## Rock

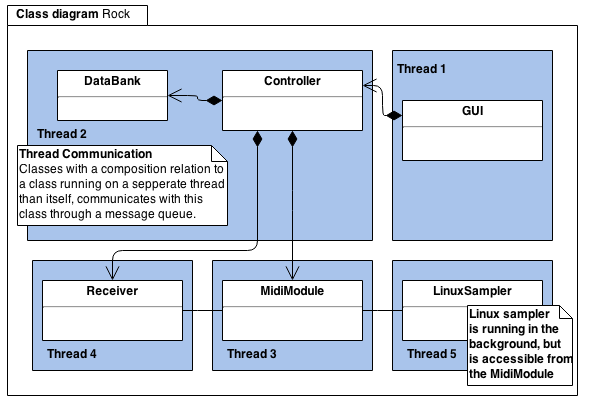
Softwaren på Rock implementeres event-baseret og indeholder således en række selvstændige tråde. De vil være opdelt i to hovedkategorier, bestemt ud fra hvor kritisk afviklingshastigheden er for den musikalske oplevelse ved brug af systemet. Disse er benævnt som følger:

* **Slow Lane**
* **Fast Lane**

Visse softwaremoduler vil gå igen for begge baner, men hastigheden af disse (prioriteten[[1]](#footnote-1) i systemets scheduler), vil afhænge af hvilken tråd, de indgår i.

Der vil i følgende afsnit være udfærdiget applikationsmodeller for de individuelle moduler. Hver af disse moduler har sin egen tråd, og applikationsmodellerne rummer dermed de enkelte trådes funktionalitet.

### Overordnet Klassediagram Rock

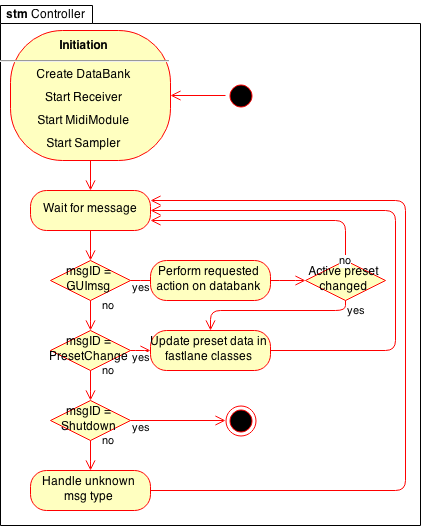


### Trådkommunikation

I dette afsnit skitseres overordnet, hvordan kommunikation, trådene imellem, vil foregå.

### Controller

Systemets centrale klasse er Controller. Denne klasse står for at starte resten af systemets tråde, håndtere systemets databank og kontrol af fast lane klasserne ved skift af preset. Herunder ses et stm diagram som viser klassens opførsel:



### Slow Lane

Denne bane håndterer de softwaremoduler i systemet, hvis afvikling ikke er kritisk for den musikalske oplevelse ved benyttelse af BodyRock3000.

#### SW-moduler i Slow Lane

Bruger-indstillinger af systemet (UC 2-4) tildeles en lav prioritet i systemets scheduler, da disse kun anvendes sjældent, og ikke er latens-kritiske. Følgende softwaremoduler tager sig af disse indstillinger:

* GUI
* DataBank
* Controller

Controller og DataBank vil blive brugt i hhv. Slow og Fast Lane afhængigt af hvilken sammenhæng de benyttes i.

#### Applikationsmodel for Slow Lane

*Slow Lane* benyttes i systemet som betegnelse for trådene *GUI* og *Controller*. Denne del af systemet står for at modtage input fra brugeren, samt for at administrere *DataBank* i henhold til disse.

Herudover administrerer *Controller* ligeledes trådene i *Fast Lane*, hvilket bl.a. indebærer at modtage presets fra *Receiver*, samt at sende opdaterede data fra *DataBank* til *MidiModule* og *ALSA-Functions*.

##### Sekvensdiagram

##### 

I dette diagram ses en beskrivelse af kommunikation mellem hhv. *GUI-* og *Controller*-tråden, når brugeren skal oprette en ny sensorkonfiguration.  
Denne kan betragtes, som værende repræsentativ for den generelle kommunikation mellem de to tråde.

*GUI*-delen i diagrammet er en samlet betegnelse for diverse klasser i User Interface. Disse klasser kommunikerer gennem *MsgHandler,* ned til *Controller*.

### Fast Lane

Fast Lane håndterer de softwaremoduler i systemet, hvis afvikling er kritisk for den musikalske oplevelse ved benyttelse af BodyRock3000.

#### SW-moduler i Fast Lane

Benyttelse af systemet som instrument (UC 5-8) tildeles en høj prioritet i systemets scheduler, da disse er latens-kritiske. Følgende softwaremoduler benyttes ved brug af Rocks MIDI- og lydgenereringsfunktion:

* Receiver
* MidiModule
* ALSA-functions
* (Controller)
* (DataBank)

*Controller* og *DataBank* indgår i UC5, ”Vælg preset”. UC5 er en afstikker-funktionalitet i Fast Lane, idet den kan anses som en (re-)initieringsfunktion for banen Fast Lane, hvis hovedfunktion er at omdanne sensordata til hhv. MIDI-signaler og lyd.

#### Receiver

##### Sekvensdiagram



**Figur 1** Sekvensdiagram over operationerne udført af klassen Receiver

##### Klassediagram



**Figur 2** Klassediagram over klassen Receiver

##### Funktionsbeskrivelser

|  |  |
| --- | --- |
| **Funktion** | Constructor |
| **Parametre** | Pointer til køen fra klassen *MidiModule* |
| **Returværdi** | Ingen |
| **Beskrivelse** | Kalder funktionen *connect* |

|  |  |
| --- | --- |
| **Funktion** | Destructor |
| **Parametre** | Ingen |
| **Returværdi** | Ingen |
| **Beskrivelse** | Afbryder den eventuelt kørende tråd, og kalder funktionen *disconnect* |

|  |  |
| --- | --- |
| **Funktion** | start |
| **Parametre** | Ingen |
| **Returværdi** | Ingen |
| **Beskrivelse** | Starter en tråd som kalder funktionen *receive* i en uendelig løkke |

|  |  |
| --- | --- |
| **Funktion** | receive |
| **Parametre** | Ingen |
| **Returværdi** | Ingen |
| **Beskrivelse** | Aflæser og tømmer Rx-bufferen fra UART-noden *ttyAMA0*, som indeholder data modtaget fra Body over Bluetooth. Reagerer herefter på den først modtagne byte:  Startbyte = 0x0F: Modtaget sensordata pakkes. Registreres det, at en ny omgang sensordata er påbegyndt, sendes den sidste pakke videre til *MidiModule*.  Startbyte = 0xF0: Modtaget preset videresendes til *Controller*.  Ellers: Modtaget data ignoreres. |

|  |  |
| --- | --- |
| **Funktion** | connect |
| **Parametre** | Ingen |
| **Returværdi** | Ingen |
| **Beskrivelse** | Opretter *read only*, blocking forbindelse til noden *ttyAMA0*, som håndterer UART-forbindelsen på de benyttede ben på Rock. |

|  |  |
| --- | --- |
| **Funktion** | disconnect |
| **Parametre** | Ingen |
| **Returværdi** | Ingen |
| **Beskrivelse** | Afbryder forbindelsen til, og frigiver, noden *ttyAMA0*. |

#### Applikationsmodel for MidiModule

I dette afsnit findes en applikationsmodel for MidiModule-tråden.

##### Klasseidentifikation

MidiModule opdeles i underklasser for at vise systemets interne virkemåde.

**Boundary klasser:**

* ***MsgQueue***: Inderholder *messages* fra hhv. Receiver (data) og Controller (preset)
* ***AlsaAdapter***: Interface til ALSA-funktionerne

**Domæne klasser:**

* ***SensorConfig***: Et objekt af klassen SensorConfiguration i det aktive preset, der indeholder konfiguration for hvilket datapunkt, et specifikt MidiSignal skal genereres ud fra
* ***MappingScheme:*** Et objekt af klassen MappingScheme i et specifikt SensorConfig-objekt. Denne indeholder konfigurationen og funktionaliteten for, hvordan et bestemt MidiSignal genereres

**Controller klasse:**

* ***MidiModule:*** Det overordnede objekt, der indeholder den samlede funktionalitet for, hvordan en mængde af datapunkter omdannes til en række MidiSignaler, jf. det nuværende aktive preset.

##### Sekvensdiagram

Herunder ses et sekvensdiagram for det interne programflow i MidiModule. MsgQueue blokerer indtil der forekommer en ny *message*. Derefter håndteres den modtagne *message* jf. dens type.

For data messages loopes gennem de aktive sensorkonfigurationer, hvor SensorConfig tilgår et specifikt datapunkt, og MappingScheme dernæst opdaterer det givne SensorConfig objekts MidiSignal.

For preset messages nedlægges først de aktive sensorkonfigurationer, hvorefter et preset svarende til den modtagne message findes og initieres.



##### State Machine Diagram

MidiModule har to hovedtilstande: *Handling* og *Idle*. *Handling* igangsættes ved indkommende *messages* og igangsættes med undertilstand *Handling Queue*, hvori *msgQueue* tømmes. Afhængigt af beskedens type, igangsættes enten *Handling Data* eller *Handling* *Preset*. Når *msgQueue* er håndteret, går *MidiModule* i tilstand *Idle*.



##### Klassediagram

Herunder er et klassediagram for*MidiModule* og dens bestanddele angivet på figur XX.



Figur Samlet Klassediagram for MidiModule

##### AlsaAdapter

Følgende afsnit omhandler funktionaliteten hos, og implementeringen af, klassen AlsaAdapter. Klassen modtager en vektor af MidiSignal structs og sender dem vha. ALSA videre til enten LinuxSampler eller et eksternt lydkort.

###### Anvendte RawMidi funktioner

Følgende er en beskrivelse af de anvendte metoder fra RawMidi interfacet som beskrevet i *ALSA*. For at muliggøre anvendelsen af disse metoder blev headerfilen ”alsa/asoundlib.h” inkluderet.

|  |  |
| --- | --- |
| **Funktion** | snd\_rawmidi\_open |
| **Parametre** | snd\_rawmidi\_t \*\* inputp,  snd\_rawmidi\_t \*\* outputp,  const char \* name,  int mode |
| **Returværdi** | 0 ved succes. Ved fejl returneres en nagativ fejlkode. |
| **Beskrivelse** | Denne funktion åbner en ny forbindelse til den RawMidi kompatible ALSA port specificeret ved ACSII identifieren name.  Name defineres som ”hw:CARD,DEVICE,SUBDEVICE”, hvor CARD, DEVICE og SUBDEVICE hentyder til enten navnene og/eller numrene for den ønskede port.  Mode specificerer måden, hvorpå porten åbnes. Der forefindes i frameworket en række defines, som kan anvendes her i stedet for den faktiske int værdi. |

|  |  |
| --- | --- |
| **Funktion** | snd\_rawmidi\_close |
| **Parametre** | snd\_rawmidi\_t \*\* rawmidi |
| **Returværdi** | 0 ved succes. Ved fejl returneres en nagativ fejlkode. |
| **Beskrivelse** | Denne funktion lukker den RawMidi port specificeret ved rawmidi og frigiver alle associerede resourser. |

|  |  |
| --- | --- |
| **Funktion** | snd\_rawmidi\_write |
| **Parametre** | snd\_rawmidi\_t \*\* rawmidi,  const void \* buffer,  size\_t size |
| **Returværdi** | 0 ved succes. Ved fejl returneres en nagativ fejlkode. |
| **Beskrivelse** | Denne funktion skriver MIDI beskeden i buffer til den RawMidi port specificeret ved rawmidi. Buffer oprettes som et array af bytes indeholdende den ønskede MIDI besked. Size angiver antallet af bytes i buffer. |

##### AlsaAdapterens funktioner

|  |  |
| --- | --- |
| **Funktion** | AlsaAdapter |
| **Parametre** | Int type |
| **Returværdi** | - |
| **Beskrivelse** | AlsaAdapterens constructor tager en enkelt int, type, som parameter. Denne afgør, hvor AlsaAdapteren outputter de modtagede midi signaler. |

|  |  |
| --- | --- |
| **Funktion** | ~AlsaAdapter |
| **Parametre** | - |
| **Returværdi** | - |
| **Beskrivelse** | AlsaAdapterens anvender snd\_rawmidi\_close til at lukke de / den åbnede RawMidi porte og sætter dem / den til at pege på NULL. |

|  |  |
| --- | --- |
| **Funktion** | open |
| **Parametre** | - |
| **Returværdi** | bool |
| **Beskrivelse** | Denne funktion åbner den RawMidi port, der er specificeret ved type\_, der bliver sat ved AlsaAdapterens oprettelse. Funktionen anvender RawMidi funktionen snd\_rawmidi\_open, med flaget Append. Dette medfører, at beskeder sendt til porten lægges til output bufferen. |

|  |  |
| --- | --- |
| **Funktion** | send |
| **Parametre** | Const std::vector<MidiSignal>& |
| **Returværdi** | bool |
| **Beskrivelse** | Denne funktion modtager en vektor af MidiSignal structs, som specificeret i INDSÆT REFERENCE. I en for-løkke omdannes hvert MidiSignal element i vectoren til et char array på formen {command\_ + channel\_ , param1\_ , param2} og sendes derefter vha. snd\_rawmidi\_write til den RawMidi specificeret ved type\_. |

1. I den endelige version af systemet vil der forekomme trådprioritering. I denne iteration er det ikke implementeret. [↑](#footnote-ref-1)