Modularity and Design Patterns

# Class diagram

This section defines the system’s class diagram for future reference.

A diagram of a computer program

Description automatically generated

Figure: Class diagram

The implementation consists of only two classes, Sentry, and Module. The application’s core consists of a single instance of Sentry. The class is responsible for managing the connected modules. It reads the data they produce, and delivers it to the output handler, in our case a websocket.

# Sampling rate difference

A fundamental problem within this niche domain is that external modules can have varying output frequencies. This is a potential risk of a blocking operation, if the Sentry should wait on data from one or more Modules at any point in time. One might recognize a potential case for the producer/consumer arrangement, as illustrated by Dijkstra [https://www.cs.utexas.edu/users/EWD/transcriptions/EWD13xx/EWD1303.html]. I.e., the Module produces data in an unknown interval (producer), and the Sentry reads data in a different, arbitrary interval (consumer).

In the classic producer/consumer arrangement, each package of data from the producer must be processed by the consumer, which is handled by a buffer between the two. For the present application however, the Sentry’s frequency is the bottleneck - if a Module produces faster than the Sentry consumes, data sent between consumption intervals can be seen as loss of detail and be discarded. Hence, this application does not need a buffer and can suffice with storing the latest value from each Module, whenever the respective Module produces new data. Whenever the Sentry is ready, it can read the latest produced value from the Module.

The application of the producer-consumer design avoids blocking operations and guarantees a stable output frequency with the most up to date values of the input Modules.

# Simplifying Module Integration

The following software design choices aid the ease of integrating external modules.

* Fail safety: on software failure in any external module, the module is removed from the Sentry. This is done in the wrapper for the consumer, to guarantee internal responsibility over managing healthy external modules.
  + This ensures uptime for the remaining modules.

## Interface for External Modules

To allow any external modules to connect to Sentry, Modules