Anmerkung: Erste Version der Dokumentation es fehlen noch viele Diagramme
und die meisten Abschnitte sind momentan nur mit kurzen Abschnitten oder Kommentaren
beschrieben die mir als Hilfe dienen und noch ausformuliert werden muss.
Abgesehen davon muss noch die Rechtschreibung überprüft werden und der Text noch formatiert werden.
Einige Bennungen stehen noch nicht fest und eine endgültige Bennung stehet sowohl im Code als auch im Dokument noch an.

Verteilte Systeme mit OSC-Kopplung
//Name für das Programm finden
Inhaltsverzeichnis:

6. Code
6.1 Test
5.2 Beispiel Code für Configurationen

7 OSC Implementierungen
7.1 CodeQuellen
7.2 OSC OpenSource Beispiele

8 Glossar:

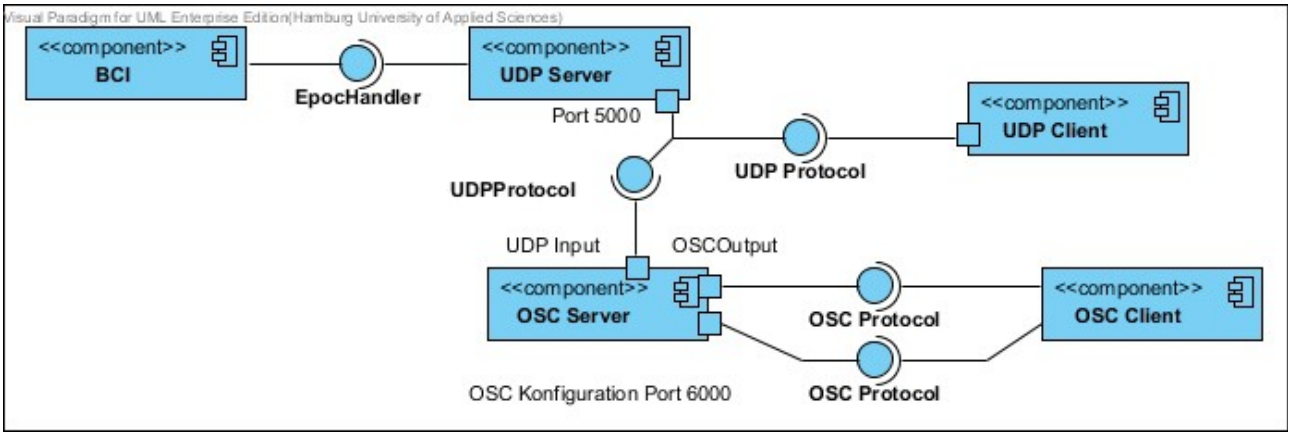
1. Einleitung

1.1 Aufgabe des Programmes

Das Programm wurde dafür entwickelt um die Daten des BCI auf andere Prozesse zu verteilen, unabhängig von deren Betriebssystem und Sprache.
Dafür wird OSC als Kommunikationsprotokoll verwende, welches auf UDP aufbaut.

1.2 Aufbau

Das Programm besteht aus zwei Prozessen. Einen grundlegenen UDP Server mit eigenen Protokoll, geschrieben in C++, welcher entfernten Zugriff auf die Daten realisiert und den OSC Server welcher erweiterte Funktionalitäten liefert, geschrieben in Java. Die Kommunikation zwichen den



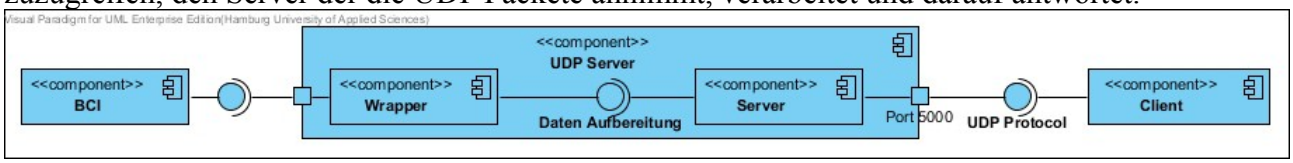
2. UDP Server:

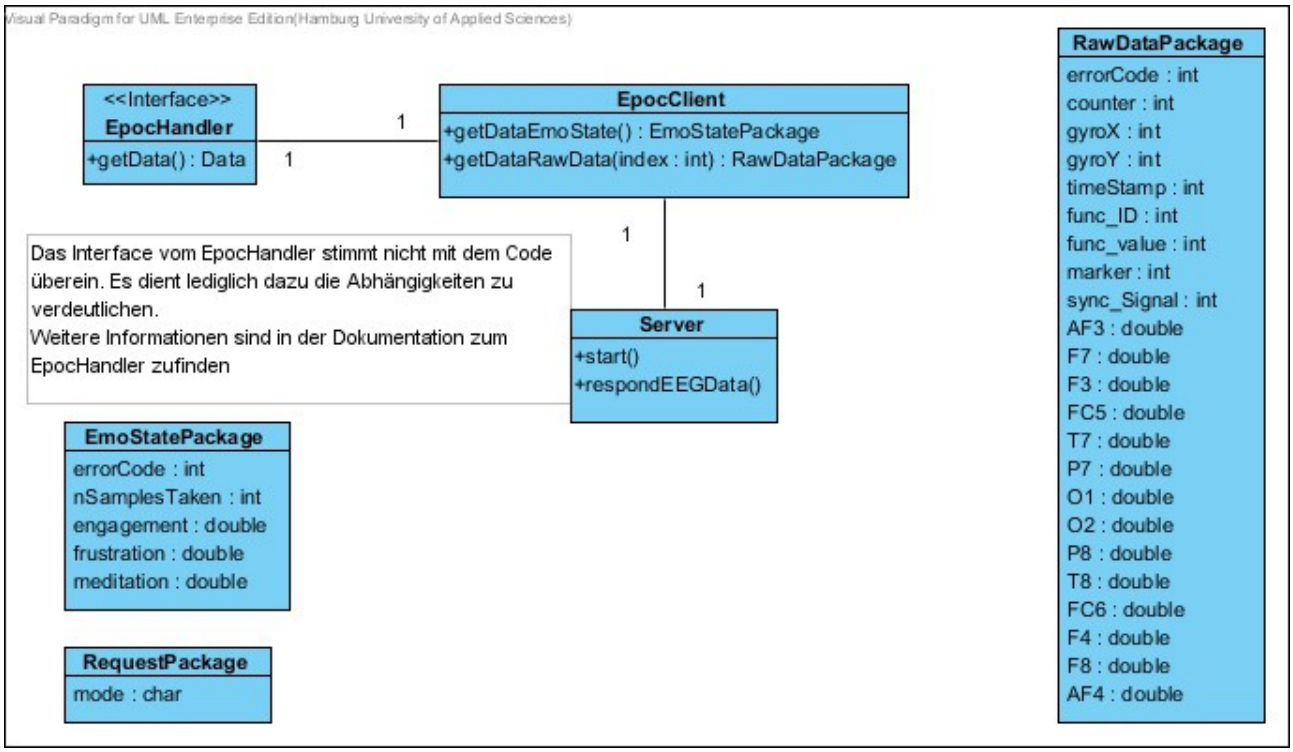
2.1 Einleitung:

Der UDP Server ermöglicht einen Prozess auf die Daten des BCI zuzugreifen mithilfe eines eigenen UDP Protokoll.
Er ist in C++ geschrieben und benutzt zur erstellung der Sockets die Win 32 api.
Als Kommunikationsschnittstelle zum BCI dient der BCI-Wrappe.

2.2 Aufbau:

Der UDP Server unterteilt sich in zweiKomponenten den BCI-Wrapper um auf die Daten des BCI zuzugreifen, den Server der die UDP Packete annimmt, verarbeitet und darauf antwortet.





Class EpocHandler:
Schnittstelle zu dem BCI. Weiter informationen sind der entsprechenden Dokumentationen zu entnehemen.

Class EpoCClient:
Wandelt die daten die der EpocHandler liefert in die vom Server benutzten Packages um.

Class Server:
Öffnet den Socket und verarbeitet ankommende Nachrichten.
Es wird dabei kein neuer Thread erstellt.

Pseudocode.

```
start()
init()
while(true){
package = receivePackage();
if(package->mode == EEGDATA)
        respondEEGData();
}
```

2.3.1 UDP Protocol:

Die Laborrechner verwenden eine 64 Bit Intel CPU und als Betriebssystem Win 7 64 Bit (Speicherbelegung ist little Endian).

```
struct REQUESTPACKAGE{
char mode;
}
```

1 Byte

mode definiert welche Daten geschickt werden sollen.
Standard 1 für die EEGDaten.

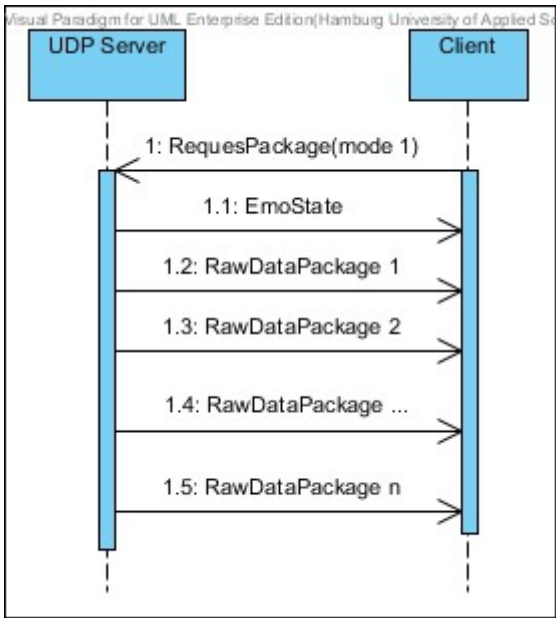
```
struct EMOSTATEPACKAGE{
int32_t errorCode;
int32_t nSamplesTaken;
double engagement;
double frustration;
double meditation;
double excitement;
}
```

40 Byte

```
struct RAWDATAPACKAGE{
int32_t errorCode;
int32_t counter;
int32_t gyroX;
int32_t gyroY;
int32_t timeStamp;
int32_t func_ID;
int32_t func_Value;
int32_t marker;
int32_t sync_Signal;
4 Byte data Alignment; (Siehe Grundlagen Technischer Informatik)
double AF3;
double F7;
double F3;
double FC5;
double T7;
double P7;
double O1;
double O2;
double P8;
double T8;
double FC6;
double F4;
double F8;
double AF4;
}
152 Byte
```

2.3.2 Nachrichtenablauf:

Versendet werden immer die Structs aus 2.3.1.
Da für die Berrechnung von den EmoStates mehrere RawData Samples gebraucht werden,
wird mit den anfragen der EEGDaten,
so viele RawDataPackages mitgeschickt wie Samples benötigt wurden.
Die gesamte Kommunikation ist asynchron.



2.4 UDP Server Konstanten

SDTPORT 5000 Der Port auf dem sich der Socket bindet.
EEGData 1 Konstante für den Request mode

3. OSC Server:

3.1 Einleitung
Die Grundfunktion des OSC Server ist die Verteilung der EEGDaten in Form von OSC Paketen. Er verfügt aber gleichzeitig auch über eine observerartige Funktion die es einen Erlaubt ein Event zu definieren wenn es eintritt, wird eine vorher festgelegte Nachricht abgeschickt (Kapitel 3).

3.3 Controller

Der Controller dient zur Verwaltng der Message Threads.
Alle Message Threads sind in einer HashMap gespeichert mit ihren Namen als Key.
Der Controller leitet die Methodenaufrufe zu den jeweiligen Thread weiter und verwaltet die Daten des BCI.

Class Controller
package controll
//Class Diagramm einfügen

3.4 OSC Input

Beim OSC Input werden OSCNachrichten empfangen um Message Threads
zu manipulieren(siehe Kapitel 3)

Class OSCInput
package com.osc

Methoden:
OSCInput(int port, Controller c)
Erstellt einen OSC Inputport und fügt die Listener für die Konfiguration hinzu(siehe Kapitel 3), welche mit dem Controller c interagieren.

3.5 Pollthread

Um die Netzwerkbelastung zwischen den UDP Server und den OSC Server möglichst gering zu halten, liegen auf dem Java Server immer die aktullen Nachrichten des BCI vor, mit diesen arbeiten die Message Threads.

Class Pollthread
package controll

Methoden:
Pollthread(UDPClient u, double f, Controller c)
u = Schnittstelle zwichen OSC Server und UDP Server
f = Frequenz wie häufig die Daten vom UDP Server gepollt werden soll
c = Der Controller an den die Daten weitergeleitet werden sollen

3.6.1 UDP Client

Schnittstelle zwischen UDP und OSC Server

Class UDPClient
package com.udp

Schnittstelle zwischen den UDP Server und den OSC Server

UDPClient(String serverName, int port)

serverName = Name oder Textdarstellung der IP des UDP Servers

port = port des C++ Servers

EEGData getEEGData()

Holt die Daten des BCI vom UDP Server und wandelt sie in die EEGData Klasse um.

3.6.2 Datenpakete

Statt der C++ Strukts werden Klassen verwendet.

Class EmoState
package dataPackages

Variablen:
int nSamplesTaken
double engagement
double frustration
double excitement
double meditation

Methoden:
EmoState(byte[] data)
Erstellt einen EmoState Klasse aus den Daten einer UDP Nachricht

Object[] toArray()
Wandelt die Variablen in ein Object[] um für die OSC Nachricht
Object[] {nSamplesTaken, engagement, frustration, meditation, excitement}

Class RawData
package dataPackages

Variablen:
public final int counter;
public final int gyroX;
public final int gyroY;
public final int timeStamp;
public final int func_ID;
public final int func_Value;
public final int marker;
public final int sync_Signal;

public final double AF3;
public final double F7;
public final double F3;
public final double FC5;
public final double T7;
public final double P7;
public final double O1;
public final double O2;
public final double P8;
public final double T8;
public final double FC6;
public final double F4;
public final double F8;
public final double AF4;

Methoden:
RawData(byte[] data)
Erstellt einen RawData Klasse aus den Daten einer UDP Nachricht

Object[] toArray()
Wandelt die Variablen in ein Object[] um für die OSC Nachricht
Object[] {counter, gyroX, gyroY, timeStamp, func_ID, func_Value, marker, sync_Signal
AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, AF4}

Class EEGData
package dataPackages

EEGData(EmoState e, LinkedList<RawData> r)
Erstellt einen EEG Datensatz aus den EmoState und den zugehörigen RawDatasamples

3.7 Message Thread

Überprüft in einer festgelegten Frequenz seine Events mit den Daten des BCI,
sollte das Event eintreten werden die Nachrichten abgeschickt.

```
while(true){
if(check Event)
send(Messages)
}
```

3.8 Event

Event
Interface welches die Methoden die der Message Thread nutzt einfordert
boolean checkCondition(EEGData eeg)
prüft ob das Event Eintritt anhand der EEGDaten

to Message(EEGData eeg)
gibt die Messages des Events zurück anhand der EEGDaten

AbstactEvent
Das Event baut auf eine Liste mit Condition und eine mit Messages auf und stellt

die Methoden für CheckCondition und toMessage bereit.
Die Subklassen dieses Events kann man über OSC Konfigurieren

AlwaysSendEvent
Gibt super.CheckCondition zurück

OneTimeEvent
Gibt true zurück wenn ein wechsel von false auf true stattfindet
Gibt super.CheckCondition xor last

3.8.1 Condition

Condition
Interface welches die Methoden einforder welche das Event benutzt

public boolean checkCondition(EEGData eeg)

Überprüft anhand der EEGDaten ob die Condition true ist

Abstract Condition
Bietetet Methoden zum vergleichen und um die Variablen aufzulösen

Float Condition
Vergleich mit einen Float

String Condition
Vergleich mit einer Variable

Int Condition
Vergleich mit einen Int

3.8.2 Message

Message
Interface welches die Methoden einforder welche das Event benutzt

public LinkedList<OSCMessage> toMessage(EEGDaten eeg)
gibt die Message zurück welche anhand der eeg Daten erstellt werden

Float Message
Es soll ein selbstdefinierter Float geschickt werden

Int Message
Es soll ein selbstdefinierter Integer geschickt werden

String Message
Es soll ein selbstdefiniertert String, oder ein Datensatz geschickt

3.9 Konstanten

Sämtliche Konstanten sind in der Klasse Constants definiert
byte REQUESTEEGDATA = 1
int RESPONSEEMOSTATESIZE = 40
int RESPONSERAWDATASIZE = 152

//Konstanden aus der Java Datei einfügen

4 Jave Server Konfiguration

4.1 OSCMessages

Eingabe an den Controller über OSCMessages
//Aufrufe sind im projekt gespeichert

4.2 Wrapper

OSC Kommunikation abstrahiert
Nur Java
//Muss im Zuge von Umbauarbeiten noch angepasst werden

4.3 FileReader

Liest aus einer Datei die OSC Befehle aus
Java
Man kann dem Server sagen eine Datei auszulesen über OSC

4.3.1 StringParser

Teil aspekt des FileReaders
Einzeln benutzbar
Einsatzmöglichkeit wäre eine Gui

4.4 Selbst erstellte Events

//gute möglichtkeiten an den man den Code verändern kann finden

5 Benutzung des Programmes
5.1 Start Programm
//UDP Server Starte dann OSC Server etc.
5.2 Fehlerquellen
//mögliche Fehlerquellen aufschreiben
5.3 möglichkeiten für Lokale Test
//Anstarten des Programmes mit Dummywerten für die Nachrichten

6. Code
//auf tests verweisen andere Versionen etc.
6.1 Test
6.2 Beispiel Code für Configurationen

7 OSC Implementierungen

7.1 CodeQuellen
//quelle des benutzten OSC Codes
7.2 OSC OpenSource Beispiele
//andere quellen für andere Sprachen

8 Glossar:

//Begriffe erklären