VAS Group

Masaccio - Monitoring for urbAn Safety with the IoT

Valentina Cecchini - mat.: 255596

Andrea Perelli - mat.: 254758

Stefano Valentini - mat.: 254825

Application Domain

The Architecture that we are proposing is designed to be **adaptable** to various situations. We decided to instantiate the problem to the monitoring of the **UnivAQ**'s existing buildings.

Our Architecture is designed to **manage** and **monitor emergencies** that may occur in the different places of UnivAQ's building, **manage accesses** to restricted areas and **perform analysis** on collected data.

Main services

The main services we want to provide are:

- Access control
- Security monitoring
- Alarm dispatching
- Smart Information Service
- Crowd Monitoring
- Data Storing and Analysis
- Attendance Registration

Design Decisions (1)

- **Data Acquisition:** we have chosen the **Publish/Subscribe** architectural pattern, in particular we have used **Apache Kafka.**

  Pros: *reliability*, *fault-tolerance*, *performance*, *scalability*.

- **Storing:** we decided to store the collected data in the UnivAQ's server. In particular, we organised them in two different DB: a **NoSQL DB (MongoDB)** and a **relational DB**.

  Pros: *scalability*, *dependability*, *costs efficiency*, *privacy/security*, *performance*.

Design Decisions (2)

- **Sensors and Actuators organisation:** we have chosen to organise the sensors in areas (classrooms, laboratories, …). Each sensor publishes the collected data on the topic of the specific area in which it is deployed.

  Pros: *scalability*, *decoupling*.

- **Visualisation and analysis:** A dedicated Kafka Client is responsible to read the incoming data from the area topics and transmit them in real time with a secure connection to a dashboard (through web-socket). The data can even be subjected to analysis and refactoring before being sent to the dashboard, the eventual alarms are displayed on the dashboard and double checked by the personnel before being forwarded.

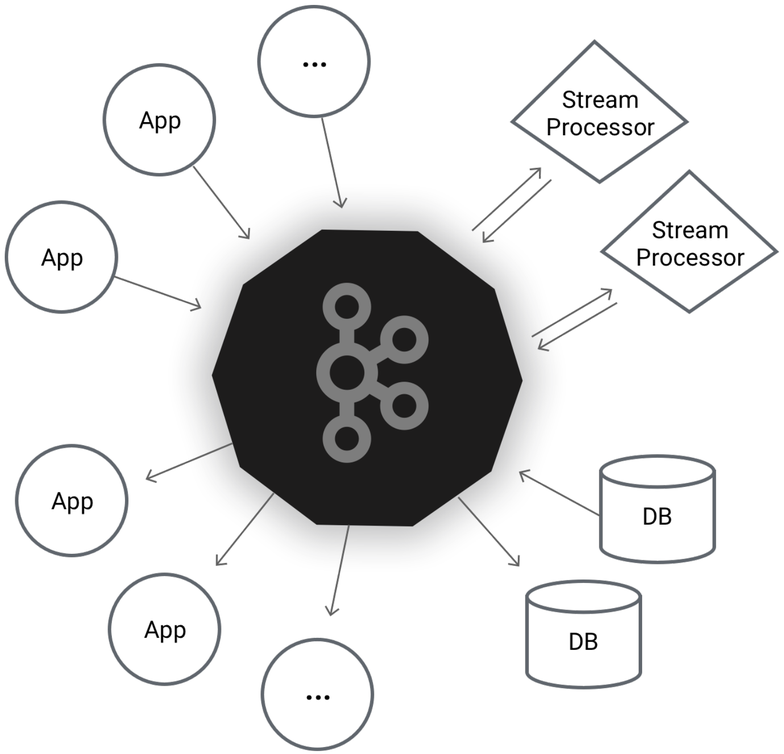
  Pros: *false alarms avoidance*, *real-time monitoring/graphical visualization*.

Architectural core: Apache Kafka

Apache Kafka allows to implement secure real-time streaming data pipelines that reliably get data between systems or applications.

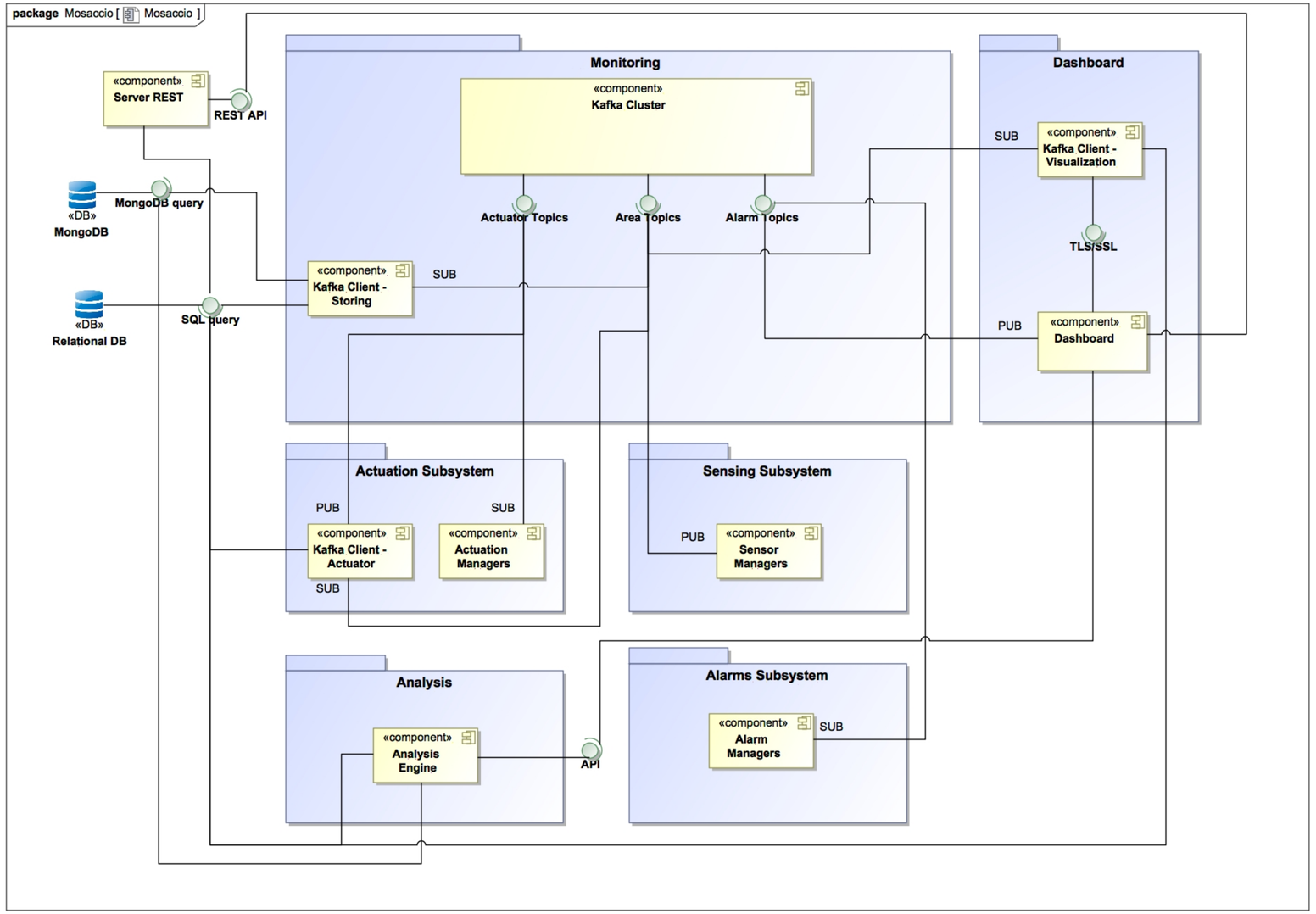
Kafka can be used to stream data, as a message dispatcher and as a storage system it also allow for a throughput of millions of records/second, replication, redundancy, scalability, decentralisation and fault tolerance techniques that are the crucial points of our system, such as:

- in case of a cluster fails another one takes its place, with all the updated data: every member of the cluster has all the replicated/up-to-date data;
- data can be saved for an arbitrary amount of time on the brokers;
- for a topic with replication factor N, it will tolerate up to N-1 server failures without losing any records committed to the log;
- Kafka replicates the log for each topic's partitions across a configurable number of servers. This allows automatic failover to these replicas when a server in the cluster fails, so messages remain available in the presence of failures.



System Architecture

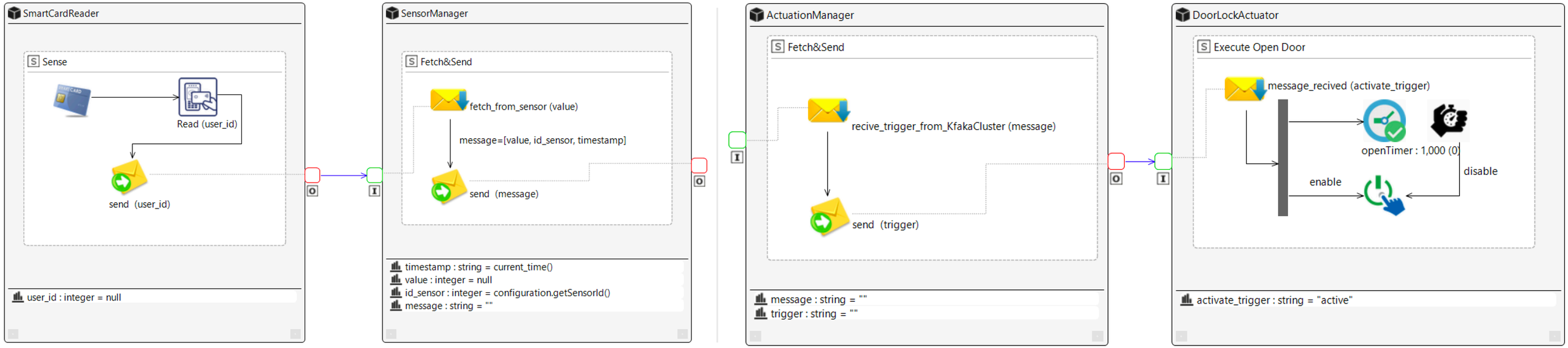
The system is composed by a **Sensor Network**, an **Actuators Network,** a **Kafka Cluster**, **two DB** (NoSQL and Relational), an **Administration Dashboard** and **three Kafka Clients** (**Storing**, **Visualize** and **Actuator**).



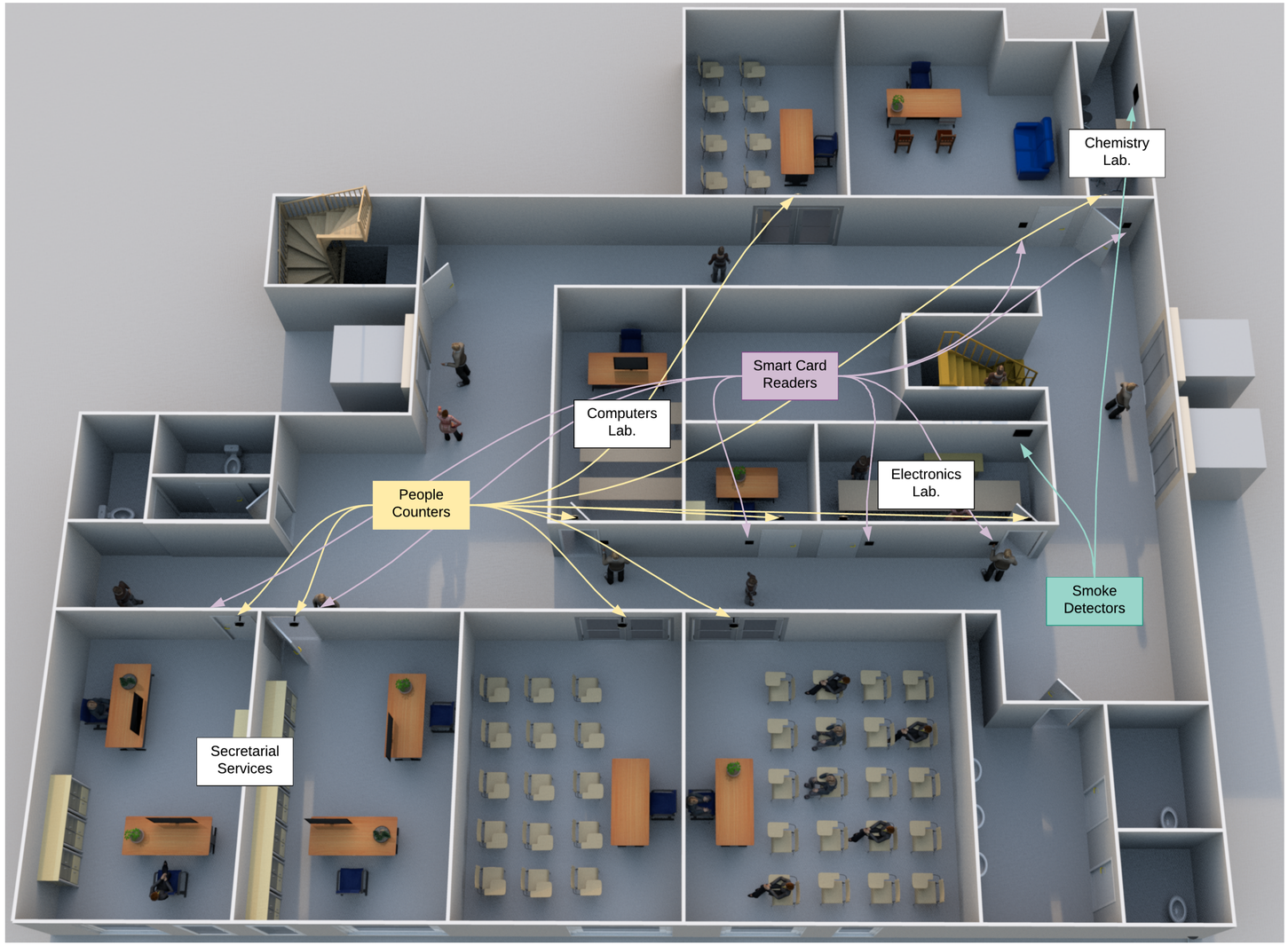
CAPS - SAML

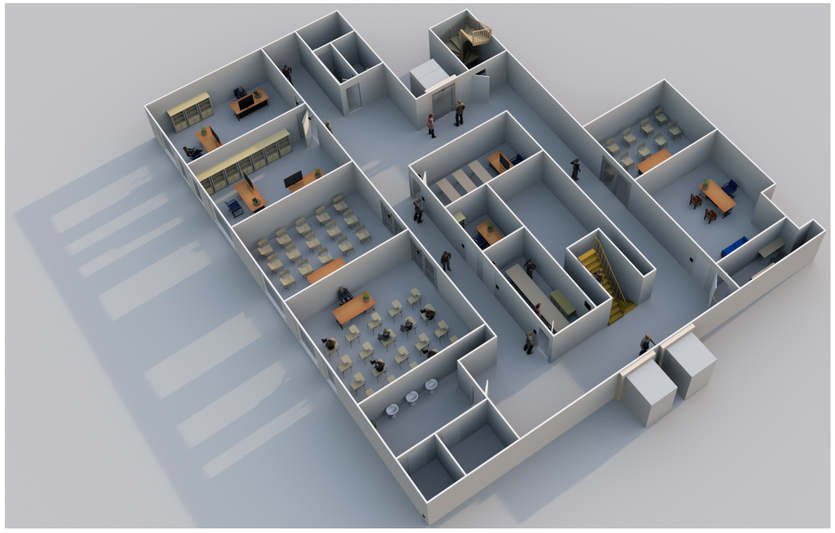
In our system a sensor simply senses the environment and broadcast the sensed value to its **Sensor Manager**. The Sensor Manager publishes the message on the respective **Kafka Topic** along with the sensor id and the current time stamp.

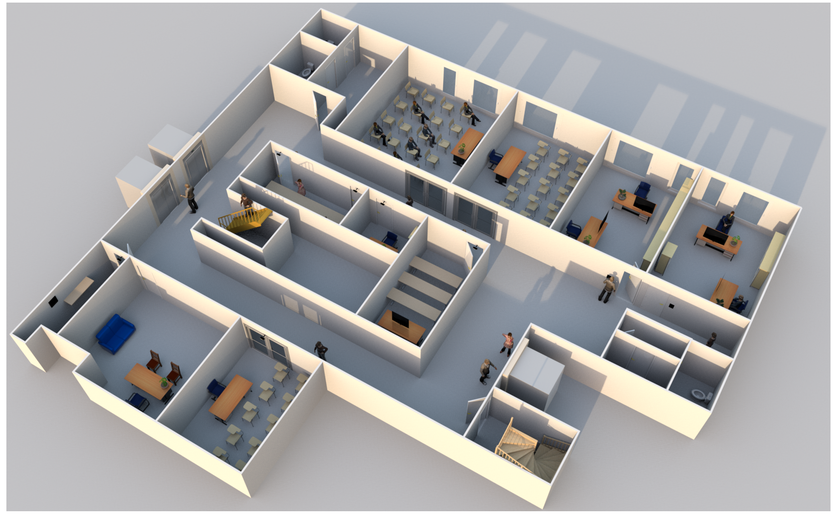
Similarly, when the **Actuation Manager** receives a triggering message (by listening on the respective **Kafka Topic**) it forwards it to its actuator, the actuator then performs its action.



CAPS - ENVML

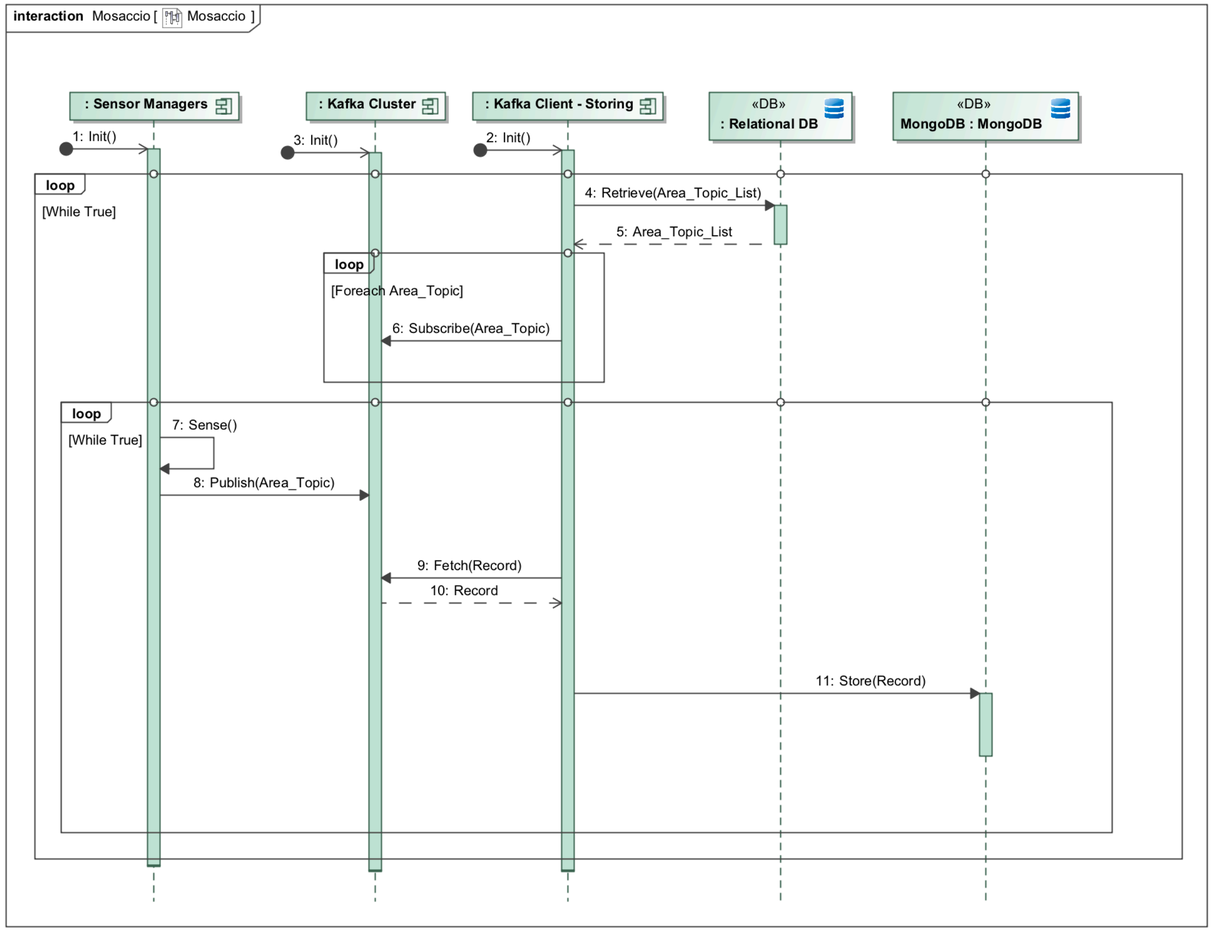

Chemistry Lab.
Smart Card Readers
Computers Lab.
People Counters
Electronics Lab.
Smoke Detectors
Secretarial Services





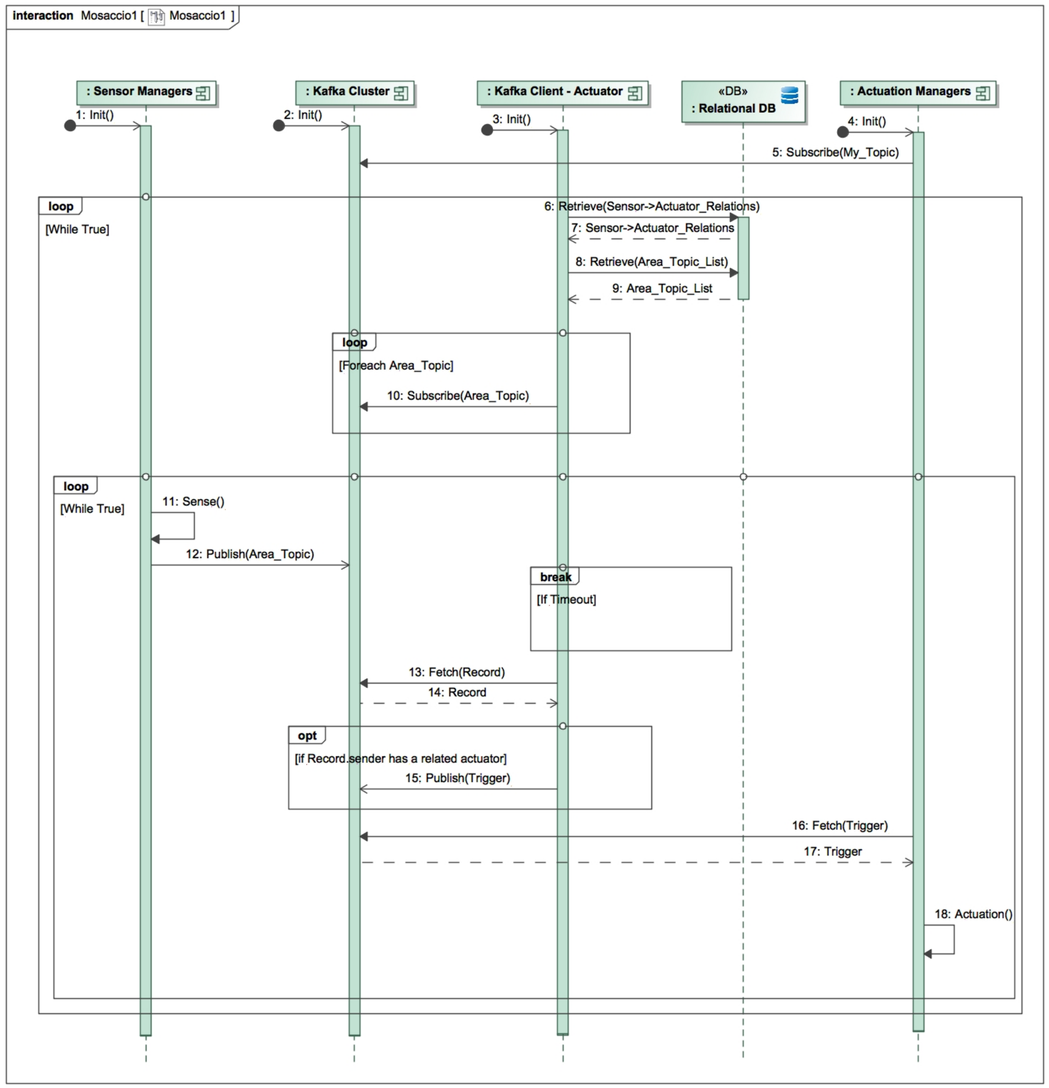
Implemented Service (D2) - Storing

This is the procedure that is responsible to the collection of the sensed data and the storing on the databases.
We decided to first implement this feature because it is the core functionality on which all the other ones are based on.
This allowed us to test the performance/usability/behavior of the system.



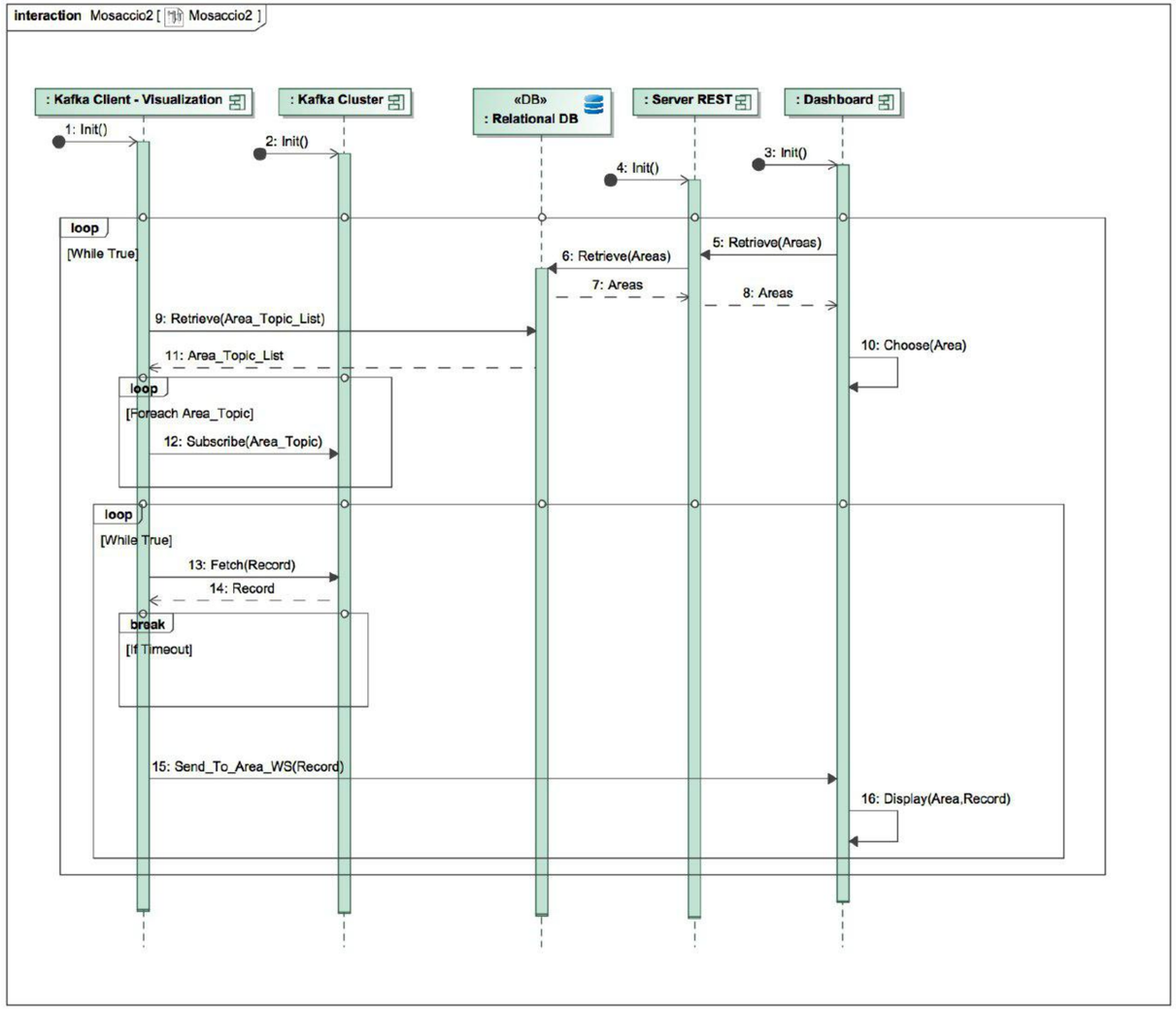
Implemented Service (D3) - Actuation

The actuation service consists of the triggering of some actuators when a certain sensor senses a certain event (e.g. a user slides his card on a card reader → the door opens).
The implementation of this service allowed us to further test our architecture, adding components and seeing how they can work together.



Implemented Service (D4) - Visualization

This service allows the personnel to visualize the data sensed by the sensors in a particular area.
In general, it allows to manage the system.
The implementation of this service allowed us to demonstrate how the data can be visualized and handled in real-time, even if other crucial components such as the Storing Client and the Actuation Client are not running.



Masaccio

Home

Choose an area

by clicking on an area you will be redirected to the list of the sensors that are deployed in that area

a1.1 a1.2 a1.3 a1.4 a1.5

file:///Users/valent0ne/Projects/MASACCIO/implementation/dashboard/index.html#

Masaccio

Home

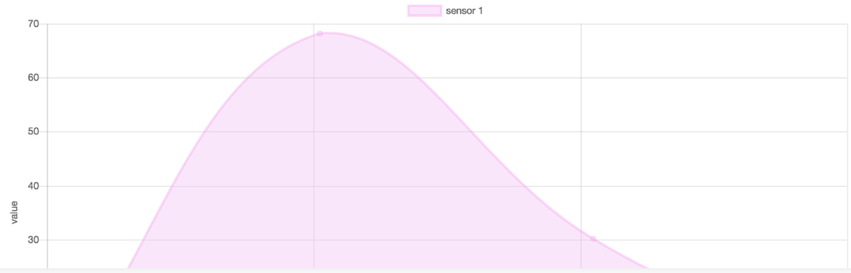
Area: a1.1

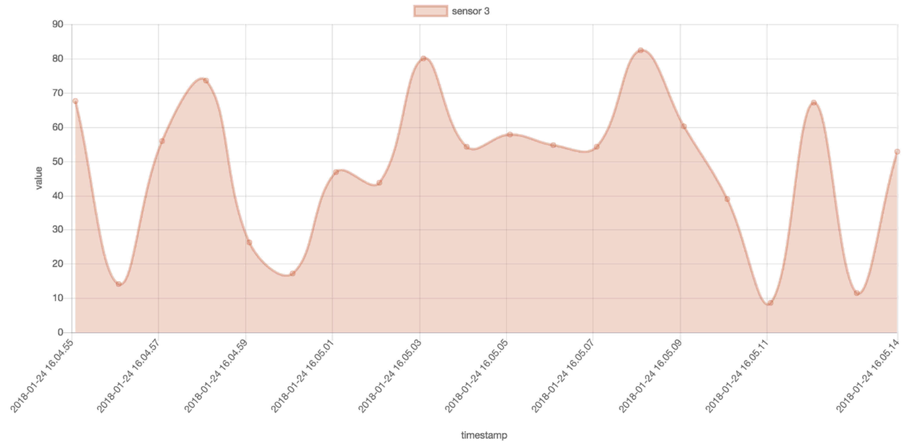
Description: this area describes a lecture room situated on the first floor of the Coppito 0 building.

Latitude: 40.435642

Longitude: 53.234176

Elevation: 654





Real-time sensor's readings

#	Sensor ID	Payload	Timestamp
69	1	17.56557183895725	2018-01-24 16.05.15
68	3	52.70817873619386	2018-01-24 16.05.14

Real-time sensor's readings

#	Sensor ID	Payload	Timestamp
83	3	43.66751851192085	2018-01-24 16.05.20
82	1	95.95218781349189	2018-01-24 16.05.20
81	2	30.842315473680436	2018-01-24 16.05.20
80	3	24.38350602777801	2018-01-24 16.05.19
79	1	42.38378202134771	2018-01-24 16.05.19
78	3	94.70859410723126	2018-01-24 16.05.18
77	1	62.138789830885244	2018-01-24 16.05.18
76	2	63.58717507586717	2018-01-24 16.05.18
75	3	61.61435610367828	2018-01-24 16.05.17
74	1	34.58066222693786	2018-01-24 16.05.17
73	3	89.62695461119833	2018-01-24 16.05.16
72	1	46.84544413147189	2018-01-24 16.05.16
71	2	37.16820646607195	2018-01-24 16.05.16
70	3	62.43567455166793	2018-01-24 16.05.15

Performance Analysis

This test was made on a network composed by three laptops.

From the table at the right we can assert that the numbers we got are much higher with respect to the customer's required numbers (40.000/hour <<< 240.000/13s → 17.000/s).

So, we can conclude that the system is fully capable of handling the requested amount of data traffic.

kind of operation	# of topics	# of messages per topic	# of total messages	time (seconds)	kind of message
Publish (produce)	6	40.000	240.000	13.817	String
Subscribe (consume)	6	40.000	240.000	15.341	String
Store (from String to .json)	-	-	240.000	real-time/ asynchronous	.json

Thank you for the attention

Questions?