Software architecture & design

Introduction

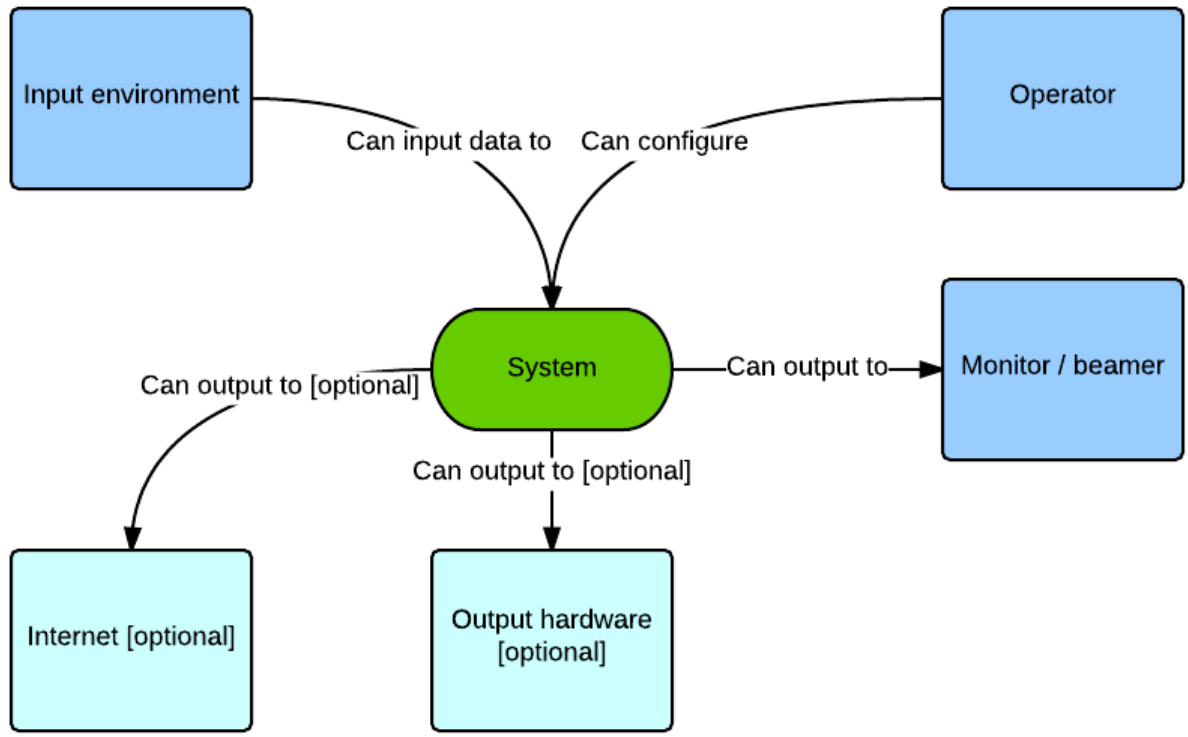
This page describes the overall structure and components for the system. Models are displayed using (UML) diagrams and textual descriptions, which are explained by the reference list.

Found in this document are context models, flow diagrams and architectural decisions.

The software will be using headsets of the IMEC company, provided by Baltan Labs.

Context models

An UML context diagram is used to describe the in- and output(s) of the system. The system can communicate with several devices. The interaction between the system and the devices can be seen in the figure below.



Type	Name	Description
Input	Input environment	Inputs for the system. These include, but not limited to, input files and live streams from headset(s)
Input	Operator	Human interaction with the software
Output	Monitor/beamer	Software can render graphical user interfaces which can be displayed on monitor / beamer
Output	Internet [optional]	Software can “share” a kiss via the internet

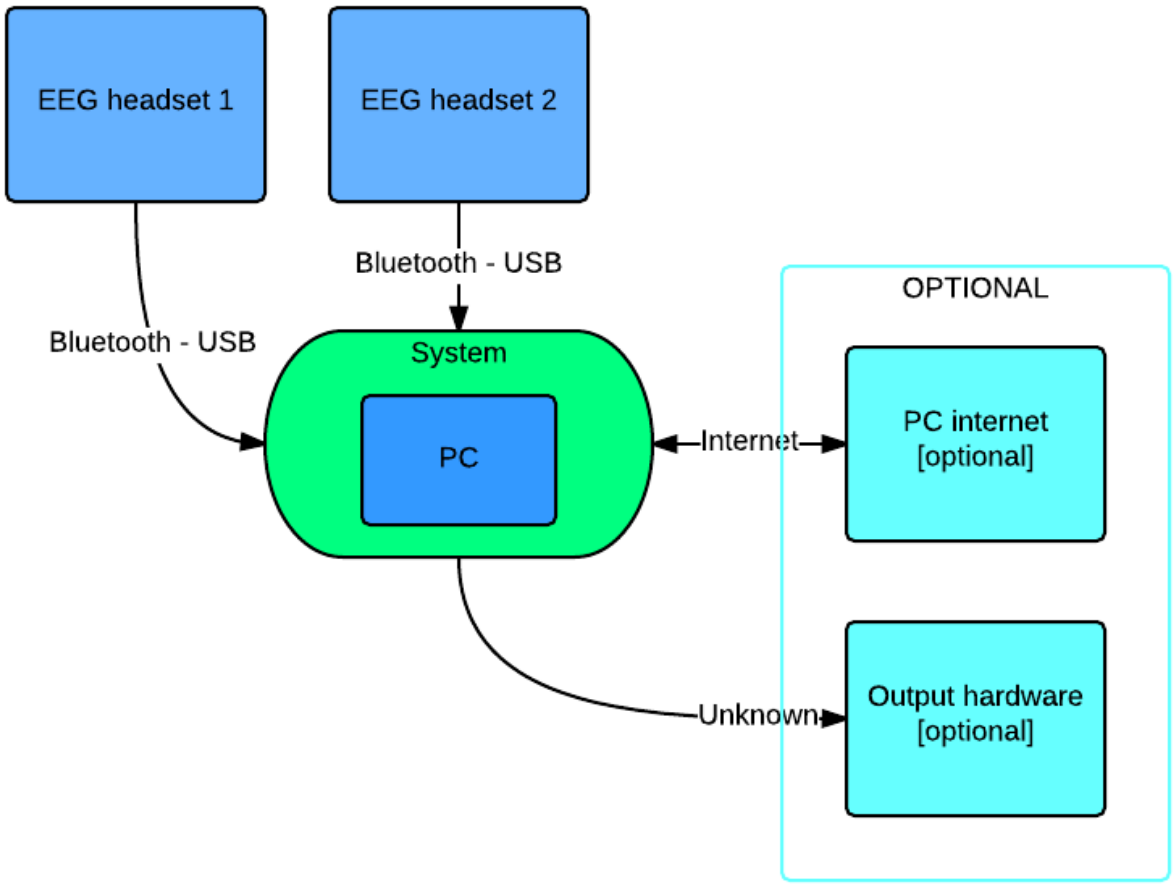
Output	Output hardware [optional]	Software can make a kiss “feelable” via external hardware

Context interaction model

An UML context interaction diagram is used to describe the sensors and actuators of the system. Please see the figure below.

The system can communicate with multiple sub systems. These subsystems have been designed to fulfill a certain function.

The internet network is drawn as optional, just like the output hardware. The reason is that this is not yet clear on how this will work - TODO.

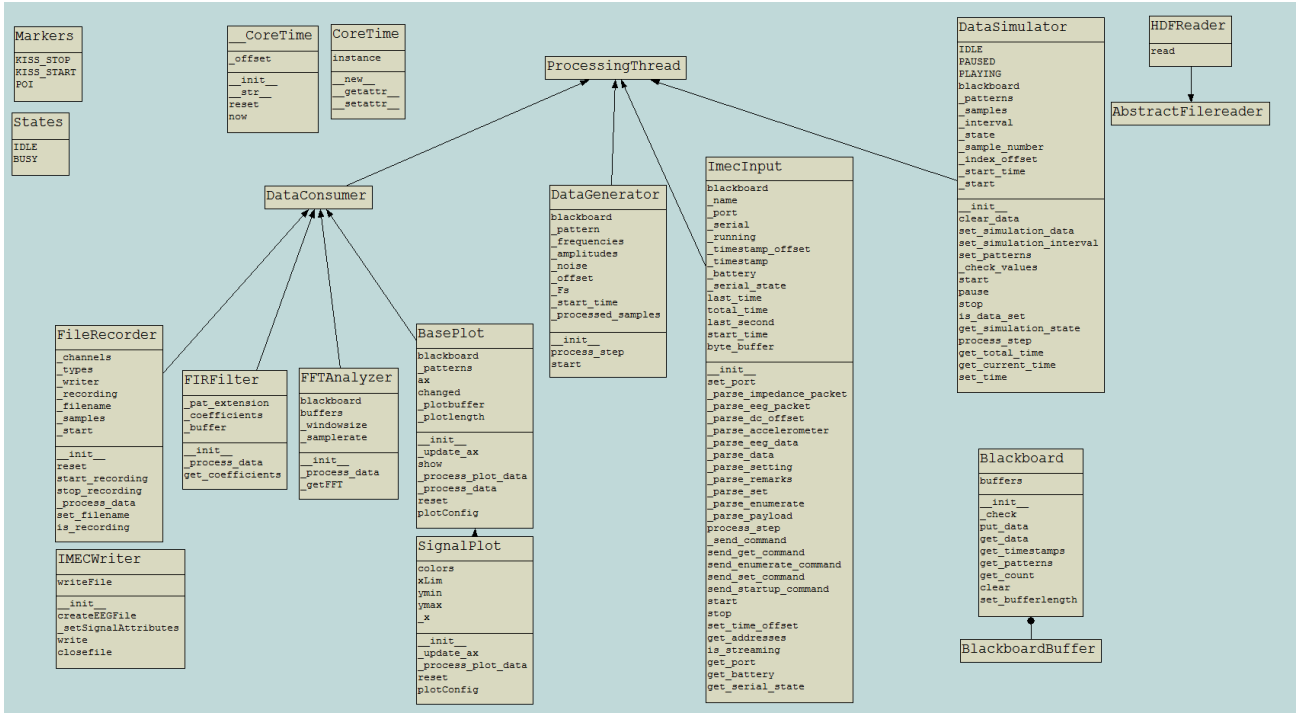


Type	Name	Description
Sensor	EEG headset 1	An IMEC EEG headset which will monitor user’s EEG data and input it to software system
Sensor	EEG headset 2	An IMEC EEG headset which will monitor user’s EEG data and input it to software system
Sensor	PC internet [optional]	Software can “share” a kiss to another user’s PC via the internet
Actuator	Output hardware [optional]	Software can make a kiss “feelable” via external hardware

Class diagrams

MVC pattern

Model diagram



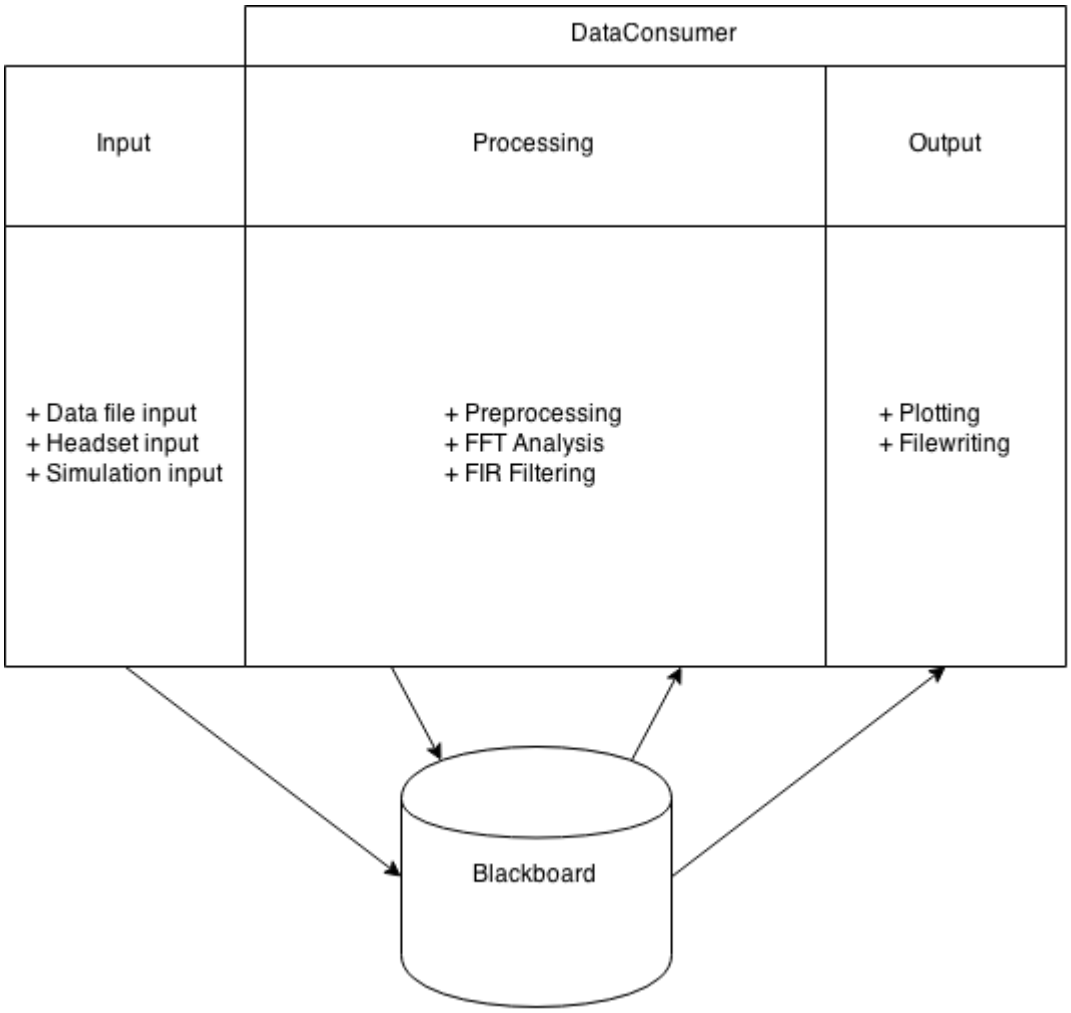
How-to-UML

De software om vanuit python code een UML diagram te genereren is pyNsource:

> https://code.google.com/p/pynsource/downloads/detail?name=pyNsource-1.60-win32-standalone.zip&can=2&q=

Door één voor één alle modules te importeren wordt de hele software structuur duidelijk.

System architecture



The system will have a number of requirements which we will use for our design decisions. One of the main functions of the system will be to get data from multiple sources to multiple destinations with multiple processing steps in between. All these parts, inpu, processing and output, will have to usable and modifyable in real-time. Data plays a central role in all this. Based on the FieldTrip software, we use a Blackboard Architecture to distribute all this data.

The blackboard architecture is based on the idea that there are multiple knowledge sources or expert systems that are all able to see, modify and delete data from a central knowledge bank, hence the metaphor of experts around a blackboard. In the general case of a blackboard system, there should be a control component to make sure that the data is modified only by one expert system at a time. In our system, based on the Fieldtrip software, there is no need for such a control component, as expert systems are only allowed to add new knowledge to the blackboard, rather than modify or delete knowledge from it. As our system will be mostly processing signals, the blackboard will consist of a set of signal buffers. Expert systems that use the buffered data need to check the blackboard regularly to make sure they process all the presented data. By leaving out a control component, this responsibility of processing all the data is with the data consuming components rather than with the data producing components.

Alternative solutions for the data distribution we considered were to use the Observer pattern or Producer-Consumer pattern. A drawback of the observer pattern is that two objects that need to share data always need to be explicitly coupled wheras to data sharing objects act completely independently in the blackboard pattern. This makes it easier to modify inputs, processing and output components in real-time. The producer-consumer pattern has the same drawback as well as the problem that we may have multiple data producer for one data consumer or multiple data consumers for one data producer.

Example: input and output

Consider the following situation: we have two headsets with four channels. We need to record one of the headset's data, and we need to plot one of the channels for both the headsets. The two headsets are the data producing components, the data recorder and the data plotter are the data consumers. The blackboard buffers the data produced by the headsets. The headsets will store their data under the following patterns:

```
"/eeg1/channel_1"
"/eeg1/channel_2"
"/eeg1/channel_3"
"/eeg1/channel_4"
"/eeg2/channel_1"
"/eeg2/channel_2"
"/eeg2/channel_3"
"/eeg2/channel_4"
```

The data consumers will run on their own thread and poll the blackboard for data. The data recorder will search for all buffers starting with "/eeg1" and store all that data in a file. The data plotter will search for all buffers with "/channel_1" as second element in the pattern.

Example: processing

An example of a processing component can be a low-pass filter. Let's say we have the same two headsets with the same four channels as the previous example. Let's say we want to filter the data from headset two, channel 1 with a low-pass filter at 50 Hz. This filter would search the blackboard for a buffer with pattern "/eeg2/channel_1". The filtered data will be stored to a buffer with the pattern "/eeg2/channel_1/lpf_50". Such a component is therefor a data consumer and a data producer at the same time.