HAW HAMBURG

BACHELOR THESIS
Jonas Krukenberg

Greifbare Benutzerinteraktion mit einem Musiklernprogramm

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Jonas Krukenberg

Greifbare Benutzerinteraktion mit einem Musiklernprogramm

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Thomas Lehmann
Zweitgutachter: Prof. Dr. Kai von Luck

Eingereicht am: 18. November 2022

Jonas Krukenberg

Thema der Arbeit

Greifbare Benutzerinteraktion mit einem Musiklernprogramm

Stichworte

Tangible Tabletop Interaction, Human Computer Interaction, Systemdesign, Softwarearchitektur

Kurzzusammenfassung

In dieser Bachelorarbeit wird untersucht, wie mithilfe von realen Objekten eine Interaktion mit einem Musiklernprogramm für eine Steigerung des Lernerfolgs erreicht werden kann. Als Schnittstelle für die Objektinteraktion wird ein Tabletop als eine Ausprägungsform eines Tangible User Interface verwendet. Es erfolgt eine Integration von Informationen über Bewegungen von Objekten auf der Tischoberfläche in eine in diesem Rahmen entwickelte Webanwendung. Das Ziel der Anwendung ist es, einfach musiktheoretische Zusammenhänge der Harmonielehre auf interaktive Weise zu visualisieren. Gleichzeitig ist das Lernprogramm alternativ auch ohne Tabletop und unabhängig vom Betriebssystem benutzbar. Der Entwicklungsprozess vom Entwurf über Implementierung bis zur Qualitätssicherung wird ausführlich erläutert.

Jonas Krukenberg

Title of Thesis

Tangible User Interaction with an Educational Music Software

Keywords

Tangible Tabletop Interaction, Human Computer Interaction, System Design, Software Architecture

Abstract

This bachelor thesis examines how real objects can be used to interact with an educational music program in order to achieve an increase of learning success. As an interface for the object interaction, a tabletop is used as a type of a tangible user interface. Information about the movements of objects on the table surface is being integrated into a web application which will be developed in this context. The main goal of the application is to visualize simple music-theoretical connections of harmony theory in an interactive way. At the same time, the tutorial program can also be alternatively used without a tabletop device and independently of the operating system. The development process from design to implementation to quality assurance is explained in detail.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Der digitale Wandel wird allmählich auch in deutschen Schulen sichtbar: *SMART Board*s lösen Kreidetafeln ab, Schüler lesen Lehrbücher in digitaler Form auf Tablets und Overhead-Projektoren und Röhrenfernseher haben vermehrt das Zeitliche gesegnet. Neben dem einfachen Austausch der Hardware kann seitdem der klassische Unterricht durch dedizierte Lernsoftware an Varianz und Attraktivität bei der jungen Generation Z gewinnen.
Auch der Musikunterricht bleibt von dieser Entwicklung nicht unberührt. Musiksoftware wird im Unterricht gerne für Notation (Aufschreiben von Musiknoten), Audio-Editing, automatische Begleitung (durch eine "virtuelle Band") und Musikliteratur-Training verwendet (Vgl. Xu [21, Kap. 3]).
Es existiert das allgemeine Verständnis, dass die Aktivierung mehrerer Sinne im Rahmen des handlungsorientierten Lernens die Wahrnehmung und das längerfristige Behalten von Informationen fördern. So wird dies auch in Kátai u. a. [13] als ein wichtiger Bestandteil der Wissensvermittlung identifiziert und gleichzeitig werden Verbindungen zu entsprechender Didaktik-Software hergestellt. In Kolb [12] wird ein Modell des Lernzyklus "Piaget's Model of Learning and Cognitive Development" vorgestellt, bei denen *konkrete Erfahrung*, *reflektierende Beobachtung*, *abstrakte Begriffsbildung* und *aktives Experimentieren* für den Lernerfolg von Bedeutung sind. Die Phasen in diesem Lernzyklus können mithilfe von Software unterstützt werden. Ein möglicher Ansatz einer Realisierung soll im Rahmen dieser Bachelorarbeit untersucht und umgesetzt werden.

1.1 Problemstellung

Im klassischen Musikunterricht einer allgemeinbildenden deutschen Schule ist ein großer Praxisanteil bereits nicht ungewöhnlich. Es wird gesungen und auf einfachen Musikinstrumenten gespielt, um den Unterricht lebhafter und unterbewusst musikalische Konzepte einprägsamer zu gestalten. Diese Methodik stößt jedoch an ihre Grenzen, wenn ein Thema sehr theoretisch ist oder bereits Vorerfahrung auf einem Musikinstrument voraussetzt. Die Harmonielehre (siehe Erklärung in Kapitel 2) ist ein Beispiel dafür, da Strukturen und Relationen von Harmonien theoretische Konzepte sind. Diese können zwar gut visualisiert werden, wenn sie jedoch in die Praxis umgesetzt werden, verlieren Lernende schnell den theoretischen Hintergrund und spielen lediglich motorisch die vorgegebenen Akkorde. So heißt es auch in Sikora [18, S.5]: "Das *unbewusste* (intuitive) Ohr braucht man, um Musik zu *machen*. Das *bewusste* (kognitive) Ohr braucht man, um Musik zu *lernen*.". An dieser Stelle kann eine Musiksoftware unterstützen, eine Lernbrücke zwischen Theorie und Praxis zu schaffen. Bestenfalls sollten visuelle Eindrücke, wie die Darstellung von Musikharmonien, an die Möglichkeit der spielerischen Einflussnahme in das Dargestellte gekoppelt werden. Dafür bietet sich unter Anderem der Einsatz von Tangible User Interfaces (TUI) an, mithilfe dessen Möglichkeit von haptischer Erfahrung der Lernerfolg begünstigt wird. Schließlich ermöglichen TUIs eine Mensch-Computer-Interaktion mithilfe greifbarer Objekte, die im Schulkontext in ihrer Bauweise einfach und robust ausgeprägt sein können.

Eine Einschränkung ist allerdings, dass mit der Verwendung von TUIs eine starke technische Abhängigkeit zur Hardware, dem zugehörigen Betriebssystem und den zugrundeliegenden Kommunikationsschnittstellen besteht, da diese oft individuell für ein bestimmtes Produkt entworfen sind und es nur wenige Design- und Schnittstellenstandards gibt. Je nach Größe, Gewicht und Beschaffungsmöglichkeiten der Hardware kann dies sogar dazu führen, dass dessen Nutzung an einen bestimmten Ort gebunden ist. Dies weist implizit auf ein Problem der Skalierbarkeit im ökonomischen und realistischen Sinn dar, wie es auch in Holmquist u. a. [7] thematisiert wird.
Die technische Abhängigkeit schafft zugleich eine Unflexibilität bei der Realisierung neuer Software für das gegebene TUI, da das Programm kompatibel zu dem benötigten Betriebssystem sein muss. Dadurch werden die Möglichkeiten eingeschränkt, ein TUI möglichst vielseitig und dadurch nachhaltig verwenden zu können.

1.2 Ziele

Aus der Problemstellung leiten sich einige konkrete Ziele ab, die im Rahmen dieser Bachelorarbeit erreicht werden sollen.

Es wird ein System entwickelt, dessen Kern ein exemplarisches Lernprogramm für die Harmonielehre beinhaltet. Die Darstellung erfolgt über eine grafische Benutzeroberfläche und die Bedienung soll mit greifbaren, realen Objekten erfolgen, um das haptische Erlebnis zu ermöglichen. Hierfür eignet sich der Einsatz eines Tabletop, mit dem es erlaubt ist, physische Objekte und deren absolute Position und Orientierung auf dem Gerät optisch zu erkennen und gegen andere Objekte zu diskriminieren (Mehr dazu in Abschnitt 1.3). Um die Voraussetzungen an die Hardware für die Bedienung des Programms möglichst gering zu halten, sollen die Objekte kostengünstig und einfach produzierbar sein. Zugleich soll eine universelle Anwendbarkeit die Nachhaltigkeit und allgemeine Verfügbarkeit für zukünftige Applikationen ermöglichen. Als populäres Beispiel gibt es bereits den *Microsoft Surface Dial*[1] als aktives kapazitives Sensor-Objekt, das jedoch nur im Zusammenhang mit proprietären SDKs und Hardware- und Softwareplattformen (Windows) reibungslos verwendbar ist. Somit erfüllt der Dial nicht die genannten Anforderungen insbesondere aufgrund des Beschaffungsaufwands (Kaufpreis) und der Herstellerabhängigkeit und des damit einhergehenden eingeschränkten Anwendungsbereichs. Deshalb sollen die für die Objekterkennung lediglich auf Optik beruhenden Mechanismen eines Tabletops genutzt werden.

Das angemerkte Problem der technischen Abhängigkeit, beim Tabletop sogar der Ortsgebundenheit, verhindert, dass das Musiklernprogramm ohne Verfügbarkeit eines solchen speziellen Tischs beispielsweise zuhause für Hausarbeiten genutzt werden kann. Aus diesem Grund soll die Software statt mit einem TUI alternativ mit einer Computermaus bedient werden können, da davon ausgegangen wird, dass heutzutage nahezu alle Schüler Zugang zu einem Computer oder Laptop haben. Da auf ebendiesen Geräten unter Umständen verschiedene Betriebssysteme installiert sind, sollte das Musiklernprogramm plattformunabhängig, also mindestens auf den drei verbreiteten Betriebssystemen Windows, MacOS und einer Linux-Distribution wie Ubuntu lauffähig sein. Eine weitere zeitgemäße Möglichkeit der alternativen Interaktion besteht in der Verwendung eines touchfähigen Geräts (ergo: Smartphone oder Tablet). Da spezielle Gesten, die sich mit einem traditionellen Cursor nicht realisieren lassen, für diese Anwendung nicht notwendig

[1] Microsoft Surface Dial Produktseite: https://www.microsoft.com/de-de/d/surface-dial/925r551sktgn

sind, ist eine zusätzliche Unterstützung für jene Eingabeart mit geringem Mehraufwand verbunden und daher aufgrund Komfortaspekten aus Nutzersicht vorgesehen.

Damit Ergebnisse dieser Bachelorarbeit für zukünftige Arbeiten verwendet und weiterentwickelt werden können, soll eine Schnittstelle für die Kopplung von benötigten UI Komponenten mit Tabletop-Objekten geschaffen werden. Sie soll die Basisfunktionalitäten der Informationsweitergabe über erkannte Objekte und deren Zustand anbieten. Allerdings beschränken sich diese auf die für das Lernprogramm relevanten Funktionen innerhalb der technischen Randbedingungen.

1.3 Einordnung Tangible User Interfaces

Das Forschungsgebiet der *Human Computer Interaction* (HCI) beschäftigt sich mit Schnittstellen, mit denen der Mensch mit Software interagiert. Sehr etablierte Teilgebiete sind *Graphical User Interfaces* (GUI), Touchscreens, Sprachassistenz und Virtual- und Augmented Reality. In einer weiteren der HCI untergeordneten Teildisziplin werden *Tangible User Interfaces* erforscht. Dabei geht es im Wesentlichen um die Verbindung von der digitalen und der physischen Welt. Eine Ausprägung ist die Interaktion mit Software mithilfe von realen Objekten, die meist mit auf den Anwendungsfall zugeschnittener Hardware wie Sensorik, Aktorik oder Konnektivität ausgestattet sind. Dadurch wird erzielt, den Umgang mit Technologien auf eine natürlicher Weise greifbar zu machen, haptisches Feedback zu erhalten und so Menschen unter Anderem in der Bildung und im künstlerischen Handeln zu unterstützen.

In Rodić und Granić [15] wurden bereits eine Vielzahl von Publikationen bezüglich des Einsatzes von TUIs in der Wissens- und Kompetenzvermittlung für Kinder ausgewertet und Zusammenhänge zwischen der Art der Ausprägung des TUI und der Anwendungsdomäne aufgezeigt. Das Tabletop wird in jener Veröffentlichung als eigenständige Kategorie[2] der Ausprägungsformen von TUIs in der Bildung identifiziert, die in 21 von 155 untersuchten Arbeiten in Bereich wie Kollaboration, Problemlösung oder auch im Künstlerischen zum Einsatz kommen. *Tangible Tabletop Interaction* vereint nach Shaer und Hornecker [17, Kap. 3.1.2] Multitouch-Oberflächen mit TUIs, in der die heutzutage geläufige Interaktion mittels Finger- oder Stiftberührungen inbegriffen ist. Darüber hinaus steigt die Anzahl der Forschungsarbeiten, die sich mit einem Technologie-Mix von Touch und "tangible handles" befassen, an.

[2] Weitere Kategorien sind die der "manipulatives" (Lernspielzeuge) und der Tablets

Entsprechend der Größe des Forschungsgebiets der TUI finden jährlich dedizierte ACM gelistete Konferenzen wie die CHI (Conference on Human Factors in Computing Systems) und TEI (Tangible Embedded and Embodied Interaction) statt. Bereits 1997 wurde in Ishii und Ullmer [8] die Vision der Transformation von "Painted Bits" (im Bereich GUI) zu "Tangible Bits" (im Bereich TUI) prototypisch vorgestellt - ein erster Ansatz des *Tabletop*-Gedankens.

Daran ist zu erkennen, dass es eine für diese Bachelorarbeit stark ausgeprägte Basis mit Forschungsarbeiten im Bereich der TUIs vorhanden ist. Insbesondere können Erkenntnisse bezüglich Tabletop in die Planungs- und Durchführungsphase für das zu entwickelnde System einfließen.

2 Musiktheoretische Grundlagen

Für das zu entwickelnde Musiklernprogramm sind einige Grundkenntnisse in der Anwendungsdomäne der Musiktheorie erforderlich, die in diesem Kapitel in Kurzform festgehalten werden und sich an den einführenden Abschnitten in Sikora [18] orientieren. Hierbei wird sich im Allgemeinen auf Musik der uns bekannten westlichen Kulturen beschränkt. Im Folgenden werden die wesentlichen Elemente der Musik erläutert: Rhythmus, Melodik und Harmonik.

2.1 Rhythmus und Melodik

Der Rhythmus beschreibt die zeitliche Gliederung von Musik wie Tondauern und Pausen. Rhythmen weisen meist sich wiederholenden Strukturen auf und orientieren sich an Takten, einer statischen Einteilung von Zeiten und Betonungen. Ein Rhythmus entsteht oft im Einklang mit einer Melodie (melodischer Rhythmus) oder einer Harmoniefolge (harmonischer Rhythmus), kann aber auch alleinstehend sein.

Die Melodik befasst sich mit der zeitlich aufeinanderfolgenden Sequenz von Tönen, der Melodie. Solch eine Tonfolge kann einen melodischen Rhythmus haben und folgt meist einem zugrundeliegenden harmonischen Kontext. Oft ist es die Melodie, die einem Musikstück einen großen Wiedererkennungswert gibt, weil sie einzigartig wirkt. Demnach lassen sich Konventionen für eine "gute" Melodie aufgrund der großen Varianz und Kontextabhängigkeit nur schwer definieren und setzen oft bereits eine gegebene Harmoniefolge oder einen Rhythmus voraus.

2.2 Harmonik

In der Harmonik geht es um die vertikale Anordnung von Tönen im Notensystem, also dem gleichzeitigen Erklingen von Tönen, die als *Akkord* oder *Harmonie* bezeichnet werden. Auf diesem Fachgebiet gibt es die meisten lern- und lehrbaren Fakten und bietet im Gegensatz zu subjektiven Bereichen wie Sound, Form, Timing etc. viele Konventionen und Regeln, die in Büchern für Harmonielehre analysiert und vermittelt werden.

Die **Funktionsharmonik** ist ein Teilgebiet der Harmonik und beschäftigt sich mit den Zusammenhängen und Beziehungen von Tönen und Harmonien zueinander, dessen Erkenntnisse auch als Leitfaden für die Komposition eines neuen Musikstücks dienen können. Der Abstand zwischen zwei Tönen wird als Intervall (z.B. Terz, Quarte, Quinte) bezeichnet. Entsprechend stellt der Quintenzirkel (siehe Abbildung 2.1) Quintenverwandschaften dar, die ein sehr zentrales Beziehungsprinzip in der Musik sind. Aus ihm lassen sich unter anderem *Kadenzen* ableiten, die Akkordverbindungen beschreiben, die einem Spanungsverlauf folgen und das Bestreben haben, sich in einen Zielakkord aufzulösen. Kadenzen finden sich in nahezu jeder Komposition der Klassik- und Popularmusik, funktionieren in jeder Tonart. Eine Tonart bezeichnet das tonale Zentrum, ein absoluter Bezugspunkt für relative harmonische Zusammenhänge und setzt sich zusammen aus einem Grundton (C, Db, D etc.) und dem Tongeschlecht (Dur oder Moll).

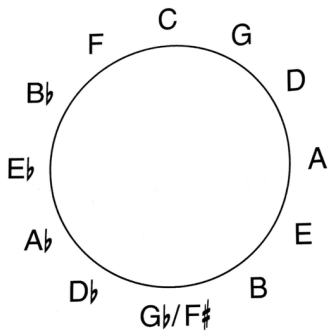


Abbildung 2.1: Der Dur-Quintenzirkel (Sikora [18, S.26]) in internationaler Schreibweise (deutsch: H = internat.: B)

Konkrete Akkordfolgen wie beispielsweise $C - Am - Dm - G - C$ in C-Dur lassen sich funktionsharmonisch mit $I - VIm - IIm - V - I$ beschreiben. Der Vorteil hierbei ist, dass es auf diese Weise möglich ist, Progressionen abstrakt und für jede Tonart allgemeingültig formulieren zu können. Schließlich beziehen sich harmonische Zusammenhänge stets auf

relative Verhältnisse von Akkorden und sind vollkommen unabhängig von der absoluten Grundstimmung.

Grundsätzliche Regeln für einfache Kadenzen und die allgemeine Notation der Funktionsharmonik können sich gut im Rahmen einer Musiksoftware zunutze gemacht werden, um Zusammenhänge abstrakt definieren zu können, deren konkrete Realisierung (Tonart und Akkordbezeichnung mit Grundton und optionalen Intervallen) zur Laufzeit eines Programms abhängig von Faktoren wie Benutzereingaben entschieden werden kann.

3 Entwurf

Um ein nachvollziehbar funktionierendes und qualitatives System für die Erfüllung der Ziele aus Abschnitt 1.2 zu entwickeln, bedarf es des Entwurfs einer Gesamtarchitektur.

Dafür wird zunächst ein Konzept der Gesamtarchitektur vorgestellt, in dem allgemein erforderliche Strukturen adressiert werden. Darauf aufbauend werden technische Randbedingungen vorgestellt, die Constraints und technische Entscheidungen für Fragmente im Kontext der Gesamtarchitektur festlegen. Erst mit den Voraussetzungen der konzeptuellen Architektur und den technischen Randbedingungen kann anschließend die Realisierung der Gesamtarchitektur in Hinblick auf die verwendeten Technologien und deren Kernaufgaben vorgestellt werden.

3.1 Konzept der Gesamtarchitektur

Die Architektur des zu entwerfenden Systems soll allgemeine nichtfunktionale Anforderungen wie Wartbarkeit, Weiterentwicklungsfähigkeit, Portabilität und Wiederverwendbarkeit und Plattformunabhängigkeit erfüllen. Dies kann im Wesentlichen erreicht werden, indem logische Komponenten mit zugewiesenen Verantwortlichkeiten eingeführt und in logischen Abstraktionsschichten getrennt werden. Zusätzlich wird das System zugunsten der Flexibilität in zwei Tiers (physikalisch voneinander unabhängige Architekturbausteine) aufgeteilt, um die generische Verarbeitung von Hardware-Input von der individuellen anwendungsspezifischen Benutzeranwendung zu trennen. Auf ein drittes Tier, das eine Trennung von Fassade (ergo UI) und Logik im Sinne einer Aufteilung in Frontend und Backend ermöglicht, soll hier verzichtet werden. Schließlich sollen Latenzen möglichst gering gehalten werden und nicht zuletzt braucht es oft Kontext und Zustand der UI, um eine korrekte Interpretation von Benutzereingaben zu erreichen.

Es wird also logisch betrachtet eine *layered architecture* (vgl. Tanenbaum und van Steen [19, Kap. 2.1]) und physikalisch eine *two-tiered architecture* (vgl. Tanenbaum und van

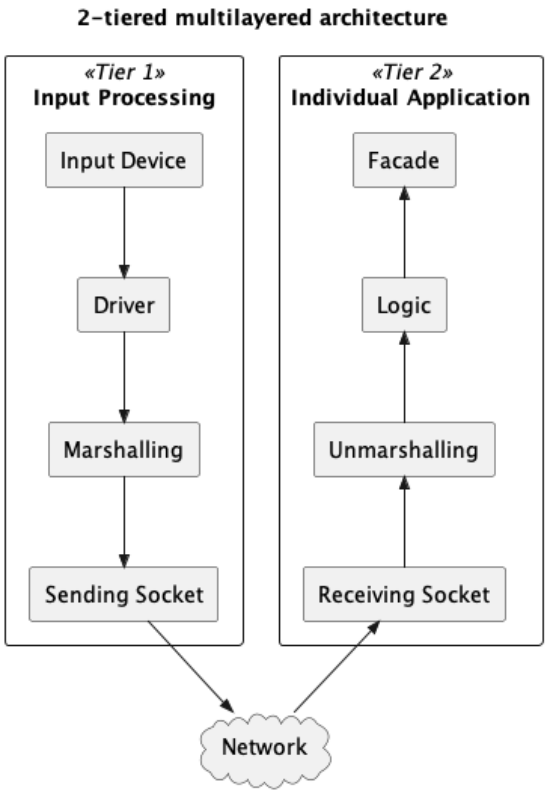


Abbildung 3.1: Schematische Darstellung einer physikalischen 2-tiered und logischen layered architecture für eine Tabletop-Anwendung. Die Pfeile weisen auf die Datenflussrichtung hin.

Steen [19, Kap. 2.2]) für das System angestrebt. Für die Kommunikation zwischen beiden Tiers soll eine Netzwerkverbindung genutzt werden, damit sie zum Beispiel für Entwicklungsarbeiten getrennt voneinander auf verschiedenen Geräten ohne eine physikalische Verbindung (Kabel) betrieben werden können. In Abbildung 3.1 wird entsprechend die Idee der Architektur für eine Tabletop-Anwendung schematisch dargestellt.

Offensichtlich sind die vorliegenden Charakteristiken des Systems bezogen auf die Eigenschaft, Eingaben zu verarbeiten, interpretieren und auf einer Benutzeroberfläche Reaktionen darzustellen nicht einzigartig. Florian Echtler, der sich unter anderem auch mit Tabletop Technologien beschäftigt hat, stellt in Echtler und Klinker [6] eine Softwarearchitektur vor, die eine Verbindung zwischen einerseits verschiedenen Eingabegeräten und andererseits unterschiedlichen grafischen Werkzeugkästen vorsieht. Die in der vorgestellten Architektur enthaltenen Schichten reichen vom Eingabegerät über die Rohda-

tenverarbeitung und Kalibrierung von erfassten Positionen bis hin zur Erzeugung und Verwertung anwendungsrelevanter Informationen über die interpretierten Gesten (vgl. Abbildung 3.2). Wenngleich sich jenes Konzept auf *Multitouch*-Schnittstellen bezieht, lässt sich die Struktur gut als Referenzarchitektur für das System in dieser Bachelorarbeit anwenden, da ähnliche Aufgaben für die Datenverarbeitung vorliegen.

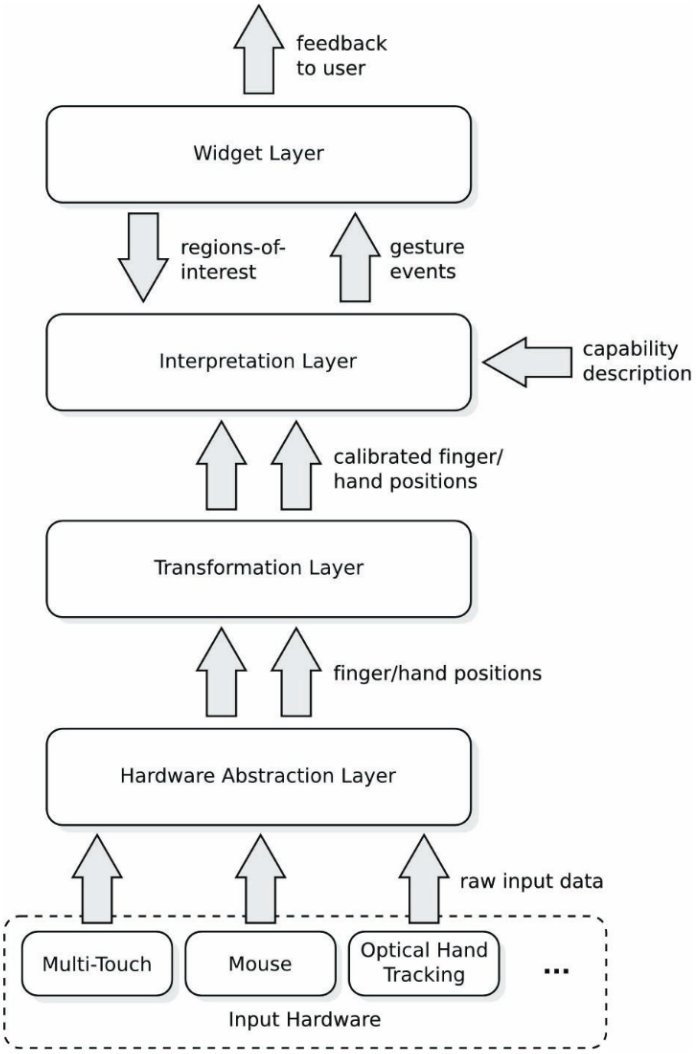


Abbildung 3.2: Multitouch Software Architecture, Echtler und Klinker [6], Kaltenbrunner [9, Kap. 8.2]

3.2 Technologische Randbedingungen

3.2.1 Tabletop

Die Auswahl eines Endgeräts für die *Tangible Tabletop Interaction* ist aus Zeit- und Kostengründen durch die an der HAW vorhandenen Mittel eingeschränkt. Grundsätzlich stehen drei Lösungen zur Verfügung:

1. Der "Küchentresen" im Living Place ist ein aus frei zusammengestellten Modulen bestehendes System. Mithilfe von Kameras unterhalb einer Platte aus Milchglas können Strukturen auf dessen Oberfläche bildlich erfasst und auf einem externen Computer ausgewertet werden. Die grafische Oberfläche wird mittels Videoprojektion auf die Unterseite des durchscheinenden Küchentresens realisiert. Der große Vorteil eines solchen Systems ist seine Modularität und Offenheit (vgl. "Openess" in Tanenbaum und van Steen [19, Kap. 1.2, Being open]). Dadurch ließe es sich sehr gut als verteiltes System betreiben, um beispielsweise die rechenaufwändige Bildverarbeitung (*image processing*) auf einem separaten Hochleistungsrechner durchzuführen. Andererseits befindet sich der Küchentresen zum Zeitpunkt der Konzeption in einem nicht betriebsfähigen Zustand, wodurch ein unverhältnismäßig großer Einrichtungsaufwand bezüglich zu beschaffener und installierender Hardware entstünde, welcher außerhalb des hiesigen fachlichen Fokus läge und den zeitlichen Rahmen der Bachelorarbeit sprengte.

2. Das CSTI[1] besitzt einen Wandmonitor mit einem Infrarot-Touchscreen, mithilfe dessen Finger und Objekte anhand resultierender Störungen im optischen Raster lokalisiert werden können (siehe auch Bhalla und Bhalla [3, Kap. 4]). Diese Technologie ist jedoch eher für die Erkennung von Finger- und Stiftpositionen geeignet, da Objekte nur anhand ihrer Formfaktoren unterschieden werden können und dies die Anzahl verwendbarer Elemente und damit die Skalierbarkeit und Flexibilität einer Applikation mit Objektinteraktion deutlich einschränkt.

3. Im Living Place gibt es zwei *Samsung SUR40 with Microsoft PixelSense* (kurz: SUR40), ursprünglich *Microsoft Surface 2.0*[2], zu sehen in Abbildung 3.3. Die Technologie basiert auf optischen Sensoren an den Pixeln des Bildschirms, wodurch vergleichbar mit einem Kamerabild eine Aufnahme des Bereichs unmittelbar über

[1] Creative Space for Technical Innovations: https://csti.haw-hamburg.de/
[2] Microsoft PixelSense: https://en.wikipedia.org/wiki/Microsoft_PixelSense (sämtliche aussagekräftige offizielle Quellen wurden von Microsoft offline gestellt)

Abbildung 3.3: SUR40 im HAW Living Place

der Oberfläche ermöglicht wird. Im Gegensatz zu kapazitiven Touchscreens ist es dadurch auch hier möglich, neben Fingern auch Objekte anhand visueller Charakteristiken wie Form und Grautönen zu identifizieren. Da das SUR40 ein Bestandteil der Produktpalette der großen IT-Unternehmen Microsoft und Samsung ist, muss man bei der Entscheidung für diese Lösung berücksichtigen, dass hier nicht alle Komponenten frei einsehbar und modifizierbar sind. Ebenso muss bei der Entwicklung von nativen Applikationen für das herstellerseitig unterstützte Betriebssystem Windows 7 das dafür bereitgestellte *Microsoft Surface SDK 2.0* verwendet werden, dessen Dokumentation aufgrund des Alters und herstellerseitiger Einstellung des Supports sehr lückenhaft ist. Gleichzeitig ist der Entwickler aufgrund des SDK darauf angewiesen, die vorgegebene Programmiersprache (C-Sharp) zu verwenden, welches die applikationszentrische Entscheidungsfreiheit erneut einschränkt.

Trotz der Nachteile des SUR40 aufgrund seiner proprietären Technologie bietet es im Vergleich zu den ersten beiden Alternativen hardwaretechnisch die besten Voraussetzungen, da es sich bereits um ein geschlossenes und funktionierendes System handelt, das mehrere gleichartige Objekte optisch voneinander unterscheiden kann. Da sich diese Arbeit letztendlich auf die Integration eines Softwaresystems fokussieren soll und die Hardware des SUR40 bereits als gegeben und im Vergleich am robustesten betrachtet werden kann, kann sich darauf mit geringstem Projektrisiko gestützt werden.

3.2.2 Betriebssystem

Wie bereits erwähnt, wurde die Unterstützung und Weiterentwicklung des SUR40 mitsamt des Microsoft Surface SDK 2.0 herstellerseitig eingestellt und des Weiteren wurden allgemein hohe Latenzen bei der Reaktion auf Touch-Eingaben von 153ms (± 10ms) in Echtler und Kaltenbrunner [5] gemessen. Außerdem existiert das bekannte Problem vieler False-Positive Reaktionen aufgrund einer sehr hohen Sensitivität bezüglich der Beleuchtungsverhältnisse ("Ghostpoints"). Daher ist es zukunftsperspektivisch gesehen nicht nachhaltig, dies als Basis für ein neues Softwaresystem zu verwenden. Vielmehr sollte man das Gerät anstelle von Windows 7 mit einem auf Linux basierendem Betriebssystem wie Ubuntu betreiben, das einen erweiterten Eingriff in Funktionsweise und Datenströme erlaubt. Da in Echtler und Kaltenbrunner [5] genau dies aus ebendiesen Gründen erreicht und in dem Rahmen ein entsprechender Linux Kernel-Treiber[3] entwickelt wurde, darf man hier davon ausgehen, dass das Vorhaben grundsätzlich umsetzbar ist. Obwohl davon auszugehen ist, dass die Einrichtung von Ubuntu als ein nicht ursprünglich für das SUR40 vorgesehenes Betriebssystem bezüglich der Signalverarbeitung von Pixelsensordaten im Vergleich zur Verwendung des bereits installierten Windows 7 nicht frei von Komplikationen sein wird, soll dieser einmalige Aufwand aufgebracht werden, um den Installationsaufwand für Folgeprojekte möglichst gering zu halten.

Damit es im Rahmen anderer Projekte weiterhin möglich ist, die vom Hersteller für Windows 7 bereitgestellten Programme und das SDK nutzen zu können, wird die Installation von Ubuntu auf einer neuen SSD Festplatte separat erfolgen und der Betrieb als Dual-Boot-System ermöglicht. Außerdem wird der Arbeitsspeicher des SUR40 bei dieser Gelegenheit von 4GB auf 8GB aufgerüstet. Beide Maßnahmen sollen Latenzen bei der Bedienung des Geräts reduzieren, da diese einen starken Einfluss auf die User Experience haben, wie beispielsweise in Anderson u. a. [1] belegt wird.

3.2.3 Bildverarbeitung

Da auf die Verwendung von Windows 7 verzichtet wird, muss eine alternative Lösung für die Analyse und Interpretation des Kamerabilds der Bildschirmoberfläche gefunden werden. Dafür wird in Echtler und Kaltenbrunner [5] reacTIVision als etabliertes Framework für auf Optik basierende Interaktion für Multitouch und Objekterkennung verwendet. In

[3] SUR40 Kernel-Treiber: https://github.com/torvalds/linux/blob/master/drivers/input/touchscreen/sur40.c

Abbildung 3.4 wird die Aufgabe von reacTIVision schematisch gezeigt. Die abgebildete Kamera wird in diesem Fall durch die Pixelsensoren und der Projektor durch den Bildschirm des SUR40 realisiert.

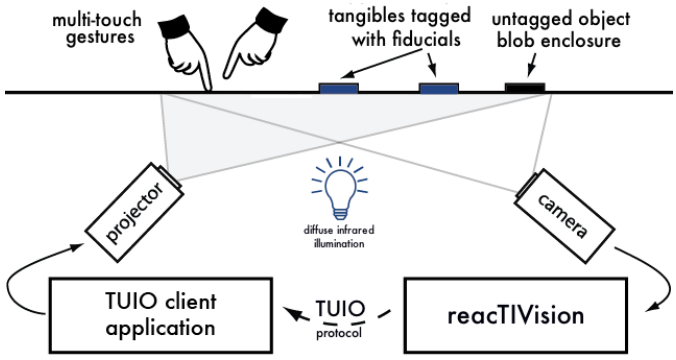


Abbildung 3.4: reacTIVision Framework Architektur aus Kaltenbrunner und Bencina [10]

TUIO Das in Abbildung 3.4 erwähnte TUIO-Protokoll (TUIO 1.1: "A Protocol for Table-Top Tangible User Interfaces", Kaltenbrunner u. a. [11]) ist als De-Facto-Standard das Nachrichtenformat für die Weitergabe von Informationen über erkannte Fingerspitzen und Objekte. Es basiert auf der OSC Spezifikation[4], die Vorgaben zu Größe und Format eines *OSC Packet* (Presentation Layer im OSI Referenzmodell nach Zimmermann [22, Kap. 7]) und dessen enthaltener *OSC Message* oder *OSC Bundle*. Eine TUIO-Nachricht enthält, je nach Klasse, Informationen über den Objektzustand bestehend aus beispielsweise Position und Orientierung (Klasse *set messages*) oder über die aktuell auf der Bildschirmoberfläche erkannten Objekte (Klasse *alive messages*). Mehrere miteinander zusammenhängende TUIO-Nachrichten werden in *frames* mithilfe von *fseq*-Nachrichten (*fseq* für "frame sequence ID") gruppiert. Das Protokoll ist zustandslos, enthält also keine expliziten Informationen über einmalige Vorkommnisse wie das Hinzufügen oder Entfernen eines Objekts, damit mögliche Paketverluste weniger kritisch sind. Ein *TUIO Bundle* besteht aus mehreren TUIO-Nachrichten und hat nach Kaltenbrunner u. a. [11][5]

[4] OpenSoundControl: https://opensoundcontrol.stanford.edu/spec-1_0.html
[5] siehe auch TUIO 1.1 Spezifikation: https://www.tuio.org/?specification

eine Struktur, wie sie im Anhang unter Abschnitt A.1 beschrieben wird. Da das TUIO-Protokoll unter anderem die Objekterkennung unterstützt, dediziert für Tangible Interaction entwickelt wurde und ein standardisiertes Nachrichtenformat die Interoperabilität eines Systems begünstigt, soll dieses auch im Rahmen dieser Arbeit Anwendung finden.

Fiducial Marker Zur optischen Identifizierung eines Objekts wird sich für eine höhere Zuverlässigkeit an einem Hilfsmittel, dem fiducial marker, bedient. Dabei handelt es sich um eine klar definierte, meist strukturierte, bildliche Codierung einer ID. Schließlich unterstützt reacTIVision neben der einfachen auf Formfaktoren basierten *blob object detection* und *finger touch point detection* auch die Erkennung von *Fiducials*. Standardmäßig werden hier sogenannte *amoeba symbols* verwendet, die schwarze und weiße Regionen aufweisen. Anhand adjazenter Regionen lässt sich ein Baum erzeugen, dessen Struktur als eindeutiges Merkmal auf die Fiducial-ID schließen lässt. In Abbildung 3.5 werden drei exemplarische Topologien und deren Adjazenzgraphen und es wird anhand der beiden Graphen b und c mit identischen Graphen deutlich, dass geometrische Formen des Symbols keinen Einfluss auf den erzeugten Baum haben.
Die Position und Orientierung eines Symbols wird mithilfe des berechneten Mittelpunkts von allen Blattknoten und des Vektors von dort zum Mittelpunkt aller schwarzen Blattknoten festgestellt, so wie es in Abbildung 3.6 für ein amoeba Symbol gezeigt wird.
Die amoeba Symbole werden aufgrund der genannten Irrelevanz der Geometrie mit-

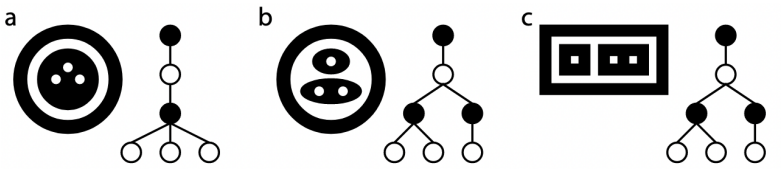


Abbildung 3.5: Schematische Adjazenzgraphen für fiducial markers aus Bencina und Kaltenbrunner [2]

tels eines genetischen Algorithmus erzeugt und erscheinen daher tendenziell organisch und ästhetisch. Für eine möglichst geringe Fehlerquote in der Erkennung der Symbole, ist es allerdings notwendig, dass die Fiducials einen Durchmesser von fünf bis sechs Zentimeter haben. Aus dem weiteren Grund, dass die steigende Komplexität der Adjazenzgraphen bei Bedarf an mehr amoeba Symbolen die Performance spürbar beeinflusst, wurde in Kaltenbrunner [9] mit *Yamaarashi* Symbolen eine weiterentwickelte Struktur vorgestellt, exemplarisch in Abbildung 3.7 dargestellt. Hier ist es möglich, mit einer 20-Bitfolge 2^{20} verschiedene IDs zu erhalten, ohne auf die effiziente Methode zur Ermittlung

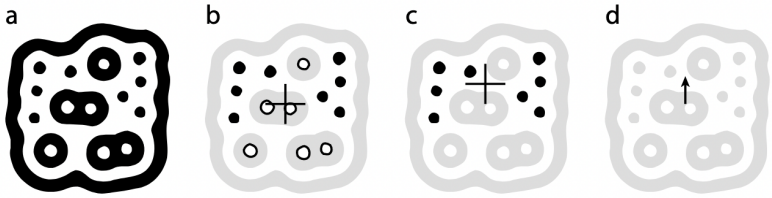


Abbildung 3.6: a) ein amoeba fiducial symbol, b) Mittelpunkt aller weißen und schwarzen Blattknoten, c) Mittelpunkt aller schwarzen Blätter,d) Der resultierende Vektor für die Bestimmung der Orientierung; aus Bencina und Kaltenbrunner [2]

der Orientierung mittels eines einfachen amoeba Symbols in der Mitte zu verzichten. Die vereinfachte Struktur erlaubt es, jene Symbole kleiner (vier Zentimeter) und dennoch effizienter zu detektieren. Da außerdem die kreisrunde Form ästhetisch besser zu den für das Musiklernprogramm angedachten runden Objekten passt, soll also mit Yamaarashi Symbolen gearbeitet werden.

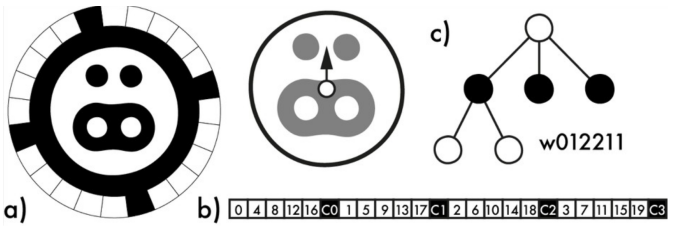


Abbildung 3.7: a) Yamaarashi Symbol, b) Binärcodefolge; c) amoeba Kernsymbol; aus Kaltenbrunner [9]

Bewertung reacTIVision Abschließend stellt reacTIVision eine geeignete Engine für die Bildverarbeitung dar, da die geforderte Erkennung von Objekten anhand von Fiducials sehr effizient und dadurch latenzarm erfolgt. Weiterhin ist die Engine mit diversen Betriebssystemen, insbesondere Windows 7 und Ubuntu, kompatibel[6] und dadurch für den Einsatz auf dem SUR40 für den Fall, dass eine Ubuntu-Installation nicht möglich ist, geeignet. Die Plattformunabhängigkeit von reacTIVision rührt daher, dass im Gegensatz zu vielen Computer Vision Frameworks keine GPU-Prozessierung (etwa mit OpenCL) zur Steigerung der Performance genutzt wird, sondern der Code in Standard C++ geschrieben ist (vgl. Kaltenbrunner [9, Kap. 7.3.6]). Prinzipiell ist es jedoch durchaus

[6] Gemäß http://reactivision.sourceforge.net/

möglich, alternative TUIO-konforme Softwarelösungen zu wählen. So wirbt *BullsEye*[7] mit einem präziseren Tracking von Fiducials, ist allerdings nur für Windows verfügbar und nicht für das SUR40 erprobt. Eine weitere Möglichkeit wäre es, *CCV* (Community Core Vision)[8] zu nutzen, allerdings kann diese Engine TUIO-Nachrichten nicht über das WebSocket-Protokoll übertragen. Aus diesen Gründen, und nicht zuletzt weil es bereits den Proof-Of-Concept für die Kompatibilität mit dem SUR40 gibt (siehe Echtler und Kaltenbrunner [5]), soll reacTIVision also als Engine für die Bildverarbeitung verwendet werden.

3.3 Realisierung der Gesamtarchitektur

Da nun die technischen Randbedingungen gegeben sind, lassen sich diese in den Entwurf einer Systemarchitektur als eine Implementierung der Referenzarchitektur aus der technischen Perspektive einfließen. Diese ist in Abbildung 3.8 zu sehen. Die informelle Syntax der gezeigten Grafik orientiert sich an jener aus der Literatur, um die schematische Äquivalenz zu verdeutlichen. Zwecks eines besseren Verständnisses im Folgetext werden die konkreten Namen der verwendeten Komponenten wie SUR40, Ubuntu und reacTIVision verwendet, da sich an dieser Stelle bereits für die Verwendung ebendieser entschieden wurde. Zugleich visualisiert die Abbildung den grundsätzlichen Datenfluss für die Verarbeitung von Benutzereingaben des Tabletop Geräts (SUR40).

Die Kamera-Rohdaten des SUR40 werden mit dem bereits in Unterabschnitt 3.4.2 erwähnten SUR40 Linux Kernel-Treiber auf Byte-Ebene strukturiert und für die Weiterverarbeitung als USB-Eingabegerät auf Betriebssystemebene provisioniert.

Der Datenstrom dieses Geräts kann von der *reacTIVision vision engine* gelesen und in TUIO-Nachrichten, die Informationen über Finger-, Fiducial- und Objektzustände enthalten, transformiert werden.

In diesem Anwendungsfall werden jene Nachrichten an die TuioSupport-Komponente der Frontend-Anwendung geschickt, die TUIO-Nachrichten interpretiert und zu für die Anwendung relevanten Events transformiert. Für die Übertragung der TUIO-Nachrichten in diese Systemschicht wurde sich für das WebSocket Protokoll entschieden, weil dadurch die erforderliche Echtzeitkommunikation gegeben ist und ressourceneffizient nach dem

[7] BullsEye: http://bullseye.projects.cavi.au.dk/
[8] CCV: http://ccv.nuigroup.com

Push-Prinzip realisiert werden kann. Wenngleich hier keine bidirektionale Kommunikation notwendig ist, würde ein alternativer unidirektionaler HTTP Polling-Mechanismus einen zu großen netzwerk- und performancetechnischen Overhead bedeuten, da zu vielen Zeitpunkten keine neuen Objektinformationen verfügbar sind und die Abfragefrequenz für ein Echtzeitverhalten sehr hoch sein müsste. Ein Nachteil der Verwendung von WebSockets ist, dass eine unbeabsichtigte Unterbrechung der zugrundeliegenden persistenten TCP Verbindung zu einem Abbruch der Kommunikation führt. Dieses Szenario muss programmatisch mit einhergehender erhöhter Anwendungskomplexität im Gegensatz zum Polling-Mechanismus durch manuelle Neuverbindungen oder den Einsatz geeigneter Bibliotheken wie *Socket.IO*[9] gelöst werden. Beim Betrieb beider Tiers auf demselben Gerät (SUR40) ist jedoch nicht mit Ausfällen der lokalen Netzwerkverbindung zu rechnen. Es überwiegen also die genannten Vorteile der WebSocket-Lösung für das gegebene Anwendungsszenario.

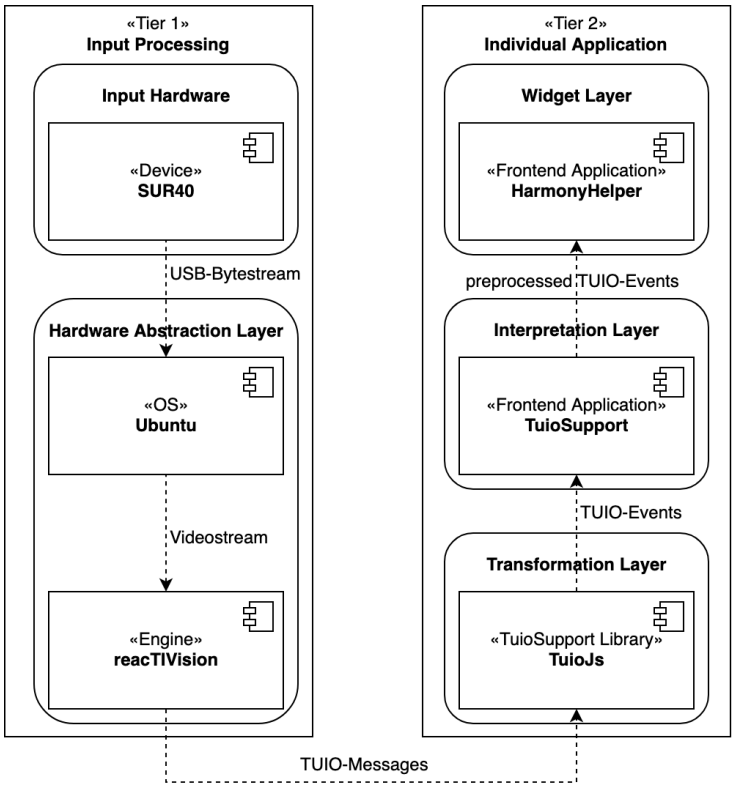


Abbildung 3.8: Systemarchitektur nach Referenzarchitektur in Abbildung 3.2

[9] Socket.IO ist eine Bibliothek für eine eventbasierte, bidirektionale und latenzarme Echtzeitkommunikation: https://socket.io/

3.4 Entwicklung des Musiklernprogramms

Für die Kommunikation und Dokumentation der zu entwerfenden Architektur wurde sich dafür entschieden, zwecks Übersichtlichkeit und Konformität Vorschläge für Kapitel und Darstellungsformen aus dem arc42 Template[10] zu entnehmen.

3.4.1 Ziele

Das Musiklernprogramm wird im Stil der agilen Softwareentwicklung entworfen und realisiert. Demnach ist es für eine inkrementelle Vorgehensweise von Vorteil, Ziele und Aufgaben in kleine User Stories zu formulieren und auf einer User Story Map zu kategorisieren und priorisieren, die in Abbildung A.1 zu sehen ist. Es wird sich hierbei auf essentielle Funktionalitäten beschränkt, die für die Grundidee der Visualisierung von harmonischen Zusammenhängen bei Interaktion mit (TUIO Modus) und ohne (Non-TUIO Modus) einem Tabletop notwendig sind. Als Kernfunktionalität des Programms soll es dem Anwender möglich sein, eine beliebige Harmoniefolge zusammenzustellen und begleitend Vorschläge für wohlklingende Folgeakkorde (im Rahmen gegebener funktionsharmonischer Regeln) zu erhalten.

Der Entwurf für die grafische Benutzeroberfläche hat *Mapping Tonal Harmony Pro*[11] als Vorbild für das Layout der Darstellung harmonischer Funktionen (Tonika, Subdominante, Dominante) und wird in einem Wireframe in Abbildung A.2 dargestellt. Hierbei wurden besondere Eigenschaften der Ansicht im TUIO- und Non-TUIO Modus in einem gemeinsamen Wireframe zusammengefasst. Demnach sind die gekennzeichneten "TUIO Objekte" und "Toolbar Buttons" jeweils im gegenseitige Ausschluss aktiv. Außerdem soll die alternative Non-TUIO Bedienung für die Harmonieauswahl (a2) per Mausklick bzw. Touchberührung auf die zur Verfügung stehenden Optionen erfolgen. Die Toolbar Buttons funktionieren beide wie ein toggle switch: Ist der Playbutton (b1) gedrückt, wird an der Stelle ein Stopp-Button angezeigt, sowie der Button für das Tongeschlecht (b2) nach Aktivierung "Moll" anzeigt und eine Möglichkeit des Zurückschaltens nach "Dur" bietet.

[10] arc42: https://www.arc42.de/
[11] Mapping Tonal Harmony Pro: https://mdecks.com/mapharmony.phtml

3.4.2 Technische Randbedingungen

Da für das Lernprogramm eine sehr individuell für den Anwendungsfall gestaltete grafische Benutzeroberfläche erhalten soll, muss die verwendete Programmierumgebung dies gut ermöglichen. Dafür könnte beispielsweise ein GUI-Toolkit von Java wie *Swing* verwendet werden. Jedoch ist es naheliegender, sich anstatt einer Lösung mit modularen UI-Funktionalitäten direkt in einer Welt der Visualisierung von Informationen zu bewegen. An dieser Stelle stellen sich Technologien für den Web-Browser wie HTML, CSS und JavaScript als dedizierte Mittel für Benutzerinteraktion als für diese Aufgabe geeignet dar. Außerdem ist zu erwarten, dass es für den Bereich eine große Auswahl an Bibliotheken und Frameworks gibt, die hier von Nutzen sind.

Aus den dargestellten Zielen in Abschnitt 1.2 geht hervor, dass das Lernprogramm alternativ von Schülern daheim bedient werden können soll, die gegebenenfalls unterschiedliche Betriebssysteme nutzen. Des Weiteren ist anzustreben, dass der Installationsaufwand für die Verwendung möglichst gering sein soll, um die Hemmschwelle der tatsächlichen Nutzung klein zu halten.

Es stehen drei verschiedene Ausprägungsformen zur Auswahl, in denen die Software bereitgestellt werden könnte, die den Cross-Platform-Ansatz verfolgen und allesamt der Anforderung des "Single Point of Truth" (Eine gemeinsame Codebasis für alle Plattformen) gerecht werden:

1. Eine für die Plattform erzeugte native App (React Native, Flutter). Diese Lösung bietet inhärent die beste Performance und ermöglicht Zugriff auf Betriebssystem-Ebene. Allerdings muss hier auf Basis eines spezifischen View-Models gearbeitet werden, was die Auswahl an Bibliotheken und den Community-Support erheblich einschränkt.

2. Eine für den Betrieb im Browser vorgesehene Web App (HTML und JavaScript basiert: React, Angular, Vue). Der Vorteil hierbei ist, dass die Technologie weit verbreitet ist und es eine große Auswahl an Templates und Bibliotheken gibt.

3. Eine Hybrid App (Ionic). Im Wesentlichen handelt es sich um eine Web App, wird aber in einem nativen Rahmen als eigenständige Anwendung installiert und ermöglicht daher bei Bedarf native Systemaufrufe. Aufgrund des Web-basierten Ansatzes profitiert auch diese Lösung von der großen JavaScript Community.

Für das Musiklernprogramm eignet sich die Umsetzung in einer Web App, die lediglich auf einem Webserver in Form von statischen Dateien bereitgestellt werden muss und demnach - abgesehen von einem Webbrowser - nahezu keinen Installationsaufwand für Nutzer bedeutet. Eine Hybrid App würde nur dann infrage kommen, wenn das Programm in einem deutlich späteren Entwicklungsstadium die Möglichkeit für Systemaufrufe bräuchte, beispielsweise um die Vision Engine (reacTIVision) im Hintergrund selbstständig zu starten. Da eine Portierung von einer Web App zu einer Hybrid App mit einem Framework wie Electron[12] als äußere Schale möglich ist, besteht diese Option weiterhin für die Zukunft.

Für die Entwicklung der Web App bietet sich an, ein Frontend Framework zu verwenden, das eine komponentenorientierte *Separation of Concerns* und bezüglich Zustandsänderungen ein reaktives Verhalten für Laufzeitvariablen als Grundlage für Elemente im HTML-Dokument unterstützt. Das Angebot ebensolcher Frameworks ist groß und die auf diversen Ranking-Webseiten hoch gelisteten Beispiele sind React[13], Angular[14] und Vue.js[15]. Aufgrund der im Vergleich flachen Lernkurve für Vue.js und persönlicher Vorerfahrungen wird dafür sich entschieden, dies zu verwenden. Des Weiteren wird sich im gleichen Zuge darauf festgelegt, Typescript als Javascript-Erweiterung für die Entwicklung zu benutzen, da durch die Typsicherheit Laufzeitfehler vermieden werden und die implizite Dokumentation durch den Code deutlich verbessert wird.

3.4.3 Kontextabgrenzung

Der fachliche Kontext, gezeigt in Abbildung 3.9, lässt sich leicht erschlagen: Ein Benutzer interagiert mit dem Programm entweder indirekt mit einem Objekt für ein Tabletop (hier: SUR40) oder direkt (im Non-TUIO Modus) per Maus oder Toucheingaben.

Die gezeigte *Application* wird als das zu entwerfende Teilsystem betrachtet und entspricht der *Individual Application* (Tier 2) aus der bereits vorgestellten Systemarchitektur in Abbildung 3.8.

[12] https://www.electronjs.org/
[13] https://reactjs.org/
[14] https://angular.io/
[15] https://vuejs.org/

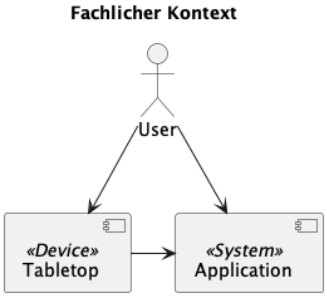


Abbildung 3.9: Fachlicher Kontext des Musiklernprogramms (System)

3.4.4 Bausteinsicht

Die *Application* unterteilt sich in die beiden wesentlichen Komponenten *TuioSupport* und *HarmonyHelper*. *TuioSupport* ist für Aufgaben zuständig, die allgemein für die Unterstützung eines TUIO-fähigen Geräts notwendig und daher auch für andere Anwendungsfälle anwendbar sind, während *HarmonyHelper* den fachlichen Kern des Musiklernprogramms enthält. Dies und die verwendeten externen Bibliotheken zeigt die Bausteinsicht in Abbildung 3.10.

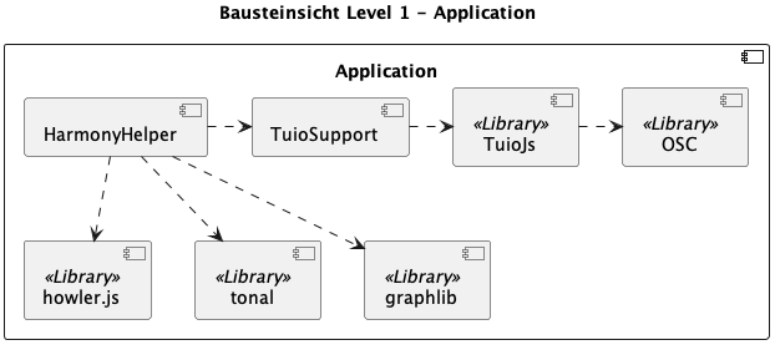


Abbildung 3.10: Bausteinsicht Level 1 - System

Die *HarmonyHelper*-Komponente ist in Abbildung 3.11 zu sehen und unterteilt sich in vier Subkomponenten:

- **UI-Elements**: Enthält alle Komponenten, die für die grafische Anzeige und Interaktion zuständig sind.
- **Music**: Verwaltet Logik für Vorschläge, Wiedergabe und Modellierung von Akkordfolgen.

- **SuggestionLogic**: Hält die abstrakte Logik für das auf Graphen basierte Vorschlagswesen. Die Trennung in einer separaten Komponente ermöglicht die generische Wiederverwendbarkeit für Vorschläge mit einer anderen Fachlichkeit.
- **Configuration**: Beinhaltet die applikationsspezifische Informationen wie beispielsweise Fiducial IDs, Durchmesser und die Wiedergabelautstärke.

Erwähnenswert ist auch hier, dass Abhängigkeiten stets in dieselbe Richtung entlang der Schichten gerichtet sind, um eine lose Kopplung zu erreichen. Abhängigkeiten zwischen den Klassen verschiedener Komponenten werden in der Darstellung zugunsten der Übersichtlichkeit zu einer allgemeinen Komponenten-Abhängigkeit zusammengefasst.

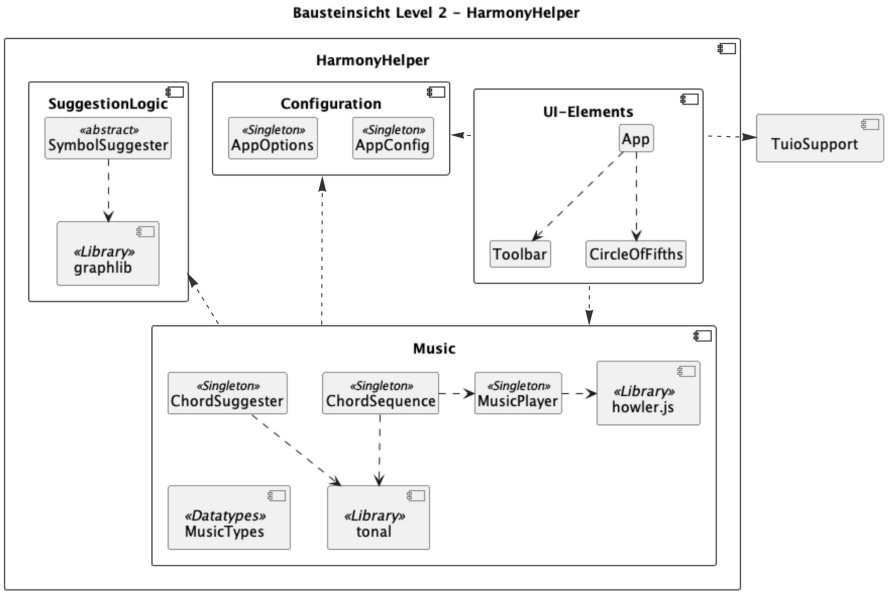


Abbildung 3.11: Bausteinsicht Level 2 - HarmonyHelper

3.4.5 Laufzeitsicht

Für ein leichteres Verständnis, wie Komponenten der Bausteinsicht miteinander wechselwirken, sollen zwei Sequenzdiagramme von Use Cases Zusammenhänge und Aufgaben verdeutlichen. Dabei wurde sich für Abläufe im TUIO Modus entschieden, die ein besonders hohes Maß an Kommunikation zwischen den Teilnehmern bedarfen.

Die Sequenz für die Erzeugung von Harmonievorschlägen in Abbildung A.3 zeigt, wie das Signal eines Auswahlobjekts (SelectorObject) durch die Hierarchie von UI-Komponenten jeweils partiell verarbeitet und/oder weitergereicht wird, um schließlich empfohlene Harmonien in der Auswahl des Quintenzirkels hervorzuheben. Der zentrale Baustein ist hier die *CircleOfFifths*-Komponente, die schließlich auch verantwortlich für die angezeigten Auswahlmöglichkeiten ist. Die Logik für die Berechnung der Vorschläge ist in *ChordSuggester* und der abstrakten Superklasse *SymbolSuggester* verortet und wird im nächsten Unterabschnitt erläutert.

Anhand der Sequenz für die Auswahl und Wiedergabe einer Harmonie Abbildung A.4 ist zu sehen, dass der allgemeine Verarbeitungsprozess ähnlich ist. Allerdings erkennt hier die *CircleSelect*-Komponente, dass das Auswahlobjekt entfernt wurde und sendet anstelle eines "update"- nun ein "toggle"-Signal, wobei sich als Nebeneffekt in *CircleSelect* das Auswahlmenü schließt. Die Darstellung verdeutlicht außerdem, dass der generische *MusicPlayer* keine Kenntnisse über Akkorde und Musiknoten benötigt, da *ChordSequence* bereits genügend Wissen über die enthaltenden Akkorde verfügt, um daraus die für die Wiedergabe notwendigen MIDI-IDs zu erhalten und nur diese weiterzugeben.

Harmonievorschläge

Für die Generierung von Harmoniefolgen wurde sich bereits in diversen Forschungsarbeiten beschäftigt. Diesbezüglich kristallisieren sich zwei wesentliche Ansätze heraus.

Einerseits kann das vorliegende Problem mit dem der Textvervollständigung im Bereich der *Natural Language Generation* verglichen werden, da es sich strukturell lediglich um eine Folge von Symbolen (Wörter oder Harmonien) handelt, auf Basis dessen eine Vorhersage für das Nächste getroffen wird. Häufige Lösungsansätze sind hier die Verwendung von Machine Learning Verfahren, insbesondere neuronale LSTM Netze und RNNs. Demnach werden in Wallace und Martin [20] ebendiese Netze verwendet, um auf Basis von Trainingsdaten in Form von bekannten Musikstücken Harmonievorschläge zu generieren. Vorausgesetzt, es ist ein geeigneter Datensatz mit Trainingsdaten verfügbar, ist mit diesem Ansatz wenig Fachwissen über die Anwendungsdomäne der Funktionsharmonik notwendig. Es besteht allerdings die Gefahr, dass Harmonien bei sehr diversen Trainingsdaten willkürlich erscheinen und Entscheidungen des Modells schwer manuell beeinflusst werden können. Außerdem ist JavaScript nicht die Programmiersprache der Wahl für Machine Learning, das am populärsten im Python-Umfeld vertreten ist. Daher

fügt sich eine solche Lösung schlecht in die gegebenen technischen Randbedingungen und die Architektur ein.

Andererseits finden sich Methoden für das Vorschlagswesen, die auf kontextfreien Grammatiken basieren. Entsprechend wird in Magalhães und Koops [14] eine Harmoniefolge algorithmisch mit einer gegebenen Grammatik erzeugt. Der Vorteil dieser Herangehensweise ist, dass die Vorschläge durch Anpassungen der Grammatik leicht definiert werden können und ein deterministischer Algorithmus die Entscheidungen des Programms nachvollziehbar macht. Aus diesem Grund soll dieser Ansatz als jener mit dem geringeren technologischen Overhead und Projektrisiko weiterverfolgt werden.

Modellierung von Regeln der Funktionsharmonik Da im Bereich der Funktionsharmonik allgemeine Aussagen getroffen werden können, wie "Eine Dominante möchte sich in die Tonika auflösen", lässt sich ein solches Regelwerk in Form einer kontextfreien Grammatik (KFG) formulieren, so wie es beispielsweise in Rohrmeier [16] getan wurde. Es folgt eine exemplarische KFG $G = \{V, T, P, S_0\}$ mit Variablen V, Terminalsymbolen T, Produktionen P, Startsymbol S zur Modellierung von Regeln für fundamentale Harmoniefolgen.

$$
\begin{aligned}
V &= \{P_0, A, D, S, T\} \\
T &= \{I, IIIm, IV, V\} \\
S &= P_0 \\
P &= \{P_0 \rightarrow AA \mid \epsilon \\
&\quad A \rightarrow TDT \\
&\quad T \rightarrow I \\
&\quad D \rightarrow II\ V \mid V \mid SD \\
&\quad S \rightarrow IIIm\ IV \mid IV\}
\end{aligned}
\tag{3.1}
$$

Bei genauer Betrachtung akzeptiert G die Folge *Tonika-Dominante-Tonika*, woraus sich auch die bekannte musikalische Kadenz *Tonika-Subdominante-Dominante-Tonika* ableiten lässt. Die gezeigte Grammatik ist beliebig austausch- und erweiterbar und soll hier als Fallbeispiel für die Implementierung dienen.

Erzeugen von Harmonievorschlägen Mit einer kontextfreien Grammatik ist nun also ein Regelwerk für Harmoniefolgen gegeben. Darauf aufbauend wird ein Mechanismus gesucht, der ausgehend von einer konkreten Harmoniefolge eine Vorschlagsliste erzeugt,

die alle möglichen regelkonformen Optionen für einen Folgeakkord enthält. Auf die KFG bezogen soll also ein Algorithmus entwickelt werden, der ausgehend von einem Teilwort (das auch leer sein darf) eine Vorschlagsliste erzeugt, die alle möglichen Symbole enthält, sodass eine Produktion der Grammatik angewendet werden kann, also wieder ein gültiges (Teil-)wort entsteht.

Eine KFG kann unter anderem mit einem Kellerautomaten implementiert werden. Ein solcher Automat kommt auf dem ersten Blick mit sehr wenigen Zuständen aus, da die Eigenschaft, dass die Produktionen einer Grammatik rekursiv mit einem Stack bearbeitet werden können, mit dem Keller dargestellt werden kann. In A.5 wird entsprechender Kellerautomat für die KFG in 3.1 dargestellt. Die Vorgehensweise für die Erzeugung von Harmonievorschlägen ist hier, solange die auf dem Keller-Stack befindliche Variable zu verarbeiten (einlesen von ϵ), bis ein Terminalsymbol (also eine Harmonie) auf dem Stack liegt. Dies darf erst dann wieder eingelesen werden, wenn der Benutzer tatsächlich diese Harmonie ausgewählt hat. Die Komplexität dieser Implementierung wird durch das nichtdeterministische Verhalten erhöht. Es kann schließlich mehrere Aktionen für eine Variable auf dem Stack geben (Im Beispiel betrifft es $\epsilon, D|SD$, $\epsilon, D|V$), sodass alle Alternativen in Kopien des PDA simuliert werden müssen, bis in allen Kopien ein Terminalsymbol auf dem Stack liegt. Bei Benutzereingabe eines passenden Symbols einer Kopie, müssen alle anderen Kellerautomaten verworfen werden. Die Erzeugung und Verwaltung der parallel existierenden Zustände und die Zweckentfremdung eines Kellerautomaten - üblicherweise sind sie für das Parsen bereits erzeugter Wörter gedacht - sprechen trotz der Einfachheit der Grundstruktur des Automaten aufgrund der erhöhten Komplexität bei horizontaler Skalierung gegen die Verwendung dieses Ansatzes der Problemlösung.

Alternativ wäre es wünschenswert, eine KFG in einem einfachen Graphen darstellen zu können, da Algorithmen auf ihnen gut visualisiert werden können und ein Graph eine gleichbleibende endliche Menge diskreter Knoten hat. Die Knoten können Terminal- und Nonterminalsymbole repräsentieren und die Kanten kennzeichnen erlaubte Übergänge, wodurch jeder Pfad eine mögliche Harmoniefolge darstellt. Ein Graph für eine KFG ohne Linksrekursion lässt sich folgendermaßen erzeugen:

1. Weise dem aktuellen Nonterminal $v \in V$ das Startsymbol S zu: $v = S$.
2. Für alle Symbole $s \in \{V \cup T\}$ auf der rechten Seite der Produktionen von v:
   a) Erzeuge einen Knoten n_s für das Symbol.

b) Füge eine gerichtete Kante vom Vorgängersymbol (falls vorhanden) von s zu n_s hinzu.

c) Falls $s \in V$ (ist ein Nonterminalsymbol): Gehe zu Punkt 2 mit $v = s$.

Das Ergebnis der Transformation in einen Graphen ist in Abbildung 3.12 zu sehen. Nonterminale werden farbig hervorgehoben, weil sie im nachfolgenden Algorithmus für das Erzeugen von Vorschlagslisten eine besondere Rolle spielen.

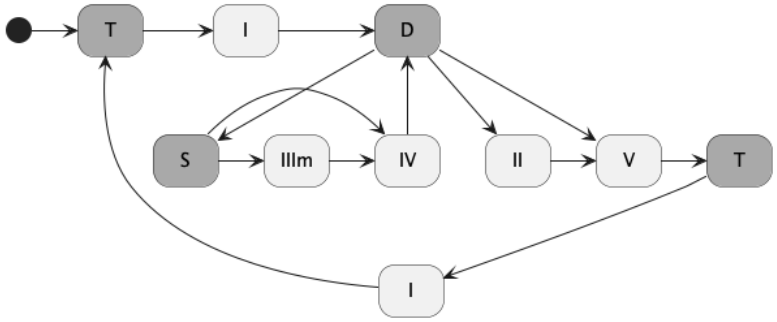


Abbildung 3.12: Graph-Repräsentation der exemplarischen Grammatik in 3.1 für Harmoniefolgen

Algorithmus zum Erzeugen einer Vorschlagsliste Da nun die Basis für die Repräsentation einer KFG in einem Graphen ist, kann diese nun genutzt werden, um einen Algorithmus für die Verwaltung einer Vorschlagsliste basierend auf der bisher ausgewählten Folge von Terminalsymbolen zu entwerfen. Die Vorschlagsliste enthält stets alle möglichen Eingabesymbole (später konkret: Harmonien), die innerhalb der gegebenen Grammatik mit gegebener Historie erlaubt sind. Die Historie von Harmonien wird durch das Traversieren durch den Graphen ausgehend vom Startknoten realisiert. Da der Nutzer auch Harmonien auswählen können soll, die nicht vorgeschlagen wurden, stellt die Historie nicht zwingend ein gültiges Wort der Grammatik dar.

Wichtig ist für den Algorithmus, dass ein Unterschied zwischen Nonterminalen (T, D und S in dunkelgrau) und Terminalen (hellgrau) gemacht wird, da ein Terminal als konkrete Harmonie Bestandteil eines Vorschlags sein kann und Nonterminale lediglich die harmonische Funktion aller Kindknoten bis zum nächten Nonterminal markieren, wie in Abbildung 3.12 zu sehen ist. Im Folgenden wird der für dieses Problem entwickelte Algorithmus unterteilt in Vorbedingungen, Initialisierung und der Ausführung, beschrieben.

Vorbedingungen
Gegeben ist eine Liste H mit bereits ausgewählten Akkorden (kann auch leer sein), als Constraints das gewünschte Tongeschlecht (Dur/Moll) g und die harmonische Funktion f (Tonika, Dominante, Subdominante). Außerdem hat jeder Terminal-/Akkord-Knoten des zugrundeliegenden Graphen ein Attribut über die zugehörige harmonische Funktion, also dem nächsten Nonterminal-Elternknoten, und über die Stufe (I, II etc.) des Funktionsakkords. Als Grundton wird "C" festgelegt.

1. **Initialisierung**: Setze v_{cur} auf den Startknoten.
2. Erzeuge eine leere Vorschlagsliste V.
3. **Ausführung**: Für jeden zu v_{cur} adjazenten erreichbaren Knoten v_{ad}:
4. Falls v_{ad} ein Terminalsymbol repräsentiert: Füge v_{ad} zu V hinzu.
5. Sonst: Führe Schritt 3 rekursiv aus mit $v_{cur} = v_{ad}$.
6. Wiederhole die Schritte 2 bis 5 mit $c_{cur} = h$ für jedes $h \in H \wedge h \in V$.
7. Entferne alle Knoten aus V, die nicht der gegebenen harmonischen Funktion f entsprechen.
8. Gebe eine Liste konkreter Akkorde aus den Funktionsakkorden in V auf Basis des gegebenen Grundtons und Tongeschlechts g aus.

Ermitteln der harmonischen Funktion

Abhängig von der Position des Quintenzirkels auf der grafischen Benutzeroberfläche wird eine bestimmte harmonische Funktion adressiert, die sich auf die Harmonievorschläge auswirkt. Um zu ermitteln, in welchem Bereich sich ein Punkt (die Position des Quintenzirkels) in einem Koordinatensystem (der Anzeigefläche des HTML-Dokuments) befindet, ist die Anwendung von linearer Algebra notwendig.

Das Problem lässt sich anhand Abbildung 3.13 skizzenhaft darstellen. Hierbei ist zu erwähnen, dass h die Höhe und w die Breite der Gesamtfläche zur Laufzeit bekannt sind und daher als gebundene Variablen betrachtet werden. Zunächst werden die Geradengleichungen für die Diagonalen a und b anhand der bekannten Punkte ermittelt. Gesucht

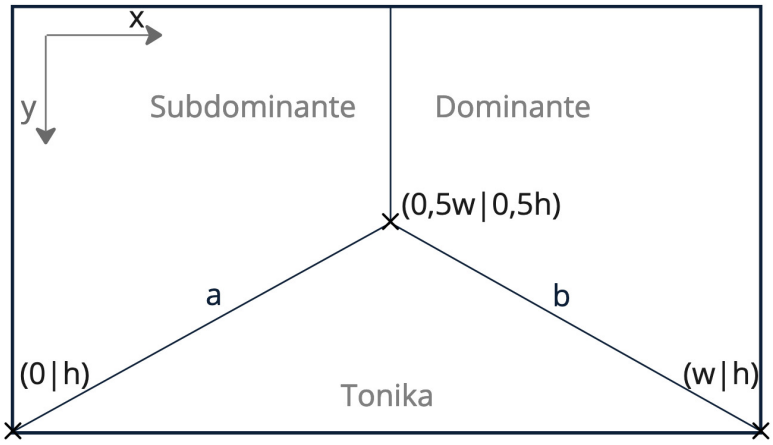


Abbildung 3.13: Flächen für harmonische Funktionen

sind also jeweils m und n in $a(x) = mx + n$ und $b(x) = mx + n$. Das Einsetzen der Punkte $(0|h)$ und $(0,5w|0,5h)$ in $a(x)$ ergibt:

$$\begin{aligned} n &= h \\ \frac{h}{2} &= m * \frac{w}{2} + h \\ m &= -\frac{h}{w} \\ a(x) &= -\frac{h}{w}x + h \end{aligned} \tag{3.2}$$

Das Einsetzen der Punkte $(w|h)$ und $(0,5w|0,5h)$ in $a(x)$ ergibt:

$$h = mw + n \tag{1}$$

$$\frac{h}{2} = m\frac{w}{2} + n \tag{2}$$

$(1) - (2):$

$$\begin{aligned} \frac{h}{2} &= m\frac{w}{2} \\ m &= \frac{h}{w} \end{aligned}$$

$m\ in\ (1):$

$$\begin{aligned} n &= h - \frac{h}{w}w = 0 \\ b(x) &= \frac{h}{w}x \end{aligned}$$

Mithilfe von $a(x)$, $b(x)$ lässt sich nun ermitteln, welcher harmonischen Funktion ein Quintenzirkel auf einem beliebigen Punkt P der Gesamtfläche zugeordnet wird. Algebraisch ist es dafür entscheidend, ob sich P oberhalb oder unterhalb der Geraden befindet und ob $P(x|y)$ links oder rechts der vertikalen Mittellinie (trivial anhand der x-Koordinate erkennbar) befindet. P befindet sich im Bereich "Subdominante", wenn gilt: $a(x) > y \wedge x < \frac{w}{2}$. Äquivalent gilt für "Dominante" $b(x) > y \wedge x \geq \frac{w}{2}$ und in allen anderen Fällen muss P im "Tonika"-Bereich verortet sein.

3.4.6 Entwurfsentscheidungen

Eine erwähnenswerte Entwurfsentscheidung ist jene über das Design der Tangible Objects, da dies ein wichtigen Faktor bezüglich der Usability der Software ist. Grundsätzlich gibt es hierfür zwei Anforderungen. Technisch bedingt muss das Objekt einen Yamaarashi Fiducial Marker mit ausreichend großem Durchmesser (ca. 4cm nach Kaltenbrunner [9]) für den Tracking-Mechanismus besitzen. Außerdem sollte das Objekt aus fachlicher Sicht seinem Nutzer eine Information über den Verwendungszweck vermitteln.

Beide Anforderungen sollen mit entsprechende Beschriftungen oder Symbolen realisiert werden, wobei der Formfaktor aller Objekte aufgrund der Wiederverwendbarkeit identisch ist. Es handelt sich konkret um die vier Objekte, die bereits im Wireframe (Abbildung A.2) gezeigt werden. Da insbesondere die Objekte "Dur" und "Moll" in ihrer Verwendung nur im wechselseitigen Ausschluss sinnvoll sind (es soll nur ein Quintenzirkel zugleich eingestellt werden können), könnte dies sehr gut forciert werden, indem deren Fiducial Marker auf den Wendeseiten eines einzigen Objekts platziert werden. Allerdings kann bei dieser Lösung die Beschriftung ambivalent interpretiert werden: "Bedeutet es, wenn ich Moll sehe, dass Moll gerade aktiv ist, oder muss ich die Moll-Seite auf das Display legen, damit der daneben befindliche Fiducial Marker den Moll-Modus aktiviert?". Außerdem wäre diese Lösung der beidseitigen Doppelfunktion nicht nachvollziehbar auf das Play- und Löschobjekt übertragbar, da es hier keinen semantischen Zusammenhang gibt.

Schließlich wird sich dafür entschieden, aus Konsistenz- und ästhetischen Gründen die Objekte einheitlich zu designen: Auf einer Seite befindet sich der Fiducial Marker und auf der Kehrseite die für den Anwender sichtbare und verständliche Kennzeichnung der Funktion.

3.5 API für TUIO-UI Kopplung

Wie bereits in Unterabschnitt 3.4.4 zu sehen, ist die Komponente *TuioSupport* Bestandteil des zu entwerfenden Systems. Diese realisiert aus der makroskopischem Perspektive betrachtet den Interpretation Layer der Gesamtarchitektur in Abschnitt 3.1. Der Vorteil der strikten Trennung von der Benutzeranwendung liegt in der Wiederverwendbarkeit durch die Abstraktion der Problemstellung auf eine rein technische Betrachtungsweise: Diese Komponente soll als Programmierschnittstelle Möglichkeiten für die Kopplung von Informationen aus TUIO-Nachrichten (beispielsweise die Fiducial-Objekterkennung) an grafische Elemente der Benutzeroberfläche bereitstellen. Diese Oberfläche beschränkt sich hier aufgrund technischer Randbedingungen auf ein HTML-Dokument, das es zu modifizieren gilt.

3.5.1 Kontextabgrenzung

Wie im technischen Kontext in Abbildung 3.14 zu sehen ist, kapselt die *TuioSupport* die Verarbeitungslogik der rohen TUIO-Nachrichten und soll für die Verwendung in einer Benutzeranwendung relevante Events erzeugen. Die Kapselung von TuioJs bewirkt dass, ein möglicher Austausch durch eine andere Bibliothek nur weniger Anpassungen innerhalb der TuioSupport-Komponente bedarf. Gleichzeitig bietet die Komponente weitere Funktionalitäten an, wie die Bereitstellung von HTML-Templates in Form von Vue Components, die bereits mit TUIO Objekten synchronisiert werden können. Mit dieser Entscheidung geht einher, dass *TuioSupport* aus diesem Grund nur mit Vue-Anwendungen kompatibel ist. Dies ließe sich bei Bedarf jedoch lösen, indem UI Komponenten modular bereitgestellt würden.

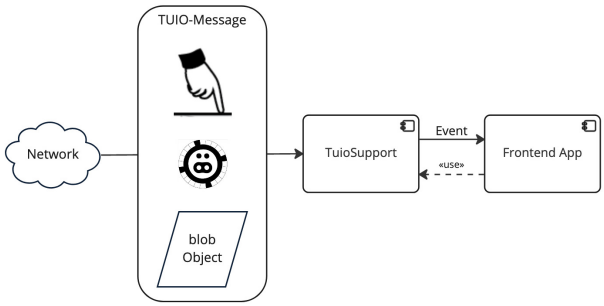


Abbildung 3.14: Technischer Kontext - TuioSupport

3.5.2 Bausteinsicht

Abbildung 3.15 zeigt die Bausteinsicht, die sich in die Architekturdokumentation aus Unterabschnitt 3.4.4 eingliedert. Es folgen einige Anmerkungen zu Aufgaben der gezeigten Klassen.

- **TuioClient**: Deserialisiert OSC-Nachrichten und transformiert diese in TUIO-Events
- **TuioEventDispatcher**: Verwaltet die Verbindung zum TuioClient und leitet empfangene TUIO-Events an registrierte Event-Listener weiter.
- **TuioEventListener**: Verarbeitet und filtert TUIO-Events für einen bestimmten Zweck. Diese abstrakte Klasse registriert sich im Sinne des Observer Pattern am TuioEventDispatcher.
- **TuioObjectEventListener**: Implementiert TuioEventListener und behandelt TUIO-Events, die für das an diesen Listener gebundene FrontendTuioObject relevant sind.
- **FrontendTuioObject**: Repräsentiert ein TUIO Objekt mit Informationen über dessen Zustand, ermöglicht die Erweiterung um in der UI benötigte Informationen und kapselt im Stil des Adapter Pattern die Abhängigkeit zu TuioJs.
- **FrontendSelectorTuioObject**: Hält zusätzliche Informationen, die für die Verwendung als Auswahlobjekt (ähnlich zu einem Cursor) in der UI benötigt werden.

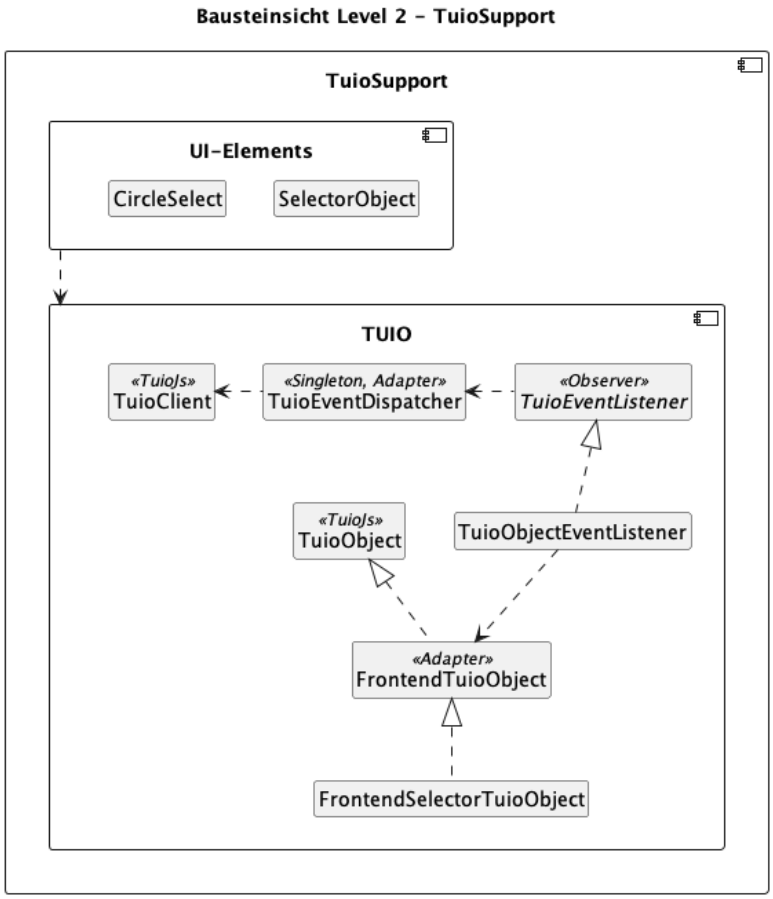


Abbildung 3.15: Bausteinsicht Level 2 - TuioSupport

3.5.3 Laufzeitsicht

Die Aufgaben der Bausteine werden in der Sequenz für die Aktualisierung der Position und Ausrichtung eines Auswahlobjekts (SelectorObject) der UI in Abbildung 3.16 veranschaulicht. Erwähnenswert ist, dass das SelectorObject ebenfalls ein Event emittiert, das von referenzierenden Vue Components (hier: *App*) behandelt werden kann. Die absolute Position und Rotation der grafischen Repräsentation des Objekts werden anhand des aktualisierten Zustands des FrontendTuioObject reaktiv (mittels einer Funktion, die Vue aufgrund einer detektierten Änderung einer Variable ausführt) ermittelt.

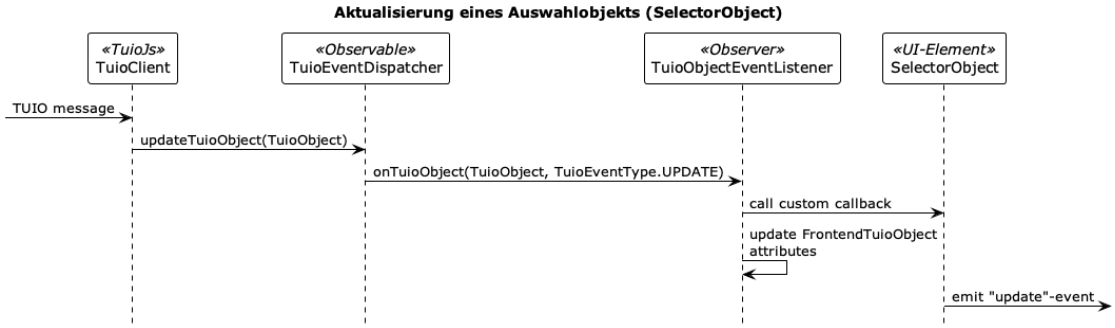


Abbildung 3.16: Sequenzdiagramm: Aktualisierung eines Auswahlobjekts

Umgang mit Messungenauigkeiten

Während der Realisierung der Rotationsindikation des Auswahlobjekts konnten Schwankungen des in der TUIO-Nachricht enthaltenen Winkels beobachtet werden. Dieses Verhalten muss auf Mess- oder Rundungsfehler im Transformation-, Hardware Abstraction- oder Input Hardware-Layer in Tier 1 zurückzuführen sein, die in Abhängigkeit von Rotationswinkel und Lichtverhältnissen sichtbar werden. Aufgrund des Aufwands soll dies als Umgebungsbedingung gelten, weshalb eine Lösung innerhalb des Interpretation Layer angebracht ist. Dadurch soll ein "Flackern" des im Frontend angezeigten Winkels erheblich reduziert werden.
Bei genauerer Analyse fällt während des Fehlverhaltens ein Muster in der Zeitreihe von Messwerten auf: Der Zahlenwert oszilliert recht zuverlässig, sodass jeder Wert α zum Zeitpunkt t relativ zum vorherigen und nachfolgenden entweder größer oder kleiner ist. Dies ist auch in einer exemplarischen Messreihe in Abbildung 3.17 zu sehen. Dafür wurde ein Objekt mit einem Fiducial Marker auf das SUR40 gelegt und die emittierten

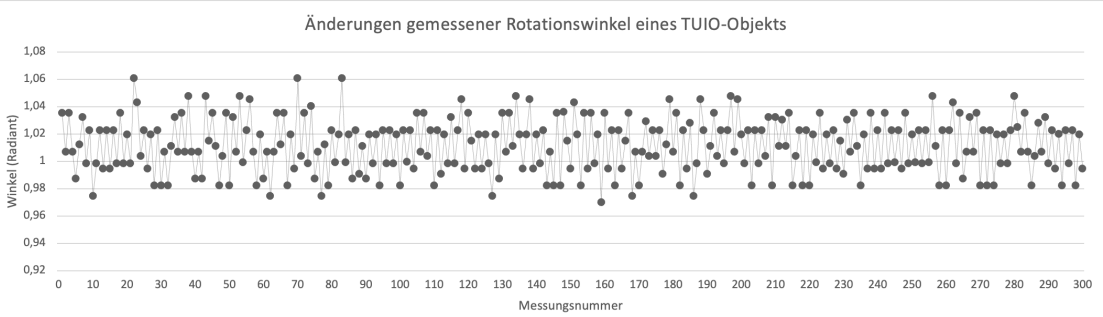


Abbildung 3.17: Messreihe: Änderungen gemessener Rotationswinkel (Radiant) eines unbewegten TUIO-Objekts mit reacTIVision auf dem SUR40 (im Zeitraum von wenigen Sekunden)

TUIO-Nachrichten von reacTIVision gesammelt, wobei jene mit unverändertem Rotationswinkel herausgefiltert wurden. Bei der vorliegenden Streubreite von $0,09\,rad = 5,16°$ (Standardabweichung ist $0,02\,rad = 1,15°$) ist dies schließlich signifikant für ein sichtbares Oszillieren der Rotationsindikation verantwortlich.

Als Lösung für das beschriebene Problem sollen nur solche Werte als gültig interpretiert werden, die Teil einer monoton steigenden oder fallenden Messreihe sind.
Also muss gelten: $\alpha_{t-1} < \alpha_t < \alpha_{t+1} \vee \alpha_{t-1} > \alpha_t > \alpha_{t+1}$. Die Empfindlichkeit des Mechanismus lässt sich anhand der Anzahl n der betrachteten letzten Werte $\alpha_{t-n} \dots \alpha_t$ einstellen.

Die Implementierung des Filters wird mithilfe des Strategy-Patterns realisiert. Wie in Abbildung 3.18 zu sehen, kann dieser Mechanismus in eine separate Klasse (*TuioObjectAngleFilter*) extrahiert und für andere Implementationen wiederverwendet werden. Durch das Überschreiben von Methoden wird das Update-Verhalten des Objekts ausreichend gekapselt, sodass externe Komponenten wie der Event-Listener und jene aus der UI keine Kenntnis davon benötigen.

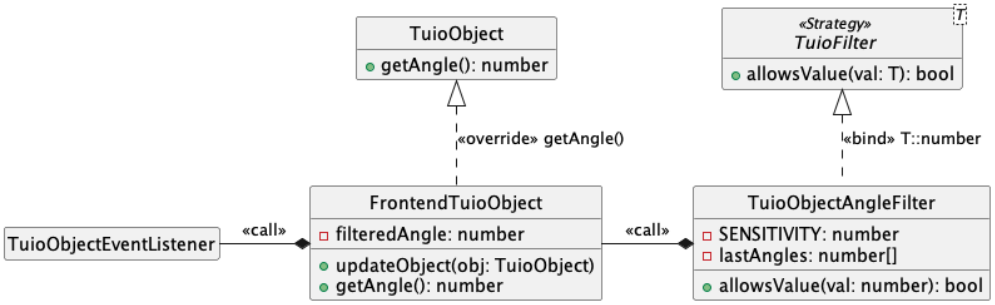


Abbildung 3.18: Klassendiagramm mit relevanten Teilnehmern für den Umgang mit Messungenauigkeiten des Rotationswinkels eines TUIO Objekts. Für den Filtermechanismus irrelevante Attribute, Beziehungen und Methoden wurden zur Vereinfachung der Darstellung weggelassen.

3.6 Teststrategie

Für die Qualitätssicherung des Systems und Früherkennung von Fehlern auf mehreren Ebenen wird eine Teststrategie basierend auf dem allgemeinen V-Modell nach Boehm [4] entworfen, das -in ihrer Granularität zunehmend- Abnahmetests, Systemtests, Integrationstests und Komponententests vorsieht. Die Ausführung der Testobjekte soll weitgehend automatisiert erfolgen, um eine Integration in das *Continuous Development* zu ermöglichen.

3.6.1 Komponententests

Auf Komponentenbasis sollen Methoden in Typescript-Klassen in Form von Unit-Tests mithilfe der Unit Testing Library *Jest*[16] durchgeführt werden. Hierbei sind Komponenten, die für fachlichen Anforderungen wichtigen komplexe Kernlogiken wie die Erzeugung der Harmonievorschläge und Modellierung der Akkordfolge verantwortlich sind, als besonders hoch zu priorisieren.

3.6.2 Integrationstests

Integrationstests prüfen auf einer höheren Ebene die Zusammenarbeit von Komponenten anhand deren Schnittstellen. In Bezug auf das zu entwickelnde System kommen hierfür

[16] https://jestjs.io/

die Vue Components infrage, die viele Referenzen auf Units in der Logikschicht besitzen. Hier sollte also sichergestellt werden, dass die Kommunikation einer Vue Component mit anderen Klassen (MusicPlayer, ChordSuggester etc.) und mit anderen Vue Components korrekt funktioniert. Jedoch liegt diesbezüglich eine technische Einschränkung vor, da die verwendeten Test-Frameworks auf das Testen auf Komponenten- oder Systemebene spezialisiert sind. Daher werden an dieser Stelle einige *Component Tests* mit *Cypress*[17] mit Beschränkung auf die Prüfung der Schnittstellen zwischen Vue Components entworfen und andere Aspekte der Integrationstests müssen manuell oder mit System- bzw. Abnahmetests abgedeckt werden. Ein Beispiel für automatisierte Testfälle ist hier das Prüfen auf emittierte Events, die beispielsweise nach Initialisierung oder bei Positionsänderung eines Quintenzirkels von anderen Components behandelt werden.

3.6.3 Systemtests

In Systemtests wird das gesamte System als Blackbox getestet und so mit Entwurf und Spezifikationen abgeglichen. Hierbei zielen Tests im Allgemeinen sowohl auf funktionale als auch auf nicht-funktionale Anforderungen in einer realen Systemumgebung ab. Zum System gehören hier die Frontend Applikation und die gesamte Pipeline der Verarbeitung des Tabletop Inputs. Ein besonders wichtiger Testfall wäre die korrekte Repräsentation eines Fiducial Objekts in dem Musiklernprogramm auf dem SUR40 einschließlich der Kommunikation zwischen beiden Tiers. Dieser Teil der Systemtests werden allerdings nur manuell ausgeführt, da der Aufwand für das Testsetup für eine automatisierte Testumgebung mit simuliertem Hardware-Inputstream unverhältnismäßig hoch wäre. Andererseits lässt sich die Lernsoftware zumindest im Non-TUIO Modus gut automatisiert testen. Auch hierfür wird das Testframework *Cypress* verwendet, mithilfe dessen Interaktionen mit dem HTML-Dokument in einem Browser simuliert und Erwartungen über das optische Erscheinungsbild (z.B. Position, Existenz von Elementen) formuliert und überprüft werden (auch End-to-End Test genannt). Für die Überprüfung der Unterscheidung zwischen TUIO- und Non-TUIO Modus wird bei Bedarf ein Mock eines nicht funktionalen Websocket-Servers erstellt.

[17] https://cypress.io

3.6.4 Abnahmetests

Der Abnahmetest soll hier der abschließende manuelle Test vor dem Abschluss eines Inkrements bzw. Anwendungsfalls sein. Dieser findet auf dem SUR40 mit der im HAW Pub aktuell installierten Version des Musiklernprogramms. Die Version entspricht, durch CI/CD abgesichert, dem Stand aus dem Entwicklungsbranch des Versionsverwaltungssystems (Git), der die vorhergehenden Testphasen ohne Beanstandungen durchlaufen ist.

3.7 Entwicklungsumgebung

In diesem Kapitel wird dargelegt, mithilfe welcher Methoden und Werkzeuge die Realisierung des Systems erleichtert wird.

3.7.1 Simulation

Um auch in der Entwicklungsphase die technische Abhängigkeit des Tabletop und somit zugleich eine Bindung an den Ort im Living Place aufzulösen, bietet sich der Einsatz eines Programms an, das ein solches Gerät simuliert und entsprechende TUIO-Nachrichten verschickt. Dafür gibt es den von Martin Kaltenbrunner bereitgestellten *TUIO Simulator*[18]. Allerdings unterstützt dieser nur UDP, weshalb er im Rahmen dieser Arbeit um einen konfigurierbaren WebSocket-Support erweitert wurde. Es wurde auf das Springframework umgestellt, damit eine dateibasierte Konfiguration und Austauschbarkeit von Implementierungen für den Server-Socket via Dependency Injection auf eine einfache Art ermöglicht wird. Die Erweiterung des Simulators wurde als Fork auf GitHub der Öffentlichkeit zu Verfügung gestellt[19].

3.7.2 CI/CD

Das mit Git verwaltete Code-Repository befindet sich auf der GitLab Instanz der HAW. Zum Zweck der Automatisierung von Testprozessen und des Deployments wird das Toolset der dort verfügbaren CI/CD Infrastruktur benutzt. Die auszuführenden Aufgaben

[18] TUIO Simulator: https://github.com/mkalten/TUIO11_Simulator
[19] TUIO Simulator mit WebSocket Support: https://github.com/jkruke/TUIO11_Simulator

Tabelle 3.1: CI/CD Pipeline

stage	step	Funktion
prepare	prepare	Lokales Setup (Dependencies herunterladen)
test	lint	Statische Codeanalyse (z.B. Code-Konventionen und Typsicherheit)
test	unit test	Durchführen von Komponententests
test	integration test	Durchführen von Integrationstests
test	e2e test	Durchführen von Systemtests (End-to-End)
deploy	deploy	Installation der Code-Artefakte im persönlichen HAW Pubverzeichnis hier geht es noch weiter

werden in *stages* unterteilt, die wiederum in mehrere parallel ausführbare *steps* unterteilt sind und werden in Tabelle 3.1 aufgeführt. Da die Ausführung einer stage eine fehlerfreie vorhergehende stage bedingt, ist sichergestellt, dass das Deployment ausschließlich für erfolgreich geprüfte Artefakte durchgeführt wird.

3.7.3 Tooling

Für die Arbeit mit dem Vue.js Framework bieten sich dedizierte Plugins für die IDE (Unterstützung von spezifischen Sprachkonstrukten) und den Webbrowser (Analyse von Laufzeitvariablen für Debugging-Zwecke) an. Im Fall eines Fehlers im Zusammenhang mit dem TUIO Modus auf dem SUR40 ist das Debuggen im Browser auf demselben Gerät recht unkomfortabel. Aus diesem Grund gibt es für das Vue Projekt eine zweite Startkonfiguration für den Betrieb auf einem anderen Laptop, sodass die Applikation dort bedient werden kann und zugleich eine Netzwerkverbindung zum Tabletop aufbaut und daher auch TUIO-Nachrichten erhält.

4 Bewertung und Fazit

4.1 Bewertung der Realisierung

Die Realisierung des Entwurfs kann grundsätzlich im Rahmen einer *Functional Evaluation* und einer *User Evaluation* bewertet werden. Auf die Letztere wird in dieser Arbeit verzichtet, da dieser Prozess idealerweise eine Nutzerstudie mit der Zielgruppe "Schüler einer allgemeinbildenden Schule" mithilfe des SUR40 durchgeführt werden müsste und einen unverhältnismäßigen organisatorischen Aufwand bedeuten würde. Daher wird sich auf eine funktionelle Prüfung der Anforderungen beschränkt, welche in Form der in Abschnitt 3.6 erläuterten Teststrategien durchgeführt wurde. In Abbildung 4.1 ist die auf dem SUR40 im Living Place ausgeführte Anwendung zu sehen und zeigt eine Übereinstimmung mit dem Entwurf der grafischen Benutzeroberfläche, den Tangible Objects und den Kernfunktionalitäten der Software.

Es ist allerdings anzumerken, dass der reibungslose Betrieb von *reacTIVision* nicht ohne Komplikation erreicht wurde. Schließlich war ein Debugging auf Low-Level-Ebene im SUR40 Kerneltreiber und der Engine notwendig, bis die Signale der Pixelsensoren als Kamera-Stream interpretiert werden konnte (Siehe Details dazu in Unterabschnitt A.3.2). Trotz der hier gewonnenen Erkenntnisse ist es daher nicht auszuschließen, dass das initiale Setup auf einem anderen Tabletop erneut nicht ohne Probleme ablaufen wird.

Auf der Applikationsebene war es zunächst nicht leicht, eine adäquate JavaScript-Bibliothek für die Verarbeitung von TUIO WebSocket-Nachrichten zu finden. Als Ergebnis musste das verwendete *TuioJs* angepasst werden, indem die Implementierung des OSC-Parsers (für Deserialisierung der OSC Nachrichten) durch eine aktuellere Version ausgetauscht wurde und die Schnittstellen mit Typdeklarationen für die Verwendung mit Typescript Code erweitert wurde. Diese Problematik ist auf die geringe Aktivität der TUIO Community zurückzuführen, wodurch Projekte innerhalb der Domäne stark gealtert sind und Kompatibilitätsprobleme mit sich bringen.

Abbildung 4.1: Das entwickelte Musiklernprogramm “HarmonyHelper” im Einsatz auf dem SUR40 im HAW Living Place

Insgesamt konnte das Vorhaben jedoch gut umgesetzt werden und bietet eine Grundlage für potenzielle Erweiterungen technischer und fachlicher Natur. Beispielsweise ist eine Erweiterung um die Erkennung von Fingern (Touch) per TUIO oder eine Möglichkeit, die Grundtonart (C) zu ändern, durchaus denkbar.

4.2 Fazit

Bezüglich des Lösungskonzepts ist anzumerken, dass aufgrund technischer Randbedingungen mit dem SUR40 nicht mit dem heutigen Stand der Technik gearbeitet werden konnte. Schließlich wird das Gerät seit vielen Jahren nicht mehr verkauft und die PixelSense Technologie scheint sich angesichts der Produktpalette und der verfügbaren Software nicht durchgesetzt zu haben. Dies gefährdet die Nachhaltigkeit der Ergebnisse dieser Bachelorarbeit, da die Hardware dafür unter Umständen in einigen Jahren nicht mehr auf einem aktuellen Betriebssystem lauffähig sein könnte.

Die Lösung, das System anstelle von Windows unter Ubuntu zu betreiben, bietet andererseits eine hohe Flexibilität und somit mehr Möglichkeiten, das SUR40 länger verwenden zu können. Das *reacTIVision* Framework für Linux funktioniert zuverlässig und stellt

einen robusten Adapter zwischen Hardware und Software dar. Bezüglich des Musiklernprogramms ermöglicht die Umsetzung als Web App für die Bedienung im Browser eine sehr leichtgewichtige Installation und Nutzung. Der Nachteil ist hierbei lediglich, dass die Applikation für den TUIO Modus darauf angewiesen ist, dass eine TUIO Vision Engine (*reacTIVision*) erreichbar ist, die manuell auf dem Tabletop gestartet werden muss. Damit das Programm die Engine als "soft dependency" selbständig starten kann, könnte die Web App stattdessen als eine Hybrid App provisioniert werden, wodurch Systemaufrufe und somit auch das Starten von Drittprogrammen möglich wäre. Eine Integration der bestehenden Lösung in diese Applikationsform ist bei weiteren Anforderungen für native Eigenschaften durchaus machbar, aber in dieser Arbeit nicht notwendig.

Eine weitere Erkenntnis ist, dass für eine zu entwerfende Software stets zu hinterfragen ist, inwiefern sich der Einsatz von Objekten als Interaktionsmedium für den Anwendungsfall eignet. Schließlich ist eine UI nur bedingt skalierbar, wenn die Bedienung ausschließlich mit Objekten erfolgen soll, da es mit dem gewählten Konzept für einen einfachen neuen Button im Non-TUIO Modus meist ein neues Objekt für den TUIO Modus geben müsste. Dabei stellt sich die Frage, inwiefern die Bedienung bei wachsender Anzahl verschiedenster Objekte dann noch möglichst einfach bleibt und ob es nicht intuitiver ist, virtuelle Buttons mit einem allgemeinen Cursor-Objekt - vergleichbar mit einer Maus - betätigen zu können. Denkbar ist auch ein hybrider Ansatz, der die Benutzerinteraktion je nach Use Case sowohl mit realen Objekten als auch auf klassische Weise per Touch oder Maus ermöglicht. Dies würde die Grenzen zur Tangible User Interaction verschwimmen lassen und haptische Aspekte zugunsten der Bedienbarkeit etwas in den Hintergrund verschieben.

Abschließend bleibt unklar, ob ein Lernprogramm in der hier erarbeiteten Ausprägungsform tatsächlich den Lernerfolg im Musiktheorieunterricht steigert. Dazu bräuchte es Bewertungen aus der bildungswissenschaftlichen Perspektive und Nutzerstudien, die auf dieser Bachelorarbeit aufsetzen können. Es wurde jedoch eindeutig gezeigt, dass es möglich ist, mit einem Musiklernprogramm auf greifbare Weise zu interagieren und somit eine Grundlage für zukünftige Arbeiten geschaffen.

Literaturverzeichnis

[1] Anderson, Glen ; Doherty, Rina ; Ganapathy, Subhashini: User Perception of Touch Screen Latency. In: Marcus, Aaron (Hrsg.): *Design, User Experience, and Usability. Theory, Methods, Tools and Practice*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2011, S. 195–202. – ISBN 978-3-642-21675-6

[2] Bencina, Ross ; Kaltenbrunner, Martin: The design and evolution of fiducials for the reactivision system. In: *Proceedings of the Third International Conference on Generative Systems in the Electronic Arts* Bd. 2 Monash University Publishing Melbourne, VIC, Australia (Veranst.), 2005

[3] Bhalla, Mudit R. ; Bhalla, Anand V.: Comparative Study of Various Touchscreen Technologies. In: *International Journal of Computer Applications* 6 (2010), S. 12–18

[4] Boehm, B. W.: Guidelines for Verifying and Validating Software Requirements and Design Specifications. In: Samet, P. A. (Hrsg.): *Euro IFIP 79*, North Holland, 1979, S. 711–719

[5] Echtler, Florian ; Kaltenbrunner, Martin: SUR40 Linux: Reanimating an Obsolete Tangible Interaction Platform. In: *Proceedings of the 2016 ACM International Conference on Interactive Surfaces and Spaces*. New York, NY, USA : Association for Computing Machinery, 2016 (ISS '16), S. 343–348. – URL https://doi.org/10.1145/2992154.2996778. – ISBN 9781450342483

[6] Echtler, Florian ; Klinker, Gudrun: A Multitouch Software Architecture. In: *Proceedings of the 5th Nordic Conference on Human-Computer Interaction: Building Bridges*. New York, NY, USA : Association for Computing Machinery, 2008 (NordiCHI '08), S. 463–466. – URL https://doi.org/10.1145/1463160.1463220. – ISBN 9781595937049

[7] Holmquist, Lars E. ; Zuckerman, Oren ; Ballagas, Rafael ; Ishii, Hiroshi ; Ryokai, Kimiko ; Zhang, Haiyan: The Future of Tangible User Interfaces. In:

Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems. New York, NY, USA : Association for Computing Machinery, 2019 (CHI EA '19), S. 1–6. – URL https://doi.org/10.1145/3290607.3311741. – ISBN 9781450359719

[8] Ishii, Hiroshi ; Ullmer, Brygg: Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms. In: *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA : Association for Computing Machinery, 1997 (CHI '97), S. 234–241. – URL https://doi.org/10.1145/258549.258715. – ISBN 0897918029

[9] Kaltenbrunner, Martin: *An Abstraction Framework for Tangible Interactive Surfaces*, Dissertation, 01 2018

[10] Kaltenbrunner, Martin ; Bencina, Ross: ReacTIVision: A Computer-Vision Framework for Table-Based Tangible Interaction. In: *Proceedings of the 1st International Conference on Tangible and Embedded Interaction*. New York, NY, USA : Association for Computing Machinery, 2007 (TEI '07), S. 69–74. – URL https://doi.org/10.1145/1226969.1226983. – ISBN 9781595936196

[11] Kaltenbrunner, Martin ; Bovermann, Till ; Bencina, Ross ; Costanza, Enrico u. a.: TUIO: A protocol for table-top tangible user interfaces. In: *Proc. of the The 6th Int'l Workshop on Gesture in Human-Computer Interaction and Simulation* Citeseer (Veranst.), 2005, S. 1–5

[12] Kolb, D.A.: *Experiential learning: experience as the source of learning and development*. Englewood Cliffs, NJ, 1984. – URL http://www.learningfromexperience.com/images/uploads/process-of-experiential-learning.pdf!(dateofdownload:31.05.2006)

[13] Kátai, Zoltán ; Juhász, Katalin ; Adorjáni, Alpár K.: On the role of senses in education. In: *Computers and Education* 51 (2008), Nr. 4, S. 1707–1717. – URL https://www.sciencedirect.com/science/article/pii/S0360131508000663. – ISSN 0360-1315

[14] Magalhães, José P. ; Koops, Hendrik V.: Functional Generation of Harmony and Melody. In: *Proceedings of the 2nd ACM SIGPLAN International Workshop on Functional Art, Music, Modeling and Design*. New York, NY, USA : Association for Computing Machinery, 2014 (FARM '14), S. 11–21. – URL https://doi.org/10.1145/2633638.2633645. – ISBN 9781450330398

[15] Rodić, Lea D. ; Granić, Andrina: Tangible interfaces in early years' education: a systematic review. In: *Personal and Ubiquitous Computing* 26 (2022), Feb, Nr. 1, S. 39–77. – URL https://doi.org/10.1007/s00779-021-01556-x. – ISSN 1617-4917

[16] Rohrmeier, Martin: A generative grammar approach to diatonic harmonic structure. In: *Proceedings of the 4th Sound and Music Computing Conference* (2007), 01

[17] Shaer, Orit ; Hornecker, Eva: Tangible User Interfaces: Past, Present, and Future Directions. In: *Found. Trends Hum.-Comput. Interact.* 3 (2010), jan, Nr. 1–2, S. 1–137. – URL https://doi.org/10.1561/1100000026. – ISSN 1551-3955

[18] Sikora, Frank: *Neue Jazz-Harmonielehre Buch. Buch.* Mainz : Schott, 2003. – ISBN 3795751241

[19] Tanenbaum, Andrew S. ; Steen, Maarten van: *Distributed Systems: Principles and Paradigms*. 2. Upper Saddle River, NJ : Pearson Prentice Hall, 2007. – ISBN 978-0-13-239227-3

[20] Wallace, Benedikte ; Martin, Charles: *Comparing Models for Harmony Prediction in an Interactive Audio Looper*. S. 173–187, 03 2019. – ISBN 978-3-030-14811-9

[21] Xu, Pan: Research on the Application of Computer Music Software in College Traditional Music Course. In: *Journal of Physics: Conference Series* 1992 (2021), aug, Nr. 2, S. 022178. – URL https://doi.org/10.1088/1742-6596/1992/2/022178

[22] Zimmermann, H.: OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection. In: *IEEE Transactions on Communications* 28 (1980), Nr. 4, S. 425–432

A Anhang

A.1 TUIO Protokoll

```
/tuio/2Dobj source application@address
     1.1     1.2         1.3
/tuio/2Dobj alive s_id0 ... s_idN
             2.1         2.2
/tuio/2Dobj set sessionID classID xPos yPos xVel \
                yVel rotationVel motionAccel rotationAccel
            3.1          3.2
/tuio/2Dobj fseq id
            4.1  4.2
```

1. **source**-Message (optional):
   1.1. Profil (Bezeichner) für Objekttyp (Alternativen z.B. /tuio/2Dcur für Cursor/Finger und /tuio/2Dblb für Blob/Freiform)
   1.2. Nachrichtentyp “source”
   1.3. Benutzer/Anwendung und Adresse des Absenders
2. **alive**-Message:
   2.1. Nachrichtentyp “alive”
   2.2. Liste der SessionIDs aller aktuell detektierten Objekte
3. **set**-Message:
   3.1. Nachrichtentyp “set”
   3.2. Informationen über den Objektzustand

- sessionID: Das Objekt erhält für den Zeitraum der Präsenz eine eindeutige ID, um Objekte mit derselben markerID differenzieren zu können.
- classID: Das Objekt wird beispielsweise per *fiducial marker* einer Klasse/Funktion zugewiesen. Diese ist meist einzigartig, kann aber auch mehrfach vergeben sein, da mittels sessionID eine eindeutige Zuweisung möglich ist.
- xPos yPos: x- und y-Koordinaten des Objekts
- xVel yVel rotationVel: x-, y- und Rotationsgeschwindigkeit des Objektes
- motionAccel rotationAccel: (Rotations-)beschleunigung des Objekts

4. **fseq**-Message:

   4.1. Nachrichtentyp “fseq”

   4.2. eindeutige frame sequence ID (int32)

A.2 Abbildungen

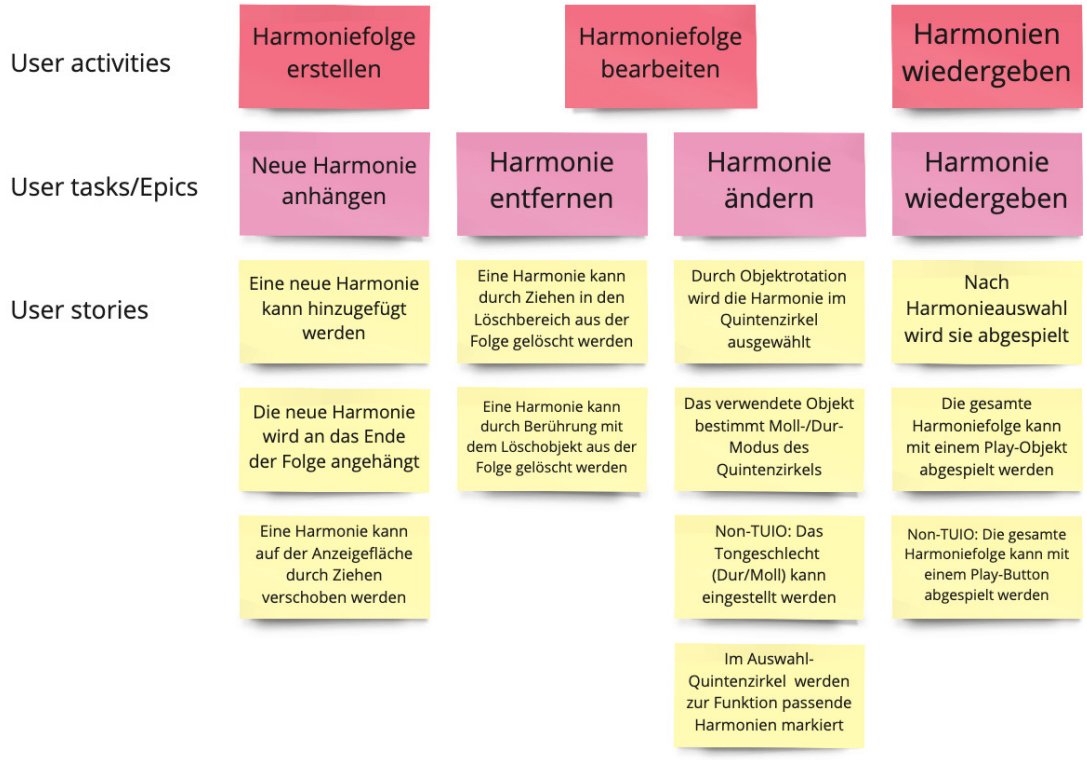
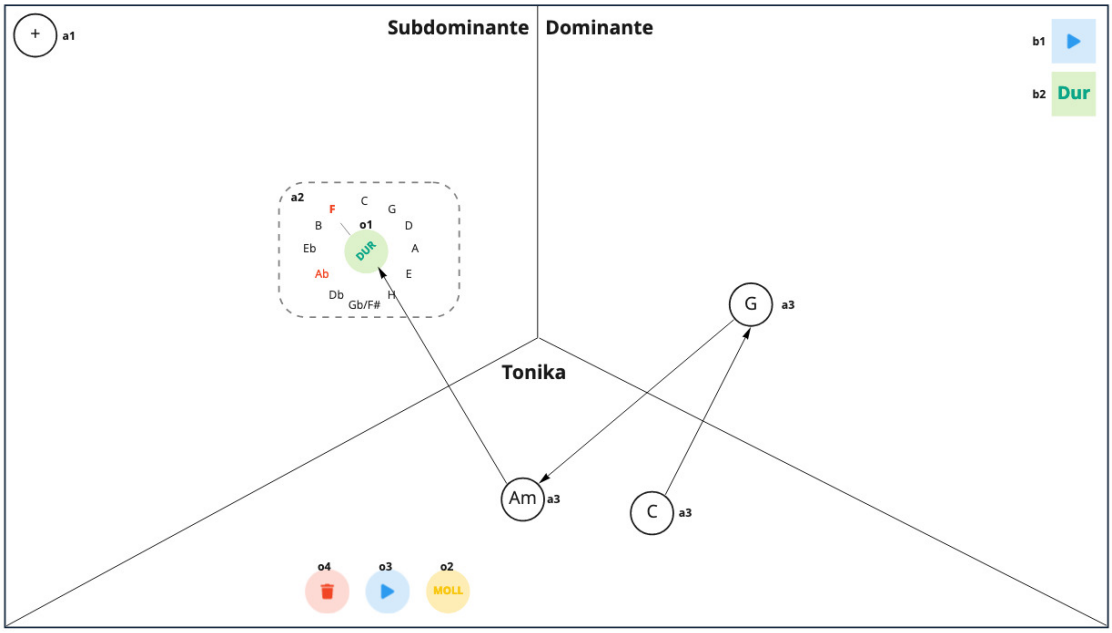


Abbildung A.1: UserStoryMap für das Musiklernprogramm



TUIO Objekte (TUIO Modus)
o1 Dur-Auswahlobjekt liegt auf dem Tisch
o2 Moll-Auswahlobjekt liegt auf dem Tisch
o3 Play-Objekt liegt auf dem Tisch
o4 Löschobjekt liegt auf dem Tisch

Anzeige-Elemente
a1 Triggerfläche: Neue Harmonie hinzufügen (neue a3 herausziehen)
a2 Quintenzirkel Harmonieauswahl aktiv (F und Ab werden vorgeschlagen, F ist ausgewählt)
a3 Harmonie der Folge (verschiebbar, editierbar)

Toolbar Buttons (Non-TUIO Modus)
b1 Harmoniefolge abspielen/stoppen
b2 Dur-/Moll-Auswahlmodus für Quintenzirkel umschalten

Abbildung A.2: Wireframe für das Musiklernprogramm

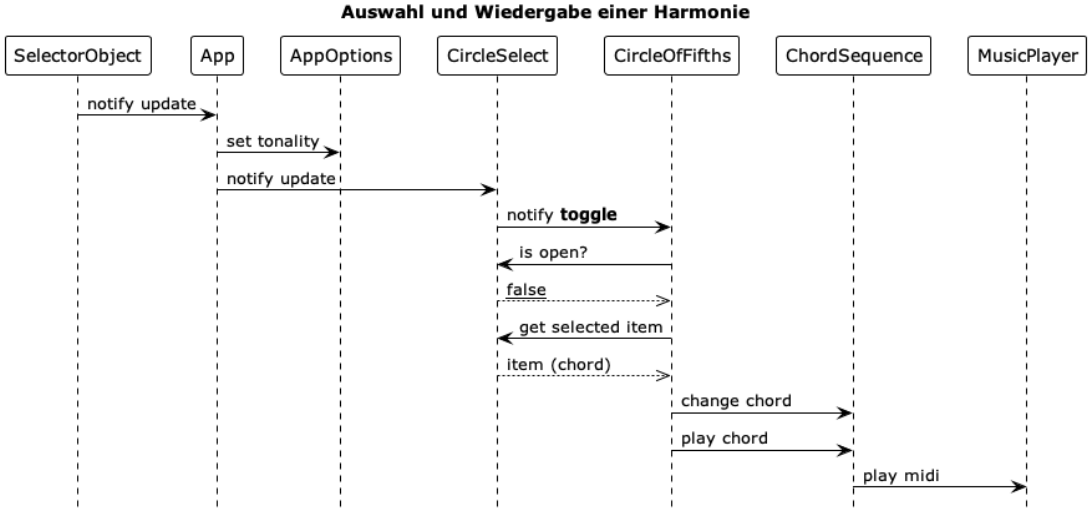


Abbildung A.3: Sequenzdiagramm: Auswahl und Wiedergabe einer Harmonie

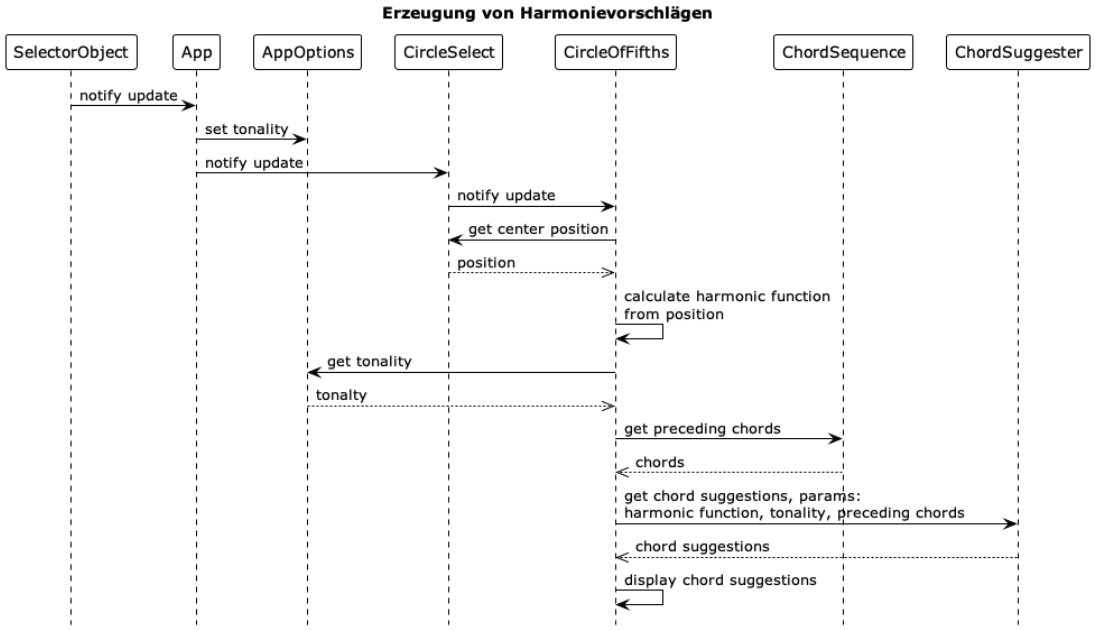


Abbildung A.4: Sequenzdiagramm: Erzeugung von Harmonievorschlägen

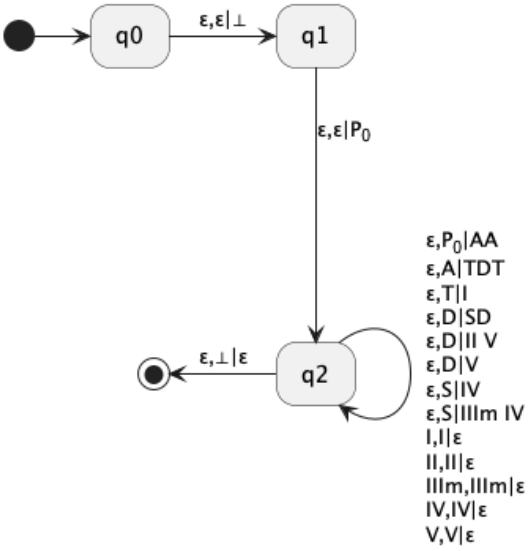


Abbildung A.5: Nichtdeterminisitischer Kellerautomat (Pushdown Automata) für die kontextfreie Grammatik in 3.1

A.3 Informationen für Entwickler

A.3.1 SUR40 mit Ubuntu betreiben

Werksseitig wurde das SUR40 mit Windows 7 ausgeliefert. Die Installation eines alternativen Betriebssystems wie Ubuntu ist unproblematisch mit bekannten Methoden parallel zu Windows oder alleinstehend möglich. Allerdings kann der Betrieb insbesondere dann mit Komplikationen verbunden sein, wenn man spezielle Hardware wie die *PixelSense* Sensorik für die Bilderfassung des Bildschirms nutzen möchte.

Bei grundsätzlichen Problemen mit der Erreichbarkeit von Bilddaten des SUR40 sollte mit Kommandozeilentools wie *lsusb* und *v2l4-utils* geprüft werden, ob die Datenquelle als Eingabegerät erkannt wurde. Unter Umständen kann es nützlich sein, im System-Log (*/var/log/syslog* bzw. *dmesg*) nach Problemen beim Laden des Kernel-Treibers "sur40" [1] zu suchen. Für Debugging-Zwecke kann der Entwickler den Treiber auch manuell um Lognachrichten erweitern, kompilieren und via *modprobe* installieren.

A.3.2 reacTIVision auf dem SUR40 nutzen

Die Bildanalyse-Software reacTIVision [2] ist laut Hersteller kompatibel mit Windows und Linux-Distributionen. Für den Betrieb unter Ubuntu ist es notwendig, dass einige benötigte Treiber manuell installiert werden müssen. Fehlende Module lassen sich am einfachsten aufspüren, indem das reacTIVision Projekt [3] lokal auf dem SUR40 kompiliert wird, sodass benötigte Abhängigkeiten aufgrund von Compilerfehlern und -warnungen auffallen und nachträglich installiert werden können.

Im aktuellen Stand von reacTIVision muss das Kamera-Gerät unter */dev/video** verortet sein. In neueren Kernelversionen wird das Gerät jedoch beispielsweise den Namen */dev/v4l-touch0* tragen. Um dieses Problem einfach zu umgehen, kann ein symbolischer Link */dev/video0 -> /dev/v4l-touch0* erzeugt werden. Beim Start von reacTIVision sollte schließlich das Kamerabild des Bildschirms zu sehen sein.

[1] https://github.com/torvalds/linux/blob/master/drivers/input/touchscreen/sur40.c
[2] http://reactivision.sourceforge.net/
[3] https://github.com/mkalten/reacTIVision

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original