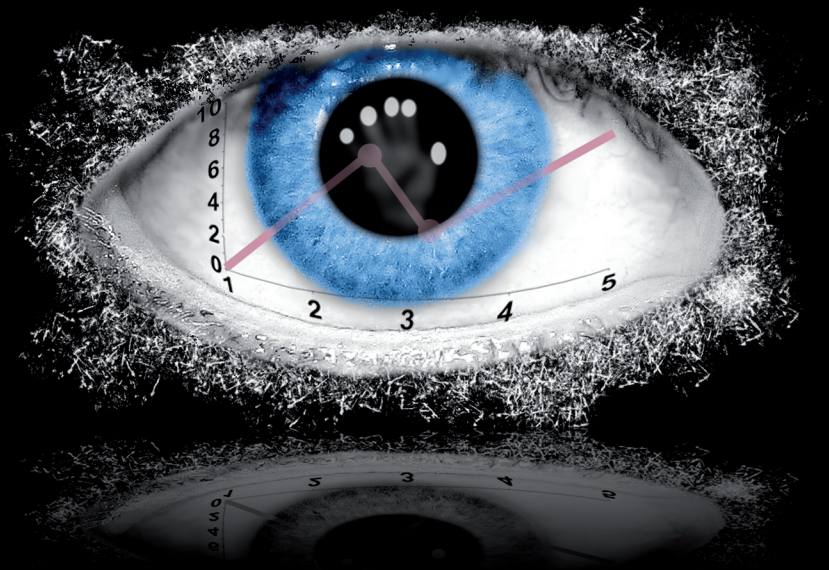
Abschlussbericht
Projektgruppe Visual Analytics



CARL VON OSSIETZKY universität OLDENBURG

Abschlussbericht

Projektgruppe Visual Analytics

Markus Cramer,
Jutta Fortmann,
Thole Klingenberg,
Kai Maschke,
Alexandr Puchkovskiy,
Plamen Rusinov,
Marwin Schmitt,
Konstantin Sleumer,
Arne Uphoff

Betreut von:
Dipl.-Inform. Stefan Flöring,
Dipl.-Inform. Tobias Hesselmann

Carl von Ossietzky Universität Oldenburg
Fakultät II - Department für Informatik
Prof. Dr. Dr. h.c. H.-J. Appelrath

Sneak Preview

Was ist *TaP*?

Unser System ermöglicht die visuelle Analyse großer, multidimensionaler Datenmengen auf der berührungssensitiven Oberfläche eines multitouch-fähigen TableTop Computers (s. Abbildung 1).

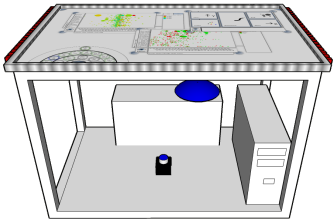


Abb. 1: Aufbau des eingesetzten Multitouch-Tisches

Direkte Interaktion

Die Arbeitsfläche von *TaP* lässt sich vollständig über Gesten steuern, die mit einem oder mehreren Fingern durchgeführt werden können. Ein Diagramm kann an einer beliebigen Stelle auf der Arbeitsfläche erstellt und anschließend vergrößert, verkleinert und wie ein reales Objekt verschoben werden.

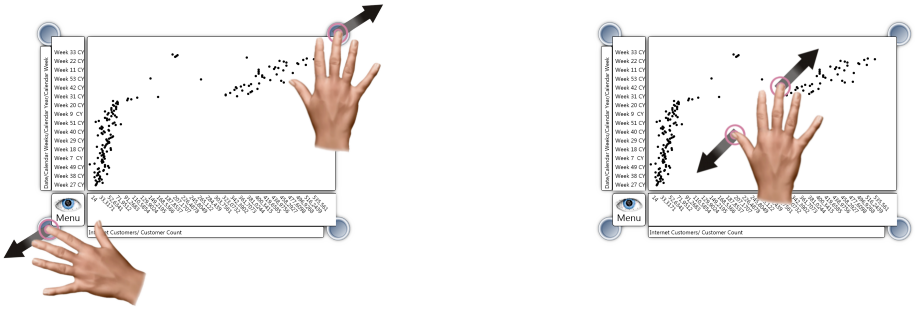


Abbildung 2: Skalierung (links) und Zoomen (rechts) von Diagrammen durch Gesten

Pie-Menü

Die Navigation durch die Daten eines OLAP-Systems löst *TaP* durch ein an die Form der Hand angepasstes Menü. Es besteht aus mehreren drehbaren Ringen, die je nach Bedarf ein- und auch ausgeklappt werden können.

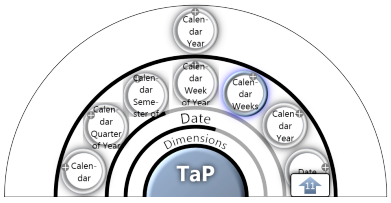


Abb. 3: Pie-Menu

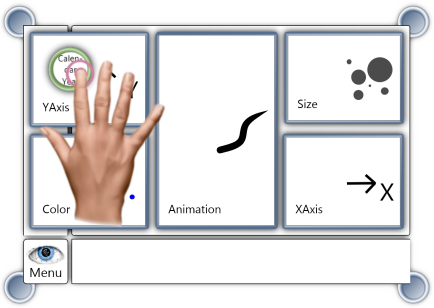


Abb. 4: Belegung per *Drag and Drop*

Erstellen benutzerdefinierter Diagramme

Durch *Drag and Drop* können die Elemente des Pie-Menüs auf ein Diagramm gezogen werden um so Abhängigkeiten zwischen kompatiblen Dimensionen und Kennzahlen zu visualisieren.

Mehrdimensionalität

Mit Hilfe eines Punktdiagramms können zunächst nur zwei Größen miteinander in Beziehung gesetzt werden. *TaP* kann weitere Größen berücksichtigen, welche dann durch den Flächeninhalt bzw. die Farbe der Diagrammpunkte repräsentiert werden. Zusätzlich ermöglicht *TaP* eine Animation der Daten, um z. B. zeitliche Verläufe zu visualisieren.

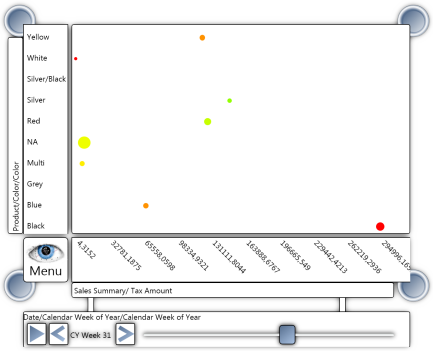


Abb. 5: Größen- und Farbdimension und Animation

Kollaboratives Arbeiten

TaP unterstützt exploratives Arbeiten an mehreren Diagrammen zugleich, wodurch kollaborative Teamarbeit gefördert wird. Ein Analyst kann z. B. ein Diagramm an einer beliebigen Stelle der Arbeitsfläche erzeugen und seine Ergebnisse anschließend präsentieren. Seine Kollegen können bei Bedarf die Arbeit an dem Diagramm fortsetzen.

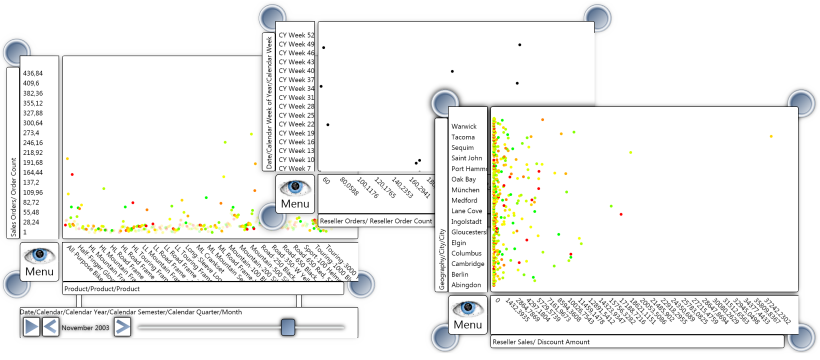


Abbildung 6: Kollaboration

Inhaltsangabe

Abbildungen

Tabellen

Teil I

Einführung

Kapitel 1

Einleitung

Dieser Abschlussbericht dokumentiert die Ergebnisse unserer Projektgruppe *Visual Analytics*, die in Kooperation mit dem OFFIS-Bereich Gesundheit in der Zeit von Anfang Oktober 2008 bis Ende September 2009 an der Carl von Ossietzky Universität Oldenburg stattfand. In diesem Zeitraum haben wir uns mit der Entwicklung einer Software zur visuellen Datenanalyse beschäftigt. Die Zielsetzung des Projektes sah vor, dass unsere Anwendung auf einem Multitouch-TableTop-Computer[1] eingesetzt werden soll. Die Interaktion mit dem System ist daher vollständig gestenbasiert umgesetzt worden. Das heißt, dass alle Interaktionen auf einer berührungssensitven Oberfläche ausgeführt werden. Dieser Einleitungsabschnitt motiviert zunächst den Einsatz der visuellen Analyse und den von Multitouch-Technologien und es folgt eine Darstellung dessen, was wir uns von der Kombination dieser beiden Verfahren versprechen. Nach dieser kurzen Einführung folgt ein Überblick, der den weiteren Aufbau dieses Dokuments beschreibt.

Motivation

Die rasante Entwicklung in der Informationstechnologie ermöglicht die Speicherung enormer Datenmengen. Diese Daten füllen riesige Datenbanken und können längst nicht mehr manuell überblickt werden. Die Schwierigkeit besteht darin, aus der Flut von Daten, die häufig versteckten Informationen zu gewinnen. Durch eine Analyse wird es ermöglicht, Rückschlüsse zu ziehen und Entscheidungen zu treffen. Klassische *Data Mining* Verfahren scheitern bisher an der Analyse dieser komplexen und enormen Datenmengen. Dies ist in der Datenqualität, den häufig komplexen Auswertungsalgorithmen und an der großen Datenmenge begründet [Sch07]. Die visuelle Datenexploration stellt eine mögliche Lösung dar, in der der Mensch mit seinem intuitiven Vorgehen, seinem Allgemeinwissen und seiner Fähigkeit zu Kombinieren in den Explorationsprozess eingebunden wird. Der Analyst steuert dabei den Analyseablauf, beurteilt Teilergebnisse und entscheidet daraufhin, ob er in eine

[1] „*Als TableTop-Arbeitsplatz bezeichnet man tischförmige Arbeitsplätze mit horizontaler Oberfläche, auf der Aufgaben erledigt werden.*“ [FHTA09]

Richtung weiter exploriert oder ob er seine bisherigen Ergebnisse verwirft. Durch die Einbindung des Menschen in den Explorationsprozess entstehen folgende Vorteile:

- Der Mensch kann im Gegensatz zu *Data Mining* Verfahren sehr leicht auch inhomogene Daten erfassen.
- Klassische Analysetechniken müssen in der Regel speziell für neuartige Datensätze eingerichtet werden. Bei der visuellen Datenexploration entfällt dieser Schritt.
- Der visuelle Datenexplorationsprozess gestattet durch seine Einfachheit, dass auch Nichtexperten eine Analyse durchführen können.

Klassische Eingabegeräte, beispielsweise Maus und Tastatur, ermöglichen nur eine indirekte Interaktion mit einer Anwendung. Eine direkte Interaktion, wie sie beispielsweise über Touchscreens möglich ist, gilt hingegen als intuitiver, effektiver und schneller erlernbar [Sch08, Shn83]. Ein typisches Beispiel stellen Fotoanwendungen für Multitouch-Geräte dar, die es auf sehr intuitive Weise erlauben, Bilder mit den Fingern, ähnlich wie auch reale Objekte, zu verschieben.

Multitouch-Eingabegeräte zeichnen sich dadurch aus, dass sie mehrere Berührungen zugleich verarbeiten können. Diese können zum einen als komplexe Gesten interpretiert werden, können zum anderen aber auch als voneinander unabhängige Eingaben behandelt werden. Multitouch-Eingabegeräte sind schon seit langer Zeit im akademischen Bereich bekannt (vgl. [Bux07]). Mit der Entwicklung der *Frustrated Total Internal Reflection (FTIR)*-Technologie durch Jefferson Y. Han wurde auch das allgemeine Interesse an diesen Eingabegeräten geweckt (vgl. [SBD+08]). Da gleichzeitige Eingaben unabhängig voneinander verarbeitet werden können, eröffnen große Multitouch-Touchscreens zudem Möglichkeiten für kollaboratives Arbeiten.

Der typische Analyseablauf besteht aus einem Zyklus, in dem der Analyst die Visualisierung als Verifikations- und Hypothesenwerkzeug nutzt. Hierfür muss der Analyst verschiedene Operationen auf der Visualisierung durchführen, wie zum Beispiel Zoomen, Verfeinern oder ein neues Diagramm erstellen. Mittels eines Multitouch-Touchscreens ist eine direktere Interaktion mit den Visualisierungen möglich, was zu einer dynamischeren Art der Analyse führt. Da bei Visual Analytics besonders die Auffassungsgabe des Menschen eine Rolle spielt, bietet sich die Möglichkeit des kollaborativen Arbeitens auf einem Multitouch-Eingabegerät an, um im Team neue Erkenntnisse aus Visualisierungen zu gewinnen.

Aufbau des Berichts

Dieser Bericht fasst die gestellte Problemstellung und die entwickelten Lösungen der Projektgruppe *Visual Analytics* zusammen. Der Bericht ist in sechs Teile gegliedert. Nach dieser Einleitung folgt der Teil *Grundlagen*, in dem Grundlagen zu den Themen Visual

Analytics, Multitouch, *Graphical User Interface (GUI)*-Entwicklung für Multitouch-Technologien und Usability vorgestellt werden.

Der Abschnitt *Visual Analytics Prozess* stellt den Prozess der visuellen, explorativen Analyse vor. Durch den darauf folgenden Abschnitt *Visualisierungsformen* erhält der Leser einen Einblick in bestehende Möglichkeiten der Visualisierung von Daten, welche u. a. Anwendung im Bereich der visuellen Analyse haben können. Die zu analysierende Datenmenge ist häufig multidimensional [Kei02]. Für diese Datenformen ist eine spezielle Datenbankform entwickelt worden, welche im anschließenden Abschnitt dargestellt wird.

Im Kapitel *Multitouch* wird die Integration eines Multitouch-Eingabegerätes in eine Anwendung beschrieben. Hierzu werden zunächst einige Technologien zur Erkennung von Berührungen im Abschnitt *Visuelle Multitouch-Technologien* vorgestellt und anschließend beschrieben, wie die erfassten Daten zu Mustern umgesetzt werden und mit Hilfe eines speziellen Protokolls zur Hauptanwendung weitergeleitet werden können. Die Erkennung von Gesten für die Umsetzung der Berührungen in Steuerungsereignisse schließt das Kapitel ab.

Das Kapitel über *Programmoberflächenentwicklung für Multitouch-Technologien* enthält eine Abwägung einiger Technologien zur *GUI*-Entwicklung. Dort werden die Eigenschaften der betrachteten Toolkits aufgeführt und bewertet. Es folgt eine Einführung in die Verwendung der *Windows Presentation Foundation (WPF)*. Das Kapitel schließt mit einem Überblick über bestehende Multitouch-Frameworks und ihrer Bewertung ab.

Im Kapitel *Usability* gibt es eine Einleitung in die Grundlagen des Usability-Engineering. Dort wird u. a. beschrieben, wann eine Geste als angenehm empfunden wird. Auf das Prototyping wird als eine Methode des Usability-Engineering in einem eigenen Abschnitt eingegangen. Als weitere Technik zur Untersuchung der Gebrauchstauglichkeit wird die Methode der Evaluation vorgestellt. Die Einführung von Programmfunktionen, die dem Anwender nicht bekannt sind, wird im Abschnitt *Hilfe und Dokumentation* behandelt. Das Kapitel schließt mit der Nennung von Standards für den Entwurf von Benutzungsoberflächen ab.

Leser, die auf einem dieser Gebiete keine bis wenig Erfahrung haben, sollten in dem entsprechenden Grundlagenkapitel nachlesen.

Projektplanungsinteressierte Leser finden im anschließenden Teil zur *Projektplanung* Informationen zur Projektorganisation und zur Anforderungsanalyse.

Für Entwickler ist besonders der Teil *Umsetzung* interessant. Hier wird zunächst das gesamte System beschrieben, bevor dann die Entwurfsentscheidungen der einzelnen Komponenten von *TaP* dargestellt werden. Nach dem Kapitel *Entwurf* wird die Umsetzung der Entwurfsentscheidungen im Kapitel *Implementierung* erläutert.

Anschließend werden mögliche Erweiterungen der Komponenten analysiert und Schritte zur Umsetzung dieser Erweiterungen gezeigt.

Damit geprüft werden kann, ob die Bedienbarkeit des Systems den Erwartungen entspricht, sind Evaluationen durchgeführt worden, welche im Teil *Qualitätssicherung und Evaluationen* beschrieben sind. Außerdem sind hier Maßnahmen zur Qualitätssicherung festgehalten worden.

Abschließend werden im Teil *Abschluss* die Ergebnisse der Projektgruppe zusammengefasst und ein Ausblick gegeben. Im Anschluss daran gibt die Projektgruppe noch ein persönliches Fazit ab.

Teil II

Grundlagen

Kapitel 2

Visual Analytics

In *Visual Analytics*-Anwendungen wird der Anwender stark in den Analyseprozess einbezogen. Solche Verfahren profitieren von den Fähigkeiten des Menschen und können rein automatisierten Verfahren überlegen sein. Eine Anwendung, die für solch eine visuelle Analyse genutzt werden soll, muss überaus flexibel sein und es dem Analysten jederzeit erlauben den Kontrollfluss zu steuern. Der Schwerpunkt dieser Arbeit besteht in der Umsetzung einer Umgebung zur visuellen Analyse. In diesem Kapitel werden daher die Grundlagen des *Visual Analytics*-Prozesses sowie verwandte Technologien vorgestellt. Im ersten Abschnitt erfolgt eine Definition des *Visual Analytics*-Begriffs und es werden einige Anwendungsmöglichkeiten vorgestellt. In dem darauf folgenden Abschnitt wird allgemeiner auf verschiedene Formen der Datenvisualisierung eingegangen, die in der visuellen Analyse eingesetzt werden können. Eine besondere Herausforderung stellen hierbei hochdimensionale Daten dar, denn diese erfordern zum einen geeignete Visualisierungsformen, zum anderen aber auch ein angemessenes Datenbanksystem, das in der Lage ist sie zu verwalten. Den Abschluss dieses Kapitels bildet daher eine Einführung in *Online Analytical Processing*-Systeme.

2.1 Der Visual Analytics Prozess

Visual Analytics ist eine Methode zur visuellen Datenanalyse. Im Gegensatz zur automatisierten Analyse durch spezialisierte Algorithmen wird der Mensch stärker in die Auswertungsphase eingebunden. Bereits der Name des Verfahrens lässt auf eine Art Visualisierung schließen, die für Verkaufs- und Präsentationszwecke wesentlich besser geeignet ist.

Die visuelle Exploration von Daten hielt bereits in der zweiten Hälfte der 70er Jahre Einzug in die Statistik. Die Grundlagen zur visuellen Datenanalyse wurden 1977 durch J.W. Tukey in seinem revolutionären Lehrbuch gelegt. Mit der Idee „*Looking at data to see what it seems to say*“ [Tuk77] beschreibt J.W. Tukey einen völlig neuen Ansatz. Sein Konzept, die Daten zunächst grafisch darzustellen und anhand dieser auf eine Hypothese

zu schließen, welche durch die konfirmatorische Statistik bestätigt oder widerlegt wird, war zur damaligen Zeit bahnbrechend. Seit Anfang der 80er Jahre haben die statistisch grafischen Methoden zum Aufdecken von Mustern wachsende Bedeutung erhalten. Hier wurden unter dem Stichwort „*Visual Exploratory Data Analysis*“ [Tuf83] rein grafische und auf mehrdimensionalen Daten bezogene Visualisierungen zur Erkennung von Wesenszügen und Anomalien verwendet. Auf den Beweis durch die schließende Statistik wird häufig verzichtet. (vgl. [Deg06])

Im Folgenden wird nun diese andere Herangehensweise an die Informationsextraktion dargestellt. Dem Leser wird durch zwei Definitionen ein Einblick in das Verfahren gewährt und anschließend ein allgemeines Vorgehensmodell zur visuellen Analyse vorgestellt.

2.1.1 Definition

Nach [SM00] wird *Visual Analytics* als „*...das Erzeugen visueller Repräsentationen gegebener Daten, um deren effektive Auswertung durch den Anwender zu ermöglichen...*“ aufgefasst. Somit ist klargestellt, dass eine Visualisierung im Sinne von *Visual Analytics* dem Analysten als Hilfe dienen soll, die Daten in Informationen zu transformieren. Hingegen definieren [TC05] *Visual Analytics* als „*...the science of analytical reasoning facilitated by interactive visual interfaces...*“ und erweitern mit ihrer Auslegung die Definition um den Aspekt der interaktiven Steuerung. Zusammengefasst ist *Visual Analytics* die Technik visuelle Repräsentationen zu erstellen, welche die Daten auf eine geeignete Weise wiederspiegeln. Hierbei ist zu beachten, dass selten der gesamte Datenbestand auf einmal angezeigt werden kann. Der Aspekt der intuitiven Bedienung, woran diese Projektgruppe forscht, ist ein besonderes, unabdingbares Merkmal der Visualisierung.

2.1.2 Allgemeines Vorgehensmodell

Nach [Shn92] kann die visuelle Analyse in vier Schritte „*Analyse first – show the important – zoom, filter and analyse further – details on demand*“ untergliedert werden, was auch als *Visual Analytics Mantra* bezeichnet wird (vgl. [TC05]). *Visual Analytics* basiert auf sukzessiven Vertiefungen. Hierbei werden die Stärken zweier Welten, die des Computers und die des Menschen, miteinander vereint. Datenverarbeitende Geräte haben sicherlich Vorteile bei Berechnungen und bei der Suche von Datensätzen. Der Mensch hat mit seinem visuellen Kortex hingegen die Möglichkeit, sehr schnell Zusammenhänge zu erkennen oder zu filtern. Des Weiteren verfügt er über ein großes Allgemeinwissen und kann intuitiv Daten auswerten. Dieser Zusammenhang ist in Abbildung 2.1 dargestellt.

Abbildung 2.1: Vergleich zwischen den Fähigkeiten des Computers und des Menschen [SN]

Visual Analytics wird als Zyklus aus vier Elementen verstanden. Diese sind in der Abbildung 2.2 dargestellt. Die Grundidee hierbei ist, die Datensätze mit Hilfe von automatischen Algorithmen zu visualisieren. Somit dient die Visualisierung als Verifikation der Algorithmen. Aber auch der umgekehrte Weg, zunächst die Visualisierung und die daraus resultierende Hypothese durch Algorithmen beweisen zu lassen, ist denkbar. Dieses Vorgehen führt zu Wissen, welches wiederum auf die Datensätze angewendet werden kann.(vgl. [KMS+])

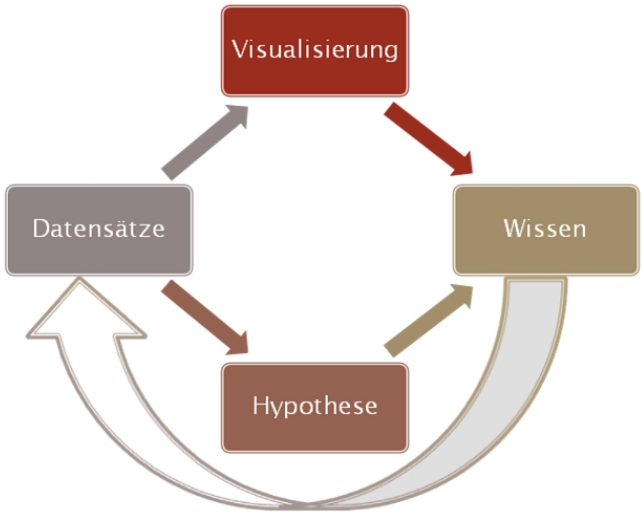


Abbildung 2.2: Allgemeines Vorgehensmodell Visual Analytics (nach [KMS+])

Auf Grund der Aufteilung der Analyse zwischen Computer und Mensch entsteht eine Symbiose, durch die einige Nachteile des klassischen *Data Mining* Ansatzes behoben werden. Es werden keine Experten für die Auswertung benötigt, was zu einer erheblichen Kostenreduktion führt. Natürlich wird für die Analyse weiterhin qualifiziertes Personal benötigt, welches jedoch nur geringe Kenntnis vom Datenanalyseverfahren haben muss. Somit können sich die Mitarbeiter mehr auf die eigentliche Auswertung konzentrieren. Außerdem erhält der Analytiker durch die Arbeit mit den Daten ein besseres Verständnis dieser, wodurch eine bessere Verifikation der Ergebnisse möglich ist. Im Vergleich mit *Data Mining* wird auch der Nachteil der Glaubwürdigkeit des Endergebnisses behoben. Da durch die sukzessive Approximation im *Visual Analytics* - Prozess die Zwischenergebnisse vom Analytiker kontrolliert werden, ist das Endergebnis nachvollziehbar und damit vertrauenswürdiger als beim herkömmlichen *Data Mining*. Zusammengefasst erhält der Anwender durch *Visual Analytics* die Möglichkeit, seine Intuition in den abstrakten algorithmischen Analyseprozess des *Data Mining* mit einzubringen. Hierdurch erhält er Einsicht in die charakteristische Struktur der Daten und kann daraus die Parametrierung der analytischen Methoden präziser und sicherer durchführen. (vgl. [SN, Ban06])

2.1.3 Anwendungsmöglichkeiten von Visual Analytics

Nachdem im vorhergegangenen Abschnitt das Grundmodell von *Visual Analytics* vorgestellt wurde, werden nun einige Anwendungsmöglichkeiten in Form einer nicht geschlossenen Auflistung präsentiert. Diese Beispiele dienen als Motivation für die Einbindung des *Visual Analytics* Prozesses in die Analyse.

Physik und Astronomie Ständig werden Daten über das Universum durch Teleskope und andere Messgeräte gesammelt. Die gesammelten Daten sind zunächst unstrukturiert und können nicht direkt weiterverarbeitet werden. Nur durch eine geeignete Analyse können Informationen aus den Daten gewonnen werden. Allerdings werden auch viele *Rauschdaten* mit aufgezeichnet. Aufgabe des Analysesystems ist es, die wichtigen Daten herauszufiltern. Die Komplexität der Daten führt dazu, dass automatische Algorithmen extrem kompliziert und dadurch langsam und unüberschaubar werden. Anhand einer visuellen Analyse gelingt die Auswertung wesentlich schneller und effektiver. (vgl. [KMS$^+$])

In der Abbildung 2.3 ist eine Supernova mit ihren diversen Kräften (Rotation, Magnetismus, Gravitation, etc.) dargestellt. Die Abbildung lässt einen Kern und eine Hülle vermuten. Diese Annahmen können auch von Nicht-Experten getroffen werden. Dies wäre nicht möglich, wenn nur die Daten ausgegeben werden würden. (vgl. [KMS$^+$])

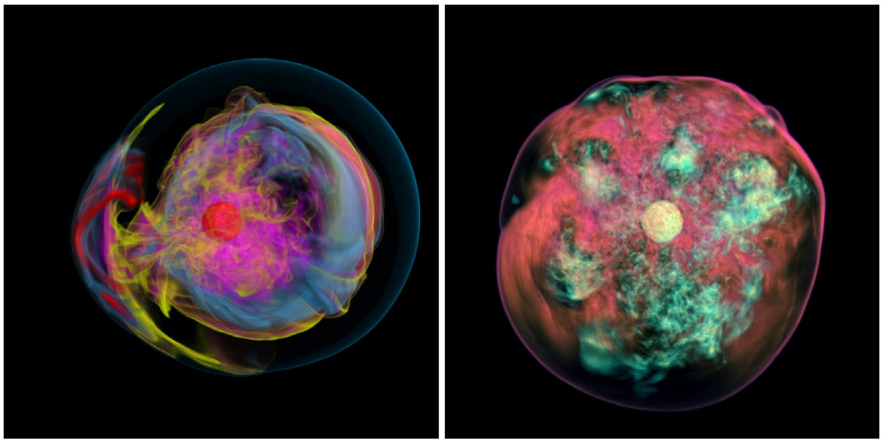


Abbildung 2.3: Darstellung einer Supernova mit Ihren Kräften [KMS+]

Wirtschaft Ein anderes großes Gebiet sind Wirtschaftsanwendungen. Der Finanzmarkt produziert mit seiner enormen Menge an Aktien, Fonds, Zertifikaten usw. Datenmengen, welche die Kursbewegungen anzeigen. Wunschziel jedes Analytikers ist die Vorhersage dieser Kurse. *Visual Analytics* stellt einen Schritt in diese Richtung dar, denn zumindest einige Kursbewegungen können mit dieser Datenanalysetechnik begründet werden. Abbildung 2.4 ist ein Beispiel aus dem FinDEx-System, ein Programm zur Begründung von Kursbewegungen. Es ermöglicht den Vergleich einer Aktie mit dem Leitindex und stellt die verschiedenen Kursbewegungen grafisch dar. Hierbei werden sowohl die absolute (kleines Dreieck) als auch die relative (großes Dreieck) Performance aufgezeigt.(vgl. [KMS+])

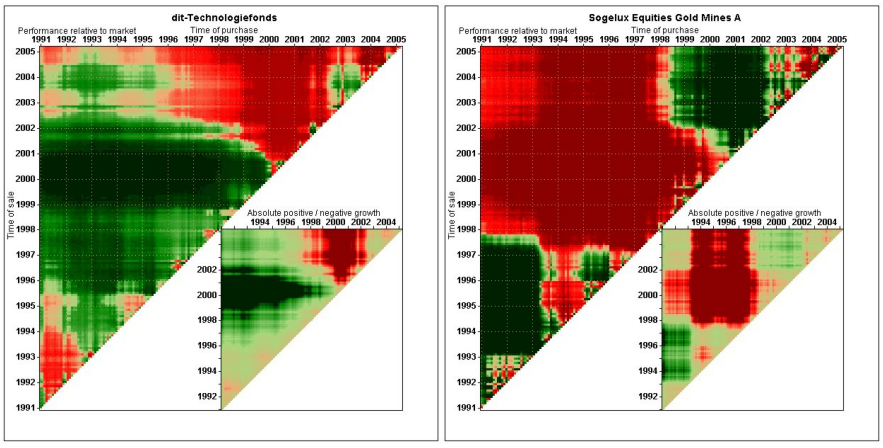


Abbildung 2.4: FinDEx System zur Darstellung der Marktperfomanz [KMS+]

Softwareanalyse Ziel der Softwareentwicklung ist die Realisierung performanter und korrekter Anwendungen. Jedoch hat sich erwiesen, dass zumindest bei nichttrivialen Programmen die Komplexität schnell steigt und sich hierdurch Fehler in ein System einschleichen können. Auch hier kann *Visual Analytics* genutzt werden, um Fehler zu finden, da der Quellcode im Grunde eine große Datenmenge ist, die nach Anomalien durchsucht werden kann. Gerade hier liegt das Spezialgebiet der visuellen Datenanalyse. Des Weiteren kann auch nach Optimierungsmöglichkeiten gesucht werden.(vgl. [KMS+])

Diese Auflistung stellt nur einen kleinen Einblick in eine Vielzahl von Anwendungsbereichen dar. Weitere Bereiche sind zum Beispiel Sicherheit oder aber auch Biologie, Medizin und Gesundheit.

2.2 Visualisierungsformen

Visual Analytics arbeitet, wie der Name bereits verrät, mit visuellen Darstellungen. In diesem Kapitel werden einige Visualisierungsformen vorgestellt. Zunächst wird auf Beispiele von Diagrammen eingegangen, um ein Gefühl für die Visualisierungsproblematik zu erhalten. Anschließend wird eine Klassifizierung von Diagrammen gegeben.

2.2.1 Diagrammformen

- Liniendiagramme

  Das *Liniendiagramm* ermöglicht die Darstellung des Bezugs zweier Merkmale zueinander. Dabei wird jedes Merkmal auf eine der zwei Achsen des Koordinatensystems aufgetragen. In der Regel wird jeder Messwert als Punkt an der entsprechenden Stelle gezeichnet. Diese Punkte werden häufig mit Linien verbunden. Dadurch entsteht ein Linienzug. Es ist zu beachten, dass oft keine Zwischenwerte existieren und die Linien nur zur Verdeutlichung dienen. Sie eignen sich besonders gut, wenn ein Merkmal pro Zeiteinheit betrachtet werden soll. Darüber hinaus lassen sich mit *Liniendiagramme* Trends erkennen. Abbildung 2.5 zeigt ein *Liniendiagramm* mit beispielhaften Daten.

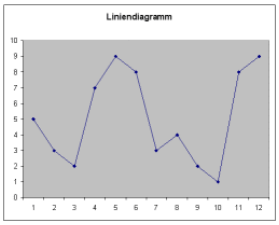


Abb. 2.5: Bsp. für ein *Liniendiagramm* [Wik09a]

- Säulendiagramme

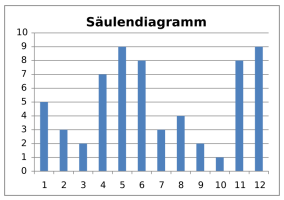


Abb. 2.6: Bsp. für ein *Säulendiagramm* [Wik09c]

Wie beim *Liniendiagramm* werden auch beim *Säulendiagramm* pro Achse ein Merkmal aufgetragen, allerdings wird statt eines Punktes eine vertikale Säule verwendet. Ein *Säulendiagramm* umschließt alle Werte einer Säule bis zu ihrem Endpunkt. *Säulendiagramme* eignen sich für Vergleiche und Profildarstellungen. Ein gewisser Nachteil ist, dass sie nur anwendbar sind, wenn es sich um eine relativ kleine Menge von Messwerten handelt. Zu empfehlen sind bis zu 15 Säulen, ansonsten verliert diese Darstellungsart schnell an Übersichtlichkeit. Angewendet werden die *Säulendiagramme*, wenn Verhältnisse verdeutlicht werden sollen. Abbildung 2.6 zeigt die Datensätze aus Abbildung 2.5 als *Säulendiagramm*.

- Kreisdiagramme

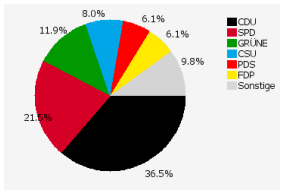


Abb. 2.7: Bsp. für ein *Kreisdiagramm* [Par09]

Das *Kreisdiagramm* ist eine Darstellungsform, die Teilwerte eines Ganzen aufzeigt. Das Ganze wird als Kreis abgebildet und ist in mehrere Kreissektoren aufgeteilt. Jeder Kreissektor besitzt einen Teilwert, somit ist der Kreis die Summe aller Teilwerte. Das *Kreisdiagramm* dient zur Übersicht von Verteilung und Anteilen vom Ganzen. Um das Prinzip aufrecht zu erhalten, sollte eine gewisse Anzahl von Sektoren nicht überschritten werden. Ferner empfiehlt sich eine Sortierung der Anteile nach ihrer Größe. In der Regel sollte immer mit dem größten Wert begonnen werden, zum Beispiel auf der positiven X-Achse des Koordinatensystems, welches durch den Mittelpunkt gebildet werden kann. Abbildung 2.7 zeigt eine beispielhafte Stimmenverteilung nach Parteien.

- Streudiagramme

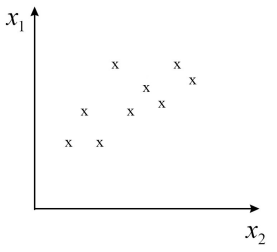


Abb. 2.8: Bsp.: *Streudiagramm* [Deg06]

Dieser Darstellungstyp der Visualisierung ist wohl einer der bekanntesten und gebräuchlichsten im wissenschaftlichen Bereich. Es wird die Streuung zweier Merkmale dargestellt und somit kann die Art, Richtung und Intensität des Zusammenhanges zwischen diesen beiden Merkmalen visualisiert werden. Die Gruppenbildung und Identifikation ungewöhnlicher Beobachtungen ist leicht möglich. Für große Datenmengen bzw. bei großer Streuung ist es empfehlenswert, das Diagramm mit weiteren Informationen anzureichern,

wie zum Beispiel durch Glättung (Regressionsgerade) oder Eintragung von Dichte-Ellipsen für bestimmte Prozentwerte des Datenmaterials. Ein drittes Merkmal kann zum Beispiel durch Farbinformationen aufgenommen werden oder durch Bubbles (der Datenpunkt wird je nach Ausprägung des dritten Merkmales in der Größe verändert). Eine andere Möglichkeit ist das sogenannte Slicing, welches eine Projektionsvorschrift für die 2D-Visualsierung des resultierenden Würfels ist.(vgl. [Deg06, CM84])

- 3D-Streudiagramme

  3D-*Streudiagramme* sind perspektivisch – räumlich gezeichnete zweidimensionale *Streudiagramme*. Allerdings ist es hier schwierig, Muster genau zu erkennen und von daher nur für eine niedrige Anzahl von Beobachtungen empfehlenswert. Mitte der 70er Jahre wurden erstmals dynamisch - grafische Analysetechniken, wie zum Beispiel die Rotation, an diesen Diagrammen durchgeführt. (vgl. [Deg06])

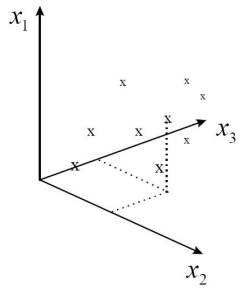


Abb. 2.9: Bsp.: 3D-*Streudiagramm* [Deg06]

- Blasendiagramme

  Das *Blasendiagramm* ist ein weiterentwickeltes *Streudiagramm*. Zwei der möglichen Merkmale werden, wie in einem *Streudiagramm*, aufgetragen. Hingegen wird das dritte Merkmal mittels der Größe der Blase, auch Bubble genannt, aufgezeigt. Ferner kann ein weiteres Merkmal durch die Farbe der Blase miteinbezogen werden. Abbildung 2.10 zeigt den Marktanteil eines Beispielunternehmens im Vergleich zu Mitarbeiterzahl und Umsatz. Nachteil dieser Darstellungsform ist die geringe Anzahl von Blasen, die in einem Diagramm vorkommen können. Bei einer zu großen Anzahl von Blasen besteht die Gefahr, dass sich diese überlappen.

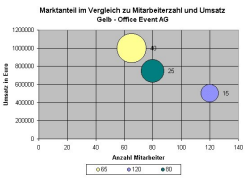


Abb. 2.10: Bsp. für ein *Blasendiagramm* [Hof09]

- Netzdiagramme

  Ein *Netzdiagramm* ist auch als Spinnennetz- oder Radardiagramm bekannt. Das Besondere an dieser Darstellungsform ist, dass die Wertepaare nicht in einem Koordinatensystem aufgetragen werden, sondern dass ein Netz als Basis dient. Hierbei werden mehrere gleichwertige Kategorien in Form eines Spinnennetzes präsentiert.

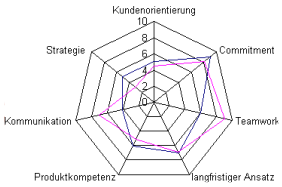


Abb. 2.11: Bsp. für ein *Netzdiagramm* [Klu09]

Für jede Kategorie ist genau eine Achse vorgesehen. Die Achsen werden von einem Mittelpunkt aus kreisförmig und gleichmäßig angeordnet. Die Bewertung ergibt sich durch den relativen Abstand des Bewertungspunktes zum Mittelpunkt. Außen liegende Punkte würden dabei den Idealzustand oder 100% einer Leistung oder Qualität symbolisieren. Außerdem werden die Bewertungspunkte mit Linien verbunden. Dadurch ergeben sich Flächen, die zur weiteren Verdeutlichung oft gefärbt werden. Für das Aufzeigen mehrerer Ebenen werden unterschiedliche Farben verwendet. Damit das Prinzip funktioniert, müssen mindestens drei Kategorien vorhanden sein. Folglich wäre bei nur zwei Achsen keine Verbindung ersichtlich. Bei mehr als zehn Achsen verliert das Diagramm an Übersicht. Optimal sind fünf bis sieben Achsen. Das *Netzdiagramm* eignet sich gut für den Vergleich z.B. zwischen Ist- und Sollzustand. Abbildung 2.11 zeigt die Situation in einem Unternehmen.

- Mosaikplot

  Der *Mosaikplot* ist ein grafisches Verfahren, welches der Visualisierung von Datensätzen mit zwei oder mehr qualitativen Merkmalen dient. In erster Linie bietet das *Mosaikplot* einen Überblick über die Daten und ermöglicht das Erkennen von Zusammenhängen. Im Allgemeinen sind die darzustellenden Merkmale in Rechtecke aufgeteilt. Die Flächen der rechteckigen Felder sind proportional zur Anzahl der Beobachtungen. Die Anzahl der Beobachtungen ist nicht limitiert, aber auch nicht ablesbar. Ferner sollten mindestens zwei Merkmale existieren. Es gibt keine Regel, die die maximale Anzahl von Merkmalen begrenzt. Es sollte aber beachtet werden, dass eine zu große Anzahl eher hinderlich ist, da die Grafik unübersichtlich wird. Abbildung 2.12 zeigt ein Beispiel eines *Mosaikplot*s aus [Wik09b]. Der für dieses Beispiel verwendete Datensatz hat 2201 Beobachtungen und 3 Merkmale.

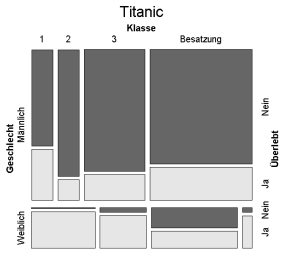


Abb. 2.12: Bsp. für einen *Mosaikplot* [Wik09b]

- Bildsymbole Ein ganz anderer Weg als die bisher vorgestellten Diagramme wird durch die Gesichtserkennung durchgeführt. Die Idee besteht darin, Merkmalen bestimmte Gesichtsformen zuzuweisen und damit einen Datensatz in ein Gesicht zu transformieren. Hierdurch lassen sich in der größten Ausführung maximal 36 Merkmale gleichzeitig darstellen. Zur Auswertung wird die menschliche Fähigkeit der schnellen Wiedererkennung von Gesichtern ausgenutzt. Die Schwierigkeit besteht hier jedoch darin, geeignete Kombinationen der Gesichtszüge für die verschiedenen Merkmale zu finden. Außerdem ist dieses Verfahren nur für einige wenige Datensätze zu verwenden, da für jeden Datensatz ein Gesicht angezeigt werden muss, und somit die Übersicht leicht verloren geht. (vgl. [Deg06])

Weitere Möglichkeiten der bildlichen Auswertung sind Histogramme oder Glyphs (Relieffiguren). Diese Verfahren können allerdings nur bei ca. acht bis zwölf Merkmalen genutzt werden, da ansonsten ein Vergleich zwischen den Objekten nicht mehr möglich ist. [Deg06]

- Kurvenprofile

  Die auch Andrew-Plots genannten Darstellungen verfolgen eine funktionale Interpretation des Datensatzes durch Sinus- und Cosinus-Funktionen und stellen diese dar. Damit ist ein Datensatz eine Linie im Plot. Gleiche bzw. ähnliche Datensätze werden durch eine sehr ähnliche Funktion repräsentiert. Damit lassen sich leicht Anomalien erkennen, da diese komplett anders dargestellt werden. (vgl. [Deg06, And72])

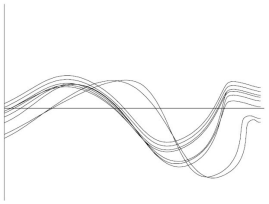


Abb. 2.13: Beispielhafter Andrew's Plot [Deg06]

2.2.2 Kartenformen

- Standortkarten

  Standortkarten kommen zum Einsatz, wenn qualitative Informationen lagegetreu dargestellt werden sollen. Der Betrachter erhält Auskunft darüber, an welchen Ort die Sachdaten gebunden sind. Das bedeutet aber nicht, dass sich dieser Zusammenhang exakt auf einen einzelnen Ort bezieht, sondern dieser kann sich auch auf eine größere Fläche, wie z. B. Gemeinde oder Wahlbezirke, beziehen. Die qualitativen Informationen werden mit Hilfe von Signaturen, die an die entsprechende Stellen platziert werden, verdeutlicht. Signaturen sind in der Kartographie Kartenzeichen oder Symbole. Es wird zwischen punkthaften, linienhaften und flächenhaften Signaturen unterschieden. Punkthafte Signaturen können nach [HGM02] in verschiedene Kategorien aufgeteilt werden. Sie können von einfachen Punkten, über kleine Bilder und geometrische Symbole, bis hin zu Buchstaben, Ziffern und Zahlen variieren. Durch ihre Form stellen z. B. bildhafte Signaturen einen direkten Bezug zur dargestellten Information dar. Abbildung 2.14 zeigt eine beispielhafte Standortkarte.

Abb. 2.14: Beispiel *Standortkarte*

- Choroplethenkarten

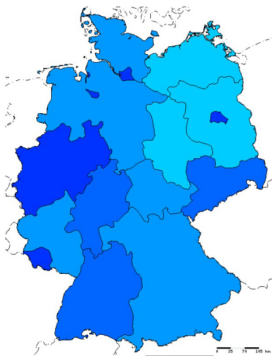


Abb. 2.15: Beispiel *Choroplethenkarte* [Phi]

Bei der *Choroplethenkarte* handelt es sich um eine flächenbezogene Karte, die über eine Vielzahl von raumbezogenen Informationen verfügt. Hierbei werden quantitative Daten in Relation zueinander betrachtet. Nach [OQS] ist dies die am häufigsten verwendete flächenhafte Darstellungsform.
Choroplethenkarten bestehen aus *echten* und *unechten* Flächen. Eine Fläche wird als *echt* bezeichnet, wenn die durch sie dargestellten Eigenschaften der Realität entsprechen. Das sind u. a. Wasser- und Anbauflächen. Echte Flächen werden jedoch selten auf der *Choroplethenkarte* dargestellt. Entsprechend liefern *unechte* Flächen Informationen, die sich aus dem Bereich durchschnittlich oder vorherrschend ergeben. Abbildung 2.15 zeigt eine Karte der Bevölkerungsdichte in Deutschland, getrennt nach Bundesländern. Dabei gilt, je dunkler die blaue Farbe, desto dichter ist das Gebiet besiedelt. Ein Nachteil dieser Darstellungsform ist die unpräzise Information, die sie übermittelt. Diese *Choroplethenkarte* liefert nur Auskunft über die durchschnittliche Bevölkerungsdichte eines bestimmten Gebiets und keine genauen Zahlen. Tatsächlich kann die Bevölkerungsdichte innerhalb dieser Gebiete sehr unterschiedlich sein. Wie [Phi] warnt, sollten die Gebiete sehr gewissenhaft ausgewählt werden, um die Karte aussagekräftig und informativ zu halten.

- Mehrschichtige Karten

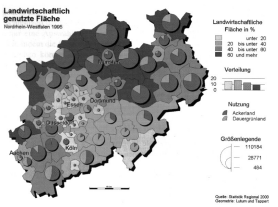


Abb. 2.16: Beispiel *Mehrschichtige Karte* [OQS]

Die *Mehrschichtige Karte* ist in der Lage, verschiedene Sachdaten in einer einzigen Karte abzubilden. Dies findet durch eine Überlagerung mehrerer Kartenschichten statt. Ziel dieser besonderen Kartenform ist die Untersuchung von möglichen, durch den räumlichen Bezug miteinander in Verbindung stehenden, Daten. Es können z. B. Informationen über Fischvorräte und die Wasserverschmutzung auf der Karte untergebracht und deren Zusammenhänge analysiert werden. Abbildung 2.16 zeigt an einem Beispiel, wie *Kreisdiagramme* in eine *Choroplethenkarte* eingebettet sein können. Die *Choroplethenkarte* liefert Informationen über die prozentuale landwirtschaftliche Flächennutzung und die *Kreisdiagramme* ihrerseits informieren zusätzlich über den verhältnismäßigen Anteil der Flächen als Dauergrün- bzw. Ackerland.

2.2.3 Klassifizierung von Visualisierungsformen

Die Visualisierungen aus dem vorherigen Abschnitt können in verschiedene Kategorien eingeteilt werden, wodurch sie sich leichter vergleichen lassen. In diesem Abschnitt soll diese Kategorisierung vorgenommen werden.

- Standard 2D/3D Techniken

  Klassische Darstellungstechniken, wie zum Beispiel XY-Graphen, Histogramme oder Karten fallen unter dieser Kategorie. Die bereits vorgestellten *Streudiagramme* sind ebenfalls in dieser Kategorie einzuordnen. (vgl. [KMS+, Oel02, Kei02])

- Geometriebasierte Techniken

  Die Grundidee geometrischer Techniken basiert auf der Transformation und Projektion multidimensionaler Datensätze, um eine bessere Darstellung zu erreichen. Es werden hierbei sowohl 2D als auch 3D Visualisierungen verwendet. Es existiert eine Vielzahl von Darstellungsformen, die diese Grundidee verfolgen. Beispielhaft seien Scatterplots und Parallel Coordinates genannt. Scatterplots werden häufig verwendet um *Data Mining* Ergebnisse zu visualisieren und werden von den meisten Tabellenkalkulationsprogrammen direkt unterstützt. Jede Datendimension wird einer der beiden Achsen (X oder Y) zugeordnet und in dessen Abhängigkeit gezeichnet. Eine dreidimensionale Darstellung ist ebenso denkbar. Hingegen wird bei der wohl bekanntesten Technik, dem Parallel Coordinate, jede Dimension durch vertikale Achsen repräsentiert. Jeder Datensatz wird als Linie aufgetragen, welche die Achsen (Dimension) an der entsprechenden Wertigkeit schneidet. Es entsteht so, pro Datensatz, ein charakteristisches Polygon (siehe Abbildung 2.17). Das Tool XmdvTool[1] kann solche Graphen erzeugen. (vgl. [KMS+, Oel02, Kei02])

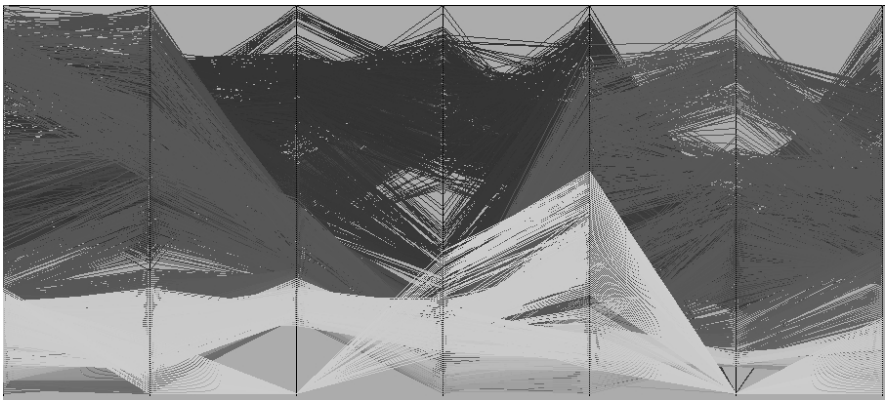


Abbildung 2.17: Beispielhafte Parallel Coordinate – Darstellung [Kei02]

[1] `http://davis.wpi.edu/~xmdv/`

- Icon-Techniken

  Eine andere Technik der Visualisierung sind Icon- oder Glyph-Techniken. Diese Techniken werden vor allem zur Darstellung von mehrdimensionalen Datensätzen verwendet und beruhen darauf, dass der Mensch Formen und Farben sehr schnell unterscheiden und wiedererkennen kann. Für jeden Datensatz wird meist eine eigene Form erzeugt. Bei der Icon-Technik wird eine bestimmte Eigenschaft einer Form der Ausprägung eines Merkmales zugeordnet, wie zum Beispiel Form oder Farbe. Die aus dem vorherigen Abschnitt bekannten Bildsymbole sind nur ein Beispiel für solch eine Zuordnung. Bei diesem Verfahren besteht die Schwierigkeit darin, die entsprechenden Eigenschaften der Symbole den Dimensionen zuzuordnen. (vgl. [KMS$^+$, Oel02, Kei02])

- Pixel- und Voxel Techniken

  Die Idee der Pixel-Techniken beruht darauf, jedem Bildpunkt einen Datenpunkt zuzuordnen, indem der Bildpunkt eine entsprechende Farbe darstellt. Außerdem werden die Bildpunkte entsprechend der Dimension gruppiert. Die Datensätze sind somit über das Bild verteilt und stehen nur über die relative Position innerhalb der Teilbereiche in Beziehung zueinander. Der zweidimensionale Ansatz der Pixel-Techniken kann durch sogenannte Voxel-Techniken auf eine dreidimensionale Darstellung erweitert werden, wodurch noch weitere Datensätze angezeigt werden können (siehe Abbildung 2.18). (vgl. [KMS$^+$, Oel02, Kei02])

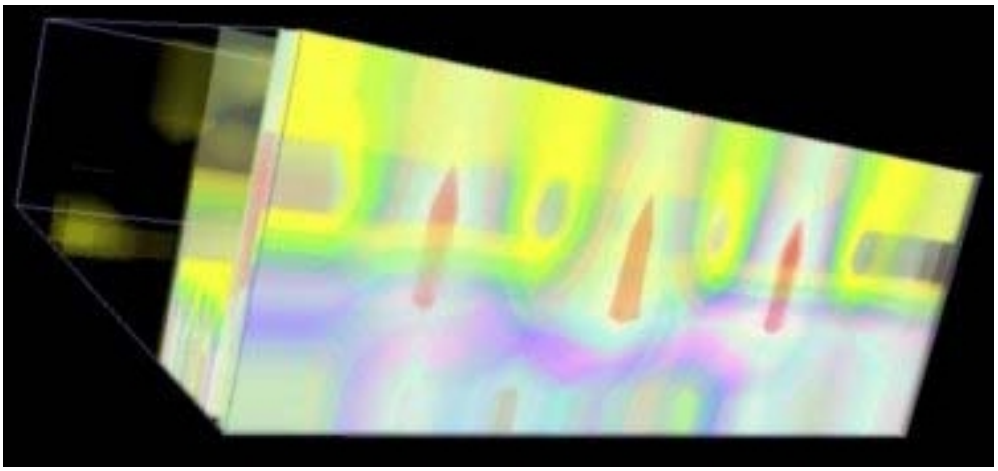


Abbildung 2.18: Beispielhafte Voxel–Darstellung [OIEE01]

- Geschachtelte Visualisierungen

  Geschachtelte Visualisierungen oder auch hierarchische und graphbasierte Techniken stellen Daten in Form hierarchisch untereinander aufgeteilter Untereinheiten dar. Die Attribute, die zum Aufbau der Hierarchie verwendet werden, müssen sorgfältig gewählt werden, da ansonsten das Ergebnis sehr stark verfälscht werden kann. Bekannte Vertreter dieser Technik sind Dimensional Stacking [LWW90], World-within-Worlds [FB90] und Treemaps [Shn92]. (vgl. [KMS+, Oel02, Kei02])

- Gemischte Techniken

  Neben diesen klar voneinander getrennten Ansätzen, gibt es auch gemischte Ansätze zur Darstellung der Daten. Zum Beispiel entsteht durch die Kombination von Icon und Scatterplots eine Art 3D-Glyph (siehe Abbildung 2.19). Durch die räumliche Aufteilung können drei Dimensionen (X,Y,Z-Achse) direkt dargestellt werden und müssen nicht mehr durch das Icon kodiert repräsentiert werden, wodurch eine einfachere und intuitivere Handhabung gewährleistet ist. (vgl. [KMS+, Oel02, Kei02])

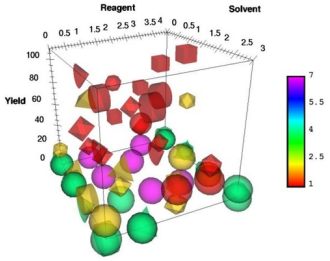


Abb. 2.19: Dreidimensionale Glyphen [Oel02]

Ein Problem ist jedoch bei allen vorgestellten Verfahren gleich. D.W. Scott hat dies wie folgt zusammengefasst: „...it is difficult, for example, that we could find true 7D features in data by purely graphical means without the guidance of a mathematical model ...“ [Sco92]. Aus diesem Grund ist eine Vorauswahl der Dimensionen und Kennzahlen, welche dargestellt werden sollen, in den meisten Fällen unabdingbar.

2.3 Online Analytical Processing

In diesem Abschnitt wird das *Online-Analytical Processing (OLAP)*-System vorgestellt. Dabei handelt es sich um eine spezielle Art von Datenbank, welche auf die Datenanalyse ausgerichtet ist. Nach der Einführung der wichtigsten Begriffe wird näher darauf eingegangen, wie bedeutend ein *OLAP*-Datenbanksystem für eine Analyse-Software ist. Diese Wichtigkeit wird mit Hilfe der Regeln aus (Codd & FASMI [CCS93, Pen08]) unterstützt.

2.3.1 Einführung

Im Gegensatz zu einer relationalen Datenbank werden die Daten in einer *OLAP*-Datenbank multidimensional verwaltet. Das heißt es stehen *Kennzahlen* im Vordergrund, z. B. Umsatz oder Bevölkerungszahl. Diese lassen sich mit Hilfe von *Dimensionen* aus unterschiedlichen Perspektiven betrachten. Orts- oder Zeitangaben können beispielsweise als *Dimensionen* aufgefasst werden.

Eine Abbildung solcher Daten wird *Cube* oder *Datenwürfel* genannt. Ein *Cube* besitzt eine Reihe von *Dimensionen* und *Kennzahlen*. Allerdings sind *Dimensionen* und *Kennzahlen* nicht an einen *Cube* gebunden, sondern können global auf einer multidimensionalen Datenbank verwendet werden.

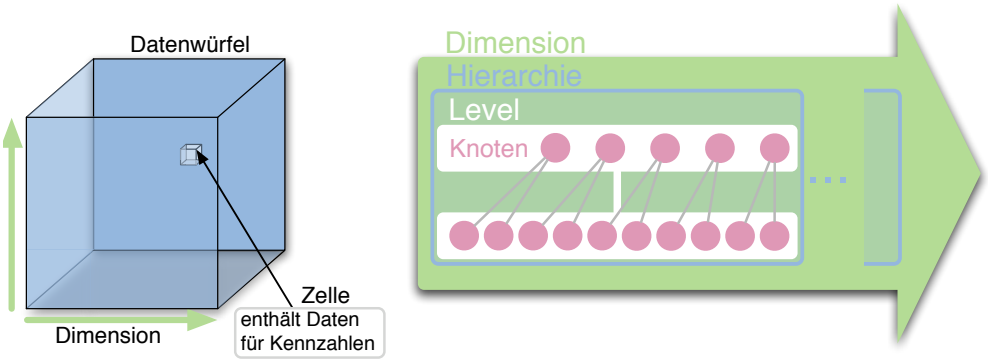


Abbildung 2.20: Schematische Darstellung eines Datenwürfels

2.3.2 Dimensionen

Eine *Dimension* beschreibt eine mögliche Sicht auf Daten. Sie besteht aus unterschiedlichen *Hierarchien*, die ihrerseits in mehrere Stufen unterteilt sind. Beispielsweise kann *Zeit* als *Dimension* folgende *Hierarchien* besitzen:

1. Jahr → Quartal → Monat → Tag
2. Jahr → Woche → Tag

Die unterschiedlichen *Hierarchiestufen* werden auch *Level* genannt. Ein *Level* hat eine Zahl von untergeordneten Elementen, welche *Knoten* genannt werden und erlauben daher jeweils die Navigation zu bestimmten Daten. Diese *Knoten* (engl. *Nodes*) repräsentieren die Koordinaten innerhalb des *Cube*s. Knoten der Hierarchiestufe *Jahr* sind z. B. *2004, 2005, 2006*.

2.3.3 Kennzahlen

Kennzahlen, auch *Fakten* oder engl. *Measure* genannt, sind in der Regel numerische Werte. Diese beinhalten z. B. die Daten wie *Bevölkerungszahl* . Diese können über mehrere *Hierarchiestufen* zusammengefasst werden, z. B. durch Addition oder Mittelwertbildung. Wenn *Kennzahlen* über die gesamte *Hierarchie* zusammengefasst werden, werden sie *additive Fakten* genannt. Falls keine Berechnung möglich ist, wird hier von *Nicht-additive Fakten* gesprochen.

Mehrere *Kennzahlen* können zu *Kennzahl-Gruppen* zusammengefasst werden. Dies dient einerseits der Übersicht und andererseits der Bildung von *Subcubes*, was jedoch nicht in allen multidimensionalen Datenbanken der Fall ist. Diese *Subcubes* sind Ausschnitte aus einem gesamten *Cube* in dem nicht alle *Kennzahlen* und *Dimensionen* zur Verfügung stehen. Nur *Dimensionen* und *Kennzahl*-Gruppen innerhalb eines *Subcubes* sind in einer Darstellung zusammen verwendbar. Somit ist es also wichtig zu wissen, zu welcher *Dimension* eine *Kennzahl* und anders herum verfügbar ist.

2.3.4 Operationen

Zur Datenanalyse stehen folgende Operationen für die Navigation in einem *Datenwürfel* zur Verfügung (vgl. [Kös07]).

Pivotierung Der *Datenwürfel* wird gedreht, wodurch eine neue Perspektive auf die Daten entsteht.

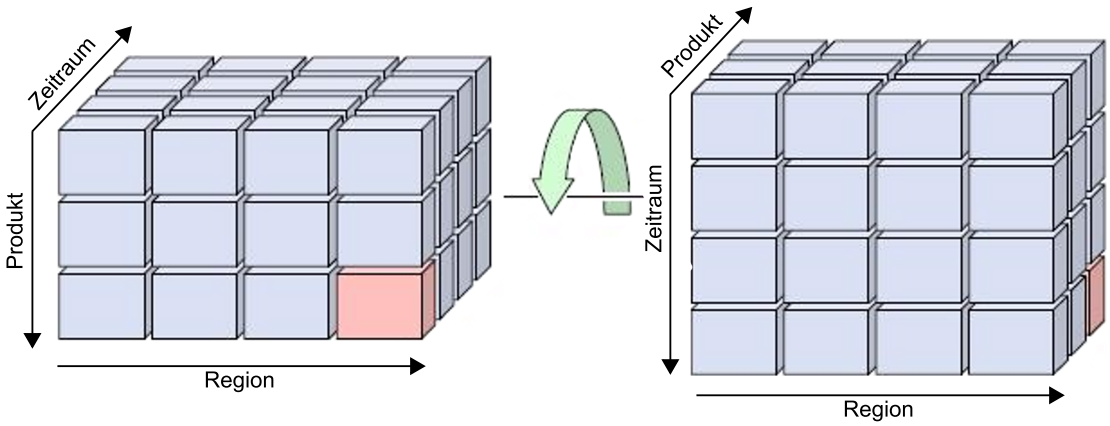


Abbildung 2.21: Pivotierung eines Datenwürfels [Kös07]

RollUp In der *Hierarchie* einer *Dimension* wird auf eine höhere Aggregationsstufe gewechselt.

DrillDown Hierbei entsteht eine höhere Detailstufe in den Daten durch das Wechseln auf eine tiefere Aggregationsstufe einer Hierarchie.

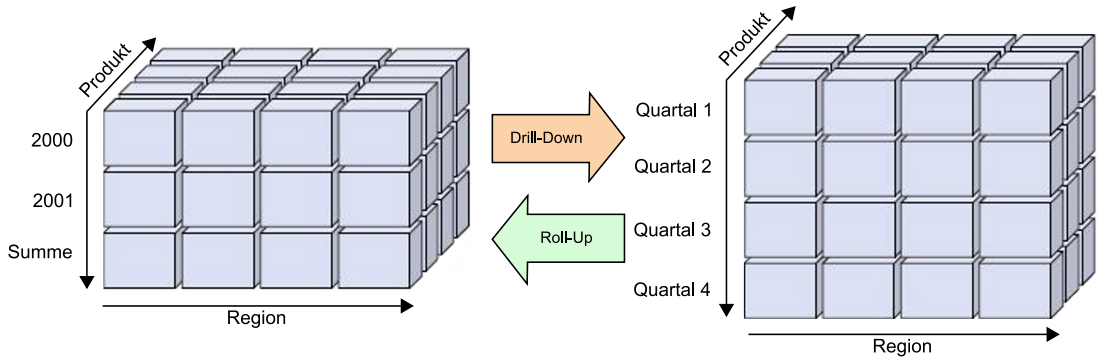


Abbildung 2.22: *RollUp* und *DrillDown* in einem *Datenwürfel* [Kös07]

DrillAcross Hiermit wird zwischen unterschiedlichen *Datenwürfel*n gewechselt.

DrillThrough Verzweigung auf eine andere meist feinere Datenquelle.

Slicing Herausschneiden einer Scheibe aus einem *Datenwürfel*. Hierbei wird eine *Dimension* reduziert auf ein einzelnes Element der Aggregationsstufe.

Dicing Herausschneiden eines Teilwürfels aus einem *Datenwürfel*. Sozusagen ein *DrillDown* auf allen *Dimensionen*.

2.3.5 Evaluation von OLAP-Datenbanken

Ein Datenbank-System lässt sich mit Hilfe zweier Tests und Regeln analysieren, um sie als *OLAP*-fähig zu identifizieren. Hierin erkennt man auch die Unterschiede einer *OLAP*-Datenbank zu einem *RDBMS*. Dies soll auch die Wichtigkeit einer *OLAP*-Datenbank für dieses Projekt unterstützen.

Evaluation nach Codd

1993 veröffentlichte Dr. Edgar Codd, der bereits Meilensteine in der Entwicklung relationaler Datenbanken in den 60ern und 70ern setzte, die so genannten *Codd rules and features*. Diese beinhalten 18 Regeln – ursprünglich waren es 12, welche 1995 erweitert wurden –, welche ein *OLAP*-System erfüllen sollte. (vgl. [CCS93, Pen08])

Basic Features B

- **Multidimensional Conceptual View** Diese Regel schreibt vor, dass eine *OLAP*-Datenbank multidimensional konzipiert sein muss. Dies ist eine der wichtigsten Voraussetzungen für eine *OLAP*-Datenbank.
- **Intuitive Data Manipulation**

  Datenmanipulation soll nach Dr. Codd durch direkte Interaktionen mit den Datenzellen geschehen und nicht über Umwege, wie z. B. durch ein Menü. Dies sollte über Benutzereingaben, wie mit einer Maus o. Ä. geschehen. Jedoch unterstützen heutzutage noch nicht sehr viele *OLAP*-Anwendungen einen hohen Umfang an Eingabemöglichkeiten wie beispielsweise *Drag and Drop*.
- **Accessibility: *OLAP* as Mediator**

  OLAP ist konzeptionell *Middleware*, also eine Komponente zwischen einer heterogenen Datenquelle und einer Datenvisualisierung.
- **Batch Extraction vs Interpretive**

  Diese Regel von Codd besagt, dass eine *OLAP*-Datenbank in der Lage sein soll, eigene sowie auch externe Daten zu verarbeiten. Es gibt bisher nur sehr wenige *OLAP*-Systeme, die diese Hybridität erfüllen.
- ***OLAP* Analysis Models**

  Dr. Codd gibt hiermit vor, dass ein *OLAP*–Produkt alle vier Analysemodelle unterstützen soll: kategorisch, erklärend, kontemplativ, formelhaft. Es soll Funktionen unterstützen, wie parametrisierte statische Berichte, *Slicing* und *Dicing* mit *DrillDown*, „*was wenn*“-Analysen und Zielsuche.
- **Client Server Architecture**

  Da Analysen von mehreren Clients durchführbar sein sollen, ist eine Client/Server-Architektur wichtig. Außerdem sollten für Anfragen an den Server keine Programmierkenntnisse erforderlich sein.
- **Transparency**

  Diese Regel besagt, dass die Benutzungsschnittstelle klar von der Datenarchitektur getrennt sein soll.
- **Multi-User Support**

  Obwohl die meisten *OLAP*-Anwendungen nur einen Lesezugriff bieten, sollten sie nach Codd auch die Möglichkeit der Bearbeitung der Daten beinhalten, mitsamt der notwendigen Integrität und Sicherheit.

Special Features S

- **Treatment of Non-Normalized Data**

  Eine Veränderung in der *OLAP*-Datenbank darf sich nicht auf eine unnormalisierte Datenbank auswirken.

- **Storing *OLAP* Results: Keeping Them Seperate from Source Data**

  Mit dieser Regel sagt Dr. Codd aus, dass die *OLAP*-Daten nie live durch die Quelle aktualisiert werden. Also muss die *OLAP*-Datenbank separiert sein, aber es muss leicht möglich sein, eine Aktualisierung durchzuführen.

- **Extraction of Missing Values**

  Dies bedeutet, dass nicht vorhandene Werte nicht als 0-Werte gesehen werden dürfen.

- **Treatment of Missing Values**

  Diese Regel sagt aus, dass `null`-Werte in einer *OLAP*-Analyse völlig ignoriert werden müssen.

Reporting Features R

- **Flexible Reporting**

  Dr. Codd fordert, dass alle *Dimensionen* gleich verwendet werden können, so wie der Nutzer es wünscht.

- **Uniform Reporting Performance**

  Die Leistung eines Berichtes darf nicht abhängig von der Anzahl der *Dimensionen* und Größe der Datenbank sein.

- **Automatic Adjustment of Physical Level**

  Die Form der Datenspeicherung soll, je nach Datenmodel und -größe, dynamisch sein.

- **Generic Dimensionality**

  Dimensionen sollen alle eine gleichartige Struktur und Funktionsfähigkeit haben. Jedoch ist es nach Codd erlaubt, ausgewählte *Dimensionen* in der Funktionsfähigkeit zu erweitern. Doch sollten diese Zusatzfunktionen für jede *Dimension* möglich sein.

Dimension Control D

- **Unlimited Dimensions and Aggregation Levels**

  Unbegrenzte *Dimensionen* und Aggregationsstufen sind natürlich technisch nicht möglich. Jedoch sollte eine *OLAP*-Anwendung nicht diese beiden Attribute zu stark begrenzen. Nach Codd sollte das Maximum der Aggregationsstufen zwischen 15 und 20 liegen.

- **Unrestricted Cross-dimensional Operations**

  Diese Regel sagt aus, dass Berechnungen nicht nur auf *Kennzahlen* sondern auch auf *Dimensionen* möglich sein sollen.

FASMI-Test

1994 beschäftigte sich Nigel Pendse von *OLAP*-Report mit der Analyse von *OLAP*-Systemen. Die Codd-Regeln befand er jedoch als zu parteiisch, da Tedd Codd von der Firma Arbor Software – aus dem Bereich *Business Intelligence* – gesponsert wurde. Außerdem seien 12 Regeln zu viele, um sie sich zu merken. Daher entwickelte er im *OLAP*-Report eine eigene Evauluationsmethode. Dieser leicht zu merkende Test lautet abgekürzt *FASMI* und steht für ***F**ast **A**nalysis of **S**hared **M**ultidimensional **I**nformation*. (vgl. [Pen08])

- **Fast**

  Anfragen an ein *OLAP*-System sollten durchschnittlich innerhalb von 5 Sekunden angezeigt werden können. Die eher kleineren Anfragen sollten unter einer Sekunde liegen, die komplexesten Anfragen nicht über 20 Sekunden. Um langen Ladezeiten entgegenwirken zu können, wird empfohlen, vorkalkulierte Daten vorzuhalten, jedoch ist dies bei sehr großen Datenmengen kaum mehr möglich. Die meisten Probleme bei *OLAP*-Datenbanken beziehen sich auf langsame Anfragen.

- **Analysis**

  Das *OLAP*-System ist so ausgelegt, dass es mit jeder Art von Businesslogik und statistischen Daten arbeiten kann. Es soll dem Benutzer möglich sein selbstdefinierte Berechnungen in der Analyse durchzuführen, ohne dass der Nutzer dafür Programmierkenntnisse benötigt.

- **Shared**

  Das System muss über ausreichende Sicherheit verfügen. Falls auch ein Schreibzugriff erfolgen kann, ist es notwendig, simultane Schreibvorgänge zu verwalten.

- **Multidimensional**

  Dies ist die wichtigste Eigenschaft eines *OLAP*-Systems. Nicht nur die Mehrdimensionalität selbst, sondern auch die Nutzung von Hierarchien ist für ein *OLAP*–System unabkömmlich.

- **Information**

  Alle Informationen müssen dem Nutzer zur Verfügung stehen. Der Nutzer darf bei einer Analyse nicht durch Beschränkungen des *OLAP*–Systems beeinflusst werden.

Anhand der beiden erläuterten Tests lässt sich erkennen, dass eine *OLAP*-Datenbank für dieses Projekt von hoher Wichtigkeit ist, da es viele der Anforderungen an dieses Projekt unterstützt oder gar direkt erfüllt. Darunter fallen die Performanz, intuitive Datenanalyse und Transparenz.

2.3.6 Architekturen

Es gibt grundsätzlich vier unterschiedliche Architekturen, um ein *OLAP*-System zu realisieren. Sie unterscheiden sich meist in der Datenhaltung und Geschwindigkeit.

***ROLAP* (Relational-OLAP)** Diese Datenbank basiert auf einem relationalen Datenbank-Managementsystem. Eine solche Datenbank zu verwenden ist oft sehr naheliegend, da oftmals Daten bereits in relationaler bzw. flacher Form existieren (z. B. Tabellenkalkulationsdateien). Um diese Daten für Analysezwecke nutzen zu können, wird hier eine nicht-normalisierte Datenstruktur in einem *Starschema* oder *Snowflakeschema* verwendet. Vorkalkulierte Daten können auch in Form von aggregierten Relationen verwendet werden (vgl. [Pen08]). Die Daten werden, wie bei relationalen Datenbanksystemen üblich, auf der Festplatte gehalten und von dort, abgesehen von Daten, die in einem Cache liegen, geladen.

***MOLAP* (Multidimensional-*OLAP*)** Die Daten für ein multidimensionales *OLAP*-System werden meist aus bestehenden Datenbanken extrahiert. Die Datenhaltung während des Betriebes wird entweder wie bei *ROLAP* auf der Festplatte oder im Arbeitsspeicher realisiert. Die Verwendung des Arbeitsspeichers bietet den großen Vorteil höherer Performanz, allerdings ist die Datenbankgröße hier auf die Größe des Arbeitsspeichers beschränkt. In fast allen *MOLAP*-System gibt es vorkalkulierte Daten. Nicht viele *MOLAP*-Systeme unterstützen mehrere Benutzer mit Schreibzugriffen, eher ist es möglich, mehrere Nutzer nur mit Leserechten zugreifen zu lassen (vgl. [Pen08]). Schreibrechte von mehreren Nutzern ist auch grundsätzlich nicht im Sinne der Datenanalyse, da nach einer der Regeln von Tedd Codd die Datenquelle separiert ist von der *OLAP*-Datenbank (vgl. [CCS93]),

sodass nicht alle Daten immer live, sondern nach bestimmten Zeitintervallen in das System eingespielt werden.

***DOLAP* (Desktop-*OLAP*)** Diese Art der *OLAP*-Architektur besteht aus einem Server, welcher die Daten hält, und Clients, welche relativ kleine Datenmengen aus dem Datensatz extrahieren. Die Weiterverarbeitung der Daten erfolgt auf dem Client.

***HOLAP* (Hybrid-*HOLAP*)** Hierbei handelt es sich um eine Mischvariante aus *MOLAP* und *ROLAP*. Ein kleiner Datensatz wird hier multidimensional gehalten und bei Bedarf werden neue Daten aus der relationalen Datenbank in eine multidimensionale Datenstruktur nachgeladen.

2.4 Zusammenfassung

Im vorangegangenem Abschnitt wurden die Grundlagen des *Visual Analytics*-Prozesses vorgestellt. Unter *Visual Analytics* wird eine Methode zur visuellen Datenanalyse verstanden, welche den Bereich der automatisierten Algorithmen verlässt und den Menschen stärker in den Explorationsprozess mit einbezieht. Dieses Verfahren ermöglicht den Analysten, die Struktur von Daten schnell zu erfassen, explorativ in Daten zu navigieren und Schlussfolgerungen zu ziehen. Zentralle Komponente des Explorationsprozesses sind Visualisierugen, wodurch dem Menschen Daten in verständlicher Form automatisiert dargestellt werden.

Nachdem das Analyseverfahren vorgestellt wurde, sind verschiedene Visualisierungsarten aufgezeigt worden, wie z. B. *Liniendiagramme*, *Säulendiagramme* und *Streudiagramme*. Des Weiteren ist ein Einblick in komplexere Darstellungsformen, wie z. B. Kurvenprofile und thematischen Karten, gegeben worden.

Im nächsten Abschnitt ist das *OLAP*-System eingeführt worden. *OLAP* ist eine spezielle Datenbankform, bei der die Daten multidimensional verwaltet werden. Die *OLAP* zugrunde liegende Struktur ist ein *OLAP-Datenwürfel*, der eine Reihe von *Dimensionen* und *Kennzahlen* besitzt. *OLAP*-Operationen, wie z. B. *RollUp*, *DrillDown*, *Slicing* und *Dicing* dienen der Navigation in einem *Datenwürfel* und ermöglichen das Betrachten zahlreicher *Kennzahlen* aus unterschiedlichen Perspektiven. Dies eignet sich besonders zur visuelle Analyse, da die Datenmenge in verschiedenen Aggregationsstufen zueinander korreliert vorliegen und somit ein schnelles Navigieren erleichtert wird.

Kapitel 3

Multitouch

Unter Multitouch wird die Eingabe von Befehlen über eine Fläche mit berührungsempfindlichen oder positionsbestimmenden Sensoren verstanden. Die Sensoren erkennen die Position beziehungsweise den Druckpunkt eines Objektes auf der meist rechteckigen Fläche. Diese Objekte werden bei Touch-Technologien meistens durch Finger oder sogenannte *Stylus-Stifte* dargestellt. Einige Touch-Geräte erkennen zum Teil nicht nur die Position, sondern auch die Druckstärke oder sogar den Berührungswinkel der Objekte. Das charakteristische Merkmal von Multitouch-Eingabegeräten ist, dass sie mehrere Berührungspunkte zu einem Zeitpunkt erkennen können. Bei Singletouch-Eingabegeräten hingegen würde nur ein Berührungspunkt zu einem Zeitpunkt erkannt werden. Darüber hinaus erkennen einige Multitouch-Technologien nicht nur mehrere Punkte, sondern auch ganze Flächen. Dieser Unterschied eröffnet viele neue Möglichkeiten. So sind Eingaben an einem solchen Touchscreen mit mehreren Berührungspunkten und somit mit mehreren Benutzern möglich. Auch kann auf die Eingabe mittels Maus und Tastatur komplett verzichtet werden, da jegliche Mauseingaben nun mit der Hand getätigt werden können. Bei Multitouch-Geräten können viele Funktionen der Mauseingabe (wie zum Beispiel die des Mausrades) mit einfachen Gesten simuliert werden. Mausklicks und eine einfache Bewegung des Cursors sind bereits mit Singletouch simulierbar. Auch die Tastatur kann bei Touchscreens recht einfach nachgebildet werden, indem eine virtuelle Tastatur auf dem Bildschirm dargestellt wird. (vgl. [Röd07])

Im Rahmen der Projektgruppe wird ein Tisch genutzt, welcher sich durch einen multitouch-fähigen Touchscreen auf der Oberseite des Tisches auszeichnet. Damit dieses Konstrukt funktionieren kann, müssen viele verschiedene Technologien kombiniert und genutzt werden. Im Folgenden werden die von der Projektgruppe genutzten Technologien vorgestellt und erläutert. Dabei werden anfangs die nutzbaren visuellen Technologien vorgestellt, mit denen ein Multitouch-Tisch betrieben werden kann. Da die visuellen Technologien mit Hilfe von einer Kamera die Finger aufnehmen und die Einzelbilder analysiert werden müssen, werden im nachfolgenden Abschnitt die Mustererkennung und Protokolle erläutert. Zum Abschluss dieses Kapitels werden Gesten und deren Erkennung behandelt.

3.1 Visuelle Multitouch-Technologien

Damit der Multitouch-Tisch Multitouch-Funktionalität unterstützt, muss dieser mit einer geeigneten Technologie ausgestattet werden. Der Projektgruppe werden während ihrer Gesamtlaufzeit zwei Multitouch-Tische zur Verfügung gestellt. Während der Erste mit der *FTIR*-Technologie betrieben wurde, wird der aktuelle Multitouch-Tisch mit der *Diffused Surface Illumination (DSI)*-Technologie betrieben. Diese Technologie kann als Mischtechnologie aus *FTIR* und *Diffused Illumination (DI)* verstanden werden, welche im Folgenden erläutert werden.

3.1.1 Frustrated Total Internal Reflection

Frustrated Total Internal Reflection beruht auf der *Totalreflexion*. Unter der *Totalreflexion* wird „*ein optisches Phänomen, bei dem elektromagnetische Strahlung an der Grenzfläche zweier Medien nicht gebrochen, sondern vollständig reflektiert wird*“ [Wik] verstanden. Bevor die Funktionsweise der Technik erläutert wird, wird nun der Aufbau eines solchen Touchsensors erklärt. Wie in Abbildung 3.1 dargestellt, besteht der Touchsensor aus einer Plexiglas-Scheibe, an deren Seite Infrarot-LEDs angebracht werden. Unter der Plexiglas-Scheibe wird eine Infrarot-Kamera befestigt, welche die Plexiglas-Scheibe abfilmt. Häufig wird anstatt einer Infrarot-Kamera lediglich eine Webcam benutzt, aus welcher der Infrarot-Sperrfilter entfernt und stattdessen ein Infrarot-Bandpass eingesetzt wird. Dieses hat zur Folge, dass die Webcam nur noch Infrarot-Licht erkennt. (vgl. [SBD+08, Han05, NGC])

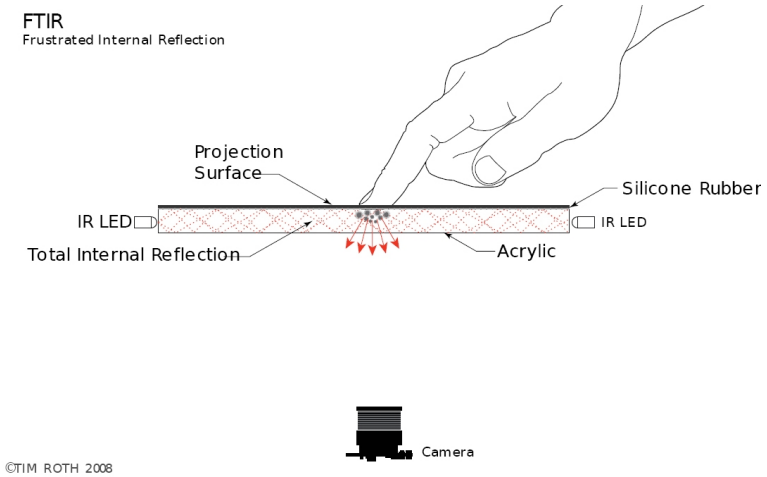


Abbildung 3.1: *FTIR* - Aufbau [SBD+08]

Die Funktionsweise ist nun die Folgende: Die Infrarot-LEDs senden Infrarot-Licht in das Plexiglas. Aufgrund der Totalreflexion wird das Licht innerhalb des Plexiglases gefangen gehalten und vollständig reflektiert. Berührt ein Finger die Plexiglas-Scheibe, verändert sich der Brechungsindex der Scheibe an der berührten Stelle und das Infrarot-Licht wird *verhindert*, also nach unten in Richtung der Infrarot-Kamera gestrahlt. Die Kamera erkennt das Licht als große, helle Punkte, sogenannte *Blobs* (siehe Abbildungn 3.2) und kann durch diese Punkte die Berührungspunkte bestimmen. (vgl. [SBD+08, Han05, NGC])

Die Technik wäre mit den bis nun beschriebenen Komponenten schon funktionsfähig, um jedoch einen Touchscreen zu erhalten, werden zwei weitere Schichten benötigt. Zum einen eine Projektionsschicht, zum anderen eine *kompatible Oberfläche* (im Englischen *Compliant Surface*).

Auf die Projektionsschicht kann dann ein Bild projiziert werden. Dieses geschieht meistens durch einen Projektor, welcher ebenfalls wie die Webcam unterhalb der Plexiglas-Scheibe angebracht ist. Die *kompatible Oberfläche* wird zwischen die Projektionsschicht und die Plexiglas-Scheibe gelegt. Diese Schicht ist zwar nicht zwingend notwendig, doch ohne sie müsste die Projektionsschicht sehr stark eingedrückt oder der Touchsensor mit öligen Fingern bedient werden. Denn diese zusätzliche Schicht, welche die Projektionsschicht und die Plexiglas-Scheibe miteinander verbindet, verhindert Luftbläschen, welche zwischen diesen beiden Schichten entstehen können. So wird durch diese Schicht eine geschmeidigerere Bewegung der Finger auf der Projektionsschicht ermöglicht. Gleichzeitig sind dadurch die *Blobs* für die Kamera besser zu erkennen. In den meisten Fällen wird als *kompatible Oberfläche* Silikon Kautschuk verwendet. (vgl. [NGC, Han05])

Abbildung 3.2: *FTIR - Blobs* [NGC]

Wie zu vermuten, ist diese Technik sehr leicht umzusetzen und auch recht kostengünstig in der Herstellung. Das teuerste Element an einem solchen Multitouch-Tisch ist dabei der Projektor. Zudem ist diese Technik sehr gut skalierbar, da der *FTIR*-Effekt nicht durch die Größe der Fläche beeinflusst wird und auch das Projektionsbild sich einfach

durch den Projektor skalieren lässt. Durch Überlagerungen von mehreren Infrarot-Kameras kann zudem nicht nur die generell schon sehr hohe, sondern eine noch höhere Auflösung erreicht werden. Weiterhin nimmt die *Blob*-Größe bei stärkerem Druck zu, wodurch die Technik drucksensitiv wird. Auch können bei bestimmten *kompatiblen Oberflächen* dünne Gegenstände genutzt werden, um den *FTIR*-Effekt auszulösen. Als wichtigster Punkt ist jedoch zu erwähnen, dass diese Technik voll multitouch-fähig ist, denn das Infrarot-Licht kann im Inneren des Plexiglases nicht durch andere Elemente behindert werden. Ein Nachteil beim eigenen Bau dieser Technik ist, dass ein LED-Rahmen benötigt wird. Dieser ist nur mit einiger elektrotechnischer Erfahrung und handwerklichem Geschick aufzubauen. Zudem ist der Touchsensor nicht in der Lage, die Objekte, die auf ihm liegen, zu identifizieren. Ein weiterer Nachteil ist, dass man bei der Konstruktion im Bereich der Stabilität eingeschränkt ist, da keine zusätzliche Glasscheibe genutzt werden kann. (vgl. [NGC, Han05, Rot07])

3.1.2 Diffused Illumination

Diffused Illumination nutzt, wie auch *FTIR*, Infrarot-Licht und eine Infrarot-Kamera für die Positionsbestimmung. Es muss zwischen zwei Arten von *DI*-Techniken unterschieden werden: zum einen gibt es die *Rear-DI*-Technik, welche die besseren Ergebnisse erzielt, und zum anderen *Front-DI*. Abbildung 3.3 zeigt dabei den Aufbau des Touchsensors der *Rear-DI*-Technik.

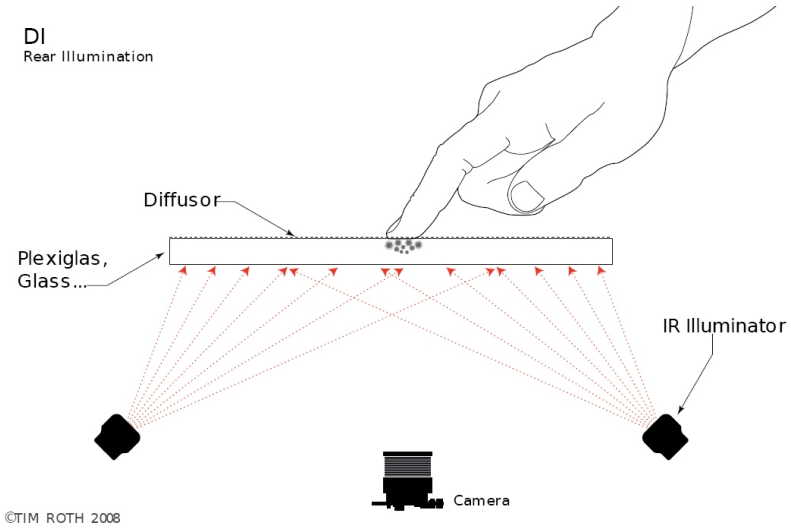


Abbildung 3.3: *DI* - Aufbau [SBD+08]

Auf den ersten Blick ähnelt der Aufbau dieser Technik sehr dem Aufbau der *FTIR*-Technik. Tatsächlich ist die Funktionsweise auch sehr ähnlich. So gibt es auch bei der *DI*-Technik eine Infrarot-Kamera, welche Infrarot-Licht erkennt, und die Plexiglas-Scheibe,

welche allerdings auch durch eine Glasscheibe ersetzt werden kann. Oberhalb der Plexiglas-Scheibe wird eine Diffusorschicht gelegt, welche auftreffendes Licht stärker zerstreut als die Plexiglas-Scheibe und in den meisten Fällen auch als Projektionsschicht fungiert. Unterhalb der Plexiglas-Scheibe sind Infrarot-Leuchter angebracht, die die Scheibe mit Infrarot-Licht bestrahlen. Dieses Licht wird von der Diffusorschicht zerstreut, also nur sehr schwach reflektiert. Wird die Oberfläche der Diffusorschicht mit einem Finger oder einem Gegenstand berührt, wird an dieser Stelle mehr Infrarot-Licht als sonst üblich reflektiert. Die Infrarot-Kamera kann also, wie auch bei der *FTIR*-Technik, die Position der Finger durch das reflektierte Infrarot-Licht als helle *Blobs* erkennen (siehe Abbildung 3.4) und diese berechnen. (vgl. [SBD+08, NGC])

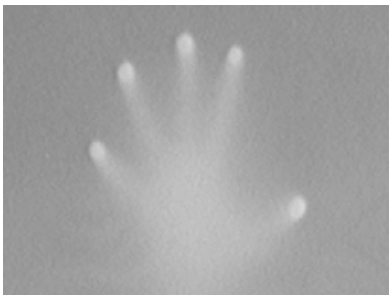


Abbildung 3.4: *DI - Blobs* [NGC]

Bei der *Front-DI*-Technik wird die obere Schicht des Touchsensors nicht von unten mit Infrarot-Licht bestrahlt, sondern von oben. Die Infrarot-Kamera würde dann keine hellen *Blobs* erkennen, sondern einen Schatten an der Stelle, an der die Finger den Touchsensor berühren (siehe Abbildung 3.5). Somit sind bei dieser Technik Konstruktionen möglich, welche gänzlich auf künstliche Infrarot-Lichtquellen verzichten. Das Prinzip ist leicht umsetzbar, beispielsweise schon durch einen Abfalleimer, in den unten eine Infrarotkamera integriert wird und Pauspapier, welches mit der Funktion der Diffusorschicht die Öffnung abdeckt. Diese Schicht wird dann vom Umgebungslicht bestrahlt und es würden dort Schatten entstehen, wo ein Gegenstand die Fläche berührt. (vgl. [NGC])

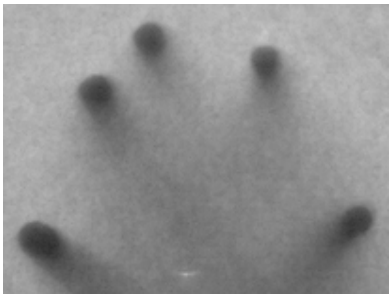


Abbildung 3.5: *Front-DI - Blobs* [NGC]

Diese Technik ist einfacher in der Konstruktion als die der *FTIR*-Technik. Da Infrarot-Strahler schon fertig zu kaufen sind, muss kein aufwendiger LED-Rahmen aufgebaut werden und es ist kein Löten notwendig. Zusätzlich wird keine *kompatible Oberfläche* benötigt, welche ebenfalls schwer zu montieren sein kann. Es kann jedes transparente Material genutzt werden, denn es bestehen keine Bedingungen für das Material, wie zum Beispiel die Möglichkeit der Totalreflexion. Wie auch die *FTIR*-Technik hat die *DI*-Technik eine sehr hohe Auflösung und ist voll multitouch-fähig. Ein weiterer Vorteil der *Rear-DI*-Technik ist, dass auch Objekte oder Erhebungen der Finger erkannt werden können, da diese von unten bestrahlt werden. Es kann sogar ermittelt werden, um welche Objekte es sich handelt. So ist eine Anwendung von bestimmten Codes, welche das dazugehörige Programm zu interpretieren weiß, unterhalb von diesen Objekten denkbar. Jedoch hat diese Technik auch einige Nachteile. Wird beispielsweise Abbildung 3.2 mit Abbildung 3.4 und Abbildung 3.5 verglichen, so ist zu erkennen, dass die *Blobs* bei den *DI*-Techniken ungenauer sind als bei der *FTIR*-Technik. Außerdem ist es schwierig, eine gleichmäßige Beleuchtung der Diffusorschicht zu erreichen, welche für eine zuverlässige Funktionsweise des Touchsensors unbedingt erforderlich ist. Ein weiterer Nachteil ist die hohe Fehleranfälligkeit, welche durch das ständig wechselnde Umgebungslicht verursacht wird. Die *DI*-Techniken sind empfindlicher gegenüber dem Umgebungslicht, als die *FTIR*-Technik. So muss der Touchsensor bzw. das Auswertungsprogramm für die *Blobs*, ständig neu kalibriert werden, um sich dem stetig wechselnden Umgebungslicht anzupassen. Dieser Nachteil kommt besonders beim *Front-DI* zum tragen, da diese Technik zum Teil nur mit Umgebungslicht betrieben wird und nicht mit zusätzlichen Strahlern. (vgl. [NGC, Rot07])

3.1.3 Diffused Surface Illumination

Wie schon erwähnt ist eine große Schwierigkeit bei der *DI*-Technologie die gleichmäßige Beleuchtung der Diffusorschicht. Diese Schwierigkeit wird bei der *DSI*-Technologie durch eine spezielle Acrylplatte und der Art des Aufbaus verhindert. Der Aufbau ist dem von *FTIR* ähnlich, nur dass hier keine normale Plexiglas-Scheibe, sondern eine spezielle Plexiglas-Scheibe genutzt wird. Diese enthält kleine Partikel, welche sich wie eine Vielzahl von kleinen Spiegeln verhalten. Hierdurch wird das Infrarot-Licht nicht innerhalb der Scheibe gefangen gehalten, sondern gleichmäßig in Richtung der Oberfläche heraus gestrahlt (siehe Abbildung 3.6). (vgl. [NGC, SBD$^+$08])

Der Effekt, der bei Berührung dieser Plexiglas-Scheibe auftritt, ist ähnlich dem Effekt der *DI*-Technologie, nur ist die Beleuchtung gleichmäßiger und es gibt keine Hotspots. Ein weiterer Vorteil ist, dass dieser Aufbau zu einem *FTIR*-Aufbau umgewandelt werden kann, indem einfach die Plexiglas-Scheibe ausgetauscht wird. Weiterhin können *Fiducials* mit dieser Technologie erkannt und verfolgt werden. Allerdings ergeben sich auch Nachteile durch den Aufbau. Dadurch, dass das Infrarot-Licht auch in

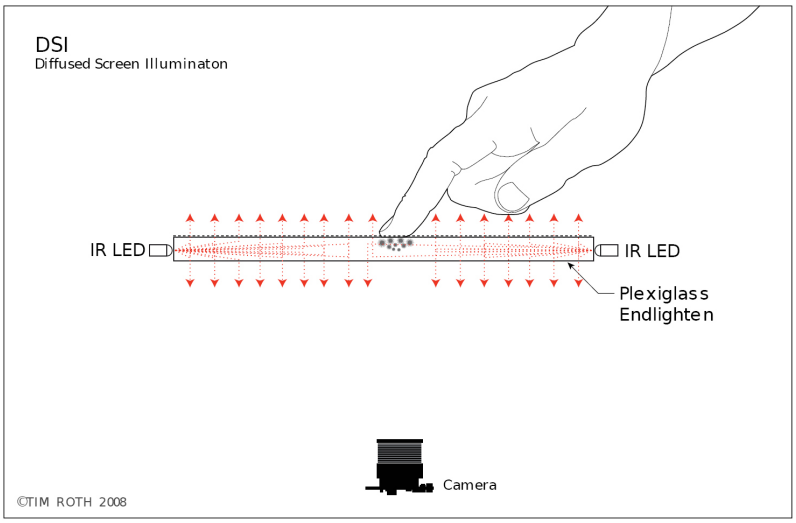


Abbildung 3.6: *DSI* - Aufbau [SBD+08]

Richtung der Kamera gestrahlt wird wenn kein Objekt die Oberfläche berührt, ist der Kontrast der *Blobs* geringer als bei einem normalen *DI*-Aufbau. Durch den niedrigeren Kontrast können sich ebenfalls mehr Probleme mit dem Umgebungslicht ergeben. (vgl. [NGC, SBD+08])

3.2 Mustererkennung und Protokolle

Der Projektgruppe steht ein Multitouch-Gerät zur Verfügung, welches mit Hilfe eines *DSI*-Setups realisiert wurde. Wie in Abschnitt 3.1.3 deutlich wurde, handelt es sich dabei um ein kostengünstiges, visuelles Verfahren, bei dem die Eingaben des Benutzers mit Hilfe einer Kamera videoüberwacht werden. Ob und welche Eingaben der Benutzer auf der Eingabefläche durchgeführt hat, muss bei dieser Technologie aus einer Analyse der Videoaufnahme geschlossen werden.

Die Eingaben können daher nicht unmittelbar weitergeleitet werden, sondern bedürfen zunächst einer umfangreichen Vorverarbeitung. Der erste Schritt besteht darin zu erkennen, welchen Inhalt die Einzelbilder (*Frames*) des Videos darstellen. Hier kommen Verfahren zur Mustererkennung zum Einsatz, wie sie in der Bildverarbeitung verbreitet sind, diese werden in [MHB99] und [BB05] ausführlich vorgestellt. Die Abschnitte 3.2.1 und 3.2.2 gehen auf die Grundzüge dieser Verfahren ein. Erkannte Eingabeobjekte müssen schließlich über aufeinander folgende Bilder hinweg erneut identifiziert und somit verfolgt werden. Abschnitt 3.2.3 befasst sich mit dieser Thematik.

Da das Erkennen und Verfolgen von *Blobs* eigenständige und wiederkehrende Probleme darstellen, liegt es nahe, diese Funktionen auszulagern, damit sie von verschiedenen Programmen verwendet werden können. In solch einer Architektur muss eine geeignete Schnittstelle für die Kommunikation zwischen den Komponenten sorgen. Für Umgebungen mit multitouch-fähigen Geräten wurde zur Regelung dieser Kommunikation das *TUIO*-Protokoll entworfen, welches auf *Open Sound Control (OSC)* basiert. Auf diese beiden Protokolle geht Abschnitt 3.2.4 ein.

3.2.1 Relevante Muster

Typische Muster ergeben sich bei Multitouch-Geräten durch Berührungen mit der Hand, wie es links in Abbildung 3.7 dargestellt ist. Die Berührungsfläche tritt besonders deutlich in Form heller Regionen, sogenannter *Blobs*, hervor.

Zusätzlich können *Tangibles* für die Bedienung von Multitouch-Geräten eingesetzt werden. Dabei handelt es sich um Gegenstände, welche auf die Eingabefläche gelegt werden können, um herkömmliche Dreh- und Schieberegler zu simulieren. Bewegungen oder Drehungen eines *Tangibles*, die durch den Benutzer durchgeführt werden, können so verwendet werden um den Programmfluss zu steuern. Abbildung 3.7 zeigt rechts einige Beispiele für *Tangibles*, die entweder über ihre Form oder durch ein aufgedrucktes Muster (*Fiducial*) unterschieden werden können.

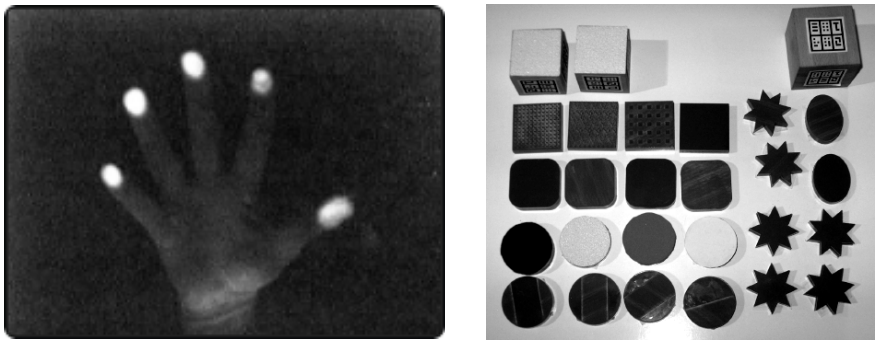


Abbildung 3.7: Beispiel für die Muster, die sich durch Berührungen mit der Hand ergeben (links) und Beispiele für *Tangibles* mit unterschiedlichen Formen (rechts). [NGC08a, KOC04]

Während jede der in Abschnitt 3.1.3 vorgestellten Technologien das Verfolgen von einem oder mehreren Fingern erlaubt, ist es nicht mit jeder von ihnen möglich *Tangibles* einzusetzen. Im Falle von *FTIR* strahlen Infrarot-LEDs von den Seiten in eine Acrylscheibe ein. Durch eine Berührung an der Oberfläche wird das Infrarotlicht dann in Richtung der Kamera gestreut. Für die Mustererkennung stehen also allein

die Informationen über die Berührungsfläche zur Verfügung. Darüber hinausgehende räumliche Informationen gehen dabei verloren. *Tangibles*, die, ohne großen Druck auszuüben, auf der Tischoberfläche liegen, können damit kaum erkannt werden.

Wird *DI* anstelle von *FTIR* verwendet, dann wird die Eingabefläche aus der Kameraebene heraus beleuchtet. Die Kamera registriert dann alle Objekte, welche die Infrarotstrahlung reflektieren. So stehen auch Informationen über das zur Verfügung, was sich wenige Zentimeter oberhalb der Eingabefläche befindet. Insbesondere können dann auch Muster erkannt werden, die sich durch unterschiedliches Reflektionsverhalten ergeben, u. a. *Tangibles*. Für Tische mit der *FTIR*-Technologie lässt sich etwas ähnliches realisieren, indem Infrarot-LEDs in das *Tangible* integriert werden.

3.2.2 Mustererkennung

Die Mustererkennung bei der Verarbeitung der Einzelbilder kann im Wesentlichen auf zwei verschiedene Weisen angegangen werden. Die Analyse des Bildes kann entweder kanten- oder regionenbasiert erfolgen. Ein kantenbasierter Ansatz durchsucht das Bild zunächst nach Kanten und versucht anschließend, diese den Formen von bekannten Objekten zuzuordnen. Ein regionenbasierter Ansatz sucht stattdessen nach zusammenhängenden Flächen und versucht, diese bekannten Formen zuzuordnen. [MHB99]

Der Inhalt der Kameraaufnahmen bei einem Multitouch-Tisch setzt sich relativ einfach zusammen, nämlich aus dem dunklen Hintergrund und den hellen Berührungsflächen. Da keine komplexen Strukturen erkannt werden müssen, ist ein regionenbasierter Ansatz einem kantenbasierten vorzuziehen. Aus diesem Grund wird die Mustererkennung optischer Multitouch-Geräte im Folgenden mit Hilfe von regionenbasierten Bildbearbeitungsmethoden erläutert, was den Einsatz kantenbasierter Verfahren jedoch nicht ausschließt.

Eine *Region* bzw. ein *Blob* entspricht dabei einem zusammenhängenden Verbund von Pixeln, welche alle den gleichen Farbton aufweisen. Damit diese Bedingung so gut wie möglich erfüllt wird, wird jedes Eingabebild zunächst einem Schwellwertverfahren (*Thresholding*) unterzogen. Dieses wandelt das Graustufenbild der Kamera in ein Binärbild um. In einem solchen Binärbild wird nur noch zwischen Vordergrund- und Hintergrundpixeln unterschieden. Es gibt also nur noch zwei Farben. In unserem Fall sind die Hintergrundpixel schwarz, während sich die hell reflektierenden Objekte auf der Tischoberfläche in der Vordergrundebene des Bildes wiederfinden.

Im Anschluss an das *Thresholding* ist die *Blob Detection* relativ einfach möglich. Die *sequentielle Regionenmarkierung* ist ein Verfahren, welches dafür eingesetzt werden kann. Es arbeitet in zwei Durchläufen: Im ersten Durchlauf erhalten alle Vordergrundpixel einen neuen Wert, der einer Nummerierung der gefundenen Regionen

entspricht. Das höchste Label, das auf diese Art vergeben wird, entspricht also gleichzeitig der Anzahl der unterscheidbaren Regionen. Welches Label für ein Pixel vergeben wird, wird in Abhängigkeit von seiner direkten Nachbarschaft entschieden.

Die Nachbarschaft eines Bildpunktes ist leider nicht klar definiert. Es ist jedoch üblich, entweder eine 4-er Nachbarschaft oder eine 8-er Nachbarschaft zu betrachten. Beide werden durch Abbildung 3.8 verdeutlicht. Welche der beiden verwendet wird, macht im Folgenden meist keinen Unterschied. Ist dies dennoch der Fall, wird an geeigneter Stelle darauf hingewiesen.

Abbildung 3.8: Unterschiedliche Definitionen für Nachbarschaften. 4er-Nachbarschaft (links) und 8er-Nachbarschaft (rechts). Abbildung ähnlich in [BB05] zu finden.

Befindet sich noch keine Regionsnummer innerhalb der Nachbarschaft eines Pixels, so wird ein neues Label erzeugt. Liegt stattdessen bereits genau ein Label vor, wird dieses für das aktuell betrachtete Pixel übernommen. Enthält die Nachbarschaft allerdings zwei unterschiedliche Labels, so liegt offensichtlich ein Konflikt vor. Dann wird zunächst irgendeines der beiden übernommen. In einer zusätzlichen Datenstruktur werden die betroffenen Labels gleichzeitig als äquivalent vermerkt.

Der zweite Durchlauf dient dazu, solche Unstimmigkeiten zu beseitigen. Das ganze Bild wird erneut durchlaufen, wobei äquivalente Markierungen durch eine Einheitliche ersetzt werden [BB05].

3.2.3 Blob Tracking

Die detektierten Regionen müssen schließlich über mehrere Bilder hinweg verfolgt werden. Hierzu werden sie mit einer eindeutigen *Identifier (ID)* versehen, welche sie in jedem Bild kennzeichnet. Die Regionenmarkierung muss daher für jedes Einzelbild durchgeführt werden. Auf diese Weise wird zunächst eine Menge von Regionen $\mathcal{M}_n$ bestimmt, die im n-ten Einzelbild $\mathcal{B}_n$ enthalten sind, und eine weitere Menge von Regionen $\mathcal{M}_{n+1}$, welche das Folgebild $\mathcal{B}_{n+1}$ enthält. Nun gilt es, diejenigen Elemente von $\mathcal{M}_{n+1}$ zu finden, die bereits in $\mathcal{M}_n$ enthalten waren, also Regionen, die in beiden Bildern enthalten sind. Diese erhalten dann diejenige *ID*, die dem entsprechenden

Element aus $\mathcal{M}_n$ bereits zugeordnet ist. Regionen, die keiner bereits vorhandenen Region zugeordnet werden können, erhalten eine neue *ID*.

Wann eine Region $\mathcal{R}_1 \in \mathcal{M}_n$ als wiedererkannt gewertet wird, ist eine Festlegungssache, für die dem Autor kein Standardverfahren bekannt ist. Denkbar ist hier, dass zunächst alle Regionen aus $\mathcal{M}_{n+1}$ heraus gefiltert werden, deren Abstand zu den Bildkoordinaten von $\mathcal{R}_1$ einen gewissen Maximalabstand überschreitet. Bei einfachen Implementierungen, wie etwa im Falle der Touchlib, die später in Abschnitt 3.2.5 vorgestellt wird, ist dies bereits das einzige Kriterium, das beim *Blob Tracking* Anwendung findet. Denkbar ist aber auch, dass weitere Kriterien zur Entscheidung herangezogen werden, falls mehrere Regionen einen angemessenen Abstand aufweisen. Es könnte z. B. überprüft werden, ob sich die Fläche und die Form einer Region wesentlich verändert haben. Wie solch eine Überprüfung umzusetzen ist, kann u. a. in [MHB99, BB05] nachgeschlagen werden.

3.2.4 Das TUIO Protokoll

Eine Trennung zwischen einer Komponente zur Mustererkennung und darauf aufsetzenden Komponenten ist wünschenswert, da die Erkennung von *Blobs* und ihre Verfolgung eigenständige Problemstellungen darstellen, die vom Rest des Programms unabhängig sind. Es gibt also keinen Grund sie mehrfach zu implementieren. In diesem Zusammenhang findet das *TUIO*-Protokoll, welches im Folgenden vorgestellt werden soll, verbreitete Anwendung. Dabei handelt es sich jedoch um eine Implementierung des *OSC* Protokolls, welches die allgemeine Kommunikation zwischen Computern und beliebigen Multimedia-Geräten regelt. Daher wird zunächst *OSC* näher erläutert.

Open Sound Control

Bei *OSC* handelt es sich um eine Verallgemeinerung des *Musical Instrument Digital Interface (MIDI)*-Standards. Dieser bildete ursprünglich eine Schnittstelle zwischen Musikinstrumenten und Synthesizern. Wird beispielsweise eine Taste eines Keyboards niedergedrückt, so wird eine *MIDI*-Nachricht erzeugt, deren Inhalt angibt, welcher Ton zu welchem Zeitpunkt gespielt wird und wie kraftvoll der Tastenanschlag war. Eine zweite Nachricht wird erzeugt, wenn die Taste wieder losgelassen wird. Der Synthesizer empfängt diese Nachrichten über eine Hardwareschnittstelle und ist mit ihrer Hilfe zu einem späteren Zeitpunkt in der Lage das Musikstück zu rekonstruieren. Der *MIDI*-Standard erreicht bei der Beschreibung von Musik eine Genauigkeit, welche diejenige von Notenschrift bereits übertrifft.

Open Sound Control überträgt dieses Konzept auf eine allgemeine Client-Server-Architektur, wobei das Musikinstrument durch einen Client ersetzt wird, und der

Synthesizer durch einen Server. Die Kommunikation zwischen den beiden findet über *OSC*-Pakete statt, die der Client über ein Netzwerkprotokoll an den Server sendet. Für diese Übertragung wird oft das *User Datagram Protocol (UDP)* verwendet, da es durch den Verzicht auf Bestätigungsmeldungen für den Empfang von Paketen gegenüber dem *Transmission Control Protocol (TCP)* eine höhere Geschwindigkeit bei der Übertragung der Nutzdaten aufweist. Daraus resultiert die Möglichkeit, dass einzelne Pakete verloren gehen können, was durch den Server berücksichtigt werden muss [Osc03].

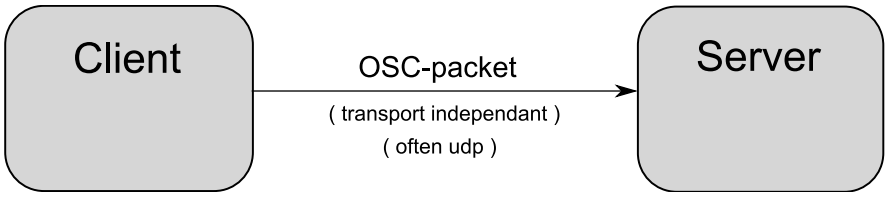


Abbildung 3.9: Schematische Übersicht über ein System aus *OSC*-Client und Server.

Damit der Server die Nachrichten des Clients verarbeiten kann, müssen sich diese an eine bestimmte Syntax halten, wie sie in der *OSC*-Spezifikation [Wri02] festgehalten ist. Eine *OSC*-Nachricht besteht danach immer aus den drei Komponenten

- *OSC Address Pattern*,
- *OSC Type Tag String* und
- und einem oder mehreren *OSC Arguments*.

OSC Address Pattern Das *OSC Address Pattern* bestimmt den oder die Empfänger der Nachricht. Im Gegensatz zum *MIDI*-Standard kann der Server nun beliebige Funktionen anbieten. Diese müssen in einer baumförmigen Hierarchie, ähnlich Abbildung 3.10, organisiert sein. Eine *OSC*-Adresse entspricht dem vollständigen Pfad von der Wurzel bis zu einem Knoten in diesem Baum, z. B. „`/touch/down`“.

Ein *OSC Address Pattern* entspricht im Wesentlichen solch einer Adresse, kann aber zusätzlich Wildcards wie „`?`“ oder „`*`“ enthalten, so dass z. B. „`/touch/*`“ alle drei Funktionen aus Abbildung 3.10 als Empfänger festlegen würde.

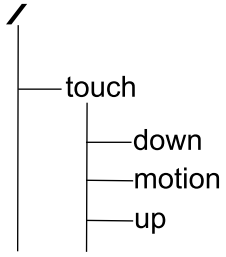


Abb. 3.10: Beispielhafter Namensraum für einen OSC-Server

OSC Type Tag String Auf das Address Pattern folgt der sogenannte *OSC Type Tag String*. Dieser hängt eng mit den darauf folgenden *OSC Arguments* zusammen, wird aber oft auch weggelassen. Der *OSC Type Tag String* ist dazu gedacht, die

Datentypen der darauf folgenden Argumentliste festzulegen. Er entspricht einer Liste von Kürzeln für die entsprechenden Datentypen (rot in Tabelle 3.1).

OSC Arguments Über die *OSC Arguments* wird eine Liste von Werten angegeben, die als Parameter an den Server übergeben werden soll. Hierfür stehen standardmäßig die Datentypen `int32(i)`, `float32(f)`, *OSC*-`String(s)` und *OSC*-`Blob(b)` zur Verfügung.

Vollständige *OSC*-Nachrichten, könnten wie die Beispiele in Tabelle 3.1 aussehen.

address pattern	type tag string	arguments		
/touch/down	(iff)	785	636.000000	579.000000
/touch/motion	-	773	474.000000	603.000000
/touch/up	-	771	495.000000	580.000000

Tabelle 3.1: Beispiele für vollständige OSC-Nachrichten[1]

Schließlich können eine oder mehrere *OSC*-Nachrichten zu einem *OSC Bundle* zusammengefasst werden. Ein solches Bundle wird atomar interpretiert, d.h. alle enthaltenen Nachrichten werden gleichzeitig ausgeführt. Zusätzlich kann ein Bundle auch selbst Bundles enthalten. Jedem Bundle ist ein *Time Tag* zugeordnet, der festlegt, an welchem Zeitpunkt es wirksam wird [Wri02].

TUIO

Das *TUIO* Protokoll ist speziell darauf ausgelegt worden, den Zustand der Objekte, die sich auf einer Multitouch-Eingabefläche befinden, zu überwachen. Es wurde im Zusammenhang mit *ReacTIVision* für den *reactable* (siehe Abschnitt 3.2.5) entwickelt, und ist daraufhin optimiert, dass sich viele Objekte zugleich auf der Eingabefläche befinden.

Es wird zwischen Cursorn und Objekten unterschieden. Cursor entsprechen dabei *Blobs*, wie sie durch die Berührung mit den Fingern hervorgerufen werden. Objekte entsprechen den *Tangibles*, über die mehr Informationen an den Server übermittelt werden, beispielsweise ihre Orientierung.

Da *TUIO* in *OSC* implementiert ist, verwendet es auch die in Abschnitt 3.2.4 eingeführte Syntax. Zudem macht das Protokoll Vorgaben darüber, welche Funktionen der Server zur Verfügung zu stellen hat. Die Kommunikation zwischen Client und Server soll nach [KBBC05] vollständig über drei Nachrichtentypen geregelt werden: *set messages*, *alive messages* und *fseq messages*.

[1] Beispiel übernommen von [Zad07].

Set Messages dienen dazu, den Zustand eines Objektes zu übergeben. Sie werden erzeugt, wenn ein neues Objekt detektiert wird oder wenn sich der Zustand, z.B. Position oder Orientierung, eines bekannten Objektes verändert.

Alive Messages enthalten immer eine Liste aller Objekte, die sich derzeit auf der Tischoberfläche befinden. Abgesehen von ihren *ID*s wird dabei keine Information übergeben. *Alive Message*s werden zum einen in regelmäßigen Abständen erzeugt, zum anderen aber auch dann, wenn ein Objekt von der Eingabefläche verschwunden ist.

Fseq Messages enthalten eine *ID* für das *Frame*, welches gerade verarbeitet wird. Über *fseq Messages* kann zugeordnet werden, welche Ereignisse aus dem gleichen Bild entnommen worden sind.

Auf explizite Nachrichten für die Detektion eines neuen Objektes, sowie für das Verschwinden eines bereits bekannten, ist bewusst verzichtet worden, denn wenn *UDP* verwendet wird, und eine dieser beiden Nachrichten verloren geht, wird es schwierig Konsistenz herzustellen. Stattdessen muss der Server das Auftauchen von Objekten aus eingehenden *set message*s ableiten. Das Verschwinden eines Objektes muss aus aufeinander folgenden *alive message*s erschlossen werden (vgl. [KBBC05]).

In der Spezifikation wird zudem ein unverbindlicher Ablauf vorgeschlagen, zu welchen Zeitpunkten zusätzliche Nachrichten versendet werden. Um die Größe eines *UDP*-Pakets voll auszunutzen, werden mehrere *set messages* zu einem Bundle zusammengefasst, wobei jedes Bundle zugleich auch eine *alive message* enthält. Letztere werden auch dann in regelmäßigen Abständen gesendet, wenn sich nichts an der Eingabefläche ändert. Zudem werden in regelmäßigen Abständen *set messages* für Objekte versandt, die ihren Zustand gar nicht verändert haben. Dies betrifft jedoch immer nur eine zyklisch wechselnde Teilmenge der bekannten Objekte.

Schließlich werden durch *TUIO* Profile für 2D, 2.5D und 3D-Oberflächen festgelegt, die sich jeweils darin unterscheiden, welche Information mit einer *set message* übertragen wird. Um zweieinhalb- oder dreidimensionale Ereignisse zu beschreiben, müssen jeweils mehr Argumente übermittelt werden, als für zweidimensionale. Für 2D-Oberflächen haben die Profile die folgende Form:

/tuio/2Dobj	set	s i x y a X Y A m r
/tuio/2DCur	set	s x y m r
/tuio/[profilname]	alive	[Liste aktiver sessionIDs]
/tuio/[profilname]	fseq	int32

Die Bedeutungen der Kürzel, die anstelle der Parameter angegeben sind, sind Tabelle 3.2 zu entnehmen (siehe auch [KBBC05]).

Parameter	Bedeutung
s	sessionID, temporary object ID, int32
i	classID, fiducial ID number, int32
x,y,z	position, float32, range 0...1
a,b,c	angle, float32, range 0...2π
A,B,C	movement vector (rotation speed and direction), float32
m	motion acceleration, float32
r	rotation acceleration, float32
P	free parameter , type defined by *OSC* packet header

Tabelle 3.2: Bedeutung der Parameter für *set messages*.

3.2.5 Bibliotheken zur Mustererkennung

In diesem Abschnitt wird kurz auf bereits bestehende Softwarebibliotheken eingegangen, die speziell für die Mustererkennung optischer Multitouch-Geräte entwickelt worden sind. Vorab werden Kriterien aufgestellt, anhand derer diese Bibliotheken zu bewertet werden können. Die Auswahl der Kriterien orientiert sich zum einen natürlich an dem Funktionsumfang, welchen die betrachtete Software liefert, zum anderen aber auch an den Rahmenbedingungen der Projektgruppe. Das Symbol **+** entspricht im Folgenden einer positiven Bewertung, **–** einer negativen und • einer neutralen Wertung. Schließlich soll es auf dieser Grundlage möglich sein, ein Urteil darüber zu fällen, ob der Einsatz einer bestimmten Programmbibliothek im Rahmen der Projektgruppe sinnvoll ist oder nicht.

Kriterien

Der multitouch-fähige Tisch, der als Eingabegerät verwendet wird, ist über die eingebaute Kamera an einen Computer angeschlossen. Daher bedarf es einer Softwarekomponente, welche die Videobilder auswertet und Koordinaten für Cursor bzw. Objekte liefert.

Die Anforderungen an diese Komponente beginnen daher mit der *Kameraunterstützung*. Diese ist von Programm zu Programm unterschiedlich stark ausgeprägt und hängt zudem vom verwendeten Betriebssystem ab. Da innerhalb der Projektgruppe eine Entwicklung innerhalb von Microsofts .Net-Umgebung vorgesehen ist, werden nur

Windows kompatible Versionen der Bibliotheken betrachtet. Solche, die überhaupt keine Kameraunterstützung bieten, scheiden von vornherein aus.

Ein weiterer Schwerpunkt des Funktionsumfangs soll in der *Blob Detection* bestehen. Das heißt, dass *Blobs* in den Einzelbildern, welche die Kamera liefert, aufgespürt und als Cursor bzw. als *Tangible* identifiziert werden sollen. Im Rahmen der Projektgruppe spielen *Tangibles* eine eher untergeordnete Rolle. Das heißt, sie müssen nicht zwingend unterstützt werden.

Weiter sollen einmal erkannte Objekte in Folgebildern wiedererkannt und verfolgt werden.

Die anfallenden Positionsdaten für erkannte Objekte sollen schließlich mit Hilfe eines Protokolls wie *OSC* oder *TUIO* weitergegeben werden.

Zusammengefasst ergeben sich die folgenden Kriterien bezüglich der Funktionalität:

- Kameraunterstützung
- Blob Detection
- Blob Tracking
- Protokollunterstützung

Entscheidend ist zudem, wie gut und auch auf welche Weise bei der Konfiguration Einfluss auf die Mustererkennung genommen werden kann. Auf jeden Fall erforderlich sind Filtermöglichkeiten, mit deren Hilfe das Bildrauschen oder auch ungewollte Artefakte eliminiert werden können. Wünschenswert wäre beispielsweise eine Möglichkeit, selbst Muster festzulegen, die anschließend wiedererkannt werden können.

Zur Beurteilung der Qualität der Programme wird der Programmcode betrachtet. Der Schwerpunkt wird dabei darauf gelegt, wie verständlich der Code ist. Hier ist natürlich kein rein objektives Urteil möglich.

Weitere Kriterien sind damit:

- Konfigurierbarkeit
- Codequalität

OpenCV

Die *Open Source Computer Vision Library (OpenCV)* wurde 1999 von *Intel* ins Leben gerufen. Hintergrund war, dass bis zu diesem Zeitpunkt kein in sich geschlossenes Framework für den Bereich der Computer Vision existierte. Bis zum heutigen Tage hat sich eine beachtliche Community, von über 14.000 eingetragenen Forenmitgliedern, um das Open Source Projekt versammelt. Insgesamt umfasst das Projekt über 500 Computer Vision-Algorithmen, die weit mehr Funktionen liefern, als die Projektgruppe benötigen [Pis07].

Gerade für Anwendungsentwickler ist *OpenCV* interessant, da sich die zahlreichen Funktionen leicht und flexibel in eigene Projekte einbinden lassen. Dies ist jedoch grundsätzlich nur unter Zuhilfenahme von eigenem Programmcode möglich. Aus diesem Grunde erhält *OpenCV* für Kameraunterstützung[2] und *Blob Detection* jeweils ein durchschnittliches Urteil, denn grundlegende Funktionen werden geboten, müssen aber durch eigenen Programmieraufwand zusammengefügt werden. Die Verfolgung von *Blobs* hingegen muss vollständig selbst umgesetzt werden. Hierfür und für die nicht vorhandene Protokollunterstützung gibt es jeweils eine negative Bewertung.

Konfiguriert wird die Mustererkennung im Falle einer eigenen *OpenCV*-Implementierung vollständig durch den Entwickler. Hier gibt es also kaum Beschränkungen, also wird ein ✚ vergeben. Bei der Entwicklung dieser Bibliothek wurde großer Wert auf Effizienz gelegt, sodass der resultierende C++-Code recht maschinennah gehalten ist. Hierunter leidet die Verständlichkeit des Codes. Die Effizienz ist damit positiv zu bewerten, der Mangel an Verständlichkeit dagegen eher negativ. *OpenCV* erhält deshalb ein • für seine Codequalität.

TouchLib

Bei der Touchlib handelt es sich um eines der ersten frei verfügbaren Projekte, welches eine Mustererkennung für optische Multitouch-Geräte umsetzt. Sie wurde Ende 2006 von David Wallin ins Leben gerufen und ist prinzipiell sowohl für den Einsatz mit der *FTIR*-Technologie als auch für *DI* geeignet. Mittlerweile wird sie durch die NUI-Group[3] weitergeführt [Wal08].

[2] Die unterstützten Kameramodelle sind dem *OpenCV*-Wiki unter `http://opencv.willowgarage.com/` zu entnehmen. Zuletzt eingesehen am 24.12.2008.

[3] http://nuigroup.com/, zuletzt besucht am 20.09.09

Die Touchlib setzt auf *OpenCV* auf. Sie kann daher entweder Gebrauch von ihrer Kameraunterstützung machen oder auf *VideoWrapper*[4] oder *DSVideoLib*[5] zurückgreifen. Die Kameraunterstützung ist also relativ umfangreich umgesetzt worden (✚).

Die *Blob Detection* ist relativ einfach gehalten, reicht aber bereits für die gleichzeitige Verfolgung von Fingern und *Tangibles* aus. Letztere unterscheiden sich in diesem Falle durch eine quadratische Form und eine größere Fläche von den Fingern. Maximal neun derartige *Tangibles* können automatisch durch die Touchlib verfolgt werden. Finger werden als Cursor interpretiert, von denen beliebig viele zugleich verfolgt werden können. Da *Blob Detection* und -Tracking ohne größeren Aufwand beinahe uneingeschränkt funktionieren, werden beide positiv bewertet. Ereignisse, die das Auftauchen oder die Bewegung eines Objektes auf der Oberfläche des Tisches signalisieren, werden in Form des *TUIO*-Protokolls kommuniziert, sodass auch die Protokollunterstützung positiv zu bewerten ist.

Filter	**Funktion**
cvcapture	Touchlibs Wrapper für *OpenCV*s capturing Funktionen
vwcapture	VideoWrapper als Quelle für capturing verwenden (Windows only)
dsvlcapture	Verwendung von Directshow (Windows only)
mono	Wandelt das eingehende Bild in ein Graustufenbild um
rectify	Thresholding Filter, der alle Pixelwerte unterhalb einer Grenze auf null setzt
highpass	Filter, verschwommene Strukturen entfernt, so dass das nur scharfe Kanten erhalten bleiben (Di).
invert	Invertiert die Farben
smooth	Gaußscher Weichzeichner
brightnesscontrast	Regler für Helligkeit und Kontrast
backgroundremove	Legt einen Schnappschuss an, der von allen Folgebildern abgezogen wird
resize	Ändert die Größe des Camera inputs

Tabelle 3.3: Filtereinstellungen, zur Konfiguration der Touchlib.

Dies gilt auch für die Konfigurierbarkeit der Touchlib. Das eingehende Videobild wird zunächst durch einige Vorfilter verarbeitet, welche von *OpenCV* entliehen sind. Mit Hilfe einer Konfigurationsanwendung kann eine Feineinstellung dieser Filter vorgenommen werden, sodass die Mustererkennung mit den bestmöglichen Bedingungen durchgeführt werden kann. Tabelle 3.3 enthält die einstellbaren Filter, deren Werte nach der Feineinstellung in einer *eXtensible Markup Language (XML)*-Datei gespeichert werden.

[4] Ein Wrapper für verschiedene camera libraries, verfügbar unter `http://sourceforge.net/projects/videowrapper`. Zuletzt eingesehen am 24.12.2008.

[5] Ein DirectShow Wrapper, verfügbar unter `http://sourceforge.net/projects/dsvideolib/`. Zuletzt eingesehen am 24.12.2008.

Der Code der Touchlib ist im Großen und Ganzen relativ verständlich und überschaubar. Teilweise gibt es jedoch auch Code der vollkommen unkommentiert ist, und dessen Funktion nicht ohne weiteres nachvollzogen werden kann. Ein Beispiel hierfür stellt die Umsetzung der *Blob*-Verfolgung dar. Auf Effizienz wurde bei der Umsetzung nicht immer geachtet, so wird beispielsweise *Bubblesort* eingesetzt um *Blobs*, die sich in einem früheren Frame befunden haben, nach ihrer Entfernung zu einem *Blob* aus dem aktuellen Frame zu sortieren. Aufgrund dieser Mängel erhält die Touchlib eine negative Bewertung für ihre Codequalität. Der ursprüngliche Mangel an Programmdokumentation wird übrigens seit kurzem zum Teil behoben, denn mittlerweile hat ein Mitglied der NUI-Group ein Doxygen-Projekt für die Touchlib angelegt [Moo08].

Community Core Vision

Am 22.10.2008 wurde Tbeta ([NGC08b]) als Nachfolger der Touchlib veröffentlicht. Kurze Zeit später wurde das Projekt unter dem Namen *Community Core Vision (CCV)* fortgesetzt. Die Verwendung des Programms ist ähnlich wie bei der Touchlib geblieben. Bisher wird die Verfolgung von *Fiducials* nicht unterstützt. *Community Core Vision* setzt auf eine einfachere Benutzungsoberfläche, welche intuitiver zu bedienen ist, als es beim Vorgänger der Fall war. Auch der Funktionsumfang wurde erweitert, unter anderem um einstellbare Filter, die z. B. auch einen dynamischen *Background Remove* umfassen. Wie bereits der Background Remove Filter der Touchlib entfernt dieser den Hintergrund aus dem Bild, der immer gleich ist. Dies ist in *CCV* jedoch auch dynamisch möglich, wodurch sich beispielsweise die Lichtverhältnisse während des Betriebs leicht ändern können, ohne dass die *Blob Detection* beeinträchtigt wird. Zudem können weitere Filter als Module eingefügt werden. Zusätzlich wird Unterstützung geboten, indem das Kamerabild horizontal oder vertikal gespiegelt werden kann. Auch die Kalibrierung wurde gegenüber dem Vorgänger stark verbessert (vgl. [NGC08b]).

ReacTIVision

ReacTIVision[6] wurde an der Universität Pompeu Fabra in Barcelona entwickelt. Dort dient es zur Mustererkennung für den sogenannten reacTable. Dieser multitouch-fähige Tisch ist ein Synthesizer, bei dem jegliche Kontrollmöglichkeit über eine Multitouch-Eingabefläche realisiert worden ist. Die unzähligen Dreh- und Schieberegler eines gewöhnlichen Synthesizers werden in diesem Fall durch *Tangibles* ersetzt. Daher liegt der Schwerpunkt der Mustererkennung von ReacTIVision in der Unterscheidung und Verfolgung dieser *Tangibles*. Aber auch Finger werden als Eingabercursor erkannt. Im

[6] `http://mtg.upf.es/reactable/?software`, zuletzt besucht: 20.09.2009

Gegensatz zu *Tangibles* können sie auch verfolgt werden, wenn *FTIR* anstelle von *DI* verwendet wird (vgl. [MTG08]).

Ähnlich wie es auch bei der Touchlib der Fall ist, übernimmt ReacTIVision das Capturing der Videobilder selbst. Die Anzahl der unterstützten Kameratypen ist etwas geringer, weshalb die Kameraunterstützung eher als durchschnittlich bewertet wird. Die Qualität des Fingertrackings kann gesteuert werden, indem ein Wert für die durchschnittliche Größe eines Fingers angegeben wird. Da die Einstellungsmöglichkeiten damit im Vergleich zur Touchlib relativ gering sind, erhält auch die Konfigurierbarkeit eine durchschnittliche Bewertung (vgl. [MTG08]).

Da das *TUIO*-Protokoll im Rahmen desselben Projektes entwickelt worden ist, ist es nicht verwunderlich, dass ReacTIVision dieses verwendet. Bei Bedarf kann jedoch auch auf eine Ausgabe über das *MIDI*-Protokoll umgeschaltet werden.

Die Identifizierung der *Tangibles* geschieht im Falle von ReacTIVision basierend auf topologischen Merkmalen. Bei der Segmentierung der Kamerabilder wird ein Nachbarschaftsgraph für die gefundenen Regionen erstellt (siehe Abbildung 3.11). Dieser erfasst den Wechsel von dunklen und hellen Flächen. Immer wenn eine Region durch eines oder mehrere Löcher unterbrochen ist, erscheinen die Löcher als Kindknoten dieser Region im Nachbarschaftsgraphen. Die Löcher können nun selbst wieder helle Regionen enthalten, so dass sich nach und nach ein Baum konstruieren lässt, der sämtliche Regionen enthält.

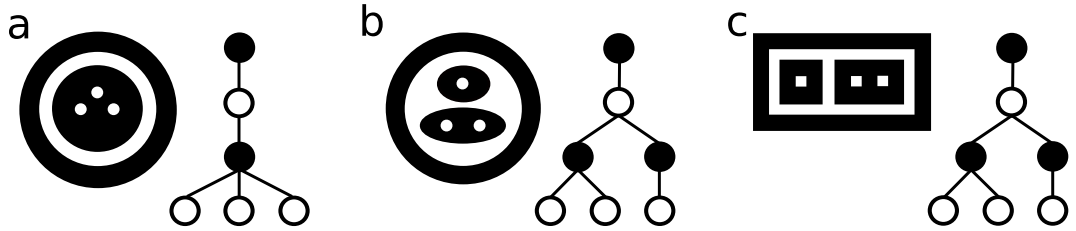


Abbildung 3.11: Beispiele für einfache Topologien und ihre Nachbarschaftsgraphen. Quelle: [BKJ05]

Die Nachbarschaftsgraphen können eindeutig identifiziert werden, so dass ein zugehöriges *Tangible* ebenfalls eindeutig bestimmt werden kann.

Weitere Bibliotheken

Zahlreiche weitere Projekte beschäftigen sich mit der Mustererkennung für optische Multitouch-Geräte. Oft ist es jedoch so, dass diese nur Teilaspekte erfüllen, oder gänzlich auf *DI* ausgelegt sind. Daher werden sie nicht näher aufgeführt.

Zusammenfassung

Es wurde zu Beginn im Detail vorgestellt warum die Mustererkennung für optische Multitouch-Geräte relevant ist. Daraufhin wurden die verschiedenen Arbeitsschritte von *Blob Detection* und *Blob Tracking* erläutert. Nach einem *Thresholding* werden mit Hilfe der sequentiellen Regionenmarkierung oder ähnlichen Verfahren Bildregionen identifiziert, die entweder durch einen Finger oder durch ein *Tangible* verursacht werden.

Schließlich wurden einige Softwarelösungen vorgestellt, die den gesamten Prozess von der Kameraaufzeichnung hin zum *Blob Tracking* automatisieren. Diese sind anhand der in Abschnitt 3.2.5 eingeführten Kriterien bewertet worden. Die Ergebnisse dieser Bewertung sind in Tabelle 3.4 zusammengefasst worden.

Kriterium	OpenCV	ReacTIVision	Touchlib	CCV
Kameraunterst.	•	•	+	+
Blob Detection	•	•	+	++
Blob Tracking	−	•	+	+
Protokolle	keine	TUIO, Midi	TUIO	TUIO
Konfigurierbarkeit	+	•	+	++
Technologie	keine	*DI*/*FTIR**	*DI*/*FTIR*	*DI*/*FTIR*
Code-Qualität	•	•	−	unbekannt
Lizenzmodell	BSD	GPL v2.0	BSD	MPL/MIT

Tabelle 3.4: Übersicht über die abgegebenen Bewertungen

Wie der Tabelle zu entnehmen ist, hat die Touchlib mit vier positiven Bewertungen ein sehr gutes Ergebnis erzielt. *Community Core Vision* bringt einige Neuerungen mit sich, die es gegenüber dem Vorgänger hervorheben. Daher wurde in der Relation zwischen diesen beiden zum Teil eine höhere Wertung für *CCV* abgegeben. Die Touchlib hat momentan noch den Vorteil, dass sie etwas älter ist und das daher eine große Community existiert, auf deren Erfahrungsschatz zurückgegriffen werden kann. *Community Core Vision* befindet sich dagegen noch in der Beta-Phase, weshalb auch keine Bewertung für den Code abgegeben wurde. Wie stabil und zuverlässig dieser ist, muss sich erst noch zeigen. Es ist allerdings abzusehen, dass *CCV* einen würdigen Nachfolger für die Touchlib darstellt. Aufgrund des größeren Funktionsumfangs und auch der Tatsache, dass das Projekt, im Gegensatz zur Touchlib, in absehbarer Zukunft weiter gepflegt wird,

sollte der Einsatz von *CCV* im Rahmen der Projektgruppe angestrebt werden. Sollten sich allerdings Probleme durch den sehr frühen Entwicklungsstand ergeben, sollte auf die Touchlib oder ReacTIVision zurückgegriffen werden.

Von einem gemeinsamen Einsatz von zwei oder mehr der aufgeführten Bibliotheken ist abzusehen, da jede von ihnen den gesamten Verarbeitungsprozess umfasst. Es gibt keine geeignete Möglichkeit, ihre Ergebnisse sinnvoll zu koppeln.

3.3 Gesten und ihre Erkennung

Als Mittel zur nonverbalen Kommunikation sind Gesten sehr ausdruckskräftig. Sie treten daher in vielen unterschiedlichen Formen auf. Viele Gesten werden unbewusst eingesetzt und wahrgenommen, was darauf schließen lässt, dass es sich bei ihnen um eine natürliche und intuitive Ausdrucksform handelt. Viele Forscher beschäftigen sich damit, wie Gesten zur Interaktion mit Maschinen eingesetzt werden können. Bekannt geworden ist die Multitouch-Technik durch Jeff Han, der bereits auf einigen Präsentationen erfolgreich Gesten wie das *Zoomen* mit zwei Fingern zeigen konnte.

Dieses Kapitel befasst sich mit den Grundsätzen von Gesten, ihrer Erkennung und dem aktuellen Stand der Techniken, die dazu eingesetzt werden. Die Arbeit der Projektgruppe *Visual Analytics* beschränkt sich aus technischen Gründen auf Gesten im zweidimensionalen Raum. Des Weiteren wird im Folgenden nicht jedes Detail der Erkennungstechniken betrachtet, da die Techniken für sich sehr komplex sind und eine ausführliche Darlegung den Rahmen dieser Arbeit sprengen würde. Stattdessen wird mehr Wert darauf gelegt, die grundlegenden Ideen der Gestenerkennung zu vermitteln.

3.3.1 Gesten

Gesten können Bewegungen oder eine Haltung des Körpers oder eines Körperteils beinhalten. Sie haben immer einen Bezug zu Raum und Zeit, da z. B. die Geschwindigkeit oder aber auch der Ort einer Geste ihre Bedeutung verändern könnten. Nach [KH90] lässt sich eine Geste wie folgt definieren und in die Mensch-Maschine Interaktion einordnen:

„*A gesture is a motion of the body the [sic!] contains information. Waving goodbye is a gesture. Pressing a key on a keyboard is not a gesture because the motion of a finger on it's [sic] way to hitting a key is neither observed nor significant. All that matters is which key was pressed.*“

Demnach ist es wichtig, verschiedene Arten von Gesten zu unterscheiden und zu divergieren, was eine Geste ist und was nicht. Das angegebene Beispiel *Drücken einer*

Taste sagt also aus, dass der Weg des Fingers bis zur Taste irrelevant ist. Es zählt nur die direkte Ausführung des Drucks auf die Taste. Die Tastatur kann hierbei auch nicht unterscheiden, wie schnell oder kräftig gedrückt worden ist.

Gesten sind nicht angeboren und müssen daher erlernt werden. Aus diesem Grund haben die meisten Gesten auch einen kulturellen Bezug. Gesten können danach gruppiert werden, ob sie isoliert erfolgen oder mit Hilfe von Objekten.

Im folgenden Abschnitt werden unterschiedliche Arten von Gesten genauer beschrieben.

Gestenarten

Es gibt grundsätzlich vier unterschiedliche Arten von Gesten: *symbolische*, *deiktische*, *ikonische* und *pantomimische* Gesten.

Symbolische Gesten Symbolische Gesten haben immer einen kulturellen und gesellschaftlichen Bezug. Innerhalb dieser Kulturen und Gesellschaften sind sie jedoch eindeutig zu verstehen. Ein Beispiel stellt die Geste für *OK* dar, die sich ergibt, wenn Daumen und Zeigefinger zu einem Ring zusammengeführt und die restlichen Finger abspreizt werden. Ein anderes Beispiel ist die Gebärdensprache, welche auch von Kultur zu Kultur unterschiedlich sein kann.

Deiktische Gesten Diese Art der Gestik beinhaltet das Zeigen und das Weisen einer Richtung mit einem Finger, der ganzen Hand oder aber auch mit Hilfe eines Zeigestocks. Das Zeigen wird bereits seit längerem in der Mensch-Maschine-Interaktion (auch *Human-Computer-Interaction (HCI)* genannt) verwendet. Man findet es beispielsweise in der Handhabung eines Track-Pads wieder. Hier wird mit dem Zeigen auf einen Punkt des Track-Pads ein Ort angezeigt.

Ikonische Gesten Hierbei handelt es sich um das Nachahmen von Bewegungen, Größenzuordnungen oder Formen. Es handelt sich um eine bildhafte Darstellung von Objekten oder Eigenschaften. Als Beispiel kann man hier die Geste *so klein mit Hut* angeben: Daumen und Zeigefinger werden so nahe aneinander gehalten, dass sie einen kleinen Zwischenraum einschließen.

Pantomimische Gesten Bei dieser Art von Gesten geht es darum, etwas nicht vorhandenes/unsichtbares darzustellen. Pantomimisch stellt man die Bewegungen oder Haltungen dar, welche mit dem zu zeigenden Objekt durchzuführen wären. Als Beispiel kann hier das pantomimische Fahren eines Motorrads eingeordnet werden: Die Hände werden so gehalten, als würden sie einen Lenker halten und eine Drehung der rechten Hand deutet das Gasgeben an.

Gestennotation

Die Art der Ausführung einer Geste muss notiert werden, damit andere Menschen oder der Computer sie korrekt verstehen können. Hier wird auf zwei unterschiedliche Arten der Gestennotation eingegangen. Eine Weitere folgt zu einem späteren Zeitpunkt.

Schriftliche Notation Die wohl einfachste Form zur Notierung einer Geste ist es, die Bewegung in vollständigen Sätzen zu beschreiben. Hierbei ist es wichtig, so genau wie möglich zu formulieren. Eine Notation kann z. B. so aussehen:

„*Den Daumen und drei Fingerspitzen aufsetzen und nach links bewegen*“

Piktogramme Piktogramme beinhalten die Abbildung einer Hand oder vergleichbarem mit dazugehörigen Symbolen und Zeichen, welche Bewegungen und Haltungen anzeigen. Als Beispiel werden hier Piktogramme der Internetpräsenz Fingerworks [Fin] verwendet.

Bei diesen Piktogrammen (s. Abb. 3.12) zeigen Pfeile eine Bewegung in entsprechender Richtung an. Eine Markierung mit der Farbe rot bedeutet eine Berührung der Multitouch-Fläche und blau steht für ein Tippen. Grün bedeutet, dass dieser Finger auf der Position bleibt, während eine Bewegung mit den rot-markierten Fingern durchgeführt wird.

Abbildung 3.12: Eine Multitouch-Geste von Fingerworks [Fin]

3.3.2 Von Singlepoint zu Multitouch

Es gibt verschiedene Formen von Gesteninteraktionen. Diese werden im Folgenden kurz erläutert, wobei insbesondere auf ihre Unterschiede eingegangen wird. Hauptsächlich werden Single-/Multipoint Eingaben von Single-/Multitouch-Eingaben unterschieden. Zudem können Eingaben diskret oder stetig erfolgen. Diskrete Eingaben sind z. B. Betätigungen von Knöpfen oder Schaltern in einer Benutzungsoberfläche. Stetige Eingaben erfolgen z. B. bei dem Zoomen von Objekten. Höhere Wichtigkeit für die Zukunft werden vor allem stetige Eingaben haben.

Singlepoint

Mit Singlepoint sind Interaktionen gemeint, welche nur deiktische Gesten ausführen, also das Zeigen auf etwas. Sie werden mit nur einem Finger ausgeführt, welcher z. B. die Multitouch-Fläche berührt. Sie sind schon länger bekannt und werden oft verwendet. Selbst die Ausführung eines *Drag and Drop* ist nach Bill Buxton [Bux07] nur eine Single-/Multipoint Interaktion. Die wohl am weitesten verbreitete und bekannteste Singlepoint-Hardware ist das Track-Pad in Notebooks. Es hat nahezu den gleichen Funktionsumfang wie eine normale Maus, da es Bewegungen und Klicks verarbeiten kann. Hier wird ein Klick entweder mit Hilfe einer Maustaste oder aber auch durch ein kurzes Antippen der Track-Pad-Fläche ausgeführt.

Singlepoint-Gesten werden auch auf Touchscreens verwendet. Hierzu zählen Geräte wie PDAs, Smartphones und auch Notebooks mit Touchscreens.

Singletouch

Singlepoint erlaubt nur eine sehr geringe Anzahl von Gesten. *Mausgesten* stellen eine naheliegende Erweiterung einfacher Singlepoint-Gesten dar. Eine solche Erweiterung führt zu dem Begriff *Singletouch*.

Eine vorgegebene Bewegung wird mit gedrückter Maustaste ausgeführt. Dies kann einfache Richtungsangaben oder auch ganze Formen beinhalten (s. Abbildung 3.13). Singletouch-Techniken sind bereits bekannt von PDAs mit Schrifterkennung.

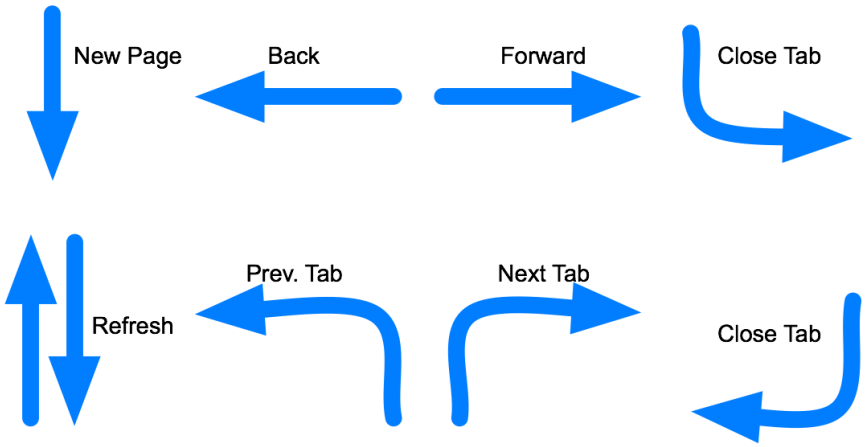


Abbildung 3.13: Einige Mausgesten

Doch auch Singletouch-Gesten können bei entsprechender Software schon sehr mächtig sein. Ein gutes Beispiel hierfür ist die Texterkennungs-Software Swype.

Abbildung 3.14: Swype [Inc08]

Die Software dient dazu, eine innovative, virtuelle Tastatur auf einem Touchscreen anzuzeigen. Grundsätzlich ähnelt sie einer virtuellen Tastatur, allerdings ist die Bedienungstechnik angeknüpft an eine Texterkennung, wie sie bei Mobiltelefonen eingesetzt wird. Dabei bewegt der Benutzer seinen Finger ohne Absetzen über die Buchstaben und schreibt so den Text. Hat man einen multitouch-fähigen Touchscreen, ist es möglich, auch mit mehreren Fingern auf der Tastatur zu tippen, um z. B. die Umschalttaste gedrückt halten zu können.

Multitouch

Anders als Singletouch-Gesten erlauben Multitouch-Gesten, mehrere Eingaben gleichzeitig zu verarbeiten und zu verwenden. Dadurch können z. B. Tastaturbefehle ergänzt oder ganz ersetzt werden. Hierbei spielt eine andere Form der Interaktion eine Rolle. Statt eine bestimmte Bewegung mit nur einem Finger auszuführen, kann bereits die Anzahl der Berührungen auf der Multitouch-Fläche unterschieden werden, so dass beispielsweise das Antippen mit drei oder vier Fingern bereits als Geste gedeutet werden kann. Dadurch kann Zeit für die Eingabe der Gesten eingespart werden.

Multitouch-Gesten erlauben auch eine größere Vielfalt der Eingabemöglichkeiten. Allein rechnerisch ergibt sich aus der Anzahl der benutzbaren Finger (z. B. alle fünf Finger einer Hand) eine höhere Anzahl an Kombinationen der Fingerbewegungen, als mit einem einzigen Finger. In der Abbildung 3.15 sind einige Gesten aufgeführt, welche mit mehreren Fingern arbeiten.

Nicht nur die Positionen der Finger auf der Multitouch-Fläche kann von Relevanz sein, sondern auch weitere Attribute wie die Druckstärke der Berührung. Dies ließe sich entweder durch weitere Komponenten, welche den Druck messen können, oder aber auch mit Hilfe der Größe der *Blobs* implementieren. Hierbei ist jedoch zu beachten, dass unterschiedliche Personen mit unterschiedlich großen Händen in gleicher Weise interagieren können müssen.

Abbildung 3.15: Links oben: Geste für *in Zwischenablage kopieren* / Rechts oben: Geste für *Einfügen* / Links unten: Geste für *Neu* / Rechts unten: Geste für *Drucken* [Fin] (Zeichenerklärung siehe Abschnitt 3.3.1)

Weiterhin kann es für eine multitouch-fähige Software wichtig sein, die Orientierung des Nutzers zum Multitouch-Tisch zu erkennen. Dies kann, wenn es die Technik zulässt, anhand des Winkels der Hand zum Tisch festgestellt werden. Somit kann die Ausrichtung der Schrift automatisch an den Benutzer angepasst werden.

Mit Multitouch-Techniken ist aber nicht nur die Interaktion einer Einzelperson möglich, auch sind Eingaben mehrerer Personen zugleich erlaubt. Allerdings ist es schwierig, unterschiedliche Benutzer mit Hilfe eines Multitouch-Tisches voneinander zu unterscheiden. Bisherige Techniken verwenden hierzu zusätzlich angebrachte Sensoren an den Händen der Benutzer oder durch unterschiedlich schwache Ströme, die jede der Personen durchfließen.

Nach [Bux07] sollte es völlig natürlich sein, mehr als nur einen Finger für *HCI* zu verwenden, da Menschen auch in der Realität nicht nur mit einem Finger Dinge ausführen, sondern Hände oder gar Arme für Gesten benutzen.

3.3.3 Gestenerkennung

Um Gesten erkennen zu können bedarf es Technologien der Mustererkennung, da jede Geste immer einem festen Muster folgt. Im Folgenden wird auf unterschiedliche Techniken der Mustererkennung eingegangen.

Dynamic Time-Warping

Das Dynamic Time-Warping verwendet Methoden der dynamischen Programmierung, da es ein Problem in mehrere Teilprobleme zerlegt. Bei diesem Verfahren geht es

hauptsächlich darum, zweidimensionale Sequenzen miteinander zu vergleichen. Dies wird hauptsächlich bei der Spracherkennung verwendet, lässt sich aber auch für die Erkennung von Gesten auf Multitouch-Eingabegeräten verwenden.

Um eine Geste mit einer anderen vergleichen zu können, müssen die Bewegungen jeder Geste zunächst auf ein zweidimensionales Muster heruntergerechnet und gespeichert werden. Trennt man die Daten der horizontalen und vertikalen Bewegung auf, so kann man zwei Diagramme erstellen. Hier befindet sich dann auf der X-Achse die Zeit und auf der Y-Achse jeweils die horizontale oder vertikale Bewegung. Im ersten Prozess werden die X-Achse der gespeicherten Geste und die X-Achse der soeben ausgeführten Bewegung angeglichen (s. Abb. 3.16). Dies erfolgt über eine Matrix, welche die Distanzen der Y-Werte enthält. Durch Linearisierung (Abb. 3.17) der Distanzen werden die Werte angeglichen.

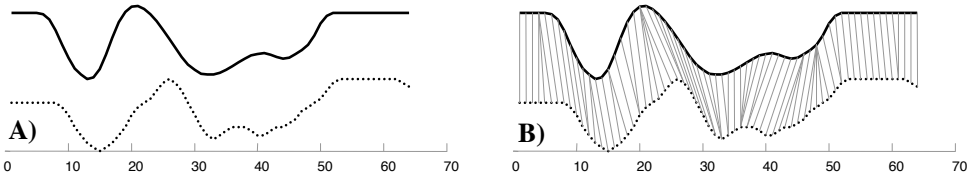


Abbildung 3.16: Angleichung der X-Achse beim Dynamic Time-Warping [Keo01]

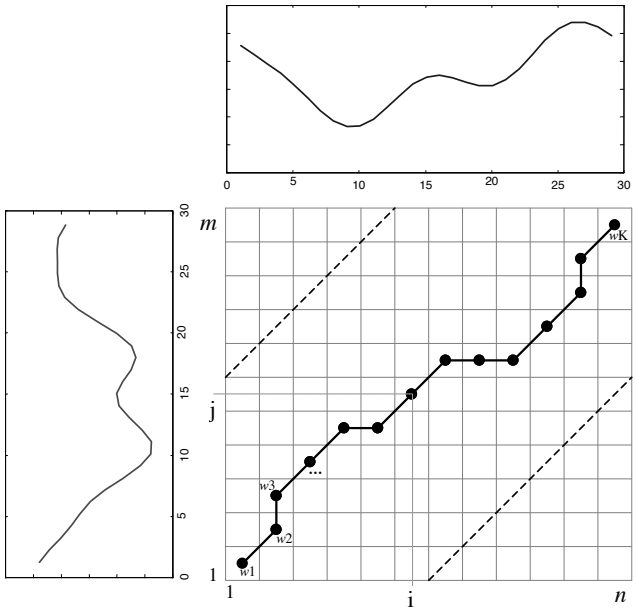


Abbildung 3.17: Die Abbildung zeigt die Abstände zwischen den beiden Sequenzen, welche anschließend linearisiert werden [Keo01]

Die Funktion (3.1) berechnet rekursiv die Linearisierung mit Hilfe von dynamischer Programmierung. $\lambda(i,y)$ ist hierbei der kumulative Abstand von i und j.

$$\lambda(i,j) = d(q,c) + \min\{\lambda(i-1,j-1), \lambda(i-1,j), \lambda(i,j-1)\} \tag{3.1}$$

Falls es keine großen Abweichungen mehr gibt, kann davon ausgegangen werden, dass beide Sequenzen zur gleichen Geste gehören.

Künstliche neuronale Netze

Eine weitere Möglichkeit zur Erkennung von Gesten ist die Nutzung von künstlichen neuronalen Netzen (*KNN*). Diese Vorgehensweise entstammt der Idee, die Funktionsweise des menschlichen Gehirns nach zu bilden.

Künstliche neuronale Netze sind dazu in der Lage, Muster zu erkennen und wichtige Merkmale der Eingabedaten zu extrahieren. Als Ergebnis erhält man eine Klassifikation. Diese gibt an, ob es sich bei einer Eingabe um eine entsprechende Geste gehandelt hat, oder nicht.

Zur Veranschaulichung der Vorgehensweise wird ein Beispiel zur Rechnung mit der *inklusive oder*-Operation aus [Str97] verwendet. Hier gibt es zwei binäre Eingabevariablen X_1 und X_2. Die jeweiligen Ergebnisse der Variablenpaare (3.3) sind bekannt und können für den Lernprozess des *KNN* verwendet werden. In diesem Lernprozess passt das *KNN* nach und nach die so genannten Eingangsgewichte (w_i) an, bis die Eingabewerte die erwartete Ausgabe ergeben. Das Ergebnis eines Durchlaufs ergibt sich aus der Funktion

$$net = w_1 x_1 + w_2 x_2 \tag{3.2}$$

wofür anschließend ein Schwellenwert Θ eingeführt werden muss, um als Ausgabewert nur zwei Klassen zuzulassen. Hier wird $\Theta = 0,5$ gesetz,t um je eine Klasse für Eins und Null zu erhalten, da alle Ausgangswerte $< 0,5$ Null ergeben und alle Ausgangswerte $\geqslant 0,5$ Eins ergeben.

$$\begin{aligned} 0 \cup 0 &= 0 \\ 0 \cup 1 &= 1 \\ 1 \cup 0 &= 1 \\ 1 \cup 1 &= 1 \end{aligned} \tag{3.3}$$

Als Gewichtungen werden zufällige oder manuell vorgegebene Werte eingesetzt und der Lernprozess beginnt. Falls bei einem Test einer Eingabe mit erwarteter Ausgabe (Δ)

dieser fehlschlägt, wird das näherliegende Gewicht geändert (mit einem vorgegebenen Wert, Lernrate δ).

$$w_i(t+1) = w_i(t) + \delta * \Delta * X_i \tag{3.4}$$

Die Gewichte werden in diesem Prozess so lange angepasst, bis alle erwarteten Werte erreicht werden.

Künstliche neuronale Netze bieten somit die Möglichkeit, mit relativ wenig Trainingsdaten eine Gestenerkennung zu erstellen, da nicht jedes Detail einer Geste programmiert werden muss, sondern automatisch durch das *KNN* erkannt werden kann. Probleme können auftauchen, wenn das *KNN übertrainiert* wird, da die Muster dann zu genau sind und nicht mehr, wie gewünscht, generell. Somit ist es z. B. nicht mehr gegeben, dass Gesten von unterschiedlichen Benutzern erkannt werden können.

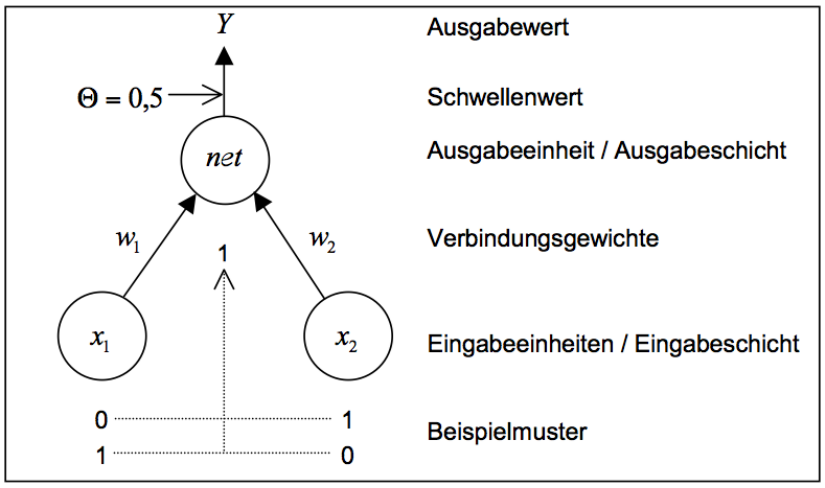


Abbildung 3.18: Ein simples Beispiel eines *KNN* [Str97]

Um mehr Eingabedaten verwenden und mehr Ausgabeergebnisse erhalten zu können, bedarf es mehrschichtiger *KNN*s. Diese Schichten vereinen mehrere Verarbeitungseinheiten, welche eine ähnliche Verhaltensweise bei der Datenverarbeitung aufweisen.

Ein Problem eines *KNN* ist, dass es keine Möglichkeit gibt, zeitabhängige Daten einzugeben. Hierzu werden erweiterte dynamische *KNN*s wie rekursive *KNN*s und *Time Delay Neural Network (TDNN)* benötigt.

Bei rekursiven *KNN*s dienen die Ausgaben nach einer vorgegebenen Zeit wieder als Eingabe [San97], ähnlich wie bei *Feedback-Registern* (s. Abb. 3.19). Hierbei ist jedoch zu beachten, dass das Trainieren der Daten gut überwacht werden muss, falls das

Problem nicht detailliert genug beschrieben ist. Andernfalls kann das *KNN* nicht korrekt laufen.

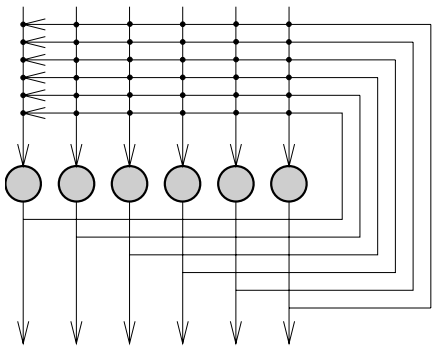


Abbildung 3.19: Ein einschichtiges rekursives *KNN* [San97]

Bei dem *TDNN* werden die aktuellen Daten und folgenden Daten (also zeitbasierend) in Shift-Registern gespeichert. Da es grundsätzlich auf statischen *KNN*s basiert, kann es so einfach implementiert und trainiert werden wie statische *KNN*s. Gesten mit unterschiedlicher Geschwindigkeit korrekt zu erkennen, stellt jedoch ein Problem dar [San97].

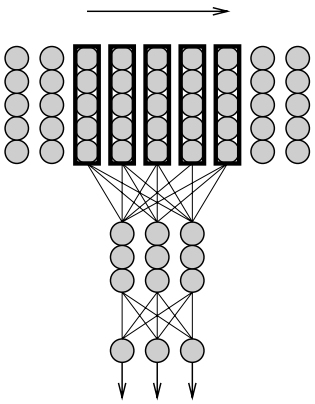


Abbildung 3.20: Beispiel eines *TDNN* [San97]

Eine weitere Möglichkeit der Gestenerkennung mit *KNN*s ist die Verwendung von Bayes *KNN*s. Hier sind die Eingabeeinheiten stochastische Ereignisse, wobei die Zustände die Wahrscheinlichkeit beinhalten, dass das Ereignis zu einem bereits erfolgten Ereignis gehört [San97].

Hidden-Markov Modell

Anschaulicher und bei großen Problemen auch verständlicher als *KNN*s sind *Hidden Markov Models (HMMs)*. Hierbei handelt es sich um ein stochastisches Modell, welches aus n Zuständen besteht, die nicht direkt beobachtet werden können. Zu jedem Zeitpunkt t emittieren diese Zustände sichtbare Symbole. Ein *HMM* lässt sich nach [Pop07, Dör03] wie folgt formal definieren:

HMM:

$$\lambda = (S, \pi, V, A, B) \tag{3.5}$$

Zustandsmenge

$$S = \{s_1, s_2, \ldots, s_n\}, n \in \mathbb{N} \tag{3.6}$$

Wahrscheinlichkeiten der Zustandsübergänge von s_i nach s_j:

$$a_{ij} = P(q_{t+1} = s_j \mid q_t = s_i), 1 \geq i, j \geq n) \tag{3.7}$$

Emissionswahrscheinlichkeiten $B = \{b_j(k)\}$ der Beobachtungswahrscheinlichkeit von v_k nach s_j:

$$b_j = P(O_t = v_k \mid q_t = s_j), 1 \geq j \geq n, 1 \geq k \geq m, O_t \in V \tag{3.8}$$

Beobachtungssequenz:

$$O = \{O_1, =_2, \ldots, O_n\} \tag{3.9}$$

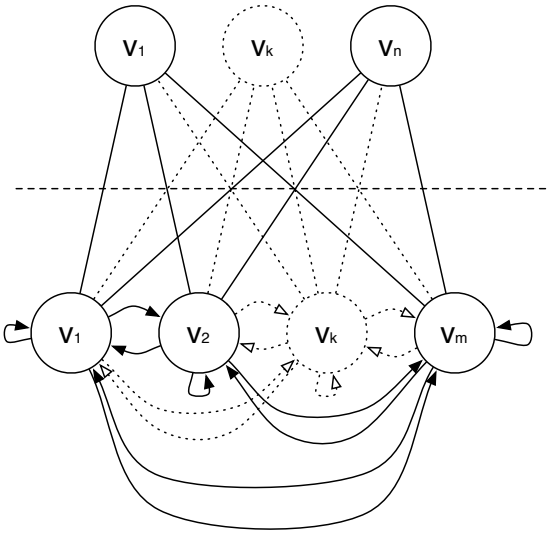


Abbildung 3.21: Beispiel eines ergodischen Hidden-Markov Modells

Bei dem ergodischen Modell (s. Abb. 3.21) kann jeder Zustand in einem Schritt erreicht werden. Anders ist es z. B. bei dem *Links-Rechts HMM*, welches sich für zeitabhängige Modelle, bei denen sich Eigenschaften über die Zeit ändern können, besser eignet [LK99].

Zu einer Beobachtungssequenz O (3.9) muss die Wahrscheinlichkeit $P(O \mid \lambda)$ für die Zustandsfolge des *HMM* $\lambda = (A, B, \pi)$ ermittelt werden [Pop07]. Unter Verwendung des *Vorwärts-Algorithmus* kann diese Wahrscheinlichkeit P mit einer *Vorwärtsvariable* $v_t(i)$ errechnet werden:

$$v_t(i) = P(O_1, O_2, \ldots, q_t = s_i \mid \lambda) \tag{3.10}$$

$$v_1(i) = \pi_i b_i(O_1), \text{ mit } 1 \geq i \geq n \tag{3.11}$$

$$v_{t+1}(j) = \left(\sum_{i=1}^{n} v_t(i) a_{ij} \right) b_j(O_{t+1}), \text{ mit } 1 \geq t \geq T-1 \text{ und } 1 \geq j \geq n \tag{3.12}$$

$$P(O \mid \lambda) = \sum_{i=1}^{n} v_T(i) \tag{3.13}$$

Anschließend muss die Wahrscheinlichkeit P maximiert werden [Pop07], um das *HMM* zu trainieren. Dies kann mit dem *Baum-Welch-Algorithmus* erreicht werden, indem dieser eine Wahrscheinlichkeitsverteilung errechnet. Dies wird hier nicht weiter beschrieben und kann bei [Pop07] nachgelesen werden.

3.3.4 Bibliotheken

Das Thema Multitouch-Gesten ist noch recht jung, weswegen es noch nicht allzu viele existierende Bibliotheken dazu gibt. Das Framework MultiTouchVista[7] unterstützt bereits rudimentär Multitouch-Gesten. Es baut auf dem .NET Framework *StylusInput* auf, welches u.a. zur Erkennung von Gesten auf Tablet PCs dient. Das Framework arbeitet mit einer *ContactHistory*, welche alle Berührungen der Multitouch-Oberfläche zunächst zwischenspeichert. Sobald kein Kontakt mehr zur Multitouch-Oberfläche existiert, wird der *GestureRecognizer* des .NET aufgerufen. Mit MultiTouchVista können also die gleichen Gesten erkannt werden, wie auch auf Tablet PCs, was bedeutet, dass Multitouch-Gesten hier nur aus gleichzeitigen Singletouch-Gesten bestehen können, die das Framework bereits kennt.

Einen weiteren Ansatz verfolgt die NUI-Group[8] mit der GestureLib [Nui08]. Sie soll direkt an TBeta (ehemals TouchLib) anknüpfen und Gesten an die Anwendungsschicht

[7] `http://www.codeplex.com`, zuletzt besucht: 20.09.2009
[8] `http://www.nuigroup.com`, zuletzt besucht: 20.09.2009

weiterleiten können. Hier soll eine auf *XML*-basierende Sprache implementiert werden, die *Gesture Definition Markup Language (GDML)*. Dieses Framework befindet sich noch in der konzeptionellen Entwicklung.

3.4 Zusammenfassung

Im vorangegangenen Abschnitt wurde gezeigt, wie mit Hilfe von verschiedenen Technologien Berührungen auf einem Multitouch-Tisch erkannt werden können. Die vorgestellten Technologien *FTIR*, *DI* und *DSI* basieren auf dem Auswerten eines Infrarot-Signals, welches durch Berührung direkt oder indirekt beeinflusst wird. Somit ist unter anderem die Erkennung von Fingern möglich. Die *Blob Detection* ermöglicht schließlich die softwareseitige Erkennung dieser Berührungen, *Blobs* genannt, welche durch die *Blob Tracking*-Software analysiert werden und als Bewegungen interpretiert werden. Damit nicht jede Multitouch-Software diesen komplexen Vorgang reimplementieren muss, ist die Software als eigene Komponente ausgelagert und kommuniziert mit anderen Programmen über das *TUIO*-Protokoll.

Aus den gesendeten *TUIO*-Daten ist es schließlich möglich, Gesten zur Bedienung des Programms zu erkennen. Wie gezeigt wurde, bestehen zur Erkennung von Mustern bereits viele ausgereifte Techniken, allerdings scheint die Forschung im Bereich der Multitouch-Gesten noch keine alltagstauglichen Ergebnisse zu haben. Die bekannten Techniken weisen gute Ansätze auf, doch meist verlangen diese eher nach Gesten, die auf einem einzigen Pfad basieren. Multitouch-Gesten beinhalten jedoch mehr als nur einen Pfad, weshalb sie weitaus komplexer sind.

Kapitel 4

Programmoberflächenentwicklung für Multitouch-Technologien

In diesem Kapitel werden Technologien der Oberflächenprogrammierung in Bezug zu Multitouch-Technologien untersucht. Dazu werden zunächst verschiedene Technologien für die Darstellung von Diagrammen verglichen und gegeneinander abgewogen. Es folgt anschließend eine genauere Einführung in eine dieser Technologien, der *WPF*. Da die Elemente der Benutzungsschnittstelle mit Hilfe von Gesten gesteuert werden sollen, folgt schließlich eine Betrachtung und ein Vergleich von bestehenden *WPF* Multitouch-Frameworks, die solch eine Anbindung gewährleisten.

4.1 Visual Programming Toolkits

In diesem Abschnitt wird erläutert, welche Ziele und Anforderungen an die Grafik-Programmierung gestellt werden. Die Visualisierung der Daten wird in Diagrammform erfolgen, was vom Grundprinzip her eine recht einfache Aufgabe ist, sich allerdings in sehr unterschiedlicher Qualität umsetzen lässt. Bei den Diagrammen gibt es die klassischen 2D Diagramme (z. B. Liniendiagramm oder Tortendiagramm) und die 3D Diagramme. Mit 3D Diagrammen sind hier Diagramme gemeint, die Daten in einem 3D Raum visualisieren, nicht 2D Diagramme, die um 3D Effekte erweitert wurden (z.B. ein Balkendiagramm, das Quader anstelle von Rechtecken verwendet). Solche und andere Effekte stellen eine weitere Anforderung da. Neben dem erwähnten 3D Effekt sind auch Transparenz, Schatten oder komplexe Farbverläufe typische Beispiele. Das letzte wichtige Kriterium ist die Animierbarkeit der Diagramme. Dies bedeutet, dass sich z. B. die Balken eines Balkendiagrammes *ausfahren* oder das eine Animation benutzt wird, um ein Diagramm um die Zeitdimension zu erweitern. Diagramme im Allgemeinen wurden in Abschnitt 2.2 erläutert. Neben den Diagrammen ist die weitere grundlegende Aufgabe, mit dem ToolKit eine Benutzungsoberfläche, die speziell an die Gegebenheiten des Multitouch-Tisches angepasst ist, zu entwickeln. Das bedeutet, dass sie nicht auf die Möglichkeiten eines Standard *GUI* ToolKits eingeschränkt ist.

4.1.1 Kriterien

Folgende Kriterien der ToolKits werden bei der Betrachtung der ToolKits verwendet:

- Expression Power: Gibt an, ob mit dem ToolKit alles umgesetzt werden kann oder ob es Einschränkungen gibt (z.B. keine 3D Grafiken möglich sind). Hier wird nochmals zwischen 2D und 3D Grafiken, Effekten (Schatten, Transparenz etc.) und Animierbarkeit unterschieden.
- High Level: Gibt an, wieviel Arbeit dem Programmierer durch das ToolKit abgenommen wird und wieweit es darunter liegende Schichten abstrahiert.
- Tools: Beurteilt die Tools wie *Integrated Development Environment (IDE)*, Debugger und ähnliche, die es für das ToolKit gibt.
- Performanz: Gibt die Leistungsfähigkeit zum Darstellen komplexer Grafiken an.
- Dokumentation: Beurteilt die Dokumentation des ToolKits, zu der Handbücher und Onlinedokumentationen der Hersteller, Bücher Dritter, Communities und andere Hilfeseiten gehören.
- Integration: Beurteilt, wie sich das Programm mit Multitouch-Bibliotheken verwenden lässt.
- Anmerkungen: Betrachten von besonderen Eigenschaften, die noch nicht erwähnt wurden und nicht zu den primären Zielen gehören, aber trotzdem erwähnenswert sind.

4.1.2 ToolKits

Die hier ausgewählten ToolKits haben alle prinzipiell das Potential für das Projekt in Frage zu kommen. ToolKits, die von vornherein nicht in Frage kommen, da sie z. B. über keinerlei Hardwarebeschleunigung verfügen, wurden nicht betrachtet.

Betrachtete ToolKits

- Low Level ToolKits – *OpenGL*, *DirectX*
- Diagramm Bibliotheken
- GUI ToolKits (*WPF* und Qt)
- *WPF*

Low Level ToolKits

Die wesentlichen *Low Level ToolKit (LLTK)* sind *OpenGL*[KG08] und *DirectX*[Mic08b] bzw. ToolKits, die darauf aufbauen, wie der *Simple DirectMedia Layer (SDL)* [Tea08]. Diese ToolKits ermöglichen direkten Zugriff auf die Grafikhardware und erlauben damit die höchste Performanz.

Expression Power **2D:** Die *Graphics Processing Unit (GPU)*s aktueller Grafikkarten sind auf das Verarbeiten von 3D Grafiken ausgelegt. Da es sich bei 2D Grafiken prinzipiell um einen Unterbereich der 3D Grafiken handelt, lassen sich die *LLTK* auch sehr gut dafür einsetzen. Dies lässt sich daran erkennen, dass 2D Bibliotheken manchmal *OpenGL* einsetzen, um das Rendern zu beschleunigen. Cairo[Mac08] oder Adobe in ihrer CS4 Programm Serie setzt auf *DirectX* 9.0 und *OpenGL* 2.0[Asi08b].
3D: Da die *LLTK*s gerade für schnelle 3D Grafiken entwickelt wurden, können eigentlich alle 3D Diagramme umgesetzt werden.
Effekte: Mittels Shadern lassen sich alle Effekte in guter Geschwindigkeit realisieren.

High Level Die Vorteile in Ausdruckskraft und Performanz der *LLTK*s werden durch große Abstriche im Bereich der Abstraktion und Komfortabilität erreicht. Die Grundbibliotheken an sich konzentrieren sich auf eine sehr hardwarenahe Umsetzung von 3D Grafiken und sind kaum darauf ausgelegt, leicht oder schnell benutzbar zu sein.

Performanz Aufgrund des direkten Hardwarezugriffs und dadurch, dass keine Bindung an ein Framework gegeben ist, können problemspezielle Lösungen entwickelt werden, welche in der Regel performanter sind als Toolkit-Lösungen. Auch wird bei manchen anderen Lösungen ein weiterer Interpreter (z. B. Flash und Java) eingesetzt, anstatt native zu arbeiten, oder es kommt Verwaltungsaufwand durch ein Framework hinzu (z. B. *WPF*[Fer06]). Allerdings gibt es bei *LLTK*s einen höheren Arbeitsaufwand und damit die Gefahr (im Vergleich zu ToolKits, die auf einem höherem Level arbeiten) durch eine schlechte Umsetzung trotz allem ein langsameres Programm zu erzeugen. Adam Nathans schreibt dazu: „*it's easy to write a DirectX program that is faster than the same program in* WPF *– but it's even easier to write a program in DirectX that is slower than the* WPF *application* “ [Nat06].

Tools Da *LLTK*s meist als einfache Bibliotheken realisiert sind, lassen sie sich in alle größeren Programmiersprachen und *IDE* einbinden und zeigen sich auch hier wieder sehr flexibel. Dies ist allerdings meist eine recht rudimentäre Unterstützung. Es werden noch Debugger für *OpenGL* oder Direct3D benötigt. Zusätzlich gibt es Tools zum Erstellen und Debuggen von Shadern. Prinzipiell muss bei *LLTK*s

aber öfter auf diese meist nicht in die eigentliche *IDE* integrierten Tools zurück gegriffen werden, was sie oft unkomfortabler macht.

Dokumentation Sowohl *OpenGL* als auch *DirectX* sind seit vielen Jahren etablierte ToolKits, die daher über eine sehr große Dokumentation verfügen. Sowohl die Hersteller haben ausführliche und vollständige Dokumentationen, als auch Dritte mit einem großen Angebot an Büchern. Außerdem sind auch viele Hilfestellungen in Form von Communities oder Schulungen verfügbar.

Integration Die Flexibilität führt auch bei der Integration mit Multitouch-Bibliotheken dazu, dass sich *LLTK*s auf der einen Seite sehr gut mit beliebigen Bibliotheken kombinieren lassen, es aber auf der anderen Seite keine Frameworks gibt, die speziell auf die Probleme der *LLTK*s zugeschnitten wären oder hier Arbeit abnehmen würden.

Anmerkungen *OpenGL* als *LLTK* würde die größte Flexibilität bezüglich der Betriebssystemauswahl erlauben. Momentan gibt es drei verschiedene Shader Programmiersprachen (OpenGL Shader Languages (GLSL), Microsoft DirectX High–Level Shader Language (HLSL) und Nvidias Cg), die alle unterschiedliche Merkmale aufweisen. Durch die Wahl der Sprache, z. B. MS DirectX High-Level, wird gegebenfalls eine Plattformbindung erreicht. Ein *LLTK* würde wahrscheinlich auch immer mit einem anderen *GUI* ToolKit kombiniert werden, da eine performante Darstellung nur bei den Diagrammen kritisch ist. Meist wirkt der z.B. *OpenGL* Bereich eines Programmes optisch signifikant anders, so dass eventuell die Programmkontinuität gestört wird. Außerdem ist keine Kombination von (Standard) Widgets mit dem Diagramm möglich.

Graph Bibliotheken

Graph Bibliotheken sind darauf spezialisiert, Graphen oder Diagramme darzustellen. Meist setzen sie auf einem größeren Framework auf, wodurch sich ergibt, dass das komplette Programm dieses Framework einsetzen sollte. z. B. das Graphical Editing Framework, das eine Abhängigkeit zum SWT-Framework erzeugt, oder JGraph, das an Swing bindet.

Expression Power Die Möglichkeiten dieser Bibliotheken sind auf einen sehr speziellen Bereich begrenzt. Einfache, von dieser Bibliothek unterstützte Diagramme sind problemlos einsetzbar und meist vielfältig gestaltbar. Außerhalb des normalen Funktionsumfangs sind jedoch schnell eigene Erweiterungen nötig, deren Aufwand auf Grundlage der benötigten Einarbeitungszeit in das Framework häufig groß ist. Des Weiteren ist es oft technisch nicht möglich, diese Frameworks auf 3D zu erweitern.

High Level Graph-Bibliotheken erledigen sehr viel der internen Logik von Diagrammen. Bei der Verwendung dieser Bibliotheken liegt der Fokus auf dem Aussehen und der Anordnung der Diagramme. Allerdings bieten die Diagramme meist eine eingeschränkte Funktionalität. Ist es notwendig diese zu erweitern, müssen die Interna der Bibliothek berücksichtigt werden. Eine Einarbeitung in eine Bibliothek, um diese zu erweitern ist meist sehr aufwendig.

Tools Da Graph Libraries nur Erweiterungen bestehender ToolKits sind, werden meist keine speziellen Tools angeboten, um sie zu unterstützen. Diese sind auch nicht in dem Maße nötig wie z.B. bei den *LLTK*s, welche auf spezieller Hardware laufen.

Performanz Die Performanz ist meist in kleinen Bereichen akzeptabel, aber nicht auf große Datenmengen ausgelegt. Auch sind die Frameworks, auf denen sie aufbauen, oft nicht für z.B. Animationen ausgelegt, was sich oft in schlechter Performanz in diesem Bereich bemerkbar macht.

Dokumentation Die Dokumentation dieser Projekte ist oft auf die (nicht immer vollständige) Herstellerdokumentation beschränkt, Bücher oder Onlineangebote sind sehr selten. Es existiert aber in aller Regel mindestens eine Hilfe-Plattform im Internet.

Integration Die Integration an Multitouch-Bibliotheken hängt von dem darunter liegenden Framework ab. Für alle Frameworks sind prinzipiell Anbindungen vorhanden – wenn auch in sehr unterschiedlicher Ausprägung.

Anmerkungen Dies sind alles allgemeine Eigenschaften von Diagramm Bibliotheken, die natürlich bei einer speziellen Bibliothek stark davon abweichen können.

Windows Presentation Foundation

Windows Presentation Foundation ist Microsofts Nachfolger von Windows Forms (Windows GUI), der mit Windows Vista eingeführt wurde, sich aber auch unter Windows XP installieren lässt.

High Level *WPF* ist eines der wenigen *GUI* ToolKits, das von Grund auf mit Unterstützung für Animationen, Vektorgrafik und Multimedia entwickelt wurde. Die Umsetzung erfolgt daher wesentlich integrativer als bei ToolKits, die erst nachträglich um entsprechende Unterstützung erweitert wurden. Mit *WPF* hat Microsoft auch die *eXtensible Application Markup Language (XAML)* eingeführt. *XAML* ist eine *XML*-basierte Sprache zum Beschreiben von *GUI*s. *WPF* ermöglicht durch *XAML* eine gute Abstraktion von Aussehen und Verhalten.

Tools *WPF* wird durchweg von den Microsoft Visual Studio Tools unterstützt. Zusätzlich gibt es noch Microsoft Expression Blend zum Gestalten von *GUI*s, sowie Microsoft Expression Design zum Zeichnen.

Dokumentation *WPF* verfügt über eine gute Herstellerdokumentation. Außerdem ist eine Community mit zusätzlichen Artikeln und Foren von Microsoft verfügbar. Auch von Drittanbietern gibt es zahlreiche Bücher, Tutorials und Communities.

Integration Es gibt Multitouch-Frameworks, die sich in die Struktur von *WPF* integrieren.

Performanz *WPF* ist hauptsächlich darauf ausgelegt, möglichst vielfältige und einfache Möglichkeiten bereitzustellen, als besonders performante Darstellung. Es existiert keine Möglichkeit für den direkten Hardwarezugriff und, da *WPF* die Daten verwaltet, entsteht auch hier ein gewisser Overhead. *WPF* ist eher für herkömmliche Desktopprogramme als für *„high–end scientific vizualizations“* optimiert[Fer06].

Anmerkung Der Einsatz von *WPF* führt zu einer starken Plattformabhängigkeit. Eventuell könnte es möglich sein, eine Silverlight Version des Programmes zu erstellen.

Qt

Qt ist ein plattformunabhängiges *GUI* ToolKit von Qt Software, auch unter dem Namen TrollTech bekannt. Die Umbenennung erfolgte vor kurzem nach der Übernahme durch Nokia.

Expression Power Qt zeichnet sich durch eine direkte Integration von Vektorgrafiken im *Scalable Vector Graphics (SVG)* Format aus. Dabei können die Vektorgrafiken nicht nur in dafür vorgesehen Container Widget verwendet werden, sondern auch als Aussehen für Standard Widgets eingesetzt werden. Qt bringt allerdings keinerlei Unterstützung für 3D Grafiken oder Effekte mit. Diese ließen sich also nur durch aufwendig selbst entwickelte Softwarelösungen oder durch die Integration eines *LLTK*s realisieren.

High Level Das Qt ToolKit beinhaltet eine große Anzahl von Funktionalität für Standardaufgaben, die über den reinen *GUI* Bereich hinausgehen. Dies nimmt einem viel Arbeit ab. Allerdings ist die Trennung von *GUI* und Logik nicht so weit ausgereift wie bei *WPF* und *XAML*.

Tools Qt verfügt über einen eigenen Designer und Tools für die Build ToolChain, außerdem Plugins für Visual Studio und Eclipse. Qt lässt sich damit sehr gut in bestehende Entwicklungsumgebungen integrieren und in einen durchgängigen Arbeitsablauf mit weiteren Tools für diese Entwicklungsumgebung kombinieren (z.B. Testframeworks).

Performanz Qt läuft native auf jeder Plattform ohne spezielle Laufzeitumgebungen, was ihm einen gewissen Geschwindigkeitsvorteil verschaffen sollte. Des Weiteren setzt Qt OpenGL ein, um SVG Grafiken schneller zu rendern.

Dokumentation Es gibt eine gute und vollständige Herstellerdokumentation zu Qt, auch wenn sie nicht so umfangreich ist (besonders im Bezug auf Beispiele) wie die MSDN Library oder Suns Java API. Es sind des Weiteren zahlreiche Bücher, Online-Hilfen, wie Foren etc., vorhanden, die wiederum aufgrund der geringeren Verbreitung von Qt im Vergleich zu *WPF* quantitativ geringer ausfällt.

Integration Es existieren Bibliotheken für C/C++, die sich natürlich mit Qt kombinieren lassen. Es sind allerdings keine Erweiterungen, die sich direkt in das Event Modell von Qt integrieren würden.

Anmerkungen Qt ist plattformunabhängig.

Flash

Adobe Flash ist eine ursprünglich für nur kleine Multimediainhalte entwickelte Technologie. In den letzten Jahren wurde Flash jedoch erweitert, um es zu einer vollwertigen Plattform für Programme aller Art zu machen. Die neueste Generation der Skriptsprache ActionScript 3.0 unterstützt alle Paradigmen, die für eine höhere objektorientierte Programmiersprache notwendig sind. Im Herbst 2008 hat Adobe auch eine Betaversion von Alchemy [Asi08a], einem Compiler, der C/C++Programme für die *ActionScript Virtual Machine 2 (AVM2)* erstellen kann, veröffentlicht.

Expression Power Adobe Flash ist für 2D Multimedia Projekte ausgelegt und kann in diesem Bereich sowohl mit Vektorgrafiken als auch mit pixelbasierten Bildern problemlos umgehen. Seit der Version 10 ist es auch möglich, die *GPU* zu benutzen, um eigene Effekte zu erzeugen. Es ist jedoch nicht möglich, einen direkten Zugriff auf die *GPU* zu erhalten, und, da 3D Grafiken nicht vorgesehen sind, bietet Flash hier keine Lösungen. Diese könnten also nur mit einer eigenen Softwarelösung in Flash erfolgen, für die Flash nicht genügend performant ist.

High Level Die Bibliothek von Flash ist recht umfangreich und erleichtert besonders das Arbeiten mit Multimediainhalten. Im Flash Authoring Tool – auch Flash genannt – lassen sich sehr viele multimediabezogene Aufgaben grafisch erledigen.

Tools Adobe stellt unter dem Namen Flash – oder auch Flash Authoring Tool – eine Umgebung bereit, die besonders für die grafische Bearbeitung geeignet ist. Außerdem vertreibt Adobe auch eine *IDE* auf Basis von Eclipse unter dem Namen Flex. Allerdings sind beide Tools kostenpflichtig.

Performanz Die Multimedia-Performanz von Flash ist recht gut, besonders, wenn sie sich mit den *GPU* Effekten realisieren lässt. Allerdings laufen alle Flash

Programme in einer virtuellen Maschine, wodurch sie langsamer werden. Adobe gibt als Richtlinie an, dass eine in C/C++geschriebene Bibliothek in Flash meist um den Faktor zwei bis zehn langsamer ist als die entsprechende native Version. In ActionScript 3 geschriebene Programme sind meist nochmal um den Faktor zehn langsamer. Natürlich ist dies programmabhängig und es bezieht sich darauf, wenn keine Flash Funktionen genutzt werden. [Asi08a]

Dokumentation Eine gute Herstellerdokumentation ist vorhanden und auch viele zusätzliche Materialien, wie Online-Hilfen und Bücher. Besonders ActionScript 3 ist nicht so verbreitet wie *WPF* oder Java, wodurch hier weniger Angebote von Dritten verfügbar sind.

Integration Eine Kommunikation mit Touchlib ist möglich, aber es gibt keine direkte Anbindung für Flash.

Anmerkungen Flash ist plattformunabhängig und würde auch eine Webversion ermöglichen.

4.1.3 Fazit

In der Tabelle 4.1 wurden die Eigenschaften bewertet und so quantisiert einander gegenüber gestellt.

		Low Level	Graph Lib	Qt	WPF	Flash
Expression Power	2D	++	+	++	++	++
	3D	++	−−	−	++	−−
	Effekte	++	−−	−	+	+
High Level		−	++	•	+	+
Tools				+	++	•
Performanz		+++	•	+	++	+
Dokumentation		++	•	+	++	+
Plattform		w,l,m		w,l,m	w	w,l,m
Multitouch		•	•	•	+	•

Tabelle 4.1: Tabelle der Einschätzungen der einzelnen ToolKits

Wägt man die Kriterien Aufwand, Möglichkeiten und Geschwindigkeit, gegeneinander ab, zeigt sich, dass *WPF* die eindeutig beste Plattform ist. Nur *LLTK*s hätten einen Vorteil bezüglich der Geschwindigkeit, sind aber mit einem überproportional größerem Aufwand behaftet. Die Zielplattform ist recht genau definiert (PC, der den Multitouch steuert), wodurch die Plattformunabhängigkeit vernachlässigt werden kann. Ein weiterer Vorteil von WPF ist die Möglichkeit, Expression Blend einzusetzen, das beim Erstellen der *GUI* viel Arbeit abnehmen kann.

4.2 Einführung in die Windows Presentation Foundation

Im Jahr 1985 kam die erste Windows Version auf den Markt. Für die Programmierung wurde eine in C geschriebene Schnittstelle, Windows *Application Programming Interface (API)*, angeboten, die aus drei *Dynamic Link Library (DLL)* bestand: gdi32.dll, kernel32.dll und user32.dll. Mit Hilfe dieser Bibliotheken war es möglich, Fenster und weitere grafische Komponenten zu erzeugen und sie zu manipulieren. Es standen auch weitere Systemfunktionen zur Verfügung. Die Verwendung dieser *API* war aber äußerst kompliziert. Die Funktionsaufrufe waren auf Betriebssystemebene umgesetzt. Dies machte die Entwicklung langsam und fehleranfällig. Später kamen objektorientierte Wrapper für die Windows-*API*: die Microsoft Foundation Classes und *Object Window Library (OWL)*. Die Wrapper haben die Nutzung der *API* vereinfacht. Die Schwächen der *API* und die damit verbundenen Probleme blieben aber unverändert.

Das erste Problem ist die hohe Komplexität in der Oberflächenentwicklung [WW07]. Die Oberflächen werden immer komplexer. Die Implementierung mit alten Technologien wird zu einer schwierigen Aufgabe, die nur mit großem Aufwand erledigt werden kann.

Die Anforderungen für Benutzeroberflächen werden immer weiter verändert [WW07]. Die Verwendung von Standardsteuerelementen reicht für Benutzer nicht mehr aus. Die modernen Anwendungen sollen gut aussehen und einfach benutzbar sein. Um dieses Problem zu lösen, werden oft Designer in den Entwicklungsprozess einbezogen. Dadurch entsteht aber ein weiteres Problem.

Die Arbeitsaufteilung zwischen Designer und Entwickler ist schwierig umzusetzen. Denn gewöhnlich ist die Wissensschnittmenge des Designers und des Entwicklers eher klein.

4.2.1 WPF Architektur

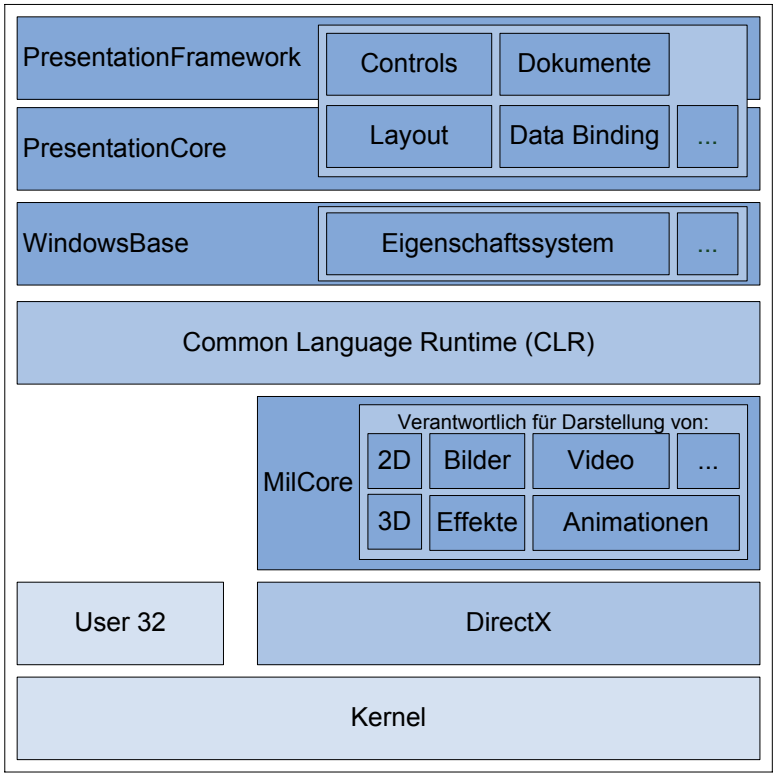


Abbildung 4.1: Die *WPF*-Architektur [Hub08]

Um die im Kapitel 4.2 beschriebenen Probleme zu lösen, wurde im Jahr 2007 die *WPF* entwickelt. Wie Abbildung 4.1 zeigt, baut *WPF* nicht auf der Windows-*API* auf, sondern auf *DirectX*. Für die Anwendung bedeutet das, dass mit *WPF* nicht nur 3D- sondern auch 2D-Oberflächen von der Grafikkartenleistung profitieren können. Des Weiteren können Transparenz-Effekte eingesetzt werden, denn *WPF*-Controls sind nicht nur einem bestimmten Bildschirmausschnitt zugeordnet und können daher auf anderen Controls gezeichnet werden. Außerdem übernimmt *WPF* die Zeichnen-Aufgabe, was die Programmierung deutlich einfacher macht. Dies geschieht durch eine Verwaltung der verschiedenen Bäume.

Die nativ implementierte MilCore verwaltet einen Composition Tree. Diese besteht aus einfachen grafischen Elementen, wie Linien oder Rechtecken. MilCore konvertiert die Veränderungen im Baum in *DirectX*-Befehle. Der Visual Tree, der vom PresentationCore verwaltet wird, besteht ebenfalls aus einfachen Elementen, allerdings sind diese Elemente *Common Language Runtime (CLR)*-Objekte. PresentationCore und MilCore halten beide Bäume synchron und können große Objekte, wie Bitmaps, sogar gemeinsam verwenden. Diese Architektur ist leistungsfähiger als ältere Windows

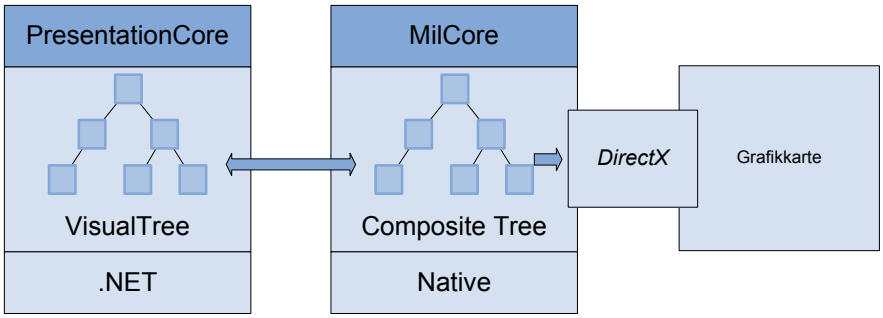


Abbildung 4.2: *WPF*-Bäume [Hub08]

Programmiermodelle, denn es müssen nicht alle Elemente neu gezeichnet werden, sondern nur die, die sich verändert haben.

4.2.2 WPF-Konzepte

Dieser Abschnitt dient dazu, einem Leser, der keine Kenntnisse von WPF besitzt, die nötigen Grundlagen von WPF zu vermittelt, die benötigt werden, um diese Arbeit zu verstehen.

Die Extensible Application Markup Language

Der Einsatz von *XAML* bringt weitere Möglichkeiten. *XAML* ist deklarativ und eignet sich daher gut zur Oberflächenbeschreibung. Durch Trennung von Design und Code wird der Einsatz von Code-Generatoren vereinfacht. Des Weiteren wird hierdurch die Arbeitsaufteilung zwischen Designer und Programmierer gefördert, da diese relativ unabhängig voneinander entwickeln können. Während sich der Programmierer mit der Anwendungslogik beschäftigt, kann der Designer mit Hilfe von speziellen Werkzeugen, wie z.B. Expression Blend, die Benutzeroberfläche entwickeln, oder eine bereits entwickelte Oberfläche bearbeiten.

XAML ist eine auf *XML*-basierende deklarative Sprache, die zur Erstellung von *WPF GUI* Komponenten verwendet wird. Listing 4.1 zeigt ein einfaches *XAML* Beispiel zur Deklaration eines `Windows`, das zwei `Buttons` enthält. Im nachfolgenden Abschnitt wird immer auf die Zeilennummern dieses Beispiels referenziert, um Beispiele zu geben. In *XAML* werden *XML-Tags* zur Erstellung eines Objektes einer Klasse verwendet. Über *XML-Namespaces* wird festgelegt, in welchen Assemblies diese Klassen zu finden sind. Damit es möglich ist, eine Klasse in *XAML* zu instanziieren, muss sie einen parameterlosen und öffentlichen Konstruktor besitzen. Alle öffentlichen *Eigenschaften* der Klasse können über *XML-Attribute* gesetzt werden. Zum Beispiel wird in Zeile acht die Hintergrundfarbe des Buttons auf Weiß gesetzt. Da die Werte in *XML* immer

`Strings` sind, muss eine implizite Methode vorhanden sein, die einen `String` zu dieser Klasse konvertieren kann. Um Attributen auch komplexere Werte zuweisen zu können, besitzt *XAML* die *erweiterte Notation* für XML-Attribute. Durch diese Notation kann ein Attribut auch in der Form eines neuen *XML-Tags* geschrieben werden, welches das Format „Klasse“.“Attribut“ hat und ein Kind-Element besitzen muss, welches dann der Wert des Attributes ist. So wird in den Zeilen 11 bis 16 für den zweiten Button ein Farbverlauf erstellt. Wie Kind-Elemente eines *XML-Tags* behandelt werden, hängt vom Typ des *Tags* ab. Bei Objekten, die eine *Content*-Eigenschaft besitzen, wird diese gesetzt (Vergleiche Zeile neun und Zeile 18). In diesem Fall darf es nur ein Kind-Element geben. Andere Elemente, wie zum Beispiel das `StackPanel` (Zeile sechs), können auch mehrere Kind-Elemente haben (die Buttons in Zeile sieben und Zeile neun).

```
1  <Window x:Class="BasicXAMLExample.Window1"
2   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4   xmlns:sys="clr-namespace:System;assembly=mscorlib"
5   Title="Window1" Height="300" Width="300">
6      <StackPanel>
7          <Button x:Name="button1" Background="White"
8           Content="Content setzen 1. Moeglichkeit"/>
9          <Button x:Name="button2" Click="button2_Click">
10             <Button.Background>
11                 <LinearGradientBrush>
12                     <GradientStop Color="Red" Offset="0"/>
13                     <GradientStop Color="Blue" Offset="1"/>
14                 </LinearGradientBrush>
15             </Button.Background>
16             <!--Content wird implizit gesetzt.-->
17         </Button>
18     </StackPanel>
19 </Window>
```

Listing 4.1: XAML Beispiel

Das `Window`-Element definiert mittels des `Class`-Attributs eine neue Klasse mit dem Namen `Window1` (Zeile eins). In der Regel sind sie in einer Datei im Format „Klassenname“.xaml gespeichert. Zu solch einer *XAML*-Datei gehört noch eine *Code-Behind*-Datei, die regulären C#-Code enthält und im Format „Klassenname“.xaml.cs gespeichert wird. In dieser Datei kann die Klasse um beliebige C#-Konstrukte ergänzt werden. Beide Dateien werden dann zusammen zu einer fertigen Klasse kompiliert. Listing 4.2 zeigt den C#-Quellcode zu dem *XAML*-Beispiel in Listing 4.1. Besonders häufig werden im Code-Behind Teil der Klasse `EventHandler` definiert, damit sie in *XAML* benutzt werden können. So wird der in Listing 4.2 definierte `EventHandler` Namens `button2_Click` in Zeile neun von Listing 4.1 benutzt.

```
1  namespace BasicXAMLExample
2  {
3      /// <summary>
4      /// Interaction logic for Window1.xaml
5      /// </summary>
6      public partial class Window1 : Window
7      {
8          public Window1()
9          {
10             InitializeComponent();
11         }
12
13         public void NeueFunktion()
14         {
15             Console.WriteLine("Ich bin eine Funktion der Klasse Window1");
16         }
17
18         private void button2_Click(object sender, RoutedEventArgs e)
19         {
20             Console.WriteLine("Button2 wurde geklickt");
21         }
22     }
23 }
```

Listing 4.2: Code-Behind des XAML Beispiels

Content Konzept

Die meisten Controls in *WPF* besitzen je eine öffentliche Eigenschaft *Content* vom Typ `object`. Diese beschreibt den grafischen Inhalt des Controls. Da diese Eigenschaft von Typ `object` ist, können beliebige Elemente als Inhalt von anderen Controls dienen [Per06]. Des Weiteren wird ein zusätzlicher Baum aufgebaut, der Elementen-Baum. Dieser hat als Wurzelelement ein Window und als Kind-Elemente beliebige *WPF*-Controls. Die *WPF*-Programmierung besteht im Grunde genommen nur aus dem Aufbau und der Veränderung des Element-Baums.

Routed-Events

Da ein beliebiges Element ein weiteres beliebiges Element aufnehmen kann, ist ein neues Event-Model notwendig. Wenn z.B. in einem Button ein `StackPanel` liegt und darin ein `Rectangle`, ist unklar, welches Element das MouseDown-Event empfangen soll. Um Unklarheiten zu beheben, sind Routed-Events entwickelt worden. Es kann in *WPF* eine von drei verschiedenen Routing-Strategien verwendet werden (vgl. [Hub08]):

- *Tunnel* - das Event wird von oben durch den Visual Tree in niedrigere Hierarchiestufen geroutet.
- *Bubble* - das Event von einem im Visual Tree tiefer liegenden Element nach oben gereicht.
- *Direct* - das Event wird nur auf dem geklickten visuellen Element gefeuert.

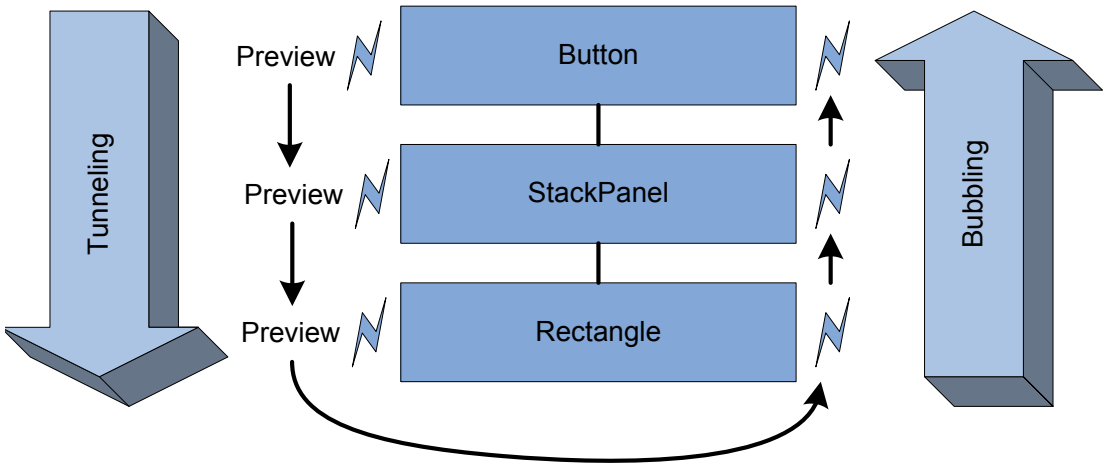


Abbildung 4.3: Routed-Events [Hub08]

Das Routing von Events wird durch eine spezielle Routed-Event-Engine verwaltet. Diese ermöglicht auch die Attached Events.

„Das Konzept eines angefügten Ereignisses ermöglicht es, einen Handler für ein bestimmtes Ereignis zu einem beliebigen Element hinzuzufügen, das nicht das Element sein muss, das dieses Ereignis tatsächlich definiert oder erbt.“[Micc]

DependencyProperties

`DependencyProperties` sind allgemein ausgedrückt .Net `Properties`, deren Wert in Abhängigkeit zu anderen Faktoren ermittelt wird. Diese anderen Faktoren können, neben dem klassischen Wert, den eine `Property` haben kann (sogenannte `local values`), z.B. von `Styles` oder `Animations` ermittelt werden. Des Weiteren verfügt eine `DependencyProperty` über die `ChangeNotifications`, also die Möglichkeit, darüber zu informieren, wenn sich der Wert verändert hat. `DependencyProperties` sind ein grundlegender Bestandteil des *WPF*-Frameworks. Insbesondere visuelle Eigenschaften (wie z.B. Hintergrundfarbe) sind meist als `DependencyProperties` realisiert. Dies erlaubt eine besonders leichte Integration in grafische Tools wie Expression Blend. Tatsächlich kann eine eigens erstellte `DependencyProperty` eines selbst geschriebenen Widgets sofort in Expression Blend verwendet werden. Damit eine Klasse `DependencyProperties` verwenden kann, muss sie von der Klasse `DependencyObject` ableiten. Die Definition einer eigenen DependencyProperty erfolgt als eine statische Klassenvariable, erzeugt durch eine Factory-Methode der `DependencyProperty` Klasse. Dies mag auf dem ersten Blick etwas unlogisch wirken (da die Werte einer Eigenschaft objekt-und nicht klassenweit gesetzt werden sollen), das eigentliche Zugreifen auf die Werte wird dann aber über die von `DependencyObject` bereitgestellten Methoden `GetValue(DependencyProperty)`

und `SetValue(DependencyProperty, object)` erledigt. Die `DependencyProperty` dient mehr als eine Art Typ für diese Funktionen. Um nun dem Programmierer weiterhin einen bequemen Zugriff auf die `Property` zu erlauben, wird ein sogenannter Property Wrapper definiert, der entsprechend wie eine normale .Net `Property` aussieht, aber die `Set-` und `GetValue` Methoden aufruft. Es muss beachtet werden , dass interne Prozesse vom .Net Framework oft die `Set-` und `GetValue` Methoden direkt aufrufen; in diesen Wrapper sollten also keine Nebeneffekte eingebaut werden. Abbildung 4.4 zeigt, welche Schritte bei der Wertermittlung einer `DependencyProperty` durchgeführt werden. Beim Schritt *coercion* wird, falls ein entsprechendes Callback definiert wurde, der Wert nochmals verändert, damit eventuelle Rahmenbedingungen erfüllt werden. Beispielweise könnte es einen Minimal- oder Maximalwert geben, auf den der Wert gesetzt wird, falls dieser über- bzw. unterschritten wird. Im letzten Schritt, der *validation*, wird noch einmal überprüft, ob der Wert nun gültig ist. Wenn ein ungültiger Wert vorliegt wird eine Exception geworfen. [Mic08a]

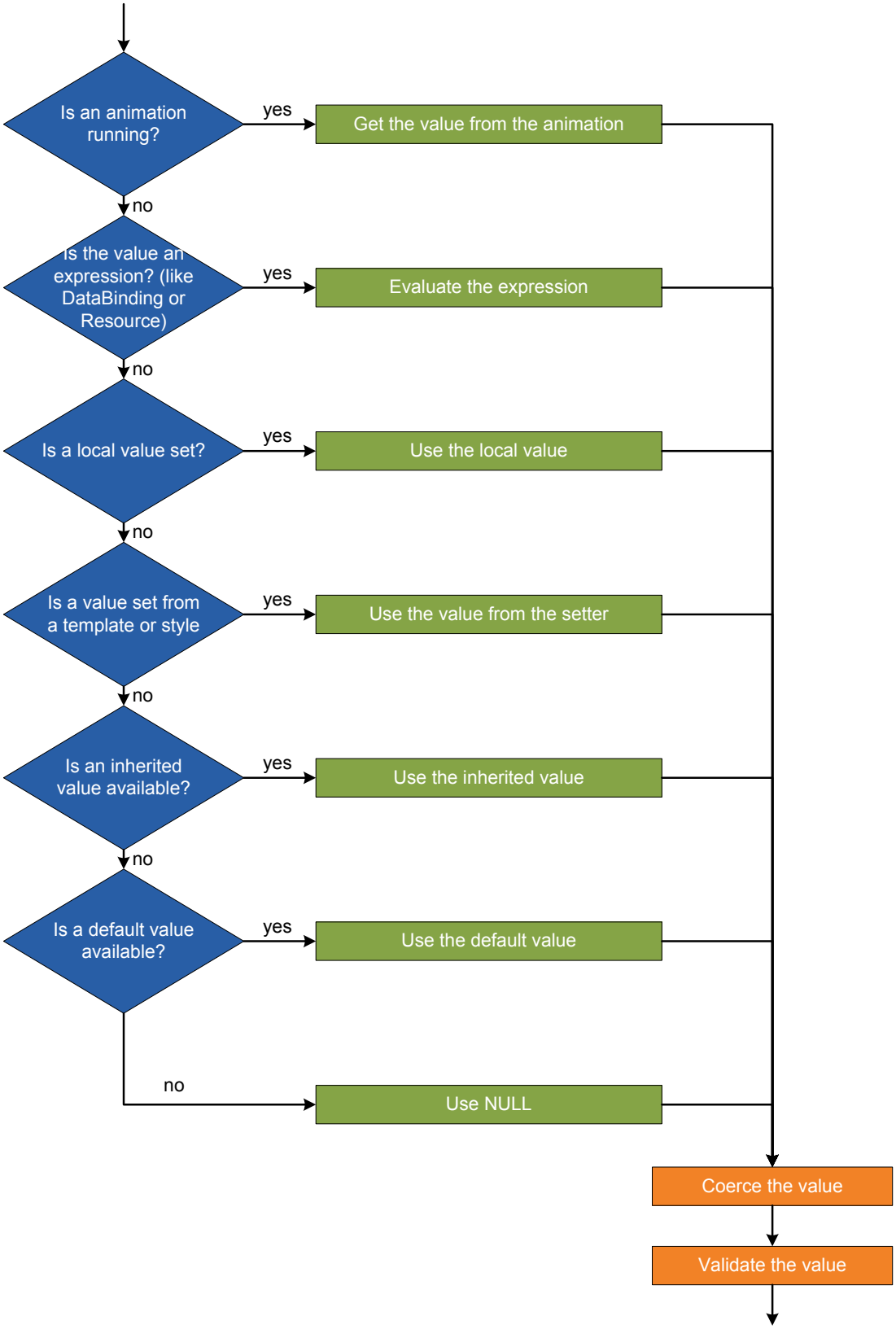


Abbildung 4.4: Die *Value Resolution Strategy* von `DependencyProperties` [Chr08]

Der in vorletzte Schritt der Abbildung 4.4 wird oft als *Visual Inheritance* oder *Property Inheritance* bezeichnet, um ihn von der klassischen Vererbung der Objektorientierung abzugrenzen. Hierbei wird ermittelt, ob der Wert vielleicht in einem der Vorfahren im VisualTree gesetzt wurde.

Genauso wie Attached Routed Events gibt es auch *Attached Dependency Properties*. Sie können auf Elemente gesetzt werden, die diese Eigenschaften nicht definieren. So kann eine `Canvas.Left`Eigenschaft zu einem beliebigen Control zugewiesen werden, obwohl Control diese Eigenschaft nicht definiert.

Styles

Ein Style definiert eine Reihe von Settern für Eigenschaften, damit sie einheitlich auf mehrere Widgets angewendet werden können. Ein Style kann einen `key` haben, mit dem man ihn referenzieren kann, und einen `TargetType`, der festlegt, auf welche Elemente er angewendet werden kann. Auch kann ein Style einen anderen Style erweitern (mit dem Attribut `BasedOn`). Des Weiteren können in Styles auch Trigger definiert werden (siehe Abschnitt Trigger).

Listing 4.3 zeigt ein einfaches Beispiel für einen Style.

```
1 <Style x:Key="Style1" TargetType="{x:Type Button}">
2     <Setter Property="Background" Value="#FF140C56"/>
3     <Setter Property="Foreground" Value="#FFFFFFFF"/>
4 </Style>
```

Listing 4.3: Beispiel Styles

Templates

Während ein Style nur Eigenschaften eines Elements setzt, kann ein Template den kompletten VisualTree eines Elements austauschen und damit ein völlig anderes Aussehen des Elements erreichen. Templates sind die wesentliche Technik, um die Eigenschaften und das Verhalten eines Widgets von seinem Aussehen zu trennen. Es gibt drei verschiedene Arten von Templates: `ControlTemplates`, mit denen sich das Aussehen eines Widgets anpassen lässt; `DataTemplates`, die es im Wesentlichen erlauben, für beliebige (Daten) Objekte eine visuelle Repräsentation zu erzeugen und `ItemsControlTemplates`, mit denen `ItemsPanels` (z.B. ListBoxes) anpasst werden können.

Trigger

Trigger definieren, ähnlich wie die Styles, eine Reihe von Settern für Eigenschaften. Diese werden dann angewandt, wenn ein spezieller Zustand ausgelöst wurde. Bei `PropertyTriggern` und `DataTriggern` ist dies, wenn zugehörige Eigenschaften einen bestimmten Wert angenommen haben, oder bei `EventTriggern`, falls ein

bestimmtes Ereignis eintritt. `DataTrigger` und `PropertyTrigger` sind immer Teile eines Templates. `EventTrigger` müssen nicht Teil eines Styles oder Templates sein, dies ist aber die gängige Praxis [LL07].

Resources

Alle von `FrameworkElement` abgeleiteten Klassen haben die `Resources` Eigenschaft, in der nicht visuelle Elemente wie z.B. Styles und Templates definiert werden können. Dieses Element und all seine visuellen Kind-Elemente können auf die Ressourcen zugreifen.

Databinding

`Databinding` ist eine Technik, die es erlaubt, `DependencyProperties` an Datenquellen zu binden, um ihren Wert daraus zu ermitteln. Dies ist eine wichtige Technik, um die Darstellung von den Daten und der Logik zu trennen. Damit ist es möglich, *GUI* Elemente für spezielle Daten in *XAML* zu definieren, ohne dabei auf klassische Programmierung zurückgreifen zu müssen. Als Datenquellen können dabei im Prinzip beliebige *CLR*-Objekte benutzt werden. Allerdings müssen eine Reihe von Interfaces implementiert werden, wenn das Binding in beide Richtungen funktionieren oder das Ziel bei Veränderungen benachrichtigt werden soll. Insbesondere `DependencyProperties` (oft von z. B. anderen Elementen in der *GUI*) eignen sich gut, da sie alle Implementierungen dafür schon bereitstellen. Auch *XML*-Dokumente sind als Quelle beliebt. Diese beiden Typen können in *XAML* direkt eingebunden werden. Die Klasse `FrameworkElement` definiert die Eigenschaft `DataContext`, die im Normalfall das *CLR*-Objekt enthält, das für das Binding benutzt werden soll.

Behavior, Trigger und Action Konzept

Die Konzepte der Behaviors, Trigger und Actions wurden erstmalig in Silverlight 3 eingeführt und danach auch zu *WPF* portiert. Im Grunde genommen handelt es sich bei den Konzepten um *WPF* Erweiterungen. Um diese Erweiterungen nutzen zu können, muss man die *DLL* Microsoft.Expression.Interactivity.dll in sein Projekt einbinden. Nach Einbinden dieser *DLL* bekommt jedes `UIElement` die *Attached Dependency Properties* `Interaction.Triggers` und `Interaction.Behavior`.

Eine Action ist eine Klasse, die von der generischen Klasse `TargetedTriggerAction<T>` abgeleitet wird, wobei das `T` den Typ angibt, auf welche Klasse die Action angewendet werden kann. Eine Action ist im Prinzip eine Abstraktion einer Handlung und enthält eine Methode `Invoke`, welche diese Handlung beschreibt.

Ein Trigger ist eine Klasse, die von der generischen Klasse `TriggerBase<T>` abgeleitet wird, wobei das `T` den Typ angibt, auf welche Klasse der Trigger angewendet werden kann (z. B. Button). Jeder Trigger hat eine Property `AssociatedObject`, in welcher

das assoziierte Objekt gespeichert wird. Zusätzlich dazu können einem Trigger mehrere Actions zugeordnet werden. Dadurch kann der Trigger die Veränderungen vom assoziierten Objekt beobachten. Sobald ein definierter Zustand von dem assoziierten Objekt erreicht wird, werden die `Invoke`-Methoden der Actions ausgeführt.

Behavior bilden eine Kombination aus Triggern und Actions. Sie beobachten ein Element und führen gleichzeitig eine Action aus, sobald ein bestimmter Zustand erreicht worden ist. Dazu muss auch hier das assoziierte Objekt angegeben werden, durch die `AssociatedObject`-Property.

4.2.3 Das Model-View-ViewModel Muster

Dieser Abschnitt basiert auf dem Artikel *Das Model-View ViewModel-Pattern für WPF-Anwendungen* von Thomas Huber und Christoph Pletz [HP07].

Das *Model-View-ViewModel (MVVM)* Softwaremuster ist eine abgewandelte Variante des *Model View Controller (MVC)* Musters, das speziell auf *WPF* angepasst wurde. Beim *MVC* Muster wird eine Client-Software in drei Komponenten geteilt: Das *Model*, welches die Daten bereitstellt, der *View*, der die Daten dem Benutzer anzeigt, und der *Controller*, der die Verarbeitung der Nutzerinteraktion und des Programmablaufs übernimmt. Abbildung 4.5 zeigt die Beziehungen der drei Komponenten zueinander. Dabei besitzt das Model keinerlei Wissen darüber, wie und ob es angezeigt wird – es muss nur melden, wenn es sich verändert hat. Die Trennung von Controller und View ist in der Praxis jedoch oft sehr schwer. Oft müssen die Daten noch für die Ausgabe formatiert werden oder bei der Eingabe neuer Daten müssen diese erst verifiziert werden.

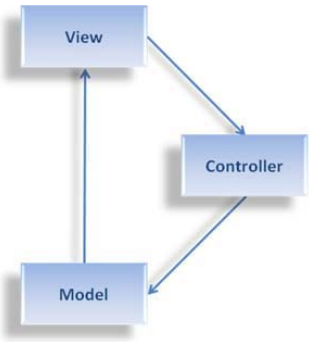


Abbildung 4.5: Abhängigkeitsbeziehungen im *MVC* Muster [Kle07].

Bei dem *MVVM* Muster bleibt die Rolle des Models und des Views im Prinzip gleich. Dabei greift der View nicht direkt auf das Model, sondern bekommt diese Daten – eventuell aufbereitet – vom ViewModel. Eine *WPF* Besonderheit ist, dass der View möglichst komplett in *XAML*, also rein deklarativ, erstellt wird. Das ViewModel ist eine

abstrakte Repräsentation des Views. Es speichert den Zustand und das Verhalten des Views aber nicht das Aussehen. So speichert das ViewModel beispielsweise, welche Elemente in einer Liste ausgewählt worden sind, diese aber anders farblich zu markieren ist Aufgabe des Views. Abbildung 4.6 zeigt die Architektur des *MVVM* Musters.

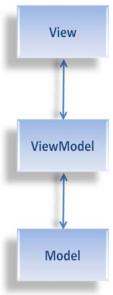


Abbildung 4.6: Abhängigkeitsbeziehungen im *MVVM* Muster (vgl. [Kle07])

Die wichtigen Techniken dafür sind *Databinding*, *DataTemplates* und *DataTrigger*. Um Verhalten umzusetzen, werden Commands benutzt. Diese vom ViewModel erstellten Commands können dann im View an entsprechende Elemente gebunden werden. Das Ziel des *MVVM* Musters ist es, eine stärkere Abstraktion von Aussehen und Verhalten des Views zu erhalten. Außerdem lassen sich ViewModels leichter testen, da sie keine graphische Oberfläche haben.

4.3 Multitouch-Frameworks

Im Bereich von Multitouch-Tischen und Multitouch-Programmierung ist in den verschiedensten Forschungsprojekten bereits sehr viel geforscht und entwickelt worden. Gerade auch für die Gestenerkennung stehen einige Frameworks zur Verfügung. Es stellt sich die Frage, ob nicht eines dieser Frameworks direkt verwendet werden kann oder ob ein eigenes Framework zur Gestenerkennung und zur Einbindung von Multitouch in *WPF* entwickelt werden muss. Um festzustellen, ob eines der bereits existierenden Frameworks verwendet werden kann, ist ein Vergleich dieser durchgeführt worden.

Für die Untersuchung wurden die folgenden Multitouch-Frameworks ausgewählt: *WPF-Multitouch*, *Multitouch Vista* und *Multitouch Core*. In den folgenden Kapiteln werden sie auf die Kriterien wie Architektur, Lizenz, Integration in Visual Studio und *Microsoft Expression Blend*, Code-Konvention Einhaltung, Erweiterbarkeit, Anzahl an Features und Multitouch-Simulationsmöglichkeit hin untersucht und bewertet. *WPF-Multitouch* und Multitouch-Core sind *Closed Source* Projekte. Daher können die Annahmen über die Architektur von der tatsächlichen Architektur abweichen. Des Weiteren sind die

Lizenzen für *WPF-Multitouch* und *Multitouch Core* entweder unbekannt oder wurden von Autoren widersprüchlich angegeben.

4.3.1 WPF-Multitouch

Die Autoren von *WPF-Multitouch*[1] geben als Lizenz *GNU is Not Unix (GNU) General Public License (GPL)* v3 an. Allerdings ist der Source Code nicht verfügbar, was der Lizenz nicht entspricht. Die Architektur des Frameworks ist in Abbildung 4.7 abgebildet.

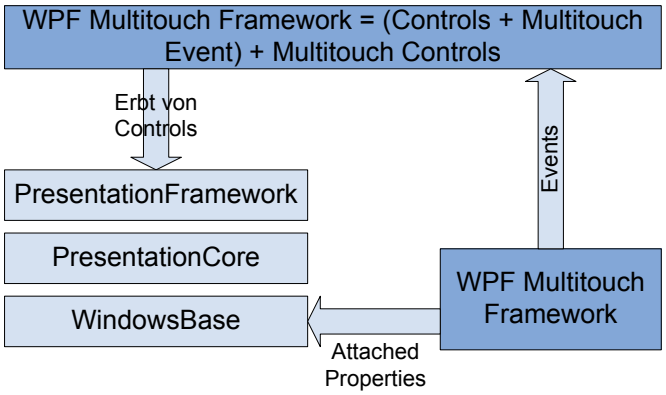


Abbildung 4.7: WPF-Multitouch Architektur

Das *WPF-Multitouch* Framework kann in zwei Module unterteilt werden. Das erste, rechts unten auf Abbildung 4.7 dargestellt, initialisiert das Framework und verarbeitet empfangene *TUIO*-Nachrichten. Außerdem benutzt dieses Modul *Attached Dependency Properties*, um die Funktionalität von bestehenden *WPF*-Controls zu erweitern. Damit kann ein `Rectangle` mit einem Finger bewegbar gemacht werden, indem folgende Zeilen Code hinzugefügt werden:

```
1 Rectangle rect = new Rectangle();
2 MTProperties.SetTouchEnabled(rect, true);
3 MTProperties.SetCanBeDragged(rect, true);
```

Listing 4.4: Rectangle bewegbar machen

Oder in *XAML*:

```
1 <Rectangle mt:MTProperties.TouchEnabled="True"
2  mt:MTProperties.CanBeDragged="True" />
```

Listing 4.5: Rectangle bewegbar machen (in XAML)

[1] `http://code.google.com/p/multitouchframework/` letzter Zugriff 10.12.2008

Des Weiteren bietet dieses Modul auch einige Events (s. Abbildung 4.8). Darunter ist das Event `GestureRecognised`, welches über eine erfolgreiche Gestenerkennung informiert. Insgesamt werden sieben verschiedenen Gesten erkannt.

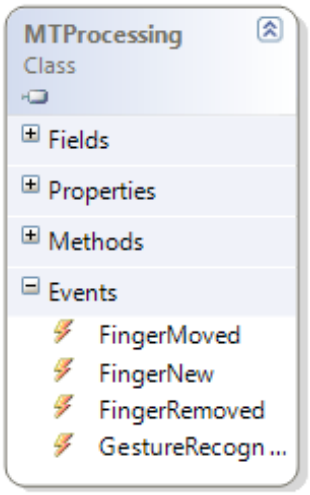


Abbildung 4.8: WPF-Multitouch Events

Die Nutzung dieser Events ist aber mit zusätzlicher Arbeit verbunden, denn die grafische Komponente muss selber berechnen, ob die Events für sie bestimmt sind.

Das zweite Modul (Abbildung 4.7, links oben) bildet sich aus verschiedenen Controls, die von *WPF*-Controls erben `MTButton` beispielsweise erbt von `Button`, `MTSlider` von `Slider` usw. Diese Controls können in einer Multitouch-Anwendung einfach eingesetzt werden.

Die Integration in Visual Studio und in *Microsoft Expression Blend* ist problemlos. Die Eigenschaften der Controls können in beiden Anwendungen angepasst werden.

Mit *WPF-Multitouch* lassen sich einfache Multitouch-Anwendungen sehr leicht schreiben. Sobald aber eine Funktionalität abweichend von klassischer Touch-Drag-Resize-Rotate-Funktionalität gewünscht ist, wird es schnell kompliziert. *WPF-Multitouch* bietet nur eine Dualtouch-Simulation auf der *WPF*-Ebene. Sie ist einfach zu nutzen, allerdings lassen sich damit komplexe Gesten nicht nachbilden.

4.3.2 Multitouch Vista

Multitouch Vista[2] ist *Open Source* und ist unter *GNU GPL* v2 frei verfügbar. Die Architektur des Frameworks ist in Abbildung 4.9 abgebildet.

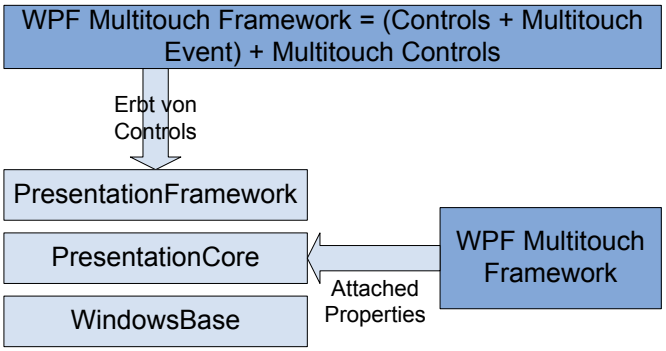


Abbildung 4.9: *Multitouch Vista* Architektur

Die Architekturen von *WPF-Multitouch* und *Multitouch Vista* sind sehr ähnlich. Allerdings verwendet *Multitouch Vista Attached Dependency Property*s anstelle von *Attached Dependency Properties*. Somit können die grafischen Komponenten selber entscheiden, wie sie auf Events reagieren. Sie erhalten nur die Informationen über die Events, die für sie bestimmt sind. Es werden die Routing-Strategien Bubble und Tunnel unterstützt. Die Events haben eine ähnliche Funktionalität wie die in *WPF-Multitouch* (vgl. Abbildung 4.8). Sie sind mit der Attached Events Syntax definiert und in Abbildung 4.10 dargestellt.

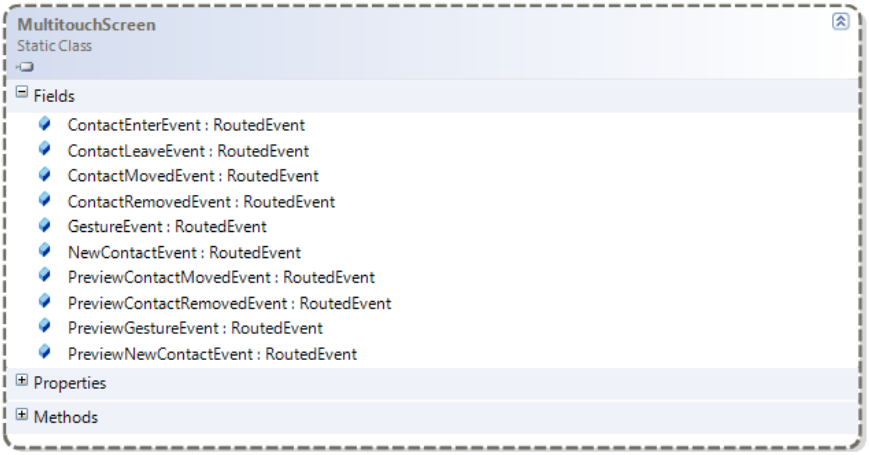


Abbildung 4.10: *Multitouch Vista* Events

[2] `http://www.codeplex.com/MultiTouchVista`, letzter Zugriff 10.12.2008

Die Anbindung an Events erfolgt mit der Attached Event Syntax:

```
MultitouchScreen.ContactMoved(rect, rect_ContactMoved);
```

Listing 4.6: Bindung an Events

Oder in *XAML*:

```
<Rectangle  x:Name="rect"
 mt:MultitouchScreen.ContactMoved="rect_ContactMoved"/>
```

Listing 4.7: Bindung an Events (in XAML)

Es werden über 30 Gesten erkannt. Die Gesten sind aber entweder Single- oder Dualtouch-Gesten.

Die Integration in *Visual Studio* und in *Microsoft Expression Blend* ist auch mit *Multitouch Vista* problemlos.

Multitouch Vista bietet eine Simulation auf Betriebssystem-Ebene durch mehrere Mäuse, dadurch lassen sich allerdings in der Praxis nur Dualtouch-Gesten simulieren. Da *Multitouch Vista* einen deutlich größeren Funktionumfang hat als *WPF-Multitouch*, ist es möglich, mit *Multitouch Vista* mehr zu erreichen. Durch Offenheit des Codes sind Weiterentwicklung und Anpassung denkbar. Allerdings ist *Multitouch Vista* nicht dokumentiert. Deswegen ist die Anpassung bzw. Weiterentwicklung mit langen Einarbeitungsphasen verbunden.

4.3.3 Multitouch Core

Multitouch Core[3] wurde von Roman Kalantari entwickelt. Eine Anfrage über die Lizenz beantwortete er folgendermaßen: „I did not release the dlls under any license you can use them as you want.“ Die Architektur des Frameworks ist in Abbildung 4.11 dargestellt.

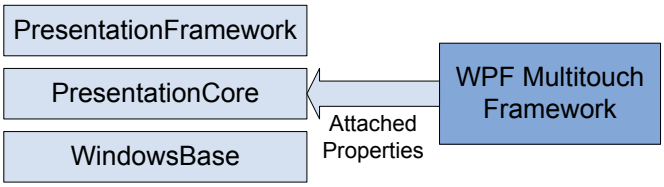


Abbildung 4.11: *Multitouch Core* Architektur

Die Architektur von *Multitouch Core* unterscheidet sich von *WPF-Multitouch* und *Multitouch Vista*. Hier werden keine speziellen Multitouch Controls entwickelt, es

[3] http://nuigroup.com/forums/viewthread/1616/, letzter Zugriff 14.12.2008

werden, wie in Multitouch Vista, nur die Attached Events zur Verfügung gestellt. Diese sind in Abbildung 4.12 abgebildet. *Multitouch Core* unterstützt nur die Routing-Strategie Tunnel. Außerdem ist *Multitouch Core* das einzige Framework, das *Drag and Drop*-Unterstützung bietet.

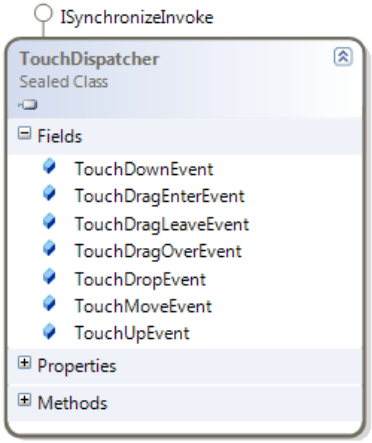


Abbildung 4.12: *Multitouch Core* Events

Durch den Verzicht auf Multitouch Controls ist keine Integration in *Visual Studio* notwendig. Die Simulation findet auf *WPF*-Ebene statt und ist genauso wie in *WPF-Multitouch* implementiert. Sie unterstützt aber nur Singletouch und ist nicht fehlerfrei. Da *Multitouch Core* keine Gestenerkennung unterstützt, ist der Einsatz nur beschränkt möglich.

4.3.4 Ergebnisse

Die untersuchten Frameworks befinden sich noch im frühen Entwicklungsstadium. Daher ist eine Erweiterungs- bzw. Weiterentwicklungsmöglichkeit gewünscht. Der Source Code ist aber nur von *Multitouch Vista* verfügbar. Die Erweiterung von *Multitouch Vista* ist allerdings mit weiteren Problemen verbunden, denn es liegt kaum Dokumentation vor und der Source Code ist eher *zusammengewürfelt* als überlegt entwickelt worden. Daher eignen sich die untersuchten Frameworks für die Entwicklung einer konkurrenzfähigen Multitouch-Anwendung nicht. Die Entwicklung eines eigenen Frameworks mit *WPF* ist relativ einfach. Voraussetzung dafür sind tiefe *WPF*-Kenntnisse. Hauptschwierigkeit ist dabei die Gestenerkennung.

4.4 Zusammenfassung

Zu Beginn dieses Kapitels wurden verschiedene Toolkits, die für die Programmierung von Benutzungsoberflächen geeignet sind, vorgestellt und auf ihre Tauglichkeit für die Projektgruppe hin untersucht. Betrachtet wurden unter anderem *OpenGL*, *DirectX*, *WPF*, Qt und Adobe Flash. Die Kriterien zur Bewertung dieser Toolkits beziehen unter anderem ein, ob bereits Multitouch-Unterstützung möglich oder vorhanden ist, und wie die Performanz des Toolkits zu bewerten ist. Aus der abschließenden Bewertung ging hervor, dass die Windows Presentation Foundation den am besten geeigneten Kandidaten für die Projektgruppe darstellt. Der zweite Abschnitt dieses Kapitels lieferte daher eine Zusammenstellung der Grundlagen für den Umgang mit *WPF*.

Nach der Festlegung des Toolkits, das für die GUI-Programmierung eingesetzt wurde, musste zudem entschieden werden auf welchem Wege die Anbindung von Multitouch-Gesten an die GUI gestaltet werden sollte. Im dritten Abschnitt des Kapitels wurden daher die Multitouch-Frameworks *WPF-Multitouch*, *Multitouch Vista* und *Multitouch Core* vorgestellt und einander gegenüber gestellt. Allerdings genügte keines dieser Frameworks den angelegten Maßstäben, was vor allem darauf zurückzuführen war, dass sie sich zum Zeitpunkt des Vergleichs alle noch im Entwicklungsstadium befanden. Zudem war es nachteilig, dass der Quellcode der Frameworks nur in wenigen Fällen zur Verfügung gestellt wurde. Da davon auszugehen war, dass durch die Projektgruppe zumindest kleinere Anpassungen am eingesetzten Framework möglich sein sollten, wurde beschlossen ein eigenes Multitouch-Framework zu enwickeln. Dieses kann dann an die individuellen Bedürfnisse Projektgruppe angepasst werden.

Kapitel 5

Usability

Die Benutzungsschnittstelle ist die wesentliche Kommunikation zwischen dem Produkt und dem Benutzer. Sie prägt daher wesentlich den Eindruck, den der Benutzer von dem Produkt erhält. So haben z. B. Studien gezeigt, dass bei gleichem Funktionsumfang Benutzer eines Bankautomaten den mit der optisch ansprechenderen Oberfläche bevorzugen. Usability (Deutsch: Benutzbarkeit, Bedienbarkeit, Gebrauchstauglichkeit, Ergonomie) bezeichnet nach der [ISO] „*das Ausmaß, in dem es [das Produkt] von einem bestimmten Benutzer verwendet werden kann, um bestimmte Ziele in einem bestimmten Kontext effektiv, effizient und zufrieden stellend zu erreichen*“. In der Literatur wird immer häufiger der Begriff *joy of use* verwendet, der als Steigerung zur Zufriedenstellung zu sehen ist und Freude an der Benutzung als Ziel setzt. Daraus abgeleitet ergibt sich der wissenschaftliche Themenbereich des Usability Engineerings, der sich mit der systematischen Planung, Entwicklung und Evaluation von Benutzungsschnittstellen befasst. Als Grundlagen dienen hier Fakten und Standards statt der sonst oft eher intuitiven, unsystematischen Gestaltung von Benutzungsoberflächen. Die Ziele des Usability Engineerings sind dabei zufriedene Benutzer, höhere Produktivität und die Reduzierung von Kosten durch frühzeitige Erkennung von Fehlern.

Im Folgenden wird als Grundlage für Usability auf den menschlichen Körper eingegangen und nur auf Aspekte, die für unser Projekt relevant sind. Danach folgen Abschnitte über Prototyping und Evaluationen, die die wesentlichen Werkzeuge des Usability Engineering bilden. Im Anschluss werden kurz Hilfsfunktionen für den Benutzer erläutert und Standards, welche zu einer guten Usability führen sollen, erklärt.

5.1 Der menschliche Körper

In diesem Abschnitt werden die Grundlagen des menschlichen Körpers vorgestellt, um einen Überblick zu erhalten, welche Interaktionsmöglichkeiten prinzipiell möglich sind. Besonders beim Entwurf von Gesten ist es wichtig zu wissen, welche Gesten als besonders angenehm empfunden werden, bzw. welche zu vermeiden ist.

5.1.1 Sehvermögen

Das menschliche Auge ermöglicht eine wesentlich bessere Helligkeits- als Farbwahrnehmung. Dadurch entstehen einige Farbkombinationen, die schlecht zu erkennen sind, wie z.B. Gelb auf Weiß oder Grün auf Rot. Neben den physischen Eigenschaften des Auges wird unsere Wahrnehmung sehr stark durch die Interpretation bestimmt. Dadurch kann es zu optischen Täuschungen oder auch zur Gruppierung von Elementen kommen. (vgl. [Bol09])

5.1.2 Bewegung

Der menschliche Körper hat ein mit Gelenken versehenes Skelett, das durch Muskeln bewegt werden kann. Eine Bewegung wird von den Agonist-Muskeln ausgeführt und von den Antagonist-Muskeln wieder in den initialen (entspannten) Zustand gebracht. Man unterscheidet folgende Bewegungsarten:

- Beugen und Strecken: Den Winkel zwischen zwei zusammenhängenden Körperteilen verändern.
- Drehung: Einen Körperteil an seiner Achse drehen.
- Abspreizen und Anziehen: Einen Körperteil von der Mittellinie des Körpers — bei Fingern der Hand, bei Zehen entsprechend des Fußes — entfernen bzw. ranführen.
- Interne und Externe Rotation: Drehung einer Gliedmaße in bzw. gegen Richtung der Mittellinie.
- Heben und Senken: Auf- und Abwärtsbewegungen.
- Protraktion und Retraktion: Vorwärts- bzw. Rückwärtsbewegung des Arms.

Speziell bei Touchscreens sollte man beachten, dass Fingernägel hinderlich sein können, da einige Touchscreens nicht auf Fingernägel reagieren. Außerdem sind Finger meist leicht fettig, wodurch sich schnell Schlieren auf dem Display bilden. Helle oder gemusterte Hintergründe verringern diesen Effekt etwas, während Schwarz ihn hervorhebt. Einer der wichtigsten Aspekte ist jedoch die Verdeckung eines Teils des Bildschirms durch die Hand selbst, der bei Maussteuerung sichtbar wäre. [Saf08]

Für einen Menschen ist es im Allgemeinen schwieriger und anstrengender etwas anzutippen, je kleiner und je weiter es weg ist. Der Psychologe Paul Fitts hat dazu eine Formel aufgestellt, das sogenannte Gesetz von Fitts, welche berechnet, wie lange man für die Bewegung, um auf etwas zu zeigen, braucht. Diese Formel lautet:

$$MT = \alpha + \beta \log_2 \left(\frac{D}{W} + 1 \right) \tag{5.1}$$

Erklärung:

MT: Die Zeit, die für die Bewegung benötigt wird.

α: Eine empirisch ermittelte Konstante, die die Verzögerung angibt, bis das Gerät beginnt bzw. stoppt.

β: Eine empirisch ermittelte Konstante, die Beschleunigung bzw. Geräte-spezifische Veränderung erfasst.

D: Die Entfernung vom Start zum Ziel.

W: Die Breite des Ziels in Bewegungsrichtung.

Der Term

$$\log_2 \left(\frac{D}{W} + 1 \right) \tag{5.2}$$

wird dabei oft als Schwierigkeitsindex bezeichnet. Er gibt an, wie schwierig es ist, das Ziel von einer gegebenen Entfernung aus zu treffen [Zha09]. Bei dem Design der Oberfläche und der Gesten sollte auch darauf geachtet werden, wie die Ruheposition des Benutzers ist. Besonders die häufig gebrauchten Elemente sollten so nah wie möglich oder sogar in der Ruheposition genutzt werden können. Im Allgemeinen ist zu beachten, dass neutrale und entspannte Positionen als angenehm empfunden werden. Lässt sich ein häufiger Positionswechsel nicht vermeiden, sollte darauf geachtet werden, dass auch Aktionen vorgesehen sind, die die Gliedmaßen wieder in entspannende Positionen bringen.

Bei größeren Geräten sollten folgende Positionen und Bewegungen vermieden werden [Saf08]:

Extrempositionen Extrempositionen, die die Gliedmaßen spannen, sind sehr unangenehm und sollten auf jeden Fall vermieden werden.

Wiederholungen Die Oberfläche sollte so ausgelegt sein, dass der Benutzer nicht immer wieder die gleiche Bewegung ausführen muss.

Statische Position Keine Position sollte darauf ausgelegt sein, über einen langen Zeitraum gehalten werden zu müssen. Gerade bei Multitouch-Touchscreens muss darauf geachtet werden, dass nicht eine Hand eine Aufgabe, wie z. B.das dauerhafte Drücken, bekommt.

Gelenkbelastung Positionen, die Belastung auf die Gelenke ausüben, sind in jedem Fall zu vermeiden.

Bei der Gestaltung von Gesten sollte das gesamte Spektrum der Zielgruppe beachtet werden. Insbesondere bezüglich der Körpergröße sollte dabei in Betracht gezogen werden, dass in Deutschland das Spektrum ohne Ausreißer zwischen 1.50m und 1.90m liegt [sta09]. Dies gilt auch für Geräte in Handheld-Größe, bei denen die Gesten schnell für eine zu große Hand ungeeignet sind. Auch sollte versucht werden, zu komplexe Gesten zu vermeiden, da sonst Gruppen mit motorischen Einschränkungen ausgeschlossen werden. Touchscreens bieten (von sehr wenigen Ausnahmen abgesehen) keinerlei haptische Rückmeldung, womit der Benutzer komplett auf eine visuelle Rückmeldung angewiesen ist. Dies ist problematisch, da schlechte Sehkraft ein verbreitetes Phänomen ist. Ca. 7% aller Menschen sind Linkshänder, sodass die Oberfläche auch gespiegelt bedient werden können sollte [Saf08].

5.2 Prototyping

Ein Prototyp ist ein Modell des Endproduktes, das das Testen bestimmter Attribute erlaubt. Es zeigt, ob erstellte Designspezifikationen so umgesetzt wurden, wie sie geplant wurden. Diese Prototypen erlauben erstmals, Rückmeldung von Benutzern zu erhalten, und so frühzeitig Fehler im Interaktionsdesign zu finden. Außerdem können sie zum Überzeugen von Auftraggebern oder Vorgesetzen benutzt werden.

Ein Prototyp kann von einfachen Papiermodellen (Low Fidelity) bis hin zu fast fertigen Produkten (High Fidelity) reichen. Je detaillierter der Prototyp ist, desto detailliertere Rückmeldung erlaubt er. Je einfacher der Prototypen umsetzt wird, desto eher kann er eingesetzt werden und frühzeitig zur Fehlererkennung dienen. Außerdem hat sich gezeigt, dass bei einem Prototyp, der schon wie ein fertiges Produkt wirkt, eher selten grundlegende Bedenken geäußert werden. Ein Prototyp kann darauf ausgelegt sein, die Gesamtheit eines Produkts, was auch horizontaler Prototyp genannt wird, zu testen, wie z.B. konsistenter Gesamteindruck einer Website. Der vertikale Prototyp ist für das Testen von speziellen Features, wie z.B. hierarchische Auswahlmenüs für Touchscreens, geeignet.

Eine Methode, wie Funktionen des Prototypen, die schwer umsetzbar sind, simuliert werden können, ist die sogenannte *Wizard of Oz*-Technik. Hierbei simuliert ein Mensch Teile des Prototypen, indem er entsprechende Aufgaben übernimmt. Zum Beispiel kann

Spracherkennung sehr leicht durch einen Menschen simuliert werden, ist aber sehr aufwendig in der technischen Umsetzung [Bol09].

5.3 Evaluation

Als wesentliches Werkzeug zur Überprüfung der Tauglichkeit der entwickelten Benutzungsschnittstellen dient die Evaluation. Sarodnick und Brau definieren eine Evaluation folgendermaßen:

„Evaluation allgemein bezeichnet eine systematische und möglichst objektive Bewertung eines geplanten, laufenden oder abgeschlossenen Projekts. Ziel ist es, spezifische Fragestellungen zu beantworten und/oder den Zielerreichungsgrad eines bestimmten Vorhabens zu erheben.“[SB06]

Es werden die empirischen und die analytischen Evaluationen unterschieden. Heuristische Methoden werden von Usability-Experten durchgeführt, die aufgrund ihres Wissens und ihrer Erfahrung eine Bewertung abgeben. Empirische Methoden analysieren das Verhalten von tatsächlichen Benutzern. Bei diesen experimentellen Evaluationen müssen die Benutzer Aufgaben an einem System durchführen und werden dabei beobachtet. Die Daten können neben der einfachen Beobachtung auch aus Äußerungen der Nutzer während des Tests, Interviews nach dem Test oder Messungen, z.B. benötigte Zeit für die Aufgabe, gewonnen werden. Dabei sollte ein Experiment immer folgende Kriterien beachten:

Objektivität Werden die Ergebnisse vom Versuchsleiter beeinflusst?

Reliabilität Sind die Ergebnisse reproduzierbar?

Validität Wurde das Richtige gemessen bzw. gefragt?

5.4 Hilfe und Dokumentation

In der Psychologie gibt es den Begriff der *Affordance*, auf Deutsch *Angebotscharakter*, der beschreibt, wie gut man einem Alltagsgegenstand ansieht wie er zu benutzen ist. Don Norman wandte 1988 *Affordance* auch auf Benutzungsoberflächen an. Dieser Begriff wird auch im Interaktionsdesign verwendet. Der Angebotscharakter wird dabei von Faktoren wie physikalischer Beschaffenheit, visueller Erscheinen, logischen Aspekten sowie kulturellen Erfahrungswerten bestimmt. Norman argumentiert, dass virtuelle Objekte keine wirkliche Affordance haben. Es handelt sich also mehr um eine wahrgenommene oder virtuelle Affordance, die beschreibt, wie intuitiv ein *GUI*-Element

mit der Aktion, die es ausführt, verknüpft wird (vgl. [Bol09]). Zum Beispiel ist es oft üblich, dass ein Element, welches anklickbar ist, sich visuell leicht verändert, wenn die Maus über ihm verharrt. Diese Affordance-Eigenschaft kann auf Touchscreens nicht benutzt werden, weshalb Affordance auf einem Touchscreen zur Herausforderung wird. Es bietet sich an, Metaphern zu verwenden, also virtuelle Gegenstände wie reale Objekte zu designen, die eine ähnliche Funktion haben (vgl. [Wal09]).

Eine Möglichkeit, den Benutzer auf die Existenz von Gesten aufmerksam zu machen, ist mittels sogenannter *intelligenter* Hilfen. Diese Hilfen bieten, in Abhängigkeit vom Status und vom Kontext, dem Benutzer Hilfestellungen an. Der Webbrowser Opera macht den Benutzer beim ersten — vielleicht unabsichtlichen — Ausführen einer Mausgeste darauf aufmerksam, dass er gerade eine Mausgeste ausgeführt hat, fragt, ob er diese Funktion aktivieren oder deaktivieren möchte und bietet mehr Informationen zum Thema Mausgesten an. Dabei sollten diese Systeme die Regel von Ben Shneiderman beachten, dass die Kontrolle immer vom Benutzer ausgehen sollte. Die Hilfestellungen sollten daher unaufdringlich und nicht unterbrechend gestaltet sein [Saf08].

Um die Geste zu erklären, ist bei einfachen Gesten oft ein kurzer Text mit einer Illustration ausreichend. Dabei bieten sich besonders Metaphern und Vergleiche für ein besseres Verständnis an. Bei komplexeren Gesten ist es aber meistens erforderlich, eine Demonstration in Form einer kleinen Animation oder eines Videos anzubieten. (vgl. [Saf08])

5.5 Standards

Standards dienen dazu, Benutzungsoberflächen systematisch zu gestalten. Es ist zwar möglich, Oberflächen intuitiv gut zu gestalten, die Wahrscheinlichkeit ist aber groß, Aspekte zu übersehen. Daher dienen Standards zur Vermeidung von Fehlern. Standards werden aus Theorien, empirischen Untersuchungen oder Erfahrungen gebildet. Dabei unterscheidet man zwischen Prinzipien, die eher generelle Sachverhalte darlegen und Richtlinien, die sich mit konkreten Kriterien wie z.B. der Festlegung von Schriftarten, Farben oder dem Aussehen von Buttons beschäftigen.

5.5.1 Die acht goldenen Regeln des Interface Design

Ben Shneiderman hat acht Prinzipien aufgestellt, die sich auf alle Benutzungsoberflächen anwenden lassen.

1. **Konsistenz** Einheitlichkeit bei Terminologie, Aufbau der Oberfläche, Verwendung von Icons, Design und Farben.

2. Universelle Zielgruppe Berücksichtigung von möglichst verschiedenen Benutzern. Z.B. Alt - Jung, Technophil - Technophob, Anfänger - Experte.

3.Rückmeldungen Der Systemzustand muss zu jeder Zeit ersichtlich sein. Rückmeldungen für unwichtige Aktionen oder häufige Aktionen sollten dabei dezent gestaltet sein.

4. Abgeschlossene Dialoge / Aktionen Es muss klar sein, wann eine Aktion anfängt, in Bearbeitung ist und wann sie wieder beendet ist.

5. Fehlerbehandlung und Fehlervermeidung Aktionen, die keinen Sinn machen, sollten nicht angeboten oder auswählbar sein. Tritt ein Fehler auf, sollte der Benutzer eine potentielle Fehlerursachenanalyse erhalten, die zur Lösung dienlich ist. Der Benutzer darf das System nicht in einen ungültigen Zustand versetzen können.

6. Umkehrung von Aktionen Dem Benutzer müssen konsistente und leichte Undo-Funktionen angeboten werden, damit eine explorative Benutzung des Systems möglich ist.

7. Der Benutzer hat die Kontrolle Aktionen sollten immer vom Benutzer ausgeführt werden. Von der der Benutzungsoberfläche sollten selbst keine überraschenden Aktionen ausgeführt werden. Außerdem müssen sich angefangene Dialoge zu jedem Zeitpunkt abbrechen lassen.

8. Verringertes Abfragen des Kurzzeitgedächtnisses Die Oberfläche sollte die Belastung des Kurzzeitgedächtnisses möglichst verringern. Sie sollte daher nicht zu viele und nur notwendige Elemente aufweisen und sich strukturiert und aufgeräumt präsentieren.

5.5.2 Eigenschaften von guten gestischen Benutzungsschnittstellen

Die folgenden Eigenschaften hat Dan Saffer als Kriterien für eine gute gestengesteuerte Benutzungsschnittstelle ausgemacht (vgl. [Saf08]).

Entdeckbarkeit / Feststellbarkeit Diese Eigenschaft beschreibt, wie leicht sich die Funktionen des Gerätes dem Benutzer offenbaren. Dies ist besonders wichtig für Gesten, da diese in der Regel erlernt werden müssen.

Reaktionsfähigkeit In der Realität sind wir es gewohnt, dass sofort eine Reaktion erfolgt, wenn wir einen Gegenstand berühren. Bei Touchscreens wird durch das Wegfallen der Maus eine Abstraktionsschicht weniger eingesetzt. Daher orientieren sich Touchscreen-Benutzungsoberflächen nach dem Modell der

direkten Interaktion näher an der Realität. Entsprechend erwartet der Benutzer hier auch eine schnellere Reaktion des Systems. Bei einer langsamen Reaktion wird der Benutzer schnell annehmen, dass ein Fehler vorliegt und entsprechend die Aktion noch einmal ausführen. Falls es sich also um eine Aktion handelt, die länger dauert, ist es wichtig dem Benutzer anzuzeigen, dass sie schon in Bearbeitung ist.

Angemessen Besonders für Gesten ist es wichtig, den kulturellen Kontext, die Situation und die Umgebung zu berücksichtigen. So kann zum Beispiel das wilde Schütteln eines Mobiltelefons zum Ablehnen eines Anrufes zwar sehr intuitiv sein, aber auch wenig dezent, wenn man sich gerade in Gesellschaft befindet.

Aussagekräftig Eine Geste ist nur dann einprägsam und verständlich, wenn ihre Bedeutung sinnvoll mit der Aktion verknüpft werden kann.

Verspielt Gesten und Touchscreens fördern ein spielend, exploratives Erlernen des Systems. Dafür muss das System sehr robust sein und dem Benutzer leicht ermöglichen, Aktionen wieder rückgängig zu machen.

Angenehm Die Gesten sollten ästethisch und funktional angenehm sein. Benutzer arbeiten lieber mit Geräten, die für alle Sinne ansprechend sind, auch wenn weniger ansprechende Systeme die gleiche Funktionalität besitzen.

5.5.3 GUI Konventionen für Touchscreens

Die folgenden Konventionen stammen aus dem Buch *Designing Gestural Interfaces* von Dan Saffer [Saf08] und aus den Richtlinien *Interaction Design Guide for Touchscreen Applications* von Gerd Waloszek [Wal09].

- Touchscreen-Anwendungen sind meist nicht darauf ausgelegt, mit anderen Programmen im Wechsel zu stehen und sollten immer im *Vollbildmodus* ausgeführt werden. Generell sollte die Anwendung auch keine anderen Fenster oder Dialoge starten, sondern diese im Hauptfenster integrieren.
- Die Benutzungsoberfläche sollte immer in zwei Versionen, eine für *Linkshänder* und eine für *Rechtshänder*, verfügbar sein.
- *Helle Hintergründe* sollten benutzt werden, um Spiegeleffekte und Fingerabdrücke, die durch die Oberfläche des Touchscreens bedingt sind, zu verringern.
- *Leicht gemusterte Flächen* verringern Spiegelungen.

- Da der Finger direkt anzeigt, was berührt wird, kommen Touchscreen-Systeme in aller Regel *ohne Mauszeiger* aus. Wenn die Genauigkeit des Systems sehr schlecht ist, wird manchmal ein Hilfszeiger eingesetzt, der anzeigt, wo der erkannte Punkt, im Gegensatz zur tatsächlichen Position des Fingers, ist.
- Effekte, die erzielt werden, wenn der Mauszeiger über einem Element schwebt, es aber noch nicht angeklickt wurde, lassen sich nicht realisieren. Dies hat zur Folge, dass *keine Tooltips* eingesetzt werden können, die normalerweise zur Erklärung von Icons benutzt werden.
- Während eine Maus mit verschiedenen Tasten ausgestattet ist, kann ein Element nur mit einem Finger berührt werden. Diese Funktionalität lässt sich zwar über Gesten (z.B. zwei Finger gleichzeitig bedeutet rechte Maustaste) nachbilden, dies wird aber in der Regel selten gemacht, da es sehr unnatürlich ist. Somit kann in der Regel *keine rechte Maustaste* benutzt werden.
- Da der Finger eine sehr unpräzise Eingabe gewährleistet, sollten *Buttons* wesentlich größer gestaltet werden, als es für die Maus üblich ist. Als Richtlinie kann eine *Mindestgröße von 2x2cm bei einem Abstand von mindestens 3mm* zueinander verwendet werden. Um dennoch mehr Freiheiten bei der Gestaltung der Elemente zu haben, bieten sich die sogenannten *Eisbergspitzen* und die *adaptive Vergrößerung* an. Als *Eisbergspitze* bezeichnet man ein Element, wenn es virtuell einen größeren Bereich hat, der es aktiviert, als es visuell gestaltet ist. *Adaptive Vergrößerung* bedeutet, dass ein Button kontextabhängig vergrößert wird, wenn es wahrscheinlich wird, dass er gedrückt wird. Zum Beispiel wird bei der englischen Version des iPhones nach dem Eintippen der Buchstaben *t* und *h* das *e* vergrößert, da *the* in der englischen Sprache eines der häufigsten Wörter ist.
- Texteingabe ist unvorteilhaft am Touchscreen. Daher sollte die *Notwendigkeit, Text einzugeben, minimiert* werden.
- Wenn Nummern eingegeben werden müssen, sollten nach Möglichkeit *Schieberegler* verwendet werden.
- Bäume (z.B. Dateibäume) sind nicht geeignet für Touchscreens, da sie relativ filigran sind und auch viel von der Verdeckung betroffen sind. Es sollten also *keine Bäume* eingesetzt werden. Hierarchien sollten so in einzelne *GUI*-Elemente zerlegt werden, dass sie als einzelne Listen dargestellt werden können.
- In der Regel werden *keine aufklappenden Menüs*, wie Kontextmenüs oder Dateimenüs, eingesetzt. Diese sind sehr unpraktisch, da sie oft von dem Phänomen der Verdeckung betroffen sind.
- Es gibt *keinen standardmäßig selektierten Button*, wie es in den meisten normalen Dialogen der Fall ist. Da bei Touchscreen keine Entertaste vorhanden ist, die diesen Button ausführen könnte, ergibt sich kein Vorteil.

- Die Undo-Funktion, die die letzte Aktion widerruft, wird meist nicht umgesetzt, da sie sehr abstrakt ist und sich nicht natürlich in die Oberfläche integrieren lässt. Wie schon Ben Shneidermann in seinen acht goldenen Regel des Designs feststellte, ist es aber sehr wichtig für den Benutzer, dass er Aktionen leicht rückgängig machen kann. Aus diesem Grund sollte die Oberfläche immer einen leichten und offensichtlichen Weg bieten, um durch eine gegenteilige Aktion die Änderung widerrufen zu können.

5.6 Zusammenfassung

Im Abschnitt Usability wurde erläutert, dass die Gebrauchstauglichkeit ein wichtiger Aspekt eines Systems ist, da sie die wesentliche Eindrücke des System für den Benutzer prägt und effizientes Arbeiten mit dem System ermöglicht. Zuerst wurden die Grundlagen des Usability-Engineerings vorgestellt. Es wurde auf den menschlichen Körper eingangen und z. B. erklärt wie eine Geste entworfen sein muss, um angenehm zu sein. Weitere Grundlagen waren Richtlinien wie die acht goldenen Regeln von Sheiderman und Konventionen für den Entwurf von Touchscreen-Bedienungsschnittstellen. Als Methoden des Usability-Engineerings wurde das Prototyping vorgestellt, bei dem einzelne Aspekte eines Systems modelliert werden, um sie auf ihre Gebrauchstauglichkeit untersuchen zu können. Eine weitere Methode war die Evaluation, bei der ein Produkt oder Prototyp von Probanden auf die Gebrauchstauglichkeit untersucht wurde. Die vorgestellten Grundlagen und Methoden konnten z. B. durch Evaluationen (siehe Abschnitt 13) genutzt werden um die Gebrauchstauglichkeit von *TaP* zu verbessern.

Teil III

Projektplanung

Kapitel 6

Projektorganisation

In diesem Kapitel wird das Vorgehen des Projektmanagements bei der Projektplanung beschrieben. Als erstes wird hierbei auf die Form des Projektmanagements eingegangen. In den darauf folgenden Abschnitten wird die Zeitplanung genauer erläutert und der Meilensteinplan erklärt sowie auf die Rollenverteilung innerhalb der Projektgruppe eingegangen. Den Abschluss bildet eine Beschreibung der Planungsentwicklung, in der nennenswerte Abweichungen vom Projektplan erläutert werden.

6.1 Agiles Projektmanagement nach dem *oose Engineering Process*

Für die Durchführung dieser Projektgruppe wurde die Verwendung eines agilen Vorgehensmodells vorgeschrieben. Agile Modelle haben gegenüber klassischen Modellen, wie zum Beispiel dem Wasserfall- oder V-Modell, den Vorteil, dass zu Beginn des Projektes noch nicht alle Anforderungen an das Produkt bekannt sein müssen. Bei agilen Vorgehensmodellen entwickeln sich die Dokumente und das Programm inkrementell. Da die Software-Entwicklung in der Projektgruppe in nur einem Jahr stattfindet, ist die Nutzung eines agilen Modells sehr sinnvoll. Die verfügbare Zeit kann komplett für die Entwicklung des Produktes genutzt werden, denn es muss zu Beginn des Projektes keine Zeit verwendet werden, um eine Anforderungsanalyse zu machen, die alle zukünftigen Features des Produktes abdeckt.

In dieser Projektgruppe ist agiles Projektmanagement nach dem *oose Engineering Process* durchgeführt worden. Dieser Prozess wurde jedoch auf die Gegebenheiten der Projektgruppe angepasst. Die Abweichungen werden nach einer kurzen Beschreibung des *oose Engineering Process* erläutert.

6.1.1 Oose Engineering Process

Der *oose Engineering Process* ist ein Vorgehensleitfaden für die objektorientierte Softwareentwicklung. Hierbei stellt er nicht den Anspruch, den Prozess dieser Softwareentwicklung bis ins Detail zu beschreiben, sondern geht davon aus, dass Projekte nur zu einem gewissen Grad planbar sind (vgl. [nul06]).

Ein wichtiger Planungsschritt zum Erreichen des gesetzten Ziels ist die Unterteilung des gesamten Projektzeitraums in gleich große Zeitabschnitte, die sogenannten Iterationen. Abbildung 6.1 zeigt den prinzipiellen Aufbau einer Iteration.

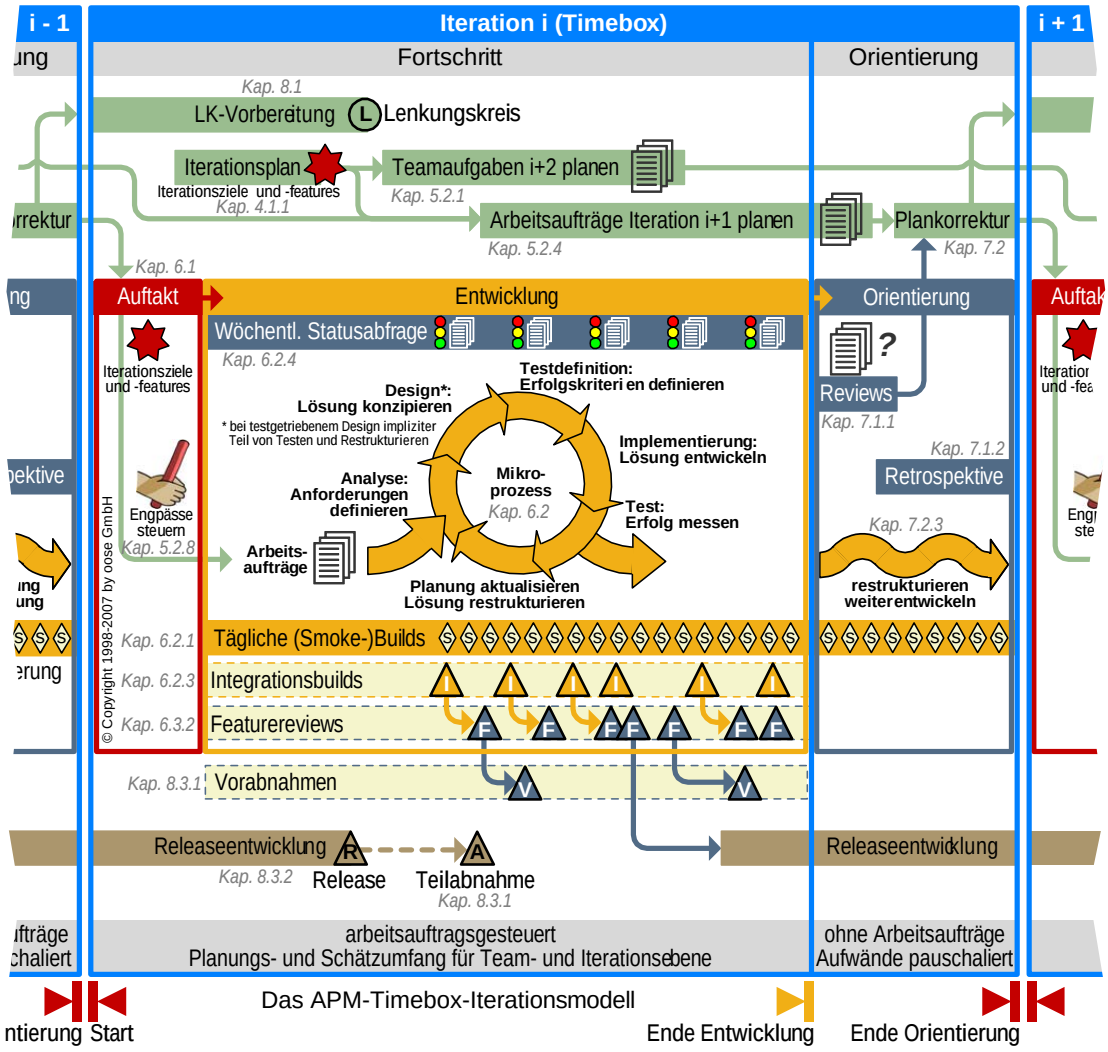


Abbildung 6.1: Prinzipieller Aufbau einer Iteration

Bei Iterationen handelt sich um ähnliche, sich wiederholende Zeiteinheiten, auch Timeboxes genannt. Eine Iteration kann je nach Projektgröße und -dauer zwischen zwei und zehn Wochen dauern. Bei einem mittleren Projekt, d. h. einem Projekt mit ca. 8-20 Mitarbeitern, wird eine Iterationsdauer von 3-5 Wochen empfohlen. In diesem Rahmen sollte sich demnach auch die Iterationsdauer der Projektgruppe befinden. Der Begriff Timebox verdeutlicht den festen Zeitrahmen, den eine Iteration definiert. Dieser Zeitrahmen kann nicht verschoben werden, denn entscheidend ist hier, was zu einem bestimmten Zeitpunkt erreicht wurde. Das heißt am Ende einer Iteration sollen bestimmte Ergebnisse in einem bestimmten Detailgrad vorliegen. Konnten nicht alle Aufgaben in der dafür vorgesehenen Iteration umgesetzt werden, werden diese offenen Aufgaben notfalls in die nächste Iteration verschoben, keinesfalls wird jedoch die Iteration verlängert. Während einer Iteration finden alle elementaren Entwicklungsaktivitäten wie Analyse, Implementierung und Test statt. In diesem Zusammenhang spricht man auch vom Iterations-Mikroprozess, der im sogenannten Fortschrittsabschnitt der Iteration durchgeführt wird.
Der zweite wichtige Teilabschnitt einer Iteration ist der Orientierungsabschnitt. Im Orientierungsabschnitt wird die vergangene Iteration betrachtet und der Fortschritt im Bezug auf das gesamte Projekt bewertet. Auf diesen Rückblick und die daraus gewonnenen Erkenntnisse aufbauend wird die Planung für die nächste Iteration aktualisiert und verfeinert.
Ein weiteres wichtiges Merkmal einer Iteration ist, dass sie stets eine teilfertige und ausführbare Version des zu erstellenden Produktes liefert (auch *Build* genannt). Hier spiegelt sich der iterativ-inkrementelle Charakter der agilen Entwicklung wieder. In kurzen, sich wiederholenden, Zeitabschnitten wird das Zielprodukt prototypartig weiterentwickelt. Durch die ständige Erweiterung des Prototypen *wächst* dieser letztlich solange, bis er alle Eigenschaften des Zielprodukts aufweist, d. h. dieses schließlich repräsentiert.

Um den Fortschritt innerhalb des Projektes messen und überprüfen zu können, setzt man im *oose Engineering Process Meilensteine* ein.

Definition 1 (Meilenstein) *„Ein Meilenstein definiert einen Termin, zu dem eine Menge von Ergebnissen in einer bestimmten Detaillierung und Vollständigkeit nachprüfbar und formal vorliegen muss. Liegen die Ergebnisse zum geplanten Termin nicht vor, wird der Meilensteintermin verschoben. Ein Meilenstein ist ein Hilfsmittel zur Planung und Überwachung eines Entwicklungsprozesses.“ [OW08]*

Zu der Menge von Ergebnissen, die zum Meilensteintermin vorliegen müssen, können Dokumentation, aber auch *Releasefeatures* gehören.

Definition 2 (Releasefeature) *„Ein Releasefeature ist eine Ausbaustufe eines Produktfeatures für genau ein Release und umfasst alle Iterationsfeatures des Produktfeatures, die bis zum Releasetermin erstellt werden sollen.“ [OW08]*

6.1.2 Abweichungen vom oose Engineering Process

Um den *oose Engineering Process* in der Projektgruppe durchführen zu können, mussten einige Änderungen vorgenommen werden. Im *oose Engineering Process* stellt der Projektleiter eine Instanz dar, die zwischen dem Kunden und dem Entwicklerteam steht. Als solche versucht er, die Kundenwünsche effizient in einen Projektplan einzuarbeiten und eine realistische Zeitplanung für die Durchführung des Projektes zu erstellen, jedoch ohne in der Implementierung tätig zu sein. In der Projektgruppe ist dies nicht möglich, da sich jedes Teammitglied an der Implementierung beteiligen muss. Um diesem Mehraufwand entsprechen und die Aufgaben des Projektleiters weiterhin wahrnehmen zu können, wurde die Projektleiter-Position doppelt besetzt.

Als weitere Abweichungen vom *oose Engineering Process* wurden die ursprüngliche Bedeutung der Begriffe *Meilenstein* und *Releasefeature* angepasst. Ein *Meilenstein* ist zwar weiterhin „*[...] ein Hilfsmittel zur Planung und Überwachung des Entwicklungsprozesses*“ [OW08], jedoch gibt es keinen Unterschied mehr zum Begriff des Releasefeatures. Ein Release besteht aus mindestens einem *Meilenstein*, sodass in den Iterationsfeinplanungen die Subfunktionalitäten der *Meilensteine* in Iterationsfeatures und entsprechende Arbeitspakete heruntergebrochen werden können.

Im *oose Engineering Process* vorgesehene tägliche *Smoke-Builds* (siehe Abbildung 6.1) sind für die Projektgruppe ungeeignet und nicht sinnvoll, da nicht jeden Tag Fortschritte erzielt werden. Außerdem ergibt sich aus der Tatsache, dass die Projektgruppe einen geplanten Wochenumfang von 20 Stunden hat und dass die Teilnehmer der Projektgruppe Studenten mit unterschiedlichen zeitlichen Verfügbarkeiten sind, dass die für den Iterationsmikroprozess angedachten täglichen Steuerungstreffen (siehe hierzu [OW08]) wegfallen; eine wöchentliche Rückmeldung zum momentanen Entwicklungszustand gibt es jedoch.

6.2 Zeitplanung

Die Projektgruppe ist auf den Zeitraum eines Jahres ausgelegt. Dieses wird in zwei Semester unterteilt, zu deren Ende jeweils eine Abnahme stattfindet, die zum einen einen Bericht über das vergangene Semester und zum anderen eine Programmversion mit allen bis dahin geplanten Features vorstellt. Zur ersten Abnahme soll die Pflicht-Umsetzung (siehe Abbildung 6.2, Abnahme 1) des Produktes fertiggestellt sein. Am Ende des zweiten Semesters findet die zweite Abnahme statt, die gleichzeitig den Abschluss des Projektes bildet. Zu dieser abschließenden Abnahme soll die Kür, d.h. das erweiterte Produkt, welches auf der Pflicht-Umsetzung aufbaut, präsentiert werden. Unmittelbar nach der ersten Abnahme ist eine einwöchige kreative Phase geplant, in der die Ziele und *Meilensteine* hinsichtlich der Kür, auf Basis der Pflicht-Umsetzung, diskutiert und ggf. ergänzt bzw. verändert werden.

Den Weg zu den Abnahmen und Programmversionen unterstützen Releases und *Meilensteine*. Diese unterteilen den Entwicklungsprozess und sorgen dafür, dass bestimmte Teilumsetzungen rechtzeitig fertiggestellt und zu einem funktionierenden Teilprodukt zusammengefügt werden. Für die erste Abnahme sind die Releases 1 und 2 umzusetzen, die sich aus insgesamt neun *Meilensteine*n zusammensetzen. Die zweite Abnahme setzt sich aus den Releases 3 und 4 zusammen, die wiederum auf den vorherigen Releases aufbauen. In diesen beiden letzten Releases werden fünf weitere *Meilensteine* umgesetzt. Auf die Inhalte der *Meilensteine* wird in Abschnitt 6.2.1 eingegangen. Die Projektlaufzeit ab Januar 2009 ist in zwölf Iterationen à drei Wochen aufgeteilt. Diese kurze Iterationsdauer wurde gewählt, weil es sich um ein mittleres Projekt mit neun Entwicklern handelt, deren Arbeitszeit pro Woche stark begrenzt ist (vgl. [OW08]). Der Entwicklungsabschnitt einer Iteration wurde auf 13 Werktage festgelegt, sodass der Orientierungsabschnitt zwei Werktage umfasst. Der Orientierungsabschnitt wurde bewusst kurz gehalten, um soviel Zeit wie möglich für die Entwicklung investieren zu können. Die Projektleitung entschied, die Aufteilung in Iterationen erst zeitnah zu dem Beginn der Implementierungszeit vorzunehmen, da sie diese Aufteilung für die Einarbeitungszeit, die insbesondere der Ideenfindung und Zieldefinition diente, nicht für vorteilhaft erachtete. Aus diesem Grund beginnt die Planung mit den Iterationen erst ab Januar 2009. Die gesamte zeitliche Planung der Projektgruppe wurde in einem Projektplan illustriert, der in den Abbildungen 6.2 und 6.3 zu sehen ist. Dort ist auch die Einteilung in drei Phasen zu erkennen. Die Startphase des Projektes umfasst die erste Iteration, in der eine konkretere Zieldefinition und der Projekt- und Releaseplan entsteht. Die Iterationen zwei bis elf bilden die Hauptphase. Während dieser Phase wird das Produktziel der Projektgruppe entwickelt. Die Abschlussphase dient u. a. der Durchführung von Abschlusstests, der Eliminierung letzter Fehler, der optischen Verschönerung der Benutzungsoberfläche und der Fertigstellung des Abschlussberichts. Diese Phase findet in der zwölften und letzten Iteration statt.

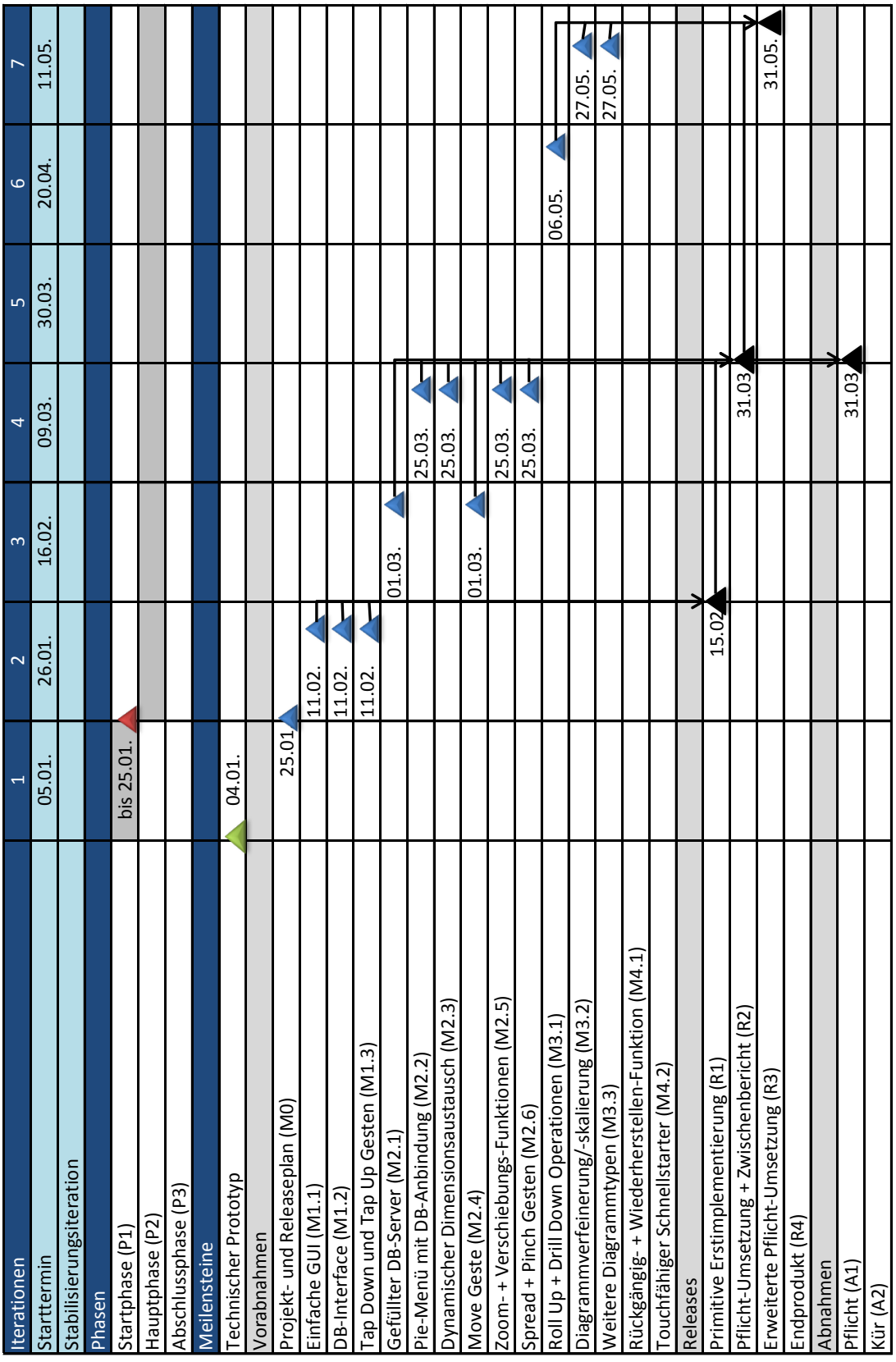


Abbildung 6.2: Teil 1 des Projektplans

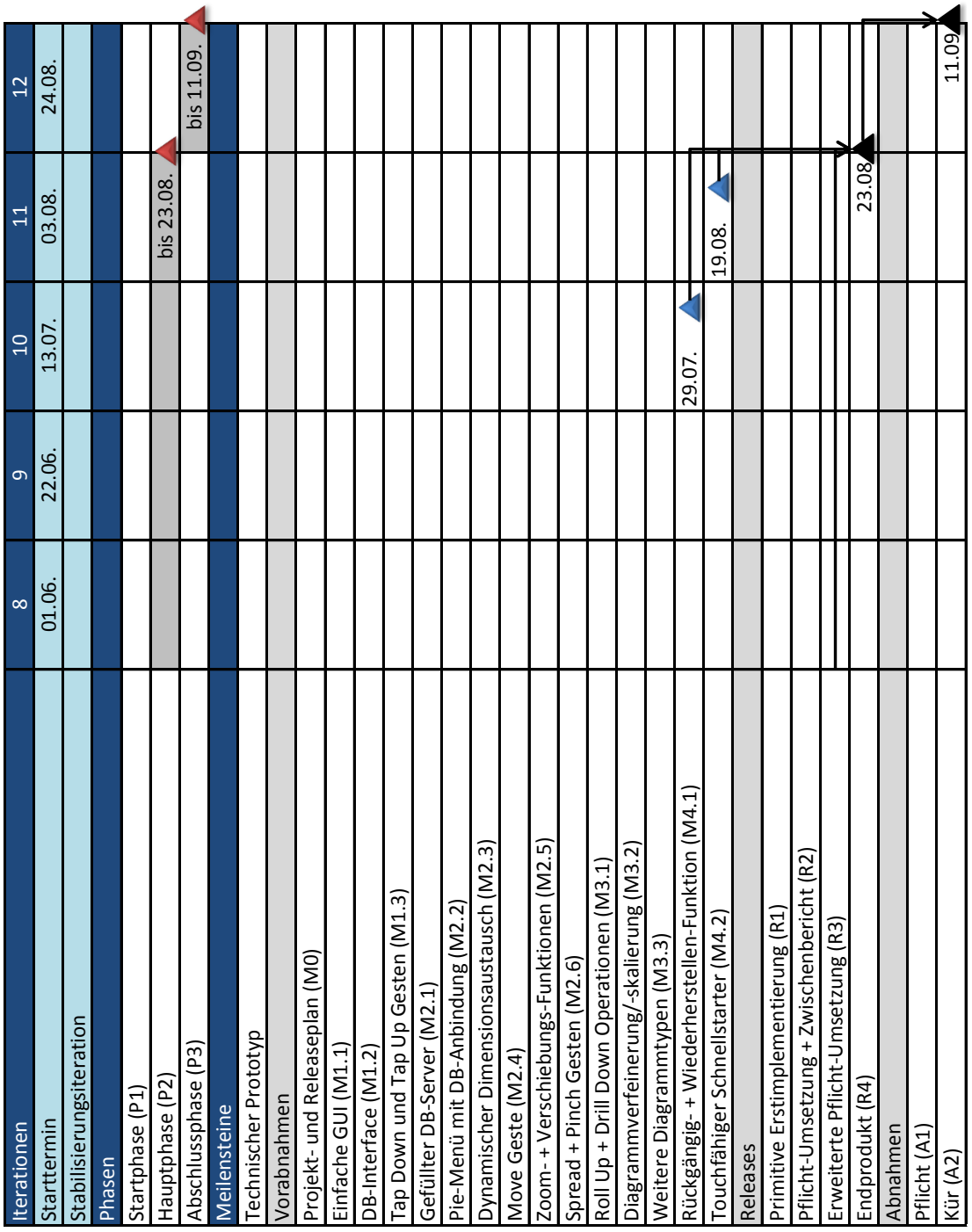


Abbildung 6.3: Teil 2 des Projektplans

6.2.1 Meilensteine

Dem Projektplan in den Abbildungen 6.2 und 6.3 sind die einzelnen *Meilensteine* über den gesamten Projektverlauf hinweg zu entnehmen. Dabei werden die einzelnen *Meilensteine* mit blauen Dreiecken gekennzeichnet, während Releases und Abnahmen mit schwarzen Dreiecken dargestellt sind. Die Endpunkte der Phasen sind durch rote Dreiecke markiert. Bei dem ersten *Meilenstein* handelt es sich um einen primitiven technischen Prototypen, der als Resultat einer Einarbeitungsphase entstand, in der ein grober Gesamtüberblick über alle Systemkomponenten vorgenommen wurde. Dieser Prototyp diente zur Aufdeckung von Problemen, zur Ideenfindung und Einarbeitung in die verwendeten Technologien und ist deshalb kein Bestandteil eines Releases. Zur Abgrenzung wurde dieser *Meilenstein* durch ein grünes Dreieck dargestellt.

Nach der Erstellung des bereits mehrfach zitierten Projektplans wurden drei *Meilensteine* angesetzt, die zusammen das erste Release *Primitive Erstimplementierung* R1 bilden.

Einfache GUI (M1.1) Der erste *Meilenstein* steht für die Umsetzung einer einfachen grafischen Benutzungsoberfläche. Diese beinhaltet ein aus den Daten der angebundenen Datenbank generiertes Punkt-Diagramm. Außerdem verfügt die *GUI* über einen Button zum Beenden des Programms, sowie über einen Button zum Erstellen des Diagramms. Darüber hinaus ist das Hauptfenster standardmäßig maximiert und rahmenlos.

DB-Interface (M1.2) Das Datenbank-Interface implementiert den Verbindungsaufbau zur Datenbank. Außerdem stellt es Methoden für das Abrufen der Daten und Metadaten aus der Datenbank bereit, sodass mit Hilfe dieser Methoden das Diagramm inklusive der Achsenbeschriftungen erstellt werden kann.

***TapDown*- und *TapUp*-Gesten (M1.3)** Die *TapDown*- und *TapUp*-Gesten werden implementiert und an die Buttons gebunden. Sie sorgen dafür, dass die o.g. Buttons durch *Touch*-Gesten betätigt werden können, die über den *TUIO*-Simulator simuliert werden.

Das zweite Release *Pflicht-Umsetzung und Zwischenbericht (R2, TaP v1.0)* baut auf R1 auf und beinhaltet folgende weitere *Meilensteine*:

Gefüllter DB-Server (M2.1) Die Datenbank wird mit allen verfügbaren Daten gefüllt, damit die Anwendung auf existierende Daten zurückgreifen kann.

Pie-Menü mit DB-Anbindung (M2.2) Ein drehbares und ruckelfreies Pie-Menü wird erstellt und an die Datenbank angebunden. Mit Hilfe des Pie-Menüs kann in den verfügbaren Dimensionen und Kennzahlen der Datenbank navigiert werden. Außerdem werden mittels *Drag and Drop* die anzuzeigenden Dimensionen

bzw. Kennzahlen auf die Achsen des Punkt-Diagramms gezogen, welches die dazugehörigen Daten dann visualisiert.

Dynamischer Dimensionsaustausch (M2.3) Es wird eine Funktion erstellt, die jederzeit ein Wechseln der Achsenbelegungen ermöglicht. Dies geschieht durch Ziehen neuer Dimensionen bzw. Kennzahlen aus dem Pie-Menü auf die entsprechende Achse des Diagramms.

***Move*-Geste (M2.4)** Eine *Move*-Geste wird implementiert und die Anbindung an entsprechende Komponenten. Diese dient dazu, *GUI*-Elemente wie z. B. das Pie-Menü oder das Diagramm beliebig zu verschieben.

Zoom- und Verschiebe-Funktionen (M2.5) Es wird die Funktion bereitgestellt, innerhalb des Diagramms zu vergrößern und zu verkleinern. Hierdurch kann z. B. in Anhäufungen von Datenpunkten hineingezoomt werden. Darüber hinaus ermöglicht die Verschiebefunktion das Verschieben eines vergrößerten Ausschnitts des Diagramms. Die Zoomfunktion wird durch eine spezielle Kombination von *TapDown*- und *Move*-Gesten ausgeführt, der sogenannten *Pinch*-Geste.

***Spread*- und *Pinch*-Gesten (M2.6)** Die *Spread*- und *Pinch*-Gesten werden implementiert und dienen später dazu, die Daten-Verfeinerungsfunktion (M3.1) auszulösen.

In das dritte Release fließen zusätzlich folgende drei *Meilensteine* ein:

Roll Up und Drill Down Operationen (M3.1) Die *RollUp* und *DrillDown* Operationen ermöglichen die Navigation zwischen den einzelnen Detailierungsstufen der Daten. Diese Funktion ermöglicht es in Verbindung mit M3.2, in einem angezeigten Diagramm zwischen einzelnen Detailierungsstufen zu wechseln und sich diese unmittelbar visualisieren zu lassen.

Diagrammverfeinerung/-skalierung (M3.2) Das Diagramm wird dahingehend erweitert, dass *RollUp* und *DrillDown* Operationen direkt im Diagramm ausgeführt werden können. Dies geschieht mittels der *Spread*- und *Pinch*-Gesten aus M2.6.

Weitere Diagrammtypen (M3.3) Neben dem Punktdiagramm werden weitere Diagrammtypen, deren Gestalt noch nicht spezifiziert wird, entwickelt.

Das vierte Release *(TaP v2.0)* fordert zwei weitere *Meilensteine*.

Rückgängig- und Wiederherstellen-Funktion (M4.1) Eine Rückgängig- und Wiederherstellen-Funktion ermöglicht dem Nutzer, bereits getätigte Schritte rückgängig zu machen und wiederherzustellen. Diese Funktion bezieht sich auf bestimmte Ansichten der Daten-Visualisierung.

Touchfähiger Schnellstarter (M4.2) Ein Programm, welches dazu dient, mittels Touch-Bedienung die Zielanwendung zu starten, wird in diesem *Meilenstein* umgesetzt. Sie dient zur ausschließlichen Touch-Bedienung der Anwendung inklusive deren Start.

Die Anzahl der zu diesem Zeitpunkt festgelegten *Meilensteine* für die Kür-Umsetzung wurde bewusst gering gehalten, um weiteren Ideen, die im Anschluss an die Pflicht-Abnahme entstehen, Raum zu gewähren.

6.2.2 Planungsentwicklung

Bis zur ersten Abnahme wurden keine nennenswerten Abweichungen von dem ursprünglichen Projektplan 6.2 und 6.3 vorgenommen. Nach dieser Abnahme, d. h. bis zur zweiten und damit Endabnahme, wurde der Projektplan mehrmals angepasst. Die Abbildungen 6.4 bis 6.7 zeigen den an den tatsächlichen Projektverlauf angepassten Projektplan.

Da der Plan erhebliche Änderungen beim Funktionsumfang, der in den Releases 3 und 4 realisiert werden sollte, aufzeigt, mussten auch die in diese Releases einfließenden *Meilensteine* neu definiert werden. Alle in die letzten beiden Releases bearbeiteten *Meilensteine* werden im Folgenden beschrieben.

In das dritte Release fließen vier *Meilensteine* ein:

Erweiterung des Punktdiagramms (M3.1) Das Punktdiagramm soll dahingend erweitert werden, dass Punkte sowohl über eine Farb- als auch über eine Größendimension verfügen.

Abstraktes Datenmodell (M3.2) Die bisher speziell auf eine *ADOMD.NET*-Datenbank ausgerichtete Datenbankzugriffslogik soll auf ein allgemeines Datenmodell abstrahiert werden, sodass *TaP* auch andere Datenbanken, ohne größere Anpassungen des Datenmodells, verwenden kann.

Usability-Verbesserungen (M3.3) Anhand einer Evaluation wurden bei der Anwendung *TaP* einige Usability-Schwächen aufgedeckt. Diese sollen in diesem *Meilenstein* weitestgehend behoben werden, um *TaP* so gebrauchstauglich wie möglich zu gestalten.

Umrüstung auf MUSTANG-Backend (M4) Mit der Erstellung des abstrakten Datenmodells (M3.2) findet die Umrüstung der Datenbankschnittstelle auf das *Multidimensional Statistical Data Analysis Engine (MUSTANG)*-Backend statt. Auch die Daten, die von *MUSTANG* bereitgestellt werden, sollen für *TaP* verwendet werden.

Im Vergleich zum ursprünglichen Projektplan haben sich alle dort geplanten *Meilensteine* verändert. Die ursprünglichen *Meilensteine* M3.1 DrillDown *und* RollUp *Operationen* und M3.2 *Diagrammverfeinerung/-skalierung* wurden in das 4. Release verschoben. Der *Meilenstein* M3.3 *Weitere Diagrammtypen* wurde komplett entfernt, da sich dieses Feature in der zur Verfügung stehenden Zeit nicht mehr realisieren ließ. Stattdessen wurde beschlossen, das Punktdiagramm um mehrere Dimensionen zu erweitern. Außerdem kam die neue hochpriorisierte Anforderung der Kunden, *MUSTANG* anzubinden, hinzu. Darüber hinaus galt es, *TaP* gebrauchstauglich und ausgereift zu gestalten, wodurch M3.3 entstand.

Das vierte Release fordert insgesamt neun *Meilensteine*.

Roll Up und Drill Down Operationen (M5.1) Die *RollUp* und *DrillDown* Operationen ermöglichen die Navigation zwischen den einzelnen Detailierungsstufen der Daten. Diese Funktion ermöglicht es in Verbindung mit M5.2, in einem angezeigten Diagramm zwischen einzelnen Detailierungsstufen zu wechseln und sich diese unmittelbar visualisieren zu lassen.

Diagrammverfeinerung/-skalierung (M5.2) Das Diagramm wird dahingehend erweitert, dass *RollUp* und *DrillDown* Operationen direkt im Diagramm ausgeführt werden können. Dies geschieht mittels der *Spread*- und *Pinch*-Gesten aus M2.6.

Zeitliche Animation (M6.1) Die einzelnen Punkte des Punktdiagramms sollen animiert werden können, sodass z. B. Entwicklungen über einen bestimmten Zeitraum hinweg besonders anschaulich dargestellt werden können. Diese Animation stellt, z. B. neben Größe und Farbe, eine weitere Dimension des Punktdiagramms dar.

Parallele Darstellung mehrerer Diagramme (M6.2) Um dem Nutzer den direkten Vergleich zwischen bestimmten Datensätzen zu ermöglichen, wird in diesem *Meilenstein* die gleichzeitige Darstellung mehrerer Diagramme realisiert.

Verschiebbarer Desktop (M6.3) Die parallele Darstellung mehrerer Diagramme mit Hilfe einer unendlichen, verschiebbaren Desktopfläche wird ermöglicht.

Manuelle Zustandsspeicherung (M7.1) Der Nutzer soll beliebige Zustände eines Diagramms bzw. einer Diagrammansicht manuell speichern und wiederherstellen können.

History-Funktion (M7.2) Dieser *Meilenstein* realisiert die Zustandsspeicherung wie unter M7.1 beschrieben, allerdings automatisch, d.h. ohne dass der Nutzer bewusst einen zu speichernden Zustand auswählt. M7.1 und M7.2 realisieren den ursprünglichen *Meilenstein* M4.1 *Rückgängig- und Wiederherstellen-Funktion* (s. Abbildung 6.3).

Filterfunktion für Diagramme (M7.3) Um dem Nutzer zu ermöglichen, die Visualisierung auf einen bestimmten Datenbereich, welcher aus einzelnen Knoten unterschiedlicher Dimensionen besteht, zu beschränken, soll eine zusätzliche Filterfunktion implementiert werden.

Usability-Evaluation (M8) Am Ende der Entwicklung von *TaP* ist die Durchführung einer weiteren Usability-Evaluation geplant, um die Verbesserungen und Neuentwicklungen hinsichtlich ihrer Gebrauchstauglichkeit zu überprüfen. Weitere sich daraus ableitende Verbesserungen sollen anschließend umgesetzt werden.

Neben den verschobenen *Meilenstein*en M5.1 und M5.2 werden zum 4. Release einige neue Funktionen bereitgestellt. Dazu zählt die zeitliche Animation als weitere Dimension des Punktdiagramms, sowie die Möglichkeit, mittels eines verschiebbaren Desktops, mehrere Diagramme anzuzeigen. Geplant war außerdem eine zusätzliche Filterfunktion zur gezielten Betrachtung spezieller Dimensionsbereiche (M7.3), die jedoch auf Grund von Zeitmangel und höherer Priorisierung anderer Features nicht mehr umgesetzt werden konnte. Am Ende der Entwicklung von *TaP* findet eine erneute Usability-Evaluation statt, deren Erkenntnisse Aufklärung über die Verbesserung der Anwendung im Vergleich zur ersten Evaluation gibt. Der ursprünglich geplante *Meilenstein* M4.2 *Touchfähiger Schnellstarter* musste aus gleichem Grund wie M7.3 entfernt werden.

Iterationen	1	2	3	4		5	6
Starttermin	05.01.	26.01.	16.02.	09.03.		20.04.	11.05.
Stabilisierungsiteration					30.03.		
Phasen							
Startphase (P1)	bis 25.01.						
Hauptphase (P2)							
Abschlussphase (P3)							
Meilensteine							
Technischer Prototyp	04.01.						
Vorabnahmen							
Projekt- und Releaseplan (M0)	25.01						
Einfache GUI (M1.1)		11.02.					
DB-Interface (M1.2)		11.02.					
Tap Down und Tap Up Gesten (M1.3)		11.02.					
Gefüllter DB-Server (M2.1)			04.03.				
Pie-Menü mit DB-Anbindung (M2.2)				25.03.			
Dynamischer Dimensionsaustausch (M2.3)				25.03.			
Move Geste (M2.4)			04.03.				
Zoom- + Verschiebungs-Funktionen (M2.5)				25.03.			
Spread + Pinch Gesten (M2.6)				25.03.			
Erweiterung des Punktdiagramms (M3.1)							27.05.
Abstraktes Datenmodell (M3.2)							27.05.
Usability-Verbesserungen (M 3.3)							27.05.
Umrüstung auf MUSTANG-Backend (M4)							27.05.
Roll Up + Drill Down Operationen (M5.1)							
Diagrammverfeinerung/-skalierung (M5.2)							
Zeitliche Animation (M6.1)							
Parallele Darstellung mehrerer Diagramme (M6.2)							
Verschiebbarer Desktop (M6.3)							
Manuelle Zustandsspeicherung (M7.1)							
History-Funktion (M7.2)							
Filterfunktion für Diagramme (M7.3)							
Usability-Evaluation (M8)							

Abbildung 6.4: Teil 1 des aktuellsten Projektplans

Releases							
Primitive Erstimplementierung (R1)		15.02.					
Pflicht-Umsetzung + Zwischenbericht (R2)				31.03.			
Erweiterte Pflicht-Umsetzung (R3)							31.05.
Endprodukt (R4)							
Abnahmen							
Pflicht (A1)				31.03.			
Kür (A2)							

Abbildung 6.5: Teil 2 des aktuellsten Projektplans

Iterationen	7	8	9	10	11
Starttermin	01.06.	22.06.	13.07.	03.08.	24.08.
Stabilisierungsiteration					
Phasen					
Startphase (P1)					
Hauptphase (P2)				bis 23.08.	
Abschlussphase (P3)					bis 11.09.
Meilensteine					
Technischer Prototyp					
Vorabnahmen					
Projekt- und Releaseplan (M0)					
Einfache GUI (M1.1)					
DB-Interface (M1.2)					
Tap Down und Tap Up Gesten (M1.3)					
Gefüllter DB-Server (M2.1)					
Pie-Menü mit DB-Anbindung (M2.2)					
Dynamischer Dimensionsaustausch (M2.3)					
Move Geste (M2.4)					
Zoom- + Verschiebungs-Funktionen (M2.5)					
Spread + Pinch Gesten (M2.6)					
Erweiterung des Punktdiagramms (M3.1)					
Abstraktes Datenmodell (M3.2)					
Usability-Verbesserungen (M 3.3)					
Umrüstung auf MUSTANG-Backend (M4)					
Roll Up + Drill Down Operationen (M5.1)		08.07.			
Diagrammverfeinerung/-skalierung (M5.2)		08.07.			
Zeitliche Animation (M6.1)		08.07.			
Parallele Darstellung mehrerer Diagramme (M6.2)	26.06.				
Verschiebbarer Desktop (M6.3)	23.06.				
Manuelle Zustandsspeicherung (M7.1)		03.07.			
History-Funktion (M7.2)		03.07.			
Filterfunktion für Diagramme (M7.3)				14.08.	
Usability-Evaluation (M8)				28.08.	

Abbildung 6.6: Teil 3 des aktuellsten Projektplans

Releases						
Primitive Erstimplementierung (R1)						
Pflicht-Umsetzung + Zwischenbericht (R2)						
Erweiterte Pflicht-Umsetzung (R3)						
Endprodukt (R4)				30.08.		
Abnahmen						
Pflicht (A1)						
Kür (A2)					11.09.	

Abbildung 6.7: Teil 4 des aktuellsten Projektplans

Auf dem Weg zu dieser letzten Version des Projektplans fanden zahlreiche Überarbeitungen der Features und *Meilensteine* statt. Dieses Vorgehen spiegelt die schrittweise klarer werdende Zieldefinition beim agilen Vorgehen wieder. Während des Projektverlaufs wurde das tatsächliche Ziel immer wieder verändert und näherte sich mehr und mehr dem Endziel an. Mit dem Projektfortschritt wurde außerdem klar, welche Features realisierbar sind und wie diese, je nach Kundenwunsch, zu priorisieren sind. Im Laufe des Projekts wurden auch neue Entwicklungswege aufgedeckt, die teilweise verfolgt wurden, wodurch dann ursprünglich geplante Features hintenangestellt oder komplett verworfen wurden.

6.2.3 Zeitliche Planungsinstrumente

Neben dem o. g. Projektplan verwendete das Projektmanagement weitere Planungsinstrumente zur Planung vom gesamten Projektverlauf, von Iterationen und von Wochen.

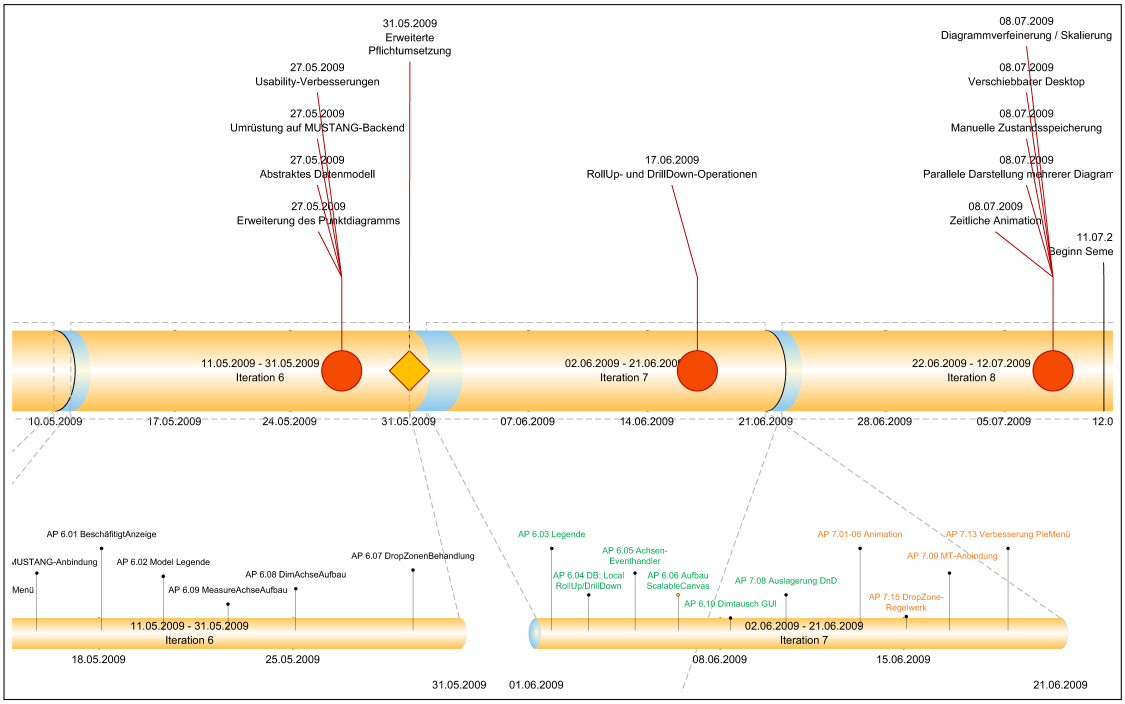


Abbildung 6.8: Ausschnitt des Zeitstrahls

Zeitstrahl

Der Zeitstrahl (s. Abbildung 6.8) wurde zur Veranschaulichung des gesamten Projektverlaufs verwendet. Er zeigt die Iterationen im Zeitverlauf mit den zugehörigen Features und *Meilensteine*n. Die Darstellungsform ermöglicht sowohl eine gröbere als auch eine

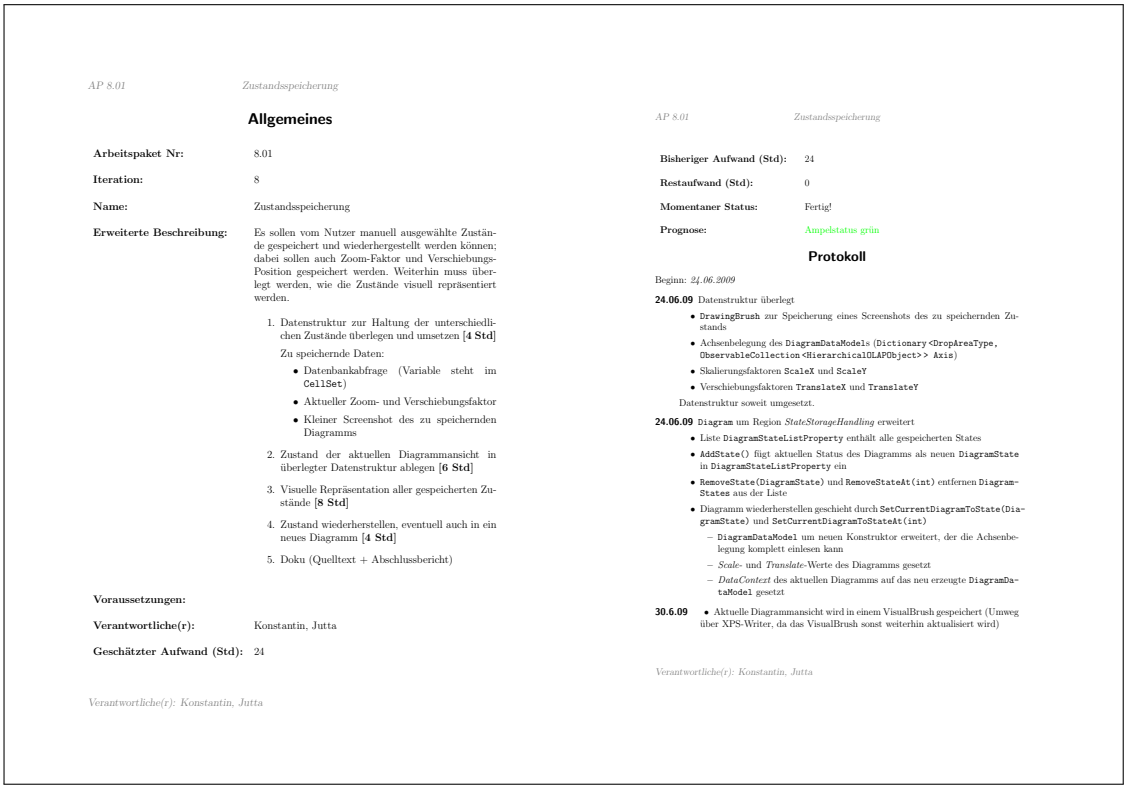
detailliertere Sicht auf den Verlauf und die einzelnen Iterationen. Insgesamt wurde der Zeitstrahl verwendet, um zu Beginn und am Ende einer Iteration (Auftakt und Abschluss) dem Team und den Kunden die für die jeweilige Iteration geplanten bzw. umgesetzten Arbeitspakete aufzuzeigen.

Präsentationsfolien und Statusdokumente

Die genauere Sicht auf die einzelnen Tätigkeiten einer Woche wurde in Form einer Präsentation veranschaulicht, die wöchentlich um Folien ergänzt wurde. Diese Folien (s. Abbildung 6.9) zeigten den Wochenausblick und -rückblick. In jeder wöchentlichen Sitzung wurde zunächst präsentiert, was für die vergangene Woche geplant war, was tatsächlich in der vergangenen Woche umgesetzt wurde und was für die folgende Woche geplant war. Auch Probleme und Terminschwierigkeiten einzelner Arbeitspakete wurden mit Hilfe der Präsentation diskutiert und veranschaulicht. Um den Status eines Arbeitspaketes zu verdeutlichen, wurde auf ein Ampelmodell zurückgegriffen, wie es auch beim OEP empfohlen wird.

Abbildung 6.9: Beipielfolie aus dem Wochenrückblick

Als Grundlage für die Erstellung der Folien dienten Statusdokumente (s. Abbildung 6.10), die von den einzelnen Entwicklerteams gepflegt wurden. Dort wurde aufgeführt, welche Teilaufgaben ein Arbeitspaket beinhaltete, von wem dieses bearbeitet wurde, in welcher Zeit welche konkreten Teilaufgaben erledigt wurden bzw. wo es Probleme gab und wie der aktuelle Restaufwand eingeschätzt wurde. Trotz dieser Statusdokumente war eine persönliche Rücksprache mit den einzelnen Entwicklerteams in der Regel sinnvoll, um einen umfassenden Einblick in die Problemstellung und damit verbundenen Tätigkeiten zu gewinnen. Somit war aktive Kommunikation unter den einzelnen Projektmitgliedern stets sehr wichtig und eins der effektivsten Planungsinstrumente.



AP 8.01 *Zustandsspeicherung*

Allgemeines

Arbeitspaket Nr:	8.01
Iteration:	8
Name:	Zustandsspeicherung
Erweiterte Beschreibung:	Es sollen vom Nutzer manuell ausgewählte Zustände gespeichert und wiederhergestellt werden können; dabei sollen auch Zoom-Faktor und Verschiebungs-Position gespeichert werden. Weiterhin muss überlegt werden, wie die Zustände visuell repräsentiert werden.

1. Datenstruktur zur Haltung der unterschiedlichen Zustände überlegen und umsetzen **[4 Std]**
   Zu speichernde Daten:
   - Datenbankabfrage (Variable steht im `CellSet`)
   - Aktueller Zoom- und Verschiebungsfaktor
   - Kleiner Screenshot des zu speichernden Diagramms
2. Zustand der aktuellen Diagrammansicht in überlegter Datenstruktur ablegen **[6 Std]**
3. Visuelle Repräsentation aller gespeicherten Zustände **[8 Std]**
4. Zustand wiederherstellen, eventuell auch in ein neues Diagramm **[4 Std]**
5. Doku (Quelltext + Abschlussbericht)

Voraussetzungen:	
Verantwortliche(r):	Konstantin, Jutta
Geschätzter Aufwand (Std):	24

Verantwortliche(r): Konstantin, Jutta

AP 8.01 *Zustandsspeicherung*

Bisheriger Aufwand (Std):	24
Restaufwand (Std):	0
Momentaner Status:	Fertig!
Prognose:	Ampelstatus grün

Protokoll

Beginn: *24.06.2009*

24.06.09 Datenstruktur überlegt
- `DrawingBrush` zur Speicherung eines Screenshots des zu speichernden Zustands
- Achsenbelegung des `DiagramDataModel`s (`Dictionary<DropAreaType, ObservableCollection<HierarchicalOLAPObject>> Axis`)
- Skalierungsfaktoren `ScaleX` und `ScaleY`
- Verschiebungsfaktoren `TranslateX` und `TranslateY`

Datenstruktur soweit umgesetzt.

24.06.09 `Diagram` um Region *StateStorageHandling* erweitert
- Liste `DiagramStateListProperty` enthält alle gespeicherten States
- `AddState()` fügt aktuellen Status des Diagramms als neuen `DiagramState` in `DiagramStateListProperty` ein
- `RemoveState(DiagramState)` und `RemoveStateAt(int)` entfernen `DiagramStates` aus der Liste
- Diagramm wiederherstellen geschieht durch `SetCurrentDiagramToState(DiagramState)` und `SetCurrentDiagramToStateAt(int)`
  - `DiagramDataModel` um neuen Konstruktor erweitert, der die Achsenbelegung komplett einlesen kann
  - *Scale-* und *Translate*-Werte des Diagramms gesetzt
  - *DataContext* des aktuellen Diagramms auf das neu erzeugte `DiagramDataModel` gesetzt

30.6.09
- Aktuelle Diagrammansicht wird in einem VisualBrush gespeichert (Umweg über XPS-Writer, da das VisualBrush sonst weiterhin aktualisiert wird)

Verantwortliche(r): Konstantin, Jutta

Abbildung 6.10: Beispiel für ein Statusdokument

Tabellenkalkulation

Um einen Überblick über die einzelnen Iterationen und die in diesen umzusetzenden Features zu erhalten, wurde ein Iterationsplan erstellt, wie er in [OW08] vorgestellt wird. Darüber hinaus wurde mit Hilfe eines Tabellenkalkulationsprogramms eine Tabelle angelegt, die die Verfügbarkeit der einzelnen Projektmitglieder tagesgenau, nach Iterationen geordnet, darstellte. In die Berechnung der Verfügbarkeit flossen Urlaube, krankheitsbedingte Ausfälle, Sonderbelastungen und die unterschiedlichen rollenspezifischen Belastungen der Mitglieder ein. Diese Verfügbarkeitstabelle diente dem Projektmanagement dazu, einzelne Projektmitglieder gezielt einzuplanen und die tatsächlich pro Iteration zur Verfügung stehenden Mannstunden festzustellen, um darauf wiederum die Iterationsplanung aufzubauen.

Für die besonders kritischen letzten Iterationen der Implementierungszeit wurde die Tabellenkalkulation zusätzlich als Hilfsmittel zum Risikomanagement eingesetzt. Pro Arbeitspaket zeigte diese Tabelle die für die Fertigstellung des Arbeitspaketes geplante Zeit, die tatsächlich bisher benötigte Zeit und den geschätzten Restaufwand. In einer separaten Tabellenspalte wurde die Abweichung zwischen dem geplanten und zu erwartenden Zeitaufwand berechnet und daraus das Verhalten des Zeitaufwands

bezüglich der kompletten Iteration ermittelt. Besonders kritische Arbeitspakete konnten so schnell erkannt und ggfs. durch entsprechende Maßnahmen stabilisiert bzw. entschärft werden.

6.3 Struktur der Projektgruppe

Über die Laufzeit von einem Jahr der Projektgruppe ergeben sich vielfältige Aufgaben, die erledigt werden müssen. Um dem nachkommen zu können, wurden diese Aufgaben zu Beginn der Projektgruppe identifiziert und in verschiedene Aufgabenbereiche unterteilt. Diesen Aufgabenbereichen wurden dann Rollen zugeordnet und die Mitglieder der Projektgruppe haben sich auf diese Rollen verteilt. Im Folgenden werden die Rollen und deren Aufgabenbereiche vorgestellt.

Projektmanager/Gruppensprecher Die Rolle des Projektmanagers /Gruppensprechers wurde, wie in Kapitel 6.1.2 begründet, doppelt besetzt. Die Aufgaben, die die Rolleninhaber wahrzunehmen haben, sind:

Langfristiges Projektmanagement Zu Beginn muss ein Vorgehensmodell für die Software-Entwicklung ausgewählt werden, das konsequent und mit Hilfe von Werkzeugen durchgesetzt wird.

Werkzeuge zum Projektmanagement Um das Projekt strukturiert und sinnvoll zu verwalten, müssen Werkzeuge wie z. B. *Microsoft Project* oder *Microsoft Excel* (aber auch andere) verwendet werden.

Kommunikation zwischen Betreuern und Gruppe Als Gruppensprecher-in muss der/die Rolleninhaber-in die Kommunikation mit den Betreuern regeln. Hierzu gehört auch das Weitertragen von Problemen, wenn das gruppeninterne Konfliktmanagement nicht funktioniert.
Ferner sind die Projektmanager natürlich für den Kontakt mit dem Kunden, in diesem Fall repräsentiert durch die Betreuer, zuständig und müssen Anforderungsänderungen und Entscheidungen an die Gruppe weiterleiten.

Wöchentliche Sitzung Neben der Planung der wöchentlichen Sitzungen übernimmt eine-r der Projektleiter-innen jeweils die Leitung der aktuellen Sitzung. In dieser Sitzung wird der Fortschritt des Entwicklerteams in Bezug zum Gesamtprojekt gesetzt und Probleme besprochen.

Iterationsplanung Die aus Kundenrücksprachen hervorgehenden Anforderungen müssen zu Arbeitspaketen heruntergebrochen werden. Diese Kundenrücksprache geschieht immer im Orientierungsabschnitt einer Iteration und beeinflusst die Planung der nächsten Iteration. Auch diese Planung ist Aufgabe der Projektmanager.

Soziale Gruppentreffen Um die Gruppenarbeit und die vollständige Integration aller Teammitglieder in die Gruppe zu unterstützen, sorgen die Projektmanager ferner für verschiedene soziale, außeruniversitäre Aktivitäten. Dies kann ein Grillen wie eine sportliche Aktivität beinhalten.

Coach/Chef Programmierer: Der Coach/Chef Programmierer trägt die Verantwortung für die Einhaltung des Entwicklungsmodells und das funktionierende Zusammenspiel aller Entwickler. Als solcher hat er stets einen guten Überblick über den bisher produzierten Code und kann die anderen Mitglieder der Projektgruppe nach ihren Fähigkeiten optimal einsetzen. Die Koordination von Entwicklungs- und Architekturentscheidungen gehört ebenso zu seinen Aufgaben wie die Zusammenarbeit mit den Projektmanagern bei der Iterationsplanung.

Qualitätsbeauftragter Die Rolle des Qualitätsbeauftragten beinhaltet die Überprüfung der Einhaltung von Richtlinien bei der Entwicklung. Hierzu hat der Beauftragte zunächst die Coding Richtlinien definiert, und anschließend darauf geachtet, dass der Quellcode den definierten Richtlinien genügt

Als weitere Aufgabe achtet der Qualitätsbeauftragte auch darauf, dass jeder Entwicklungsschritt stets vollständig und gut dokumentiert ist. Als Hilfsmittel dienen ihm hierbei verschiedene Werkzeuge, sowie die Verteilung und Auswertung von Reviews über die verfassten Dokumente. Weitere Aufgaben und die Dokumentation der Durchführung des Qualitätsmanagement sind in Kapitel 12 zu finden.

Dokumentenbeauftragter Für die gesamte Dokumentation des Projektverlaufs wird das Textsatzsystem LaTeX verwendet. Der Dokumentenbeauftragte ist verantwortlich für das Erstellen und Pflegen von Vorlagen für die Dokumente, die im Laufe der Zeit erstellt werden müssen. Dies umfasst eine Vorlage für die Protokolle, in denen die Ergebnisse der wöchentlichen Sitzungen festgehalten werden können, ebenso wie Vorlagen für den Zwischen- und den Endbericht, welche den Entwicklungsstatus des Produktes vollständig dokumentieren.

Im Zusammenhang mit den beiden zuletzt genannten Dokumenten umfasst der Aufgabenbereich des Dokumentenbeauftragten das Anlegen und Pflegen verschiedener Verzeichnisse, wie etwa das Glossar oder das Stichwortverzeichnis. Da nicht jeder Projektgruppenteilnehmer mit dem Gebrauch dieser Verzeichnisse vertraut ist, muss außerdem eine Anleitung für ihren Gebrauch verfasst werden, beispielsweise über das Anlegen eines neuen Akronyms und das Erstellen eines Verweises, der sich darauf bezieht.

Des Weiteren stellt der Dokumentenbeauftragte für die übrigen Teilnehmer einen Ansprechpartner in allen LaTeX-bezogenen Fragen dar und ist für das Lösen diesbezüglicher Probleme zuständig.

Um den Dokumenten ein möglichst einheitliches Aussehen zu geben, ist es notwendig, das Schriftbild unterschiedlicher Autoren aneinander anzugleichen. Das Erstellen einer Guideline mit entsprechenden typographischen Vorgaben zählt ebenfalls zu den Aufgaben des Dokumentenbeauftragten.

Dokumentationsbeauftragter Für die Koordinierung der Dokumentation der Entwicklung ist der Dokumentationsbeauftragte zuständig. In Absprache mit dem Dokumenten- und Qualitätsbeauftragten trägt er Sorge dafür, dass stets aller Quellcode hinreichend dokumentiert wird und die entsprechenden Entwurfsdokumente und Berichte verfasst werden. Auch die Planung der Inhalte der abschließenden Berichte und die letztendliche Überprüfung dieser fiel mit in seinen Aufgabenbereich.

Des Weiteren war der Dokumentationsbeauftragte für die Auswahl und das Testen der Dokumentationswerkzeuge und das Dokumentieren der Anforderungen verantwortlich.

Administrator Die Aufgaben des Administrators umfassen zum einen Installationstätigkeiten (Installation von Betriebssystem und Tools auf dem Multitouch-Rechner), zum anderen auch die Einrichtung von Versionskontrollsystemen (SVN), Mailing-Listen und Wikis. Außerdem sorgt der Administrator für eine Internetrepräsentation der Projektgruppe und stellt diese unter einer aussagekräftigen Domain zur Verfügung.

Datenmodell-Experte Der Datenmodell-Experte setzt die zu verwendende multidimensionale Datenbank in Abstimmung mit dem Administrator und dem Coach auf. Er kennt sich mit dem verwendeten System aus und kann den weiteren Teammitgliedern bei Fragen zur Anbindung und Aufbau der *Online Analytical Processing*-Datenbank helfen. Er kennt die Inhalte der verwendeten Datenbanken, um etwaigen Problemen vorbeugen zu können.

Für die Anbindung der *MUSTANG*-Datenbank hält er Kontakt zu den Verantwortlichen um bei potenziellen Problemen diese mit deren Unterstützung lösen zu können.

Der Datenmodell-Experte hat Einfluss auf die Entwicklung des verwendeten Datenmodells, da dieses eng an die Strukturen der Datenbanken angelehnt ist. Er kennt sich bestens mit der Implementierung der Schnittstellen zu den Datenbanken aus und kann hierbei, falls nötig, Hilfestellung leisten.

Usability Engineer Der Usability Engineer ist dafür zuständig, darauf zu achten, dass bei der Entwicklung des Systems die Aspekte der Gebrauchstauglichkeit berücksichtigt werden. Dafür gibt er eine Einführung in die Grundlagen von Usability mit besonderem Bezug zu Touch- und Gestensystemen (siehe Abschnitt 5). Des Weiteren hat er die Usability-Richtlinien zusammengestellt, die für das

TaP Projekt geeignet sind und später für Usability-Reviews benutzt werden können. Um frühzeitig die Brauchbarkeit zu testen, muss zu Anfang ein Papierprototyp entwickelt werden. Um die Qualität der Nutzerinteraktion des Systems zu testen, werden zwei Evaluationen durchgeführt. Außerdem sollen die einzelnen Komponenten mittels heuristischen Reviews überprüft werden.

Kapitel 7

Anforderungsanalyse

Dieses Kapitel widmet sich der Anforderungsanalyse. Hierzu wird anfangs ein Leitszenario beschrieben, welches den möglichen Umgang mit *TaP* zeigt. Daraufhin werden die funktionalen Anforderungen, gefolgt von den nicht-funktionalen Anforderungen, aufgelistet und erläutert. Abschließend werden Anwendungsfälle in Form von Tabellen dargestellt.

7.1 Zielgruppe

Als Anwender kommt insbesondere folgende Personengruppe in Frage:

> Analysten mit Hintergrundwissen über die Anwendungsdomäne, die den Datenbestand explorieren wollen. Weiterführendes Wissen über komplexe Auswertungsverfahren sind nicht erforderlich.

7.2 Leitszenario

Die fiktive Firma „Fahrrad Flömann“ fertigt und verkauft weltweit Fahrradteile, -zubehör, -bekleidung und Fahrräder. Diese Produkte werden über das Internet und an Vertriebspartner verkauft. Die Firma hat mehrere regionale Vertriebsteams in unterschiedlichen Ländern.

„Fahrrad Flömann“ will visuelle Analysen durchführen, um Zusammenhänge in ihren großen, multidimensionalen Datenmengen zu erkennen. Visuelle Analysen weisen ein hohes Maß an Interaktivität und Dynamik auf. Daher wollen die Analysten der Firma dafür einen Multitouch-Tisch verwenden, der einen intuitiven, *begreifbaren* Zugang zu den Daten ermöglicht.

Die Geschäftsleiterin Floria Hesselring möchte ihre Verkaufszahlen der letzten Jahre analysieren lassen. Hierfür beauftragt sie den Analysten Steff Han, welcher mit

Hilfe einer multitouch-fähigen Analysesoftware die Daten der Firma untersuchen soll. Zunächst betrachtet Steff Han den Bruttogewinn über die letzten Geschäftsjahre. Anschließend möchte er einen Vergleich zu den Verkaufszahlen herstellen. Hierbei wechselt er zwischen unterschiedlichen Visualisierungsformen. Die Geschäftsleiterin und der Analyst stellen fest, dass der Verlauf der Daten eine Anomalie im Bereich des Jahres 2002 aufweist. Der Analyst schlägt vor, die Daten des Jahres 2002 genauer zu betrachten, indem er auf Monatsebene wechselt. Bei der Betrachtung der neu angezeigten Daten vermuten sie, dass eine genauere Analyse auf Tagesebene anschaulicher ist. Auf Grund der hohen Menge an Daten ist die folgende Darstellung jedoch sehr ungenau. Der Analyst verfeinert also die Darstellung, um den genauen Tag der Anomalie ermitteln zu können. Die Geschäftsleiterin und der Analyst erkennen das genaue Datum der Anomalie und beschließen, diese Darstellung für einen späteren Einsatz zu speichern.

Um feststellen zu können, ob sich diese Anomalie auf eine bestimmte Produktkategorie bezieht, wollen sie sich den Verlauf des Bruttoeinkommens, verteilt auf die Produktkategorien, über die Zeit anschauen. Steff Han wählt hierfür eine über die Zeit animierte Darstellung, sodass immer nur ein bestimmter Zeitabschnitt angezeigt wird und dieser automatisch wechselt. Auch hier stellen Floria Hesselring und Steff Han eine Anomalie im Jahr 2002 fest. Durch die Ansicht detaillierterer Zeitabschnitte stellen sie fest, dass am 1. März 2002 der Bruttogewinn für die Produktkategorie *Fahrräder* weit im negativen Bereich liegt. Die Geschäftsleiterin kann auf Grund ihres Hintergrundwissens das Datum einer Buchung zu einer großen Bestellmenge an Fahrrädern zuordnen, welche die Firma zum Weiterverkauf bestellt hat.

7.3 Funktionale Anforderungen

Wie im Leitszenario beschrieben ist, benötigt das Programm eine Vielzahl von bestimmten Funktionen. Die genauen Anforderungen an diese Funktionen sind in diesem Abschnitt festgehalten.

Allgemeine Anforderungen

Der Analyst Steff Han benötigt eine gestenbasierte Anwendung mit einer größtmöglichen Darstellungsfläche.

[FA.1] Gesten-Bedienung: Der Analyst soll die Möglichkeit haben so direkt wie möglich mit dem Programm bzw. dessen Bestandteilen zu interagieren. Die Bedienung soll daher sowohl auf Single- als auch auf Multitouch-Gesten ausgelegt sein.

[FA.2] Vollbildmodus: Um den maximal möglichen Platz für die Analyse ausnutzen zu können, soll die Benutzungsoberfläche des Programms stets maximiert sein.

Anforderungen an die Visualisierung

Es werden unterschiedliche Arten von Darstellungsformen für Daten in einem Diagramm benötigt. Hierbei ist es erforderlich, dass der Analyst Daten verfeinern und in diesen navigieren kann. Das Programm muss dauerhaft Überblick über die angezeigten Daten liefern.

[FA.3] Punktdiagramm: Mit dem Programm sollen multidimensionale Daten in Form von Diagrammen dargestellt werden. Da Punktdiagramme allgemein bekannt und einfach zu interpretieren sind, soll die Visualisierung in dieser Form geschehen. Ein Punktdiagramm besitzt Achsen (X-Achse, Y-Achse), die jeweils eine bestimmte Einteilung haben, und Beschriftungen, welche die Daten näher beschreiben.

[FA.4] Skalierung der Größe des Diagramminhaltes: Der Inhalt eines Diagramms soll beliebig vergrößert und verkleinert werden können, um auf Wunsch eine detaillierte Ansicht der Daten zu erhalten.

[FA.5] Verschieben des Diagramminhaltes: Nach der Vergrößerung eines Diagramms soll es möglich sein, die Umgebung des sichtbaren Ausschnitts genauer explorieren zu können. Anstatt die Vergrößerung wieder rückgängig zu machen und in einen neuen Bereich hinein zu zoomen, soll es möglich sein, den sichtbaren Diagrammausschnitt zu verschieben.

[FA.6] Datenverfeinerung: Multidimensionale Datenbanken erlauben den Zugriff auf die Daten in unterschiedlichen Hierarchiestufen über die Funktionen *DrillDown* und *RollUp* (s. Abschnitt 2.3). Während der Analyse soll ein einfacher Wechsel zwischen diesen Aggregationsstufen möglich sein. Eine *DrillDown*-Operation in der Datenbank entspricht einer Verfeinerung der Granularität des Diagramminhalts, während eine *RollUp*-Operation einer Vergröberung entspricht.

Die Verfeinerung soll auf zwei unterschiedliche Weisen umgesetzt werden. Es soll zwischen einem *lokalen* und einem *globalen DrillDown* unterschieden werden.

Globaler DrillDown Die Verfeinerung geschieht auf der gesamten Ebene einer Achse.

Lokaler DrillDown Die Verfeinerung wird an einem bestimmten Punkt im Diagramm ausgelöst. Das verfeinerte Diagramm zeigt dann nur diejenigen Werte an, aus welchen sich der ausgewählte Punkt zusammensetzt.

[FA.7] Größenausprägung: Um mit einem einzigen Diagramm mehr als zwei Größen in Beziehung zu setzen, soll es Punkte in verschiedenen Größen anzeigen können. Ein Punktdiagramm mit dieser Ausprägung wird *Bubble-Chart* genannt. Je kleiner ein Punkt dargestellt wird, desto kleiner ist der Wert den er repräsentiert. Je größer ein Punkt dargestellt wird, desto größer ist dieser Wert.

[FA.8] Farbausprägung: Um mit einem einzigen Diagramm mehr als zwei Größen in Beziehung zu setzen, soll es Punkte in verschiedenen Farben anzeigen können. Verschiedene Farben bzw. Farbabstufungen repräsentieren unterschiedliche Werte.

[FA.9] Animation: Um sich z. B. die zeitlichen Veränderungen der ausgewählten Größen anzuzeigen, soll das Diagramm in der Lage sein, seinen Inhalt über eine Dimension zu animieren. Diese Dimension kann z. B. eine zeitliche Dimension sein. Die Animation soll als Ganzes abgespielt werden können, wobei es ferner möglich sein soll, den Ablauf zu pausieren. Zusätzlich zu einer automatisierten Animation sollen die Einzelbilder manuell durchlaufen werden können, um Auffälligkeiten genauer betrachten zu können.

[FA.10] Legende: Damit dem Analysten bekannt ist, welche Farbe bzw. welche Punktgröße welchem Wert entspricht, muss jedem Diagramm eine Legende zugeordnet sein, in welcher ebendiese Zuordnung vermerkt wird.

Multitouch-Framework

Zur Anbindung der Multitouch-Eingaben an die Software wird eine gemeinsame Schnittstelle benötigt.

[FA.11] TUIO-Anbindung: Damit die Anwendung unabhängig von der zur Bloberkennung (siehe Abschnitt 3.2.2) eingesetzten Software ist, sollen die Benutzerinteraktionen in Form des *TUIO*-Protokolls vom Multitouch-Tisch zur Anwendung übertragen werden.

Anforderungen an die Datenbank

Die zu analysierenden Daten der Firma werden in einer multidimensionalen Datenbank gehalten.

[FA.12] Multidimensionale Datenbank: Das Programm soll in der Lage sein, Daten aus einer multidimensionalen Datenbank auszulesen. Eine Speicherung von Daten ist nicht vorgesehen.

[FA.13] Austauschbarkeit der Datenbank: Das verwendete Datenbanksystem soll durch eine Abstrakionsschicht von dem Rest des Programms getrennt werden, sodass es ggf. ausgetauscht werden kann. Die Abstraktionsschicht soll nicht abhängig von einem bestimmten Datenbanksystem sein. Programmintern soll der Umgang mit der Abstraktionsschicht einheitlich sein: d. h. Unterschiede zwischen den Datenbanksystemen müssen ausgeglichen werden.

[FA.14] MUSTANG-Anbindung: *MUSTANG* soll als Datenquelle unterstützt werden.

Anforderungen an die Datenauswahl

Wie das Leitszenario gezeigt hat, muss es eine Möglichkeit zur Datenauswahl geben.

[FA.15] Grafische Darstellung hierarchischer Daten: Da eine multidimensionale Datenbank genutzt wird, muss es ein Menü zur Auswahl geben, welches in der Lage ist, die hierarchische Struktur darzustellen. Mit diesem Menü muss es möglich sein, sowohl Kennzahlen als auch Dimensionen auswählen zu können. Da häufig nicht alle Kennzahlen und Dimensionen kompatibel zueinander sind, muss der Benutzer auf entsprechende Kompatibilitäten hingewiesen werden. Damit der Benutzer einfacher mit dem Menü arbeiten kann, muss seine aktuelle Position im Menü ebenfalls erkennbar sein.

7.4 Nicht-Funktionale Anforderungen

In diesem Abschnitt werden die Eigenschaften des Programms beschrieben.

[NFA.1] Bedienbarkeit: Um die Bedienbarkeit des Programmes und dessen Bestandteilen möglichst einfach zu halten, sollen die üblichen Usability-Anforderungen eingehalten werden: Dazu gehört unter anderem eine intuitive und reaktionsschnelle Benutzerführung. Auch soll der aktuelle Systemzustand zu jedem Zeitpunkt offensichtlich sein. Darüber hinaus soll dem Nutzer zu jeder Zeit klar sein, welche Aktionen er ausführen kann. Der Programmfluss soll ihn unterstützen und ihm, sofern dies möglich ist, kognitive Last abnehmen.

[NFA.2] Reaktion der Gesten: Damit dem Analysten die Interaktion mit dem Programm bzw. dessen Bestandteilen so direkt wie möglich erscheint, sollen die Reaktionen jeder einzelnen Geste (siehe Anforderung [FA.1]) unmittelbar und ohne Wartezeiten erfolgen, also zeitgleich mit der Ausführung der jeweiligen Geste.

[NFA.3] Natürliche Gesten: Um dem Analysten die Interaktion mit dem Programm bzw. dessen Bestandteilen so intuitiv wie möglich zu gestalten, sollen als Gesten (siehe Anforderung [FA.1]) möglichst natürliche Gesten verwendet werden. Unter natürlichen Gesten werden Gesten verstanden, welche intuitiv sind.

[NFA.4] Form des Menüs: Da das Programm auf einem Multitouch-Tisch betrieben wird, sollte das Menü aus Anforderung [FA.15] für diese Art der Steuerung optimiert sein. Eine Anpassung des Menüs an eine Hand sollte in Betracht gezogen werden. Insgesamt muss das Menü übersichtlich gestaltet werden und die Auswahl von Daten sollte schnell und einfach möglich sein.

[NFA.5] Darstellung hierarchischer Daten im Menü: Damit der Analyst die Struktur der multidimensionalen Datenbank im Menü aus Anforderung [FA.15] wiederfindet, sollte die hierarchische Struktur deutlich erkennbar sein.

[NFA.6] Reaktion der multidimensionalen Datenbank: Um dem Benutzer eine möglichst effiziente Arbeit zu ermöglichen, soll die Antwortzeit einer Datenbankabfrage (siehe Anforderung [FA.12]) möglichst gering gehalten werden. Der Benutzer muss darauf hingewiesen werden, dass eine Datenbankabfrage stattfindet.

[NFA.7] Sprache: Damit das Programm auch von Analysten aus anderen Ländern bedient werden kann, sollen alle Texte und Meldungen in der englischen Sprache verfasst werden.

[NFA.8] Implementierung: Während einer Evaluation von verschiedenen Visual Programming ToolKits stellte sich heraus, dass *WPF* am geeignetsten für die Implementierung der grafischen Oberfläche ist (s. Abschnitt 4.1). Hieraus folgt, dass C# als Entwicklungssprache genutzt werden soll. Um die Arbeit im Team zu vereinfachen, soll als Entwicklungswerkzeug *Visual Studio 2008* mit dem *Visual Studio Team System 2008* verwendet werden.

[NFA.9] Beschränkungen des Diagramms: Es muss sichergestellt werden, dass auf die Achsen des Diagrammes aus Anforderung [FA.3] jeweils nur eine Dimension gelegt werden kann. Kennzahlen können mehrere eingefügt werden. Diese werden dann durch unterschiedliche Farben dargestellt und die Farben-Ausprägungsachse (siehe Anforderung [FA.8]) kann nicht mehr belegt werden. Weiterhin muss sichergestellt werden, dass nur Kennzahlen auf die Größen-Ausprägung (siehe Anforderung [FA.8]) gezogen werden dürfen. Die unterschiedlich großen Punkte dürfen sich jedoch überschneiden.

[NFA.10] Diagrammverfeinerung: Während der Vergrößerung und Verkleinerung des Diagramminhaltes (siehe Anforderung [FA.4]) verändert sich nicht die Größe der einzelnen Punkte, sondern nur deren Abstände zueinander.

7.5 Anwendungsfälle

Ein Anwendungsfall beschreibt, mit welchen Aktionen Analysten ein vorgegebenes Ziel erreichen können. Im Folgenden werden die Anwendungsfälle der benutzten Software beschrieben.

[UC.1] Auswahl eines Dimensionselements

Beschreibung	Der Benutzer öffnet das Menü und wählt in der vorliegenden hierarchischen Struktur ein Element einer Dimension aus. Anschließend muss mit einer *Drag and Drop*-Geste das Element auf eine der Drop-Zonen[1] im Diagramm gezogen werden.
Akteure	Analyst
Beinhaltet	*Drag and Drop*-Geste
Auslöser	–
Vorbedingungen	1. Das Programm ist gestartet. 2. Das Menü wird angezeigt. 3. Die auszuwählende Dimension ist kompatibel zu den bereits im Diagramm verwendeten Objekten.
Nachbedingungen	Der Akteur hat eine Dimension ausgewählt
Ausnahmen	–
Erweiterungen	–

[1] bestimmter Bereich zur Zuordnung eines Elementes zu einer Achse oder Ausprägung.

Standardablauf	1. Der Akteur berührt den Basisknopf des Menüs, um die erste Ebene des Menüs anzeigen zu lassen. 2. Der Akteur berührt das Element *Dimensions* in der ersten Ebene des Menüs, um sich die verfügbaren Dimensionen in der nächsten Ebene des Menüs anzeigen zu lassen. 3. Der Akteur berührt ein Element der Dimensionsebene, um sich die zugehörigen Level in der nächsten Ebene anzeigen zu lassen. 4. Der Akteur wählt eines der angezeigten Levelelemente aus und zieht es mit der *Drag and Drop*-Geste auf eine der Drop-Zonen des Diagramms.

[UC.2] Auswahl einer Kennzahl

Beschreibung	Der Benutzer öffnet das Menü und wählt in der vorliegenden hierarchischen Struktur eine Kennzahl aus. Anschließend muss mit einer *Drag and Drop*-Geste das Element auf eine der *Drop-Zones* im Diagramm gezogen werden.
Akteure	Analyst
Beinhaltet	*Drag and Drop*-Geste
Auslöser	–
Vorbedingungen	1. Das Programm ist gestartet. 2. Das Menü wird angezeigt. 3. Die auszuwählende Kennzahl ist kompatibel zu den bereits im Diagramm verwendeten Objekten.
Nachbedingungen	Der Akteur hat eine Kennzahl ausgewählt
Ausnahmen	–
Erweiterungen	–
Standardablauf	1. Der Akteur berührt den Basisknopf des Menüs, um die erste Ebene des Menüs anzeigen zu lassen. 2. Der Akteur berührt das Element *Measures* in der ersten Ebene des Menüs, um sich die verfügbaren Kennzahl-Gruppen in der nächsten Ebene des Menüs anzeigen zu lassen. 3. Der Akteur wählt eine der angezeigten Kennzahlen aus und zieht diese mit der *Drag and Drop*-Geste auf eine der Drop-Zonen des Diagramms.

Alternativablauf	1. Da nicht in allen Datenbanken zwingend Kennzahl-Gruppen vorhanden sind, werden direkt nach dem Berühren des Elements *Measures* die verfügbaren Kennzahlen angezeigt.

[UC.3] Ausführung einer *Pinch*-Geste

Beschreibung	Der Akteur führt eine *Pinch*-Geste in einem bestimmten Bereich des Multitouch-Tisches aus.
Akteure	Analyst
Beinhaltet	*Move*-Geste
Auslöser	–
Vorbedingungen	–
Nachbedingungen	Der Akteur hat eine *Pinch*-Geste durchgeführt.
Ausnahmen	–
Erweiterungen	–
Standardablauf	1. Der Akteur berührt mit mindestens zwei Fingern einen festgelegten Bereich. 2. Er verringert die Distanz zwischen beiden Fingern.
Alternativablauf	-

[UC.4] Ausführung einer *Spread*-Geste

Beschreibung	Der Akteur führt eine *Spread*-Geste in einem bestimmten Bereich des Multitouch-Tisches aus.
Akteure	Analyst
Beinhaltet	*Move*-Geste
Auslöser	–
Vorbedingungen	–
Nachbedingungen	Der Akteur hat eine *Spread*-Geste durchgeführt.
Ausnahmen	–
Erweiterungen	–
Standardablauf	1. Der Akteur berührt mit mindestens zwei Fingern einen festgelegten Bereich. 2. Er vergrößert die Distanz zwischen beiden Fingern.
Alternativablauf	-

[UC.5] Ausführung einer *Rotate*-Geste

Beschreibung	Der Akteur führt eine *Rotate*-Geste in einem bestimmten Bereich des Multitouch-Tisches aus.
Akteure	Analyst
Beinhaltet	*Move*-Geste
Auslöser	–
Vorbedingungen	–
Nachbedingungen	Der Akteur hat eine *Rotate*-Geste durchgeführt.
Ausnahmen	–
Erweiterungen	–
Standardablauf	1. Der Akteur berührt mit mindestens zwei Fingern einen festgelegten Bereich. 2. Er dreht alle Finger in eine Richtung relativ zum Mittelpunkt aller Finger.
Alternativablauf	-

[UC.6] Ausführung einer *Drag and Drop*-Geste

Beschreibung	Der Akteur führt eine *Drag and Drop*-Geste in einem bestimmten Bereich des Multitouch-Tisches aus.
Akteure	Analyst
Beinhaltet	*Move*-Geste
Auslöser	–
Vorbedingungen	–
Nachbedingungen	Der Akteur hat eine *Drag and Drop*-Geste durchgeführt.
Ausnahmen	–
Erweiterungen	–
Standardablauf	1. Der Akteur berührt und hält mit mindestens einem Finger ein Objekt, welches für eine *Drag and Drop*-Geste benutzt werden kann. 2. Er bewegt den/die Finger auf dem Tisch zu dem Bereich, wo das Objekt abgelegt werden soll. 3. Bei Erreichen des Zielbereiches hebt er die Finger vom Tisch.
Alternativablauf	-

[UC.7] Ausführung einer *Move*-Geste

Beschreibung	Der Akteur führt eine *Move*-Geste in einem bestimmten Bereich des Multitouch-Tisches aus.
Akteure	Analyst
Beinhaltet	–
Auslöser	–
Vorbedingungen	–
Nachbedingungen	Der Akteur hat eine *Move*-Geste durchgeführt.
Ausnahmen	–
Erweiterungen	–
Standardablauf	1. Der Akteur berührt einen Bereich auf dem Multitouch-Tisch. 2. Er bewegt den/die Finger auf dem Tisch.
Alternativablauf	-

[UC.8] Ausführung eines *RollUp*

Beschreibung	Der Akteur führt einen *RollUp* auf der X-, Y- oder Animations-Achse aus. Dieser führt dazu, dass die angezeigte und verwendete Ebene auf dieser Achse durch die nächsthöhere Ebene in der *Hierarchie* der Datenbank ersetzt wird. Durch den Austausch wird das Diagramm anschließend direkt mit den neuen Daten erzeugt.
Akteure	Analyst
Beinhaltet	*Pinch*-Geste
Auslöser	–
Vorbedingungen	Ein Dimensionselement wurde für die betreffende Achse ausgewählt.
Nachbedingungen	Das Diagramm wurde mit der neuen Ebene erstellt. Alle anderen Diagramm-Belegungen entsprechen denen des Ausgangszustands.
Ausnahmen	–
Erweiterungen	–
Standardablauf	1. Der Akteur berührt und hält mit mindestens zwei Fingern, in Richtung der längeren Seite, die X-, Y- oder Animations-Achse. 2. Er führt eine *Pinch*-Geste aus. 3. Das Programm beschränkt den *RollUp* auf eine einzige Ausführung der *Pinch*-Geste. 4. Es ersetzt die Ebene durch die Nächsthöhere. 5. Das Programm erzeugt das Diagramm mit der neuen Ebene.
Alternativablauf	-

[UC.9] Ausführung eines globalen *DrillDown*

Beschreibung	Der Akteur führt einen globalen *DrillDown* auf der X-, Y- oder Animations-Achse aus. Dieser führt dazu, dass die angezeigte und verwendete Ebene auf dieser Achse durch die nächst niedrigerere Ebene in der *Hierarchie* der Datenbank ersetzt wird. Durch den Austausch wird das Diagramm anschließend direkt mit den neuen Daten erzeugt.
Akteure	Analyst
Beinhaltet	*Spread*-Geste
Auslöser	–
Vorbedingungen	Ein Dimensionselement wurde für die betreffende Achse ausgewählt.
Nachbedingungen	Das Diagramm wurde mit der neuen Ebene erstellt.
Ausnahmen	–
Erweiterungen	–
Standardablauf	1. Der Akteur berührt und hält mit mindestens zwei Fingern, in Richtung der längeren Seite, die X-, Y- oder Animations-Achse. 2. Er führt eine *Spread*-Geste aus. 3. Das Programm beschränkt den globalen *DrillDown* auf eine einzige Ausführung der *Spread*-Geste. 4. Es ersetzt die Ebene durch die Nächstniedrigere. 5. Das Programm erzeugt das Diagramm mit der neuen Ebene.
Alternativablauf	-

[UC.10] Ausführung eines lokalen *DrillDown*

Beschreibung	Der Akteur führt einen lokalen *DrillDown* auf einem *Knoten* der X-, Y- oder Animations-Achse aus. Dieser führt dazu, dass die angezeigte und verwendete Ebene auf dieser Achse durch die Kinder des *Knoten* (Beschriftungsobjekt auf der Achse) in der *Hierarchie* der Datenbank ersetzt wird. Durch den Austausch wird das Diagramm anschließend direkt mit den neuen Daten erzeugt.
Akteure	Analyst
Beinhaltet	–
Auslöser	–
Vorbedingungen	Ein Dimensionselement wurde für die betreffende Achse ausgewählt.
Nachbedingungen	Das Diagramm wurde mit der neuen Ebene erstellt.
Ausnahmen	–
Erweiterungen	–

Standardablauf	1. Der Akteur berührt mit mindestens einem Finger einen *Knoten*. 2. Er hebt den Finger von dem *Knoten*. 3. Das Programm ersetzt die angezeigte und verwendete Ebene durch die Kinder des *Knoten*s. 4. Es erzeugt das Diagramm mit der neuen Ebene neu.
Alternativablauf	-

[UC.11] Erstellung eines neuen Diagramms

Beschreibung	Der Akteur möchte ein neues Diagramm erstellen. Dieses erfolgt über die Aktivierung einer Schicht, auf welcher frei gezeichnet werden kann. Zeichnet der Akteur ein Rechteck, wird in der entsprechenden Größe und an der entsprechenden Position ein neues Diagramm erstellt.
Akteure	Analyst
Beinhaltet	*Move*-Geste
Auslöser	–
Vorbedingungen	–
Nachbedingungen	Ein neues Diagramm wurde erstellt.
Ausnahmen	–
Erweiterungen	–
Standardablauf	1. Der Akteur berührt eine der Flächen in den Ecken der Anwendung, um die Schicht für den Zeichnungsvorgang zu aktivieren. 2. Das Programm zeigt eine halbtransparente weiße Schicht über der gesamten Anwendungsfläche an. Dieses bestätigt die Aktivierung. 3. Der Akteur beginnt die Zeichnung des Rechtecks mit der oberen linken Ecke des Rechtecks mit einem Finger. 4. Das Programm zeigt die bereits zurückgelegte Strecke, in Form eines Pfades, an. 5. Die Zeichnung des Rechtecks erfolgt linksdrehend bis die obere linke Ecke wieder erreicht wurde. 6. Der Akteur hebt seinen Finger von der Oberfläche. 7. Das Programm zeigt das Diagramm an der Stelle der Zeichnung, entsprechend der Größe des Rechtecks, an.

Alternativablauf	1. Falls der Akteur die Zeichnung des Rechtecks nicht korrekt durchgeführt hat, wird kein Diagramm erstellt und der Pfad der zurückgelegten Strecke wird ausgeblendet.

[UC.12] Verschieben des Diagramms

Beschreibung	Der Akteur möchte ein Diagramm innerhalb des Arbeitsbereiches an eine andere Position verschieben.
Akteure	Analyst
Beinhaltet	*Move*-Geste
Auslöser	–
Vorbedingungen	–
Nachbedingungen	Das Diagramm wurde an eine andere Position verschoben.
Ausnahmen	–
Erweiterungen	–
Standardablauf	1. Der Akteur berührt mindestens mit einem Finger eine der äußeren Flächen an den Ecken des Diagramms. 2. Er führt eine *Move*-Geste aus. 3. Das Diagramm wird an die Position des Fingers verschoben.
Alternativablauf	–

[UC.13] Verschiebung im Diagramm

Beschreibung	Der Akteur möchte den Bildausschnitt des angezeigten Bereiches im Diagramm verschieben. Der Bildausschnitt ist durch einen Rahmen und die Achsen begrenzt.
Akteure	Analyst
Beinhaltet	*Move*-Geste
Auslöser	–
Vorbedingungen	–
Nachbedingungen	Der angezeigte Bereich im Diagramm wurde verändert.
Ausnahmen	–
Erweiterungen	–

Standardablauf	1. Der Akteur führt eine *Move*-Geste aus. 2. Das Programm verschiebt den angezeigten Bereich des Diagramms in die gleiche Richtung wie die Bewegung des Fingers.
Alternativablauf	–

[UC.14] Verschieben der Arbeitsfläche

Beschreibung	Der Akteur verschiebt mit einem oder mehreren Fingern den angezeigten Bereich der Arbeitsfläche.
Akteure	Analyst
Beinhaltet	*Move*-Geste
Auslöser	–
Vorbedingungen	–
Nachbedingungen	Der angezeigte Bereich auf der Arbeitsfläche wurde verändert.
Ausnahmen	–
Erweiterungen	–
Standardablauf	1. Der Akteur führt eine *Move*-Geste aus. 2. Das Programm verschiebt den angezeigten Bereich der Arbeitsfläche in die gleiche Richtung wie die Bewegung des Fingers. 3. Die beinhalteten Diagramme werden in gleicher Weise verschoben.
Alternativablauf	–

[UC.15] Hinein-Zoomen im Diagramm

Beschreibung	Der Akteur möchte einen Ausschnitt des Diagramms detaillierter betrachten. Hierzu führt er eine *Spread*-Geste auf der Oberfläche des Diagramms durch.
Akteure	Analyst
Beinhaltet	*Spread*-Geste
Auslöser	–
Vorbedingungen	Auswählen einer *Kennzahl*, Auswählen eines Dimensionselements.
Nachbedingungen	Es wird ein detaillierterer Ausschnitt des Diagramms angezeigt.
Ausnahmen	–
Erweiterungen	–

Standardablauf	1. Der Akteur führt eine *Spread*-Geste in dem Anzeigebereiches des Diagramms aus. 2. Das Programm skaliert die Bildfläche des Diagramms hoch, wobei die Größe des Diagrammfensters nicht verändert wird. Die Größe der Diagrammpunkte bleibt konstant.
Alternativablauf	–

[UC.16] Hinaus-Zoomen im Diagramm

Beschreibung	Der Akteur möchte einen gröberen Ausschnitt im Diagramm betrachten. Hierfür führt er eine *Pinch*-Geste auf der Oberfläche des Diagramms durch.
Akteure	Analyst
Beinhaltet	*Pinch*-Geste
Auslöser	–
Vorbedingungen	Auswählen einer *Kennzahl*, Auswählen eines Dimensionselements.
Nachbedingungen	Es wird ein gröberer Ausschnitt des Diagramms angezeigt.
Ausnahmen	–
Erweiterungen	–
Standardablauf	1. Der Akteur führt eine *Pinch*-Geste in dem Anzeigebereiches des Diagramms aus. 2. Das Programm skaliert die Bildfläche des Diagramms herunter, wobei die Größe des Diagrammfensters nicht verändert wird. Die Größe der Diagrammpunkte bleibt hierbei konstant.
Alternativablauf	–

[UC.17] Vergrößern des Diagramms

Beschreibung	Der Akteur vergrößert das Diagramm mit einer *Spread*-Geste auf den Ecken des Diagramms.
Akteure	Analyst
Beinhaltet	*Spread*-Geste
Auslöser	–
Vorbedingungen	–
Nachbedingungen	Das Diagramm wurde vergrößert.
Ausnahmen	–
Erweiterungen	–

Standardablauf	1. Der Akteur berührt zwei gegenüberliegende Ecken des Diagramms. 2. Er führt eine *Spread*-Geste aus. 3. Das Programm skaliert das Diagramm hoch. Die Inhalte des Diagramms passen sich prozentual an die Größe an.
Alternativablauf	–

[UC.18] Verkleinern des Diagramms

Beschreibung	Der Akteur verkleinert das Diagramm mit einer *Pinch*-Geste auf den Ecken des Diagramms.
Akteure	Analyst
Beinhaltet	*Pinch*-Geste
Auslöser	–
Vorbedingungen	–
Nachbedingungen	Die Größe des Diagramms wurde verringert.
Ausnahmen	–
Erweiterungen	–
Standardablauf	1. Der Akteur berührt zwei gegenüberliegende Ecken des Diagramms. 2. Er führt eine *Pinch*-Geste aus. 3. Das Programm skaliert das Diagramm herunter. Die Inhalte des Diagramms passen sich prozentual an die Größe an.
Alternativablauf	–

[UC.19] Ändern der Achsenbelegung

Beschreibung	Der Akteur wechselt die Belegung der Achsen, ohne neue Elemente hinzuzufügen. Hierfür müssen die Drop-Zones angezeigt werden. Der Austausch erfolgt über das Berühren eines Elements und das anschließende Berühren der Drop-Zonen, auf die das Element gelegt werden soll.
Akteure	Analyst
Beinhaltet	–
Auslöser	–
Vorbedingungen	Anzeige der Drop-Zonen, Auswahl eines Dimensionselements, Auswahl einer *Kennzahl*.

Nachbedingungen	Die Achsenbelegung wurde verändert und das Diagramm neu erzeugt.
Ausnahmen	–
Erweiterungen	–
Standardablauf	1. Der Akteur berührt eines der Elemente innerhalb einer Drop-Zone. 2. Der Akteur berührt anschließend mit einem Finger einen freien Bereich innerhalb einer anderen Drop-Zone. Hierbei kann die Berührung des Elements bereits abgeschlossen oder noch aktiv sein. 3. Das Programm versetzt das ausgewählte Element auf die entsprechende Drop-Zone. 4. Die Daten des Diagramms werden entsprechend angepasst.
Alternativablauf	1. Falls das zu wechselnde Element nicht auf diese Achse verschoben werden kann, wird keine Änderung durchgeführt.

[UC.20] Löschen eines Elements von einer Drop-Zone

Beschreibung	Der Akteur löscht eine der Achsenbelegungen, indem er eine Schaltfläche neben einem Element berührt.
Akteure	Analyst
Beinhaltet	–
Auslöser	–
Vorbedingungen	Anzeige der Drop-Zonen, Auswahl eines Dimensionselements, Auswahl einer *Kennzahl*.
Nachbedingungen	Ein Element wurde von einer Drop-Zone entfernt und das Diagramm neu erzeugt.
Ausnahmen	–
Erweiterungen	–
Standardablauf	1. Der Akteur berührt eine Schaltfläche neben einem der Elemente innerhalb einer Drop-Zone. 2. Das Programm entfernt dieses Element von der Drop-Zone. 3. Das Programm aktualisiert das Diagramm entsprechend der neuen Belegung der Achsen.
Alternativablauf	–

[UC.21] Anzeigen der Drop-Zonen

Beschreibung	Um eine Veränderung an der Belegung der Achsen vornehmen zu können, müssen die Drop-Zonen angezeigt werden.
Akteure	Analyst
Beinhaltet	*Move*-Geste
Auslöser	–
Vorbedingungen	–
Nachbedingungen	Die Drop-Zonen werden angezeigt.
Ausnahmen	–
Erweiterungen	–
Standardablauf	1. Der Akteur berührt und hält den Knopf "Menü". 2. Das Programm zeigt die Unterpunkte des Menüs an. 3. Der Akteur führt eine *Move*-Geste zum Unterpunkt für das Anzeigen der Drop-Zonen aus. 4. Er hebt den/die Finger von der Oberfläche des Tisches. 5. Das Programm blendet die Unterpunkte des Menüs aus. 6. Es zeigt die Drop-Zonen auf dem Diagramm an.
Alternativablauf	–

[UC.22] Speichern eines Diagramms

Beschreibung	Der Akteur kann den vollständigen Zustand eines Diagramms speichern.
Akteure	Analyst
Beinhaltet	*Move*-Geste
Auslöser	–
Vorbedingungen	
Nachbedingungen	Ein Diagramm wurde gespeichert.
Ausnahmen	–
Erweiterungen	–

Standardablauf	1. Der Akteur berührt und hält den Knopf "Menü". 2. Das Programm zeigt die Unterpunkte des Menüs an. 3. Der Akteur führt eine *Move*-Geste zum Unterpunkt für das Speichern aus. 4. Er hebt den/die Finger von der Oberfläche des Tisches. 5. Das Programm blendet die Unterpunkte des Menüs aus. 6. Es speichert den aktuellen Zustand des Diagramms.
Alternativablauf	–

[UC.23] Laden eines Diagramms

Beschreibung	Der Akteur kann den vollständigen Zustand eines Diagramms wiederherstellen. Alle gespeicherten Zustände werden in einer Auswahlliste angezeigt.
Akteure	Analyst
Beinhaltet	*Move*-Geste
Auslöser	–
Vorbedingungen	
Nachbedingungen	Ein Diagramm wurde geladen.
Ausnahmen	–
Erweiterungen	–
Standardablauf	1. Der Akteur berührt und hält die Schaltfläche "Menü". 2. Das Programm zeigt die Unterpunkte des Menüs an. 3. Der Akteur führt eine *Move*-Geste zum Unterpunkt für das Laden aus. 4. Er hebt den/die Finger von der Oberfläche des Tisches. 5. Das Programm blendet die Unterpunkte des Menüs aus. 6. Anstelle des Diagramms wird eine Auswahlliste mit bereits gespeicherten Zuständen angezeigt. 7. Falls der Platz nicht ausreicht, kann mit einer *Move*-Geste nach unten gescrollt werden. 8. Der Akteur sucht sich ein Listenelement aus. 9. Er berührt ein Listenelement. 10. Das Programm stellt den ausgewählten Diagrammzustand wieder her.
Alternativablauf	–

[UC.24] Anzeigen der Legende

Beschreibung	Damit der Akteur erkennen kann, welche Größe und Farbe der Punkte zu welchen Werten gehört, kann er die Zuordnungen in einer Legende nachvollziehen. Diese wird erst nach Betätigung eines entsprechenden Schaltfläche angezeigt.
Akteure	Analyst
Beinhaltet	*Move*-Geste
Auslöser	–
Vorbedingungen	
Nachbedingungen	Die Legende wird angezeigt.
Ausnahmen	–
Erweiterungen	–
Standardablauf	1. Der Akteur berührt und hält die Schaltfläche "Menü". 2. Das Programm zeigt die Unterpunkte des Menüs an. 3. Der Akteur führt eine *Move*-Geste zum Unterpunkt für das Anzeigen der Legende aus. 4. Er hebt den/die Finger von der Oberfläche des Tisches. 5. Das Programm blendet die Unterpunkte des Menüs aus. 6. Neben dem Diagramm wird ein neuer Bereich angezeigt, in dem die Legende zu sehen ist.
Alternativablauf	–

[UC.25] Abspielen einer Animation

Beschreibung	Ist die Visualisierungsdimension „Animation“ belegt, hat der Akteur die Möglichkeit, sich den Verlauf des Dimensionselementes über die Zeit anzeigen zu lassen. Ein Knopf zum Abspielen der Animation beginnt diesen Zeitverlauf und schaltet automatisch, in einem festgelegten Zeitintervall, zum nächsten Zeitabschnitt.
Akteure	Analyst
Beinhaltet	–
Auslöser	–
Vorbedingungen	Die Animations-Achse wurde belegt.
Nachbedingungen	Eine Animation wird abgespielt.
Ausnahmen	–
Erweiterungen	–

Standardablauf	1. Der Akteur berührt die Schaltfläche zum Abspielen der Animation 2. Die Schaltfläche verändert ihr Aussehen, sodass sie nun als Schaltfläche zur Betätigung einer Pause fungiert. 3. Die Animation wird abgespielt.
Alternativablauf	–

[UC.26] Pausieren einer Animation

Beschreibung	Eine bereits laufende Animation kann über eine Schaltfläche pausiert werden, um den aktuellen Zustand auf unbegrenzte Dauer betrachten zu können.
Akteure	Analyst
Beinhaltet	–
Auslöser	–
Vorbedingungen	Die Animations-Achse wurde belegt, eine Animation wird abgespielt.
Nachbedingungen	Eine Animation wurde pausiert.
Ausnahmen	–
Erweiterungen	–
Standardablauf	1. Der Akteur berührt die Schaltfläche zum Pausieren der Animation 2. Die Schaltfläche verändert ihr Aussehen, sodass sie nun als Schaltfläche zum Abspielen der Animation fungiert. 3. Die Animation wird pausiert.
Alternativablauf	–

[UC.27] Wechseln zu einem anderen Zeitabschnitt in einer Animation

Beschreibung	Der Akteur kann manuell durch die Zeitabschnitte in einer Animation navigieren. Hierzu gibt es zwei Schaltflächen, welche jeweils zum nächsten bzw. vorherigen Zeitabschnitt wechseln.
Akteure	Analyst
Beinhaltet	–
Auslöser	–
Vorbedingungen	Die Animations-Achse wurde belegt.
Nachbedingungen	Eine Animation wurde pausiert.
Ausnahmen	Falls kein Zeitabschnitt vor bzw. nach dem aktuellen Zeitabschnitt existiert, werden keine Veränderungen am Diagramm vorgenommen.

Erweiterungen	–
Standardablauf	1. Der Akteur berührt die Schaltfläche zum Wechseln in den nächsten bzw. vorherigen Zeitabschnitt der Animation. 2. Das Diagramm wechselt in den nächsten bzw. vorherigen Zeitabschnitt.
Alternativablauf	–

Teil IV

Umsetzung

Kapitel 8

Systembeschreibung

In diesem Kapitel werden die Hauptkomponenten des *TaP*-Systems vorgestellt. Zunächst wird die Zielumgebung des Systems beschrieben. Anschließend werden die Komponenten von *TaP* näher erläutert.

8.1 Systemumgebung

Folgende technische Voraussetzungen erfüllt die Zielumgebung:

Software

TaP soll für die .NET-Umgebung entwickelt werden und muss damit unter Windows-Betriebsystemen mit installiertem .NET-Framework 3.5 lauffähig sein. Eine Unterstützung für UNIX-Systeme und deren Derivate ist nicht vorgesehen. Als Datenquelle muss eine *OLAP*-Datenbank verfügbar sein. Diese kann entweder auf demselben Rechner laufen, wie das Endprodukt, oder ebenfalls über eine Netzwerkanbindung verwendet werden. Auf dem Zielsystem wird *Microsoft SQL Server* in Verbindung mit *Microsoft Analysis Services* verwendet. Als Betriebssysteme sind Microsoft Windows Vista und Microsoft Windows 7 installiert.

Hardware

TaP soll auf dem Multitouch-Tisch des *OFFIS* lauffähig sein. Der schematische Aufbau des Tisches kann Abbildung 8.1 entnommen werden. *TaP* wird auf dem in den Tisch integrierten PC (a) installiert. Die grafische Ausgabe findet über den Projektor (b) auf der Projektionsfläche (c) des Tisches statt. Von den Seiten strahlen Infrarot-LEDs (d) in die Scheibe der Projektionsfläche ein. Berührungen des Anwenders führen dazu, dass das Infrarot-Licht in Richtung der Kamera (e) reflektiert wird, die mit einem Infrarot-Bandpassfilter ausgestattet ist.

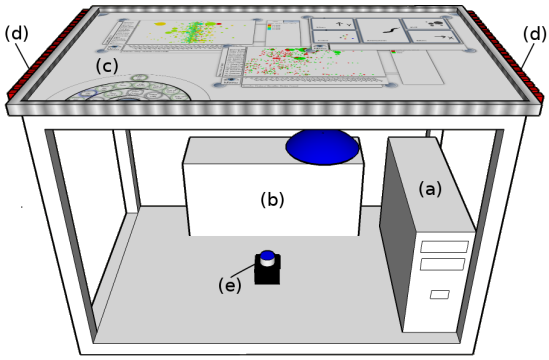


Abbildung 8.1: Aufbau des Multitouch-Tisches

Der 95 cm hohe Multitouch-Tisch hat eine Bildschirmdiagonale von 145 cm und hat eine Auflösung von 1280 x 800 Pixeln. Bei dem eingebauten PC handelt es sich um einen Intel Core 2 Quad Q9400 4 x 2.67 GHz mit 5 GB Arbeitsspeicher und einer NVIDIA GeForce 9800 GT Grafikkarte.

Schnittstellen

TaP muss einerseits die Eingaben des Multitouch-Tisches verarbeiten können. Diese werden mit Hilfe der Software *Community Core Vision* empfangen und mit Hilfe des *TUIO*-Protokolls an *TaP* weitergegeben. Die erhaltenen Nachrichten müssen sinnvoll zu Gesten oder Singlepoint-Eingaben verarbeitet werden. Andererseits muss *TaP* mit *Online Analytical Processing*-Systemen zusammenarbeiten, um seine Daten zu beziehen. Als Zwischenschicht zwischen *TaP* und der Datenbank soll zudem MUSTANG eingesetzt werden, um auf die darunter liegende Datenbank zuzugreifen.

8.2 Komponenten von *TaP*

Das Gesamtsystem besteht aus drei Komponenten, die in den folgenden Unterabschnitten behandelt werden. Im ersten Abschnitt wird die Anbindung der Datenquellen beschrieben. Darauf folgt eine Beschreibung der benötigten *GUI*-Komponenten und des Multitouch-Frameworks. Die Abbildung 8.2 zeigt das Zusammenspiel der drei Komponenten und ihrer Umgebung.

Datenbankschicht

Als Datengrundlage wird ein multidimensionales Datenbankmanagementsystem verwendet. Hierfür wird eine Datenbanbindung benötigt, die zum einen Datenbankanfragen formuliert und zum anderen eingehende Ergebnisse verarbeitet. Unser System unterstützt

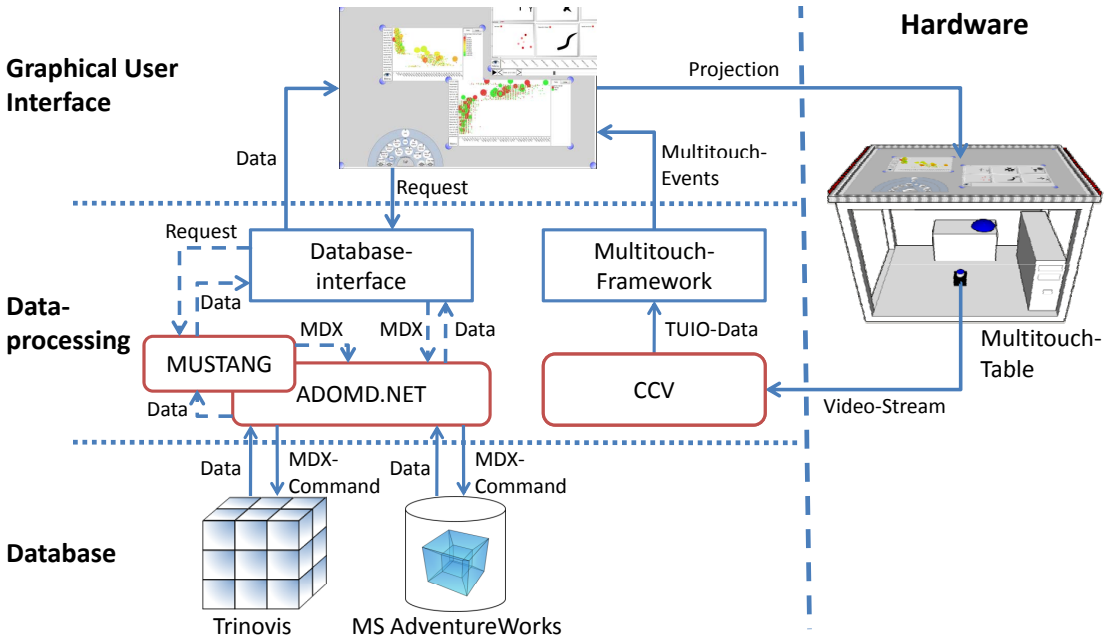


Abbildung 8.2: Übersicht der Komponenten von *TaP* und ihrem Zusammenspiel mit der Systemumgebung

zwei Datenbanken, die alternativ verwendet werden können. Bei der Einen handelt es sich um *AdventureWorks*, die von *Microsoft* öffentlich zur Verfügung gestellt wird. Es handelt sich um eine Sammlung von Wirtschaftsdaten eines fiktiven Unternehmens, die in multidimensionaler Form gespeichert worden ist. *AdventureWorks* wird über die *ADOMD.NET-API* angesprochen.

Bei der zweiten Datenbank handelt es sich um einen vom Unternehmen *Trinovis* gepflegten Gesundheitsdatensatz, welcher ebenfalls in multidimensionaler Form vorliegt. Dieser Datenbestand wird nicht direkt mit Hilfe von *ADOMD.NET* eingebunden. Stattdessen wird *MUSTANG* als Zwischenschicht verwendet. *MUSTANG* ist eine vom *OFFIS* entwickelte Plattform, welche die Entwicklung spezialisierter analytischer Informationssysteme für das Gesundheitswesen ermöglicht. Bei den zur Verfügung stehenden Daten handelt es sich um hoch aggregierte Daten über Krebserkrankungen. Rückschlüsse auf betroffene Einzelpersonen sind nicht möglich.

Die Daten werden aus einer der beiden Datenquellen ausgelesen und anschließend in ein gemeinsames übergeordnetes Datenmodell überführt. Die übrigen Komponenten von *TaP* verwenden ausschließlich dieses eigene Datenmodell.

Benutzungsschnittstelle

Die Benutzungsschnittstelle bildet den Kern von *TaP*. Sie übernimmt zum einen die Aufgabe die Daten mittels geeigneter, multidimensionaler Diagramme zu visualisieren.

Zum anderen ermöglicht sie es dem Benutzer durch eine Menüstruktur in den vorhandenen Daten zu navigieren.

Die Benutzungsschnittstelle besteht aus einer Arbeitsfläche, die ein oder mehrere Diagramme und ein hierarchisches Pie-Menü beinhaltet. Alle Komponenten lassen sich mit Multitouch-Eigenschaften versehen, sodass sie mit Hilfe von Gesten auf der Arbeitsfläche verschoben und skaliert werden können. Das Pie-Menü dient zur Auswahl von *Dimensionen* und *Kennzahlen*, die in einem Diagramm dargestellt werden können. Bei den Diagrammen handelt es sich die zentralen Komponenten für die Analyse. Sie sind als einfache Punktdiagramme realisiert, die um weitere Kenngrößen erweitert werden können.

Multitouch-Framework

TaP muss in der Lage sein, die Eingaben des Multitouch-Tisches, dessen Aufbau in Abschnitt 8.1 erläutert wurde, zu verarbeiten. Die entsprechende Komponente ist das Multitouch-Framework. Es empfängt *TUIO*-Signale und wandelt diese in Touch-Ereignisse um, auf welche die Komponenten der Benutzungsschnittstelle reagieren können. Das Multitouch-Framework erkennt einfache, natürliche Gesten, die vom Benutzer leicht zu erlernen und umzusetzen sind. Das Erscheinen eines *Blobs* löst beispielsweise ein TouchDown-Ereignis aus, und sein Verschwinden löst ein TouchUp-Ereignis aus.

Kapitel 9

Entwurf

Dieses Kapitel gibt einen Überblick über die Entwurfsgedanken, welche die Entwicklung von *TaP* vorangetrieben haben. Das System besteht aus fünf Hauptkomponenten, die im Folgenden detailliert betrachtet werden. Begonnen wird im nächsten Abschnitt mit dem Entwurf der Datenvisualisierung, gefolgt von einer Beschreibung des Desktops, in den die Diagramme eingebettet werden. Im Anschluss daran wird die Entwicklung eines Pie-Menüs motiviert und sein Entwurf beschrieben. Es folgen Abschnitte, in denen der Entwurf des Multitouch-Frameworks und der Datenbankanbindung erläutert werden.

9.1 Punktdiagramm

Dieser Abschnitt befasst sich mit der Visualisierung innerhalb von *TaP*. Es können Daten in Form von Punktdiagrammen repräsentiert werden, wodurch mehrere *Dimensionen* und *Kennzahlen* darstellbar sind.

Die Anforderungen an die visualisierten Diagramme umfassen unter anderem die Funktionalitäten *Zoom* [FA.4] und *Pan* [FA.5]. Das bedeutet, dass es möglich sein muss, das Diagramm beliebig stark zu vergrößern, um einen besseren Einblick in die Daten zu erhalten. Dabei ist zu beachten, dass die Datenpunkte selbst ihre Größe beibehalten, wenn das Diagramm vergrößert oder verkleinert wird. Sonst würden keine neuen Informationen gewonnen werden, wie es beispielsweise der Fall ist, wenn viele Punkte so nahe beieinander liegen, dass sie optisch nicht mehr voneinander getrennt werden können.

Neben den genannten Funktionalitäten *Zoom* und *Pan* muss sichergestellt sein, dass die zugrunde liegenden Daten korrekt angezeigt werden. Die Skalierung der Datenwerte auf den Wertebereich der Bildschirmkoordinaten darf durch das Zooming oder Panning nicht in Mitleidenschaft gezogen werden.

Zusätzlich zu den Multitouch-Gesten, die schlussendlich zur Steuerung des Diagramms eingesetzt werden sollen, findet eine Anbindung von Mausgesten statt. Dies ist zum

einen erforderlich, weil sich die Fertigstellung des Multitouch-Tisches mehr und mehr verzögerte. Zum anderen ist es während der Entwicklung generell einfacher eine Funktionalität mit der Maus zu testen, wenn kein einsatzbereiter Multitouch-Tisch zugegen ist.

Für die Entwicklung von Diagrammen mit diesen Funktionen ist zunächst in Betracht gezogen worden, *Visifire* zu verwenden. Die Umsetzung mit dieser Programmbibliothek stieß schnell auf unerwartete Hürden. Es stellte sich heraus, dass *Visifire* nicht allen Anforderungen an die Visualisierungen von *TaP* gerecht wird. Damit das Aussehen des Programms in späteren Iterationen leicht modifiziert oder auch gänzlich ausgetauscht werden kann, ist die Verwendung von *Templates* unabdingbar. *Visifire*-Diagramme können jedoch nicht innerhalb solcher *Templates* verwendet werden, sodass der Entschluss gefasst wurde, stattdessen eigene Visualisierungen zu entwickeln.

9.1.1 Überblick über das Punktdiagramm

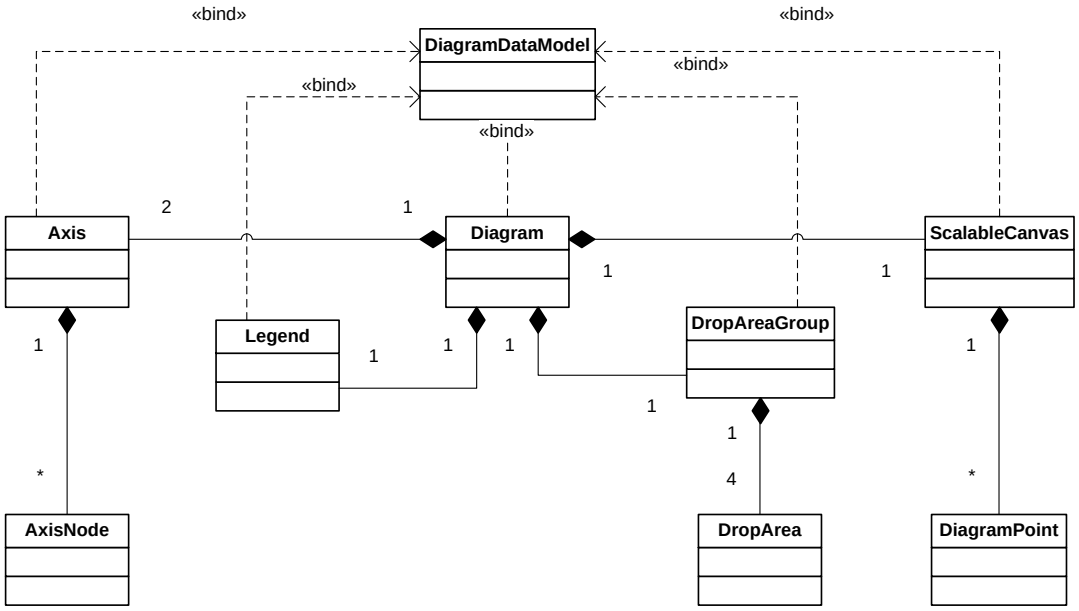


Abbildung 9.1: Klassendiagramm der am Punktdiagramm beteiligten Klassen

Das Klassendiagramm in Abbildung 9.1 zeigt die Komponenten des Punktdiagramms. Ihre Funktion und ihre Beziehungen zueinander werden im Folgenden beschrieben.

9.1.2 Die Containerklasse Diagram

Die Klasse `Diagram` stellt alle notwendigen Eigenschaften und Elemente für die Anzeige des Diagramms bereit. Dabei wurde das Diagramm mit Hilfe des Entwurfsmusters *MVVM* entwickelt. Die Model-Komponente gehört zum Namespace `TAP.Model` (siehe 9.5.2). Das `DiagramDataModel` stellt die ViewModel-Komponente des Entwurfmusters dar, während die anderen Klassen der View-Komponente zugeordnet werden. Wie in Abbildung 9.1 zu erkennen ist, stellt die Klasse `Diagram` eine zentrale Sammelstelle dar, in welche alle Klassen integriert werden. Das Diagramm besteht aus zwei Achsen, die Beschriftungen beinhalten. Zusätzlich ist in der Klasse `Diagram` eine Scalable-Canvas enthalten, in welcher die Punkte erzeugt und angezeigt werden. Ferner gibt es eine `DropAreaGroup`, die aus fünf verschiedenen DropZonen, je eine für Farb-, X-, Y- , Animations- und Größen-Achse, besteht. Weiterhin ist dem Diagramm eine animierte Fortschrittsanzeige zugeordnet, welche allerdings nur erscheint, wenn dies von der Datenbank auch unterstützt wird. Eine Legende gibt zusätzliche Informationen zu den Datenpunkten an.

Eigenschaften der `Diagram` Klasse:

Das Diagramm definiert vier *Attached Dependency Properties*, die im visuellen Baum vererbt werden.

- `TranslateX` und `TranslateY`: Geben an, um wie viel der Inhalt des Diagramms nach rechts oder links und nach oben oder unten verschoben ist. Diese Properties werden durch das Verschieben gesetzt.
- `ScaleX` und `ScaleY`: Bestimmen den Skalierungsfaktor für den Inhalt des Diagramms. Die Properties werden durch das Zoomen beeinflusst.

Das Diagramm hört auf die folgenden Events:

- `PropertyChanged` aus dem `DiagramDataModel`: Wenn diese den Wert `true` für `IsWorking` hat, dann wird die Fortschrittsanzeige angezeigt. Während diese angezeigt wird, sind die Interaktionen mit dem Diagramm unterbunden. Liefert `IsWorking` den Wert `false`, wird die Fortschrittsanzeige wieder geschlossen.
- `DragEnter`: Durch dieses Event wird die `DropAreaGroup` angezeigt.
- `DragLeave` und `Drop`: Wird dieses Event geworfen, wird die `DropAreaGroup` wieder ausgeblendet.

9.1.3 DiagramDataModel

Die Klasse `DiagramDataModel` ist Teil des ViewModels. Sie benachrichtigt andere angemeldete Elemente, wie `Axis`, `Legend`, `DropAreaGroup`, `ScalableCanvas` und das `Diagram` selbst, über Eigenschaftsveränderung mit Hilfe von `PropertyChangedEvent`s. Im `DiagramDataModel` werden die Daten für das Diagramm gespeichert.

Eigenschaften der `DiagramDataModel` Klasse:

- `Axis`: Ein `Dictionary`, in welchem die Zuordnung einer Achse zu *Dimensionen* oder Measures erfolgt.
- `CellSet`: Hier werden die Punkte nach einer Datenbankabfrage hinterlegt (siehe Abschnitt 9.5.2).
- `IsWorking`: Gibt an, ob gerade eine Datenbankabfrage ausgeführt wird oder nicht.
- `ColorModel`: Ist das Farbmodell für die Datenpunkte. Es ist abgeleitet von `Dictionary<String, Brush>`, welches eine Abbildung der Namen auf die entsprechenden `Brush`es realisiert.
- `AniModel`: Stellt das Modell für die Animationsachse bereit (siehe 9.1.9).
- `AxisMinMaxRelation`: Speichert den kleinsten und den größten Wert jeder Achse, der dargestellt werden muss.

Abbildung 9.2 zeigt das `DiagramDataModel`, welches ein Teil des *MVVM*-Models ist.

9.1.4 Axis und AxisNode

Das `UserControl Axis` übernimmt die Darstellung einer Achse in X- oder Y-Richtung. Sie baut sich auf, wenn sich im `DiagramDataModel` die Zuordnung der `Axis`-Komponente ändert. Es gibt zwei mögliche Szenarios:

- Wenn ein *Level* zur Achse zugeordnet wird, werden dessen *Nodes* sofort angezeigt.
- Wenn ein *Measure* zur Achse zugeordnet wird, wartet die Achse mit ihrem Aufbau solange, bis ein Diagramm dargestellt werden kann.

Außerdem können die Operationen *RollUp* und lokaler sowie globaler *DrillDown* über die Achse angestoßen werden. Bei einem *RollUp* handelt es sich um eine *Online Analytical Processing*-Operation, bei der der nächsthöhere *Level* eines bereits angezeigten *Level*s geholt und angezeigt wird. Der *DrillDown* liefert den nächsttieferen *Level*. Hierbei gibt es zwei Unterscheidungen: *lokal* und *global*. Bei einem lokalen

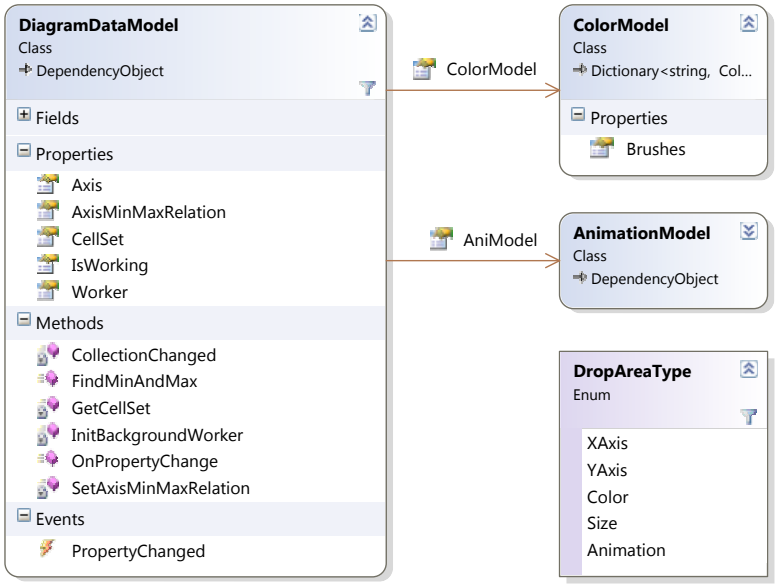


Abbildung 9.2: Klassendiagramm der Diagramm-Datenhaltung

DrillDown werden die Kind-Knoten eines ausgewählten Knotens angezeigt. Global bedeutet, dass alle Knoten des kompletten Kind-*Level*s angezeigt werden.

Bei einem *DrillDown* handelt es sich um eine Verfeinerung der angezeigten Daten, welche der Zoomfunktionalität nicht unähnlich ist. Daher wird es ebenfalls mit Hilfe der *Pinch*-Geste ausgelöst. Um einen globalen *DrillDown* zu erreichen, muss die Geste auf dem Bereich durchgeführt werden, der die Achse darstellt. Im Falle des lokalen *DrillDown*s kann die Analogie des Kneifens, um eine Verfeinerung zu erreichen, nicht beibehalten werden. Da ein spezieller Knoten ausgewählt wird, muss sich auch die Geste auf eine spezielle Achsenbeschriftung beziehen. Damit die Geste, die den lokalen *DrillDown* auslöst nicht fälschlicherweise mit der *Pinch*-Geste verwechselt wird, kann es sich nicht um ein TouchDown oder ein Move handeln. Daher wird ein TouchUp als Auslöser verwendet. Die *RollUp*-Operation, die grundsätzlich global ist, wird entsprechend durch die *Spread*-Geste ausgelöst.

Die Klasse `Axis` wird allgemein aus `AxisNode`s aufgebaut. Eine `AxisNode` stellt die sichtbare Beschriftung eines Elements einer Skala dar. Damit sich die Beschriftungen nicht überlappen, wird nur eine gewisse Anzahl von `AxisNode`s angezeigt. Je nach Skalierungsfaktor im Diagramm werden dynamisch neue `AxisNode`s sichtbar gemacht. Dies erfolgt in der Methode `HideNodes`, die dafür verantwortlich ist, dass die `AxisNode`s auf der richtigen Position sind und nicht überlappend angezeigt werden.

9.1.5 DropAreaGroup und DropArea

`DropAreaGroup` ist ein Container bestehend aus mehreren `DropArea`s. Dieser reagiert auf DragOver- und Drop-Events, wobei bei einem `DragOver` untersucht wird, ob ein Drop-Element an dieser Stelle gedroppt werden darf, welches anhand vorgeschriebener Regeln von der `Check`-Methode überprüft wird. Die Regeln werden im Folgenden aufgelistet:

- Es dürfen nur *Level* und *Measure*s gedroppt werden.
- Es sind nur Kombination von *Online Analytical Processing*-Objekten erlaubt, welche aus Datenbanksicht möglich sind.
- Es ist nicht erlaubt, *Measure* und *Level* auf eine Drop-Zone zu kombinieren.
- Auf die Farb-, Animations- und Größenachse können entweder eine *Kennzahl* oder ein *Level* gedroppt werden.
- Auf die Größen-Dimension dürfen nur *Measure*s gezogen werden.
- Auf die Animation-Drop-Zone dürfen nur *Level* gezogen werden.
- Maximal ist pro Drop-Zone ein *Level* erlaubt.
- Es ist nicht erlaubt, Elemente aus der gleichen *Dimension* auf das Diagramm zu ziehen.
- Es ist nicht erlaubt, Elemente auf eine der Drop-Zonen zu ziehen, die schon auf einer der Achsen sind.

Bei einem legitimen Drop wird das Drop-Element durch die `Drop`-Methode in das Dictionary für die Zuordnungen der ausgewählten Elemente zu den Achsen eingefügt. Wird ein Element von einer Drop-Zone in eine andere Drop-Zone verschoben, wird das Drop-Element aus dem Dictionary entfernt und dann wieder neu zugeordnet. Die *GUI* passt sich dann neu an.

9.1.6 ScalableCanvas und DiagramPoint

`ScalableCanvas` ist ein Container, welcher `DiagramPoint`s beinhaltet. Die `Scaleable-Canvas` wird aufgebaut, sobald sich das `CellSet`-Property ändert. Falls ein neues `CellSet` gesetzt wird, findet ein Neuaufbau des Diagramms statt. Dieser Aufbau beinhaltet die Berechnung der `DiagramPoint`s, also die Bestimmung der Position, der Größe und der Farbe der Punkte. Außerdem sorgt die `ScaleabaleCanvas` dafür, dass die Größe der Punkte sich bei einer Skalierung nicht verändert.

9.1.7 DiagramState und DiagramStateList

Um eine Zustandsspeicherung des aktuellen Diagramms zu ermöglichen, wird die Klasse `Diagram` um einige Methoden erweitert, sowie die Klasse `DiagramState` erstellt. `DiagramStateList` sorgt für die visuelle Repräsentation der gespeicherten Zustände. Gespeicherte Zustände werden dort in Form von Screenshots aufgelistet und mittels *TouchUp*-Geste ausgewählt.

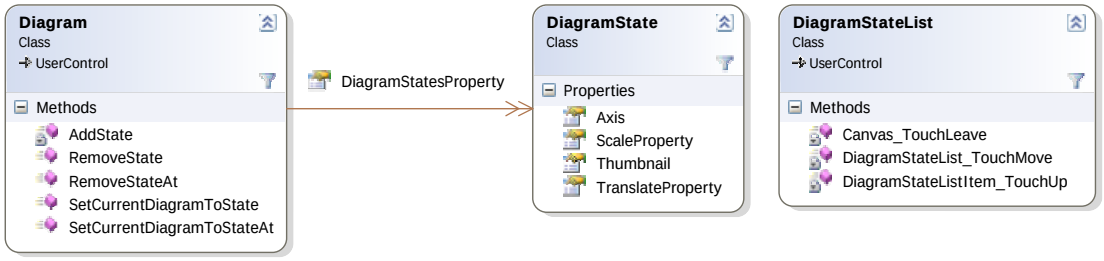


Abbildung 9.3: Klassendiagramm der an der Zustandsspeicherung beteiligten Klassen

Das Speichern geschieht über den Menüpunkt *Save*, während der Nutzer über *Load* zu der Liste der gespeicherten Zustände gelangt. Pro Speichervorgang wird ein neues Objekt vom Typ `DiagramState` erstellt, in dem alle für die Wiederherstellung nötigen Daten gehalten werden. Dazu gehören die Belegung der Achsen, der Zoom- und der Verschiebungsfaktor sowie ein Screenshot des Diagramms. Alle `DiagramState`-Objekte werden der `DiagramStateList` unmittelbar nach Erstellung über die Methode `AddState` hinzugefügt. Abbildung 9.3 zeigt das Klassendiagramm[1] der drei beteiligten Klassen.

9.1.8 Legend

Zur Erklärung der im Diagramm dargestellten Farb- und Größenwerte stellt die Legende zwei Reiter zur Verfügung. Der Farbreiter stellt die einzelnen Farbe-Bezeichner-Paare als eine Liste von farbigen Rechtecken und dazugehörigen Textfeldern dar. Diese Liste wird aktualisiert, sobald vom `DiagramDataModel` das PropertyChanged-Event *ColorModel* gefeuert wird. Im Größenreiter sind, analog zum Farbreiter, die Größe-Bezeichner-Paare untereinander angeordnet. Dabei wird der aktuell verfügbare Platz innerhalb der Legende ausgenutzt. Die Punktgrößen werden proportional zum Wertebereich der dargestellten *Kennzahlen* berechnet. Die Aktualisierung des Größenreiters erfolgt sobald das `DiagramDataModel` das PropertyChanged-Event *CellSet* feuert oder die aktuelle Größe des Diagramms und damit auch die der Legende verändert wird.

[1] Die Darstellung der Klasse `Diagram` ist in dieser Abbildung der Übersicht halber auf die im Rahmen der Zustandsspeicherung erweiterten Funktionen beschränkt.

Die Klasse `Legend` sorgt für das Layout und die Funktionalität der Legende. Das `DiagramDataModel` stellt mit dem `ColorModel` bereits eine Datenstruktur zur Verfügung, aus der der Inhalt des Farbreiters automatisch generiert werden kann. Zum Befüllen des Größenreiters dient das in `Legend` durch die Methode `GenerateSizeModel` erstellte Modell, das äquivalent zum `ColorModel` Größe-Bezeichner-Paare erstellt. Abbildung 9.4 stellt die Klassenstruktur[2] der Legende dar.

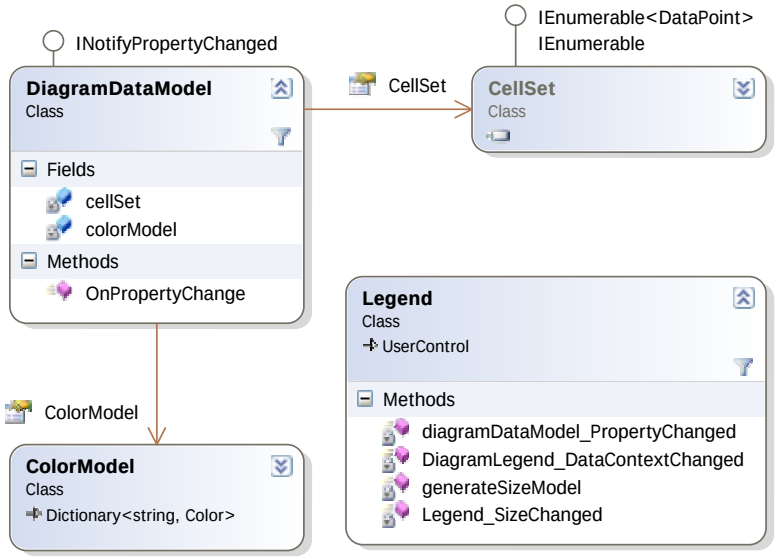


Abbildung 9.4: Klassendiagramm der an der Legende beteiligten Klassen

9.1.9 AnimationModel und AnimationPlayer

Das `AnimationModel` kapselt die Operationen, welche für die Animation der Diagrammpunkte benötigt werden. Eine Animation wird erzeugt, sobald ein Anwender ein *Level* auf die Animations-Dropzone zieht. Am anschaulichsten ist dies, wenn es sich um ein *Level* einer Zeitdimension handelt, dann spiegelt die Animation der Punkte den Verlauf der Daten über die Zeit wieder. Da es stark von der Datenbank abhängt, bei welcher *Dimension* es sich um eine Zeitdimension handelt und da dies nicht – ohne einen zusätzlichen Mechanismus einzuführen – abfragbar ist, ist die Auswahl von *Level*, die für die Animation verwendet werden, nicht auf bestimmte *Dimensionen* beschränkt. Eine Animation entsteht aus mehreren Diagrammen, die dem Anwender in einer zeitlichen Abfolge angezeigt werden. Für jedes der Einzelbilder werden die Eigenschaften *Größe*, *Position*, *Farbe* und *Sichtbarkeit* aller Diagrammpunkte neu berechnet und in das Diagramm eingefügt. Das `AnimationModel` hat Interaktionsfunktionen, die es

[2] Die Darstellung der Klassen `DiagramDataModel`, `ColorModel` und `CellSet` sind in dieser Abbildung der Übersicht halber auf die im Rahmen der Legende benötigten Funktionen und Properties beschränkt.

ermöglichen, die Animation zu starten und sie zu stoppen. Diese werden durch den `AnimationPlayer` visualisiert und dem Anwender zugänglich gemacht. Die Buttons zum starten und stoppen der Animation sind ähnlich einem Videoplayer umgesetzt. Zusätzlich gibt es Vorwärts- und Zurückbuttons, die es erlauben die Einzelbilder manuell durchzuschalten, um sie somit einzeln zu betrachten. Damit dem Anwender immer klar ist, welchen Knoten des Animationslevels er gerade betrachtet, wird der Name des Knotens ebenfalls durch den `AnimationPlayer` angezeigt.

9.2 Desktop

Die Arbeitsfläche von *TaP* besteht aus zwei aufeinanderliegenden Schichten, welche als Canvas implementiert sind. Zum einen die Desktopschicht, die die Diagramme enthält und zum anderen eine Befehlsschicht, auf welcher pfadbasierte Gesten ausgeführt werden können. Damit der Benutzer sehen kann, wie der aktuelle Pfad der Geste aussieht, wird dieser mittels einer blauen Linie angezeigt. Die Schicht wird bei Berührung von einer der vier Ecken der Software ein- bzw. ausgeblendet. Diese Befehlsschicht wird durch eine halbdurchsichtige Canvas dargestellt, welche in das `MainWindow` gelegt wird. Für die Zeichnung der Linie und der Gestenerkennung wird die Klasse `DiagramEventTrigger` implementiert. Wenn ein Viereck erkannt wird, wird die Action `AddDiagramAction` ausgeführt, welche ebenfalls den Pfad übergeben bekommt. Diese Action untersucht den Pfad, berechnet die Diagrammgröße und die Diagrammposition und fügt anschließend ein neues Diagramm in die Desktopschicht ein. Da die Diagramme auch aufeinander liegen können sollen, muss die Funktionalität gegeben sein, ein Diagramm zu fokussieren.

9.3 Entwurf des Pie-Menüs

Das Pie-Menü soll eine Auswahlstruktur zur Verfügung stellen, die es ermöglicht, *Kennzahlen* und *Dimensionen* für die Darstellung auszuwählen. Sowohl für die Auswahl von *Kennzahlen* als auch für die von *Dimensionen* wird dasselbe Pie-Menü verwendet. Damit *Kennzahlen* und *Dimensionen* unterschieden werden können, enthält der erste Ring nur diese beiden Elemente. Das Pie-Menü selbst ist durch seine halbrunde Darstellung an die Form einer Hand angepasst, sodass die Bedienung auf dem Multitouch-Tisch intuitiv und einfach erfolgen kann.

9.3.1 Funktionalität

Das Pie-Menü stellt die aus der Datenbank erhaltenen *Dimensionen* und *Kennzahlen* hierarchisch in verschiedenen Ebenen dar. Sollten mehr Elemente anzuzeigen sein, als eine Ebene des Pie-Menüs aufnehmen kann, wird es automatisch rotierbar und die nicht dargestellten Elemente lassen sich durch eine *Move*-Geste in eine sichtbare Position drehen. Um das Pie-Menü zu rotieren, können ebenso Buttons verwendet werden, welche an den Enden des jeweiligen Rings angebracht sind. Sie zeigen außerdem die Anzahl der verdeckten Elemente an. Damit der Anwender die Orientierung im Menü nicht verliert, ist ein Statusbalken in jedem Ring implementiert. Dieser ist an der Unterseite des entsprechenden Ringes plaziert und stellt in dunkel-grau den aktuell angezeigten Bereich dar. Zu Beginn ist das Pie-Menü komplett eingeklappt. Durch einen *TouchDown* auf einem Element wird dieses selektiert und, sofern eine weitere Hierarchiestufe verfügbar ist, die nächste Stufe als neuer Ring um das Menü dargestellt. Der Status aller Selektionen wird im ViewModel gespeichert. Durch das Herausziehen eines Elementes aus dem Pie-Menü wird eine Drag-Operation gestartet. Anschließend kann es mit Hilfe eines Drops auf eine *Drag and Drop*-Zone losgelassen werden, wodurch es für eine Achse selektiert wird. Wird der `BasisButton` des Menüs berührt, wird es in den Anfangszustand zurückgesetzt, der Selektionszustand aller Elemente bleibt jedoch weiterhin gespeichert.

Das Pie-Menü stellt hierarchische Daten dar, welche unter anderem viele Ebenen haben können. Auf dem Multitouch-Tisch steht allerdings nur ein begrenzter Platz zur Darstellung zur Verfügung. Aus diesem Grund werden immer nur maximal drei Ringe (unter- und oberhalb der Auswahl, und der aktuell ausgewählte Ring) angezeigt. Die darunter liegenden Ebenen werden durch einen dünneren Ring, auf dem nur das ausgewählte Element dargestellt ist, repräsentiert.

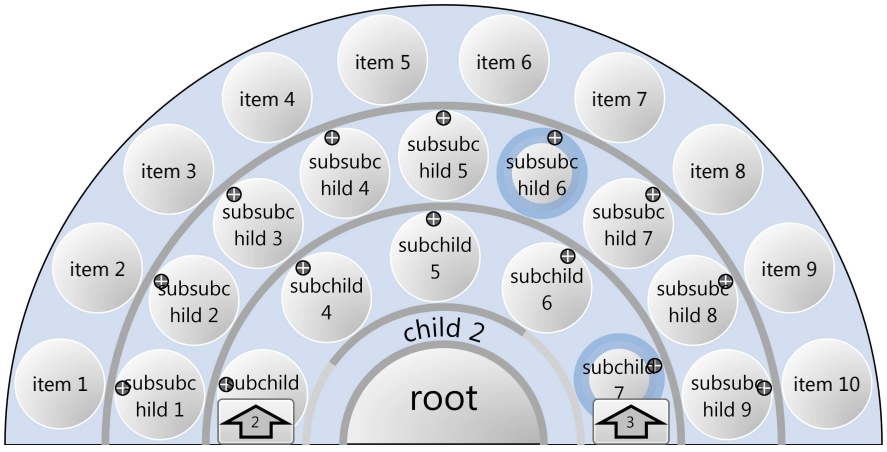


Abbildung 9.5: Überblick über das Pie-Menü

Das Pie-Menü kann durch einen Touch auf die Arbeitsfläche ausgeblendet werden, wodurch die gesamte Arbeitsfläche für die Analyse zur Verfügung steht. Soll das Menü wieder eingeblendet werden, ist dies durch das Auflegen einer Hand auf dem Multitouch-Tisch möglich.

Für die Entwicklung ist zusätzlich zu der Multitouch-Unterstützung eine Maussteuerung implementiert, da nicht jeder Entwickler immer einen Multitouch-Tisch zum Testen seines Entwicklungsstandes zur Verfügung hat.

9.3.2 Komponenten des Pie-Menüs

Im Folgenden Klassendiagramm wird der Aufbau des Pie-Menüs aus den einzelnen Komponenten dargestellt.

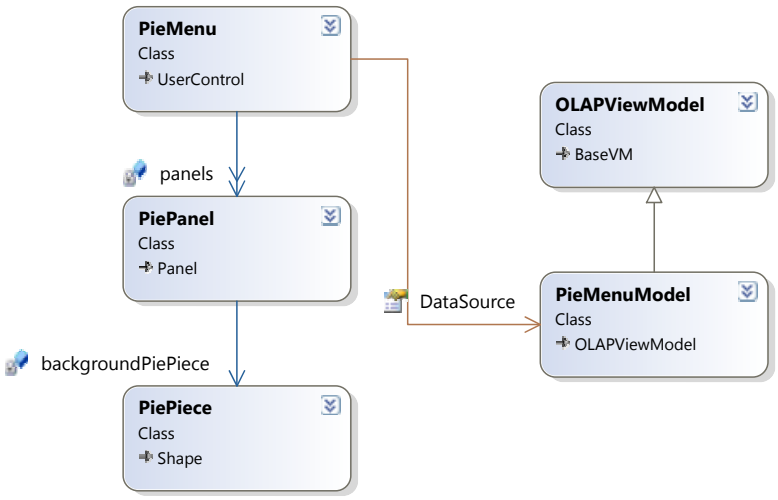


Abbildung 9.6: Klassendiagramm des Pie-Menüs

PiePiece

Die Hintergrund- und Rahmengestaltung, sowie die Darstellung des Pie-Menüs als Halbkreis, wird von der Klasse `PiePiece`[3] übernommen. Es wird jedoch auch eine Möglichkeit vorgesehen, den sichtbaren Kreisbogen des `PiePieces` und damit auch den des Pie-Menüs anzupassen, um das Pie-Menü beispielsweise in einer Ecke platzieren zu können.

[3] `www.codeproject.com`, Lizenz: `Code Project Open Licence (Cpol) 1.02`

PiePanel

Die Anordnung der Schaltflächen, mit denen die *Dimensionen* und *Kennzahlen* dargestellt werden, übernimmt die Klasse `PiePanel`. Je nach Anzahl der anzuzeigenden Elemente wird das `PiePanel` automatisch rotierbar.

PieMenu

Die Klasse `PieMenu` ist die Klasse, die von der Projektgruppe genutzt wird, um das Pie-Menü darzustellen. Sie beinhaltet das `PiePanel` und übergibt diesem die darzustellenden Elemente. Bei einem `Touch` auf einem Element reagiert `PieMenu` auf das entsprechende Event und erzeugt bei Bedarf ein neues `PiePanel` mit den entsprechenden Elementen.

OLAPViewModel

Das `OLAPViewModel` kapselt ein `HierachicalOLAPObject` und bildet das ViewModel zur Darstellung. Es stellt ein Modell zur Selektion bereit. Außerdem werden die Kind-Elemente im Hintergrund geladen, wodurch der *GUI*-Thread nicht blockiert wird. Das Laden der Kind-Elemente im Hintergrund kann durch eine Property deaktiviert werden.

PieMenuModel

Das ViewModel des Pie-Menüs wird durch die Klasse `PieMenuModel` realisiert. Es erweitert die Klasse `OLAPViewModel` um die Fähigkeit zu speichern, wie weit das Pie-Menü gedreht ist.

9.4 Entwurf des Multitouch-Frameworks

Das Multitouch-Framework besteht im Wesentlichen aus fünf Klassen (s. Abbildung 9.7).

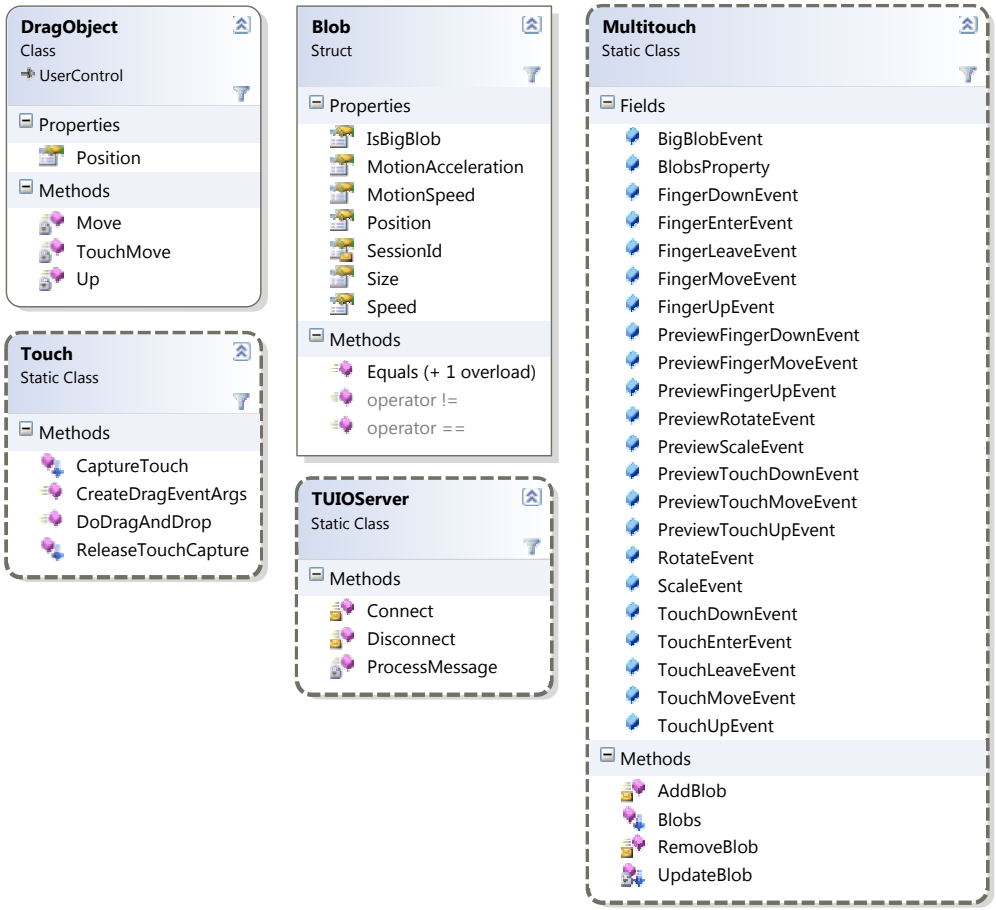


Abbildung 9.7: Wichtige Multitouch-Framework Klassen

9.4.1 Blob

Die Struktur `Blob` modelliert die Berührung eines Fingers auf dem Tisch. Sie besitzt die Informationen über die aktuelle Position, Geschwindigkeit und Beschleunigung eines Fingers, sowie eine eindeutige *ID* des Fingers.

9.4.2 TUIOServer

Der `TUIOServer` empfängt und verarbeitet Nachrichten im *TUIO*-Format. Die *TUIO*-Nachrichten werden geparst, in `Blob`-Objekte umgewandelt und für die weitere Verarbeitung an die `Multitouch`-Klasse weitergereicht.

9.4.3 Multitouch

Die `Multitouch`-Klasse definiert eine Reihe von *Attached Dependency Properties*, die aus den Anforderungen abgeleitet sind. Die Events TouchDown, TouchUp und entsprechende Preview-Events informieren die *GUI* über die Geste *Berühren*. Die Events TouchMove, TouchEnter, TouchLeave informieren über die Geste *Verschieben*. Über das *Rotieren* und *Skalieren* wird mittels Rotate- und Scale-Event informiert. Die Touch-, Scale- und Rotate-Events sollen unabhängig von der Anzahl der ausführenden Finger ausgelöst werden. Um dennoch Events auszulösen, wenn neue Finger auf die Tischoberfläche gebracht werden, sollen `Finger`-Events bereitgestellt werden. Das Wichtigste in der `Multitouch`-Klasse ist eine *Attached Dependency Property Blobs*, welche auf Grundlage der Attached-Eigenschaft auf beliebige `UIElemente` gesetzt werden kann. Da das `UIElement` direkt von `Visual` erbt, kann diese Eigenschaft auf beliebige *GUI*-Elemente gesetzt werden. Damit können *GUI*-Elemente zu jedem beliebigem Zeitpunkt die Informationen über ihre darauf liegenden Finger erhalten.

Der Datenfluss im Framework ist in Abbildung 9.8 dargestellt. Nachdem die Fingertracking-Software Veränderungen auf der Tischoberfläche erkennt, sendet sie eine *TUIO*-Nachricht. Der *TUIO*-Server empfängt diese, konvertiert sie in ein `Blob`-Objekt um und gibt es weiter an die `Multitouch`-Klasse. Diese filtert heraus, welche *GUI*-Elemente von diesen Veränderungen betroffen sind und aktualisiert die Liste `BlobsProperty`. Anschließend versendet sie Events, falls entsprechende Gesten erkannt worden sind.

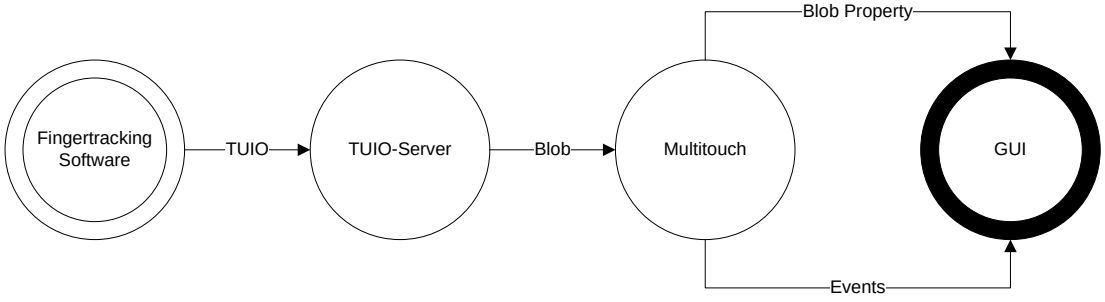


Abbildung 9.8: Datenfluss im Multitouch-Framework

9.4.4 Touch

Das Multitouch-Framework besitzt eine Klasse `Touch`, welche die Funktionalität für *Touch-Capture* und *Drag and Drop* enthält. Die Umsetzung dieser Funktionen ist aus den *WPF*-Funktionen *Mouse Capture* und *Drag and Drop* bekannt. Ein *Drag and Drop* wird mittels der statischen Methode `DoDragAndDrop` der statischen Klasse `Touch` ausgeführt. Diese Methode ist der Methode `DoDragDrop` von der `DragDrop`-Klasse aus *WPF* sehr ähnlich, es wird lediglich ein zusätzlicher Parameter vom Typ `Point` übergeben, welcher die Mitte des zu ziehenden Elementes relativ zum *TaP*-Window angibt. Diese Methode erstellt ein `DragObject`, fügt es in das Fenster ein und setzt es an die Position des übergebenen `Point`s.

9.4.5 DragObject

Um *Drag and Drop*-Funktionalität bereitstellen zu können, muss neben den in der Klasse `Touch` bereitgestellten Funktionen noch eine visuelle Repräsentation des zu ziehenden Elementes geschaffen werden. Dies wird durch die von `UserControl` abgeleitete Klasse `DragObject` geleistet.

9.5 Entwurf der Datenbankanbindung

Das Programm soll in der Lage sein multidimensionale Daten sowohl direkt über *ADOMD.NET* als auch mit Hilfe von *MUSTANG* zu verwenden. Die Verwendung der multidimensionalen Daten durch andere Komponenten (z.B. Menüs) soll unabhängig von der Struktur der Datenbank sein, sodass ein Wechsel zwischen diesen beiden problemlos möglich ist. Die Daten aus beiden Datenquellen werden hierfür in ein einheitliches Datenmodell überführt. In beiden Fällen muss das Abrufen von Metadaten und Daten gewährleistet sein. Zu den Metadaten zählen folgende Elemente (in den Klammern sind die zugehörigen Klassen angegeben):

- *Dimensionen* (`Dimension`)
- *Level* (`Level`)
- *Hierarchie* (behandelt wie `Level`)
- *Knoten* (`Node`)
- *Kennzahlen* (`Measure`)
- *Kennzahl-Gruppen* (behandelt wie `Measure`)

9.5.1 DatabaseFactory

Um die Anbindung an die beiden Datenbanken realisieren zu können, kommt das Fabrik-Muster zum Einsatz (s. Abb. 9.9). Dazu wird eine abstrakte Datenbank-Klasse verwendet, welche von den beiden jeweiligen Datenbank-Klassen implementiert wird. Die Fabrik liefert eine Instanz einer der Datenbanken zurück. Der zu verwendende Datenbanktyp wird in einer Konfigurationsdatei eingestellt.

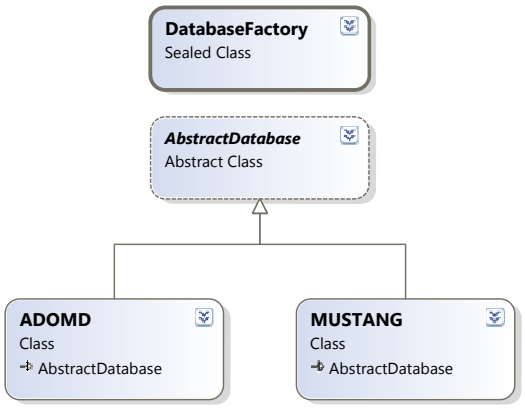


Abbildung 9.9: Klassendiagramm der Datenbank-Schnittstelle

9.5.2 OLAP-Abstraktionsschicht

Bei dem Entwurf der Datenbankschnittstelle wird zwischen Metadaten, einer Beschreibung über die Daten und den konkreten Daten aus der jeweiligen Datenbank unterschieden. In den folgenden Abschnitten werden diese beiden Modelle zur Repräsentation der Metadaten und Daten dargestellt.

Metadaten

Die *Online Analytical Processing*-Abstraktionsschicht (s. Abb. 9.10) beinhaltet zu jedem Datentyp der jeweiligen Anbindung eine entsprechende Klasse. Alle Klassen der *Online Analytical Processing*-Abstraktionsschicht erben von der gemeinsamen Oberklasse `OLAPObject`, die ihre Gemeinsamkeiten zusammenfasst. Weiter wird zwischen *Dimensionen*, *Kennzahlen* und Dimensionselementen unterschieden, die *Knoten* und *Level* umfassen. Diese Datentypen befinden sich in einer hierarchischen Struktur, wobei der Datentyp `Cube`, der alle *Dimensionen* und *Kennzahlen* beinhaltet, das oberste Element darstellt. Es muss sichergestellt sein, dass auf die jeweiligen Kind-Elemente in der Hierarchie (s. Abb. 2.20) zugegriffen werden kann. Dies geschieht mit Hilfe der Methode `GetChildren` der Datenbankschnittstelle.

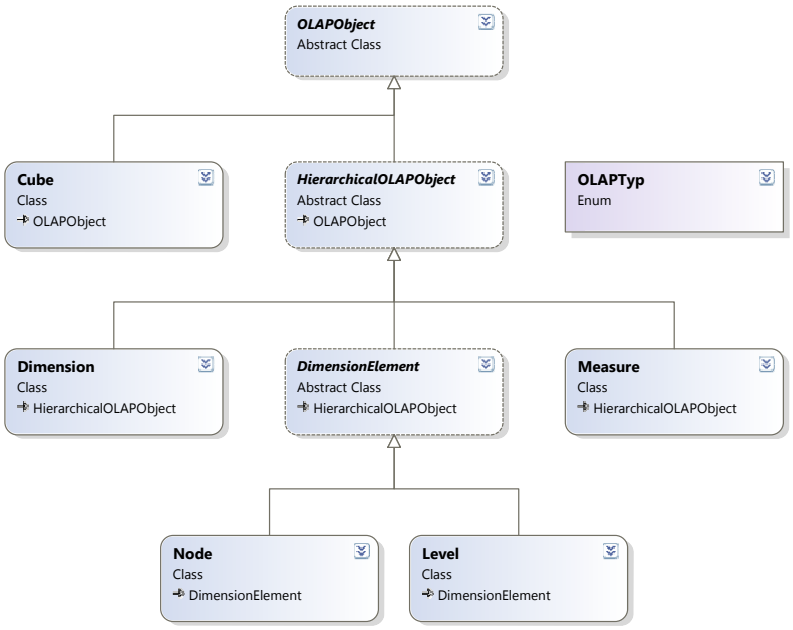


Abbildung 9.10: Klassendiagramm der OLAP Abstraktionsschicht

Bei den Kind-Elementen einer *Dimension* handelt es sich immer um *Level*. *Hierarchien*, wie sie in *ADOMD.NET* vorkommen, werden in diesem Modell ebenfalls wie *Level* behandelt. Kind-Elemente eines *Level*s sind ebenfalls *Level*. Zusätzlich enthält jeder *Level*-Knoten, bei deren Kind-Elementen es sich wiederum um Knoten handelt, welche dem nächsttieferen *Level* zugeordnet sind. Das Abrufen von Knoten zu einem bestimmten *Level* erfolgt durch die Methode `GetNodes` der Datenbankschnittstelle.

Da *MUSTANG* keine getrennte Behandlung von Kennzahlgruppen vorsieht, werden *Kennzahlen* und Kennzahlengruppen beide durch die Klasse *Measure* repräsentiert. Kennzahlgruppen können *Kennzahlen* als Kinder enthalten. *Kennzahlen* haben nie Kind-Elemente.

Es ist nicht immer gegeben, dass eine *Dimension* mit einer beliebigen *Kennzahl* kombinierbar ist. Daher muss vor einer Abfrage an die Datenbank eine Kompatibilitätsprüfung durchgeführt werden. Bevor eine weitere *Kennzahl* oder *Dimension* ausgewählt werden kann, wird durch die Methode `GetCompatibles` ermittelt, welche Kombinationen mit den zuvor ausgewählten *Dimensionen* und *Kennzahlen* möglich sind.

Daten

Da die Rückgabe-Objekte der beiden Schnittstellen zum Abrufen von Daten unterschiedlich sind, muss auch hier ein einheitliches Modell (s. Abb. 9.11) verwendet

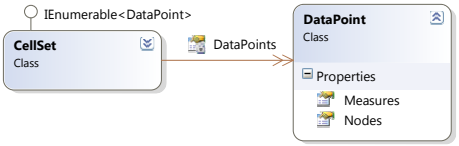


Abbildung 9.11: Klassendiagramm der OLAP Datenbehandlung

werden. Die *MUSTANG*-Schnittstelle liefert ein `CubeVO`-Objekt, *ADOMD.NET* ein `CellSet`-Objekt. Der einheitliche Datentyp wird durch die eigene Klasse, welche ebenfalls den Namen `CellSet` trägt, repräsentiert. Ein solches `CellSet` ist eine Liste aus `DataPoint`-Objekten. Ein `DataPoint` besteht aus einer Liste von *Knoten*, welche die Position im Datenwürfel bestimmen, und aus einem `Dictionary` mit *Kennzahlen* und den zugehörigen Werten des Datenpunktes.

Zum Abrufen von Daten aus dem Datenwürfel wird die Methode `GetData` verwendet. Für das erfolgreiche Abrufen von Daten werden Listen mit Knoten, welche den Datenbereich bestimmen, und eine Liste der abzurufenden Kennzahlen benötigt.

9.6 Zusammenfassung

In den vorangegangenen Abschnitten wurde das Grundkonzept des Systems vorgestellt. Teil dieses Konzepts ist der in Abbildung 9.12 dargestellte *LogicalTree*. Dieser beschreibt den logischen Aufbau der visuellen Komponenten von *TaP*. Hieran ist gut zu erkennen, wie die zuvor beschriebenen Komponenten zueinander in Beziehung stehen und aus welchen weiteren Teilkomponenten diese bestehen. Erkennbar ist diese Zuordnung durch die verschiedenfarbigen Rechtecke, wobei die Farbe nur der Übersichtlichkeit dient. Ein Rechteck, dass in einem anderen Rechteck enthalten ist, stellt eine WPF-Komponente dar, die Unterkomponente einer anderen ist. Beispielsweise verfügt die *XAxis*-Komponente des Typs `Axis` über beliebig viele Unterkomponenten vom Typ `AxisNode`. Ein Rechteck mit dem Text „...“ zeigt an, dass beliebig viele Unterkomponenten innerhalb dieser Komponenten existieren können.

Während des Entwurfs hat sich ergeben, dass kein existierendes Framework zur Erstellung von Diagrammen für die Zwecke von *TaP* einsetzbar ist. Somit war es erforderlich die gesamte Diagramm-Komponente, welche auf dem *MVVM*-Model basiert, selbst zu entwickeln. Der Entwurf hat gezeigt, dass die Punktdiagramm-Komponente eines erheblichen Aufwandes bedarf. Weiterhin wurde in diesem Abschnitt der Entwurf des Desktop und des Pie-Menüs behandelt. Das Pie-Menü ist eine eigenständige, austauschbare Komponente, welches speziell für Multitouch-TableTop-Computer entworfen wurde.

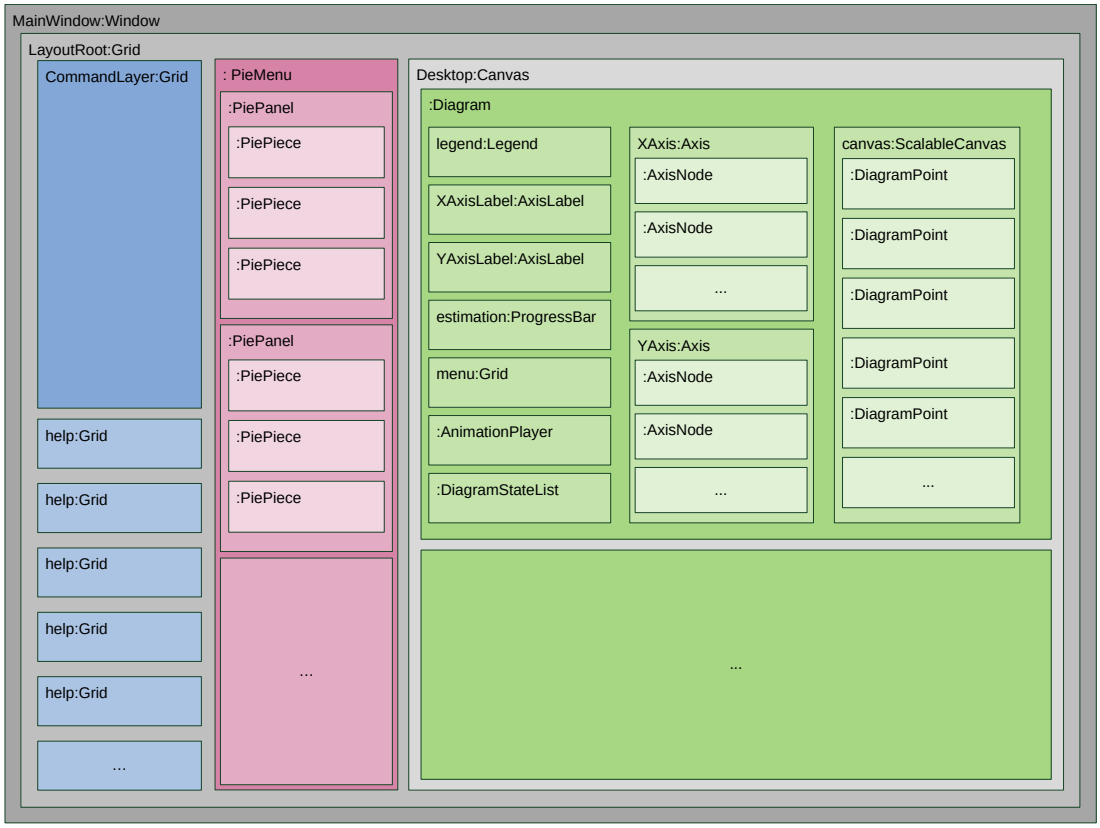


Abbildung 9.12: Logischer Überblick über die visuellen Komponenten von *TaP*

Nachdem die visuellen Komponenten erklärt wurden, folgten die Entwurfsentscheidungen für die Multitouch- und Datenbank-Komponente.

Bei dem Entwurf des Multitouch-Frameworks wurde festgestellt, dass eine Vielzahl von Events eingeführt werden musste. Neben den Klassen, welche die Multitouch-Funktionalität bereit stellen, musste z. B. auch eine von `UserControl` abgeleitete Klasse `DragObject` erstellt werden, um die Interaktion *Drag and Drop* für Multitouch möglich zu machen.

Der Entwurf der Datenbank hat gezeigt, dass ein allgemeines Datenmodell für die Anbindung unterschiedlicher Datenbanktypen notwendig ist. Somit mussten auch die Funktionen zum Abruf der Metadaten verallgemeinert werden. Die Klasse `DatabaseFactory` ermöglicht die Verwendung der unterschiedlichen Datenbanktypen.

Kapitel 10

Implementierung

Dieses Kapitel gibt einen Überblick über die Umsetzung der Entwurfsgedanken, welche die Entwicklung von *TaP* vorangetrieben hat. Das System besteht aus fünf Hauptkomponenten, die im Folgenden detailliert betrachtet werden. Begonnen wird im nächsten Abschnitt mit der Implementierung des Punktdiagramms, gefolgt von einer Beschreibung des Desktops, in den die Diagramme eingebettet werden. Im Anschluss daran wird die Implementierung des Pie-Menüs beschrieben. Es folgen Abschnitte, in denen die Implementierung des Multitouch-Frameworks und der Datenbankanbindung erläutert werden.

10.1 Punktdiagramm

Die Analyse findet in *TaP* hauptsächlich durch ein Punktdiagramm statt. Dieses besteht aus einem Modell, dem `DiagramDataModel`, und der eigentlichen grafischen Repräsentation, welche im `Diagram` realisiert ist. Diese beiden Komponenten werden im Folgenden dargestellt. Außerdem werden noch die Drop-Zonen erklärt, welche zur Bestimmung der ausgewählten *Kennzahlen* und *Dimensionen* benötigt werden.

10.1.1 DiagramDataModel

Die Klasse `DiagramDataModel` implementiert das Interface `INotifyPropertyChanged`, um andere Elemente über Veränderungen zu informieren. Diese enthält ein Property `Axis`, welches eine Zuordnung von den Achsen zu *Level* oder *Measures* darstellt. Bei diesem Property handelt es sich um ein Dictionary. Bei der Initialisierung melden sich die Achsen bei der zugehörigen Collection an. Hierfür holt sich die Achse ihre eigene Liste von `HierarchicalOLAPObjects` und meldet sich bei dieser als Beobachter für CollectionChanged-Events an. Des Weiteren enthält die Klasse einen booleschen Wert `isWorking`, welcher angibt, ob gerade eine Datenbankabfrage ausgeführt wird. Sobald eine Datenbankabfrage gestartet wird, wird dieser Wert vom `DiagramDataModel`

gesetzt. Weiterhin existiert ein `CellSet`-Property `CellSet`, welches das Ergebnis einer Datenbankabfrage aufnimmt.

10.1.2 Die Containerklasse Diagram

Die Klasse `Diagram` besteht aus zwei `Axis`, einer `Legend` und `DropAreaGroup`. Die Klasse hört auf das PropertyChanged-Event von dem `isWorking`-Property der Klasse `DiagramDataModel`. Wenn `isWorking` auf `true` gesetzt wurde, startet die Beschäftigt-Animation, wird es auf `false` gesetzt, wird diese Animation beendet. Dabei wird auf die funktionalen Bereiche des Diagramms ein `BlurEffect` gesetzt und Events blockiert. Ebenfalls enthält diese Klasse vier *Attached Dependency Properties*, `scaleX`, `scaleY`, `translateX` und `translateY`, welche die Ausrichtung und den Skalierungsfaktor des Diagramms enthalten. Jede dieser Eigenschaften wird von allen Elementen in der logischen Struktur der untergeordneten Elemente geerbt. Des Weiteren enthält die Klasse ein kleines Menü, welches in der linken unteren Ecke eines Diagramms angezeigt wird. Dieses ist in der Lage, kleinere Funktionen auszuführen, welche im Folgenden aufgelistet werden:

- Legende einblenden
- Drop-Zonenanzeigen lassen
- Diagramm schließen
- Speichern und Laden von Diagrammzuständen

10.1.3 DropAreaGroup und DropArea

Die Klasse `DropAreaGroup` wird von der Klasse `Grid` abgeleitet und kann somit eine beliebige Anzahl an `DropArea`s steuern. Die am meisten genutzte Methode ist die boolesche `Check(DropArea area, HierarchicalOLAPObject data)`-Methode, welche `true` zurückgibt, wenn das übergebene `HierarchicalOLAPObject` in die angegebene `DropArea` und als Kombination zu den anderen `HierarchicalOLAPObject`s in den anderen `DropArea`s passt. Das heißt, dass mit diesen `HierarchicalOLAPObject`s ein Diagramm erstellt werden kann. Die `DropAreaGroup` behandelt Drag and Drop-Events und benutzt die `Check`-Methode für die Überprüfung. Außerdem ist in dieser Klasse die Funktionalität für die Löschenfunktionen implementiert. Die Klasse `DropArea` definiert die grafische Darstellung einer zu einer Achse zugeordneten Drop-Zone. Diese Klasse enthält ebenfalls eine Liste von `HierarchicalOLAPObjects`. Ansonsten enthält diese Klasse keine weitere Logik.

10.1.4 Axis und AxisNode

Die Klasse `Axis` enthält ein Property `DropAreaTyp` vom Typ `DropAreaTyp`, welches angibt, um welchen Typ von Achse es sich handelt. `DropAreaTyp` ist dabei eine Aufzählung, welche die Werte `XAxis`, `YAxis`, `Color`, `Size` und `Animation` enthält, wobei der `DropAreaTyp` der Klasse `Axis` nur `XAxis` oder `YAxis` sein kann. Weiter hat die Achse eine `Translatetransform`, dessen X- und Y-Properties an die Properties `TranslateX` und `TranslateY` des Diagramms gebunden sind. Handelt es sich bei der Achse um eine X-Achse, wird das Binding des Property X der `TranslateTransform` entfernt. Stellt die Achse eine Y-Achse dar, wird analog dazu verfahren, jedoch mit dem Y-Property. Die Achse hört auf PropertyChanged-Events, welche vom `DiagramDataModel` abgefeuert werden, reagiert dabei allerdings nur auf Änderungen der Eigenschaft `CellSet`. Diese Änderung sorgt dafür, dass sich die *Measure*-Achse aufbaut. Die *Dimension*s-Achse ist in der Lage, sich noch vor der *Measure*-Achse aufzubauen, da die *Node*s der *Level* sich nicht verändern können. Der Aufbau der *Dimension*s-Achse wird gestartet, sobald ein neues *Level* in die zur *Dimension*s-Achse gehörende Liste im Dictionary hinzugefügt wird. Wird das *Level* aus dieser Liste entfernt, wird die Achse wieder gelöscht. Wenn `ScaleX` und `ScaleY` sich verändern, werden die Abstände zwischen den `AxisNodes` angepasst oder gegebenenfalls ausgeblendet. Dieser Vorgang wird nun für die X-Achse erläutert, der Vorgang für die Y-Achse läuft analog dazu.

Zuerst müssen einige Variablen definiert werden:

- $s =$ Skalierungsfaktor
- $W =$ Diagrammbreite
- $w =$ Beschriftungsbreite (`AxisNode`)
- $c =$ Anzahl von Beschriftungen auf der Achse

Der Abstand d zwischen den einzelnen Beschriftungen wird mit folgender Formel berechnet:

$$d = \frac{s \cdot W - w \cdot c}{c - 1} \tag{10.1}$$

Dabei wird nur jedes i-te Element angezeigt, wobei i wie folgt berechnet wird:

$$i = \left\lfloor \frac{c \cdot w}{W} \right\rfloor \tag{10.2}$$

Die *DrillDown*- und die *RollUp*-Funktionen sind so umgesetzt worden, dass jeweils nur eines der *Level*, die einer Achse zugeordnet sind, ausgetauscht wird. Diese Änderung bewirkt, dass eine neue Datenbankabfrage gestellt und dass anschließend ein neues `CellSet` für die Darstellung verwendet wird. Es werden also sowohl die Achsen als auch

das Diagramm von neuem erstellt. Bei einem globalen *DrillDown* bzw. *RollUp* wird das ausgewählte *Level* daher durch sein jeweiliges Kind-Element bzw. durch sein Parent-Element ersetzt. Dieses Vorgehen muss für den lokalen *DrillDown* variiert werden, da nur ein Ausschnitt des nächsttieferen Levels angezeigt werden soll. Zu diesem Zweck wurde die *Online Analytical Processing*-Abstraktionsschicht um die Klasse `Partiallevel` erweitert. Sie erbt alle Eigenschaften eines *Level*s und kann folglich ebenso verwendet werden wie ein *Level*. Die Data-Eigenschaft wird jedoch leer belassen; stattdessen werden der `Nodes`-Eigenschaft des `PartialLevel`s die `ChildNodes` der ausgewählten `Node`s zugewiesen. Das ausgewählte *Level* wird in diesem Fall also durch solch ein `PartialLevel`-Objekt ersetzt.

10.1.5 DiagramState und DiagramStateList

In Abbildung 9.3 sind die für die Zustandsspeicherung benötigten Klassen dargestellt. Die Klasse `DiagramState` stellt eine Datenstruktur zur Verfügung, die es ermöglicht, einen Diagrammzustand zu speichern. Wie aus Abbildung 9.3 hervorgeht, enthält sie dazu mehrere `Properties`. Das `Dictionary Axis` enthält die Zuweisung von `HierarchicalOLAPObject`s zu den Achsen bzw. *Drop-Zonen* im Diagramm. Um einen kompletten Diagrammzustand wiederherstellen zu können, speichern die `Properties ScaleProperty` und `TranslateProperty` die Skalierungs- und Verschiebungsfaktoren des gespeicherten Diagramms. Weiterhin wird mit dem `Property Thumbnail` ein Screenshot zur Verfügung gestellt, sodass dass das Auffinden eines gewünschten Zustandes in der Liste von Zuständen erleichtert wird.

Die Liste der Zustände wird im Diagramm mit Hilfe des neuen Properties `DiagramStatesProperty` gespeichert. Durch die Methode `AddState` wird dieser Liste der aktuelle Diagrammzustand hinzugefügt, wobei die Methode durch den Diagramm-Menüpunkt *Save* aufgerufen wird. `RemoveState` und `RemoveStateAt` können genutzt werden, um Zustände aus der Liste zu entfernen, wobei der Unterschied der beiden Methoden darin besteht, dass `RemoveState` als Parameter einen `DiagramState` erhält und diesen aus der Liste entfernt, und `RemoveStateAt` ein `int` übergeben wird, der den Listen-Index des zu löschenden Zustandes angibt. Mit den Methoden `SetCurrentDiagramToState` und `SetCurrentDiagramToStateAt` kann der aktuelle Zustand eines Diagramms auf einen zuvor gespeicherten Zustand gesetzt werden.

Die Klasse `DiagramStateList` stellt die Liste der gespeicherten Zustände visuell dar. Wird im Diagramm-Menü *Load* ausgewählt, legt sich eine Ebene über das Diagramm, in der die Screenshots der gespeicherten Zustände zur Auswahl dargestellt werden. Per Multitouch kann nun ein Zustand ausgewählt werden. Im Diagramm wird die Methode `SetCurrentDiagramToState` mit dem entsprechenden Diagrammzustand als Parameter aufgerufen. Für den Fall, dass viele Diagrammzustände gespeichert wurden, ist die `DiagramStateList` per Multitouch verschiebbar.

10.1.6 Legend

Im Konstruktor wird an den Eventhandler `DataContextChanged` der Legende die Methode `DiagramLegend_DataContextChanged` angehängt. Ebenso wird an den `SizeChanged`-Eventhandler die Methode `Legend_SizeChanged` angefügt.

`DiagramLegend_DataContextChanged` sorgt für die Anbindung der Legende an das `DiagramDataModel` des Diagramms. Das Handling der vom `DiagramDataModel` gesendeten Events geschieht von da an in der Methode `DiagramDataModel_PropertyChanged`.

Der Farbreiter verfügt bei Bedarf über einen Scrollbalken, sodass durch die Liste der angezeigten Farbe-Bezeichner-Paare navigiert werden kann.

Der Inhalt des Größenreiters wird stets an die aktuelle Größe des Diagramms angepasst. Dies wird durch die Methode `Legend_SizeChanged` ausgelöst, in der wiederum die Methode `GenerateSizeModel` aufgerufen wird.

`GenerateSizeModel` nutzt die aktuelle Größe der Legende zur Berechnung der maximal anzuzeigenden Größe-Bezeichner-Paare und berechnet anhand dieser die anzuzeigenden Punktgrößen und die dazugehörigen Werte. Die Berechnung der Punktgröße findet mit der folgenden Formel statt:

$$PointSize = \frac{maxPointSize - minPointSize}{maxElem - 1} \cdot i + minPointSize \tag{10.3}$$

Dabei sind die folgenden Größen beteiligt:

maxPointSize	*Höchster Wert der anzeigbaren Punktgrößen*
minPointSize	*Niedrigster Wert der anzeigbaren Punktgrößen*
maxElem	*Maximal anzeigbare Elemente in der Legende*
i	*Das i-te Element in der Liste*

In der folgenden Formel werden die zu den Punktgrößen gehörenden Werte der darzustellenden Kennzahl berechnet:

$$Value = \frac{CellSetMax - CellSetMin}{maxElem - 1} \cdot i + CellSetMin \tag{10.4}$$

Mit den folgenden Größen:

CellSetMax	*Höchster Wert des Wertebereichs der anzuzeigenden Kennzahl*
CellSetMin	*Niedrigster Wert des Wertebereichs der anzuzeigenden Kennzahl*
maxElem	*Maximal anzeigbare Elemente in der Legende*
i	*Das i-te Element in der Liste*

10.1.7 Zeichenfläche

Das Diagramm wird mit Hilfe der Methode `UpdateCanvas` erstellt, die aufgerufen wird, wenn sich das `DiagramDataModel` ändert. Diese benötigt als Parameter ein gültiges `DiagramDataModel`, welches ein `CellSet` beinhaltet. Innerhalb der Methode wird zunächst geprüft, ob ein gültiger Datensatz vorliegt. Als gültig wird anerkannt, wenn sowohl *Measure* wie auch *Dimension* auf den Positionsachsen liegen, aber auch ein `CellSet` vorhanden ist. Im nächsten Schritt werden die alten Datenpunkte im Diagramm gelöscht, da diese nicht mehr gültig sind. Die Berechnung der Farbwerte der `DiagramPoint`s geschieht durch die Methode `CreateColorGradient` der Unterstützungsklasse `DiagramUtils`. Ebenso stellt die Klasse die Methode `CreateSizes` zur Bestimmung der Größen der Datenpunkte bereit. Nachdem sowohl Größe wie auch Farbe für jeden Datenpunkt bestimmt wurden, erfolgt die Berechnung der Position jedes Punktes im Diagramm. Damit die Legende das Farbmodell erhalten kann, wird durch das Setzen des `ColorModel`s ein PropertyChanged-Event gefeuert.

10.1.8 DiagramUtils

Die Klasse `DiagramUtils` ist als Unterstützungsklasse zu sehen, die Methoden für Farbberechnung, Größenberechnung und Positionsbestimmung bereit hält. Für die Bestimmung der Farbe von Datenpunkten stehen zwei Farbverläufe zur Verfügung. Für Kennzahldaten, also Daten mit Zwischenwerten wird ein Farbverlauf von grün über gelb nach rot gewählt. Für Dimensionswerte ist ein größeres Farbspektrum von rot nach blau über zehn Stützpunkte gewählt worden. Es ist dabei sichergestellt, dass in jeder der zehn Klassen in etwa gleich viele Elemente enthalten sind, da Dimensionswerte immer im gleichen Farbabstand dargestellt werden müssen. Die Bestimmung der exakten Farben erfolgt durch lineare Interpolation.

Die Bestimmung der Größe erfolgt ebenfalls durch lineare Interpolation zwischen Minimum- und Maximum-Wert der Achse.

Weiterhin enthält diese Klasse noch eine Hilfsmethode zur Berechnung des Skalierungsfaktors, welcher für die Positionierung von Datenpunkten bei *Measure*s benötigt wird, und eine Methode zur Bestimmung des Abstandes von zwei Punkten bei Dimensionswerten.

Die Methode `PrintToVisual` wird genutzt, um eine beliebige Erweiterung der Klasse `UIElement` in eine Instanz der Klasse `Visual` umzuwandeln. Dies wird z. B. benötigt, um Screenshots der aktuellen Diagrammansicht zu generieren, die in der Zustandsspeicherung verwendet werden.

10.1.9 AnimationModel und Animationplayer

Die *WPF* umfasst bereits Funktionen, mit deren Hilfe Properties animiert werden können. Hierzu muss eine Animation des entsprechenden Typs erstellt werden. Für die X- und Y- Koordinaten eines `DiagramPoints`, sowie für seine Breite und Höhe, werden `DoubleAnimationUsingKeyFrames` verwendet. Solch einer Animation können nach und nach verschiedene Werte für das Property zugefügt werden, die bei ihrem Abspielen automatisch durchlaufen werden. Für jedes dieser Properties wurde eine Animation in die Klasse `DiagramPoint` eingefügt. Mit der Methode `AddKeyFrame` werden dem `DiagramPoint` Werte übergeben, welche als `KeyFrames` an die Animation angehängt werden. Die Werte zwischen zwei `KeyFrames` können durch die *WPF* auf verschiedene Weisen interpoliert werden, z. B. linear. Durch solch eine Interpolation kann beim Anwender allerdings leicht ein verfälschter Eindruck des Datenbestandes entstehen. Daher werden `DiscreteDoubleKeyFrames` verwendet, die keinerlei Interpolation vornehmen. Die Füllfarbe und die Sichtbarkeit der Diagrammpunkte haben nur diskrete Werte und können nicht mit einer `DoubleAnimation` animiert werden. Mit diesen Properties wird mit Hilfe von `ObjectAnimationUsingKeyFrames` ähnlich verfahren wie mit den anderen.

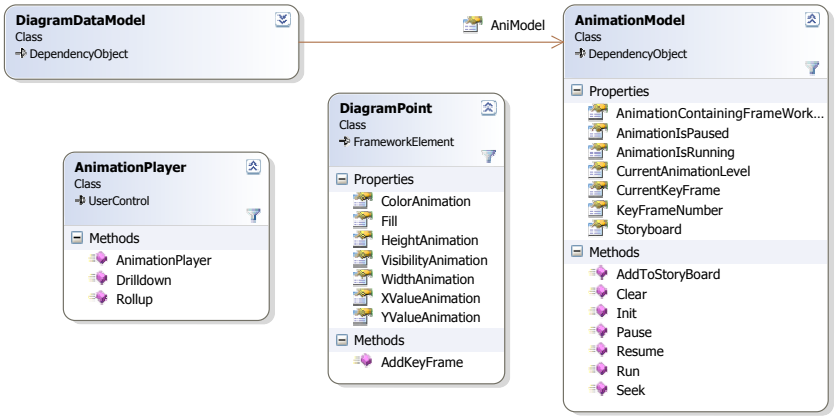


Abbildung 10.1: Klassen, die an der Punktanimation beteiligt sind.

Um eine Animation abzuspielen, wird in *WPF* ein *Storyboard* benötigt, das den zeitlichen Ablauf steuert. Ein solches Storyboard stellt den Kern des `AnimationModels` dar. Über die Methode `AddToStoryboard` werden alle AnimationsProperties eines `DiagramPoints` übernommen und dem Storyboard zugefügt. Die Methoden `Run`, `Pause`, `Resume` und `Seek` des `AnimationModels` verwenden die entsprechenden Methoden des Storyboards. Damit diese Kontrollfunktionen uneingeschränkt verwendet werden können, muss bei jedem der Aufrufe das Frameworkelement angegeben

werden, welches die zu animierenden Punkte enthält. Dieses kann über das Property `AnimationContainingFrameworkElement` des `AnimationModels` einmalig gesetzt werden, sodass es bei der Verwendung des `AnimationModels` sonst nicht weiter beachtet werden muss. Damit die Animationen aller Diagrammpunkte über ein einziges Storyboard kontrolliert werden können, müssen die Diagrammpunkte im *Namescope* des `AnimationContainingFrameworkElements` registriert werden. Sie erhalten daher in der Methode `AddToStoryboard` einen eindeutigen Namen, müssen aber, bevor das `AnimationModel` für den Aufbau einer neuen Animation verwendet wird, wieder aus diesem *Namescope* entfernt werden. Hierfür ist die Methode `Clear` vorgesehen, welche das `AnimationModel` und das `AnimationContainingFrameworkElement` in ihre Anfangszustände zurückversetzt.

Der Status des `AnimationModels` wird durch die beiden boolschen Properties `AnimationIs-Paused` und `AnimationIsRunning` beschrieben. Bevor die Animation gestartet wurde, haben beide den Wert `false`. In diesem Fall ist nur der Aufruf der `Run` Methode sinnvoll, denn Versuche, ein Storyboard, welches noch nie gestartet wurde, zu pausieren oder an eine bestimmte Stelle zu springen, sind nicht möglich. Wurde das Abspielen der Animationen durch einen Aufruf von `Run` gestartet, hat `AnimationIsRunning` den Wert `true`. Von nun an wird das Property `AnimationIsPaused` automatisch mit dem Zustand der `Clock` synchronisiert, welche das Storyboard intern steuert. Dies geschieht in dem Eventhandler `Storyboard_CurrentTimeInvalidated`. Ein Aufruf von `Pause` wird also direkt an das Storyboard weitergeleitet, und das Property nimmt daraufhin automatisch den richtigen Wert an. Ebenso verhält es sich, wenn die pausierte Animation fortgesetzt wird. Nachdem die Animation einmal vollständig abgespielt worden ist, wird der Status des `AnimationModels` im EventHandler `Storyboard_Completed` so gesetzt, dass `AnimationIsRunning false` ist, aber, anders als im Anfangszustand, `AnimationIsPaused true`. Ein Aufruf von `Seek` ist nur dann sinnvoll, wenn die Animation gerade pausiert oder noch nicht gestartet wurde. Die Methode erhält einen Zeitpunkt, an den das Storyboard springen soll. Da jedes `KeyFrame` für eine feste Dauer von einer Sekunde angezeigt wird, entspricht die Zeit in Sekunden gleichzeitig der Nummer des anzuzeigenden `KeyFrames`. Nachdem das Storyboard an die gewünschte Stelle gesprungen ist, wird es pausiert und kann über einen Aufruf der `Resume`-Methode fortgesetzt werden.

10.2 Desktop

Für die Realisierung des Desktops werden dem `MainWindow` zwei Canvas hinzugefügt, welche grafisch übereinander liegen. Eine Canvas dient als Command Layer zur Eingabe pfadbasierter Gesten, die andere als Container für die Diagramme und das Pie-Menü. Zunächst wird der Command Layer vorgestellt.

10.2.1 Command Layer

Die Befehlsschicht, auch Command Layer genannt, enthält einen `DiagramEventTrigger`, der selber unter anderem einen `Stroke` und einen `GestureRecognizer` enthält. Sobald die Befehlsschicht berührt wird, wird ein neues `Stroke`-Objekt erzeugt. Ein `Stroke`-Objekt ist ein grafisches Element, welches eine Menge von Punkten enthält. Diese Punkte sind in ihrer Reihenfolge miteinander verbunden. Sobald ein Move-Event aufgefangen wird, wird die neue Position zum `Stroke` hinzugefügt. Sobald die Berührung abgeschlossen ist, also ein TouchUp-Event aufgefangen wird, wird die letzte Position ebenfalls zum `Stroke` hinzugefügt und letztendlich dieser `Stroke` vom `GestureRecognizer` untersucht. Erkennt der `GestureRecognizer` ein Viereck, wird die Action `AddDiagramAction` ausgeführt. Des Weiteren stehen Methoden zum Aufrufen der Hilfefunktion und zum Schließen des Programms zur Verfügung, welche mit einer Kreisbewegung bzw. einer Wischgeste durchgeführt aufgerufen werden. Zusätzlich wird das `Stroke`-Objekt der Action übergeben. Die Klasse `AddDiagramAction` enthält dabei drei Hilfsmethoden, `Width`, `Height` und `Position`.

Die Methode `Width` ermittelt durch den übergebenen `Stroke` die Breite des Diagramms. Dafür werden der kleinste und der größte X-Wert aus dem `Stroke` ermittelt. Analog dazu verfährt die Methode `Height`, jedoch hier mit den Y-Werten. Um die Breite und Höhe zu berechnen wird das Minimum jeweils vom Maximum subtrahiert. Die Methode `Position` ermittelt die Position des Diagramms. Hierfür werden jeweils die minimalen Werte von den X- und Y-Werten ermittelt. Nachdem diese Werte berechnet werden, wird ein neues Diagramm erzeugt, die Größe dieses Diagramms gesetzt und mit einer `TranslateTransform` positioniert. Da die Diagramme aufeinander liegen können, gibt es das `LastZOrder`-Property, welches eine statische Variable ist. Es gibt an, welchen Z-Index das oben liegende Diagramm hat. Soll ein Diagramm in den Vordergrund geholt werden, so wird dieser Index inkrementiert und auf das Diagramm wird das `AttachedProperty Canvas.ZIndex` mit diesem Wert gesetzt. Ein Diagramm kann durch Berühren desselben in den Vordergrund geholt werden.

10.2.2 Einblendung des Menüs

Das Menü kann durch Auflegen des Handballens angezeigt werden. Die Erkennung der Hand wird mit Hilfe der Größeninformation der *Blobs* realisiert. In *TUIO* sind verschiedene Profile zur Übertragung von *Blobs* spezifiziert. Zur Zeit unterstützt *Community Core Vision* allerdings nur das Datenformat '/tuio/2Dcur', welches keine Größeninformationen enthält. Es besteht dennoch die Möglichkeit durch eine Option diese Informationen mitzusenden. *Community Core Vision* fügt die Größendaten dann an ein *TUIO*-Paket an. Dieses entspricht nicht der Spezifikation, wodurch das Programm evtl. bei einer neueren Version von *Community Core Vision* angepasst werden muss. Die Größeninformationen werden durch das Multitouch-Framework ausgewertet und als BigBlob-Event an die Komponenten weitergeleitet, falls es es sich um einen großen *Blob* handelt. Dieses Event wird im `MainWindow` bearbeitet, damit das Pie-Menü an der Position der Hand angezeigt wird. Um die Menüführung zu vereinfachen, wird das Menü nicht direkt an der Handposition angezeigt, sondern etwas höher. Zu Entwicklungszwecken kann das Menü auch über Rechtsklick eingeblendet und über Linksklick auf die Arbeitsfläche ausgeblendet werden.

10.3 Implementierung des Pie-Menüs

Das Pie-Menü ist das zentrale Element, das dem Benutzer zur Verfügung steht, um eine Auswahl der Daten zu treffen, die im Diagramm dargestellt werden sollen. Das Pie-Menü verfügt hierzu über verschiedene Ebenen, die entsprechend den Hierarchien in der Datenbank aufgebaut sind.

PiePanel

Die Klasse `PiePanel` ist eine Erweiterung eines `Panels` und stellt dem Pie-Menü verschiedene Dependency Properties zur Verfügung, die das Aussehen des gesamten Menüs beeinflussen. Neben den Eigenschaften `Background`, `Bordercolor` und `BorderThickness`, wodurch das Pie-Menü seine Farbgebung und Rahmengestaltung erhält, sind noch weitere Eigenschaften festzulegen. `PiePieceInnerRadius` definiert den inneren Radius des jeweiligen `PiePanels`. Mit `PiePieceWidth` wird die Breite des `PiePanels` festgesetzt. `PiePieceOuterRadius` ist die Summe von `PiePieceInnerRadius` und `PiePieceWidth`. Gleichzeitig ist `PiePieceOuterRadius` auch der innere Radius des nächst höher gelegenen `PiePanels`. Mit `VisibleAngle` wird eine Eigenschaft gesetzt, die angibt, wie groß der Winkel des `PiePanels`, der dargestellt und zur Berechnung der Anzahl der angezeigten Elemente verwendet wird,

sein muss. `VisibleAngleOffset` gibt einen Offset zum Winkel an, sodass das Pie-Menü eventuell auch am linken oder rechten Rand des Bildschirms angezeigt werden kann.

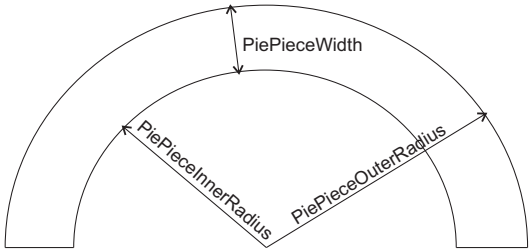


Abbildung 10.2: Zusammenhang der Radien im `PiePanel`

Des Weiteren steht das Property `ElementsAreLooped` als boolean-Wert, welches für die unendliche Drehbarkeit des Panels gesetzt werden muss. Das Property `ShowLeftRightButton` stellt die Möglichkeit zur Verfügung, die Schaltflächen zum Rotieren aus- bzw. einblenden zu lassen.

Elemente können über die Property `ItemsSource` dem `PiePanel` hinzugefügt werden. Zu jedem Element aus der `ItemsSource` wird ein entsprechendes *GUI*-Element erzeugt. Da der angezeigte Platz eventuell nicht alle Elemente fasst, wird mit der privaten Methode `CalcNumberOfVisibleItems` berechnet, wie viele Elemente dargestellt werden können. Sollten alle Elemente im verfügbaren Bereich Platz finden, wird ein boolean-Wert gesetzt, wodurch das `PiePanel` nicht rotierbar ist.

Wie im Entwurf zum Pie-Menü beschrieben ist, soll nur eine bestimmte Anzahl von Ringen sichtbar sein. Dies bedeutet, dass ein Ring eingeklappt werden kann und über die Property `ExpansionsState` angezeigt wird. Die Property ist ein `Enum`, welches die Werte `Collapsed`, `Expanded`, `Expanding` oder `Collapsing` annehmen kann. Die Methoden `Collapse` und `Expand` führen das Ein- bzw. Ausklappen eines Ringes durch. Hierbei wird eine Animation abgespielt.

Um Multitouch-Events und zu Testzwecken auch ohne Multitouch-Gerät Mouse-Events zu verarbeiten, stellt die Klasse `PiePanel` Methoden bereit, die auf eben diese Events reagieren. Für den Fall von `MouseDown`-, `MouseUp`-, `MouseMove`, `FingerMove`- und FingerUp-Events wird die Routing-Strategie *Tunneling* verwendet, um das `PiePanel` auf eine Rotation reagieren lassen zu können, ohne auf einem dargestellten Element ein Event auszulösen. So kann erreicht werden, dass das `PiePanel` auch rotiert wird, wenn die Rotation auf einem Button beginnt. Anderenfalls würde der eventuelle Klick/Touch auf das Element möglicherweise eine nicht gewünschte Aktion auslösen.

PieMenu

Das Aussehen des Pie-Menüs und das der darin liegenden Elemente ist in *XAML* programmiert. In der entsprechenden Code-Behind-Datei enthält das `Dictionary uielements` alle Elemente, die im `PieMenu` angezeigt werden sollen. Diese Elemente werden den `PiePanel`s in der entsprechenden Hierarchiestufe hinzugefügt. Die Datenquelle für die Erstellung der Hierachiestufen wird mittels der `DataSource`-Eigenschaft gesetzt. `CreateFakeModel` ist eine Methode, die ein `PieMenu` ohne Datenbankzugriff mit Elementen füllt, welches zu Testzwecken sinnvoll ist. Abbildung 10.3 zeigt eine Instanz der Klasse `PieMenu`.

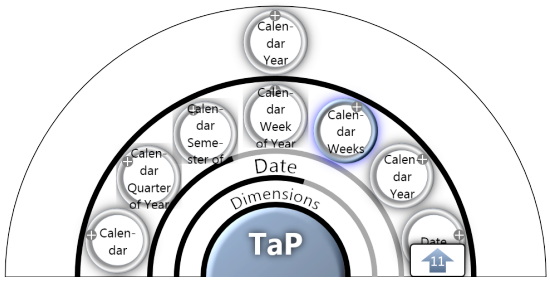


Abbildung 10.3: Das Pie-Menü hat einen inneren nicht drehbaren Ring und einen äußeren drehbaren Ring, was durch die Buttons signalisiert wird.

Die Klasse `PieMenu` registriert alle Elemente, die in ihr dargestellt werden sollen sowohl für Multitouch-Events als auch zu Testzwecken für Mouse-Events. Hierbei wird im Gegensatz zu `PiePanel` allerdings die Routing-Strategie *Bubble* verwendet, damit die auftretenden Events nicht an im VisualTree darunterliegende Elemente weitergegeben werden. Die wichtigsten Funktionalitäten, die `PieMenu` seinen Elementen verleiht, ist *Drag and Drop* und TouchUp. Die TouchUp-Eventbehandlung (für den Fall eines Tests ohne Multitouch-Gerät ist auch die Nutzung des MouseClick-Events implementiert) sorgt dafür, dass, sobald ein Element berührt wird, die eventuell vorhandene Hierarchieebene in einem weiteren `PiePanel` ausgeklappt wird und auch ein optisches Feedback an den Benutzer erfolgt, das anzeigt, welches Element gerade ausgewählt wurde (siehe Abbildung 10.3, bläulich hinterlegt *Dimension*). Die *Drag and Drop*-Funktionalität ermöglicht es dem Benutzer, ein Element auszuwählen. Hierzu muss der Anwender das entsprechende Element auswählen und mit dem Finger oder der Maus aus dem Pie-Menü herausziehen. Es wird eine Kopie erzeugt, die am Mauszeiger oder Finger angezeigt wird, und in die Drop-Zonen abgelassen werden kann.

Abbildung 10.4: *Drag and Drop* aus einem Pie-Menü

Eine Berührung des Basisbuttons des Pie-Menüs lässt alle Ebenen verschwinden.

ViewModel

Die Klasse `PieMenuModel` stellt das ViewModel der Klasse `PieMenu` dar. Jedes Element, das durch die Klasse `PieMenu` erstellt wird, speichert im `PieMenuModel` seinen Anzeigezustand. Die Klasse `OLAPViewModel` besitzt die Eigenschaft `IsSelected`, in dem der aktuelle Selektionszustand gespeichert ist. Zum Laden von Kind-Elementen wird ein `BackgroundWorker` eingesetzt. Neue Kind-Elemente werden durch die Methode `CreateChild` erzeugt, diese muss eventuell von Unterklassen überschrieben werden.

10.4 Implementierung des Multitouch-Frameworks

Dieser Abschnitt beschreibt die Umsetzung des Multitouch-Frameworks. Es wird zunächst die Anbindung der *Blob Detection* beschrieben und anschließend auf die Implementierung der Erkennung von Gesten eingegangen.

10.4.1 Blob

Die Struktur *Blob* repräsentiert eine Fingerberührung auf dem Tisch. Sie speichert in den Properties `Position`, `MotionSpeed`, `MotionAcceleration`, `Size` und `Speed` die Eigenschaften der Berührung auf dem Tisch. Außerdem wird durch `SessionId` eine eindeutige *ID* des Fingers gespeichert. Die Gleichheitsoperatoren `!=` und `==` sowie die `Equals`-Methode sind überladen worden, damit *Blobs* mit gleicher `SessionId` als gleich eingestuft werden.

10.4.2 TUIOServer

Mittels der Methode `Connect()` erstellt `TUIOServer` eine `UdpClient`-Instanz, die die von der Fingertracking-Software gesendeten *TUIO*-Daten empfängt und an die Methode `ProcessMessage()` weiterleitet. `ProcessMessage()` fügt die so erkannten *Blobs* in der Klasse `Multitouch` durch Aufruf der Methode `AddBlob()` ein. Ferner trägt die Methode auch dazu bei, dass beim Verschwinden eines *Blobs* von der Tischoberfläche dieser auch wieder aus der Liste der erkannten Finger entfernt wird.

10.4.3 Multitouch

Die wichtigste Funktion der Klasse `Multitouch` ist die Umwandlung der von der Fingertracking-Software erkannten *Blob*-Positionsveränderungen in Gesten und daraufhin das Feuern von Events, die diesen Gesten entsprechen.

Im Folgenden werden die erkennbaren Gesten und die daraufhin gefeuerten Events beschrieben.

TouchDown Das TouchDown-Event wird abgefeuert, wenn ein neuer Finger die Tischoberfläche berührt und die darunterliegenden *GUI*-Elemente keine weiteren *Blobs* in `BlobsProperty` haben. Damit wird gewährleistet, dass eine Berührung unabhängig von der Fingeranzahl erkannt wird.

TouchEnter Das TouchEnter-Event wird ausgelöst, wenn ein Finger auf der Tischoberfläche seine Position ändert und dabei auf einen Bereich eines *GUI*-Elements kommt, dass noch keinen *Blob* in der `BlobsProperty`-Liste hat.

TouchUp Ein TouchUp-Event wird gesendet, wenn ein Finger von der Tischoberfläche abgehoben wird und das darunterliegende *GUI*-Element nur eine *Blob*-Objektinstanz in der `BlobsProperty` hatte. Es wird hierdurch gewährleistet, dass eine Berührung unabhängig von der Fingeranzahl ist und ein TouchUp-Event nur ausgelöst wird, wenn alle Finger vom entsprechenden *GUI*-Element abgehoben wurden.

TouchLeave Das TouchLeave-Event wird abgefeuert, wenn der letzte Finger auf einem *GUI*-Element dieses verlässt und die `BlobsProperty` somit leer wird. Hierdurch wird gewährleistet, dass das TouchLeave-Event von der Fingeranzahl unabhängig ist.

TouchMove Das TouchMove-Event wird ausgelöst, wenn ein Finger auf der Tischoberfläche die Position ändert. Für jedes darunterliegende *GUI*-Element wird

der Schwerpunkt der Geste vor und nach der Veränderung berechnet. Der Schwerpunkt wird dabei mittels folgender Formel berechnet:

$$x = \sum_{k=0}^{n} \frac{X_k}{n}, \text{ für } n > 0, \text{ sonst } 0 \tag{10.5}$$

n ist die Anzahl der *Blobs* in `BlobsProperty` und x die X-Koordinate aus der `Blob.Position`-Property.

Die Differenz zwischen diesen zwei Punkten wird an die *GUI*-Elemente weitergegeben. Hierdurch wird gewährleistet, dass *Verschieben* unabhängig von der Anzahl der Finger ist und dessen Bewegungsrichtung erkannt wird.

Scale Wenn ein Finger auf der Tischoberfläche die Position ändert und der Finger dabei am weitesten Rechts, Links, Oben oder Unten auf dem *GUI*-Element steht, wird ein Scale-Event gesendet. Die Differenz zwischen altem und neuem Zustand wird an das *GUI*-Element weitergegeben.

Rotate Genauso wie `Scale` wird das Rotate-Event nur dann abgefeuert, wenn ein Finger auf der Tischoberfläche die Position ändert und der Finger dabei am weitesten Rechts, Links, Oben oder Unten auf dem *GUI*-Element steht. Im Unterschied zum Scale-Event wird jedoch die Winkeldifferenz zwischen dem alten Zustand der gegenüberliegenden Finger und dem neuen Zustand der gegenüberliegenden Finger an das *GUI*-Element weitergegeben.

BigBlob Das BigBlob-Event wird gefeuert, wenn die Größe eines *Blobs* einen spezifizierten Schwellwert überschreitet. Dieser kann über die Einstellungsdatei des Multitouch-Frameworks eingestellt werden. Der Schwellwert muss in Abhängigkeit zu den Einstellungen von *Community Core Vision* ermittelt werden. Bei einem TouchDown ist die *Blob*-größe anfangs meist recht klein, da sich diese erst durch Druck aufbaut. Sobald die *Blob*-größe den Schwellwert überschreitet, wird ein BigBlob-Event gesendet.

10.4.4 Touch

Zur Bereitstellung der Funktionalität von *Touch-Capture*, die der *WPF*-Funktion Mouse-Capture nachempfunden ist, implementiert die Klasse `Touch` die Methoden `CaptureTouch` und `ReleaseTouchCapture`. Da Mouse-Capture allerdings bekannt ist, wird auf die Erklärung dieser Methoden nicht weiter eingegangen. Um jedoch Kollisionen zwischen dem nativen Mouse-Capture und dem eigenen *Touch-Capture* zu verhindern, musste in der Klasse `Multitouch` das bool'sche Property `isMultittouch` eingeführt werden. Ist dieses auf `true` gesetzt, werden nur Multitouch-Event-Handler eingebunden, andernfalls nur Mouse-Event-Handler.

Um die Funktion des *Drag and Drop* auch für `Touch` nachzubilden, implementiert die Klasse `Touch` die statische Methode `DoDragAndDrop`. Diese Methode ist der Methode `DoDragDrop` von der `DragDrop`-Klasse aus *WPF* sehr ähnlich, es wird lediglich ein zusätzlicher Parameter vom Typ `Point` übergeben, welcher die Mitte des zu ziehenden Elementes relativ zum *TaP*-Window angibt. Die Methode erstellt ein `DragObject`, fügt es in das Fenster ein und setzt es an die Position des übergebenen `Point`s. Das `DragObject` „haftet“ sich dann an den Finger, bzw. bei Maussteuerung an die Maus. Wird der Finger oder der Mauszeiger verschoben, führt das `DragObject` einen `HitTest` durch und findet dadurch heraus, welches *WPF*-Element unter ihm liegt. Dieses Element wird dann mit dem Element des vorherigen `HitTest`s verglichen. Wenn das neue Element dem alten entspricht, wird ein DragOver-Event abgefeuert. Sind dies beiden Elemente jedoch verschieden, wird beim alten Element ein DragLeave-Event und beim neuen Element ein DragEnter-Event abgefeuert. Bei einem DragOver-Event überprüft das darunter liegende Element, ob ein Drop auf ihm möglich ist. Wenn es nicht möglich ist, wird das Drag-Objekt auf halbdurchsichtig gesetzt, ansonsten bleibt die visuelle Erscheinung unverändert. Eine Anforderung ist es, dass die *WPF*-Events genutzt werden sollen. Da diese Klassen jedoch zum *WPF*-Framework gehören und als `internal` deklariert sind, lassen sich deren Objektinstanzen nur per *Reflection* erzeugen. Dies wird durch die Methode `CreateDragEventArgs` geregelt.

10.4.5 DragObject

Eine Instanz der Klasse `DragObject` ist eine visuelle Repräsentation eines zu ziehenden Elementes. Der `DataContext` des `DragObject`s ist dabei stets der gleiche, den auch das Element hatte, bei dem der *Drag and Drop* gestartet wurde. Die Funktion `TouchMove` des `DragObject`s sorgt für die Auslösung der in der Klassenbeschreibung der Klasse `Touch` (siehe 10.4.4) erwähnten Drag-Events, sodass die optisch unter dem `DragObject` liegenden Elemente das darüber *schwebende* Element bemerken. Im Fall eines erfolgreichen Drops wird dann der `DataContext` des `DragObject`s für die weitere Datenverarbeitung verwendet.

10.5 Implementierung der Datenbankanbindung

Damit die Anwendung unabhängig von der Art der Datenbank arbeiten kann, benutzen und liefern alle Methoden Instanzen der *Online Analytical Processing*-Abstraktionsschicht zurück. Die Metadaten der Datenbank werden beim ersten Aufruf nahezu komplett geladen. Ausnahme hierbei sind die Knoten, da es hiervon eine sehr große Anzahl geben kann. Für alle weiteren Metadaten und Abfragen gibt es vier öffentliche Methoden, welche im Folgenden genauer erklärt werden.

Zu beachten ist, dass sowohl *MUSTANG* als auch *ADOMD.NET* bereits einige Funktionen, beispielsweise für bestimmte Datenbankabfragen, bieten. Damit diese auch weiterhin verwendet werden können, haben die Objekte der *Online Analytical Processing*-Abstraktionsschicht ein `Data` Property vom Typ `object`. Dieses könnte im Falle einer *Dimension* beispielsweise ein `DimensionVO` von *MUSTANG* oder eine *ADOMD.NET*-`Dimension` aufnehmen. Operationen auf einem `OlapObjekt` müssen daher fast immer durch die entsprechende Implementierung der Datenbankanbindung vorgenommen werden. So wird gewährleistet, dass sich die Objekte der Abstraktionsschicht nicht voneinander unterscheiden, selbst wenn unterschiedliche Datenbanken verwendet werden.

10.5.1 Abrufen von Kind-Elementen

Die *Online Analytical Processing*-Datenbank ist in einer komplexen hierarchischen Struktur aufgebaut. Werden die Kinder eines Elements angefordert, so werden diese ggf. durch die Datenbankimplementierung nachgeladen. Dies geschieht durch einen Aufruf der Methode `GetChildren` (s. Listing 10.1).

```
1  private List<HierarchicalOLAPObject> _Children;
2
3  public override List<HierarchicalOLAPObject> Children
4  {
5      get
6      {
7          // ueberpruefe, ob bereits Kind-Elemente geladen worden sind
8          if (_Children == null)
9          {
10             // falls nicht, fordere sie von der Datenbank an
11             _Children = DatabaseFactory.getDBInstance().GetChildren(this);
12         }
13         return _Children;
14     }
15     ...
16 }
```

Listing 10.1: Das Abrufen von Kind-Elementen wird ggf. an die Datenbankinstanz delegiert

Beide Implementierungen unternehmen zuerst eine Überprüfung zu dem übergebenen Datentyp, um diesen korrekt behandeln zu können. Handelt es sich bei dem Argument der `GetChildren`-Methode um ein `OlapObject`, das nie Kinder haben kann (z. B. einen `Cube`), wird eine leere Liste als Ergebnis geliefert.

ADOMD.NET Beim Abruf der Kind-Elemente eines Level muss in diesem Falle zunächst eine Prüfung auf den *ADOMD.NET*-Datentyp vorgenommen werden, da zusätzlich *Hierarchie* aus *ADOMD.NET* als *Level* behandelt werden. Da der Aufruf der *Level* einer *Hierarchie* alle *Level*, unabhängig von der Tiefe im Baum, zurück liefert, müssen diese vorerst in eine hierarchische Struktur umgewandelt werden. Dies wird mit Hilfe einer Sortierung auf die Eigenschaft `LevelNumber`

realisiert. Anschließend werden alle *Level* rekursiv geladen, sodass kein erneutes Laden von *Level* notwendig ist.

Im Falle des Aufrufens der Kinder einer Dimension muss hier beachtet werden, dass bei der Verwendung von *ADOMD.NET Hierarchien* und nicht *Level* als Kinder geliefert werden.

MUSTANG Für die *MUSTANG*-Anbindung muss der Abruf von Kind-Elementen nur für *Dimensionen*, *Level* und *Nodes* implementiert werden. Da Kennzahlengruppen in *MUSTANG* nicht explizit behandelt werden, kann es nicht vorkommen, dass ein `Measure`-Objekt Kinder hat.

Hier kann problemlos auf die Funktionen der zugrunde liegenden *MUSTANG*-Klassen zurückgegriffen werden. Die Ergebnisse müssen lediglich in neue Objekte der Abstraktionsschicht verpackt werden.

10.5.2 Abrufen kompatibler Elemente

Innerhalb eines Datenwürfels kann es möglich sein, dass bestimmte *Kennzahlen* nicht zu allen *Dimensionen* kompatibel sind. Eine Abfrage trotz Inkompatibilität würde ein ungültiges Ergebnis zurückliefern. Um dieses zu vermeiden, liefert die Methode `GetCompatibles` alle *Kennzahlen* und *Dimensionen*, welche zu den übergebenen *Kennzahlen* und *Dimensionen* kompatibel sind.

ADOMD.NET Bei Instanziierung der *ADOMD.NET*-Implementierung wird direkt ein Dictionary mit *Kennzahlen* und ihren kompatiblen *Dimensionen* erstellt. Die Abfrage der Kompatibilität wird mit Hilfe von Metadaten des Datenwürfels, welche im *XML*-Format vorliegen, durchgeführt. Anschließend wird das *XML*-Dokument geparst und das Dictionary kann gefüllt werden. In der Methode `GetCompatibles` werden bei Aufruf mit *Language Integrated Query (LINQ)* die passenden Daten herausgesucht (s. Listing 10.2).

```
1 ...
2 else if (olapObject.OLAPTyp == OLAPType.Dimension)
3 {
4   // Durchsuche das Dictionary nach kompatiblen Kennzahlen
5   var adomdMeasures = (from i in measureDimensionRelations
6                        where i.Value.Contains(olapObject.Data as Adomd.Dimension
                            )
7                        select i.Key).Distinct().ToList();
8 ...
```

Listing 10.2: LINQ-Statement zum Auffinden kompatibler Kennzahlen für eine Dimension

MUSTANG *MUSTANG* ermöglicht es bereits, direkt abzufragen, welche *Dimensionen* zu einer bestimmten Kennzahl kompatibel sind. Um die Zuordnung auch andersherum zu gewährleisten, wird auch bei der Instanziierung der *MUSTANG*-Implementierung ein `Dictionary` angelegt, welches allen Kennzahlen die zugehörigen *Dimensionen* zuordnet. Sollen die passenden Kennzahlen zu einer *Dimension* gefunden werden, wird dieses `Dictionary` mit Hilfe einer *LINQ*-Abfrage durchsucht.

10.5.3 Abrufen von Knoten

Jedes *Level* hat eine Anzahl von *Knoten*, welche die Koordinaten in einem Datenwürfel repräsentieren. Diese *Knoten* sind für die Darstellung und Abfrage wichtig.

ADOMD.NET Der Abruf von Knoten muss für *Hierarchien* und *Level* unterschiedlich geschehen, da eine *Hierarchie* keine `Member`-Objekte hat. Stattdessen muss hier ein *Knoten* mit dem `DefaultMember` der *Hierarchie* erstellt werden. Dieser repräsentiert dann die *Hierarchie* als ein *Level*.

MUSTANG Der Abruf von Knoten ist über den `DimensionService` von *MUSTANG* problemlos möglich. Zu den erhaltenen Resultaten werden lediglich neue Repräsentationen aus der *Online Analytical Processing*-Abstraktionsschicht erzeugt.

10.5.4 Abrufen von Daten

Um ein Diagramm mit Daten füllen zu können, wird mindestens eine *Kennzahl* und eine Anzahl von *Knoten* benötigt. Die Methode `GetData` liefert ein `CellSet` zurück, welches mit Datenpunkten gefüllt ist. Bei beiden Datenbank-Typen werden die gelieferten Daten gespeichert, sodass ein erneuter Abruf auf dieselben Daten nicht mehr auf die Datenbank erfolgen muss und somit die Daten viel schneller zurückliefert.

ADOMD.NET Bei dem Abruf von Daten mit *ADOMD.NET* muss ein *MDX*-Befehl mit den übergebenen Metadaten generiert werden. Hierfür werden alle *Kennzahlen* auf die Zeilen und alle Knoten auf die Spalten der Abfrage gelegt.

MUSTANG Der *MDX*-Befehl wird automatisch durch *MUSTANG* generiert. Dazu müssen lediglich *Knoten* und *Kennzahlen*, die für die Abfrage benötigt werden, übergeben werden. Anschließend liefert *MUSTANG* ein `CubeVO`-Objekt als Ergebnis. Dieses dient als Grundlage, um ein `Cellset` aufzubauen, wobei die Zahlenwerte jeweils unter der zugehörigen *Kennzahl* in einem `Dictionary` abgelegt werden.

Kapitel 11

Erweiterbarkeit

In diesem Kapitel werden Erweiterungspunkte für *TaP* genannt. Jeder der folgenden Abschnitte beschreibt das empfohlene Vorgehen zur Erweiterung um eine bestimmte Funktion. Zuerst wird beschrieben, wie eine zusätzliche Datenbank-Anbindung gestaltet werden kann. Im Anschluss daran werden mögliche Erweiterungen für das Multitouch-Framework erläutert. Darüber hinaus wird beschrieben, wie das Pie-Menü durch eine andere hierarchische Menüform ausgetauscht werden kann. Schließlich wird im vorletzten Abschnitt die Erweiterung um neue Diagrammformen thematisiert, während sich der letzte Abschnitt mit Verbesserungen des bestehenden Diagramms beschäftigt.

11.1 Datenbank

Die Datenbank-Anbindung ist um weitere Datenbank-Typen erweiterbar. Für die Anbindung eines weiteren Datenbank-Typs muss eine neue Klasse, die die abstrakte Klasse `AbstractDatabase` implementiert, erstellt werden. Dies betrifft alle Methoden und Properties der Klasse, außer der Methode `GetMinMax`, da diese speziell für Datenbanken mit einer MDX-Schnittstelle implementiert wurde. Somit müsste diese Methode überschrieben werden. Es ist vorstellbar, dass nicht nur Datenbanken Lieferanten für Daten sein können, sondern z. B. Datenstrommanagementsysteme.

11.2 Multitouch-Framework

Um das Multitouch-Framework zu erweitern, könnte eine *Attached Dependency Property* in die `Multitouch`-Klasse eingefügt werden, in der gespeichert wird, welche Events ein Element empfangen soll. Hierdurch könnte eine Performanz-Verbesserung erreicht werden, da nur noch die erwarteten Interaktionen erkannt werden sollen. Eine weitere Verbesserung der Performanz könnte dadurch erlangt werden, indem nicht jedes *TUIO*-Event zur Berechnung herangezogen wird, sondern nur die Events,

die relevante Veränderungen aufweisen. Um die Usability zu verbessern, könnte ein visuelles Feedback implementiert werden, sodass der Benutzer merkt, welche Stelle des Touchscreens er berührt.

Damit dem Multitouch-Framework weitere Gesten hinzugefügt werden können, müssen folgende Schritte durchgeführt werden:

1. *Attached Dependency Properties* müssen in der `Multitouch`-Klasse definiert werden.
2. Die Methoden `AddBlob()`, `RemoveBlob()` und `MoveBlob()` müssen um die Erkennung der neuen Geste erweitert werden.
3. `eventQueue` muss mit den entsprechenden Events gefüllt werden, sodass sie abgefeuert werden können.

Eine weitere Möglichkeit, das Programm multitouch-fähig zu machen, ist der Umstieg auf .NET 4 oder das Surface SDK. Bei beiden Frameworks ist die Multitouch-Fähigkeit bereits integriert.

11.3 Einbindung eines weiteren hierarchischen Menüs

Zur Einbindung eines anderen hierarchischen Menüs sollte das `OlapViewModel` verwendet werden. Das `PieMenuModel`, als Datenmodel für das Pie-Menü, ist von der Klasse abgeleitet und implementiert spezielle Eigenschaften, die für das Pie-Menü relevant sind. Damit das Menü mit dem Punktdiagramm interagieren und damit als Auswahlmenü verwendet werden kann, muss noch die *Drag and Drop*-Unterstützung eingebunden werden. Hierzu ist die Methode `Touch.DoDragAndDrop` auszuführen. Als Datenobjekt ist das ausgewählte *OLAP*-Objekt des *OLAP*-Models zu verwenden. Beispielhaft ist dies im Folgenden Quellcode dargestellt.

```
1 DataObject data = new DataObject("TaPData", hierachicalOlapObject);
2 Touch.DoDragAndDrop(draggedUiElement, data,
3     mouseEvent.GetPosition(Application.Current.MainWindow));
```

Listing 11.1: Erstellung eines DragObjects

Im *OLAP*-Datenbanksystem kann es vorkommen, dass nicht alle *Dimensionen* und *Kennzahlen* zueinander passen. Gerade bei der verwendeten *Trinovis*-Datenbank kann dies häufig vorkommen. Es ist aus diesem Grund sinnvoll, die kompatiblen *OLAP*-Elemente hervorzuheben. Zur Anzeige, ob ein Element zur bisherigen Diagrammbelegung kompatibel ist, sollte die Eigenschaft `IsValid` verwendet werden. Diese Eigenschaft wird durch die Methode `CheckCompatibility` gesetzt, welche als Übergabeparameter eine Liste der kompatiblen Elemente bekommt. Die kompatiblen Elemente werden

durch die `GetCompatibles` Methode der Datenbankschnittstelle bereitgestellt. Das folgende Listing enthält einen beispielhaften Aufruf dieser Methode.

```
List<HierarchicalOLAPObject> compatibles = Cube.GetCompatibles(currElementsOnDiagram);
rootOlapViewModel.CheckCompatibility(compatibles);
```

Listing 11.2: Beispielhafter Aufruf der Check-Compatibility Funktion

11.4 Einbindung weiterer Diagramme

Es besteht die Möglichkeit, weitere Diagramme in *TaP* einzubinden. Im Folgenden wird ein konzeptueller Ablauf gegeben, wie diese Einbindung gestaltet werden kann.

Es wird empfohlen, das neue Diagramm mit einer neuen Geste zu verknüpfen. Diese kann in die *XAML*-Datei des `MainWindows` eingebunden werden. Listing 11.3 zeigt, wie ein entsprechender Trigger angesprochen werden muss. Wie zu erkennen ist, wird für jede erkennbare Geste eine Action angebunden.

```
<Canvas Background="#8FFFFFFF" >
    <i:Interaction.Triggers>
        <tap:DiagramEventTrigger>
            <tap:AddNewDiagrammAction TargetName="desktop"
             tap:DiagramEventTrigger.EventName="newGesture"/>
        </tap:DiagramEventTrigger>
    </i:Interaction.Triggers>
</Canvas>
```

Listing 11.3: Hinzufügen eines Actiontriggers zur Erstellung eines neuen Diagramms

Die Action muss abgeleitet sein von `TargetTriggerAction<Panel>`. Diese abstrakte Klasse schreibt unter anderem die Methode `Invoke` vor, die bei einer bei einer Action aufgerufen wird. Das übergebene Objekt ist der `Stroke`, welcher von der Gestenerkennung gesetzt wird. Anhand des `Strokes` kann die Position und die Größe des neuen Diagramms ermittelt werden. Dies ist im folgenden Listing beispielhaft dargestellt:

```
// ... Process Stroke...

// Create Diagram
Diagram diagram = new Diagram();
diagram.Width = stroke.Width();
diagram.Height = stroke.Height();
Point position = stroke.Position();
Canvas.SetLeft(diagram, position.X - deltaX);
Canvas.SetTop(diagram, position.Y - deltaY);
this.Target.Children.Add(diagram);
```

Listing 11.4: Berechnung der Position und Größe eines Diagramms mit Hilfe des `Stroke`

Damit das neue Diagramm ein ähnliches Verhalten wie das Punktdiagramm erhält, steht das Behavior `Resizable` zur Verfügung. Dieses kann einfach auf visuelle Elemente angewandt werden. Hierdurch wird es ermöglicht das Diagramm zu verschieben und seine Größe zu ändern.

Wenn das Diagramm mit dem bestehenden Pie-Menü interagieren soll, muss es in der Lage sein, `DragObjects` aufzunehmen. Das Property `DataObject` enthält ein `HierarchicalOlapObject`, welches einer ausgewählten *Kennzahl* oder *Dimension* entspricht. Um später eine Datenbankabfrage durchführen zu können, müssen diese Objekte vorgehalten werden. Damit Daten aus der Datenbank gelesen werden können, kann der folgende Befehl verwendet werden:

```
1 CellSet cellset = cube.LoadData(dimensions, measures, backgroundWorker);
```

Listing 11.5: Laden von Daten mit Hilfe der Datenbankschnittstelle

Der BackgroundWorker muss angegeben werden, wenn eine Fortschrittanzeige beim Laden erwünscht ist, ist aber optional. Anschließend kann das Ergebnis, das in Form eines `CellSets` zurückgegeben wird, für die Darstellung im Diagramm verwendet werden. Anstatt ein eigenes Datenmodell für das neue Diagramm zu entwickeln, ist es auch möglich, das bestehende `DiagramDataModel`, welches für das Scatterplotdiagramm entwickelt wurde, wiederzuverwenden.

11.5 Bestehendes Diagramm (Scatterplotdiagram)

Dieser Abschnitt beschreibt, wie Teile des Diagramms um neue Funktionen erweitert werden können und in welchen Klassen die entsprechenden Änderungen dann durchzuführen sind.

11.5.1 Erweiterung um neue Drop-Zonen

Um eine neue Drop-Zone in das Diagramm einzufügen, müssen an mehreren Stellen im Code Änderungen gemacht werden. In der *XAML*-Datei `Diagram` muss in der `DropAreaGroup` das `Grid` um entsprechende Zeilen bzw. Spalten erweitert werden und eine neue Drop-Zone nach dem in Listing 11.6 dargestellten Beispiel erstellt werden.

```
1 <local:DropAreaGroup Visibility="Visible" x:Name="DropArea" Margin="0,0,0,75">
2   <Grid.RowDefinitions>
3     <!-- already existing RowDefitions -->
4     <RowDefinition Height="newHeight"/>
5   </Grid.RowDefinitions>
6   <Grid.ColumnDefinitions>
7     <!-- already existing ColumnDefitions -->
8     <ColumnDefinition Width="newWidth"/>
```

```
9    </Grid.ColumnDefinitions>
10   <!-- already existing dropzones -->
11   <local:DropArea DropAreaType="NewDropAreaType" Margin="newMargin"
12    VerticalAlignment="Stretch" Grid.Column="newColumn" Grid.Row="newRow"
13    Grid.RowSpan="newRowspan"/>
14 </local:DropAreaGroup>
```

Listing 11.6: Einfügen einer neuen Drop-Zone

Um eine neue Drop-Zone erstellen zu können, ist `DropAreaType` weiterhin ein neuer Drop-Zonen-Typ hinzuzufügen, im Falle von Listing 11.6, Zeile 11, der Typ *NewDropAreaType* (siehe Listing 11.9).

```
1 public enum DropAreaType
2 {
3   // already existing DropAreaTypes
4   .
5   .
6   .
7   NewDropAreaType
8 }
```

Listing 11.7: Hinzufügen eines neuen DropAreaTypes

In der Klasse `DropAreaGroup` muss die Methode `Check(DropArea, HierarchicalOLAPObject)` um eine Behandlung der Drops auf die neu angelegte Drop-Zone erweitert werden. Für die grafische Darstellung der neuen Drop-Zone ist die Klasse `DropArea` sowohl in der *XAML*- als auch in der Code-Behind-Datei verantwortlich und muss entsprechend dem neuen `DropAreaType` angepasst werden.

11.5.2 Erweiterung des Diagrammmenüs

Um neue Buttons in das Diagramm einzufügen, müssen die folgenden Änderungen vorgenommen werden. In der *XAML*-Datei `Diagram` muss in dem `Menu` das `Grid` um ein weiteres `Grid` erweitert werden. An der Stelle sind mögliche Buttons folgendermaßen einzufügen.

```
1  <Grid x:Name="buttons" Visibility="Collapsed" Background="#10101010">
2      <!-- already existing buttons -->
3      <Grid global:Multitouch.TouchUp="newButtonUp"/>
4          <Rectangle RadiusX="newRadiusX" RadiusY="newRadiusY" Fill="newFill"
5           Stroke="newStroke" HorizontalAlignment="newAlignment"
6           Margin="newMargin" VerticalAlignment="newAlignment" Width="Auto"
7           Height="Auto"/>
8          <TextBlock HorizontalAlignment="Center" VerticalAlignment="Center"
9           Text="newButton" TextWrapping="Wrap"/>
10   </Grid>
11 </Grid>
```

Listing 11.8: Einfügen eines neuen Buttons

Um eine Funktionalität am neuen Button anzubinden, muss in der Code-Behind-Datei `Diagram.xaml.cs` weiterhin die neue Methode `newButtonUp(object sender, RoutedEventArgs e)` hinzugefügt werden, die auf ein ausgelöstes TouchUp-Event reagiert. Diese sieht folgendermaßen aus (siehe Listing 11.9).

```
1 private void newButton(object sender, RoutedEventArgs e)
2 {
3     // implement some new eventhandling-method
4 }
```

Listing 11.9: Hinzufügen eines neuen DropAreaTypes

11.5.3 Erweiterung der Diagrammpunkte

Die visuelle Repräsentation der darzustellenden Daten wird durch die Klasse `DiagramPoint` geleistet. Sollen die darzustellenden Daten geändert werden, muss die hinter den `DiagramPoints` liegende Datenhaltung geändert werden. Die Zuweisung von beispielsweise der Positionierung der Punkte geschieht durch die Methode `PositionDiagramPoint(int, ref DataPoint, int, DiagramPoint)` in der Klasse `ScalableCanvas`. Für eine Erweiterung der darzustellenden Daten sollten hier die `DiagramPoints` mit den neu darzustellenden Daten versehen werden.

Da `DiagramPoint` von `FrameworkElement` abgeleitet ist, kann, um z. B. ein anderes Rendering der `DiagramPoints` zu bewirken, die Methode `OnRender(DrawingContext)` überschrieben werden. Listing 11.10 zeigt dies anhand eines Beispiels.

```
1 protected override void OnRender(DrawingContext drawingContext)
2 {
3     // implement some new renderingMethod
4 }
```

Listing 11.10: Überschreiben der Methode `OnRender(DrawingContext)`

Natürlich kann so jede Methode, die `DiagramPoint` von `FrameworkElement` erbt, überschrieben werden und es können auch Methoden, die `FrameworkElement` bereitstellt, zur Darstellung weiterer Daten verwendet werden.

11.5.4 Layout verändern

Das Design des Diagramms wurde hauptsächlich mit *Microsoft Expression Blend* erstellt. Aus diesem Grund empfiehlt sich die Benutzung des Tools bei weiteren Änderungen.

Das Diagramm besteht aus einigen Bereichen, deren Design einzeln anpassbar ist. Im Folgenden werden beispielhafte Layout-Änderungen demonstriert:

Standardfarbwert der Diagrammpunkte ändern Die meisten Farbtöne sind in der Klasse `DiagramUtils` des `ScatterplotDiagrams` zu ändern. Die Standardfarbe der Diagrammpunkte ist von der statischen Variable `DefaultColor` gesetzt.

```
1 public readonly static Brush DefaultColor = Brushes.newColor;
```

Listing 11.11: Ändern der Standardfarbe der Diagrammpunkte

Farbverlauf der Diagrammpunkte ändern Der Farbverlauf der Diagrammpunkte nach Verwendung von Farbdimension ist in der Klasse `DiagramUtils` zu ändern. Standardmäßig beginnt der Farbverlauf mit grün und wechselt über gelb zu rot.

```
1 static DiagramUtils(){
2     colorDictionaryDimensionAxis.Add(0.0, Color.FromScRgb(newColor0));
3     .
4     .
5     .
6     colorDictionaryDimensionAxis.Add(1.0, Color.FromScRgb(newColorN));
7 }
```

Listing 11.12: Ändern der Standardfarbverlauf der Diagrammpunkte

Schriftgröße der Achsenbeschriftung ändern Die Schriftgröße ist standardmäßig auf 20.0 Pixel gesetzt. Die Konstante, die die Größe angibt, ist in der Klasse `Axis` zu finden.

```
1 private const double TextHeight = newValue;
```

Listing 11.13: Ändern der Schriftgröße der Diagrammbeschriftung

11.5.5 Erweiterung der Legende

Für den Fall, dass die Legende erweitert werden soll, kann dies in der Klasse `Legend`, bestehend aus einer *XAML*- und einer Code-Behind-Datei, gemacht werden. In Listing 11.14 wird gezeigt, wie die Legende um einen Tab erweitert werden kann.

```
1  <Grid>
2    <TabControl x:Name="tabControl" Background="White" Margin="1" BorderBrush="{x:Null}"
3     Style="{DynamicResource TabControlStyle1}" BorderThickness="0">
4      <TabItem x:Name="newTab" mt:Multitouch.TouchDown="TabItem_TouchUp" Header="newTab"
5       Height="newHeight" Width="newWidth" VerticalAlignment="newAlignment"
6       HorizontalAlignment="Center" Margin="newMargin" Background="White"
7       BorderBrush="{x:Null}" Style="{DynamicResource TabItemStyle1}">
8          <!-- newLayout -->
9      </TabItem>
10   </TabControl>
11 </Grid>
```

Listing 11.14: Hinzufügen eines Tabs zur Legende

Das Layout der einzelnen Tabs wird in Listing 11.14 durch die `DynamicResource` *TabItemStyle1* beeinflusst. Dies kann ebenfalls geändert werden, indem ein neues `Style`-Element definiert wird. Der einfachste Weg ist die Legende in *Microsoft Expression Blend* weiterzubearbeiten.

Teil V

Qualitätssicherung und Evaluationen

Kapitel 12

Qualitätssicherung

Unter Qualitätssicherung werden Maßnahmen verstanden, die zur Sicherstellung eines Qualitätsniveaus dienen. In der Projektgruppe sind verschiedene Maßnahmen zur Sicherung der Qualität des Endproduktes, bestehend aus Programm und Abschlussbericht, durchgeführt worden. Diese werden im folgenden Kapitel beschrieben. Zunächst werden die Coding Konventionen beschrieben, welche die Codequalität sicherstellen. Anschließend werden Sicherungsmaßnahmen für den Abschlussbericht vorgestellt. Den Abschluss dieses Kapitels bilden heuristische Evaluationen zur Sicherstellung und Verbesserung der Benutzerbarkeit des Programms.

12.1 Coding Konventionen

Sobald mehr als eine Person an einem Bericht oder einem Programm arbeitet, gibt es unterschiedliche Bearbeitungsstile. Eine Teamarbeit oder Projektarbeit soll für den Kunden allerdings immer so aussehen, als ob sie nur von einer Person bearbeitet wurde, also ein einheitliches Design besitzen. Konventionen bilden ein Regelwerk, woran sich jedes Teammitglied halten muss, damit das Ziel eines einheitlichen Stiles erreicht werden kann. In der Projektgruppe *TaP* sind zwei Regelwerke festgelegt worden. Zum einen die Coding- und zum anderen die Abschlussberichts-Konventionen. Im Folgenden wird das Regelwerk zur Codeerstellung erläutert, da dieses für eine Interpretation und eventuelle Weiterentwicklung hilfreich ist.

Im Allgemeinen lassen Programmiersprachen bezüglich der Strukturierung und Namensgebung im Programmcode einen relativ großen Freiraum zu. Dies hat zur Folge, dass der Code bei der Bearbeitung von mehreren Personen unübersichtlich und schlecht lesbar wird. Coding-Konventionen beschreiben eine einheitliche Struktur, wie Programmcode geschrieben wird. Gerade in größeren Projekten, wo jeder seinen eigenen Programmierstil besitzt, ist eine einheitliche Programmiervorschrift notwendig, um die Lesbarkeit und Wartbarkeit des Codes zu verbessern. (vgl. [HD05, Mos03])

> „*Completed source code should reflect a harmonized style, as if a single developer wrote the code in one session.*“ ([Micg])

Die Richtlinien sind dabei nicht als unumstößliche Regeln zu sehen. Im Einzelnen kann es sinnvoll sein, gezielt die vorher festgelegten Regeln zu verletzen. Dieses ist dann mit Hilfe von Kommentaren zu begründen. Im Allgemeinen können die Coderichtlinien in folgende Bereiche gegliedert werden:

- Dateinamen und Verzeichnisstrukturen
- Namenskonvention
- Formatierung des Codes
- Kommentare
- Programmierpraktiken

Meistens ist es sinnvoll, zunächst allgemeine Regeln festzulegen, die sich über die verschiedensten Bereiche ausdehnen.

Globale Regeln

In der Projektgruppe ist festgelegt worden, dass Codevariablen komplett in Englisch zu schreiben sind. Dass heißt, die Klassennamen, Variablennamen, Methodennamen und die Kommentare sind in Englisch zu halten. Als softwareseitige Unterstützung zur Code Analyse wird die in Microsoft Visual Studio Team Suite 2008 integrierte Code Analyse verwendet. Des Weiteren sind folgende allgemeine Richtlinien festgelegt, die teilweise softwareseitig nicht überprüft werden können (vgl. [Krü03, HD05, Edm09]):

- Code wird so einfach wie möglich gehalten, da er von anderen gelesen werden muss.
- Klassen werden so einfach wie möglich gehalten. Anstelle von großen monolithischen Klassen wird der Code auf mehrere kleine verteilt, wodurch die Übersichtlichkeit erhöht wird. Des Weiteren wird hierdurch der objektorientierte Gedanke unterstützt. Es ist nicht notwendig, jede einzelne Klasse in eine eigene Datei auszulagern. Zusammenhängende Klassen sollten, wenn möglich, in einer Oberdatei zusammengefasst werden.
- Noch unvollständige Funktionen einer Klasse oder Methode werden mit `// TODO:` kommentiert.

- Für Schleifen-Konstrukte werden nur `while` oder `foreach` Schleifen verwendet, da diese einfacher zu debuggen und logisch einfacher sind. Wird der Index des aktuellen Schleifendurchlaufes in ein Statement miteinbezogen, ist auch die Verwendung einer `for` Schleife möglich.
- Falls zu einem String-Objekt ein weiterer String hinzugefügt werden soll, wird ein Objekt des Typs `Stringbuilder` verwendet, und nicht der Konkatenations-Operator += des String-Objektes verwendet.
- Debug-Meldungen werden mit Hilfe der `Debug.Assert()` - Methode ausgegeben, da das Ausgabe-Fenster der Debug-Konsole sehr schnell voll wird. Außerdem wird so automatisch die Codezeilennummer mit angegeben. Das Listing 12.1 zeigt eine beispielhafte Anwendung dieses Debug-Konstrukts.

```
1 // Create an index for an array
2 int index;
3 // Perform some action that sets the index
4 // Test that the index value is valid.
5 Debug.Assert(index > -1);
```

Listing 12.1: Bedingungsfestlegung für eine Assertion

  Grafische Komponenten, wie zum Beispiel `MessageBox`, werden nicht für Debug-Meldungen verwendet.

Namenskonvention

In diesem Abschnitt wird die Freiheit zur Wahl von Namen für Dateien, Klassen, Variablen, Methoden, etc. neben den allgemeinen Regeln aus dem vorherigen Abschnitt weiter eingeschränkt. (vgl. [Krü03, HD05, Edm09])

Variablen werden klein geschrieben und erhalten einen mnemonischen Namen. Dass heißt, dass zum Beispiel Indexvaribalen nicht als `i` deklariert werden, sondern als `index` oder für eine temporäre Variable zum Beispiel `swapInt` gewählt wird. Variablen werden so früh wie möglich deklariert. Eine Instanziierung ist bei primitiven Datentypen nicht vorgesehen. Die Namen ergeben sich aus der Konvention `Adjektiv + Nomen + Qualifizierer`, zum Beispiel `lowestCommonDenominator`. Der Typ der Variable wird nicht im Variablennamen mit angegeben, es wird damit die sogenannte hungarian notation nicht verwendet. Für Exceptions und EventArgs wird der Variablenbuchstabe `e` verwendet, sofern dieses möglich ist.

> „*Do not use Hungarian notation for field names. Good names describe semantics, not type.*“[Micf]

Konstanten werden komplett groß geschrieben. Unterstriche zur Trennung von Wörtern werden dabei nicht verwendet. Die Namen folgen dem Prinzip `Adjektiv + Nomen + Qualifizierer`.

Methoden sollten nach dem Namensprinzip `Verb + Adjektiv + Nomen + Qualifizierer` gewählt werden. Die Parameter einer Methode sind, wenn möglich, in Gruppen zusammenzufassen, wobei für jede Gruppe eine eigene Codezeile verwendet wird. Pro Methode wird nur ein `return`-Statement verwendet.

Event Handler reagieren auf Events. Diese Events werden mit einem Unterstrich an den Objektnamen angehängt. Somit ergibt sich die Regel `ObjectName_EventName` für EventHandler, zum Beispiel `HelpButton_Click(object sender, EventArgs e)`.

Interfaces bekommen ein vorangestelltes `I`, gefolgt von einem Namen, der die Eigenschaft, die dieses Interface repräsentiert, beschreibt. Der erste Buchstabe dieser Eigenschaft wird wiederum groß geschrieben, also zum Beispiel `IComponent`.

Klassen und Strukturen sollten, wenn möglich, mit Hilfe der `#region`-Befehle in Regionen unterteilt werden. Alle öffentlichen Methoden, außer eindeutige Getter- und Setter- Methoden, werden mit Hilfe der Kommentierungsfunkton `///` dokumentiert. Die Namen der Klassen folgen dem Prinzip `Nomen + Qualifizierer`, zum Beispiel `CustomerForm`. Klassen und Strukturen beginnen mit einem Großbuchstaben. Ebenso werden komplexere nicht-öffentliche Methoden kommentiert.

Namespaces repräsentieren die logische Struktur von Subsystemen. Im Allgemeinen sollte die Regel `CompanyName.ProjectOrDomainName.PackageName.SubsystemName` zur Wahl des Namespace-Namens gefolgt werden, zum Beispiel `Microsoft.Logging.Listeners`.

Formatierung des Codes

Die Formatierung schreibt vor, wie der Code strukturiert werden soll, wo welche Klammern zu setzen und wo Leerzeichen einzufügen sind. Da die meisten Projektgruppenmitglieder bisher nur Java programmiert haben, sind auch hier einige Änderungen gegenüber dem Java-Coding-Style festzuhalten. Ein beispielhaftes Klassenlayout ist im folgenden Listing (Listing 12.2) dargestellt. (vgl. [Krü03, HD05, Edm09]):

```
1 // Class layout based on accessibility
2 class Purchasing
3 {
4     #region var
5
6     #region constants
7
8     #region Start
9
```

```
10      #region Scale
11
12      #region Rotate
13
14      #region Translate
15  }
```

Listing 12.2: Definition von Regions zur Code-Strukturierung

Dies hat zur Folge, dass die Methoden einer Klasse nach der Funktionalität gegliedert sind. Jede Methode ist entsprechend ihrer Funktionalität einzuordnen. Ebenso sind Felder in die entsprechende Region einzuordnen. Eine weitere Unterstrukturierung sollte, wenn möglich, eingefügt werden, damit die Übersicht erweitert wird. (vgl. [Krü03, HD05, Edm09])

Des Weiteren gelten folgende Regeln:

- Der `this`-Zeiger ist vor eigenen Methodenaufrufen anzugeben.
- Bei der Verwendung von statischen Methoden oder Variablen einer Klasse ist der Klassenname voranzustellen.
- Nach Methodenklammern folgt kein Leerzeichen, zum Beispiel `MessageBox(String message)`.
- Lange Befehlszeilen und Übergabeparameter werden in der Zeile umgebrochen. Hierbei sollten nach Möglichkeit die Übergabeparameter in einer Zeile gruppiert werden.

Kommentare

Zur Kommentierung von Klassen und Methoden wird die C# Dokumentierungsfunktion verwendet, welche mit `///` eingeleitet wird. Mit Hilfe dieser Dokumentierungsfunktion sind alle Methoden und Klassen zu dokumentieren. Dabei ist nicht anzugeben, wie die Klassen, bzw. Methoden etwas machen (wird bereits durch den Quellcode ausgedrückt), sondern welchen Zweck diese Funktionalität erfüllt. Außerdem stehen verschiedene Möglichkeiten zur Verfügung, um im Quellcode Kommentare einzufügen. Von dieser Möglichkeit ist ausgiebig Gebrauch zu machen, besser ist es jedoch, Kommentare nicht einfügen zu müssen, da der Code und die Variablennamen bereits aussagekräftig sind. Um einzelne Zeilen zu kommentieren, wird diese direkt nach dem Abschluss des Befehls dokumentiert.

```
1 private string name = string.Empty; // Name of control (defaults to blank)
```

Listing 12.3: Beispielhafter Kommentar

Auch die Zusammenfassung zu Blöcken ist möglich. Die Auskommentierungsfunktion mit Hilfe von `/*..*/` ist nur kurzzeitig zu verwenden. Wenn Code nicht mehr benötigt wird, ist dieser zu löschen. Falls der Code ohne das Auskommentieren eines Codeabschnittes nicht eingecheckt werden kann, so ist ein `// TODO:`-Kommentar hinzuzufügen. (vgl. [Krü03, HD05, Edm09])

Programmierpraktiken

Programmierpraktiken haben das Ziel, fehleranfällige Programmierkonstrukte zu verhindern und das Lesen des Codes gerade für unerfahrene oder weniger geschulte Programmierer zu vereinfachen. In diesen Code-Konventionen werden lediglich die Verwendung von `goto`-Anweisungen und Fallthroughs verboten. Fallthroughs sind Switch-Case-Anweisungen, wobei einige Case-Anweisungen kein `break`-Statement besitzen, und damit der darauf folgende Case-Fall auch ausgeführt wird (vgl. [Mos03]). Des Weiteren wird die Verwendung einer erweiterten If-Bedingung `Bedingung ? True-Ausführung : False-Ausführung` für einfache If-Abfragen vorgeschlagen. Bei komplexen Bedingungen ist die Standard-If-Bedingung zu verwenden.

Im Laufe der Projektphase hat sich gezeigt, dass einige Konventionen auf Grund der wachsenden Erfahrung der Programmierer nicht mehr benötigt werden. Zum Beispiel mussten Anfangs *LINQ*-Abfragen ausführlich kommentiert werden, da nur wenige Projektmitglieder mit dieser deklarativen Abfragesprache Erfahrung hatten. In der Projektphase konnte diese Konvention entfallen, da alle Mitglieder so viel Erfahrung gesammelt hatten, sodass immer eine Person den *LINQ*-Ausdruck deuten konnte.

Für Perfomanzsteigerungen des Systems stehen in Visual Studio Team System 2008 in der Code Analyse sehr viele weitere Optionen zur Verfügung, welche für das Programm entsprechend gewählt worden sind.

12.2 Testvorgehen

In der Projektgruppe sind verschiedene Mechanismen eingeführt worden, um die Qualität des Endberichtes und des Programmes sicherzustellen. Zunächst ist zwischen der Programm- und Endberichtsqualität zu unterscheiden. Für Programme kann es durchaus sinnvoll sein, automatische Tests durchzuführen, allerdings existieren keine automatischen Tests für Dokumentationen. In der Projektgruppe sind für Dokumentenüberprüfungen *Reviews* durchgeführt worden, welche vom Qualitätsbeauftragten koordiniert und beaufsichtigt wurden. Diese werden im Folgenden vorgestellt; anschließend werden die weiteren zur Qualitätssicherung durchgeführten Maßnahmen vorgestellt.

12.2.1 Review

Reviews dienen zur Fehlerüberprüfung eines Arbeitsergebnisses. Sie zählen zu der Gruppe der analytischen Maßnahmen zur Qualitätssicherung. Im Gegensatz zu den testenden Verfahren, wie zum Beispiel *Unit-Tests*, wird auf eine Ausführung des Prüflings mit konkreten Eingaben verzichtet. Ziel eines *Reviews* ist das Auffinden von statischen Fehlern, wie zum Beispiel:

- Abweichungen von Standards
- Fehler im Design und den Anforderungen
- Falsche oder inkorrekte Schnittstellenspezifikation

Es gibt verschiedene Ausprägungen von *Reviews*, die sich in der Durchführungsart stark unterscheiden (siehe [ICS98]). Allen Verfahren gemein ist, dass Mängel sachlich genannt werden, ohne dem Autor einen Vorwurf zu machen. Der Autor wiederum muss in der Lage sein, sachliche Kritik zu akzeptieren.

Für die Qualitätssicherung von Dokumentationen sind *informelle Reviews* verwendet worden. Im Gegensatz zu *Inspektionen*, die nach IEEE 610 und IEEE 1028 einen in sieben Phasen festgeschriebenen Ablauf haben, sind *informelle Reviews* wesentlich ungeregelter und können dadurch in der Projektgruppe flexibler gehandhabt werden. Für den aufwändigen Ablauf von *Inspektionen* fehlt häufig die Zeit.

Für den Ablauf von *informellen Reviews* sind einige Regeln festgelegt worden, wodurch die Verantwortung auf die beiden Rollen *Autor* und *Inspekteur* aufgeteilt wurde.

- Der Autor ist nicht gleichzeitig Inspekteur seines eigenen Dokumentes.
- Der Autor stellt den Inspekteuren das Dokument in einer geeigneten Form zur Verfügung.
- Die Inspekteure korrigieren das Dokument und stellen dem Autor eine fehlermarkierte Version zur Verfügung. Sie korrigieren nicht selbst. Dies ist Aufgabe des Autors, da er die Auswirkungen einer Korrektur selbst am besten einschätzen kann.

Die Koordination dieser *informellen Reviews* ist durch den Qualitätsbeauftragten erfolgt. Damit dieser einen Überblick über die aktuellen Ergebnisse der *Reviews* behält, ist ein sogenanntes *Feedback-Formular* entwickelt und eingeführt worden. Dieses als *Portable Document Format (PDF)*-Formular entwickelte Dokument ist von allen beteiligten Personen, dem Autor und den Inspekteuren, ausgefüllt worden und an den Qualitätsbeauftragten gesendet worden. Somit sind dem Beauftragten die wesentlichen Eckdaten (Fehleranzahl, Fehlerverteilung, Ende der Inspektion, Ende der Einarbeitung der korrigierten Fehler) des *Reviews* bekannt. Damit die *Reviews* termingetreu

durchgeführt wurden, ist ein spezielles Strafensystem eingeführt worden. Die Termine zur Erstellung und Einarbeitung von *Reviews* sind durch den Qualitätsbeauftragten festgelegt worden. Das Zusenden des in Abbildung 12.1 dargestellten *Feedback-Formular*s dokumentiert die erfolgreiche Abarbeitung einer Aufgabe.

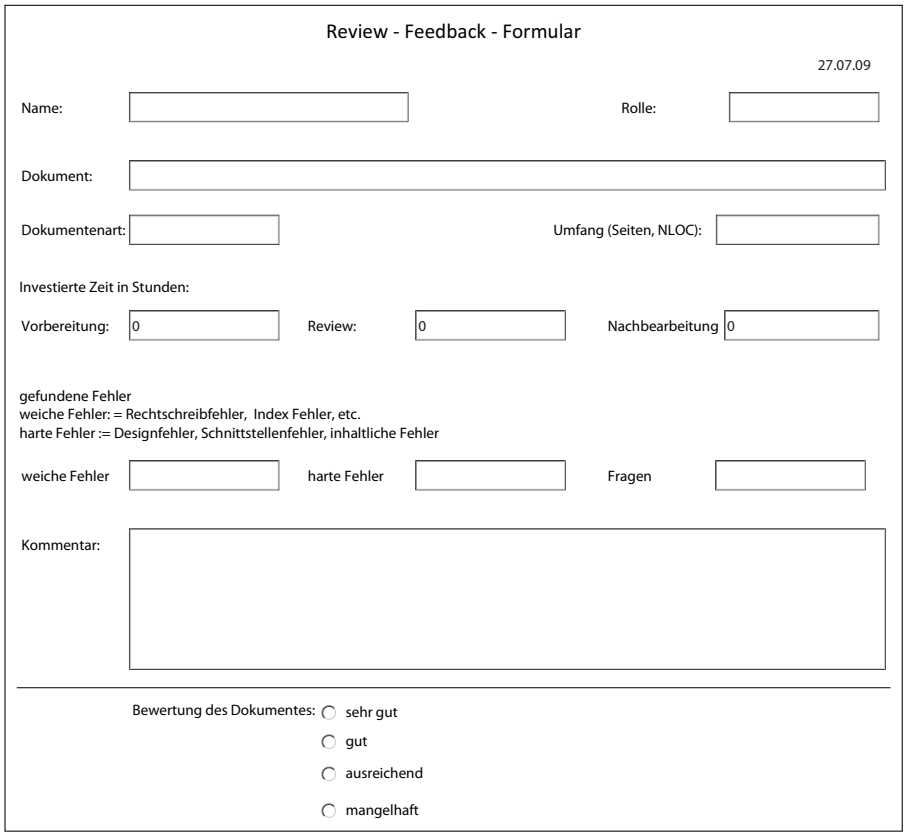


Abbildung 12.1: Feedback-Formular zu Reviews

Die ersten Felder des Formulars in Abbildung 12.1 dienen dazu, das Formular einem *Review* zuzuordnen. Mit Hilfe der Felder zur investierten Zeit ist eine Gleichverteilung der Last, die durch die *Reviews* entsteht, zwischen den Projektgruppenmitgliedern möglich. Die letzten Felder sind für die Bewertung des inspizierten Dokumentes zuständig. Durch die Fehleranzahl und die Seitenanzahl des Dokumentes kann die Fehlerquote ermittelt werden. Wichtig hierbei ist die Unterscheidung zwischen harten und weichen Fehlern. Als weiche Fehler werden Rechtschreib- und Grammatikfehler verstanden. Unkorrekte oder unvollständige Aussagen werden als harte Fehler betrachtet. Kommen in einem Dokument harte Fehler vor, wird nach der Korrektur ein erneutes *Review* angesetzt, damit sichergestellt ist, dass dieser Fehler ohne Auswirkungen auf die restliche Qualität des Dokumentes behoben worden ist.

Neben der Information über ein *Review* gestattet das verwendete Formularsystem auch die Sammlung der Formulare in einer Tabellenstruktur, wodurch Auswertungen über die Reviews möglich sind.

12.2.2 Auswertung Review

Die Auswertung ist durch die Sammlung der *Review*-Formulare ermöglicht worden. Die Ergebnisse werden im Folgenden diskutiert. In der Abbildung 12.2 ist der Verlauf der durchgeführten *Reviews* und der Einarbeitungen dargestellt.

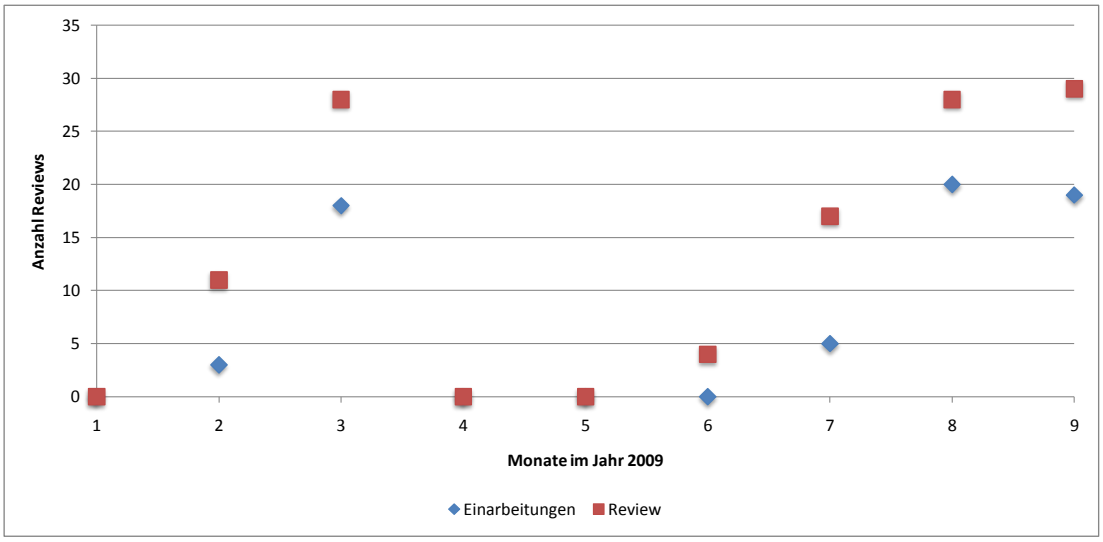


Abbildung 12.2: Verlauf der Reviews und Einarbeitungen

Es ist ersichtlich, dass immer mehr *Reviews* als Einarbeitungen durchgeführt wurden. Dieses ist darin begründet, dass die meisten Dokumente von mindestens zwei Inspekteuren begutachtet wurden. Bei kleinen, weniger fehleranfälligen Dokumenten, ist teilweise die Inspektion von nur einem Begutachter durchgeführt worden. Der Verlauf, dargestellt ab Januar 2009, zeigt außerdem auf, dass in den Monaten der Abgabe von Dokumenten (Zwischenbericht und Abschlussbericht) die Anzahl der *Reviews* erhöht wurde, um die Qualität der Dokumente zu gewährleisten. In den Monaten April und Mai sind keine *Reviews* durchgeführt worden, da im April planungsgemäß eine Orientierungsphase und anschließend eine Programmierphase eingeleitet wurde. Übereinstimmend mit der Projektplanung ist entschieden worden, *Reviews* erst ab Januar 2009 zu koordinieren, wodurch die Zeitspanne bis Januar 2009 optimal zur Ideenfindung und Zieldefinition genutzt werden konnte. Dieser Zeitraum wird bei der weiteren Auswertung nicht betrachtet. Somit können zwei zeitliche Abschnitte definiert werden. Zum einen die Zeitspanne der *Reviews* die zum Zwischenbericht Ende März

durchgeführt worden sind, genannt *Reviewphase Zwischenbericht*. Zum anderen die Zeitspanne, welche nach Fertigstellung des Zwischenberichtes bis zur Abgabe des Abschlussberichtes reicht, die *Reviewphase Abschlussbericht*. Der Zeitraum von Januar bis September 2009 wird als *Review-Zeitraum* definiert.

Die Abbildung 12.3 verdeutlicht den Verlauf der Qualitätsbewertungen über den Review-Zeitraum. Es ist für jede Monatshälfte der arithmetische Mittelwert der Bewertungen ermittelt worden. Die Bewertungsskala reicht dabei von 1,0 (sehr gut) bis zu 4,0 (ausreichend).

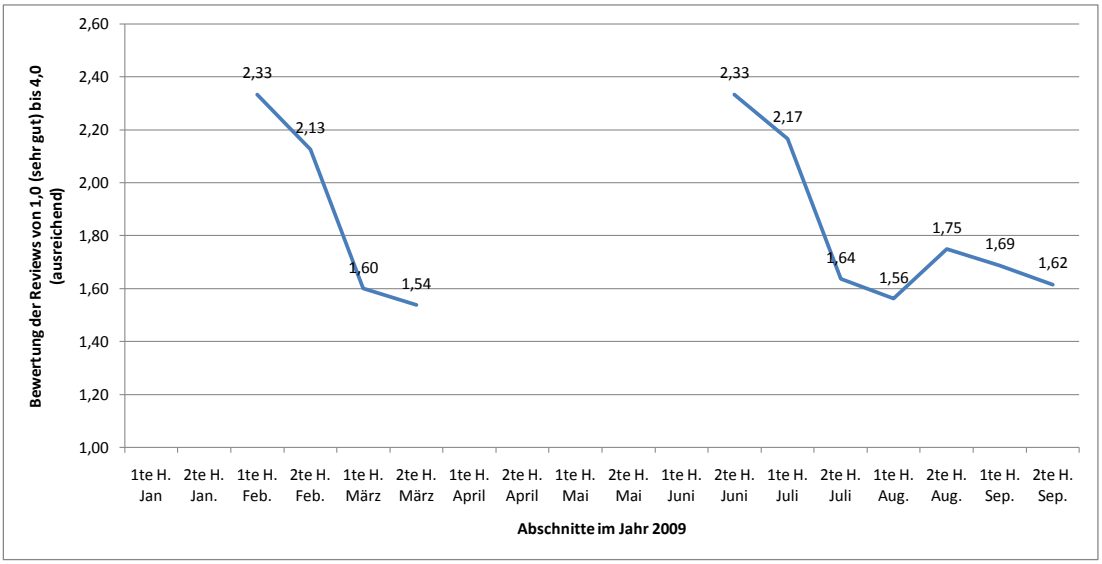


Abbildung 12.3: Verlauf Qualität pro Monatshälfte

Zum Ende der jeweiligen Review-Phasen ist der Trend erkennbar, dass sich die Bewertungen, die an das deutsche Schulnotensystem (1=sehr gut; 2=gut; 3=befriedigend; 4=ausreichend) angelehnt sind, der Dokumente auf einem Bereich zwischen 1,5 und 1,9 eingependelt haben. Damit ist die durchschnittliche Bewertung in der Endphase besser als gut. Zudem zeigt die Qualitätskurve auf, dass zu Beginn jeder Phase Qualitätsprobleme herrschten. Zur Lösung dieser Probleme ist die Maßnahme getroffen worden, dass rechtschreib- und grammatikschwache Autoren durch andere Autoren unterstützt wurden. Durch diese Maßnahme ist die Qualität der Dokumente, welche zum Review eingereicht wurden, erheblich angestiegen.

Insgesamt sind durch die 117 durchgeführten Reviews und 65 Einarbeitungen 2221 weiche und 65 harte Fehler entdeckt und behoben worden. Hierfür sind 254 Arbeitsstunden investiert worden. Umgerechnet auf Arbeitstage (8 h pro Tag) sind somit über 31 Tage in die Qualitätssicherung des Abschlussberichts und Zwischenberichts geflossen. Pro gelesener Seite eines Begutachters wurden ca. zwei weiche Fehler entdeckt. Zum Lesen einer Seite wurden im Schnitt neun Minuten benötigt, die

Inspektionsrate beträgt damit sechs Seiten pro Stunde. Im Vergleich mit typischen Softwareprojekten, welche eine Rate von 5-20 Seiten pro Stunde haben [GG93], ist die Inspektionsrate damit am unteren Niveau. Von der in [GG93] genannten optimalen Inspektionsrate von einer Seite pro Stunde besteht allerdings eine erhebliche Differenz. Dies bedeutet, dass einerseits noch mehr Zeit in die Reviews einfließen könnte, andererseits im Vergleich mit echten Softwareprojekten die Qualität der Reviews bereits sehr hoch ist, denn mehr Zeit für *Reviews* bedeutet eine gründlichere Fehlersuche.

Das stetige Erstellen von Zwischendokumenten, die zeitnah einem *Review* unterzogen wurden, gestattete die frühe Zusammenstellung des Abschlussberichts. Dieser konnte noch vor der Abgabe durch externe Inspekteure kontrolliert werden und die dort gefundenen Fehler korrigiert werden.

12.2.3 Programmtests

Ein weiterer Bestandteil der Qualitätssicherung ist die Sicherung der Qualität des Programmes. Diese drückt sich zum einen durch einen lesbaren Code aus, der durch die Coding-Konventionen (vgl. 12.1) sichergestellt ist, und zum anderen durch ein fehlerfreies und benutzungsfreundliches Programm.

Das entwickelte Programm ist sehr stark oberflächenlastig, was das Testen des Programms und damit das Finden von Fehlern äußert schwierig gestaltet. *Unit-Tests* sind für das Testen von Algorithmen ausgelegt, allerdings nicht für das Oberflächentesten. Aus diesem Grund hat sich die Projektgruppe dazu entschieden, *Unit-Tests* nicht als vorrangige Testmethode zu verwenden. Als weitere Schwierigkeit gestaltet es sich, für ein automatisches Testframework Daten zu entwickeln, die alle Randfälle behandeln. Wegen den zwei genannten Schwierigkeiten ist auf automatisierte Tests verzichtet worden. Stattdessen sind Live-Tests direkt am System durchgeführt worden. Hierfür ist eine ganze Iteration, die Stabilisierungs-Iteration, eingeplant worden.

In dieser Iteration ist das ganze Programm nach einem vorgegeben Schema getestet worden. Dabei ist die Funktionalität durch Anwendungsszenarios geprüft worden, wodurch nicht nur die Benutzungsoberfläche, sondern auch implizit die darunter liegenden Schichten, wie zum Beispiel das Multitouch-Framework, getestet worden sind. Zur Dokumentation der Testfälle ist ein *Wiki* eingerichtet worden. Ein *Wiki* hat den Vorteil, dass es von jedem editiert und ergänzt werden kann, wodurch jedes Projektmitglied die aktuellen Änderungen sofort sieht. Damit die Einträge im *Wiki* alle eine gleiche Form haben, sind diese nach einer Vorlage erstellt worden. Ein Eintrag besteht aus folgenden Elementen:

Titel: Ein aussagekräftiger Titel zu diesem Testszenario

Beschreibung: Eine genaue Beschreibung des Szenarios

Erwartet: Was wird bei dem Ablauf des Szenarios erwartet

***Log4Net*:** Mit Hilfe dieses Logging-Systems sind Statusmeldungen in das System integriert worden, z. B. welche Elemente auf die Achsen eines Diagramms hinzugefügt worden sind. Diese Meldungen werden in diesem Bereich als genaue Beschreibung, was der Tester ausgeführt hat, eingefügt.

Fehler: Eine Auflistung der Fehler, die während des Testens aufgetreten sind.

Ampel: Damit ein Projektmitglied sich schnell einen Überblick machen kann, welche Tests fehlerhaft (rot), noch ausgeführt (gelb) oder korrekt (grün) durchgelaufen sind, ist zusätzlich eine Ampel eingefügt worden.

Die Live-Tests sind in die Bereiche Desktop, Pie-Menü, Diagramm und Command-Layer aufgeteilt worden. Dies hat zum einen den Vorteil, dass sich unterschiedliche Personen mit den einzelnen Tests befassen können; zum anderen erhöht dies die Übersicht. Im Folgenden ist beispielhaft ein Auszug aus dem *Wiki* bezüglich eines Szenarios dargestellt.

Belegung X-und Y Achse mit Kennzahl, Dimension

Testfall (Beschreibung): Belegung der X und Y Achsen mit Kennzahl und Dimension.

Erwartet:

- Es werden alle Datenpunkte angezeigt.
- Die Beschriftung ist bei den Datenpunkten und ist korrekt
- Die Punkte sind an der korrekten Position.

Log4Net:

- Test1 :
  - Y Achse:
  - 2009-07-21 11:57:25,655 [9] DEBUG TaP.Diagram (E:\Studium Arne\Visual-Analytics \Tap-Source\TAP\Touch and Pray\TaPControls\DropArea\DiagramDataModel.cs) - adding Date/ Calendar Year
  - X-Achse:
  - 2009-07-21 11:57:50,335 [9] DEBUG TaP.Diagram (E:\Studium Arne\Visual-Analytics \Tap-Source\TAP\Touch and Pray\TaPControls\DropArea\DiagramDataModel.cs) - adding Internet Orders/ Internet Order Count

Fehler

- Test 1: Jahr 2006 hat einen Diagrammpunkt, obwohl es keine Daten hat. Es scheint so, als wäre die X-Achse horizontal gespiegelt.

Abbildung 12.4: Auszug aus einem Testfall

Das *Wiki* hat noch eine weitere wichtige Aufgabe: Es stellt das Bug-Tracking System für die Projektgruppe zur Verfügung.

12.2.4 Bug-Tracking

Bugs, also Fehler im Programm, müssen einheitlich dokumentiert werden, da sie sonst eventuell nicht behoben werden. Hierzu gibt es zahlreiche, teilweise kostenlose, Tools, wie zum Beispiel *Bugzilla* (`http://www.bugzilla.org/`). Es stellt sich jedoch schnell die Frage, ob solch ein komplexes System mit seinen ganzen Funktionen überhaupt benötigt wird oder ob nicht ein einfacheres Verfahren sinnvoller ist. Vor dem Hintergrund, dass *TaP* kein Community-Projekt ist, sondern von einer begrenzten Anzahl von Entwicklern realisiert wird, ist entschieden worden, auf ein aufwendiges Tool zum Bug-Tracking zu verzichten. Stattdessen ist das bereits erwähnte *Wiki* um entsprechende Seiten für jedes Unterprojekt (Datenbank, Multitouch-Framework, Grafische Ebene) ergänzt worden. In diesen Seiten sind die gefundenen Bugs eingearbeitet worden. Falls einem Anwender, der durch das Anwenden des Programms im Entwicklungszustand automatisch zu einem Tester wird, ein Fehler auffällt, kann dieser in den meisten Fällen in die genannten Bereiche eingegliedert werden. Es kann allerdings auch vorkommen, dass ein Fehler nicht zuzuordnen ist, hierfür ist eine Sonstiges-Seite eingerichtet worden. Ein Eintrag im *Wiki*-Bugtracking besteht aus den folgenden Punkten:

Titel: Ein aussagekräftiger Titel zu diesem Fehler

Achsenbelegung: Die Belegung der Achsen, als dieser Fehler auftrat

Exception: Position einer eventuell auftretenden Fehlermeldung

Problembeschreibung: Eine kurze Beschreibung des Fehlers und eine Beschreibung, wie es zu dem Auftreten des Fehlers gekommen ist.

Ampel: Ähnlich wie beim Testen dient auch hier eine Ampel als symbolische Anzeige, welchen Status dieser Fehler gerade hat. Grün als Ampelfarbe stellt einen behobenen Bug dar, rot einen neuen, noch nicht behobenen Fehler und gelb einen Bug, der gerade behoben wird.

Es kann sein, dass eventuell keine Exception geworfen wird, oder dass die Achsenbelegung keinen Einfluss auf den Fehler hat. Aus diesem Grund sind diese Punkte optional und können weggelassen werden. Im Folgenden ist ein beispielhafter Auszug aus dem Bug-*Wiki* dargestellt, um das System zu verdeutlichen.

Kompatibilitätscheck liefert falsche Ergebnisse

Achsenbelegung:

- x-Achse: ICD HD Fallzahl
- y-Achse: [Dim Aufnahmegrund].[Aufnahmegrund].[Aufnahmegrund]
- Größe: -
- Farbe: -
- Animation: -

Exception (Zeile, Datei und Exception): ScaleableCanvas.xaml.cs Z.390: KeyNotFoundException: Der angegebene Schlüssel war nicht im Wörterbuch angegeben.

Problembeschreibung: Vermutlich haben Dimension und Kennzahl nicht zusammengepasst und es wurde nie abgefangen, so dass beim Diagrammaufbau DataPoints ohne Inhalt ankommen.

Reproduziert mit: [Dim Altersgruppe].[Altersgruppe].[Altersgruppe] und Einwohner, die beiden passen laut SQL Management Studio aber zusammen.

Abbildung 12.5: Auszug eines Bugs aus dem Wiki

Neben reinen Bugs ist es auf den entsprechenden Seiten auch möglich, Ideen und Vorschläge zu den einzelnen Komponenten zu speichern, wodurch es eine zentrale Stelle zur Speicherung dieser Ideen gibt. Beim Testen des Programms sind sehr viele Bugs aufgefallen, mit unterschiedlichen Auswirkungen auf das Programm. Damit die kritischen Probleme als erstes behoben werden können, ist eine Priorisierung der Bugs in drei Gruppen vorgenommen worden. Durch diese Maßnahmen sind insgesamt 68 Fehler identifiziert und korrigiert worden.

12.2.5 GUI-Tests mit Hilfe von heuristischen Evaluationen

Im Gegensatz zu den vorherigen Tests zielen heuristische Evaluationen auf die Benutzungsfähigkeit des Programms und nicht auf die Funktionalität. Es wird versucht, durch diese Art des Testens Interaktionsfehler zu finden. Diese gelten im Vergleich zu Usability-Evaluationen als relativ unaufwendige Methode, um Usability-Fehler zu finden [SB06]. Die verwendeten Heuristiken sind die im Abschnitt 5.5.1 genannten acht goldenen Regeln des Interface Designs von Shneidermann, sowie ausgewählte Konventionen für Touchscreens (siehe 5.5.3) verwendet worden. Bei der Durchführung dieser Testform sollten die Inspekteure Usability-Experten mit einem großen Domänenwissen sein. Diese Tester sind allerdings sehr selten anzutreffen [SB06]. Da solche Experten nicht zur Verfügung standen, ist auf die Entwickler von *TaP* zurückgegriffen worden.

Zur Durchführung der Evaluation ist das Programm in verschiedene funktionale Bereiche aufgegliedert worden, um abgrenzbare Testeinheiten zu erhalten. Im Folgenden ist die Aufteilung aufgelistet:

Pie-Menü: Das hierarchische Menü wird in diesem Bereich getestet.

Diagramm: Das Scatterplotdiagramm als Darstellungsfläche der Daten wird überprüft.

Desktop: Dieser Bereich umfasst das Command Layer und das Zusammenspiel aller Komponenten.

Die Bereiche sind jeweils zwei Entwicklern zugewiesen worden, welche möglichst wenig an der Implementierung der jeweiligen Komponente beteiligt waren. Anhand eines erstellten Fragebogens auf Grundlage der genannten Regeln ist der Test durchgeführt worden. Jede der Regeln ist von den Inspekteuren überprüft worden. Wenn ein Kriterium nicht erfüllt wurde, ist von dem Inspekteur der Grund für das Verletzen des Kriteriums angegeben worden.

Auswertung

Abbildung 12.6 zeigt einen Überblick über das Gesamtergebniss aller drei Teilkomponenten. Es ist ersichtlich, dass mindestens 50% den jeweiligen Kriterien zugestimmt haben. Die Bedienung des Programms wurde von allen Inspekteuren als angenehm empfunden. Hingegen wurden Mängel im Bereich der *Umkehrbarkeit* und *Präzision* festgestellt. Die Präzisionsmängel können teilweise auf die gewählte Hardwarekonfiguration zurückgeführt werden, da mit einer Touch-Bedienung nicht so präzise ausgewählt werden kann wie mit einer klassischen Maus. Von den Inspekteuren sind keine Unstimmigkeiten beim Merkmal *Zielgruppe* festgestellt worden, wobei diese von den Testern auch schwierig aufzudecken sind, da sie nicht der Zielgruppe angehören.

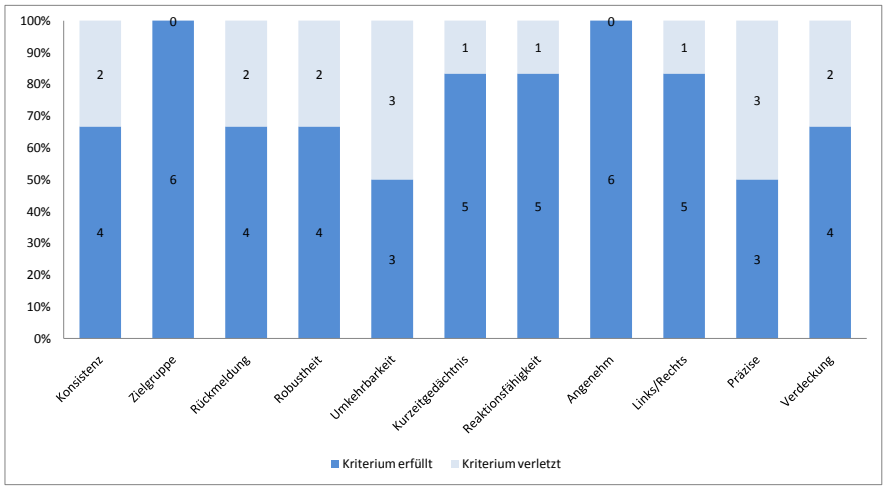


Abbildung 12.6: Ergebnisse heuristische Evaluation - Gesamtüberblick

In der Evaluation wurde entdeckt, dass das Scatterplotdiagramm Unstimmigkeiten in der Kategorie *Konsistenz* aufweist. Diese sind von Testern auf Grundlage des unter-

schiedlichen Designs, z. B. zwischen Pie-Menü und Scatterplottdiagramm bemängelt worden. Außerdem sind einige Komponenten des Diagramms unterschiedlich gestaltet, z. B. haben einige Komponenten abgerundete Ecken, während andere unveränderte Ecken haben. Des Weiteren sind Mängel im Bereich *Rückmeldung* festgestellt worden, da beim *DrillDown* bzw. *RollUp* keine Rückmeldung an den Benutzer erfolgt.

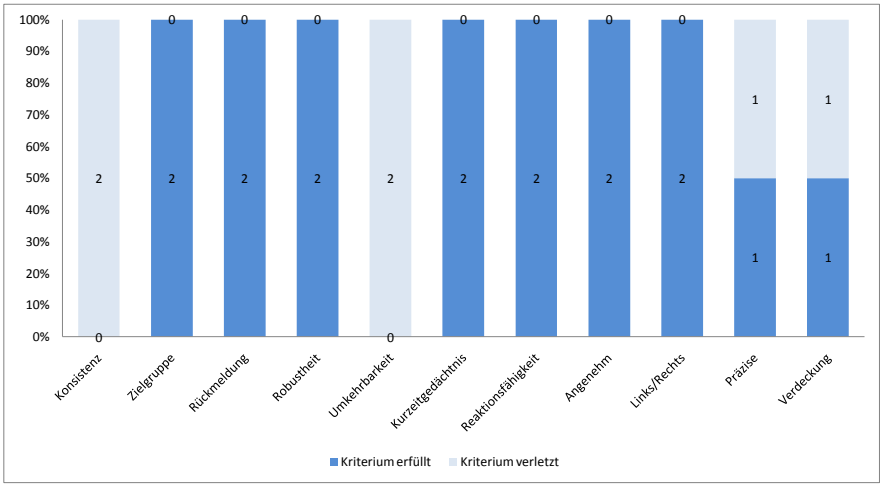


Abbildung 12.7: Ergebnisse heuristische Evaluation - Diagramm

Die Komponente Desktop zeichnet sich durch die *angenehme Bedienung* sowohl für *Links-* wie auch für *Rechtshänder* aus. Dies ist wahrscheinlich darin begründet, dass der Nutzer eine freie Positionswahl am Multitouch-Tisch hat. An dieser Komponente ist wiederum der Mangel der unzureichenden *Rückmeldung* an den Benutzer aufgefallen.

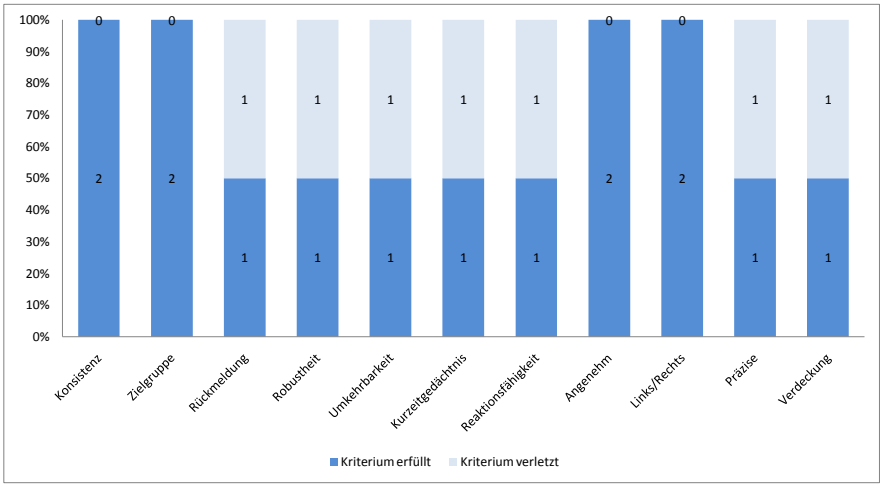


Abbildung 12.8: Ergebnisse heuristische Evaluation - Desktop

Das Pie-Menü als Evaluationskomponente entsprach weitestgehend den Kriterien. Besonders der *Verdeckungseffekt*, welcher bei dem Pie-Menü nur gering auftritt, ist positiv bewertet worden. Allerdings ist das Merkmal *Kontrolle* verletzt worden, da sich das Pie-Menü selbstständig verkleinert bzw. vergrößert und dieser Prozess nicht abgebrochen werden kann. Von den Inspekteuren wurde bemängelt, dass alle Elemente anfangs auf der rechten Seite des Pie-Menüs sind, und sie empfanden damit das Merkmal *Links/Rechts* verletzt.

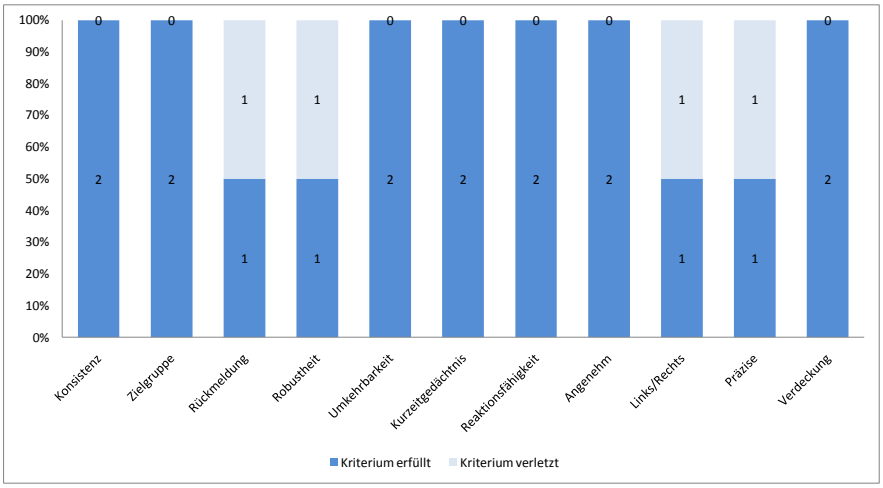


Abbildung 12.9: Ergebnisse heuristische Evaluation - Pie-Menü

Konsequenzen

Auf Grund von technischen Schwierigkeiten mit dem Multitouch-Tisch ist diese Evaluation erst relativ spät durchgeführt worden. Aus diesem Grund konnten nicht alle Usability-Fehler behoben werden. Trotzdem konnten einige Mängel behoben werden. Das Pie-Menü ist in der Farbgebung dem Diagramm angeglichen worden, wodurch die Fehleranzahl in der *Konsistenz* verringert wurde. Des Weiteren ist in dem Menü eine optische Rückmeldung implementiert worden. Durch Blinken des jeweiligen Menübuttons ist dem Anwender bewusst, welchen Menüpunkt er gerade ausgewählt hat. Der Mangel der unterschiedlichen Kantenführung im Diagramm konnte jedoch nicht behoben werden, da dieses auf der Entwurfsentscheidung beruht, die Achsen durch ein `StackPanel` zu repräsentieren und *WPF* einen Fehler beim *Clipping* der Komponenten aufweist [Mica].

Kapitel 13

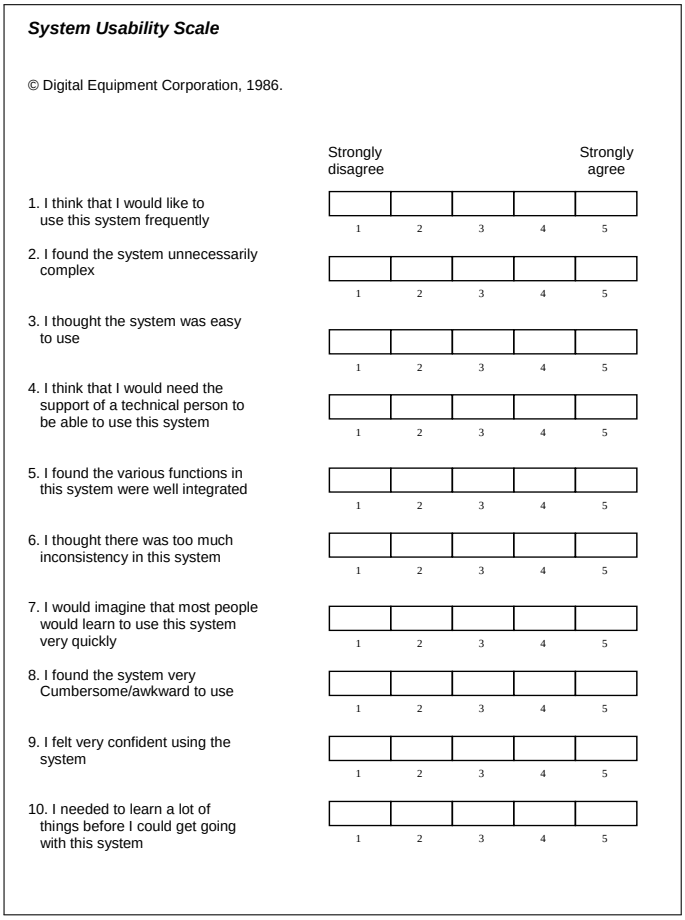
Usability-Evaluationen

Usability-Evaluationen sind die wirksamste Methode, um die tatsächliche Gebrauchstauglichkeit des entwickelten Produkts zu testen. Entwickler besitzen eine natürliche Blindheit gegenüber Fehlern im Interaktionsdesign der Komponenten, die sie selbst entwerfen. Sie ist dadurch gegeben, dass der Entwickler in jedem Stadium vollständig mit der Bedienung vertraut ist. Des Weiteren haben Informatiker oft eine andere Sicht auf Programme, da sie mit der hinter dem System stehenden Architektur und Datenstrukturen besser vertraut sind. In Evaluationen wird das System von unabhängigen Probanden, die der Zielgruppe möglichst nahe kommen, gmit dem Ziel getestet, Fehler in der Gebrauchstauglichkeit aufzudecken.

Dabei wurden jeweils nach dem ersten und zweiten Semester die bis zu diesem Zeitpunkt erstellten Versionen (*TaP* v1.0 und *TaP* v2.0) getestet. Die Evaluation waren dabei zum einen als *induktive Tests*, die Schwachstellen im Interaktionsdesign und der Gestaltung aufzudecken sollen, angelegt, als auch als *deduktive Tests*, die einen Vergleich der beiden Versionen ermöglichen sollen [SB06].

Um die Schwachstellen aufzudecken, wurden von den Testleitern die Beobachtungen während der Tests notiert. Für den Vergleich der beiden Systeme wurden verschiedene Werte gemessen. Außerdem füllte jeder Proband nach der Evaluation den *System Usability Scale (SUS)*-Fragebogen aus. Dieser Fragebogen ist in Abbildung 13.1 dargestellt. Er enthält zehn Fragen, die mit einer *Likert-Skala*, im Bereich von 1 bis 5, messen, wie gut der Proband das System bedienbar fand und wie zufrieden er damit war. Der *SUS* wird zusammengefasst in einer Skala 0-100 angegeben, wobei höhere Werte eine höhere Zufriedenheit bedeuten. [Bro96].

TaP ist ein Tool für Analysten, also im Wesentlichen Domänenexperten mit einem sehr gutem Wissen über die Daten, die sie analysieren wollen (vergleiche 7.1). Bei der Fertigstellung von *TaP* v1.0 war nur die Beispieldatenbank *AdventureWorks* verfügbar. Um eine Vergleichbarkeit der Evaluationen beizubehalten wurde auch bei der zweiten Evaluation wieder *AdventureWorks* eingesetzt. Da es sich um eine Beispieldatenbank handelt, gab es natürlich keine Domänenexperten. Die Probanden erhielten daher eine



System Usability Scale

© Digital Equipment Corporation, 1986.

	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently	1	2	3	4	5
2. I found the system unnecessarily complex	1	2	3	4	5
3. I thought the system was easy to use	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this system	1	2	3	4	5
5. I found the various functions in this system were well integrated	1	2	3	4	5
6. I thought there was too much inconsistency in this system	1	2	3	4	5
7. I would imagine that most people would learn to use this system very quickly	1	2	3	4	5
8. I found the system very Cumbersome/awkward to use	1	2	3	4	5
9. I felt very confident using the system	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this system	1	2	3	4	5

Abbildung 13.1: System Usability Scale Formular

kleine Einführung in die Datenbank und wurden außerdem ermuntert, bei Unklarheiten nachzufragen.

Nach Faulkner[Fau03] werden mit fünf Testpersonen im Durschschnitt ca. 85% Prozent der Fehler gefunden. Bei zehn Testpersonen sind es schon ca. 94%. Daher können schon mit einer relativen kleinen Anzahl an Testpersonen gute Ergebnisse erzielt werden.

13.1 Evaluation *TaP* v1.0

Da zum Zeitpunkt der ersten Evaluation noch kein Multitouch-Tisch verfügbar war, musste die Evaluation mit normaler Mausbedienung durchgeführt werden. Dadurch lassen sich einige Ergebnisse nur bedingt auf die Interaktion am Multitouch-Tisch übertragen. Es wurde allerdings trotzdem beschlossen die Evaluation zu diesem

Zeitpunkt durchzuführen, um die Ergebnisse gut in die weitere Planung einbeziehen zu können.

Die Evaluation wurde von jedem Mitglied der Projektgruppe mit 2 Probanden durchgeführt. Da *TaP* normalerweise in einer Büroumgebung eingesetzt werden soll, war Voraussetzung, dass die Umgebung entsprechend ungestört war.

13.1.1 Ablauf

Als Vorbereitung auf die Evaluation bekam der Proband eine Einführung durch den Testleiter. Dabei wurden folgende Punkte behandelt:

- Eine grobe Einführung in *Visual Analytics*, ohne Einführung in das multidimsionale Datenmodel, *Dimensionen* oder *Kennzahlen*.
- Eine Einführung in *AdventureWorks*, so dass ein Verständnis über die Art und die grobe Struktur der Daten herrscht.
- Keine Einführung in *TaP*, damit die intuitive Bedienung von *TaP* evaluiert werden kann.
- Eine Erklärung, welche Ziele diese Evaluation verfolgt und eine Einführung in die *Thinking Aloud* Methode 5.

Im nächsten Schritt wurde die eigentliche Evaluation, also die vier Aufgaben, am PC durchgeführt.

Nach Abschluss aller Testfälle wurde dem Probanden der *SUS* Fragebogen gestellt und demographische Daten erhoben. Im Anschluss an die Evaluation wurde dem Probanden die weitere Entwicklung unseres Projektes erläutert.

13.1.2 Aufgabenstellung

Jeder Proband musste vier Aufgaben erledigen, die typische Analyseszenarien darstellen sollten. Dabei wurden die Aufgaben bewusst zielorientiert gestellt, ohne dass darauf eingegangen wurde, mit welchen Funktionen von *TaP* eine Lösung erzielt werden könne. Für jede Aufgabe wurde dabei vom Testleiter notiert, wie lange der Proband für die jeweilige Aufgabe gebraucht hat und wie seine konkrete Lösung aussah.

Hier die gestellten Aufgaben:

Aufgabe 1 Lassen Sie sich anzeigen, wie viel Ihre Verkäufer (Reseller) pro Land (Country) bestellt haben. Die Länder finden Sie unter Dimension -> Geography -> Country und die Anzahl der Bestellungen Ihrer Reseller unter Kennzahl-> Reseller Orders -> Reseller Order Count.

Aufgabe 2 Verschaffen Sie sich einen Überblick über die Gesamtbestellungen (Order Quantity) in jedem Geschäftsjahr (Fiscal Year).

Aufgabe 3 Sie haben die Vermutung, dass Kunden mit mehr Kindern auch öfter im Internet bestellen. Versuchen Sie Zusammenhänge zu finden, die diese Vermutung unterstützen.

Aufgabe 4 Wählen Sie für die Anzeige die Kennzahl -> Internet Orders -> Internet Order Count und die Dimension -> Date -> Date. Nennen Sie das Datum, nach dem ein großer Sprung zu erkennen ist.

13.1.3 Auswertung

Die Auswertung der Messwerte und der *SUS* erfolgt im Abschnitt 13.3 im Vergleich mit der zweiten Evaluation.

Diese Auswertung stützt sich komplett auf die Kommentare und ist nach den einzelnen Komponenten gegliedert. Es gab auch einige Kommentare, die sich auf Unklarheiten mit der Datenbank bezogen. Diese Kommentare sind nicht extra aufgeführt, da dies dem Fakt geschuldet ist, dass keine echten Domänenexperten als Testpersonen zur Verfügung standen.

Pie-Menü

In Tabelle 13.1 sind die Kommentare der Benutzer über das Pie-Menü zusammen gefasst. Es fällt auf, dass es vielen Probanden nicht auffiel, dass sie das Pie-Menü drehen können. Außerdem wurde die Unübersichtlichkeit dieser Menüform bemängelt.

Problem	Häufigkeit
Rotierbarkeit des Pie-Menüs nicht erkannt	58.82 %
Allgemeine Unübersichtlichkeit bemängelt	35.29 %
Basis Buttons nicht als Buttons erkannt	11.76 %
Pfeile zur Anzeige der Drehbarkeit gewünscht	11.76 %
Nicht drehbare Ebenen versucht zu drehen	11.76 %
Umbruch der Beschriftungen bemängelt	11.76 %
„+“-Anzeige verwirrt oder missfallend	11.76 %
Physikalisches verhalten gewünscht	5.88 %

Hilfestellung (Text etc.) gewünscht	5.88 %
Drehen des Pie-Menüs führte zum Verschieben im Diagramm	5.88 %
Unendlichkeit des Pie-Menüs bemängelt	5.88 %

Tabelle 13.1: Tabelle mit den zusammengefassten Kommentaren für das Pie-Menü

Folgende Verbesserungsmaßnahmen wurden aus den aufgedeckten Schwächen geschlossen:

Drehbarkeit Die mehrfach gewünschte Anzeige von Pfeilen *Links* und *Rechts* am Pie-Menü zur Anzeige, dass das Pie-Menü drehbar ist. Diese Pfeile werden, wenn es nicht drehbar ist, weggelassen. Zusätzlich dienen die Pfeile als Buttons, mit denen sich das Pie-Menü drehen lässt.

Basis-Buttons Den Basis-Button vom Design näher an Standard Buttons orientiert.

Unübersichtlichkeit Eine Statusanzeige, ähnlich einem Scrollbalken, für das Pie-Menü, damit dem Nutzer gezeigt wird, wo er sich gerade befindet. Zusätzlich eine Anzeige, wieviele Elemente sich noch links beziehungsweise rechts verstecken. Außerdem klappen sich Ringe aus vorherigen Hierachien zusammen, um Platz zu sparen und nur noch den Pfad zum aktuellen Element anzuzeigen.

Unendlichkeit des Pie-Menüs Das Pie-Menü wird links und rechts gestoppt.

Zusätzlich wurde eine Evaluation geplant, um herauszufinden, ob andere Menüs besser zum Navigieren geeignet wären. Dabei wurden mehrere Prototypen für andere Menüformen entwickelt, aus Zeitgründen konnte die Evaluation jedoch nicht durchgeführt werden.

Drag and Drop

Wie Tabelle 13.2 zeigt, waren relativ viele Probleme mit *Drag and Drop* festzustellen, die daher rührten, dass ein Bug nur sehr langsames *Drag and Drop* zu ließ. Auch wurde *Drag and Drop* mit der rechten Maustaste als nicht gut empfunden.

Problem	Häufigkeit
Allgemeine Probleme mit *Drag and Drop*	47.06 %
Rechte Maustaste nicht erkannt	17.65 %
Drop-Zonen nicht erkannt	11.76 %
Unklarheiten ob Drop erfolgreich war oder nicht	11.76 %
Neustart des Programms wegen *Drag and Drop* Problemen	11.76 %
Unklar das *Drag and Drop* zur Auswahl der *Dimensionen* und *Kennzahlen* verwendet wird	11.76 %

Tabelle 13.2: Tabelle mit den zusammengefassten Kommentaren für *Drag and Drop*

Zur Verbesserung wurde der Bug behoben, so dass *Drag and Drop* auch bei sehr schnellen Bewegungen problemlos funktioniert. Außerdem wurde davon abgesehen, *Drag and Drop* mit einer speziellen Geste, wie der rechten Maustaste bzw. zwei Fingern, zu starten.

Diagramm

Wie die Tabelle 13.3 zeigt, gab es relativ wenig Kommentare zum Diagramm. Insbesondere bemängelten einige Benutzer, dass sie nur das alte Diagramm verändern konnten, es aber keine Funktion gab, um es zu löschen, zurückzusetzen oder ein neues Diagramm zu erstellen. Auch wurden oft die Achsenbeschriftungen nicht richtig erkannt, da sie sich überdecken können und die Schrift nicht mehr lesbar ist.

Problem	Häufigkeit
Diagramm resetten, löschen oder neu Funktion erwartet	29.41 %
Nicht gewusst, wie ein genaues Datum herausgefunden werden soll	23.53 %
Zoom Funktion nicht erkannt	17.65 %
Beschriftungen der Achsen nicht erkannt	17.65 %
Tooltip etc. bei einem Datenpunkt gewünscht	5.88 %
Vermutet das *Kennzahl* und *Dimension* statischer einer Achse zugeordnet sind	5.88 %

Tabelle 13.3: Tabelle mit den zusammengefassten Kommentaren für das Diagramm

Es wurden Funktionen zum Neu-Erstellen und Schließen von Diagrammen entwickelt. Um die Achsenbeschriftung zu entlasten wurden die Achsen nun so implementiert, dass nur in bestimmten Intervallen Beschriftungen angezeigt werden. Eine Umsetzung einer ToolTip-artigen Funktion wurde angedacht, da diese aber auf einem Touchscreen nicht trivial ist nicht umgesetzt.

13.2 Evaluation *TaP* v2.0

Die zweite Evaluation wurde kurz vor Ende der Projektgruppe mit der finalen Version von *TaP* durchgeführt. Dadurch war es nicht mehr möglich, entdeckte Fehler zu beheben. Trotzdem wurde der Test erst am Ende durchgeführt, da, bedingt durch die parallele Entwicklung der Komponenten, sonst nur sehr beschränkte Tests möglich gewesen wären. Des Weiteren wäre bei den beschränkten Tests keine Vergleichbarkeit zu der ersten Evaluation möglich gewesen. Da sich der Funktionsumfang von *TaP* erweitert hatten wurden die vier Fragen der ersten Evaluation beibehalten, um auf diesen Vergleiche machen zu können, und um weitere Fragen erweitert. Die Evaluation wurde

diesmal am Multitouch-Tisch durchgeführt. Insgesamt wurden fünf Testpersonen für die Evaluation ins *OFFIS* eingeladen.

Der Ablauf der ersten Evaluation wurde beibehalten (siehe 13.1.1).

13.2.1 Aufgabenstellung

Dies sind die Aufgaben, die den Probanden in der zweiten Evaluation gestellt wurden. Sie sind so gestellt, dass der gesamte Funktionsumfang von *TaP* genutzt werden kann. Bei einigen Aufgaben sind auch mehrere Lösungswege vorhanden: es wurde notiert, welcher Weg gewählt wurde. Die Aufgaben 1 - 4 der ersten Evaluation (siehe 13.1.2) entsprechen den Aufgaben 1a, 2b, 3a und 4a.

Aufgabe 1

a) Lassen Sie sich anzeigen, wie viel Ihre Verkäufer (Reseller) pro Land (Country) bestellt haben. Die Länder finden Sie unter TaP->Dimension -> Geography -> Country->Country und die Anzahl der Bestellungen Ihrer Reseller unter TaP->Measure-> Reseller Orders -> Reseller Order Count.

b) Erweitern Sie das Diagramm, so dass Sie auch den Reseller Gross Profit ablesen können. In welchem Land wurde der höchste Brutto-Profit erzielt und wie hoch fiel er aus?

Aufgabe 2

a) Sie möchten sich einem anderen Problem zuwenden. Schließen Sie das alte, nicht mehr benötigte Diagramm. Erstellen Sie ein neues Diagramm.

b) Verschaffen Sie Sich einen Überblick über die Gesamtbestellungen (Order Quantity) in jedem Geschäftsjahr (Fiscal Year).

c) Erweitern Sie dieses Diagramm um den Brutto-Profit (Gross Profit) und die Brutto-Gewinnspanne (Gross Profit Margin). In welchem Jahr wurde der höchste Profit gemacht und in welchem Jahr war die Gewinnspanne am höchsten?

d) Sie wollen die Gewinnspannen genauer untersuchen. Gibt es auf Quartalsebene auffällig niedrige Gewinnspannen?

Aufgabe 3

a) Sie haben die Vermutung, dass Kunden mit mehr Kindern auch öfter im Internet bestellen. Versuchen Sie Zusammenhänge zu finden, die diese Vermutung unterstützen.

b) Finden Sie heraus, ob die Erkenntnisse aus der vorherigen Aufgabe konstant über alle Jahre sind.

Aufgabe 4

a) Wählen Sie für die Anzeige die Kennzahl -> Internet Orders -> Internet Order Count und die Dimension -> Date -> Date. Nennen Sie das Datum nach welchem ein großer Sprung zu erkennen ist.

b) Sie finden diese Stelle besonders interessant. Speichern Sie diesen Zustand des Diagramms, damit Sie ihn später wiederfinden. Tauschen Sie nun Internet Order Count gegen Internet Customer Count aus und betrachten Sie das Ergebnis. Sie möchten nun die alte, interessantere Stelle einem Kollegen zeigen. Laden Sie diese dazu wieder.

13.2.2 Auswertung

Die Auswertung der Messwerte und SUS Punktzahl erfolgt im Abschnitt 13.3. Hier erfolgt analog zur Auswertung der ersten Evaluation nur die Auswertung der Kommentare unterteilt nach Komponenten.

Hilfe

Die Vermutung, dass die Geste zum Aufrufen der Hilfe bekannt wäre, wenn sie am Anfang eingeblendet wird, konnte, wie in Tabelle 13.4 dargestellt ist, nicht bestätigt werden. Im Allgemeinen wurde die Hilfe-Funktion nicht gerne benutzt. *TaP* sollte sich noch besser explorativ bedienen lassen, so dass die Hilfe im Normalfall nicht benötigt wird.

Problem	Häufigkeit
Hilfe wurde nicht gelesen	60.00 %
Hilfe erst nach Aufforderung benutzt	60.00 %
Geste zum Öffnen war nicht bekannt	40.00 %
Hilfe wurde missverstanden	40.00 %
Übersichts- oder Suchfunktion in der Hilfe gewünscht	20.00 %

Tabelle 13.4: Tabelle mit den zusammengefassten Kommentaren für die Hilfe

Diagramm

Beim Diagramm zeigte sich, dass die Benutzer mit den Drop-Zonen nicht zurecht kamen. Tabelle 13.5 zeigt, dass nur ein Benutzer die DropZone korrekt verstanden hatte. Die meisten Probanden belegten zwar nach einigen Versuch die X- und die Y-Achse richtig, übertrugen dieses Prinzip aber nicht auf die anderen DropZones. Auch tappten die meisten Anwender die DropZones in der Erwartung, dass sich nun Möglichkeiten zur Auswahl für diese Zonen ergeben würden. Die *Spread*-Geste wurde von jedem Benutzer für das Zoomen verwendet, ohne dass er Hilfe benötigte.

Problem	Häufigkeit
Drill Down nicht erkannt	80.00 %
Probleme „x“ zu treffen	80.00 %
Drop-Zonen nicht erkannt	80.00 %
Drop-Zonen wurden versucht zu tappen	80.00 %
Legende erst nach Hinweis entdeckt	60.00 %
Tauschen Funktion auf den Drop-Zonen gewünscht	60.00 %
Animation erst nach Hinweisen benutzt	60.00 %
Menüpunkt „Change“ unklar	60.00 %
Zoom-Geste auf Achse angewendet	40.00 %
„ToolTip“-Artige Funktion gewünscht	40.00 %
Kennzahl-Achse bemängelt	40.00 %
Diagramm Menü korrekt benutzt	40.00 %
Achsen tauschen Funktion gewünscht	40.00 %
„x“ zum Löschen der Elemente auf dem Diagramm nicht erkannt	40.00 %
Probleme mit der Kompatibilität auf den Drop-Zonen	20.00 %
Versucht Achsenbeschriftung zu tappen	20.00 %
Verkleinertes Diagramm für fehlerhaft gehalten	20.00 %

Tabelle 13.5: Tabelle mit den zusammengefassten Kommentaren für das Diagramm

Pie-Menü

Das Pie-Menü konnte in *TaP* v2.0 durch das Handauflegen beliebig platziert werden. Tabelle 13.6 zeigt, dass die Anwender erwarteten, das Pie-Menü nun auch verschieben zu können. Auch waren die Probanden meist verwirrt, wenn das Pie-Menü auf einmal verschwand und erst durch eine Geste wieder zum Vorschein gebracht werden musste. Die Farbgebung des Pie-Menüs wurde zwar nur in einem Fall korrekt gedeutet, allerdings kam es auch ohne die korrekte Deutung nur zweimal zu Problemen mit der Kompatibilität.

Problem	Häufigkeit
Verschieben Funktion für das Pie-Menü gewünscht	80.00 %
Farbgebung des Pie-Menüs nicht erkannt	80.00 %
Drehfunktion über Drag nicht entdeckt	60.00 %
Pie-Menü konnte nur mit Hilfe wieder sichtbar gemacht werden	60.00 %
Pie-Menü als umständlich oder unübersichtlich empfunden	40.00 %
Pie-Menü schließt sich ungewollt beim Tappen der Dreh-Buttons	40.00 %
Versucht Elemente zu draggen die nicht dragbar sind	40.00 %
Automatisches Schließen des Pie-Menü als störend empfunden	20.00 %

Tabelle 13.6: Tabelle mit den zusammengefassten Kommentaren für das Pie-Menü

Desktop

Bei dem Desktop gab es, wie in Tabelle 13.7 zu sehen ist, wenig Kommentare. Einige Benutzer scheinen, Gesten die recht lange Aktionen auf dem Tisch enthalten, wie die *Neues-Diagramm*-Geste oder *Drag and Drop*, als unangenehm zu empfinden. Außerdem wurde einmal erwartet, dass für ein neues Diagramm ein *L* zu zeichnen ist, wie es bei vielen Programmen, die Mausgesten besitzen der Fall ist.

Problem	Häufigkeit
Fände „*L*“-Geste für neues Diagramm intuitiv	20.00 %
Empfindet *Neues-Diagramm*-Geste umständlich	20.00 %
Drag and Drop als umständlich empfunden	20.00 %

Tabelle 13.7: Tabelle mit den zusammengefassten Kommentaren für den Desktop

13.3 Vergleich

Die beiden Evaluationen dienten nicht nur zur Aufdeckung von Bedienungsschwachstellen, sondern auch zum Vergleich der Versionen v1.0 und v2.0 von *TaP*. Dabei wird anhand der Kommentare betrachtet, ob die Schwachstellen aus v1.0 behoben wurden. Außerdem werden die SUS Punktzahlen und Messwerte verglichen.

Im Diagramm 13.3 werden die durschnittlichen SUS Punktezahlen verglichen. Dabei ist festzustellen, dass sich die Gesamtzufriedenheit um ca. 15 Punkte verbessert hat. In Abbildung 13.2 sind die einzelnen Fragen im Detail gezeigt. Da jede zweite Frage des SUS eine negative Usability bedeutet wurden bei diesen Fragen die Antworten invertiert, um eine bessere Vergleichbarkeit zu erreichen. Es fällt auf, dass in allen Bereichen

Verbesserungen erzielt wurden. Besonders Frage 2 („I found the system unnecessarily complex“) und Frage 8 („I found the system very cumbersome/awkward to use“) wurden vom Benutzer besser bewertet. Dies bedeutet, dass die Benutzer *TaP* v2.0 trotz gestiegen Funktionsumfang subjektiv als weniger komplex und intuitiver empfanden. Es lassen sich keine eindeutigen Rückschlüsse über die Ursachen für die Verbesserungen ziehen. Eine Vermutung ist aber, dass die Bedienung am Tisch als angenehmer empfunden wurde als die Bedienung mit der Maus. Besonders sollte dieser Effekt beim Pie-Menü, das speziell an die Form der Hand angepasst wurde, auftreten. Die Messwerte bestehen sowohl aus den Zeiten, die bei den vier Aufgaben gemessen wurden, als auch aus dem Datum, das in Aufgabe 4 ermittelt wurde.

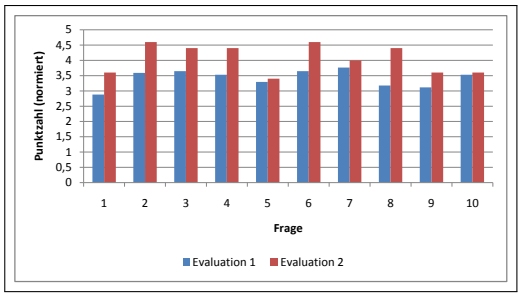


Abbildung 13.2: Vergleich der SUS Punktzahlen nach Fragen

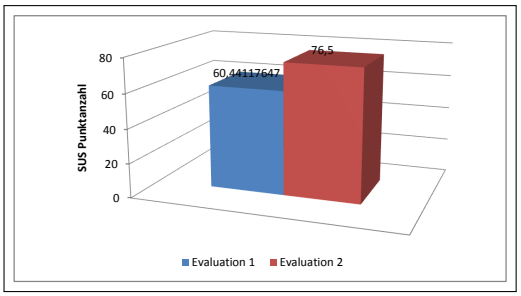


Abbildung 13.3: Vergleich der SUS Punktzahlen

Abbildung 13.4 zeigt den Vergleich der einzelnen Zeiten der beiden Evaluationen. Die Zeiten sind sehr ähnlich, nur in Aufgabe 3 zeigt sich eine merkbare Verbesserung der *TaP* v2.0 im Vergleich zu v1.0. Die Schwierigkeit in dieser Aufgabe war, eine passende Dimension und Kennzahl auszusuchen. Hier ist anzunehmen, dass die Anpassung des Pie-Menüs nach der ersten Evaluation zu Verbesserungen geführt hat. Der Vergleich der Gesamtzeiten in Abbildung 13.5 zeigt wieder, dass keine signifikanten Unterschiede festzustellen sind.

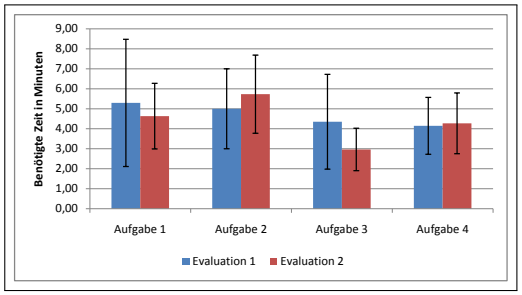


Abbildung 13.4: Vergleich der benötigten Zeiten für die vier Aufgaben

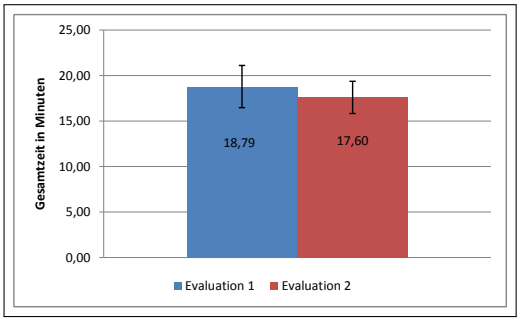


Abbildung 13.5: Vergleich der insgesamt benötigten Zeit für die 4 bei beiden Evaluationen gleichen Aufgaben.

Bei der vierten Aufgabe sollte ein Tag rausgesucht werden, an dem die Daten einen auffälligen Sprung machen. Es wurde gemessen, wie stark das von den Probanden ermittelte Datum von dem erwarteten Datum abwich. Wie in Abbildung 13.6 zu sehen ist, besitzen die Ergebnisse der zweiten Evaluation stärkere Abweichung. Dies ist wahrscheinlich auf die neue Darstellung des Diagramms zurückzuführen, welche für die bessere Lesbarkeit modifiziert wurde. Nun werden nur noch so viele Achsenbeschriftungen angezeigt, wie auch an die Achse passen. In der Evaluation wurde dabei entdeckt, dass dies dazu führt, dass dem Benutzer nicht klar ist, dass noch mehr Werte verfügbar wären und er daher nicht auf die Idee kommt, weiter in diesen Bereich zu zoomen. Hier müsste angezeigt werden, dass Werte ausgelassen wurden.

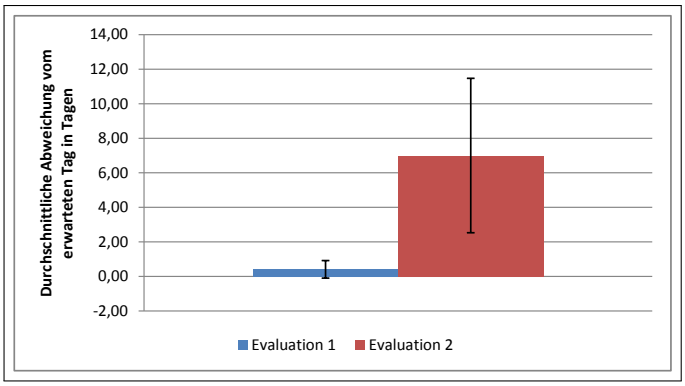


Abbildung 13.6: Vergleich der Abweichungen vom erwarteten Ergebniss

Auch in der zweiten Evaluation sind viele Schwächen in der Bedienbarkeit aufgedeckt worden. Allerdings ist zu beachten, dass in der zweiten Evaluation viele Probleme, die in der ersten Evaluation bemängelt wurden, nicht mehr auftraten. So konnte von allen Probanden das Pie-Menü ohne Hilfe benutzt werden. Die Rotierbarkeit über die Buttons wurde korrekt erkannt und benutzt. Auch traten keine Probleme mit *Drag and Drop* mehr auf. *Drag and Drop* war meist nicht die erste Methode, welche die Testpersonen zur Datenauswahl versuchten, aber sie versuchten es immer recht schnell ohne Hilfestellung. Beim Diagramm wurden die Schließen- und Neu-Erstellen-Funktionen angenommen, die von einigen Probanden in der ersten Evaluation gewünscht waren. Die Achsenbeschriftungen waren nun problemlos lesbar, aber hatten nun die oben erwähnten Nachteile. In beiden Evaluationen traten einige Probleme mit dem Verständnis der Datenbank auf. Dabei kamen die Benutzer immer recht gut mit der hierarchischen Struktur an sich klar, hatten aber Probleme mit dem konkreten Aufbau von *AdventureWorks*. Auch die Unterscheidung von *Dimensionen* und *Kennzahlen* war den Benutern bei beiden Evaluationen oft unklar.

Teil VI

Abschluss

Kapitel 14

Zusammenfassung

Dieser Bericht stellt die Ergebnisse des einjährigen Projektes *Visual Analytics* vor. Im Rahmen dieses Projektes wurde eine Anwendung zur visuellen Datenanalyse entwickelt, die mittels eines Multitouch-Tisches bedient wird. Die Anwendung ermöglicht die Darstellung mehrerer Dimensionen und Kennzahlen in Form eines Punktdiagramms, das über zusätzliche Visualisierungsdimensionen erweitert werden kann. Diese Erweiterungen sind Größe und Farbe der Punkte, sowie eine (zeitliche) Animation der Punkte. Als weitere Funktion bietet die Anwendung *TaP* die Speicherung und Wiederherstellung von Diagrammzuständen. Innerhalb des Diagramms kann über Funktionen wie *Verschieben*, *Vergrößern* und *Verkleinern* mit den Daten interagiert werden. Auch ein *Hinein-* und *Herauszommen* in die bzw. aus den Daten, d. h. die schrittweise Verfeinerung der Daten, wird direkt im Diagramm ermöglicht.

TaP verfügt über ein Datenbank-Framework, das so erweitert werden kann, dass beliebige multidimensionale Datenbanken von *TaP* angesprochen werden können.

Die Auswahl der Daten wird über ein Half-Pie-Menü vorgenommen. Dieses Menü ist speziell für die Bedienung mittels eines Multitouch-Tisches entwickelt worden, was beispielsweise an der runden, an die Handform angepassten Menüführung deutlich wird.

Die gesamte Interaktion findet über Gesten statt, die mit den Händen und Fingern auf der Oberfläche des Multitouch-Tisches ausgeführt werden. Die Wahl der Gesten wurde insbesondere anhand der Faktoren Intuition und Natürlichkeit vorgenommen. So wird ein Element z. B. durch eine Berührung der Tischoberfläche an entsprechender Stelle ausgewählt und mittels einer Fingerbewegung bewegt.

Eine Usability-Evaluation rundete die Entwicklung ab und gab Erkenntnisse hinsichtlich der Gebrauchstauglichkeit des Systems. Daraus abgeleitete und anschließend durchgeführte Verbesserungsmaßnahmen von *TaP* führten zu dem hier vorgestellten Endprodukt.

Kapitel 15

Ausblick

Die vorgestellte Anwendung ermöglicht die Erstellung von Punktdiagrammen in Verbindung mit weiteren Visualisierungsdimensionen wie Farbe, Größe und Animation. Um beispielsweise ortsbezogene Daten besonders anschaulich darzustellen, bieten sich Landkarten an. *TaP* kann um weitere Visualisierungsformen, wie z. B. eine thematische Karte, ergänzt werden und dadurch eine noch anschaulichere visuelle Datenanalyse ermöglichen.

Um gezielter in den Daten navigieren zu können, bietet sich darüber hinaus die Integration einer Filterfunktion in *TaP* an. Eine solche Funktion ermöglicht dem Nutzer, die Visualisierung auf einen bestimmten Datenbereich zu beschränken. Dieser Datenbereich kann beispielsweise aus einzelnen Knoten verschiedener Dimensionen bestehen.

TaP ist für die Nutzung durch eine Person konzipiert worden. Damit mehrere Personen gleichzeitig an unterschiedlichen Diagrammen arbeiten können, ist die Unterstützung mehrerer Nutzer notwendig. TaP kann um einen solchen Multi-User-Support erweitert werden, sodass die Anwendung auf dem Multitouch-Tisch für parallele, unabhängige Datenanalysen genutzt werden kann.

Derzeit ist die Seite des Multitouch-Tisches, von der aus *TaP* bedient wird, festgelegt. Damit ein Nutzer unabhängig von seinem Standpunkt *TaP* ideal bedienen kann, ist eine Erweiterung der Anwendung notwendig. Legt der Nutzer zu Beginn der Interaktion seine Hand auf den Multitouch-Tisch, so kann beispielsweise mit Hilfe der Handposition festgestellt werden, an welcher Stelle des Tisches der Nutzer steht. Dementsprechend könnte sich *TaP* dann in diese Position ausrichten.

TaP verwendet multidimensionale Datenbanken als Datenquelle. Um auch Live-Daten, die z. B. mit Hilfe von Messgeräten erstellt wurden, mit *TaP* visualisieren zu können, kann beispielsweise ein Datenstrommanagementsystem angebunden werden.

Sind mehrere Visualisierungsformen für *TaP* entwickelt worden, so kann die Verwendung einer eigenen Beschreibungssprache in Betracht gezogen werden. Angelehnt an eine *Domain Specific Language (DSL)* würde eine solche Beschreibungssprache

die Austauschbarkeit verschiedener Visualisierungen und die Entwicklung weiterer Visualisierungsformen erleichtern. Insbesondere würde dadurch die Spezifizierung neuer Visualisierungsformen für Personen ohne Programmierkenntnisse ermöglicht werden. Darüber hinaus bestünde die Möglichkeit, Interaktionsschritte, die auf einer Visualisierung und bestimmten Daten durchgeführt wurden, in ihrer Abfolge zu speichern, um diese anschließend auf anderen Daten oder einer anderen Visualisierung durchzuführen bzw. anzuzeigen.

Kapitel 16

Persönliches Fazit

Die vergangenen zwölf Monate haben uns viel über das Arbeiten im Team und den Umgang mit neuen Technologien gelehrt. Wir hatten die Möglichkeit, ein agiles Projektmamangement auszuprobieren und in die Programmiersprache C♯ mit der Technologie *WPF* Einblick zu erhalten.

Auf Grund des verwendeten agilen Projektmanagements war es uns möglich, spontan auf Anforderungsänderungen zu reagieren und terminliche Verschiebungen ohne größere Probleme zu bewältigen, wie sie sich beispielsweise bei der Durchführung der Evaluationen ergaben. Hierdurch ist es uns gelungen, ein funktional vollständiges und stabiles System zu entwickeln. Wir, die Projektgruppe, sind der Meinung, dass wir auf das entstandene System stolz sein können. In diesem Glauben werden wir durch mehrere Artikel bestärkt, die bereits über *TaP* und unser Pie-Menü veröffentlicht worden sind.

Die Projektgruppenkasse mit Gesamteinnahmen von 745,92 €, die besonders durch Strafen gefüllt wurde, freut sich bereits darauf, geleert zu werden, was wir uns reichlich verdient haben. Insgesamt sind wir nun zwar weiser, dafür aber auch ärmer. Neben der Bewältigung der Arbeitspakete, die allesamt auf den Standardzeitaufwand „halbe Stunde, ohne Doku!“ geschätzt wurden, gab es auch eine Exkursion nach München, sowie spezielle Social Events wie zum Beispiel einige Grill- oder Videoabende, die uns ebenfalls in guter Erinnerung bleiben werden. Auch beim Bowling haben wir gezeigt, dass wir mit der Punktzahl von 191 Punkten gut dabei sind. Schade nur, dass der Geschwindigkeitsrekord von 38,3 km/h von dem Betreiber als Vollpfostenrekord abgetan wurde.

Wir bedanken uns bei den Betreuern Tobias und Stefan für die sehr gute Betreuung und freuen uns auf weitere Zusammenarbeit zum Beispiel in Master- oder Diplomarbeiten wie auch bei individuellen Projekten.

Bis dahin,

Touch and Pray

Teil VII

Anhang

Literatur

[And72] Andrews, D. F.: Plots of High-Dimensional Data. In: *Biometrics* 28 (1972), Nr. 1, S. 125–136. – ISSN 0006341X

[Asi08a] Adobe Systems Incorporated: *Alchemy:FAQ*. Website, 2008. – Online verfügbar unter `http://labs.adobe.com/wiki/index.php/Alchemy:FAQ`; letzter Abruf: 30.11.2008

[Asi08b] Adobe Systems Incorporated: *Graphics Acceleration (GPU) support in Adobe Creative Suite 4 applications*. Website, 2008. – Online verfügbar unter `http://kb.adobe.com/selfservice/viewContent.do?externalId=kb405445`; letzter Abruf: 26.11.2008

[Ban06] Bange, Carsten: *Werkzeuge für analytische Informationssysteme*. Springer Berlin Heidelberg, 2006

[BB05] Burger, Wilhelm ; Burge, Mark J.: *Digitale Bildverarbeitung – Eine Einführung mit Java und ImageJ – 2. Auflage*. Springer-Verlag Berlin Heidelberg, 2005

[BKJ05] Bencina, Ross ; Kaltenbrunner, Martin ; Jordà, Sergi: Improved Topological Fiducial Tracking in the reacTIVision System. In: *Proceedings of the IEEE International Workshop on Projector-Camera Systems (Procams 2005)*. San Diego, USA, 2005

[Bol09] Boll, Susanne: *Vorlesung Mensch-Maschine-Interaktion*. Vorlesung, 2009. – Vorlesung an der Carl-von-Ossietzky Universität Oldenburg, Wintersemester 2008 / 2009.

[Bro96] Brooke, J.: SUS: A quick and dirty usability scale. In: Jordan, P. W. (Hrsg.) ; Weerdmeester, B. (Hrsg.) ; Thomas, A. (Hrsg.) ; Mclelland, I. L. (Hrsg.): *Usability evaluation in industry*. London : Taylor and Francis, 1996

[Bux07] Buxton, Bill: *Multi-Touch Systems that I Have Known and Loved*. `http://www.billbuxton.com/multitouchOverview.html`. Version: 2007, Abruf: 14.12.2008

[CCS93] Codd, E F. ; Codd, S B. ; Salley, C T.: Providing OLAP to user-analysts: An IT mandate / Technical Report, E.F. Codd and Associates. 1993. – Forschungsbericht

[Chr08] Christian, Moser: *WPF Tutorial - Value Resolution Strategy*. Website, 2008. – Online verfügbar unter http://www.wpftutorial.net/ValueResolutionStrategy.html; letzter Abruf: 07.12.2008

[CM84] Cleveland, W. S. ; McGill, R.: M. E.: Dynamic Graphics for Statistics. In: *Journal of the American Statistical Association* 79 (1984), S. 807 – 822

[Deg06] Degen, Horst: *Statistische Methoden zur visuellen Exploration mehrdimensionaler Daten*. Springer Berlin Heidelberg, 2006

[Dör03] Dörfler, Nikolas: *Hidden Markov Models*. November 2003

[Edm09] Edmonson, Clint: *Coding Standards Reference Documents*. http://www.notsotrivial.net/blog/post/2008/12/Holiday-Goodie-Bag-Free-C-and-VB-Coding-Standards-Reference-Documents.aspx, 01 2009

[Fau03] Faulkner, Laura: Beyond the Five-User Assumption: Benefits of Increased Sample Sizes in Usability Testing. In: *Behavior Research Methods, Instruments, & Computers* 35 (2003). http://tc.eserver.org/27411.html

[FB90] Feiner, S. ; Beshers, C.: Visualizing n-dimensional virtual worlds with n-vision. In: *Computer Graphics* 24 (1990), S. 37–38

[Fer06] Fernicola, Pablo: *When to use WPF and when to use other technologies*. Website, 2006. – Online verfügbar unter http://www.fernicola.org/loquitor/index.php?/archives/19-When-to-use-WPF-and-when-to-use-other-technologies.html; letzter Abruf: 26.11.2008

[FHTA09] Flöring, Stefan ; Hesselmann, Tobias ; Teiken, Yvette ; Appelrath, Hans-Jürgen: Kollaborative visuelle Analyse multidimensionaler Daten auf Surface-Computern. In: *DBSpektrum* (2009)

[Fin] *Fingerworks*. http://www.fingerworks.com, Abruf: 15.12.2008

[GG93] Gilb, Tom ; Graham, Dorothy: *Software Inspection*. Addison-Wesley, 1993

[GW] Gruber, Hermann ; Wimmer, Felix: *BubbleSort*. http://www2.tcs.ifi.lmu.de/~gruberh/lehre/sorting/Bubble/Bubble.html, Abruf: 09.09.2009

[Han05] Han, Jefferson Y.: Low-Cost Multi-Touch Sensing through Frustrated Total Internal Reflection / Media Research Laboratory - New York University. 2005. – Forschungsbericht

[HD05] Hartog, Vic ; Doomen, Dennis: *Coding Standard: C*. http://www.tiobe.com/content/paperinfo/gemrcsharpcs.pdf, 05 2005

[HGM02] Hake, Guenter ; Gruenreich, Dietmar ; Meng, Liqiu: Kartographie: Visualisierung raum-zeitlicher Informationen. (2002)

[HMDa] *Praxis der Wirtschaftsinformatik | Gesamtglossar | S*. http://hmd.dpunkt.de/glossar/gesamt/s.html#smoke-build, Abruf: 16.09.2009

[HMDb] *Praxis der Wirtschaftsinformatik | Gesamtglossar | S*. http://hmd.dpunkt.de/glossar/gesamt/s.html#smoke-test, Abruf: 16.09.2009

[Hof09] Hofert, Svenja: *Sicher auftreten, gekonnt überzeugen*. Version: 2009. http://www.teialehrbuch.de/Kostenlose-Kurse/Sicher-Auftreten-Gekonnt-Ueberzeugen/images/17.jpg, Abruf: 29.12.2008

[HP07] Huber, Thomas ; Pletz, Christoph: Das Model View ViewModel-Pattern für WPF-Anwendungen. In: *dot.net magazin* 10 (2007), S. 2–10. – Online verfügbar unter http://www.thomasclaudiushuber.com/articles/ModelViewViewModelArticle.pdf; letzter Abruf: 22.03.2009

[Hub08] Huber, Thomas C.: *Windows Presentation Foundation*. Galileo Press, 2008

[ICS98] IEEE Computer Society, Software Engineering Review Working G. t.: IEEE standard for software reviews. (1998), März. http://www.ieee.org/

[Inc08] Inc., Swype: *Swype, Inc. Text Input for Screens*. Version: 2008. http://www.swypeinc.com/, Abruf: 30.03.2009

[ISO] ISO: ISO-Norm 9241.

[KBBC05] Kaltenbrunner, M ; Bovermann, T ; Bencina, R ; Costanza, E: TUIO: A protocol for table-top tangible user interfaces. In: *Proc. of the The 6th International Workshop on Gesture in Human-Computer Interaction and Simulation*, 2005

[Kei02] Keim, Daniel: Datenvisualisierung und Data Mining. In: *Datenbank-Spektrum* 2 (2002), S. 30–39

[Keo01] Keogh, M E. & P. E. & Pazzani: Derivative Dynamic Time Warping. In: *First SIAM International Conference on Data Mining*. Chicago, USA : SDM, 2001

[KG08] Khronos Group: *OpenGL*. Website, 2008. – Online verfügbar unter http://www.opengl.org; zuletzt besucht am 26.11.2008.

[KH90] Kurtenbach, G. ; Hulteen, E.A.: Gestures in Human-Computer Communication. In: *The Art of Human-Computer Interface Design*. Kurtenbach G. und Hulteen, 1990, S. 309–317

[Kle07] Kleinau, Jens P.: *(MVC) Model-View-Controller - reloaded as (MVVM) Model-View-ViewModel*. http://www.codecomplete.de , 2007

[Klu09] Klumpp, Bruno: *Radar-Diagramme*. Version: 2009. http://www.methode.de/am/di/amdi05.htm, Abruf: 29.12.2008

[KMS+] Keim, Daniel A. ; Mansmann, Florian ; Schneidewind, Jorn ; Thomas, Jim ; Ziegler, Hartmut: Visual Analytics: Scope and Challenges / Universität Konstanz und Pacifc Northwest National Laboratory, National Visualization and Analytics Center (NVAC). http://nvac.pnl.gov. – Forschungsbericht

[KOC04] Kaltenbrunner, Martin ; O'Modhrain, Sile ; Costanza, Enrico: Object Design Considerations for Tangible Musical Interfaces. In: *Proceedings of the COST287-ConGAS Symposium on Gesture Interfaces for Multimedia Systems*. Leeds, UK, 2004

[Krü03] Krüger, Mike: *C Coding Style Guide*. Version: 2003. http://www.icsharpcode.net/TechNotes/SharpDevelopCodingStyle03.pdf

[Kös07] Köster, Frank: *Multidimensionale Datenbanken*. Carl-von-Ossietzky Universität Oldenburg - Vorlesung Informationssysteme II, 2007

[LK99] Lee, Hyeon kyu ; Kim, Jin H.: An HMM-based Threshold Model Approach for Gesture Recognition. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 21 (1999), S. 961–973

[LL07] Leete, Gurdy ; Leete, Mary: *Microsoft Expression Blend Bible*. Wiley, 2007

[LWW90] LeBlanc, J. ; Ward, M. O. ; Wittels, N.: Exploring n-dimensional databases. In: *Proc. Visualization* (1990), S. 230–239

[Mac08] MacSlow: *Using cairo with OpenGL*. http://www.cairographics.org/OpenGL/ : Website, 2008

[MHB99] Milan, Sonka ; Hlavac, Vaclav ; Boyle, Roger: *Image Processing, Analysis, and Machine Vision – 2nd ed.* PWS Publishing, 1999

[Mica] *Rounded corners on a StackPanel?* http://social.msdn.microsoft.com/Forums/en-US/wpf/thread/3364bdd1-0e74-41cb-9cb9-d91f02443ceb, Abruf: 14.09.2009

[Micb] Microsoft: *AdventureWorks-Beispieldatenbanken*. http://msdn.microsoft.com/de-de/library/ms124501.aspx, Abruf: 09.09.2009

[Micc] Microsoft: *Übersicht über angefügte Ereignisse*. http://msdn.microsoft.com/de-de/library/bb613550.aspx, Abruf: 30.03.2009

[Micd] Microsoft: *Downloaddetails: DirectX-Endbenutzer-Runtime*. http://www.microsoft.com/downloads/details.aspx?FamilyID=2da43d38-db71-4c1b-bc6a-9b6652cd92a3&displaylang=de, Abruf: 09.09.2009

[Mice] Microsoft: *Informationen zu Expression Blend*. http://msdn.microsoft.com/de-de/library/cc296376.aspx, Abruf: 08.09.2009

[Micf] Microsoft: *Microsoft Corp .net framework general reference — design guidelines for class library developers*. http://msdn.microsoft.com/library/en-us/cpgenref/html/cpconnetframeworkdesignguidelines.asp

[Micg] Microsoft: *Microsoft Corp. Visual studio .net coding techniques and programming practice*. http://msdn.microsoft.com/library/en-us/vsent7/html/vxconCodingTechniquesProgrammingPractices.asp

[Mic08a] Microsoft: *Dependency Property Value Precedence*. Website, 2008. – Online verfügbar unter http://msdn.microsoft.com/en-us/library/ms743230.aspx; letzter Abruf: 07.12.2008

[Mic08b] Microsoft: *DirectX*. Website, 2008. – Online verfügbar unter http://msdn.microsoft.com/de-de/directx/default(en-us).aspx; letzter Abruf: 26.11.2008

[Moo08] Moore, J. A.: *Touchlib Documentation*. Version: 2008. http://easterisland-llc.com/touchlib/main.html, Abruf: 11.12.2008

[Mos03] Moser, Heinrich: *Auswirkungen von Code Conventions auf Software Wartung und Evolution*. Version: 08 2003. http://www.heinzi.at/texte/codeconv.pdf

[msd] *SQL Server 2008-Onlinedokumentation - ADOMD.NET*. http://msdn.microsoft.com/de-de/library/ms123483.aspx, Abruf: 15.09.2009

[MTG08] Music-Technology-Group: *reacTIVision*. Version: 2003-2008. http://mtg.upf.es/reactable/?software, Abruf: 11.12.2008

[Nat06] Nathan, Adam: *Windows Presentation Foundation Unleashed (WPF) (Unleashed)*. Indianapolis, IN, USA : Sams, 2006. – ISBN 0672328917

[NGC] Nui-Group-Community: *Getting Started With MultiTouch*. http://nuigroup.com/forums/viewthread/1982/, Abruf: 14.12.2008. Forum

[NGC08a] Nui-Group-Community: *Touchlib – A Multi-Touch Development Kit*. Version: 2006-2008. http://www.nuigroup.com/touchlib/, Abruf: 18.10.2008

[NGC08b] Nui-Group-Community: *tbeta preview*. Version: 2008. http://tbeta.nuigroup.com/, Abruf: 11.12.2008

[Nui08] Version: September 2008. http://www.nuigroup.com/touchlib/, Abruf: 17.09.2008

[nul06] Software entwickeln mit System: oose Engineering Process (OEP). (2006), 11. http://www.oose.de:8080/oep/

[Oel02] Oellien, Frank: *Algorithmen und Applikationen zur interaktiven Visualisierung und Analyse chemiespezifischer Datensätze*, Naturwissenschaftlichen Fakultäten der Friedrich-Alexander-Universität Erlangen-Nürnberg, Diplomarbeit, 2002. http://www2.ccc.uni-erlangen.de/people/Frank_Oellien/diss/

[OIEE01] Oellien, F. ; Ihlenfeldt, W.-D. ; Engel, K. ; Ertl, T.: Multi-Variate Interactive Visualization of Data from Laboratory Notebooks. In: *In: Proceedings ECDL: Workshop 'Generalized Documents'* Sep. 2001 (2001)

[oos] oose: *oose Innovative Informatik GmbH*. www.oose.de, Abruf: 30.03.2009. Webseite

[Ope] OpenGL: *OpenGL Overview*. http://www.opengl.org/about/overview/, Abruf: 09.09.2009

[OQS] Olbrich, Gerold ; Quick, Michael ; Schweikart, Jürgen: Desktop Mapping: Grundlagen und Praxis in Kartographie und GIS.

[Osc03] *Open Sound Control: State of the Art 2003*. Montreal, 2003 . – 153–159 S. – OpenSound Control

[Osh08] Osherove, Roy: *the art of UNIT TESTING with Examples in .NET*. Manning Publications, 2008 http://www.manning-sandbox.com/forum.jspa?forumID=355

[OW08] Oestereich, Bernd ; Weiss, Christian: *APM - Agiles Projektmanagement*. dpunkt.verlag, 2008. – ISBN 978–3–89864–386–3

[Par09] Parlament, Europäisches: *Kreisdiagramm*. Version: 2009. http://www.europarl.de/export/system/galleries/bilder/stimmen_parteien_kreis.gif, Abruf: 29.12.2008

[Pen08] Pendse, Nigel: *The OLAP Report*. Version: März 2008. http://www.olapreport.com/, Abruf: 27.03.2009

[Per06] Perzold, Charles: *Anwendung=Code+Markup*. Microsoft Press, 2006

[Phi] Philipp, Dennis: Interaktive Visualisierung von Kennzahlen mit Geografiebezug auf Basis von Kartendiensten.

[Pis07] Pisarevsky, Vadim: *Introduction to OpenCV*. Version: 2007. http://downloads.sourceforge.net/opencvlibrary/opencv_introduction_2007June9.pdf?modtime=1181770251&big_mirror=1, Abruf: 11.12.2008

[Pop07] Poppinga, Benjamin: *Beschleunigungsbasierte 3D-Gestenerkennung mit dem Wii-Controller*. Oldenburg, August 2007

[Röd07] Rödiger, Marcus: Studienausarbeitung - Multitouch - Technik und Technologien / Hochschule für Angewandte Wissenschaften Hamburg. Version: 6 2007. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2007/roediger/bericht.pdf. 2007. – Studienausarbeitung

[Rot07] Roth, Tim: *FTIR vs DI at*. Version: 10 2007. http://iad.projects.zhdk.ch/multitouch/?p=47, Abruf: 14.12.2008

[Saf08] Saffer, Dan: *Designing Gestural Interfaces*. O'Reilly, 2008

[San97] Sandberg, Anders: *Gesture Recognition using Neural Networks*. Juni 1997

[SB06] Sarodnick, Florian ; Brau, Henning: *Methoden der Usability Evaluation: Wissenschaftliche Grundlagen und praktische Anwendung*. 1. Huber, Bern, 2006

[SBD+08] Schöning, Johannes ; Brandl, Peter ; Daiber, Florian ; Echtler, Florian ; Hilliges, Otmar ; Hook, Jonathan ; Löchtefeld, Markus ; Motamedi, Nima ; Muller, Laurence ; Olivier, Patrick ; Roth, Tim ; Zadow, Ulrich von: Multi-Touch Surfaces: A Technical Guide / University of Munster. Version: 2008. http://ifgiweb.uni-muenster.de/~j_scho09/pubs/TUM-I0833.pdf. 2008. – Forschungsbericht

[Sch] Schwichtenberg, Dr. H.: *Reflection - Begriffserklärung im Entwickler-Lexikon/Glossar auf www.IT-Visions.de*. http://www.it-visions.de/glossar/alle/303/Reflection.aspx, Abruf: 19.08.2009

[Sch07] Schneidewind, Jörn: *Scalable Visual Analytics : Solutions and Techniques for Business Applications*, Universität Konstanz, Diplomarbeit, 2007. http://www.ub.uni-konstanz.de/kops/volltexte/2007/3746/

[Sch08] Schmidt, Toni: *Interaction Concepts for Multi-Touch User Interfaces: Design and Implementation*. http://kops.ub.uni-konstanz.de/volltexte/2009/7239. Version: 2008

[Sco92] Scott, D. W.: *Multivariate Density Estimation*. Wiley-Interscience, 1992

[Shn83] Shneiderman, Ben: Direct Manipulation: A Step Beyond Programming Languages. In: *IEEE Computer* 16 (1983), Nr. 8, S. 57–69

[Shn92] Shneiderman, B.: Tree visualization with treemaps: A 2D space-filling approach. In: *ACM Transactions on Graphics* 11 (1992), S. 92–99

[SM00] Schumann, Heidrun ; Müller, Wolfgang: *Visualisierung: Grundlagen und allgemeine Methoden*. Bd. 1. Auflage. Berlin, Heidelberg, New York, 2000

[SN] Schulz, Hans-Jörg ; Nocke, Thomas: *Maschinelle Datenanalyse im Informationszeitalter ? Können oder müssen wir ihr vertrauen?*

[sta09] Statista, Statistik Portal: *Körpergröße*. Website, 2009. – Online verfügbar unter `http://de.statista.com/statistik/diagramm/studie/341/umfrage/koerpergroesse/`; letzter Abruf: 19.02.2009

[Str97] Strecker, Stefan: *Künstliche Neuronale Netzwerke - Aufbau und Funktionsweise*. Mainz, 1997

[TC05] Thomas, James J. ; Cook, Kristin A.: *Illuminating the Path - The Research and Development for Visual Analytics*. National Visualizaition and Analytics Center, 2005

[Tea08] Team libSDL: *Simple DirectMedia Layer*. Website, 2008. – Online verfügbar unter `http://www.libsdl.org`; zuletzt besucht am 26.11.2008

[Tuf83] Tufte, E. R.: *The Visual Display of Quantitative Information*. 1983

[Tuk77] Tukey, J. W.: *Exploratory Data Analysis*. 1977

[Wal08] Wallin, David: *Whitenoise Audio Blog*. Version: 2006-2008. `http://www.whitenoiseaudio.com/blog/archive/2006_08_01_archive.html`, Abruf: 11.12.2008

[Wal09] Waloszek, Gerd: *Interaction Design Guide for Touchscreen Applications*. Website, 2009. – Online verfügbar unter `http://www.sapdesignguild.org/resources/TSDesignGL/`; letzter Abruf: 19.02.2009

[Wik] Wikipedia: *Totalreflexion - Wikipedia*. `http://de.wikipedia.org/wiki/Totalreflexion`, Abruf: 14.12.2008

[Wik09a] Wikipedia: *Liniendiagramm*. Version: 2009. `http://de.wikipedia.org/wiki/Liniendiagramm`, Abruf: 29.12.2008

[Wik09b] Wikipedia: *Mosaikplot*. Version: 2009. `http://de.wikipedia.org/wiki/Mosaikplot`, Abruf: 29.12.2008

[Wik09c] Wikipedia: *Säulendiagramm*. Version: 2009. http://de.wikipedia.org/wiki/Säulendiagramm, Abruf: 29.12.2008

[Wri02] Wright, Matthew: Open Sound Control 1.0 Specification. (2002). http://opensoundcontrol.org/spec-1_0

[WW07] Westphal, Ralf ; Weyer, Christian: *.NET 3.0 kompakt*. Spektrum Akademischer Verlag, 2007

[Zad07] Zadow, U.: *Using libavg as a Tracking Library*. Version: 12:18 Uhr, 28.10.2007. https://www.libavg.de/wiki/index.php/Using_libavg_as_a_Tracking_Library, Abruf: 14.12.2008

[Zha09] Zhao, Haixia: *Fitts Law: Modeling Movement Time in HCI*. Website, 2009. – Online verfügbar unter http://www.cs.umd.edu/class/fall2002/cmsc838s/tichi/fitts.html; letzter Abruf: 19.02.2009

Abkürzungen

API	Application Programming Interface.
AVM2	ActionScript Virtual Machine 2.
CCV	Community Core Vision.
CLR	Common Language Runtime.
DI	Diffused Illumination.
DLL	Dynamic Link Library.
DSI	Diffused Surface Illumination.
DSL	Domain Specific Language.
FTIR	Frustrated Total Internal Reflection.
GDML	Gesture Definition Markup Language.
GNU	GNU is Not Unix.
GPL	General Public License.
GPU	Graphics Processing Unit.
GUI	Graphical User Interface.
HCI	Human-Computer-Interaction.
HMM	Hidden Markov Model.
ID	Identifier.
IDE	Integrated Development Environment.
KNN	Künstliches neuronales Netz.
LINQ	Language Integrated Query.
LLTK	Low Level ToolKit.
MIDI	Musical Instrument Digital Interface.
MUSTANG	Multidimensional Statistical Data Analysis Engine.
MVC	Model View Controller.
MVVM	Model-View-ViewModel.

OLAP	Online-Analytical Processing.
OpenCV	Open Source Computer Vision Library.
OSC	Open Sound Control.
OWL	Object Window Library.
PDF	Portable Document Format.
RDBMS	Relationales Datenbank Management System.
SDL	Simple DirectMedia Layer.
SUS	System Usability Scale.
SVG	Scalable Vector Graphics.
TCP	Transmission Control Protocol.
TDNN	Time Delay Neural Network.
UDP	User Datagram Protocol.
WPF	Windows Presentation Foundation.
XAML	eXtensible Application Markup Language.
XML	eXtensible Markup Language.

Glossar

Additives Faktum
Ein Fakt, dass sich aus der Aggregation anderer Fakten einer Hierarchiestufe ergibt (siehe auch *Fakt*).

ADOMD.NET
„*ADOMD.NET ist ein Microsoft.NET Framework-Datenanbieter, der auf die Kommunikation mit Microsoft SQL Server Analysis Services ausgelegt ist. ADOMD.NET verwendet das XML for Analysis-Protokoll für die Kommunikation mit analytischen Datenquellen, indem entweder TCP/IP- oder HTTP-Verbindungen für die Übertragung oder den Empfang von SOAP-Anfragen oder -Antworten eingesetzt werden, die mit der XML for Analysis-Spezifikation kompatibel sind. [...]*“ [msd] .

AdventureWorks
AdventureWorks ist eine multidimensionale Datenbank von Microsoft für den SQL Server, in welcher Daten wie Produkte, Kunden und Mitarbeiter einer fiktiven Firma enthalten sind. Die AdventureWorks Datenbank gibt es für Onlinetransaktionsverarbeitung, analytische Onlineverarbeitung und Data Warehouses. (vgl. [Micb]).

Attached Dependency Property
Eine Funktionalität von WPF die es ermöglicht, Klassen außerhalb ihrer Definition nachträglich um Eigenschaften zu erweitern.

Blasendiagramm
Ein Blasendiagramm ist ein erweitertes Streudiagramm, wobei weitere Merkmale durch die Größe oder die Farbe des Punktes möglich sind.

Blob
Engl. für Klecks. Region, die bei der Berührung einer Eingabefläche mit optischem Verfahren entsteht und sich als heller Fleck in den Kamerabildern zeigt. Im Zusammenhang mit Multitouch-Gesten wird der Begriff Blob häufig synomym zu der verursachenden Berührung verwendet .

Blob Detection
Vorgang, der *Blobs* auf der Multitouch-Oberfläche detektiert.

Blob Tracking
Vorgang, der die *Blobs* auf der Multitouch-Oberfläche über die Zeit verfolgt.

Bubblesort
„*BubbleSort ist ein einfach zu verstehender Sortieralgorithmus: Es werden soviele Runden absolviert wie die Reihung lang ist; In jeder Runde werden von vorne bis hinten je zwei Werte verglichen. Wenn der "linke-Wert größer ist als der rechte, werden die beiden Werte vertauscht. So steigen in jeder Runde größere Werte – wie Blasen im Wasser – von unten nach oben (daher der Name).*“ [GW].

Choroplethenkarte
Eine Choroplethenkarte ist eine flächenbezogene Karte, die über eine Vielzahl von raumbezogenen Informationen verfügt.

Closed Source
Unter Closed Source wird die nicht-Preisgabe des Quelltextes einer Software verstanden. Damit hat nur der Softwareentwickler, welcher die Software entwickelt hat, die Möglichkeit, die Software zu verändern.

Community Core Vision
Community Core Vision ist ein Open Source Programm, welches *Blobs* innherhalb eines Videostreams erkennt und diese mit Hilfe von TUIO an eine Anwendung weiterleiten kann (siehe auch *TUIO*).

Cube
Aus dem englischen für Datenwürfel (siehe auch *Datenwürfel*).

Data Mining
Data Mining ist ein systematisches Verfahren zur Erkennung von Mustern und Anomalien auf meistens großen Datenbeständen. Die verwendeten Verfahren sind häufig statistisch-mathematisch begründet.

Datenwürfel
Ein Datenwürfel ist ein mehrdimensionaler Würfel, welcher zur logischen Darstellung von multidimensionalen Daten verwendet wird.

Dicing
Bei einem *Dicing* handelt es sich um eine Operation auf einem Datenwürfel. Es handelt sich um das Herausschneiden eines Teilwürfels aus einem Datenwürfel (siehe auch *Datenwürfel*).

Diffused Illumination
Diffused Illumination ist eine Technologie, mit welcher ein Multitouch-Tisch betrieben werden kann. Da hier *Blobs* mit Hilfe von Infrarotlicht und einer Kamera

erkannt werden, handelt es sich hier um eine visuelle Technologie. Für weitere Informationen siehe Abschnitt 3.1.2.

Dimension
Mögliche Sicht auf Daten in einer multidimensionalen Datenbank, welche unterschiedliche Hierarchien und Stufen besitzen kann.

DirectX
„*Microsoft DirectX stellt Technologien bereit, durch die auf Windows-Computern Anwendungen mit vielfältigen Multimedia-Elementen ausgeführt und angezeigt werden können, wie z. B. Farbgrafiken, Video, 3D-Animationen und Audio. DirectX enthält Updates zur Verbesserung der Sicherheit und Leistung des Computers sowie viele neue Features für alle Technologien, auf die von Anwendungen über die DirectX-API zugegriffen werden kann.*“ [Micd].

DOLAP
Desktop Online-Analytical Processing. Diese Art der OLAP-Architektur besteht aus einem Server, welcher die Daten hält und Clients, welche relativ kleine Datenmengen aus dem Datensatz extrahieren. Die Weiterverarbeitung der Daten erfolgt auf dem Client (siehe auch *Online Analytical Processing*).

DrillAcross
Bei einer *DrillAcross*-Operation wird zwischen unterschiedlichen Datenwürfeln gewechselt.

DrillDown
Bei einem *DrillDown* handelt es sich um eine Operation auf einem Datenwürfel. Hierbei entsteht eine höhere Detailstufe in den Daten durch das Wechseln auf eine tiefere Aggregationsstufe einer Hierarchie.

DrillThrough
Bei einem *DrillThrough* handelt es sich um eine Operation auf einem Datenwürfel. Hiermit wird eine Verzweigung auf eine andere meist feinere Datenquelle erreicht.

Fakt
Synomym für Kennzahl (siehe auch *Kennzahl*).

Fiducial
engl. für Bezugswert/Bezugspunkt. Gemeint sind optische Markierungen, die beispielsweise auf *Tangibles* angebracht werden, um sie eindeutig identifizieren zu können.

Front-DI
Front-DI ist eine Abwandlung der *Diffused Illumination*-Technologie, bei der die

obere Schicht des Touchsensors von oben mit Infrarotlicht bestrahlt wird. Für weitere Informationen siehe Abschnitt 3.1.2.

Frustrated Total Internal Reflection
Frustrated Total Internal Reflection ist eine Technologie, mit welcher ein Multitouch-Tisch betrieben werden kann. *Blobs* werden mit Hilfe von Infrarotlicht und einer Kamera erkannt werden, somit handelt es sich um eine visuelle Technologie. Für weitere Informationen siehe Abschnitt 3.1.1.

Graphical User Interface
Grafische Benutzungsoberfläche die als Schnittstelle zur Bedienung des Programms durch den Benutzer fungiert.

Hidden Markov Model
Stochastiches Modell zur Mustererkennung, basierend auf einem dynamischen Bayes'schen Netz.

Hierarchie
Ordnung in der Elemente anderen untergeordnet sein können. Dimensionen enthalten Hierarchien.

Hierarchiestufe
Dies ist eine Stufe in einer Hierarchie. Zum Beispiel sind *Jahr, Monat, Tag* Hierarchiestufen einer Hierarchie in der Dimension *Zeit*.

HOLAP
Hybrid Online-Analytical Processing. Hierbei handelt es sich um eine Mischvariante aus *MOLAP* und *ROLAP* (siehe auch *Online Analytical Processing*).

Inspektion
Nach DIN Verfahren aus der Qualitätsicherung bei dem ein Dokument von einem Prüfer daraufhin untersucht wird, ob es den definierten Ansprüchen genügt.

Kennzahl
Eine Kennzahl (*analog: Fakt / engl. measure*) ist in der Regel ein numerischer Wert, welcher über mehrere Stufen aggregiert werden kann.

Kennzahl-Gruppe
Eine Sammlung von meist semantisch zusammgehörigen Kennzahlen.

Knoten
Ein Knoten ist ein Element innerhalb einer Hierarchiestufe. Zum Beispiel sind *2003, 2004, 2005* Knoten der Hierarchiestufe *Jahr*.

Kreisdiagramm
Ein Kreisdiagramm ist eine Darstellungsform für Teilwerte eines Ganzen als Teile eines Kreises.

Künstliches neuronales Netz
Model das den menschlichen neuralen Netzen nach gebildet wurde um Entscheidungsprozesse zu simulieren.

Level
siehe Hierarchiestufe.

Likert-Skala
Skala zur Messung persönlicher Einstellungen aus der Sozialforschung.

Liniendiagramm
Ein Liniendiagramm ist die graphische Darstellung mehrerer Merkmale in Linienform.

Log4Net
Log4Net ist ein Logger für .Net-Anwendungen, welcher über eine XML-Datei konfiguriert werden kann. Er steht kostenlos unter `http://logging.apache.org/log4net/index.html` zur Verfügung.

MDX
MDX steht für Multidimensional Expressions und ermöglicht das Abfragen von mehrdimensionalen Objekten mit einer SQL-ähnlichen Syntax. Auch die Rückgabe von mehrdimensionalen Datensätzen, welche die Inhalte der merhdimensionalen Objekte enthalten sind mit MDX möglich.

Measure
Synonym für Kennzahl. (siehe auch *Kennzahl*).

Mehrschichtige Karte
Eine mehrschichtige Karte ist eine Karte, bei der eine Überlagerung mehrerer Kartenschichten erfolgt.

Meilenstein
„*Ein Meilenstein definiert einen Termin, zu dem eine Menge von Ergebnissen in einer bestimmten Detaillierung und Vollständigkeit nachprüfbar und formal vorliegen muss. Liegen die Ergebnisse zum geplanten Termin nicht vor, wird der Meilensteintermin verschoben. Ein Meilenstein ist ein Hilfsmittel zur Planung und Überwachung eines Entwicklungsprozesses.*“[OW08].

Microsoft Expression Blend

„*Microsoft Expression Blend ist ein professionelles Designtool mit umfangreichen Features zum Erstellen ansprechender und komplexer Benutzeroberflächen für Anwendungen auf Basis von Microsoft Windows und Microsoft Silverlight.*“[Mice].

MOLAP

Multidimensional Online-Analytical Processing. Multidimensionale Architektur für OLAP-Systeme (siehe auch *Online Analytical Processing*).

Mosaikplot

Ein Mosaikplot ist das graphische Verfahren zur Visualisierung von Datensätzen mit zwei oder mehreren qualitativen Merkmalen in Form mehrerer Flächen.

Namescope

Beschreibt den Gültigkeitsbereich eines Variablennamens in einer Programmiersprache.

Netzdiagramm

Ein Netzdiagramm ist die grafische Darstellung von Werten mehrerer, gleichwertiger Merkmalen in einer Spinnennetzform.

Nicht-additives Faktum

Ein Fakt, das nicht aus anderen Fakten berechnet ist.

Node

Siehe *Knoten*.

Online Analytical Processing

Bei einem *Online Analytical Processing*-System handelt es sich um ein analytisches Informationssystem. Es kann seine Daten aus einem *Data-Warehouse* beziehen und speichert hochdimensionale Daten in Form von Datenwürfeln.

oose Engineering Process

Der *oose Engineering Process* ist ein Vorgehensleitfaden für die objektorientierte Softwareentwicklung. Nähere InforFür nähere Informationen lesen Sie Kapitel 6.1.1 oder besuchen [oos].

Open Sound Control

Protokoll das ursprünglich zur Übertragung von Multimediainhalten über Netzwerke entwickelt wurde. Wird auch für die Übertragung von TUIO-Daten verwendet.

Open Source

Unter Open Source wird die preisgabe des Quelltextes einer Software verstanden. Der Softwareentwickler stellt den gesamten Quelltext seiner Software evtl. unter einer speziellen Lizenz anderen Entwicklern zur Verfügung.

OpenGL

OpenGL ist eine Umgebung für die Entwicklung portabler, interaktiver 2D- und 3D-Grafik-Anwendungen. Seit seiner Einführung im Jahr 1992 wurde OpenGL zur am meisten in der Industrie genutzten und unterstützten 2D- und 3D-Grafik-Programmierschnittstelle (vgl. [Ope]).

Pivotierung

Bei einer *Pivotierung* handelt es sich um eine Operation auf einem Datenwürfel. Der Datenwürfel wird gedreht, wodurch eine neue Perspektive auf die Daten entsteht.

Rauschdaten

Rauschdaten sind meistens nicht determnistisch berechenbare ungewollte Daten, welche das Nutzdatensignal stören oder sogar überlagern können.

Rear-DI

Rear-DI ist eine spezialisiertere Bezeichnung der *Diffused Illumination*-Technologie. Bei dieser Technologie wird die Projektionsschicht von unten mit Infrarotlicht bestrahlt. Für weitere Informationen siehe Abschnitt 3.1.2.

Reflection

„*Die Gewinnung von Metainformationen aus einer Komponente selbst wird im Allgemeinen als Relexion (engl. Reflection) bezeichnet [Griffel: Componentware, Siete 471]. Reflection ist die Möglichkeit, per Programmcode auf die Meta-Daten einer Softwarekomponente zuzugreifen.*“[Sch] .

Review

Reviews dienen zur Fehlerüberprüfung eines Arbeitsergebnisses. Sie zählen zu der Gruppe der analytischen Maßnahmen zur Qualitätssicherung. Im Gegensatz zu den testenden Verfahren, wie zum Beispiel Unit-Tests, wird auf eine Ausführung des Prüflings mit konkreten Eingaben verzichtet. Ziel eines Reviews sind das Auffinden von statischen Fehlern. Für weitere Informationen siehe Abschnitt 12.2.1.

ROLAP

Relational Online-Analytical Processing. Eine Möglichkeit ein OLAP-System umzusetzen, indem es auf einer relationalen Datenbank aufsetzt (siehe auch *Online Analytical Processing*).

RollUp
Bei einem *RollUp* handelt es sich um eine Operation auf einem Datenwürfel. Innerhalb einer *Hierarchie* einer *Dimension* wird auf eine höhere Aggregationsstufe gewechselt.

Slicing
Ausschneiden von Scheiben aus einem Datenwürfel.

Smoke-Build
„*Ein Smoke-Build ist ein Build, das mit dem Ziel erstellt wird, einen* Smoke-Test *durchzuführen.*“ [HMDa].

Smoke-Test
„*Dieser normalerweise automatisierte Test soll möglichst alle Hauptfunktionen eines Testobjektes auslösen, ohne die Ausgaben des Testobjektes mit vorgegebenen Sollergebnissen zu vergleichen, mit dem Ziel, die grundsätzliche Testbarkeit und Robustheit (z.B. Absturzfreiheit) zu prüfen. [...]*“ [HMDb].

Snowflakeschema
Ein Schema für relationale Datenbanken, welches beim OLAP und Data Warehousing eingesetzt wird. Erweiterung des Starschemas, wobei die Dimensionstabellen klassifiziert und normalisiert werden.

Standortkarte
Eine Standortkarte ist die lagegetreue Visualisierung qualitativer Informationen auf einer Karte.

Starschema
Ein Schema für relationale Datenbanken, welches beim OLAP und Data Warehousing eingesetzt wird. Denormalisiert und somit optimiert auf schnelle Leseoperationen.

Streudiagramm
Ein Streudiagramm ist die graphische Darstellung mehrerer Merkmale, die in ein kartesisches Koordinatensystem in Form mehrerer Punkten eingetragen werden.

Stylus-Stift
Ein *Stylus-Stift* ist eine Art minenloser Stift, der genauere Eingaben auf Touch-Geräten ermöglicht, als sie mit den Fingern möglich sind.

Subcube
Ein Subcube ist kleinerer Datenwürfel innerhalb eines Datenwürfels. Meistens umfassen Subcubes nur bestimmte Kennzahlen aus einer Kennzahl-Gruppe.

System Usability Scale
Test zur subjektiven Bewertung einer Benutzungsoberfläche [Bro96].

Säulendiagramm
Ein Säulendiagramm ist die graphische Darstellung mehrerer Merkmale in Form mehrerer Balken.

Tangible
Tangible ist ein Sammelbegriff für zusätzliche Gegenstände, die zur Bedienung eines Multitouch-Geräts eingesetzt werden können. Der Programmfluss kann beispielsweise gesteuert werden indem ein Tangible von einem Ort an einen anderen verschoben wird oder wenn es gedreht wird. Außerdem kann das Programm in einen bestimmten Modus wechseln oder eine bestimmte Aktion durchführen, wenn ein Tangible auf der Oberfläche des Multitouch-Geräts platziert wird.

Template
Eine WPF-Komponente kann in ihrem Aussehen durch ein Template verändert werden. Dabei ersetzt das Template das Erscheinungsbild der Komponente komplett.

Thinking Aloud
Thinking Aloud (deutsch: lautes Denken) ist eine Methode aus der Psychologie die im Usability-Engineering benutzt wird. Dabei erzählt der Proband laut alle seinen Gedanken bei der Benutzung des Systems und erklärt dadurch seine Handlungen und Eindrücke [SB06].

Thresholding
Bildverarbeitungsverfahren zur Binärisierung, das mit einem Schwellenwert arbeitet. Pixel, die unterhalb dieses Wertes liegen, erhalten den Farbwert 0. Pixel mit einem höheren Farbwert erhalten den Wert 255.

Totalreflexion
Unter der Totalreflexion wird „*[...]ein optisches Phänomen, bei dem elektromagnetische Strahlung an der Grenzfläche zweier Medien nicht gebrochen, sondern vollständig reflektiert wird [...]*“ verstanden (vgl. [Wik]).

TUIO
TUIO ist ein offenes, allgemeines Protokoll für Multitouchanwendungen. Es beschreibt Touchereignisse auf abstrakte Weise.

Unit-Test
Ein Unit-Test beschreibt einen Code-Abschnitt (in den meisten Fällen eine Methode), welcher einen anderen Code-Abschnitt aufruft und anschließend die Korrektheit und einige Annahmen testet. Wenn die Ergebnisse der Methode unterschiedlich zu den Annahmen sind, dann gilt der Unit-Test als fehlgeschlagen. (vgl. [Osh08]).

Visifire

Visifire ist eine Sammlung von Charting-Controls für die Datenvisualisierung in WPF und Silverlight.

Visual Analytics

Visual Analytics ist die Wissenschaft der analytischen Begründung, welche durch interaktive visuelle Interfaces unterstützt wird [TC05].

Visual Analytics Mantra

Gängiges Vorgehen bei der visuellen explorativen Analyse: „*Analyse first - show the important - zoom, filter and analyse further - details on demand*“ [KMS+].

Wiki

Ein Wiki ist ein so genanntes Hypertext-System. Die Inhalte dieses Systems können in den meisten Fällen von allen gelesen werden und zusätzlich von registrierten Benutzern auch verändert werden. Diese Funktion ähnelt einem Content-Management-System und wird durch die Wiki-Software bereitgestellt. Um z. B. Tabellen oder Bilder als Inhalte einzufügen, wird eine einfach zu erlernende Markup-Sprache verwendet.

Stichworte