Fakultät Informatik, Elektrotechnik und Informationstechnik
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Fachstudie Nr. 158

Framework zur Übertragung, Sicherung und Visualisierung von Sensordaten

Stefanie Baur Steffen Hanikel Dominik Olp

Studiengang: Softwaretechnik

Prüfer: Prof. Dr. Albrecht Schmidt

Betreuer: Dipl.-Inf. Bastian Pfleging

begonnen am: 23. Mai 2012

beendet am: 16. Juli 2012

CR-Klassifikation: J.2, J.3, H.5.1

Kurzfassung

Diese Arbeit beschreibt zwei Tools, die zur Unterstützung der Forschung im Bereich der Mensch-Computer-Interaktion entwickelt wurden. Zum einen ein Framework, das das einfache Versenden von Datenpaketen über ein Netzwerk erlaubt, und zum anderen ein Visualisierungstool, das über das Netzwerk eingehende Daten anzeigt und speichert.

Neben einer Beschreibung der Funktionen und dem Entwurf der beiden Systeme werden auch die entsprechenden Anforderungen sowie die Vision, die der Implementierung zugrunde liegt, dargestellt.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Die Mensch-Computer-Interaktion (MCI) beschäftigt sich mit der Entwicklung und Gestaltung möglichst intuitiv zu bedienender Schnittstellen zwischen Mensch und Computer. Häufig wird bei der Forschung im Bereich der MCI ein Proband mittels verschiedener Geräte beobachtet, um herauszufinden, auf welche Weise er eine bestimmte Aufgabe löst. Um die dabei entstehenden Messdaten verschiedener Geräte leicht auswerten zu können, wird ein Tool benötigt, das die Daten einlesen und darstellen kann.

Besonderes Augenmerk liegt hierbei auf der Visualisierung der eingehenden Daten. Neben generischen Darstellungen für alle eingehenden Daten soll es die Möglichkeit geben, spezialisierte Visualisierungen für spezifische Geräte und Anwendungsfälle zur Verfügung zu stellen. Dies soll helfen, die Daten in einer Form zu abstrahieren, dass ihre zeitlichen und inhaltlichen Zusammenhänge für einen menschlichen Beobachter leicht visuell erfassbar sind. Das Ziel ist es, auf diesem Weg auch unerwartete Zusammenhänge leichter erkennbar zu machen.

Des Weiteren wird ein neues Framework für die Kommunikation der Geräte über das Netzwerk benötigt, mit dessen Hilfe auf einfache Weise neue Geräte eingebunden werden können. Eine sehr ähnliche Arbeit existiert bereits unter dem Namen „EI-Toolkit“ (siehe Kapitel 2.1 „Das ursprüngliche EI-Toolkit“) und wird auch schon verwendet. Das neu zu entwickelnde Framework ist im Grunde eine Neuimplementierung des vorhandenen EI-Toolkits, wobei die benötigten Interfaces so klein und einfach wie möglich zu halten sind. Da die Hauptaufgabe des Frameworks weiterhin dieselbe bleibt, haben wir uns dazu entschieden, den Namen „EI-Toolkit“ beizubehalten.

1.1.1 Gliederung der Arbeit

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 1 enthält die Einleitung und den Überblick über die Gliederung.

Kapitel 2 stellt benötigte Grundlagen zur Weiterentwicklung der Anwendungen sowie das ursprüngliche EI-Toolkit vor.

Kapitel 3 beschreibt Entwurf und Funktionalität des neuen EI-Toolkits.

Kapitel 4 beschreibt Entwurf, Funktionalität und Vision für Octopus.

Jedes der Kapitel 3 und 4 enthält Unterkapitel zu Konzept, Umsetzung und Evaluierung sowie einen Ausblick auf die mögliche Weiterentwicklung der jeweiligen Anwendung.

Kapitel 5 beschließt die Arbeit mit einer Zusammenfassung der Ergebnisse.

2 Hintergrund und verwandte Arbeiten

Dieses Kapitel beschreibt die Grundlagen, die nötig sind, um Octopus beziehungsweise das EI-Toolkit weiter zu bearbeiten und stellt die verwandten Arbeiten vor, die es bereits auf diesem Gebiet gibt.

2.1 Das ursprüngliche EI-Toolkit

Das ursprüngliche EI-Toolkit entstand während einer Dissertation von Paul Holleis [Hol09, S. 84ff]. Diese Anwendung stellt eine Plattform zur Kommunikation zwischen unterschiedlichen Geräten zur Verfügung. Sie erlaubt die Anbindung der Geräte in verschieden Programmiersprachen und unterstützt neben UDP noch einige weitere Kommunikationsprotokolle.

Unsere Recherche zur Verwendung des bestehenden EI-Toolkits zeigte, dass das Toolkit im derzeitigen Stand relativ komplex ist, da Artefakte aus unterschiedlichen Projekten und Überarbeitungsphasen im Code vorhanden sind. Die allgemeine Struktur des Code sowie die Kapselung der Module ist durch die mehrfache Überarbeitung beeinträchtigt. Weiterhin ist das Nachrichtenformat nur sehr lose definiert.

Der Aufwand, ein Stück Software zu schreiben, das mittels des EI-Toolkits Pakete versendet, war schon vor der Überarbeitung gering. Dies war besonders den bereits existierenden Beispielanwendungen zu verdanken. Die Benutzbarkeit des ursprünglichen EI-Toolkits aus Benutzersicht war also relativ gut. Aus Wartungssicht jedoch ergab sich ein anderes Bild: So hätte beispielsweise das Einarbeiten eines strikteren Nachrichtenformats einen hohen Aufwand gefordert.

Auf dieser Grundlage wurde entschieden, das Erstellen und Versenden von UDP-Paketen, die zentrale Funktion des EI-Toolkits, auszulagern und ein separates, ähnliches Framework mit einem strikteren Nachrichtenlayout zu erstellen. Genaueres hierzu unter 3.2.2 „Nachrichten-Struktur“.

2.2 Grundlagen zu dieser Arbeit

Zur weiteren Bearbeitung des EI-Toolkits sind neben fundierten C++ Kenntnissen Grundkenntnisse in der Netzwerkprogrammierung und der asynchronen Programmierung entschei-

dend. Informationen zum Entwurf des EI-Toolkits sind unter 3.2.1 „Software-Architektur“ zu finden.

Um Octopus bearbeiten oder erweitern zu können, sind sowohl Kenntnisse in C++ wie auch in Qt [qtD] Voraussetzung. Tutorials zu einfachen Qt Anwendungen gibt es online [qtT]. Informationen zum Entwurf von Octopus finden sich unter 4.2.1 „Software-Architektur“.

Weitere Links zu den für die Entwicklung von Octopus und dem EI-Toolkit verwendeten Tools und Bibliotheken sind in den Kapiteln 3.2.3 „Eingesetzte Tools/Libraries“ beziehungsweise 4.2.2 „Eingesetzte Tools/Libraries“ zu finden.

3 EI-Toolkit

Dieses Kapitel beschreibt die Überarbeitung des EI-Toolkits. Dazu gehören die neuen Anforderungen sowie deren Umsetzung.

3.1 Konzept

Das EI-Toolkit dient der Entwicklung von lose gekoppelten Prototypen für Pervasive-Computing-Anwendungen. Dafür bietet es eine Plattform für die Kommunikation von Geräten untereinander.

Nachfolgend die Anforderungen an das bestehende EI-Toolkit:

- Es soll keine Einschränkungen für die benütze Software und Hardware geben.
- Es soll einfach sein, dass Toolkit in eigene Anwendungen einzubinden.
- Es soll einfach sein, Anwendungen zu erweitern und zu debuggen.
- Es soll einfach sein, das Toolkit in bestehende Projekte zu integrieren.

3.1.1 Anforderungen

Daraus ergeben sich folgende konkrete Anforderungen, die auch für das neue EI-Toolkit gelten:

- Das Application Programming Interface (API) soll so einfach wie möglich sein.
- Die folgenden Sprachen müssen unterstützt werden: C++, C# und Java.
- Das Toolkit muss weiche Echtzeitanforderungen erfüllen. D.h. Nachrichten sollen mit einer möglichst geringen Latenz zugestellt werden, es besteht aber keine Garantie für eine Höchstdauer.
- Das Toolkit muss auf Performanz ausgelegt sein, d.h. Nachrichten sollen auch mit hohem Durchsatz noch schnell zugestellt werden. Das Toolkit selbst darf daher nur geringen Zusatzaufwand generieren.
- Das Toolkit muss unabhängig von einem bestimmten Transportmedium sein, d.h. es soll sowohl über LAN, WLAN, Bluetooth etc. funktionieren können.

- Das Toolkit muss erweiterbar sein, d.h. es soll einfach sein, neue Nachrichtentypen zu integrieren und neue Konzepte zu implementieren.

Außerdem wurden an das neue EI-Toolkit folgende zusätzliche Anforderungen gestellt:

- Es soll eine automatische Erkennung aller Sender möglich sein.
- Die Geräte sollen Metainformationen über sich selbst zur Verfügung stellen können.

3.2 Umsetzung

Nach einer ausführlichen Evaluation des ursprünglichen Toolkits kamen wir zu dem Schluss, dass die neu geforderte Funktionalität nicht ohne weiteres integriert werden kann. Ausschlaggebend dafür waren vor allem die drei unterschiedlichen Implementierungen in verschiedenen Programmiersprachen sowie das nicht vollständig standardisierte Datenformat des alten Toolkits.

3.2.1 Software-Architektur

Abb. 3.1 zeigt einen Überblick über die Architektur des Toolkits. Diese ist in mehrere Ebenen untergliedert.

Auf der obersten Ebene stehen Sender und Receiver. Sie dienen dazu, dem Benutzer des EI-Toolkits eine hohe Abstraktion für das Senden und Empfangen von Paketen zu liefern.

Darunter liegt die Präsentationsschicht. Sie dient der Umwandlung von EI-Toolkit-spezifischen Datenstrukturen in ein Paket, das verschickt oder gespeichert werden kann.

Auf der untersten Architekturebene schließlich befindet sich die Transportschicht, die für die Nachrichtenübermittlung auf Hardwareebene steht. Sie stellt Funktionen zur simplen Übermittlung von Daten auf Byte-Basis bereit.

Sender und Receiver verwenden die Präsentations- und Transportschicht für die eigentliche Kommunikation. Die Implementierungen dieser beiden Schichten können per Dependency-Injection ausgetauscht werden, sodass der Benutzer sehr leicht Anpassungen vornehmen und beispielsweise ein anderes Kommunikationsprotokoll verwenden kann.

So ergibt sich nach außen hin eine sehr kleine API, wodurch wiederum der Aufwand für die Einbindung des Toolkit in die eigene Anwendung sehr gering ist.

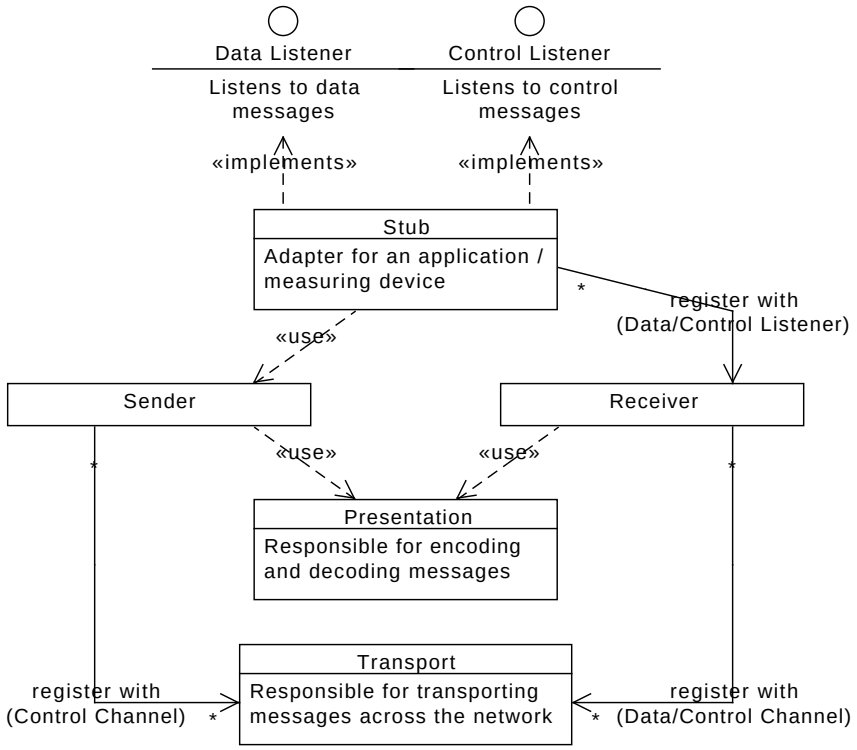

Abbildung 3.1: Entwurf EI-Toolkit

3.2.2 Nachrichten-Struktur

Das Nachrichtenformat beinhaltet unter anderem Angaben zum Sender der Nachricht sowie die aktuellen Sensordaten des Geräts. Dies erleichtert sowohl das Anlegen von Datenreihen, wie beispielsweise zur Anzeige in Octopus benötigt (siehe Kapitel 4.1.2 „Datenmodell“), als auch das Speichern der empfangenen Daten.

Das Nachrichtenformat enthält aktuell nur die nötigsten Informationen, kann bei Bedarf jedoch leicht erweitert werden, um zusätzlichen Anforderungen zu genügen.

3.2.3 Eingesetzte Tools/Libraries

Um das Toolkit nicht in mehreren Sprachen implementieren zu müssen und dadurch den Wartungsaufwand in die Höhe zu treiben, ist die eigentliche Bibliothek vollständig in C++ geschrieben. Die Schnittstellen für anderen Sprachen sind nur dünne Wrapper (Bindings), die auf die C++ Bibliothek zugreifen. Dieser Ansatz hat gleich mehrere Vorteile:

1. Die Bibliothek kann in C++ implementiert werden. C++ erlaubt eine performante, portable und trotzdem abstrakte Implementierung zu gleich.

2. Bindings lassen sich automatisch mittels SWIG [SWI] generieren, was den Wartungsaufwand für die Bindings auf ein Minimum reduziert.
3. Es lassen sich ohne viel Aufwand Bindings für weitere Sprachen erzeugen.
4. Bei Änderungen an der Bibliothek werden auch die Bindings neu generiert und sind damit automatisch immer auf dem neusten Stand.

Da die C++-Standardbibliothek keine Schnittstellen für Netzwerkzugriffe und Zeitmessung enthält, wird als Ergänzung die Boost-Bibliothek [booa] verwendet. Sie eignet sich dafür besonders gut, da sie im Gegensatz zu den meisten anderen Bibliotheken selbst keine weiteren Abhängigkeiten mit sich bringt, portabel und effizient ist, von erfahrenen Entwicklern betreut und ständig gewartet wird.

Für die Dokumentation der API wird Doxygen [dox] eingesetzt. Dieses Werkzeug erlaubt das Erstellen einer HTML-basierten Referenz und Anleitung für die API durch direkt im Quellcode eingefügte Annotationen. Das hat den Vorteil, dass die Dokumentation nicht extern erstellt werden muss und dadurch leichter aktuell zu halten ist.

Durch die hohen Anforderungen an Portabilität und Verwendbarkeit in verschiedenen Sprachen wurde kein systemspezifisches Build-Werzeug eingesetzt, sondern stattdessen das portable CMake [cma] verwendet. Dieses erlaubt das automatische Finden externer Bibliotheken und die automatische Generierung systemspezifischer Builddateien auf allen wichtigen Plattformen. So können mit einem Aufruf alle Bibliotheken, die Dokumentation, sowie die Bindings automatisch generiert und ausgeliefert werden.

Auf der Transportschicht ist als Standardprotokoll UDP Broadcast implementiert, da das auch im ursprünglichen EI-Toolkit die am häufigsten eingesetzte Protokollvariante war. Auf der Präsentationsschicht wird JSON [jso] verwendet, da es einfach zu verwenden und leicht zu erweitern ist. Weitere Transport- oder Präsentationsvarianten können einfach durch Implementierung des entsprechenden Interfaces integriert werden.

3.3 Evaluierung

Eine Messung mit 4 PCs ergab eine durchschnittliche Sendegeschwindigkeit von $\approx$ 40k Nachrichten pro Sekunde und eine durchschnittliche Empfangsgeschwindigkeit von $\approx$ 120k Nachrichten pro Sekunde. Dabei ist der größte Flaschenhals der IP-Stack des Betriebssystems. Dieser ließe sich nur durch ein Versenden der Nachrichten in größeren, zusammenhängenden Paketen beheben, da hier der Aufwand pro Systemcall ausschlaggebend ist.

Einschränkungen gibt es vor allem beim Senden über WLAN. Dort ist die Broadcastbandbreite auf 1 bis 2 Mbit/s beschränkt; somit kann nur ein geringer Durchsatz von < 1000 Nachrichten pro Sekunde erreicht werden.

3.4 Ausblick

Verbesserungswürdig sind unter anderem die generierten Java Bindings für maps. Über diese lässt sich nämlich in der aktuellen Version nicht iterieren. Dazu müssten allerdings die von SWIG eingesetzten Templates stark überarbeitet werden.

Zudem unterstützen die Bindings bisher keine Exceptions. Stattdessen werden alle Fehler intern im EI-Toolkit abgefangen, können aber nicht an den Aufrufer weitergegeben werden.

Weitere Ideen umfassen das Erstellen weiterer Bindings für andere Sprachen wie z.b. Python [pyt] oder Ruby [rub]. Durch eine Verwendungsmöglichkeit des EI-Toolkit in Skriptsprachen ließe sich die Iterationszeit bei der Erstellung von Prototypen stark verkürzen.

Die Implementierung weiterer Transport- und Präsentationsschichten liegt ebenfalls nahe. Es existiert bereits eine rudimentäre Implementierung für das hochoptimierte Serialisierungprotokoll Protobuf [pro], mit dem weitere Geschwindigkeitssteigerungen möglich würden.

Der Einsatz eines speziellen Speicherallokierers könnte ebenfalls auch deutliche Geschwindigkeitssteigerungen bringen, da beim Versenden und Empfangen von Nachrichten viele kleine Objekte auf dem Heap angelegt und wieder freigegeben werden müssen. Darauf ist der Standardallokierer nicht vollständig optimiert.

Da es Pläne gibt, das EI-Toolkit auch für Bilddaten zu verwenden, läge außerdem die Erweiterung der Nachrichten um ein Feld für binäre Daten nahe, um diese Daten effizient versenden und empfangen zu können.

Eine größere Erweiterung wäre die Bereitstellung einer Konfigurationsmöglichkeit für angeschlossene Geräte über eine einheitliche Schnittstelle.

4 Octopus

Octopus ist ein Visualisierungstool, das über das Netzwerk eingehende Daten aufzeichnet und mit Hilfe verschiedener Visualisierungen anzeigt. Dazu identifiziert Octopus zunächst alle erreichbaren Sender sowie die von ihnen gesendeten Daten und bietet dem Benutzer dann die Möglichkeit, sich einzelne Datenreihen anzeigen zu lassen. Aufgezeichnet wird in erster Linie in eine Datenbank, es besteht aber auch die Möglichkeit, die aufgezeichneten Daten als CSV-Datei exportieren zu lassen.

4.1 Konzept

4.1.1 Anforderungen

Die Anforderungen an Octopus beschränkten sich auf drei Bereiche. Zum einen sollte eine leichte Benutzbarkeit gewährleistet sein, um neuen Benutzern einen unkomplizierten Einstieg bereitzustellen. Dies ist insbesondere dahingehend wichtig, dass Octopus in erster Linie als Visualisierung genutzt wird, um Daten hinsichtlich einer Hypothese effizient auswertbar zu machen.

Des Weiteren sollte eine Visualisierungsmethode gewählt werden, die intuitiv erfassbar ist. Bei interpolierbaren Daten haben wir uns daher für einfache Graphen entschieden, die einzelne Datenpunkte mit einer Linie verbinden. Diese Art der Visualisierung ist allgemein bekannt, leicht zu lesen und für eine Vielzahl verschiedener Daten beziehungsweise Sender anwendbar. Diskrete Ereignisse (zum Beispiel Tastatureingaben) werden als beschriftete Impulse dargestellt.

Eine weitere wichtige Anforderung war die einfache Erweiterbarkeit, da spätere Studien- beziehungsweise Diplomarbeiten Octopus unter Umständen um neue Funktionalitäten ergänzen müssen, um ihren eigenen Anforderungen gerecht zu werden.

4.1.2 Datenmodell

Nachdem Octopus beim Start alle vorhandenen Sender identifiziert, werden je nach Anzahl und Typ der gesendeten Daten für jeden Sender entsprechende Datenreihen angelegt. Sendet beispielsweise eine Kinect x-, y- und z-Koordinaten für jedes erkannte Gelenk, werden pro Gelenk drei Datenreihen angelegt, die Koordinaten (Zahlen) enthalten können. Für diskrete Ereignisse hingegen werden Datenreihen verwendet, die Text speichern.

Neue Datenreihen können auch während dem laufenden Betrieb der Anwendung angelegt werden. Kommen Datenpakete von einem neuen Sender an, fügt Octopus diese automatisch als neue Datenreihe hinzu.

4.1.3 Visualisierung

Die Anzeige der Datenreihen erfolgt innerhalb eines oder mehrerer Plots. Jeder Plot kann beliebig viele Datenreihen anzeigen, wobei jede Datenreihe als eigener Graphen dargestellt wird. Jeder Plot ist ein einer Spur eingebettet, welche die Bearbeitungsoptionen für die Graphen bereitstellt. Darüber kann die Art der Skala (linear oder logarithmisch) verändert werden, auf die die Graphen abgebildet werden, und die Zeitstempel eines Graphen können um beliebige Werte korrigiert werden, beispielsweise um Latenzen auszugleichen.

Zusätzlich zu dieser generischen Visualisierung sind als Erweiterung auch spezielle, auf den Sender zugeschnittene Visualisierungen denkbar (siehe Kapitel 4.1.5 „Vision“).

4.1.4 Realisierter Funktionsumfang

Aufgrund der beschränkten Zeit, die für eine Fachstudie zur Verfügung steht, konnten wir nicht den gesamten aus unserer Sicht wünschenswerten Funktionsumfang implementieren. Daher haben wir uns auf die unserer Meinung nach für ein effektives Arbeiten unentbehrlichen Funktionen beschränkt. Das Ergebnis ist in Abb. 4.1 zu sehen.

Die implementierte Funktionalität umfasst:

Generische Darstellung Alle eingehenden Daten, die entweder als Zahlen oder Text vorliegen, können mittels der Standardvisualisierung als Graphen angezeigt werden.

Skalenart Die Art der Skala, auf die die Graphen abgebildet werden, kann auf linear oder logarithmisch eingestellt werden.

Zoom Da verschiedene Sender auch unterschiedlich schnell senden, gibt es die Möglichkeit den Detailgrad der Ansicht zu erhöhen oder zu verringern. So sind auch bei Sendern mit hohen Frequenzen noch Details erkennbar. Die Zoom-Stufe kann durch Buttons in der Toolbar oder stufenlos per Mausrad angepasst werden.

Markierung Für die Auswertung eines Bereichs der aufgenommenen Daten gibt es die Möglichkeit, einen Abschnitt zu markieren. Der Inhalt der Markierung kann entweder exportiert, oder der aktuell sichtbare Bereich auf die Markierung reduziert werden.

Speichern und Laden Gespeichert wird neben der Datenbank unter anderem die Anzahl und Reihenfolge der angezeigten Spuren sowie die ihnen zugeordneten Graphen. Weitere Einstellungen, die die Graphen betreffen, wie beispielsweise die Skalenart (logarithmisch oder linear) oder deren zeitliche Korrektur einzelner Datenreihen werden ebenfalls gespeichert. Weiterhin werden auch der zuletzt angezeigte Bereich und die

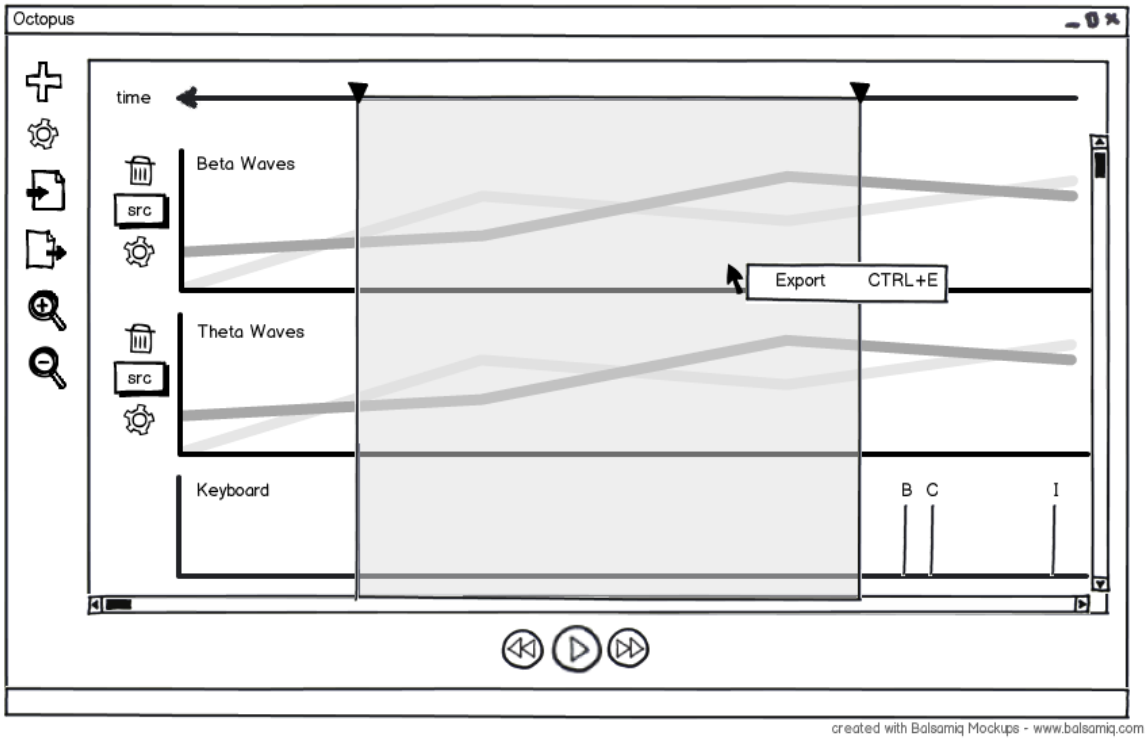

Abbildung 4.1: Benutzeroberfläche mit markiertem Bereich

Position des Cursors gespeichert. Alle diese oberflächenspezifischen Einstellungen werden beim Laden einer Aufnahme aus einer Projektdatei wieder hergestellt.

Export Beim Export werden ausschließlich die Daten der ausgewählten Datenreihen exportiert, nicht jedoch keine oberflächenspezifischen Einstellungen. Die Exportfunktionalität ist erweiterbar gehalten, sodass zukünftige Nutzer leicht eigene Exportformate einführen können.

Separate Aufnahme Während Live-Daten empfangen werden, kann eine separate Aufnahme gestartet werden. Diese speichert alle während der Aufnahme eingehenden Daten sowie die aktuellen Einstellungen bezüglich der Anzeige zusätzlich als separate Aufnahme.

Abspielen Beim Abspielen werden die Daten genauso wie beim Empfangen mit den gewählten Visualisierungsmethoden angezeigt. Kontrolliert wird das Abspielen durch Versetzen des Cursors; dessen Position gibt an, an welchem Zeitstempel man sich aktuell befindet. Entsprechend dem Fortschritt beim Abspielen wird auch der Cursor über den Plots automatisch versetzt. Zur einfacheren Handhabung kann die

Geschwindigkeit, in der abgespielt wird, verändert werden. Alle Visualisierungen passen sich dann entsprechend an.

Neuen Daten folgen Während Daten empfangen werden, hat der Benutzer die Möglichkeit, die neuen Daten stets im sichtbaren Bereich zu halten. Ist diese Funktion deaktiviert, kann der Benutzer stattdessen bereits aufgenommene Daten betrachten oder abspielen.

4.1.5 Vision

Die Vision der Anwendung, zu sehen in Abb. 4.2, enthält neben den bereits implementierten Funktionen auch alle weitere Funktionalität, die aus unserer Sicht wünschenswert wäre.

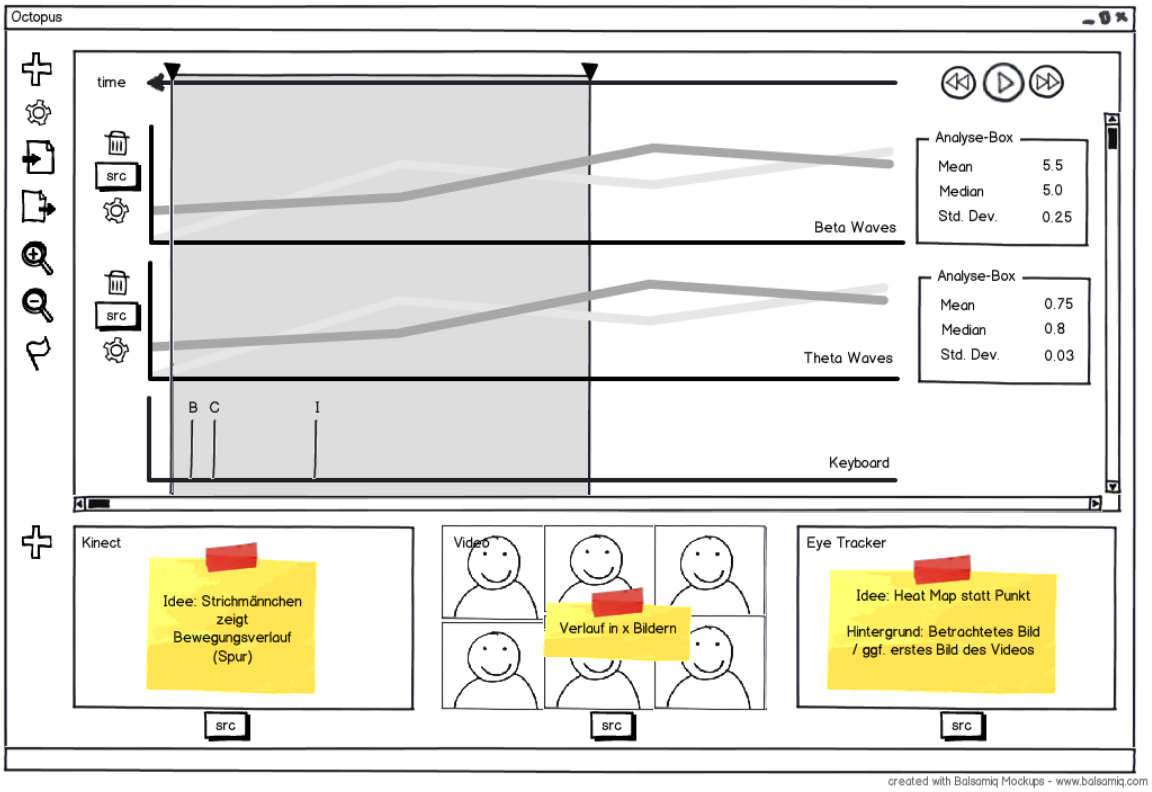

Abbildung 4.2: Vision mit markiertem Bereich

Neben der generischen Graphdarstellung sind hier auch speziell auf einzelne Sender ausgelegte Visualisierungen vorgesehen (in der Abbildung unterhalb der Spuren). Diese Art der Darstellung wird in Octopus als „Display“ bezeichnet.

Für die Kinect gibt es beispielsweise die Idee, die eingehenden Daten anhand eines Strichmännchens zu visualisieren. So könnte das Display eine vereinfachte Darstellung der Person zeigen, anhand derer die von der Kinect zum gegebenen Zeitpunkt übermittelten Gelenkpositionen auf einen Blick erkennbar sind. Existiert eine Markierung, könnte der Bewegungsverlauf über den markierten Zeitraum als Bewegungsspur angezeigt werden.

Ein anderes Beispiel wäre der Eye Tracker, bei dem es sich anbieten würde, die Koordinaten des auf einem Bildschirm fokussierten Punkts auf das Display abzubilden. Eventuell wäre eine Hinterlegung des Punkts mit dem Bildschirminhalt hilfreich, sodass der Octopus-Benutzer direkt sehen kann, worauf der Träger des Eye Trackers zum gegebenen Zeitpunkt sein Augenmerk gerichtet hat. Deshalb ist auch für die Displays eine Möglichkeit zur dynamischen Auswahl der angezeigten Datenreihen vorgesehen. Soll der Verlauf der Daten über eine Zeitspanne dargestellt werden, läge die Darstellung mit Hilfe einer Heat-Map nahe. Auch in diesem Fall könnte eine Hinterlegung mit dem Bildschirminhalt die Analyse erleichtern.

Spezialisierte Visualisierungen dieser Art sind für viele weitere Eingabegeräte denkbar.

Die Analyseboxen, von denen eine pro Spur vorgesehen ist, sind als weitere Auswertungshilfe gedacht. Sie enthalten allgemeine Informationen über die in der Spur enthaltenen Daten, so zum Beispiel den aktuellen Mittelwert, Median, oder Minima und Maxima. Diese Informationen könnten bei Bestehen einer Markierung auf die Daten des markierten Bereichs angepasst werden.

Um wichtige Vorkommnisse schon beim Empfang der Daten festhalten zu können, ist außerdem die Möglichkeit vorgesehen, sogenannte „Flags“ zu setzen. Damit sind kleine Markierungen gemeint, die mit Kommentaren versehen werden können, um bei der späteren Auswertung der Daten schnell wieder die interessanten Punkte zu finden.

Um für die Plots schnell die gewünschten Datenreihen auswählen zu können, sind in der Vision spezialisierte Dialoge vorgesehen, die die verfügbaren Daten übersichtlicher darstellen können. Dies ist besonders für Geräte interessant, die sehr viele Daten liefern. Für die Kinect beispielsweise wäre ein Dialog denkbar, der ein Strichmännchen zeigt, dessen Gelenke durch Klicken auswählbar sind (siehe Abb. 4.3 „Spezialisierter Kinect-Dialog“). So könnte man die interessante Datenreihe sehr viel schneller finden, als dies mit einer einfachen Baumstruktur möglich wäre.

Auch für ein Brain-Computer-Interface wie das Emotiv Headset [emo] wäre eine solche individualisierte Darstellung bei der Auswahl der anzuzeigenden Sensoren vorstellbar (siehe Abb. 4.4).

4.2 Umsetzung

Die Implementierung von Octopus basiert auf der Annahme, dass das EI-Toolkit eingesetzt wird, um Sensordaten über das Netzwerk zu übermitteln. Es wird davon ausgegangen, dass die Daten entweder in Form von Fließkommazahlen (z.B. Messwerten) oder Text (z.B.

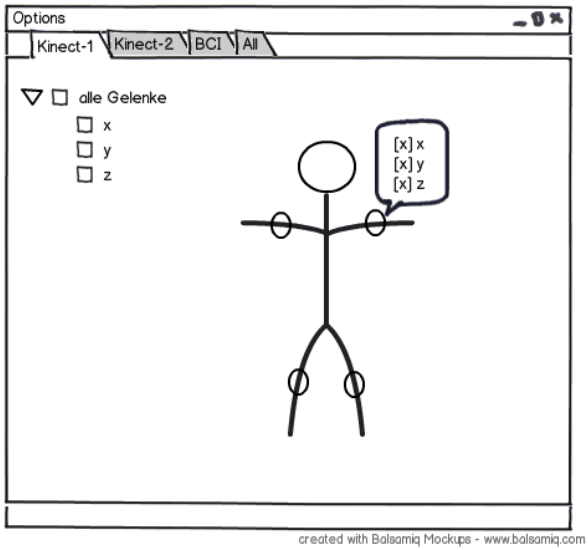

Abbildung 4.3: Spezialisierter Auswahl-Dialog für Kinect-Datenreihen.

Bezeichner für Ereignisse) vorliegen. Des Weiteren wird vorausgesetzt, dass jeder Sender über die Description-Message des EI-Toolkits eine aussagekräftige Beschreibung seiner Daten zur Verfügung stellt, auf Basis derer Octopus entscheiden kann, wie die empfangenen Daten darzustellen sind. Ansonsten ist lediglich ausreichend Speicherplatz für die Daten zur Verfügung zu stellen; dieser ist in der bisherigen Version von Octopus jedoch vernachlässigbar ($<$ 100MB). Weitere Voraussetzungen sind nicht zu erfüllen.

4.2.1 Software-Architektur

Abb. 4.5 zeigt die verschiedenen Pakete und Unterpakete, in die der Octopus-Code gegliedert ist, sowie ihre Beziehungen zueinander. Die Pakete Common und Dialogs stellen Datentypen und Klassen zur Verfügung, die an zahlreichen Stellen im Programm zum Einsatz kommen. Sie können daher von allen anderen Paketen verwendet werden. Alle weiteren Abhängigkeiten sind in Form von Pfeilen dargestellt. Die Sichtbarkeitsindikatoren „+“ und „-“ zeigen an, welche Klassen außerhalb („+“) und welche nur innerhalb („-“) ihres Paketes verwendet werden.

Wie aus Abb. 4.5 deutlich ersichtlich ist, ist das Main-Paket das zentrale Paket des Systems. Es ist für die zentrale Koordination zuständig und enthält die Implementierung des

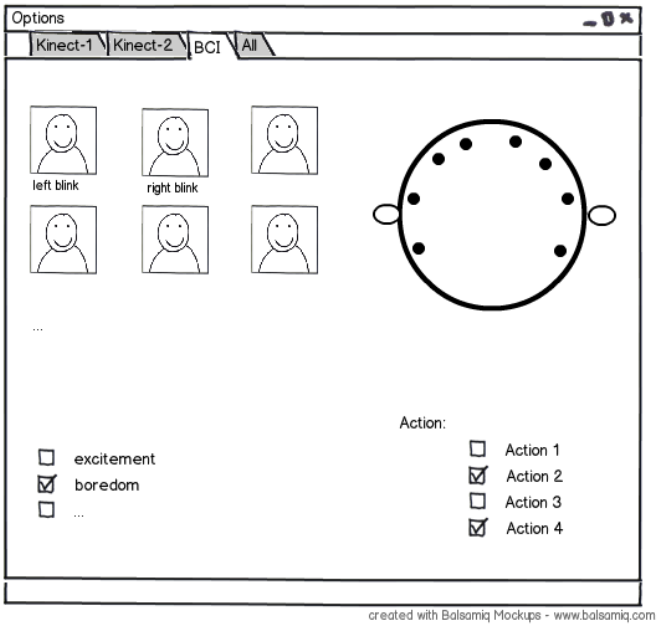

Abbildung 4.4: Spezialisierter Auswahl-Dialog für BCI-Datenreihen.

Hauptfensters. Das Network-Paket sorgt mit dem NetworkAdapter für die Übermittlung der über das Netzwerk eingehenden Daten an das DataModel-Paket, das daraufhin die Verwaltung, Speicherung und Bereitstellung der Daten übernimmt. Das MainView-Paket ist für die Darstellung der Daten verantwortlich und enthält daher auch unter anderem die Unterpakete Tracks und Displays (siehe auch Abb. 4.6). Das Export-Paket ist für den Export der gespeicherten Daten zuständig, das Record-Paket für die Erstellung separater Aufnahmen. Das Time-Paket enthält nur den TimeManager, der für die Konvertierung zwischen Zeit und Pixeln zuständig ist. Er gibt beispielsweise die Distanz zwischen den Strichen der Skalierung der Zeitleiste vor.

Das Tracks-Paket enthält die Klassen, die zur Erstellung und Verwaltung von Spuren nötig sind. Dazu gehört auch das QCustomPlot-Widget, das an dieser Stelle ins Programm eingebunden wird. Das Paket Displays ist noch leer und für die Klassen vorgesehen, die individuelle Visualisierungen für spezifische Geräte realisieren. Das Analysis-Paket ist ebenfalls noch leer. Es ist für die Klassen vorgesehen, die die in der Vision vorgestellten Analyseboxen implementieren.

Detailliertere Informationen zu einzelnen Klassen oder Methoden sind der mit doxygen generierten Quellcode-Dokumentation zu entnehmen.

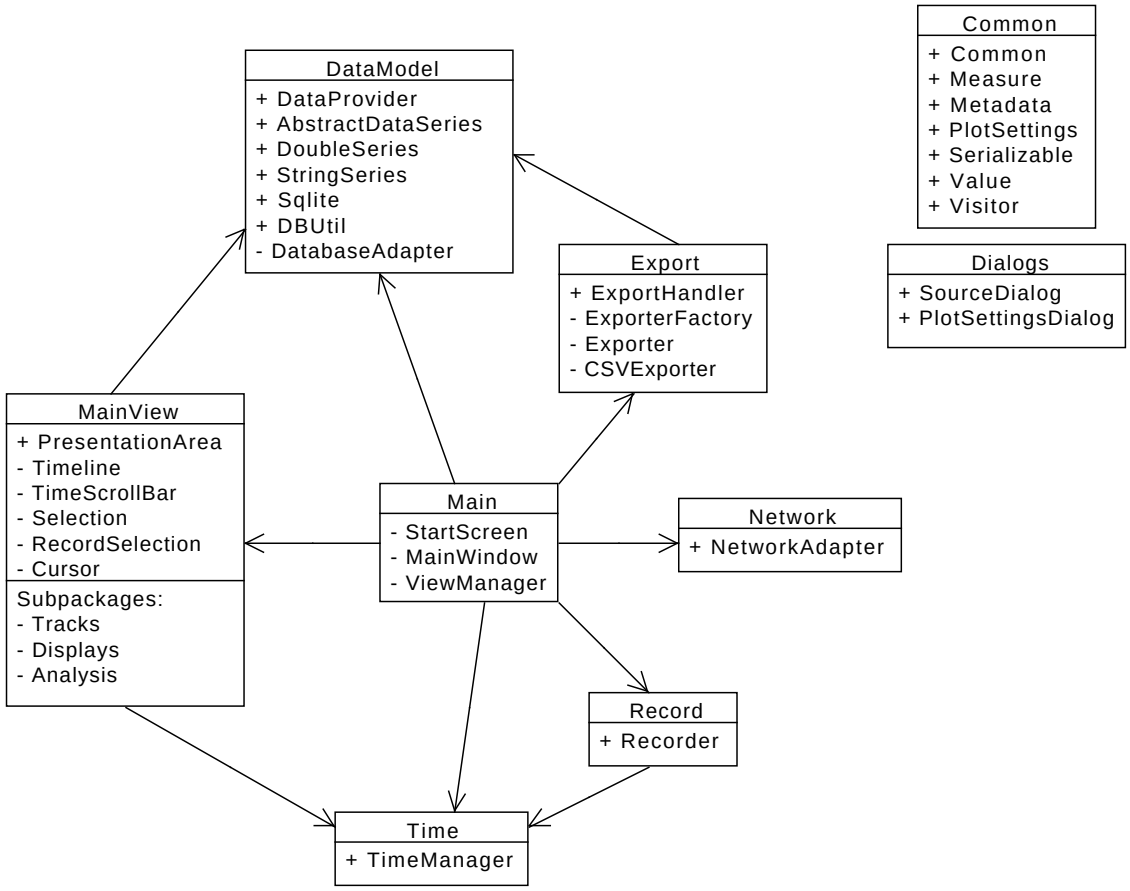

Abbildung 4.5: Paketstruktur Octopus

4.2.2 Eingesetzte Tools/Libraries

Zur Realisierung der vereinbarten Funktionalität wurden verschiedene Tools und Libraries eingesetzt.

Programmiersprache Bei der Wahl der Programmiersprache haben wir uns für C++ entschieden. Da die Anwendung die Daten mehrerer Sender gleichzeitig verarbeiten und anzeigen können muss und die Daten mit Frequenzen von bis zu 1.000 Hz eingehen, war uns die Performanz der Anwendung besonders wichtig. Gleichzeitig wollten wir eine Sprache, die ein höheres Abstraktionsniveau bietet als zum Beispiel C. C++ war daher die richtige Wahl.

Oberfläche Ein weiteres Argument für den Einsatz von C++ war die damit mögliche Verwendung der Qt-Bibliothek [qtD]. Diese bietet unter anderem ein Signals-and-Slots-System [sig] zur Kommunikation zwischen Objekten. Dadurch können Informationen über

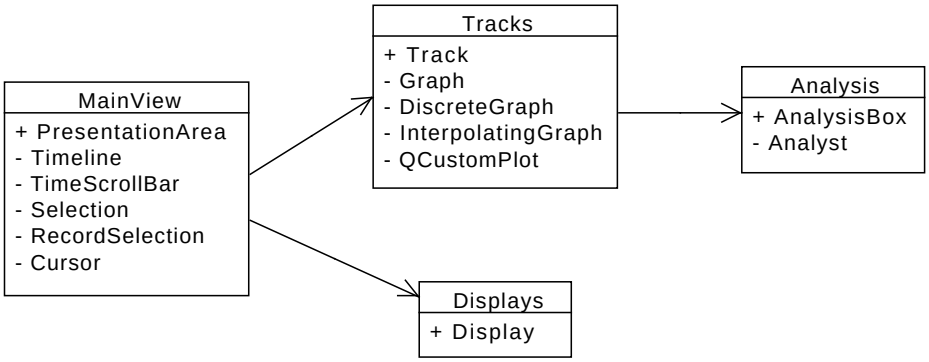

Abbildung 4.6: Innere Struktur des MainView-Pakets

Änderungen an einem Objekt leicht an andere Objekte weitergeleitet werden, ohne dass das geänderte Objekt die zu informierenden Objekte zwingend kennen muss.

Die Notwendigkeit einer solchen Informationsweitergabe tritt in GUI-Anwendungen besonders häufig auf. Das Signals-and-Slots-System hilft hier, direkte Abhängigkeiten zwischen Objekten zu verringern.

Für die Realisierung der Oberfläche mit Qt gibt es verschiedene Optionen. Standardmäßig bietet der Qt Creator [qtC], eine Entwicklungsumgebung für Qt, mit dem Qt Designer eine grafische Oberfläche zum Design von Dialog- und anderen Fenstern oder Fensterausschnitten. Diese nutzt die Standard-Widgets von Qt. Auch Teile des Verhaltens der Oberfläche, zum Beispiel die Anzeige von Tooltips oder eines Kontextmenüs lassen sich mit Hilfe des Qt Designers relativ einfach konfigurieren.

In diesem Fall kann neben den Qt-Standard-Widgets auch das Graphics View Framework [gra] zur Umsetzung der Oberfläche eingesetzt werden. Das Framework bietet Funktionalität zur Verwaltung und Interaktion mit zahlreichen 2D-Objekten. Es verwendet Binary Space Partitioning („binäre Raumpartitionierung“) für die schnelle Lokalisierung angezeigter Objekte und kann dadurch selbst Szenen mit Millionen von Objekten in Echtzeit darstellen.

Alternativ dazu bietet Qt die CSS- und JavaScript-ähnliche Sprache QML [qml] zur Implementierung von Oberflächen und deren Verhalten. Mit deren Hilfe soll eine effiziente Entkopplung von interner Anwendungslogik und Oberflächenlogik erreicht werden.

Nachdem die Erstellung eines kleineren Prototypen mit QML gezeigt hatte, dass die Funktionalität von QML unseren Anforderungen nicht genügt, haben wir die Oberfläche zunächst mit Hilfe des Graphics View Framework implementiert. Wegen der ständigen Aktualisierungen der Plots ergaben sich hieraus jedoch Performanzprobleme,

aufgrund derer wir später wieder auf die Qt-Standard-Widgets umgestellt haben. Näheres hierzu in Kapitel 4.2.3 „Verzögerungen beim Scrolling“.

Plots Zur Darstellung der Plots in den Spuren wurde das externe Widget QCustomPlot [qcp] verwendet. Dieses Widget ist speziell darauf ausgelegt, Datenreihen in Form von Graphen anzuzeigen. Jeder QCustomPlot besitzt vier Achsen (oben, unten, links, rechts), von denen jedem Graphen je eine als Abszisse und eine als Ordinate zugeordnet wird.

Man kann zwischen verschiedenen Typen von Graphen wählen, wobei das Aussehen jedes Graphen in Bezug auf Farbe, Darstellung der Datenpunkte usw. individuell konfiguriert werden kann. Außerdem bietet das Widget bereits von Haus aus einen gewissen Umfang an Interaktionsmöglichkeiten mit dem Plot, darunter das Selektieren von Graphen und das Verändern des sichtbaren Ausschnitts durch Drag & Drop.

Datenbank Die Speicherung der über das Netzwerk eingehenden Daten erfolgt in einer SQLite-Datenbank [sqlb]. SQLite ist das weltweit am stärksten verbreitete und meistgenutzte Datenbanksystem. Es unterstützt die Mehrheit der im SQL-92-Standard definierten SQL-Befehle und ist direkt integrierbar, ohne dass zusätzliche Server-Software nötig wird. Dadurch entsteht keine Abhängigkeit zu externen Modulen.

Dokumentation Für die Dokumentation der API wurde wie beim EI-Toolkit doxygen [dox] eingesetzt.

4.2.3 Probleme und Lösungen

Bei der Implementierung von Octopus sind wir auf einige Unwegbarkeiten gestoßen. Diese und unsere Lösungen dafür sind im Folgenden beschrieben.

Verzögerungen beim Scrolling

Anfangs wurde die Benutzeroberfläche unter Einsatz des Graphics View Framework implementiert. Wir hatten gehofft, dass das Zeichnen sich gegenseitig überlagernder Widgets, wie zum Beispiel der Spuren und des Cursors, dadurch besonders einfach würde. Dazu wurde jede Spur sowie Zeitleiste und Cursor in ein QGraphicsItem eingebettet, und diese QGraphicsItems in eine gemeinsame QGraphicsScene, die den gesamten Spurenbereich umfasste.

Leider stellte sich jedoch bald heraus, dass das häufig erforderliche Neuzeichnen der Spuren in diesem Zusammenhang ein Problem darstellte. Bei jedem Pixel, das der Cursor über den sichtbaren Bereich hinauswandert, muss der sichtbare Bereich der Spuren angepasst und jede Spur neu gezeichnet werden. Je mehr Spuren gleichzeitig sichtbar waren, desto stärker war die sichtbare Verzögerung - es war keine flüssige Anzeige mehr möglich. Auch beim manuellen Scrollen war dieser Effekt deutlich sichtbar. Besonders extrem war die

Verzögerung bei Spuren mit diskreten Ereignissen, da hier neben den Datenpunkten auch deren Beschriftung gezeichnet werden muss.

Zunächst sah es so aus als läge das Problem vor allem an der Performance des QCustomPlot. Profiling ergab, dass das Neuzeichnen besonders viel Zeit kostete, und dabei vor allem die Berechnung der Beschriftung für diskrete Ereignisse, die jedes Mal wieder neu durchgeführt wurde. Optimierungen an dieser Stelle erzielten jedoch nur einen geringen Effekt.

In einem weiteren Schritt wurden probehalber die Spuren aus der QGraphicsScene extrahiert. Daraufhin war eine deutliche Verbesserung zu erkennen, allerdings zeigte die Zeitleiste immer noch eine merkliche Verzögerungen. Daran ließ sich erkennen, dass die Einbettung der Widget in die QGraphicsScene einen großen Teil des Problems verursacht. Daher schafften wir die Verwaltung der Widgets durch die QGraphicsScene vollständig wieder ab. Das Graphics View Framework kam damit auch im Rest der Anwendung nicht mehr zum Einsatz.

Da die Verzögerung trotzdem noch nicht ganz verschwunden war, haben wir weitere Nachforschungen angestellt und dabei herausgefunden, dass QCustomPlot Änderungen in einem eigenen Buffer cacht, bevor er die Änderungen weitergibt und sein Widget zum Neuzeichnen auffordert. Diese Vorkehrung ist unnötig, da Qt-Widgets ein solches Buffering selbst vornehmen. Puffert das Qt-Widget selbst, kann Qt außerdem zusätzliche Optimierungen vornehmen, wie zum Beispiel ein Neuzeichnen des Widgets bei jedem Durchlauf der EventLoop nur einmal durchzuführen und nicht sichtbare Bereiche nicht neu zu zeichnen.

Nach Abschaffung des eigenen Caching durch den QCustomPlot war ein flüssiges Abspielen auch mit zahlreichen Spuren und hochfrequenten Datenreihen möglich.

Ausrichtung von Spuren und Zeitleiste

Von Haus aus bietet der QCustomPlot nur zwei Möglichkeiten für das Setzen seiner Ränder: Entweder aktiviert man den automatischen Modus, in dem anhand der Beschriftung der Achsen die optimalen Ränder berechnet und automatisch gesetzt werden, oder man stellt auf den manuellen Modus um, in dem die Breite der Ränder als Wert übergeben werden kann. Die Funktion zur Berechnung der optimalen Ränder ist von außen nicht zugänglich.

Um die Ordinaten aller Spuren und die Nulllinie der Zeitleiste korrekt auszurichten, den Plots selbst aber gleichzeitig aber so viel Platz wie möglich zu lassen, haben wir daher ein Signal eingeführt, das die intern berechneten optimalen Ränder nach außen kommuniziert, ohne sie gleich automatisch zu setzen. Stattdessen hat die Presentation Area als übergeordnetes Objekt die Aufgabe übernommen, aus den optimalen Werten aller Spuren den minimalen für alle Spuren verwendbaren Rand zu bestimmen und dann entsprechend zu setzen. Der so bestimmte Wert wird dann auch an die Zeitleiste weitergegeben, zur Positionierung der Nulllinie.

Anpassung der Ordinaten

Der QCustomPlot bietet eine Funktion zur Anpassung all seiner Achsen auf den aktuellen Wertebereich seiner Graphen. Jeder Graph bietet dieselbe Funktion, und zudem je eine Funktion zur einzelnen Anpassung seiner Ordinate oder Abszisse.

In unserem Fall war vor allem die Anpassung der Ordinate des Plots interessant. Um dies in einer Zeile tun zu können anstatt über alle Graphen des Plots iterieren zu müssen, haben wir den QCustomPlot um eine entsprechende Funktion erweitert.

4.3 Lizenzierung

Der Sourcecode für die SQLite-Bibliothek ist public domain, also ohne Copyright, und kann daher in beliebiger Weise verwendet, verändert und neu veröffentlicht werden [sqla].

Die Boost-Bibliotheken sind unter der Boost Software License Version 1.0 [boob] veröffentlicht. Wird eigener Code gemeinsam mit Boost-Code veröffentlicht, gilt die Boost Software License ausschließlich für den Boost-Code. Der eigene Code kann unter beliebigen Bedingungen veröffentlicht werden [booc, „How is the Boost license different from the GNU General Public License (GPL)?“].

Das QCustomPlot-Widget ist unter Version 3 der GNU General Public License (GPLv3) [GNUb] veröffentlicht. Die Verwendung dieses Moduls impliziert, dass auch Octopus unter der GPLv3 oder einer kompatiblen Lizenz veröffentlicht werden muss [GNUa]. Wir haben uns daher ebenfalls für die Veröffentlichung unter der GPLv3 entschieden.

4.4 Evaluierung

Eine Benutzerstudie mit Octopus konnte aufgrund der begrenzten Zeit leider nicht mehr durchgeführt werden. Gleichwohl haben wir uns während der gesamten Entwicklung bemüht, die Interaktion so intuitiv zu gestalten wie möglich.

Ansatzpunkte hierfür waren die Verwendung von so möglichst wenigen Buttons und das weitgehende Verzichten auf Menüs. Neben der Interaktion durch Buttons in der Toolbar ist auch die direkte Manipulation des Bildausschnitts per Maus möglich. So kann man sowohl zoomen als auch den sichtbaren Bereich direkt mit der Maus verschieben.

Außerdem wurde viel Wert darauf gelegt, dass in verschiedenen Zuständen des Systems (zum Beispiel während einer Live-Aufnahme oder nach dem Laden einer gespeicherten Aufnahme) immer dieselben Buttons sichtbar sind und nicht nutzbare lediglich deaktiviert werden. Dies soll den Nutzer dabei unterstützen, sich jederzeit schnell in Octopus orientieren zu können und nicht bei jedem Wechsel zwischen Live-Aufnahme und gespeicherter Aufnahme eine andere Oberfläche vorzufinden.

4.5 Ausblick

Einige Ideen für mögliche Erweiterungen wurden bereits im Kapitel 4.1.5 „Vision“ vorgestellt. Besonders viel Potenzial liegt dabei aus unserer Sicht in den Displays (vgl. Abb. 4.2). Spezialisierte Visualisierungen könnten für jedes beliebige Gerät erstellt werden und würden die intuitive Auswertbarkeit der Daten sicher drastisch verbessern.

Die Individualisierung des Dialogs zur Auswahl von Datenreihen halten wir ebenfalls für besonders hilfreich. Die Vorstellung ist hier, dass der Dialog wie bisher einen Reiter enthält, der alle verfügbaren Spuren in einer Baumstruktur anzeigt, aber zusätzlich weitere Tabs mit individualisierten Darstellungen für Sender, die viele Datenreihen liefern. Beispiele hierfür wurden in Abb. 4.3 und Abb. 4.4 vorgestellt. Eine solche Individualisierung des Dialogs würde nicht nur die Benutzererfahrung angenehmer machen, sondern könnte auch Fehler bei der Auswertung der Daten vermeiden helfen.

Darüber hinaus sind Verbesserungen der allgemeinen Benutzbarkeit möglich, so zum Beispiel die direkte Auswahl von Graphen eines Plots für die Auswahl der zu exportierenden Datenreihen. Anstatt in einem Dialog aus den verfügbaren Datenreihen die gewünschten auszuwählen, könnte man die Auswahl direkt in der Benutzeroberfläche erstellen und dadurch eine Ebene der Indirektion vermeiden.

Nach einer Erweiterung des EI-Toolkits um Konfigurationsnachrichten wie im Kapitel 3.4 „Ausblick“ beschrieben, könnte man in Octopus eine grafische Oberfläche für diese Funktion bereitstellen; beispielsweise um Sender ein- oder auszuschalten.

Octopus wurde bewusst so entworfen, dass auch zukünftige Änderungen mit relativ wenig Aufwand eingearbeitet werden können. Hierfür wurde auf eine ausführliche Dokumentation und eine gute Architektur viel Wert gelegt.

5 Abschluss

Laut Aufgabenstellung waren im Wesentlichen vier Punkte zu erfüllen:

1. Daten visualisieren und zeitliche Zuordnung ermöglichen
2. Breites Spektrum an Daten darstellbar machen
3. Gespeicherte Daten für andere Anwendungen verwendbar machen
4. Neue, leichtgewichtige EI-Toolkit Version erstellen

Alle Punkte der Aufgabenstellung konnten erfolgreich umgesetzt werden.

Eingehende Daten können mittels der Graphen dargestellt werden, wodurch die zeitliche Zuordnung verschiedener Ereignissen möglich ist.

Ein großes Spektrum verschiedener Daten kann visualisiert werden, da die anzeigbaren Datentypen (Fließkommazahlen und Text; siehe hierzu auch Kapitel 4.2 „Umsetzung“) sowohl für eine große Anzahl Messwerte als auch für diskrete Ereignisse gut einsetzbar sind.

Eingehende Daten werden einerseits in einer SQL-Datenbank gespeichert (siehe Kapitel 4.2.2 „Eingesetzte Tools/Libraries“, Stichpunkt „Datenbank“), andererseits können beliebige aufgezeichnete Daten exportiert werden. Derzeit wird nur der Export nach CSV angeboten, aber die Export-Schnittstelle ist sehr leicht erweiterbar, sodass bei Bedarf schnell weitere Exportformate hinzugefügt werden können.

Das EI-Toolkit wurde neu entwickelt. Alle Sender, die bisher das alte EI-Toolkit unterstützt haben müssen daher angepasst werden. Der Anpassungsaufwand ist jedoch sehr gering.

Literaturverzeichnis

[booa] Boost C++ Libraries. URL http://www.boost.org/. (Zitiert auf Seite 14)

[boob] Boost Software License. URL http://www.boost.org/users/license.html. (Zitiert auf Seite 27)

[booc] Boost Software License - FAQ. URL http://www.boost.org/users/license.html#FAQ. (Zitiert auf Seite 27)

[cma] CMake - Cross Platform Make. URL http://www.cmake.org/. (Zitiert auf Seite 14)

[dox] Doxygen. URL http://www.stack.nl/~dimitri/doxygen/. (Zitiert auf den Seiten 14 und 25)

[emo] Emotiv | EEG System | Electroencephalography. URL http://www.emotiv.com/. (Zitiert auf Seite 20)

[GNUa] Frequently Asked Questions about the GNU Licenses - GNU Project - Free Software Foundation (FSF). URL http://www.gnu.org/licenses/gpl-faq.html#IfLibraryIsGPL. (Zitiert auf Seite 27)

[GNUb] The GNU General Public License v3.0 - GNU Project - Free Software Foundation (FSF). URL http://www.gnu.org/copyleft/gpl.html. (Zitiert auf Seite 27)

[gra] Graphics View Framework | Documentation | Qt Developer Network. URL http://qt-project.org/doc/qt-4.8/graphicsview.html. (Zitiert auf Seite 24)

[Holo9] P. Holleis. *Integrating Usability Models into Pervasive Application Development*. Dissertation, LMU München: Faculty of Mathematics, Computer Science and Statistics, 2009. URL http://edoc.ub.uni-muenchen.de/9571/. (Zitiert auf Seite 9)

[jso] JSON. URL http://www.json.org/. (Zitiert auf Seite 14)

[pro] protobuf - Protocol Buffers - Google's data interchange format - Google Project Hosting. URL http://code.google.com/p/protobuf/. (Zitiert auf Seite 15)

[pyt] Python Programming Language – Official Website. URL http://www.python.org/. (Zitiert auf Seite 15)

[qcp] Qt plotting widget | Works Like Clockwork. URL http://www.workslikeclockwork.com/index.php/components/qt-plotting-widget/. (Zitiert auf Seite 25)

[qml] Introduction to the QML Language | Documentation | Qt Developer Network. URL http://qt-project.org/doc/qt-4.8/qdeclarativeintroduction.html. (Zitiert auf Seite 24)

[qtC] Qt Creator IDE and tools — Qt - A cross-platform application and UI framework. URL http://qt.nokia.com/products/developer-tools/. (Zitiert auf Seite 24)

[qtD] Qt library 4.8 | Documentation | Qt Developer Network. URL http://qt-project.org/doc/qt-4.8/. (Zitiert auf den Seiten 10 und 23)

[qtT] Tutorials | Documentation | Qt Developer Network. URL http://qt-project.org/doc/qt-4.8/tutorials.html#qt-essentials. (Zitiert auf Seite 10)

[rub] Ruby Programming Language. URL http://www.ruby-lang.org/en/. (Zitiert auf Seite 15)

[sig] Signals & Slots | Documentation | Qt Developer Network. URL http://qt-project.org/doc/qt-4.8/signalsandslots.html. (Zitiert auf Seite 23)

[sqla] SQLite Copyright. URL http://www.sqlite.org/copyright.html. (Zitiert auf Seite 27)

[sqlb] SQLite Home Page. URL http://www.sqlite.org/. (Zitiert auf Seite 25)

[SWI] Simplified Wrapper and Interface Generator. URL http://www.swig.org/. (Zitiert auf Seite 14)

Alle URLs wurden zuletzt am 16.07.2012 geprüft.

Erklärung

Hiermit versichern wir, diese Arbeit
selbständig verfasst und nur die angegebenen
Quellen benutzt zu haben.

_____

(Stefanie Baur Steffen Hanikel Dominik Olp)