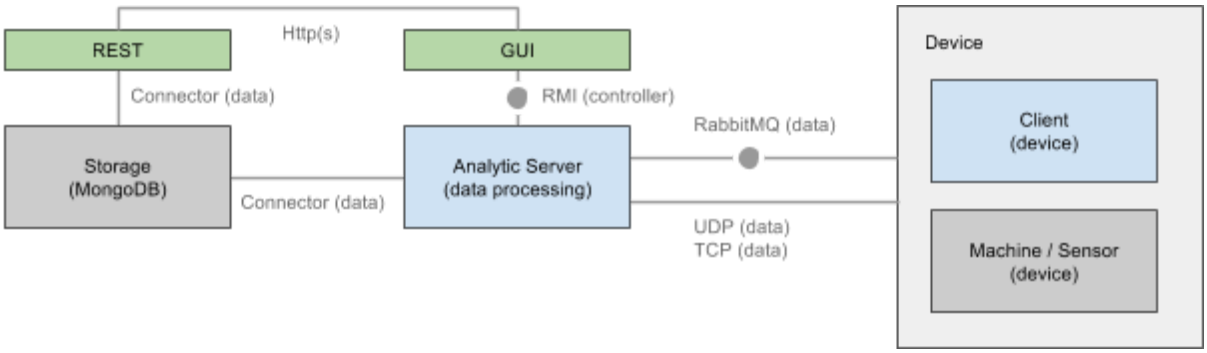
Randy Wind (s1791540)
Andrew Peter Prifer (s2535009)

Smart Service Center

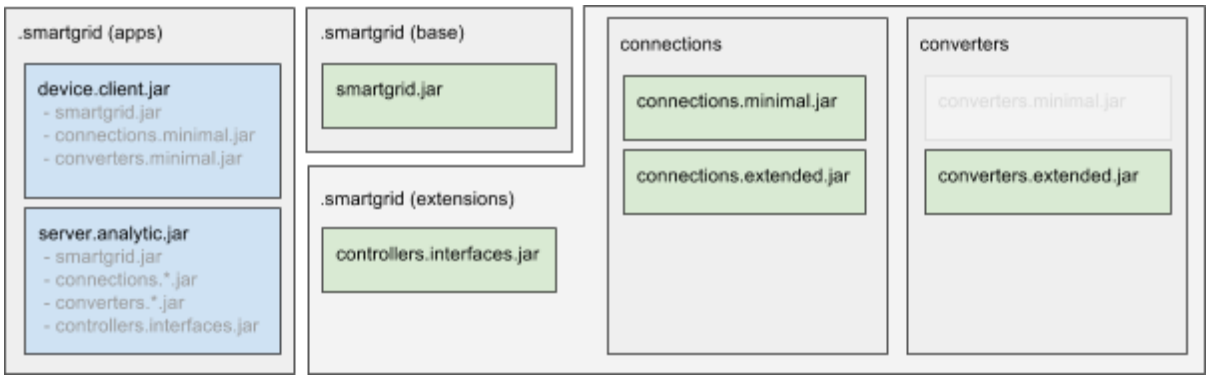
Project Document



In this project we prepare a system that can use Machine Learning on real-time data streams from sensors and machines (servers) process this and return actions to the devices.

Project Layout

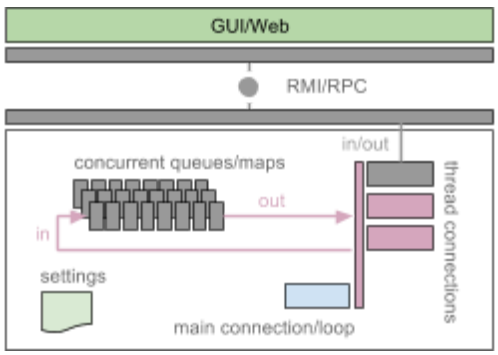
The project is separated into multiple packages, the base of the project, extended connections, extended converters, controllers interfaces, minimal connections and minimal converters.

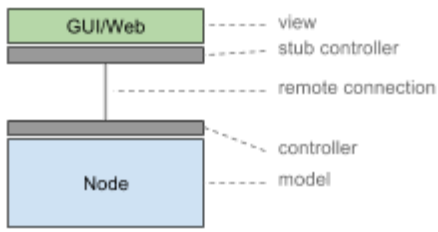


Entities

Node (basic entity)

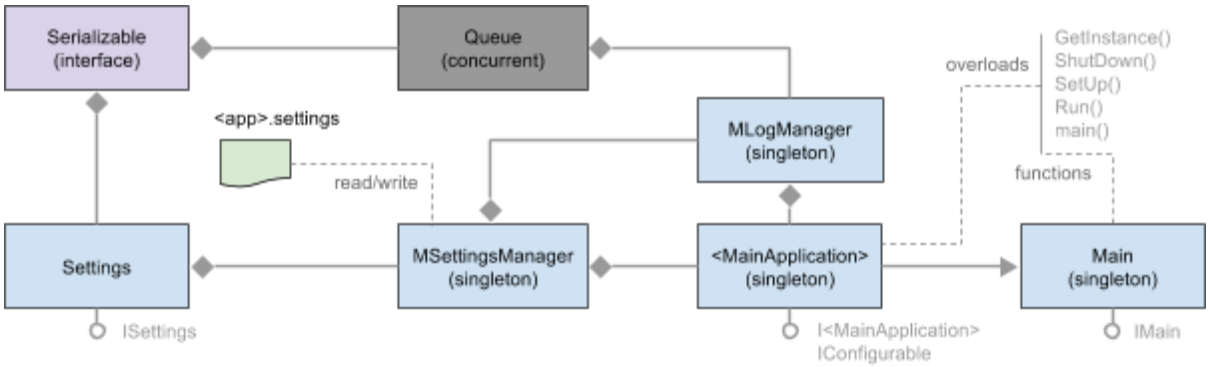
A node represent the singleton application's (instance of Main class) e.g. Client, Server and optionally the GUI. They consists of queues for exchange between connections. Connection and the main loop pull and offer data from and to the queues, some connections represent remote calls or com device calls. The main loop executes the functionality of the node itself.





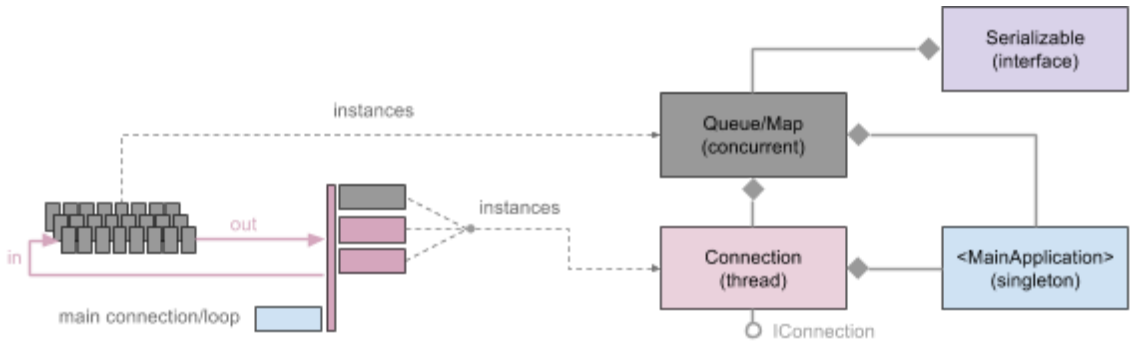
Main (application)

The main class is a singleton, they are used as a basis for the stand alone applications containing the main (program entrypoint) function. Each stand alone application inherits from the Main class and overloads the base functions main, GetInstance, ShutDown, SetUp and Run. Each standalone application uses the LogManager and SettingsManager, these are both singletons as well. The SettingsManager simply reads settings from a file. The LogManager handles logging and the log Queue (not yet supported).



Connection (thread)

The connection objects pull and offer serializable objects to the queues/maps, to consider a reasonable level of concurrency, we chose to use concurrent queues. In some cases multiple connection offer and pull from the same queue.



Multiple connections from one queue poses a problem when we want to pull specific routed objects from the queue. To prevent the queues from getting blocked by unwanted routes Deques work well, using (instead of peek) pullFirst followed by offerFirst if no match, ensures other threads can continue pulling during the check. Adding a timestamp to the route wrap threads can drop data if they remain to long on the queue.

Analytic Server Node (data processing)

Collects data from any devices, processes it, routes actions back to the Devices and stores some or all results to MongoDB. Data is collected through UDP, TCP and RabbitMQ Connection Consumers to the DataQueue, processed by the Analytic Server, than actions are generated, internally routed to their incoming connection type and routed back to the corresponding devices.

Client Node (Device/Machine)

Software daemon that collects data from its host machine / sensor. Sends data to the real time data queue, receives actions from the Analytic Server entity though RabbitMQConsumer with routing option. Data is generated and offered to the DataQueue and send by RabbitMQ Producer Connection to The analytic Server. Some smaller device

may not have the ability to use RabbitMQ as a communication means, so therefore we allow (direct) UDP and TCP Consumer Connections to the Analytic Server.

REST (API to Results)

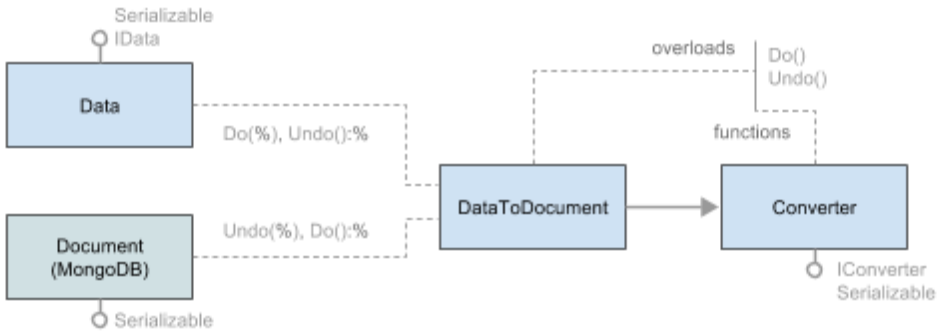
To allow external observation (or optional manipulation) of the results produced by the Analytic server a REST API is implemented using NodeJS express and mongoDB driver. Current GET operations are supported.

```
<query> : <attribute> | <time_range> | <%_range> | <query>&<query>
<%_range> : &%_min=<value> | &%_max=<value>
<time_range> : &from=<time> | &to=<time>

domain/results
domain/results?<query>
domain/results/<_id>

domain/data
domain/data?<query>
domain/data/<_id>
```

Storage (of Results)



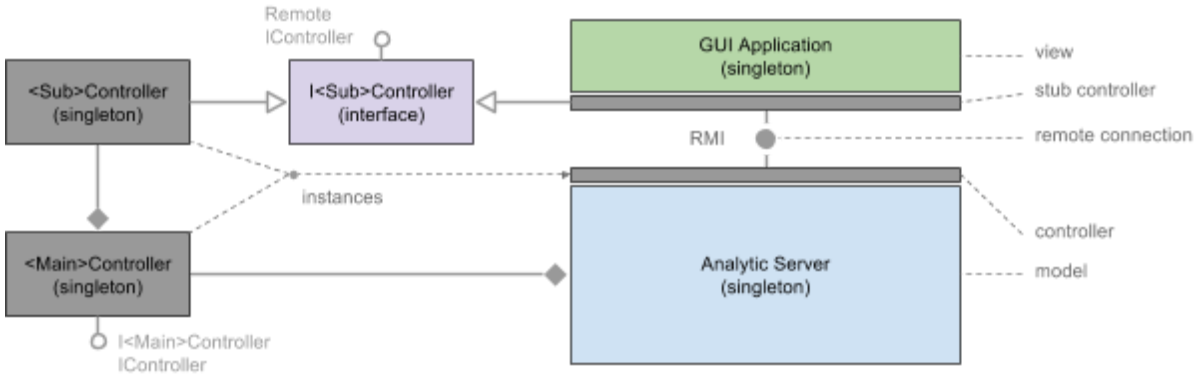
MongoDB is used to store the results computed by the Analytic Server. Only the results collection is inserted, although compressed instances of data for later use in analytics could also be stored in the data collection.

```
result { device_id, timestamp, interval, actions, weight, flag }

data#sensor { sensor_id, timestamp, interval, signal, flag }
data#machine { machine_id, timestamp, interval, activity, handled, unhandled, flag }
```

GUI (Controller Analytic Server, Dashboard)

The GUI entity uses RMI Caller Connection to connect with the Analytic server, It stores a stub controler int a promise object when the connection is successful. The GUI entity automatically reconnects if connection has been broken, waiting for the RMI service to be running, connection is don through the Analytics Server UUID. This was chosen to strictly separate the GUI/View from the model and controller.



The Dashboards (specific GUI element) allows the users to trace and observer data and results through the REST interface. Most of the date representations and views is computed client side and not by the rest server.