützungen, wie zum Beispiel die Verwendung von Split-Screens, bereitstellen.

Ein bekanntes Problem stellen Konflikte bzgl. der Zuständigkeitsbereiche dar, dabei wird eine Aufgabe mehrfach von unterschiedlichen Personen ohne Absprache bearbeitet oder Elemente werden von einem Benutzer bearbeitet, die nicht in seinen Zuständigkeitsbereich fallen. Weiterer Kommunikationsbedarf entsteht an den Schnittstellen *zwischen* Expertengruppen. Diese müssen sich absprechen, um eine geeignete Verbindung zwischen den unterschiedlichen Problem- bzw. Aufgabenbereichen zu schaffen. Weil sich meist jede Gruppe nur in ihren sehr speziellen Fachgebieten auskennt, fehlt das Verständnis für die Themengebiete der jeweils anderen Expertengruppe. Dadurch kann jede Kleingruppe nur eine Seite

der Schnittstelle festlegen und nicht auf die Anforderungen der Anderen eingehen. Der Tisch sollte diese Probleme eingrenzen oder zumindest vor Zuständigkeitskonflikten warnen, wobei Nutzer nicht zu stark eingeschränkt werden sollten, um ein flexibles Bearbeiten beizubehalten. Weiter sollte die fehlende Kommunikation, die als Ursache dieses Problems zugrunde liegt, behoben oder wenn nötig sogar erzwungen werden.

Die Arbeiten erfolgen dabei auf weiteren digitalen Medien, die außerhalb des Anwendungsbereiches des MTT liegen, und werden daher nicht näher beschrieben.

2.4 Zweite Phase: Fortschritte zusammentragen und besprechen

Nach Erarbeitungen in Kleingruppen oder Einzelarbeiten sollen in dieser Phase die gewonnen Fortschritte in einem weiteren Meeting zusammengetragen und besprochen werden. Innerhalb externer Arbeiten erzielte Fortschritte können erste Lösungsansätze, aber auch neu aufgetretene Problemstellungen sein, die sich erst aus der genaueren Bearbeitung der Aufgabe heraus ergeben haben. Dazu sollte die Editor-Software zuvor ausgelagerte bzw. veränderte Speicherstände importieren und integrieren können. Aus disem Grund muss berücksichtigt werden, dass es zu Konflikten während der Integration von mehreren extern veränderten Speicherständen kommen kann. So könnte im einfachsten Beispiel der Name einer Klasse o. Ä. von zwei verschiedenen Benutzern geändert worden sein. Nach dem Laden beider gespeicherter Diagramme stellt sich die Frage, welcher Name übernommen werden soll. Daher sollte der Editor Möglichkeiten anbieten, um verschiedene Speicherstände zu vereinen und im Falle von Konflikten den Benutzern Funktionen zum Auflösen bereitstellen. Durch erhebliche und separate Änderungen pro Person oder Kleingruppe außerhalb der Meetings kann beim Zusammentragen das resultierende Gesamtbild des Diagramms unstrukturiert und unübersichtlich ausfallen. Ein Neustrukturieren und -anordnen nach dem Integrieren externer Inhalte könnte dieses Problem auf dem Tabletop schrittweise minimieren. In tieferen Stadien des Entwicklungsprozesses ist es schwer, einen Überblick über den Verlauf der Arbeiten und Änderungen beizubehalten. So werden während Besprechungen oftmals Aufgaben, Ideen, Lösungen, Konzepte o. Ä. besprochen, die nicht festgehalten werden und in Vergessenheit geraten. Außerdem sind nach Umfragen, die Collabee durchführte und auswertete, nicht alle vorherigen Änderungen einsehbar. Deswegen können Teammitglieder getroffene Entscheidungen ggf. nicht nachvollziehen, wie bereits im Collabee-Projekt festgestellt wurde.[3] Auch können Lösungswege bestimmter Problemstellungen in Sackgassen führen, sodass der Entwicklungsstand bestimmter Elemente zurückgesetzt werden muss. Dazu könnte ein

[3] Vgl. Totolici, Alex et al.: Collabee – Multi-touch Collaborative Diagramming. Online im Internet:
`http://www.jrejre.net/2010/08/collabee-multi-touch-collaborative-diagramming`⟩ – Zugriff am 04.01.2013, Meilenstein 2 S. 5.

Protokoll sowie eine Historie auf dem MTT Abhilfe schaffen. In diesen sollten alle Arbeiten, Eingaben und insbesondere alle Änderungen, Erstellungen, Löschungen und Zuordnungen von Elementen auf dem Tisch aufgezeichnet und Nutzern entsprechende Optionen zur leichten Aufbereitung der Daten angeboten werden: Beispielsweise eine Suche oder ein Filter, um nur Änderungen bestimmter Elemente einzusehen. Während einer Besprechung wird oft ein Protokoll angelegt, um beispielsweise Einigungen, Reviews oder Resultate von Machbarkeitsstudien niederzuschreiben, aber auch um Besprechungspunkte, die in einem nächsten Meeting angesprochen bzw. besprochen werden sollen, festzulegen. Eine Art Checkliste auf dem Tisch, wie bereits in COSMOS zu sehen,[4] könnte dabei den Abarbeitungsprozess der Besprechungspunkte visuell im nächsten Meeting unterstützen sowie das Gedächtnis entlasten und Mitschriften an das Projektdiagramm binden. So werden sämtliche Daten vor Ort angeboten und können persistent gespeichert werden. Fortgeschrittene Teambesprechungen können langwierig und komplex werden. In Reviews werden noch einmal alle Elemente des Diagramms genauer betrachtet und korrekturgelesen. Der Überblick der besprochenen und noch zu besprechenden Elemente verliert sich gerade bei großen und fortgeschrittenen Diagrammen. Der Besprechungsfortschritt sollte durch Unterstützung des MTTs markiert bzw. ersichtlich sein. Dazu bietet sich zum Beispiel eine visuelle Einfärbung der Elemente an. Insbesondere, wenn das Diagramm sehr groß gestaltet ist, sodass es nicht vollständig auf dem Tisch angezeigt werden kann, muss dieses oder einzelne Elemente ständig verschoben werden. Auch bei größeren Teams als in diesem Szenario beschrieben, lassen sich gleichzeitiges Arbeiten und Besprechungen sehr schlecht durchführen, da nur eine kleine Anzahl von Entwicklern mit gutem Sichtverhältnis um den Tisch herum platziert werden kann. Um weiterhin ein kollaboratives Arbeiten zu gewährleisten, sollte der MTT multimodale Alternativen anbieten oder sich mit anderen Systemen verbinden lassen, wie zum Beispiel mit einem Beamer.

Innerhalb von folgenden Teammeetings sollten weitere anstehende Aufgaben sowie Problembeschreibungen besprochen und verteilt werden.

2.5 Zyklus: Teammeetings, Fortschritte und Reviews

Typischerweise bilden im Entwicklungsprozess eines Klassendiagramms Teambesprechungen, Arbeiten in Kleingruppen oder Einzelarbeiten, Zusammentragen von Fortschritten sowie Reviews einen zyklischen Ablauf, der sich fortlaufend wiederholt. Dabei verändert sich das Aufgabengebiet von ungenau formulierten Aufgaben zu letztendlich der Erstellung von strikten UML Klassendiagramm-Elementen, sodass nach einigen Zyklen ein ausgereiftes Klassendiagramm entsteht. Aufgaben

[4] Vgl. Löffelholz, Daniel et al.: COSMOS: Multi-touch support for collaboration in user-centered projects. In Heiss, Hans-Ulrich (Hrsg.): Informatik 2011. Bonn: Ges. für Informatik. Online im Internet: `http://www.user.tu-berlin.de/komm/CD/paper/061521.pdf` – Zugriff am 04.01.2013, ISBN 978–3–88579–286–4, [S. 11].

und Problemen werden in immer kleinere Teile und zu konkreteren Formulierungen zerteilt. Dadurch wird das Verständnis der Aufgaben größer und erste Lösungsansätze für Problemstellungen können gefunden werden. Das noch grobe und notationsfreie Modell des Problembereichs, wie in 2.2 „Erste Phase: Aufgabenplanung und Verteilung“ beschrieben, wird durch weitere Zyklen umfangreicher ausgebaut und folgend weiter beschrieben. Durch das Ausbauen des Diagramms werden immer höhere Detailschichten erreicht, bis letztendlich langsam UML-Elemente eingeführt werden. Die Einführung strikter UML-Elemente erfolgt bei großen Projekten eher langsam und in kleinen Schritten. Außerdem werden diese auch weiterhin in mehreren Durchläufen kontinuierlich bearbeitet und weiterentwickelt. Dabei können auch verwendete UML-Klassendiagramm-Elemente unvollständig sein oder nicht der strikten UML-Notation entsprechen. Eingeführte UML-Elemente werden im ersten Schritt schlicht gehalten und durchlaufen ebenfalls verschiedene Detailstufen, so können Attribute einer neu erstellten Klasse lediglich mit einem Attributnamen bezeichnet sein. Die UML-Elemente können teilweise unvollständig oder sogar fehlerhaft sein, indem zum Beispiel Angaben über Rückgabetyp und Parameter von Methoden fehlen. Das resultierende Diagramm soll den Entwicklern zunächst mehr Verständnis des *Systems* beibringen. Im fortlaufenden Entwicklungsprozess werden diese Details ausgebaut, sodass das Gesamtbild an technischen Informationen gewinnt. Allerdings können sich Aufgabenbereiche von Klassen verschieben oder verlegen. Sobald sich Aufgaben und Zusammenhänge zwischen Klassen größtenteils gefestigt haben, können auch Assoziationen beständige Werte über Kardinalitäten, Rollen und Bezeichner annehmen. Zusätzliche Angaben über Sichtbarkeiten von Methoden und Attributen werden meist erst in abschließenden Phasen pro Klasse erstellt. Trotzdem können zu jedem Zeitpunkt des Entwicklungsprozesses gravierende Änderungen anfallen, die das gesamte System betreffen. Der Editor sollte dazu benötigte informelle Übergangselemente, wie Elemente für die Aufgabenplanung, bereitstellen und strikte sowie unvollständige UML-Elemente visualisieren können. Durch den Entwicklungsverlauf sollte die Anwendung am MTT Bearbeitungen dieser Elemente benutzerfreundlich unterstützen. Der hohe Grad an Informationen verursacht einen Verlust der Orientierung und Übersicht. Gerade bei vielen Assoziationen, die sich teilweise überschneiden, kann sich die Sichtbarkeit, bei Hinzufügen von Kardinalitäten, Rollen und Bezeichnern, erheblich verschlechtern. Der Editor sollte hierzu verschiedene Detailschichten anbieten bzw. unnötige Informationen für den derzeitigen Bearbeitungsvorgang ausklammern.

Als letzte Phase sind abschließende Teamtreffen in Form von Reviews am MTT denkbar.

2.6 Dritte Phase: Ausgereiftes Diagramm

Nach einigen durchlaufenen Zyklen soll in dieser Phase von einem ausgereiften Klassendiagramm ausgegangen werden. Dabei muss das Diagramm weder zwangs-

weise vollständig noch perfekt sein. Allerdings sollten bereits alle informellen Diagrammelemente verarbeitet und zu strikten UML-Elementen übergeleitet worden sein. Oftmals werden auch nach einem vermeintlich vollständigen Klassendiagramm Änderungen vorgenommen, um Mängel bzw. Fehler zu beheben, das Produkt zu optimieren, zu erweitern oder Ähnliches. Bei neuen Problemen oder Aufgabenstellungen, durch Ausdehnungen des Aufgabengebiets der Anwendung o. Ä., werden erneut informelle Elemente benötigt. Die Bearbeitung dieser Elemente erfolgt in mehreren Schritten und Besprechungen, bis schlussendlich UML-Elemente aus diesen entworfen sind. Der Arbeitsablauf dieser Schritte wurde bereits zuvor untersucht, sodass in der weiteren Untersuchung des Entwicklungsprozesses von der Einhaltung strikter UML-Notation ausgegangen wird. Die nachfolgend analysierten Anforderungen und Probleme können bereits in früheren Phasen, durch die Verwendung der UML, auftreten. Der Multi-Touch-Tisch sollte hierzu die Erarbeitung von UML-Elementen anbieten sowie ein späteres Bearbeiten des gesamten Dokuments, auch nach einer ersten Fertigstellung des Diagramms. Ein ausgereiftes Klassendiagramm wird meist den Softwaredokumenten beigefügt, Kunden vorgestellt oder von Entwicklern als Hilfestellung für die Implementierungsphase benötigt. Dazu sollte der MTT das fertige Klassendiagramm in andere Medien, beispielsweise durch Speichern auf einen USB-Stick oder durch Drucken, exportieren lassen. Um die darauf folgende Implementierungsphase einzuleiten und zu unterstützen, könnte der Editor bereits Funktionen, wie zum Beispiel eine automatische Generierung von Programmcode, bereitstellen.

Da in dieser Analyse die Entwicklung des Klassendiagramms unter dem beschriebenen Szenario analysiert werden sollte, werden nachfolgende Phasen, wie die der Implementierungsphase, nicht weiter betrachtet.

3 Konzepte zur Unterstützung des Entwicklungsprozesses

In diesem Kapitel werden eigenständig entwickelte Konzepte vorgestellt, die den Entwicklungsprozess von Klassendiagrammen auf einem Multi-Touch-Tisch unterstützen. Diese Konzepte basieren grundlegend auf der im vorigen Kapitel erfolgten Analyse des Entwicklungsprozesses und bieten Lösungsentwürfe für angesprochene Anforderungen, Aufgaben und Schwierigkeiten an. Weiterhin sind die folgenden Konzepte aus unseren eigenen Ideen entstanden. Sofern Ideen und Konzepte anderer Autoren aufgegriffen und verwendet wurden, sind diese explizit erwähnt worden. Die aufgeführten Konzepte wurden in verallgemeinerten Kategorien, entsprechend ihrer Anforderungen, zusammengefasst.

3.1 Übersicht, Anzeigeplatz und Kollaboration

Die nachstehenden Konzepte befassen sich mit dem Anzeigeplatz des Multi-Touch-Tisches sowie der Übersicht und Kollaboration eines arbeitenden Teams am MTT.

3.1.1 Replizierte Menüs

Das COSMOS-Projekt hat bereits herausgestellt, dass, obwohl die Anzeigefläche eines Multi-Touch-Tisches um ein Vielfaches größer ist als die eines üblichen Monitors, nicht von mehr Platz für die Anzeige von Elementen eines Diagramms ausgegangen werden kann. Das liegt zum einen daran, dass die Auflösung eines Monitors geringer ist.[1] Zum anderen ist, unserer Erkenntnis nach, der Betrachtungsabstand größer, da der Tisch im Stehen weiter von den Augen entfernt ist als ein Monitor, der beim Sitzen üblicherweise direkt vor dem Benutzer aufgebaut ist. Zudem müssen alle Elemente so groß sein, dass sie auch von entfernteren Positionen am Tisch erkannt werden. Außerdem wird der Multi-Touch-Tisch durch Berührungen und eventuell zusätzlichen Stiften oder Tangibles bedient, weswegen die Elemente laut den Entwicklern von COSMOS so groß sein müssen, um diese eindeutig auswählen zu können.[2] Ein weiteres Problem in diesem Zusammenhang,

[1] Vgl. Löffelholz, Daniel et al.: COSMOS: Multi-touch support for collaboration in user-centered projects. In Heiss, Hans-Ulrich (Hrsg.): Informatik 2011. Bonn: Ges. für Informatik. Online im Internet: `http://www.user.tu-berlin.de/komm/CD/paper/061521.pdf` – Zugriff am 04.01.2013, ISBN 978–3–88579–286–4, [S. 5].

[2] Vgl. Löffelholz, Daniel et al., a. a. O., [S. 8].

wie auch die Projekte COSMOS und Collabee aufzeigen, sind die Anwendungsmenüs, die für gewöhnlich dauerhaft Platz verbrauchen und für alle Benutzer gut zugänglich sein müssen. Naheliegend ist, diese zentral in der Mitte des Tisches anzusiedeln, sodass sie von jeder Position am Tisch erreicht werden können. Dadurch ist die Leserichtung allerdings nur für eine Tischseite optimal und der Platz, den alle Benutzer gemeinsam am MTT nutzen, mit Menüs belegt. Die Collabee-Entwickler haben dieses Problem gelöst, indem sie replizierte Menüs in zwei gegenüberliegenden Ecken anzeigen lassen, wie es in Abbildung 3.1 zu sehen ist.

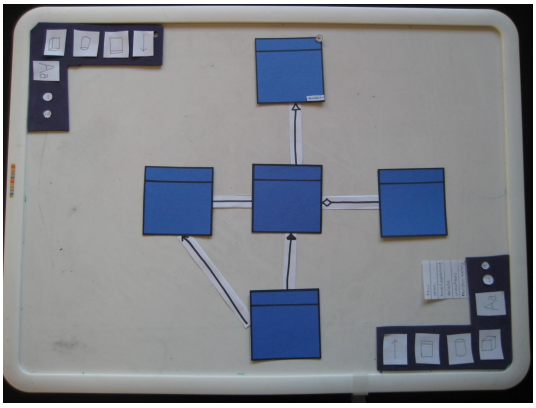


Abbildung 3.1: Collabee benutzt replizierte Menüs in den Ecken des MTTs.[3]

So können die angrenzenden Personen jeweils eins der Menüs nutzen, wodurch die Erreichbarkeit und Lesbarkeit dieser erheblich verbessert wird. Damit sind sie jedoch dauerhaft sichtbar, obwohl sie nur einen Bruchteil der Zeit genutzt werden. Besser wäre es Menüs nach Bedarf anzeigen zu lassen, wofür Gesten oder Tangibles verwendet werden sollten, wie es in der Ausarbeitung zu COSMOS beschrieben wird, um den Anzeigeplatz möglichst effizient zu nutzen. Eine solche Umsetzung würde unseren Erkenntnissen nach einen Mehraufwand bedeuten, da beispielsweise ein Tangible vom Tischrand auf den Tisch abgestellt oder eine Geste aus dem Gedächtnis gezeichnet werden muss, anstelle des Betätigens eines Buttons. Andererseits erscheint das benötigte Menü an der gewünschten Stelle, das die Wege zu und von einem Menü erheblich verkürzen kann. Zudem lassen sich je nach Geste oder Tangible verschiedene Menüs auswählen, sodass nicht sämtliche Menüs bzw. Menüeinträge auf einmal angezeigt werden. Ein weiterer Vorteil dieses Ansatzes ist, Menüs über anderen Elementen anzuzeigen, welche in dem kurzen Moment nicht benötigt werden. So muss ein Benutzer nicht nach freiem Platz zum Öffnen eines Menüs auf dem Tisch suchen und kann diese in seiner unmittelbaren Umgebung öffnen. Wurde ein Menüpunkt ausgewählt, das Menü geschlossen oder das

[3]Entnommen aus: Totolici, Alex et al.: Collabee – Multi-touch Collaborative Diagramming. Online im Internet:
`http://www.jrejre.net/2010/08/collabee-multi-touch-collaborative-diagramming`⟩ – Zugriff am 04.01.2013, Meilenstein 3, S. 7.

Tangible entfernt, so wird auch das angezeigte Menü wieder ausgeblendet, um die ungestörte Weiterarbeit zu ermöglichen.[4]

3.1.2 Anzeige von verschiedenen Detailstufen

Bei umfangreichen Klassendiagrammen bzw. Diagrammen mit Übergangselementen geht oft aufgrund der vielen unterschiedlichen, angezeigten Informationen, wie im Kapitel 2.5 „Zyklus: Teammeetings, Fortschritte und Reviews“ der Analyse geschildert, der Überblick verloren. Dabei können Klassen, Assoziationen oder Übergangselemente zu viele Details anzeigen, die für den momentanen Besprechungs- oder Bearbeitungsvorgang unnötig sind. Eigene Erfahrungen zeigen, dass sich Assoziationen mit sämtlichen Verbindungsinformationen häufig überlappen und dadurch nur noch schwer zu entziffern sind. Auch in frühen Entwicklungsphasen, wie bei der Visualisierung des Problembereichs, deren Elemente noch willkürlich aus ersten Gedankengängen entstanden und dementsprechend angeordnet sind, tritt dieses Problem auf. Oft werden die zuvor zusammengetragenen Elemente und Verbindungen weiterentwickelt oder erneut besprochen. Dabei sollen Elemente des Diagramms durch die Anzeige verschiedener Abstraktionsstufen jeweils hervorgehoben bzw. abgegrenzt werden können, um an Übersicht zu gewinnen. Wobei für jedes Element die Abstraktionsstufe separat angegeben bzw. ausgewählt werden kann. Die Autoren von COSMOS geben an, dass durch das Ausblenden von Informationen wieder Anzeigeplatz freigegeben wird.[5]In der UML gibt es bereits verschiedene Detailgrade der Darstellung, indem Informationen anfänglich schlichtweg nicht angegeben werden. Rupp et al. weisen darauf hin, dass in frühen Phasen der Modellierung die Erfassung der Zusammenhänge im Vordergrund steht.[6] So können Attribute in der einfachsten Darstellung nur mit ihrem Bezeichner angegeben werden oder für eine detailliertere Schreibweise mit Voranschreiben des Datentyps. Bei Methoden werden in der Planungsphase der Klasse Bezeichner von Methoden zuerst ohne Angaben von Rückgabewerten und Parametern aufgelistet. Nach Absprachen und Untersuchungen des Aufgabenbereichs werden diese Werte später ergänzt. Ähnlich detailliert bzw. unvollständig können Assoziationen zwischen Klassen oder Übergangselementen, nur mit einer Linie und ihrem Verbindungstypen, wie bei einer Generalisierung, erstellt werden. Mit der Zeit können weitere Informationen, wie Kardinalitäten, Bezeichner und Rollen hinzugefügt werden. Um auch in späteren Entwicklungsphasen zu einer beliebigen Granularität der Informationen zu wechseln, und nur benötigte Informationen anzuzeigen, sollen Diagrammelemente, wie im COSMOS-Framework, in verschiedenen Detailstufen dargestellt werden können.[7]

[4] Vgl. Löffelholz, Daniel et al., a. a. O., [S. 8].
[5] Vgl. Löffelholz, Daniel et al., a. a. O.
[6] Vgl. Rupp, Chris/Queins, Stefan/Zengler, Barbara: UML 2 glasklar: Praxiswissen für die UML-Modellierung. 3. Auflage. München und Wien: Hanser, 2007, ISBN 978–3446411180, S. 106f..
[7] Vgl. Löffelholz, Daniel et al., a. a. O., [S. 6].

In diesem Konzept werden die Detailstufen in drei Schichten aufgeteilt und Informationen, wie in der Entwicklungsprozess-Analyse festgestellten Reihenfolge (2.5 „Zyklus: Teammeetings, Fortschritte und Reviews“), schrittweise aufgenommen. Dabei beziehen wir uns auf eigene Erfahrungen der Entwicklung von UML-Klassendiagramm-Elementen und betrachten weiterhin nur grundlegende Informationsangaben der UML-Elemente. Die minimale Anzeige soll Bezeichner von Attributen und Methoden anzeigen sowie Assoziationen nur als Linie mit den Navigationstypen darstellen. Die zweite Detailstufe soll zusätzlich den Datentyp bei Attributen, Rückgabewert und Parameter bei Methoden als auch den Verbindungsbezeichner mit Leserichtung bei Assoziationen darstellen. Die letzte Schicht erweitert die Attribute und Methoden von Klassen um Sichtbarkeitsangaben und Assoziationen um Kardinalitäten und Rollen. Weiterhin werden sonstige Informationen, die durch die UML vorgegeben werden, wie beispielsweise Stereotypen, eingeblendet. Unter anderen Gesichtspunkten eines Szenarios könnte der Entwicklungsprozess andere Anforderungen an ein solches Konzept bringen, sodass die zu einem bestimmten Zeitpunkt in der Entwicklung dargestellten Informationen variieren. Daher sollte eine durchdachte Aufteilung und Reihenfolge der Detailstufen zunächst an der eigenen Entwicklung gemessen und festgelegt werden.

Mit einer entsprechend konformen Geste sollen diese Stufen für das gesamte Diagramm, aber auch für einzelne Elemente, erhöht oder erniedrigt werden. Alternativ kann auch eine grafische Benutzerschnittstelle auf ein Smartphone ausgelagert werden, um einerseits weiteren Platz einzusparen und andererseits eine direkte Kontrolle über Anzeigestufen des Diagramms vor Ort zu haben. Eine solche GUI könnte leicht durch grafische Steuerelemente, wie mit Hilfe eines Sliders, dargestellt werden und nötige Funktionalitäten bereitstellen.

Bei Besprechungen bezieht sich das derzeitige Besprechungsthema meist nur auf einzelne Bereiche von Elementen, sodass unnötige Informationen ausgeblendet werden könnten, um damit den Anzeigeplatz effizienter zu nutzen. Eine zusätzliche Detailstufe, die unter der Minimalen eingeführt wird, soll ausgewählte Elemente, die für das laufende Meeting nicht relevant sind, ausgrauen und in den Hintergrund versetzen, wodurch ein höherer Fokus auf dem themenbezogenen Bereich im Diagramm liegt und unwichtige Elemente aufgrund der farblichen Abschwächung leichter ignoriert werden können. Elemente auf einer höheren Schicht sollen im Vordergrund dargestellt werden, um einen hohen Kontrast zu ausgegrauten Elementen zu erzielen und die Lesbarkeit bei Überschneidung beider Elemente anzuheben.

Des Weiteren sollen Klassen ein- und ausgeklappt werden können, weshalb das Konzept der Detailstufen erneut erweitert wird. Eine eingeklappt Klassen soll dabei in einfache Klassen-Notation darstellen werden und eine aufgeklappte Klasse in detaillierterer Notation mit allen Attributen und Methoden. Beide Klassen Notationen entsprechen der UML.[8] Der eingeklappte Modus einer Klasse stellt nur den Klassennamen in ihrem Klassenelement dar. Ist die Klasse ausgeklappt,

[8] Rupp, Chris/Queins, Stefan/Zengler, Barbara, a. a. O., S. 108ff..

wird sie entsprechend ihrer Detailstufe mit weiteren Informationen angezeigt. Für die Attribute und Methoden lassen sich analoge Funktionen umsetzen, allerdings ist der Gebrauch dieser nur in seltenen Fällen begründet, da sich eine Klasse hauptsächlich aus Attributen und Methoden zusammensetzt. Somit ist entweder die *gesamte* Klasse für den momentanen Besprechungsaufwand wichtig oder kann mit sämtlichen Attributen und Methoden ignoriert werden.

Bei der Bearbeitung eines Elements ist ein Gesamtüberblick des Dokuments nicht erforderlich. Allerdings werden für dessen Bearbeitung sämtliche Details des Elements sowie einige außerhalb liegende Randinformationen, wie zum Beispiel Assoziationen, benötigt. Diese Informationen sollen in *jeder* Detailstufe während des Bearbeitungsprozesses angezeigt werden, auch wenn diese nach der aktuellen Detailstufe ausgeblendet sind. Wird ein Übergangs- oder Klassenelement bearbeitet, soll dies sowie alle Elemente, die durch eine direkte Assoziation verbunden sind, ausgeklappt und auf höchster Detailstufe angezeigt werden. Ebenfalls sollen alle Assoziationen, die von dem zu bearbeitenden Element ein- oder ausgehen, vollständig angezeigt werden. Das resultierende Anzeigeergebnis entspricht einer „Sternansicht“. Wird eine Assoziation bearbeitet, soll diese und die verbundenen Elemente vollständig angezeigt werden, sodass bereits während der Bearbeitung der Assoziation, die die Beziehung zwischen den Elementen darstellt, Aufgabengebiete der verknüpften Elemente nachgelesen werden können. Diese (Stern-)Ansicht ermöglicht Benutzern Zugriff auf alle relevanten Informationen, die für die Bearbeitung von Elementen hilfreich sind, und unterstützt zudem das schrittweise Besprechen von jeweils einzelnen Elementen.

3.1.3 Anmeldung am Multi-Touch-Tisch

Sobald mehrere Personen gleichzeitig an einem Multi-Touch-Tisch arbeiten, werden für verschiedene Funktionalitäten Nutzerprofile o. Ä. benötigt, mit denen sich Personen identifizieren. Auf diese Weise können Personen in einer Anwendung unterschieden werden, um ihnen beispielsweise Aufgaben oder Verantwortlichkeiten, wie im Konzept 3.1.4 „Zuweisung von Verantwortlichkeiten“ beschrieben, zuzuweisen. Eine naheliegende und von traditionellen Systemen bekannte Möglichkeit ist, Profile mit WIMP-Eingabemasken von Benutzern auf dem Tisch anzulegen. Dies ist eine simple Möglichkeit und sollte als Alternative für komplexere Anmeldeverfahren, die im Folgenden beschrieben sind, beibehalten werden.

Eine bequemere Möglichkeit wäre, die Benutzer anhand ihrer persönlichen Gegenstände zu identifizieren, die sie häufig während der Arbeit am Tisch bei sich tragen. Häufig wird dazu ein Gegenstand mit einem (Bild-)Code versehen, sodass dieser, durch eine Kamera unterhalb der Tischfläche, als Tangible erkannt wird.

Einige verschiedene Typen dieser für die Tangible-Erkennung eingesetzten (Bild-) Codes zeigen Oppold et al. in ihrer Arbeit auf.[9] So ließe sich ein Code, wie beispielsweise ein QR-Code, auf Firmen-, Studenten- oder vergleichbaren Ausweisen drucken sowie unter die persönliche Kaffeetasse oder auf die Rückseite des eigenen Mobiltelefons kleben, um Benutzer zu identifizieren. Bei Ausweisen könnten auch andere Erkennungsmerkmale, wie die Personalnummer oder ein zugehöriger Barcode sowie das Mitarbeiterfoto zur Wiedererkennung genutzt werden. Bei Smartphones ließe sich der Code auch als Bild auf dem Bildschirm anzeigen und das Telefon, mit dem Display nach unten, auf den Tisch legen. Es muss jedoch bedacht werden, dass das Handydisplay dabei zerkratzen könnte und dementsprechend eine passende Unterlage benötigt wird, damit sich Smartphone-Besitzer auf diese Möglichkeit einlassen. Zudem könnten Mitarbeiter einen spezifischen Gegenstand nicht besitzen oder ausnahmsweise nicht bei sich haben. Ein weiteres Problem ist, dass auf dem Tisch liegende Tangibles dauerhaft Platz verbrauchen und daher nur kurz für die Anmeldung aufgelegt werden sollten. Damit ließe sich der genutzte Platz anschließend zum Arbeiten verwenden. Es sei denn, der Gegenstand besitzt weitere Funktionalität, wie ein personalisiertes Menü, welches um den aufgelegten Gegenstand angezeigt wird. In diesem Fall kann der Gegenstand weiterverwendet werden.

Ein anderer Ansatz ist eine im oder am Tisch integrierte Kamera, die mit Hilfe einer Gesichtserkennung die an den Tisch getretenen Benutzer erkennen kann, wie es bereits bei Smartphones möglich ist.[10] Die Benutzer müssen sich lediglich vor der Arbeit am MTT der Reihe nach kurz vor die Kamera stellen, wenn nicht mehrere davon verbaut werden sollen. Diese Lösung erfordert neben der Kamera auch eine Software zur Gesichtserkennung.

Weiterhin lassen sich Smartphones, welche bekanntermaßen in ihrer Verbreitung zunehmen, über Funktechnologien, speziell mit geringen Reichweiten, zur Anmeldung nutzen. Mit Hilfe einer Anwendung könnten sich Benutzer sogar bei großen Teams bequem von ihrem Mobiltelefon aus am Multi-Touch-Tisch anmelden, sobald sie in dessen Nähe sind. Zusätzlich ist eine automatische Anmeldung durch eine dauerhaft auf dem Smartphone laufende Anwendung denkbar, sodass ein Benutzer automatisch erkannt und sein Name bzw. Profil auf dem Tisch angezeigt wird. Vorteilhaft ist dies, wenn der MTT abseits steht, beispielsweise in einem Besprechungsraum, wodurch es offensichtlich ist, dass herangetretene Personen an diesem arbeiten wollen. Nachteilig ist eine automatische Anmeldung, sobald der Tisch an einem Platz steht, an dem viele Personen vorbeikommen und daher umgehend angemeldet werden, obwohl sie dieses nicht wünschen. Währenddessen ist

[9] Oppold, Pete et al.: Multi-Touch Table with Object Recognition: codename Planck. Online im Internet: `http://eecs.ucf.edu/seniordesign/fa2011sp2012/g14/Documents/SD1%20Group%2014%20Final%20Documentation_v1.1.pdf` – Zugriff am 12.08.2013, S. 13.

[10] Vgl. Anonymus: Mobilgeräte entsperren: Google patentiert Methode zur Gesichtserkennung. In: Spiegel Online. Online im Internet: `http://www.spiegel.de/netzwelt/gadgets/google-erhaelt-%20patent-auf-entsperren-per-gesichtserkennung-a-854294.html` – Zugriff am 12.07.2013.

eine manuelle Anmeldung durch den Nutzer in jeder Situation ohne Einschränkung möglich. Eine ähnliche Lösung könnte mit Fingerabdrücken erreicht werden, die durch Lesegeräte oder Kameras vom Tisch erkannt werden.

Werden bestehende Diagramme geladen, so lassen sich Nutzerprofile automatisch aus diesen entnehmen, sofern Profile mitgespeichert werden. Das Vorgehen garantiert nicht, dass die entsprechenden Personen tatsächlich anwesend sind, ermöglicht jedoch eine Einplanung von abwesenden, am Diagramm beteiligten Teammitgliedern, wie beispielsweise bei der Zuweisung von Aufgaben und Verantwortlichkeiten. Neu hinzukommende Nutzer müssten sich weiterhin mit den bereits vorgestellten Verfahren anmelden.

Um das Arbeiten mit einem MTT komfortabler zu gestalten, können in Nutzerprofilen zusätzliche Daten gehalten werden. So könnten am Tisch arbeitende Personen jeweils festlegen, wo die bearbeiteten Klassendiagramme für sie, in einem Netzwerk bzw. auf einem portablen Speichermedium, abgelegt werden sollen oder ob sie diese per Bluetooth, E-mail o. Ä. zugeschickt bekommen möchten. Selbiges wäre auch für die zu ladenden Diagramme möglich wodurch ein Suchen der Diagramme im Dateisystem entfällt. Wiederum eignet sich hierfür ein Smartphone, welches das Profil seines Besitzer dezentral zur Verfügung stellen könnte, wenn eine zentrale Lösung, beispielsweise mit Hilfe einer Datenbank, unerwünscht ist.

Abschließend lässt sich sagen, dass eine Kombination von mehreren Anmeldeverfahren genutzt werden sollte, um Benutzerfreundlichkeit und Barrierefreiheit sicherzustellen. Welche Verfahren kombiniert werden sollten, hängt vom Verhalten der Nutzer des Multi-Touch-Tisches und dessen Örtlichkeit ab. Ebenso spielt die Teamgröße eine Rolle. Während es beispielsweise für ein kleines Team unproblematisch ist, sich direkt am Tisch anzumelden, kann dieses Verfahren bei größeren Teams zu Zeit- und Koordinationsproblemen führen.

3.1.4 Zuweisung von Verantwortlichkeiten

Während der Entwicklung eines Diagramms besteht dieses aus vielen Klassen, Assoziationen und Übergangselementen, die von verschiedenen Teammitgliedern bearbeitet werden. Damit diese sich nicht gegenseitig behindern oder Arbeiten mehrfach verrichten, werden Elemente zwischen den Teammitgliedern aufgeteilt. So hat jedes Element maximal einen Verantwortlichen, mit Ausnahme der Assoziationen. Diese verbinden Elemente und liegen somit im Zuständigkeitsbereich der Verantwortlichen, deren Elemente durch diese Assoziation verbunden sind. Zudem müssen Elemente nicht zwangsweise einem Teammitglied zugeordnet sein. Das heißt, es kann Elemente geben, die zu einem Zeitpunkt von niemandem betreut werden, wie beispielsweise Aufgaben in Übergangselementen, die erst in Zukunft bearbeitet werden müssen. Weiterhin soll es möglich sein, den Verantwortlichen eines Elements zu ändern, sodass das Team in der Arbeitsverteilung flexibel bleibt. Damit die Teammitglieder sich nicht ihre Verantwortlichkeiten merken müssen und die Zuständigen weitere Elemente auf einen Blick sehen können, soll der Name des Verantwortlichen direkt am Element notiert sein. Denkbar ist

auch ein kleines Profilfoto des Zuständigen für eine schnellere Erkennung. Damit die Mitarbeiter den Überblick über die Aufteilung behalten und Elemente eines Einzelnen schneller erkennen, sollen diese zudem farblich hinterlegt oder umrandet werden, da Farben schneller wahrgenommen werden als Texte. Allerdings sollen diese so dezent gewählt werden, dass sie die Mitarbeiter nicht von ihren Aufgaben abhalten, indem sie die Aufmerksamkeit auf sich ziehen.

3.2 Darstellung und Bearbeitung von Elementen

Nachfolgend werden Konzepte zur Darstellung und Bearbeitung von Diagrammelementen beschrieben, um die Bearbeitung dieser möglichst kollaborativ und benutzerfreundlich zu gestalten. Diese entstammen sowohl der formalen UML--Klassendiagramm-Notation als auch einer eigenen informellen Notation.

3.2.1 Aufgaben und Problemstellungen visualisieren

Ein ausgereift entwickeltes Klassendiagramm, für beispielsweise eine umfangreiche Software, entsteht in der Regel erst nach mehreren Teamtreffen und wiederholten Überarbeitungen. Dabei ist eine gute Planung und Teamkommunikation Voraussetzung. Vor der eigentlichen Entwicklung eines vollständigen und strikten UML-Klassendiagramms werden zunächst Ideen gesammelt und mögliche Lösungen sowie Umsetzungen diskutiert. Dabei wird anfänglich auf allgemeine Fragen, bezogen auf beispielsweise Architektur der Software und verwendete Technologien, eingegangen.

Nachdem konkrete Aufgabenbereiche und Komponenten definiert sind sowie Problemstellungen herausgestellt wurden, können Aufgaben und Probleme in kleinere Teile aufgeteilt werden. Diese können sukzessiv weiter zerlegt werden, bis letztendlich ggf. eine Aufgabe oder eine Problemlösung einer UML-Klasse entspricht. Der Übergang zu einem striktem UML-Klassendiagramm wird dabei von Ideen, Lösungsansätzen, unvollständigen Klassen und Übergangselementen, die nicht Teil der UML-Notation sind, aber einen erheblichen visuellen Wert für den Entwicklungsverlauf ergeben, begleitet. Um diesen Entwicklungsprozess für alle Teammitglieder transparent zu halten, soll es nicht nur möglich sein UML-Elemente zu erstellen, sondern ebenfalls Aufgaben und Problembeschreibungen auf dem MTT zu visualisieren und festzuhalten. Diese können bei Bedarf weiter in Teilaufgaben zerlegt und Benutzern zugewiesen werden. Im Folgenden sollen diese informellen Elemente als Aufgabenelemente bezeichnet werden. Die Informationen die ein solches Element beinhalten sollte, wären zum Beispiel der Aufgaben- bzw. Problemtitel und dessen Beschreibungstext.

Auch im Collabee-Projekt, das bereits im Kapitel 1.4.3 „Collabee“ erwähnt wird, ist der Bedarf an informellen Elementen für die Entwicklung von Klassendiagrammen erkannt worden. Durch Befragungen haben die Projektmitglieder festgestellt, dass Diagrammentwickler zunächst eine flexible Notation, in der sie

beispielsweise auch Zeichnen können, gegenüber der strikten UML-Notation bevorzugen.[11] Während der weiteren Entwicklung sollten, den Befragungsergebnissen nach, striktere Notationen eingeführt werden, um Einigkeit über die Bedeutung der verwendeten Symbole herzustellen. Deswegen sollte, den Autoren nach, der Collabee-Prototyp eine Verbindung zwischen solchen unstrukturierten und strukturierten Eingaben schaffen.[12] Auf diesen Erkenntnissen basierend, haben wir das im folgenden weiter beschriebene und von uns entwickelte Aufgabenelement als Verbindung zwischen unstrukturierten und strukturierten Eingaben eingeführt.

Die Visualisierung solcher Aufgabenelemente sollte sich möglichst von den in UML-Notation definierten und meist rechteckigen Klassendiagramm-Elementen abheben, weshalb Aufgabenelemente durch beispielsweise eine Kreis-, Blasen- oder Wolkenform dargestellt werden sollten. Um ein Aufgabenelement und damit die repräsentierte Aufgabe in kleinere Teilaufgaben zu zerlegen, kann das Aufgabenelement mit einem Finger von oben nach unten „zerschnitten“ werden, wodurch dieses in mehrere (Teil-)Aufgabenelemente zerfällt. Dabei enthalten die neu gebildeten Aufgabenelementen als Orientierungs- und Gedächtnisstütze den Inhalt der zerlegten Aufgabe, auf dessen Basis nun eine Teilaufgabe formuliert werden soll. Damit diese Elemente nicht mit dem ursprünglichen (Haupt-)Aufgabenelement verwechselt werden, könnten weiterhin Aufgabenelemente, die durch das Teilen einer Aufgabe gebildet wurden, in der Titelbezeichnung einen Prefix von „Teilaufgabe von ...“ mit sich führen. Wobei „...“ den Titel bzw. Namen des zerteilten Elements bezeichnet. Dieser Titel dient als Platzhalter der Teilaufgabe und sollte nach dem Teilen sinnvoll umbenannt werden. Soll eine Teilaufgabe umbenannt werden, so kann im Bearbeitungsmodus des Textfeldes der Inhalt automatisch markiert werden, um ein manuelles Entfernen des Vorgabewertes zu erleichtern.

Damit der Ursprung einer Teilaufgabe und Verbindungen zwischen Teilaufgaben des selben Ursprungs bzw. der selben Hauptaufgabe visuell erkennbar ist, soll zudem jeweils eine gemeinsame Hintergrundfarbe verwendet werden, um den Aufgabenbereich zu kennzeichnen. Wird eine neue Aufgabe erstellt, so wird ihr eine zufällige Farbe, die möglichst noch kein Element benutzt, zugewiesen. Beim Teilen von Aufgabenelementen wird die Farbe an die Teilaufgabenelemente *weitergegeben*, um einerseits eine Zugehörigkeit unter den erstellten Teilaufgaben zu bilden und andererseits gedanklich auf die nicht mehr existente Hauptaufgabe zu referieren. Des Weiteren soll der Hintergrund dezent dargestellt werden, da die zu übertragende Information der Zugehörigkeit zur Hauptaufgabe eine geringere Priorität als andere Informationen des Elementes, wie beispielsweise der Aufgabenstellung oder der, nach dem Konzept 3.1.4 „Zuweisung von Verantwortlichkeiten“ beschriebenen, Zuweisung eines Verantwortlichen, besitzt. Somit wird Benutzern die Zugehörigkeit des Aufgabenbereichs nicht durch zu hohe Kontrastwerte aufge-

[11] Vgl. Totolici, Alex et al.: Collabee – Multi-touch Collaborative Diagramming. Online im Internet: `http://www.jrejre.net/2010/08/collabee-multi-touch-collaborative-diagramming`⟩ – Zugriff am 04.01.2013, Meilenstein 2 S. 4.

[12] Vgl. Totolici, Alex et al., a. a. O., Meilenstein 2 S. 6.

zwungen. Mögliche Konflikte aufgrund der farblichen Markierung von Elementen, durch beispielsweise das Konzept für Verantwortlichkeiten, sind denkbar und sollten vermieden werden.

Die hier verwendete Geste des Zerlegens soll dem Nutzer ein optisches Feedback der ausgeführten Aktion geben und eine intuitive Benutzung der Geste ermöglichen. Eine Animation für das Teilen der Aufgabenelemente könnte dieses Feedback weiterhin verstärken.

Zusätzlich zum Erstellen und Teilen von Aufgaben sollen diese mit anderen Elementen, durch zum Beispiel Assoziationen (ähnlich) der UML-Notation oder einer freien notationslosen Darstellung, verbunden werden können. Auch Einschränkungen zwischen Verbindungen, wie `XOR`, sollen hier bereits nutzbar sein. Beim Teilen von Aufgabenelementen könnten diese an alle oder nur an eine Teilaufgabe weitergegeben werden, wodurch bestehende Assoziationen des geteilten Elements überprüft und ebenfalls aufgeteilt werden müssen. Beim Kopieren dieser Assoziationen an *alle* Teilaufgaben, müssen nach dem Zerlegen keine Assoziationen neu erstellt werden und können leicht durch einfaches Löschen der redundanten Verbindungen aufgeteilt werden. Ein Nachteil ergibt sich beim Zerlegen in viele Unterelemente, da die Übersicht bei vielen kopierten Assoziationen verloren geht. Daher bietet es sich an, bestehende Assoziationen nur an *ein* Teilaufgabenelement weiterzugeben und Assoziationen manuell an die anderen Aufgabenelemente zu verschieben. Dies kann zum Beispiel durch Dockingpunkte an den Assoziationsenden, wie im Kapitel 3.2.8 „Assoziationspfad mit Fix- und Dockingpunkten beliebig führen“ beschrieben, erreicht werden, was allerdings einen erhöhten Implementierungsaufwand mit sich bringt.

Durch den Einsatz dieser Aufgabenelemente lassen sich somit bereits in frühen Phasen, ohne UML-Notation, Gedankengänge und Ideen festhalten und visualisieren. Da diese Elemente bis zu Aufgaben, die eine konkrete Klasse beschreiben, heruntergebrochen werden sollen, kann das Erstellen dieser entworfenen Klassen durch eine weitere Geste, die ein Aufgabenelement in eine UML-Klasse umwandelt, erleichtert werden. Der Titel des Aufgabenelements könnte dabei als Klassenname und der Aufgabeninhalt als Kommentar übernommen werden. Kommentare können mit Hilfe der UML-Notation durch Rechtecke, die mit der Klasse durch eine gestrichelte Linie verbunden sind und deren rechte obere Ecke umgeknickt ist, dargestellt werden.[13] Alternativ können diese durch eine entsprechende Kommentarfunktionalität, wie beispielsweise im Kapitel 3.4.2 „Kommentarfunktion“ erläutert, direkt am Klassenelement angezeigt werden.

3.2.2 Optisches Feedback

In der Realität kann nahezu jede Zustandsveränderung, aufgrund diverser Arten von Feedback, mitverfolgt werden. Computer hingegen liefern nur soweit Feed-

[13] Vgl. Rupp, Chris/Queins, Stefan/Zengler, Barbara: UML 2 glasklar: Praxiswissen für die UML-Modellierung. 3. Auflage. München und Wien: Hanser, 2007, ISBN 978–3446411180, S. 37f..

back wie es in ihren Systemen einprogrammiert ist und können in kürzester Zeit Zustände und Informationen soweit verändern, dass sie vom Benutzer nicht mehr nachvollzogen oder verstanden werden können. Daher muss ein Computerprogramm bzw. -system dem Nutzer deutlich machen, welche Zustandsänderungen eingetreten und wie sie zustande gekommen sind, sodass der Benutzer diese nachvollziehen kann. Eine Möglichkeit wäre es das Ergebnis oder die Veränderung textuell zu beschreiben, was für den Nutzer mit Lese- und Zeitaufwand verbunden ist. Eine andere Feedback-Möglichkeit, die COSMOS für Anwendungen fordert,[14] ist dem Nutzer mit Hilfe von Animation zu zeigen was geschieht und so zu erklären wie das Ergebnis zustande kommt. Dies kann weniger Leseaufwand bedeuten und einem Benutzer zudem das Geschehen bildlich darstellen. Weil Animationen zum Abspielen eine feste, wenn auch kleine Zeitspanne brauchen, ist es nicht ratsam diese für jede Aktion, als optisches Feedback, zu nutzen. Deshalb sollten Animationen nur Zustandsveränderungen darstellen, die beim Anwender zu Verständnisproblemen führen oder die Nutzer nur schwer nachvollziehen können, sodass weitere Erklärungen unnötig werden.

Weiterhin betonen die Mitglieder des COSMOS-Projekts, dass Touch-Eingabesysteme kein haptisches Feedback liefern, wie es von Computertastaturen und -mäusen bekannt ist. Um dies auszugleichen sollen alle Eingaben durch Touch-Systeme in ihrem Prototypen mit optischem Feedback bestätigt werden.[15] Uns sind keine weiteren sinnvollen Feedback-Möglichkeiten des MTTs als die der visuellen Bestätigung bekannt, weshalb wir uns der Aussage der COSMOS-Autoren anschließen. Ein akustisches Feedback wäre möglich, wirkt jedoch entgegen eines kollaborativen Arbeitens, weil Signale von anderen Teilnehmern ebenfalls wahrgenommen werden und zu Missverständnissen führen bzw. als störend empfunden werden.

3.2.3 Undo und Redo

Eine Funktionalität, um vollzogene Änderungen rückgängig zu machen bzw. diese Änderungen anschließend erneut durchzuführen, wird unter dem Namen *Undo* bzw. *Redo*, in den verschiedensten Arten von Software, angeboten. Da an einem Multi-Touch-Tisch mehrere Personen gleichzeitig arbeiten und damit parallel Daten innerhalb der Anwendung manipulieren, ist ein diagrammweites Undo bzw. Redo, wie es aus Ein-Benutzer-Systemen bekannt ist, nicht sinnvoll. Nach eigenen Änderungen können verschiedene Manipulationen anderer Tischbenutzer folgen, die im Falle eines allgemeinen Undos bzw. Redos, beim Zurücksetzen ebenfalls rückgängig gemacht werden müssten, um den Entwicklungsstand vor den eigenen

[14] Vgl. Löffelholz, Daniel et al.: COSMOS: Multi-touch support for collaboration in user-centered projects. In Heiss, Hans-Ulrich (Hrsg.): Informatik 2011. Bonn: Ges. für Informatik. Online im Internet: `http://www.user.tu-berlin.de/komm/CD/paper/061521.pdf` – Zugriff am 04.01.2013, ISBN 978–3–88579–286–4, [S. 6].

[15] Vgl. Löffelholz, Daniel et al., a. a. O., [S. 9].

Änderungen zu erreichen. Stattdessen sollte, unseren Erkenntnissen nach, einem Benutzer eine Auflistung der zuletzt durchgeführten Änderungen, die rückgängig gemacht werden können, angezeigt werden. Dadurch ist jeder Tischbenutzer in der Lage gezielt Änderungen zurückzusetzen bzw. zu wiederholen ohne den Zwang weitere oder fremde Bearbeitungsfortschritte rückgängig machen zu müssen. Diese Funktionalität sollte sich jeweils auf einzelne Klassen, Aufgabenelemente und Assoziationen beziehen, sodass die Auflistung übersichtlich bleibt und gesuchte Änderungen schnell gefunden werden können. Insbesondere, wenn mehrere Personen gleichzeitig arbeiten, ist zu erwarten, dass eine allumfassende Auflistung sämtlicher Arbeiten am Tisch keine Übersichtlichkeit bieten kann. Dieser Nachteil einer globalen Auflistung wird durch kontinuierlich hinzukommende Änderungen weiter verstärkt. Explizit sollen in der Auflistung Manipulationen von Informationen eines Elementes, wie das Hinzufügen und Löschen von Attributen und Methoden als auch das Verschieben des Elements, durch die Undo-/Redo-Funktionalität erfasst werden. Filteroptionen in der Auflistung können dabei die Suche nach Änderungen einschränken und somit diese verbessern bzw. verkürzen.

Eine andere Alternative wäre ein Undo/Redo-Konzept personenabhängig umzusetzen, sodass jeder Bearbeiter eines Diagramms nur eigene Änderungen zurücksetzen kann. Dieses ist problematisch, weil einerseits Touchpoints, und somit Eingaben und Änderungen, einzelnen Personen bislang nicht ohne Weiteres zugewiesen werden können und andererseits die Teammitglieder voneinander abhängig sind, da sie Manipulationen von Anderen nicht eigenständig zurücksetzen können.

Die diagrammweite Anwendung der Undo-/Redo-Funktionalität soll, mit Hilfe der Auflistung, nur die Erstellung bzw. Löschung von Klassendiagramm-Elementen erfassen, sodass gelöschte Elemente bei Bedarf wiederverwendet werden können. Alternativ können gelöschte Elemente in einem digitalen Papierkorb gesammelt und von dort zur erneuten Verwendung auf die Diagrammfläche gezogen werden. Das Papierkorb-Konzept für gelöschte Elemente entstammt dem COSMOS-Projekt, in welchem ein Tangible in Form eines Papierkorbs genutzt wurde.[16]

3.2.4 Verschieben von gruppierten Elementen

Bei der Arbeit an Diagrammen werden diese immer wieder umstrukturiert und Elemente des Diagramms verschoben. Dazu müssen oftmals mehrere Elemente oder Gruppen von Elementen verschoben werden, wobei sich die Anordnung innerhalb einer Gruppe nicht verändern sollte. In diesen Fällen hat es sich bei erfolgreichen Ein-Benutzer- und webbasierten Diagrammeditoren nicht bewährt, Diagrammelemente einzeln zu verschieben und anschließend erneut anzuordnen. Analog wird der Bedarf eines gruppierten Verschiebens an Multi-Touch-Diagrammeditoren erwartet, weswegen die betroffenen Elemente des Diagramms erst mit einer Geste zusammengefasst und anschließend als eine Einheit verschoben werden. Dazu wird ein Rahmen um alle zu verschiebenden Elemente gezeichnet, wobei keine

[16] Vgl. Löffelholz, Daniel et al., a. a. O., [S. 10].

besondere Form eingehalten werden muss, um die Auswahl möglichst einfach und präzise zu halten. Sobald der Rahmen geschlossen wurde, wird der umrandete Bereich optisch deutlich hervorgehoben. Alle Elemente innerhalb dieses Rahmens können als gruppierte Einheit, durch dieselbe Aktion wie beim Verschieben eines Elements, beliebig oft verschoben werden. Der Rahmen kann anschließend beispielsweise durch eine simple Wisch-Geste zerstört werden, um die Auswahl aufzulösen. Sollte währenddessen ein Element zu wenig oder zu viel umrandet worden sein, so kann dies korrigiert werden, indem eine neue Linie vom Rahmen, um ein Element herum, und zurück zu diesem gezeichnet wird, die nur das besagte Element enthält. Durch diese Aktion wird das Element ein- bzw. ausgeschlossen. Dieselbe Aktion ermöglicht das gleichzeitige Hinzufügen bzw. Herausnehmen mehrerer Elemente. Das einzelne Aufnehmen bzw. Ausschließen eines Elements

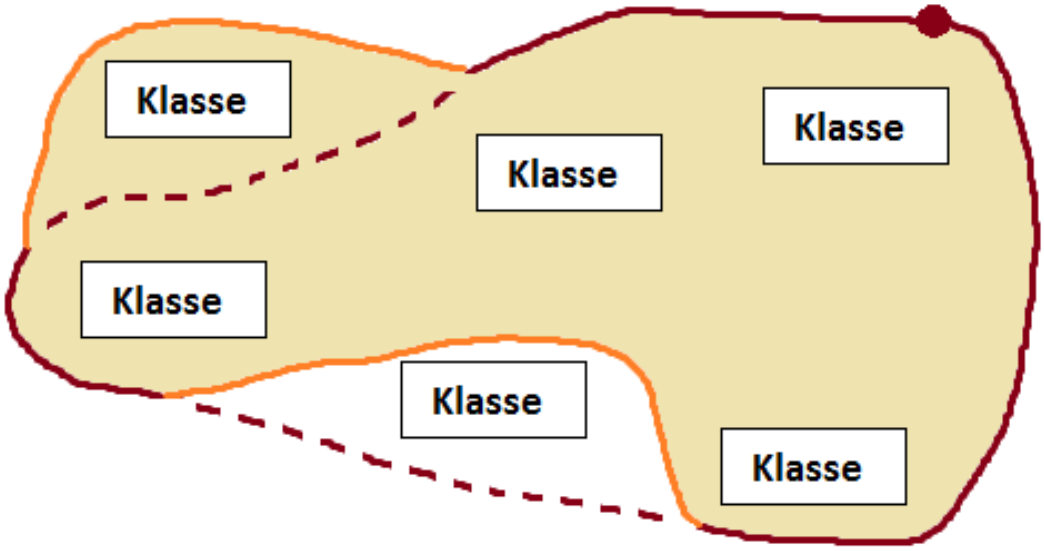


Abbildung 3.2: Markieren von mehreren Elementen

ist deutlich in Abbildung 3.2 zu sehen, wobei die roten Linien zusammen mit den gestrichelten Linien den ursprünglichen Rahmen darstellen und mit den beiden orangen Linien den neuen Rahmen wiedergeben. Durch dieses Konzept muss ein unter Umständen aufwendiger Rahmen nicht erneut gezogen werden, was insofern nicht nur benutzerfreundlich sondern auch zeitsparend ist. Diese Funktion setzt einen gewissen Kommunikationsaufwand voraus, weil die gruppierte Einheit nicht in mehrere Richtungen verschoben werden kann und Tischnutzer sich auf eine Person einigen müssen, die den umrandeten Bereich verschiebt. Im Idealfall können die Elemente während des Verschiebens von weiteren Benutzern bearbeitet werden, da Eingabegeräte, wie die standardmäßige virtuelle Tastatur, nicht direkt an Elemente gekoppelt sind und deswegen nicht mit verschoben werden.

Damit ein Klassendiagramm am MTT nicht, wie an einem Whiteboard, von der Anzeigefläche begrenzt wird, können alle Elemente über den sichtbaren Rand des Tisches hinaus verschoben werden. Dies gilt sowohl für einzelne Elemente, eine

umrandete Einheit als auch für sämtliche Elemente des Diagramms. Für Letzteres kann die gesamte angezeigte Fläche, und damit zusammenhängend das gesamte Klassendiagramm, durch beispielsweise ein Tangible oder einen speziellen Modus zum Verschieben des Diagramms, beliebig weit in jede Richtung verschoben werden. Damit haben Arbeitsgruppen wie auch Einzelnutzer beliebig viel Platz zum Modellieren ihrer Klassendiagramme, wenn auch der sichtbare Ausschnitt ihres Diagramms auf die Tischgröße begrenzt ist. Im Collabee-Projekt wurde diese Notwendigkeit ebenfalls erkannt. Deswegen sollte in ihrem Prototyp eine unbegrenzte Diagrammfläche umgesetzt werden, womit eines der, ihrer Aussage nach, größten Probleme von Medien wie einer Tafel oder ein Whiteboard, die nur einen begrenzten Platz zum Modellieren und keine Funktionalität zum Skalieren bzw. Zoomen anbieten, gelöst wird.[17]

3.2.5 Verschieben von Attributen und Methoden

Gerade in Gruppengesprächen beschäftigen sich die Teammitglieder mit der optimalen Struktur ihres Klassendiagramms. Dabei werden während der Gespräche Attribute und Methoden anderen Klassen zugeordnet bzw. Klassen aufgeteilt oder zusammengelegt. Um die betroffenen Attribute und Methoden nicht umständlich kopieren zu müssen, sollen sie bequem per Drag & Drop von einer Klasse in eine Andere verschoben werden können. Um den Benutzern weiter entgegen zukommen, sollen die Elemente, wenn sie in einer neuen Klasse abgelegt werden, ganz gleich an welcher Stelle, sich selbstständig im Attribut- bzw. Methoden-Bereich eingliedern. Wird ein Attribut zwischen zwei andere Attribute gezogen soll es zwischen diesen eingegliedert werden. Dieses gilt ebenfalls für Methoden.

Zusätzlich sollen mit diesem Verhalten Attribute und Methoden, innerhalb ihres Bereiches der Klasse, neu angeordnet werden können. Entweder ordnet ein Benutzer diese manuell oder lässt sie automatisch, beispielsweise alphabetisch oder der Sichtbarkeit der Attribute und Methoden nach, sortieren.

3.2.6 Oberklasse erstellen und Klassen vereinen

Eine Vererbungshierarchie entwickelt sich im Klassendiagramm aus Klassen, die gemeinsame Eigenschaften oder Funktionalitäten haben. Dabei fällt dies meist erst während der Entwicklung auf, wenn bereits Klassen mit gemeinsamen Eigenschaften erstellt wurden. Darauffolgend wird eine neue Klasse erstellt, in der gleiche Attribute, Methoden und Assoziationen erstellt und dadurch gesammelt werden. Zudem müssen diese Gemeinsamkeiten aus den Unterklassen entfernt sowie Vererbungs-Assoziationen zwischen der neuerstellten Oberklasse und den Unterklassen gebildet werden. Um diese Arbeitsschritte zu optimieren, soll dieses Konzept die beschriebenen Arbeitswege zur Erstellung einer Oberklasse automatisieren und verbessern.

[17] Vgl. Totolici, Alex et al.: Collabee – Multi-touch Collaborative Diagramming, a. a. O., Meilenstein 2 S. 5.

Dazu sollen im ersten Schritt alle Klassen markiert werden, deren Gemeinsamkeiten in eine Oberklasse ausgelagert werden sollen. Das Markieren könnte dabei durch eine Touch-Geste geschehen, wie zum Beispiel mit einem Drei-Finger-Tap. Negativ fällt bei dieser *einfachen* gestengesteuerten Lösung auf, dass nur jeweils *eine* Oberklasse während des Markiervorgangs gebildet werden kann, da nicht zwischen verschiedenen Markierungen unterschieden werden kann. Sollen zu einem Zeitpunkt zwei unterschiedliche Oberklassen durch das Markieren von Elementen erstellt werden, muss dies nacheinander geschehen, dadurch wird das kollaborative Arbeiten erschwert bzw. verlangsamt. Eine bessere Möglichkeit bietet das Markieren durch Tangibles. Mit einem Tangible sollen alle Elemente nacheinander von einem User markiert werden. Dabei könnte jedem Benutzer ein eigenes Tangible zugeordnet werden, sodass jeder Benutzer eigene Markierungen auf Elementen erzeugen kann.

Im nächsten Schritt soll, durch eine benutzerfreundliche Aktion, eine neue Klasse, die als Oberklasse der markierten Elemente dienen soll, auf dem Diagramm erstellt werden. Für eine gestenorientierte Steuerung könnte dies eine Pinch-Geste sein (Siehe Abb. 3.3), die das Zusammenführen aller Elemente verdeutlicht. Bei der Benutzung von Tangibles ist alternativ das Auflegen dieser auf eine freie Stelle vorstellbar, an der die Oberklasse erstellt werden soll, um nicht zwischen Tangibles und Gesten wechseln zu müssen.

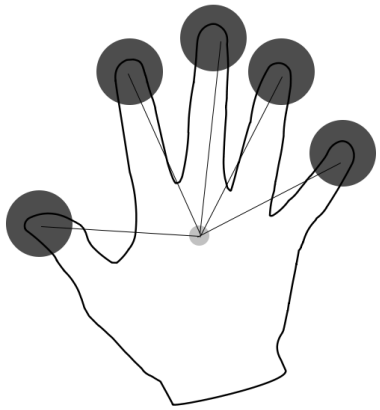


Abbildung 3.3: Bei einem Fünf-Finger-Pinch werden alle aufliegenden Finger einer Hand zu einem Mittelpunkt zusammengezogen

Bei der Erzeugung der Oberklasse sollen alle Attribute, Methoden und Assoziationen von allen Elementen zueinander verglichen werden. Stimmt eine dieser Eigenschaften bzw. Funktionalitäten bei *allen* Klassen überein, so soll diese in die Oberklasse ausgelagert werden. Das beschriebene automatisierte Auslagern beruht bei Attributen und Methoden auf identischen Bezeichnern, bzw. nur mit kleinen Abweichungen durch z.B. Groß- und Kleinschreibung, sowie bei Assoziationen auf gleichen Angaben bei Kardinalitäten, Rollen und Bezeichner. Andernfalls kann

nicht automatisch davon ausgegangen werden, dass beispielsweise zwei Attribute mit Namen „ID“ und „Bezeichner“ dieselbe Eigenschaft ausdrücken wollen. Dabei können bei gleichen Bezeichnern von Attributen bzw. Methoden Konflikte auftreten, wenn diese beispielsweise mit verschiedenen Datentypen definiert sind. Diese Attribute bzw. Methoden sollen nicht ausgelagert werden und erst bei Bedarf durch einen Benutzer in die Oberklasse verschoben werden, wenn diese sich auf eine Lösung geeinigt haben. Angaben, die nicht automatisch ausgelagert wurden, aber dennoch ausgelagert werden sollen, können von dem Benutzer manuell hinzugefügt bzw. bei den Unterklassen gelöscht werden. Dazu können Attribute und Methoden mit der Unterstützung des Konzeptes 3.2.5 „Verschieben von Attributen und Methoden“ direkt von einer Unterklasse in die Oberklasse verschoben werden. Wird das Verschieben von Assoziationsenden (3.2.8 „Assoziationspfad mit Fix- und Dockingpunkten beliebig führen“) unterstützt, so können diese ebenfalls verschoben werden. Der Name der Oberklasse kann von dem Benutzer frei gewählt werden.

Dieses Konzept kann zudem sehr leicht mit einer Klassen-Vereinen-Funktionalität erweitert werden. Dabei sollen Klassen, die redundant erstellt wurden, da zum Beispiel zwei Personen an unterschiedlichen Seiten des MTTs gearbeitet haben und die Arbeit des Anderen nicht vollständig mitverfolgen konnten, zu einer Klasse vereint werden. Diese Funktion soll Eigenschaften und Funktionalitäten aus beiden Klassen zusammenführen. Zum Vereinen sollen zunächst, wie beim Erstellen einer Oberklasse, alle Elemente markiert werden. Anschließend soll durch ein Tangible oder eine Pinch-Geste ein Auswahl-Menü von „Oberklasse erstellen“ und „Klassen vereinen“, statt einer direkten Erstellung einer Oberklasse, geöffnet werden. Dieses Menü könnte weiterhin mit sinnvollen Aktionen, die auf mehreren Elementen gleichzeitig ausgeführt werden sollen, erweitert werden. Bei der Auswahl von „Klassen vereinen“ soll eine neue Klasse an dem Pinch-Ausführungs- bzw. Tangible-Ort abgebildet und alle markierten Klassen gelöscht werden. Die neu erstellte Klasse soll alle Attribute und Methoden der markierten Klassen enthalten. Dabei sollen gleiche Elemente nur einmalig auftauchen. Analog dazu sollen alle Assoziationen übernommen werden. Wobei Assoziationen zwischen markierten Elementen an der vereinten Klasse als reflexive Assoziationen übernommen werden sollen. Auch hier muss der Benutzer ggf. nachhelfen. Insbesondere beim Vereinen von vielen Klassen mit Assoziationen kann die Übersicht nach dem Vereinen zu einer Klasse verloren gehen, da viele und eventuell redundante Assoziationen an der vereinten Klasse schlagartig und ohne einen benutzerdefinierten Assoziationsweg entstehen können. Eine automatische Setzung von Assoziationspfaden, statt direkten Pfaden, könnte verhindern, dass diese Assoziationen andere Klassen überlappen und somit andere Mitglieder behindern. Redundante Assoziationen können dabei von dem Benutzer leicht gelöscht werden. Der Name der vereinten Klasse könnte weiterhin durch eine Auswahl von Klassennamen der markierten Klassen sowie eines textuell frei wählbaren Namens durch den Anwender bestimmt werden.

Das Oberklassen-Erstellen- und Klassen-Vereinen-Konzept soll ein umständli-

ches Kopieren und Löschen von Elementen oder Eigenschaften minimieren, wobei nicht alles automatisiert werden kann.

3.2.7 Erzeugung von Assoziationen und Bearbeitung ihrer Attribute

Assoziationen werden zwischen verschiedensten Elementen gebildet. Dies sind häufig Verbindungen zwischen Klassen, können aber auch zwischen Übergangselementen, wie Aufgabenelementen, oder beiden Elementtypen bestehen. Auch Verknüpfungen zwischen Assoziationen, sogenannte Restriktionen aus der UML, sollen dargestellt werden können. Dabei kann der Assoziationsweg von unterschiedlichen Positionen, am Rand eines Elements, starten und in ebenso unterschiedliche Richtungen verlaufen sowie andere Assoziationen überschneiden. Kardinalitäten und Rollen, sowie Steuerelemente für die Bearbeitung dieser, liegen nah an den verknüpften Klassen. Diese sollten allerdings nicht überlappt werden, da die Klassen für das Verständnis der Verknüpfung wichtig sind und somit weiterhin einsehbar sein sollten. Ebenso können Assoziationen, und der damit verbundene Verwendungsplatz für detaillierte Verbindungsinformationen, verschiedene Längen haben. Dies erschwert die übersichtliche Platzierung der Informationselemente einer Assoziation sowie Steuerelemente für deren Bearbeitung. Ein Konzept für eine mögliche Erstellung und Bearbeitung von Assoziationen, hinsichtlich der genannten Probleme, wird im Folgenden beschrieben.

Zum Erzeugen einer Assoziation soll ein Double-Tap auf dem ersten zu verbindenden Element ausgeführt und gehalten, dann mit dem gehaltenen Touchpoint das erste mit dem zweiten Element verbunden und dort losgelassen werden. Der Weg der dabei mit dem Finger gewählt wird, setzt bereits bei der Erzeugung den Pfad der Assoziation. Da direkte Verbindungswege schnell zu Überlappungen führen, soll der Pfad individuell angepasst werden können. Die Erzeugung und Bearbeitung von sogenannten Docking- bzw. Fixpunkten, die den Pfad der Assoziation beeinflussen, wird im Konzept 3.2.8 „Assoziationspfad mit Fix- und Dockingpunkten beliebig führen“ genauer beschrieben. Anstelle eines Double-Taps ist auch eine andere Geste denkbar. Diese sollte allerdings möglichst nicht mit anderen Interaktionen kollidieren, wie beispielsweise die Geste zum Verschieben von Elementen, die statt eines Double-Taps mit einem einfachen Touchpoint ausgeführt wird, und aufgrund des häufigen Gebrauchs einfach gehalten werden.

Beim Erstellen einer Verbindung werden einige Eigenschaften der Verknüpfung vorgegeben, die jederzeit geändert werden können. Vorgegeben wird ein unspezifizierter Verbindungstyp sowie leere Angaben von Rollen und Kardinalitäten, um den Detailgrad der Assoziation zum Erstellungszeitpunkt gering zu halten. Bei der Erstellung einer Assoziation geht es im ersten Schritt um die Verbindung zwischen Elementen. Erst in späteren Phasen wird die genaue Beziehung zwischen diesen Elementen durch Assoziationsattribute vervollständigt und festgelegt. Der Bezeichner einer Verbindung, inklusive der Leserichtung, ist dabei eine Ausnah-

me. Dieser liefert bereits am Anfang einen Wiedererkennungswert der Assoziation und hält den Grundgedanken der Beziehung fest. Um diesen direkt nach der Erstellung ohne weiteren Aufwand einzutragen, wird der Tastatur-Fokus auf das Eingabefeld des Assoziationsbezeichners gesetzt, um dem Benutzer ein direktes Eingeben des Bezeichners zu ermöglichen. Alle Eigenschaften der Verbindung lassen sich zu jedem beliebigen Zeitpunkt bearbeiten. Die Leserichtung kann durch einen einfachen Touchpoint auf das Symbol der Leserichtung gewechselt werden. Auch der Verbindungstyp kann durch einen einfachen Touchpoint auf ein Assoziationsende und einer darauf folgenden Auswahl geändert werden. Die Auswahl über Verbindungstypen, die jeweils durch Pfeilspitzen repräsentiert werden, wird dabei, ähnlich wie im Video des DTDiagram-Projekts zu sehen ist, als kreisförmige Liste dargestellt.[18] Das in diesem Konzept verwendete Auswahlmenü wird in

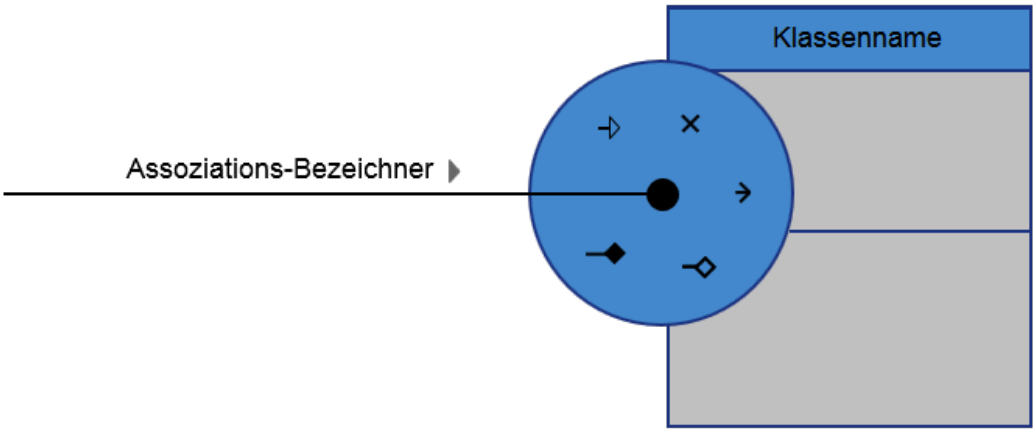


Abbildung 3.4: Kreisauswahl von Verbindungstypen bei Bearbeitung einer Assoziation

Abbildung 3.4 skizzenhaft dargestellt. Dabei kann sich die Auswahl eines Assoziationsendes, wie zum Beispiel bei einer Generalisierung, auf beide Assoziationsenden auswirken sowie auf den Assoziationsbezeichner oder die Kardinalitäten. So entfällt der Bezeichner bei Generalisierungen, Aggregationen und Kompositionen und schränkt die Auswahl des Zahlenbereichs von Kardinalitäten ein. Bei einer Komposition beispielsweise, wird eine Teil-Ganzes-Beziehung dargestellt, wobei das „Teil"-Element nicht ohne das „Ganzes"-Element bestehen kann und somit zwangsweise mit genau einem „Ganzes"-Element in Beziehung stehen muss.[19]

Bei einer Erstellung solcher Verbindungstypen entfällt die Angabe des Bezeichners, die in diesem Konzept automatisch nach Erstellung der Assoziation im

[18] Vgl. Selber, Thomas/Pongelli, Stefano/Valente, Giulio: DTDiagram - a Pen and Touch UML Class Diagram editor for DiamondTouch tabletops: Semester project for the Information Systems Lab 2012 at ETHZ (www.ethz.ch). Online im Internet: `http://www.youtube.com/watch?v=hsZOsjf5un4` – Zugriff am 04.01.2013.

[19] Vgl. Rupp, Chris/Queins, Stefan/Zengler, Barbara, a. a. O., S. 147.

Tastatur-Fokus liegen sollte. Daher sollte in diesem Fall direkt zur Auswahl des Verbindungstypen übergegangen werden. DTDiagram lässt nach dem Erzeugen von Assoziationen direkt den Verbindungstypen auswählen, anders als, in diesem Konzept beschrieben, den Assoziationsbezeichner. Dies ermöglicht einen einfacheren Erstellungsweg für Aggregationen und Kompositionen, allerdings werden, unseren Erfahrungen nach, diese verhältnismäßig seltener verwendet als uni- und bidirektionale Verbindungen, deren darauf folgender Bearbeitungsschritt das Setzen eines Bezeichners ist. Zudem steht der Verbindungstyp einer Assoziation in frühen Entwicklungsphasen meist noch nicht eindeutig fest. Daher haben wir uns in diesem Konzept für das direkte Bearbeiten des Bezeichners entschieden, anstelle des Verbindungstypen. Vererbungen werden, nach unseren Erfahrungen aus Softwareprojekten, öfters verwendet als Aggregationen, allerdings werden diese meistens erst erstellt, wenn mindestens zwei Elemente bestehen, die gleiche Eigenschaften vereinen. Dieser Vorgang wird durch eine differenzierte Geste im Konzept 3.2.6 „Oberklasse erstellen und Klassen vereinen“ erläutert.

Zum Bearbeiten bzw. späterem Vervollständigen der Assoziationen sollen auch Rollen und Kardinalitäten durch Textfelder eingetragen bzw. geändert werden können. Kardinalitäten werden durch eine *exakte* Anzahl oder einen *Zahlenbereich* mit einem Minimum und einem Maximum angegeben. Zusätzlich kann für eine beliebige Anzahl das Zeichen „`*`“ eingefügt werden. Daher sollen bei Kardinalitäten jeweils ein Textfeld für das Minimum und Maximum der Beziehung angeboten werden. Zwischen diesen Feldern soll die Zeichenkette „`..`“ angeben werden, die bereits die Schreibweise der Zahlenbereiche von Kardinalitäten in UML-Notation vorgibt und Benutzern so eine schnellere Interpretation der Textfelder ermöglicht. Um Platz für Bezeichner der Eingabefelder zu sparen, können graue Platzhalter mit Inhalt „Min“ bzw. „Max“ in den entsprechenden Textfeldern platziert werden. Diese werden bei einem Fokus automatisch gelöscht und nur bei leeren Feldern ohne Fokus erneut angezeigt. Exakte Beziehungsangaben lassen sich durch äquivalente Eingaben des Minimums und Maximums einstellen. Damit wird auf separate Steuerelemente für die exakte Beziehungsangabe bewusst verzichtet, um weiteren Verwendungsplatz zu minimieren. Ist das „Max“-Feld leer oder fällt dieses unter den Wert der Minimumangabe, passt sich dieses automatisch dem Minimum an. Exakte Beziehungen können dadurch leichter eingetragen werden, indem nur das Minimumfeld benutzt wird. Wenn eine Kardinalität nicht in Bearbeitung ist, soll statt Eingabefeldern nur eine textuelle Beschreibung der Kardinalität angezeigt werden, wodurch eine bessere Übersicht gewährleistet ist. Bei exakter Angabe von Kardinalitäten soll auf die kurze UML-Schreibweise zurückgegriffen und auf den zusätzlichen Bezeichner „`..Max`“ verzichtet werden. Durch einen einfachen Touchpoint auf die textuelle Beschreibung werden Bearbeitungsmöglichkeiten wiedergegeben. Da Fehler durch Benutzereingaben reduziert werden sollen, soll es nur möglich sein ganze Zahlen und das Zeichen „`*`“ in den vorgegeben Feldern einzutragen. Des Weiteren beschränkt sich die Eingabe durch den Verbindungstypen der Assoziation. Daher soll bei fehlerhafter Eingabe in ein Textfeld der Kardinalitäten, der Eingabewert auf den nächst möglichen Wert zurückgesetzt und ein

entsprechendes Feedback an den Anwender geliefert werden.

Für die Einstellung der Kardinalitäten sind ebenso Slider denkbar. Da Kardinalitäten allerdings von kleinen bis hin zu großen Zahlenbereichen variieren, muss weiterhin zusätzlich ein Eingabefeld angeboten werden, weil nicht sämtliche Eingabemöglichkeiten mit Hilfe eines Sliders übersichtlich festgehalten werden können. Weiterhin bietet es sich an, häufig vorkommende Eingaben am Slider darzustellen. Allerdings zeigen eigene Erfahrungen, dass Kardinalitäten zu stark variieren, sodass für häufige Kardinalitäten nur Null, Eins und „*“ angenommen werden. Ein solcher Slider hätte in den meisten Fällen, hinsichtlich der eingenommenen Anzeigefläche und dem Nutzen, einen negativen Gesamteffekt. Daher soll in diesem Konzept auf die Verwendung von Slidern verzichtet werden.

Assoziationen zwischen Verbindungen, sogenannte Einschränkungen bzw. Restriktionen, sollen analog zu Assoziationen zwischen Klassen- oder Aufgabenelementen erstellt werden. Einschränkungen sind sehr einfache Assoziationen. Sie bestehen nur aus einer gestrichelten Linie und dem Assoziationsbezeichner, der in geschweiften Klammern die Einschränkung, wie beispielsweise „{XOR}“, enthält. Nach dem Erstellen einer Einschränkung wird ebenfalls automatisch der Bezeichner zum Bearbeiten fokussiert, der wiederum jederzeit bearbeitet werden kann.

Ist die Länge einer Assoziation unzureichend, um sämtliche Informationen ohne Überlappungen darzustellen, sollen einige Informationen minimiert bzw. ausgeblendet werden. Überlappen sich Rollen und Kardinalitäten mit einer der verbunden Klassen oder dem Assoziationsbezeichner, so sollen diese ausgeblendet werden. Bei einer Verlängerung der Assoziation durch Bearbeitung des Assoziationspfades oder durch Verschiebung einer der Klassen, sollen diese Angaben bei hinreichendem Anzeigeplatz wieder eingeblendet werden. Dieses Verhalten kann mit einem Herab- und Heraufsetzen von Detailstufen der Assoziation verglichen werden, wie im Konzept 3.1.2 „Anzeige von verschiedenen Detailstufen“ beschrieben. Um Benutzer auf verdeckte Informationen hinzuweisen, sollte dies an der Assoziation symbolisiert werden. Allgemein ist es zu empfehlen, Klassen oder Assoziationswege bei ungenügendem Anzeigeplatz neu anzuordnen, damit bei Bedarf sämtliche Informationen angezeigt werden. Reicht der Platz selbst bei minimierten Rollen und Kardinalitäten nicht für den Assoziationsbezeichner, soll dieser verkürzt, mit einem „...“-Suffix, angezeigt werden. Die Steuerelemente zur Bearbeitung der Assoziation sollen bei fehlendem Platz nicht direkt an der Assoziationslinie dargestellt, sondern in ein Popup-Fenster ausgelagert werden. Mit einem Konnektor soll visuell die Verbindung zur bearbeiteten Assoziation dargestellt werden. Das Popup-Fenster kann somit beliebig verschoben werden, wodurch Assoziation und die zugehörigen Klassen nicht überdeckt werden und im Blickfeld des Nutzers liegen.

3.2.8 Assoziationspfad mit Fix- und Dockingpunkten beliebig führen

Die Verwendung von Docking- und Fixpunkten, die im Folgenden weiter beschrieben werden, soll das Neuplatzieren von vorhandenen Assoziationen sowie das Erstellen und Bearbeiten von Assoziationspfaden erleichtern.

Neuplatzieren von vorhandenen Assoziationen

Im Entwicklungsprozess von Klassendiagrammen kann das Problem auftreten, dass durch Verlegen der Aufgabengebiete von Klassen Assoziationen neu platziert werden müssen. Ein umständlicher Weg stellt das Löschen und Neuerstellen dieser Verbindungen dar. Dabei müssen zudem sämtliche Informationen, wie Bezeichner, Kardinalitäten, Rollen und Leserichtung erneut eingegeben werden. Eine benutzerfreundliche Variante zur Lösung dieser Aufgabe bietet das folgende Konzept.

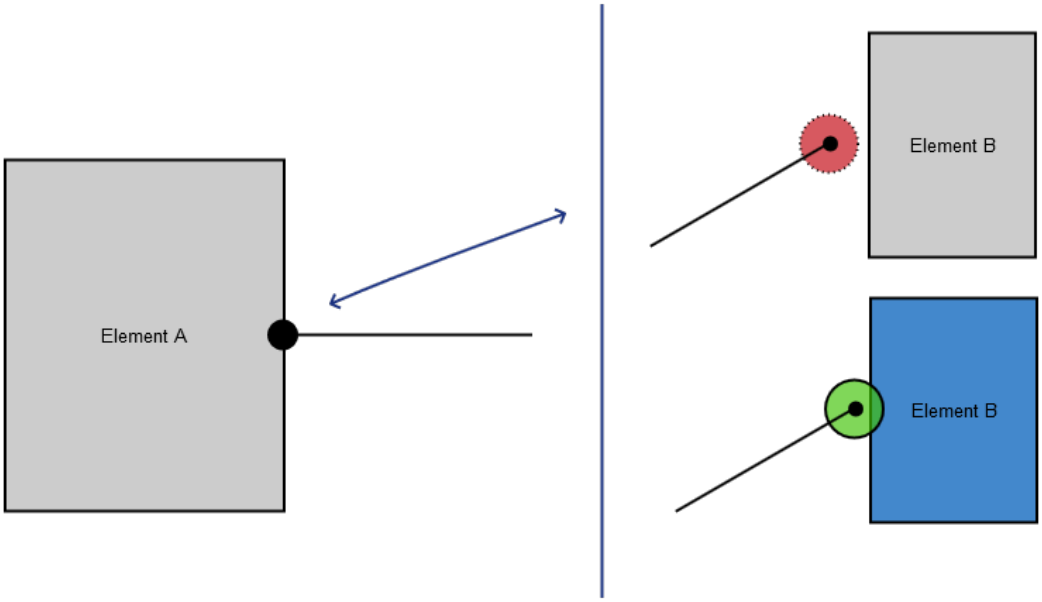


Abbildung 3.5: Neu platzieren einer Assoziation mit visueller Hilfe und Unterstützung der Docking-Ausführung

Für ein benutzerfreundliches Neuplatzieren von Assoziationen sollen „Dockingpunkte“ verwendet werden. Diese Punkte stellen die Verknüpfung zwischen Klassen oder Aufgabenelementen und einer Assoziation dar. Da diese Verbindung ebenfalls für ein Assoziationsende zutrifft und soll im Folgenden ein Assoziationsende zu einem Dockingpunkt erweitert werden. Während der Bearbeitung der Assoziation soll der Dockingpunkt bzw. das Assoziationsende verschoben werden können. Das Verschieben wird durch die Standard-Verschieben-Geste vorgenommen, wobei ein Anknüpfen *innerhalb* von Aufgabenelementen, Klassen und Assoziationen

erfolgreich ist. So können keine Assoziationsenden fehlerhaft gesetzt werden, indem zum Beispiel der Dockingpunkt nicht genau auf das zu verbindende Element gesetzt wurde und so unverbunden neben diesem platziert ist. Um dieses Problem weiterhin auszuschließen, soll beim fehlerhaften Verbinden der Dockingpunkt wieder zur Anfangsposition der Drag-Aktion zurückgeführt werden. Dabei soll dies nicht schlagartig geschehen, sondern durch eine animierte Bewegung mit benutzerorientierter Geschwindigkeit, sodass der Benutzer ein Feedback über den Fehler erhält und dies schnellstmöglich durch erneutes Greifen und Setzen, sogar noch während der Bewegung des Zurücksetzens, beheben kann. Für ein weiteres erleichtertes Setzen des Punktes, soll ein geeigneter Radius um den Docking-Punkt herum gewählt werden, der bereits beim Überschneiden mit Diagrammelementen eine Verbindung zulässt. Dieser Radius sollte nicht zu groß gewählt werden, damit bei dicht platzierten Elementen zwischen diesen differenziert werden kann. Es sollte also keinesfalls die Möglichkeit bestehen mit *einem* Dockingpunkt mehr als ein Element zu verknüpfen. Um das Erkennen des Radius und einer erfolgreichen Verbindung durch einen Drop bereits während der Ausführung der Aktion zu visualisieren, könnte das Element, welches als Verbindungselement erkannt wurde, visuell hervorgehoben werden. Der Radius könnte dabei durch einen Kreis mit einer gestrichelte bzw. durchgezogene Kontur und einer Füllfarbe kenntlich dargestellt werden. Dabei wird eine gestrichelte Kontur während einer möglichen fehlerhaften Platzierung, die durch einen sofortigen Drop erzeugt werden würde, gezeichnet und eine durchgezogene Kontur bei einer möglichen erfolgreichen Platzierung. Das erfolgreiche bzw. fehlerhafte Platzieren kann ebenfalls durch die Füllfarben rot und grün visuell unterstützt werden, wie in Abbildung 3.5 zu sehen ist.

Benutzerfreundliches Erstellen und Bearbeiten von Assoziationspfaden an einem MTT

Ein weiteres Problem, welches durch Fixpunkte bzw. Dockingpunkte gelöst werden soll, ist, dass sich Assoziationen häufig gegenseitig überschneiden, wenn die Assoziationen auf direktem Wege gezeichnet werden. Dadurch wird ein erheblicher Verlust der Übersicht, Orientierung und Informationen verursacht, weil Elemente wie Kardinalitäten, Rollen, Leserichtung und Bezeichner sich ebenfalls mit anderen Texten überlappen und somit ein Entziffern dieser Informationen fast unmöglich machen. Dazu sollen Konzepte zur Platzierung von Assoziationspfaden vorgestellt werden, anstatt den direkten Weg von A nach B zu wählen. Assoziationswege können durch mehrere Verfahren angelegt werden, die jeweils Vor- und Nachteile mit sich bringen, und eine Verwendung dieser sollte nach dem vorliegenden Anwendungsfall entschieden werden. Am Ende dieses Abschnitts soll eine Kombination aus allen Konzepten für Assoziationswege beschrieben werden, welche möglichst alle Vorteile der Anderen vereint und weniger nachteilige Auswirkungen hat.

Als einfachste Möglichkeit können Assoziationswege automatisch und aus *geraden* Wegen, die nur senkrecht oder waagerecht verlaufen, erstellt werden. Ein Beispiel dazu ist in der Abbildung 3.6 unter „A) Automatische Geraden“ zu fin-

den. Dies hat den Vorteil, dass Endnutzern der Arbeitsaufwand abgenommen wird Assoziationswege manuell zu platzieren und eine bessere Übersichtlickkeit als die bei direkten Assoziationswegen geboten wird. Auf direktem Pfad werden die meisten Assoziationen in diagonalen Lagen gelegt, wodurch häufig Überschneidungen von Assoziationen, aber vor allem auch längere Überschneidungen, hingenommen werden müssen. Durch das automatische Anlegen gerader Pfade werden größtenteils Assoziationslinien bei vorhandenen Überschneidungen nur an einem minimalen Teil mit eventuellen Bezeichnern überlappt. Dennoch kann der Fall eintreten, dass zwei Assoziationen einen gleichen Teilassoziationsweg gewählt haben und so sich ein längerer Abschnitt überschneidet. Dieses Problem kann durch den Benutzer, in dieser Möglichkeit, nur durch das Verschieben der verbundenen Elementen behoben werden.

Eine andere Möglichkeit wäre den Assoziationsweg durch sogenannte Fixpunkte, die durch eine Assoziation verlaufen, beliebig vom Benutzer verändern zu lassen, wie durch „B) Docking- und Fixpunkte“ in Abbildung 3.6 dargestellt. Anstatt den direkten Pfad zwischen zwei Elementen zu wählen, können so weitere Punkte auf dieser Geraden generiert und dadurch Umleitungen erstellt werden, um einige Überschneidungen zu vermeiden. Dabei könnte der erstellte Pfad aber willkürlich und über unnötige Umwege vom Benutzer fixiert worden sein, sodass der Pfad schwer nachvollzogen werden kann. Zudem ist der Arbeitsaufwand im Vergleich der obigen vorgestellten Konzepte deutlich höher, wenn auch eine benutzerorientierte Freiheit beim Bearbeiten des Assoziationspfades gegeben ist. Für eine Assoziation muss ein Benutzer einzelne Fixpunkte auf der Assoziationslinie anlegen und anschließend verschieben. Bei längeren und komplexen Assoziationswegen fällt dieser Arbeitsaufwand sehr stark ins Gewicht.

Eine andere, benutzerorientierte, einfache Lösung zum Erstellen eines Assoziationsweges ist das Setzen des Pfades nach dem Verlauf der Gestenführung zum Erstellen einer Assoziation. Dies ist in Abbildung 3.6 unter „C) Freies Zeichnen“ verdeutlicht. Dazu soll der Weg, der während der Ausführung der Erstellungs-Geste von Assoziationen zwischen den zu verbindenden Elementen geführt wird, als Assoziationsweg angelegt werden. Das Resultat ergibt einen benutzerdefinierten Weg, der gleichzeitig ohne großen Aufwand angelegt wird. Soll der Pfad bearbeitet werden, könnte eine Funktion zum erneuten Eingeben bereitgestellt werden. Dazu müsste der Benutzer allerdings einen neuen vollständigen Weg eingeben, auch wenn dieser nur einen kleinen Abschnitt ändern möchte. Zudem kann der durch den Benutzer angelegte Pfad sehr komplex erstellt worden sein.

Werden alle obigen vorgestellten Konzepte zum Anlegen eines Assoziationspfades betrachtet, fällt auf, dass viele Konzepte Eigenschaften unterstützen, die bei anderen negativ ausfielen und andersherum. Das folgende kombinierte Konzept soll die Nachteile durch die Verwendung der anderen Konzepte untereinander ausgleichen. Der Assoziationsweg soll sich zunächst an der Gestenführung bei der Erstellung der Assoziation orientieren, wie in Variante „C) Freies Zeichnen“ erläutert. Dabei soll der Weg der Gestenführung automatisch in Geraden umgewandelt werden, wie in Abbildung 3.6 unter „D) In Kombination“ zu sehen ist.

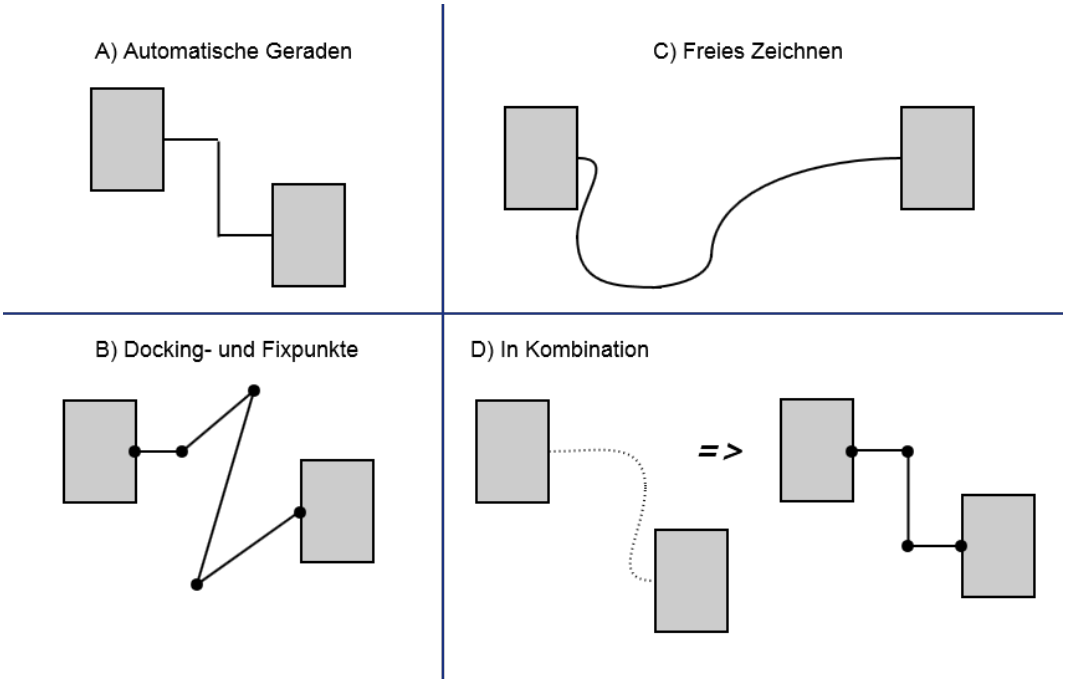


Abbildung 3.6: Assoziationswege in verschiedenen Ausführungen

An den Eckpunkten sollen zusätzlich Fixpunkte erzeugt werden, die vom Benutzer beliebig verschoben werden können. Außerdem können von dem Benutzer weitere Fixpunkte auf einer Assoziationslinie zwischen Docking- bzw. Fixpunkten erstellt werden, die wiederum den Pfad zwischen diesen beeinflusst. Um weiterhin einen geraden Verlauf der Assoziation zu gewährleisten, wird der Pfad automatisch bei Verschiebungen von Fixpunkten gerichtet, wodurch ggf. neue Kanten entstehen, die zusätzliche Fixpunkte erzeugen. Ebenso sollen unnötige Fixpunkte automatisch reduziert werden. Bilden zum Beispiel drei Fixpunkte eine durchgehende *Gerade*, so soll der mittlere entfernt werden, um weiterhin Arbeitsaufwand bei erneutem Verschieben zu reduzieren. Diese Kombination aus allen zuvor vorgestellten Konzepten ermöglicht es Benutzern, einen geeigneten Pfad ohne großen Arbeitsaufwand anzulegen. Dieser Pfad wird automatisch gerichtet und sichert damit eine geordnete Übersicht. Zudem kann der Pfad jederzeit flexibel und beliebig durch den Benutzer verändert werden.

Ebenfalls sollen Assoziationsenden bzw. Dockingpunkte an Elementen benutzerdefiniert verändern werden können, sodass diese sich am Rand der Elemente verschieben lassen. Damit wird es ermöglicht bei beispielsweise zwei Assoziationen, die zwischen den selben Klassen erzeugt worden sind, eine bessere Übersicht zu erreichen. Dies kann durch einen Benutzer erreicht werden, indem beide Assoziationsenden an beiden Klassen so platziert werden, dass diese nicht aufeinander liegen und damit keine Überlappung des Assoziationswegs, der geradlinig von bzw. zu einem Element verläuft, erzeugen. Für mehrere Vererbungsassoziationen einer Oberklasse könnten weiterhin Dockingpunkte an der Oberklasse zu einem vereint

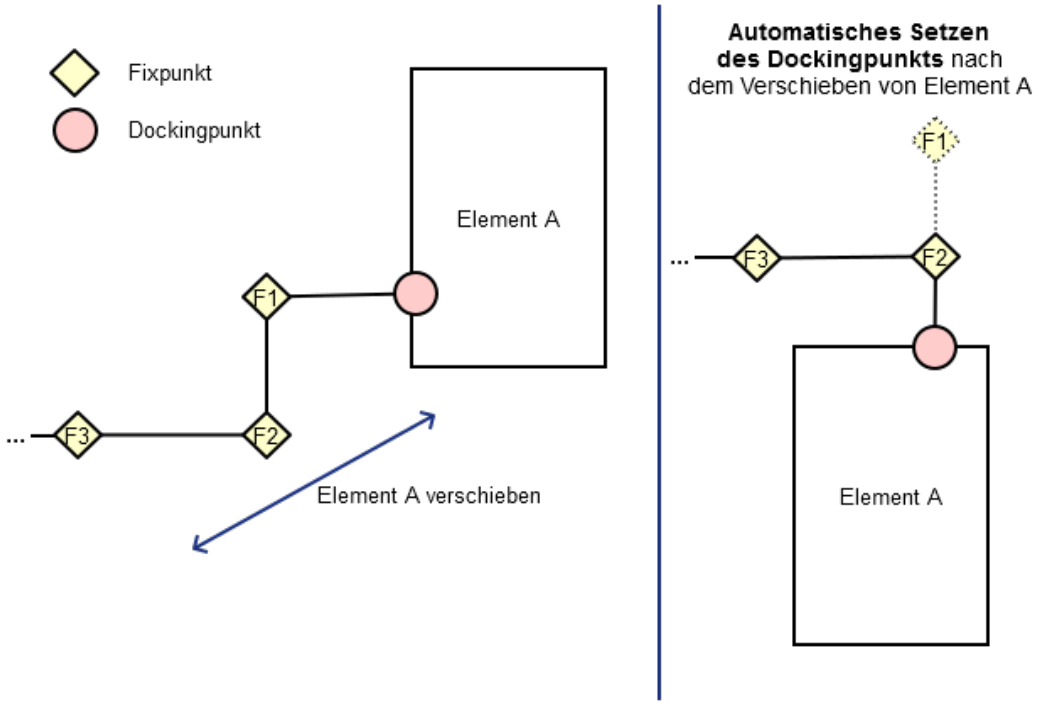


Abbildung 3.7: Automatisches Setzen von Dockingpunkten

werden. Dies entspricht ebenfalls der UML-Notation.[20] Aufgrund der nicht vorhanden Angaben über Kardinalitäten und Rollen an den Assoziationsenden ist dies übersichtlicher, weil es keine Überlappungen von Bezeichnern verursacht und die Anzahl der Vererbungslinien an einer Oberklasse auf eine minimiert. Nachteilig wirkt sich dieses Konzept jedoch beim Verschieben der verbundenen Elementen aus. Nach der benutzerdefinierten Variante können sehr schlecht gewählte Positionen der Dockingpunkte entstehen. Ein Beispiel dazu ist eine Assoziation, die auf den *oberen* Rand eines verbundenen Elements trifft, deren Dockingpunkt aufgrund der vorherigen Positionierung aber am *unteren* Rand des Elements gesetzt ist. Alternativ könnte erneut eine kombinierte Variante gewählt werden, die ein benutzerdefiniertes Setzen der Dockingpunkte erlaubt und beim Verschieben des Elements diese automatisch neu positioniert.

Bei erstellen einer Assoziation wird der Dockingpunkt auf den Schnittpunkt zwischen dem gestengeführten Assoziationspfad und dem Rand des Elements gesetzt. Beim Verschieben dieses Elements wird der Dockingpunkt automatisch neuausgerichtet, indem der Schnittpunkt zwischen dem kürzesten Weg vom Element zum *nähesten* Fixpunkt und dem Rand des Elements die neue Position bestimmt. Fixpunkte und Wege die dadurch abgeschnitten sind, werden entfernt. Dieser Vorgang wird in Abbildung 3.7 verdeutlicht. Reflexive Assoziationen stellen eine Ausnahme dar, diese sollen mit dem Element verschoben werden, d. h. alle Docking- und

[20] Vgl. Rupp, Chris/Queins, Stefan/Zengler, Barbara, a. a. O., S. 128.

Fixpunkte werden relativ zum Element verschoben.

Das automatische Setzen von Dockingpunkten könnte negativ ausfallen, sobald Elemente nur in einem kleinen Radius verschoben werden, wodurch sämtliche vom Benutzer geordnete Positionierungen überschrieben werden. Dieser Nachteil kann Benutzern das Arbeiten sehr erschweren, gerade wenn Elemente nur kurzzeitig verschoben werden, um Arbeitsplatz zu schaffen. Zusätzlich könnten komplexere Konzepte geschaffen werden, die zum Beispiel nur außerhalb eines definierten Verschiebungsradiuses Dockingpunkte automatisch setzen und damit ein Überschreiben oder automatisches Setzen nur durch beispielsweise einer spezielle Geste ausführen. Weiterhin müssen entstehende Nachteile bedacht werden, wie das Überladen von Gesten. Dies soll in diesem Konzeptumfang nicht weiter betrachtet werden. Daher wird zum Konzept der benutzerdefinierten Dockingpunkt-Setzung geraten, wodurch der Benutzer beim Verschieben von Elementen die Dockingpunkte ggf. neu organisieren muss.

Für Restriktionen und n-äre Assoziationen lassen sich diese Konzepte ebenfalls umsetzen.

3.3 Speichern und Laden

Die beiden folgenden Konzepte beschäftigen sich mit dem digitalen Festhalten und Wiederherstellen von Ergebnissen während der Diagrammerstellung, welches bereits als Anforderung im Kapitel 2 „Analyse des Entwicklungsprozesses eines Klassendiagramms“ festgestellt wurde. Insbesondere soll das externe Arbeiten, gegeben durch benutzerspezifische Teildiagramme, ermöglicht werden.

3.3.1 Speichern von Diagrammen

Die Funktionalität erstellte oder bearbeitete Diagramme zu speichern ist von elementarer Bedeutung, um Ergebnisse festzuhalten und auf diesen zu einem späteren Zeitpunkt aufzusetzen. Damit Teammitglieder in externen Arbeitsphasen weiterarbeiten können, wie in 2.3 „Arbeiten in Kleingruppen oder Einzelarbeit“ beschrieben, ist ein einzelnes Gesamtdiagramm für alle Beteiligten nicht ausreichend. Dies liegt einerseits daran, dass alle Beteiligten in diesem Fall das gesamte Diagramm bearbeiten und trotz Absprachen in Meetings Zuständigkeitsprobleme auftreten können. Andererseits können schwerwiegende Konflikte bei der Bearbeitung entstehen, wenn beispielsweise zwei Personen jeweils gleichzeitig an demselben Diagramm arbeiten und es speichern. Ein Algorithmus zum automatischen Zusammenführen der Änderungen und Beheben der Konflikte kann dabei nicht alle Fälle abdecken, daher müssten die Teammitglieder in Besprechungen zusätzlich Änderungen und Konflikte manuell behandeln.

In diesem entwickelten Konzept sollte jedes Teammitglied eine eigene Version des Diagramms bekommen, in der nur Elemente bearbeitet werden können für die das Teammitglied aufgrund des Konzeptes 3.1.4 „Zuweisung von Verantwort-

lichkeiten“ verantwortlich ist, um Zuständigkeiten klar abzugrenzen und damit Teammitglieder sich nicht gegenseitig in einem Diagramm behindern. Weil bei einer Assoziation zwischen Elementen verschiedener Zuständiger beide für diese verantwortlich sind und sie bearbeiten dürfen, soll die Assoziation in den benutzerdefinierten Diagrammen beider Beteiligten manipulierbar bleiben. Diese Assoziationen werden folgend als Grenzassoziationen bezeichnet. Um den Kontext der eigenen Elemente im gesamten Diagramm zu erhalten, sollen Elemente anderer Teammitglieder zwar angezeigt werden, aber nicht manipulierbar sein. Visuell soll dies zum Beispiel durch das bekannte *Ausgrauen* der fremden Elemente unterstützt werden, wie in Abbildung 3.8 zu sehen ist. Ein solches benutzerspezifisches Diagramm, samt der ausgegrauten fremden Elemente, wird folgend als Teildiagramm bezeichnet. Änderungswünsche zu fremden Elementen können während einer ex-

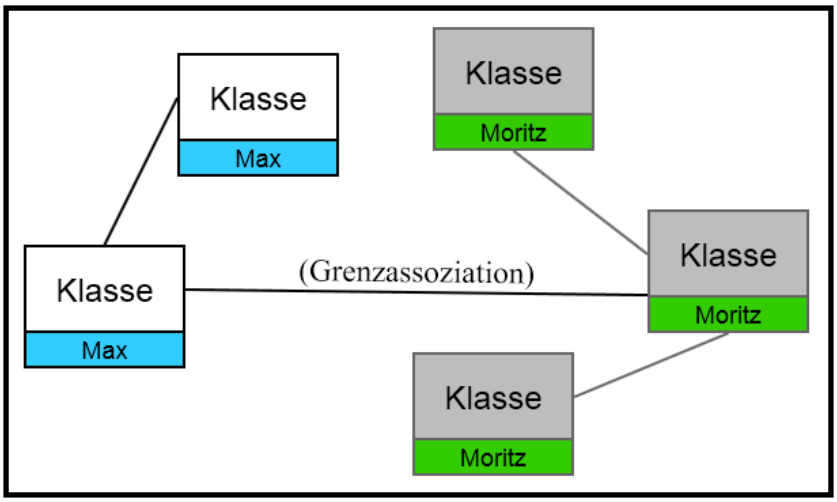


Abbildung 3.8: Teildiagramm von Max mit einer Grenzassoziation zu ausgegrauten Elementen von Moritz

ternen Bearbeitung über verschiedene Kommunikationsmöglichkeiten, beispielsweise per Email, mitgeteilt oder über die im Konzept 3.4.3 „Checkliste“ beschriebene Checkliste erfasst werden. Durch die beschriebene Trennung des Diagramms können, beim Zusammensetzen der bearbeiteten Teildiagramme, Konflikte nur an den Grenzassoziationen entstehen. Sollten aufgrund der Einzelarbeit Konflikte an diesen Grenzassoziationen entstehen, so werden diese erst im anschließenden Treffen am MTT erkannt und können dort direkt aufgelöst werden. Dies wird ausführlich im Konzept 3.3.2 „Laden von Diagrammen“ erläutert. Die Funktionalität zum Speichern der Teildiagramme und des Gesamtdiagramms sollte einfach und komfortabel ausgelöst werden können. Daher soll es möglich sein das Teildiagramm für eine einzelne beteiligte Person zu sichern, aber auch für alle Beteiligten auf einmal, sodass nach dem Besprechungsende nicht jedes Teammitglied sein Teildiagramm selbst sichern muss. Zusätzlich zum Letzteren soll das Gesamtdiagramm automatisch gespeichert werden. Diese Datei dient dem Teamleiter als Backup, falls

ein Teildiagramm verloren geht oder beschädigt wird, und kann jederzeit manuell erstellt werden. Alle Elemente ohne Verantwortlichkeit werden zu einem Teildiagramm zusammengefasst sowie automatisch unter dem Benutzer *Anonym* o. Ä. gesichert. Alternativ könnten nicht zugewiesene Elemente in allen Teildiagrammen bearbeitbar sein, um jedem Mitglied zu ermöglichen an diesen zu arbeiten. Jedoch bleiben durch die Auslagerung in eine separate Datei einerseits die Teildiagramme übersichtlich, weil nur die eigenen Elemente hervorgehoben werden, andererseits werden nicht zugewiesene Aufgaben bzw. Elemente erst vergeben und bearbeitet nachdem dies mit dem Teamleiter abgesprochen wurde. Dadurch kann es nicht passieren, dass mehrere Mitglieder dasselbe Element ohne Absprache bearbeiten.

Standardmäßig werden diese Dateien im Dateisystem des MTTs gespeichert und können dort von den Beteiligten entnommen werden. In Verbindung mit dem Konzept 3.1.3 „Anmeldung am Multi-Touch-Tisch“ könnte in einem Nutzerprofil hinterlegt werden, wo das eigene Teildiagramm abgelegt bzw. wohin dieses automatisch übermittelt werden soll. Dazu ließen sich, abhängig vom jeweiligen Teammitglied, traditionelle Speichermedien, wie ein USB-Stick, eine SD-Karte oder ein Netzlaufwerk, sowie Versionsverwaltungssysteme und Übertragungen per Email nutzen. Wird ein Smartphone zur Anmeldung verwendet, könnte dies gleichzeitig als Speichermedium fungieren. Als Speicherformat für UML-Klassendiagramme wird von COSMOS *XML Metadata Interchange*(XMI) vorgeschlagen, weil viele bekannte UML-Editoren für den PC oder im Web mit diesem Standard umgehen können.[21]

3.3.2 Laden von Diagrammen

Dieses Konzept ist aus der Notwendigkeit entstanden gespeicherte Diagramme auf einem Multi-Touch-Tisch zu laden, damit ein Team seine Arbeit kollaborativ fortsetzen kann und baut auf dem Konzept 3.3.1 „Speichern von Diagrammen“ auf. Um Teamarbeiten fortzusetzen, treffen sich die Teammitglieder nach ihrer eigenständigen Arbeit entweder als Gesamtgruppe oder als Kleingruppe am MTT und wollen ihre Ergebnisse zusammentragen. Nach diesem Konzept sollen die Mitglieder *nacheinander* ihre Teildiagramme auf dem Tisch laden. Alternativ können alle Teildiagramme auf einmal durch die Anwendung geladen werden, wenn diese gesammelt an einem Ort im Dateisystem abgelegt sind. Eine weitere Alternative in Verbindung mit dem Konzept 3.1.3 „Anmeldung am Multi-Touch-Tisch“ sieht vor, dass nach der Anmeldung aller anwesenden Mitglieder nur ein Teammitglied sein Teildiagramm lädt und die Teildiagramme der restlichen Anwesenden automatisch dazu geladen werden. Die dadurch notwendigen Angaben bzw. Speicherpfade werden im Benutzerprofil hinterlegt, sofern die Mitarbeiter ein Teildiagramm des

[21] Vgl. Löffelholz, Daniel et al.: COSMOS: Multi-touch support for collaboration in user-centered projects. In Heiss, Hans-Ulrich (Hrsg.): Informatik 2011. Bonn: Ges. für Informatik. Online im Internet: `http://www.user.tu-berlin.de/komm/CD/paper/061521.pdf` – Zugriff am 04.01.2013, ISBN 978–3–88579–286–4, [S. 10].

zugehörigen Diagramms besitzen. Dabei wird das erste geladene Teildiagramm wie bei der Einzelarbeit dargestellt. Das heißt, dass nur die Elemente des Besitzers des Teildiagramms und Grenzassoziationen editierbar sind, während alle anderen Elemente nicht bearbeitet werden können und ausgegraut sind. Beim Laden jedes weiteren Teildiagramms werden alle Elemente des jeweiligen Besitzers und deren Veränderungen automatisch im angezeigten Diagramm aktualisiert und zum Bearbeiten freigegeben. Anschließend werden diese automatisch ausgewählt und können von einer beliebigen Person am MTT gruppiert, wie nach dem Konzept 3.2.4 „Verschieben von gruppierten Elementen“, an eine freie Stelle im Diagramm verschoben werden. Dabei werden die Grenzassoziationen entsprechend angepasst. Sollten nicht alle Teammitglieder anwesend sein, bleiben die Elemente der Abwesenden weiterhin nicht editierbar bzw. ausgegraut. Diese können ggf. trotzdem aktualisiert werden, wenn Anwesende aktuellere Versionen ausgegrauter Elemente von stattgefundenen Arbeiten in Kleingruppen mitbringen, an denen derzeit abwesende Teammitglieder beteiligt waren. Dabei wird für jedes Teildiagramm, mit einem abwesenden Verantwortlichen, überprüft, welcher Anwesende die aktuellste Version des jeweiligen Teildiagramms besitzt, um es mit diesem zu aktualisieren. Dadurch werden Fortschritte des gesamten Teams auch bei Treffen von Kleingruppen weitergegeben.

Durch das Weitergeben von Informationen über Dritte kann ein Konflikt beim Neuerstellen und Löschen einer Grenzassoziation auftreten, der an folgendem Beispiel verdeutlicht wird. Während eines Kleingruppentreffens wird eine Grenzassoziation zu einem Element eines abwesenden Teammitglieds erstellt. Ein dabei anwesendes Mitglied, das nicht für die Grenzassoziation verantwortlich ist, teilt diese in einem weiteren Kleingruppentreffen, bei dem ein weiterer Zuständiger dieser Assoziation anwesend ist, durch sein Teildiagramm mit. Bei n-ären Grenzassoziationen kann dies ein weiterer anderer Zuständiger sein. Durch das Mitteilen wird diese Grenzassoziationen bei dem anwesenden Zuständigen angezeigt und zum Teildiagramm hinzugefügt, selbst wenn dieser sie bereits durch andere Meetings erhalten und gelöscht hat. Dieser Konflikt wird kontinuierlich auftreten, solange diese Grenzassoziation nicht während eines Treffens mit allen Zuständigen der Grenzassoziation und dem Dritten gelöscht wird. Eine technische Lösung müsste feststellen, ob die durch Dritte hinzugefügte Grenzassoziation tatsächlich neu ist oder bereits gelöscht wurde. In speziellen Fällen kann es nötig werden die ausgegrauten Elemente abwesender Teammitglieder, auch während eines Gruppentreffens, zu bearbeiten, weswegen eine Funktionalität für den Team- bzw. Projektleiter angeboten wird, mit der er ausgegraute Elemente bearbeiten kann.

Nach dem Laden der Teildiagramme können anwesende Mitarbeiter mit dem Besprechen ihrer Fortschritte beginnen und mögliche Konflikte in den Grenzassoziationen auflösen. Dabei trifft für jede dieser Grenzassoziationen einer der folgenden vier Fälle zu, welche nur für binäre Grenzassoziationen beschrieben werden, jedoch genauso auf n-äre Grenzassoziationen bezogen werden können. Alle Änderungen eines Mitarbeiters innerhalb seines Teildiagramms, wie das Erstellen, Löschen und Bearbeiten von Elementen, mit Ausnahme von Grenzassoziationen, können kei-

ne Konflikte enthalten, da diese Änderungen von keinem Anderen durchgeführt werden können. Daher brauchen diese Fälle folgend nicht weiter beachtet werden.

Fall 1: Grenzassoziation wurde nicht verändert

In diesem Fall wurde die Grenzassoziation nicht verändert und die Elemente, die sie verbindet existieren noch. Diese Situation ist konfliktfrei und bedarf keiner weiteren Beachtung durch die Diagrammersteller.

Fall 2: Grenzassoziation wurde überarbeitet

Wurde eine Grenzassoziation nur von einem Bearbeiter verändert, so werden die Änderungen übernommen und diese Assoziation mit „Klärungsbedarf“ markiert, sofern das Konzept 3.4.1 „Visuelle Marker für Besprechungsfortschritte“ eingesetzt wird. Haben mehrere Bearbeiter Veränderungen an dieser Grenzassoziation vorgenommen, so wird versucht die Änderungen zu vereinen. Dies ist am Ende des Konzepts in „Automatisches Zusammenführen von Änderungen“ beschrieben. Wenn dieses nicht möglich ist, wird die Grenzassoziation mit „Ungeklärt“ markiert, ansonsten mit „Klärungsbedarf“. Sobald die Teammitglieder diese Assoziation besprechen und bearbeiten wollen, erhalten sie die Möglichkeit sich für eine der beiden Veränderungen zu entscheiden oder einen Kompromiss zu finden, indem sie Teile aus beiden Änderungen übernehmen. Anschließend wird die Assoziation als „Erledigt“ markiert und kann weiter bearbeitet werden. Sollte der Konflikt während der Sitzung nicht besprochen werden, so behalten die jeweiligen Beteiligten ihre Version der Grenzassoziation und der Konflikt wird in der nächsten Besprechung erneut angezeigt.

Fall 3: Grenzassoziation wurde gelöscht

Wurde eine Grenzassoziation von einem der Zuständigen während einer Einzelarbeit gelöscht, so wird sie in seinem Teildiagramm nicht mehr angezeigt. Bei einem Treffen mit dem zweiten Zuständigen dieser Verbindung wird automatisch festgestellt, dass die Grenzassoziation von einem Beteiligten gelöscht wurde, weswegen diese mit „Gelöscht“ markiert wird. Bei der Nutzung des Konzeptes 3.4.1 „Visuelle Marker für Besprechungsfortschritte“ kann dieses um einen „Gelöscht“-Marker ergänzt werden, der automatisch gesetzt wird und Teammitgliedern nicht zur Verfügung steht. Des Weiteren kann es passieren, dass der zweite an der Grenzassoziation beteiligte Mitarbeiter diese zusätzlich während seiner eigenständigen Arbeit verändert hat. In dem Fall wird die Änderung automatisch übernommen *und* die Assoziation markiert. Nur wenn alle an einer Grenzassoziation beteiligten Teammitglieder diese in Einzelarbeit gelöscht haben, wird sie entfernt ohne nochmals angezeigt zu werden. Eine mit „Gelöscht“ markierte Grenzassoziation kann während des Treffens besprochen werden, um ihre Markierung aufzuheben oder die Assoziation wie abgesprochen zu entfernen. Wird sie nicht besprochen,

so behalten die einzelnen Mitglieder ihre Version dieser Grenzassoziation und der Konflikt wird bei der nächsten Sitzung erneut angezeigt.

Fall 4: Grenzassoziation wurde hinzugefügt

In dem Fall, dass während der Einzelarbeit eine neue Grenzassoziation erstellt worden ist, wird diese in einer Besprechung mit „Neu“ markiert. Diese Markierung wird jedoch nur angezeigt, wenn der Zuständige, dessen Element mit der Grenzassoziation verbunden wurde, anwesend ist und sein Teildiagramm lädt. Weitere Besprechungsteilnehmer bekommen dies andernfalls als normale Grenzassoziation angezeigt, weil die Verantwortung und Umsetzung dieser Beziehung bei den Verantwortlichen der verbundenen Elemente liegt. Bei Einsatz des Konzeptes 3.4.1 „Visuelle Marker für Besprechungsfortschritte“ kann dieses um einen entsprechenden Marker ergänzt werden, der automatisch gesetzt wird und ansonsten nicht von den Teammitgliedern ausgewählt werden kann. Dabei sind zwei Situationen zu unterscheiden. In der Ersten ist das Element, welches mit der neuen Grenzassoziation verbunden ist, beim Zusammenfügen vorhanden. In der zweiten Situation wurde dieses von seinem Bearbeiter gelöscht bzw. geteilt, sodass dieses nicht mehr existiert. Damit die Assoziation nicht ins Leere zeigt, wird ein Aufgabenelement, wie im Konzept 3.2.1 „Aufgaben und Problemstellungen visualisieren“ beschrieben, erstellt und mit der neuen Grenzassoziation verbunden. Dabei trägt das neue Aufgabenelement den Klassennamen, den Assoziationsbezeichner bzw. den Titel des Aufgabenelements, auf das die Grenzassoziation ursprünglich verbunden war. Wiederum gilt, dass der „Neu“-Marker verschwindet, sobald die Grenzassoziation besprochen und akzeptiert wurde. Zum Auflösen des Konflikts kann die Assoziation auf ein anderes Element gezogen, wie im Konzept 3.2.7 „Erzeugung von Assoziationen und Bearbeitung ihrer Attribute“ beschrieben, und anschließend das automatisch erstellte Aufgabenelement gelöscht werden. Sollte die Assoziation nicht besprochen werden, wird der Konflikt weiterhin dargestellt.

Automatisches Zusammenführen von Änderungen

Sollten mehrere Teammitglieder in Einzelarbeit oder in Kleingruppen dieselbe Grenzassoziation verändert haben, so entsteht ein Konflikt. Um den Teammitgliedern eine Konfliktlösung bei unabhängigen Änderungen zu ersparen, sollen diese automatisch zusammengeführt werden. Dabei werden die veränderten Informationen von Grenzassoziationen, wie beispielsweise Bezeichner, Rollen, Kardinalitäten und verknüpfte Elemente, akzeptiert, solange nicht dieselbe Information durch mehrere Bearbeiter verändert wurde. Wurden mehrere Veränderungen an einer Information vorgenommen, ist eine Konfliktlösung durch die zuständigen Bearbeiter notwendig. Dabei wird die Assoziation mit den beteiligten Diagrammelementen grafisch als UML-Element in einer gesonderten Ansicht dargestellt, beispielsweise in einem Popup mit einer Liste der verschiedenen Versionen der Grenzassoziation, sodass die Änderungen der verschiedenen Benutzer unterscheidbar sind und aus-

gewählt werden können. Als weitere Auswahlmöglichkeit zur Konfliktlösung wird eine Grenzassoziation mit den Informationen aufgeführt, die bei allen Beteiligten gleich waren. Die restlichen Informationen können in dieser gemeinsam, und für alle zufriedenstellend, angepasst werden, sofern sich nicht für die Lösung eines Beteiligten entschieden wird.

3.4 Besprechungen

Die nachstehend vorgestellten Konzepte dienen der Unterstützung von Besprechungen am Multi-Touch-Tisch. In diesen sollen vor allem Besprechungen von Arbeitsfortschritten, Formulierungen von Ideen und Lösungen sowie das Planen von weiteren Aufgaben bestärkt werden.

3.4.1 Visuelle Marker für Besprechungsfortschritte

Während der Klassendiagrammentwicklung werden einige Aufgaben deutlicher, umfangreicher und detaillierter besprochen als Andere sowie vorrangige Aufgaben umgesetzt. Dabei erschwert sich der Überblick über den gesamten Bearbeitungsfortschritt sowie die Unterscheidung von bearbeiteten, besprochenen und noch anfallenden Aufgaben. Des Weiteren werden häufig verbundene Probleme bzgl. einer Aufgabe oder Umsetzung erst während einer Besprechung deutlich, zu deren Bewältigung auf Anhieb kein Lösungsvorschlag gegeben ist. Um einen Kontrollverlust zu vermeiden, sollen sogenannte *Marker*, die in diesem Konzept entworfen werden, zum Einsatz kommen. Marker sollen farblich und in Form eines Symbols den Besprechungs- bzw. Entwicklungsstand von Aufgabenelementen, Klassen, Attributen, Methoden und weiteren im Diagramm präsenten Informationen widerspiegeln. Der Zustand wird hierbei unterteilt in die Kategorien „Vollständig besprochen/ Erledigt“, „Teilweise besprochen/ Klärungsbedarf“ und „Nicht besprochen/ Ungeklärt“. Für die farbliche Hervorhebung können Signalfarben genutzt werden, sodass ein unterbewusster Erkennungswert übertragen wird. Ein ähnlicher Effekt lässt sich ebenfalls durch Symbole erwirken. Die farblichen Symbole der Marker sollen folgendermaßen mit den Kategorien des Entwicklungsstandes in Verbindung gebracht werden:

- ein grüner Haken steht für „Vollständig besprochen/ Erledigt“,
- ein blaues Fragezeichen für „Teilweise besprochen/ Klärungsbedarf“ und
- ein rotes Kreuz stellvertretend für den Entwicklungsstand „Nicht besprochen/ Ungeklärt“.

Auf diese Art und Weise wird der Fortschritt der Bearbeitung für alle Teammitglieder ersichtlich und verfolgbar. Offene Fragen und Probleme bzgl. der Modellierung des Diagramms, die in einem Teammeeting nicht geklärt werden konnten, können

durch das gezielte Markieren von Diagramminformationen mit diesen Markern gekennzeichnet werden und bleiben somit im Blickfeld. Dabei übernehmen Marker eine Zweckaufgabe, vergleichbar mit dem eines Lesezeichens.

Weiterhin werden Aufgaben selbständig in Kleingruppen, die außerhalb der Teammeetings stattfinden, erarbeitet und besprochen. Dabei kann eine Aufgabe in einer Kleingruppenbesprechung als „Erledigt“ markiert worden sein, die für das gesamte Team allerdings noch ungeklärt ist, da zunächst in einem Teammeeting mit allen Mitentwicklern diese Fortschritte mitgeteilt werden müssen. Der Einsatz von Markern soll auch in Kleingruppen Hilfestellungen für Besprechungen liefern, jedoch unterscheidbar zu Markern sein, die in einer Teambesprechung gesetzt worden sind.

Um den Erkennungsaufwand zwischen Kleingruppen-Markern und Team-Markern für Benutzer möglichst gering zu halten, soll die Anzahl der verwendeten Symbole und Farben minimiert werden. Daher sollen Team-Marker sich durch eine kontrastbetonte Kontur abheben, wohingegen Kleingruppen-Marker durch eine ausgegraute Kontur dargestellt werden. Zudem kann es wünschenswert sein, dass Kleingruppen selbständig arbeiten und intern besprochene Elemente nicht erneut im gesamten Team geklärt werden müssen. So können Kleingruppen-Marker mit dieser Kontur leichter ignoriert werden als bei einer Verwendung dieser mit verschiedenen Formen oder grellen Farben. Wird ein Kleingruppen-Marker eines Elements innerhalb einer Besprechung des gesamten Teams geändert, so wechselt dieser automatisch die Kontur und wird damit zu einem Team-Marker, andernfalls bleibt die Kontur erhalten und der Marker bleibt ein Kleingruppen-Marker. Analog erfolgt das Überschreiben von Team-Markern in Kleingruppentreffen. Bei Konflikten, die beispielsweise durch verschiedene Markierung von unterschiedlichen Kleingruppen auftreten können, soll der Marker mit der höchsten Priorität übernommen werden. Die Priorität der Marker gilt dabei von „Nicht besprochen/ Ungeklärt“ als höchste Priorität, „Vollständig besprochen/ Erledigt“ als mittlere Priorität und „Teilweise besprochen/ Klärungsbedarf“ als niedrigste Priorität.

Ein Marker soll direkt neben Diagramminformationen mit Bearbeitungsaufwand, wie Aufgabenelemente, Klassen, Assoziationen, Attribute und Methoden, den Klassennamen sowie Kardinalitäten, Rollen, dem Bezeichner als auch der Leserichtung von Assoziationen, geheftet werden. Wird ein Element erstellt oder bearbeitet, wird der Marker auf den Vorgabewert „Ungeklärt“ gesetzt, wobei Marker nur innerhalb von Kleingruppen, bestehend aus mindesten zwei Personen, und Teambesprechungen geändert werden können. Da die eingeführten Marker den Besprechungsfortschritt festhalten sollen und nicht zum Markieren von Fragen und Problemen einer einzelnen Person gedacht ist, sollen diese Marker nicht in externen Einzelarbeiten geändert werden können. Zum festhalten von Fragen und Problemen einzelner Personen sollen dagegen Kommentare und Notizen benutzt werden, wie zum Beispiel in dem Konzept 3.4.2 eingeführt. Für externe Einzelarbeiten hat dies zur Folge, dass alle vorgenommenen Änderungen im nächsten Teammeeting besprochen werden müssen, da diese durch „Ungeklärt“ markiert sind. Aber auch innerhalb von Besprechungen des *gesamten* Teams werden Be-

arbeitungen an Elemente automatisch als „Ungeklärt“ markiert, wodurch diese erst durch Einverständnis des Teams als „Besprochen“ markiert werden müssen und somit die Aufmerksamkeit des gesamten Teams bei Bearbeitungen gefördert wird. Die zusätzliche Bearbeitung und Organisation der Marker führt einerseits zu einem weiteren Arbeitsaufwand, sorgt aber andererseits für eine bessere und gestärkte Kommunikation des Teams indem Unklarheiten direkt beseitigt werden.

Des Weiteren kann ein Klassen-Marker eingeführt werden, der den gesamten Status einer Klasse mit sämtlichen Attributen, Methoden und Assoziationen an einer zentralen Stelle zusammenfasst. Das heißt, sollten alle Attribute und Methoden innerhalb einer Klasse und alle Marker von anliegenden Assoziationen als „Erledigt“ markiert worden sein, so gilt dies auch für den Klassen-Marker. Existiert nur *ein* Element in dieser Klasse oder einer anliegenden Assoziation mit einem „Klärungsbedarf“- oder „Ungeklärt“-Marker, so wird der Klassen-Marker auch dementsprechend angezeigt. Wobei der „Ungeklärt“-Marker bei mehreren unterschiedlichen Markern eine höhere Priorität innehat. Der Fortschritt von anliegenden Assoziationen soll mitaufgenommen werden, da diese die Beziehungen einer Klasse symbolisieren und somit Teil der Klassenkonstruktion sind. Für Assoziationen könnte ebenfalls ein Assoziations-Marker analog zum Klassen-Marker erstellt werden, um eine bessere Übersicht über den Entwicklungsstand von Assoziationen unabhängig von Klassen zu erhalten. Allerdings sollten nicht zu viele Marker eingesetzt werden, um einen übersichtlichen Anzeigeplatz beizubehalten. Für die weitere Übersicht bei detaillierten Diagrammen, soll hingegen das Konzept 3.1.2 „Anzeige von verschiedenen Detailstufen“ eingesetzt werden, dabei können verschiedene Informationen ein- und ausgeblendet werden. Besonders zum Besprechen und Bearbeiten von Assoziationen bietet sich dieses Konzept an.

3.4.2 Kommentarfunktion

Teambesprechungen sollen am Multi-Touch-Tisch durchgeführt werden. Dabei soll die Planung des Teamprojekts durch visuelle Elemente, wie Aufgabenelemente nach dem Konzept 3.2.1 „Aufgaben und Problemstellungen visualisieren“, unterstützt werden. Dies beinhaltet unter anderem die Erstellung und Zuweisung von Aufgaben, die im späterem Verlauf des Projekts von einem oder mehreren Teammitgliedern eigenständig an anderen digitalen Medien bearbeitet werden sollen. Die Informationsmasse zum Bearbeiten der Aufgabe ist insbesondere beim Kick-Off-Meeting und frühen Phasen der Projektplanung für den Bearbeiter unübersichtlich und grobgranular formuliert, da das Thema noch zu ungenau besprochen wurde oder Kenntnisse bzgl. Technologie und Aufgabenverständnis fehlen. Auch beginnt der eigentliche Arbeitsprozess der Aufgabe meist nicht direkt nach der Teambesprechung bzw. Aufgabenzuweisung, sondern ist durch einen gewissen Zeitraum getrennt, wodurch Informationen und Gedankengänge verloren gehen können.

Ein, unserer Auffassung nach, hilfreicher Lösungsansatz ist das Hinzufügen von Kommentaren bzw. Notizen direkt an erstellten Aufgabenelementen, Klassen oder

Assoziationen. Dies ermöglicht Benutzern sämtliche Arten von Informationen zum Bearbeiten der zugewiesenen Aufgaben am benötigten Ort zu sichern und jederzeit abzurufen, ohne in protokollierten Aufzeichnungen danach zu suchen. Dabei kann der Bearbeiter im Nachhinein besprochene Ideen, Lösungen und vereinbarte Umsetzungsschritte gedanklich erneut auffrischen. Ein Verlust von brauchbaren Informationen wird somit verringert. Des Weiteren werden Arbeitsabläufe und -Prozesse beschleunigt, weil das redundante Auflösen von Problemen mit bereits besprochenen Lösungsansätzen größtenteils vermieden und ein angesprochener Lösungsweg direkt umgesetzt werden kann. Ein weiterer Vorteil dieses Konzepts ist, dass eine Information nicht mehrfach von verschiedenen beteiligten Personen mit unterschiedlicher Auffassung der Aufgabe notiert wird, das bei weiteren Arbeitstreffen zu Streitigkeiten und sogar konfliktreichen Ansichten führen kann. Dadurch, dass eine Information im Idealfall nur einmal geschrieben wird und von Allen gelesen werden kann, müssen alle Beteiligten dieser zustimmen bzw. können diese nach Bedarf erweitern, um Missverständnisse und unterschiedliche Interpretationen durch ungenaue Formulierungen zu vermeiden.

Damit Kommentarfelder nicht dauerhaft den ohnehin schon knappen Anzeigeplatz eines Multi-Touch-Tisches verbrauchen, sollen Benutzer diese Felder separat ein- bzw. ausklappen können. So sind Kommentare jederzeit verfügbar, werden jedoch erst im Bedarfsfall angezeigt.

Eine weitere Anforderung an die Umsetzung eines solchen Konzepts stellt eine schnelle und leichte Eingabe-Möglichkeit dar. Wie nach der Analyse des Entwicklungsprozesses in Kapitel 2.2 „Erste Phase: Aufgabenplanung und Verteilung“ beschrieben, sollte bei dieser Funktionalität der Vorteil des schnellen Schreibens durch Notieren auf einem Blatt Papier erhalten bleiben, da Kommentare und Notizen während der Besprechung aufgezeichnet werden und nicht den Besprechungsverlauf stoppen sollen. Der Umgang mit Touch-Tastaturen ist für viele Benutzer meist ungewohnt. Das fehlende haptische Feedback gegenüber physikalischen Tastaturen ist nur eine Ursache dessen. Daher könnte der MTT beispielsweise durch eine Spracherkennung Eingaben für Kommentare erleichtern. Dies ist allerdings nicht ratsam, da die Kollaboration, beispielsweise bei gleichzeitigen Eingaben von verschiedenen Personen, gestört wird und Besprechungen für die Stimmeingabe unterbrochen werden müssen. Besser wäre die Kombination mit einem (Touch-) Stift. Ein Stift erleichtert ebenfalls die Eingabe, hat aber nicht die Nachteile der Spracherkennung. Weitere Vorteile, die sich durch diese multimodale Eingabe am MTT ergeben, werden im Konzept 3.5.2 „Der (Touch-)Stift als Eingabegerät“ erläutert.

3.4.3 Checkliste

Neben den Kommentaren, die zu jedem Element im Diagramm verfasst werden können und im Konzept 3.4.2 „Kommentarfunktion“ beschrieben werden, soll eine zentrale Checkliste angeboten werden, die der Liste des COSMOS-Frameworks

nachempfunden ist.[22] Im Gegensatz zu den Kommentaren, die sich auf bestimmte Elemente beziehen, und dementsprechend in das zum Element gehörende Kommentarfeld eingetragen werden, ist die Checkliste eine zentrale Stelle für allgemeine Notizen und alle Beteiligten. Neben den Notizen können dort Aufgaben zur Organisation oder Kleinigkeiten bzw. To-dos zusammengetragen werden, sodass diese, wie die Kommentare, nicht extern festgehalten werden müssen. Die einzelnen Einträge können weitere Angaben enthalten, wie beispielsweise eine Deadline, einen Verantwortlichen oder eine Prioritätseinstufung. Eine automatische Sortierung der Liste nach verschiedenen, selbstgewählten Kriterien soll die Übersichtlichkeit sowie den Nutzen der Liste steigern. Damit ein Projektmitarbeiter während der Einzelarbeit keine Punkte löscht, die für einen Anderen von Bedeutung sind, soll es in Einzelarbeit nur möglich sein weitere Punkte hinzuzufügen. Gelöscht werden können die Einträge der Checkliste während der gemeinsamen Besprechungen, im Einverständnis aller Beteiligten.

3.4.4 Der MTT als Präsentationsmedium

Diagramme werden von unterschiedlich großen Gruppen besprochen bzw. vor solchen vorgestellt. Dabei reicht die Gruppenstärke bei einer Vorstellung eines Diagramms von ein paar Entwicklern, die sich über neue Änderungen beraten, bis hin zu einem größeren Publikum, wie einem Projektteam oder Vertretern anderer Abteilungen und Kunden. Löffelholz et al. sind in ihrer Ausarbeitung zu COSMOS der Auffassung, dass Multi-Touch-Tische besser für kollaborative Reviews geeignet sind als bisher typisch genutzte Medien, wie ein PC oder Whiteboard. Diese Auffassung basiert zum einen darauf, dass am MTT eine lockere Atmosphäre mit viel Blickkontakt möglich ist, ohne dass ein einzelnes Teammitglied die alleinige Kontrolle über das Medium besitzt. Zum anderen müssen die Beteiligten, während der bereits angespannten Atmosphäre bei Reviews, weder eng vor einem kleinen Monitor noch direkt vor einer Wand sitzen.[23] Diese Begründung ist, unserer Erkenntnis nach, bei einer kleinen Personenzahl sehr einleuchtend. Haben jedoch nicht alle Beteiligten Platz an derselben Tischseite und müssen sich um den Tisch verteilen, kann es zu Verzögerungen und Problemen kommen, weil das Diagramm oder Teile davon zum Lesen und Bearbeiten gedreht werden müssen. Steigt die Personenzahl weiter, sodass nicht alle Personen einen Platz am Tisch finden, wird eine andere Lösung benötigt.

In diesem Konzept wurde deshalb eine Kombination aus MTT und Beamerpräsentation gewählt. Während eine oder mehrere Personen ein Diagramm am MTT vorstellen, können die restlichen Besprechungsteilnehmer das Diagramm

[22] Vgl. Löffelholz, Daniel et al.: COSMOS: Multi-touch support for collaboration in user-centered projects. In Heiss, Hans-Ulrich (Hrsg.): Informatik 2011. Bonn: Ges. für Informatik. Online im Internet: `http://www.user.tu-berlin.de/komm/CD/paper/061521.pdf` – Zugriff am 04.01.2013, ISBN 978–3–88579–286–4, [S. 11].

[23] Vgl. Löffelholz, Daniel et al., a. a. O., [S. 4].

aus Sicht der Redner an der Leinwand sehen und gleichzeitig den Erläuterungen folgen. Der Aufbau ist an eine Präsentation mit Rednerpult angelehnt und kann skizzenhaft in Abbildung 3.9 betrachtet werden. Eine Präsentation mit einem Red-

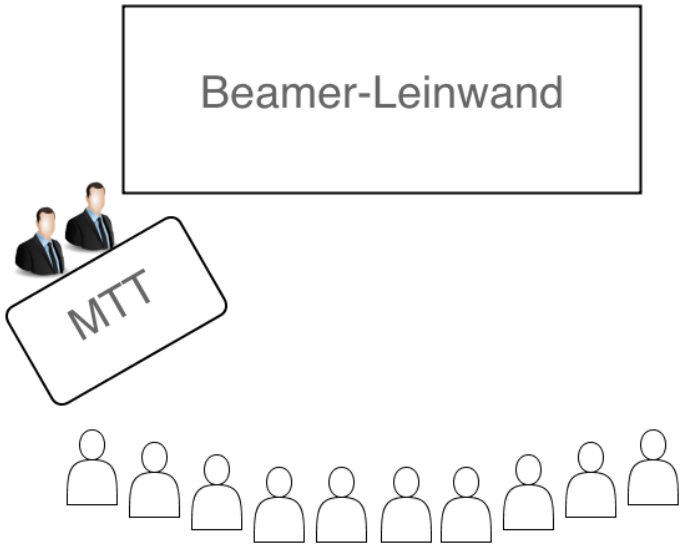


Abbildung 3.9: Aufbau einer Präsentation mit MTT und Beamer

nerpult wird häufig genutzt, um diese vor einem größeren Publikum vorzustellen. Auf die Nutzung eines PCs, um Diagramme zu zeigen und in diesen zu navigieren, wird jedoch allgemein verzichtet. Dies liegt unter anderem daran, dass die Navigation mit Maus und Tastatur zu lange dauert und der Redner mit dem PC beschäftigt ist, anstatt mit seinem Publikum zu reden. Als Ausweg werden deshalb Ausschnitte der Diagramme präsentiert, in denen bereits zu den Problemzonen navigiert wurde. Außerdem werden mehrere Ausschnitte in verschiedenen Detailstufen für die Besprechung erzeugt. Mit einem MTT kann ein Redner schnell, intuitiv und präzise, vor allem durch Gesten, in Diagrammen navigieren und hinter dem Tisch, stehend sein Publikum hervorragend adressieren. Eine Aufbereitung des Diagramms in verschiedene Problemzonen und Detailstufen sowie die Erstellung einer Präsentation ist nach Löffelholz et al. überflüssig und spart Vorbereitungszeit. Zudem wird die Menge der Artefakte reduziert, da durchgehend an einem einzigen Diagramm gearbeitet bzw. vorgestellt wird.[24]

Das Vorstellen des Diagramms wird, unserem Konzept nach, durch einen Präsentationsmodus unterstützt, welcher optimalerweise durch das Auflegen eines Tangibles aktiviert wird. In diesem Modus können Redner ihren Zuschauern direkt auf dem Tisch etwas verdeutlichen, indem sie den Tisch gestenartig berühren. Auf der Leinwand wird dieses beispielsweise durch einen animierten Laserpointer-Punkt dargestellt, wie es aus anderer Präsentations-Software bereits bekannt ist. Weiterhin wäre es möglich auf der Leinwand einen größeren Ausschnitt des Diagramms, als auf dem MTT zu sehen ist, darzustellen, um einen besseren Überblick

[24] Vgl. Löffelholz, Daniel et al., a. a. O., [S. 3].

zu gewähren. Weil das Diagramm in solchen Fällen in der Regel nicht bearbeitet werden soll, werden im Präsentationsmodus alle Funktionen zum Erstellen, Bearbeiten und Löschen von Diagrammelementen deaktiviert. So können diese nicht versehentlich ausgeführt werden und die Redner können sich auf ihre Präsentation konzentrieren.

Durch eine lockere, halbkreisförmige Sitzordnung wird Blickkontakt zwischen allen Teilnehmern, ähnlich dem am MTT, ermöglicht. Zudem können Teilnehmer an den Tisch herantreten, wenn sie etwas anmerken möchten oder auf ein Problem stoßen. Aufgrund der Verwendung des Multi-Touch-Tisches und eines einzigen Diagramms müssen weder Laptops noch Präsentationen gewechselt werden, wie es bei anderen Besprechungen üblich ist. Dadurch wird ein schnelles Wechseln der Redner ermöglicht, sodass die für verschiedene Diagrammteile Zuständigen ihre Fortschritte und Ergebnisse in einem Diagramm präsentieren können.

3.4.5 Bearbeitungsprotokoll eines Diagramms

Innerhalb einer im Collabee-Projekt durchgeführten Befragung von Softwareentwicklern und Informatikstudenten, die regelmäßig, im Team und überwiegend auf Whiteboards oder ähnlichen Hilfsmitteln UML-Diagramme erstellen, wurde festgestellt, dass vereinzelte Zwischenstände eines Diagramms nicht ausreichend sind für das Verständnis des Diagramms. Die Betroffenen gaben an, dass sie während der Diagrammentwicklung oftmals Probleme hatten in der Vergangenheit getroffene Modellierungsentscheidungen nachzuvollziehen, weil ihnen Gedankengänge und Argumentationen, die zum Zeitpunkt der Entscheidung vorlagen, fehlten.[25] Zudem wird, ihrer Ansicht nach, der Aussagegehalt eines Diagramms erst vollständig von einer Person erfasst, wenn diese den vollständigen Entwicklungsverlauf des Diagramms mit allen Zwischenständen und Entscheidungen mitbekommen hat.

Die im Collabee-Projekt vorgeschlagene Lösung zu diesem Problem ist, sämtliche Veränderungen des Diagramms aufzunehmen und bei Bedarf abzuspielen.[26] Nachteil dieser Lösung ist, unserer Auffassung nach, dass es selbst im Schnelldurchlauf lange dauert die Aufnahmen zu betrachten, weil auch Phasen aufgenommen werden, in denen Nutzer inaktiv sind oder in denen Unwichtiges passiert. Besser wäre es nur wichtige Veränderungen und deren Gründe aufzunehmen. Welche Veränderungen wichtig sind und ihre Anlässe, können jedoch nur die Bearbeiter eines Diagramms bestimmen und somit müsste die Protokollierung dieser manuell erfolgen. Nicht selten werden den Beteiligten wichtige Änderungen und ihre Anlässe erst im Nachhinein klar, weil sie im Moment der Veränderung offensichtlich erscheinen, weswegen von einer Protokollierung abgesehen wird. Weiterhin ist diese Alternative für die Beteiligten sehr aufwendig und es ist zu erwarten, dass

[25] Vgl. Totolici, Alex et al.: Collabee – Multi-touch Collaborative Diagramming. Online im Internet: `http://www.jrejre.net/2010/08/collabee-multi-touch-collaborative-diagramming` – Zugriff am 04.01.2013, Meilenstein 2 S. 5.

[26] Vgl. Totolici, Alex et al., a. a. O.

schnell auf das manuelle Festhalten verzichtet wird. Deswegen sollte das Bearbeitungsprotokoll automatisch angelegt werden, sodass dieses jederzeit vorhanden ist. Die von uns entwickelte Kompromisslösung sieht vor, dass alle Änderungen automatisch mitgeschrieben werden, also auch Unwichtige, wobei der jeweilige Bearbeitungsvorgang und die Inaktivität zwischen Änderungen nicht protokolliert werden. So können die Betrachter beim Durchsehen des Protokolls die Geschwindigkeit vorgeben, um unwichtige Änderungen schnell zu überfliegen und sich auf die Wichtigen zu konzentrieren. Festgehalten wird beispielsweise bei Texteingaben die neue Eingabe und nicht der Eingabevorgang durch den Bearbeiter sowie beim Verschieben nur der Start- und Endpunkt. Bei der Nutzung eines Versionsverwaltungssystems können bei Zwischenständen ebenfalls Notizen angegeben werden, um Veränderungen zu protokollieren. Diese Zwischenstände zeigen jedoch nur Sprünge im Entwicklungsverlauf, weswegen die zuvor beschriebene automatische Protokollierung den Entwicklungsverlauf sehr viel detaillierter beschreibt. Zusätzlich wäre es möglich eine Art Logbuch zu führen, in dem wichtige Begründungen, Gedankengänge, Erkenntnisse und Entscheidungen manuell festgehalten werden. Logeinträge können dabei an die protokollierten Veränderungen gekoppelt bzw. in das Bearbeitungsprotokoll aufgenommen werden, um noch mehr Aufschluss bei einem erneuten Betrachten des Diagramms zu geben. Um Einträge und Veränderungen schneller wiederzufinden oder nur Einträge eines bestimmten Elements bzw. Verantwortlichen zu sehen, sollten mitgeschriebene Ereignisse nach verschiedenen Kriterien gefiltert werden können. Um Protokolleinträge anzuzeigen, wird ein neues Fenster, beispielsweise ein Popup, geöffnet. Weil die Anzeige eines einzelnen Protokolleintrags in Textform für ein Diagramm wenig Sinn macht, sollen alle Änderungen bis zu diesem Eintrag gemeinsam als ein neues Diagramm angezeigt werden. Die Änderung des ausgewählten Protokolleintrags wird dabei deutlich angezeigt. Auf diese Weise entsteht eine Abbildung des Diagramms zum Zeitpunkt des Protokolleintrags, welches folgend als Revisionsdiagramm bezeichnet wird. Die einzelnen Revisionsdiagramme können beispielsweise anhand eines Zeitstempels oder der Reihenfolge der Protokolleinträge unterschieden werden. Damit ein bestimmter Protokolleintrag und das dazugehörige Revisionsdiagramm intuitiver als über eine nicht aussagekräftige Nummer des Eintrags wiedergefunden werden kann, soll ihr ein Bezeichner zugewiesen werden können, über den das Revisionsdiagramm ebenfalls aufgerufen werden kann.

3.4.6 Anzeige von Revisionsdiagrammen und Vergleichen von Diagrammen

Während der Arbeit an einem Diagramm werden unterschiedliche Entwicklungsstände des Diagramms betrachtet. Diese werden anhand von Zwischenständen, wie beispielsweise eines eingesetzten Versionsverwaltungssystems, bzw. Protokolleinträgen, nach dem Konzept 3.4.5 „Bearbeitungsprotokoll eines Diagramms“, ausgewählt und stellen das Diagramm zum Zeitpunkt des Zwischenstandes bzw.

Protokolleintrags dar. Solche Diagramme werden folgend als Revisionsdiagramme bezeichnet und in Teammeetings gewöhnlich aus zwei Gründen betrachtet. Der erste Fall ist der Vergleich zwischen zwei verschiedenen Versionen desselben Diagramms, um Veränderungen und die Struktur zu vergleichen. Dies ist oftmals der Fall, wenn größere Änderungen vorgenommen wurden oder Teammitglieder mit der aktuellen Entwicklung unzufrieden sind und nach besseren Lösungen suchen. Der zweite Fall ist die Suche nach einer spezifischen Information bzw. Veränderung. Dieser kommt vor, wenn ein Teammitglied wissen möchte, wie ein ausgewähltes Element zu einem bestimmten vergangenen Zeitpunkt ausgesehen hat oder wann und von wem es zuletzt verändert wurde. Die Darstellung eines ausgewählten Revisionsdiagramms soll zusätzlich zu dem bereits angezeigten Diagramm und nicht wie gewöhnlich im Splittscreen dargestellt werden. Dies kann beispielsweise mit Hilfe eines Popups realisiert werden, welches wie eine zweite Ebene über dem ersten Diagramm liegt, jedoch nicht das ganze Diagramm verdeckt. Dabei soll gleichzeitig im aktuellen Diagramm gearbeitet und in beiden Diagrammen intuitiv navigiert werden können, ohne eins der beiden Diagramme ein- und ausblenden zu müssen. Dies hat den Vorteil, dass sich das angezeigte Revisionsdiagramm am Ort des Geschehens befindet, also dort wo der Benutzer arbeitet, und der Benutzer zum Vergleichen nicht ständig von einer Tischhälfte zur anderen schauen muss. Zudem kann die Größe der Popups angepasst werden, sodass andere Tisch-Benutzer *ungestört* weiterarbeiten können. Zur Auswahl des gewünschten Revisionsdiagramms sollen drei Möglichkeiten zur Verfügung stehen. Die Erste ist eine direkte Auswahl über die Revisionsnummer, sofern diese bekannt ist. Die zweite Möglichkeit ist eine Ansicht des Protokolls (Abbildung 3.10 Links), in dem eine Revision anhand eines Eintrags gewählt werden kann, wobei standardmäßig nur die manuell gespeicherten Versionen angezeigt werden und keine der automatischen Einträge des Konzepts 3.4.5 „Bearbeitungsprotokoll eines Diagramms“. Die dritte Möglichkeit ist ein schrittweises Navigieren durch die Revisionsdiagramme, die als grafische Darstellung repräsentiert werden (Abbildung 3.10 Rechts), wobei die Navigation mit dem aktuellsten Revisionsdiagramm beginnt. Die Fußleiste dieser Ansicht zeigt den Protokolleintrag der ausgewählten Änderung textuell an. Um die Benutzer bestmöglich zu unterstützen, sollten alle drei Möglichkeiten angeboten werden und miteinander verknüpft sein. In die Revisionsdiagramm-Ansicht gelangen die Nutzer, wenn sie eine Revisionsnummer bzw. -eintrag über die ersten beiden Möglichkeiten auswählen. Dadurch, dass alle drei Möglichkeiten miteinander verbunden sind, soll bei einem Wechsel die Auswahl übernommen werden, damit ein Benutzer den Eintrag nicht erneut auswählen muss. Über eine *Filter*-Einstellung können Kriterien eingestellt werden, nach denen die Einträge im Protokoll gefiltert werden. Damit lassen sich gesuchte Revisionen schneller auffinden oder automatisch protokollierte Veränderungen des Konzepts 3.4.5 „Bearbeitungsprotokoll eines Diagramms“ einblenden.

Für den ersten zu Anfang dieses Konzepts beschriebenen Fall, dem Vergleich zweier unterschiedlicher Stände eines Diagramms, kann das Popup zusätzlich an einen Tischrand gezogen werden, sodass es sich anschließend automatisch über

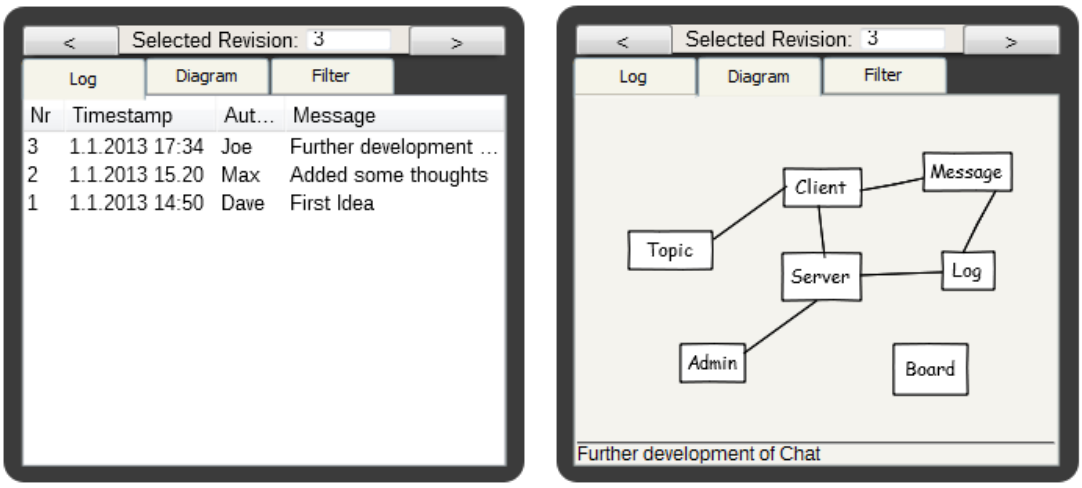


Abbildung 3.10: Skizzen des Revisionsauswahl-Popups: Protokoll-Ansicht(l), Diagramm-Ansicht(r)

eine Tischhälfte legt. Währenddessen wird das darunterliegende Diagramm, an dem entwickelt wird, verkleinert auf der gegenüberliegenden Tischseite angezeigt, womit ein Vergleich im Splittscreen wie gewohnt stattfinden kann. Ist die Anzeigefläche dafür zu klein, kann das Popup auf den ganzen Tisch vergrößert werden. Allerdings muss dann zwischen den beiden Diagrammen umgeschaltet werden.

Für den zweiten Fall, der Suche nach einer spezifischen Information bzw. Veränderung, ist die Anzeige des gesamten Revisionsdiagramms für den Benutzer und der Suche nach nur einem Element nicht sinnvoll, weil das gesuchte Element gesucht, identifiziert und gegebenenfalls zu diesem navigiert werden muss. Zusätzlich angezeigte Informationen bzw. Elemente sind in diesem Fall unwichtig und stören hauptsächlich. Deshalb soll in diesem Fall nur das ausgewählte Element im Popup-Fenster angezeigt werden, damit dieses an Ort und Stelle ohne viel Aufwand verglichen werden kann.

Zusätzlich soll es aus der Diagrammansicht möglich sein Elemente oder Teilelemente, wie beispielsweise Attribute und Methoden, ins aktuelle Diagramm zu ziehen bzw. die aktuellen Versionen mit diesen zu ersetzen. Zudem sollen in dieser Ansicht alle Änderungen optisch hervorgehoben und fokussiert werden, sodass ein Betrachter nicht lange nach diesen suchen muss. Den aktuellen Stand des Diagramms auf ein ausgewähltes Revisionsdiagramms zurückzusetzen, ist zu manchen Entwicklungsständen gewünscht und soll daher ebenso angeboten werden.

Revisionsanzeige auf externen Geräten

Ist ein längeres Durchsuchen des Protokolls von Nöten oder werden Teammitglieder am Tisch durch die Revisionsansicht eines einzelnen gestört, kann die Ansicht auf ein Smartphone oder Tablett ausgelagert werden. Dies erlaubt einem Benutzer sich vom Tisch zu entfernen und eine bequemere Arbeitsposition als über den

Tisch gebeugt einzunehmen. Dadurch haben die Teammitglieder mehr Platz zum Arbeiten am Tisch. Protokolleinträge können am Smartphone betrachtet werden, wobei die Diagrammansicht aufgrund der Displaygröße effizienter auf Tabletts benutzt werden kann.

Vergleichen von unterschiedlichen Diagrammen

Sollen zwei Diagramme verglichen werden, die nicht unterschiedliche Revisionsdiagramme des selben Diagramms sein müssen, so kann dazu die Diagrammansicht der Revisionsanzeige wiederverwendet werden. Dies könnte der Fall sein, wenn völlig unterschiedliche Diagramme, beispielsweise alternative Lösungsansätze für ein Problem, miteinander verglichen und besprochen werden sollen. Dementsprechend ist vor allem der Aufbau bzw. die Strukturierung der Diagramme interessant, weswegen diese direkt im Splittscreen angezeigt werden. Sollen einzelne Details der Diagramme verglichen werden, können die MTT-Benutzer wie gewohnt in beiden Diagrammen zu diesen navigieren und anschließend besprechen.

3.5 Eingabemöglichkeiten

Im Folgenden werden alternative Eingabemöglichkeiten zur standardmäßig eingesetzten, virtuellen Tastatur des Multi-Touch-Tisches vorgestellt. Diese können Eingaben am MTT vereinfachen bzw. intuitiver gestalten und erweitern den Tisch zu einem multimodalen System.

3.5.1 Das Smartphone als Eingabegerät

Damit ein Team gut zusammen arbeiten kann und Fortschritte erzielt, muss auch jedes einzelne Teammitglied effizient arbeiten können. Nach Erfahrung der Nutzer, ist die virtuelle Tastatur als Eingabesystem jedoch nicht optimal für Tabletops. Diese Tastaturen verdecken einen großen Teil des Bildschirms bei einer ununterbrochenen Darstellung und überdecken ggf. Elemente. Zudem bekommt der Benutzer kein haptisches Feedback, sodass dieser nicht erfühlen kann, ob eine Taste erfolgreich gedrückt wurde. Eine virtuelle Tastatur ist, obwohl diese einer PC-Tastatur nachempfunden ist, ein neues Eingabesystem, an das sich Benutzer erst gewöhnen müssen.

Smartphones besitzen ebenfalls eine virtuelle Tastatur. Durch die häufige Nutzung von Smartphones werden diese Tastaturen oft benutzt, daher sind Smartphone-Besitzer an Eingaben mit diesen gewöhnt und erfahrener. Somit sind Smartphones als zusätzliches Eingabegerät am MTT für Smartphone-Besitzer wesentlich intuitiver. Da die in Klassendiagrammen enthaltenen Daten in der Regel nur kurze Texte sind, deren Bearbeitung teilweise durch Eingabemasken unterstützt werden, können Smartphone-Nutzer diese während der Eingabe gut auf ihrem Display lesen. Zusätzliche Vorteile dieses Eingabegeräts sind die meist eingebaute

Rechtschreibprüfung, Autokorrektur und ein persönliches Wörterbuch, welche die Eingabe von langen Texten, wie bei Aufgaben und Kommentaren, erleichtern. Weiterhin unterstützen Smartphones die Barrierefreiheit, beispielsweise mit einer Sprachsteuerung wie im Kapitel 1.5 „Betrachtung des MTTs als multimodales System“ beschrieben. An Multi-Touch-Tischen arbeitende Personen würden somit durch ihre Mobiltelefone ihre eigenen Tastaturen mitbringen und keinen Anzeigeplatz auf dem Tisch verbrauchen. Im Collabee-Projekt wurden Eingabefelder während einer Bearbeitung jeweils mit derselben Hintergrundfarbe versehen, wie die dazugehörende virtuelle Tastatur auf dem MTT, um ihre Zugehörigkeit zueinander optisch zu verdeutlichen.[27] Dies kann ebenfalls bei der Kombination von MTT und Smartphone gut eingesetzt werden, sodass während Eingaben am Smartphone dazugehörige Textfelder auf dem MTT schneller wiedergefunden werden können. Weil Touchpoints bislang nicht ohne Weiteres Personen zugeordnet werden können, ist es problematisch die Smartphone-Eingaben den Elementen und Textfeldern automatisch zuzuordnen. Sobald dieser technische Stand erreicht ist, zum Beispiel über Fingerabdruckerkennung, können Texte automatisch auf das Smartphone, oder andere Geräte, der jeweiligen Person übertragen bzw. Eingaben von diesen entgegengenommen werden. Folgend werden zwei unterschiedliche Ansätze vorgestellt, die diese Zuordnung vorerst manuell erstellen. Dabei unterscheiden sich die Ansätze in ihrer Abhängigkeit zum Konzept 3.1.4 „Zuweisung von Verantwortlichkeiten“.

Zuordnung von Eingabefeldern ohne zugewiesenen Verantwortlichen

Ist einem Element, wie einem Aufgabenelement oder einer Klasse, kein zuständiger Bearbeiter zugewiesen und soll dieses bearbeitet werden, so wird eine virtuelle Tastatur am MTT geöffnet. Zu diesen Elementen gehören ebenfalls Assoziationen, die keinem festen Verantwortlichen zugeordnet und deshalb nur mit diesem Ansatz bearbeitet werden können. Dies ist die einfachste Möglichkeit Texteingaben am Multi-Touch-Tisch zu bearbeiten und wird daher als Standardlösung angeboten. Um eine Bearbeitung von Eingabefeldern oder -masken trotzdem auf einem Smartphone zu ermöglichen, können Tangibles eingesetzt werden. Anstatt diesen Tangibles jedoch eine bestimmte Funktionalität zuzuordnen, wird ihnen ein spezifischer Anwender zugeteilt. So kann ein Benutzer das Tangible, welches ihn identifiziert, auf ein Eingabefeld setzen, damit der Inhalt auf seinem Smartphone angezeigt wird und dort bearbeitet werden kann. Zur Bestätigung der Eingabe wird das Tangible wieder entfernt, damit der Bearbeiter dieses nicht vergisst und so andere in ihren Tätigkeiten blockiert. Soll eine Eingabe ohne Änderung abgebrochen werden, so lässt sich dies, vor dem Entfernen des Tangibles, auf dem Smartphone auswählen. Zu jedem Zeitpunkt kann ein Eingabefeld nur von einer

[27] Vgl. Totolici, Alex et al.: Collabee – Multi-touch Collaborative Diagramming. Online im Internet:
`http://www.jrejre.net/2010/08/collabee-multi-touch-collaborative-diagramming`⟩ – Zugriff am 04.01.2013, Meilenstein 4 S. 2.

Person bearbeitet werden. Während auf einem Element mit mehreren Eingabefeldern durchaus mehrere Personen zugleich arbeiten können.

Zuordnung von Eingabefeldern mit zugewiesenem Verantwortlichen

Ist einem Element ein Verantwortlicher zugewiesen, so ist es offensichtlich besser, wenn Änderungen an dem Element von diesem vorgenommen werden. Daher wird der Inhalt eines ausgewählten Eingabefelds direkt auf dem Smartphone des Verantwortlichen angezeigt und kann dort bearbeitet werden, sofern dieser zur Zeit nicht mit einer anderen Eingabe beschäftigt ist. Solange ein Verantwortlicher mit einer Eingabe beschäftigt ist, wird bei einer Bearbeitungsanfrage auf Eingabefelder seiner Elemente eine virtuelle Tastatur auf dem MTT geöffnet. Ein Benutzer braucht für die Bearbeitung der eigenen Elemente keine Tangibles für die Zuordnung und kann seine Bearbeitung direkt auf seinem Smartphone bestätigen oder ohne Änderung abbrechen. Tangibles zur Zuordnung lassen sich weiterhin wie beschrieben nutzen und werden erwartungsgemäß angewendet, um in fremden Elementen zu arbeiten. Ebenfalls ist davon auszugehen, dass die in diesem Absatz beschriebene Vorgehensweise überwiegend genutzt wird, um die eigenen Elemente weiterzuentwickeln, da die virtuelle Tastatur des Smartphones bei textuellen Eingaben der des MTTs vorgezogen wird und Benutzer den Elementen ohnehin zugeordnet sind.

Auslagerung von Funktionalität auf Smartphones

Eine Weiterentwicklung der Idee, Smartphones als Eingabegeräte für einen MTT zu nutzen, sieht nicht nur die Bearbeitung von Texteingaben vor, sondern auch eine weitere Auslagerung von Funktionalität auf ein Smartphone. So ist es denkbar Aufgabenelemente und Klassen mit Hilfe von Smartphones zu erstellen, während andere Personen, die direkt am Multi-Touch-Tisch stehen und arbeiten, diese anordnen und miteinander verbinden. Problematisch an diesem Ansatz ist, dass keine Position für das neu erstellte Element existiert, weswegen unklar ist, wo das Element auf dem MTT angezeigt werden soll. Eine Möglichkeit wäre diese Diagrammelemente an einem festgelegten Platz oder an der Position eines bestimmten Tangibles auf dem Tisch erstellen zu lassen. Eine andere sähe vor, die Elemente am Tischrand, möglichst in der Nähe des Smartphone-Nutzers, der das jeweilige Element erstellt hat, anzuzeigen.

3.5.2 Der (Touch-)Stift als Eingabegerät

Um aus erster Sicht eine hohe Benutzerfreundlichkeit zu erreichen, sollten zusätzlich zur Touch-Eingabe des Multi-Touch-Tisches alternative Eingabesysteme betrachtet werden. Texteingaben am MTT erfolgen standardmäßig durch eine virtuelle Tastatur. Wie bereits im Kapitel 1.5 „Betrachtung des MTTs als multimodales System“ beschrieben, ist diese Art der Eingabe für längere Texte ungeeignet.

Für Notizen und Kommentare, die während Besprechungen ebenfalls am Tisch eingetragen werden sollen, muss eine einfache und schnelle Eingabe möglich sein. Dazu soll ein (Touch-)Stift mit dem MTT kombiniert werden, sodass ein einfaches handschriftliches Schreiben, wie auf einem Blatt Papier, nachempfunden werden kann. Mit diesem Stift sollen handschriftliche Eingaben in allen Texteingabefeldern möglich sein. Zudem sollen Gesten zum Zeichnen bzw. Malen alternativ mit Hilfe des Stiftes ausgeführt werden können. Damit wird Nutzern ebenfalls ein natürliches Zeichnen ermöglicht, mit dem beispielsweise durch das Malen eines Rechtecks eine Klasse erstellt werden kann. Da diese Art der Eingabe natürlicher wirkt als die einer virtuellen Tastatur, bietet dieses Eingabesystem eine sehr gute Alternative mit der einige Eingaben einfacher und schneller umgesetzt werden können. Bei dem gewohnten Schreiben mit einem Stift auf einem Blatt Papier werden oft die Hand bzw. einige Finger zur Unterstützung der Stiftführung aufgelegt. Dieses Verhalten könnte auf dem MTT zu ungewollten Aktionen führen, indem die aufgelegte Hand als weiterer Touchpoint interpretiert wird. Zudem verdeckt diese Stift-Haltung weiteren Anzeigeplatz, das als störend in der Zusammenarbeit mit anderen Teilnehmern empfunden werden könnte. Im Vergleich zu virtuellen Tastaturen muss bedacht werden, dass diese ebenfalls Anzeigeplatz verdecken. Externe Eingaben über ein Smartphone, wie im Kapitel 3.5.1 „Das Smartphone als Eingabegerät“ beschrieben, haben diesen Nachteil nicht. Die Stifteingabe ist allerdings intuitiver als eine Eingabe durch ein Smartphone, wenn auch der tägliche Gebrauch von Smarthphones den Umgang mit den bereitgestellten Eingabemöglichkeiten des Mobilgeräts fördert. Für den Einsatz von (Touch-)Stiften spricht weiterhin die einfache Handhabung. Der Stift kann nach vollendeten Eingaben leicht zwischen zwei Fingern geklemmt werden, ohne zu stören. Ein Smartphone hingegen muss abgelegt und eine virtuelle Tastatur des MTT geschlossen werden. Ein weiterer Punkt, der für die Eingabekombination mit einem Stift spricht, ist das Unterscheiden von Benutzereingaben. Dies könnte folgendermaßen realisiert werden. Durch verschiedene Erkennungsmerkmale an der Stiftspitze könnten Eingaben einem bestimmten Stift zugeordnet werden. Zur Erkennung wären Marken, die gewöhnlich der Erkennung von Tangibles dienen, verwendbar. Wird jedem Benutzer ein Stift zugewiesen, können Eingaben durch Stifte verschiedener Anwender unterschieden werden. Im Zusammenhang mit Verantwortlichkeiten (Kapitel 3.1.4 „Zuweisung von Verantwortlichkeiten“) kann somit anstelle von undefinierten Verantwortlichkeiten beispielsweise direkt festgehalten werden, wer welches Element erstellt hat und der jeweilige Benutzer kann automatisch als Verantwortlicher eintragen werden.

4 Anforderungsanalyse und Systementwurf

Die Anforderungsanalyse beschäftigt sich mit den Anforderungen an den zu konzipierenden Prototypen sowie der bestehenden Entwicklungsumgebung, wie zum Beispiel dem gegebenen Multi-Touch-Tisch, unter deren dieser Prototyp entwickelt wird. Die Anforderungen werden dabei durch Anwedungsfälle beschrieben.

Der Systementwurf soll zudem notwendige Hardwarevoraussetzung für den Prototyp festlegen und mit Hilfe eines Klassendiagramms einen softwareseitigen Entwurf des Prototypen beschreiben. Dazu gehören einerseits Entscheidungen über die verwendeten Tools, das Betriebssystem und die Programmiersprache sowie die Architektur mit eingesetzten Pattern, Modellen und dem Entwurfs-Klassendiagramm des Prototypen. Des Weiteren wird auf die geplante Gestensteuerung des prototypischen Editors eingegangen.

4.1 Entwicklung eines Prototypen

In dieser Arbeit soll ein Prototyp eines Klassendiagramm-Editors entwickelt werden. Dabei soll der Editor einige Konzepte aus dem Kapitel 3 „Konzepte zur Unterstützung des Entwicklungsprozesses“ umsetzen, die zur Unterstützung des Entwicklungsprozesses von Klassendiagrammen sowie der Kollaboration am MTT einen vergleichsweise hohen Nutzen haben. In einer anschließenden Evaluierung, die im Kapitel 6 „Evaluierung“ beschrieben wird, sollen unter anderem diese implementierten Konzepte durch Benutzertests geprüft werden. Weil der Test ein Entwickeln eines Klassendiagramms am MTT bewerten soll, müssen Konzepte zur Erstellung von UML-Elementen in dem Prototyp umgesetzt sein. Da hauptsächlich die Konzepte aus den Kapiteln: 3.2.1 „Aufgaben und Problemstellungen visualisieren“, 3.4.2 „Kommentarfunktion“, 3.4.1 „Visuelle Marker für Besprechungsfortschritte“ und 3.1.4 „Zuweisung von Verantwortlichkeiten“ zur Unterstützung des Entwicklungsprozesses von Klassendiagrammen sowie der Kollaboration am MTT umgesetzt werden sollen, wird aus Prioritäts- und Zeitgründen die Erstellung und Bearbeitung von UML-Elementen auf eine grundlegende Implementierung minimiert.

4.2 Evaluierung geeigneter Multi-Touch-Frameworks und -Bibliotheken

In diesem Kapitel sollen verschiedene Frameworks zur Entwicklung von Multi-Touch-Anwendungen betrachtet und eine Entscheidung für die Verwendung einer dieser für die Entwicklung des Prototypen gefällt werden.

Die Entwicklung im Multi-Touch-Bereich nimmt immer mehr zu. Schon heute existiert eine Menge von Technologien um den Entwicklungsprozess von Multi-Touch-Anwendungen zu unterstützen und zu erleichtern. Neben Simulationsprogrammen für Touch-Eingaben gibt es eine Vielzahl von verschiedenen Frameworks und Bibliotheken, die bereits wichtige Funktionen implementiert haben und frei zugänglich anbieten. Das Simulieren von Touch-Gesten ermöglicht das Testen von entwickelten Multi-Touch-Programmen auf nicht multi-touch-fähigen Geräten.

4.2.1 Übersicht und Vergleich

Die meisten Frameworks und Bibliotheken sind für einen spezifischen Anwendungsbereich entwickelt worden. So sind beispielsweise nicht alle Frameworks für jede Plattform geeignet. Die Grafik in Abbildung 4.1 zeigt eine Liste der bekanntesten Frameworks und Bibliotheken und vergleicht diese in einer Übersicht. Zusätzliche Schlüsseldaten für die für uns relevanten Frameworks werden in der Tabelle 4.1 dargestellt.

Im Folgenden wird einzeln näher auf die für uns relevanten und geeigneten Frameworks eingegangen und diese durch gegebene Quellen zusammengefasst. Wobei hier die auf Python basierende Alternative Kivy, der Nachfolger von PyMT, untersucht wird.

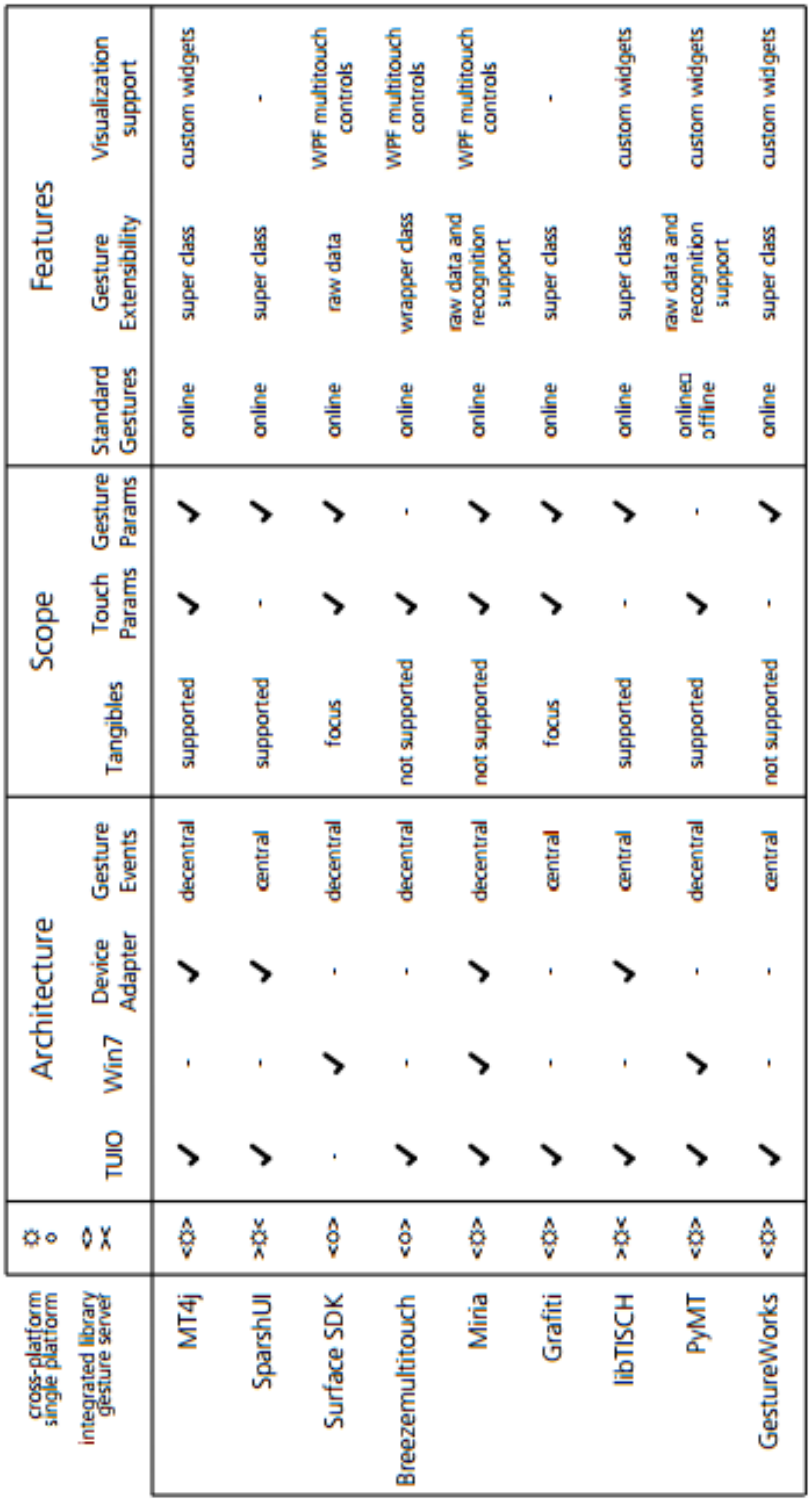
4.2.2 MT4j

Folgende Zusammenfassung des Open-Source-Java-Frameworks *Multitouch for Java* (*MT4j*) bezieht sich auf die Funktionsliste der offiziellen Webseite von MT4j.[2]

Das Framework ist für die schnelle Entwicklung von visuell anspruchsvollen Anwendungen und für die Unterstützung von verschiedenen Eingabegeräten gestaltet, mit einem besonderen Fokus auf den Multi-Touch-Bereich. MT4j ist in Java geschrieben und somit Plattformunabhängig. Die aktuelle Version 0.98 wurde unter Windows 7, Windows XP, Windows Vista, Ubuntu Linux und Mac OSX

[1] Entnommen aus: Kammer, Dietrich et al.: Taxonomy and Overview of Multi-touch Frameworks: Architecture, Scope and Features. Online im Internet: `http://vi-c.de/vic/sites/default/files/Taxonomy_and_Overview_of_Multi-touch_Frameworks_Revised.pdf` – Zugriff am 12.08.2013, [S. 4].

[2] Fraunhofer-Institut für Arbeitswirtschaft: MT4j Webseite. Online im Internet: `http://www.mt4j.org` – Zugriff am 12.08.2013.



cross-platform / single platform; integrated library / gesture server	☼ o <> ><	Architecture: TUIO	Architecture: Win7	Architecture: Device Adapter	Architecture: Gesture Events	Scope: Tangibles	Scope: Touch Params	Scope: Gesture Params	Features: Standard Gestures	Features: Gesture Extensibility	Features: Visualization support
MT4j	<☼>	✓	-	✓	decentral	supported	✓	✓	online	super class	custom widgets
SparshUI	>☼<	✓	-	✓	central	supported	-	✓	online	super class	-
Surface SDK	<o>	-	✓	-	decentral	focus	✓	✓	online	raw data	WPF multitouch controls
Breezemultitouch	<o>	✓	-	-	decentral	not supported	✓	-	online	wrapper class	WPF multitouch controls
Miria	<☼>	✓	✓	✓	decentral	not supported	✓	✓	online	raw data and recognition support	WPF multitouch controls
Grafiti	<☼>	✓	-	-	central	focus	✓	✓	online	super class	-
libTISCH	>☼<	✓	-	✓	central	supported	-	✓	online	super class	custom widgets
PyMT	<☼>	✓	✓	-	decentral	supported	✓	-	online□ offline	raw data and recognition support	custom widgets
GestureWorks	<☼>	✓	-	-	central	not supported	-	✓	online	super class	custom widgets

Abbildung 4.1: Frameworks und Bibliothenken für Multi-touch-Anwendungen im Vergleich.[1]

	MT4j	WPF + Surface SDK	Kivy
Verwendete Programmiersprache	Java	C#, XAML	Python
Plattformen	Windows, Linux, iOS, Android	Surface basierte Systeme, Windows 7	MacOSX, Linux, Windows, iOs, Android, Geräte mit TUIO-Unterstützung
Benötigt	Java 1.5+	-	Python 2.x ($2.6 \leq x < 3.0$)
Grafik-Engine	OpenGL	DirectX 9.0c	OpenGL ES 2
Lizenz	GPL	Microsoft Software License Terms	LGPL 3

Tabelle 4.1: Frameworks und Programmiersprachen für MTTs im Vergleich

getestet. Gesten werden von vielen heterogenen Eingabetypen unterstützt. Windows 7 Touch-Funktionen und kompatible Multi-Touch-Hardware werden ebenso unterstützt wie Apples Multi-Touch-Maus und Trackpads. Das Framework setzt auf das TUIO Protokoll, welches für Finger- und Objekt-Tracking-Software wie reacTIVision, CCV oder Touché vorgesehen ist. Unterstützungen für weitere Arten von Eingabegeräten können durch eine abstrakte Eingabeschicht von MT4j leicht und schnell hinzugefügt werden. Die häufigsten und bekanntesten Multi-Touch-Gesten sind bereits enthalten. Diese können modular mit jeder Komponente verknüpft werden. Auch nicht vorhandene frei kreierte Gesten können effizient über das flexible Multi-Touch-Gestensystem eingepflegt werden.

Etliche Grafikobjekte sind bereits vorhanden und können ohne direktes arbeiten mit OpenGL verwendet werden. Wie zum Beispiel Rechtecke, runde Rechtecke, Ellipsen, Polygone, Linien, Dreiecke, Gitter, Kugeln, Würfel und weitere mehr. Auch Texturen, Farbverläufe, Füll- und Konturfarben können Grafikobjekten leicht hinzugefügt werden. Des Weiteren offeriert MT4j mittels OpenGL Software oder Hardware beschleunigtes Grafikrendering. Um Entwicklern entgegenzukommen werden vorgefertigte UI-Komponenten angeboten. Buttons, Listen, Slider, Labels, Bilder und eine virtuelle Multi-Touch-Tastatur sind nur einige nennenswerte Elemente des vorgegebenen User Interfaces.

Es können 2D, 3D sowie 2.5D (Pseudo-3D) Anwendungen geschrieben werden. Dabei kann auf eine erweiterbare, komponentenorientierte Szenengraph-Struktur zurückgegriffen werden - Identisch zu Javas Swing Framework. Zusätzlich zu gängigen Bildformaten erlaubt MT4j das Verwenden von Scalable Vector Graphics (SVG). Durch das Einsetzen von Vektoren erlaubt SVG ein pixelfreies verkleinern und vergrößern von Grafiken. Schriften können als Bitmap oder vektorisierte

Schrifttypen wie SVG oder True Type Fonts geladen werden. Im grafischen Bereich und insbesondere für 3D Anwendungen lassen sich 3D-Objekte von .3ds und .obj Dateien mit Texturen importieren. Um Grafikobjekten weiche und glatte Schattierungen anzuheften, können des Weiteren Normalen erstellt werden.

In eine MT4j-Anwendungen können Animationen eingebaut werden um Inhalte zu verdeutlichen, bewegliche Hintergründe zu simulieren oder spielerische Effekte wirken zu lassen. MT4j ist hierarchisch über dem Processing aufgebaut und erlaubt somit die Nutzung vieler Funktionen und Bibliotheken von dieser und der sich darunter befindenden Ebenen. Eigene Multi-Touch-Anwendungen können mit Hilfe des Frameworks für Windows und Linux auch ohne vorhandene multi-touchfähige Endgeräte getestet werden, indem eine oder mehrere PC-Mäuse mit dem Rechner verbunden werden. MT4j ist Open-Source und kostenlos unter der GPL Lizenz erhältlich.

4.2.3 WPF/.NET + Surface SDK

Microsoft stellt für das hauseigene Betriebssystem Surface, welches für Hardware mit Touch-Bildschirmen konzipiert wurde, ein eigenes Software Development Kit (SDK) bereit. Mit dem *Surface SDK* können Anwendungen entwickelt werden die auf Geräten mit Surface und berührungsempfindlichen PCs mit Windows 7 einsetzbar sind. Folgende Details des SDKs wurden von der Webseite des Surface SDKs zusammengefasst.[3]

Das SDK läuft mit den aktuellen Technologien WPF 4.0, XNA 4.0 und Windows 7 (32- und 64-Bit). Um Programme mit dem aktuellen Surface SDK 2.0 zu entwickeln wird Microsofts Visual Studio 2010 vorausgesetzt. Damit lassen sich Windows-Anwendungen wie gewohnt mit WPF oder XNA erstellt. Das Surface SDK bietet zuzüglich Gestenerkennung für Multi-Touch-Oberflächen von Windowsgeräten. Die Erfassung der Gesten bezieht sich neben Punkt- und Fingererkennung auch auf das Tagging von Objekten. Dabei können bestimmte Gegenstände von dem Tisch erkannt werden. Dies bietet Entwicklern und Benutzern weitere Möglichkeiten der Interaktionen mit Multi-Touch-Geräten. Zudem werden bereits die gängigsten Gesten im Multi-Touch-Bereich unterstützt, zu welchen das Verschieben, Verkleinern, Vergrößern, Schwenken und Drehen von grafischen Objekten der Anwendung gehört.

Microsoft offeriert für die Entwicklung neben Visual Studio weitere Produkte, die das Entwickeln erleichtern sollen. So kann neben dem eingebauten WYSIWYG-Editor für GUI-Elemente von Visual Studio zusätzlich Expression Blend für die Erstellung eigener aber auch vorhandener UI Elemente verwendet werden. Mit Expression Studio können Designer Interaktionen, grafische Elemente und weitere Medien hinzufügen. Mit Hilfe dieser Software lassen sich leicht Eigenschaften, wie

[3] Microsoft: MSDN Surface. Online im Internet:
`http://msdn.microsoft.com/en-us/library/ff727864.aspx` – Zugriff am 12.08.2013.

Position, Farben, Lage, Form, Stil, verknüpfte Ereignisse und weiteres mehr über WIMPs konfigurieren. Bekannte und gewohnte UI-Elemente, wie Buttons, Labels, Slider, Textfelder, Layouts, etc., stehen bei der Wahl der grafischen Anwendungsoberfläche in einer umfangreichen Auswahlliste zur Verfügung.

Bereitgestellte Funktionen und Verfahren können in einer Dokumentierten API nachgelesen werden. Eine große Online-Gemeinschaft, bestehend aus Foren, Blogs und Fanseiten, liefert Publikationen, wie Einführungen in WPF und dem Surface SDK, Dokumentationen, Hilfestellungen bei Fragen und stellt Diskussionsrunden für häufig auftretende Probleme. Zum Testen und Debuggen werden mehrere Tools zum Surface SDK mitgeliefert. Darunter Testsoftware, wie *InputVisualizer* und *InputSimulator*, mit der ohne vorhandene Multi-Touch-Geräte und nur mit Hilfe von ein oder mehreren Zwei-Button-Mäusen Touch-Gesten als Eingabe simuliert werden können. Eingegebene Gesten können mit dem Debugging-Tool *Event Tracking* nachverfolgt werden, um so Eingaben besser nachzuvollziehen und mögliche Fehler bei der Gestenerkennung auszuschließen.[4]

4.2.4 Kivy

Das Community Projekt Kivy, geleitet von erfahrenen Softwareentwicklern,[5] ist eine Open-Source Python Bibliothek um das Entwickeln von Anwendungen mit innovativen Benutzerschnittstellen wie Multi-Touch-Eingaben zu erleichtern.[6] Diese Zusammenfassung beruht auf Informationen und Kapiteln bzw. Abschnitten der Kivy-Dokumentation.[7]

Dank der plattformunabhängigen Programmiersprache Python kann Kivy auf allen gängigen Betriebssystemen wie Windows, Linux und OS X sowie vielen verschiedenen Variationen von Geräten betrieben werden. Um eine hohe Flexibilität zu gewährleisten bietet das Framework Unterstützungen für neue externe Geräte und bietet Lösungen mit Hilfe von Software-Protokollen an. Dabei werden auch Lösungen von Drittanbietern so genannte „Third-party solutions“, wie zum Beispiel WM_TOUCH für Windows 7 Stift- und Touch-Geräte und HID kernel input events auf Linux-Systemen, verwendet. Ebenfalls kann unter OS X auf Apples multi-touch-fähige Geräte zurückgegriffen werden. Zusätzlich wird TUIO (Tangible User Interface Objects) und weitere Eingabequellen unterstützt.[8]

Kivy ist auf viele Arten optimiert worden. Dies betrifft unter anderem die Geschwindigkeit der Entwicklung sowie die Ausführungsgeschwindigkeit von erstellten Anwendungen mit Kivy. Um dieses Ziel zu erreichen wurden zeitkritische

[4] Vgl. Microsoft, a. a. O., MSDN Surface Overview.
[5] Vgl. Virbel, Mathieu/Hansen, Thomas/Denter, Christopher: Kivy Webseite: About us. Online im Internet: `http://kivy.org/#aboutus`⟩ – Zugriff am 12.08.2013.
[6] Vgl. Virbel, Mathieu/Hansen, Thomas/Denter, Christopher: Kivy Dokumentation: Release 1.8.0-dev. Online im Internet: `http://kivy.org/docs/pdf/Kivy-latest.pdf`⟩ – Zugriff am 12.08.2013, S. 1.
[7] Virbel, Mathieu/Hansen, Thomas/Denter, Christopher, a. a. O.
[8] Vgl. Virbel, Mathieu/Hansen, Thomas/Denter, Christopher, a. a. O., S. 21f..

Funktionen auf C-level[9] optimiert und kostspielige Operationen durch den Einsatz von intelligenten Algorithmen minimiert. Zudem wurde großen Wert auf eine zweckerfüllende Auslastung der GPU gesetzt. Da die GPU für grafische Aufgaben erstellt und optimiert wurde, hat diese einen erheblichen Vorteil in der Ausführung von grafischen Operationen gegenüber der CPU. Durch bevorzugtes und gezieltes Einsetzen der GPU wird somit die CPU entlastet und die Performance gesteigert.[10]

Kivy ist ein fokussiertes Framework. Mit nur wenigen Zeilen kann bereits eine einfache Anwendung erstellt werden. Dabei wird nicht ohne Grund auf Python gesetzt. Python ist eine vielseitige und leistungsstarke aber dennoch einfach zu handhabende Scriptsprache. Des Weiteren stellt Kivy eine eigene Beschreibungssprache, die *Kivy Language*, für anspruchsvolle Benutzeroberflächen zur Verfügung. Damit wird Entwicklern ein mächtiges Werkzeug zur schnellen und einfachen Verbindung sowie Erstellung von GUI-Elementen bereit gestellt. Kivy übernimmt sämtliche Compiler-Einstellungen und enthält in der portablen Version sämtliche benötigte Bibliotheken. Dies erlaubt Entwicklern unnötige Aufgaben auszublenden und sich auf ihre Hauptanwendung zu konzentrieren.[11]

Kivy ist sehr Einsteigerfreundlich. Zusammen mit einer ausführlichen Dokumention, der API mitsamt allen Widgets, zwei Programming Guidelines, einem Tutorial und Demos gibt es eine große hilfsbereite Community, die durch Foren und Chat für Fragen offen steht. Widgets sind zahlreich vorhanden und bieten neben Eigenkreationen Grundelemente der UI. Dazu zählen Buttons, Akkordions, Popup-Fenster, verschiedene Layouts (Box-, Float-, etc. Layout), Slider, Ladebalken, Labels, usw. Ähnlich zu MT4j wird hier ebenfalls eine virtuelle Tastatur angeboten. Beim Umgang mit Gesten können in Kivy verschiedene Multi-Touch-Gesten aufgenommen und komprimiert in eine Datenbank gespeichert werden. Dies ermöglicht während der Ausführung des Programms Eingabegesten mit Signaturen aus der Gestendatenbank zu vergleichen und so auf verschiedenste und kreativste Gesten zu reagieren.

Zusammengefasst bietet Kivy eine flexible plattformunabhängige Grundstruktur für Multi-Touch-Anwendungen. Eingaben werden durch eine Vielzahl von verschiedenen Geräten unterstützt. Dabei ist Kivy fokussiert um Entwicklern zeit zu sparen, das Arbeiten zu erleichtern und unnötige Arbeiten abzunehmen. Durch Optimierungen können Anwendungen annähernd auf Geschwindigkeiten von C entwickelt werden und bieten damit einen Performance-Vorteil gegenüber anderen Anwendungen. Kivy kann kostenlos und selbst in kommerziellen Anwendungen verwendet und verbreitet werden. Kivy steht unter der LGPL 3 Lizenz.

[9] Die Programmiersprache C ist aufgrund der Hardware-Nähe bekannt für optimierte Funktionen und deren schnelle Ausführung.

[10] Vgl. Virbel, Mathieu/Hansen, Thomas/Denter, Christopher: Kivy Dokumentation: Release 1.8.0-dev, a. a. O., S. 21.

[11] Vgl. Virbel, Mathieu/Hansen, Thomas/Denter, Christopher, a. a. O., S. 22.

4.2.5 Evaluierung der Frameworks

Mit unter den wichtigsten und relevanten Punkten eines geeigneten Frameworks für den zu entwickelnden Prototypen stellt, unserer Ansicht nach, die Plattformunabhängigkeit dar. Zusätzlich zum Multi-Touch-Tisch der Universität Paderborn mit dem Betriebssystem Windows 7 steht uns ein Android-Tablet zur Verfügung, auf dem wir ebenfalls Tests und insbesondere die Eignung und Unterstützung von Einzelarbeiten im Gegensatz zu Teamarbeiten am MTT untersuchen möchten. Wir sehen großes Potenzial in einem UML-Editor für MTTs und eventuelle Erweiterungen. Somit möchten wir uns Möglichkeiten für Portierungen auf andere Systeme und MTTs offen halten, wenn auch erst für spätere Weiterentwicklungen des Prototypen.

Da mit dem Surface SDK nur Anwendungen für Surface Geräte und Windows 7 unterstützt werden, haben wir uns gegen dieses Produkt entschieden. Auch bietet Surface viele Funktionen, Unterstützungen und Effekte im 3D Bereich, Umgang mit getaggten Objekten und leichte Erstellungen bzw. Setzung von festen GUI-Elementen, die für unsere 2D Anwendung mit schlichtem UI nicht benötigt werden.

Das java-basierte Framework MT4j sowie Kivy, der auf Python basierende Nachfolger von PyMT, sind plattformunabhängig. Beide bieten das Hinzufügen von eigenen Gesten, wobei MT4j zusätzlich bekannte Gesten wie Zoomen, Verschieben, Rotieren, o. Ä. erkennt. Kivy verfügt über keine expliziten vordefinierten Gesten, stellt aber bereits Funktionen wie `touch.grab(self)` oder `touch.is_double_tap` zur Verfügung, mit deren Einbindung sich leicht bekannte Gesten implementieren lassen, sowie eine `Scatter`-Klasse, die die obigen Funktionen unterstützt.

Durch bereits absolvierte Veranstaltungen mit Java haben wir Erfahrungen und grundlegende Kenntnisse in dieser Programmiersprache erworben. Kivy hingegen setzt auf Python, eine Programmiersprache, in der wir bisher noch keine Kenntnisse erworben haben, diese aber leicht zu erlernen sind. Beide Frameworks haben eine ausführliche Dokumentation und Tutorials, wodurch sich das Einarbeiten in beiden Fällen erleichtert. Wobei für Kivy weitere Programming Guidlines und sehr ausführliche Tutorials zur Verfügung stehen und kontinuierlich erweitert und verbessert werden.

Für unseren Prototypen haben wir uns für Kivy entschieden. Kivy profitiert von den Erfahrungen des Vorgängers PyMT, ebenfalls ein Python Multi-Touch-Framework. Dabei setzt Kivy unter anderem auf Cython, ein Python-Interpreter für die Zielsprache C, wodurch eine hohe Performance auf annähernd C-Level erreicht wird. Des Weiteren ist Kivy sehr aktuell und lebendig in der Entwicklung. Neue Updates folgen meist bereits nach Wochen. MT4j hingegen veröffentlichte die letzte stabile Version am 31 März 2011. Der Quellcode von Kivy ist auf der Git-Hosting-Webseite GitHub hochgeladen. Dies ermöglicht der gesamten Online-Gemeinschaft Entwicklungen von Kivy zu verfolgen und durch eigene Quellcode-Einsendungen zu beeinflussen. Ein weiterer Vorteil, der die enge und aktive Zusammenarbeit zwischen Community und Kivy erbringt, ist die schnelle

Erkennung und Behebung von Fehlern. Diese können ebenfalls auf der Hostingseite mit den Entwicklern ausgetauscht werden. Bereits erste Erfahrungen mit Kivy zeigten positive Resonanz. Mit nur wenig Erfahrung und Entwicklungsaufwand wurde nach wenigen Stunden eine funktionstüchtige Zeichen-Anwendung in Python geschrieben. Selbst eigene Gestenmuster ließen sich schnell und einfach integrieren. Auch in Gesprächen mit derzeitigen Projektgruppen im MTT-Umfeld bekamen wir Fürsprache und positive Erfahrungsberichte im Umgang mit Kivy. Aus diesen Gründen sollen die Möglichkeiten die Kivy bereit stellt näher erforscht werden. Ebenfalls soll diese Gelegenheit genutzt werden um eine neue Programmiersprache zur weiteren Fortbildung zu erlernen.

4.3 Hardware- und softwarebasierte Entwicklungsumgebung

Der zu erstellende Prototyp dieser Arbeit soll für folgende Hardware- und Software-Spezifikationen errichtet werden.

4.3.1 Stichdaten des Multi-Touch-Tisches

Die Entwicklung orientiert sich hauptsächlich an der Verwendung des Multi-Touch-Tisches „Mister T.“ (Multitouch Interactive Surface and Tangible Exploration Research Table) der eine Entwicklung der Universität Paderborn und der Masterarbeit von Adrian Hülsmann ist und sich aus folgenden Stichdaten zusammensetzt. Der Tisch besitzt eine Breite von 160cm, eine Tiefe von 95cm und fasst mindestens sechs Personen zum kollaborativen Arbeiten. Dabei ist der MTT eine projektionsbasierte Konstruktion die eine Full-HD Bildschirmauflösung von 1080p mit einem 16:9 Auflösungsformat und einer Gesamtauflösung von 1280x480 hat. Das Bild wird mit einem ACER P7500 Projektor, der eine Helligkeit von bis zu 4000 Lumen hat, erzeugt, wobei Spiegel den Lichtweg des Beamers umlenken, damit trotz Konstruktionsgröße und Projektionsabstand von 1,75m eine hohe Auflösung erreicht werden kann. Die Oberfläche des Touch-Tisches besteht aus einer robusten 8mm dicken Glasscheibe, die auf der Oberseite mit einer „digiline-white“ Projektionsfolie von Ifoha beckt ist, sodass sich eine kratzfeste und blickwinkeltreue Tischoberfläche ergibt. Für die Erkennung von Touch-Eingaben werden vier Infrarot-Laser eingesetzt, die sich in jeweils einer der Tischecken befinden. Diese erzeugen eine Laserschicht, die maximal einen Millimeter über der Tischoberfläche aufgespannt ist und so Eingaben wahrnehmen. Die 120° Line-Generator-Linsen werden in selbstgebauten Laserhalterungen kalibriert und einem 5V Netzteil betrieben. Zwei PS3 Kameras zeichnen Toucheingaben von der Unterseite des Tisches auf. Als Trackingsoftware wird das Programm CCV benutzt. Der MTT ist an einem Windows 7 Computer mit einem i7 Prozessor und einer ATI Radeon HD 5970 Grafikkarte angeschlossen. Von diesem werden, wie auch die Trackingsoftware, Multi-Touch-Anwendungen gestartet.

4.3.2 Technische Voraussetzungen

Für den Prototypen werden folgende technische Voraussetzungen benötigt. Der Prototyp soll mit dem Kivy Framework in der Version 1.6.0 entwickelt werden und kommt in einer portablen Version mit allen benötigten Bibliotheken, wie unter anderem Python, daher werden keine weiteren Voraussetzungen benötigt um den Prototypen auszuführen.[12] Allerdings wird eine OpenGL 2.0 Unterstützung des Systems empfohlen, da Testsausführungen unter einem System mit Windows 7 32-Bit und einer OpenGL Version von 1.4 fehlschlugen und keine Kivy-Applikationen ausgeführt werden konnten. Dabei wurde ein Fehler über eine fehlendes Update von mindestens OpenGL v.2.0 ausgegeben. Tests unter Windows 32-Bit mit OpenGL 3.3 und Windows 64-Bit mit OpenGL 4.2 waren erfolgreich. Zu beachten ist das für das ausführen des in dieser Arbeit bereitgestellten Pakets für Windows Systeme außer einer OpenGL 2.0 Unterstützung ebenfalls keine weiteren Voraussetzungen gebunden sind und sich der Editor über die ausführbare EXE-Datei ohne weiteres starten lässt. Kivy enthält Bibliotheken in der 32-Bit Version, wodurch sich auch erstellte Pakete für ein bestimmtes Betriebssystem weiterhin auf 32- sowie 64-Bit dieses Systems ausführen lassen.

4.3.3 Verwendete Hilfsprogramme

Folgende Entwicklungs-Programme und Hilfstools werden in dieser Arbeit verwendet.

- Eclipse Classic Juno v. 4.2.2 [13]
- Eclipse PyDev Plugin v. 1.5.2 [14]
- Microsoft Surface SDK 2.0 - Input Simulator [15]
- PyInstaller 2.0 [16]

Der Prototyp wird mittels Pyhton und Kivy in *Eclipse Juno* entworfen und getestet. Dabei wird das Eclipse Plugin *PyDev* verwendet und das Framework Kivy unter den Projekt-Einstellungen als Bibliothek eingebunden. Um das Projekt auch innerhalb Eclipse mit Kivy auszuführen wird zusätzlich ein neuer Python-Interpreter in Eclipse eingetragen der Kivys portablen Python Interpreter benutzt.

[12] Vgl. Virbel, Mathieu/Hansen, Thomas/Denter, Christopher: Kivy Dokumentation: Release 1.8.0-dev. Online im Internet: `http://kivy.org/docs/pdf/Kivy-latest.pdf` – Zugriff am 12.08.2013, S. 5.

[13] Eclipse Classic Juno: http://www.eclipse.org/downloads/packages/eclipse-classic-422/junosr2 - Zugriff am 10.08.2013

[14] Eclipse PyDev Plugin: http://marketplace.eclipse.org/content/pydev-extensions - Zugriff am 10.08.2013

[15] Microsoft Surface SDK 2.0: http://www.microsoft.com/en-us/download/details.aspx?id=26716 - Zugriff am 10.08.2013

[16] PyInstaller 2.0: http://www.pyinstaller.org - Zugriff am 10.08.2013

Um weiterhin die Funktionsweise der Multi-Touch-Anwendung auf ein Verhalten auf einer Multi-Touch-Oberfläche zu testen werden Touch-Simulations-Tools benutzt. Unter diesen Fallen die bereits im Kapitel 4.2.3 „WPF/.NET + Surface SDK“ angesprochenen Tools des Microsoft Surface SDKs. Diese Tests und Tools sollen hauptsächlich eingesetzt werden um das Entwickeln ohne Zugang zum MTT zu ermöglichen, zu erleichtern und dadurch auch zu beschleunigen. Tests unter realen Bedingungen am MTT sollen ebenfalls durchgeführt werden. Dazu soll der Quellcode der entstandenen Software bereits in ein ausführbares Paket gebunden werden und sich auf die Ausführung des Programms und insbesondere die Gesten-Steuerung in realer Umgebung beschränken. Das Tool *PyInstaller* soll dazu verwendet werden um ein ausführbares Paket für den am MTT eingesetzten Windows Computer zu erzeugen.

4.4 Anwendungsfälle des prototypischen UML-Editors

Das Anwendungsfalldiagramm (Use-Case-Diagramm) in Abbildung 4.2 dient der übersichtlichen Darstellung der Anwendungsfälle die im prototypischen UML-Editor unterstützt werden sollen. Für ein besseres Verständnis werden diese Anwendungsfälle im Weiteren genauer beschrieben und ihre Funktionalität erklärt. Dazu gehört jeweils eine Auflistung der genutzten Konzepte aus Kapitel 3 „Konzepte zur Unterstützung des Entwicklungsprozesses“ sowie eine optionale Beschreibung in welchem Umfang diese umgesetzt werden sollen und zuletzt eine Begründung für den Einsatz dieser Konzepte in der prototypische Implementierung. Die im Anwendungsfall-Diagramm dargestellten Akteure sind Teammitglieder bzw. Personen, die gemeinsam am Multi-Touch-Tisch ein Klassendiagramm erstellen. Da einem Teamleiter im Rahmen des Prototypen keine gesonderten Funktionen zugeordnet sind, werden diese ebenfalls zu den Teammitgliedern gezählt. Sämtliche Aktionen bzw. Anwendungsfälle werden ausschließlich auf dem MTT ausgeführt. Fälle mit der Beschriftung „Elemente“ vereinen in diesem Diagramm Fälle für Aufgabenelemente, Klassen und Assoziationen.

4.4.1 Am Multi-Touch-Tisch anmelden

In diesem Anwendungsfall wird das Anlegen eines persönlichen Nutzerprofils durch ein Teammitglied am MTT beschrieben. Mit diesen Profilen können nachfolgend Mitglieder Elementen des Klassendiagramms als Verantwortlichkeit zugewiesen werden. Das verwendete Konzept 3.1.3 „Anmeldung am Multi-Touch-Tisch“ wird im Prototypen über WIMP-Elemente realisiert, weil Tangibles am genutzten Multi-Touch-Tisch nicht eingesetzt werden können und eine Implementierung alternativer Umsetzungsmöglichkeiten einen zu großen Aufwand für den vorgegebenen Zeitrahmen darstellt.

Abbildung 4.2: Anwendungsfalldiagramm des prototypischen UML-Editors

4.4.2 Klassendiagramm erstellen

Ein Benutzer kann über das Hauptmenü ein neues Klassendiagramm erstellen. Dies ist eine grundlegende Funktion eines UML-Klassendiagramm-Editors.

4.4.3 Klassendiagramm speichern

Durch ein Teammitglied im Hauptmenü angestoßen, wird das gesamte Klassendiagramm gespeichert, womit die Arbeitsergebnisse festgehalten werden. Das Sichern des gesamten Diagramms wird nach dem Konzept 3.3.1 „Speichern von Diagrammen“ umgesetzt. Das Speichern sowie das damit einhergehende Laden von benutzerspezifischen Teildiagrammen ist für den Prototypen von großem Interesse, lässt sich jedoch nicht im Rahmen dieser Arbeit umsetzen. Zudem basiert das in der prototypischen Implementierung eingesetzte Speicherformat auf XML und nicht auf dem im Konzept erwähnten XMI-Format.

4.4.4 Klassendiagramm laden

Das gesamte Diagramm, welches im Anwendungsfall 4.4.3 „Klassendiagramm speichern“ erstellt wurde, wird durch ein Teammitglied auf dem Multi-Touch-Tisch über das Hauptmenü geladen. Dies wird im Konzept 3.3.2 „Laden von Diagrammen“ erläutert und ist in der prototypischen Implementierung notwendig, um bestehende Klassendiagramme weiter bearbeiten zu können.

4.4.5 Elemente erstellen und löschen

Das Erstellen und Löschen von Klassen und Assoziationen ist existentiell für das Erstellen eines Klassendiagramms. Sowie Aufgabenelemente eines der wichtigsten Elemente zur Unterstützung der Entwicklungsphase solcher Diagramme darstellen. Daher wird dieser Anwendungsfall in den geplanten Prototypen mit aufgenommen. Darunter zählen ebenfalls das Erstellen von Attributen und Methoden unter Klassen sowie das Erstellen von Restriktionen und n-ären Assoziationen, die Spezialfälle einer Assoziation. Für die Umsetzung der Assoziationen soll eine einfache Variante des Konzepts 3.2.7 „Erzeugung von Assoziationen und Bearbeitung ihrer Attribute“ verwendet werden. Aufgrund des Zeitrahmens sollen Assoziationen direkt gezeichnet werden, d. h. ohne Umwege bzw. Docking- und Fixpunkte, die im Konzept 3.2.8 „Assoziationspfad mit Fix- und Dockingpunkten beliebig führen“ erläutert werden. Da für eine prototypische Implementierung vorerst die Umsetzung von *einfachen* Klassendiagrammen angedacht ist, soll sich der Editor auf maximal eine Verbindung zwischen zwei Elementen beschränken.

4.4.6 Elemente anordnen

Das Anordnen von Elementen stellt eine intuitive Aufgabe dar, die ebenfalls notwendig für den Editor ist und somit ein Anwendungsfall des Prototypen be-

schreibt. Dabei existieren bereits verbreitete Gesten für Touch-Oberflächen zum Verschieben sowie Rotieren von Elementen, die nachfolgend auch im Editor eingesetzt werden. Diese beruhen auf keinem Konzept unsererseits, aber auf der Erfahrung und Verbreitung der Multi-Touch-Gesten. Weiterhin kann das gesamte Diagramm am Tisch verschoben werden, wie es im Konzept 3.2.4 „Verschieben von gruppierten Elementen“ beschrieben wird. Da Assoziationen an Elementen gebunden sind und keine Docking- bzw Fixpunkte, wie in Kapitel 3.2.8 „Assoziationspfad mit Fix- und Dockingpunkten beliebig führen“ beschrieben, sondern direkte Assoziationspfade umgesetzt werden, sind Assoziationen nur durch die Position der verbundenen Elemente verschiebbar.

4.4.7 Elemente bearbeiten

Nach dem Erstellen von Elementen sollen diese bearbeitet werden, um eine Weiterentwicklung des Klassendiagramms zu ermöglichen. Die im Konzeptkapitel 3.5 „Eingabemöglichkeiten“ aufgeführten intuitiven alternativen Eingabemöglichkeiten werden im Prototypen nicht umgesetzt, weil sie einen zu großen Aufwand für den Zeitrahmen der Arbeit bedeuten und teilweise aufgrund der Hardwarespezifikation des MTTs nicht umsetzbar sind. Ebenfalls wird das Konzept 3.2.7 „Erzeugung von Assoziationen und Bearbeitung ihrer Attribute“ für die Bearbeitung von Assoziationen umgesetzt, beschränkt sich dabei allerdings auf die Kreisauswahl der Assoziationsenden. Die Bearbeitung von Informationselementen, zu denen Attribute, Methoden, Kardinalitäten, Rollen, Klassennamen, Aufgabentitel und -beschreibung sowie Assoziationsbezeichner gehören, wird im Prototyp innerhalb einfacher Textfelder geschehen, um Anwender nicht durch feste unvollständige Eingabemöglichkeiten, wie Datentypen oder Ähnliches, einzuschränken. Die Bearbeitung von UML-Elementen soll sich im Prototypen vorerst auf grundlegende Elemente beschränken, d. h. es sollen keine Textfelder für weitere Informationsangaben wie beispielsweise Stereotypen von Klassen angezeigt werden. Diese Angaben können aber teilweise durch die uneingeschränkte Eingabe der Textfelder angegeben werden, wie Stereotypen beispielsweise zum Klassennamen ergänzt werden können. Insbesondere ist keine Highlighting-Funktion für Attribute und Methoden, wie zuerst in der Zielsetzung geplant, vorgesehen, da diese Funktionalität eine geringe Priorität für den Prototypen darstellt.

4.4.8 Aufgabenelemente teilen und in Klassen umwandeln

Um den Entwicklungsprozess bereits im Prototypen grob zu unterstützen sollen Aufgabenelemente geteilt und in Klassen umgewandelt werden können. Dabei wird das Konzept 3.2.1 „Aufgaben und Problemstellungen visualisieren“ mit einigen Einschränkungen eingesetzt. Da nach dem Konzept Aufgabenelemente beliebig in Teilaufgaben zerlegt werden können, wird beim Zerlegen eine Kopie des Elements erstellt, mit sämtlichen Inhalten und dem zugewiesenen Benutzer. Nach dem Zerlegen können weitere Kopien durch erneutes direktes Zerlegen erstellt werden,

um beliebig viele Teilaufgabenelemente zu erzeugen. Da ein Umsetzen von Assoziationen auf andere Elemente mit Hilfe des Konzepts 3.2.8 „Assoziationspfad mit Fix- und Dockingpunkten beliebig führen“ nicht implementiert wird, werden verknüpfte Assoziationen an alle Teilaufgaben kopiert, wie im Konzept der Aufgabenelemente beschrieben. Überschüssige Assoziationen können anschließend leicht gelöscht werden. Zudem sollen Aufgabenelemente in Klassen umgewandelt werden können, dabei wird der Aufgabentitel zum Klassennamen und der Inhalt bzw. die Aufgabenstellung des Aufgabenelements in das Kommentarfeld der Klasse übertragen. Die eingetragene Verantwortlichkeit wird bei diesem Prozess beibehalten.

4.4.9 Marker setzen

Für die zu bearbeitenden Elemente sowie ihre Informationselemente sollen Marker durch Teammitglieder gesetzt werden um den Besprechungs- bzw. Entwicklungsstatus anzugeben. Wie im Konzept 3.4.1 „Visuelle Marker für Besprechungsfortschritte“ beschrieben, fördert diese Funktionalität die Kollaboration und stellt neben den Aufgabenelementen ein wichtiges Element zur Unterstützung der Entwicklungsphase dar, weswegen diese in die prototypische Implementierung aufgenommen wird.

4.4.10 Verantwortlichkeiten zuweisen

Den Diagrammelementen werden Teammitglieder zugewiesen um Verantwortlichkeiten und Zuständigkeitsbereiche festzulegen. Das dazu verwendete Konzept 3.1.4 „Zuweisung von Verantwortlichkeiten“ wird im Prototyp umgesetzt, um die Arbeitsteilung zu unterstützen, welche ein wesentlicher Bestandteil der Kollaboration ist.

4.4.11 Kommentare verfassen und bearbeiten

In Kommentarfeldern, die jeweils einem Element des Klassendiagramms angehören, können Teammitglieder Notizen zu Gedankengängen, Ideen und besprochenen Inhalten hinterlassen. Daher kann auf den Einsatz von zusätzlichen Medien zum Festhalten von Informationen beim Einsatz des Prototypen verzichtet werden. Das zugehörige Konzept 3.4.2 „Kommentarfunktion“ wird im Prototypen implementiert, weil Notizen in nahezu jeder Besprechung festgehalten bzw. verwendet werden.

4.4.12 Detailstufen von Elementen

Die Anwender des Prototypen sollen die Anzahl der angezeigten Informationen zu den vorhandenen Diagrammelementen geringfügig verändern können. So lassen sich Klassen, wie im Konzept 3.1.2 „Anzeige von verschiedenen Detailstufen“ beschrieben, ein- und ausklappen und Rollen, Kardinalitäten sowie der Bezeichner

von Assoziationen bei Überlappungen mit anderen Informationselementen automatisch ausblenden. Diese Funktionalität wird in die prototypischen Implementierung aufgenommen, weil sie den Benutzern des MTTs die Anzeige der jeweils notwendigen Informationen und einen besseren Umgang mit dem Anzeigeplatz ermöglicht. Die Umsetzung wird aus Gründen des Aufwands für eine prototypische Implementierung begrenzt.

4.5 Architektur des prototypischen UML-Editors

In diesem Kapitel wird die Architektur des prototypischen Editors erläutert. Dabei werden Softwaretechniken beschrieben die größtenteils bereits durch das Kivy-Framework vorgegeben werden und im Prototypen zum Einsatz kommen. Zudem wird ein Entwurf des Klassendiagramms vorgestellt, wobei näher auf wichtige Bausteine bzw. Klassen des Prototypen eingegangen wird.

4.5.1 Zum Einsatz kommende Softwarepattern

Mit dem Gebrauch des Kivy Frameworks werden bereits Entwurfsmuster der Softwareentwicklung genutzt bzw. können benutzt werden. So wird ein Beobachter-Entwurfsmuster verwendet um Entwicklern die Möglichkeit zu bieten auf Änderungen von Kivy-Attributen wie zum Beispiel die Größe oder Position von Elementen zu reagieren. Gewünschte Ereignisse müssen zuvor registriert werden und einer Callback-Funktion zugewiesen werden, die ausgeführt wird sobald das Ereignis eintrifft. Dabei wurden Python-Properties für die Umsetzung von Kivy-Attributen eingesetzt. Python-Properties erlauben es versteckte Setter und Getter für Attribute zu definieren, die je nach Schreib- bzw. Lesezugriff im Hintergrund aufgerufen werden. Dies ermöglicht auch Kivy-Attribute selbst als Callback-Funktionen einzusetzen. Somit können beispielsweise Kind-Elemente auf die gleiche Größe des Eltern-Elements gebunden werden, sodass sich Kind-Elemente bei jeder Veränderung der Größe ihres Eltern-Elements dieser automatisch anpassen. Eine Art des Model-View-Controller Pattern strukturiert Applikationen im Framework Kivy. Dabei können `.kv`-Dateien als Views interpretiert werden. Diese enthalten die Darstellung eines Widgets. `Widget`-Elemente sind Kivy-Klassen die als Controller dienen. Diese verarbeiten Touchpoint-Eingaben durch drei vererbte Methoden (`on_touch_down`, `on_touch_move`, `on_touch_up`), verarbeiten Daten von Models und aktualisieren Views entsprechend. Models können durch einfache Python-Klassen dargestellt werden, ein Beispiel dafür stellt die in diesem Prototypen zu erstellende „User“-Klasse. Für das Speichern des Diagramms bzw. speichern sämtlicher Widget-Elemente mit allen Attributen bietet sich ein Visitorpattern an, das allerdings in diesem Prototypen nicht zum Einsatz kommen soll, da Kivy Widgets über eine verlinkte Listen-Struktur mit „Parent“- und „Children“-Zeigern verarbeitet. Daher soll stattdessen diese Liste genutzt werden um über alle Elemente zu iterieren. Siehe dazu auch das folgende Kapitel 4.7 „Speichern und Laden“.

4.5.2 Klassendiagramm des Prototypen

Das in Abbildung 4.3f. dargestellte Klassendiagramm wurde als Entwurf vor der Implementierung erstellt und soll einen Überblick über die in der prototypischen Implementierung verwendete Architektur geben. Das Diagramm wurde auf eigenständig entwickelte Klassen begrenzt und führt nur Attribute, Methoden und Parameter auf, die das Verständnis der Architektur fördern sollen. Zu den nicht explizit aufgeführten Methoden gehören neben den Konstruktoren die Touch- und Hilfs-Methoden sowie Setter und Getter. Weiterhin erben alle Klassen mit einer grafischen Darstellung von der Kivy-Klasse: `Scatter` oder `Widget`. Kivy-Klassen werden ebenfalls für eine bessere Übersicht über das Diagramm nicht dargestellt. Die genutzte Programmiersprache Python ermöglicht Mehrfachvererbung. Im geplanten Prototypen soll dies, wie im Klassendiagramm sichtbar, genutzt werden. Auf die zur Implementierung geplanten Gesten sowie Speicher- und Lade-Funktionalitäten wird in den Unterkapiteln 4.6 „Eingesetzte Gesten“ und 4.7 „Speichern und Laden“ eingegangen, da diese umfangreiche Funktionalitäten beschreiben. Zu beachten gilt, dass `void` als Rückgabewert bei Methoden durch das verwendete Werkzeug ArgoUML[17], mit Hilfe dessen das folgende Klassendiagramm erstellt wurde, nicht dargestellt wird. Deswegen sind Methoden mit fehlender Angabe des Rückgabewerts stellvertretend für Methoden mit Rückgabewert `void` anzusehen. Zudem wurde das Klassendiagramm für eine bessere Lesbarkeit auf zwei Seiten aufgeteilt. Damit Assoziationen dabei nicht über die Seitenränder gezeichnet werden müssen, sind in Abbildung 4.4 die gelb markierten Klassen der Abbildung 4.3 erneut und eingeklappt aufgeführt. Folgend werden die *wichtigsten* Klassen und ihre Verbindungen zu Weiteren erläutert.

UMLEditor

Die in dieser Arbeit entwickelte prototypische Anwendung wird über die Klasse `UMLEditor` gestartet, welche ihrerseits von der `APP`-Klasse des Kivy-Frameworks erbt. Die `build`-Methode weist der `editor`-Variable eine Instanz zu, die Diagramme verwaltet und zeigt anschließend das Hauptmenü (`MainmenuPopup`) an. Die weiteren Methoden dienen der Konfiguration der Anwendung.

MainmenuPopup

Das Hauptmenü wird durch die Klasse `MainmenuPopup` realisiert und nutzt die Klasse `MainmenuDialog` für die grafische Darstellung. Das Hauptmenü ist dafür zuständig neue Klassendiagramme zu erstellen, zu laden und zu speichern. Zudem kann es bei einer Weiterentwicklung der Anwendung um weitere Diagrammarten ergänzt werden. Weiterhin können über diese Klasse neue Benutzer (`User`) einem geöffneten Diagramm hinzugefügt werden, wofür die Klasse `UserAdministration` genutzt wird. Die dem `MainmenuPopup` bekannten Klassen `SaveFilePopup` und

[17] ArgoUML: http://argouml.tigris.org - Zugriff am 10.07.2013

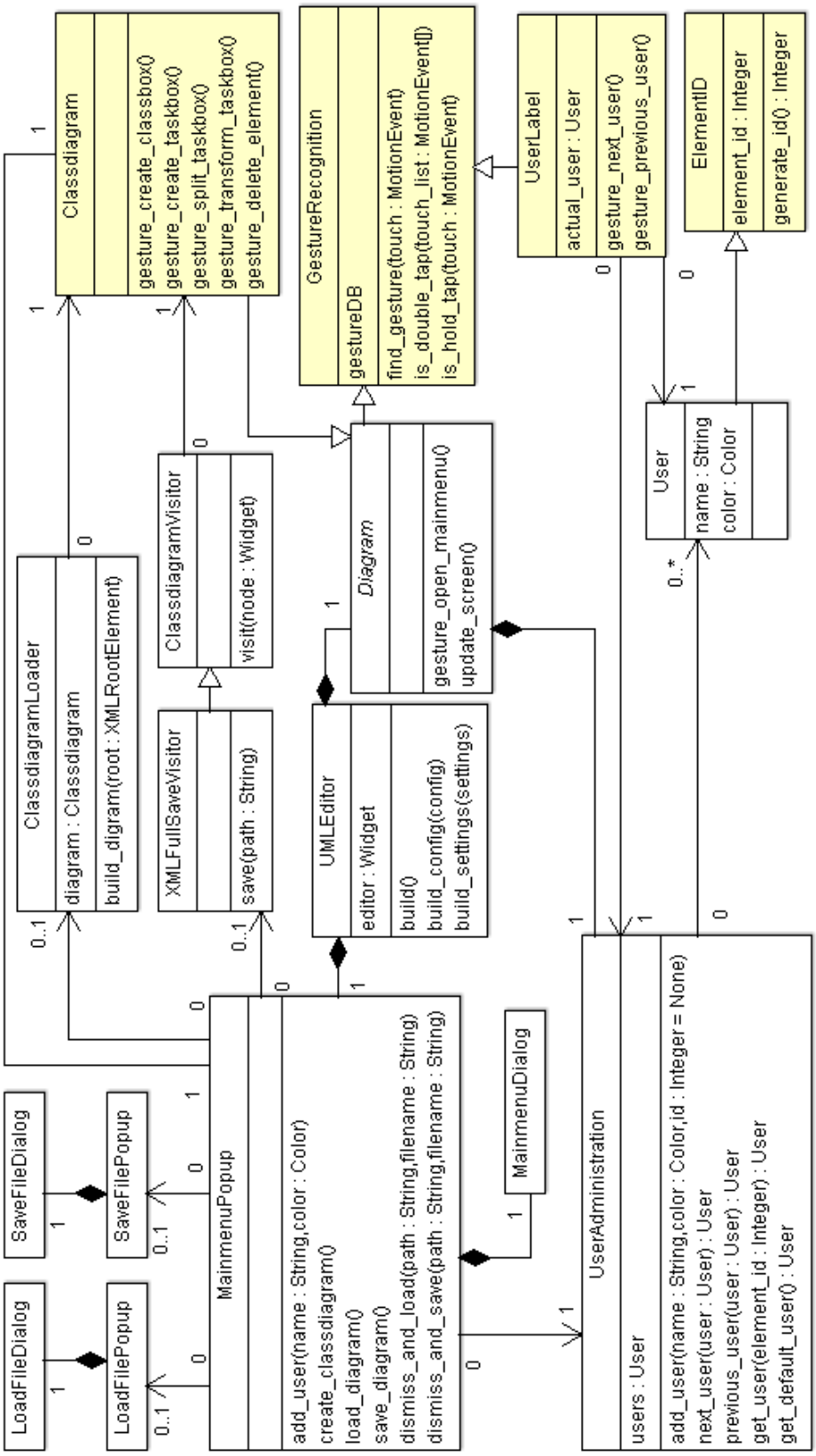


Abbildung 4.3: Klassendiagramm des prototypischen UML-Editors (Teil A)

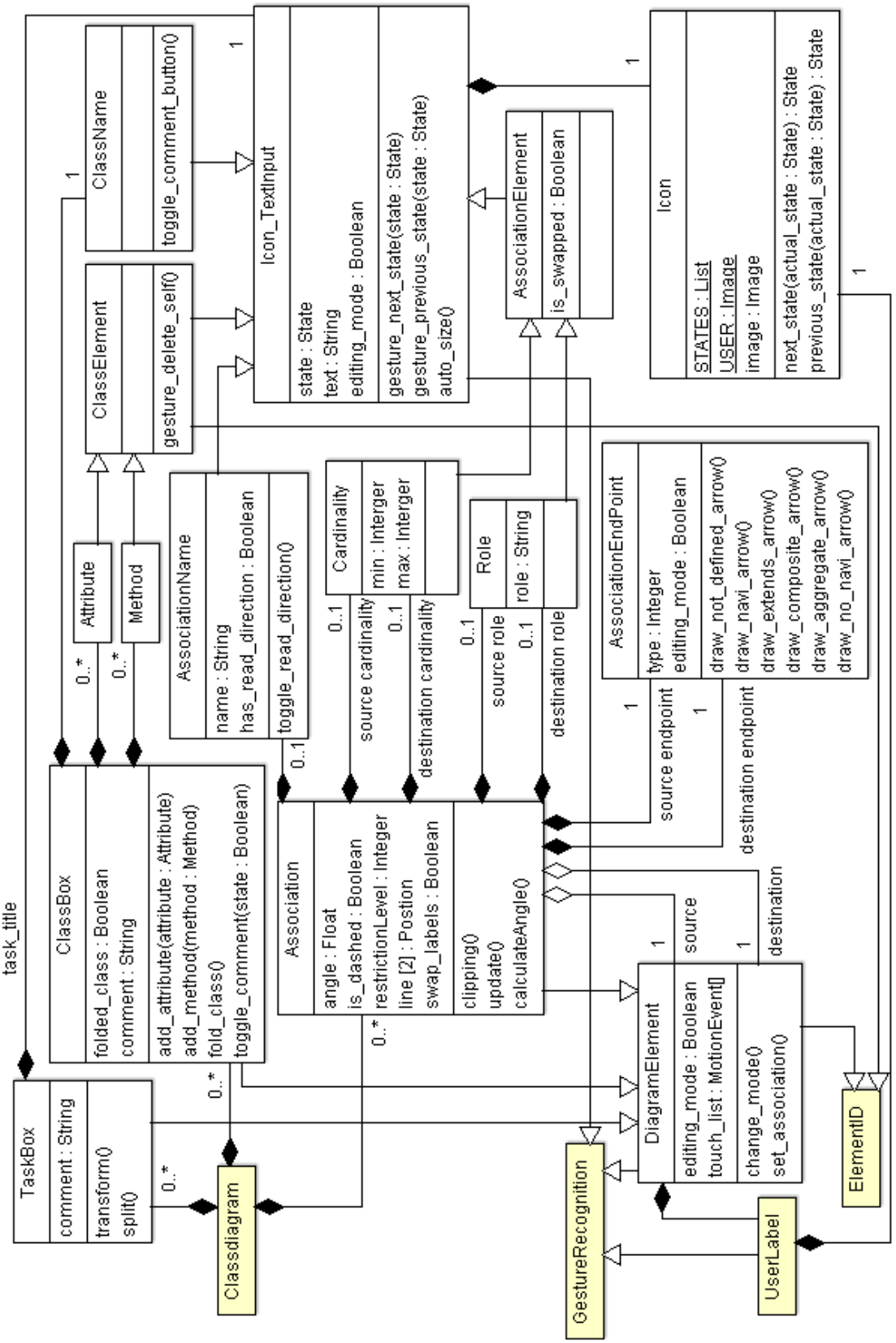


Abbildung 4.4: Klassendiagramm des prototypischen UML-Editors (Teil B)

`LoadFilePopup` helfen bei der Auswahl der Datei in die geschrieben bzw. die eingelesen werden soll. Die Klassen `XMLFullSaveVisitor` und `ClassdiagramLoader` übernehmen das anschließende Speichern oder Laden eines Klassendiagramms.

Classdiagram

Diese Klasse erbt von der `Diagram`-Klasse, welche allgemeine Methoden für Diagramme bereitstellt, wie beispielsweise das Öffnen des Hauptmenüs, und selbst von der Klasse `GestureRecognition` erbt, die eine Gestenerkennung zur Verfügung stellt. `Classdiagram` beinhaltet die angelegten Klassen (`ClassBox`), Aufgaben (`TaskBox`) und Assoziationen (`Association`) und kann Klassen und Aufgabenelemente aufgrund der erkannten Gesten erstellen und löschen. Letzteres gilt auch für Assoziationen. Weiterhin ist diese Klasse für das Teilen und Umwandeln von Aufgabenelementen zuständig.

DiagramElement

Gemeinsamkeiten von Klassen, Aufgabeelemente und Assoziationen sollen in der Klasse `DiagramElement` ausgelagert werden. Diese erbt sowohl von `ElementID`, zur eindeutigen Identifikation des Elements, als auch von `GestureRecognition`, wegen der Gestenerkennung. Sie kann zudem zwischen einem Ansichts- und einem Bearbeitungsmodus wechseln und Assoziationen zwischen `DiagramElement`en erstellen. Zudem beinhaltet diese Klasse die `UserLabel`-Klasse, um die jeweiligen Verantwortlichen anzuzeigen.

UserLabel

Die Klasse `UserLabel` besteht aus der `Icon`-Klasse um grafisch zu verdeutlichen, dass in ihrem Anzeigefeld Benutzer (`User`) dargestellt werden. Um die Unterscheidung der Benutzer zu vereinfachen besitzen diese neben einem Namen auch eine Farbe, die als Hintergrundfarbe für das Anzeigefeld gewählt wird. Zur eindeutigen Identifizierung erbt die `User`-Klasse von der Klasse `ElementID`, sodass die Benutzerverwaltung (`UserAdministration`) der `UserLabel`-Klasse immer den richtigen Benutzer zurückgibt, wenn dieser gewechselt werden soll.

ClassBox

Die Klasse `ClassBox` stellt eine Klasse in UML-Notation mit einem Klassennamen (`ClassName`), Attributen (`Attribute`) und Methoden (`Method`) dar, wobei letztere durch diese Klasse hinzugefügt werden und von der Klasse `ClassElement` erben. Diese bietet den Erbenden unter anderem eine Funktion um sich selbst aus der zugeordneten Klasse zu entfernen. Durch das Erben von `DiagramElement` kann der `ClassBox`-Klasse ein Benutzer zugewiesen werden. Zusätzlich besitzt sie ein Kommentarfeld, das über die Klasse `ClassName` ein- und ausgeblendet werden

kann. Über das Ein- und Ausblenden von Attributen und Methoden kann zwischen verschiedenen Detailstufen von Klassen gewechselt werden.

TaskBox

Die Klasse `TaskBox` soll die beschriebenen Aufgabenelemente des Konzepts 3.2.1 „Aufgaben und Problemstellungen visualisieren“ umsetzen. Diese erbt von der Klasse `DiagramElement`, wodurch ihr ein Verantwortlicher zugewiesen werden kann. Der Aufgabentitel wird mit Hilfe der Klasse `Icon_TextInput` dargestellt und der Inhalt bzw. die Aufgabenstellung durch ein mehrzeiliges Textfeld. Des Weiteren werden Methoden bereit gestellt mit denen Aufgabenelemente durch die Klasse `Classdiagram` geteilt und umgewandelt werden können.

Association

Die Klasse `Association` bildet die Assoziationen zwischen `DiagramElementen` ab und erbt selbst von `DiagramElement`. Je nach Typ der `Source`- und `Destination`-Rolle sollen somit Einschränkungen zwischen Assoziationen und n-äre Assoziationen repräsentiert durch eine Assoziationsklasse (d. h. eine Assoziation soll mit einer Klasse verbunden werden) dargestellt werden. Ist die Assoziation an zwei Assoziationen befestigt so ist diese eine Einschränkung und soll folglich gestrichelt und ohne Rollen- sowie Kardinalitäten-Angabe gezeichnet werden. Wurde die Assoziation zwischen einer Assoziation und einer Klasse erstellt ist diese eine Verknüpfung zu einer Assoziationsklasse und es entfällt zu der Darstellung einer Einschränkung auch noch der Assoziationsbezeichner. Der Typ der Assoziation zeichnet sich im Attribut `restriction_level` nieder, sowie `is_dashed` ob die Assoziation gestrichelt gezeichnet werden soll. Um eine Linie an den Seiten zweier Elemente zu zeichnen sollen Assoziationsendpunkte mit Hilfe von Algorithmen und auf Grundlage des Mittelpunktes und der Position der Elemente, in der Methode `clipping` berechnet werden. Diese Punkte sollen im Attribut `line` gespeichert werden. Je nach Lage der Assoziation müssen alle Beschriftungen der Verbindung im Winkel `angle`, der mit durch die Methode `calculateAngle` berechnet wird, gedreht werden und im Falle das Labels auf dem Kopf stehen durch `swap` umgedreht werden. Eine `update`-Methode sorgt dafür das berechnete Winkel und Assoziationsendpunkte bei veränderten Parametern der verbundenen Elemente erneut berechnet und aktualisiert werden. Da die Verantwortlichkeit von Assoziationen sich bereits durch die angrenzenden Klassen repräsentiert sollen Attribute und Methoden der `DiagramElement` Klasse die zur Funktionalität des User-Labels gehören ignoriert werden. Zusätzlich zu zwei Verbindungselementen in den Rollen `Source` und `Destination` besteht eine Assoziation je nach Typ aus den weiteren Klassen `Role`, `Cardinality`, `AssociationName` und `AssociationEndPoint` um Verbindungsinformationen zu visualisieren.

Dabei soll `AssociationEndPoint` Methoden bereitstellen die einzelne Pfeile und Symbole der Navigierbarkeit von Assoziationen zeichnen. Die Klassen `Role` und

`Cardinality` enthalten beide das Attribut `is_swapped`, welches durch die Oberklasse `AssociationElement` vererbt wird und durch `Associationen` gesetzt wird. `Role` und `Cardinality` werden weiterhin als `Icon_TextInputs` dargestellt.

Der Associationsname repäsentiert durch `AssociationName` ist ebenfalls eine Unterklasse von `Icon_TextInputs`. Zusätzlich soll diese neben dem Textfeld auch die Leserichtung darstellen sowie das Wechseln dieser durch die Funktion `toggle_read_direction`

Icon_TextInput

Die `Icon_TextInput`-Klasse besteht aus einem Textfeld des Kivy-Frameworks und der Klasse (`Icon`), die einen Marker passend zum Besprechungsstatus des Textfeldes anzeigt. Das Textfeld kann sich in der Breite dem enthaltenen Text anpassen, sowie die Höhe dem enthaltenen Zeilen, und mit Hilfe von Gesten kann der Besprechungsstatus des Textfeldes gewechselt werden. Zur Erkennung von Gesten erbt die Klasse `Icon_TextInput` von der `GestureRecognition`-Klasse und vererbt anschließend alles an die Klassen `ClassName`, `ClassElement`, `AssociationName` und `AssociationElement`. Weiterhin kann über eine Variable eingestellt werden, ob das Textfeld editierbar ist.

4.6 Eingesetzte Gesten

Nachfolgend werden wichtige Gesten der geplanten Software beschrieben. Diese wurden zum großen Teil vor der Implementierung festgelegt, wobei darauf geachtet wurde, dass diese möglichst intuitiv sind und nicht miteinander kollidieren. In bestimmten Fällen, wie beispielsweise dem Erstellen und Umwandeln eines Aufgabenelements, wurde eine Geste gewollt mit unterschiedlichem Verhalten belegt. Zudem ist zu beachten, dass die Gesten der prototypischen Implementierung nur von einer Tischseite aus in Leserichtung gezeichnet werden können und nur eine bestimmte Ausführungsrichtung dieser Gesten implementiert wird. So lässt sich zum Beispiel ein „M“ zum Öffnen des Hauptmenüs nur von links unten startend nach rechts zeichnen. Die Grafiken zur Veranschaulichung der Gesten sind der Anwendung nach der Implementierung entnommen. Dabei stellt ein Punkt den ersten Touchpoint der Ausführungsgeste dar und eine mit dem Punkt verbundene Linie die Gestenführung nach dem auflegen des Fingers bzw. ersten Touchpoints der Geste.

Geste zum Öffnen des Hauptmenüs

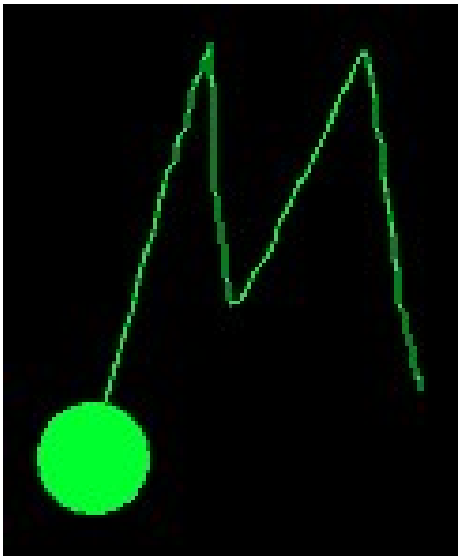


Abbildung 4.5: Geste zum öffnen des Hauptmenüs

Das Hauptmenü öffnet sich, sobald ein großes M auf der Diagrammfläche gezeichnet wird (Abb. 4.5), wobei es nicht auf einem Diagrammelement beginnen darf. Dabei werden alle eingeblendeten Tastaturen auf dem Multi-Touch-Tisch automatisch ausgeblendet.

Geste zum Erstellen einer Aufgabe

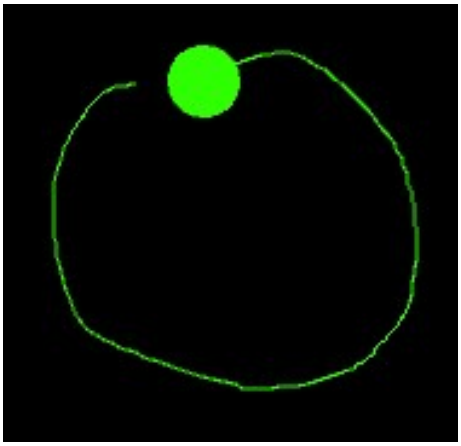


Abbildung 4.6: Geste zum Erstellen einer Aufgabe

Ein neues Aufgabenelement entsteht in der Mitte eines, mit einem Finger im Uhrzeigersinn gezeichneten Kreises, wobei am obersten Punkt des Kreises begonnen wird. Mit dieser Geste (Abb. 4.6) wurde versucht eine Anlehnung an die Geste zum Erstellen von Klassen zu schaffen, die sich trotzdem eindeutig von jener unterscheidet.

Geste zum Teilen einer Aufgabe

Abbildung 4.7: Geste zum Teilen einer Aufgabe

Um ein Aufgabenelement in 2 Teilaufgaben zu zerlegen kann die Teilen-Geste (Abb. 4.7) genutzt werden. Dabei wird ein senkrechter Strich durch ein Aufgabenelement gezogen, der bereits über der Aufgabe beginnt. Die Geste kann sowohl im Aufgabenelement, als auch unter diesem enden und soll dem Benutzer signalisieren, dass das Element ab dem Zeitpunkt aus zwei Teilen besteht. Nach dem Ausführen der Geste wird das Originalelement um seine halbe Breite nach links verschoben, während rechts neben dem Element eine exakte Kopie der Aufgabe mitsamt allen Inhalten, derselben Hintergrundfarbe, dem gleichen Benutzer und allen Assoziationen des Originals erstellt wird. Diese Geste bzw. Funktion ist nur für Aufgabenelemente verfügbar.

Geste zum Umwandeln einer Aufgabe in eine Klasse

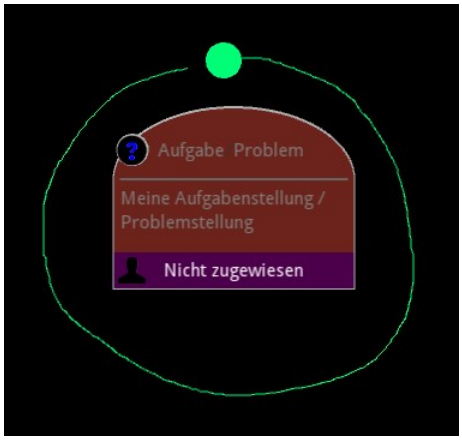


Abbildung 4.8: Geste zum Umwandeln einer Aufgabe

Zum Umwandeln einer Aufgabe in eine Klasse wird ein gezeichneter Kreis, wie er zuvor in 4.6 „Geste zum Erstellen einer Aufgabe“ beschrieben wurde genutzt, wobei sich in dessen Mitte ein Aufgabenelement befindet. Diese Geste (Abb. 4.8) überlagert absichtlich die Geste zum Erstellen von Aufgaben um die Gedächtnisbelastung der Benutzer in sofern zu reduzieren, als das keine neue Geste gelernt werden muss. Beim Umwandeln wird anstelle der Aufgabe eine neue Klasse erstellt, wobei der Aufgabentitel zum Klassennamen wird und die Aufgabenbeschreibung im Kommentarfeld der neu erstellten Klasse erscheint während der zugewiesene Verantwortliche beibehalten wird. Diese Geste bzw. Funktion ist nur für Aufgabenelemente verfügbar.

Geste zum Erstellen einer Assoziation

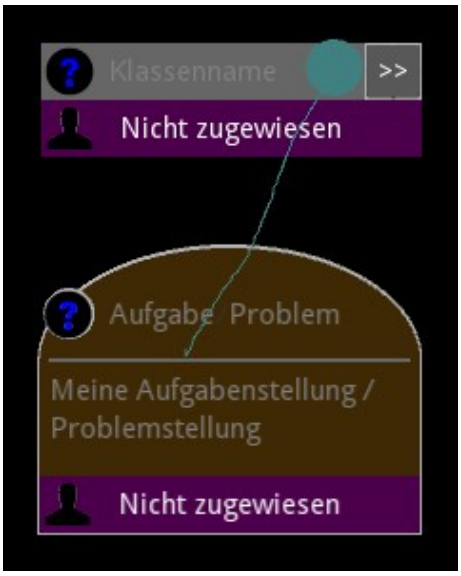


Abbildung 4.9: Geste zum Erstellen einer Assoziation

Eine Assoziation wird über einen Double-Tap, also dem zweifachen Antippen eines Elements und dem anschließenden Ziehen einer Verbindung zu einem anderen Element erstellt, wie es in Abbildung 4.9 zu sehen ist. Dabei wird beim zweiten Antippen nicht losgelassen, bis das gewünschte Element erreicht worden ist. Diese Geste gilt auch für Spezialformen von Assoziationen.

Geste zum Erstellen einer Klasse

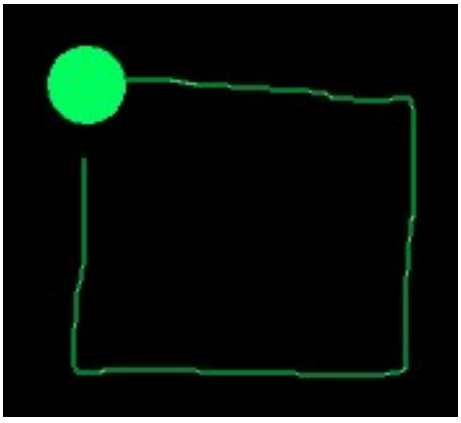


Abbildung 4.10: Geste zum Erstellen einer Klasse

Zum Erstellen einer Klasse wird ein Rechteck mit einem Finger gezogen (Abb. 4.10). Die Geste beginnt dabei am linken, oberen Eckpunkt und zieht sich im Uhrzeigersinn über die weiteren Eckpunkte. Beim Loslassen der Geste entsteht eine leere Klasse, deren linke, obere Ecke auf dem Startpunkt der Geste liegt. Die Idee entstammt eigenen Beobachtungen, in denen der Rahmen für neue Klassen an eine Tafel oder ein Whiteboard von verschiedenen Personen gezeichnet wurde.

Geste zum Löschen von Attributen und Methoden

Abbildung 4.11: Geste zum Löschen von Attributen und Methoden

Zum Löschen eines Attributs oder einer Methode einer Klasse, wird dieses bzw. diese von links nach rechts mit einem Finger durchgestrichen. Diese Geste (Abb. 4.11) wurde der Löschen-Geste der Android-App Any.do nachempfunden und muss im Attribut bzw. der Methode beginnen und enden.[18] Das Löschen von Attributen bzw. Methoden ist nur im Bearbeitungsmodus einer Klasse möglich.

Geste zum Ein- & Ausklappen einer Klasse

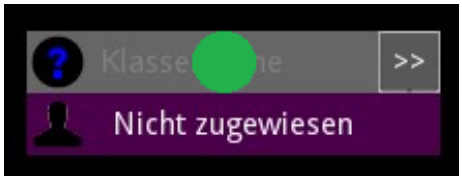


Abbildung 4.12: Geste zum Ein- & Ausklappen einer Klasse

Zum Ein- & Ausklappen einer Klasse muss diese wie im COSMOS-Projekt lediglich mit dem Finger angetippt werden[19](Abb. 4.12), wobei die Anzeige des verantwortlichen Benutzers davon ausgenommen ist. Diese Funktion lässt sich nur im Ansichtsmodus und nicht im Bearbeitungsmodus nutzen.

[18] Vgl. Any.do: Frequently Asked Questions. Online im Internet: `http://www.any.do/faq⟩` – Zugriff am 12.08.2013

[19] Vgl. Löffelholz, Daniel et al.: COSMOS: Multi-touch support for collaboration in user-centered projects. In Heiss, Hans-Ulrich (Hrsg.): Informatik 2011. Bonn: Ges. für Informatik. Online im Internet: `http://www.user.tu-berlin.de/komm/CD/paper/061521.pdf⟩` – Zugriff am 04.01.2013, ISBN 978–3–88579–286–4, [S. 10]

Geste zum Löschen eines Elements

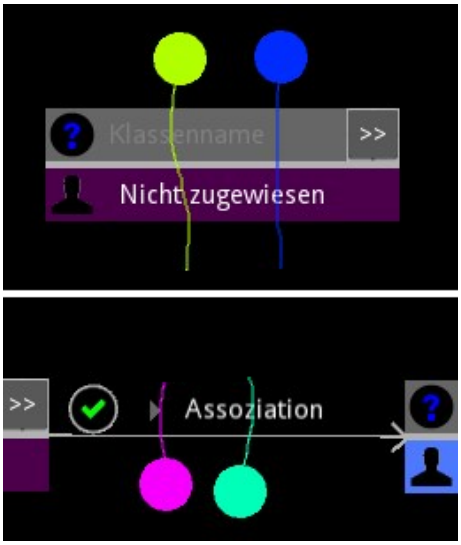


Abbildung 4.13: Geste zum Löschen eines Elements

Damit ein Element wie eine Assoziation, Aufgabe oder Klasse gelöscht wird muss dieses mit zwei gleichzeitig aufgesetzten Fingern durchgestrichen werden (Abb. 4.13). Dabei ist darauf zu achten, dass die Geste in der Nähe, jedoch außerhalb eines Elements beginnt und aufhört. Diese Geste wurde für vier Richtungen (von oben, von unten, von links und von rechts) umgesetzt, weil Elemente beispielsweise an Tischrändern schwer zugänglich sind und eine einzelne Richtung für diese Geste daher nicht ausreicht.

Geste zum Wechseln von Markern und Verantwortlichen

Abbildung 4.14: Geste zum Wechseln von Markern und Verantwortlichen

Während für das Wechseln von Verantwortlichen eine Streichgeste nach links bzw. rechts (Abb. 4.14) auf der gesamten Benutzeranzeige erfolgen kann, muss diese bei Markern genau auf einem Markericon beginnen.

Geste zum Wechseln von Modi

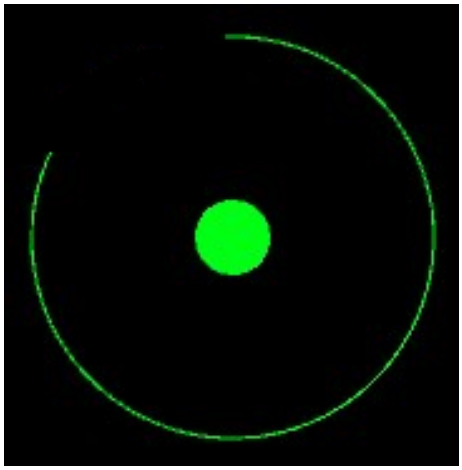


Abbildung 4.15: Geste zum Wechseln von Modi

Modi können in der prototypischen Implementierung über ein Hold-Tap gewechselt werden. Dabei sind zwei unterschiedliche Modi-Wechsel zu beachten. Zum einen ist dies der Wechsel auf Diagrammebene zwischen dem Ansichtsmodus, in dem Elemente betrachtet sowie Gesten gezeichnet werden können und dem Modus zum Drehen, Verschieben und Zoomen des gesamten Klassendiagramms. Dazu ist ein Hold-Tap von mindestens zwei Sekunden außerhalb von Diagramm-Elementen nötig, wobei eine Fortschrittsanzeige in Form eines entstehenden Kreises (Abb.4.15) angibt, wann diese Zeit erreicht. Zum anderen ist der Wechsel zwischen Ansichtsmodus und Bearbeitungsmodus von Klassendiagramm-Elementen gemeint, der bereits von COSMOS genutzt wurde,[20] jeweils nur für ein einzelnes Element gilt und durch einen Hold-Tap von einer halben Sekunde ausgelöst wird, was einem kurzen Verharren vor dem Loslassen des Touchpoints entspricht. Diagramm-Elemente können nur im Bearbeitungsmodus bearbeitet werden.

Gesten zum Drehen, Verschieben und Zoomen

Zum Drehen, Verschieben und Zoomen des Klassendiagramms, sowie dem Drehen und Verschieben von einzelnen Klassen- bzw. Aufgabenelementen wurden die von Kivy vorgegebenen Standard-Gesten genutzt.

4.7 Speichern und Laden

Um Diagramme zwischen Arbeitsphasen festzuhalten werden diese in einer XML-Datei gespeichert. Dabei werden alle benötigten Daten in einem eigenständig entwickelten Format festgehalten das im Anhang B „Das verwendete Speicherformat“ dokumentiert ist. Der Aufbau besteht aus vier Teilen: einer Auflistung der am

[20] Vgl. Löffelholz, Daniel et al., a. a. O.

Diagramm beteiligten Personen, einer Aufzählung der vorhandenen Klassen- und Aufgabenelemente, sowie einer Liste von Assoziationen. Umschlossen werden diese Angaben von einem Tag der den gespeicherten Diagrammtyp, in diesem Fall ist es ein Klassendiagramm, samt der benötigten Informationen wiedergibt. Weil die Bezeichner von Diagrammelementen, wie ein Klassen- oder Benutzername, nicht eindeutig sein müssen und verändert werden können, wird allen Elementen und Teilelementen eine eindeutige Identifikationsnummer zugeteilt. Auf diese Weise ist es möglich alle Diagrammelemente eindeutig zu identifizieren und beispielsweise beim Laden eines Diagramms sicherzustellen, dass eine Assoziation die richtigen Klassendiagramm-Elemente mit einander verbindet.

Beim Sichern werden zuerst die Informationen des Klassendiagramms extrahiert und im Anschluss die Benutzer, die Klassendiagramm-Elemente und die Verbindungen durchsucht. Die Reihenfolge ist dabei unerheblich, da das Diagramm in der Anwendung existiert und somit alle Informationen vorhanden sind. Beim Laden eines Diagramms hingegen ist die Reihenfolge von großer Bedeutung, weil Assoziationen beispielsweise auf weitere Diagrammelemente verweisen und diese dafür schon geladen sein müssen. Deswegen werden nach der Diagrammerstellung zunächst die Benutzer geladen, sodass die im Anschluss eingelesenen Klassen- und Aufgabenelemente jene als Verantwortliche angeben können. Zuletzt werden die Assoziationen eingelesen, sodass sie die bereits geladenen Elemente miteinander verbinden können.

5 Implementierung

In diesem Kapitel werden eigene Erfahrungen, Eindrücke und vor allem Probleme, die während der Umsetzung des Prototypen gesammelt werden konnten beschrieben. Dazu zählt der Umgang mit dem eingesetzten Multi-Touch-Tisch sowie die Verwendung von unterstützenden Tools, dem Kivy-Framework und der für uns neuen Programmiersprache Python. Auch soll die Umsetzung einiger Konzepte beschrieben werden, die in dieser prototypischen Implementierung eingebaut wurden. Dabei wird hauptsächlich die *Implementierung* dieser unter Augenschein genommen. Ob die hier eingesetzten Konzepte ihren Zweck erfüllen, soll in dieser Arbeit durch Benutzertests ermittelt und untersucht werden. Diese werden nachfolgend im Kapitel 6 „Evaluierung“ genauer erläutert. Eine Installationsanleitung am Ende dieses Kapitels soll es ermöglichen den entwickelten Prototypen auf verschiedenen Plattformen zu installieren und zu testen.

Um einen Rückschlag bei der Implementierung des Prototypen mit dem ausgewähltem Framework zu vermeiden, fand bereits während der Erstellung des Proposals dieser Arbeit eine Einarbeitung in Python und Kivy statt. Aus diesem Grund konnte in der Implementierungsphase sofort mit der Erstellung des Prototypen begonnen werden. Die in dieser Arbeit festgelegte Vorgehensweise sah vor, dass zunächst die Grundfunktionalität eines Klassendiagramm-Editors rudimentär umgesetzt werden soll. Anschließend sollen die Kernfunktionen der umzusetzenden Konzepte implementiert werden. Nach der Fertigstellung des Grundgerüsts des Prototypen sollen sowohl der Klassendiagramm-Editor als auch die Konzepte verfeinert werden, sodass sie in der Evaluation sinnvoll getestet werden können. Zu dem Zeitpunkt sind ferner Tests der prototypischen Implementierung am Multi-Touch-Tisch vorgesehen, sodass Schwierigkeiten und Fehler zeitgleich mit der abschließenden Entwicklung behoben werden können. Dieses Vorgehen sollte sicherstellen, dass ein testbarer Prototyp möglichst schnell verfügbar war und dass nicht Zeit für die Feinabstimmung eines Konzepts genutzt wurde, die zur Umsetzung eines anderen geplanten Konzepts notwendig wäre. Durch diese Vorgehensweise konnte die Implementierungsphase nach Zeitplan beendet werden und war nicht vollständig vom Fortschritt der Implementierung abhängig.

5.1 Eingesetzte Hard- und Software

In diesem Kapitel wird der Nutzen und Umgang der Verwendeten Hardware und Softwaretools, sowie Probleme die dabei aufgetreten sind näher erläutert. Dabei wird insbesondere die Verwendung des Tisches, der Simulationstools und Erstel-

lung der ausführbaren Pakete und Tests auf verschiedenen Systemen beschrieben.

Das Testen des Prototypen und insbesondere die damit verbundenen Gesten- und Touch-Eingaben wurden einerseits mit den von Kivy unterstützten Maus-Eingaben, sowie mit den Touch Simulations-Programm *Input Simulator* getestet. Dabei hatten beide Varianten einen annähernd gleichen Simulationswert, der sich zum schnellen ausprobieren und für die grobe Funktionsweise eignet, aber für genauere Touch-Eingaben und -Arbeiten ungeeignet ist. Zum Kalibrierung der Gesten und speichern von Gesten-Mustern mussten Tests vor Ort am MTT selbst durchgeführt werden, da diese Aufgaben durch die Eingabe einer Computer-Maus zu ungenau waren. Ein Beispiel dafür stellte die Mustereingabe sowie auch spätere Ausführung der „ZickZack"-Geste dar. Diese konnte auf dem Tisch leicht mit einem Finger ausgeführt werden. Mit einer Maus allerdings konnte diese Geste nur schwer nachgestellt werden bzw. führte zu einem völlig anderen Ausführungsmuster als durch die Finger-Eingabe.

Die geplante Verwendung der Version 1.6.0 von Kivy wurde während der Implementierung auf die Version 1.7.1 hoch gesetzt um hauptsächlich wichtige Bugfixes, aber auch neuere Funktionen, wie den `Colorpicker` in den Editor zu integrieren. Dabei wurde allerdings das Verhalten einiger Grundelemente stark verändert, sodass einige Klassen größtenteils neu aufgebaut werden mussten.

Frühere Tests der Portabilität und des Touch-Verhaltens wurden auf Tablets und Smartphones mit laufendem Android-System durchgeführt. Um eine Kivy-Applikation auf ein Android-Gerät einzurichten kann die App *Kivy Launcher* aus dem Android-Market genutzt werden, sowie ein Paket für Andoird mit dem Tool *Pyinstaller* erstellt werden. Mit dem Kivy Launcher muss der Quellcode des Prototypen lediglich per Datentransfer in einen Ordner der Kivy App geladen werden und kann anschließend über den Launcher ausgeführt werden. Die Test liefen dabei sehr erfolgreich, allerdings aufgrund des mangelnden Bildschirmplatzes im Vergleich zur Zielgruppe der MTTs, ist die Ausführung des Prototypen bzw. der Nutzen eher begrenzt.

Zum testen des Editors am MTT wurden ausführbare Pakete für Windows Systeme erstellt. Diese wurden mit dem Tool *Pyinstaller* erstellt, der dank einer Anleitung der Kivy Dokumentation schnell und leicht für Kivy eingerichtet werden konnte. Erste Builds waren allerdings fehlerhaft und führten zu abstürzten, dies war wiederum auf Fehlern und experimentelle Funktionen innerhalb des schnell fortlaufenden Kivy Frameworks zurückzuführen. Mit Hilfe der Community und Bug-Fixes einer aktuelleren Entwicklerversion des Frameworks konnten die Fehler allerdings schlussendlich behoben werden.

5.2 Entwicklung mit Python und Kivy

Kivy ist ein junges Framework, welches deswegen durchgehend gepflegt und um nötige bzw. gewünschte Funktionen und GUI-Elemente erweitert wird. Das heißt wiederum, dass Funktionen und Elemente, wie sie aus ausgereiften Programmier-

sprachen und Frameworks bekannt sind, nicht alle gegeben sind oder sich noch in der Entwicklung befinden. Dazu gehört beispielsweise das Deaktivieren von GUI-Elementen wie Buttons, sodass diese nur unter bestimmten Voraussetzungen benutzt werden können und Benutzer sehen, dass die Funktionalität vorhanden ist, selbst wenn Anwender diese zu dem Zeitpunkt nicht benutzen können. Solche nicht vorhandene Funktionalität wird oftmals von der Community entwickelt und an die Kivy-Entwickler weitergereicht, die diese anschließend ins Framework einbinden.

Problematisch ist weiterhin, dass das Verhalten von Funktionen und Elementen nicht immer dem dokumentierten Verhalten oder den Erwartungen von Entwicklern entspricht, die diese von anderen Sprachen und Frameworks mitbringen. In diesem Zusammenhang kann beispielsweise der Double-Tap erwähnt werden, der dem zweifachen Antippen des Multi-Touch-Tisches, wie bei einem Doppelklick, gleicht. Dieser besitzt als Variablen eine Distanz und den Zeitunterschied zwischen den beiden einzelnen Touchpoints. Für beide Variablen werden in einer Konfigurationsdatei Werte festgelegt die besagen wann genau zwei Touchpoints einen Double-Tap ergeben. Obwohl dies entsprechend dokumentiert ist, werden Double-Taps erkannt, die nach den vorgegebenen Werten keine sind. Ein Beispiel für nicht erfüllte Erwartungen ist das TextInput-GUI-Element, welches auch als mehrzeiliges Textfeld eingesetzt werden kann. Dieses bricht bei einer festen Zeilenbreite jedoch keine Zeilen selbständig um und besitzt keine Scrollfunktionalität für den Fall, dass mehr Zeilen eingegeben werden als angezeigt werden können.

Während der Einarbeitungsphase in Python und Kivy sowie der Implementierungsphase erschienen mehrere Updates des Kivy-Frameworks. In diesen wurde das Verhalten von verschiedenen Funktionen wie auch GUI-Elementen soweit verändert, dass der in dieser Arbeit implementierte Prototyp nicht mehr funktionsfähig war. Hierzu zählt der zuvor beschriebene Double-Tap, dessen Verhalten mit einem Update umgestellt wurde, sodass die in der Implementierungsphase entwickelte Unterscheidung zwischen Single- und Double-Tap ein unerklärliches Verhalten aufwies. Weil die Updates jedoch viele neue Funktionen und GUI-Elemente, Vereinfachungen sowie Bugfixes mit sich brachten, wurden diese trotz der Probleme die sie verursachten als Mehrwert angesehen und eingesetzt. Bei der großen Anzahl an protokollierten Veränderungen in den Changelogs der Updates war es nicht möglich diese nachzuvollziehen ohne die eigenen Aufgaben dadurch zu vernachlässigen.

Insgesamt kann in dieser Arbeit bestätigt werden, dass Kivy mit Python schnell erlernt und dass bei dieser Kombination mit wenig Programmcode viel erreicht werden kann. Aufgrund mangelnder Erfahrung mit der Python-Kivy-Kombination, vor allem zu Beginn der Implementierungsphase, wurde das Entwurfs-Klassendiagramm mit leichten Abweichungen umgesetzt. Weiterhin stellten wir mit fortschreitender Entwicklung und zunehmender Erfahrung fest, dass der Programmcode an verschiedenen Stellen optimaler umgesetzt bzw. effizienter programmiert sein könnte. Dies wurde jedoch nicht an allen Stellen verbessert, weil in dieser Arbeit beispielsweise der Weiterentwicklung des Prototypen und der Behebung

kritischer Fehler mehr Priorität zugemessen wurde.

5.3 Umgang mit Touch-Ereignissen

Um den Umgang mit Touch-Ereignissen in Kivy zu verstehen muss zunächst kurz auf die Organisation der GUI-Elemente, die alle von der Kivy-Klasse `Widget` erben und daher *Widgets* heißen, eingegangen werden. Diese sind in Kivy-Anwendungen in einer Baumstruktur angeordnet, sodass eine Anwendung ein Widget als Wurzel-Element besitzt, das weitere Widgets enthält, die ihrerseits wiederum GUI-Elemente enthalten können. In der grafischen Darstellung liegen die Kind-Elemente auf dem Element, das sie enthält, sodass das Wurzel-Element die grafisch unterste Ebene im Anzeigefenster bildet.[1]

Ein Touch-Ereignis ist ein spezielles Objekt der Kivy-Klasse `MotionEvent`, das mindestens eine X- und Y-Position enthält. Um solche Ereignisse zu verarbeiten, können in allen Widgets die Methoden `on_touch_down`, `on_touch_move` und `on_touch_up` überschrieben werden.[2] Die `on_touch_down`-Methode wird aufgerufen, sobald ein Touch-Ereignis das Widget erreicht und ist dafür zuständig dieses an alle Kind-Elemente weiterzuleiten, in denen das Ereignis aufgrund seiner Position liegt. Soll ein Touch-Ereignis von einem Widget verarbeitet werden, so kann es zunächst prüfen, ob das Ereignis bereits von einem Kind-Element bearbeitet wurde und nach der Bearbeitung den Wahrheitswert `True` zurückgeben um dem Eltern-Element zu signalisieren, dass auf das Ereignis bereits reagiert wird. Weil Ereignisse nur über diese Methode propagiert werden, muss ein Touch-Ereignis über die Anweisung `touch.grab(self)` ergriffen werden, damit die beiden weiteren Methoden aufgerufen werden.[3] Die Methode `on_touch_move` ist zum Verarbeiten von Positionsänderungen beispielsweise während einer Geste gedacht und wird dementsprechend nur aufgerufen wenn ein Touchpoint sich bewegt und in der `on_touch_down`-Methode ergriffen wurde. Die Methode `on_touch_up` wird aufgerufen sobald bei einem Touchpoint der Finger vom Tisch genommen wird. An dieser Stelle kann beispielsweise geprüft werden, ob eine Geste erkannt wurde oder zwischen einem Double-Tap, einem Single-Tap und einem Hold-Tap unterschieden werden. Abschließend muss das ergriffene Touch-Ereignis mit der Anweisung `touch.ungrab(self)` wieder freigegeben werden.

Aufgrund der Touch-Methoden und der Verteilung der Touch-Ereignisse über die Widget-Baumstruktur erschien eine zentrale Klasse zur Gestensteuerung nicht sinnvoll, weswegen auf die Mehrfachvererbung zurückgegriffen wurde um Programmcode wieder verwenden zu können. Je mehr dies geschehen ist und je tiefer der Umgang mit den Touch-Ereignissen weitervererbt wurde desto komplizierter

[1] Vgl. Virbel, Mathieu/Hansen, Thomas/Denter, Christopher: Kivy Dokumentation: Release 1.8.0-dev. Online im Internet: `http://kivy.org/docs/pdf/Kivy-latest.pdf` – Zugriff am 12.08.2013, S. 77f..
[2] Vgl. Virbel, Mathieu/Hansen, Thomas/Denter, Christopher, a. a. O., S. 74.
[3] Vgl. Virbel, Mathieu/Hansen, Thomas/Denter, Christopher, a. a. O., S. 76.

wurde das gesamte Verhalten. So sorgte es für mehrere Rückschläge während der Implementierung bei dem Versuch die Ereignisse über mehrere Vererbungsstufen mit verschiedenen Reaktionen der Anwendung zu belegen und gleichzeitig die geerbten Reaktionen beizubehalten.

5.4 Mehrfachvererbung

Entgegen der uns bisher bekannten Vererbungskonzepte in Programmiersprachen ist in Python die Mehrfachvererbung erlaubt. Sie erlaubt es von mehreren Klassen zu erben, wobei diese in Form einer Auflistung angegeben werden und wodurch Interfaces wie sie beispielsweise aus Java bekannt sind nicht benötigt werden. Bei gleichnamigen Attributen bzw. Methoden, wird implizit dasjenige verwendet, dessen Klasse zuerst in der Auflistung genannt wurde, weshalb im Konstruktor einer solchen Klasse die Konstruktoren der vererbenden Klassen explizit mit den passenden Parametern aufgerufen werden müssen.[4] Aus diesem Grund ist besondere Vorsicht bei der Mehrfachvererbung mit gleichnamigen Attributen bzw. Methoden geboten.

Im Prototypen wurde während der Implementierung die Entscheidung gefällt Funktionalität wie die Gestenerkennung und die Vergabe von Element-IDs über die Mehrfachvererbung anzubieten. Dadurch konnte Programmcode wiederverwendet werden und Methoden die lediglich auf andere Klassen weiterleiteten um dort eine zur Verfügung gestellte Methode auszuführen wurden somit überflüssig. Um die vorhin beschriebenen Probleme mit gleichnamigen Attributen und Methoden zu vermeiden wurde darauf geachtet diese unterschiedlich und sprechend zu bezeichnen.

5.5 Algorithmen des Prototypen

Für den Prototypen wurde Algorithmen hauptsächlich in den Assoziationen und Aufgabenelementen benutzt. Die Darstellung von Assoziationen benötigt für die richtige Lage, d. h. Winkel und Position einigen Rechenaufwand. Da die Linie der Assoziation, sowie die Pfeile bzw. Symbole dieser genau an den Kanten von Klassen- oder Aufgabenelementen gezeichnet werden, müssen Schnittpunkte an den Kanten berechnet werden. In diesem Prototypen wurde das Problem auf rechteckige Elemente und Assoziationen mit direkten Verbindungen eingegrenzt. Dadurch reduziert sich das Problem der Schnittpunktberechnungen auf Schnitte an der unteren, oberen, linken oder rechten Kanten sowie an Eckkanten eines Elementes. Diese Schnittpunktberechnung kann mit Hilfe eines „Line-Clipping-“Algorithmus, wie der Cohen-Sutherland-Algorithmus oder mit Cyrus-Beck und

[4] Vgl. Lutz, Mark/Ascher, David/Gherman, Dinu C.: Einführung in Python: [moderne OO-Programmierung ; behandelt Python 2.5]. 2. Auflage. Beijing und Cambridge und Farnham und Köln und Paris und Sebastopol und Taipei und Tokyo: O'Reilly, 2007, ISBN 978–3–89721–488–0, S. 405.

Liang-Barsky, die Erweiterungen des Cohen-Sutherland sind, gelöst werden. In diesem Prototypen wurde der Cohen-Sutherland-Algorithmus[5] implementiert. Dabei wird eine Line vom Mittelpunkt des ersten Elements bis zum Mittelpunkt des zweiten Elements gebildet, die letztendlich an den Außenkanten von beiden Elementen geschnitten werden soll.

Mit Hilfe des Cohen-Sutherland-Alogithmus wird allerdings eine *Schnittlinie* ermittelt, die innerhalb des Rechtecks liegt. Da der Mittelpunkt eines Elements jeweils den Anfang bzw. das Ende der zu schneidenden Assoziation bildet, ist dieser Mittelpunkt immer teil der resultierenden Schnittlinie. Der zweite Punkt der resultierenden Schnittlinie, ist der Schnittpunkt der Assoziation an diesem Element. Um die Assoziation an den Kanten zu zeichnen, werden nur die Schnittpunkte gespeichert. Damit beide Schnittpunkte der Assoziation berechnet werden, muss ein Schnittpunkt jeweils einmal mit einem der verbundenen Elemente berechnet werden.

Dazu wird die Linie jeweils im *lokalen* Koordinatensystem des Rechteckelements, anstatt im Weltkoordinatensystem, geschnitten, um Schnittpunkte von rotierten Rechtecken bzw. Elementen ebenfalls berechnen zu können. Würde der Schnittpunkt eines rotierten Rechtecks im Weltkoordinatensystem berechnet, werden falsche Clippinglinien, und damit ein falsche Schnittpunkt, berechnet. Dies liegt daran, dass ein rotiertes Rechteck, im Weltkoordinatensystem, einer Raute, einem Drachen o. Ä. und keinem, für die Berechnung notwendigen, Rechteck entspricht. Im lokalem Koordinatensystem des Elements bleibt das dargestellte Rechteck unverändert, da dieses im Weltkoordinatensystem rotiert wurde und somit ebenfalls sein eigenes Koordinatensystem Dieses Problem wird rechts in der Abbildung 5.1 grafisch verdeutlicht. Oben rechts werden die Clippinglinien des Elements im Weltkoordinatensystem und unten rechts im lokalen Weltkoordinatensystem des Elements dargestellt. Wie deutlich zu sehen ist entsprechen im oberen Bild die Clippinglinien nicht den Kanten des Elements.

Für die Beschriftungen einer Assoziation, die in jeder Lage und Position auf der Assoziationslinie dargestellt werden soll, wurde der Rotierungswinkel der Assoziation berechnet. Der Rotierungswinkel der Assoziation entspricht dem Winkel zwischen X-Achse und der Linie. Mit diesem Wert werden die Bezeichner ebenfalls rotiert, sodass diese auf der Assoziationslinie dargestellt werden. Dabei wechseln die Beschriftungen ihre relative Position innerhalb der Assoziation je nach positiver bzw. negativer Steigung der Linie, damit Beschriftungen nicht auf dem Kopf angezeigt werden. Zu beachten gilt das der Prototyp sich auf das Arbeiten von Personen auf einer Seite beschränkt.

Weitere Algorithmen wurden in diesem Prototypen nicht verwendet. Allerdings wurde für die Visualisierung von Aufgabenelementen eine optisch passende Farbe berechnet. Aufgabenelemente bekommen zum Erstellungszeitpunkt eine zufällige Hintergrundfarbe zugewiesen, da diese Farbe einen Großteil des Elements belegt,

[5] Rauber, Thomas: Algorithmen in der Computergraphik. 1. Auflage. Stuttgart: Teubner Verlag, 1993, ISBN 3–519–02127–7, S. 59ff..

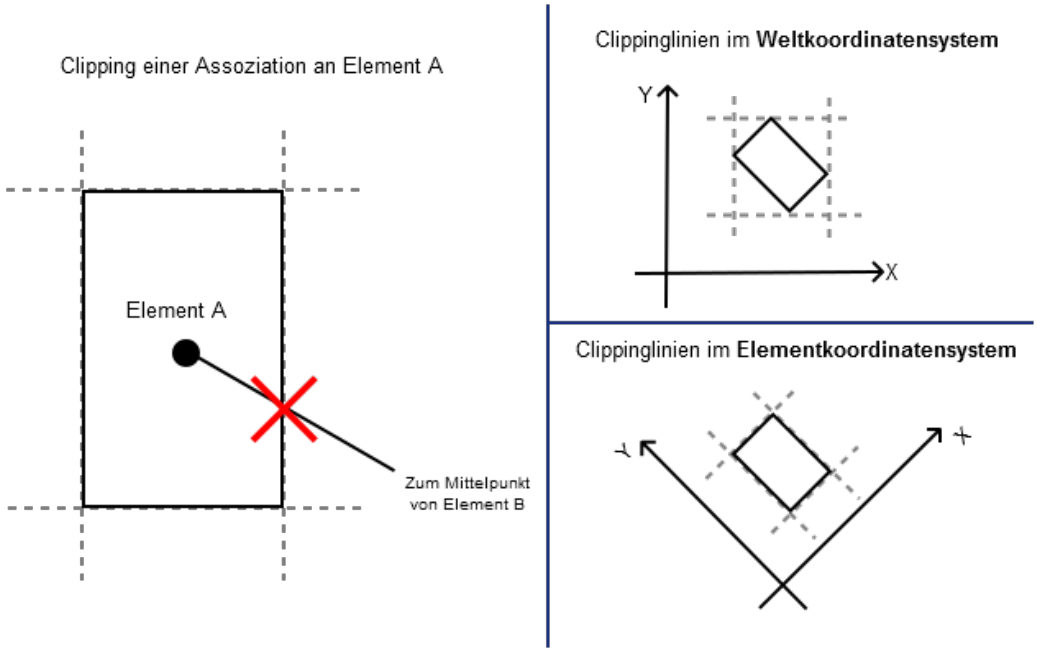


Abbildung 5.1: Schnittpunktberechnung für ein Element mit dem Cohen-Sutherland Algorithmus (l) sowie falschen (oben r) und richtigen (unten r) Clippinglinien eines rotierten Elements

soll diese möglichst freundlich und passend zur Rahmen- und Schriftfarbe erzeugt werden. Dazu wird ein *Color-Matching*-Verfahren verwendet, welches, bei Eingabe einer Farbe, eine Zufällige Farbe, aber mit gleicher Farb-Intensität, erzeugt. Das Ergebnis liefert eine Hintergrundfarben die optisch mit der eingegeben Farbe harmoniert.

5.6 Implementierung der Konzepte

Durch eine gute Abschätzung der Implementierung im Kapitel 4 „Anforderungsanalyse und Systementwurf“ und Einschränkungen in der Umsetzung einiger Konzepte, konnte der Umfang der geplanten Funktionen bzw. Konzepte eingehalten werden. Zudem wurde Syntax-Highlighting als eine weitere Funktionalität aufgenommen, da diese durch die neue Version des Kivy-Framework von den Entwicklern bereitgestellt wurde. Diese Funktionalität wurde allerdings nur benutzt und nicht weiterentwickelt, da sie mit einer niedrigen Priorität für Prototypen eingestuft wurde. Als problematisch und schwierig stellte sich das Implementieren von Gesten heraus. Daher wurde auf umfangreichere und komplexe Gesten verzichtet und zudem ein Bearbeitungs- und Ansichts-Modus für Elemente geschaffen. Dieser Modus sollte einerseits mehr Übersicht schaffen indem Bearbeitungselemente, wie Schaltflächen für das Hinzufügen von Attributen bei einer Klasse, ausgeblendet

werden. Anderseits konnten so Elemente im Ansichtsmodus beispielsweise leichter verschoben werden, da keine Gesten für Bearbeitungen innerhalb des Elements geprüft werden mussten und weniger Gesten mit ähnlichen Gesten kollidierten.

Ungeachtet dieser kleineren Schwierigkeiten konnten sämtliche für die Implementierung geplanten Konzepte erfolgreich implementiert werden.

5.7 Installationsanleitung

Auf der in dieser Arbeit beigefügten *DVD 3: Benutzertests und Prototyp*, die im Anhang D „Prototypische Implementierung und Benutzertest-Aufnamen“ zu finden ist, befindet sich ein ausführbares Paket bzw. Build des Prototypen für Windows 32- und 64-Bit-Systeme. Um den Prototypen am MTT „Mister T.“ zu starten sollten vorher die Trackingsoftware *CCV* sowie das Tool *BSQ Simulator* gestartet werden. Bei einem Ausführen bzw. Testen ohne Multi-Touch-Tisch reicht das starten der *umleditor.exe* im *umleditor*-Ordner. Bei der Verwendung gilt zu beachten dass dies lediglich ein Prototyp ist, weshalb Fehler und Anwendungsabstürze nicht ausgeschlossen werden können.

Da Python plattformunabhängig ist lässt sich der Prototyp laut Kivy Dokumentation auch auf Linux, Windows, MacOSX, Android und IOS ausführen. Davon wurden bereits Windows und Android erfolgreich getestet. Mit Hilfe von *PyInstaller* oder anderen Tools können Pakete auch für diese Systeme zur einfacheren Handhabung und Einrichtung erstellt werden. Eine Anleitung dazu ist in der Kivy-Dokumentation zu finden.[6]

[6] Virbel, Mathieu/Hansen, Thomas/Denter, Christopher: Kivy Dokumentation: Release 1.8.0-dev. Online im Internet: `http://kivy.org/docs/pdf/Kivy-latest.pdf`⟩ – Zugriff am 12.08.2013, S. 115ff..

6 Evaluierung

In diesem Kapitel soll der in der Implementierungsphase entstandene Prototyp und die darin umgesetzten Konzepte evaluiert werden. Im Folgenden wird zunächst der Versuchsaufbau näher erläutert, gefolgt von unseren Erwartungen des Versuchsablaufs. Dabei werden zwei Gruppen von Testpersonen jeweils ein UML-Klassendiagramm innerhalb des Versuchs mit Hilfe des Prototypen erstellen. Dabei sollen Beobachtungen schriftlich festgehalten werden und schlussendlich mit den im Testversuch bearbeiteten Fragebögen, ausgewertet und vorgestellt werden. Wobei zunächst die Gemeinsamkeiten beider Testgruppen aufgeführt werden, bevor die Besonderheiten beider Gruppen getrennt beschrieben werden. Abschließend werden allgemeine Meinungen der Versuchsteilnehmer gegenüber der Entwicklung am MTT sowie Resultate über den Nutzen der implementierten Konzepte zusammengefasst, die aus den Tests gewonnen werden konnten.

6.1 Der Versuchsaufbau

Zur Evaluierung des Prototypen sollen vor allem Kollaboration, der Umgang der vorgegebenen Gesten und Elemente zur Unterstützung des Entwicklungsprozesses von Klassendiagrammen durch den Prototyp von Benutzern getestet sowie bewertet werden. Beim Benutzertest haben wir uns für einen qualitativen Test entschieden, entgegengesetzt zu einem quantitativen Test, da vor allem der Nutzen und die Benutzung der eingesetzten Konzepte geprüft und diese ohne eine Meinung vorzugeben von den Testkandidaten bewertet werden sollen. Ein quantitativer Test ist allerdings nicht auszuschließen und lässt sich wegen Zeitgründen nicht mehr in dieser Arbeit ausführen. Dieser Test könnte zu einem späteren Zeitpunkt durchgeführt werden und weitere Kenntnisse erbringen.

Der Test soll mit zwei Gruppen bestehend aus jeweils drei Personen stattfinden. Dabei sollen die Testpersonen Kenntnisse der UML-Klassendiagramm-Entwicklung und -Umsetzung besitzen sowie Erfahrungen aus Projekten mit sich bringen. Nach einer kurzen Einleitung und Motivation, die unter anderem das Testen der Software und nicht der Kandidaten betonen sowie verdeutlichen soll, dass keine *Fehler* auf Benutzerseite entstehen können, sollen die Kandidaten zunächst in alle vorhandenen Gesten und Elemente eingeführt werden. Da die Anzahl der Gesten für den *Einstieg* erheblich ist und diese selbst nach der Einführung schnell vergessen werden können, gerade bei seltenem Einsatz von bestimmten Gesten, wird jeder Person eine Gestenübersicht ausgehändigt. Die verwendete Gestenübersicht kann im Anhang C „Anlagen des Benutzertests“ unter der Abbildung C.1

gefunden werden. Dabei soll der Nutzen der Elemente möglichst nicht vorgegeben werden, wie beispielsweise der genaue Nutzen des Marker-Konzepts, um zu prüfen ob diese von den Benutzern auch erkannt und wie gedacht angenommen werden oder nicht. Das Konzept der Aufgabenelemente wird als einziges Konzept genauer erklärt, da der Umgang mit diesen Elementen, und damit insbesondere die Zerlegung und Umwandlung zu Klassen, weiteres Verständnis und Hintergrundwissen erfordert.

Im nächsten Schritt soll die Aufgabenstellung erläutert werden. Dazu werden die Teilnehmer aufgefordert ein textuell vorgegebenes Modell eines Problembereichs, das ebenfalls im Anhang C „Anlagen des Benutzertests“ (Abb. C.2) zu finden ist, zu bearbeiten und am MTT gemeinsam zu einem Klassendiagramm zu entwickeln. Die Bearbeitung erfordert dabei den Nutzen der implementierten Konzepte. Diese sollen mit der vorgegebenen Aufgabenstellung durch Versuchsteilnehmer geprüft werden. Unsere Erwartung des Einsatzes dieser werden im Kapitel 6.2 „Erwartungen an den Testversuch“ weiter beschrieben.

Die Aufgabe dreht sich um die Modellierung eines Flughafens und ist aus Zeitgründen simpel gehalten und nicht sehr aufwändig. Die Aufgabenstellung setzt sich aus vier wesentlichen Bereichen zusammen. Dabei sollen die Testpersonen Rollen als Verantwortliche für *Gepäckorganisation*, *Gebäude* und *Personal* übernehmen, um Projektarbeiten nachzuempfinden und Zugehörigkeiten für Fachbereiche zu vermitteln. Der vierte Bereich der Aufgabenstellung wird absichtlich *nicht* als Rolle verteilt, um einen Konflikt in der Gruppe zu erzwingen und damit die Kommunikation, Absprache und Zusammenarbeit des Team zu provozieren.

Beim ersten Arbeiten am MTT sollen die Benutzer ihren Namen samt Rollenzuordnung eintragen, um anschließend die erläuterte Aufgabe zu bearbeiten. Des Weiteren soll die Aufgabe möglichst ohne fremde Hilfe bearbeitet und von gängigen Benutzertest-Methoden, wie *lautes Denken*, begleitet werden.

Da die Aufgabenstellung simpel gehalten ist, soll nach der Erstellung eines zugehörigen, ausgereiften Klassendiagramms die Aufgabenstellung erweitert werden. Die Testpersonen werden dazu aufgefordert ihr Modell beliebig zu einem *internationalen* Flughafen zu erweitern. Als Beispiel dazu wird die Einführung von Terminals für internationale Flugrouten genannt. Diese Aufgabenstellung soll den Benutzern ermöglichen Aufgaben für ein theoretisches nächstes Teamtreffen als auch zwischenzeitliches, externes Arbeiten zu formulieren und untereinander aufzuteilen. Dazu sollen die Kandidaten möglichst informelle Elemente nutzen und ihre Erweiterungen *nicht* in UML-Notation formulieren oder sogar ausarbeiten. Durch Beobachtungen der Ausführung dieses Aufgabenteils soll den Umgang der Aufgabenelemente und die Zuweisung der Verantwortlichkeiten innerhalb des gesamten Teams bewusst untersucht werden, da nach unseren Erwartungen Aufgabenelemente aufgrund der simplen Aufgabenstellung kaum genutzt werden.

Der Test wird mit einem Fragebogen beendet, der nicht ausgesprochene Gedanken und Meinungen der Testpersonen sowie entgangene Beobachtungen erfassen soll. Der Fragebogen kann ebenfalls im Anhang C „Anlagen des Benutzertests“ (Abb. C.3 ff.) eingesehen werden.

6.2 Erwartungen an den Testversuch

Für den zuvor beschriebenen Benutzertest werden folgend unsere Erwartungen bezüglich des Versuchsablaufs, der (gewollt) auftretenden Probleme und das zu erwartende Verhalten der Benutzer beschrieben. Diese Erwartungen beruhen auf Erfahrungen aus bisherigen, ähnlichen Aufgaben und aus der eigenen Bearbeitung der Versuchs-Aufgabenstellung, die zu zweit sowohl an einer großen Tafel als auch am Prototypen selbst durchgeführt wurde.

Aufgrund der prototypischen Implementierung können technische Probleme nicht ausgeschlossen werden, weswegen für den Versuch das Diagramm während der Bearbeitung im Hintergrund gesichert wird. Dies erfolgt automatisch in kurzen Zeitabständen.

Zudem erwarten wir Probleme bei der Ausführung der Gesten, da diese nur von einer Tischseite und in eine Richtung ausgeführt werden können. Weiterhin ist die Erkennung sowie Unterscheidung der Gesten im Prototypen nicht eindeutig falls diese ungenau gezeichnet werden. Weitere Schwierigkeiten werden erwartungsgemäß überlappende Elemente darstellen, wobei Eingabefelder blockiert bzw. Gesten nicht präzise ausgeführt werden könnten, neben kleineren Anzeigefehlern.

Bei der Erstellung des Modells des Problembereichs erwarten wir, dass die Versuchsteilnehmer zur Analyse der Aufgabenstellung und dem Zusammentragen von Informationen die Aufgabenelemente verwenden. Dazu zählt das Verbinden der Aufgabenelemente über Assoziationen, die Zuweisung von Verantwortlichen anhand der zugewiesenen Rollen der Aufgabenstellung und letztendlich das Umwandeln einer Aufgabe in eine Klasse, wenn diese ausreichend analysiert wurde. Andererseits rechnen wir ebenfalls damit, dass die Versuchsteilnehmer die Aufgabenstellung direkt in UML-Notation umsetzen, weil die gegebenen Informationen in der von uns gestellten Aufgabe simpel gehalten als auch gut aufbereitet sind und nicht durch die Testpersonen umständlich erarbeitet werden müssen. Daraus folgt, dass die Aufgabenelemente nicht in dem Umfang genutzt werden, wie es im dazugehörigen Konzept vorgesehen ist. Diese Erwartungshaltung lässt sich ebenfalls auf die Tatsache zurück führen, dass ein größeres Software-Projekt, in dem Informationen umständlich erarbeitet werden müssen, nicht in einem kleinen, zeitlich begrenzten Test simuliert werden kann. Bei der anschließenden Erweiterung der Aufgabenstellung gehen wir davon aus, dass die Nutzung der Aufgabenelemente und der Zuweisung von Verantwortlichen steigt, weil diese sich zum Notieren von Ideen und der Planung des weiteren Vorgehens besonders eignen.

Bei der Kommunikation zwischen den Versuchsteilnehmern erwarten wir, dass sie trotz der anfänglichen, selbstständigen Arbeit, gemäß den ihnen zugeteilten Rollen, sich bei Fragen und Problemen gegenseitig unterstützen. In der anschließenden, rollen-übergreifenden Gruppenarbeit wird, unserer Erwartung nach, eine enge Zusammenarbeit aller beteiligten stattfinden, die eine schnelle und dynamische Arbeitsverteilung aufzeigen wird. Dabei werden Tätigkeiten, wie das Lesen der Aufgabenstellung, die Erstellung und Bearbeitung von Diagrammelementen

und das Verbinden dieser mit Hilfe von Assoziationen, fließend und ohne großen Kommunikationsaufwand zwischen den Teilnehmenden gewechselt. Diese Erwartung beruht auf der für alle Beteiligten einsehbaren Arbeitsfläche, auf der alle aktuellen Tätigkeiten und Elemente wahrnehmbar sind und alle Elemente von jedem bearbeitet werden können. Zudem können die genannten Tätigkeiten von verschiedenen Personen, kollaborativ und parallel ausgeführt werden.

Beim Verschieben bzw. Anordnen von Diagrammelementen rechnen wir damit, dass wenn ein Element am MTT erstellt werden soll angrenzende Elemente bei Bedarf beiseite geschoben werden, um einem neuen Element Platz zu machen, weil kein aufwendiges Vorausplanen der Anordnung wie beim Modellieren an einer Tafel oder auf Papier nötig ist. Im Falle einer Unübersichtlichkeit im Klassendiagramm rechnen wir damit, dass Elemente durch einen oder mehrere Teilnehmer neu anordnet werden, um die Übersicht wiederherzustellen. Erwartungsgemäß werden die Testpersonen sich am Multi-Touch-Tisch von Zeit zu Zeit bei der Bearbeitung gegenseitig behindern, weswegen sie Elemente kurzzeitig verschieben oder dieses Problem kommunikativ bewältigen müssen.

Da alle Elemente dieser Aufgabenstellung bei guter Strukturierung ohne Probleme auf dem Tisch dargestellt werden können, erwarten wir, dass die Versuchsteilnehmer das gesamte Diagramm nicht oder kaum verschieben, skalieren sowie drehen werden. Bei der Verteilung der Personen um den Multi-Touch-Tisch gehen wir davon aus, dass die Versuchsteilnehmer hauptsächlich von einer Tischseite arbeiten werden. Wobei Positionen an den Tischseiten genutzt werden, wenn Diagrammelemente nicht erreichbar sind oder der Arbeitsplatz nicht ausreicht.

Weiterhin erwarten wir keine fachlichen Schwierigkeiten bei der Aufgabenstellung sowie der UML-Notation, da die Versuchsteilnehmer bewusst mit entsprechendem Fachwissen ausgesucht worden sind. Diese werden voraussichtlich Kommentare in Aufgabenelementen und Klassen während der erweiterten Aufgabenstellung nutzen. Die Testpersonen werden Marker erwartungsgemäß ohne eine Erklärung des Konzepts intuitiv einsetzen bzw. den Sinn dieser selbstständig erkennen sobald die Kandidaten ihre Lösung auf Vollständigkeit gegenüber der Aufgabenstellung überprüfen wollen.

Obwohl Assoziationen, Aufgabenelemente und Klassen zwei verschiedene Modi (Ansichts- und Bearbeitungsmodus) haben, rechnen wir nur bei den Assoziationen mit der Nutzung von Beiden. Bei Aufgabenelementen und Klassen wird der Ansichtsmodus, unserer Erwartung nach, nicht zum Einsatz kommen, weil die gestellte Aufgabe nicht genügend Komplexität mit sich bringt. Dadurch bleibt die Möglichkeit für eine übersichtliche Bearbeitung dieser während der gesamten Entwicklung bestehen, sodass ein Wechsel in den Ansichstmodus nicht nötig ist. Bei Assoziationen werden nicht ausgefüllte Eingabefelder, wie Kardinalitäten und Rollen, im Ansichtsmodus verborgen, weswegen Assoziationen in diesem Fall übersichtlicher sind. Versuchsteilnehmer werden erwartungsgemäß vor allem bei mehreren, sich überschneidenden Assoziationen von dieser Möglichkeit Gebrauch machen. Wir rechnen nicht damit, dass das Ein- und Ausklappen von Klassen genutzt wird, da alle Elemente dieser Aufgabestellung trotz ausgeklappter Klassen

übersichtlich dargestellt werden können.

6.3 Verlauf und Beobachtungen des Versuchs

Im Folgenden werden zunächst Beobachtungen und Ergebnisse der Fragebögen aufgeführt, die in beiden Testgruppen festgestellt wurden. Anschließend werden beide Gruppen getrennt betrachtet, wobei Unterschiede zwischen den Gruppen herausgestellt werden.

6.3.1 Allgemeine Probleme und Beobachtungen

Beide Testgruppen haben innerhalb des Versuchs ein erfolgreiches UML-Klassendiagramm der vorgegebenen Problemstellung mit Hilfe des Prototypen erstellen können. Alle Testpersonen brachten durch ihr fortgeschrittenes Informatik-Studium reichlich an UML-Kenntnissen sowie Erfahrungen durch standardisierte Gestensteuerungen anderer Produkte mit sich. Dennoch fehlten einige UML-Kenntnisse, wie die Unterscheidung zwischen Aggregation und Komposition, wodurch einige Schwierigkeiten während der Modellierung auftraten, die aber durch Aushelfen von anderen Teilnehmern und der guten Teamkommunikation am MTT gelöst werden konnten. Die aus den Beobachtungen und Fragebögen entnommenen Ergebnisse jeder Gruppe spiegelten sich größtenteils in unseren Erwartungen wieder. Die Ergebnisse beider Gruppen gleichen sich zum Teil und viele Probleme wurden jeweils von *Beiden* aufgedeckt. Dabei kam es teilweise zu unerwarteten Problemen und Abläufen innerhalb der Versuchsgruppen.

Wie von uns erwartet positionierten sich die Teilnehmer zunächst an einer Tischseite und wichen nach einiger Zeit immer wieder selbständig zu den Seiten aus. Weil nach einem fortgeschrittenen Entwicklungsstand erreichbare Arbeitsflächen bereits belegt waren, mussten die Teilnehmer Elemente an weniger erreichbaren Stellen bearbeiten. So wurde vereinzelt selbst das Über-Kopf-Arbeiten und -Lesen in Kauf genommen, um an der gegenüberliegenden Seite zu arbeiten. Insgesamt war bei beiden Gruppen eine dynamische Arbeitsverteilung und Positionierung am Tisch, spätestens nach der beabsichtigten eigenständigen Arbeitsphase, sichtbar, wobei insbesondere zur Zeit nicht aktiv am Tisch arbeitende Versuchsteilnehmer zurück traten, um ihren Platz den weiteren Personen anzubieten. Aus den Fragebögen heraus, wurde die Einschränkung von einer Tischseite arbeiten zu müssen als negativ aufgefasst und eine Gestenerkennung aus allen Richtungen vorgeschlagen, um den vorhandenen Platz besser nutzen zu können. Von beiden Gruppen wurde, den Fragebögen nach, der Wunsch geäußert Elemente nicht nur abrupt zu verschieben sondern auch in einer Art zu „werfen“, damit diese selbstständig an eine gewünschte Position oder zu einer Person „gleiten“. Dadurch können Elemente an schwer erreichbare Teammitglieder übergeben oder an schwer erreichbaren Stellen grob positioniert werden.

Der Modus um das gesamte Diagramm zu verschieben, rotieren oder zu skalieren wurde kaum genutzt. Wurde dieser Modus genutzt, traten wie erwartet Konflikte aufgrund von Kommunikationsproblemen auf. Diese Kommunikationsprobleme entstanden dadurch, dass weitere Teilnehmer noch mit eigenen Aufgaben beschäftigt waren, welche ihre Aufmerksamkeit in Anspruch nahmen und somit auf ein „Finger einmal weg“ vom Initiator nicht reagierten bzw. ihre eigenen Aktionen unbedingt abschließen wollten und diesen Aufruf ignorierten. Bei Platzmangel wurde diese Funktion genutzt, um das Diagramm zu verkleinern und damit mehr Arbeitsfläche zu schaffen. Ein Verbesserungsvorschlag aus einem Fragebogen schlägt die Umsetzung einer automatischen Anpassung des gesamten Diagramms am Verhältnis von Anzahl aller Elemente zu Größe der Arbeitsfläche vor. Nach einigen Bearbeitungen des gesamten Diagramms durch den vorgegeben Modus, wurde von beiden Gruppen der Vorschlag geäußert, dass eine Funktion zum „Normalisieren“ bzw. Zurücksetzen der Ausrichtungen von Elementen bereitgestellt werden sollte. Bei den einzelnen Diagrammelementen wurde der Bearbeitungsmodus von beiden Gruppen kaum verlassen. Zudem war der aktuelle Modus (Bearbeitungs- oder Ansichtsmodus) für Benutzer bei einzelnen Elementen schlecht erkennbar und beim gesamten Diagramm gar nicht zu erkennen, welches zu Verwirrung führte. Bei diesem Problem baten die Teilnehmer um geeignetes Feedback durch die Anwendung.

Bei den Gesten wurden die meisten Probleme durch eine ungenaue Gestenerkennung verursacht, sodass Gesten mehrfach ausgeführt werden mussten um eine gewünschte Aktion auszulösen. Dazu kamen Anfangsschwierigkeiten aufgrund fehlender Erfahrung im Umgang mit dem Prototypen und der Gestensteuerung. Es wurde mehrfach kritisiert, dass die Kreis-Geste von der Rechteck-Geste nicht ausreichend unterschieden wurde, weshalb oftmals ein falsches Element erstellt worden ist. Weiterhin gaben beide Gruppen in den Fragebögen an, dass sie die Rechteck-Geste sowie das Verschieben, Rotieren Skalieren und Löschen von Elementen als intuitiv empfanden. Assoziationen mit einem Double-Tap zu erstellen wurde allerdings als schwierig bezeichnet und aus diesem Grund von einem Teilnehmer als nicht intuitiv gekennzeichnet. Zudem ist in unseren Beobachtungen aufgefallen, dass einige Teilnehmer das visuelle Feedback beim Erstellen einer Assoziationen nicht wahrgenommen haben. Dabei wird die ausgeführte Geste wie alle anderen Gesten visualisiert, wobei anstelle einer zufälligen Farbe, eine feste, graue Farbe verwendet wird. Dadurch kann bereits beim Ziehen einer Assoziation vom Anwender erkannt werden, ob die Geste korrekt erkannt bzw. richtig ausgeführt worden ist. Dies wurde von uns jedoch nicht explizit erwähnt. Die Erkennungsschwierigkeiten legten sich nach einer kurzen Einarbeitungszeit und mit steigender Erfahrung im Umgang mit dem Prototypen. Bei der Kreisauswahl des Assoziationstypen verdeckte der eigene Finger der Testperson einige Auswahlmöglichkeiten, sodass die Personen um den Finger herum lesen mussten um alle Typen zu erkennen. Dieses Problem wurde schon während der Implementierungsphase festgestellt, ließ sich aber aus technischen Gründen nicht besser realisieren. Der Benutzertest bestätigte dieses Problem. Die Auswahl der Assoziationsenden wurde von den

Befragten dennoch als intuitiv und simpel eingeordnet.

Eines der größeren Probleme das während der Nutzung des Prototypen auftrat, war der Umgang mit den virtuellen Tastaturen. Dabei überlappten sich die Tastaturen der Teilnehmer öfters gegenseitig und erschwerten die gemeinsame Arbeit erheblich. Bei Assoziationen wurde die Tastatur parallel zur Assoziations-Linie erzeugt, die ohne Drehen durch den Anwender weiter benutzt wurde. Das Verschieben der Tastaturen wurde von den Benutzern als intuitiv wahrgenommen. Allerdings wurden die Funktionen zum Drehen und Skalieren der Tastaturen erst auf direkte Ansprache erkannt, woraufhin diese effektiv genutzt wurden. Mit dieser gewonnen Erkenntnis wurde das Überlappen von Tastaturen durch die Benutzer selbständig minimiert und auch durch Assoziationen geöffnete Tastaturen wurden gedreht. Von einer Testperson wurde weiterhin der Wunsch geäußert, dass pro Benutzer eine feste virtuelle Tastatur zur Verfügung gestellt werden sollte.

Ebenfalls erwartungsgemäß wurden Aufgabenelemente für den ersten und simplen Teil der Aufgabenstellung kaum genutzt. In der erweiterten Aufgabenstellung wurden Aufgabenelemente wie gewollt vermehrt eingesetzt. Aus den Fragebögen der Teilnehmer hat sich dennoch ergeben, dass sechs von insgesamt sieben Teilnehmern die Aufgabenelemente als hilfreich für die Planung und Aufgabenzuteilung empfinden bzw. diese als hilfreich für größere Projekte sehen. Eine Person bezeichnete die Verwendung dieser Elemente für die „nächste Phase“ als „interessant“. Ebenfalls wurde das Teilen und Umwandeln von Aufgaben nicht effektiv genutzt, da das Testszenario zu begrenzt war.

Kommentarfelder von Klassen wurden von den Benutzern selbst *nicht* produktiv eingesetzt bzw. nach Angabe der Versuchsteilnehmer sogar übersehen. Im Fragebogen wurde von den Teilnehmern jedoch angegeben, dass diese Kommentarfelder für größere und komplexere Aufgaben als „nützlich“ empfinden. Der fehlende Einsatz wurde von ihnen mit der einfachen Aufgabenstellung begründet, in der Kommentare unnötig waren.

In beiden Gruppen wurden die Besprechungs-Marker korrekt eingesetzt, wenn auch erst zum Ende der Aufgabenbearbeitung, als die Gruppen auf das Problem der Konsistenz zwischen Modellierung und Aufgabenstellung gestoßen sind. Zuvor wurden diese bei anfänglichen „Erkundungen“ der Funktionen miss-interpretiert. So wurde das rote X des Markers von einer Person als Löschen interpretiert oder der Hacken ohne Teamabsprache als eigene Markierung für „Ist nun fertig für mich“ eingesetzt. Nach Auftreten des Problems, ob nun alle Elemente ordnungsgemäß für *alle* Mitglieder modelliert wurden, war das Konzept für alle Teilnehmer klar.

6.3.2 Besonderheiten der ersten Benutzergruppe

Der erste Benutzertest wurde von drei Teilnehmern durchgeführt. Nach einer Einleitung und einer sehr kurzen Einführung der Gesten wurden die Teilnehmer ihrer Aufgabe überlassen. Durch die knappe Einführung der Gesten entstanden trotz mitgelieferter Hilfestellung durch die Gestenübersicht (Anhang C.1) fortlaufend

Schwierigkeiten und Probleme, die die richtige Ausführung und den Überblick der vorhandenen Gesten betreffen.

Bei der Beobachtung der Kandidaten konnte festgestellt werden, dass Eingabefelder, sowie deren Hinweistexte, nicht als solche erkannt wurden. Dies ging besonders bei den Assoziationen hervor, die einen transparenten Hintergrund besitzen. Dieses Problem wurde von den Testpersonen allerdings schnell überwunden, sodass dies nur anfänglich auftrat. Bei der anfänglichen Einzelbearbeitung, gemäß der Rollen der Aufgabenstellung, wurden die Anzeigen der Verantwortlichkeit nur von einem Teilnehmer für seine Elemente konsequent benutzt. Nach einiger Zeit allerdings, sowie bei einem Blick auf die Elemente seiner Kollegen, forderte dieser Teilnehmer seine Teammitglieder auf, Verantwortlichkeiten einzustellen. Folgend wurden diese vom gesamten Team durchgehend genutzt. Laut Fragebogen haben die Versuchsteilnehmer die Verantwortlichkeiten-Anzeige genutzt um den Ersteller des Elements zu kennzeichnen. Ein Teilnehmer kam letztendlich eigenständig zum Schluss, dass mit dieser Anzeige besser Verantwortlichkeiten markiert werden sollten. In der erweiterten Aufgabenstellung wurden diese zusätzlich zur Zuweisung von Aufgaben benutzt. Nach dem Fragebogen wurden die Marker von der ersten Gruppe als „unbenutzbar“ bezeichnet, da diese erhebliche Probleme mit der Gestensteuerung zum Wechseln der Marker hatten. Welches nach einem Versuchsteilnehmer an den für einen Finger zu kleinen Markern lag. Der Sinn der Marker wurde allerdings als „gut“ betitelt, wobei die Benutzer den *Hacken-Marker* als Kennzeichnung von fertigen Elementen und den *Fragezeichen-Marker* als „In Bearbeitung“ interpretierten. Entgegen unseren Erwartungen, wurde bei Assoziationen der Bearbeitungsmodus ebenfalls nur selten verlassen. Dadurch wurde das Diagramm bei zunehmender Anzahl von Assoziationen unübersichtlich. Lediglich bei der Erzeugung von Vererbungen wurde sofort zum Ansichtsmodus gewechselt. Unserer Erkenntnis nach könnte dies auf der Tatsache beruhen, dass Vererbungen aufgrund fehlender Angaben von Kardinalität, Rollen und Bezeichnern keine weiteren Bearbeitungen benötigen. Bei Restriktionen wurde zudem von den Teilnehmern eine Auswahl gewünscht, wie zum Beispiel `{XOR}`. Aus den Fragebögen konnte entnommen werden, dass die Kandidaten das Erstellen, Verschieben und Löschen „OK“ fanden. Ein Teilnehmer äußerte, dass die „M“-Geste, die das Menü öffnet, und die Rechteck-Geste einen zu großen bzw. komplexen Gestenweg verlangen. Seiner Meinung nach können diese nur langsam ausgeführt werden, wodurch der Arbeitsfluss gestört wird. Dennoch gab dieser an, dass die Rechteck-Geste sehr intuitiv gestaltet ist. Weiterhin meinte der Versuchsteilnehmer, dass ähnliche Gesten nicht unterschiedliche Funktionen ausführen sollten, wie zum Beispiel das Teilen von Aufgabenelementen mit einem Strich und das Löschen von Elementen mit zwei parallelen Strichen durch diese. Seiner Erkenntnis nach, ist ein Double-Tap unüblich für Multi-Touch-Anwendungen und nicht intuitiv. Bei der *erweiterten* Aufgabenstellung wurden, wie gewollt, vermehrt Aufgabenelemente genutzt und Verantwortlichkeiten zugeordnet. Es fehlte jedoch eine ausführliche Kommunikation innerhalb des Teams. Dabei wurden bei der ersten Gruppe Aufgaben gegenseitig zugewiesen, statt im Team Aufgaben zu bestimmen

und anschließend eine geeignete Person dieser Aufgabe zuzuordnen. Des Weiteren wurden Aufgabenelemente nur erzeugt und nicht mit Assoziationen in das vorhandene Klassendiagramm integriert bzw. eingeordnet. In den Fragebögen gaben die Kandidaten an, dass mehr über Probleme und Schwierigkeiten mit der Gestensteuerung kommuniziert wurde als über die eigentliche Aufgabenstellung. Allerdings wurden innerhalb des Teams gegenseitig Hilfestellungen beim Umgang mit dem MTT gegeben. Dies wurde als positive Kommunikation am Tisch angegeben. Während der Beobachtung ist aufgefallen, dass diese Gruppe ihr Klassendiagramm wiederholt Stück für Stück neu und übersichtlicher angeordnet hat, anstatt dies nur bei komplett fehlender Übersichtlichkeit zu tun. Nach Abschluss der Bearbeitung wurde das erstellte Diagramm zusätzlich strukturiert, wobei alle Tischseiten von den Teilnehmern zur Diskussion, über die Korrektheit des Diagramms, genutzt wurden. Eine Person schlug vor das Hauptmenü-Popup, welches über den gesamten Tisch dargestellt wird, kleiner anzuzeigen oder eine Art Splittscreen einzuführen, sodass Alle weiter arbeiten können während ein Benutzer zum Beispiel das Dokument abspeichert. Von einem weiteren Benutzer wurde angegeben, dass dieser den Moduswechsel von Elementen nicht genutzt habe, da er alle Optionen sehen wollte. Von einem anderen wiederum, weil dieser die Funktion vergessen hatte.

Als negativen Punkt gab die Gruppe die Gestenerkennung an. Dabei könnten ihrer Meinung nach einige Gesten durch einfache Klicks bzw. Touchpoints ersetzt werden, welches ein schnelleres und besseres Erkennen zur Folge hätte. Als Positiv wurde die vereinfachte Kommunikation im Team und der Funktionsumfang des Prototypen angegeben sowie die Benutzererstellung mit der Farbauswahl. Dabei wurde auch das sehr schnelle Erstellen von Elementen betont, wenn Benutzer bereits Erfahrungen in der Ausführung der Gesten hatten.

6.3.3 Besonderheiten der zweiten Benutzergruppe

Die zweite Testgruppe bestand aus vier anstatt drei Versuchsteilnehmern, die jedoch den drei aufgelisteten Rollen zugewiesen wurden. Dabei bildeten zwei Personen eine Kleingruppe mit einer Rolle. Zudem wurden in der Einweisung *dieser* Gruppe die Gesten ausführlicher erklärt sowie am Multi-Touch-Tisch vorgeführt, wobei die Versuchsteilnehmer die Gesten selbst ausprobieren durften und bei Schwierigkeiten Hinweise unsererseits bekamen. Außerdem konnten dadurch Grenzen der Gesten und der Implementierung getestet werden, indem beispielsweise Gesten besonders schnell und unpräzise von den Teilnehmern ausgeführt wurden. Dadurch konnten viele Schwierigkeiten der ersten Versuchsgruppe, die mit dem Umgang des Prototypen zusammenhingen sowie das spielerische Ausprobieren des „Touch-Erlebnisses“ und einzelner Elemente vor der Aufgabenbearbeitung vermieden werden. Wodurch die Bearbeitung der Aufgabenstellung zielorientierter verlief. Dieses ließ sich vor allem bei der Nutzung der Gesten und virtuellen Tastaturen beobachten, wobei Letztere sogar mit *einer* Hand skaliert und gedreht worden sind.

Während der Auswertung der aufgezeichneten Versuche ist aufgefallen, dass im Gegensatz zur ersten getesteten Gruppe, die Versuchsteilnehmer die eingeblendeten Tastaturen zu sich zogen und passend skalierten bzw. drehten. Dadurch konnten die Personen an einer Tischseite besser gleichzeitig arbeiten. Laut Aussagen der Beteiligten im Fragebogen, ist das gleichzeitige kollaborative Arbeiten, für vier Personen an einer Tischseite, leicht eingeschränkt, aber unter Rücksichtnahme möglich. Um die Arbeit zu koordinieren hat sich die Gruppe vor Bearbeitung der Aufgabenstellung über ihre Vorgehensweise abgesprochen, zusätzlich zur Verteilung der vorgegebenen Rollen. Weil zwei Testpersonen einer Rolle zugewiesen waren und die Aufgaben gemeinsam bearbeitet haben, glich dies der Paar-Programmierung mit dazugehöriger Kommunikation. Des Weiteren konnten beide Personen den Tisch bedienen, wodurch diese während des Versuchs beispielsweise ein Element gleichzeitig bearbeiteten und verbinden konnten. Bei dieser Kleingruppe war eine schnelle und dynamische Arbeitsverteilung *ohne große Absprachen* sichtbar. Dies wurde von der Kleingruppe deutlich besser gehandhabt als von der gesamten Gruppe bei der Bearbeitung der erweiterten Aufgabenstellung. Sobald Teilnehmer ins Stocken gerieten, konnte in dieser Versuchsgruppe beobachtet werden, dass weitere Versuchsteilnehmer selbständig eingriffen, um Hilfestellungen und Tipps zu geben. Auffällig war dies besonders durch die Gruppengröße dieser Versuchsgruppe. Aufgrund eines Missverständnisses während der Rollenverteilung und einer entstandenen Unübersichtlichkeit im Diagramm, wie es im Fragebogen angegeben wurde, modellierten zwei Testpersonen jeweils am linken und rechten Tischrand dasselbe Teilproblem, wobei dies zunächst niemandem auffiel. Nach dem die Mehrfacherstellung erkannt wurde, verglichen beide Beteiligten ihre Ergebnisse und erstellten nach Auswahl der besten Teillösungen eine gemeinsame Lösung.

Da die Gruppe aus vier Personen bestand, die an einer Tischseite arbeiten mussten, verließen diese schnell ihre Positionen und wichen zu den Seiten aus. Im späteren Verlauf wurde ebenfalls die gegenüberliegende Seite benutzt, trotz suboptimalem Arbeiten. Der Moduswechsel zwischen Ansichts- und Bearbeitungsmodus erfolgte zunächst unbewusst bzw. versehentlich und sorgte bei einigen Versuchsteilnehmern für Verwirrung. Im Verlauf des Versuchs wurde das Klassendiagramm der Testgruppe neu geladen, weswegen alle Diagrammelemente im Ansichtsmodus dargestellt wurden. Zu diesem Zeitpunkt wurde der Ansichtsmodus beibehalten, weil viele Klassen und Assoziationen fertiggestellt waren und keiner weiteren Bearbeitung bedurften. In einigen Fällen wurden Klassen eingeklappt, wenn die Beteiligten diese als abgeschlossen ansahen. War das nicht der Fall, wurde der Bearbeitungsmodus der Klasse bzw. Assoziation bewusst wieder aktiviert. Lediglich ein Versuchsteilnehmer nutzte Kommentare für *OCLs*, die sich in der prototypischen Implementierung nicht modellieren ließen, und textuelle Beschreibungen von Modellierungen für deren Umsetzung ihm laut Fragebogen UML-Wissen fehlte. Des Weiteren ist den Teilnehmern aufgefallen, dass Aufgabenelemente „in der Luft hängen“, wie es eine Testperson ausdrückte, und dass Verbindungen von diesen zum restlichen Klassendiagramm sinnvoll wären. Nach einem Hinweis unse-

rerseits, dass dies möglich ist, wurden Assoziationen zwischen Aufgabenelementen und dem restlichen Diagramm gezogen, um die gewünschten Verbindungen auszudrücken. Bei Assoziationen wurde eine Versuchsperson sichtlich verwirrt, als diese eine Verbindung zwischen zwei Klassen ziehen wollte, stattdessen versehentlich eine Assoziation mit einer Klasse verbunden hat und dadurch eine n-äre Assoziation entstand. Als der Versuchsteilnehmer daraufhin hingewiesen wurde, dass es sich um eine n-äre Assoziation handelte, wurde das Problem kollaborativ vom Team selbst gelöst, indem diese Verbindung gelöscht und nochmals richtig gezogen wurde. In einer anderen Situation wollte eine Testperson eine Assoziation von der linken zur rechten Tischseite ziehen. Weil dabei die restlichen Teilnehmer behindert worden wären, ist die Testperson an der gegenüberliegenden Seite um den Tisch gegangen während diese die Assoziation gezogen hat. Dies war problemlos möglich. Weiterhin wurde im Fragebogen zwar angegeben, dass alle Gesten, bis auf den Double-Tap der Erstellungsgeste von Assoziationen, intuitiv sind, jedoch wurde bemängelt, dass Marker wechseln, Attribute löschen und das Löschen von Elementen durch die Gesten problematisch war. Unseren Beobachtungen nach war dies hauptsächlich auf die teilweise ungenaue Gestenerkennung des MTTs zurückzuführen. Die Geste zum Erstellen von Assoziationen wurde nicht als intuitiv bezeichnet, aber als schnell erlernbar und dennoch von den Teilnehmern als passende Geste zum Verbinden von Elementen genannt. Weitere Vorschläge der Teilnehmer für Gesten waren ein Double-Tap zum Wechseln von Markern, ein Zwei-Finger-Double-Tap zum Skalieren, Verschieben bzw. Platz-Schaffen und ein Fünf-Finger-Pinch (Zusammenziehen der Finger wie beim Skalieren bzw. Zoomen) um beispielsweise das Hauptmenü zu öffnen. Das Konzept der Verantwortlichkeits-Anzeige wurde von dieser Gruppe als Be- und Ausarbeitungszuständigkeit direkt richtig interpretiert. Ein Teilnehmer schrieb im Fragebogen als Stellungnahme zu diesem Element: „Macht Sinn und finde ich gut!“. Da drei Personen des Teams aus einer bestehenden Projektgruppe stammen, wovon einer der Teamleiter war, hat dieser auch im Test selbständig eine Führungsrolle eingenommen. Dadurch wurde die erweiterte Aufgabenstellung organisiert bearbeitet. Anstatt wie in der ersten Gruppe nur Aufgaben gegenseitig zu zuweisen, wurde zuerst ein Brainstorming gehalten um erweiterte Aufgaben zu erstellen, die anschließend sinnvoll verteilt wurden. Dabei wurden Elemente, Texte und Assoziationen von verschiedenen Teilnehmern simultan erstellt, wodurch Arbeitsschritte stark beschleunigt wurden. Während der Bearbeitung der Aufgabenstellung äußerten die Versuchsteilnehmer Vorschläge um Probleme komfortabler lösen zu können. So war ein gleichzeitiges, gruppiertes Verschieben von mehreren Elementen gewünscht, um ganze Strukturen von Elementen zu verschieben. Weiterhin bestand der Wunsch Elemente miteinander verschmelzen zu lassen, wenn beispielsweise versehentlich ein Element doppelt erstellt worden war. Bei den Assoziationen wurde vorgeschlagen, dass nach einer bestimmten Anzahl von Assoziationen, die an einer Stelle erstellt wurden, nur eine feste Anzahl dieser im Bearbeitungsmodus bleiben darf und alle Weiteren automatisch zum Ansichtsmodus wechseln. Dadurch wird ein bestimmter Grad an Übersichtlichkeit sichergestellt.

6.4 Resultate der Benutzertests

Beide Benutzertests verliefen sehr positiv und erzielten ein der Aufgabenstellung entsprechendes vollständiges Klassendiagramm. Trotz der beobachteten Schwierigkeiten waren beide Gruppen sehr zufrieden und begeistert von der Entwickelung eines Klassendiagramms am Multi-Touch-Tisch. Die Versuchsteilnehmer würden den entwickelten Prototypen anderen Medien bevorzugen, wenn diese in einem Team arbeiten müssten und die Gestenerkennung verbessert wird bzw. der Prototyp einen ausgereiften Entwicklungsstand erreicht hat. Weiterhin wurde von den Teilnehmern ausgesagt, dass diese den MTT für *alle* UML-Arten bevorzugen würden. Im Vergleich zu anderen von den Teilnehmern bereits eingesetzten Medien, wie ein Blatt Papier oder eine Tafel, wurde von den Teilnehmern ausgesagt, dass das Arbeiten am MTT deutliche Vorteile mit sich bringt. Gegenüber PC-Anwendungen wurde wie bereits durch eigene Tests festgestellt, dass die Navigation in Diagrammen am MTT viel intuitiver und schneller ist. Des Weiteren bezeichneten die Testpersonen das Entwickeln am Tisch als leichter und mit kurzer Einarbeitungszeit intuitiver als am Computer. Dabei zählten die Testpersonen UML-Editoren für den Vergleich auf, mit denen sie bereits Erfahrungen sammeln konnten. Die größere Arbeitsfläche, jederzeit und ohne großen Aufwand ausführbaren Bearbeitungsmöglichkeiten, die verbesserte Übersicht über laufende bzw. abgeschlossene Bearbeitungen von anderen Teilnehmern, dass (Zwischen-)Ergebnisse leicht *digital* gesichert werden können und besonders das aktive sowie inaktive *Zusammenarbeiten* am Tisch wurden als Vorteile genannt. Allerdings wird, laut den Testpersonen, nicht nur der Arbeitsfluss verbessert, sondern auch die Kommunikation sowie das Teamwork untereinander, welche erst durch das gemeinsame Besprechen und Bearbeiten an einem Tisch entstehen bzw. deutlich verbessert werden. Zudem zeigten die Tests, auch anhand der Fragebögen, dass die Kandidaten sehr animiert in der Ausübung ihrer Aufgabe waren. Dies könnte zum größten Teil an dem für viele Testpersonen neuen Medium, dem Multi-Touch-Tisch, liegen, wodurch das Arbeiten mit den teilweise spielerischen Gesten motivierend wirkte.

Die Konzepte, die in der prototypischen Implementierung zum Einsatz kamen und durch den Benutzertest geprüft werden sollten, wurden nach anfänglichen Schwierigkeiten und kurzer Einarbeitungszeit in die Editor-Software von den Testpersonen angenommen. Dazu zählen die im Prototypen hauptsächlich verwendeten Konzepte 3.2.1 „Aufgaben und Problemstellungen visualisieren“, 3.4.1 „Visuelle Marker für Besprechungsfortschritte“, 3.4.2 „Kommentarfunktion“ und 3.1.4 „Zuweisung von Verantwortlichkeiten“. Diese Konzepte wurden überwiegend erst beim Auftreten eines Problems, für dessen Problemlösung diese entwickelt wurden, richtig interpretiert und anschließend im Sinne der Konzipierung benutzt. So wurden die Marker anfänglich von einigen Personen entgegen der gedachten Verwendung zum Festhalten des Besprechungszustandes eingesetzt, wie in den Beobachtungen beschrieben ist. Sobald eine Gruppe jedoch auf das Problem stieß, die Konsistenz zwischen Modellierung und Aufgabenstellung im Team zu prüfen, wurde das Konzept von dieser als Hilfsmittel sofort erkannt und wie geplant ver-

wendet. Bei der Verantwortlichen-Anzeige wurde das Konzept spätestens dann erfolgreich erkannt als die Testkandidaten Tätigkeiten für eigenständiges externes Arbeiten sowie ein darauf folgendes Teamtreffen, in der erweiterten Aufgabenstellung, planen und zuteilen mussten. Zur Planung nutzten die Beteiligten, wie von uns vorgesehen, die Aufgabenelemente sowie die Kommentarfelder der Klassen. Das Teilen und Umwandeln von Aufgabenelementen wurde kaum genutzt, welches auf den zeitlich begrenzten Versuch und die größtenteils vorgegebenen Informationen der Aufgabenstellung zurückzuführen ist und von uns erwartet wurde. Ebenfalls ließ sich eine positive Resonanz über die implementierten Konzepte aus den Fragebögen entnehmen, in denen die Verwendung dieser Funktionen als hilfreich und sinnvoll bezeichnet wird, wenn auch erst für spätere Projektphasen. Um dieses Konzept in vollem Umfang prüfen zu können, müsste ein größerer Benutzertest durchgeführt werden. In diesem sollten einzelne Projektphasen simuliert werden, sodass Ideen bzw. Aufgaben schrittweise und selbständig von den Teilnehmern entwickelt sowie abwechselnd in Einzel- und Teamarbeit weitere anstehende Aufgaben geplant und Vorhandene ausgebaut werden. Die Konzepte zur kollaborativen und benutzerfreundlichen Bearbeitung von UML-Elementen, das Anmelden am MTT sowie das Speichern und Laden der Klassendiagramme wurden im Prototypen nur eingeschränkt implementiert, um Benutzern vorrangig das Erstellen eines vollständigen bzw. ausgereiften Klassendiagramms zu ermöglichen. Daher liefert der Benutzertest für diese Konzepte keine genauen und aussagekräftigen Ergebnisse.

Größere aber konzept-unabhängige Probleme entstanden durch prototypische Umsetzungen. So wurden Modi für Elemente und das Diagramm eingeführt, um Kollisionen von Gesten zu umgehen und für die Implementierung zu komplexe Gesten außen vor zu lassen. Dies betrifft einerseits den Bearbeitungs- bzw. Ansichtsmodus von Elementen sowie den Modus zum Verschieben, Rotieren und Skalieren des Diagramms. Dabei wurde der *aktuelle* Modus nicht eindeutig durch grafische Mittel hervorgehoben, sodass Benutzer verwirrt und orientierungslos waren. Weiterhin wurde der Bearbeitungsmodus von Elementen nur selten verlassen, da Benutzer alle Bearbeitungsoptionen vor Augen haben wollten und den Modus nur dann gewechselt haben, wenn sie sich sicher waren, dass an diesen Elementen keine weiteren Änderungen bzw. Eingaben stattfinden werden. Dies war beispielsweise bei der Erstellung von Vererbungs-Assoziationen der Fall. Da die Einführung der Modi eine schlechte Benutzerfreundlichkeit ergab, sollte die Verwendung dieser möglichst verringert und bei einer Verwendung ein geeignetes Feedback bereitgestellt werden. Durch die Einführung bzw. Implementierung sinnvoller Gesten über eine prototypische Version hinaus, sollten Kollisionen bei der Belegung von Gesten und Aktionen verringert werden, um die aus dem Prototypen heraus entstandenen Modi zu entfernen. Feedback wird ebenfalls beim Ausführen der Speichern- bzw. Laden-Funktionalität sowie beim Anlegen von Benutzerprofilen von den Versuchsteilnehmern verlangt, worauf aus Zeit- und Aufwandsgründen im Prototypen verzichtet worden war.

Ebenfalls negativ sind die vorgegeben virtuellen Tastaturen aufgefallen, deren

Rotierungs- und Skalierungs-Funktion von Nutzern nicht erkannt wurde und die besonders bei Assoziationen mit Sonderzeichen als Eingabe Probleme bereiteten. Eines dieser Benutzerprobleme stellte die Benutzung der Umschalt- bzw. Feststell-Taste dar. Diese wurden zunächst nicht auf der virtuellen Tastatur gefunden bzw. als solche Tasten interpretiert, trotz einer an physikalischen Tastaturen orientierten Tastenanordnung und -symbolik. Dadurch verlangsamten sich Eingaben bei Kardinalitäten mit einem Stern oder Restriktionen mit geschweiften Klammern. Ein Teilnehmer wies darauf hin, dass aufgabenorientierte Tastaturen angeboten werden sollten. Dieser Vorschlag wird auch von uns befürwortet.

Die beiden Benutzertests haben außerdem ein neues, bisher von uns nicht bedachtes, Problem aufgezeigt. Dadurch, dass mehrere Personen das Klassendiagramm bearbeiten und Elemente nach belieben verschieben können, wussten die Versuchsteilnehmer häufig nicht an welcher Position sich ein Element befindet bzw. ob dieses schon erstellt worden war. Als Konsequenz entstand im Versuch eine erhöhte Kommunikation, um die gesuchten Elemente zu finden. Allerdings wurden auch Klassen doppelt erstellt. In Einzelarbeiten mit anderen Medien, wie einem Personal Computer, treten diese Probleme selten ein.

Nur einige Konzepte führten zu kleinen Konflikten. Dazu zählt der Auswahlkreis über Navigationsrichtungen einer Assoziation, der durch das Verdecken von Auswahlmöglichkeiten durch den eigenen Finger dazu führte, dass die Teilnehmer um den Finger herum lesen mussten. Dieses Konzept sollte nachgebessert werden, indem zum Beispiel ein Halbkreis überhalb des Fingers angezeigt wird und somit keine Auswahl durch den Finger verdeckt wird. Ein weiteres Problem bestand, laut den Versuchsteilnehmern, bei der Erstellungs-Geste der Assoziation. Diese ist ihrer Ansicht nach durch den Double-Tap nicht intuitiv ausführbar. Dennoch wurde von den Teilnehmern im Fragebogen angegeben, dass diese Geste passend gewählt wurde und die „Richtige“ sei. Für diese Geste sollten neue Konzepte entworfen und anschließend getestet werden, um nicht nur eine passende sondern auch eine intuitive Geste für das Erstellen von Assoziationen anbieten zu können.

Allgemein lässt sich sagen, dass die implementierten Konzepte in diesen Benutzertests gut angenommen wurden und größere Probleme, die während der Tests entstanden sind, größtenteils auf die eingeschränkte Implementierung des Prototypen sowie die ungenaue Gestenerkennung zurückzuführen sind.

7 Fazit und Ausblick

Im Verlauf dieser Arbeit wurden mehrere bereits vorhandene Projekte und Arbeiten betrachtet, welche sich mit der Erstellung von Klassendiagrammen auf einem Multi-Touch-Tisch befassten. Diese sahen den MTT unter anderem als einen Hybriden zwischen Whiteboards und Desktop-PCs an, der das gleichzeitige Arbeiten von mehreren Personen erlaubte. Dabei sind wir zum Schluss gekommen, dass die Prototypen dieser Projekte lediglich als Mehrbenutzersysteme implementiert wurden und eine Unterstützung des Entwicklungsprozesses von Klassendiagrammen als auch der Kollaboration kaum gegeben war, trotz vieler nützlicher Funktionen und Konzepte.

Daher wurden in dieser Arbeit zunächst Phasen der kollaborativen Diagrammerstellung unter einem festgelegten Szenario *benutzerorientiert* analysiert. Das Szenario bestand dabei aus einer Projektgruppe mit drei bis fünf Personen, die an einem MTT ein Klassendiagramm kollaborativ erstellen will. In der Analyse wurden Probleme und Schwierigkeiten aufgedeckt, die während der Entwicklung auftreten können. Dabei baut die Analyse hauptsächlich auf Erfahrungen auf, die wir während Software-Projekten sammeln konnten. Zusätzlich wurden Anforderungen herausgestellt, die den Entwicklungsprozess unterstützen sollen oder für die Erstellung eines vollständigen Klassendiagramms notwendig sind. Auf Basis dieser Analyse wurden Konzepte entwickelt, die die erkannten Probleme vermindern bzw. lösen oder Funktionalitäten für die in der Analyse geforderten Anforderungen beschreiben. Zusätzlich wurden neben der reinen Multi-Touch-Tisch-Eingabe multimodale Möglichkeiten und die Kombination des MTTs mit weiteren Geräten und Technologien betrachtet, wenn für spezielle Aufgaben während des Entwicklungsprozesses ein MTT nicht geeignet ist. In der Konzipierung wurde besonderer Wert auf die Benutzerfreundlichkeit gelegt. Um die entworfenen Konzepte zu prüfen, sollte ein Prototyp eines UML-Klassendiagramm-Editors auf einem Multi-Touch-Tisch umgesetzt werden. Die Umsetzung aller entworfenen Konzepte war jedoch aufgrund von deren Komplexität und Zeitvorgaben innerhalb dieser Arbeit nicht möglich. Daher wurden in der Entwurfsphase des Prototyps einige der wichtigsten Konzepte zur Unterstützung des Entwicklungsprozesses von Klassendiagrammen und der Kollaboration am MTT ausgewählt. Für diesen Prototypen wurde zunächst ein geeignetes Framework evaluiert und die Entwicklungsumgebung festgelegt. Unter Betrachtung der analysierten Anwendungsfälle wurden weitere Konzepte aufgenommen und sinnvoll eingeschränkt, um einen funktionstüchtigen Klassendiagramm-Editor zu entwerfen. Die Implementierung des Prototypen erfolge in Python und mit Hilfe des Kivy-Frameworks. Eine darauffolgende Evaluation mit zwei Versuchsgruppen bestätigte überwiegend die im Prototypen umgesetz-

ten Konzepte. Dabei mussten die Versuchsteilnehmer im Team eine vorgegebene Problemstellung mit Hilfe des Prototypen modellieren. In beiden Tests wurden die Konzepte von den Versuchsteilnehmern angenommen und effektiv eingesetzt. Weiterhin wurden von den Testpersonen nicht nur Probleme mit dem Prototypen aufgezeigt, sondern auch Funktionen gewünscht, die teilweise bereits in dieser Arbeit konzipiert jedoch nicht implementiert wurden.

Eine Herausforderung für uns in dieser Arbeit war die Erstellung des Prototypen in einer für uns neuen Programmiersprache und einem uns bis zu Beginn der Arbeit unbekannten Framework. Diese ließen sich allerdings nach einer Einarbeitungszeit effektiv einsetzen. Allerdings musste der Prototyp teilweise mehrfach umstrukturiert bzw. Teile dessen überarbeitet werden, da diese auf anfänglichen Kenntnissen und mangelnden Erfahrungen bauten. Einen großen Rückschlag stellte ein Kivy-Update dar, das grundlegende Funktionalität des Prototypen gravierend veränderte. Dennoch wurde das Update integriert, da dieses viele wichtige Verbesserungen der Funktionalitäten mit sich brachte. Zudem wurde die Grundfunktionalität eines UML-Klassendiagramm-Editors unterschätzt, welche die Erstellung eines vollständigen Klassendiagramms in einer Evaluierung erst ermöglicht und damit die Prüfung der implementierten Konzepte.

Die vier wesentlichen Ziele dieser Arbeit, bestehend aus der Analyse des Entwicklungsprozesses, der Entwicklung von Konzepten zur Lösung der aufgezeigten Probleme, Schwierigkeiten und Anforderungen, der Implementierung eines Prototypen mit einer Teilmenge der entwickelten Konzepte und der abschließenden Evaluation des Prototypen, wurden erreicht. Lediglich in der prototypischen Implementierung konnten nicht alle Punkte der Zielsetzung umgesetzt werden. Zu diesen gehörte das Speichern und Laden von benutzerspezifischen Teildiagrammen, wogegen wir uns aufgrund der Komplexität bereits im Kapitel 4 „Anforderungsanalyse und Systementwurf“ entschieden haben. In diesem Zusammenhang verlor auch das Verschieben von Teildiagrammen und gruppierten Elementen an Priorität und wurde nicht umgesetzt. Ebenso wurde dies im Laufe der Arbeit für den Einsatz der Java-Schlüsselwörter entschieden, wobei diese durch eine vorhandene Funktionalität des eingesetzten Framework-Updates dennoch integriert werden konnten.

Die in dieser Arbeit erreichten Ergebnisse sind nur ein kleiner Teil, der bei der Untersuchung und Unterstützung des kollaborativen Entwicklungsprozesses von Klassendiagrammen am MTT möglich ist. Die Analyse des Entwicklungsprozesses von einer Problemstellung bis hin zu einem ausgereiften Klassendiagramm basiert auf unseren Erfahrungen und einem festgelegten Szenario, sodass nicht nur eine Analyse unter einem anderen Szenario, sondern auch Eine basierend auf anderen Erfahrungen oder Beobachtungen, neue Erkenntnisse gewinnen kann. Aus diesen neu gewonnen Erkenntnissen, können wiederum weitere Konzepte entwickelt werden. Aber auch die in dieser Arbeit aufgedeckten Probleme und Schwierigkeiten sowie Anforderungen konnten nicht alle durch Konzepte abgedeckt werden. So blieben Konzepte offen, die das Zeigen, Markieren und Zeichnen auf einem MTT un-

terstützen sollen und damit das verdeutlichen von Erklärungen ähnlich zum gestikulieren innerhalb Besprechungen erlauben. Zudem besitzen MTT-Anwendungen meist kein festes Menü und sollen durch freie Gesten gesteuert werden, die in manchen Fällen Erklärungsbedarf benötigen, sodass eine Gersteneinführung oder Tooltips hilfreich sind. Weitere Anforderungen die nicht konzipiert wurden, ist das gleichzeitige Arbeiten von zwei Gruppen an einem MTT, das Exportieren von ausgereiften Klassendiagrammen und die Unterstützung der Schnittstellen-Bestimmung durch verschiedene Expertengruppen. Weiterhin können vorhandene Konzepte dieser Arbeit erweitert werden.

Im Konzept 3.4.4 „Der MTT als Präsentationsmedium“ wurde bereits von einem Präsentationsmodus gesprochen. Dieser wurde aufgrund fehlender Informationen über das Arbeitsverhalten mit einem Klassendiagramm-Editor am MTT nicht tiefgehender betrachtet. Nach einer längeren Studie über das Benutzerverhalten sollten Anwender durch weitere Konzepte und Funktionen bei ihrer Arbeit unterstützt werden. Insbesondere im Bezug auf die multimodale Nutzung des MTTs und dessen Kombination mit anderen digitalen Medien können neue Informationen gewonnen werden.

Die Evaluierung des Prototypen hat gezeigt, dass die Benutzer eines UML-Klassendiagramm-Editors alle Seiten des Multi-Touch-Tisches zur Arbeit nutzen wollen, trotz erschwerter Bedingungen. Den Beobachtungen des Versuchs ging jedoch auch hervor, dass nicht alle Bearbeitungen von allen Tischseiten ausgeführt wurden. In weiteren Versuchen müsste deswegen herausgestellt werden, wie genau ein Team am MTT arbeitet, damit diese Arbeitsweise durch weitere Konzepte unterstützt werden kann. Insbesondere sollte das Problem mit unterschiedlichen Leserichtungen bei der Nutzung aller Tischseiten behandelt werden.

Wurde ein ausgereiftes Klassendiagramm erstellt, soll dies oft exportiert bzw. ausgelagert werden, wie zum Beispiel auf ein Blatt Papier gedruckt. Wobei das Klassendiagramm der strikten UML-Notation entsprechen soll. Bereits bei der Erstellung des Entwurfs-Klassendiagramms für den Prototypen ist aufgefallen, dass UML-Konventionen nicht durchgehend eingehalten wurden und dieses daraufhin mehrfach kontrolliert werden musste. Daher ist es vorteilhaft, wenn die genutzte Anwendung beim Validieren der UML-Notation unterstützt und den Benutzern mitteilt, ob Aufgabenelemente im Klassendiagramm vorhanden sind und alle im Diagramm enthaltenen Eingaben gültig bzw. soweit überprüfbar sinnvoll sind.

Zu weiteren Verbesserungs- und Weiterentwicklungsmöglichkeiten gehören eine bessere Unterscheidung von Benutzereingaben als durch die konzipierten Benutzer-Tangibles, die Nutzung alternativer Eingabemöglichkeiten, Funktionalitäten wie beispielsweise das Generieren von Programmcode zur Unterstützung der Überleitung in die Implementierungsphase und viele Weitere. Zusätzlich sollte ein einfacher Datenaustausch zwischen einer Editor-Anwendung auf einem MTT und dieser auf einem Desktop-PC unterstützt werden, um den Datenwechsel zwischen Gruppen- und Einzelarbeiten einfach zu halten und die Wahl eines Klassendiagramm-Editors für Benutzer nicht einzuschränken. Diese Ansicht wird auch vom

COSMOS-Projekt geteilt.[1] Außerdem ist, nach den Autoren von COSMOS, eine enge Verknüpfung zwischen Klassen- und Komponentendiagrammen sinnvoll.[2] Dies sollte auch nach unseren Erfahrungen unterstützt werden, damit das Entwickeln verschiedener Diagrammtypen der UML einfacher und übersichtlicher wird. Dazu zählen unserer Ansicht nach allerdings auch Paket- und Objektdiagramme, die eine starke logische Verknüpfung zu Klassendiagrammen haben.

Neben den Erweiterungen der Konzepte, sollten die in dieser Arbeit nicht implementierten Konzepte umgesetzt und ebenfalls evaluiert werden. Zudem konnten die von uns durchgeführten Benutzertests nur in einem kleinen Zeitrahmen durchgeführt werden, wodurch der volle Umfang eines Konzepts nicht vollständig evaluiert werden konnte. Die implementierten informellen Aufgabenelemente sind nur ein Beispiel dieser Konzepte. Erst durch einen Entwicklungsverlauf, der durch Phasen von Gruppentreffen, Besprechungen und Einzelarbeiten entsteht, können diese Konzepte vollständig geprüft werden. Des Weiteren sollten die in dieser Arbeit gewonnenen und ausgewerteten Ergebnisse der Evaluierung des Prototypen sowie Verbesserungsvorschläge durch die Versuchsteilnehmer in den Prototypen einfließen und erneut getestet werden. Der Einsatz, bzw. eine Evaluierung im größeren Rahmen, einer Weiterentwicklung des Prototypen ist in Softwaretechnik-Praktika einer Universität, Projekten und IT-Firmen, die mit den Grundbausteinen der UML-Klassendiagramm-Notation auskommen, bereits jetzt denkbar.

[1] Vgl. Löffelholz, Daniel et al.: COSMOS: Multi-touch support for collaboration in user-centered projects. In Heiss, Hans-Ulrich (Hrsg.): Informatik 2011. Bonn: Ges. für Informatik. Online im Internet: `http://www.user.tu-berlin.de/komm/CD/paper/061521.pdf` – Zugriff am 04.01.2013, ISBN 978–3–88579–286–4, [S. 10].

[2] Vgl. Löffelholz, Daniel et al., a. a. O., [S. 6].

Glossar

Artefakt

Ein Artefakt bezeichnet in der Softwareentwicklung ein erstelltes Dokument, Diagramm, Programm o. Ä..

Aufgabenelement

Ein Aufgabenelement ist ein informelles Element um Problemstellungen und Aufgaben festzuhalten. Dieses Element wurde für den Entwicklungsprozess eines Klassendaigramms in dieser Arbeit konzipiert.

Double-Tap

Der Double-Tap bezeichnet das doppelte Antippen auf einer Touch-Oberfläche. Vergleichsweise mit einem Doppelklick. Soll mit einem Double-Tap eine Geste gezeichnet werden, so muss beim zweiten Antippen der Touch gehalten und entsprechend der gewünschten Geste geführt werden. Die Eingabe dieser Geste wird mit dem loslassen des Touchs bestätigt.

Element

In dieser Arbeit soll der Begriff Element im Zusammenhang mit Diagrammelementen stellvertretend für die visuellen Diagrammelemente stehen, die in Klassendiagrammen existieren oder in dieser Arbeit als informellen Elemente konzipiert wurden. Elemente stehen für informelle Elemente, wie Aufgabenelemente, und formelle Elemente wie, wie Klassen und Assoziationen.

Framework

Ein Framework ist in der Softwareentwicklung ein Grundgerüst für einen speziellen Aufgabenbereich, wie beispielsweise in dieser Arbeit Kivy als Framework benutzt wird. Frameworks liefern hilfreiche Funktionen um die Entwicklung einer Anwendung für den Aufgabenbereich zu erleichtern. Anwendungen können auf einem Framework aufbauen, um diese hilfreichen Funktionen nutzen. Meist enthalten Frameworks Funktionen die in dem Aufgabenbereich häufig benötigt werden und so von der zu entwickelten Anwendung nicht eigenständig umgesetzt werden muss. Durch ein Framework können sich Entwickler häufig auf die Entwicklung ihrer Hauptaufgabe der Anwendung konzentrieren.

Geste

Als Gesten sind in dieser Arbeit (Multi-)Touch-Gesten gemeint, die mit Händen auf einer Touch-Oberfläche ausgeführt werden. Je nach ausgeführter Geste, kann eine Anwendung auf diese reagieren und Aktionen ausführen. Aktionen, die durch verbreitete Gesten von vielen Anwendungen ähnlich ausgelöst werden, sind: Zoomen/ Skalieren, Verschieben, Rotieren.

Grenzassoziation

Dieser Begriff wurde in dieser Arbeit durch das Speichern-Konzept eingeführt. Dabei bezeichnet eine Grenzassoziation eine Assoziation zwischen Elementen, die verschiedenen Personen als Verantwortlichkeit zugewiesen sind, genannt.

Hold-Tap

Ein Hold-Tap bezeichnet eine Touch-Geste. Diese Geste ist Vergleichbar mit einem Antippen auf der (Multi-)Touch-Oberfläche, wobei das Absetzen des Touchs erst nach einer bestimmten Zeit erfolgen darf. Ein Beispiel ist ein Hold-Tap der eine Sekunde gehalten werden soll um ein Hauptmenü aufzurufen.

Informelle Elemente

Als ein informelles Element wird in dieser Arbeit ein Element bezeichnet, dass nicht der UML-Notation entspricht. Diese Elemente werden innerhalb der Arbeit konzipiert. Dazu gehören Aufgabenelemente, Marker und Anzeigefelder für den Verantwortlichen eines Elements.

Kickoff-Meeting

Ein Kickoff-Meeting ist das erste Meeting eines Teams zu einem bestimmten Thema bzw. Aufgabe, wie beispielsweise die Erstellung eines Klassendiagramms von einem Softwareentwicklungsteam.

Multimodale (Interaktionen)

Als multimodale Interaktionen wird eine Kombination von verschiedenen Ein- und Ausgabesystemen bezeichnet. In dieser Arbeit wird beispielsweise der Multi-Touch-Tisch unter bestimmten Gesichtspunkten als ein multimodales System betrachtet, dass neben der reinen Touch-Eingabe weitere Eingabemöglichkeiten, wie zum Beispiel eine Sprach-Eingabe oder Stift-Eingabe, unterstützt.

Multi-Touch-Tisch (MTT), Tabletop

Ein Multi-Touch-Tisch ist ein Tisch mit einem Multi-Touch-Eingabesystem, wodurch mehrere Touchpoints gleichzeitig erkannt werden können, sodass unter anderem mehrere Personen gleichzeitig den Tisch bedienen können.

Object Constraint Language (OCL)

OCL steht für „Object Constraint Language“ und ist eine formale Sprache der UML, mit der beispielsweise bei UML-Diagrammen weitere Bedingungen, wie beispielsweise Invarianten in Klassendiagrammen, festgehalten werden können.

Slider

Ein Slider ist ein grafisches Eingabe-/ Steuerelement, das häufig für die Eingabe von Zahlenbereichen benutzt wird. Es stellt den gesamten Eingabebereich als einen Balken bzw. Zahlenstrahl dar, auf dem ein Wert mit einem Auswahlelement auf dem Balken ausgewählt werden kann, indem dieses Auswahlelement auf dem Balken verschoben wird.

Splittscreen

Ein Splittscreen, auch Bildschirmaufteilung genannt, bezeichnet die Aufteilung des Bildschirms in mindestens zwei von einander unabhängige Bereiche.

Tangible

Als ein Tangible wird ein physikalisches, greifbares Objekt genannt, das beispielsweise auf einem Multi-Touch-Tisch abgelegt und vom Tisch erkannt bzw. interpretiert wird. Der aufgelegte Gegenstand kann als alternative der Gesten-Eingabe verwendet werden und interagiert mit dem Tisch.

Tap

Ein (Single-)Tap ist eine (Touch-)Geste, die ein einfaches Antippen der Touch-Oberfläche beschreibt. Dies ist auch mit mehreren Fingern gleichzeitig möglich und wird je nach Anzahl der Finger als beispielsweise ein Drei-Finger-Tap bezeichnet.

Touchpoint

Touchpoints sind Berührungspunkte die beim Auflegen eines oder mehrere Finger auf der Touch-Oberfläche erkannt werden.

Übergangselement

Ein Übergangselement bezeichnet in dieser Arbeit ein informelles Diagrammelement, das während der Entwicklung von Diagrammen für den Übergang zu einem strikten ausgereiften Diagramm genutzt wird bzw. genutzt werden soll. Übergangselemente können Darstellen von Aufgaben sowie Notizen sein.

Unified Modeling Language (UML)

UML ist das Akronym für „Unified Modeling Languag“ und bezeichnet eine grafische Modellierungssprache, in der Software spezifiziert, konstruiert und dokumentiert werden kann.

Voice over IP (VoIP)

Voice over IP wird auch als Internet-Telefonie bezeichnet und überträgt Sprache sowie Steuerinformationen über eine Internet- anstatt über eine Festnetzleitung.

Widget

Als ein Widget wird in dieser Arbeit ein GUI-Element des Kivy-Frameworks bezeichnet, welches neben einer grafischen Anzeige ebenfalls das Objekt repräsentiert, dass Zustände und weitere Logik des GUI-Elements enthält. Weiterhin kann ein Widget festlegen welche Aktionen bei Benutzereingaben auf diesem Element ausgeführt werden sollen.

WIMP

WIMP ist das Akronym für „Windows, Icons, Menus, Pointer“, welche zur Darstellung von Benutzeroberflächen für 2D-Anwendungen verwendet werden.

Abbildungsverzeichnis

Literatur

(1) **Anonymus:** Mobilgeräte entsperren: Google patentiert Methode zur Gesichtserkennung. In: Spiegel Online. Online im Internet: `http://www.spiegel.de/netzwelt/gadgets/google-erhaelt-%20patent-auf-entsperren-per-gesichtserkennung-a-854294.html`⟩ – Zugriff am 12.07.2013.

(2) **Any.do:** Frequently Asked Questions. Online im Internet: `http://www.any.do/faq`⟩ – Zugriff am 12.08.2013.

(3) **Fraunhofer-Institut für Arbeitswirtschaft:** MT4j Webseite. Online im Internet: `http://www.mt4j.org`⟩ – Zugriff am 12.08.2013.

(4) **Hahn, Thomas:** Future Human Computer Interaction with special focus on input and output techniques. Online im Internet: `http://www.olafurandri.com/nyti/papers2010/FutureHumanComputerInteraction.pdf`⟩ – Zugriff am 12.08.2013.

(5) **Kammer, Dietrich et al.:** Taxonomy and Overview of Multi-touch Frameworks: Architecture, Scope and Features. Online im Internet: `http://vi-c.de/vic/sites/default/files/Taxonomy_and_Overview_of_Multi-touch_Frameworks_Revised.pdf`⟩ – Zugriff am 12.08.2013.

(6) **Löffelholz, Daniel et al.:** COSMOS: Multi-touch support for collaboration in user-centered projects. In **Heiss, Hans-Ulrich (Hrsg.):** Informatik 2011. Bonn: Ges. für Informatik. Online im Internet: `http://www.user.tu-berlin.de/komm/CD/paper/061521.pdf`⟩ – Zugriff am 04.01.2013, ISBN 978–3–88579–286–4.

(7) **Lutz, Mark/Ascher, David/Gherman, Dinu C.:** Einführung in Python: [moderne OO-Programmierung ; behandelt Python 2.5]. 2. Auflage. Beijing und Cambridge und Farnham und Köln und Paris und Sebastopol und Taipei und Tokyo: O'Reilly, 2007, ISBN 978–3–89721–488–0.

(8) **Microsoft:** MSDN Surface. Online im Internet: `http://msdn.microsoft.com/en-us/library/ff727864.aspx`⟩ – Zugriff am 12.08.2013.

(9) **Oppold, Pete et al.:** Multi-Touch Table with Object Recognition: codename Planck. Online im Internet: `http://eecs.ucf.edu/seniordesign/fa2011sp2012/g14/Documents/SD1%20Group%2014%20Final%20Documentation_v1.1.pdf`⟩ – Zugriff am 12.08.2013.

(10) **Rauber, Thomas:** Algorithmen in der Computergraphik. 1. Auflage. Stuttgart: Teubner Verlag, 1993, ISBN 3–519–02127–7.

(11) **Rupp, Chris/Queins, Stefan/Zengler, Barbara:** UML 2 glasklar: Praxiswissen für die UML-Modellierung. 3. Auflage. München und Wien: Hanser, 2007, ISBN 978–3446411180.

(12) **Selber, Thomas/Pongelli, Stefano/Valente, Giulio:** DTDiagram - a Pen and Touch UML Class Diagram editor for DiamondTouch tabletops: Semester project for the Information Systems Lab 2012 at ETHZ (www.ethz.ch). Online im Internet: `http://www.youtube.com/watch?v=hsZOsjf5un4`⟩ – Zugriff am 04.01.2013.

(13) **Totolici, Alex et al.:** Collabee – Multi-touch Collaborative Diagramming. Online im Internet: `http://www.jrejre.net/2010/08/collabee-multi-touch-collaborative-diagramming`⟩ – Zugriff am 04.01.2013.

(14) **Totolici, Alex et al.:** Collaborative UML Diagramming on a Multitouch Surface. Online im Internet: `http://www.youtube.com/watch?v=K6NCj2czZrE`⟩ – Zugriff am 12.08.2013.

(15) **Virbel, Mathieu/Hansen, Thomas/Denter, Christopher:** Kivy Dokumentation: Release 1.8.0-dev. Online im Internet: `http://kivy.org/docs/pdf/Kivy-latest.pdf`⟩ – Zugriff am 12.08.2013.

(16) **Virbel, Mathieu/Hansen, Thomas/Denter, Christopher:** Kivy Webseite: About us. Online im Internet: `http://kivy.org/#aboutus`⟩ – Zugriff am 12.08.2013.

Anhang A

Aufteilung der Arbeit zwischen den Autoren

Im Folgenden wird die Aufteilung der schriftlichen Ausarbeitung und der prototypischen Implementierung auf die Autoren dargestellt.

Aufteilung der schriftlichen Ausarbeitung auf die Autoren

Überschriften	Seiten	Autor
Abstract	0,5	Gemeinsam
1 Einleitung (Einleitungstext)	0,25	Andreas Gehle
1.1 Aufgabenstellung	0,5	Alexei Quapp
1.2 Motivation	2,25	Alexei Quapp
1.3 Zielsetzung	1,75	Andreas Gehle
1.4 Ähnliche Projekte (Einleitungstext)	0,25	Alexei Quapp
1.4.1 COSMOS	1,5	Alexei Quapp
1.4.2 DTDiagram	0,5	Alexei Quapp
1.4.3 Collabee	1	Alexei Quapp
1.4.4 Fazit der bisherigen Projekte	1	Alexei Quapp
1.5 Betrachtung des MTTs als multimodales System	3,25	Andreas Gehle
2 Analyse des Entwicklungsprozesses eines Klassendiagramms (Einleitungstext)	0,75	Andreas Gehle
2.1 Beschreibung des Szenarios	0,5	Andreas Gehle
2.2 Erste Phase: Aufgabenplanung und Verteilung	2	Andreas Gehle
2.3 Arbeiten in Kleingruppen oder Einzelarbeit	1	Andreas Gehle

2.4 Zweite Phase: Fortschritte zusammentragen und besprechen	1,5	Andreas Gehle
2.5 Zyklus: Teammeetings, Fortschritte und Reviews	1	Andreas Gehle
2.6 Dritte Phase: Ausgereiftes Diagramm	0,75	Andreas Gehle
3 Konzepte zur Unterstützung des Entwicklungsprozesses (Einleitungstext)	0,5	Andreas Gehle
3.1.1 Replizierte Menüs	1,5	Alexei Quapp
3.1.2 Anzeige von verschiedenen Detailstufen	2,5	Andreas Gehle
3.1.3 Anmeldung am Multi-Touch-Tisch	2	Alexei Quapp
3.1.4 Zuweisung von Verantwortlichkeiten	0,5	Alexei Quapp
3.2.1 Aufgaben und Problemstellungen visualisieren	2,75	Andreas Gehle
3.2.2 Optisches Feedback	0,75	Alexei Quapp
3.2.3 Undo und Redo	1	Alexei Quapp
3.2.4 Verschieben von gruppierten Elementen	1,75	Alexei Quapp
3.2.5 Verschieben von Attributen und Methoden	0,5	Alexei Quapp
3.2.6 Oberklasse erstellen und Klassen vereinen	2,25	Andreas Gehle
3.2.7 Erzeugung von Assoziationen und Bearbeitung ihrer Attribute	4	Andreas Gehle
3.2.8 Assoziationspfad mit Fix- und Dockingpunkten beliebig führen	5,25	Andreas Gehle
3.3.1 Speichern von Diagrammen	2,25	Alexei Quapp
3.3.2 Laden von Diagrammen	3,5	Alexei Quapp
3.4.1 Visuelle Marker für Besprechungsfortschritte	2,5	Andreas Gehle
3.4.2 Kommentarfunktion	1,25	Gemeinsam
3.4.3 Checkliste	0,5	Alexei Quapp
3.4.4 Der MTT als Präsentationsmedium	2	Alexei Quapp
3.4.5 Bearbeitungsprotokoll eines Diagramms	1,5	Alexei Quapp
3.4.6 Anzeige von Revisionsdiagrammen und Vergleichen von Diagrammen	2,5	Alexei Quapp
3.5.1 Das Smartphone als Eingabegerät	2,5	Alexei Quapp

3.5.2 Der (Touch-)Stift als Eingabegerät	1	Andreas Gehle
4 Anforderungsanalyse und Systementwurf (Einleitungstext)	0,5	Andreas Gehle
4.1 Entwicklung eines Prototypen	0,5	Andreas Gehle
4.2 Evaluierung geeigneter Multi-Touch-Frameworks und -Bibliotheken	0,5	Andreas Gehle
4.2.1 Übersicht und Vergleich	1,5	Andreas Gehle
4.2.2 MT4j	1,5	Andreas Gehle
4.2.3 WPF/.NET + Surface SDK	1	Andreas Gehle
4.2.4 Kivy	1,5	Andreas Gehle
4.2.5 Evaluierung der Frameworks	1,25	Andreas Gehle
4.3.1 Stichdaten des Multi-Touch-Tisches	0,75	Andreas Gehle
4.3.2 Technische Voraussetzungen	0,5	Andreas Gehle
4.3.3 Verwendete Hilfsprogramme	0,75	Andreas Gehle
4.4 Anwendungsfälle des prototypischen UML-Editors	5	Gemeinsam
4.5.1 Zum Einsatz kommende Softwarepattern	0,75	Andreas Gehle
4.5.2 Klassendiagramm des Prototypen	5,5	Alexei Quapp
4.6 Eingesetzte Gesten	6,25	Alexei Quapp
4.7 Speichern und Laden	0,75	Alexei Quapp
5 Implementierung (Einleitungstext)	0,75	Gemeinsam
5.1 Eingesetzte Hard- und Software	1	Andreas Gehle
5.2 Entwicklung mit Python und Kivy	1,25	Alexei Quapp
5.3 Umgang mit Touch-Ereignissen	1	Alexei Quapp
5.4 Mehrfachvererbung	0,5	Alexei Quapp
5.5 Algorithmen des Prototypen	2	Andreas Gehle
5.6 Implementierung der Konzepte	0,75	Andreas Gehle
5.7 Installationsanleitung	0,5	Andreas Gehle
6 Evaluierung (Einleitungstext)	0,5	Gemeinsam
6.1 Der Versuchsaufbau	1,5	Andreas Gehle

6.2 Erwartungen an den Testversuch	2	Alexei Quapp
6.3 Verlauf und Beobachtungen des Versuchs	7	Gemeinsam
6.4 Resultate der Benutzertests	2,75	Gemeinsam
7 Fazit und Ausblick	3,5	Gemeinsam

Tabelle A.1: Aufteilung der Arbeit auf die Autoren

Aufteilung der prototypischen Implementierung auf die Autoren

Die Aufteilung der Implementierung (Tabelle A.2) wird anhand der erstellten Klassen dargestellt. Dabei hat sich Andreas Gehle auf die Umrechnung zwischen den verschiedenen Koordinatensystemen und der Kollisionsberechnung spezialisiert, während Alexei Quapp vorrangig für die Verarbeitung von Touch-Ereignissen zuständig war. Zusätzlich hat Andreas Gehle die Konfiguration des Prototyps und die Erstellung der Pakete (Builds) übernommen.

Durch die unterschiedlichen Spezialisierungen der Autoren während der Implementierungsphase wurde ein Teil der Klassen gemeinsam bearbeitet. Weiterhin ist zu beachten, dass die Klassen verschieden groß sind. Insgesamt ergibt sich dadurch eine faire Aufteilung des Implementierungsaufwands auf die beiden Autoren, wobei in etwa ein Drittel der Anwendung von beiden Autoren in Zusammenarbeit implementiert wurde.

Klasse	Autor
Association	Andreas Gehle
AssociationElement	Andreas Gehle
AssociationEndPoint	Andreas Gehle
AssociationName	Andreas Gehle
Attribut	Gemeinsam
Cardinality	Andreas Gehle
ClassBox	Gemeinsam
Classdiagram	Alexei Quapp
ClassdiagramLoader	Alexei Quapp
ClassdiagramVisitor	Alexei Quapp
ClassElement	Alexei Quapp
ClassName	Alexei Quapp
Diagram	Gemeinsam
DiagramElement	Gemeinsam
ElementID	Alexei Quapp
GestureRecognition	Gemeinsam
Icon	Andreas Gehle
Icon_TextInput	Gemeinsam
LoadFileDialog	Alexei Quapp
LoadFilePopup	Alexei Quapp
MainmenuDialog	Alexei Quapp
MainmenuPopup	Alexei Quapp
Method	Gemeinsam
Role	Andreas Gehle
SaveFileDialog	Alexei Quapp
SaveFilePopup	Alexei Quapp
TaskBox	Gemeinsam

UMLEditor	Gemeinsam
User	Alexei Quapp
UserAdministration	Alexei Quapp
UserLabel	Gemeinsam
XMLFullSaveVisitor	Alexei Quapp

Tabelle A.2: Aufteilung der Arbeit auf die Autoren

Anhang B

Das verwendete Speicherformat

```
<?xml version="1.0" encoding="UTF-8"?>
<classdiagram id_counter="int" is_diagrampart="boolean">

    <association element_id="int" state="markerstate"
                                          read_direction="boolean">*
      <name>Assoziationsname</name>
      <source source_id="int" type="int">
          <cardinality state="markerstate">Kardinalität</cardinality>
          <role state="markerstate">Rolle</role>
      </source>
      <destination destination_id="int" type="int">
          <cardinality state="markerstate">Kardinalität</cardinality>
          <role state="markerstate">Rolle</role>
      </destination>
    </association>

    <classbox element_id="int" is_editable="boolean"
                                  state="markerstate" user_id="int">*
        <pos x="x_coordinate" y="y_coordinate">
        <name>Klassenname</name>
        <comment>Kommentar</comment>
        <attribut element_id="int" state="markerstate">*
            Attribut
        </attribute>
        <method element_id="int" state="markerstate">*
            Methode
        </method>
    </classbox>
```

```
    <taskbox element_id="int" is_editable="boolean"
                                    state="markerstate" user_id="int">*
        <pos x="x_coordinate" y="y_coordinate">
        <name>Klassenname</name>
        <comment>Kommentar</comment>
        <color>Farbe</color>
    </taskbox>

    <user element_id="int">*
        <name>Name</name>
        <color>Farbe</color>
    </user>
</classdiagram>
```

Anhang C

Anlagen des Benutzertests

Im folgenden befindet sich eine Gestenübersicht und Aufgabenstellung, die den Versuchsteilnehmern als Hilfestellung bei der Durchführung der Benutzertests gedient haben. Zusätzlich ist ein Fragebogen angehängt der von den Kandidaten nach den Tests am Multi-Touch-Tisch ausgefüllt wurde.

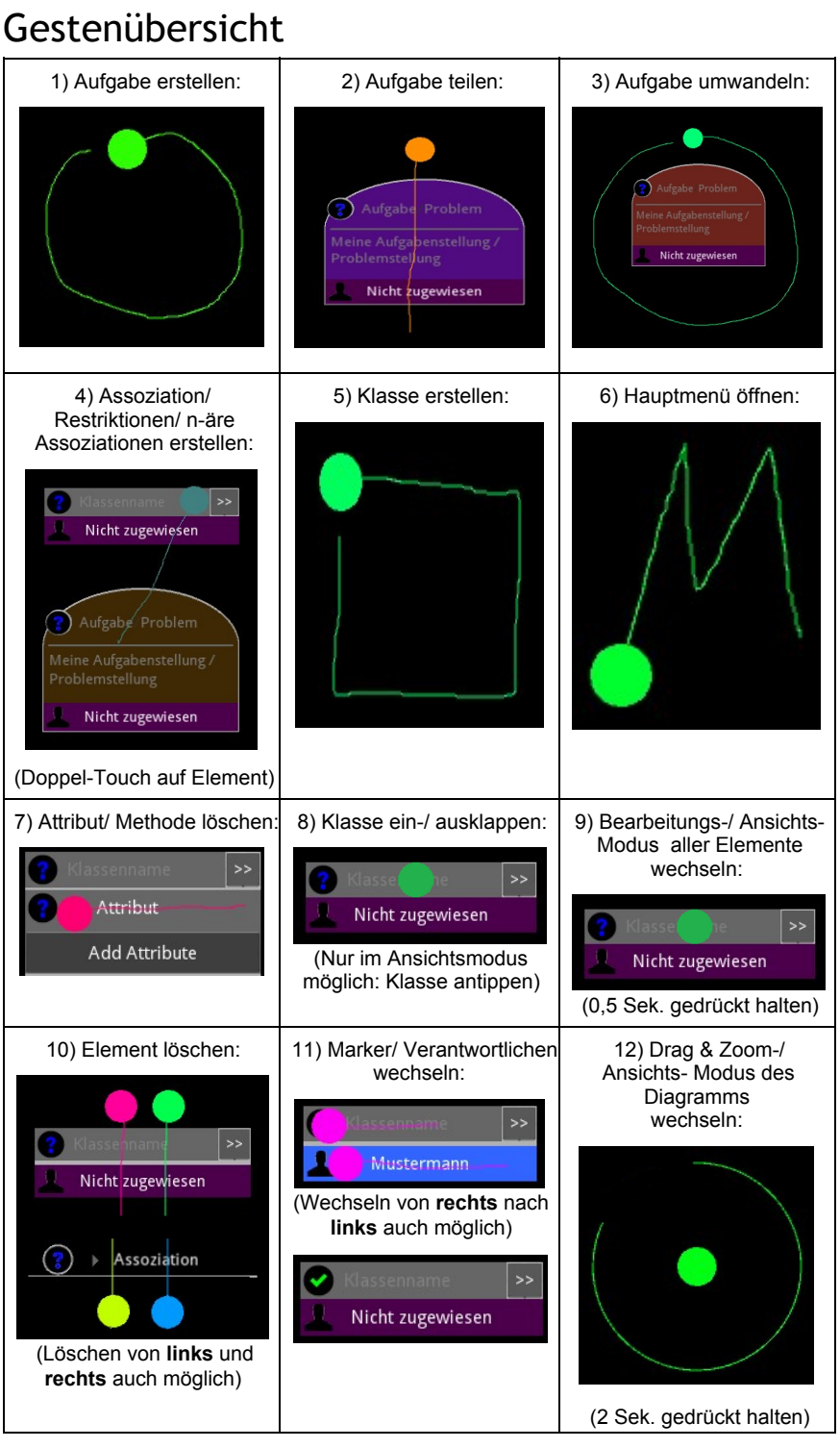


Abbildung C.1: Genstenübersicht für den Benutzertest

Aufgabenstellung

Die Aufgabe besteht darin einen vollständigen Flughafen zu modellieren.
Jedes Teammitglied soll sich dabei vorerst auf ein Teilproblem beschränken.
Folgende Aufgabenbereiche bzw. Rollen sollen untereinander aufgeteilt werden:

- Gepäckorganisation (GO)
- Gebäude des Flughafens (G)
- Personal (P)

Bearbeite Sie nun folgende Aufgabe mit Ihrer Rollenzugehörigkeit.

Aufgabe

Ein Flughafen besteht aus mindestens einem Terminal und mindestens einer Wartungshalle.
Um diese Gebäude identifizieren zu können, sind sie mit einer Bezeichnung versehen (z.B. “Terminal A”).
Weiter besteht ein Flughafen aus einer Menge von Gepäckfahrzeugen, die zur Identifizierung durchnummeriert sind.
Ein Flugzeug wird durch seine Typenbezeichnung sowie die Anzahl der Passagiere, die es transportieren kann, charakterisiert.
Wenn ein Flugzeug sich an einem Flughafen befindet, steht es entweder an einem Terminal oder wird in einer Wartungshalle gewartet.
An einem Terminal können jedoch höchstens zehn Flugzeuge gleichzeitig stehen und in einer Wartungshalle können höchstens zwei Flugzeuge gleichzeitig gewatet werde.
Wenn ein Crewmitglied Dienst hat, arbeitet es in einem Flugzeug.
Die minimale Größe einer Crew beträgt zwei Crewmitglieder, die maximale Größe sind acht Crewmitglieder.
Passagiere sitzen in einem Flugzeug oder warten in einem Terminal.
Zudem gehören jedem Passagier maximal zwei Gepäckstücke, die Gepäckstücke befinden sich entweder auf einem Gepäckfahrzeug, in einem Terminal oder in einem Flugzeug.
Sie sind durch ihr Gewicht und eine Identifikationsnummer gekennzeichnet.
Sowohl Crewmitglieder als auch Passagiere haben einen Namen und eine Passnummer.

Abbildung C.2: Aufgabenstellung für den Benutzertest

Fragebogen

Fragebogen im Rahmen der Bachelorarbeit “Konzepte und prototypische Umsetzung eines kollaborativen UML-Klassendiagramm-Editors für Multi-Touch-Tische”

Diagrammentwicklung

1. **Waren informelle Elemente (Aufgaben) hilfreich bei der Planung des Diagramms?**
   Markieren Sie nur ein Oval.
   - ◯ Ja
   - ◯ Nein

2. **Wenn nein, warum nicht?**
   Haben Sie ja angekreuzt können Sie diese Frage auslassen

   .....
   .....
   .....
   .....
   .....

3. **Haben Sie diese vollständig benutzt?**
   Aufgabe erstellt, geteilt, umgewandelt, alle Textfelder benutzt?

   .....
   .....
   .....
   .....
   .....

4. **Wie hätten Sie diese gerne noch benutzt bzw. was fehlte Ihnen dazu?**

   .....
   .....
   .....
   .....
   .....

1

Abbildung C.3: Fragebogen für den Benutzertest (Seite 1)

5. **Wie und wozu haben Sie Kommentarfelder von Klassen und Aufgaben benutzt?**

Kollaboration

6. **Wie sah die Zusammenarbeit mit den weiteren Versuchsteilnehmern aus?**
   Absprachen, Erklärungen am MTT

7. **Wie sind Sie mit dem Arbeitsplatz am Multi-Touch-Tisch ausgekommen?**
   Bedrängung/Behinderung durch Andere? Aufstellung am Tisch

8. **Was fanden Sie positiv an der Zusammenarbeit am MTT, gegenüber anderen im Team verwendeten Medien (SwtPra)?**

Abbildung C.4: Fragebogen für den Benutzertest (Seite 2)

9. **Haben Sie das gesamte Diagramm verschoben, gedreht oder vergrößert/verkleinert? Wenn ja, wieso fanden Sie dies für angebracht?**

10. **Wie haben Sie die Benutzer-Zuordnung verwendet? Was soll diese Ihrer Meinung nach ausdrücken?**

11. **Wann haben Sie Marker eingesetzt und was sollen diese Ihrer Meinung nach festhalten?**

Gesten

Abbildung C.5: Fragebogen für den Benutzertest (Seite 3)

12. **Welche Elemente haben Sie erstellt bzw. welche Gesten haben Sie benutzt?**
(Siehe Gestenübersicht)
Wählen Sie alle zutreffenden Antworten aus.

- ☐ Aufgabe erstellen
- ☐ Aufgabe teilen
- ☐ Aufgabe umwandeln
- ☐ Assoziationen erstellt
- ☐ Restriktionen erstellt (Einschränkungen mit z.B. {XOR})
- ☐ N-äre Assoziationen / Assoziationsklassen erstellt
- ☐ Attribute/ Methoden gelöscht
- ☐ Klasse ein- bzw. ausgeklappt
- ☐ Bearbeitungs- / View-Modus eines Elements gewechselt
- ☐ Elemente gelöscht
- ☐ Verantwortlichkeiten gesetzt/ gewechselt
- ☐ Marker gesetzt/ gewechselt
- ☐ Modus des gesamten Diagramms gewechselt

13. **Welche Gesten haben Ihnen Probleme bereitet bzw. sind Ihnen schwer gefallen?**

.....

.....

.....

.....

.....

14. **Haben Sie sich einige andere bzw. alternative Gesten gewünscht? Wenn ja, welche?**

.....

.....

.....

.....

.....

15. **Welche Gesten konnten Sie intuitiv verwenden bzw. schnell erlernen? Haben Sie eine Begründung dafür?**

.....

.....

.....

.....

.....

Abbildung C.6: Fragebogen für den Benutzertest (Seite 4)

16. **Weswegen haben Sie (nicht) die Möglichkeit genutzt Informationen zu verbergen (View-Modus)?**

Allgemein

17. **Was ist Ihnen besonders negativ aufgefallen bei der Entwicklung?**

18. **Was ist Ihnen besonders positiv aufgefallen bei der Entwicklung?**

19. **Würden Sie eine Entwicklung von UML Diagrammen auf einem MTT gegenüber anderen Medien bevorzugen? Warum?**

Abbildung C.7: Fragebogen für den Benutzertest (Seite 5)

20. **Weitere Anmerkungen und Ergänzungen**

.....

.....

.....

.....

.....

Abbildung C.8: Fragebogen für den Benutzertest (Seite 6)

Anhang D

Prototypische Implementierung und Benutzertest-Aufnamen

Die auf den nächsten Seiten angehefteten DVDs enthalten folgenden Inhalt.

DVD 1

- Erster Teil der Aufnahme des ersten Benutzertests

DVD 2

- Erster Teil der Aufnahme des zweiten Benutzertests

DVD 3

- Zweiter Teil der Aufnahme des ersten Benutzertests
- Zweiter Teil der Aufnahme des zweiten Benutzertests
- Ausführbare Build-Datei des Prototypen für Windows-Systeme
- Quellcode des Prototypen
- PDF der Bachelorarbeit

Eine Installationsanleitung des Prototypen befindet sich im Kapitel 5.7 „Installationsanleitung“. Weiterhin werden die im Prototypen benutzten Gesten im Kapitel 4.6 „Eingesetzte Gesten“ beschrieben. Die Build-Datei kann auf Windows-Systemen ohne weitere Einrichtungen ausgeführt werden. OpenGL 2.0 ist Mindestvoraussetzung für die Ausführung.