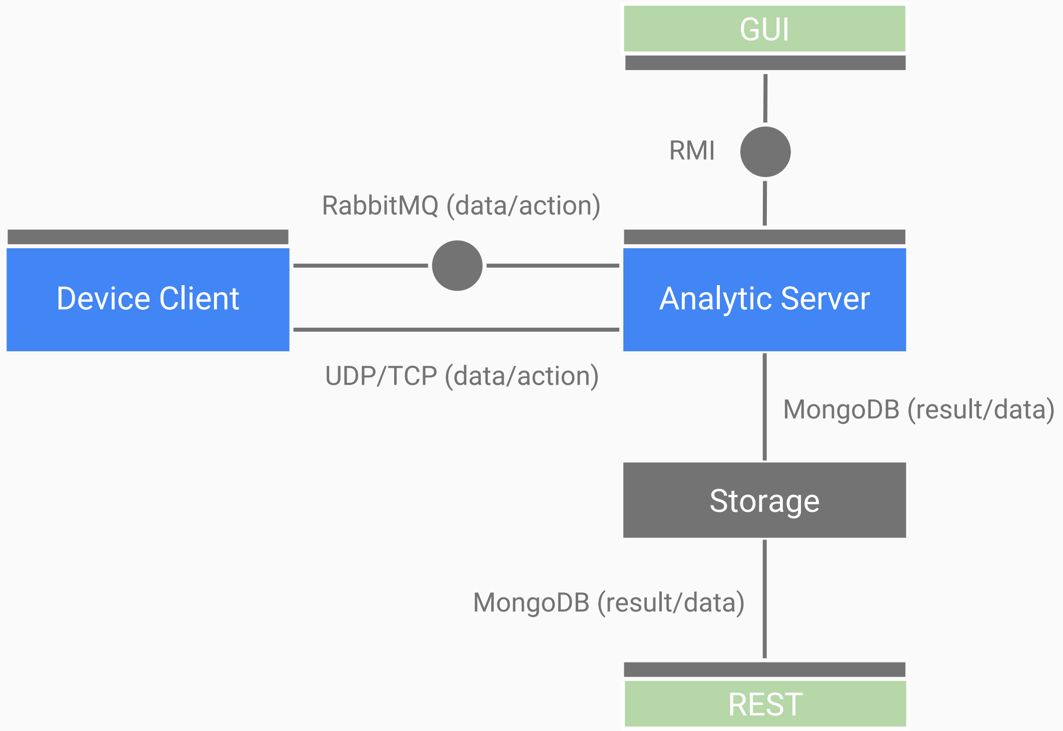
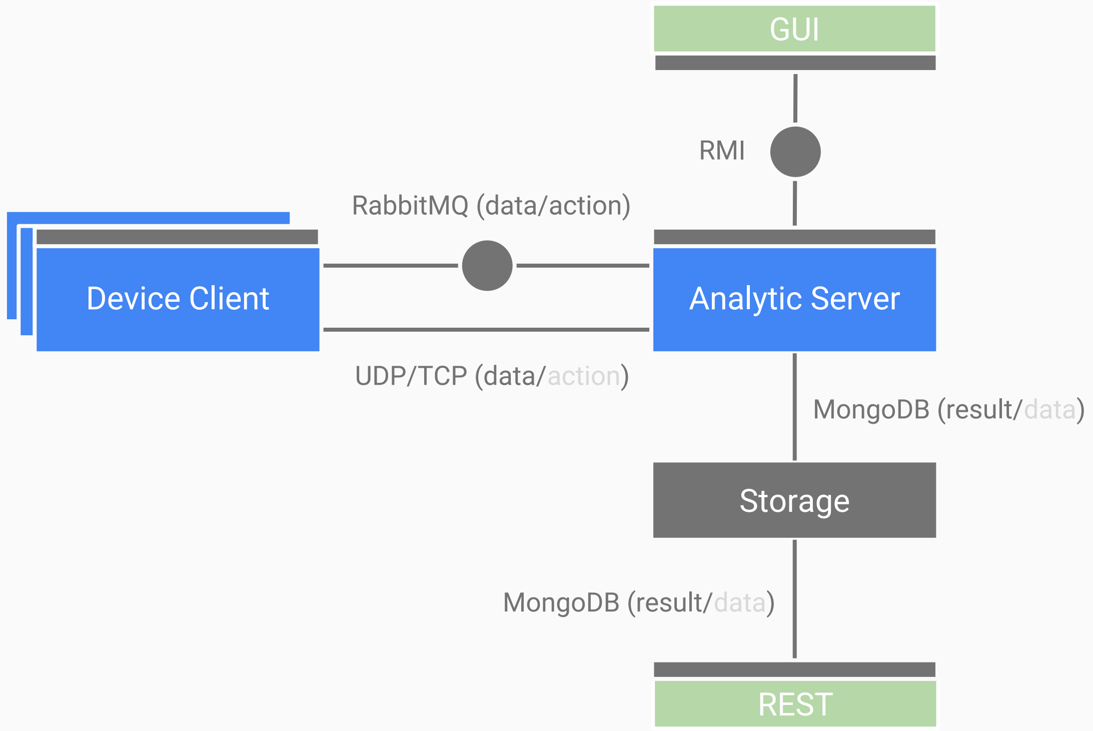
Smart Service Center

A Service of Meta Application

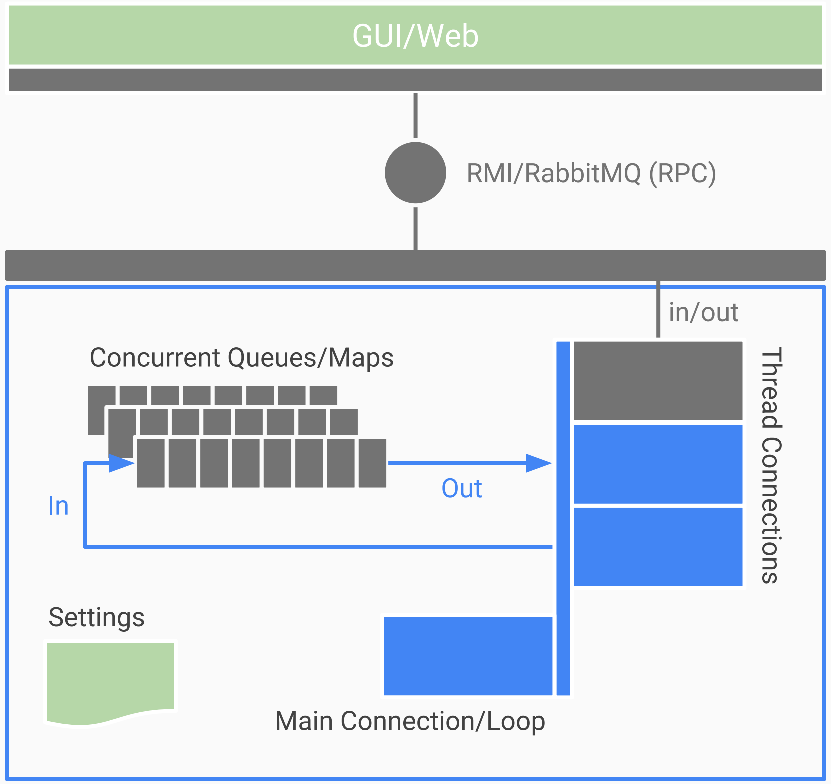
Goal


GUI
RMI
RabbitMQ (data/action)
Device Client
Analytic Server
UDP/TCP (data/action)
MongoDB (result/data)
Storage
MongoDB (result/data)
REST

Currently

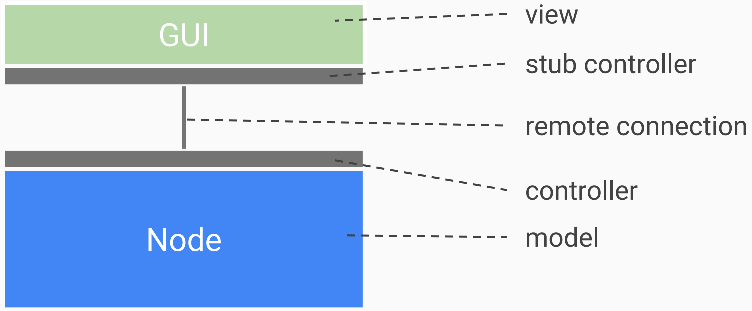

GUI
RMI
RabbitMQ (data/action)
Device Client
Analytic Server
UDP/TCP (data/action)
MongoDB (result/data)
Storage
MongoDB (result/data)
REST

Inside The Nodes

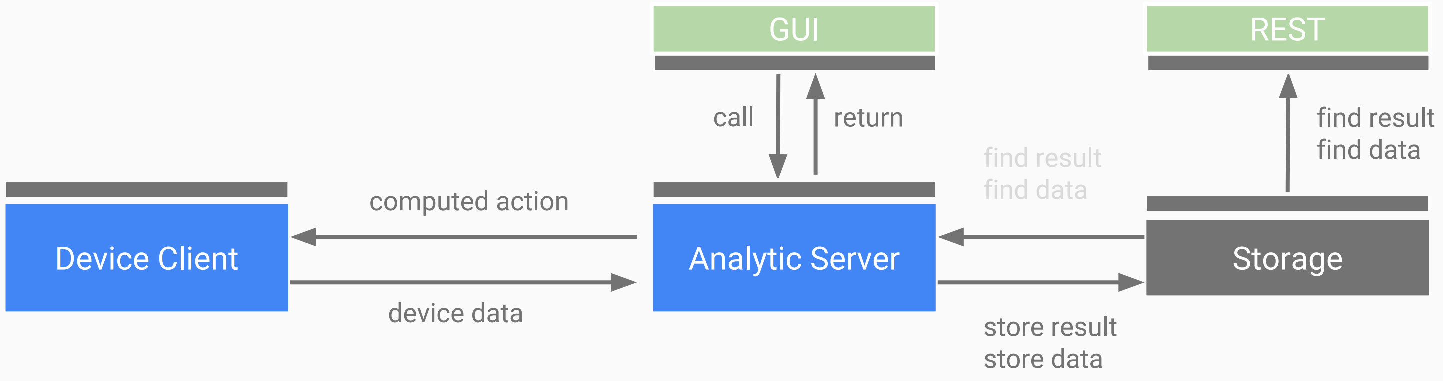


All Concurrent Queues and Maps are Serializable

View is strictly separated from the model and controller.



Main Idea



action { device_id, action_id, parameters }

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

data#sensor { sensor_id, timestamp, interval, signal, flag }

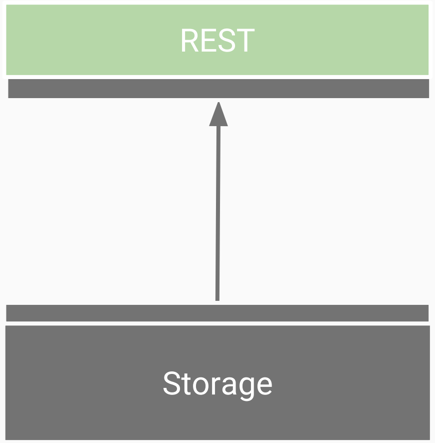
data#machine { machine_id, timestamp, interval, activity, handled, unhandled, flag }

REST Implementation

Data can be accessed through a REST API implemented using nodeJS with express and mongoDB driver.

Current urls/routes are available.

```
domain/<uuid_database>/results
domain/<uuid_database>/results?<device_id>[&<time_range>]
domain/<uuid_database>/results?<flag>[&<time_range>]
domain/<uuid_database>/results?<action_id>[&<time_range>]
domain/<uuid_database>/results?<time_range>
domain/<uuid_database>/results/<uuid_result>
```



Used Technologies

RabbitMQ/TCP/UDP (Publish/Subscribe) data exchange

RabbitMQ (routing) action command

MongoDB (database/connector) store/find results

NodeJS (service) REST implementation

RMI (GUI) a stub implementation

Some Remaining Problems

Security/Compression/Privacy/Load Attacks

Byzantine Generals Problem (partly solved using action weights)

Server Configuration/Start/Stop/Restart/Setup

Monitoring Truth

Device Registration

RabbitMQ and MongoDB Configuration

...

Any Questions