Bachelorarbeit: Ein Datenflusseditor zum Rapid-Prototyping von Multitouch-Anwendungen

vorgelegt von Frederic Raber

am 31. August 2010

Betreut von

Dr. Michael Kipp

Begutachtet von

Dr. Michael Kipp
Prof. Dr. Dr. h.c. mult. Wolfgang Wahlster

Abstract

Meine Arbeit beschäftigt sich mit dem Design eines grafischen Datenflusseditors speziell für Multitouch-Inputsysteme. Als Hardware wird hierzu ein experimentelles FTIR-Display [10] des DFKI verwendet. Dabei war es auch Teil meiner Arbeit, dieses Display aufzubauen, zu kalibrieren und die einkommenden Rohdaten aufzubereiten. Letztendlich musste ein Interface gefunden werden, um die aufbereiteten Daten mit meinem Editor empfangen zu können. Das Hauptziel meines Datenflusseditors liegt in der Erfassung des Design Space. Es soll möglich sein, verschiedene mögliche Designs einfach zu evaluieren und zu kommunizieren. Dazu soll die Erstellung dieser Systeme grafisch erfolgen. Ähnlich wie in elektronischen Schaltkreisen gibt es Module, die Anschlusspins besitzen, über die sie miteinander verbunden werden können. Dadurch soll z.B. eine Zoom-Steuerung mit einem Datenflussdiagramm designt werden können, indem zwei Module die Koordinaten der Finger auf dem Display ausgeben, und ein weiteres daraus die Distanz berechnet, welche den Zoom-Faktor angibt. Eine seperate Laufzeitumgebung analysiert den erstellten Graphen und interpretiert ihn. Die erstellten Systeme sollen auch in einem fertigen Projekt verwendet werden können. Dadurch dass neue Module sehr einfach erstellt werden können, kann das Programm auch für andere Input-Systeme verwendet werden. Es müssen lediglich die entsprechenden Module implementiert werden.

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, d. 31.08.2010

Inhaltsverzeichnis

1 Einleitung

Multitouch-Geräte finden eine immer stärkere Verbreitung, auch im privaten Umfeld. Single-Touch-Geräte können einen Punkt erkennen, an dem der Benutzer den Bildschirm beispielsweise mit seinem Finger oder einem speziellen Stift berührt. Auch die Maus ist ein solches Single-Point Interface. Beim Multitouch-Konzept können mehrere Punkte erkannt werden. Dadurch ergeben sich neue Möglichkeiten. Distanzen und Winkel können mit einer einzigen Aktion anhand zweier Punkte festgelegt werden. Die Benutzung von Multitouch-Geräten ist intuitiver als bei konventionellen Eingabegeräten. Benutzer können mit ihrem Finger Objekte verschieben oder mit einem zweiten Finger drehen, wie sie es aus der realen Welt gewohnt sind. Zum anderen können dadurch Interaktionen zwischen mehrerern Benutzern stattfinden, insofern das Display genug Platz bietet. Es können gleichzeitig verschiedene Personen am Display arbeiten und so kollaborieren. Es gibt mehrere Studien wie z.B. [6], die belegen dass Touch-Geräte der konventionellen Steuerung per Maus in vielen Disziplinen überlegen ist. Die Steuerung wird intuitiver und schneller. Lediglich die Fehlerrate erhöht sich, da die jetzigen Geräte für präzise Operationen noch zu ungenau sind.

Nachteilig ist, dass Arbeitsplätze und Heim-Computer in der Regel nicht mit multitouch-fähigen Geräten ausgestattet sind. Es sind erst Neuanschaffungen nötig. Es gibt verschiedene Implementationen des Multitouch-Prinzips. Displays für Smartphones und Handhelds nutzen beispielsweise eine Oberfläche, die elektrische Spannung trägt. Sobald der Finger die Oberfläche berührt, stört er dieses elektrische Feld. Diese Störungen werden erkannt und and die Software als Positionskoordinaten weitergegeben. Diamondtouch [4] benutzt eine solche kapazitive Sensorik. Bei ihm besitzt die Tischoberfläche ein Gitter aus Sensoren. Der Benutzer steht unter einer elektrischen Spannung. Sobald er mit seinem Finger diese Oberfläche berührt, wird in den nahegelegenen Sensoren eine Spannung induziert, durch die der Punkt der Berührung erkannt werden kann. Diese Master-Arbeit [15] beschreibt dieses und andere Multitouch-Implementationen detaillierter im Abschnitt „Technology Survey“. Eine andere bekannte Implementation ist das FTIR-Display wie es in [10] beschrieben wird. Abbildung 1 zeigt ein solches Display im Betrieb. Der Vorteil dieser Geräte ist die günstige Herstellung und beliebige Skalierbarkeit. Auch meine Arbeit beinhaltet den Aufbau, Inbetriebnahme und Anbindung eines solchen Displays an meine Hardware.

Abbildung 1. Jeff Han bei der Präsentation eines FTIR-Displays

Lemur[1] war 2005 eines der ersten Geräte, das Multitouch-Funktionen bereitstellt. Der Handheld beinhaltet Instrumente wie einen Synthesizer, verschiedene virtuelle Instrumente und ein Mischpult. Die bekanntesten Vertreter von Multitouch-Geräten sind allerdings Smartphones wie das iPhone[2] (Abbildung 2), die dieser Technologie erstmals zu einer weiten Verbreitung verhalfen. Dieses Gerät nutzt auch in den vorinstallierten Anwendungen seine Multitouch-Fähigkeit aus, um z.B. anhand der Distanz zweier Finger ein Bild oder eine Website zu zoomen. Es gibt vielfältige Applications für das iPhone, mit denen mit Multitouch-Unterstützung Bilder gemalt, oder eine Handschrift erkannt werden kann. In Abbildung 2 wird mit zwei Fingern das Skateboard in einem Skatingsport-Spiel gesteuert.

Abbildung 2. Multitouch beim iPhone

Auch *Surface*[3] (Abbildung 3) von Microsoft verwendet Multitouch zur Erfassung von Benutzeraktionen. Es ist wie ein Tisch aufgebaut, wobei die Oberfläche aus einem Multitouch-Screen besteht. Surface ist besonders auf die Interaktion zwischen mehreren Benutzern per Multitouch ausgelegt. Es können zum einen mehrere Finger von verschiedenen Benutzern erkannt werden. Zum anderen kann die Oberfläche auch andere Gegenstände wie Digitalkameras oder Handys erkennen und eine Interaktion mit ihnen ermöglichen. Das Hauptanwendungsgebiet von Surface liegt im Businessbereich, da die Anschaffung sehr teuer ist. Es soll dort besonders zur Verbesserung und Vereinfachung von kollaborativer Arbeit dienen.

1. http://www.jazzmutant.com/lemur_overview.php
2. http://www.apple.com/de/iphone
3. http://www.microsoft.com/surface

Abbildung 3. Microsoft Surface im Betrieb

Eine weiteres, momentan sehr bekanntes Multitouch-Projekt ist der reacTable. Sein Aufbau wird in [12] beschrieben. Der reacTable ist ein Tabletop Tangible Interface um einen Software-Synthesizer zu steuern. Ähnlich wie bei Surface geschieht auch hier die Steuerung nicht nur durch Finger. Es gibt verschiedene physische Bausteine, die auf der Oberfläche abgelegt werden können, wie Abbildung 4 zeigt. Benachbarte Objekte interagieren miteinander und bilden so das Synthesizer-System.

Abbildung 4. Benutzersteuerung beim reacTable

Meine Arbeit stellt ein grafisches Tool vor, welches die Erstellung eines Input-Systems für solche Multitouch-Geräte erleichtern soll. Das Design erfolgt dabei durch ein Datenflussdiagramm, wie es in Abbildung 5 zu sehen ist. Der mit „Mouse“ beschriftete Knoten schickt die

Position des Mauszeigers an den Knoten „Distance“. „ConstPoint“ sendet einen konstanten Punkt an den anderen Eingang des Distance-Knotens. Dieser berechnet daraus die Distanz der beiden Punkte und sendet diese Information an einen anderen Knoten „Print“, der diese Informationen aufnimmt und auf der Konsole ausdruckt. In Anlehung an das Anwendungsgebiet *Multitouch* und das zugrunde liegende Konzept eines Datenflussdiagramms, engl. *dataflow diagram*, wird dieses Programm daher *TouchFlow* genannt. Um die Datenflusspläne zu modellieren, existiert eine grafische Entwicklungsumgebung, die per Maus bedient wird. Die erstellten Diagramme werden durch eine Laufzeitumgebung ausgeführt.

Der Datenfluss fließt durch verschiedene Knoten, die Module genannt werden. In Codeprojekten werden oft gleiche Berechnungen durchgeführt. So werden zum Beispiel die Positionen der Finger auf dem Display abgefragt und anhand derer die Distanz oder der Winkel zwischen ihnen berechnet. Solche Berechnungen werden in meinem Projekt in Module gekapselt. Es gibt verschiedene Typen von Modulen. Input-Module besitzen keine eingehenden Datenleitungen und können Daten generieren. Sie repräsentieren in der Regel Eingabegeräte oder konstante Werte. Modify-Module führen Berechnungen wie z.B. die Errechnung der Distanz zwischen zwei übergebenen Punkten durch. Sie haben dazu Ein- und Ausgänge. Die Daten fließen also in sie hinein und in einer anderen Form wieder heraus. Output-Module besitzen lediglich Eingänge, sie nehmen also Daten auf. Diese können dann beispielsweise auf der Konsole ausgegeben werden. Die Verbindung der Module untereinander erfolgt durch Datenleitungen und Pins. Eingänge und Ausgänge eines Moduls werden durch Pins repräsentiert, die durch Datenleitungen miteinander verbunden werden können. Durch diese Leitungen werden die Daten automatisch übertragen. Auf der Entwicklungsumgebung können per Maus Module platziert und ihre Pins miteinander verbunden werden. Diese Umgebung unterstützt auch das direkte Starten eines modellierten Systems über das Laufzeitsystem. Dieses kann entweder in der Entwicklungsumgebung erstellte, oder als Datei vorliegende Graphen laden und ausführen.

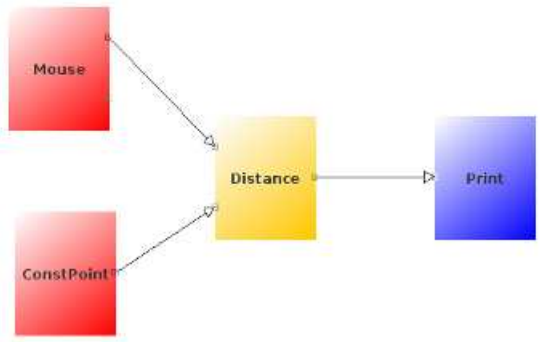

Abbildung 5. Datenflussdiagramm in TouchFlow

Im Vergleich zum Coding per Hand gibt es dadurch mehrere Vorteile. Zum einen wird dadurch die Entwicklung und das Re-Design erleichtert. In einem Datenflussdiagramm können auch grundlegende Änderungen schnell durchgeführt werden, da Berechnungen als Block ausgetauscht und Daten durch verlegen der Verbindungen zu anderen Zielen fließen können. Die Kommunikation eines grafisch erstellten Systems fällt deutlich einfacher als mit Sourcecode. Diagramme erlauben es oft auf den ersten Blick das Design eines Inputsystems zu verstehen, wohingegen Quellcode erst nach längerer Betrachtung verständlich ist.

Es gibt bereits ähnliche Datenflusseditoren für andere Anwendungsgebiete. *Diamodl* ist beispielsweise ein Tool zum Design von User-Interfaces anhand einer Mischung der beiden Konzepte Domain Modeling und Dialog Modeling. Auch hier gibt es verschiedene Arten von Modulen, die Benutzersteuerelemente, Berechnungen oder konkrete Datensätze repräsentieren können. Die Art von Modulen ist weitreichender als in meinem Projekt, wo Module nur Berechnungen oder Eingabegeräte symbolisieren. Die Interaktion der verschiedenen Module wird ebenfalls über Datenleitungen durchgeführt. Eine nähere Erläuterung zu den Konzepten des Domain und Dialog Modeling sowie der Funktionalität von diamodl enthält Kapitel 2.1.

Auch VVVV ist verwandt zu meiner Arbeit. Es benutzt ein Datenflussdiagramm, um einen Software-Synthesizer zu modellieren. Die Module sind hier Bausteine wie Oszillatoren, Filter oder Verzerrer. Die von diesen Elementen produzierten Audio-Daten werden auch über Verbindungsleitungen zum nächsten Baustein transportiert.

TouchFlow kann in vielen Anwendungsgebieten eingesetzt werden, die auch über das Gebiet des Multitouch hinaus gehen. Durch den modularen Aufbau kann meine Implementierung um beliebige andere Eingabegeräte erweitert werden. So können auch andere neuartige Eingabegeräte wie die WiiMote oder eine Bewegungssteuerung wie Microsoft Kinect eingebunden werden. Entsprechende Modify-Module können bereits eine Vorverarbeitung der Daten dieser Eingabegeräte vornehmen und so Arbeitsaufwand aus dem eigentlichen Projekt ausgliedern.

1.1 Motivation

Die klassische Herangehensweise ist die Erstellung eines Input-Systems direkt in Quellcode. Das hat allerdings einige Nachteile im Vergleich zu einer grafischen Bearbeitung.

- Der erstellte Code ist nicht flexibel, da bei jeder Änderung der Code neu compiliert werden muss.
- Das gleiche Problem ergibt sich bei der Evaluierung der Designalternativen. Oft muss für jedes Design eigener Code geschrieben und debuggt werden.
- Die Kommunikation der implementierten Designs gestaltet sich bei Quellcode am schwierigsten. Selbst für Fachexperten dauert es einige Zeit, um den Code zu verstehen und das von ihm repräsentierte Design abzuleiten. Dritte Personen, die nicht mit Programmiersprachen vertraut sind, können die Funktionsweise des Codes und so auch des Designs gar nicht nachvollziehen.

Um das Problem der untransparenten Kommunikation zu lösen, werden oft ergänzend zum Quellcode Diagramme gezeichnet. Dieses Dokument [14] beschreibt in Figure 3 das verwendete Inputsystem anhand mehrerer Dataflow-Diagramme. Die erstellten Diagramme dienen in dieser Arbeit allerdings lediglich zur Visualisierung. Die eigentliche Implementierung wurde von Hand durchgeführt. Auch in [16] wird in Figure 7 der grobe Datenfluss vom Eingabegerät, über die aufbereitenden Einheiten bis hin zur eigentlichen Applikation skizziert. Durch zusätzliche Diagramme ergeben sich zwar einige Verbesserungen, allerdings auch neue Probleme:

- Durch den zusätzlichen Aufwand für die Erstellung der zugehörigen Diagramme wird die Zeit bis zur Fertigstellung verlängert.
- Die Evaluation wird durch ein Diagramm etwas verbessert. Die Inputsysteme können als Diagramme auf dem Papier erstellt werden, bevor die eigentliche Implementierung stattfindet. Dadurch können bereits Alternativen diskutiert werden. Allerdings ist diese Diskussion rein theoretisch. Die skizzierten Konzepte können nicht sofort an einem bestehenden Produkt und verschiedenen Eingabegeräten getestet werden. Besonders die Evaluation von Multitouch-Inputsystemen gestaltet sich auf rein theoretischer Basis sehr schwierig.
- Die Kommunikation der Diagramme fällt sehr leicht, da sie für genau diesen Zweck konzipiert sind. Selbst dritte Personen können mit einer kurzen Erläuterung die grobe Funktionsweise des Systems verstehen.

Um die obigen Punkte zusammenzufassen, so entstehen einige Probleme bei der üblichen Herangehensweise an dieses Problem. Besonders transparente Kommunikation, Re-Design und Evaluation stellen oft eine Schwäche dar.

1.2 Fallbeispiel

Nehmen wir an, es soll eine Zoom-Funktion für ein Spiel implementiert werden. Besonders Strategiespiele bieten es oft an, in die Karte herein und aus ihr heraus zu zoomen, um mehr Details sehen zu können oder einen Überblick über das Schlachtfeld zu erhalten. Dem Entwickler stehen nun mehrere Möglichkeiten zur Verfügung, wie er mit verschiedenen Eingabegeräten diese Funktion implementieren könnte:

1. **Zoomen über das Scrollrad**

   Das ist die häufigste und auch die einfachste Implementierungsform. Eine Drehung des Mausrades wird von der Hardware durch einen positiven oder negativen Wert ausgedrückt, je nachdem in welche Richtung das Rad gedreht wurde. Dieser Wert kann dann direkt auf eine Bewegung in y-Richtung gemappt werden. Eine Einstellmöglichkeit ist hier der Gain-Faktor. Dieser beschreibt wieviel mit einer Einheit der Drehung gezoomt werde soll. Er kann durch einen Multiplikator ausgedrückt werden. Dieses System als Diagramm ist in Abbildung 6 abgebildet. Die Position des Scrollrades wird an ein Multiply-Modul weitergeleitet. Dieses Modul ist für die Anwendung des Gain-Faktors zuständig, indem es den erhaltenen Wert mit einer Zahl multipliziert. Diese Zahl erhält das Modul am SCALE-Pin in Form einer konstanten Fünf. Der multiplizierte Wert wird letztlich an die eigentliche Applikation (hier das Spiel) weitergeleitet, was durch das als „Output“ bezeichnete Modul symbolisiert wird.

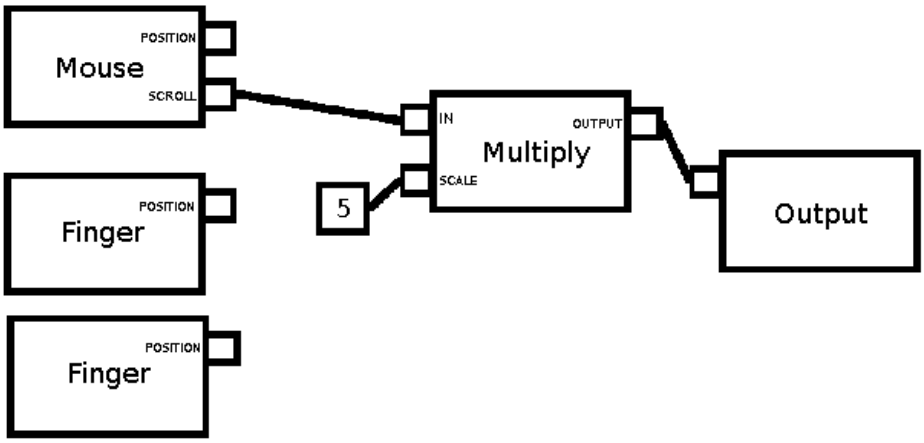

Abbildung 6. Dataflow-Diagramm der Lösung „Zoomen per Scrollrad“

2. **Zoomen über den Abstand zweier Finger**

Diese Methode ist etwas neuer und benötigt ein multitouch-fähiges Gerät. Der Zoom-Faktor wird dabei durch den Abstand zwischen den beiden Fingern bestimmt. Einen Beispielgraph zeigt Abbildung 7. Ähnlich wie in Methode 1 spielt auch hier wieder der Gain-Faktor eine Rolle. Eine zweite Designfrage ist, ob die Distanz absolut oder relativ bewertet werden soll. Wenn eine absolute Distanz verwendet wird, so wird der berechnete Abstand direkt auf einen Zoom-Faktor gemappt. Ein Abstand von 2cm bedeutet beispielsweise einen Zoom-Faktor von 5; 4cm bedeuten einen Faktor von 10. Der Nachteil dieser Methode ist, dass es zu „Sprüngen“ kommen kann, wenn der Benutzer den zweiten Finger aufsetzt. Der Zoom-Faktor wird dann direkt auf den dann ermittelten Wert gesetzt, egal wie hoch dieser zuvor war. So kann es passieren, dass ein Zoom-Faktor von 100 direkt auf 2 springt, wenn die Finger sehr dicht zusammen aufgesetzt werden. Diese Sprünge können den Benutzer verwirren. Bei einer relativen Implementierung ist die Distanz an sich nicht wichtig, sondern die Änderung dieser Distanz. Dazu ist ein Aufsetzen und Bewegen beider Finger erforderlich, damit sich eine Differenz in der Distanz ergibt. Werden die Finger so z.B. um 2cm außeinander geschoben, wird der Zoom-Faktor um 4 *erhöht*. Werden die Finger um 3cm zusammen geschoben, wird der Faktor um 6 *vermindert*, insofern das noch möglich ist. Letztere Implementierung entspricht der des iPod/iPhone.

Im Diagramm wurde die Methode mit einer *absoluten* Distanz *ohne* Gain-Faktor angewendet. Die Positionen der beiden Finger wird in den zwei mit „Finger“ bezeichneten Modulen ermittelt. Diese werden an das Distance-Modul weitergeleitet, welches den Abstand der beiden Punkte berechnet. Der Output dieses Moduls wird dann an die Hauptapplikation weitergeleitet.

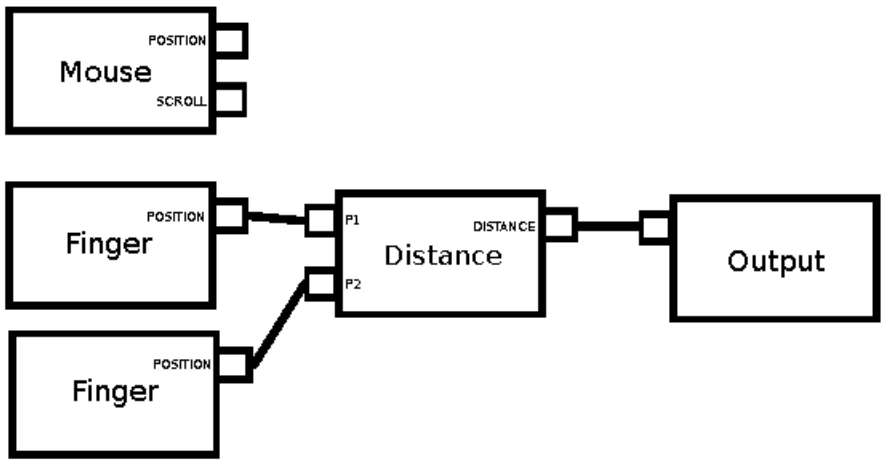

Abbildung 7. Dataflow-Diagramm der Lösung „Zoomen über den Abstand zweier Finger“

3. **Zoomen über Maus-Movement**

Diese Desginmöglichkeit ist eher unbekannt und wird in Abbildung 8 illustriert. Die Maustaste wird an einem bestimmten Punkt (1) gedrückt und gehalten. Danach wird die Maus an einem anderen Punkt (2) bewegt, und danach die Taste losgelassen. Der Zoom-Faktor wird durch die Differenz der beiden Punkte (1 und 2) bestimmt. Auch hier stellt sich wieder die Frage nach dem passenden Gain-Faktor.

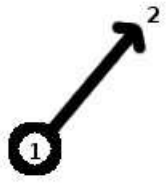

Abbildung 8. Funktionsweise des Zoomen per Maus-Movement

Man sieht direkt, dass für diese Funktion bereits mindestens drei verschiedene Lösungsansätze existieren. Welche die beste Lösung ist, kann vom Papier aus nicht entschieden werden. Weiterhin gibt es zu jeder Designmöglichkeit wiederum verschiedene Parameter wie Gain-Faktor, oder Bewertung als absolute oder relative Änderung, die gegenübergestellt und ausgewertet werden müssen. Diese Auswertung ist ohne einen direkten Test im verwendeten Programm nur schlecht möglich. Um alle Alternativen abzuwägen, müssten sie implementiert werden. Dabei wird nur eine der vielen Lösungen letztendlich verwendet, die restlichen Implementierungen sind danach nutzlos. Selbst dann ist es noch schwer, dritten Personen das Schema zu erklären, da selbst für Fachleute Quellcode nicht schnell verständlich ist. Für beispielsweise Kunden, die keine Erfahrung auf diesem Gebiet haben, ist Quellcode zum Verständnis ungeeignet. Hier wären zusätzliche Diagramme nötig, die die Kommunikation erleichtern. Diese erhöhen aber erneut den Arbeitsaufwand.

1.3 Ziele der Arbeit

Das erste Ziel ist die Implementierung eines grafischen Editors für Multitouch-Inputsysteme. Dabei spielen verschiedene Eigenschaften eine wichtige Rolle. Das zweite Ziel befasst sich mit der entsprechenden Multitouch-Hardware. Insgesamt ergeben sich also folgende Zielsetzungen:

- Software:
  - *Schnelle Entwicklung*
    Das System soll schnell erstellt und debuggt werden können. Dadurch wird der Prozess optimiert und Kosten gespart
  - *Einfaches Re-Design*
    In der Entwicklung kommt es oft vor, dass während des Entwicklungsprozesses entweder vom Softwareproduzent oder vom Abnehmer Designentscheidungen geändert werden. Diese Änderungen müssen schnell durchführbar sein. Auch dies hilft den Entwicklungsprozess zu vereinfachen und spart Kosten.
  - *Evaluation und Kalibration*
    Um verschiedene Designs gegenüberstellen zu können, müssen diese im Betrieb getestet werden können. Ein Ziel ist es daher auch, aus dem designten System schnell ein funktionierendes Inputsystem zu generieren. Dieses könnte dann in ein existierendes Produkt integriert werden und so die verschiedenen Alternativen direkt getestet werden. Auch das „fine tuning“, also das Einstellen spezifischer Parameter wie z.B. eines Multiplikators ist dadurch sehr schnell durchführbar.
  - *Transparente Kommunikation*

Die verschiedenen Alternativen müssen sehr transparent dargestellt werden, damit andere Personen, auch solche die nicht direkt am Projekt beteiligt sind, sofort das System überblicken und verstehen können. So können diese äußerst schnell in den Entwicklungsprozess eingebunden werden.

- Hardware:

  - *Aufbau und Inbetriebnahme des FTIR-Displays*
    Am DFKI existiert ein großes FTIR-Multitouch-Display. Dieses Display soll zumindest an dem damit verbundenen Computer mit meiner Software betrieben werden können. Dazu muss meine Software sowie die benötigten Elemente unter WindowsXP funktionieren. Die Verwendung in einem Client/Server-System, wie sie später in Kapitel 3 theoretisch beschrieben wird, ist hier noch nicht vorgesehen. Lediglich das Senden der Daten an localhost wird getestet. Die dazu nötigen Schritte sollen detailliert aufgezeichnet werden, damit die Inbetriebnahme später, eventuell auch an anderen Rechnern, reproduziert werden kann.

Der Hardware-Teil, sowie deren Aufbau und Inbetriebnahme wird hierbei in Kapitel 3 näher beschrieben. Konzept und Implementierung des Dataflow-Tools wird in den Kapiteln 4 und 5 erläutert.

2 Verwandte Arbeiten

2.1 Diamodl

Diamodl[4] [21] ist eine Kombination eines Editors für Domain- und Dialog-Modeling und basiert auf der *Diamodl language*. Es ist am ähnlichsten zu meiner Arbeit und wurde von *Hallvard Trætteberg* and der *Norwegian University of Science and Technology* entwickelt. Die verwendete Sprache ist Java.

Mit Diamodl werden anhand eines Datenflussplans Dialogsysteme entwickelt. Dazu werden unter anderem bausteinartige Repräsentationen von Steuerelementen (*Widgets*), Datensätzen (*Variables*) oder benutzerdefinierten Funktionen(*Computations*) über Datenleitungen miteinander verbunden. Die Daten können so von einem Steuerelement zum anderen fließen und zwischendurch modifiziert werden.

Wie bereits erwähnt, basiert das Programm auf zwei verschiedene Modellingverfahren, nämlich Domain- und Dialog-Modelling. Wobei sich Domain-Models wie UML lediglich mit dem strukturellen Aufbau der Software in Klassen, ihren Attributen und den Relationen zwischen diesen beschäftigt, so liegt die Aufgabe des Dialog-Modeling in der Beschreibung von Struktur und Verhalten des Benutzer-Interfaces.

Dieser Editor besitzt dazu zwei verschiedene Bereiche. In einem wird der domainorientierte Teil als Domain-Model modeliert, im anderen die entsprechenden Instanzen und das Datenflussidiagramm erstellt. Ein beispielhaftes Domain-Model zeigt Abbildung 9. In ihm existieren zwei Klassen, nämliche *UoD* und *Person*. UoD stellt das Klassenuniversum dar. Jede Klasse ist ein Teil dieses Universums. Die Klasse Person beschreibt eine natürliche Person und besitzt dazu Attribute wie den Namen oder die Homepage der Person. Letztlich wird im unteren Teil ein neuer Datentyp *URI* definiert, der äquivalent zur Java-Klasse java.net.URI gesetzt wird.

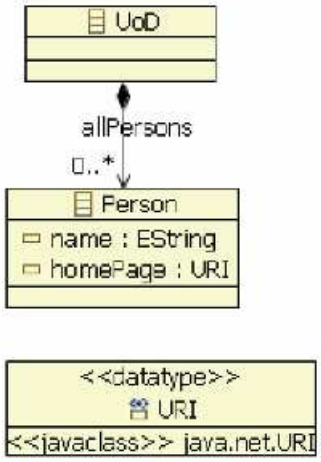

Abbildung 9. Domain-Model in diamodl

Die Module und das Konzept, welches durch ein Modul repräsentiert werden, sind in Diamodl und TouchFlow sehr verschieden. Mein Programm versucht Eingabegeräte und Berechnungen in einem Modul zu kapseln, sodass die Verbindung der Module ein größeres System ergeben. Einzelne Module haben nie eine grafische Repräsentation. In Diamodl können Module auch ein Benutzersteuerelement oder Variablen beinhalten. Modify-Module in Touchflow entsprechen Computations in Diamodl, da sie Funktionsberechnungen kapseln. Entsprechungen für Input- oder Output-Module gibt es nicht.

4. http://www.idi.ntnu.no/~hal/research/diamodl

Die verwendete *Diamodl language* ist ein Hybrid aus Datenfluss- und Zustandslogik. Für den datenflussorientierten Bereich der Sprache sind verschiedene Konzepte vorhanden. Zum einen gibt es ein Konzept *Variable*. In einer Variable werden einzelne Werte oder Objekte gespeichert. Auch eine Menge von Objekten (ähnlich einem Array) könne in einer Variable abgelegt werden. Ein weiteres Konzept ist die *Computation*. Diese enthält in ihrem Kern eine Funktion, die ähnlich wie eine Funktion aus Programmiersprachen agiert. Sie erhält eine Menge von Argumenten, und berechnet aus diesen einen Wert. Das besondere hierbei ist, dass die Computation automatisch den Wert neu berechnet, sobald sich eines der Argumente verändert. Das dritte Konzept sind die *Gates*. Gates werden ausschließlich in *Interactors* (s.u.) verwendet. Dort sind sie mit ein- und ausgehenden Kanten eines Interactors verbunden. Die Richtung des Gates gibt dabei an, ob es sich um ein Input- oder Outputgate handelt. In der Regel sind sie nur dafür zuständig, den Typ der einkommenden Daten zu prüfen und zu *casten*. Beim casten geht es darum, einen Typ in einen kompatiblen anderen Typen umzuwandeln. Beispielsweise könnte der String „123.45“ in einen double gecastet werden, da der Inhalt das Format einer double-Zahl trägt. Letztlich fehlen noch die *Connections*. Diese stellen die Verbindungsleitungen zwischen den verschiedenen Komponenten dar. Sie ermöglichen den Datenfluss von der einen zur anderen Komponente. Weiterhin können sie auch eine unäre Funktion enthalten. Mit ihr ist es möglich, den transportierten Datenfluss „on-the-fly“ zu manipulieren. All diese Elemente gehören zum domain-orientierten Teil des Editors. Alle Objekte werden mit Klassennamen typisiert. Connections können auf Attribute zugreifen um die nötigen Werte zu berechnen. Ein weiteres Konzept in diesem Bereich ist der *Interactor*. Ein Interactor beinhaltet ein *Widget*, welches ein Benutzersteuerelement darstellt. der Interactor managet die Kommunikation mit diesem Element und abstrahiert sie. Die Kommunikation mit dem darin enthaltenen Widget funktioniert über Input-Gates. Sie übernehmen die Typprüfung und nehmen die zur Darstellung nötigen Daten an. Intern werden diese vom Interactor verarbeitet und an das Widget weitergegeben. Auf der anderen Seite verarbeitet der Interactor auch die Output-Signale des inliegenden Widgets, und gibt diese über die Output-Gates nach außen hin weiter.

Die Zustandslogik beinhaltet andere Konzepte. Es gibt *Actions*, die mehrere Statements in JavaSkript beinhalten. Die Actions werden zu bestimmten Zeitpunkten ausgeführt. Es existieren bestimmte Events, die eine *Transition* von einem *State* zu einem anderen auslösen können, wie sie aus der Zustandslogik bekannt sind. Eine Transition ist mit einer Action verbunden, sodass bei dem Übergang spezieller Code ausgeführt wird. Z.B. kann die Auswahl eines bestimmten Namens in einer Liste ein Event auslösen. Auch States können Actions bei Eintritt oder Verlassen ausführen. Um zu bestimmen, ob eine Transition ausgelöst werden soll, gibt es *Conditions*. Diese prüfen bestimmte Eigenschaften und entscheiden anhand derer, ob eine Transition ausgeführt wird, oder nicht. Actions können nur auf Domain Data zugreifen, die im Kontext zu dem Element stehen, das die Aktion ausgelöst hat.

Im Bereich Dialog Modeling gibt es zwei weitere wichtige Konzepte, nämlich *Types* und *Functions*. *Types* beschreiben einen beschränkten Bereich von Werten, die einer Variablen zugewiesen werden können, die eine Function als Argument erhalten kann, oder die über eine Connection transportiert wird. Ein Type kann entweder ein Basistyp wie Integer, Boolean oder String sein, eine benutzerdefinierte Klasse, oder ein Array der vorher genannten Typen. Dabei können in Diamodl direkt keine neuen Klassen definiert werden. Sie müssen als Klasse im Java Classpath vorhanden sein, oder mit Ecore modelliert werden. Für letzteres stellt EMF und GMF spezielle Editoren zur verfügung.

Functions definieren das Verhalten von *Computations* (s.o.). Wie bereits erwähnt wurde, gibt es unäre Functions, die an Connections geknüpft werden können, um die durchfließenden Daten zu manipulieren. Auch Daten aus dem Domain Model können innerhalb einer Function verwendet werden. Zum Beispiel gibt es *property* Functions, die sich auf Attribute oder Relationen aus dem Domain Model beziehen können. Eine weitere Art ist die *script* Function. Sie ist in JavaScript geschrieben und kann auf die Domain Data der Objekte zugreifen, die ihr als Argument übergeben werden. Das Model reagiert automatisch auf Veränderungen in den Benutzerkomponenten. Es kann z.B. bei Auswahl eines Namens von einer Liste diese Person automatisch in einer anderen Komponente anzeigen lassen.

Diamodl wurde als Eclipse-Plugin geschrieben. Eine Verwendung des Editors ohne Eclipse ist daher nicht möglich. Es benutzt das Eclipse Modeling Framework (EMF) unter anderem um die Benutzeroberfläche darzustellen. Eine detallierte Beschreibung von EMF ist unter [5] verfügbar. Durch die Verwendung stehen viele nützliche Optionen, wie z.B. eine vorgefertigte grafische Umgebung, Serialisierung in XML-Format, sowie Import aus XML und Java-Files zur Verfügung. Weiteres wird am Ende dieses Abschnittes bei den Gemeinsamkeiten zu meiner Arbeit diskutiert.

Das System wird grafisch bearbeitet. Dazu gibt es eine Oberfläche, auf der verschiedene Elemente wie Interactors, Variables oder Computations platziert werden können. Per Klick können diese dann miteinander verbunden werden. Ein Ausschnitt eines Diagramms ist in Abbildung 10 zu sehen. Im Widget rechts oben kann eine URL eingegeben werden. Über den unteren Anschluss fließt dieser eingegebene Text weiter und wird in der ersten Computation (im Graph als Dreieck dargestellt) in eine URI umgewandelt. Die Computation danach ist eine Art „Tor“. An ihr ist ein Button angeschlossen. Sie sendet die Daten am Eingang erst weiter, wenn der Button geklickt wurde. Danach können die Daten weiter an das untere Widget fließen, welches daraufhin die entsprechende Webseite öffnet.

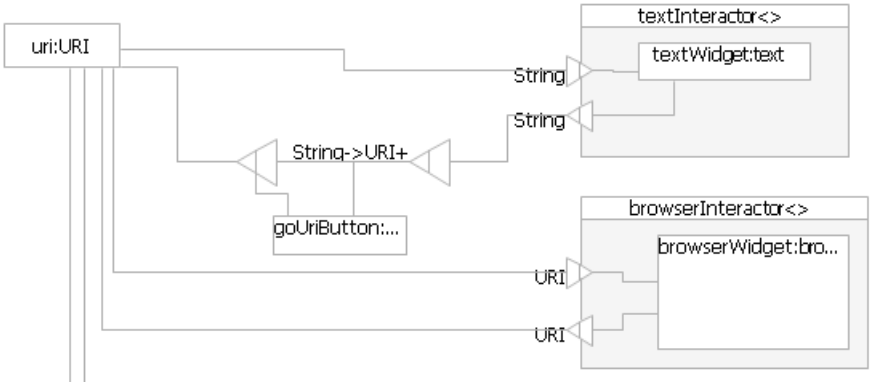

Abbildung 10. Dataflow-Diagramm in diamodl

Auch dieses Tool geht den Weg einer grafischen statt einer textuellen Implementierung, und ermöglicht so schnellen Aufbau und Redesign von Dialogsystemen. Das Re-Design ist durch die GUI sehr schnell realisierbar. Letztlich die Kommunikation ist durch die direkte grafische Visualisierung des Projektes immer sehr einfach.

Eine große Gemeinsamkeit des Tools mit TouchFlow auf Konzeptebene ist die Aufteilung in einen Editor und eine Laufzeitumgebung. Diamodl generiert keinen Code der ausgeführt werden kann, sondern interpretiert den modellierten Graphen. Das erspart die Entwicklung eines Compilers, die gerade für grafische Systeme sehr aufwändig werden kann. stattdessen werden vorhandene Ressourcen wie z.B. Java bzw. JavaScript und ihre Reflection API benutzt.

Auf konzeptueller Ebene ist der größte Unterschied, dass es mehr Typen gibt, die über die Datenleitung übertragen werden können. Hier gibt es Integer, Boolean, String und viele weitere Typen. Es können benutzerdefinierte Typen hinzugefügt werden. Mein Tool unterstützt nur double. In Diamodl können die Connections die durchfließenden Daten mit einer unären Funktion manipulieren. In TouchFlow muss dazu ein Modify-Modul zwischengeschaltet werden.

Diamodl besitzt keine direkte Anbindung an eine Fremdapplikation. Die komplette Applikation soll in Diamodl implementiert werden. Die Einspeisung der benutzerspezifischen Daten erfolgt hier durch eine Art Datenbank, nämlich der Domain Data. Diese enthält konkrete Ausprägungen der Klassen im Domain Model. Besitzt das Domain Model beispielsweise eine Klasse Person mit Attributen wie Name und Vorname, so beinhaltet die Domain Data spezielle Instanzen von dieser Klasse wie eine Person mit Namen „John Doe“. Diese Domain Data kann im Editor erstellt und von der Festplatte gelesen werden.

Dieses Programm berechnet die Daten neu, wenn sich das Objekt selbst, oder eines ihrer Properties verändert hat. Ähnlich ist es in meinem Tool, welches nur die Daten neu berechnet, wenn sich die Werte an den Input-Pins geändert haben. Diamodl hat durch die Verwendung von EMF bzw. Ecore weitere Funktionen zur Verfügung. So können z.B. die Graphen direkt als XML serialisiert und wieder geladen werden. Ecore-Modelle, die beispielsweise für die Spezifikation von Typen notwendig sind, können direkt aus Java-Code oder XML erstellt werden. Diamodl besitzt durch EMF eine Undo- und Redo-Funktion, die mein Programm zur Zeit noch nicht unterstützt.

Ein weiterer konzeptueller Unterschied zu meinem Projekt ist die Realisierung des Editors als Eclipse-Plugin. Eclipse ist auf vielen Rechnern von Entwicklern vorhanden. Weiterhin erspart EMF sehr viel Implementierungsarbeit. Die in Eclipse eingebauten Wizards, Views und Editoren können direkt verwendet werden. Allerdings verursacht dieses Konzept auch eine starke Abhängigkeit, denn ohne ein installiertes und funktionierendes Eclipse ist eine Verwendung dieser Software gar nicht möglich. Selbst die Runtime benötigt EMF. Hier ist meine Implementierung im Vorteil, denn sie ist völlig unabhängig von Eclipse. Sie benötigt lediglich wenige Libraries, die auch mit einem Installationspaket mitgeliefert werden könnten. Die Unabhängigkeit von Eclipse ist bei der Runtime besonders wichtig. Die Runtime von Diamodl benötigt ebenso EMF, wohingegen meine unabhängig ist. Inputsysteme, die mit meinem Tool erstellt werden, können somit ohne weitere Software an den Endkunden ausgeliefert werden. Auch die Installation fällt durch dieses Eclipse-basierte Design sehr schwer. Es müssen von Hand die nötigen Plugin-Dateien in den Eclipse-Programmordner kopiert werden. Werden diese nicht durch Eclipse erkannt, besteht keine Möglichkeit die Ursache herauszufinden, ob es an fehlenden Abhängigkeiten, einem nötigen Refresh von Eclipse oder einer Versionsinkompatibilität liegt. Ein weiterer Nachteil ist die Unübersichtlichkeit des Tools. Es gibt zwei verschiedene Editoren. Einen für die Erstellung eines Domain Models und einen für die Erstellung spezifischer Instanzen der Modelle. In einem dritten Fenster wird das Dialogsystem modelliert. Mein Tool besitzt nur eine grafische Anzeige und bleibt damit übersichtlich.

Diamodl besitzt lediglich die klassischen Eingabeelemente und Module, die für die Benutzung mit Maus und Tastatur optimiert sind. Spezielle Module für die Verwendung von Multitouch-Input sind nicht implementiert.

2.2 VVVV

VVVV ist ein grafischer Datenstromeditor, der besonders auf die Erzeugung und Manipulation von Video- und Grafikdatenströmen ausgerichtet ist. Wie auch bei Diamodl verläuft die Verarbeitung in Echtzeit. Das Tool wurde von „Meso Digital Media Systems Design“ entwickelt und ist für nicht-kommerzielle Nutzung kostenlos auf der VVVV-Homepage[5] zu finden. VVVV ist ausschließlich für Windows verfügbar.

Auch hier erfolgt die Bearbeitung eines Projekts komplett grafisch, wie Abbildung 11 zeigt. Jede Transformation oder jedes Input-Gerät wird als Node dargestellt, die verschiedene Ein- und Ausgänge hat. Diese Ausgänge werden grafisch durch kleine Vierecke am Rand der Nodes dargestellt. Nodes sind dabei wie Funktionen zu verstehen, wie sie z.B. in Java oder C++ existieren. die Input-Pins stellen die Parameter dar, die einer Funktion bei ihrem Aufruf übergeben werden müssen. Die Output-Pins sind dem Rückgabewert der Funktion gleichzusetzen. Dieses Konzept erinnert an Computations aus Diamodl. Die Nodes werden in einem Fenster, dem sogenannten „Patch“ platziert. Damit die Nodes miteinander kommunizieren können, werden sie per Maus mit Datenleitungen verbunden. Ein fertiges Diagramm sieht ähnlich wie ein elektronischer Schaltkreis aus. Abbildung 11, rechte Seite, zeigt ein solches System. In ihm wird das Bild, das

5. http://vvvv.org/

von einer Kamera bezogen wird (VideoIn) als Textur auf mehrere dreidimensionale Vierecke (sogenannte Quads) gelegt (Modul VideoTexture und Quad). Die Position und Ausrichtung wird durch das Transform-Modul hergestellt, welches verschiedene Positionen durch den LinearSpread erhält. Die Szene wird über das Camera-Modul in einem bestimmten Blickwinkel eingefangen, und durch das Renderer-Modul gerendert. Dieses Renderer-Modul öffnet ein seperates Fenster (linke Seite in Abbildung 11), um die berechneten Bilder auszugeben.

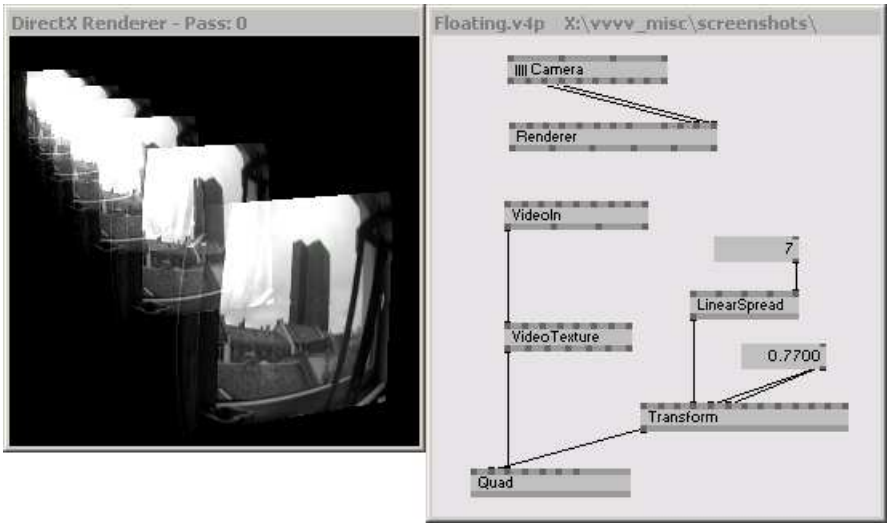

Abbildung 11.

Es gibt sehr viele verschiedene Typen von Nodes. Zum Beispiel gibt es IO-Boxen, die entweder einen Inputwert an ihrem Ausgang ausgeben oder einen vorhandenen Wert einer Leitung anzeigen können. IO-Boxen haben sehr viele verschiedene Konfigurationsmöglichkeiten. So können mit ihnen auch Slider, Buttons, oder Farbpaletten modeliert werden, um das kreirte System zu steuern. Die damit erstellten Benutzersteuerelemente können mit Widgets aus Diamodl verglichen werden. Das Aussehen der mit IO-Boxen erstellten Steuerelemente muss allerdings von Hand eingestellt werden, es gibt keine auswählbaren Vorlagen wie bei Widgets. Andere Nodes führen hingegen einfache Berechnungen mit den Werten durch, die ihnen zugeführt werden. Es gibt Nodes die von der Implementation einer simplen Addition zweier Inputs, über Vektorrechnungen bis hin zur Berechnung dreidimensionaler Wellenmuster und Audioanalyse reichen. Dabei sind die Module in verschiedene Kategorien eingeteilt. Zwei verschiedene Module können den gleichen Namen tragen, in sofern sie sich in verschiedenen Kategorien befinden. So kann beispielsweise eine Additions-Node für Zahlenwerte in der zugehörigen Kategorie, und eine andere Addition für Strings in einer anderen Kategorie existieren. Dieses Prinzip erinnert an das Overloading aus C++.

Letztlich gibt es auch Nodes, die ein berechnetes Bild ausgeben können. Dazu gibt es spezielle Nodes, die beim Start ein weiteres Fenster öffnen, in dem sie die ihnen zugeführten Informationen rendern können (Abbildung 11 links). Die Node „Renderer“ ist ein gutes Beispiel hierfür. An sie kann beispielsweise eine Text-Node angeschlossen werden. Dadurch wird auf dem Fenster des Renderers der in der Text-Node angegebene Text ausgegeben. Pins können auch von Hand mit Daten belegt werden. Zum Beispiel besitzt die Text-Node einen Pin, der die Schriftart des zu rendernden Textes angibt. Per Klick auf diesen Pin öffnet sich ein Dialog, über den dann die Schriftart ausgewählt werden kann. Um das System im Betrieb beobachten zu können, gibt es einen sogenannten „Inspector“. Sobald er gestartet ist, zeigt er Informationen über die momentan ausgewählte Node an. Dazu gehören z.B. die Daten auf den Input- und Output-Pins, Name des Moduls und Typ der angeschlossenen Leitung. Auch zur Laufzeit können im Inspektor die Werte der Pins verändert werden.

Es gibt eine weitere Klasse von Pins, die Config-Pins. Sie werden *nicht* in der grafischen Repräsentation der Node angezeigt, weil sie nicht für die Verbindung mit anderen Nodes vorgesehen sind. Sie können ausschließlich im Inspektor verändert werden und befinden sich daher dort immer an oberster Stelle. Sie verhalten sich also wie Pins, denen über ein Const-Modul bestimmte, konstante Werte übergeben werden, die sich zur Laufzeit nicht ändern lassen.

Es gibt viele verschiedenen Typen, die von einem Pin angenomment werden können. Neben den einfachen Typen wie String können z.B. auch Farben, DirectX-Layers oder Meshes von einem Pin aufgenommen und weitergegeben werden. Auch Nodes können mit einem Pin übertragen werden. Die angeschlossene Node beeinflusst dann die Arbeitsweise des damit verbundenen Nodes. So können z.B. Transformationen angewandt, oder die Render- und Samplerstates festgelegt werden.

Wie bereits angedeutet gibt es nicht nur Datenleitungen, die einen einzigen Wert übertragen können. Es können mit einer Leitung beliebig viele Daten parallel versendet werden. Diese speziellen Leitungen werden *Spreads* genannt. Wie diese Spreads von einem damit verbundenen Modul verarbeitet werden, hängt dabei von dem Baustein selbst ab. Beispielsweise kann eine Node, die einen Punkt zeichnet, die an den Positions-Pins angeschlossenen Spreads Kombinieren und somit Koordinaten-Paare erhalten. Dabei wird immer das n-te Element des einen Spreads mit dem n-ten Element des anderen Spreads verknüpft. Liegen an dem Pin für die x-Position beispielsweise **1 und 2** sowie am y-Pin **3 und 4** an, so sind die entstehenden Koordinaten **(1;3)** und **(2;4)**. Dadurch kann es an jede dieser Koordinaten einen Punkt zeichnen. Besitzt ein Anschluss weniger Dimensionen als der andere, so wird für die fehlenden Dimensionen wieder von vorne begonnen. Nehmen wir an als x-Koordinate werden **1, 2, 3** übergeben und als y-Koordinate **4, 5** . Um die ersten beiden Koordinatenpaare zu bilden, wird jeweils bei beiden Spreads das erste bzw. das zweite Element ausgewählt. Es ergibt sich also **(1;4)** und **(2;5)**. Für das dritte Element des x-Spreads fehlt nun das entsprechende dritte Element bei y. Daher wird dort wieder mit dem ersten Element begonnen. Die nächste Koordinate ist also **(3;4)**. Anhand eines Spreads kann z.B. sehr schnell ein Gittermuster erzeugt werden, indem ein Spread alle x-Koordinaten und ein anderer alle y-Koordinaten bereithält. Ein weiteres Modul (genannt *Cross*) bildet daraus alle möglichen Kombinationen, und kann diese danach als Position an ein Rechteck-Modul weitergeben. Dieses zeichnet dann die Rechtecke. Nicht alle Pins unterstützen den Anschluss von Spreads. Die Kompatibilität wird im Inspektor angezeigt.

Spezielle Nodes dürfen nur einmal auf der Arbeitsfläche existieren. Diese werden *Singleton* genannt. Beispielsweise die *Mouse-* oder die *Keyboard-* Node, die global die Mausposition und Tastaturanschläge erkennen können, dürfen nur einmal existieren. VVVV unterstützt auch die Verwendung einer Server/Client-Architektur, bei der die Clients als Render-Computer agieren. Diese Fähigkeit wird als *Boygrouping* bezeichnet. Dazu gibt es spezielle Nodes, die sich durch ihre blaue Farbe kennzeichnen. Diese Nodes existieren auch auf den Client-PCs. Wenn man also ein blaues Render-Modul einsetzt, wird auf jedem Client-PC das entsprechende Bild gerendert. Die Verbindungsleitungen zwischen den normalen (grauen) und den blauen Nodes haben dabei eine besondere Funktion. Bei diesen als *Bridges* bezeichneten Leitungen müssen Daten vom Server über das Netzwerk an die Clients übertragen werden. In der zentralen Boygroup-Node kann die gesamte Anzahl, sowie die Anzahl der momentan aktiven (also übertragenden) Bridges angezeigt werden. Damit nicht alle Clients die gleichen Daten rendern, besitzt jeder Client eine Client-ID. Anhand derer können jedem Client bestimmte Positionen oder Datenströme zugeführt werden.

Das Projekt ist sehr ähnlich zu meiner Arbeit, da es auch die Bearbeitung von Daten in einem Datenflussdiagramm realisiert. Auch der Aufbau mit Modulen, Anschlusspins und Datentypen gleicht dem meinen. Ebenso erfüllt dieses Projekt die von mir vorgegebenen Ziele in seinem Bereich. Das grafische, mausgesteuerte Interface ermöglicht eine schnelle Erstellung und Re-Design des Systems. Dadurch, dass das System direkt gestartet werden kann, und auch während dem Betrieb die Parameter kalibriert werden können, ist auch eine schnelle Evaluation und Kalibration möglich. Letztlich ist der Graph gut zu kommunizieren, da der Datenfluss und die vorgenommenen Berechnungen und Transformationen sehr schnell erkannt werden können.

Der größte Unterschied zu meinem Tool ist jener, dass VVVV auch zur Laufzeit vorgenommene Änderungen direkt umsetzt und so das Resultat direkt beobachtet werden kann. Über den Inspektor können Werte manipuliert werden. Die Auswirkung ist danach direkt sichtbar und

kann bewertet werden. Ebenso kann dieses Programm nicht nur Input-Daten empfangen oder verarbeiten, sondern auch externe Geräte über Protokolle steuern. Es gibt also auch eine, wenn auch beschränkte, Output-Funktionalität. Mein Tool hat momentan noch keine solche Fähigkeit. Diese soll aber eventuell in Zukunft nachgerüstet werden, wie Kapitel 6.2 beschreibt. VVVV ist sehr flexibel, was die bereits erstellten Nodes angeht. So besitzt die Implementierung keine Konstruktorparameter, die bei der Erstellung der Module angegeben werden müssen und danach nicht mehr geändert werden können, wie es in TouchFlow vorgesehen ist. Es werden lediglich Standart-Werte verwendet. Änderungen können danach jederzeit im Inspektor vorgenommen werden. Es ist möglich, mehrere Module gleichzeitig auszuwählen, und per Inspector die Parameter von ihnen in einem Schritt zu ändern (*Multinodural View*). VVVV unterstützt durch seine Boygrouping-Funktion die Verarbeitung und Ausgabe der Daten in einem verteilten System, das über ein Netzwerk verbunden ist. TouchFlow besitzt diese Funktion nicht, alle benutzten Berechnungen können nur lokal ausgeführt werden. In Touchflow können nur Module miteinander kommunizieren, die durch Datenleitungen miteinander verbunden sind. In diesem Tool ist das nicht unbedingt der Fall. Beispielsweise werden im Graph von Abbildung 11 im unteren Teil Objekte erstellt. Diese existieren dann in der 3D-Szene. Das Camera-Objekt im oberen Teil bezieht sich auf diese Szene, obwohl keine direkte Verbindung zwischen den Quads und der Kamera bestehen. Diese Methode macht den Graph unübersichtlich, da der Ursprung der Daten und ihr Fluss bis zur Ausgabe nicht lückenlos verfolgt werden kann. Es wird standartmäßig nur ein Inspector geöffnet, es können jedoch beliebig viele weitere geöffnet werden. Normalerweise zeigt jeder Inspektor die Daten des aktuell ausgewählten Moduls an. Man kann sie jedoch auch an ein bestimmtes Modul binden.

Hier ist meine Implementierung im Vorteil. Sie zeigt direkt einen Inspector für alle verwendeten Module an. Weiterhin besitzen meine Inspektoren ein grafisches Feedback, falls sich Daten geändert haben. Dieses Tool unterstützt das nur im speziellen Debug-Mode. Der VVVV-Inspektor zeigt bei Datenleitungen, die mehr als einen Wert transportieren, standartmäßig nicht alle Werte an. Man muss diese Option erst per Klick aktivieren. Meine Implementierung zeigt immer alle transportierten Werte an. VVVV verbietet es, Schleifen im einem Diagramm zu erstellen. Nur falls in der Schleife Module existieren, dessen Output nicht vom Input abhängen, sind diese zulässig. Mein Programm kann problemlos mit allen Formen von Schleifen umgehen. Ein weiterer Vorteil meiner Implementierung ist die Plattformunabhängigkeit. Dadurch, dass es in Java erstellt wurde, kann es auf jedem System ausgeführt werden, für das eine JVM existiert. Ebenso besitzt meine Arbeit Module, die speziell für die Bedienung von Multitouch-Geräten zugeschnitten sind. VVVV besitzt diese nicht. Die Erstellung von neuen Modulen löst TouchFlow besser. Der Programmierer kann direkt von meiner abstrakten Modul-Klasse erben und muss nur noch die abstrakten Funktionen ausfüllen. Die neuen Module können dadurch auch in ein jar-Package mit meiner Implementierung gepackt werden. In VVVV sind nur die Interfaces zur Erstellung neuer Nodes offen. Daher muss zur Entwicklung dieses COM-Interface implementiert und das Ergebnis in eine DLL-Datei exportiert werden. Näheres zum COM-Interface wird in [3] beschrieben. Die DLL kann danach in das Programm importiert werden. Dieses Programm erstellt Pins und ihre Bezeichnungen automatisch. Sobald ein Bezeichner auf Top-Level deklariert wird, wird dazu ein Pin mit dem Namen der Variable erstellt. Das erspart eine explizite Deklaration der Pins. Dadurch ist es aber nicht mehr möglich, Top-Level Variablen zu erstellen, die nicht als Pin auftauchen sollen. Mein Tool zeigt nur Pins als solche an, die explizit definiert wurden. Mit der deklaration eines Pins wird implizit auch eine entsprechende Variable erstellt, die Werte für den Pin aufnehmen kann. Der Aufwand ist somit bei beiden Programmen gleich, nur meine Lösung erlaubt beliebige Variablen auf Top-Level.

2.3 MAX/MSP

MSP konzentriert sich mehr auf den Audio-Bereich. Es ist ein modular aufgebauter Software-Synthesizer. Das ist das virtuelle Pendant zu einem Synthesizer wie er oft in der Musikproduktion verwendet wird. Dementsprechend gibt es hier Module wie Oszillatoren, Filter oder Verzögerer die bereits vorimplementiert sind. Ähnlich wie in meiner Arbeit laufen auch hier die Module simultan. Das ist bei einem Synthesizer besonders wichtig, da schon die kleinste Verzögerung direkt zu einem hörbaren Ergebnis führt. Es ist ein kommerzielles Tool das von Cycling 74[6] angeboten wird. Anders als VVVV ist diese Software auch für Mac erhältlich, nicht aber für Linux.

Die grafische Oberfläche ist in Abbildung 12 zu finden. Wie üblich werden Module, hier *Objekte* genannt, auf der Oberfläche platziert. Die Anschlüsse dieser Module, hier dargestellt durch einen schwarzen Abschnitt im Rahmen des Objekts, werden miteinander verbunden, um eine Interaktion zu gewährleisten. In diesem Beispiel wird eine gedrückte Taste eines MIDI-Keyboards erfasst (Paket *MIDI Stuff*). Aus dieser Taste wird danach die passende Frequenz berechnet (Objekt *mtof*) und mit einem Frequenzgenerator diese Frequenz generiert (Objekt *cycle*). Im Paket *Output Stuff* wird die Frequenz an das Ausgabegerät weitergeleitet. Bestimmte Objekte können Benutzersteuerelemente tragen. Dadurch können im Betrieb Parameter wie z.B. Lautstärke geändert werden.

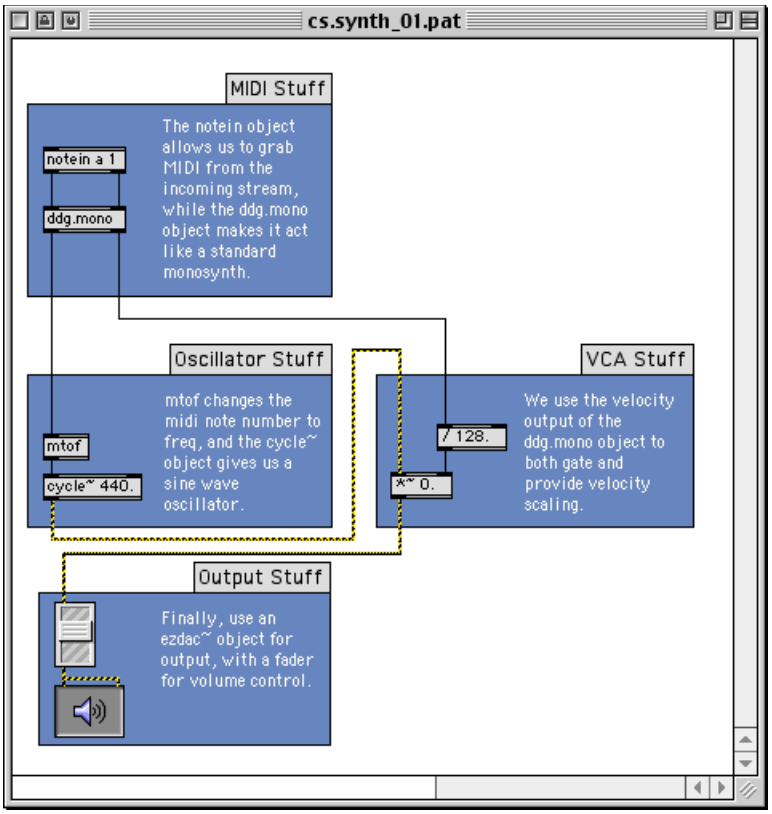

Abbildung 12. Diagramm in Max/MSP

Es können auch neue Module implementiert werden. Es gibt zum einen sogenannte „Externals“, die direkt in C geschrieben werden und dann importiert werden können. Man kann ebenso eine Kombination mehrerer solcher „Externals“ zusammenschalten und daraus ein neues Modul erstellen. Diese werden „Abstractions“ genannt. Auch dieses Tool erlaubt die Änderung des Synthesizer-Systems in Echtzeit, sodass im laufenden Betrieb Änderungen und das „fine tuning“ vorgenommen werden können. Auch ist die Einheit der Zeitsteuerung variabel. So kann z.B. nicht

6. http://cycling74.com/products/maxmspjitter/

nur angegeben werden, dass ein Modul alle x Taktzyklen arbeiten soll. Es können z.B. auch Musik-Beats als Einheit verwendet werden, sodass beispielsweise alle 3 Beats das Modul ausgelöst wird und einen weiteren Ton hineinmischt. Ein sehr wichtiges Feature ist allerdings die Möglichkeit, eigene Bedienelemente für jedes Modul zu erstellen. So kann z.B. für einen Tongenerator ein Slider-Element erstellt werden, mit dem zur Laufzeit Lautstärke und Tonhöhe verändert werden können. In Abbildung 12 besitzt das Ausgabegerät im Paket *Output Stuff* ein solches Steuerelement. Selbst dieses Interface kann zur Laufzeit verändert werden, falls z.B. neue Optionen möglich werden. Die Steuerelemente sind für die Benutzung mit Maus- und Tastatur ausgelegt. Spezielle Unterstützung oder Module für Multitouch-Geräte, wie sie mein Programm anbietet, existieren nicht.

Eine weitere Funktionalität, welches mein Programm nicht besitzt ist die Erstellung neuer Module aus einer Kombination bereits vorhandener Module, wie es die *Abstractions* sind. Dadurch können komplexe Verschaltungen, die öfter benötigt werden, in ein einzelnes Modul gekapselt und so sehr einfach wieder verwendet werden. Die Möglichkeit, mehrere Module in einem Package zu gruppieren, macht das Graphlayout übersichtlicher. Für den professionellen Einsatz ist auch die Fähigkeit, Daten über das Netzwerk zu senden sehr wichtig. So kann das System z.B. für einen Sound-Check auch von Client-PCs gesteuert werden und nicht nur von der zentralen Station aus.

Ein weiterer Unterschied ist die Kommunikation der Module und die Zeitpunkte, zu denen eine Berechnung (ein Taktzyklus) stattfindet. MSP-Objekte kommunizieren ständig miteinander, nicht nur wenn sich Eingabedaten ändern. Das ist nötig, um die entsprechende Sampling-Rate des Audio-Files zu erreichen. Dementsprechend oft müssen auch Neuberechnungen durchgeführt werden. Dadurch entsteht eine sehr hohe CPU-Last. In der Dokumentation wird ausdrücklich davor gewarnt, dass rechenintensive Module für eine Überlastung sorgen können. Neben einer erhöhten Reaktionszeit können dadurch auch Artefakte in der Sounderzeugung entstehen. Solche Probleme sind in meiner Implementierung bisher nicht vorhanden. Das liegt daran, dass zum einen die Module nicht rechenintensiv sind. Zum anderen werden nur dann Ausgabewerte neu berechnet, wenn neue Daten anliegen. MSP hat sein eigenes Speicherformat, welches an sich für den Menschen nicht lesbar oder editierbar ist. Mein Programm benutzt ein XML-File, welches schnell verstanden, erweitert oder auch von Anfang an neu geschrieben werden kann. Auch die Plattformkompatibilität ist bei TouchFlow besser, denn es ist in Java implementiert. Das erleichtert auch die Implementierung neuer Module.

3 Verwendete Hardware

Um Multitouch-Eingaben vorzunehmen und die entsprechenden Module zu testen, war entsprechende Hardware nötig. Dazu wurde ein experimentelles Display benutzt, das auf dem Prinzip der Totalreflektion (engl. „frustrated total internal reflection“) basiert [10]. Dieses spezielle Display beruht auf einer Masterarbeit [15] von H. Peters, die einige Verbesserungen in Bezug auf Tracking Performance, Ambient Light Filtration sowie Kalibration eines FTIR-Displays beschreibt. Abbildung 13 zeigt ein Foto des Versuchsaufbaus. Es sind in einem Rahmen zwei Plexiglasscheiben befestigt, zwischen denen sich eine Silikonschicht befindet (1). Im Rahmen befinden sich IR-Dioden, die die Silikonschicht beleuchten. Diese Dioden werden durch ein externes Netzteil betrieben. Das Bild wird durch einen Beamer mit extremem Weitwinkel (2) auf die Rückwand der Scheibe projiziert. Zur Erfassung der Touch-Punkte dient eine auf einem Stativ montierte IR-Kamera (3) (Firefly MV von Point Grey Research). Damit möglichst viele fremde Lichtquellen abgeschirmt werden, wird vor der Kamera ein IR-Lichtfilter befestigt.

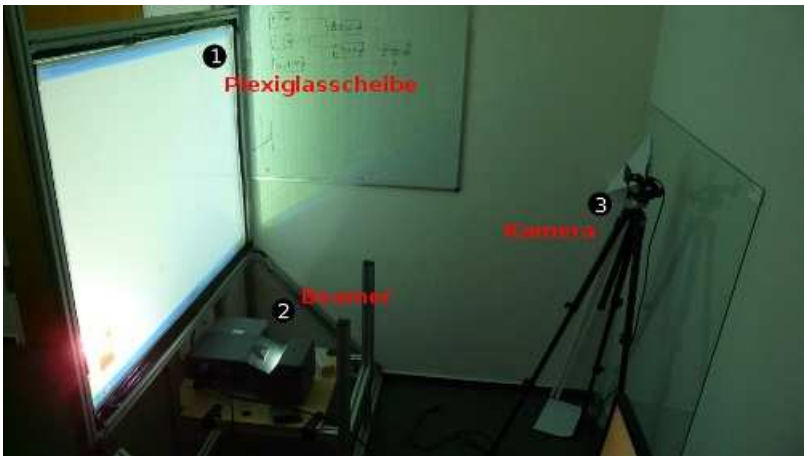

Abbildung 13. Der Versuchsaufbau

Um die von der Kamera gelieferten Bilder in Berührpunkte umzuwandeln, wird spezielle Software benötigt. Dazu verwende ich „Community Core Vision“ (CCV) der NUI Group[7]. Abbildung 14 zeigt das Hauptfenster dieses Programms. Im linken Teil sind das eingehende Kamerabild („Source Image“), die aufbereiteten Bilder aus diesen Rohdaten, sowie die erkannten Objekte („Tracked Image“) abgebildet. In diesem Bereich können durch die Regler verschiedene Kalibrierungen vorgenommen werden, um die Erkennungsrate zu verbessern. In der Praxis habe ich nicht nach einem speziellen System diese Werte geändert, sondern die Wirkung der verschiedenen Regler auf das System beobachtet und dementsprechend reagiert. Im rechten oberen Teil befinden sich einige Steuerknöpfe. Mit ihnen kann z.B. eine Kalibration oder das Senden der gewonnenen Daten aktiviert werden, wie es weiter unten beschrieben wird. In der rechten unteren Ecke befinden sich Statusinformationen wie FPS oder Auflösung der verwendeten Kamera.

Die Installation der nötigen Treiber und das stabile Betreiben dieser Software erfordert einiges an Aufwand, da mit falschen Einstellungen das Programm ohne jegliche Meldung abstürzt. Details hierzu sind in der Installationsanleitung im Anhang in Kapitel 7.1 zu finden. Nachdem die Kamera installiert wurde, muss die Software über die grafische Oberfläche kalibriert werden. Die Aufgabe der Kalibration ist es, CCV den Bildbereich des Beamers mitzuteilen, sodass der sensitive Bereich auf diesen Ausschnitt beschränkt wird. CCV projeziert dazu

7. http://www.nuigroup.com

nacheinander vier Kreise in die vier Eckpunkte des Beamer-Bildes. Diese müssen dabei mit dem Finger berührt werden, sodass die entsprechenden Touch-Punkte von der Software erkannt werden. Danach kann über die Kontrollleiste das Senden der gewonnenen Informationen über das TUIO-Protokoll aktiviert werden.

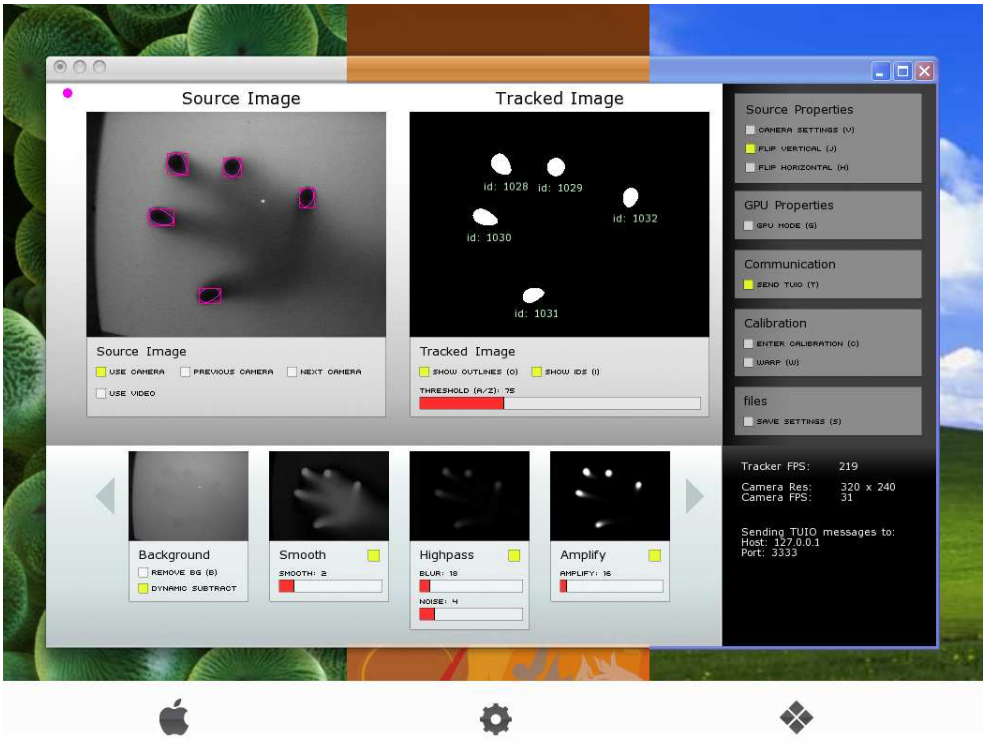

Abbildung 14. Hauptfenster von CCV

Die Kommunikation mit TouchFlow findet über das TUIO Protokoll [7] statt. Dieses Protokoll nutzt das Open Sound Control Protokoll (OSC) [13], um die Nachrichten zu codieren und als binäre Daten zu verschicken. Dadurch können TUIO-Messages über jeden Kanal verschickt werden, der durch eine OSC-Implementierung unterstützt wird. Das TUIO-Protokoll an sich besteht aus drei verschiedenen Arten von Nachrichten. *SET* Nachrichten werden benutzt, um den Zustand eines Objektes wie z.B. Position oder Ausrichtung zu übermitteln. *ALIVE* Nachrichten beschreiben, welche Objekte zum jetzigen Zeitpunkt existieren. Es gibt keine expliziten Nachrichten wie ADD oder REMOVE, um Fehler bei einem Paketverlust zu vermeiden. Um eine Session abzuschließen, wird eine *FSEQ* Message gesendet. sie beinhaltet eine unique ID, um die aktuelle Session zu identifizieren. Ein Message-Bundle, das die Statusveränderung eines Cursors angibt, hätte also dieses Format:

```
/tuio/2Dcur source application@address
/tuio/2Dcur alive s_id0 ...  s_idN
/tuio/2Dcur set s_id x_pos y_pos x_vel y_vel m_accel
/tuio/2Dcur fseq f_id
```

1. In der ersten Zeile wird die sendende Anwendung *application* sowie deren Adresse *address* übermittelt.
2. Darauf folgend werden die momentan aktiven Objekte übermittelt.

3. Im dritten Schritt folgt schließlich die eigentliche Status-Übermittlung. Die aktuelle Position für den Cursor mit der ID *s_id* ist nun (*x_pos*,*y_pos*). *x_vel* und *y_vel* geben die Beschleunigung in x- und y-Richtung an. *m_accel* beschreibt die Motion Acceleration. Sie berechnet sich aus der normalisierten Distanz zwischen den zwei letzten Punkten dividiert durch die Zeit, die zwischen der Aufnahme dieser beiden Punkte verstrichen ist.
4. Abschließend gibt die *fseq*-Nachricht die aktuelle Session id an.

Dabei sendet CCV im Versuchsaufbau die Daten an localhost. Es ist allerdings auch möglich, die Daten an jeden beliebigen anderen Rechner im Netzwerk zu schicken, und darauf die von mir entwickelte GUI und Laufzeitumgebung einzusetzen. So könnte beispielsweise die Verarbeitung der Kameradaten per CCV auf einen seperaten Server ausgelagert werden.

Zum Empfang der gesendeten Daten wird ein Client benötigt. Für die von mir verwendete Programmiersprache Java existiert bereits eine Implementierung[8]. Die zentrale Klasse dieser Implementierung ist *TuioClient*. Bei ihr können Listener registriert werden, die die TUIO-Nachrichten empfangen. Sie besitzt eine connect-Methode, mit der sich eine Instanz der Klasse für den Empfang von Nachrichten initialisieren lässt. Ab diesem Zeitpunkt empfangen alle registrierten Listener die gesendeten TUIO-Messages. Um mit TUIO zu kommunizieren, muss ein Listener-Interface *TuioListener* implementiert werden. Dieses besitzt Funktionen wie addTuioCursor, updateTuioCursor und removeTuioCursor, die aufgerufen werden sobald ein Cursor neu auf der Multitouch-Oberfläche erkannt, bewegt, oder entfernt wird. Um den Zustand eines Cursors zu kapseln, gibt es die Klasse TuioCursor. Sie enthält Informationen wie z.B. id und Position des erkannten Objektes. Jeder der oben genannten Funktionsaufrufe des implementierten TuioListener-Interfaces erhält eine Instanz von TuioCursor als Parameter. Anhand dessen kann im Funktionsrumpf der Zustand des veränderten Cursors abgefragt werden.

8. http://www.tuio.org/?software

4 Konzept

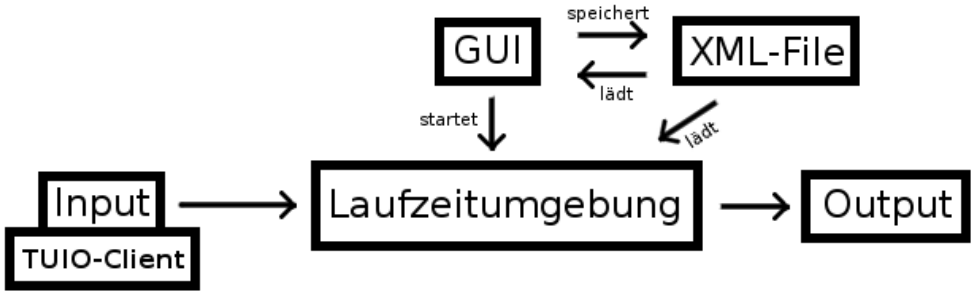

Abbildung 15. Gesamtkonzept der Implementierung

Eine grobe Übersicht zeigt Abbildung 15. Das Herzstück des Tools ist die *GUI* (Abbildung 16), in der die Inputsysteme als Diagramm modelliert werden. Die Diagramme ähneln dabei stark einem elektronischen Schaltkreis. Es gibt verschiedene Module die Daten erhalten, verarbeiten und ausgeben können. Um die Module miteinander verbinden zu können, besitzen diese Pins. Bestimmte Pins können über eine Datenleitung verbunden werden, sodass die Daten vom einen zum anderen Pin „fließen“. Im Screenshot (Abbildung 16) werden Daten von dem Mouse-Modul anhand der aktuellen Mausposition sowie vom ConstPoint-Modul durch eine eingestellten Konstante generiert. Diese fließen weiter in ein anderes Modul, dass die Distanz dieser Punkte berechnet und als Datenstrom ausgibt. Das Print-Modul nimmt letztlich diesen Strom auf und gibt ihn über die Konsole aus.

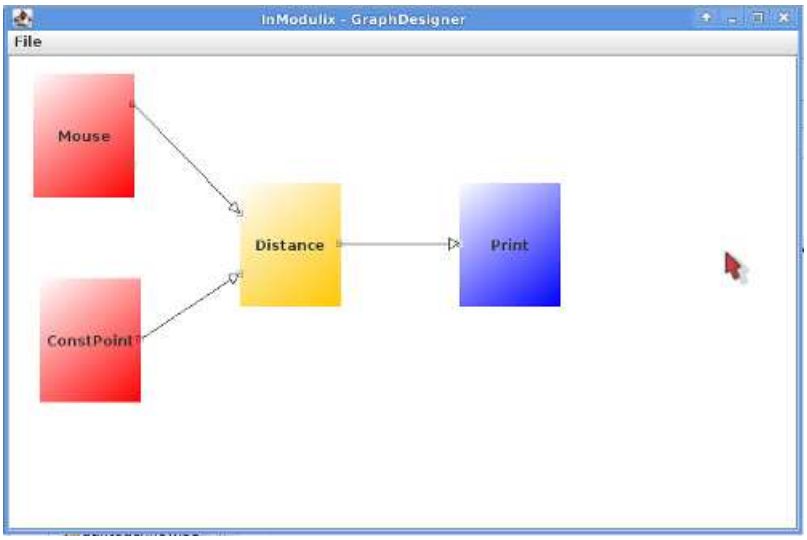

Abbildung 16. Diagramm in der GUI

Die so entworfenen Graphen werden durch die *Laufzeitumgebung* interpretiert. Sie erstellt das eigentliche Inputsystem und führt die Berechnungen der Module durch. Sie bezieht dazu die Input-Daten z.B. aus einem Eingabegerät oder dem unterstützten FTIR-Display, verarbeitet diese Daten anhand des spezifizierten Input-Graphen, und gibt sie an eine Hauptapplikation weiter, sendet sie über ein Protokoll oder gibt sie auf der Konsole aus. Die Serialisierung eines Graphen erfolgt im XML-Format. Die GUI kann diese serialisierten Graphen laden oder aus einem gezeichneten Graphen erzeugen. Das Laufzeitsystem kann nur XML laden.

Die Anbindung an eine andere Applikation kann entweder über das Observer-Interface, oder

über ein eigenes Protokoll über ein Output-Modul erfolgen. Um die erste Methode zu Nutzen, muss die Applikation in Java geschrieben sein. Danach können Klassen, die das Observer-Interface implementieren, sich bei dem entsprechenden Modul registrieren. Von da an werden sie per Funktionsaufruf über jede Änderung informiert. Die zweite Methode ist mit jeder Programmiersprache nutzbar. Sie sieht eine Kommunikation über ein Protokoll vor. Es können Output-Module geschrieben werden, die den ankommenden Datenstrom über dieses Protokoll versenden, sodass die Applikation diese Daten erhalten kann. Detallierte Informationen über die Anbindung an andere Applikationen enthält Kapitel 5.5.

Module

Module entsprechen, um das Modell eines elektronischen Schaltkreises wieder aufzunehmen, einem elektronischen Baustein. Das UML-Diagramm in Abbildung 17 zeigt die Klassenstruktur des Modulsystems.

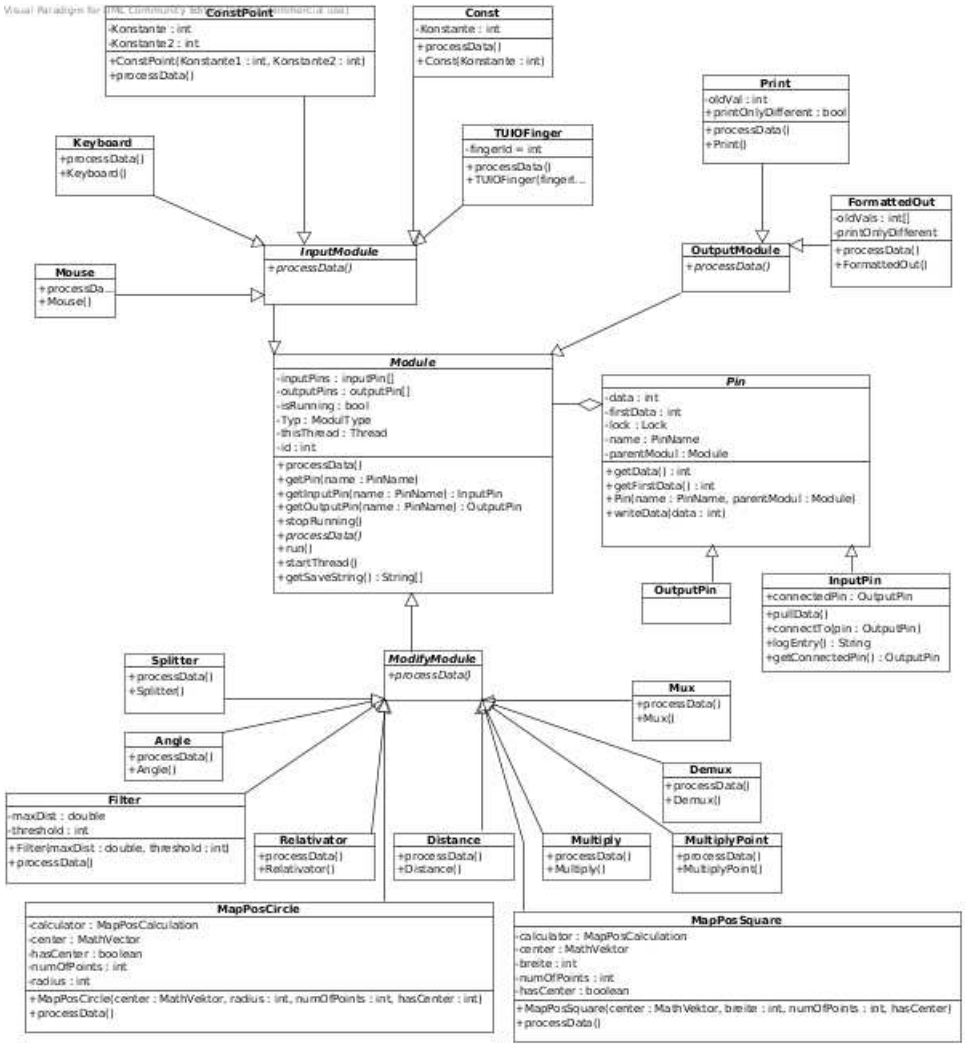

Abbildung 17. UML-Diagramm des Modulsystems

Sie erhalten Daten über Eingangspins, verarbeiten diese auf eine modulspezifische Art und Weise, und schreiben die gewonnenen Daten dann auf die Ausgangspins. Zur Zeit sind 3 verschiedene Modulklassen implementiert.

Modify-Module. besitzen sowohl Eingangs- als auch Ausgangspins und dienen zur Verarbeitung vorhandener Daten wie oben beschrieben.

Input-Module. repräsentieren in der Regel Eingabegeräte. Sie besitzen keine Eingangspins und schreiben die vom Input-Device gewonnenen Daten auf die Output-Pins.

Output-Module. sind unter anderem zu Debug-Zwecken nützlich. Sie besitzen keine Output-Pins, da sie die berechneten Werte lediglich auf der Konsole o.ä. ausgeben. Zur Zeit existieren Module, die entweder die Daten eines Pins unformatiert, oder die Daten von bis zu 4 Pins formatiert ausgeben.

Jede dieser drei Klassen besitzt verschiedene Implementierungen, wie Sektion 5.6 zeigt. Es können neue Module in Java geschrieben und hinzugefügt werden. Dazu muss entweder die abstrakte Klasse „Module“, oder eine der drei Modulklassen Input-, Modify- oder Output-Module implementiert werden. Details werden in Abschnitt 5.5 erläutert.

Pins

Pins sind die Schnittstellen der Module, um die nötigen Daten zu beziehen und weiterzugeben. Dabei werden grundsätzlich 2 verschiedene Typen unterschieden.

Eingabepins (Input-Pins). deren Aufgabe es ist die zur Berechnung nötigen Daten bereitzustellen

Ausgabepins (Output-Pins). die die berechneten Daten aufnehmen und zur weitergabe bereithalten.

Unabhängig davon unterscheiden sich die Pins durch den Datentyp, den sie bereithalten.

1D- oder Skalar-Pins. beinhalten nur einen einzigen Wert

2D- oder Punkt-Pins. beinhalten ein Tupel aus zwei Werten

Letztere sind hervorragend dazu geeignet, Koordinaten eines Punktes im zweidimensionalen Raum aufzunehmen. Die Übertragung von mehr als zwei Dimensionen für z.B. dreidimensionale Koordinaten ist nicht Teil des Konzepts.

Datenleitungenen

Das Verbinden der Pins geschieht über sogenannte Datenleitungen. Analog zu den Pins werden auch hier in Bezug auf den transportierten Datentyp zwei Arten von Datenleitungen unterschieden.

1D- oder Skalar-Leitungen. transportieren nur einen Wert

2D- oder Punktleitungen. transportieren ein Zweiertupel von Werten

Es können immer nur zwei verschiedene Pins mit dem gleichen Datentyp verbunden werden, also zwei 1D-Pins oder zwei 2D-Pins. Weiterhin muss immer ein Input- mit einem Output-Pin verbunden werden. Es ist ohne Probleme möglich und auch ausdrücklich unterstützt, einen Zyklus im Graphen aufzubauen.

GUI

Die GUI ist die zentrale Entwicklungsumgebung. Hier können die Module per Drag & Drop auf der Zeichenfläche positioniert, und ihr Pins miteinander verbunden werden. Das erstellte Inputsystem kann direkt gestartet und analysiert werden. Dazu öffnet sich ein spezielles Fenster, in dem der Status der verwendeten Module sowie die auf den dazugehörigen Pins anliegenden Daten angezeigt werden.

Laufzeitsystem

Die Hauptaufgabe des Laufzeitsystems ist es, den generierten Graphen zu interpretieren und das entsprechende Inputsystem zu simulieren. Es bezieht Daten von den verwendeten Eingabegeräten, verarbeitet sie je nach Aufbau des Modulgraphen, und gibt die berechneten Daten an eine Applikation weiter oder gibt sie auf der Konsole aus. Das Laufzeitsystem kann dabei entweder von der GUI, oder alleine mit einem XML-File als Spezifikation gestartet werden.

5 Implementation

5.1 Überblick

Zur Implementation wurde Java verwendet, da es zu vielen Plattformen kompatibel ist und ein einfaches Multithreading ermöglicht. Prinzipiell wird jedes Modul als eigener Thread modelliert, was bei Modulen mit komplexeren Berechnungen die Berechnungsdauer verringern und somit die Framerate des Moduls erhöhen kann. Kapitel 5.5 erläutert in welchen Fällen dieses Konzept die Performance verbessert, und in welchen nicht.

5.2 Verwendete Bibliotheken

JGraph

Die wichtigste externe Komponente ist JGraph[9] . Sie übernimmt die Darstellung des Modulgraphen. Durch die Verwendung muss nur noch ein kleinerer Teil der benötigten Funktionen implementiert werden. JGraph stellt die grundlegenden Datenstrukturen bereit. Es gibt Knoten, Ports die an diese Knoten angehängt werden können sowie Kanten, die zwei Ports miteinander verbinden. Die Datenstruktur ist sehr komplex und abstrakt gehalten. Es gibt beispielsweise keine connect-Methode, mit der zwei Ports miteinander verbunden werden können. Um dies zu erreichen, muss ein Edge-Objekt erstellt, und als dessen Quelle und Ziel die beiden gewünschten Ports angegeben werden. Danach wird dieses Objekt in einem Cache des Graphen eingefügt. Attribute können oft nicht an dem betreffenden Objekt eingestellt werden. Dazu besitzt ein Objekt eine Attribute-Map, die die gewünschten Eigenschaften enthält. Nur auf ihr können anhand einer statischen Klasse bestimmte Attribute geändert werden. Das Verbinden von Ports oder Löschen von Knoten und Kanten per Maus ist nicht vorimplementiert. Dazu muss ein Interface implementiert werden. Dieses fragt die Mausposition ab, erstellt die Kontextmenüs, zeichnet die Kanten während sie noch nicht mit ihrem Ziel verbunden sind und registriert eine Kante im Graphcache, sobald sie fertig angelegt wurde.

Um es zusammenzufassen, ist JGraph kein Tool das out-of-the-box verwendet werden kann. Es erfordert viel zusätzliche Implementierungsarbeit und Einarbeitung in die komplexen Datenstrukturen.

JDOM

JDOM[10] ist eine Komponente zum erzeugen und lesen von XML-Dateien. Es gibt verschiedene Alternativen wie z.B. SAX, DOM oder DOM4J. SAX und DOM sind bereits im Java Runtime Environment enthalten. SAX unterstützt lediglich das Parsen von XML-Dokumenten, nicht aber das Schreiben dieser. DOM besitzt auch die Fähigkeit XML zu schreiben, ist aber dabei sehr abstrakt. JDOM bietet hier eine konkretere Struktur, die speziell auf Java zugeschnitten ist. So werden z.B. auch Collections unterstützt. Iterationen über den geparsten Baum sind deutlich einfacher. Bei DOM ist jedes Element und jedes Attribut ein Knoten, so auch der eigentliche Text eines XML-Elements. In JDOM können solche XML-Attribute direkt als Attribut der Node-Klasse ausgelesen werden. DOM4J besitzt diese Funktionen auch. Allerdings

9. http://www.jgraph.com/

10. http://www.jdom.org/

sind in ihm auch Funktionen für die Verarbeitung nach dem klassischen DOM-Modell implementiert. Diese zusätzlichen Funktionen werden in meinem Projekt nicht benötigt. Da DOM4J eine höhere Komplexität bei ähnlichem Nutzen besitzt, habe ich mich gegen es entschieden.

Letztlich wurde JDOM verwendet, da es für Java-Programmierer einfach zu erlernen ist, alle benötigten Funktionalitäten bereitstellt und performant ist. In meiner Applikation erstellt und ließt JDOM die Modulspezifikationen, die als XML-Datei abgelegt werden.

libTUIO

libTUIO ist eine Implementierung des TUIO-Protokolls für Java und kann auf der Community-Page[11] heruntergeladen werden. Die Implementierung stellt dazu eine Klasse bereit, die sich mit dem gewünschten Port verbindet, und bei der Listener registriert werden können. Die Listener, die es selbst zu implementieren gilt, werden dann bei der Erstellung, Veränderung, oder Entfernung von Touch-Punkten benachrichtigt und erhalten entsprechende Objekte als Argument. Eine genauere Beschreibung des TUIO-Protokolls und der in libTUIO enthaltenen Klassen enthält Kapitel 3.

5.3 GUI

Die GUI wurde in Swing geschrieben. Abbildung 19 zeigt diese grafische Oberfläche, die in zwei Teile unterteilt ist. Die große linke Hälfte (1) enthält eine Zeichenfläche, auf der der Modulgraph erstellt werden kann. Rechts daneben befindet sich in einem unabhängigen Fenster der Werkzeugkasten (2). In ihm sind im oberen Abschnitt (über (2)) Steuerknöpfe untergebracht, mit dem das System gestartet („Run“), der gesamte Graph gelöscht („Clear_All“) oder gezoomt werden kann. Per Klick auf „Info“ öffnet sich zu den markierten Modulen ein Fenster mit allen wichtigen Daten wie z.B. Name und verwendeten Konstruktorparametern (Abbildung 18).

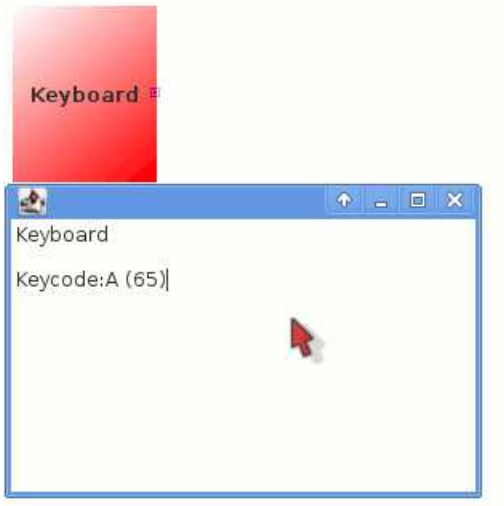

Abbildung 18. Info-Fenster

11. http://www.tuio.org/?software

Direkt darunter befinden sich die verschiedenen Module. Im oberen Teilabschnitt sind Input-, im mittleren Modify-, und im unteren Abschnitt die Output-Module zu finden. Um ein Modul im Graph einzufügen, muss zuerst im Werkzeugkasten auf das entsprechende Modul und danach im linken Fenster auf die gewünschte Position geklickt werden. Sollen weitere Module dieses Typs erstellt werden, genügt es im Graph auf die entsprechende Position zu klicken. Die Farbe des Moduls gibt dabei seine Klasse an. Input-Module sind rot, Modify-Module gelb, und Output-Module blau dargestellt.

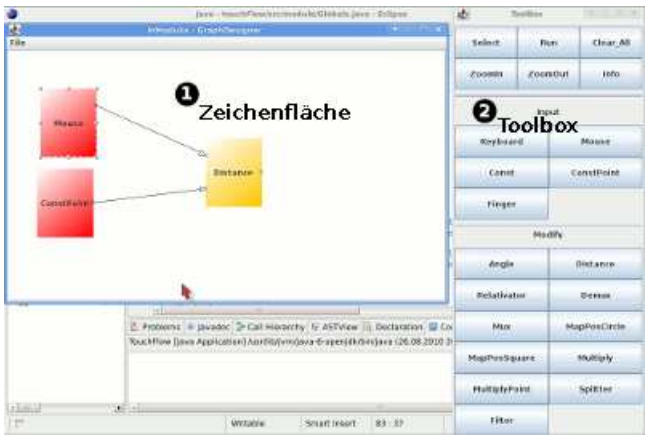

Abbildung 19. Grafische Entwicklungsumgebung von TouchFlow

Eine Modul in der GUI zeigt Abbildung 20. Jedes Modul besitzt Anschlusspins. Diese werden im Graphen durch kleine Vierecke am linken und rechten Rand der Module repräsentiert (1). Die Pins an der linken Seite sind Input-, die an der rechten Seite Output-Pins. Sobald die Maus darüber geführt wird, wird der Name des Pins angezeigt (2). Die Farbe der Schrift gibt dabei an, ob es sich um einen 1D-Pin (blau) oder einen 2D-Pin (rot) handelt. Wie bereits in Kapitel 4 beschrieben wurde, können nur jeweils 1D-Pins und 2D-Pins miteinander verbunden werden, sprich die Pins mit der selben Schriftfarbe.

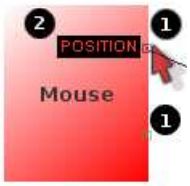

Abbildung 20. Modul in der GUI
1: Pin, 2: Pinname

Um zwei Pins miteinander zu verbinden, muss die linke Maustaste über dem ersten Pin gedrückt und gehalten werden. Danach muss die Maus zum zweiten Pin bewegt und die Taste losgelassen werden. Nur zwei kompatible Pins lassen sich dadurch verbinden.

Um ein Modul oder eine Leitung wieder zu entfernen, muss man auf dem entsprechenden Objekt per Rechtsklick das Kontextmenü aktivieren und dort „delete“ auswählen.

Um ein System zu speichern oder zu laden, existiert im linken Fenster ein Menü „File“ mit den Optionen „Load“ und „Save“.

Sobald das System per Klick auf „Run“ gestartet wurde, öffnet sich ein neues Fenster (Abbildung 21).

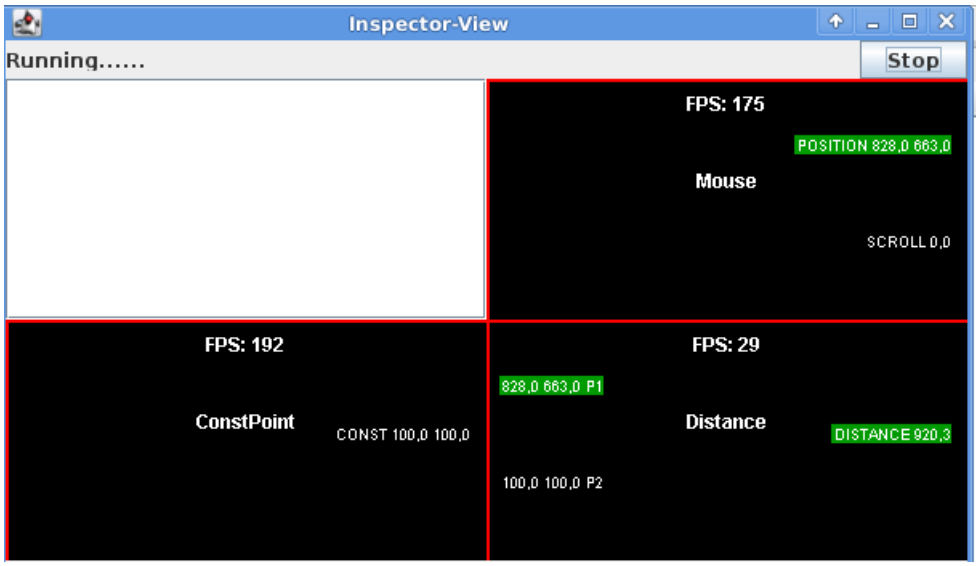

Abbildung 21. Inspector-View

Diese „Inspector-View“ zeigt den momentanen Status aller verwendeten Module sowie deren Pins während dem Betrieb an. Dort kann man die an den Pins anliegenden Daten, die FPS des Moduls sowie vorgehende Änderungen der Pindaten mitverfolgen und so das System debuggen.

5.4 Spezifikation der XML-Datei

Die XML-Datei beinhaltet eine Auflistung der Module. Das gesamte File wird durch <InModulix> ... </InModulix> umschlossen. Eine einfache Zeile könnte z.B. wie folgt aussehen:

```
<modul id="0" type="Mouse" />
```

Das bedeutet, dass ein Modul mit ID 0 existiert, und dieses ein Mouse-Modul ist. In der Implementierung der Module werden die Verbindungen der Pins dadurch gespeichert, dass ein Input-Pin den mit ihm verbundenen Output-Pin als Parameter erhält (s.u.). Diese Methode zeigt sich auch in der XML-Datei. Verbindungen werden im File dadurch hinterlegt, dass im Moduleintrag des entsprechenden Input-Pins der mit ihm verbundene Pin und dessen Modul-ID eingetragen wird. Eine beispielhafte XML-Datei mit Verbindungen und dem dazugehörigen Graphen wäre also folgende (Abbildung 22):

```
<?xml version="1.0" encoding="UTF-8"?>
<InModulix>
   <modul id="0" type="Mouse" />
   <modul id="1" type="Print" INModulID="0" INPinName="SCROLL" />
</InModulix>
```

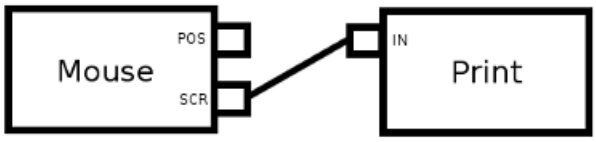

Abbildung 22.

Es existieren zwei Module. Das Modul mit ID 0 ist ein Mouse-Modul, ID 1 ist ein Print-Modul. Um nun die Verbindung zwischen dem Pin „IN“ des Printmoduls und dem Pin „SCROLL“ des Mouse-Moduls herzustellen, sind die beiden Einträge am Schluss des zweiten Moduleintrags zuständig:

```
INModulID="0" INPinName="SCROLL"
```

Das mit dem Pin „IN“ verbundene Modul hat ID 0 (erster Eintrag) und der Name des Pins an diesem Modul ist „SCROLL“.

Was hier im Beispiel fehlt, ist die Übergabe spezieller Konstruktorparameter, wie sie in Kapitel 5.6 dokumentiert sind. Zur Übergabe dieser Parameter gibt es ein spezielles Attribut *Constructor*. Mehrere Parameter werden durch ein Leerzeichen getrennt. Beispielsweise erstellt die Zeile

```
<modul id="2" type="ConstPoint" Constructor="10 100" />
```

ein neues ConstPoint-Modul mit ID 2 und übergibt dabei die beiden Parameter 10 und 100 in dieser Reihenfolge an den Konstruktor. Es gibt keine optionalen Konstruktorparameter. Sind in der Beschreibung Parameter aufgeführt, so müssen alle übergeben werden. Fehlende oder falsch getypte Parameter führen zu einer Exception und Abbruch des Laufzeitsystems.

5.5 Laufzeitsystem des Tools

Das Laufzeitsystem ist für die Interpretation des Graphen zuständig. Es erstellt die zu den Berechnungen gehörenden Threads, startet sie und sorgt für Thread-Safety. Jedes Modul führt dabei nur dann Berechnungen aus, wenn neue Daten an seinen Eingängen anliegen die nicht bereits verarbeitet wurden.

Das Design des Projektes ist auf Multithreading ausgelegt. Das Starten der Threads erzeugt einen gewissen Overhead, der sich bei der parallelen Berechnung der bisher implementierten, einfachen Module nicht unbedingt auszahlt. Für komplexere Module ist dies allerdings durchaus sinnvoll. Werden mehrere komplexe Module in unabhängigen Teilpfaden verwendet, so können diese auf mehreren CPU-Kernen parallel verarbeitet werden. Sind die Module allerdings in einer Pipeline, also einer Verkettung miteinander verbunden, so kann von der Parallelität kein großer Nutzen gezogen werden. Das erste rechenintensive Modul in der Kette gibt innerhalb seines Taktzyklus keine neuen Daten aus, sodass die folgenden Module in dieser Zeit keine Berechnungen durchführen können. Jedes einzelne Modul wird dabei durch einen eigenen Thread repräsentiert. Ein Modul arbeitet also immer in Taktzyklen. Ein typischer Zyklus sieht wie folgt aus:

1. Initialisierung der Pins
2. Berechnen der aktuellen Taktzahl (FPS)
3. Fetchen der an den Input-Pins anliegenden Daten
4. Durchführen der modulspezifischen Berechnungen, dabei schreiben der Ergebnisse auf die Output-Pins
5. Observer benachrichtigen
6. Pausieren des Threads für vorgegebene Zeit

In Schritt 1 werden die Pins mit ihren Standartwerten belegt. Das ist in der Regel 0, können aber auch andere Werte sein (z.B. RP1 des Angle-Moduls). Dadurch wird sichergestellt, dass auch bei nicht angeschlossenen Pins plausible Werte anliegen. Schritt 2 ist eher für das Debugging interessant als für das System an sich. In diesem Schritt wird ermittelt, wie oft pro Sekunde das aktuelle Modul die Daten verarbeitet. Im nächsten Schritt werden die Input-Daten von den damit verbundenen Output-Pins bezogen, insofern sie verbunden sind. Der darauffolgende Schritt ist die Berechnung an sich, die je nach der spezifischen Modulimplementierung verschieden ist. Welche Module genau existieren, behandelt Abschnitt 5.6. Um externe Komponenten über die Änderung zu informieren, werden im nächsten Schritt alle registrierten Listener (Observer) über die durchgeführte Neuberechnung informiert. Letztendlich pausiert der sechste Schritt den Modulthread für eine einstellbare Zeit. Der Grund hierfür liegt darin, dass es erst wieder sinnvoll ist die Berechnungen erneut durchzuführen, sobald neue Input-Daten vorliegen. Das aktuelle Modul gibt daher durch diese Methode den anderen Modulen die Möglichkeit, einen Taktzyklus durchzuführen bevor es selbst wieder aktiv wird.

Diese grundlegenden Aufgaben werden durch die abstrakte Klasse „Modul“ durchgeführt. Sie repräsentiert auch andere Grundfunktionen eines Moduls wie z.B. ihre Pins. Es gibt zwei Methoden um eine Kommunikation mit einer fremden Applikation zu ermöglichen. Die erste Methode sieht die Anbindung über das Java Observer Interface vor, die zweite beruht auf einem benutzerspezifischen Protokoll. Für Methode 1 muss das Projekt als Java-Quellcode vorliegen. Das Observer-Pattern wird in [18] und [22] beschrieben. Jedes Modul implementiert das Observable-Interface. Zur Kommunikation muss eine Klasse des externen Projekts das Observer-Interface implementieren und sich bei dem entsprechenden Modul registrieren. Danach wird die Klasse per Funktionsaufruf über jede neue Berechnung des beobachteten Moduls informiert, und kann dementsprechent agieren. In der zweiten Methode funktioniert die eigentliche Kommunikation über ein Protokoll. Die Anbindung kann an Projekte in einer beliebigen Programmiersprache erfolgen. Um die Daten von meinem Inputsystem zu versenden, muss ein entsprechendes Output-Modul implementiert werden. Dieses kann den Datenstrom über das Protokoll versenden. Die externe Anwendung kann die Daten dann über das Protokoll abfangen und verarbeiten. In meiner Arbeit ist kein entsprechendes Output-Modul oder Spezifikation eines Protokolls vorhanden.

5.6 Implementierte Module

Diese Sektion behandelt die momentan implementierten Module.

Um die Vektorrechnungen für zweidimensionale Vektoren zu vereinfachen, wurde eine spezielle MathVektor-Klasse implementiert. Sie kann die Länge (den Betrag) eines Vektors errechnen oder diesen um einen bestimmten Winkel drehen. Sie wird in den Implementierungen der Morph-Map (MapPosCircle und MapPosSquare) eingesetzt. Ihr Konstruktor übernimmt entweder zwei double-Werte, die die Komponenten x_1 und x_2 des Vektors angeben. Oder er übernimmt einen anderen MathVektor, den er kopieren soll (Copy-Konstruktor).

5.6.1 Input-Module

Input-Module repräsentieren in der Regel Input-Geräte oder Konstanten, die in das System eingespeist werden. Sie besitzen keine Input-Pins, da sie keine Daten verarbeiten oder annehmen, sondern nur generieren.

Mouse

Repräsentiert eine Maus mit Scrollrad. Die Mausposition kann dabei global bestimmt werden, selbst wenn der Zeiger sich außerhalb der GUI befindet. Die Scrollposition kann nur ermittelt werden, wenn der die Maus sich beim Scrollvorgang über dem zugehörigen Inspector in der Inspector-View befindet.

Konstruktorparameter: keine

Output:

- *POSITION* (PL): Position des Mauszeigers
- *SCROLL* (SL): Scrollzustand der Maus. Positive Werte geben dabei eine Drehung des Rades nach oben, negative Werte eine Drehung nach unten an.

TUIOFinger

Repräsentiert den n-ten Finger, der momentan auf der Multitouch-Oberfläche präsent ist. Die Zählung beginnt hierbei bei 0. Sind zur Zeit weniger als n+1 Finger erkannt worden, wird am Pin *POSITION* das Wertepaar (0,0) ausgegeben. n wird im Konstruktor als Parameter *fingerId* angegeben.

Dieses Modul implementiert das TUIO-Protokoll und kann somit mit jedem Steuerprogramm kommunizieren, welches dieses Protokoll unterstützt. Das von mir verwendete CCV unterstützt TUIO ebenfalls.

Konstruktorparameter:

- *fingerId* (int): Nummer des Fingers (s.o.)

Output:

- *POSITION* (PL): Position des Fingers.

Const

Konstanter Skalar. Nützlich um z.B. Multiplikator festzulegen.

Konstruktorparameter:

- *Konstante* (double): die eingestellte Konstante

Output:

- *CONST* (SL): die Konstante

ConstPoint

Wie Const, nur gibt dieses Modul einen Punkt aus, der über eine PL weitergeleitet werden kann. Anwendung z.B. in Angle, um feste Referenzpunkte anzugeben.

Konstruktorparameter:

- *Konstante* (double): X-Position des konstanten Punkts
- *Konstante2* (double): Y-Position des konstanten Punkts

Output:

- *CONST* (PL): der eingestellte Punkt

Keyboard

Gibt an, ob eine bestimmte Taste auf der Tastatur gedrückt ist. Besonders sinnvoll in Verbindung mit (De-)Multiplexer.

Konstruktorparameter:

- *keyCode* (int): der KeyCode des java.awt.event.KeyEvent der zu überwachenden Taste

Output:

- *PRESSED* (SL): 1 falls die Taste gedrückt ist, ansonsten 0.

5.6.2 Modify-Module

Modify-Module repräsentieren Berechnungen oder sonstige Datenverarbeitungen. Sie können am einfachsten mit Funktionen aus Programmiersprachen verglichen werden. Sie besitzen sowohl Input- als auch Output-Pins um die zu verarbeitenden Daten anzunehmen und die Ergebnisse wieder auszugeben.

Distance

Berechnet die Distanz zwischen zwei Punkten in euklidischer Norm.

Konstruktorparameter: keine

Input: zwei Punkte, die die Distanz begrenzen

- *A1* (PL): erster Punkt
- *A2* (PL): zweiter Punkt

Output:

- *DIST* (SL): Distanz der Punkte

Angle

Berechnet den Winkel zwischen zwei Geraden, die durch jeweils zwei Punkte definiert sind. Eine ist dabei als konstante Referenzgerade (definiert durch RP1 und RP2) vorgesehen, die andere ist eine variable Gerade, zu der der Winkel berechnet werden soll. Es könnten aber auch zwei variable Geraden verwendet werden. Sind die Pins der Referenzgeraden nicht verbunden, wird standartmäßig die y-Achse als Referenzgerade gewählt. Abbildung 23 zeigt die Berechnung des Winkels. Die obere Gerade durch RP1 und RP2 ist dabei die Referenzgerade, die untere Gerade durch P1 und P2 die variable Gerade. Der Winkel ergibt sich aus dem Schnittwinkel dieser beiden Geraden.

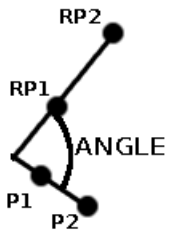

Abbildung 23. Berechnung des Winkels

Konstruktorparameter: keine

Input:

- *REF_P1* (PL): Aufpunkt 1 der Referenzgerade
- *REF_P2* (PL): Aufpunkt 2 der Referenzgerade
- *P1* (PL): Aufpunkt 1 der variablen Gerade
- *P2* (PL): Aufpunkt 2 der variablen Gerade

Output:

- *ANGLE* (SL): Winkel zwischen den zwei Geraden in Bogenmaß

Relativator

Gibt anstatt des absoluten Wertes am Eingangspin den relativen Wert aus, indem es die Differenz zum jeweils vorhergehenden Zyklus berechnet.

Konstruktorparameter: keine

Input:

- *IN*(SL): zu relativierender Inputwert

Output:

- *OUT* (SL): relativierter Inputwert

MapPosCircle

Dieses Modul implementiert eine kreisförmige Morph-Map. Bei ihr sind die Schlüsselpunkte auf einem Kreis äquidistant angeordnet (Abbildung 24) . Der erste Punkt liegt auf 12 Uhr, alle weiteren werden im Uhrzeigersinn angeordnet (Punkte 1-4). Die Numerierung erfolgt dementsprechend. Optional kann auch ein Schlüsselpunkt in der Mitte existieren, welcher die höchste Nummer erhält.

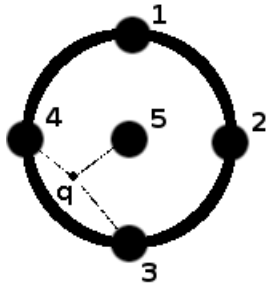

Abbildung 24. kreisförmige Morph-Map

Die genaue Funktionsweise einer Morph-Map wird in [21] beschrieben und ist sehr ähnlich zu Spatial Keyframing [19]. Im Prinzip geht es darum, eine Interpolation von verschiedenen Schlüsselpunkten zu berechnen. Die Schlüsselpunkte sind dabei feste Punkte auf der Karte. Mit einer speziellen Formel, die weiter unten erläutert wird und die vom Abstand des Punktes auf der Karte zur jeweiligen Schlüsselposition abhängt, wird der Anteil dieser Schlüsselposition an der Interpolation angegeben. Dieser „Anteil“ wird nachfolgend „Gewicht“ g genannt.

In der Praxis hat es sich gezeigt, dass die Interpolation von maximal drei verschiedenen Schlüsselelementen sinnvoll ist. Daher werden die drei Punkte ausgewählt, die dem zu untersuchenden Punkt q am nächsten sind. Die Menge dieser drei Punkte wird nachfolgend PT genannt und die darin enthaltenen Punkte P_1, P_2, P_3, sodass $\mathrm{PT} = \{P_1, P_2, P_3\}$. Alle anderen Punkte $p \notin \mathrm{PT}$ werden bei der Berechnung vernachlässigt, ihr Gewicht wird auf 0 gesetzt. Für einen ausgewählten Schlüsselpunkt p auf der Karte wird das Gewicht g_p, welches auch später auf den entsprechenden Output-Pins anliegt, wie folgt berechnet:

$$g_p = \frac{\mathrm{dist}(p)}{\sum_{r \in \mathrm{PT}} \mathrm{dist}(r)} \tag{1}$$

$$\mathrm{dist}(p) = \frac{\sum_{r \in \mathrm{PT}} \|r - q\|}{\|p - q\|} \qquad \|\cdot\| \text{ ist hierbei die euklidische Norm.} \tag{2}$$

Mit $|PT| = 2$ erhält man somit eine Art „Slider“.

Die Funktion in (2) berechnet den Abstand des Punktes p von q im Verhältnis zur Summe aller Abstände der drei ausgewählten Punkte. Dabei steht die Summe im Zähler und der einzelne Abstand im Nenner, um eine umgekehrte Proportion herzustellen. Ein kleiner Abstand soll ein großes, ein großer Abstand ein kleines Gewicht ergeben. Dieses Ergebnis muss nun noch auf ein Intervall [0...1] normalisiert werden. Diese Normalisierung wird von Formel (1) durchgeführt.

Konstruktorparameter:

- *center* (MathVector): Mittelpunkt des Kreises
- *radius* (double): Radius des Kreises
- *numOfPoints* (int): Anzahl der Punkte auf dem Kreis (eventueller Mittelpunkt *nicht* mitgezählt). Es muss gelten: numOfPoints $\geqslant 2$
- hasCenter (boolean): gibt an, ob ein Schlüsselpunkt im Mittelpunkt des Kreises existieren soll

Input:

- *POSITION* (PL): zu untersuchende Position q

Output:

So viele SL-Output-Leitungen, wie es Schlüsselpunkte gibt. Jede gibt das Gewicht des zugehörigen Schlüsselpunktes an.

MapPosSquare

Ähnlich wie MapPosCircle, nur dass die Punkte hier auf einem Quadrat mit optionalem Mittelpunkt angeordnet sind (Abbildung 25). Der erste Punkt befindet sich immer in der oberen linken Ecke. Um Sprünge zu vermeiden und Kompatibilität zu dem Projekt aus [21] zu gewährleisten, wird immer der Mittelpunkt und die zwei am nächsten liegenden Peripheriepunkte ausgewählt, insofern ein Mittelpunkt vorhanden ist.

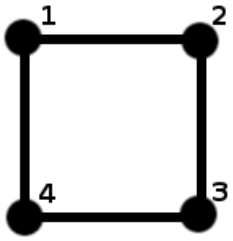

Abbildung 25. Quadratische Morph-Map

Konstruktorparameter:

- *center* (MathVector): Mittelpunkt des Rechtecks (Schwerpunkt)

- *breite* (double): Breite des Rechtecks
- *numOfPoints* (int): Anzahl der Punkte auf dem Rechteck (eventueller Mittelpunkt *nicht* mitgezählt). Es ist nur 4 und 8 erlaubt.
- hasCenter (boolean): gibt an, ob ein Schlüsselpunkt im Mittelpunkt des Rechtecks existieren soll

Input/Output: Wie bei *MapPosCircle*

Multiply

Multipliziert den Wert einer Datenleitung mit einer Zahl, die ebenfalls über eine Datenleitung zugeführt wird.

Konstruktorparameter: keine

Input:

- *INPUT*(SL): zu multiplizierende Leitung
- *SCALER* (SL): Multiplikator

Output:

- *OUTPUT* (SL): multiplizierte Leitung

MultiplyPoint

Wie *Multiply*, allerdings multipliziert dieses Modul die beiden Einzelwerte einer Punktleitung (PL) mit einem Skalar.

Konstruktorparameter: keine

Input:

- *INPUT*(PL): zu multiplizierende Leitung
- *SCALER* (SL): Multiplikator

Output:

- *OUTPUT* (PL): multiplizierte Leitung

Demultiplexer(Demux)

Das Modul ist ähnlichen einem elektronischer Demultiplexer, wie er unter anderem in [17] beschrieben ist. Abbildung 26 zeigt das Schaltsymbol (oben) und die grundlegende Funktionsweise eines Demultiplexers. Es ist ein Selektionsschaltnetz, bei dem es nur einen Eingang, aber mehrere Ausgänge gibt. Über ein Steuersignal wird festgelegt, auf welchen der Ausgänge das Eingangssignal gelegt wird. Es kann mit einem Schalter verglichen werden (Abbildung 26 unten), der je nach Eingangssignal automatisch an (Signal liegt auf Leitung 1) oder aus (Signal liegt auf Leitung 0) geschaltet wird.

Diese Implementierung umfasst nur Demultiplexer mit zwei Ausgängen. Für das Steuersignal sind Werte von 0 (Ausgang 0 wird selektiert) und 1 (Ausgang 1 wird selektiert) vorgesehen. Werte ungleich 0 oder 1 werden als 1 interpretiert und mit einer Warnung quittiert.

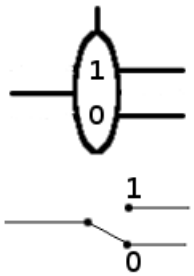

Abbildung 26. Schaltbild und Funktionsweise eines Demultiplexers

Bei Start des Programms liegt an allen Ausgängen der Wert 0 an. Wurde ein Ausgang selektiert und danach ein anderer ausgewählt, so bleibt an dem vorher selektierten Ausgang der letzte Wert des Eingangssignals vor dem Schaltvorgang bestehen. Er wird nicht auf 0 zurückgesetzt.

Konstruktorparameter: keine

Input:

- *DATA0* (SL): Eingangssignal
- *SELECT* (SL): Steuersignal

Output:

- *OUT0*: Ausgang, der bei SELECT $= 0$ selektiert wird
- *OUT1*: Ausgang, der bei SELECT $\neq 0$ selektiert wird

Multiplexer

Der Multiplexer ist das Gegenstück zum Demultiplexer. Abbildung 27 zeigt Schaltbild und Funktionsweise dieses Bausteins. Es existieren mehrere Eingänge und nur ein Ausgang. Welcher Eingang zum Ausgang durchgeleitet wird, entscheidet ein Steuersignal. Auch die grundlegene Funktionsweise in Abbildung 27 unten ist ähnlich zu der des Demultiplexers. Im Ersatzschaltbild ist lediglich der Schalter vertikal gespiegelt eingebaut. Die Stellung des Schalters entscheidet also, welche der beiden Leitungen mit der Ausgangsleitung verbunden wird. Wie auch der Demultiplexer nur zwei Ausgänge hat, hat der Multiplexer zwei Eingänge.

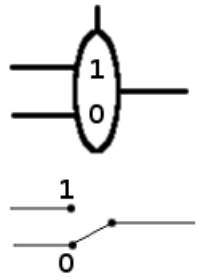

Abbildung 27. Schaltbild und Funktionsweise des Multiplexers

Konstruktorparameter: keine

Input:

- *DATA0* (SL): Eingang, der bei SELECT = 0 selektiert wird
- *DATA1* (SL): Eingang, der bei SELECT ≠ 0 selektiert wird
- *SELECT* (SL): Steuersignal

Output:

- *OUT*: selektierte Leitung

Filter

Der Filter ist zum Filtern von Punktdaten konzipiert. Er filtert fehlerhafte Koordinaten heraus, indem es Werte die stark von dem vorhergehenden Wert abweichen vorerst ignoriert, und stattdessen die alten Koordinaten weiterhin ausgibt. Im Detail gibt es zwei Parameter *Threshold* und *maxDist*. maxDist gibt die maximale Distanz in euklidischer Norm an, bis zu der die Bewegung als „normale Abweichung“ bewertet und direkt weitergeleitet wird. Ist die Distanz größer, wird der letzte Output-Wert anstelle der aktuellen Input-Daten gesendet. Werden eine bestimmte Anzahl von aufeinanderfolgenden Zyklen stark abweichende Punkte gesendet, so wird beim nächsten Zyklus der Punkt durchgeleitet, auch wenn dieser als starke Abweichung registriert wurde. Diese bestimmte Anzahl an Zyklen wird im Parameter *Threshold* definiert. Ein hoher Threshold sorgt für eine geringere Fehleranfälligkeit, macht die Steuerung allerdings träger, da z.B. ein Finger der an einer anderen Stelle aufgesetzt wird erst nach längerer Zeit durchgeleitet werden kann.

Konstruktorparameter:

- *maxDist* (double): maximale Distanz, die als normal betrachtet werden soll
- *Threshold* (int): Anzahl von Zyklen, nach denen ein stark abweichender Punkt durchgeleitet wird

Input:

- *IN* (PL): zu filternder Wert

Output:

- *OUT* (PL): gefilterter Wert

5.6.3 Output-Module

Output-Module sind lediglich zur ausgabe der berechneten Daten auf der Konsole oder anderen externen Objekten gedacht. Sie könnten auch die erhaltenen Daten über Protokolle an andere Anwendungen oder über das Netzwerk verschicken. Sie besitzen keine Output-Pins, da sie die konsumierten Daten nicht innerhalb des Modulsystems weitergeben.

Print

Gibt in jedem Zyklus die auf der Leitung *IN* anliegenden Daten unformatiert auf der Konsole aus. Mit jeder Ausgabe erfolgt ein Zeilenumbruch.

Konstruktorparameter: keine

Input:

- *INPUT* (SL): auszugebende Datenleitung

FormattedOut

Gibt die Daten von bis zu vier Leitungen formatiert aus. Dem Konstruktor wird ein Formatstring übergeben. Die Stellen, an denen der Wert der Leitungen eingefügt werden soll, wird dabei mit %1, %2 etc. für die erste, zweite Datenleitung usw. gekennzeichnet. Wenn also im Moment auf Leitung 1 der Wert 10.0 und auf Leitung 3 der Wert 5.0 anliegt, würde ein mit dem Formatstring

```
‘‘Leitung 1 hat: %1 und Leitung 3 hat:%3‘‘
```

erstelltes Modul in diesem Zyklus auf der Konsole

```
‘‘Leitung 1 hat: 10.0 und Leitung 3 hat:5.0‘‘
```

ausgeben.

Konstruktorparameter:

- *formatString* (String): der Formatstring (s.o.)

Input:

- *DATA1* (SL): auszugebende Datenleitung 1
- *DATA2* (SL): auszugebende Datenleitung 2
- *DATA3* (SL): auszugebende Datenleitung 3
- *DATA4* (SL): auszugebende Datenleitung 4

5.7 Test-Applikation

Um die Funktionstüchtigkeit des Projektes zu demonstrieren, gibt es eine entsprechende Applikation im Paket „testapp“. Abbildung 28 zeigt die Testapp bei der Bedienung am großen FTIR-Display. Diese Anwendung greift auf mein Tool zurück, um mit dem in Kapitel 3 beschriebenen Display Vierecke auf einer Zeichenfläche zu platzieren, zu skalieren und zu rotieren.

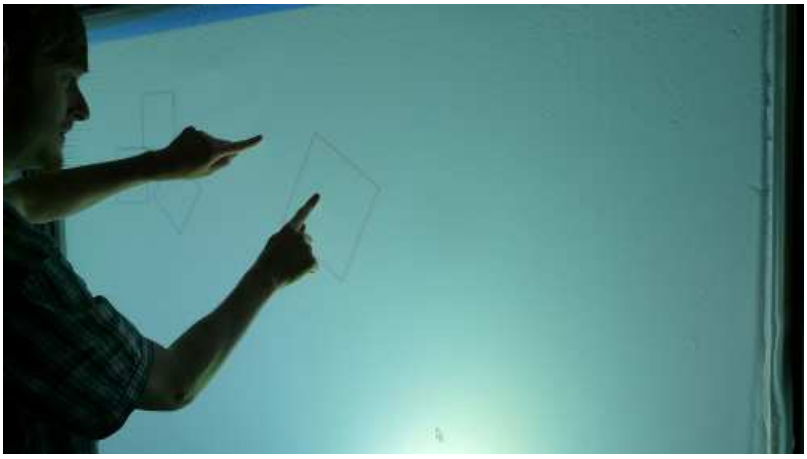

Abbildung 28. Die Testapplikation im Betrieb

Das Inputsystem wurde mit meinem Programm erstellt und ist in Abbildung 29 skizziert. Die Koordinaten der beiden Finger werden dabei zuerst gefiltert, da mit dem verwendeten Display oft durch falsch erkannte Bildpunkte fehlerhafte Koordinaten ausgegeben werden. Diese „Sprünge“ werden durch diese Module herausgefiltert. Danach werden diese Daten an ein Distance- und ein Angle-Modul weitergeleitet. Die Platzierung eines Vierecks geschieht durch zweimaliges Antippen der Oberfläche innerhalb kurzer Zeit. Für diesen Vorgang wird nur die Position des ersten Fingers benötigt. Um diese Information zu erhalten, werden die Positionsdaten vom POSITION-Pin des entsprechenden Finger-Moduls (in der Grafik als „Finger 1“ bezeichnet) abgefragt. Skalierung und Rotation eines Dreiecks wird durch zwei Finger realisiert. Die Entfernung der beiden Finger gibt dabei die Größe, und der Winkel zwischen der Gerade durch die beiden Punkte und der y-Achse den Drehungswinkel an. Diese Informationen werden vom DISTANCE-Pin des Distance-Moduls und dem ANGLE-Pin des Angle-Moduls bezogen.

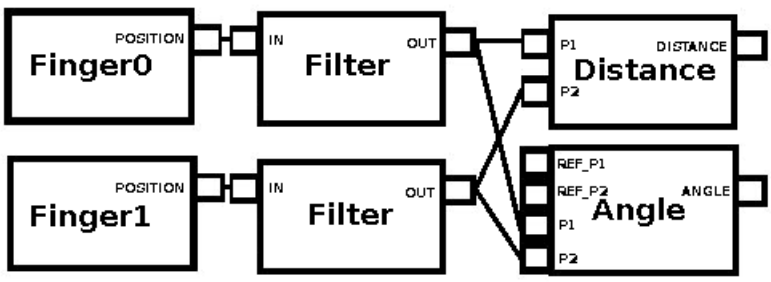

Abbildung 29. Design des Inputsystems

Um vorhandene Dreiecke wieder zu entfernen, müssen diese in den als „waste area“ gekennzeichneten, rot markierten Bereich in der rechten unteren Bildschirmecke verschoben werden.

6 Konklusion

6.1 Stärken und Schwächen

Die Entwicklungsumgebung ist einfach zu bedienen und ermöglicht eine schnelle Erstellung von Eingabesystemen. Die darunterliegende Laufzeitumgebung kann die erstellten Systeme starten, wodurch das designte System direkt ausprobiert und bewertet werden kann. Diese Kombination sorgt für einen transparenten Entwicklungsprozess, bei dem die erstellten Systeme sehr verständlich sind. Ebenso sorgt das grafische Design für ein einfaches Re-Design. Veränderungen sind innerhalb kurzer Zeit durchführbar, indem Module ersetzt oder Verbindungsleitungen umverbunden werden. Die Systeme können nach Abschluss der Evaluation in einem fertigen Produkt eingesetzt werden, da die Laufzeitumgebung unabhänig von der Entwicklungsumgebung Spezifikationen lesen und ausführen kann. So wird auch eine schnelle Fertigstellung des Eingabesystems gewährleistet.

Ein weiterer Vorteil meiner Implementierungvon TouchFlow ist der modulare Aufbau. Die Grundfunktionen eines Moduls sind in der abstrakten Modulklasse gekapselt, sodass neue Module nur die gewünschten Berechnungen sowie die Angabe der benötigten Pins enthalten müssen. Letztlich ist durch die Verwendung von Java als Programmiersprache eine weitgehende plattformunabhängigkeit gewährleistet.

Nachteilig ist an meiner Arbeit, dass sie nur als Teil eines größeren Projekts eingesetzt werden kann. Sie kann zwar mit Eingabegeräten kommunizieren und die davon gelieferten Daten aufbereiten, diese Rohdaten sind aber für den Benutzer nicht direkt verwendbar. Nur in einem größeren Projekt, das diese Daten in Aktionen umwandelt oder sie weiterverarbeitet, zeigt sich die Arbeitsersparnis die dieses Projekt mit sich bringt. Weiterhin kann das Laufzeitsystem nur begrenzt mit dem darumliegenden Framework interagieren. Es kann z.B. zu einem gegebenen Rechteck nicht ohne weiteres feststellen, welche Dateien auf dem Desktop sich innerhalb dieses Rechtecks befinden. Auch das Erfassen eines Tastendrucks ist nur innerhalb eines Steuerelements möglich, welches in meinem Framework zuvor registriert wurde. Das globale Abfragen ist nicht möglich. Letzteres und auch eine Kommunikation mit Drittanwendungen wäre durch JNI [8] möglich. Hierbei werden native Bibliotheken, die beispielsweise in C geschrieben sind, in Java eingebunden und verwendet. Dadurch verliert das Tool allerdings seine plattformunabhängigkeit. Abgesehen davon sind danach zusätzliche Bibliotheken nötig, die seperat installiert werden müssen. Schlussendlich bleibt, dass das Multithreading einen Overhead ergibt, der sich durch die parallele Bearbeitung der bisher implementierten Module eher nicht auszahlt. Wie das nächste Kapitel beschreibt, könnte sich die parallele Bearbeitung mit komplexeren Modulen allerdings durchaus rechnen.

Um alles zusammenzufassen, überwiegen die Vorteile, die sich besonders durch eine hohe Transparenz, Flexibilität in Bezug auf Re-Design und Evaluation sowie einer guten Erweiterbarkeit des bestehenden Modulsystems auszeichnen.

6.2 Mögliche Erweiterungen

Es wären mehrere neue Module denkbar, die auch schwierigere Berechnungen durchführen. Die hier beschriebenen und implementierten Module sind relativ simpel und wenig rechenintensiv. Für diese Module wäre eine serielle Verarbeitung ausreichend. Andere zukünftige Module könnten allerdings von dem jetzigen Design profitieren. Beispielsweise könnte eine Gestenerkennung implementiert werden, wie sie im PC-Spiel „Black and White“ und „MerX InMotion“[12] vorhanden ist. Um diese Implementierung durchzuführen, müsste das Modul die Positionen der letzten Frames aufzeichnen. Parallel dazu braucht man eine Hauptspeicher-Datenbank, die mögliche Gesten enthält. Diese könnte auch von außen oder über ein Netzwerk zugeführt werden. Danach können die aufgezeichneten Positionen mit der Datenbank abgeglichen, und bei einem gematchten Element der entsprechende Pin auf „1“ gesetzt werden. Ein anderes, komplexeres

12. http://www.imerx.net/

Modul wäre ein nichtlinearer Multiplikator. Dieser wäre z.B. bei einem Helikopter-Spiel von Bedeutung, um die Geschwindigkeit zu berechnen. In diesem Szenario beschleunigt der Helikopter bei geringer Geschwindigkeit sehr schnell, wohingegen mit zunehmender Geschwindigkeit und dem daraus resultierenden Luftwiderstand die Beschleunigung überproportional stark abnimmt. Um diesen Multiplikator zu berechnen, bieten sich besonders *Splines* [2] an. Zu einem solchen Spline-Modul könnten bei der Erstellung Grad, Knoten und Gewichtungsfaktoren angegeben werden, anhand derer der Spline definiert wird. Das Modul berechnet daraus die Spline-Kurve und kann anhand derer eine beispielsweise kubische Beschleunigung simulieren. Eine weitere Modulklasse könnte Feedback-Signale an die Eingabegeräte zurück senden. Beispielsweise könnte ein Modul implementiert werden, das Cut-Off Regionen definiert und ihre Einhaltung überprüft. Mit diesem Modul könnte dann ein Force-Feedback an ein kompatibles Eingabegerät gesendet werden, sobald sich die Positionswerte an den Rand der Cut-Off-Region bewegen. Dadurch wäre das System nicht nur auf die Verarbeitung und Weitergabe von Eingabesignalen beschränkt, sondern könnte auch einen kleinen Teil der Ausgabe übernehmen.

Es wäre auch eine Verbesserung des User Interfaces möglich. Es wäre denkbar, die Steuerung der GUI selbst mit einem multitouch-fähigen Gerät zu ermöglichen, ähnlich wie beim reacTable Projekt [12]. Der reacTable überträgt die Fähigkeiten eines Software-Synthesizers wie MAX/MSP auf ein Tabletop Tangible Interface und ermöglicht so eine einfachere und intuitivere Bedienung. Dabei werden nicht nur klassische Multitouch-Funktionen verwendet. Jedes Modul wird hier durch einen in der Realität existierendes, einfaches, geometrisches Objekt repräsentiert. Die auf dem Tisch platzierten Objekte interagieren untereinander und bilden dadurch ein komplexeres Synthesizer-System. Dadurch wird ein zusätzliches visuelles und vorallem haptisches Feedback erreicht, was diese Arbeit besonders für die Testuser sehr interessant macht. Die Module können dann beispielsweise gedreht werden, um Lautstärke oder Geschwindigkeit zu verändern. Durch verschieben der Modulblöcke können diese mit anderen Modulen in der Umgebung interagieren. Zur Implementierung eines Multitouch-Interfaces für mein Projekt könnten besonders gut Gesten verwendet werden, um beispielsweise verschiedene Module auf der Arbeitsfläche zu platzieren oder zu entfernen. Das würde die Bedienung noch mehr erleichtern, und auch die Produktivität aufgrund der ersparten Arbeitsschritte erhöhen. Allerdings erfordert das Erlernen der Gesten auch etwas Einarbeitungszeit. Weiterhin müsste die entsprechende Hardware angeschafft und instand gehalten werden, was einen weiteren finanziellen Aufwand darstellt.

Eine weitere Möglichkeit um den Entwicklungsprozess zu vereinfachen, wäre die Implementierung einer kleinen Entwicklungsumgebung, um neue Module zu entwerfen und direkt zu testen. Dies würde sich allerdings erst lohnen, falls das Tool häufiger benutzt wird und neue Module oft benötigt werden. Letztlich wäre ein Modul denkbar, das per JNI die momentan auf Java-interne Komponenten beschränkten Events global erfassen kann. Die nächste Stufe wäre ein abstraktes Modul, das das relativ komplexe Management von JNI übernimmt. Es erhält die gewünschte Bibliothek als Parameter, sodass diese dann in einer spezifischen Implementierung der abstrakten Klasse verwendet werden kann. Dieses abstrakte Modul könnte dann auch das Error Handling übernehmen, wenn beispielsweise die benötigte Bibliothek nicht existiert, nicht geladen werden kann oder in einer falschen Version vorliegt.

Eine andere Erweiterungsmöglichkeit wäre eine statische Modellprüfung, wie sie in Moby/PLC [20] der Universität Oldenburg angeboten wird. Moby wurde entwickelt, um SPS-Systeme zu erstellen. SPS-Systeme sind eine Verschaltung vieler kleiner elektronischer Bauteile, die kleinere Rechenaufgaben durchführen können. Unter [23] wird SPS genauer erklärt. Das Tool erlaubt es, an dieses System bestimmte Eigenschaften zu stellen, die dann automatisch geprüft werden können. Auch in meinem Programm könnte eine solche Prüfung nützlich sein. Mit ihr könnten beispielsweise unbeabsichtigte Zyklen im Graphen, vom Hauptsystem isolierte Teilgraphen oder potentiell falsche Verbindungen erkannt werden. Ein weiteres Feature von Moby ist die Generierung von Code. In Moby kann Sourcecode für die einzelnen Bausteine generiert werden, sodass dieser direkt auf die Bausteine gebrannt werden kann. Das System muss dann nur noch verbunden werden. Auch mein Programm könnte anstatt das Inputsystem zu interpretieren, speziellen Code dafür generieren. Dadurch wäre die Ausführung des Codes bzw. des Inputsystems unabhängig von meinem Laufzeitsystem. Die Verwendung meines Tools in Verbindung mit einem fremden Projekt wäre dadurch deutlich einfacher und flexibler.

6.3 Anwendungsgebiete des Tools

Multitouch-Geräte und andere neuere Eingabegeräte erfahren eine wachsende Beliebtheit auf dem Markt. Durch sie kann die Bewegungsfreiheit im Vergleich zu konventionellen Eingabegeräten deutlich erhöht werden. Das hat besonders bei Spielen einen besonders starken Effekt, weshalb auch viele der nachfolgend aufgelisteten Geräte aus diesem Segment stammen. TouchFlow kann Input-Systeme für viele solcher neuartiger Geräte modellieren, wie die nachfolgenden Beispiele zeigen.

FTIR-Displays

Wie schon im Eingangsbeispiel gezeigt wurde, können sehr einfach verschiedene Input-Systeme für FTIR-Displays erstellt werden. Spezielle Displays, wie das experimentelle Display aus Kapitel 3 sind bereits in meinem Projekt implementiert und können direkt verwendet werden. Aus diesem Grund sind auch die meisten momentan implementierten Module für dieses Einsatzgebiet konstruiert worden. Mit dem Tool können vielfältige verschiedene Designs schnell modelliert und gegenübergestellt werden. Parameter wie Gain-Faktor, oder ob das Zooming absolut oder relativ erfolgen soll, können schnell eingestellt werden.

WiiMote

Ein weiteres Einsatzgebiet wäre die WiiMote[13] von Nintento (Abbildung 30). Nehmen wir beispielsweise an, es muss ein Spiel für Schwertkämpfe implementiert werden. Nun steht der Entwickler vor der Frage, welche Bewegung er als Schlag zählen soll. Sollen nur Schläge von oben nach unter gezählt werden, so darf nur eine mehrheitliche Beschleunigung in Z-Richtung gewertet werden. Allerdings wäre es auch denkbar, dass Schläge nur horizontal, also in X-Richtung ausgeführt werden sollen. Auch eine Kombination von beidem ist plausibel. Weitere Faktoren wären ob noch zusätzlich ein Knopf gedrückt werden muss, wie groß ein eventueller Gain-Faktor sein sollte etc. Diese vielfältigen Faktoren und ihre Kombinationen könnten mit meine Programm sehr gut modelliert und getestet werden.

In diesem Dokument [1] wird die Verwendung der WiiMote behandelt, um anhand der Gesten des Benutzers seine kulturelle Abstammung zu bestimmen. Auch hier könnte ein Inputsystem mit meiner Arbeit gut modelliert werden. Ein Modul zur Gestenerkennung, wie es in Kapitel 6.2 beschrieben wird, könnte hier dazu benutzt werden um die Gesten der Probanden zu erkennen. Anhand der erkannten Gesten könnte darauf z.B. ein Verbund von Demultiplexer-Modulen die erkannte Abstammung ausgeben. Hier wäre es möglich einen sehr großen Teil des Projektes anhand meines Tools zu implementieren, wenn auch wahrscheinlich mehrere zusätzliche, komplexere Module erstellt werden müssten. Ein zusätzlicher Vorteil bei diesem Verfahren ist es, das die Kombination von Gesten, die über eine kulturelle Zugehörigkeit entscheidet, im Diagramm sehr gut sichtbar ist. Dadurch fällt es sehr einfach, die Erkennungsmechanismen zu diskutieren und zu verfeinern.

13. http://www.nintendo.com/wii/console/controllers

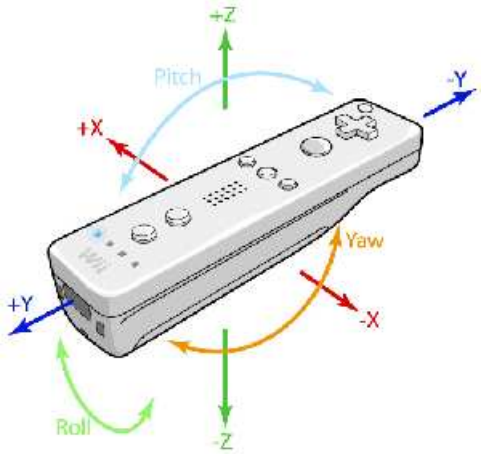

Abbildung 30. WiiMote-Controller

3D-Mäuse

Ebenso wäre ein Einsatz bei sogenannten 3D-Mäusen (Abbildung 31) denkbar, wie sie von 3Dconnexion[14] angeboten werden. 3D-Mäuse zeichnen sich durch eine sehr hohe Anzahl and DOF („Degrees of Freedom“, Freiheitsgrade) aus. Die in Abbildung 31 abgebildeten Modelle besitzen 6 DOF durch kippen, schieben, drücken, ziehen und drehen. Das ist einer mehr als bei der WiiMote. Die Effizienz dieser Mäuse wird in [9] diskutiert. 3D-Mäuse sind eher teuer in der Anschaffung und werden daher im Business-Bereich für CAD-Software verwendet. In der Regel wird der Einsatz einer 3D-Maus in der einen Hand durch eine konventionelle Maus in der anderen Hand ergänzt.

Auch hier sind viele verschiedene Designmöglichkeiten zu bewerten und einzustellen. Es stellt sich die Frage ob es besser ist, das Zoomen durch drücken und ziehen oder besser durch drehen der 3D-Maus, oder über das Scrollrad der konventionellen Maus zu steuern. Ein weiterer Parameter ist die Beschleunigung, und ob diese nichtlinear erfolgen könnte. Diese Fragen könnte man mit meinem Tool klären.

Abbildung 31. 3D-Maus von 3dconnexion

14. http://www.3dconnexion.de/

Auch könnte meine Arbeit im eher neuen Bereich der Bewegungssteuerung sehr nützlich sein. Besonders Mircosoft's Kinect[15] (Abbildung 32), wäre dabei eine mögliches Anwendungsgebiet. Kinect besteht aus mehreren Kameras, die den Spieler dreidimensional erfassen. Die dahinterliegende Logik kann daraus dann seine Bewegungen erkennen und ein kompatibles Spiel damit steuern. Kinect ist ausschließlich für die Xbox verfügbar. Daher ist auch nicht bekannt, wie genau die Interaktion mit der Xbox und der Spielsoftware stattfindet. Eine Verwendung an PC und damit auch mit meinem Programm ist daher mit Kinect nicht möglich. Ähnliche Geräte wie Kinect könnten allerdings in der Zukunft mit der PC-Plattform und meinen Implementierungen kompatibel sein.

Bei der Implementierung könnte das Gestenmodul verwendet werden. Eine Möglichkeit wäre z.B. ein InputModul, dass die Bilddaten der Kameras ausgibt. Ein weiteres Modul erkennt daraus die Position und Körperhaltung des Spielers. Letztlich das Gestenmodul vergleicht die Haltungen der letzten Frames mit Vorlagen aus einer Datenbank und gibt die wahrscheinlichste erkannte Bewegung aus.

Abbildung 32. Microsoft Kinect

Attentive UI

Für Forschung und auch das DFKI ist wahrscheinlich der Einsatz im Bereich der „Attentive User Interfaces“ interessant. Dabei geht es darum, die Aufmerksamkeit des Benutzers zu steuern, in dem das System entscheidet, wann Warnmeldungen ausgegeben werden, welche Art von Meldung, und in welcher Form. Sie spielen besonders im Bereich der Automotive IUI eine große Rolle. In diesem Bereich stellt sich nämlich beispielsweise die Frage, welche Verkehrszeichen und -meldungen wichtig sind und dem Fahrer gemeldet werden müssen, und welche vernachlässigt werden können. Dabei kommt es auch auf den Fahrer an sich an, z.B. in welcher Stimmung er sich gerade befindet. Ein Fahrer der unter Zeitdruck ist, wird sich beispielsweise nicht für einen Aussichtspunkt interessieren. Jemand, der nur auf einem Ausflug ist schon. Diese [11] Master-Arbeit, die gerade in Bearbeitung ist, behandelt das Thema im Detail.

In diesem Fall könnte mein Programm die Verarbeitung der rohen Sensordaten vornehmen, z.B. welche Hautoberflächenspannung mit welchem Puls als Anspannung gewertet wird. Die Implementierung könnte dann so vorgenommen werden, dass es für jede Emotion ein Modul gibt, das an seinem Ausgangspin die momentane Stärke dieser Emotion angibt und die Sensordaten als Input nimmt. Danach könnte ein Modul z.B. eine Emotion mit der höchsten Stärke auswählen, oder zwei Emotionen, oder eine komplexere Berechnung mit den erhaltenen Werten durchführen. Durch dieses modulare Design kann wieder sehr schnell ausprobiert werden, welche Kombination von welchen Emotionen für die eine oder andere Anzeigemöglichkeit am besten ist. In diesem Umfeld gibt es äußerst viele verschiedene Designmöglichkeiten, die ausprobiert werden müssen, wodurch mein Tool hier einen großen Nutzen bringen sollte.

15. http://www.xbox.com/de-DE/kinect

Tactile UI

Ein anderes mögliches Einsatzgebiet wäre im Bereich der Tactile User Interfaces, wie es beispielsweise mit dem EPOC Neuroheadset von Emotiv[16] möglich ist. Abbildung 33 zeigt ein solches Gerät. Das Headset empfängt und verarbeitet Neurosignale und versucht dadurch, die Gedanken und den Gefühlszustand des Benutzers zu bestimmten. Eine Anwendung für diese Hardware wird in [16] diskutiert. Dort wird versucht, per Gedankensteuerung die nativen Funktionen eines iPhone zu kontrollieren.

In meiner Implementierung wären hier Input-Module für die verschiedenen Sensoren denkbar. Dahinter liegende Module könnten diesen rohen Datenstrom dann weiterverarbeiten. So könnte es z.B. Movement-Module geben, die aus den Datenströmen Bewegungen herausfiltern, insofern welche vorhanden sind. Auch hier wäre die Entwicklung eines Projektes mit meinem Tool einfacher und transparenter. Ein Großteil der teilweise komplexen Mustererkennungen in den Datenströmen kann hier auf das Inputsystem in einzelne Module ausgelagert werden. Dies macht zum einen das Debugging einfacher, da die Berechnungen für die jeweiligen Befehle in den Modulen abgekapselt sind und somit der Fehler in einem großen System sehr schnell eingegrenzt werden kann. Zum anderen wird dadurch auch das teamorientierte Design erleichtert, weil das grobe Konzept des ganzen Systems direkt durch das Datenflussdiagramm ersichtlich ist.

Abbildung 33. EPOC Neuroheadset

16. http://www.emotiv.com/apps/epoc/299/

6.4 Zusammenfassung

Meine Arbeit und die dazugehörige Implementierung erleichtert das Design von Multitouch-Inputsystemen. Es besitzt Module, die speziell für den Bereich Multitouch ausgelegt und optimiert sind. Die Praktikabilität meines Programms zeigt sich am FTIR-Display, mit dem zum Beispiel in meiner Demo-Applikation (siehe Kapitel 5.7) einfache geometrische Objekte angelegt und verändert werden können.

Alle gestellten Zeile wurden erreicht. Das Tool ermöglicht eine *schnelle Entwicklung*. Durch das grafische Interface können per Maus Module platziert und miteinander verbunden werden. Dadurch ist es sehr schnell möglich, Inputsysteme aufzubauen. Diese können direkt in einem XML-File abgespeichert, und danach auch ohne grafischen Editor vom Laufzeitsystem gestartet werden. Nach der Erstellung eines Graphen ist nur noch wenig zusätzliche Arbeit notwendig, wie z.B. das Überwachen der gewünschten Module und die Verarbeitung der von ihnen ausgegebenen Werte. Auch ein *einfaches Re-Deisgn* ist möglich. Durch die grafische Bearbeitung und den modularen Aufbau können sehr schnell einzelne Module eingefügt werden, wenn z.B. eine relative statt absolute Steuerung erwünscht ist. Es können alternative Eingabegeräte verwendet werden, indem das entsprechende Input-Modul gegen das des alternativen Gerätes ausgetauscht wird. *Evaluation und Kalibration* wird ebenfalls durch den modularen Aufbau unterstützt. Gain-Faktoren, die hier durch ein Const-Modul repräsentiert werden, können sehr schnell geändert werden, indem das entsprechende Const-Modul durch eines mit einer anderen Konstante ersetzt wird. Auch gänzlich verschiedene Design-Möglichkeiten mit verschiedenen Eingabegeräten können schnell modelliert werden, wie das Eingangsbeispiel zeigt. Durch das Laufzeitsystem kann der erstellte Graph direkt interpretiert und gestartet werden. Anhand dessen kann das System besonders gut getestet und bewertet werden. Das Inputsystem kann direkt in eine bereits vorhandene Applikation eingebaut, und dort die Auswirkungen und Praktikabilität verschiedener Designentscheidungen ausgewertet werden. Eine *transparente Kommunikation* wird durch die grafische Bearbeitung des Systems gewährleistet. Durch die Module sieht man direkt, welche Eingabegeräte verwendet und wie die von ihnen generierten Werte weiterverarbeitet werden. Die verbindenden Leitungen ermöglichen es, den Datenfluss vom Eingabegerät durch die Berechnungen und Modifikationen bis hin zum Output, der später im Hauptprogramm verwendet wird, nachzuverfolgen. Solche Datenflusspläne sind zum einen gut geeignet, um mit Kollegen aus dem Fachgebiet über verschiedenen Alternativen zu diskutieren. Allerdings können sie ebenso potentiellen Kunden oder dritten Personen präsentiert werden, um ihnen den vorgesehenen Aufbau eines Produkts zu verdeutlichen.

Auch das Teilziel des Hardwareaufbaus wurde erreicht. Das Display kann auf dem entsprechenden Windows-PC stabil mit TouchFlow verwendet werden. Die Testanwendung lief durchgehend zuverlässig über Zeiträume von bis zu einer Stunde. Im jetzigen Setup ist eine gute Erkennungsrate möglich. Durch den Einsatz von CCV als Erkennungssoftware kann das Tool nicht nur auf diesem Windows-PC, sondern auch auf Linux oder Mac verwendet werden.

Das Programm ist speziell für Multitouch-Systeme konzipiert worden. Allerdings können auch beliebige andere Inputsysteme mit dem Tool modelliert werden. Einige Module für das Abfragen von Maus und Tastatur sind dazu bereits vorhanden. Durch den modularen Aufbau kann nahezu jedes Eingabegerät und jede Modifikation oder Berechnung der daraus resultierenden Daten in einem Modul vorgenommen werden. Wie Abschnitt 6 zeigt, gibt es viele weitere Möglichkeiten, um auch komplexe Operationen in einem Modul zu berechnen. Zwar gibt es manche konzeptuelle Schwächen, allerdings auch einige Vorteile gegenüber den jetzigen Standarts. Die Anwendungsgebiete sind ohnehin weitreichend.

7 Appendix

7.1 Installationsanleitung für verwendete IR-Kamera

Da die Installation und der stabile Betrieb von CCV mit der verwendeten Kamera einige Probleme bereitete, schildere ich hier kurz die wichtigsten Installationsschritte. Für den Versuchsaufbau wurde die Firefly MV von Point Grey Research mit Firewire-Anschluss verwendet. Laut Foreneinträgen besteht die Instabilität ausschließlich bei der Variante mit Firewire anbindung.

Wichtig: *Kamera erst anschließen, sobald die Anleitung ausdrücklich dazu auffordert.*

1. Falls das Gerät bereits installiert wurde (und nicht richtig funktioniert) im Gerätemanager die Kamera deinstallieren, FlyCapture Installer starten, Software vollständig entfernen. Neustart.
2. Neueste FlyCapture SDK von der Website laden, installieren. Nicht die Software von CD verwenden.
3. Neustart
4. Dieser Anleitung beiliegendes Treiberpaket ("pgrcam.zip", auch hier erhältlich) nach "C:/Programme/Point Grey reseach/driver/selfsigned/WindowsXP" entpacken.
5. Erst jetzt Kamera anschließen
6. NICHT die Treiber von CD, SDK oder einer anderen Quelle installieren, sondern die aus dem eben entpackten Treiberpaket.

Ebenso ist es wichtig, mindestens CCV Version 1.3 zu verwenden. Version 1.2 besitzt einen Bug, der zum sporadischen Absturz der Software mit dieser Kamera führt. War die Installation erfolgreich, erkennt CCV beim Start die richtige Auflösung der Kamera. Dies wird während dem Startvorgang in der Konsole angezeigt. Um eine höhere Auflösung zu verwenden, muss diese in der Config-Datei im CCV-Ordner angegeben werden. Ich habe 640x480 eingestellt. Werden als Auflösung Werte wie 999999999 angezeigt, war die Installation nicht erfolgreich. Wiederholen Sie die Anleitung.

Ohne diese speziellen Treiber reagiert CCV nach wenigen Sekunden nicht mehr. Mit dieser Anleitung sollte ein stabiler Betrieb hinweg über mehrere Stunden möglich sein.

Literaturverzeichnis

[1] M. Rehm, N. Bee, E. André. *Wave Like an Egyptian - Accelerometer Based Gesture Recognition for Culture Specific Interactions*. Augsburg University, 2007.

[2] C. De Boor. *A Practical Guide to Splines*. Springer, 2001.

[3] D. Box. *Essential COM*. Addison-Wesley, 1998.

[4] P. Dietz and D. Leigh. *Diamondtouch: a multi-user touch technology*. ACM Symposium on User interface Software and Technology, 2001.

[5] D. Steinberg et al. *EMF: Eclipse Modeling Framework (2nd Edition)*. Addison-Wesley Longman, 2008.

[6] F. Sasangohar et al. *Evaluation of Mouse and Touch Input for a Tabletop Display Using Fitts' Reciprocal Tapping Task*. Proceedings of the 53rd Annual Meeting of the Human Factors and Ergonomics Society, Santa Monica, CA, 2009.

[7] M. Kaltenbrunner et al. *TUIO: A Protocol for Table-Top Tangible User Interfaces*. Music Technology Group, IUA, Universitat Pompeu Fabra, Barcelona, Spain, 2005.

[8] R. Gordon. *Essential Jni: Java Native Interface*. Prentice Hall PTR, 1998.

[9] Technology Assessment Group. *The Economic Payback of 3D Mice for CAD Design Engineers*. 3Dconnexion GmbH, 2008.

[10] J. Han. *Low-Cost Multi-Touch Sensing through Frustratet Total Internal Reflection*. Media Research Laboratory, New York University, 2005.

[11] D. Ibrahimi. User-adaptive warning systems: Online assessment of driver stress and driving performance on the example of local danger alerts. DFKI.

[12] S. Jordà, G. Geiger, M. Alonso, M. Kaltenbrunner. *The reacTable: Exploring the Synergy between Live Music Performance and Tabletop Tangible Interfaces*. Music Technology Group, Pompeu Fabra University Ocata, 1, 08003 Barcelona, Spain, 2007.

[13] M. Wright, A. Freed, A. Momeni. *OpenSound Control: State of the Art 2003*. In NIME 03: Proceedings of the 3rd Conference on New Instruments for Musical Expression, Montreal, Canada, 2003.

[14] M. Kipp, Q. Nguyen. Multitouch puppetry: Creating coordinated 3d motion for an articulated arm. DFKI, Saarland University, Saarbrücken, 2010. In: Proceedings of the ACM Interactive Tabletops and Surfaces Conference.

[15] H. Peters. *Hardware and Software Extensions for a FTIR Multi-Touch Interface*. Max-Planck-Institut für Informatik Saarbrücken, Germany, 2008.

[16] A. Campbell, T. Choudhury, S. Hu, H. Lu, M. Mukerjee, K. Mashfiqui, Rabbi Rajeev, D. S. Raizada. *NeuroPhone: Brain-Mobile Phone Interface using a Wireless EEG Headset*. Dartmouth College, Hanover, NH, USA, 2010.

[17] U. Tietze, C. Schenk. *Halbleiter-Schaltungstechnik*. Springer, 2002.

[18] C. Szallies. *On Using the Observer Design Pattern*. Energotec GmbH, Germany, 1997.

[19] T. Moscovich T. Igarashi and J. Forbes Hughes. *Spatial keyframing for performance-driven animation*. In SCA 05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation, pages 107 to 115, New York, NY, USA, 2005. ACM., 2005.

[20] J. Tapken and H. Dierks. *Moby/PLC - Graphical Development of PLC-Automata*. University of Oldenburg, Oldenburg, Germany, 1998.

[21] H. Trætteberg. *Integrating Dialog Modeling and Domain Modeling - the Case of Diamodl and the Eclipse Modeling Framework*. Norwegian University of Science and Technology, Norway, 2008.

[22] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns*. Addison-Wesley Longman, Amsterdam, 1994.

[23] G. Wellenreuter, D. Zastrow. *Automatisieren mit SPS - Theorie und Praxis*. Vieweg, 2008.