Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Marcel Schöneberg

Eine auf Plug-Ins basierende Anwendungsarchitektur zur Verarbeitung von verschiedenen Eye-Tracking Datenformaten

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Marcel Schöneberg

Eine auf Plug-Ins basierende Anwendungsarchitektur zur Verarbeitung von verschiedenen Eye-Tracking Datenformaten

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. Zukunft
Zweitgutachter: Prof. Dr. Sarstedt

Eingereicht am: 11. November 2012

Marcel Schöneberg

Thema der Arbeit
Eine auf Plug-Ins basierende Anwendungsarchitektur zur Verarbeitung von verschiedenen Eye-Tracking Datenformaten

Stichworte
Architektur, Eye-Tracker, Eye-Tracking, Plug-In basiert

Kurzzusammenfassung
Dieses Dokument behandelt den Entwurf einer auf Plug-Ins basierenden Anwendungsarchitektur zur Verarbeitung von Datensätzen eines Eye-Trackers.

Der Fokus der Architektur liegt hierbei auf der Austauschbarkeit der Datenquellen (Eye-Tracker, Maus, Touch-Screen etc.), der entsprechenden Konvertierung in ein allgemeines Datenformat, sowie der Weiterverarbeitungsmethode (reine Ausgabe der Daten, statistische Analyse, Visualisierung etc.). Daher entkoppelt die Architektur diese Komponenten durch Plug-Ins, sodass die Weiterverarbeitung unabhängig vom Input der Datenquelle erfolgen kann.

Um die Nutzbarkeit der entwickelten Architektur zu untermauern umfasst diese Arbeit eine prototypische Implementierung als Proof-Of-Concept.

Marcel Schöneberg

Title of the paper
A plug in based application architecture to process various eye tracking dataformats

Keywords
architecture, eye tracker, eye tracking, plug in based

Abstract
This document deals with the design of a plug in based architectur to process eye tracking data.

The focus of the architecture is the replacebility of datasources (eye tracker, mouse, touch screen, etc.) , according conversion into a generalized dataformat and processing method (plain output of data, statistical analyse, visualisation etc.). Therefore the architecture decouples these components by using plug ins, so that the processing can take place independently of the input.

To confirm the usability of this architecture this thesis contains a protoypical implementation as a proof of concept.

Inhaltsverzeichnis

1. Einleitung

1.1. Motivation

Die Benutzung von Software ist fester Bestandteil des alltäglichen Lebens geworden. Ob auf der Arbeit eine Office-Suite oder in der Freizeit eine App auf dem Handy - Applikationen umgeben uns überall.

Ein wichtiger Aspekt von Software ist daher die Benutzbarkeit (Usability) ihrer graphischen Oberfläche (GUI). Ist diese stark ablenkend behindert sie das effektive Arbeiten [vgl. ISO 9241-1, S. 4]. Ebenso kann eine Oberfläche dazu verwendet werden Werbung zu platzieren (z.B. auf Webseiten). Daher besteht für Werbeträger ein großen Interesse, dass diese Platzierungen im Kontext von Usability-Untersuchungen gut gewählt sind [vgl. Batra/Bishu07; Tobii-c, S. 2].

Aufgrund der obigen Aspekte sind daher Usability-Untersuchungen einer Software sehr wichtig, da diese Erkenntnisse darüber liefern können wie gut eine Software u.a. die obigen Kriterien erfüllt [vgl. Batra/Bishu07; Tobii-c, S. 2].

Eine Möglichkeit solcher Usability-Untersuchungen ist das Verfolgen der Augenbewegungen bzw. des Blickpunktes einer Testperson. Diese Aufgabe können so genannte Eye-Tracker (siehe: Kapitel 2.1) erfüllen. Diese zeichnen auf wohin eine Testperson schaut (z.B. auf einem Monitor). Die auf diese Weise gesammelten Daten können auf verschiedene Weisen ausgewertet bzw. visualisiert werden. Basierend auf diesen Daten können wichtige Rückschlüsse über die Usability einer Software getroffen werden. Diese können Softwareentwickler, sowie Designer helfen eine Applikation zu dementsprechend zu verbessern.

1.2. Problemstellung

Die oben beschriebenen Untersuchungen benötigen eine Software welche sie durchführt. Das Problem besteht nun aber in der Vielfältigkeit der Datenquellen (neben den bereits erwähnten Eye-Trackern sind ebenso Mäuse, Touchscreens usw. denkbar – im Fokus dieser Arbeit stehen allerdings Eye-Tracker). Begrenzt man die Auswahl an Eingabequellen auf Eye-Tracker so stehen immer noch diverse Hersteller mit unterschiedlichen Modellen zur Verfügung.

Die Steuerung (also u.a. auch die Datenbeschaffung) dieser verschiedenen Geräte ist ebenso vielfältig wie die Formate in denen die Daten zur Verfügung gestellt werden. Da kein allgemeiner Standard existiert [vgl. Hennessey/Duchowski10, S. 1] entwickelt jeder Hersteller daher eigene Methoden. Daher braucht jedes Gerät die ihm entsprechende Software um die Beschaffung und Auswertung der Daten zu realisieren.

Ähnlich verhält es sich mit der Weiterverarbeitung der gelieferten Daten. Diese kann auf vielfältige Weise geschehen. Eine der einfachsten Möglichkeiten ist es die gesammelten Daten in einem entsprechenden Format auszugeben. Genauso denkbar ist aber auch eine statistische Analyse mit anschließender Visualisierung [vgl. Phan11, Kapitel 4.1.6]. Weitere komplexere Anwendungsfälle wären die Steuerung von Anwendungen mit Hilfe der aufgezeichneten Augenbewegungen [vgl. Tietz11].

Die oben beschriebene Vielfältigkeit der Eye-Tracker und Weiterverarbeitungsmöglichkeiten führt zu einer starken Abhängigkeit zwischen der Software (welche einen Eye-Tracker ansteuert und die Weiterverarbeitung durchführt) und der entsprechenden Hardware. Im Endeffekt brauch daher jeder Eye-Tracker eine eigene Software, da das genutzte Datenformat von der jeweiligen Hardware abhängt und damit eine Verallgemeinerung verhindert.

Zur Verdeutlichung des Problems soll folgendes Beispiel dienen: Man stelle sich eine auf Usability-Untersuchungen spezialisierte Firma vor. Diese führt ihre Studien mit verschiedenen Eye-Trackern (bzw. anderen Datenquellen) durch und visualisiert die gelieferten Daten sowohl als Heatmap sowie als Diagramme. Da die Tracker aufgrund des inhärent unterschiedlichen Datenformats jeweils ihre eigene Verarbeitungssoftware benötigen kann die Firma ihre Geschäftsprozesse nicht mit einer einzigen Software vereinfachen.

Der Wunsch nach einer Softwarearchitektur, welches es ermöglicht Daten von allen eingesetzten Datenquellen auf unterschiedliche Weise zu verarbeiten liegt also nahe um zeit- und kosteneffzient arbeiten zu können.

1.3. Ziel und Rahmen dieser Arbeit

Ziel dieser Arbeit ist es zunächst die nötigen Grundlagen des Eye-Tracking zu erläutern. Darauf basierend soll eine Softwarearchitektur entwickelt werden. Diese soll es erlauben sowohl die Datenquelle, die Konvertierung in ein universelles Datenformat sowie die Methode der Weiterverarbeitung flexibel auszutauschen. Um die Austauschbarkeit der Komponenten zu gewährleisten sind diese jeweils als Plug-Ins in die Architektur eingebunden. Daher enthält die Zielarchitektur einen Plug-In Verarbeitungsmechanismus.

Darüber hinaus soll die Architektur diverse nicht-funktionale (vgl. 3.3.2) Anforderungen erfüllen. Eine spätere Implementierung soll möglichst effizient arbeiten und dabei eine hohe Zuverlässigkeit gewährleisten.

Des Weiteren ist es nötig ein (von der Hardware unabhängiges) Datenformat zu entwerfen, welches alle Komponenten verarbeiten können und welches ggf. weitere Eigenschaften (z.B. Erweiterbarkeit, Lesbarkeit für Menschen) erfüllt.

Auch die Entwicklung einer prototypischen Implementierung der Architektur ist im Rahmen dieser Arbeit vorgesehen. Dabei wird als Datenquelle ein „Tobii X120“-Eyetracker gewählt. Das universelle Zwischenformat basiert auf XML (eXtensible Markup Lanuguage) und die Weiterverarbeitung wird durch eine Heatmap-Visualisierung (siehe: 5.1 realisiert.

Das Ende der vorliegenden Arbeit wird eine Evaluierung der entworfenen Architektur sowie ggf. darauf basierender Verbesserungen und Erweiterungen bilden.

Folgende Aspekte werden im Rahmen dieser Arbeit nicht behandelt: Eine detaillierte Ausarbeitung der Verwendung anderer Datenquellen und Weiterverarbeitungsmethoden. Auch mögliche entsprechende Erweiterungen des Datenformates werden nicht im Detail berücksichtigt. Weiterhin ist zu betonen, dass die Beispielimplementierung der Architektur nur als Prototyp, nicht aber als Vorlage bzw. vollständig gelungene Umsetzung zu verstehen ist.

1.4. Struktur dieser Arbeit

Diese Arbeit gliedert sich grob in sechs Kapitel.

Das vorliegende erste Kapitel dient der Einführung in die Thematik, sowie der Beschreibung der Problematik. Darauf basierend wird kurz der Rahmen der Arbeit abgesteckt.

Kapitel 2 erläutert die für diese Arbeit gewählte Domäne und stellt daraufhin die benutzten Technologien vor.

Kapitel 3 zeigt auf welche Anforderungen an die zu entwickelnde Architektur bzw. ihre Implementierung gestellt werden.

Kapitel 4 behandelt die zu entwerfende Architektur im Detail und geht dabei auf die Einzelkomponenten und deren Bestandteile ein.

Kapitel 5 erläutert Fragmente des entwickelnden Prototypen, stellt ein grobes Testkonzept vor und evaluiert daraufhin das Ergebnis der Entwicklungen.

Kapitel 6 bietet ein Fazit der vorliegenden Arbeit. Hierfür werden zunächst die Grundideen der Arbeit noch einmal aufgegriffen. Daraufhin wird eine Zusammenfassung samt einem möglichen Ausblick auf zukünftige Entwicklungen dieser Arbeit geboten.

2. Grundlagen

Dieses Kapitel erläutert grundlegende Begrifflichkeiten aus dem Kontext der Architektur und zählt Technologien auf welche zur Umsetzung benutzt wurden.

2.1. Was bedeutet Eye-Tracking

Der Begriff des Eye-Tracking bezeichnet im allgemeinen das Aufzeichnen der Blickrichtung/Augenposition einer Testperson [vgl. Duchowski07, S. 51; Tobii-a]. Dieses kann auf unterschiedliche Arten und Weisen passieren – ein grundlegendes Verfahren wird im Abschnitt 2.1 umrissen.

Während des Tracking-Vorganges fallen verschiedenste Daten an.

- Zu den wichtigsten gehört dabei die Fixation. Dieses ist eine Fixierung auf einen Punkt der kurzweilig betrachtet wird [vgl. Duchowski07, S. 46; Tobii-a, S. 6]. Die Fixation wird hierbei als X/Y-Koordinatensatz repräsentiert.

- Das „Gegenstück“ zur Fixation ist die Sakkade. Sie bezeichnet eine schnelle Augenbewegung mit dem Ziel das Auge neu zu positionieren [Duchowski07, S. 42]. Aufgrund ihrer kurzen Dauer von 10 ms bis 100 ms sind die Sakkaden für eine spätere Analyse der Daten zunächst von eher geringer Bedeutung.

- Darüber hinaus kann der Eye-Tracker Meta-Informationen zu dem jeweiligen Datenwert liefern (ggf. für jedes Auge einzeln) [vgl. Hennessey/Duchowski10, S. 3; Tobii-e, S. 34]. Diese Informationen können z.B. Erfassungszeitpunkt einer Fixation und Validität des X/Y-Wertepaares umfassen (ein Wertepaar kann z.B. durch das nicht-Erfassen eines/-beider Auges verfälscht werden). Ein weiteres Merkmal welches exemplarisch geliefert

werden könnte ist die Pupillengröße oder die Position des Auges im Raum. Alle diese Informationen können dann im nächsten Schritt zur Analyse benutzt werden.

Ein Gerät das die obige Aufgabe erfüllt und diese Daten liefert wird als Eye-Tracker bezeichnet.

Grundlegende Funktionsweise

Es gibt verschiedene Techniken um Eye-Tracking zu realisieren, eine davon ist das sogenannte Pupil Centre Corneal Reflection (PCCR). Da dieses die Basis des im Praxisteil verwendeten „Tobii“ Trackers ist soll es im Folgenden kurz umrissen werden.

Die grundlegende Idee des PCCR basiert darauf Reflektionen von Licht auf den Pupillen bzw. den Augen-Hornhäuten einer Versuchsperson mit Hilfe einer Kamera aufzuzeichnen (siehe Abbildung 2.1).

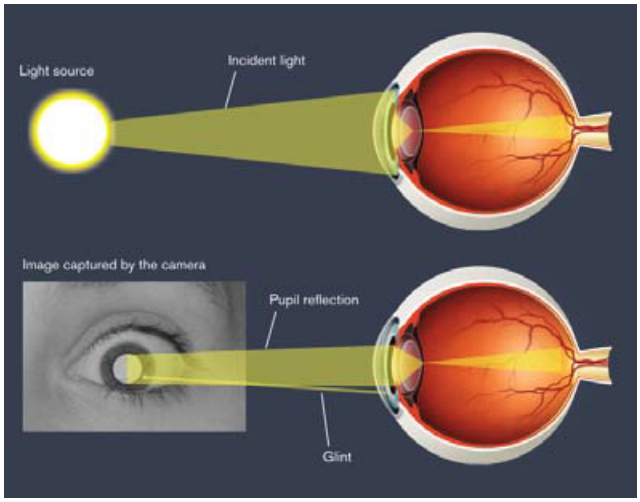

Abbildung 2.1.: Eine Lichtquelle (Eye-Tracker) bestrahlt die Hornhaut und die Pupille der Versuchsperson. Die Reflektionen werden zur Berechnung der Blickrichtung benutzt [Tobii-a]

.

Während des Trackingvorgangs werden so verschiedene Daten festgehalten. Zu diesen gehören u.a. das Reflektionsmuster auf der Pupille, der Winkel in dem sich die Pupillen zum Tracker befinden und der Entfernung zum Eye-Tracker. Auf der Basis dieser Daten ist es nun mit

Hilfe diverser Algorithmen möglich einen Vektor zu berechnen, welcher die Blickrichtung der Testperson beschreibt [vgl. Duchowski07, S. 54; Tobii-a, Abschnitt 2.1].

Diesen Vektor kann man nun nutzen um zu berechnen welchen Punkt eine Versuchsperson auf einer für den Tracker kalibrierten Fläche betrachtet

2.2. Benutzte Technologien

Die folgenden Abschnitte führen abstrakt die wichtigen im Kontext dieser Arbeit benutzten Technologien ein. Hierbei werden - sofern möglich - sowohl die theoretischen Ideen einer Technologie sowie die konkret benutzte Implementierung vorgestellt. Dieses dient der Vermeidung von Problemen aufgrund inkompatibler Softwareversionen während einer möglichen Re-Implementierung.

2.2.1. Eye-Tracker

In der prototypischen Implementierung der zu entwerfenden Architektur wird als Datenquelle ein Eye-Tracker des Modells „X120“ der Firma „Tobii“ benutzt (siehe: Abb. 2.2).

Abbildung 2.2.: Ein Tobii X120 Eye-Tracker. [Tobii-d]

Dieser ist ein stationärer Eye-Tracker welcher über die ihm beiliegende Software (Tobii Studio) oder das „Tobii“ Software Development Kit (SDK) angesteuert werden kann. Als Zugriffspunkt

dient dem „X120“ u.a. ein Ethernet-Port. Durch Nutzung des TCP/IP Protokolls kann man darüber mit dem „X120“ kommunizieren. [Tobii-b].

Die Version der Tracker-Firmware war zum Zeitpunkt der Implementierung 2.0.7.

2.2.2. Software

2.2.2.1. Message orientierte Middleware – Apache ActiveMQ

Aufgrund der potentiell großen Datenmengen welche während der Verarbeitung von Eye-Tracker Daten auftreten können als wird ein Puffermechanismus in Form einer Message orientierten Middleware (MoM) benötigt. Diese soll dem eventuellen Verlust von Daten aufgrund von unzureichender Verarbeitungsgeschwindigkeit entgegenzuwirken. Der Datenverlust kann sich u.a. darin äußern, dass Daten zum Zeitpunkt x zwar vom Eye-Tracker erfasst werden, allerdings nicht direkt von einer Form der Weiterverarbeitung entgegen genommen werden können. Daher würde die Weiterverarbeitung beim entgegennehmen/abholen der Daten statt des Datensatzes zum Zeitpunkt x die Daten des Zeitpunktes $x+i$ erhalten. Der Datensatz x wäre daher unwiderruflich verloren, sofern er nicht zwischengespeichert wird. Dieser Puffer-Mechanismus wird durch den Apache ActiveMQ Broker [siehe: ActiveMQ]. (Version: 5.6.0) bereitgestellt.

Da der ActiveMQ-Broker nur die Serverfunktionalität der Message orientierten Middleware (MoM) bietet wird zur Clientanbindung der Applikation eine Implementierung des C++ Messaging Service (CMS) verwendet. Im Rahmen dieser Arbeit ist dazu Version 3.4.4 von Apache ActiveMQ-CPP [siehe: ActiveMQ-CPP] zum Einsatz gekommen.

2.2.2.2. Apache Portable Runtime

Die Apache Portable Runtime (APR) Libraries [siehe: APR] werden vom ActiveMQ-Client zwingend zur Kompilation benötigt. Diese Libraries stellen diverse APIs u.a. für betriebssystemnahe Operationen wie Thread-/Prozessverwaltung zur Verfügung.

Die APR-Bibliotheken sind aufgeteilt in drei Unterprojekte die jeweils in der angegebenen Version benutzt wurden: APR (Version: 1.4.6), APR-iconv (Version: 1.2.1), APR-util (Version: 1.4.1).

2.2.2.3. Boost

Die Boost Libraries [siehe: Boost] stellen diverse Funktionalitäten zur Verfügung, welche im C++ Standard nicht enthalten sind. Unter anderem bieten die Libries die Möglichkeit zum Netzwerkzugriff, zur Thread-/Prozessverwaltung sowie zur Erstellung von Funktionsobjekten. Diese Funktionalitäten werden u.a. innerhalb der Tobii SDK benutzt.

Boost wurde in Version 1.50.0 verwendet.

2.2.2.4. XML Parser – Apache Xerces

In weiten Teilen der Architektur wird zur Verarbeitung von Daten die Extensible Markup Language (XML) benutzt. Dieses ist nötig da im Rahmen der Datenverarbeitung XML-Daten u.a. eingelesen und für weitere Arbeitsschritte aufgearbeitet werden.

Um diese Verarbeitung zu gewährleisten entschied sich der Autor das Xerces-Framework [siehe: Xerces] der Apache Software Foundation zu benutzen. In dieser Arbeit kam das Unterprojekt „Xerces-C++“ in der Version 2.7.0 zum Einsatz.

2.2.2.5. XPath – Apache Xalan

Im Rahmen der Weiterverarbeitung von XML basierten Daten wird im praktischen Teil der Arbeit die XML Path Language (XPath) eingesetzt. Diese definiert eine Abfragesprache auf XML-Daten, welche es erlaubt Teile eines XML-Dokuments zu adressieren.

In dieser Arbeit wird daher Apache Xalan-C [siehe: Xalan] in der Version 1.10.0 eingesetzt. Dieses Projekt bietet u.a. eine Implementierung des XPath 1.0 Standards.

2.2.2.6. Heatmap-Visualisierung – R

Im Rahmen dieser Arbeit ist eine prototypische Implementierung vorgesehen, welche zur Weiterverarbeitung die gewonnenen Daten visualisiert. Dieses soll im Rahmen einer Heatmap geschehen.

Zur Umsetzung dieses Ziels bedient sich der Autor der Programmiersprache „R“ [vgl. R]. Die Zielsetzung dieser Sprache ist statistisches Rechnen sowie das Erstellen von statistischen Grafiken zu denen auch Heatmaps zählen. ’‘R“ wurde in der Version 2.15.0 verwendet.

2.2.2.7. Microsoft Visual C++ 2010

Der Prototyp welcher ebenfalls im Rahmen dieser Arbeit entwickelt werden soll ist in C++ implementiert.

Als Entwicklungsumgebung kam daher Microsoft Visual Studio 2010 zum Einsatz.

2.2.2.8. Tobii SDK

Als Inputquelle der prototypischen Implementierung dieser Arbeit dient der „Tobii X120“-Eyetracker.

Um den Tracker anzusteuern wird das offizielle „Tobii“ SDK in der Version 3.0.2 verwendet [vgl. Tobii-e]. Dieses Software Development Kit bietet Zugriff auf alle notwendigen Funktionalitäten des „X120“ (u.a. Erkennung im Netzwerk, Kalibrierung und Datenerfassung).

3. Anforderungen an die Software

Das folgende Kapitel umreißt schemenhaft das Gesamtsystem. Darauf aufbauend werden dessen Anwendungsfälle erläutert und im Folgenden die Anforderungen welche gestellt werden erläutert.

3.1. Hauptbestandteile des Systems

Das Gesamtsystem soll die Verarbeitung von Eye-Tracking Daten übernehmen (Erfassung, Konvertierung, Weiterverarbeitung). Der entsprechende Ablauf ist in Grafik 3.1 zu sehen.

1. Zunächst müssen die Daten von einer Datenquelle (im praktischen Teil ein „TobiiX120 Eyetracker) erfasst werden.

2. Daraufhin muss eine Konvertierung der Daten in ein Format erfolgen, welches die Weiterverarbeitung erlaubt. Hierbei muss beachtet werden, dass die Definition des Formates allen späteren Schritten zugänglich sein muss.

3. Im Anschluss an diese beiden Aufgaben kann die eigentliche Weiterverarbeitung beginnen (als Beispiel wurde in dieser Arbeit eine Visualisierung in Form einer Heat-Map gewählt).

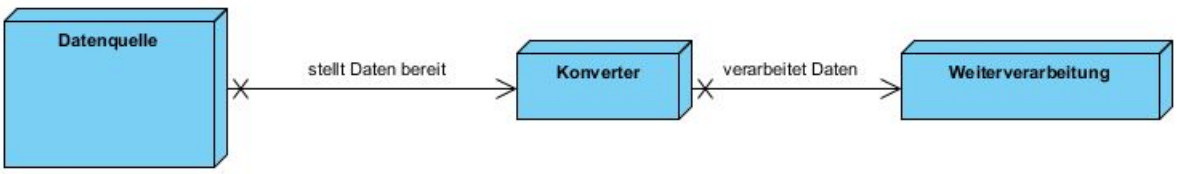

Abbildung 3.1.: Ablauf innerhalb des Gesamtsystems

3.2. Anwendungsfälle

Der folgende Abschnitt erläutert welche Anforderungen in Form von Anwendungsfällen an die Architektur bzw. deren Implementierung gestellt werden.

Die Abbildung 3.2 beschreibt die Hauptanwendungsfälle des Gesamtsystems aus Sicht des Endbenutzers. Hierbei ist anzumerken, dass es vier Gruppen von Use-Cases gibt:

1. Zunächst muss der User eine grundlegende Konfiguration durchführen, d.h er muss die entsprechenden Plug-Ins (also die Datenquelle, die dementsprechende Konvertierung und die Weiterverarbeitung) wählen. Nachdem diese Plug-Ins gewählt wurden erfolgt gegebenenfalls eine Kalibrierung der Datenquelle.

2. Ist dieses erfolgreich abgeschlossen so kann der User die Datenerfassung des Hardware-Adapter Plug-Ins starten. Hierzu wird dem Eye-Tracker das Signal übermittelt, dass die Konfiguration abgeschlossen ist. Dieser kann daraufhin seine Arbeit verrichten.

3. Weiterhin kann nach der Grundkonfiguration die Konvertierung innerhalb des entsprechenden Plug-Ins gestartet werden. Deren Aufgabe besteht darin die „Roh“-Datensätze des Trackers in ein allgemeines Format zu überführen.

4. Darüber hinaus muss auch die Weiterverarbeitung im dafür geladenen Plug-In eingeleitet werden. Diese überführt das allgemeine Format z.B. in eine Heatmap-Visualisierung der vom Tracker aufgezeichneten Datensätze.

   Ebenso ist es natürlich möglich die jeweiligen Schritte (Erfassung, Konvertierung, Verarbeitung) zu beenden.

Die jeweiligen Use-Cases werden im Folgenden noch einmal tabellarisch erläutert.

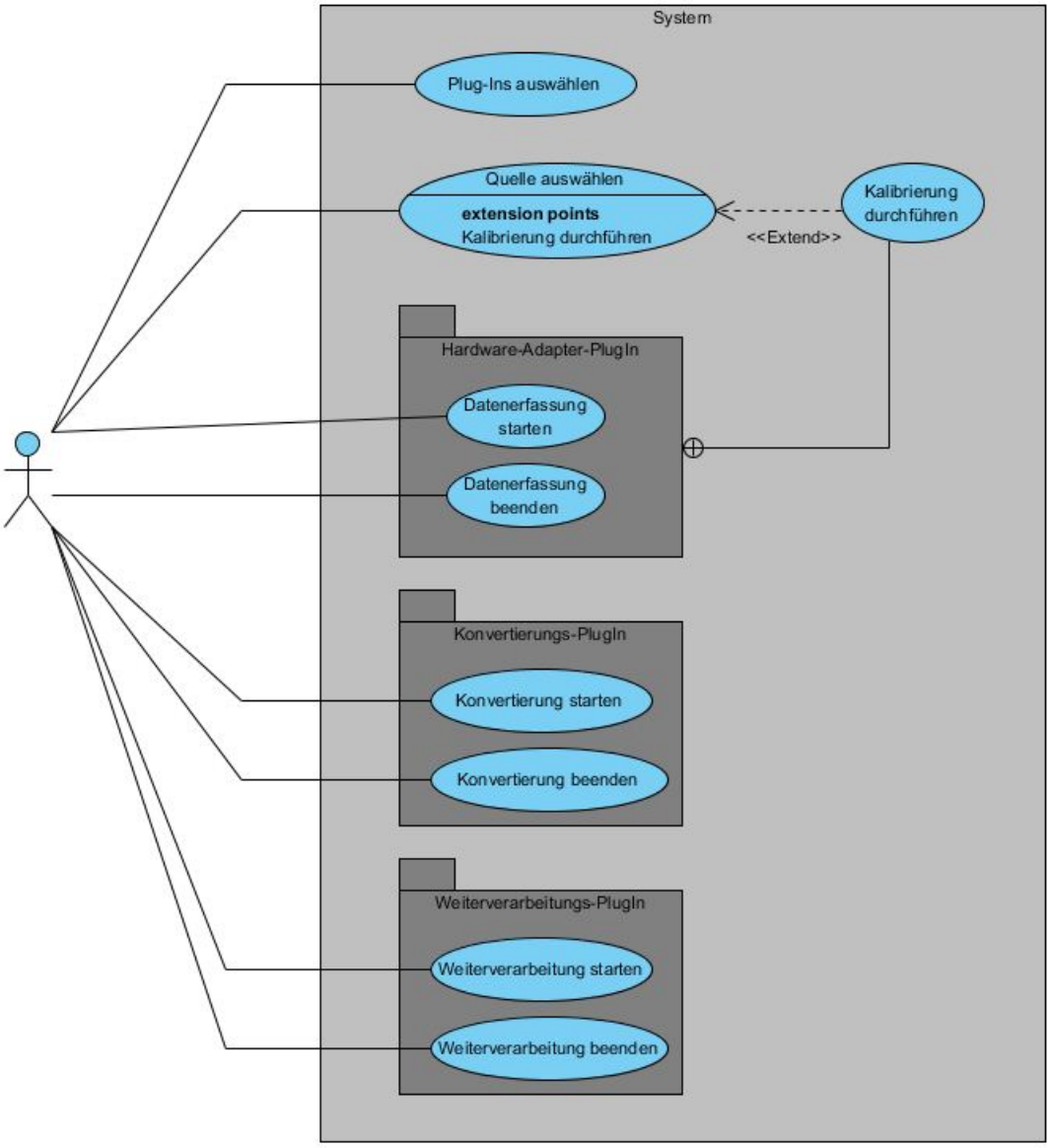

Abbildung 3.2.: Use-Case Diagramm des Systems

Tabelle 3.1.: Use-Case: Plug-Ins auswählen

Punkt	Beschreibung
Titel	Plug-Ins auswählen
Akteur	Benutzer
Auslöser	User startet die Konfiguration
Vorbedingung	1. Wissen über verfügbare Plug-Ins vorhanden 2. Jeweils mindestens ein Plug-In pro Kategorie (Datenquelle, Konvertierung, Weiterverarbeitung) vorhanden
Nachbedingung	Jeweils ein Plug-In pro Kategorie verfügbar
Erfolgsszenario	1. Benutzer wählt jeweils ein Plug-In pro Kategorie 2. Gewählte Plug-Ins werden geladen 3. Plug-Ins werden initialisiert 4. Plug--Ins können verwendet werden
Erweiterungen	–
Fehlerfälle	1. Plug-In nicht ladbar 2. Plug-In nicht initialisierbar
Häufigkeit	einmalig
Anmerkungen	Das Laden (im Sinne von „verfügbar machen“) eines Plug-Ins muss unterschieden werden von der Initialisierung („verwendbar machen“) eben dessen.

Tabelle 3.2.: Use-Case: Datenquelle auswählen

Punkt	Beschreibung
Titel	Datenquelle auswählen
Akteur	Benutzer
Auslöser	Plug-In für Datenquelle ausgewählt
Vorbedingung	Plug-In für Datenquelle wurde erfolgreich geladen
Nachbedingung	Datenquell-PlugIn ist verwendbar
Erfolgsszenario	1. Benutzer hat die Wahl zwischen evtl. mehreren Datenquellen welche zu dem geladenen Plug În passen 2. Benutzer wählt eine spezifische Datenquelle 3. Datenquelle ist verwendbar
Erweiterungen	Kalibrierung der Datenquelle durchführen
Fehlerfälle	1.Keine kompatiblen Datenquellen vorhanden 2. Gewählte Datenquelle kann nicht initialisiert werden
Häufigkeit	einmalig
Anmerkungen	Ein Plug-In für beispielsweise einen Eye-Tracker könnte mehrere zur Verfügung stehende Geräte finden. Es ist aber sinnvoll nur von einem Daten zu empfangen, daher muss ggf. eine Auswahl getroffen werden.

Tabelle 3.3.: Use-Case: Kalibrierung der Datenquelle durchführen

Punkt	Beschreibung
Titel	Kalibrierung durchführen
Akteur	Benutzer
Auslöser	Die Kalibrierung der Datenquelle wird gestartet
Vorbedingung	Datenquelle ist verfügbar (initialisiert)
Nachbedingung	Datenquelle ist kalibriert und einsatzbereit
Erfolgsszenario	1. Kalibrierung wird gestartet 2. Benutzer führt die spezifischen Maßnahmen zur Kalibrierung (z.B. festlegen der Betrachtungsfläche, Erstellen von Kalibrierungspunken etc.) durch 3. Datenquelle ist einsatzbereit
Erweiterungen	–
Fehlerfälle	Kalibrierung kann nicht durchgeführt werden (z.B. durch Probleme bei der Ansteuerung der Datenquelle, ungültige Daten)
Häufigkeit	pro Datenquelle einmalig
Anmerkungen	Die genaue Funktionsweise der Kalibrierung ist abhängig von der Datenquelle bzw. deren Plug-In

Tabelle 3.4.: Use-Case: Datenerfassung starten

Punkt	Beschreibung
Titel	Datenerfassung starten
Akteur	Benutzer
Auslöser	User startet startet die Datenerfassung
Vorbedingung	1. Plug-Ins initialisiert 2. Datenquelle ist einsatzbereit
Nachbedingung	Datenerfassung läuft
Erfolgsszenario	1. Datenerfassung wird gestartet 2. Datenquelle liefert „Roh“-Datensätze an die Konvertierung
Erweiterungen	–
Fehlerfälle	1. Probleme bei der Ansteuerung der Datenquelle (z.B. keine Verbindung) 2. Daten können nicht an die Konvertierung weitergeleitet werden
Häufigkeit	einmalig pro initialisierter Datenquelle, ansonsten beliebig oft
Anmerkungen	Die Datenerfassung kann für beliebig viele konfigurierte Datenquellen (auch äquivalente) gestartet werden.Daher können beispielsweise mehrere Konverter mit Daten versorgt werden

Tabelle 3.5.: Use-Case: Datenerfassung beenden

Punkt	Beschreibung
Titel	Datenerfassung beenden
Akteur	Benutzer
Auslöser	User beendet die Datenerfassung
Vorbedingung	Datenerfassung läuft
Nachbedingung	Datenerfassung ist beendet
Erfolgsszenario	1. Datenerfassung wird beendet
	2. Es werden keine Datensätze mehr an die Konvertierung ausgeliefert
Erweiterungen	–
Fehlerfälle	1. Probleme bei der Ansteuerung der Datenquelle
	2. Versuch eine Datenerfassung zu beenden wenn keine gestartet wurde
Häufigkeit	pro gestarteter Datenerfassung einmalig
Anmerkungen	–

Tabelle 3.6.: Use-Case: Konvertierung starten

Punkt	Beschreibung
Titel	Konvertierung starten
Akteur	Benutzer
Auslöser	User startet die Konvertierung
Vorbedingung	1. Plug-Ins sind initialisiert
	2. Eine Datenquelle liefert „Roh“-Datensätze
Nachbedingung	Die Konvertierung ist gestartet
Erfolgsszenario	1. Konvertierung ist gestartet
	2. Der Konverter liefert Datensätze an die Weiterverarbeitung
Erweiterungen	–
Fehlerfälle	1. Fehler innerhalb der spezifischen Konvertierungsmethode
	2. Keine verfügbaren „Roh“-Datensätze
Häufigkeit	beliebig oft, sofern entsprechende Datenquelle vorhanden sind
Anmerkungen	Die Konvertierung setzt auf Daten der Datenquelle auf. Dementsprechend muss eine solcher vorhanden sein um eine korrekte Funktionsweise zu garantieren

Tabelle 3.7.: Use-Case: Konvertierung beenden

Punkt	Beschreibung
Titel	Konvertierung beenden
Akteur	Benutzer
Auslöser	User beendet die Konvertierung
Vorbedingung	Konvertierung läuft
Nachbedingung	Konvertierung ist beendet
Erfolgsszenario	1. Konvertierung ist beendet
Erweiterungen	–
Fehlerfälle	1. Versuch eine Konvertierung zu beenden wenn keine gestartet wurde
Häufigkeit	beliebig oft, falls eine Konvertierung gestartet ist
Anmerkungen	–

Tabelle 3.8.: Use-Case: Weiterverarbeitung starten

Punkt	Beschreibung
Titel	Weiterverarbeitung starten
Akteur	Benutzer
Auslöser	User startet die Weiterverarbeitung
Vorbedingung	1. Plug-Ins sind initialisiert 2. Ein Konverter liefert XML-Datensätze
Nachbedingung	Die Datenweiterverarbeitung ist gestartet
Erfolgsszenario	1. Weiterverarbeitung ist gestartet 2. Das Ergebnis der Weiterverarbeitung steht bereit
Erweiterungen	–
Fehlerfälle	1. Fehler innerhalb der spezifischen Weiterverarbeitungsmethode 2. Keine verfügbaren XML-Datensätze
Häufigkeit	beliebig oft, sofern entsprechende Konverter vorhanden sind
Anmerkungen	Die Weiterverarbeitung setzt auf Daten des Konverters auf. Dementsprechend muss ein solcher vorhanden sein um eine korrekte Funktionsweise zu garantieren

Tabelle 3.9.: Use-Case: Weiterverarbeitung beenden

Punkt	Beschreibung
Titel	Weiterverarbeitung beenden
Akteur	Benutzer
Auslöser	User beendet die Weiterverarbeitung
Vorbedingung	Weiterverarbeitung läuft
Nachbedingung	Datenverarbeitung ist beendet
Erfolgsszenario	1. Weiterverarbeitung ist beendet
Erweiterungen	–
Fehlerfälle	1. Versuch eine Weiterverarbeitung zu beenden wenn keine gestartet wurde
Häufigkeit	beliebig oft, falls eine Weiterverarbeitung gestartet ist
Anmerkungen	–

3.3. Anforderungen an die Architektur

Im folgenden Abschnitt werden auf Basis der obigen Erklärungen und Anwendungsfälle die Anforderungen welche die Architektur bzw. ihre spätere Implementierung erfüllen muss vorgestellt.

Hierbei gibt es zwei Kategorien: die funktionalen sowie nicht nicht-funktionalen Anforderungen. Erstere legen fest was die Architektur fachlich leisten können muss. Letztere spezifizieren die Eigenschaften unter denen die Architektur die funktionalen Anforderungen zu erfüllen hat.

Um Missverständnisse zu vermeiden werden die jeweiligen Anforderungen noch einmal kurz umrissen.

3.3.1. Funktionale Anforderungen

Die im folgenden aufgelisteten Anforderungen entsprechen den fachlichen Kernfunktionalitäten, welche die Architektur bzw. die darauf basierende Anwendung bieten soll.

3.3.1.1. Tracker-Steuerung

(a) Die Architektur muss einen Mechanismus bieten welcher die Ansteuerung der Grundfunktionalitäten des Eye-Trackers (oder anderer Datenquellen) ermöglicht. Zu den Grundfunktionalitäten einer Datenquelle zählt u.a. die Erkennung von verfügbaren Geräten (z.B. könnten theoretisch mehrere Eye-Tracker des selben Typs verfügbar sein), die Kalibrierung eben dieser und das Starten und Beenden eine Aufnahme welche Daten erfasst und weiterleitet.

(b) Da die Datenquelle austauschbar sein soll muss ihre Funktionalität in ein ladbares Plug-In ausgelagert werden. Diese Anforderung wird in Abschnitt 3.3.1.4 näher erläutert.

3.3.1.2. Datenkonvertierung

(a) Eine weitere Kernfunktionalität ist die Konvertierung des Eye-Tracker eigenen Datenformates in ein universell nutzbares Format. Diese ist Kern der im Weiteren geforderten Austauschbarkeit von Datenquelle und Weiterverarbeitung.

Da eine Weiterverarbeitung möglichst nur einmal entwickelt werden soll, ist es unablässig sie unabhängig von ihrer jeweiligen Eingabe zu machen. Daher wird das Format des Trackers in ein offenes Format konvertiert. Nach diesem Schritt muss eine beliebige Erweiterung dann nur mit dem universellen Format umgehen können und gewinnt dadurch Unabhängigkeit.

(b) Des Weiteren soll ein dediziertes Format erweiterbar sein. Dieses führt dazu, dass es offen gegenüber neuen Datenquellen oder Informationsarten zu ist. Hierbei ist allerdings zu beachten, dass eine Weiterverarbeitungsmethode auf dem universellen Format aufsetzt. Daher ist es nötig, dass das Format einigermaßen stabil oder zumindest abwärtskompatibel ist. Ist diese Grundannahme nicht gegeben (ändert sich das universelle Format also fortlaufend) könnte des zu Inkompatibilitäten mit Weiterverarbeitungsmethoden führen.

(c) Ebenso wie die Datenquelle muss diese Komponente austauschbar sein, da die Konvertierung ggf. direkt an die Datenquelle gebunden ist. Diese Anforderung wird in Abschnitt 3.3.1.4 detaillierter erklärt. Die beiden genannten Komponenten wurden bewusst voneinander

ander getrennt, da sie verschiedene Aufgabenbereiche haben. Zudem ist es sinnvoller eine ggf. langwierige Konvertierung von der Erfassung (welche effizient sein muss) zu trennen.

3.3.1.3. Weiterverarbeitung von Daten

(a) Die Weiterverarbeitung der gesammelten Daten ist die letzte Kernkomponente der Architektur. Die von der Datenquelle erhaltenen Informationen werden in der Regel nach ihrer Konvertierung aufgearbeitet.

Im Kontext dieser Arbeit geschieht die Aufarbeitung in Form einer Visualisierung durch eine Heat-Map (siehe Beispielgrafik: 5.1). Genauso denkbar ist aber auch die Speicherung in einem Plain-Text Format oder die Verwendung der Daten zur Steuerung eines Computers.

(b) Wie auch die Datenerfassung und Konvertierung ist die Weiterverarbeitung variabel und daher als austauschbarer Baustein zu entwickeln. Diese funktionale Anforderung ist in Abschnitt 3.3.1.4 genauer beschrieben.

3.3.1.4. Plug-In Fähigkeit

Das gesetzte Ziel der Austauschbarkeit der Tracker-Steuerung, der Konvertierung sowie der Weiterverarbeitung erfordert einen Mechanismus welcher die Einzelbestandteile möglichst unabhängig voneinander macht. Dieses bedeutet, dass es ohne tiefere Eingriffe ins Programm (z.B. Änderungen am Quellcode) möglich sein muss die Teilkomponenten zu ersetzen.

Der erste Schritt in Richtung der obigen Dynamik wurde mit der Anforderung der Konvertierung von speziellen Daten in allgemeine (also Eye-Tracker eigenes Format in universelles Format) erfüllt. Das Zwischenformat zwischen Erfassung und Weiterverarbeitung entkoppelt diese beiden Bestandteile.
Der nächste Schritt ist daher die geforderte Austauschbarkeit zu erreichen. Daher wird im Rahmen dieser Arbeit ein Mechanismus benutzt welcher es erlaubt die Steuerung, Konvertierung und Weiterverarbeitung in Form von Plug-Ins nachzuladen.

3.3.2. Nicht-Funktionale Anforderungen

Die im folgenden aufgelisteten Eigenschaften stellen Anforderungen an das Verhalten der Architektur bzw. der darauf basierende Anwendung dar. Diese Eigenschaften sind grundsätzlich schwer zu quantifizieren und sind daher nur indirekt umsetzbar.
Zum besseren Verständnis sind die Anforderungen kurz umrissen um Missverständnisse zu vermeiden.

3.3.2.1. Effizienz

(a) Eine der wichtigsten nicht-funktionalen Anforderungen im Rahmen dieser Arbeit ist die Effizienz. Im Vordergrund steht hierbei zunächst die Effizienz in Bezug auf die Datenquelle, da diese zwar vom Gesamtsystem angesteuert aber nicht während der Arbeit beeinflusst werden kann. So kann dieser u.a. nicht das Aufbewahren bzw. langsamere Senden von Daten aufgezwungen werden. Die weiteren Einzelbestandteile des Gesamtsystems sind genau wie die Datenquelle ebenfalls effizient zu gestalten, diese lassen sich aber in ihrer Arbeitsweise beeinflussen, da diese selbst entwickelt wird.

Die jeweilige Inputquelle kann Datensätze mit einer sehr hohen Frequenz liefern (der für dieses Projekt gewählte „ Tobii X120“ mit maximal 120 Datensätzen pro Sekunde [Tobii-c, S. 2]). Daher sind innerhalb von kurzer Zeit (im konkreten Fall: $\frac{\text{Zeit}}{\text{Datensätze pro Sekunde}} = \frac{1\text{s}}{120} = 0,008\overline{3}\text{s} = 8,\overline{3}\text{ms}$ pro Datensatz) viele Daten zu verarbeiten.

Aus diesem Grund muss das System so effizient arbeiten, dass möglichst jeder Datensatz innerhalb des Zeitfensters von 8,3ms erfasst und weitergeleitet wird. Geschieht dieses nicht kommt es zu Latenzen in der Verarbeitung oder sogar zum Verlust von Datensätzen. Dieses Verhalten ist vor allem für komplexere bzw. ggf. interaktive Weiterverarbeitungsmethoden unvorteilhaft.

(b) Ähnliche Bedingungen gelten auch für die weitere Verarbeitungsschritte, allerdings ist die Datenquelle die primäre Problemstelle, da sie die im Weiteren benutzten Daten liefert und nicht beeinflusst werden kann.

3.3.2.2. Zuverlässigkeit

Die Anforderung nach Zuverlässigkeit dient als Versicherung falls die oben beschriebene Effizienz nicht ausreichend erfüllt wird oder falls es andere Gründe gibt welche eine hohe Zuverlässigkeit erfordern.

Im konkreten Fall kann ein Problem entstehen wenn eine Teilkomponente des Systems (Datenerfassung, Datenkonvertierung, Datenverarbeitung) nicht zeiteffizient arbeitet. Wird ein Datensatz nicht mehr schnell genug (bei der Erfassung: innerhalb von 8,3ms (vgl. a)) abgearbeitet darf er daraufhin möglichst nicht verloren gehen. Das System muss also selbst bei sinkender Effizienz noch zuverlässig arbeiten. Ist diese Anforderung nicht erfüllt kann das Ergebnis verfälscht sein.

Aus diesem Grund ist eine zentrale Forderung, dass das System die von der Inputquelle während einer Aufzeichnung gelieferten Datensätze mit einer Verlustrate von 0% an die Konvertierung bzw. die Weiterverarbeitung weiterleitet.

3.3.2.3. Geringe Kopplung

Die Architektur soll die Hauptbestandteilen des Systems (Datenerfassung, Datenkonvertierung, Datenverarbeitung) dazu befähigen soweit wie möglich unabhängig voneinander arbeiten. Diese Trennung erleichtert es u.a. Einzelbestandteile auszutauschen und das System verteilt aufzubauen. Die Verteilung kann wiederum zur Steigerung der Effizienz und Zuverlässigkeit genutzt werden.

Die Unabhängigkeit der Systemteile soll durch eine geringe Ebene der Kopplung erreicht werden. Daher wird die Anforderung gestellt, dass die Einzelkomponenten möglichst nur einer Datenkopplung (d.h. die Ausgabe von Modul 'A' dient als Eingabe von Modul 'B' [IEEE Std 610.12-1990, S. 24]) unterliegen.

4. Architektur

Das folgende Kapitel beschreibt die Architektur des zu entwickelnden Systems.

Hierzu wird zunächst der Kontext der Applikation samt der Verteilung der Einzelbestandteile vorgestellt. Zum besseren Verständnis wird dabei auch auf den groben Ablauf innerhalb des Gesamtsystems eingegangen. Basierend auf diesen Informationen werden kurz die verwendeten Architekturkonzepte vorgestellt. Daraufhin werden die Komponenten der Anwendung zunächst grob vorgestellt und im Weiteren detaillierter erläutert.

4.1. Systemübersicht

Die Abbildung 4.1 zeigt die Kontextsicht des Gesamtsystems unter Berücksichtigung eines Message-Brokers.

Der Broker dient der Entkopplung der Einzelbestandteile, also der Datenerfassung mittels Eye-Tracker, der Datenkonvertierung und der Weiterverarbeitung der Daten. Darüber hinaus stellt die Abbildung die Kommunikationswege im System dar (die Pfeile stehen für die Richtung des Informationsflusses). Die einzelnen Module kommunizieren via TCP/IP-Protokoll über ein Netzwerk. Hierbei wird konsequent das Publish-Subscribe Pattern genutzt [vgl. Publish-Subscripe Pattern GoF94, S. 250]. Durch die Entkopplung können die einzelnen Arbeitsschritte rein theoretisch sogar verteilt innerhalb des Netzwerkes ausgeführt werden, da die jeweiligen Arbeitsergebnisse über das Netzwerk an den Broker geleitet werden und dort jederzeit abrufbar sind.

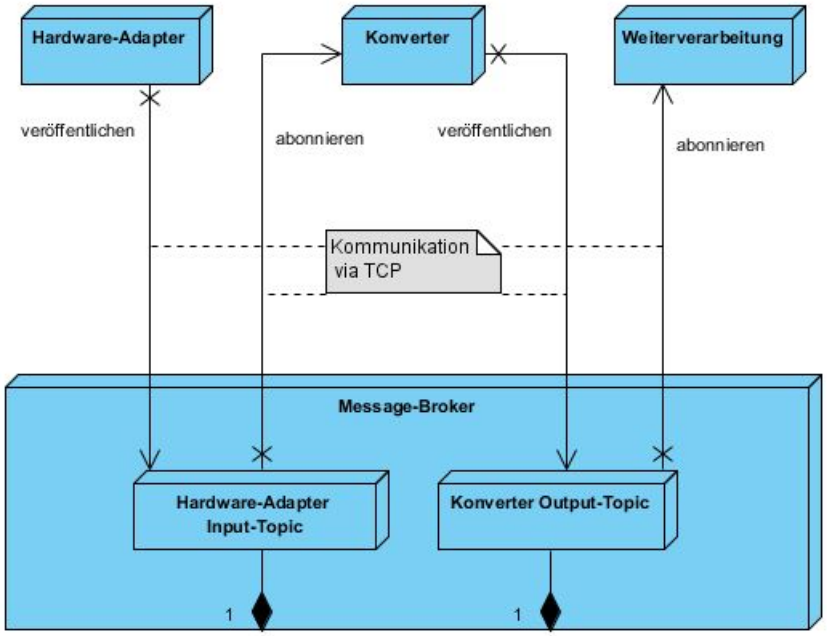

Abbildung 4.1.: Kontextsicht des Systems unter Berücksichtigung auf die Message orientierte Middleware

Ein grober beispielhafter Ablauf ist in Grafik 4.2 zu sehen.

1. Als erstes werden die Einzelkomponenten des Systems von einem Hauptprogramm (oder dem User) gestartet.

2. Ist dieses geschehen ist der Ablauf wie folgt: Zunächst stellt ein Hardware-Adapter (also z.B. ein Eye-Tracker) die von ihm empfangenen Daten in eine Topic innerhalb des Message-Brokers (publish).

3. Diese Topic ist vom Konverter abonniert (subscripe) und daher erhält dieser bei Nachrichteneingängen eine Mitteilung, so dass die neuen Daten verarbeitet werden können.

4. Die konvertierten Daten werden daraufhin vom Konverter in eine weitere Topic des Brokers gestellt und sind somit verfügbar.

5. Diese Topic wiederum wird durch die Weiterverarbeitungskomponente überwacht, so dass auch diese eine Information über neue Datensätze erhält und diese verarbeiten kann.

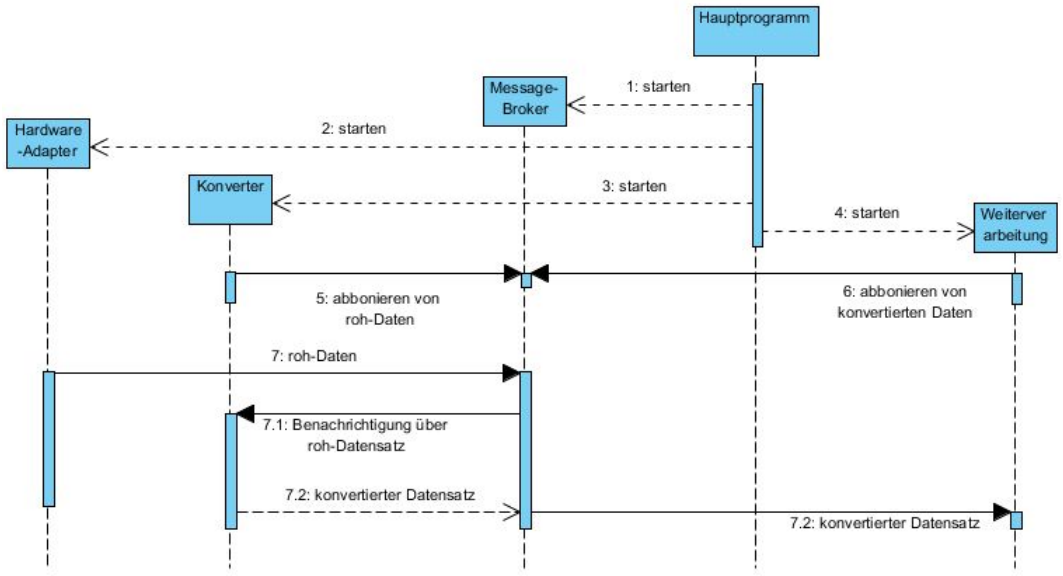

Abbildung 4.2.: Laufzeitsicht des Systems unter Berücksichtigung auf die Message orientierte Middleware

Der gewählte Aufbau unterstützt die Erfüllung der nicht-funktionalen Anforderungen (siehe: 3.3.2). Wie bereits erwähnt dient der dedizierte Message-Broker der Erhöhung der Effizienz und Zuverlässigkeit sowie der Entkopplung des Gesamtsystems.

- Die ausgetauschten Informationen können beim Eintreffen am Broker bis zu ihrer Weiterverwendung gesichert werden. Diese Sicherung stellt einen Puffermechanismus für eine ggf. langsame oder spätere Verarbeitung von Daten dar.

- Die Forderung nach Effizienz wird indirekt durch diesen Aufbau unterstützt. Da alle Komponenten aufgrund der Kommunikation über ein Netzwerk dediziert arbeiten können ist es möglich die Einzelverarbeitungen zu verteilen um vorhandene Rechenkapazität besser nutzen zu können.

- Darüber hinaus ist es auch denkbar mit den obigen Techniken mehrere Verarbeitungsmethoden parallel zu starten, sofern diese sich alle beim Broker registrieren. Der Broker kann daraufhin einen Datensatz zur Verarbeitung an nahezu beliebig viele Rechner verteilen und so beispielsweise mehrere Weiterverarbeitungsmethoden parallel arbeiten lassen.

- Die Verwendung einer MoM bietet darüber hinaus den Vorteil die Bestandteile des Systems nur bezüglich der Middleware zu koppeln, nicht aber untereinander. Daher muss jedes Subsystem in der Verarbeitungskette nur die Schnittstelle der Middleware kennen.

4.2. Architekturstil

Nun da der grobe Aufbau des Systems dargelegt wurde lässt sich erkennen, dass das System eine Komposition aus mehreren allgemeinen Architekturstilen ist. Wie und wo genau die einzelnen Grundideens angewendet werden lässt sich sehr gut an der Übersicht der Komponenten sehen.

Pipe & Filter

Zunächst ähnelt die Gesamtarchitektur dem „Pipe & Filter“-Stil [vgl. Qian08, S. 65].

Die grundlegenden Datentranformationen (Hardware-Adapter, Konvertierung, Verarbeitung) funktionieren wie der Filter innerhalb der Referenzarchitektur, während dessen übernimmt die Middleware die Aufgabe einer Pipe welche die Daten transportiert. Hierbei ist allerdings anzumerken, dass die die Transformationskomponenten aktiv an die Middleware senden während sie passiv empfangen. Daher kann die Trennung nicht haarscharf vollzogen werden. Allerdings bringt der gewählte Aufbau den selben Vorteil des Performanzgewinns. Die jeweils nächste Komponente kann theoretisch schon Datensätze verarbeiten während weitere Daten noch von ihrem Vorgänger abgearbeitet werden. Ebenso können Komponenten wie die Weiterverarbeitung immer wieder verwendet werden, selbes gilt auch für die Datenerfassung falls man die Konvertierung oder die Verarbeitung austauscht. Allerdings bestehen immer Abhängigkeiten zwischen Erfassung, Konvertierung und Verarbeitung.

Neben den Vorteilen des „Pipe & Filter“-Systems bringt die gewählte Architektur allerdings auch dessen Nachteile mit. Zum einen muss das Datenformat wie erwähnt möglichst stabil sein. Darüber hinaus entsteht auch ein zeitlicher Verlust da sowohl der Konverter als auch die Weiterverarbeitung jeweils die Datensätze aus dem Broker holen, einlesen und verarbeiten müssen.

Buffered-Message-Based

Eine weitere Grundidee welche zum Einsatz kommt ist der Ansatz Nachrichten im Sinne des „Buffered-Message-Based“-Stils [vgl. Qian08, S. 96] zwischenzulagern bzw. über einen Message-Broker zu kommunizieren. Die Vorteile die sich daraus ergeben passen sehr gut zu den Zielen dieses Projektes.

Zum einen werden die Bestandteile des Systems entkoppelt und gleichzeitig steigt die Zuverlässigkeit bzw. sinkt der Verlust von Datensätzen. Dieses ist ohne Mehraufwand möglich, da die gesamte Kommunikation darauf ausgelegt ist über die Middleware zu funktionieren.

Zum Anderen kann die Effizienz des Gesamtsystems positiv beeinflusst werden, da die MoM das Verteilen des Systems vereinfacht.

4.3. Komponentenübersicht

Die folgenden Abschnitte beschreiben das System mit Sicht auf seine Komponenten bzw. deren Zusammenarbeit.

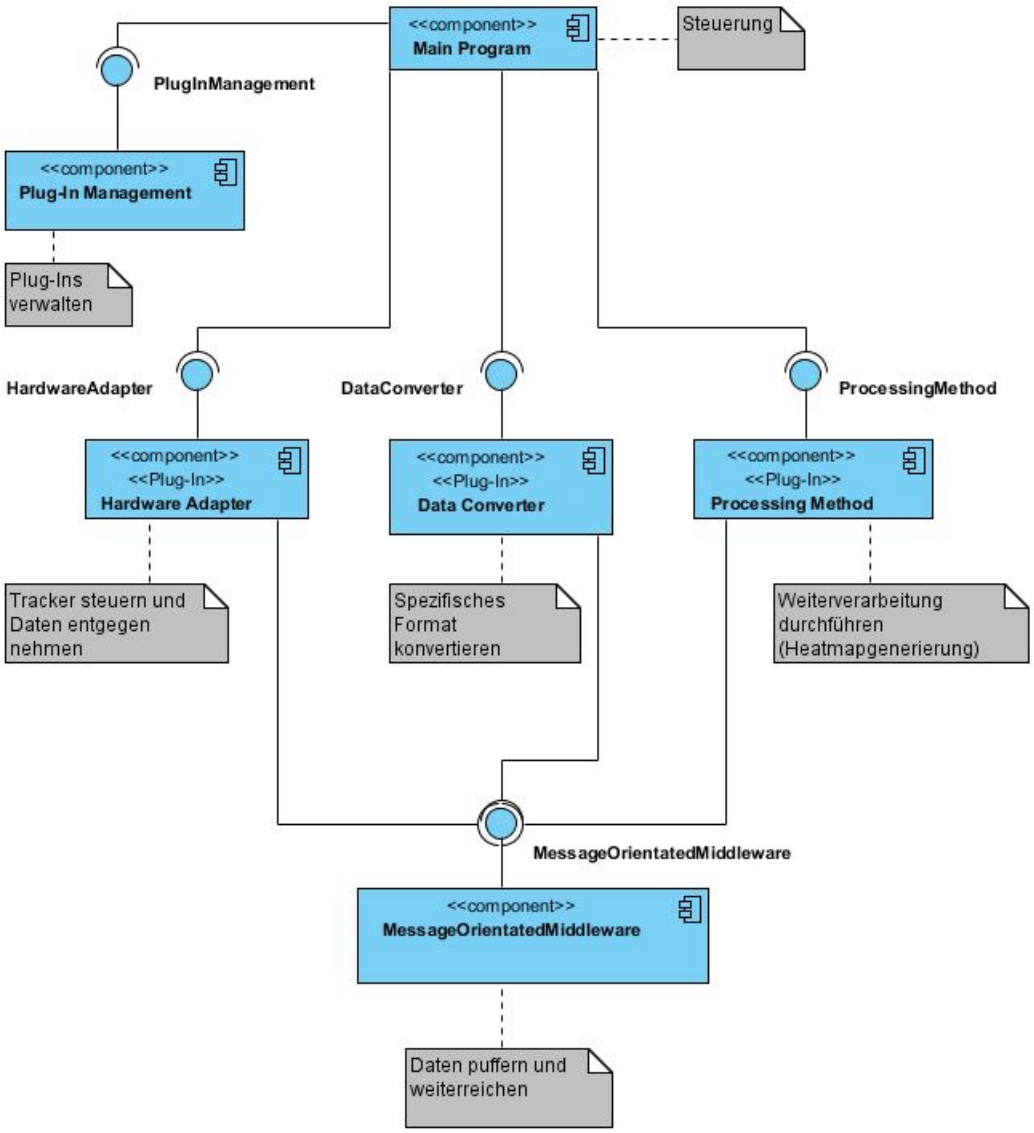

Abbildung 4.3.: Hauptkomponenten des Systems und deren Zusammenhang

Abbildung 4.3 zeigt die Komponenten des Systems.

Neben den bereits bekannten Hauptbestandteilen Datenerfassung, Datenkonvertierung und Weiterverarbeitung sind nun auch die weiteren Bestandteile zu sehen – das Plug-In Management und die bereits erwähnte Message orientierte Middleware. Darüber hinaus ist eine weitere

Komponente mit dem Namen „MainControl“ zu sehen, diese dient allein der Vollständigkeit und übernimmt die Aufgabe der Steuerung der Applikation und steht damit stellvertretend für einen User. Die Abbildung stellt darüber hinaus auch dar welche Aufgaben den einzelnen Komponenten im Groben zufallen.

Eine detaillierte Beschreibung der Komponenten samt ihrer vereinfachten Schnittstellen folgt. Hierbei ist zu beachten, dass die Schnittstellen nur die wichtigen Bestandteile zeigen. Daher wurde auf get- und set-Methoden, sowie nicht direkt verwendete Konstruktoren usw. verzichtet. Hingegen wurden zwingend notwendige Instanzvariablen mit aufgeführt.

4.3.1. Plug-In Management

Die hier beschriebene Komponente stellt die grundlegende Funktionalität für die Verwaltung von Plug-Ins dar. Daher dient sie der Erfüllung der allgemeinen funktionalen Anforderung 3.3.1.4, sowie der Umsetzung der geforderten Plug-In Fähigkeit der Einzelkomponenten „Tracker-Steuerung“ (Abschnitt: 3.3.1.1 (b)), „Datenkonvertierung“ (Abschnitt: 3.3.1.2 (c)) und „Weiterverarbeitung“ (Abschnitt: 3.3.1.3 (b)).

Ziel bei der Entwicklung des Plug-In Mechanismus war es eine möglichst simple aber zugleich ausreichend mächtige Möglichkeit zu finden Code in eine bereits bestehende Anwendung zu integrieren ohne diese essentiell zu verändern. Da das Laden von Erweiterungen zwar ein grundlegender Baustein der Anwendung ist aber letztendlich nur ein kleiner Bestandteil der Gesamtlösung ist wurde im Rahmen dieser Arbeit darauf verzichtet eine plattform-unabhängige Lösung zu finden, daher liegt der Fokus auf einer Anwendung für ein Windowsbetriebssystem.

Eine Möglichkeit zur Lösung der gestellten Aufgabe ist das von Microsoft entwickelte Component Object Model (COM ; `https://www.microsoft.com/com/default.mspx`), welches u.a. das Nachladen von Code ermöglicht. Dieses hat aber einige Nachteile. Grundsätzlich steigert COM die Komplexität der zu entwerfenden Lösung enorm und bietet mehr Funktionalität als nötig ist. Darüber hinaus ist es eine weitere Abhängigkeit welche im Folgenden beachtet werden muss. Der Autor entschied sich daher für eine einfachere und weniger komplexe Problemlösungsstrategie. Basierend auf der einer Idee von George Mihaescu [siehe: Mihaescu] wurde daher der im folgenden erläuterte Ansatz gewählt.

Die im Rahmen dieser Arbeit entwickelte Plug-In Lösung ist windows-spezifisch und basiert auf der Idee Funktionen der Windows-API zu nutzen um Dynamic Linked Librarys (DLLs) als Anwendungserweiterungen zu laden. Die entwickelten Plug-Ins bestehen hierbei aus gewöhnlichem Code welcher als DLL kompiliert wird. Die einzige Besonderheit besteht darin, dass jede Anwendungserweiterung eine Funktion zum Aufbau und zur Löschung eines Plug-In Objektes besitzt. Da die Funktionsweise zum Laden von Erweiterungen universell sein soll führt ein Aufruf der Ladefunktion dazu, dass eine Datenstruktur zurückgeben wird welche die entsprechende Aufbau- und Zerstörungsfunktion beinhaltet. Die dazu konkret implementierten Funktionen einer Erweiterung werden über die Windows-API an die Funktionen der Datenstruktur gebunden. Die Aufbaufunktion ist hierbei so entworfen, dass sie ein allgemeines Plug-In Objekt zurückgibt, welches zu einem spezifischen Objekt gecastet werden muss. Dieser Aufbau hat den Vorteil, dass der Lademechanismus soweit wie möglich unabhängig vom konkreten Aufbau eines Plug-Ins ist. Ein Plug-In muss daher nur zwei grundlegende Funktionen bereitstellen (diese können auch über eine abstrakte Klasse vorgeschrieben oder vererbt werden).

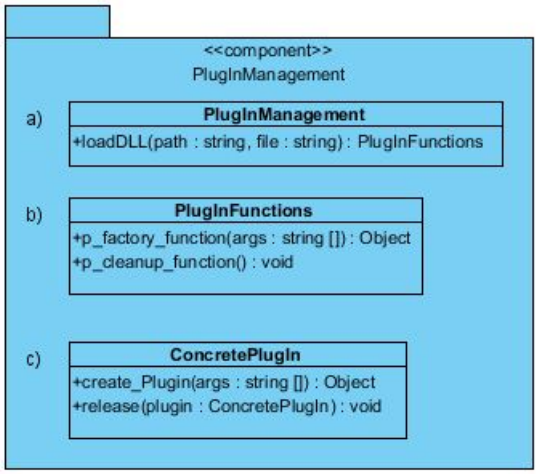

Abbildung 4.4.: Plug-In Mechanismus:
a) Schnittstelle
b) Entsprechender Datenstruktur
c) Plug-In spezifische Funktionen

Die Abbildung 4.4 zeigt die Plug-In Management Komponente. Diese besteht aus drei Teilen:

Abbildungsbestandteil a) zeigt die grundlegende Funktionalität mit der ein bestimmtes Plug-In (in Form einer DLL) geladen wird. Dabei wird über die Parameter die zu ladende DLL spezifiziert. Der Rückgabewert ist hierbei eine Instanz der Klasse PlugInFunktions.

Die als Rückgabewert angegebene Klasse ist in Teil b) der Abbildung zu sehen. Diese Klasse stellt einen reinen Container dar, dessen Aufgabe es ist die zwei Basisfunktionen eines Plug-Ins abrufbar zu machen. Die beiden gespeicherten Funktionen dienen der Erstellung eines Plug-Ins sowie seiner Zerstörung. Während die Zerstörung eines Plug-Ins ohne Argumente auskommt benötigt die Factory zur Erstellung eine Reihe von Plug-In spezifischen String-Argumenten (also Parameter welche während der Erzeugung benutzt werden wie z.B. IP-Adressen). Der Rückgabewert dieser Factory-Funktion ist ein allgemeines Objekt. Dieses muss zur Benutzung in ein konkretes Plug-In Objekt gecasted werden.

Teil c) der Abbildung dient der Veranschaulichung der Funktionen die ein zu entwerfendes Plug-In mindestens anbieten muss.

- Zwingend erforderlich ist eine konkrete Erzeugungs-Funktion (dieser werden beim Aufruf die Parameter der Factory-Funktion aus Teil b) übergeben). Diese Funktion gibt ein Objekt zurück welches auf ein konkretes Plug-In Objekt gecastet werden muss.

- Des Weiteren ist eine Release-Funktion vorgesehen welche ein Plug-In Objekt wieder zerstört (daher wird ihr dieses als Argument übergeben).

4.3.2. Message orientierte Middleware

Die im folgenden beschriebene Komponente ist die Middleware welche die Einzelteile des Systems verbindet. Darüber hinaus ist die Message orientierte Middleware nicht nur Verbindungspunkt sondern auch eine zentrale Maßnahme zur Erfüllung der nicht-funktionalen Anforderungen nach Effizienz (3.3.2.1), Zuverlässigkeit (3.3.2.2) und geringer Kopplung (3.3.2.3).

Die Middleware steht im Zentrum der Applikation da über sie die gesamte Kommunikation der Hauptkomponenten abgewickelt wird. Informationen welche von Komponente 'A' zu Komponente 'B' geschickt werden nehmen den Weg über die Nachrichtenverwaltung der Middleware. Die Verwaltung der Nachrichten findet im Nachrichten-Broker statt, dieser ist ein externer Mechanismus welcher unabhängig vom System arbeitet. Der Broker wird über

entsprechende Client-Schnittstellen über TCP/IP angesprochen und kann daher auch auf einem anderen Rechner laufen (siehe auch: 4.1).

Die gewählte Herangehensweise hat diverse Vorteile:

- Zunächst sorgt die Middleware dafür, dass die Einzelkomponenten nicht mehr direkt aufeinander aufbauen, stattdessen arbeiten die Bestandteile nur noch auf der Abstraktionsschicht der Middleware. Dieses entkoppelt die Komponenten und macht sie unabhängiger, da sie jeweils nur die Schnittstelle der Middleware und das Format der gelieferten Daten kennen müssen.

- Ein weiterer Vorteil der durch die Nutzung der Middleware gewonnen wird ist Zuverlässigkeit. Die Daten werden in jedem Fall über den Message-Broker geleitet. Daher kann dieser die Nachrichten ohne Mehraufwand für langsame Teilnehmer vorhalten oder auch die selben Daten an mehrere Teilnehmer senden.

  Um diesen Vorteil komplett zu nutzen agieren die Einzelkomponenten wie bereits angesprochen nach dem publish-subscribe-Muster. Die Produzenten von Daten schicken ihre Ergebnisse an den Broker (publish), dieser übernimmt die Verwaltung und leitet sie an alle angemeldeten (subscribe) Verbraucher weiter. Wie angedeutet ist die Beziehung zwischen Hersteller und Verbraucher nicht zwingend eine 1:1 Beziehung im Sinne einer Nachrichten-Queue. Es ist auch möglich eine 1:n Beziehung zu etablieren, dieses wird durch eine „Topic“ ermöglicht. Diese erlaubt es beliebig vielen Verbrauchern eine Nachricht zu abonnieren [vgl: Qian08, S. 97]. Da der Broker die alleinige Verwaltung übernimmt kann er Nachrichten daher auch zwischenlagern. Falls ein Verbraucher den Empfang einer Nachricht nicht bestätigt kann ein erneuter Übermittlungsversuch gestartet werden.

- Auch die Effizienz des Gesamtsystems kann durch die Nutzung der Middleware gesteigert werden. Zunächst ist der „Umweg“ den die Nachrichten über den Broker nehmen ein Zeitfaktor welcher sich negativ auswirkt. Allerdings verlagert der Broker wie bereits erwähnt die Kommunikation ins Netzwerk, da jede Komponente über den den Broker angebunden ist. Aus diesem Grund ist es ohne Mehraufwand möglich das Gesamtsystem auf verschiedene Rechner zu verteilen. So könnte die Datenerfassung auf Rechner ’A’ stattfinden, die Konvertierung auf ’B’ und die Weiterverarbeitung auf Rechner ’C’. Daher

kann eine Teilaufgabe mit der kompletten Leistung eines Rechners ausgeführt werden, da dieser im Idealfall keine weiteren Aufgaben zu erfüllen hat.

Ob diese Verteilung in der Praxis nötig wird ist natürlich von der Komplexität der Arbeitsschritte abhängig, dennoch kann es theoretisch zu einer Steigerung der Effizienz führen.

- Des Weiteren kann die Zuverlässig und Effizienz des Nachrichtentransports ggf. durch diverse weitere Einstellungen („durable“ Queues/Topics, „persistent Delivery“, Empfangsbestätigungen etc. [vgl Kapitel 8.3: Qian08]) beeinflusst werden.

Trotz aller Vorteile hat die gewählte Architektur des Systems auch Nachteile:

- Zum einen ist der dedizierte Broker zunächst ein „Single point of failure“, da sein Versagen dazu führt dass das Gesamtsystem nicht mehr funktioniert. Dieses Problem kann durch verschiedene Strategien wie z.B. durch redundante Broker angegangen werden. Dabei ist zu beachten, dass dieses allerdings sehr aufwändig werden kann, daher muss eine Entscheidung über die Notwendigkeit vom konkreten Einzelfall abhängen.

- Ein weiterer Nachteil der sich durch die Vernetzung des Systems ergibt sind Latenzzeiten innerhalb des Netzwerkes, welche ein System beeinflussen können. Durch den Umweg über das Netzwerk werden Nachrichten ggf. verzögert zugestellt. Sind diese zeitkritisch muss die Applikation mit der Verzögerung umgehen können, da sonst fehlerhafte Ergebnisse entstehen könnten.

Abbildung 4.5 zeigt nun die Schnittstelle der Middleware samt des angedeuteten externen Message-Brokers welcher genutzt wird. Die dargestellte Schnittstelle dient als Abstraktionsschicht zwischen einer konkret verwendeten Message orientierten Middleware und den wirklich im Rahmen der Anwendung benötigten Mechanismen – Also einem „Produzenten“ und „Verbraucher“ von Nachrichten.

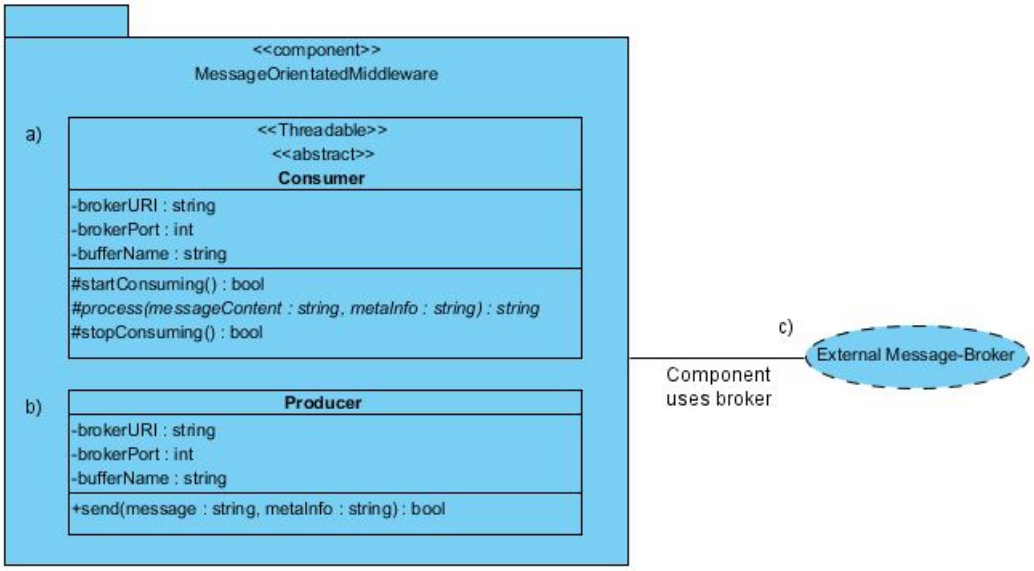

Abbildung 4.5.: Message orientierte Middleware:
a) Schnittstelle eines Consumers
b) Schnittstelle eines Producers
c) Externer Broker-Mechanismus

Teil a) der Abbildung zeigt das Interface eines Verbrauchers welcher Nachrichten empfängt und verarbeitet. Dieser ist als „Threadable“-Objekt entworfen, d.h. dieser wird als Thread gestartet, damit er nicht den weiteren Programmablauf blockiert.
Der Consumer arbeitet direkt mit dem Message-Broker zusammen, daher besitzt er Informationen über die URI sowie den Port unter dem dieser erreichbar ist. Des Weiteren wird ihm mitgegeben welchen Puffer er innerhalb des Brokers benutzen soll, also welche Topic-Nachrichten er abonniert.
Die Hauptfunktionen des Consumers sind die Aufnahme seiner Tätigkeit (startConsuming) sowie das Beenden eben dieser. Ab dem Startzeitpunkt ist er bereit Nachrichten zu empfangen bzw. zu bearbeiten – die Initialisierung hat also stattgefunden. Sowohl der Aufruf von start- und „stopConsuming“ liefern als Ergebnis einen Boolwert welcher den Erfolg bzw. Misserfolg der Aktion darstellt. Darüber hinaus besitzt der Verbraucher die Methode „process“, diese wird aufgerufen sobald eine neue Nachricht in der abonnierten Topic vorhanden ist. Diese Methode erhält vom Broker sowohl den eigentlichen Nachrichten Inhalt, sowie auch Metainformationen über die Nachricht. Mit diesen Informationen arbeitet die Methode die Nachricht ab. Die process-Funktion ist abstrakt, d.h. sie muss durch eine konkrete Funktion einer von

„Consumer“ abgeleiteten Klasse ersetzt werden. Ein Aufruf von „process“ liefert einen Text zurück, dieser enthält die einfache Repräsentation des Ergebnisses des Aufrufs.

Der Nachrichtenproduzent (Producer) ist in Teil b) der Abbildung zu sehen. Ebenso wie der Consumer enthält er Informationen über den zu adressierenden Broker, sowie den Puffer in welchem er Nachrichten ablegen soll.
Die Kernfunktion des Producers ist das Senden von Nachrichten an den in c) gezeigten Message-Broker. Die Funktionalität wird in der send-Funktion umgesetzt, diese erhält eine Textnachricht (String), sowie zur Nachricht gehörende Metainformationen. Aus diesen beiden Parametern wird eine Nachricht generiert welche an den Broker weitergeleitet wird. Über den Erfolg des Sendens informiert der Rückgabewert des Aufrufs.

4.3.3. Hardware Adapter

Die folgenden Abschnitte beschreiben die HardwareAdapter Komponente. Diese dient der Datenerfassung und bildet die Schnittstelle zu Datenlieferanten wie beispielsweise einem Eye-Tracker. Damit widmet sie sich der Erfüllung der funktionalen Anforderung 3.3.1.1 (Abschnitt (a)).

Die HardwareAdapter-Komponente besteht streng genommen aus zwei separaten Bestandteilen:

- Einer dieser Bestandteile ist das eigentliche Plug-In welches der späteren Ansteuerung eines Gerätes wie einem Eye-Tracker dient. Dieses wird als Dynamic Linked Library (DLL) kompiliert, sodass eine Anwendung durch diese austauschbare DLL erweitert werden kann.

- Der andere Bestandteil ist die Basis für das zu entwickelnde Plug-In. Diese Teilkomponente stellt Klassen zur Verfügung, welche es erlauben verschiedene Arten von HardwareAdapter Plug-Ins zu nutzen. Darüber hinaus enthält dieser Teil der Komponente Klassen zur Verwaltung Hardwareinformationen des genutzten Gerätes. Diese Informationen dienen sowohl der Identifizierung sowie der Kalibrierung einer genutzten Hardware.

Während der Aufnahme von Daten sendet der Hardware-Adapter die Resultate fortlaufend an den Broker. Diese Resultate sind in drei Gruppen aufgeteilt:

- Eine Nachricht welche den Beginn einer Aufzeichnung markiert.
- Eine weitere welche einen aufgezeichneten Datensatz enthält
- Und eine letzte welche das Ende einer Aufnahme anzeigt.

Diese Unterscheidung ist nötig um den verteilt arbeitenden Bestandteilen den jeweiligen Status der Aufnahme mitzuteilen. Diese Meta-Information erleichtern u.a. das Erstellen von validen Daten in der Konverter-Komponente.

Abbildung 4.6 stellt die fünf Bestandteile der HardwareAdapter-Komponente dar. Hierbei zeigt Teil e) der Grafik ein abstraktes beispielhaftes Plug-In zur Datenerfassung für einen netzwerkfähiges Eye-Tracker. Die restlichen Bestandteile dienen als Basis für dieses Plug-In.

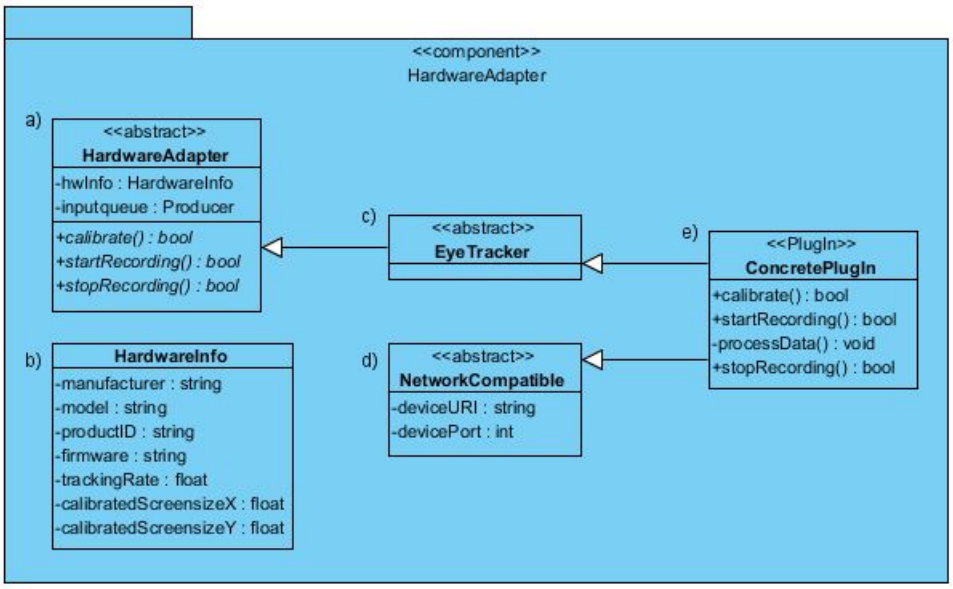

Abbildung 4.6.: HardwareAdapter:
a) Schnittstelle eines HardwareAdapters
b) Schnittstelle der HardwareInfo
c) Marker-Interface eines Eye-Trackers
d) Schnittstelle eines Netzwerkgerätes
e) Schnittstelle des eigentlichen Steuerungs-Plug-Ins

Grafikbestandteil a) zeigt den grundsätzlichen Aufbau eines Hardware-Adapters:

- Jeder Adapter hat grundsätzlich die ihm entsprechenden Hardware-Informationen verfügbar, diese können beispielsweise beim initialisieren mitgegeben oder abgefragt werden.

- Des Weiteren benötigt ein Hardware-Adapter Zugang zur Message orientierten Middleware um die von ihm gelieferten Daten an den Broker weiterzuleiten. Aus diesem Grund hält der jeder Adapter eine Instanz eines Producers der Middleware. Dieser Producer muss beim Aufbau eines Adapters mit initialisiert werden.

- Ein gültige Hardwareschnittstelle brauch drei grundlegende Funktionen. Zunächst muss eine Aufnahme von Daten gestartet und beendet werden können, darüber hinaus ist vorher ggf. eine Kalibrierung der Quelle von Nöten. Diese Basisfunktionen deklariert je-

der Hardware-Adapter, wobei der Rückgabewert über den Erfolg des Aufrufs informiert. Da eine konkrete Umsetzung der Funktionen aber vom benutzten Gerät abhängt sind diese im Adapter nur deklariert aber nicht definiert – die Funktionen sind also wie die entsprechende Klasse abstrakt. Die konkreten Definitionen wie die jeweilige Methode umgesetzt wird erfolgt daher in einer Unterklasse welche das Plug-In bildet.

Bestandteil b) der Grafik stellt den Inhalt einer HardwareInfo dar. Diese enthält neben den Informationen über Hardwarehersteller, Modell, Bezeichnung und benutzter Firmware auch Informationen über mögliche Kalibrierungsparameter. Diese können u.a. für die Verarbeitung von Daten wichtig sein. So kann z.B. die kalibrierte Displaygröße wichtig sein im Falle einer Visualisierung der aufgenommenen Daten um festzulegen wie viele Punkte maximal betrachtet worden sein können. Darüber hinaus können diese Meta-Informationen grundsätzlich dazu dienen das universelle Weiterverarbeitungsformat anzureichern, so dass die Eigenschaften der Datenquelle nicht verloren gehen.

Die unter c) zu sehende Klasse „EyeTracker“ ist ein (Marker-)Interface welches zunächst lediglich eine zusätzliche Information über den Typ einer Datenquelle bereitstellt. Es ist von der Basisklasse „HardwareAdapter“ abgeleitet und übernimmt damit alle vererbten Informationen – es ist ein Hardware Adapter welcher ein Eye-Tracker ist. Diese Klasse kann ggf. erweitert werden und Eye-Tracker spezifische Inhalte repräsentieren, dennoch bleibt sie abstrakt solange die geerbten Definitionen nicht implementiert werden.

Abbildungsbestandteil d) hat einen ähnlichen Zweck wie das unter c) beschriebene Interface – es vermittelt die Information, dass ein Gerät netzwerk-kompatibel ist. Ein solche Datenquelle hat daher mindestens eine URI (Uniform Resource Identificator) sowie einen Port unter dem sie erreichbar ist.

Ein beispielhaftes Plug-In ist unter e) zu erkennen. Dieses stellt ein Eye-Tracker mit der Fähigkeit im Netzwerk zu kommunizieren dar. Aus diesem Grund erbt es sowohl von der abstrakten Klasse EyeTracker (und ist damit letztendlich ein HardwareAdapter) als auch von „NetworkCompatible“.
Da es sich um eine konkrete Anwendungserweiterung handelt darf die Klasse keine abstrakten Methoden mehr besitzen. Daher werden alle weitervererbten Funktionen auf Basis einer konkreten Hardware implementiert. Des Weiteren ist exemplarisch eine private Funktion „processData“ aufgeführt, diese soll darstellen, dass das Plug-In natürlich spezifische Funktionen

haben kann welche der Erfüllung seiner Aufgabe dienen. So kann die processData-Funktion z.B. die vom Tracker empfangenen Datensätze verarbeiten und weiterleiten.

4.3.4. Konverter

4.3.4.1. Komponentenaufbau

Die folgenden Abschnitte beschreiben die Konverter-Komponente welche die Roh-Daten einer beliebigen Datenquelle in ein, aus Sicht der Weiterverarbeitung, universelles Format konvertiert (Anforderung 3.3.1.2 Abschnitt (a)). Diese Komponente entkoppelt die Endverarbeitung von Daten von deren Erfassung durch ein XML-basiertes Datenformat welches frei zugänglich und erweiterbar ist (Anforderungen 3.3.1.2 Abschnitt (b) sowie 3.3.2.3). Dieses Format wird in Abschnitt 4.3.4.2 detaillierter erläutert.

Da der Input der Komponente von der Datenquelle abhängt ist der Konverter ebenso wie der HardwareAdapter an die gewählte Hardware gebunden. Der Adapter und der Konverter wurden allerdings als getrennte Komponenten entworfen da es sich um inhaltlich verschiedene Aufgaben (Datenerfassung und Datenkonvertierung) handelt. Des Weiteren ermöglicht dieser Aufbau es, dass Daten von der selben Quelle unterschiedlich konvertiert werden können. Dieses ist von Vorteil sobald mehrere Format-Versionen verfügbar sind, in diesem Fall muss nur der Konverter (und ggf. die Weiterverarbeitung) ausgetauscht werden während der Hardwareadapter unangetastet bleiben kann.

Der Konverter wurde als spezieller Consumer der Middleware entworfen. Dieses bietet sich an, da sowohl ein normaler Consumer als auch ein Konverter zunächst Daten empfangen, der einzige Unterschied ist die im Konverter vorhandene Verarbeitung.
Ein Konverter erbt die Funktionalität eines Consumers und erweitert sie um die ihm eigenen Eigenschaften und die entsprechende Schnittstelle. Daher ist der Konverter grundlegend ein Consumer welcher einen Middleware-Producer zum Versenden von Ergebnissen sowie die entsprechende Logik zur Konvertierung hält.

Ein Konverter empfängt von der Middleware eine Nachricht welche er verarbeitet und daraufhin das Ergebnis als Nachricht an den Broker sendet. Diese Nachrichten sind keine direkt verwertbaren Objekte sondern reine Textnachrichten, da die Middleware unabhängig von der

benutzten Programmiersprache (und ihren Objekten) operieren soll. Diese Textnachrichten müssen dementsprechend vor ihrer Verarbeitung zunächst interpretiert werden. Das Resultat des Hardware-Adapters ist eine Nachricht in einem csv-Format (comma-separated values), daher ist es vergleichsweise simpel die Einzelbestandteile zu identifizieren.

Der Hardware-Adapter sendet wie angesprochen drei verschiedene Arten von Nachrichten welche über die jeweilige Meta-Information zu unterscheiden sind. Die eintreffenden Nachrichten enthalten die Daten welche konvertiert werden sollen, daher müssen diese zunächst aus dem csv-Strom extrahiert werden. Diese extrahierten Daten werden für die Konvertierung zwischengespeichert, hierbei werden einmalige Basisinformationen über die Hardware und regelmäßig eintreffende Datensätze unterschieden.

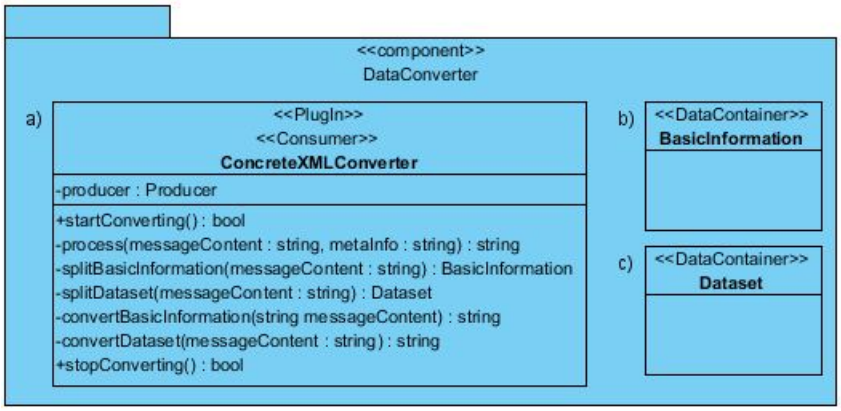

Abbildung 4.7.: DataConverter:
a) Schnittstelle eines DataConverter Plug Ins
b) Container für Basisinformationen zur Datenquelle und zum Format
c) Container für einen Datensatz der Datenquelle

Die Abbildung 4.7 zeigt die drei Bestandteile eines „DataConverter“.

Grafik a) stellt ein abstraktes DataConverter Plug-In dar. Dieses besitzt einen Producer welcher beim Aufbau des Plug-Ins initialisiert werden muss. Dieser Producer dient dem Versenden von konvertierten Daten an den Broker.

Der Konverter besitzt die Fähigkeit seine Tätigkeit mittels „startConverting“ und „stopConverting“ aufzunehmen bzw. zu beenden. Darüber hinaus hat der Konverter die eigentliche

Fachlogik: Die Funktion „process“ welche aufgerufen wird sobald ein neuer Datensatz (bestehend aus Nachricht und Metainformation/Typ) eintrifft. Entsprechend des Nachrichten Typs werden die Informationen aus dem csv-Format in ein XML-Format konvertiert (convert). Dieser Konvertiervorgang extrahiert zunächst die relevanten Daten aus dem csv-Format (split) und überträgt sie das in das Resultat. Hierbei ist es notwendig die Basisinformationen über die Hardware sowie die Kalibrierung (Bildschirmgröße, Formatversion, spezielle Zeichenkette für nicht vorhandene Informationen) und die eigentlichen Datensätzen zu unterscheiden.

Die Bestandteile b) und c) deuten die jeweiligen Container zur Zwischenspeicherung der aus dem csv-Format extrahierten Daten an. Hierbei werden wie erwähnt die Basisinformationen und die eigentlichen Datensätze unterschieden.

4.3.4.2. Datenformat

Die folgenden Abschnitte widmen sich dem bereits erwähnten Datenformat welches eine möglichst universelle Weiterverarbeitung von einmal konvertierten Daten erlauben soll (Anforderung 3.3.1.2 Abschnitt (b)). Die grundsätzliche Idee für dieses Format entstammt einem Paper von C. Hennessey und Andrew T. Duchowski [siehe: Hennessey/Duchowski10]. Der Verwendungszweck des vorgeschlagenen Formats wurde vom Autor geändert und das Format den spezifischen Bedürfnissen dieser Arbeit angepasst.

Das modifizierte Format ist XML-basiert und erbt damit alle Vorteile die XML mit sich bringt es ist u.a. also vergleichsweise leicht verständlich, es kann über XML-Sicherheitsmechanismen verschlüsselt und signiert werden und es gibt eine Reihe von Werkzeugen die das Arbeiten mit dem Format erleichtern. Darüber hinaus hat ein XML-Dokument den Vorteil das es vergleichsweise einfach erweitert/verändert werden kann. Dieses wird nötig sobald weitere Datenquellen hinzukommen (das aktuelle Format ist grundsätzlich Eye-Tracker orientiert) oder weitere Meta-Informationen eingepflegt werden müssen. Eine Grundannahme dieser Arbeit ist allerdings, dass das Datenformat grundsätzlich stabil (also nahezu unverändert) bleibt, da sonst die entsprechenden Plug-Ins angepasst werden müssen.

Der Autor entschied sich gegen den Aufbau als SOAP-Nachricht (`http://www.w3.org/TR/soap/`) obwohl dieser es ermöglicht hätte die Meta-Informationen direkt zu transportieren. Gründe für diese Entscheidung sind u.a. der mit SOAP verbundene Overhead der Nachrichten, sowie die kompliziertere Verarbeitung.

Ein komplettes beispielhaftes Dokument im genutzten Format ist unter 4.1 zu sehen. Das Format weist mehrere Hierarchieebenen auf welche verschiedene Informationen bereitstellen. Zunächst enthält das Dokument die Basisinformationen zum Format selber, der benutzten Hardware und deren Kalibrierungsparametern. Daraufhin folgt eine Liste mit beliebig vielen Datensätzen. Diese Ebenen sind im Listing 4.1 orange bzw. rot hervorgehoben. Die genannten Gliederungsebenen enthalten wiederum weitere Abschnitte z.B. für Informationen über die Datenquelle sowie Informationen über die Größe des Bereichs in dem Daten aufgezeichnet werden. Die aufgezeichneten Datensätze liegen in einem Container namens DatasetList. Hierbei sind die Datensätze aufgeteilt in verschiedene Wertekategorien (u.a. zwei- sowie drei-dimensionale Daten) welche jeweils für jedes Auge einzeln dargestellt werden.

Eine Erklärung der einzelnen Wertpaare kann der Tabelle 4.1 entnommen werden. Die im Format genutzten Werte sowie das Koordinatensystem sind an die vom Eye-Tracker „Tobii X120“ (welcher im praktischen Teil dieser Arbeit genutzt wird) gelieferten Daten angelehnt.

Darüber hinaus findet sich im Anhang dieser Arbeit ein komplettes XML-Schema des genutzten Formates. Das Schema erklärt den grundlegenden Aufbau und bietet Raum für diverse Einschränkungen von zulässigen Wertebereichen. Dieses ermöglicht es das Format den entsprechenden Bedürfnissen anzupassen und bestimmte Werte und Wertkombinationen von vornherein auszuschließen. Im Rahmen dieser Arbeit wurde das Format allerdings weitgehend offen gehalten um eine einfache Handhabung und Erklärung zu ermöglichen.

```
<RecordingSession>
 <BasicInformation>
  <FormatVersion>1.0</FormatVersion>
  <FieldUnusedSign>*NA*</FieldUnusedSign>
  <DataSource>
   <DataSourceManufacturer>Tobii</DataSourceManufacturer>
   <DataSourceModel>X120</DataSourceModel>
   <DataSourceProductID>Dummy</DataSourceProductID>
   <DataSourceFirmware>2.07</DataSourceFirmware>
   <DataSourceTrackingRate>120.0</DataSourceTrackingRate>
  </DataSource>
  <ScreenInformation>
   <Size-X>1920.0</Size-X>
   <Size-Y>180.0</Size-Y>
  </ScreenInformation>
 </BasicInformation>
 <DatasetList>
  <Dataset>
   <Timestamp>314159</Timestamp>
   <DataValidity>
```

```
21          <LeftEye>1</LeftEye>
            <RightEye>4</RightEye>
    </DataValidity>
    <PupilDiameter>
25          <LeftEye>23.0</LeftEye>
            <RightEye>42.0</RightEye>
    </PupilDiameter>
    <GazePoint2D>
29     <LeftEye>
        <X-Coord>1411.0</X-Coord>
        <Y-Coord>1412.0</Y-Coord>
       </LeftEye>
33     <RightEye>
        <X-Coord>1421.0</X-Coord>
        <Y-Coord>1422.0</Y-Coord>
       </RightEye>
37          </GazePoint2D>
            <EyePos3D>
             <LeftEye>
              <X-Coord>1511.0</X-Coord>
41            <Y-Coord>1512.0</Y-Coord>
              <Z-Coord>1513.0</Z-Coord>
             </LeftEye>
             <RightEye>
45            <X-Coord>1521.0</X-Coord>
              <Y-Coord>1522.0</Y-Coord>
              <Z-Coord>1523.0</Z-Coord>
             </RightEye>
49          </EyePos3D>
            <RelEyePos3D>
             <LeftEye>
              <X-Coord>1611.0</X-Coord>
53            <Y-Coord>1612.0</Y-Coord>
              <Z-Coord>1613.0</Z-Coord>
             </LeftEye>
             <RightEye>
57            <X-Coord>1621.0</X-Coord>
              <Y-Coord>1622.0</Y-Coord>
              <Z-Coord>1613.0</Z-Coord>
             </RightEye>
61     </RelEyePos3D>
    </Dataset>
  </DatasetList>
</RecordingSession>
```

Listing 4.1: Beispielhaftes XML-Datenformat

Tabelle 4.1.: XML-Datenformat: Erläuterung

Tag	Datentyp	Erläuterung
RecordingSession	–	Rootknoten
BasicInformation	–	Kapselt alle Basisinformationen
FormatVersion	Text	Version des XML-Formates
FieldUnusedSign	Text	Reservierte Zeichenkette für nicht genutzte Datenfelder
Datasource	–	Kapselt alle Informationen zur Datenquelle
DataSourceManufacturer	Text	Hersteller der benutzten Hardware
DataSourceModel	Text	Modell der benutzten Hardware
DataSourceProductID	Text	Serien-/Produktnummer der Hardware
DataSourceFirmware	Text	Firmware der benutzten Hardware
DataSourceTrackingRate	Fließkomma	Aufzeichnungsrate der Quelle (in Hertz)
Screeninformation	–	Kapselt die Informationen über die Fläche auf der Daten aufgezeichnet werden
Size-X	Fließkomma	Größe der Aufzeichnungsfläche in der Horizontalen (gemessen in mm)
Size-Y	Fließkomma	Größe der Aufzeichnungsfläche in der Vertikalen (gemessen in mm)
DatasetList	–	Kapselt alle aufgenommen Datensätze
Dataset	–	Kapselt einzelnen Datensatz
Timestamp	pos. Ganzahl	Zeitpunkt an dem ein Datensatz aufgezeichnet wurde
LeftEye	entsprechend	Datensatz des jeweils kapselten Elements für das linke Auge
RightEye	entsprechend	Datensatz des jeweils kapselten Elements für das rechte Auge
X/Y/Z-Coord	Fließkomma	Jeweilige Koordinate in X/Y/Z-Richtung (gemessen in mm; Null-Punkt: untere-linke Ecke der Projektionsfläche)
DataValidity	pos. Ganzahl	Gibt die Gültigkeit eines Datensatzes an (z.B. Maß für die sichere Erkennung der Augen)
PupilDiameter	Fließkomma	Durchmesser der Pupille (gemessen in mm)
GazePoint2D	Fließkomma	Betrachteter Punkt auf der Projektionsfläche (gemessen in mm)
EyePos3D	Fließkomma	Drei-dimensionale Position des Auges (gemessen in mm)
RelEyePos3D	Fließkomma	Drei-dimensionale Position des Auges relativ zum Aufnahmegerät (gemessen in mm)

4.3.5. Weiterverarbeitung

Die folgenden Abschnitte behandeln die Weiterverarbeitungskomponente. Diese überführt die Daten aus dem XML-Format in die Zieldarstellung (z.B. eine Heatmap-Visualisierung). Diese Komponente dient der Erfüllung der funktionalen Anforderung 3.3.1.3 (a). Da die Weiterverarbeitung auf dem allgemeinen Format aufsetzt ist diese für jede Aufzeichnungsquelle geeignet solange die Daten vorher überführt werden.

Die Architektur dieser Komponente ähnelt stark dem Aufbau der Konverter Komponente, da auch die Endverarbeitung der Daten nur ein spezielles Konsumieren von Daten ist. Daher ist sie wie auch der Konverter nur ein spezieller Middleware-Consumer, welcher die Daten für in ein entsprechendes Ziel konvertiert.

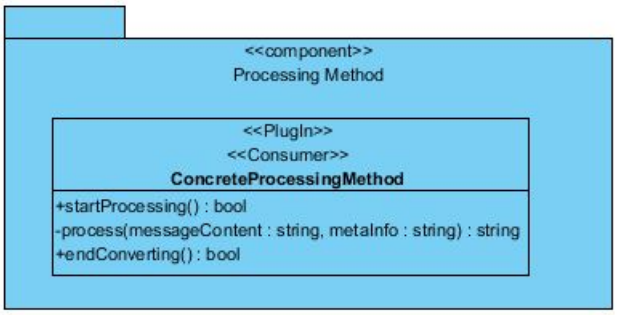

Abbildung 4.8.: Schnittstelle der ProcessingMethod-Komponente

Abbildung 4.8 zeigt den Aufbau eines Plug-Ins zur Weiterverarbeitung. Da die Verarbeitung stark dem Konverter ähnelt tauchen die drei bereits bekannten Funktionen wieder auf: Eine Weiterverarbeitung kann gestartet und beendet werden, zwischen diesen beiden Zeiträumen übernimmt die „process“-Funktion die Verarbeitung von anfallenden Datensätzen. Zu beachten ist, dass eine Weiterverarbeitung beliebig komplex ausfallen kann, daher kann diese weitere Eigenschaften aufweisen welche bei der Initialisierung ggf. mitgeliefert werden müssen. So könnte z.B. auf externe Programme zugriffen werden, deren Pfade bekannt sein müssen, genau wie etwaige Parameter für diese Programme.

5. Implementierung eines Prototypen, Testkonzept sowie Evaluierung

Das folgende Kapitel befasst sich mit einer praktischen Implementierung der zuvor beschriebenen Architektur. Diese ist als Prototyp gedacht welcher zeigen soll, dass das Grundkonzept funktionsfähig ist („Proof-Of-Concept“).

Dieses Kapitel unterteilt sich in mehrere Bereiche:

1. Zunächst wird der Umfang der Entwicklung sowie vorhandene Schwächen / fehlende Features erläutert.

2. Im Anschluss daran werden wichtige Details sowie Besonderheiten der Einzelkomponenten vorgestellt.

3. Daraufhin wird ein grobes Testkonzept für die Anwendung erläutert. Dieses stellt dar welche Komponenten besonders kritisch sein könnten und daher Priorität beim Testen erhalten sollten. Darüber hinaus werden Hinweise gegeben wie diese Komponenten getestet werden können.

4. Zum Abschluss wird eine Evaluierung des Prototypen bezüglich der Anforderungen und Schwächen durchgeführt.

5.1. Umfang

Der entwickelte Prototyp setzt den Grundaufbau der Architektur um, d.h. er besitzt die entsprechenden Plug-Ins zur Datenerfassung, Konvertierung und Verarbeitung. Daher ist auch

ein Lademechanismus für die Erweiterungen vorhanden genauso wie die Kommunikation über die Middleware.

Der Prototyp hat allerdings gewisse Einschränkungen:

- Zum einen wurde die Steuerung der Gesamtapplikation (Komponente „Main Program“) nicht für die interaktive Nutzung entworfen. Bislang startet sie eine Aufnahme, eine Konvertierung sowie eine Weiterverarbeitung für eine gewisse Zeit, so dass das entstehende Ergebnis geprüft werden kann.

- Darüber hinaus ist eine Kalibrierung der gewählten Datenquelle aus Zeitgründen nicht umgesetzt worden. Dieses kann zu Ungenauigkeiten bei der Datenerfassung führen.

- Eine weitere Schwachstelle des Prototyps ist die unzureichende Fehlerbehandlung, welche die Lokalisierung von Fehlern erschwert.

- Weitere Unzulänglichkeiten ergeben sich höchst wahrscheinlich durch die mangelnde C++ Erfahrung des Autors.

Der Prototyp enthält die im Architekturteil dieser Arbeit beschriebenen Plug-Ins:

- Als Datenerfassungsgerät wurde ein „X120“-Eyetracker der Firma „Tobii“ gewählt. Da dieser allerdings im Rahmen der Entwicklung nicht immer verfügbar war existiert darüber hinaus ein weiteres Datenerfassungs-Plug-In welches als Dummy konzipiert wurde. Dieser Dummy liefert zufällig generierte Daten ohne eine Abhängigkeit zu einer Hardware und ist damit vergleichsweise gut zum Testen geeignet.

- Des Weiteren existiert wie gefordert ein Plug-In zur Konvertierung. Dieses funktioniert sowohl für den eigentlichen Eye-Tracker als auch für den entwickelten Dummy. Der Konverter überträgt die Daten in das zuvor erläuterte Format.

- Als Weiterverarbeitung wurde wie angesprochen eine Visualisierung als Heatmap entwickelt. Diese ermöglicht es die Verteilung von während der Datenerfassung betrachteten Punkten zu analysieren.

5.2. Erläuterungen

Das folgende Unterkapitel erläutert Teile der Implementierung der Architektur. Hierbei wird sowohl auf grundlegende Entscheidungen als auch auf Einzelkomponenten eingegangen. Die Erläuterungen beschreiben jeweils nur, aus Sicht des Autors, besonders interessante Fragmente samt des entsprechenden Quellcodes. Dieses soll das Grundverständnis erleichtern – der komplette Code ist im Anhang dieser Arbeit zu finden.

5.2.1. Grundlagen

Wie angesprochen wurde der Prototyp der Anwendung in C++ implementiert. Diese Entscheidung wurde aus mehreren Gründen getroffen:

- Im Rahmen dieser Arbeit ist die Benutzung eines „Tobii X120"-Eyetrackers vorgesehen, dieser muss dementsprechend über das Software Development Kit (SDK) der Firma „Tobii" angesteuert werden. Dieses liegt in Version 3.0 RC 1 vor [siehe: Tobii-e] und unterstützt die Sprachen C# / .NET, Python, C++ sowie Objective-C. Aus diesem Grund ist zumindest für die Umsetzung des Hardware-Adapters eine dieser Sprachen zu wählen.

- Da sämtliche obigen Sprachen nie vom Autor benutzt worden waren und daher ein zusätzliches Hindernis darstellten fiel die Entscheidung die gesamte Applikation in einer Sprache zu implementieren. Dieses diente der Vermeidung von Problemen bei der Zusammenarbeit von Komponenten welche in verschiedenen Sprachen umgesetzt worden wären.

- Durch die obige Entscheidung blieben nur die vier genannten Sprachen zur Auswahl. Der Autor entschied sich zur Umsetzung mittels C++ aus Eigeninteresse eine hardwarenahe Sprache kennenzulernen.

Dieser Implementierung liegen einige Basisentscheidungen zu Grunde welche die Gesamtumsetzung beeinflussen.

- Zunächst entschied sich der Autor aufgrund seiner Unerfahrenheit mit der gewählten Sprache die Implementierung möglichst simpel zu halten. Dieses führt dazu, dass Lösungen nicht immer perfekt und höchst performant sind.

- Darüber hinaus wurden während der Implementierung C++ eigene Mechanismen verwendet. Diese beinhalten u.a. – wie in der Architektur zu erkennen – die Nutzung von Mehrfachvererbung welche relativ große Auswirkungen hat. Allerdings wurden auch kleine Änderungen wie die gelegentliche Verwendung von Strukturen anstelle von kompletten Klassen vorgenommen.

- Des Weiteren wurde zunächst die Fehlerbehandlung nur rudimentär umgesetzt. Dieses erleichtert das schnelle Entwickeln, führt allerdings zu Problemen beim Auffinden von auftretenden Fehlern. Darüber hinaus verwendet der Autor an wichtigen Stellen statische Methoden zur Erzeugung von Objekten. Dieses soll das Erstellen von ungültigen Objekten unmöglich machen, sodass Fehler welche darauf zurückzuführen wären nicht auftreten.

- Eine weitere Grundentscheidung ist es möglichst bekannte und gut unterstützte Projekte zur Umsetzung von externen Funktionalitäten wie der XML-Verarbeitung zu benutzen. Aus diesem Grund entstammen viele Projekte der Apache-Familie. Der Nachteil ist hierbei das einige Projekte relativ veraltet sind.

5.2.2. Plug In Management

Diese Komponente basiert wie angesprochen auf Aufrufen der Windows API. Da diese Aufrufe z.T. vergleichsweise kompliziert sind und mit diversen C++-Pointern etc. operieren wurde die Funktionalität durch die „PlugInManagement“ Komponente gekapselt.

```
1 wchar_t* plugin_full_name_w = builtFilePath(directory, filename);
2
3 HMODULE h_mod = ::LoadLibrary(plugin_full_name_cs);
4
5 if (h_mod != NULL) {
6 PLUGIN_FACTORY p_factory_function = (PLUGIN_FACTORY)::GetProcAddress (h_mod,
7         "Create_Plugin");
8
9 if (p_factory_function != NULL) {
```

```
10 funcs.p_factory_function = p_factory_function;
```

Listing 5.1: MainProgram Bestandteile

Listing 5.1 zeigt einige Basisbestandteile des bekannten „loadDLL(...)“ Aufrufs.

1. Zunächst wird eine weitere interne Funktion „buildFilePath“ aufgerufen (Zeile: 1) welche aus den Parameter des load-Aufrufes einen kompletten Pfad des passenden Datentyps zusammensetzt (dieser ist allerdings nicht entgültig – es folgen diverse Umwandlungen in Pointer-Typen).

2. Nun wird die eigentliche Library in Form einer DLL mittels Windows API Aufruf geladen (das Argument ist der komplette Pfad zur DLL, Zeile 3). Dieses liefert einen entsprechenden Datentyp mit den benötigten Informationen zurück.

3. Die eben gelieferten Daten (sofern korrekt) können nun mit einem Aufruf der Windows API Funktion „GetProcAddress“ nach einer Funktion mit dem Namen „Create_Plugin“ durchsucht werden (Zeile: 5 – 7).

4. Ist diese Funktion vorhanden dann wird sie als Plug-In Factory Funktion verwendet und ist abrufbar (Zeile: 10).

5.2.3. Message orientierte Middleware

Die Middleware Komponente stellt den Kommunikationsmechanismus des Systems dar, hierzu wird ein externer Message Broker benutzt. Im Rahmen dieser Arbeit wurde dazu das Apache ActiveMQ Projekt der Apache Foundation [siehe: ActiveMQ] genutzt. Da dieses sehr komplex ist wurde der Zugriff darauf, wie im Architekturteil beschrieben, auf die Bedürfnisse der Arbeit angepasst und in der Middleware Komponente gekapselt.

```
1 void Consumer::onMessage( const Message* message ) throw() {
2         const TextMessage* textMessage;
3         textMessage = dynamic_cast<const TextMessage*>(message);
4         std::string messageContent = textMessage->getText();
5         std::string messageType = message->getStringProperty("type");
6         process(messageContent, messageType);
7 }
```

Listing 5.2: Middleware-Consumer: onMessage

Listing 5.2 zeigt die „onMessage“ Methode des verwendeten ActiveMQ-Clients [siehe: ActiveMQ-CPP], diese versorgt die später verwendeten „process“ Methoden mit Daten. Die Funktion wird aufgerufen sobald eine neue Nachricht am Broker eintrifft.

1. Zeile 1 zeigt die Signatur der Funktion – das Argument ist eine generische Message und enthält alle Informationen zur am Broker eingetroffenen Nachricht.

2. Diese Message wird in eine konkrete Textnachricht umgewandelt, sodass sie interpretierbar ist. (Zeile: 3).

3. Nun werden die eigentliche Information sowie der Typ aus der Nachricht extrahiert (Zeile: 4 – 5).

4. Abschließend werden die gewonnen Informationen über die „process“ Funktion an einen konkreten Verbraucher weitergereicht.

5.2.4. MainProgram

Die Komponente MainProgram ersetzt einen interaktiven Benutzer. Ein grobes Beispiel für den Ablauf ist in Listing 5.3 zu sehen.

1. Diese Komponente baut zunächst die PlugIn-Verwaltung auf.

2. Daraufhin lädt es entsprechende Plug-Ins (Zeile: 1 – 2). Hierbei wird der Container PlugInFunctions belegt woraufhin die geladenen Erweiterungen Objekte anhand ihrer Parameter (übergeben als String-Array) erstellt werden können. Hierzu wird die durch das Laden belegte Factory-Funktion ausgerufen. Die nun erstellten Objekte müssen noch durch einen „Cast“ in ein konkretes Objekt umgewandelt werden um benutzbar zu sein (Zeile: 8).

3. Ist ein Plug-In erfolgreich initialisiert kann es seine Arbeit aufnehmen und nach einiger Zeit beenden (Zeile: 11 – 13).

```
1 PlugInManagement* pluginManagement = new PlugInManagement();
2 PlugInFuntions hardwareAdapterPlugin = pluginManagement->loadDLL("C:/",
3         "HardwareAdapter-Dummy.dll");
4
```

```
5  string adapterStrArray[5] = {"169.254.8.74", "4455", "localhost", "61616",
6          "HardwareAdapterInput"};
7
8  HardwareAdapter* adapter = static_cast<HardwareAdapterDummy*>
9          (hardwareAdapterPlugin.p_factory_function(adapterStrArray));
10
11 adapter->startRecording();
12         [...]
13 adapter->stopRecording();
```

Listing 5.3: MainProgram: Grundaufbau

5.2.5. HardwareAdapter

Ein Plug-In vom Typ HardwareAdapter dient der Datenerfassung. Dieser Arbeit liegen zwei verschiedene HardwareAdapter bei:

- Das erste Plug-In dient der Ansteuerung eines „Tobii X120“ Eyetrackers. Dieses verfügt über die vorgestellten Funktionen zur Datenerfassung, verzichtet aber auf eine vergleichsweise komplizierte Kalibrierung des Trackers.

- Als zweites Plug-In wurde ein Dummy-HardwareAdapter implementiert. Dieser liefert Daten im Format des "Tobii“-Trackers kann allerdings unabhängig von vorhandener Hardware eingesetzt werden. Dieses erleichtert das Testen der Applikation in Abwesenheit eines Eyetrackers. Die übermittelten Daten sind hierbei z.T. statisch vorgegeben bzw. werden zufällig generiert.

```
1  tracker->add_gaze_data_received_listener(
2                  boost::bind(&TobiiX120::processDataset, this, _1));
3
4  this->tracker->start_tracking();
```

Listing 5.4: Eye-Tracker: Kern der „startRecording“ Funktion

Listing 5.4 zeigt den Kern der „startRecording“ Methode des „Tobii Eye-Tracker“ Plug-Ins.

1. Zunächst werden die Daten eines vorher aufgebauten Eye-Trackers abonniert (subscribe) indem ein Event-Listener gestartet wird, welcher aufgerufen wird sobald der Tracker neue Daten aufgezeichnet hat . Dieser Listener leitet die empfangenen Daten an das als Parameter übergebene Funktionsobjekt weiter (Zeile: 1).

2. Dieses Funktionsobjekt wird mit Hilfe der Funktion bind aus den Boost-Libraries ([siehe: Boost]) erstellt. Bind erstellt hierzu ein Objekt welches die Funktion „processDataset“ der aktuellen Klasse („this“) und ein Argument (den durch den Listener erhaltenen Daten) enthält – die Funktion und das entsprechende Argument sind aneinander gebunden (Zeile: 2). Diese Herangehensweise ermöglicht es dem Listener beliebig komplexe Funktionen zu übergeben ohne deren Signatur zu kennen.

3. Zeile 4 zeigt wie dem Tracker der Befehl zum Start der Aufnahme übermittelt wird.

5.2.6. DataConverter

Diese Komponente konvertiert die vom Aufnahmegerät gelieferten Daten in ein XML-Datenformat. Hierbei werden die Nachrichten anhand ihres Typs nach dem Beginn / Ende einer Aufzeichnung bzw. einem Datensatz während der Aufzeichnung unterschieden.

Listing zeigt den Teil der „process“ Funktion des Converters welcher normale Datensätze verarbeitet.

```
1 if (messageType == "dataset") {
2         std::string output = convertDataset(messageContent);
3         this->getProducer()->send(output, messageType);
```

Listing 5.5: Converter: „process“ Funktion für Datensätze

1. Zeile 1 stellt sicher, dass in diesem Teil der Funktion nur Daten verarbeitet werden welche laut Meta-Information Datensätze sind.

2. Danach wird der Nachrichteninhalt der entsprechenden Konvertierungsfunktion übergeben. Diese filtert aus dem Inhalt zunächst die relevanten Informationen hinaus (split) und bringt diese dann in das XML-Format welches zurückgegeben wird (Zeile: 2).

3. Letztendlich wird werden die konvertierten Daten an den Broker weitergegeben, sodass die Weiterverarbeitung diese abrufen kann (Zeile: 3).

5.2.7. ProcessingMethod

Diese Komponente dient der Weiterverarbeitung von Daten. Der Autor entschied sich für diesen Teil der Arbeit dazu die aufgezeichneten Daten als Heatmap zu visualisieren.

Zur Umsetzung dieser Weiterverarbeitung bedarf es einer Reihe von Werkzeugen:

R – Heatmap Visalisierung: Die eigentliche Aufarbeitung der Daten geschieht unter zur Hilfenahme der Programmiersprache „R“ [siehe: R]. Diese bietet verschiedene Pakete an, welche u.a. auch die Erstellung einer Heatmap ermöglichen (siehe: 5.1). Die Erstellung einer solchen Visualisierung erfordert entsprechende Daten: Also die möglichen Punkte auf der Projektionsfläche des Aufnahmegerätes sowie die entsprechende Anzahl welche aussagt wie oft ein Punkt betrachtet wurde.

Xalan C – XPath: Die obigen Daten müssen zunächst aus dem XML-Datenformat extrahiert werden. Der Autor entschied sich dazu dieses mittels XPath-Ausdrücken (http://www.w3.org/TR/xpath/) umzusetzen. Mit diesen kann das Datenformat vergleichsweise einfach verarbeitet werden um die nötigen Informationen zu gewinnen. Eine geeignete Implementierung des XPath-Standards bietet das Apache Projekt Xalan an [siehe: Xalan].

Um die Handhabung der XPath-Ausdrücke zu vereinfachen entschied sich der Autor dazu eine API zur Nutzung von XPath auf Basis des Xalan-Frameworks zu implementieren (siehe: A – Projektordner XPath). Ein Beispiel für die Verwendung dieser XPath-API ist in Listing 5.6 zu sehen.

```
1 XalanXPathEvaluator* eval = XalanXPathEvaluator::create(0, this->xmlString);
2
3 XObjectPtr sizeX = eval->evaluate("/",
4       "/RecordingSession/BasicInformation/ScreenInformation/Size-X/text()");
5
6 std::string screenSizeXStr.assign (sizeX->str().begin(), sizeX->str().end());
7
8 float screensizeX = atof(screenSizeXStr.c_str());
```

Listing 5.6: Verwendung der XPath-Auswertung

1. Zunächst muss ein XPath-Evaluatpor aufgebaut werden, dieser erhält als Argumente die Art sowie die Daten eines zu ladenden XML-Dokuments. In diesem Fall wird ein XML-String aus dem Speicher als Eingabe verwendet (Zeile: 1).

2. Zeile 3 – 4 zeigt die Evaluierung eines als String übergeben XPath-Ausdruckes. Das Ergebnis der Evaluierung wird in einem xalan-internen Datentyp gespeichert.

3. Da dieses Ergebnis verschiedene Inhalte haben kann (Texte, Zahlen, Knotenmengen) muss es gesondert ausgewertet werden. In diesem Fall ist der Inhalt ein Fließkommawert. Dieser wird zunächst in einen String überführt, da nicht direkt auf die enthaltene Zahl zugegriffen werden kann (Zeile: 6).

4. Zuletzt wird die enthaltene Zahl aus dem String extrahiert womit das Ergebnis der Auswertung vorliegt (Zeile: 8).

Xerces C – XML-Parser: Das erwähnte Xalan Projekt setzt auf dem Xerces XML-Parser auf, welcher ebenfalls ein Apache Projekt ist. Dieser Parser bietet u.A. alle Grundfunktionalitäten welche durch Xalan benutzt werden.

Die angesprochene Visualisierung mit Hilfe von „R“ benutzt zur Generierung der Heatmap das „R“ eigene Paket „heatmap2“ (http://hosho.ees.hokudai.ac.jp/~kubo/Rdoc/library/gplots/html/heatmap.2.html). Die Funktionen dieses Paketes werden innerhalb eines „R“-Scripts benutzt um ein Bild im JPEG-Format zu generieren. Die Datenbasis für diese Verarbeitung wird mittels XPath-Ausdrücken aus dem XML-Format extrahiert und als csv-Format an das „R“-Script übergeben.

```
1  x <- read.table(textConnection(data), sep=";", head=T, dec=",", allowEscapes=T)
2  mat=data.matrix(x)
3
4  jpeg(filename = args[6], width = 1024, height = 768,
5          pointsize = 12, bg = "white")
6
7  heatmap.2(mat,
8          xlab = "X Coord", ylab = "Y Coord",
9          col=brewer.pal(9,"Oranges"),
10 [...]
11 )
```

Listing 5.7: Weiterverarbeitung: Teile des Heatmapscripts

Listing 5.7 zeigt Ausschnitte des erwähnten Scripts. Hierbei enthält "data"die als csv-String übergebene Datenbasis.

1. Zunächst werden die csv-Daten aufgearbeitet und in eine „R“-Matrix überführt (Zeile: 1 – 2).

2. Anschließend wird das Ausgabeformat samt Parametern (u.A. Pfad welcher als args[6] eingelesen wird) bestimmt (Zeile: 4 – 5).

3. Abschießend wird die Generierung der Heatmap gestartet. Hierbei werden die benötigten Parameter wie Daten, Achsenbeschriftungen, Farben und ggf. mehr übergeben (Zeile: 7 – 11).

Abbildung 5.1 zeigt eine beispielhafte Heatmap, welche aus den Daten des Dummy-HardwareAdapters generiert wurde.

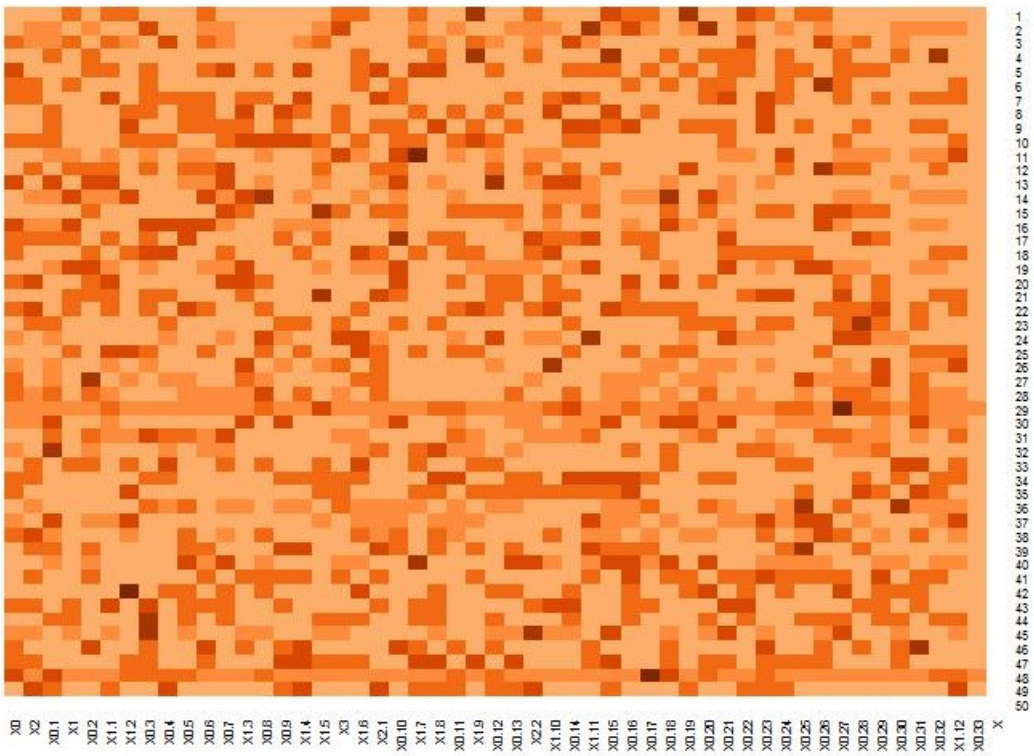

Abbildung 5.1.: Beispielhafte Heatmap

5.2.8. „Hello World“ Plug-In

Dieser Abschnitt soll zur Erläuterung des Grundaufbaus eines Plug-ins dienen. Hierzu wird ein grundlegendes „Hello World“ Plug-In entworfen. Dieses ist in Listing 5.8 zu sehen.

```
1  #include "HelloWorldPlugIn.h"
2
3  HelloWorldPlugIn::HelloWorldPlugIn(std::string description) {
4      this->description = description;
5  }
6
7  HelloWorldPlugIn::someFunction() {
8          printf("Hello World! My function is: %s", this->description);
9  }
10
11 extern "C" {
12
13         __declspec(dllexport) void* Create_Plugin(std::string* args)  {
14
15                 std::string plugInDescription = args[0];
16
17                 HelloWorldPlugIn* somePlugIn = new HelloWorldPlugIn(plugInDescription);
18                 void* plugin = static_cast<void*>(somePlugIn);
19                 printf("Plugin created");
20                 return plugin;
21         }
22
23         __declspec(dllexport) void Release_Plugin(HelloWorldPlugIn* p_plugin) {
24                 delete p_plugin;
25                 printf("Plugin deletet");
26         }
27 }
```

Listing 5.8: „Hello World“ Plug-In

1. Zunächst werden die entsprechenden Header-File für die Klasse des Plug-Ins eingebunden (Zeile: 1). Diese enthalten den grundlegenden Aufbau der Klasse samt ihrer Abhängigkeiten.

2. Grundsätzlich sollte die Klasse einen Konstruktor, einen Destruktor etc. aufweisen, dieses ist in Zeile 3 angedeutet. Die Beispielklasse hat eine Eigenschaft welche sie beschreibt.

3. Des Weiteren enthält das beispielhafte Plug-In eine Funktion „someFunction“, diese steht stellvertretend für die Logik der Klasse (Zeile: 7).

4. Zeile 11 stellt die Besonderheit der Plug-In Umsetzung dar. Die beiden zwingend benötigten Funktionen eines Plug-Ins zur Erstellung und Zerstörung sind als „C“-Funktionen umgesetzt, daher sind diese als „extern“ deklariert.

5. Die Funktion zur Erstellung eines Plug-Ins empfängt ihre Argumente als String-Array (Zeile: 15) und nutzt diese zum Aufbau eines Objektes der Plug-In Klasse (Zeile: 17). Das entstandene Objekt wird zu in einen void-Pointer gecasted (Zeile: 18) – dieses ermöglicht es ein universelles Objekt zurückzugeben ohne je vorgeschriebene create-Methode eines Plug-Ins zu ändern. Dieses würde die Architektur nachhaltig beeinflussen falls die Plug-In Methoden je vererbt werden sollten.

6. Die Funktion zum Löschen eines Plug-Ins ist in Zeile 23 zu sehen. Diese dient dazu einer Erweiterung vor ihrer Löschung die Möglichkeit zu geben gehaltene Ressourcen freizugeben.

5.3. Testkonzept

Die folgenden Abschnitte umreißen ein grobes Testkonzept für die entwickelte Applikation. Dieses Konzept dient dazu den Einstieg in eine detaillierte Testphase zu erleichtern und Testmethodiken sowie Prioritäten aufzuzeigen.

5.3.1. Testziel

Das Ziel der Testphase für diese Applikation ist es zunächst die Erfüllung der in Kapitel 3.3.1 gestellten Anforderungen sicherzustellen. Dieses gilt sowohl für die funktionalen Anforderungen sowie soweit wie möglich auch für die nicht funktionalen Anforderungen.

Darüber hinaus soll entsprechend neben den Kernkomponenten der funktionalen Anforderungen auch das gesamte System korrekt funktionieren. Hierbei ist vor allem die Middleware von Bedeutung, da diese die Kommunikationsmöglichkeit des Systems darstellt.

5.3.2. Testumgebung

Da das Gesamtsystem auf Basis bestimmter Technologien entworfen wurde empfiehlt es sich (zunächst) ein gleich aufgebautes System zu nutzen um Kompatiblitätsfehler auszuschließen.

Das zu testende System wurde explizit für ein Windows 7 Professional System entworfen. Da an verschiedenen Stellen der Implementierung windows-eigene Mechanismen und Bibliotheken verwendet wurden empfiehlt der Autor daher die Nutzung eines solchen. Dieser Hinweis gilt auch für alle in Abschnitt 2.2 aufgeführten Technologien.

Die zu testende Applikation lässt sich durch ihren Aufbau in verschiedene zu testende Schichten aufteilen. Diese Aufteilung dabei beschreibt grob den Ablauf der Applikation und damit den Fluss der zu verarbeitenden Daten.

- Plug-In Verwaltung
- Message orientierte Middleware samt Message Broker
- Netzwerkkommunikation
- Datenerfassung
- Konvertierung sowie Datenformat
- Weiterverarbeitung

Die genannten Schichten bieten einen guten Ansatzpunkt zur Strukturierung und Priorisierung von Testfällen.

5.3.3. Risikobetrachtung

Die Gefahren welche sich durch ungenügende Tests ergeben sind für das Ergebnis der Gesamtapplikation z.T. fatal. Hierbei steigt die Gefahr eines Fehlers mit seiner Tiefe in der Hierarchie der Applikation.

Während eines Programmdurchlaufes bauen die genannten Bestandteile aufeinander auf, d.h. produziert eine dieser Komponenten ein falsches Ergebnis kann dieses dazu führen, dass das gesamte System unerwünschte Resultate liefert. Daher ist ein Fehler während der Weiterverarbeitung von Daten tendenziell weniger schwerwiegend als ein Fehler bei der Übertragung der Daten an den Message-Broker über das Netzwerk. Während ein Fehler in der Weiterverarbeitung im Normalfall dazu führt, dass ggf. einen falschen Datensatz produziert wird kann ein Fehler in der Middleware zum Verlust aller Daten führen. Hierbei hängt es natürlich immer davon ab wie stark sich ein Fehler auswirkt, dennoch ist die Gefahr in den unteren Schichten des Systems inhärent höher.

5.3.4. Testgegenstände

Die zu testenden Bestandteile korrelieren mit den genannten Schichten. Besonders hervorzuheben sind hierbei die kommunikativen Bestandteile einer Komponente, sowie die entsprechende Fachlogik, da diese jeweils das Ergebnis transportieren bzw. produzieren.

Die folgende Liste zählt besonders empfindliche Bestandteile auf welche besonders intensiven Tests unterliegen sollten, da die weitere Verarbeitung von Daten auf ihnen beruht.

- Laden eines Plug-Ins
- Funktionsfähige Netzwerkkommunikation
- Message-Broker
- Korrekt funktionierende Middleware – also korrektes Konsumieren sowie Versenden von Nachrichten
- Erstellung von Datenerfassungs-, Konvertierungs- und Weiterverarbeitungsobjekten mit jeweils richtigen Daten
- Die entsprechende Fachlogik der Kernkomponenten – insbesondere die „process“-Funktionen zur Verarbeitung von eintreffenden Daten

Da die Applikation (je nach Komplexität der Plug-Ins) auf diverse externe Bibliotheken zugreift sind die Testmöglichkeiten in diesen Bereichen beschränkt. Diese lassen sich daher nur begrenzt über auf ihre korrekte Funktionsweise hin überprüfen.

5.3.5. Testmethodiken

Es empfiehlt sich die Tests entsprechend der genannten Schichten aufzubauen, da z.B. eine fehlerhafte Netzwerkkommunikation die weitere Verarbeitung unmöglich macht ist dementsprechend das Testen von höheren Komponenten wie des Konverters nicht sinnvoll. Aus diesem Grund sollte zunächst das funktionieren der tieferen Schichten sichergestellt sein.

Darüber hinaus bietet es sich grundsätzlich an die Resultate von Nachrichtenübertragungen durch den Message-Broker zu überprüfen. Dieser bietet im Falle von Active-MQ eine web-basierte Administrationsoberfläche (Standard: `localhost:8161/admin/`) an über welche verfolgt werden kann welche Nachrichten übertragen wurden.

Da konkrete Testfälle bzw. deren erwartete Resultate abhängig sind von den gewählten Plug-Ins ist es schwer beispielhafte Testdaten anzugeben. Allerdings kann man Testläufe erleichtern indem man zunächst möglichst überschaubare Testdaten generieren lässt. Zu diesem Zweck können diverse Dummys zur Datenlieferung und Verarbeitung implementiert werden. Diesen kann man z.B. fest definierte Resultate mitgeben, sodass Fehler einfacher zu entdecken sind als bei dynamisch bzw. zufällig generierten Daten.

5.4. Evaluierung

Die Entwicklung der Architektur sowie des dazugehörigen Prototypen verlief weitestgehend erfolgreich. Die funktionalen Anforderungen wurden vollständig erfüllt, das Erreichen der nicht-funktionalen Anforderungen wiederum kann allerdings nur beschränkt als Erfolg gewertet werden. Dieses wird im weiteren vertieft.

Die Tests zur Erfüllung der nicht-funktionalen Anforderungen wurden mit drei verschiedenen Konfigurationen mit jeweils verschiedenen Zeitfenstern durchgeführt. Der Testrechner war

hierbei ein durchschnittlich ausgelasteter Heimcomputer (CPU: AMD Athlon 64 X2 – 2,6 GHz, 3 GB RAM).

Konfiguration 1: Standardkonfiguration – Alle Verarbeitungsschritte aktiv, default-Einstellungen der ActiveMQ (geringe Größenbeschränkung für Topics – „memoryLimit = 1mb“). Die zugehörigen Testdaten sind in Tabelle A.1 zu finden.

Konfiguration 2: Alle Verarbeitungsschritte aktiv, größeres Speicherlimit der ActiveMQ Topic des Brokers (1000 mb). Die zugehörigen Testdaten sind in Tabelle A.2 zu finden.

Konfiguration 3: Heatmap-Auswertung deaktiviert ActiveMQ Einstellungen wie in Konfiguration 2. Die zugehörigen Testdaten sind in Tabelle A.3 zu finden.

Die Ergebnisse der obigen Messungen werden in Grafik A.1 visualisiert, sodass man die Hauptergebnisse der Messungen einfach ablesen kann. Hierbei werden die jeweiligen Durchschnittszeiten für Konvertierung (gestrichelte Linien) und Erfassung (durchgängig) von Daten in den einzelnen Konfigurationen (Rottöne = Standardkonfiguration; Gelbtöne = Konfiguration 2; Grüntöne = Konfiguration 3) in Abhängigkeit zur Aufnahmezeit dargestellt.

Effizienz

Die Forderung nach Effizienz wurde in den Anforderungen insbesondere auf die Datenquelle bezogen, da deren Übertragungsverhalten nicht beeinflusst werden kann. Anhand der Testdaten lässt sich erkennen, dass der gesetzte Grenzwert von 8,3ms (siehe: a) für die Aufnahme eines Datensatzes nur in der Standardkonfiguration bei längeren Aufnahmezeiten überschritten wird (siehe Testreihe A.1). In allen anderen Fällen wird diese Grenze allerdings weit unterschritten (siehe Testreihen A.2 und A.3). Damit kann die Anforderung der Effizienz als erfüllt angesehen werden. Anzumerken ist hierbei, dass die Tests mit der Dummy-PlugIn zur Datenerfassung durchgeführt wurden. Dieses liefert pro Sekunde weitaus mehr Datensätze (vgl. Messreihen) als der „Tobii“-Eyetracker (maximal 120 Datensätze pro Sekunde). Da der Dummyadapter ein Vielfaches an Datensätzen liefert (welche zunächst erzeugt werden müssen) steigt daher entsprechend auch die Auslastung des Testsystems. Das Testszenario ist also weitaus aufwendiger als eine reale Datenerfassung mit dem angesprochen Eyetracker.

Die Messungen zeigen, dass das Verhalten des Systems zunächst einmal von den gegebenen Limitierungen des Brokers abhängt (siehe Testreihe A.2). Ist das Speicherlimit für eine Topic zu gering so füllt sich diese bis zum Limit woraufhin neue Daten zunächst abgewiesen werden müssen bis wieder Speicher bereit steht (dieser wird erfolgreicher Versendung von gespeicherten Datensätzen an eine abonnierende Instanz freigegeben).

Ein weiterer Aspekt der die Effizienz des Systems beeinflusst ist die Auslastung des Rechners. Senkt man diese (z.B. durch das Deaktivierung der Heatmap-Weiterverarbeitung) steigt die Effizienz der anderen Verarbeitungsschritte an (siehe Testreihe A.3). Die Auslastung des Systems könnte beispielsweise durch die angesprochene Verteilung der Einzelverarbeitungsschritte auf verschiedene Rechner erfolgen. Hierbei ist allerdings zu beachten, dass der Messagebroker die zentrale Kommunikation darstellt – dieser könnte sich also bei starker Verteilung als Flaschenhals erweisen.

Weitere Faktoren, die Einfluss auf das Verhalten des Systems haben können sind:

- Die Umsetzung eines Verarbeitungsschrittes. Betrachtet man beispielsweise die Weiterverarbeitung als Heatmap so bemerkt man, dass die Evaluierung von XPath-Ausdrücken enorm aufwändig ist und damit die Performanz des gesamten Systems senkt.

- Ein stark ausgelastetes Netzwerkinterface, da die Daten samt der jeweiligen Empfangsbestätigungen aller Protokollebenen über dieses abgewickelt werden. Treten durch den hohen Verkehr zu großen Verlustraten und damit erneute Übermittlungen ein könnte ein sich selbst speisender Stau im Netzwerk entstehen.

- Ein weiterer Grund für vergleichsweise schlechte Performanz ist die sehr einfach gehaltene Implementierung. Diese übergibt viele Daten per Wert anstatt per Referenz, so entsteht ein hoher Aufwand durch das unnötiges Kopieren von Daten.

- Des Weiteren muss bedacht werden, dass die Verarbeitungsschritte aufeinander aufbauen und damit ein Folgeschritt erst arbeiten kann sobald er Daten erhalten hat. Dieses führt vor allem zu Beginn zu einer leichten Verzögerung.

Die Messreihen zeigen, dass das System die gestellten Anforderungen an die Effizienz und Performanz erfüllt. Gleichzeitig zeigen sie aber auf, dass eine Echtzeitnutzung in dieser Phase unrealistisch ist. Bestätigt wird dieses u.a. durch den Fakt, dass die Weiterverarbeitung von

Daten (also das Empfangen der konvertierten Datensätzen gefolgt von der eigentlichen Verarbeitung dieser) nicht effizient geschieht. So ist die Anzahl der von der Weiterverarbeitung empfangenen Datensätze zum Ende der Konvertierung nur ein Bruchteil der zu bearbeitenden Daten (vgl. Messreihen). Diese Performanzproblematik könnte ggf. durch eine bessere Implementierung und Verteilung der Anwendung realistischer werden.

Zuverlässigkeit

Eine weitere Anforderung stellte die Zuverlässigkeit dar, diese sollte eine Verlustrate von 0% aufweisen. Betrachtet man die Übertragungsstatistiken des Brokers so zeigt sich, dass diese Anforderung ebenfalls erfüllt wird. Hierbei ist allerdings zu beachten, dass nach Beendigung einer Aufnahme die Konvertierung sowie die Weiterverarbeitung automatisch reagieren und keine weiteren Nachrichten der Datenquelle entgegen nehmen. Aus diesem Grund gehen Nachrichten, welche zu dieser Zeit noch auf dem Transportweg sind bewusst verloren.

Geringe Kopplung

Die letzte nicht-funktionale Anforderung welche gestellt wurde war die möglichst geringe Kopplung. Diese sollte sich maximal auf die Kopplung über Daten beschränken. Die Forderung wurde auf Ebene der Kernbestandteile (Datenerfassung, Konvertierung, Verarbeitung) erfüllt, da diese nur über jeweils bekannte Datenformate aneinander gebunden sind. Allerdings arbeiten die drei Kernkomponenten jeweils mit der Middleware-Komponente zusammen um ihre Aufgabe zu erfüllen. Daher sind sie an die Middleware gebunden und verletzten auf dieser Ebene die Forderung der geringen Kopplung.

Probleme

Während der Implementierung tauchten diverse Probleme auf, die auf die unzureichenden C++ Kenntnisse des Autors, minimale Dokumentation von benötigten Libraries sowie Fehler bei der Architekturentwicklung zurückzuführen waren. Diese Probleme wurden jeweils gelöst, allerdings wurde die ursprüngliche Architektur im Laufe der Implementierung immer wieder angepasst und erweitert um Fehler zu beheben oder neue Features zu nutzen.

Bekannte Probleme im aktuellen Zustand der Implementierung sind vor allem:

- Zunächst die extrem schlechte Skalierung der Performanz bei längeren Aufnahmezeiten (siehe: 5.4).
- Die vom Autor gewählte rudimentäre Fehlerbehandlungsstrategie, welche die Stabilität der Anwendung gefährden kann.
- Ein weiteres Problem ist die Weiterverarbeitung in Form einer Heatmap. Diese ist zwar voll funktionsfähig, allerdings auch wenig performant. Dieses hängt mit der wiederholten Verwendung von XPath-Ausdrücken zusammen, welche extrem aufwendig sind.
- Darüber hinaus ergaben sich gegen Ende der Arbeit Probleme innerhalb des „Tobii X120“ Plug-Ins. Dieses ist aus bisher ungeklärten Ursachen nicht immer in der Lage eine Verbindung zum Eye-Tracker aufzubauen. Daher wurden abschließende Tests mit dem Dummy-Hardwareadapter durchgeführt.

6. Fazit und Ausblick

6.1. Methodische Abstraktion

Der folgende Abschnitt stellt noch einmal kurz die grundsätzlichen Ideen dieser Arbeit unabhängig vom Kontext dar und dient damit als methodische Abstraktion.

Datenformat

Ein allgemeines Datenformat erlaubt die Verarbeitung von semantisch ähnlichen Daten welche aber syntaktisch unterschiedlich repräsentiert werden. Darüber hinaus bietet es eine allgemeine Basis für die Verarbeitung von Daten.

Vorteile

- Alle an der Verarbeitung beteiligten Bestandteile können auf einem Datenformat arbeiten. Das Datenformat bietet daher gewissermaßen eine Abstraktionsschicht an und erleichtert damit die Verringerung der Kopplung.
- Je nach Datenformat ohne Weiteres übertragbar auf andere Plattformen und Sprachen.

Nachteile

- Das gewählte Datenformat muss zunächst so entworfen werden, dass alle Komponenten es kennen, daher ist es für eine Einzelkomponente unmöglich das Format eigenständig zu erweitern, sofern diese Erweiterung nicht von allen folgenden Verarbeitungsschritten interpretiert werden kann. Das Datenformat muss dementsprechend das kleinste

gemeinsame Vielfache alle Arbeitsschritte sein. Hierdurch enthält das Format ggf. sehr viel Overhead.

- Darüber hinaus ist die Stabilität des Datenformates essenziell - bei einer Änderung des Formats muss diese für alle Verarbeitungsschritte bedacht und gegebenenfalls angepasst werden.

Plug-Ins

Plug-Ins bieten eine gute Methode eine Anwendung zu erweitern ohne dabei den Kern der Anwendung erweitern zu müssen.

Vorteile

- Ermöglicht es eine Anwendung zu erweitern.
- Ermöglicht die Austauschbarkeit von Bestandteile und löst diese damit aus einem starren Programmcode heraus.

Nachteile

- Die Komplexität einer Anwendung wird durch die nötige Infrastruktur der Plug-Ins erhöht.
- Der Grundaufbau eines Plug-Ins hängt sowohl von der verwendeten Plattform und Sprache ab.

Aufeinander aufbauende Verarbeitungsschritte

Der Ansatz die Verarbeitung von Daten schrittweise vorzunehmen ermöglicht es die Anwendung klar zu strukturieren und ggf. die Effizienz zu steigern.

Vorteile

- Klare Struktur und Verantwortlichkeiten der Verarbeitungsschritte durch Kapselung von Einzelbestandteilen.
- Mögliche Effizienzsteigerung durch „pipelining“ zwischen den Verarbeitungsschritten - D.h. der nächste Arbeitsschritt kann ggf. seine Arbeit beginnen während sein Vorgänger weiterhin Daten liefert.
- Nahezu unabhängig von der Plattform oder der verwendeten Sprache.

Nachteile

- Mögliche Erhöhung der Komplexität der Anwendung.
- Datenübertragung zwischen den Verarbeitungsschritten wird benötigt.

Message orientierte Middleware

Eine Nachrichten-Middleware kann als Kommunikationsmethode eines Systems eingesetzt werden und dadurch Übertragungssicherheit und ggf. die Effizienz erhöhen.

Vorteile

- Datentransfersicherheit: Übertragene Nachrichten können zwischengelagert werden bis sie ihr Ziel erreicht haben.
- Mögliche Effizienzsteigerung die einfache Verteilung einer Anwendung auf mehrere Rechner.
- Einfache Verteilung von Daten an mehrere Empfänger.

Nachteile

- Erhöhung der Komplexität der Anwendung.
- Datenübertragung über ein Netzwerk muss beachtet werden aufgrund von Latenzzeiten, Laufzeiten und möglichen Angriffen auf die übertragenen Daten.

- Je nach Verwendung kann die Middleware ein Single-Point-Of-Failure sein.
- Middleware ist gegebenenfalls auf der gewählten Plattform nicht verfügbar.

6.2. Zusammenfassung

Ziel dieser Arbeit war es eine Anwendungsarchitektur für die Verarbeitung von Eye-Tracker Daten zu entwickeln. Diese sollte es erlauben die Einzelbestandteile zu separieren um eine einfache Möglichkeit der zur Ersetzung zu bieten.

Diese Architektur wurde auf Basis der bekannten Stile „Pipe & Filter“ sowie einer Nachrichten basierten Middleware erstellt. Die Möglichkeit zum Austausch von Plug-Ins wurde hierbei durch die Nutzung von windows-internen Mechanismen realisiert. Die verschiedenen Schritte des Verarbeitungsprozesses wurden jeweils durch Anwendungserweiterungen zur Datenerfassung, der Datenkonvertierung sowie der Weiterverarbeitung umgesetzt. Hierbei entkoppelt ein geräte-abhängiges Format die Datenquelle an die Konvertierung während ein erweiterbares XML-basiertes Format die Schritte der Konvertierung und Weiterverarbeitung entkoppelt.

Der gewählte Aufbau zeigte dabei mehr Vorteile als ursprünglich erwartet. So bringt die Entkopplung via Middleware nicht nur eine höhere Flexibilität und Zuverlässigkeit mit sich sondern ermöglicht auch eine einfache Verteilung der Einzelkomponenten. Dieses kann damit die mögliche Steigerung der Effizienz des Systems bewirken ohne Mehraufwand zu verursachen.

Die Implementierung der entwickelten Architektur erwies sich im Gegensatz zeitweise als zeitraubend und kompliziert. Dieses hing z.B. mit den diversen benutzten externen Bibliotheken sowie der Sprachwahl des Autors zusammen. Die Sprache „C++“ war dem Autor vorher unbekannt, daher erwiesen sich u.a. explizites Speichermanagement, Zeigerbehandlung sowie u.a. Probleme bei der Handhabung von verschiedenen String-Datentypen (samt Codierung) als ungewohnt. Ebenso waren verschiedene Bibliotheks-APIs nur minimal dokumentiert. Grundsätzlich waren die Wahlen des Autors aber lehrreich und interessant.

Insgesamt lässt sich sagen, dass das Ziel dieser Arbeit erreicht wurde auch wenn noch Verbesserungspotenzial besteht. Ein großes Problem ergibt sich durch die mangelnde Performanz

welche Echtzeitnutzung zunächst ausschließt. Darüber hinaus brauchen die genannten Bereiche der Implementierung mehr Aufmerksamkeit. Grundsätzlich bietet diese Arbeit allerdings auch die Möglichkeit eigene Ideen einzubringen und andere Möglichkeiten der Umsetzung zu testen.

6.3. Ausblick

Diese Arbeit soll eine Grundlage für aufbauende Arbeiten bieten.

- Hierzu ist es allerdings sinnvoll die angesprochen Probleme zunächst zu beheben.
- Des Weiteren wäre eine graphische Benutzeroberfläche samt funktionsfähiger Kalibrierung für Datenquellen von Vorteil.
- Ebenso ist eine Portierung auf andere Betriebssysteme möglich, da die genutzten Technologien in der ein oder anderen Form jeweils zur Verfügung stehen.
- Natürlich sind weitere Plug-Ins für Datenquellen und vor allem für Weiterverarbeitungsmethoden von Nöten.
  - Diese könnten z.B. andere Formen der Weiterverarbeitung umsetzen.
  - Auch eine gleichzeitige Darstellung des betrachten Inhalts und der entsprechenden Auswertung wäre sinnvoll. Dieses könnte z.B. durch eine Überlagerung der Heatmap mit dem Bildschirminhalt zum Aufnahmezeitpunkt geschehen. Dieses wurde aus Zeitgründen in dieser Arbeit nicht umgesetzt.
  - Darüber hinaus wäre es interessant zu untersuchen wie sich diese Architektur bei interaktiven Verwendungszwecken (wie der Steuerung der Maus via Eye-Tracker) verhält (nachdem das grundlegende Problem der dafür notwendigen Performanz behoben wurde). Auf dieser Basis ließen sich weitere Projekte umsetzen.
- Weiterhin wäre es denkbar eine Import/Export Funktionalität zu entwickeln welche die Verarbeitung von älteren Aufzeichnungen ermöglichen würde.

Grundsätzlich bietet die vorliegende Arbeit reichlich Raum für neue Ideen und Experimente in Form von Anwendungserweiterungen für andere Gerätetypen und Verarbeitungsmethoden – Dieser Raum sollte genutzt werden.

A. Anhang

Der Anhang umfasst die XML-Schema File (siehe: A.1) für das benutzten Datenformat, Tabellen mit Messdaten (und deren Visualisierung) sowie eine DVD mit weiteren Anhängen.

Die DVD ist wie folgt gegliedert:

DVD/Readme: Wichtige Hinweise zu den Inhalten der DVD

DVD/Praxis: Praktische Ergebnisse

./EyeTrackingDataProcessor: Microsoft VisualStudio 2010 Projektordner – enthält Projektdateien, sowie Unterordner für die einzelnen Unterprojekte

./Dependencies: Alle im Rahmen dieser Arbeit benutzten Software-Abhängigkeiten

./binaries: Zur Ausführung benötigte Binärdateien

./plugins: Plug-Ins für Datenerfassung, Konvertierung und Weiterverarbeitung

./Doxygen: Grundlegende API-Dokumentation der entwickelten Software. Weitere spezifischere Kommentare sind im Quellcode zu finden.

DVD/Theorie: Theoretischen Ergebnisse

./Bachelorarbeit.pdf: digitale Version dieser Arbeit

./EyeTrackingDataFormatSchema1.0.xml: XML-Schema des genutzten Datenformates

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="RecordingSession" type="RS"/>
 <xsd:complexType name="RS">
  <xsd:sequence>
   <xsd:element name="BasicInformation" type="BI"/>
   <xsd:complexType name="BI">
    <xsd:sequence>
     <xsd:element name="FormatVersion" type="xsd:string"/>
     <xsd:element name="FieldUnusedSign" type="xsd:string"/>
     <xsd:element name="DataSource" type="DS"/>
     <xsd:complexType name="DS">
      <xsd:sequence>
       <xsd:element name="DataSourceManufacturer" type="xsd:string"/>
       <xsd:element name="DataSourceModel" type="xsd:string"/>
       <xsd:element name="DataSourceProductID" type="xsd:string"/>
       <xsd:element name="DataSourceFirmware" type="xsd:string"/>
       <xsd:element name="DataSourceTrackingRate" type="xsd:float"/>
      </xsd:sequence>
     </xsd:complexType>
     <xsd:element name="ScreenInformation" type="SI"/>
     <xsd:complexType name="SI">
      <xsd:sequence>
       <xsd:element name="Size-X" type="xsd:float"/>
       <xsd:element name="Size-Y" type="xsd:float"/>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:sequence>
   </xsd:complexType>
   <xsd:element name="DatasetList" type="DL"/>
   <xsd:complexType name="DL">
    <xsd:element name="Dataset" type="DS"/>
    <xsd:complexType name="DS">
     <xsd:sequence>
      <xsd:element name="timestamp" type="xsd:time"/>
      <xsd:element name="DataValidity" type="DV"/>
      <xsd:complexType name="DV">
       <xsd:sequence>
        <xsd:element name="LeftEye" type="xsd:integer"/>
        <xsd:element name="RightEye" type="xsd:integer"/>
       </xsd:sequence>
      </xsd:complexType>
      <xsd:element name="PupilDiameter" type="PD"/>
      <xsd:complexType name="PD">
       <xsd:sequence>
        <xsd:element name="LeftEye" type="xsd:integer"/>
        <xsd:element name="RightEye" type="xsd:integer"/>
       </xsd:sequence>
      </xsd:complexType>
      <xsd:element name="GazePoint2D" type="GP2D"/>
```

```
      <xsd:complexType name="GP2D">
       <xsd:sequence>
        <xsd:element name="LeftEye" type="xsd:float"/>
        <xsd:element name="RightEye" type="xsd:float"/>
       </xsd:sequence>
      </xsd:complexType>
      <xsd:element name="EyePos3D" type="EP3D"/>
      <xsd:complexType name="EP3D">
       <xsd:sequence>
        <xsd:element name="LeftEye" type="xsd:float"/>
        <xsd:element name="RightEye" type="xsd:float"/>
       </xsd:sequence>
      </xsd:complexType>
      <xsd:element name="RelEyePos3D" type="REP3D"/>
      <xsd:complexType name="REP3D">
       <xsd:sequence>
        <xsd:element name="LeftEye" type="xsd:float"/>
        <xsd:element name="RightEye" type="xsd:float"/>
       </xsd:sequence>
      </xsd:complexType>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:complexType>
  </xsd:sequence>
 </xsd:complexType>
</xsd:schema>
```

Listing A.1: XML-Datenformat als XML-Schema

Tabelle A.1.: Testdaten: Aufnahmezeit sowie Konvertierungszeit im Durchschnitt
Konfiguration: Alle Verarbeitungsschritte aktiviert, ActiveMQ Standardkonfiguration

Aufnahmezeitraum in Sekunden	Verarbeitete Datensätze	Reale Aufnahmezeit in Sekunden	Reale Konvertierungszeit in Sekunden	Von der Weiterverarbeitung empfangene Datensätze am Ende der Konvertierung	Zeit pro Datensatz in Millisekunden: Aufnahme	Zeit pro Datensatz in Millisekunden: Konvertierung
0,1	38	0,11	0,12	38	2,94	3,24
0,5	269	0,53	0,79	269	1,97	2,94
1	458	1,01	1,42	458	2,20	3,10
5	2041	5,01	8,66	1830	2,45	4,24
10	3448	10,03	15,44	2532	2,91	4,48
30	9188	30,07	43,94	4649	3,27	4,78
60	17166	60,01	77,80	6166	3,50	4,53
150	47624	150,04	308,47	14352	3,15	6,48
200	53662	200,17	540,64	20387	3,73	10,07
240	66354	240,06	1300,42	33082	3,62	19,60
300	79506	300,24	2547,07	46218	3,78	32,04
600	87836	601,16	n.A.	n.A.	6,84	n.A.
1200	98219	1200,61	n.A.	n.A.	12,22	n.A.

Tabelle A.2.: Testdaten: Aufnahmezeit sowie Konvertierungszeit im Durchschnitt
Konfiguration: Alle Verarbeitungsschritte aktiviert, ActiveMQ Topic vergrößert im Gegensatz zur Standardmessung

Aufnahmezeitraum in Sekunden	Verarbeitete Datensätze	Reale Aufnahmezeit in Sekunden	Reale Konvertierungszeit in Sekunden	Von der Weiterverarbeitung empfangene Datensätze am Ende der Konvertierung	Zeit pro Datensatz in Millisekunden: Aufnahme	Zeit pro Datensatz in Millisekunden: Konvertierung
0,1	52	0,12	0,15	52	2,25	2,81
0,5	242	0,51	0,70	242	2,12	2,90
1	439	1,03	1,43	439	2,34	3,25
5	1889	5,01	7,62	1733	2,65	4,03
10	3623	10,01	16,63	2870	2,76	4,59
30	9261	30,02	44,39	4886	3,24	4,79
60	17581	60,04	93,62	7631	3,41	5,32
150	45967	150,01	235,10	12176	3,26	5,11
200	60975	200,08	316,29	14405	3,28	5,19
240	82503	240,02	431,49	17022	2,91	5,23
300	91755	300,03	471,55	17475	3,27	5,14
600	233503	600,01	1068,43	25185	2,57	4,58
1200	514338	1200,10	2146,91	36028	2,33	4,17

Tabelle A.3.: Testdaten: Aufnahmezeit sowie Konvertierungszeit im Durchschnitt
Konfiguration: Heatmapping deaktiviert, ActiveMQ Topic vergrößert im Gegensatz zur Standardmessung

Aufnahmezeitraum in Sekunden	Verarbeitete Datensätze	Reale Aufnahmezeit in Sekunden	Reale Konvertierungszeit in Sekunden	Zeit pro Datensatz in Millisekunden: Aufnahme	Zeit pro Datensatz in Millisekunden: Konvertierung
0,1	51	0,11	0,13	2,08	2,52
0,5	231	0,51	0,62	2,22	2,69
1	468	1,01	1,20	2,17	2,57
5	2855	5,03	6,91	2,76	2,42
10	5871	10,02	13,84	2,71	2,36
30	17702	30,01	42,10	1,70	2,38
60	35108	60,01	82,33	1,71	2,35
150	100404	150,01	209,32	1,49	2,08
200	136855	200,01	282,97	1,46	2,07
240	168154	240,01	341,17	1,43	2,03
300	211067	300,01	411,40	1,42	1,95
600	440423	600,02	877,37	1,36	1,99

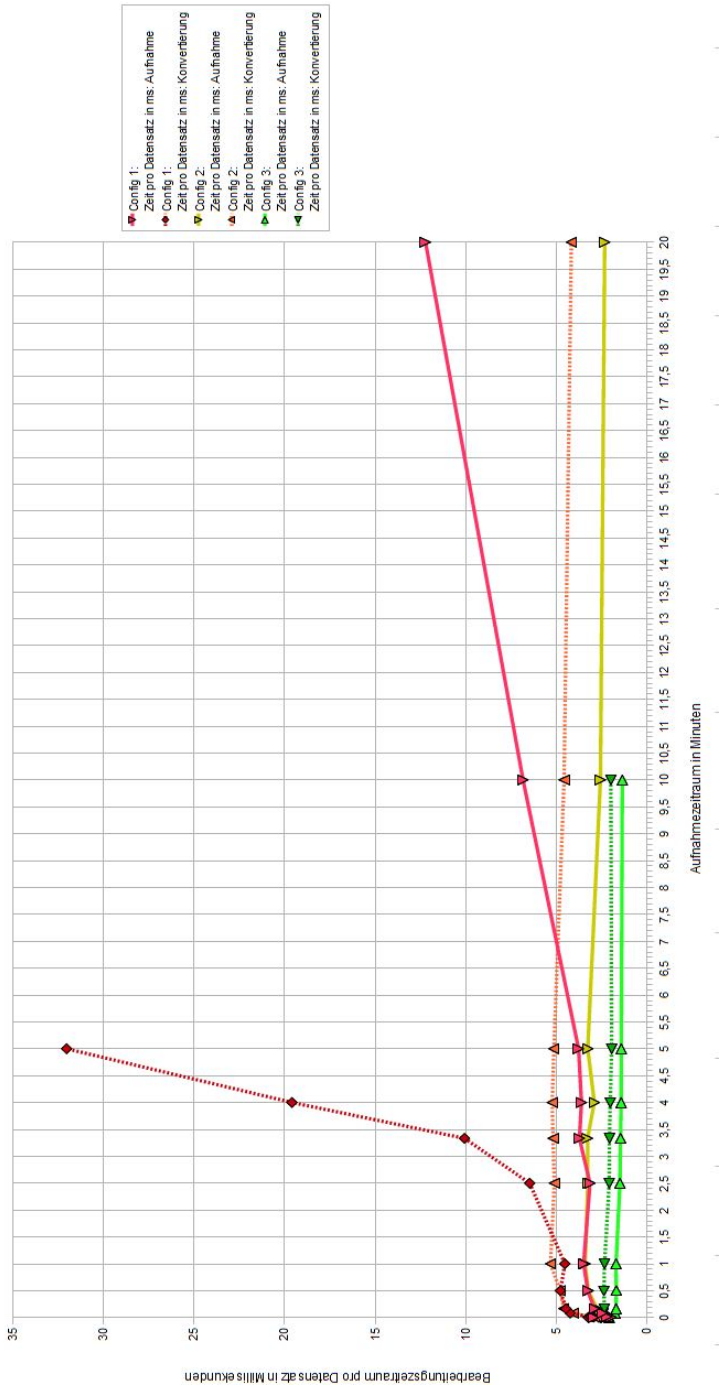

Abbildung A.1.: Visualisierung der Ergebnisse der Testdurchläufe: Anstieg des Zeitverbrauchs pro Datensatz für Aufnahme und Konvertierung

Literatur

[APR] Apache Software Foundation, Hrsg. *Apache Portable Runtime*. Englisch. URL: http://apr.apache.org/ (besucht am 14. Aug. 2012).

[ActiveMQ-CPP] Apache Software Foundation, Hrsg. *Apache ActiveMQ-CPP*. Englisch. URL: http://activemq.apache.org/cms/index.html (besucht am 11. Juni 2012).

[ActiveMQ] Apache Software Foundation, Hrsg. *Apache ActiveMQ*. Englisch. URL: http://activemq.apache.org/ (besucht am 16. Mai 2012).

[Batra/Bishu07] S. Batra und R. Bishu. „Web Usability and Evaluation: Issues and Concerns“. Englisch. In: *Usability and Internationalization. HCI and Culture*. Hrsg. von Nuray Aykin. Bd. 4559. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007, S. 243–249. ISBN: 978-3-540-73286-0. URL: http://dx.doi.org/10.1007/978-3-540-73287-7_30.

[Boost] *Boost C++ Libraries*. Englisch. URL: http://www.boost.org/ (besucht am 14. Aug. 2012).

[Duchowski07] Andrew T. Duchowski. *Eye Tracking Methodology. Theory and Practice*. Englisch. 2. Aufl. London: Springer, 2007. ISBN: 978-1-84628-608-7.

[GoF94] Erich Gamma u. a. *Design Patterns. Elements of Reusable Object-Oriented Software*. Englisch. 1. Aufl. Addison-Wesley Professional Computing Series. Boston: Addison Wesley Professional, 31. 10. 1994. ISBN: 978-0201633610.

[Hennessey/Duchowski10] Craig Hennessey und Andrew T. Duchowski. „An Open Source Eye-gaze Interface: Expanding the Adoption of Eye-ganze in Everyday- Applications“. Englisch. In: ETRA '10 Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications. Association for Computing Machinery. New York: ACM, 2010, S. 81–84. ISBN: 978-1-60558-994-7. URL: http://doi.acm.org/10.1145/1743666.1743686 (besucht am 28. Feb. 2012).

[IEEE Std 610.12-1990] Institute of Electrical and Electronics Engineers, Hrsg. *IEEE 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology*. Englisch. 1990. ISBN: 1-55937467-X.

[ISO 9241-1] International Organization for Standardization, Hrsg. *ISO EN 9241-1. Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten - Teil 1: Allgemeine Einführung.*

[Mihaescu] George Mihaescu. *A simple plug-in architecture pattern for C++ applications on Win32*. Englisch. URL: http://www.abstraction.net/ViewArticle.aspx?articleID=67 (besucht am 5. Apr. 2012).

[Phan11] Truong Vinh Phan. „Development of a custom application for the Tobii Eye Tracker“. Englisch. Bachelorarbeit. Hamburg: Hochschule für Angewandte Wissenschaften Hamburg, 24. 08. 2011. URL: http://opus.haw-hamburg.de/volltexte/2011/1387/ (besucht am 2. März 2012).

[Qian08] Kai Qian u. a. *Software Architecture and Design Illuminated*. Englisch. Jones and Bartlett Illuminated Series. Jones und Bartlett Publishers, 2008. ISBN: 978-0-7637-5420-4.

[R] R Foundation, Hrsg. *R-Project*. Englisch. URL: http://www.r-project.org/ (besucht am 11. Sep. 2012).

[Tietz11] Alexej Tietz. „Entwurf und Realisierung einer Anwendungssuite für einen Eye-Tracker“. Bachelorarbeit. Hamburg: Hochschule für Angewandte Wissenschaften Hamburg, 27. 06. 2011. URL: http://opus.haw-hamburg.de/volltexte/2011/1274/ (besucht am 3. März 2012).

[Tobii-a] Tobii Technology, Hrsg. *Tobii Eye Tracking. An introduction to eye tracking and Tobii Eye Trackers*. Englisch. Version January 27, 2010. URL: http://www.tobii.com/Global/Analysis/Training/WhitePapers/Tobii_EyeTracking_Introduction_WhitePaper.pdf?epslanguage=en (besucht am 14. Mai 2012).

[Tobii-b] Tobii Technology, Hrsg. *User Manual: Tobii X60 & X120 Eye Trackers*. Englisch. 3. Aufl. URL: http://www.tobii.com/Global/Analysis/Downloads/User_Manuals_and_Guides/Tobii_X60_X120_UserManual.pdf (besucht am 5. Feb. 2012).

[Tobii-c] Tobii Technology, Hrsg. *Tobii Eye Tracking. See through the eyes of the user*. Englisch. URL: http://www.tobii.com/Global/Analysis/Marketing/Brochures/SegmentBrochures/Tobii_Usability_Brochure.pdf (besucht am 14. Mai 2012).

[Tobii-d] Tobii Technology, Hrsg. *Tobii X60 & X120 Eye Trackers. Flexible Eye Trackers for studies of physical objects*. Englisch. URL: http://www.tobii.com/Global/Analysis/Marketing/Brochures/ProductBrochures/Tobii_X60_and_X120_Leaflet.pdf?epslanguage=en (besucht am 14. Mai 2012).

[Tobii-e] Tobii Technology, Hrsg. *Tobii SDK 3.0 Developers Guide*. Englisch. Version RC 1, June 14, 2011. URL: http://www.tobii.com/Global/Analysis/Downloads/User_Manuals_and_Guides/Tobii%20SDK%203.0%20Release%20Candidate%201%20Developers%20Guide.pdf (besucht am 5. Feb. 2012).

[Xalan] Apache Software Foundation, Hrsg. *The Apache Xalan Project*. Englisch. URL: http://xml.apache.org/xalan-c/ (besucht am 11. Sep. 2012).

[Xerces] Apache Software Foundation, Hrsg. *The Apache Xerces Project*. Englisch. URL: http://xerces.apache.org/index.html (besucht am 5. Apr. 2012).

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

Versicherung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 11. November 2012 Marcel Schöneberg