IN2DREAMS

Contract No. 777596

INtelligent solutions 2ward the Development of Railway Energy and Asset Management Systems in Europe

D6.1 Analytics platform

DUE DATE OF DELIVERABLE: 30/04/2018

ACTUAL SUBMISSION DATE: 02/05/2018

Leader/Responsible of this Deliverable: Josselin Bousquières – Evolution Energie

Reviewed: Y

Document status		
Revision	Date	Description
1	15/03/2018	Description of the table of content
2	23/03/2018	Description of the parts 1, 2, 3.2, 3.3 and 6
2.1	28/03/2018	Addition of part 5
2.2	29/03/2018	Addition of part 3.1, minor corrections and consolidation of the document
2.3	06/04/2018	Review and corrections, addition of details in part 4
3	09/04/2018	Review and corrections, executive summary
4	16/04/2018	Minor corrections, review by TMT members
5	02/05/2018	Final version after TMT approval and Quality check

Project funded from the European Union's Horizon 2020 research and innovation program		
Dissemination Level		
PU	Public	X
CO	Confidential, restricted under conditions set out in Model Grant Agreement	
CI	Classified, information as referred to in Commission Decision 2001/844/EC	

Start date of project: 01/09/2017 Duration: 24 Months

Executive Summary

The objective of this document D6.1 is to describe the data analytics platform which is the infrastructure that complements the data storage facility provided by WP3 in IN2DREAMS project. This platform is the basis for further analytics algorithms (tasks 6.2 and 6.3) and application use cases development (task 6.4), keeping in mind that the final objective is to assist infrastructure managers and railway operators to select optimal strategies and resources in order to support in a cost effective and energy efficient manner railway applications.

To develop the analytics platform, the QMiner software was selected which is designed for scaling millions of instances on high-end commodity hardware, providing efficient storage, retrieval and analytics mechanisms with real-time response. Moreover, as the framework is written in C++ for lightweight data processing, it enables ubiquitous deployment for various use case scenarios. The powerful software architecture integrates connexions with communication platforms (WP2), analysis inside QMiner, data bases and visualisation application.

A use case with real data (Reims tramway use case coming from IN2RAIL project) has been selected in order to define how the data analytics platform will be used to support a real use case scenario and to gain insight into the main required functionalities which will be used and developed in the upcoming tasks (data pre-processing, forecast modelling, power system modelling and fault detection, and operation optimisation).

Designed architecture and selected components of QMiner offer modular structure with loosely coupled components, to achieve a high level of analytical infrastructure flexibility and scalability. In such way, the analytical infrastructure will be able to support a plethora of use cases and the complexity of the designed project in the following phases. Even if time is required to adapt the component and the models to the specifications of every use case which could be developed in the future, a high level of flexibility and scalability can be achieved with QMiner because it is open source and integrated with a message broker.

Abbreviations and Acronyms

Abbreviation	Description
API	Application Programming Interface
APS	Alimentation Par le Sol or Aesthetic Power Supply
DB	Database
EU	European Union
GA	Grant Agreement
H2020	Horizon 2020 framework program
IN2RAIL	Innovative Intelligent Rail
JS	JavaScript
JU	Shift2Rail Joint Undertaking
LCC	Life-Cycle Cost
WP	Work Package

TABLE OF CONTENTS

List of Figures

List of Tables

1 Introduction

1.1 Objective of the deliverable

Work Package 6 (WP6, "User Applications") of the IN2DREAMS program is divided into 4 complementary tasks related to applications of energy management for railway operations. Task 6.1 provides the analytics infrastructure that complements the data storage facility provided by WP3. This infrastructure uses QMiner (https://github.com/qminer/qminer/wiki), an opensource software. QMiner is an analytics platform for large-scale real-time streams containing structured and unstructured data.

The objective of this deliverable D6.1 is to install QMiner software and to determine the main functions of the analytics platform which will be used in the next tasks. As it was necessary to use real data in order to identify the main required functions for the future tasks, the installation was based on the Reims tramway data which were also used within the IN2RAIL project.

The architecture of the analytics platform is described in Section 2. Section 3 presents the Reims tramway data (description of the use case for the WP6, description of the data sets, data access and storage). Section 4 describes the choice of QMiner, the installation of the software and the main data analysis frameworks which will be used in the next tasks. Finally, the last Section describes the advanced visualisation application.

1.2 Inputs

ISKRATEL and EVOLUTION ENERGIE participated in the writing of this deliverable.

The work described in this deliverable required real data. Data of Reims tramway use case coming from the IN2RAIL project were used with the help of UNIBRI and ALSTOM who are partners in IN2RAIL. The Open Data Management (ODM) platform developed in the IN2RAIL project is used as input source for the raw data feeding the analytics platform based on QMiner which is developed in this document.

1.3 Main results

The QMiner software is a very suitable choice for the analytics platform as it is designed for scaling millions of instances on high-end commodity hardware, providing efficient storage, retrieval and analytics mechanisms with real-time response.

The installation of QMiner within the data analytics platform has been successful. The software can be used for further analytics algorithms (tasks 6.2 and 6.3) and application use cases development (task 6.4). Moreover, a visualisation application is integrated into the analytics platform to allow an easier access and a better understanding of the data and of the model results.

The work done is highly correlated to other WP: the use case for QMiner implementation (Reims tramway) was also used in WP5 as a data analytics scenario which will be used in the task 5.2 to define KPI's related to models which will be developed in the task 6.2. Moreover, QMiner is a highly flexible software which will

be able to integrate new data coming from WP3 and from other projects (IN2STEMPO for example) through connexions with the communication platform developed in WP2.

2 Data Analytics Architecture and Infrastructure

2.1 Architecture overview

In order to address the issues of the Work Package 6, a specific data analytics platform has been designed and deployed. This platform is based on open source technologies and is aimed to be flexible as possible in order to be adapted to different input data, to the different aims of the Work Package and to potential other tools for other aims.

An overview of this platform and its functioning is given in figure 2.1 (a more detailed version is given in section 2.2). The architecture is the following:

- Input data are obtained from other platforms (e.g. from the IN2RAIL platform, from the IN2DREAMS WP3 platform or directly from the sensors). Specific connectors should be developed at this point to connect to the input data and send them as a stream of raw data to a message broker.
- The message broker (apache Kafka) is the central brick of the platform since it communicates with all the elements and it allows 4 main features:
    - loose coupling between components;
    - easy replacement of each components (with new technologies or other more appropriate technologies) if needed;
    - easy adding of new components (for instance for tasks 6.3 and 6.4 where the detailed functioning is not yet known);
    - working on streamed, real-time data.

  We chose apache Kafka for its streaming quality, its rapidity and easiness to connect with different technologies (Node.js, Java, InfluxDB to name but a few). This Kafka broker is hosted on a Zookeeper server.
- Streamed data (raw and processed data, analytics) are persisted into a database, namely an InfluxDB database, which fits the need of rapidity in writing and reading large amounts of data.
- Finally, APIs are set up in order to give access to data to external users (for instance via a dashboard or for other automatic tools). Such APIs should have a secure access and only selected users and/or applications should be authorised to access it in order to guaranty the privacy and security of the data.

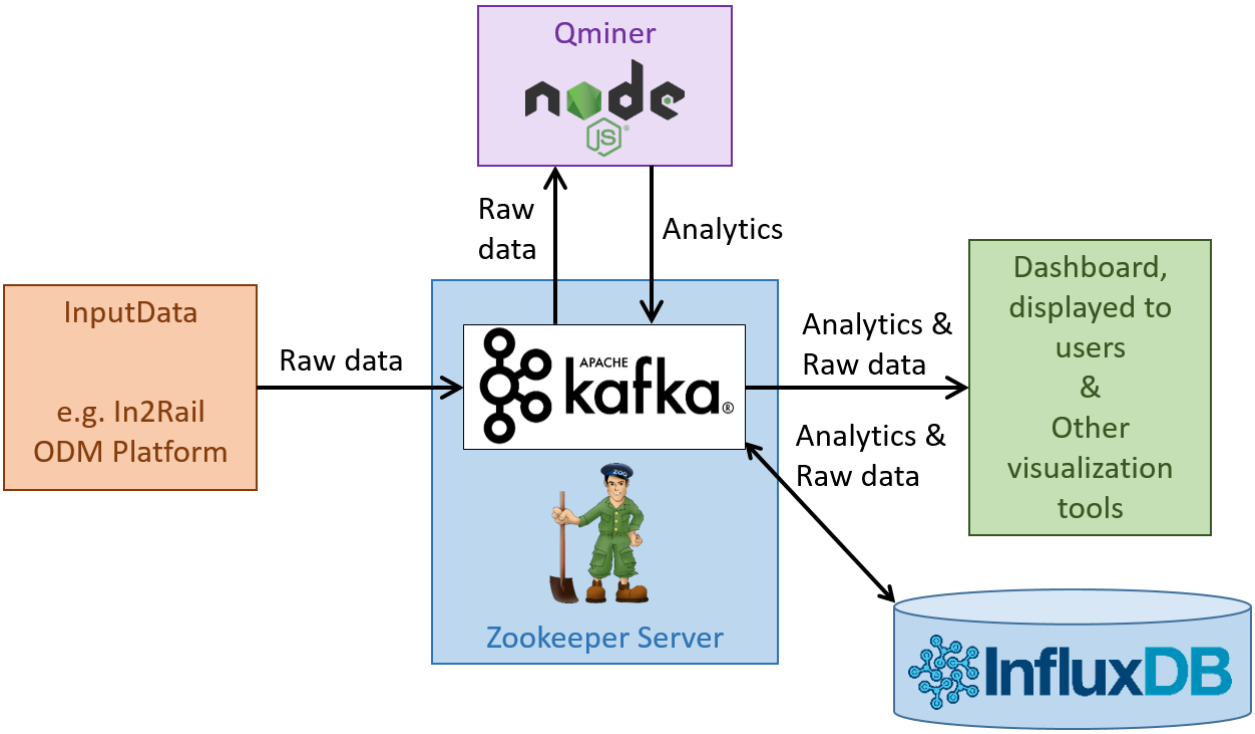

Figure 2.1: Simplified overview of the architecture

Based on this architecture and the coming tasks of the Work Package (tasks 6.2, 6.3 and 6.4), we defined the following data flow (see figure 2.2) that takes advantage of the flexible solution we designed. Note that the flow of data for task 6.4 is not defined yet, but the architecture with loosely coupled components and APIs will allow it to fit into the platform.

The data flow is the following: the work package has one input which are the raw data coming from instrumented trains and substations. These data are processed by different components (pre-processor, machine learning engine, algorithms to complete missing data, fault detection algorithms) and several output data are obtained. All of them are then re-usable in other components (whether internal or external to this platform). Examples of external components could be applications displaying to drivers or asset manager the historical and real-time electrical consumption of the train(s) as well as the forecasted consumptions. These data could also be used for decision support tools for these same users, giving them advices for the driving (drivers) or the management of the train line (asset manager).

The outputs are then the following:

- First, input raw data are pre-processed through corrections of offset or/and detection of outliers. This pre-processing is part of task 6.2. The first output data are then pre-processed data.
- Then, those pre-processed data are leading to 4 different data types:
    - In task 6.2, these data are analysed in order to get a forecast model (more specifically the parameters of this model);
    - And based on this model and the pre-processed data, the forecasted data are obtained.

- In task 6.3, fault-detection algorithms are applied on pre-processed data and alarms are obtained;
- Still in this task 6.3, in case only partial data are available (only low-level sensor, not measuring directly the voltage for instance), completed data are obtained through specific models.

- In addition, the output data obtained from 6.3 (i.e. completed data) are used to get a forecast model and forecasted data.
- Finally, the forecasted data are compared to pre-processed data to quantify afterwards the error in the modelling.

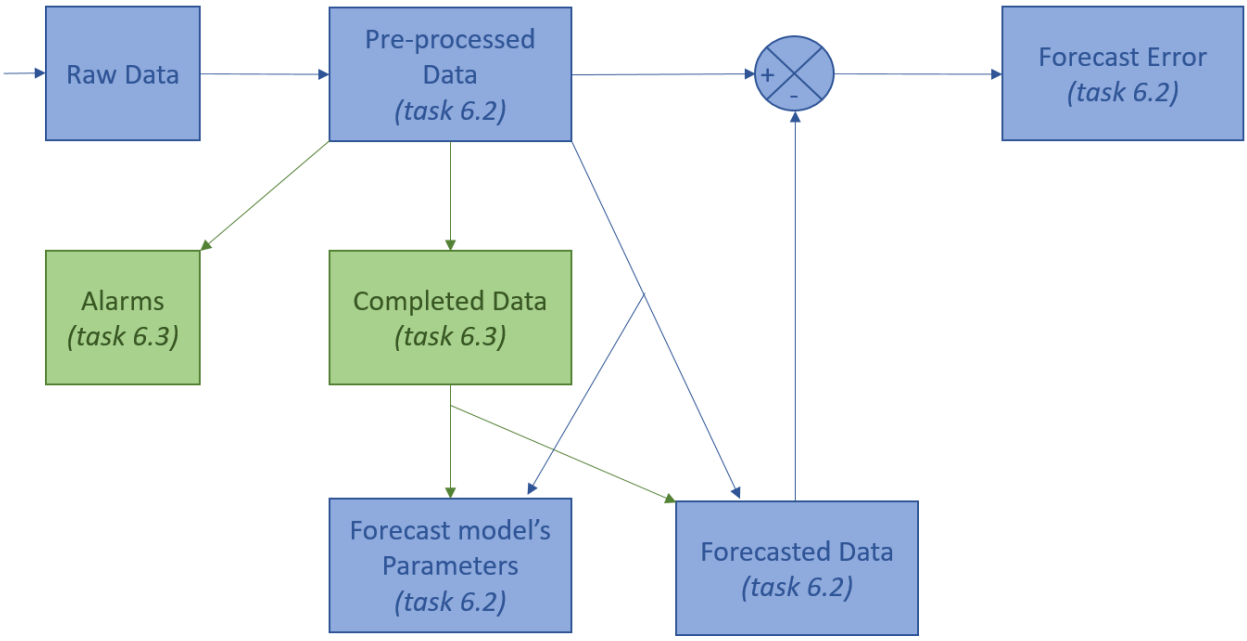

Figure 2.2: Data flow on the IN2DREAMS platform

2.2 Detailed architecture

Based on the previous data flow and the overview of the architecture, we provide below a detailed view of the architecture (Figure 2.3) and a list of the deployed tools (figure 2.4), taking into account their interfaces (input and output) with external tools or services. Based on these two figures, we can detail the functioning of the platform and its workflow:

1. The very first step consists in obtaining the historical data from the data sources, via specific connectors, and to put it all in the InfluxDB. This should normally be done only once (at the very beginning), but it may happen, during the day-to-day functioning that data are missed and that this specific connector is used once again to complete the database (DB).

2. Then, the resimulator is launched in order to feed the Machine Learning algorithms with all available historical data. Functioning is such that those historical data are taken from the influxDB (filled with

historical data in step 1. and sent exactly via the same stream as real-time data (thus allowing the analytics components to not apply a special treatment for historical data). Besides, the resimulator is highly configurable, allowing to send only a limited quantity of history or the whole data set. Note that, even when resending all the historical data as stream, time is compressed as the data are sent one after another, clearly without waiting the same time as in real-time between two data.

3. Once all the historical data have been sent to Kafka, it is possible to activate the real-time feed. A specific connector gets the data from the input data-source (via APIs or other technologies) and sends these data to the according Kafka stream.

4. The QMiner algorithms as well as the algorithms of task 6.3 are permanently linked to the Kafka streams. Thus, they are starting to process the data as soon as new data (historical then real-time) are available in the stream. And they are emitting processed data on other streams once the modelling/forecasting/analyses have been performed. For instance:

   a. QMiner takes raw data as an input;

   b. It emits pre-processed data as an output;

   c. Some of these pre-processed data are used by algorithms of task 6.3 to model complete data;

   d. And it emits these completed data on a new stream;

   e. These completed data are then used by QMiner to get a forecast model;

   f. QMiner emits the forecasted data of this model as well as the forecast error;

   g. Forecasted data (as well as pre-processed data and forecast error) are used by the dashboard to be displayed to the users.

5. In addition, the Kafka streams are always connected to a dedicated KafkaToInflux connector that takes care of the persistence of configured data-streams, from Kafka to InfluxDB.

6. Thus, the persisted - historical - data remain accessible via a secured API for future use.

7. External tools as well as a dedicated dashboard are connected to historical and real-time data (raw, pre-processed and processed data) to display them or use them for new algorithms.

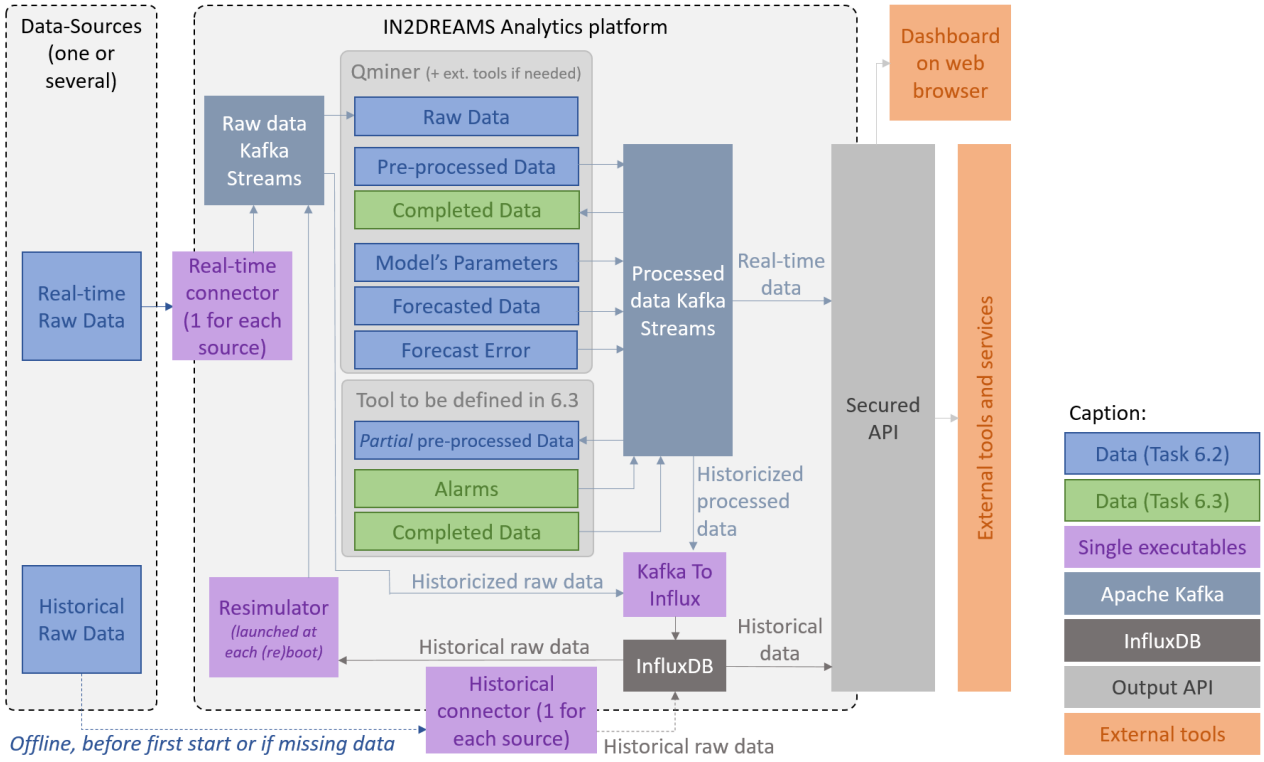

Figure 2.3: Detailed architecture of the IN2DREAMS Platform

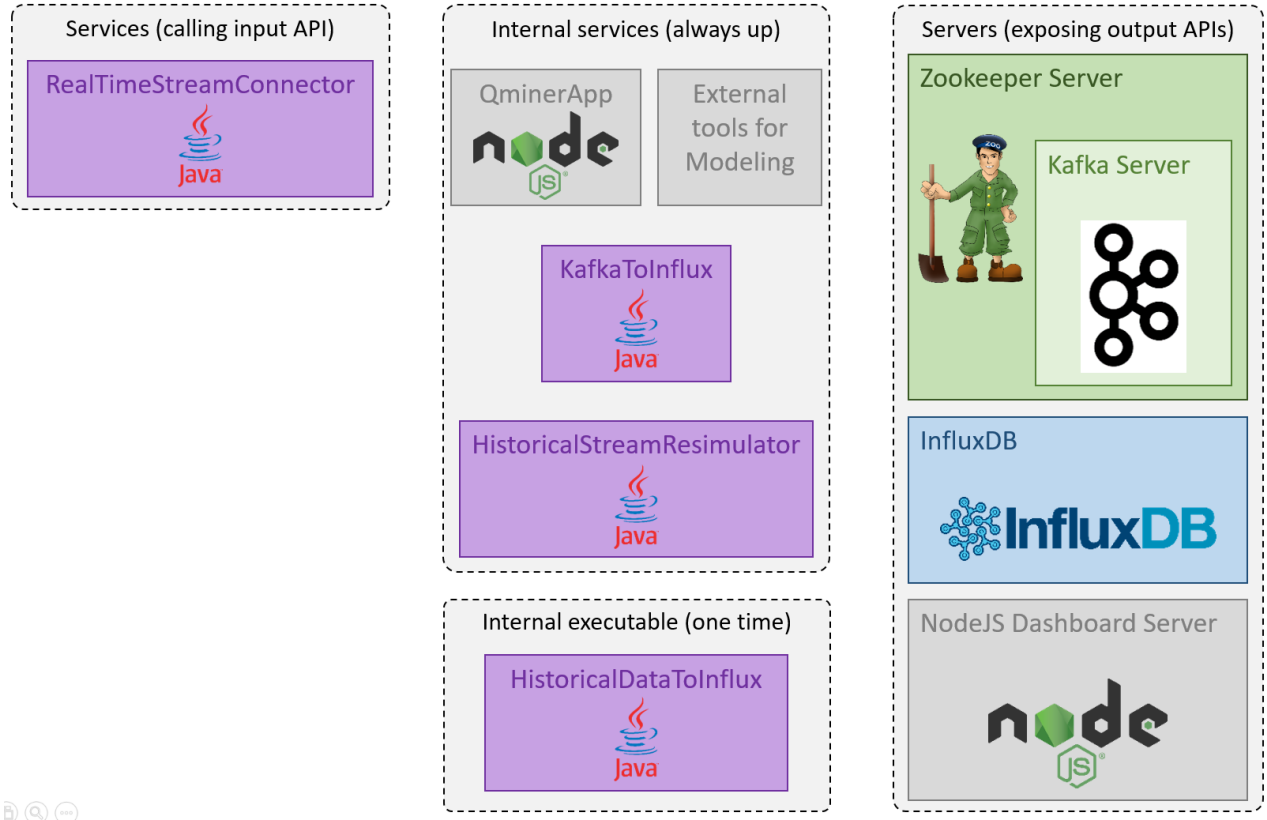

Figure 2.4: List of deployed components and their technology

2.3 Infrastructure and security

All the previously presented components are deployed on the same machine. For this project, we are working with a Microsoft windows environment, consequently all components are deployed as windows services. Once all deployed, the services should be started according to a particular schedule. There are many dependences and precedence issues to be taken care of. This leads to the following schedule to start the services (see Figure 2.5).

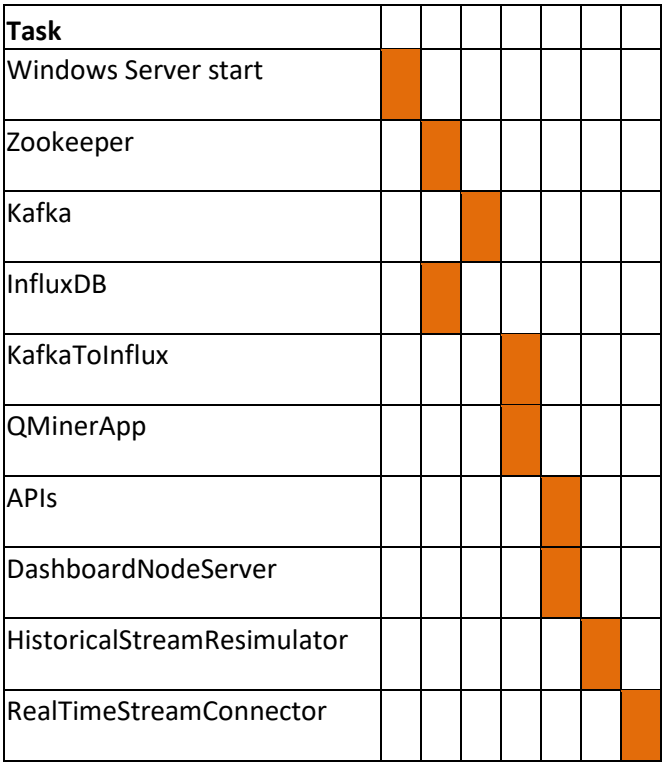

Figure 2.5: Schedule of the services

All these services should be monitored and automatically restarted in case of an unplanned stop. Of course, restarting automatically a service needs to ensure that the restart task fits into the schedule (all the services it relies on should still be running correctly and all services relying on this service should probably be restarted afterwards). Moreover, considering the substantial number of components, future work will focus on automating as much as possible the deployment of the whole platform.

All components are deployed on a secured server and only stakeholders from the WP6 shall have access to this server. The single points of access to the data for other external applications and services are APIs secured with SSL certificates. These secured APIs give access to real-time data (through Kafka streams) and historical data (through the InfluxDB).

3 The Reims tramway data

3.1 Use cases: Level 1 and Level 2

Data collected through smart metering make the infrastructure manager or the railway operator informed about energy behaviour, by way of combining real-time energy use and accurate billing. The consumer can act in two directions: energy saving, knowing the overconsumption sources, and billing reduction, switching the consumption to less expensive periods, when possible.

Smart metering easily enables these two steps, measure and analyse, which are very useful to introduce the DMAIC (Define, Measure, Analyse, Improve, Control) method for an energy management system, which supports and is in line with the ISO 50001 standard.

The IN2DREAMS project is designed to include multilevel data modelling, including building data infrastructure from the level of measurement, transportation integration and analytics, to business process modelling and data applications design. From the perspective of use case definition, this presents ~~tow~~ two very distinct levels of use case definition, referred to as:

a) Use Case Level 1
   Use case related to design and development of architecture for analytical infrastructure, including data analytics modelling and forecast scenarios. Use case Level 1 define design of forecast models to be deployed on analytical infrastructure. Use case can therefore support a plethora of use cases at the application level - Level 2, for end applications development.

b) Use Case Level 2
   Use case related to design and development of data applications, such as anomalies detection, asset management, demand side management, energy demand portfolio management, etc. These use case applications can apply data/functions derived from Use case level 1 as a feature or input for process modelling and application business function execution.

3.1.1 Use case Level 1: data analytics use cases

For data modelling and analytical infrastructure design, data from smart metering are collected from a commercially operated railway line under normal traffic conditions, as well as smart metering data from charging substations. The main objective is to observe energy data from two different scenarios:

- Data collected on train on-board unit;
- Data collected from stationary charging stations.

As real-case scenario, the data from the experimentation on the tram network of Reims (France) are used for this project.

On-train energy load forecast

Data analytics for the on-board unit will include design and development of a model for short-term forecast of energy consumption of the train (on a time horizon from few seconds to few minutes). The use case requires modelling the energy demand of a moving object, including features such as geolocation.

Substation charging forecast

Data analytics for charging substations will include mid-term energy demand forecasting (on a time horizon from 1 hour to one day). The use case requires modelling of the energy node (charging substation) as a stationary node in time and not as a moving object.

The two main subsets of analytical use cases described will be further elaborated in the following of the project development, although they serve as the main pillar for selecting methodologies and designing analytical architecture in this deliverable. To implement the analytical use case, we will use the QMiner stream analytics engine for designing and deploying a predictive model for load forecasting on grid substations and on-board trains consumption.

3.1.2 Use case Level 2: applications use cases

Level 2 use cases are focused on modelling domain-specific problems, such as: effective asset management, energy consumption optimisation, energy portfolio management and demand side management and virtualisation of demand, anomalies detection and others. While energy forecast values will serve as input data for final applications operations, analytical models designed can support a plethora of applications.

3.2 Data description

The experimentation of the tram network of Reims (France) has been started in 2016 within the IN2RAIL project framework. ALSTOM, who designed and commissioned the Reims tram, has instrumented two trains and one substation in order to collect real-time data on the energy consumption of the different systems on-board and trackside.

3.2.1 The Reims tram system

The Reims tram network crosses the city from north to south and is composed of one main line (A) starting from the terminus in the north and ending in the south and a secondary line (B) separating from the main in the south and terminating in the south-west (see Figure 3.1). The itinerary covers about 11 km of route and includes 23 passenger stations.

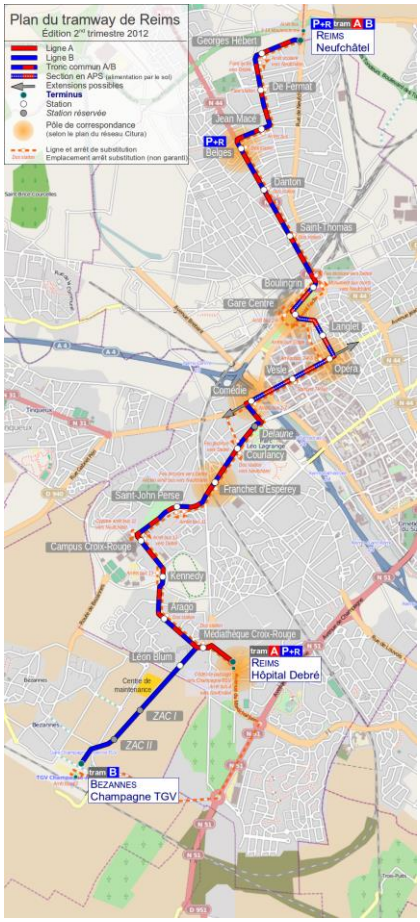

Figure 3.1: Itinerary map of the Reims tram system[1]

The trams on the line are of the ALSTOM Citadis 302 type and each tram is composed of 5 cars for a total length of 32.4 meters and a weight of 40.6 tons, when empty. Each tram can accommodate 206 passengers, can reach a maximum commercial speed of 70 km/h and is operating between 5h30 to 23h30 or 00h30 (depending on the week day) with a frequency of a passage every 5-6 minutes on the A line and every 20 minutes on the B line[2].

The train is powered through a pantograph from the overhead electric line during most of its route. Along 2 km in the city centre, to preserve the city historical view around the cathedral, the overhead line is replaced by ground-level power supply (APS, Alimentation Par le Sol), consisting of a third rail, within the running rails, supplying the power.

Along the line, seven power substations are distributed and are responsible for converting the high voltage from the public grid into the direct network feeding the trains.

[1] https://commons.wikipedia.org/wiki/File:Plan_tramway_reims.png
[2] http://www.citura.fr/fr/tram/74

3.2.2 The data set

Within the Reims tram experimentation, data from two instrumented trams and one substation are acquired and transmitted in real-time to the cloud, where they are made accessible to the IN2DREAMS consortium members. The actual delay of transmission can vary depending on the data transmission technology used.

The power injected into the tram and distributed into the different subsystems onboard the train can be monitored by sensors and meters (Figure 3.2). Most of the energy is used for the traction and breaking of the train. While the train is powered through the overhead line, when breaking, part of the train kinetic energy can be converted into electrical power and reused or stored within the train or reinjected through the pantograph into the tram line, in order to power other trains or ground equipment. However, this is not possible within the APS zone of the itinerary, where during the breaking phase all the kinetic energy in excess is dissipated into resistors.

The rest of the energy injected into the train is used to power auxiliary systems (air conditioning and heating, lightings, signalling systems, etc.) and to power filters and converters that ensure the functioning of these systems, or it is stored in on-board storage units for subsequent use.

The substation monitoring (started in 2017) provides information about the power injected into the tram line, which can be used by all the trains connected to the portion of the line where the substation is located.

Within the Reims tram experimentation, besides the measurements of current and voltage of several on-board subsystems, the position and speed of the trains are recorded in addition to some environmental measurements (e.g. inside and outside temperatures).

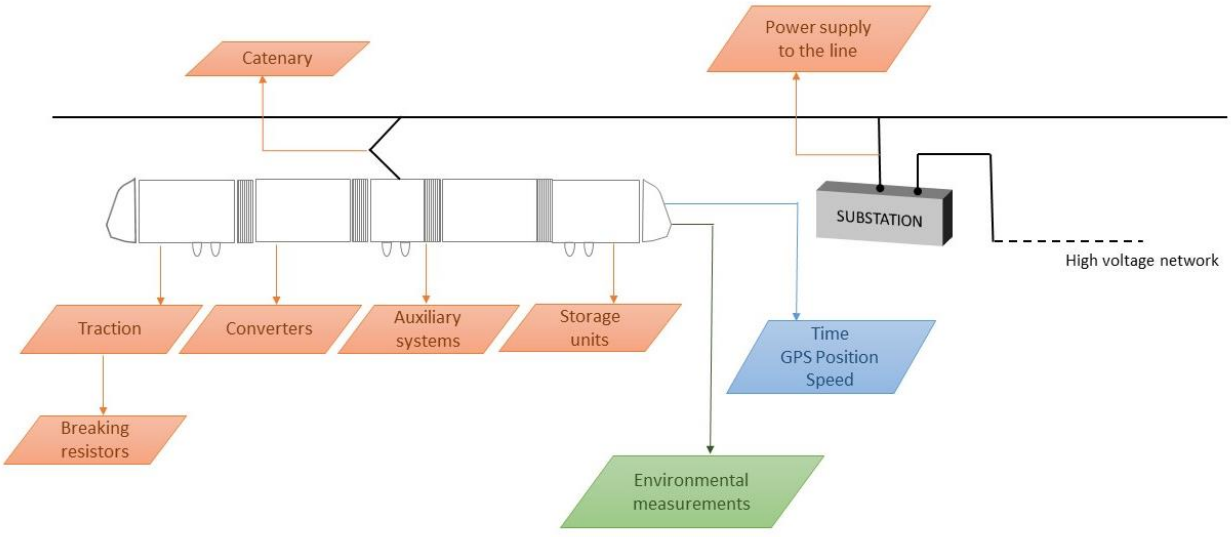

Figure 3.2 Schematic example of energy consumption monitoring for a tram network.

3.3 Data storage and access

Based on the architecture we defined in section 2, the Reims tramway data are composing the data-source referenced in Figure 2.3. Moreover, since data from 2 trams and a substation have been gathered on the same platform (the IN2RAIL ODM), this platform is the unique input data-source. Conforming to our architecture, these input-data are split in two types, real-time data and historical data:

- Historical data can be obtained in the form of csv files and a specific connector transforming these csv into InfluxDB time series has been developed and deployed on the IN2DREAMS platform;
- Real-time data are obtained via a MQTT broker. Thus, the first step consists in deploying a specific connector aimed at converting the MQTT stream into a Kafka one;
- As an alternative to this MQTT broker and to test the reliability of the chosen architecture as soon as possible in the project, it is also possible to not consider MQTT at first and to split the available historical data (4 years) into two sets: one set (the first 3.5 years for instance) actually considered as historical data and the rest (the last 0.5 years for instance) can be used to simulate 6 months of "real-time" data (with their correct arrival frequency). This approach will allow us to validate our technical choices for all types of data and to have an operational platform as soon as possible.

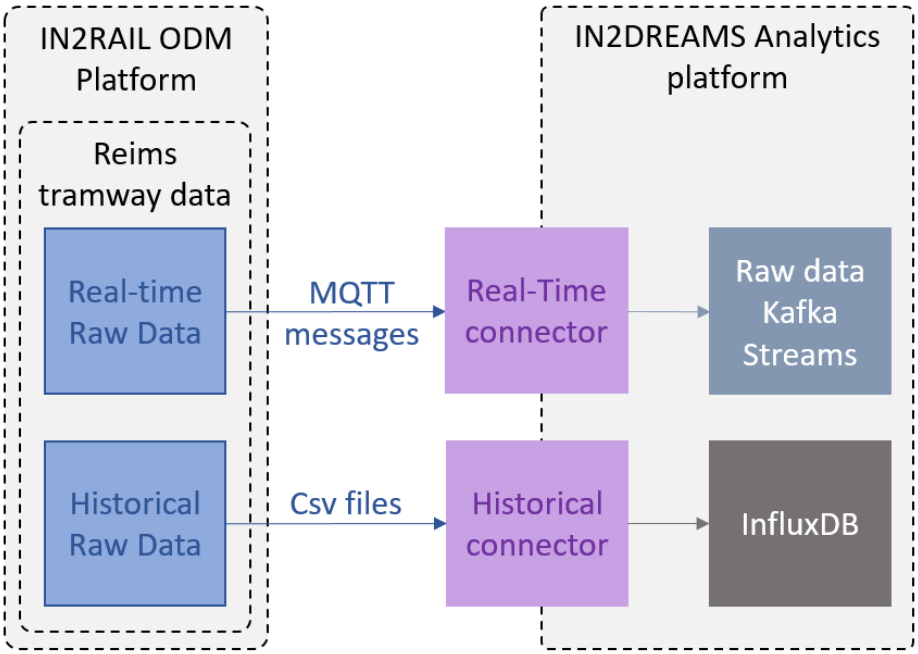

Figure 3.3: Input connection between Reims tramway data and IN2DREAMS platform

To conclude this section, it is important to note that, ultimately, the IN2DREAMS platform will be hosted on the same network as the platform hosting the Reims tramway data, thus simplifying the connection between the data-source and the analytics platform.

4 Data analysis and model prediction

4.1 QMiner

QMiner is an open source analytics module for performing large-scale data analysis in real-time. The main engine is written in highly effective C++ code, while the end functionalities are exposed using a JavaScript API as Node.js package (https://www.npmjs.com/package/QMiner). The design of the stream engine makes it very suitable for fast prototyping while still maintaining high performance, crucial for implementation in real industry settings. The main requirements for QMiner were driven by the need for fast development and prototyping of machine learning and artificial intelligence algorithms. This posed a challenge to operate on large data sets (hundreds of gigabytes) in real-time on high-end commodity hardware. These goals and constraints resulted in a unique set of features that were implemented in QMiner. Architecture design includes elements for efficient storage, online and real-time data processing, which makes QMiner a unique framework for processing streams of structured as well as unstructured data.

While QMiner is an open source project, it has an extensive knowledge base for its exploitation in streaming sensor data scenarios, being implemented in various similar project setups: from environmental intelligence, to intelligent traffic control systems and smart grid applications, as well as other fields such as streaming text analysis and anomaly detection.

The main advantage of QMiner in comparison with other open-source platforms is in its ability to support the deployment of analytical models on data streams. Moreover, the basic functions of QMiner are written in fast and reliable C++ native code, using high-performance computational libraries such as Intel MKL (Math Kernel Library). This enables QMiner to operate as a powerful and reliable stream processing engine, which, combined with using appropriate architectural design, can significantly reduce computational demands of a high number of updating models. Effective real-time processing capabilities and its platform independence, make QMiner ideal for edge computing use cases. In this project we will use QMiner to design a decentralised, light-weight, real-time, on-board analytical solution that will be possible to install directly on a train, on an average microcontroller unit. This solution will provide very short-term load forecasts (couple of seconds into the future), based on which conceptually higher-level use cases will be able to optimise the energy usage on the grid.

An important feature offered by QMiner is a built-in non-relational storage, which enables incorporating any type of data (structured or unstructured) while providing all the features for scaling and parallel computing. The built-in QMiner database is designed for implementation ~~in~~ in small modules and for fast prototyping, while for larger systems external data bases and big data technologies can be used. Using a non-relational database system constitutes an important advantage of QMiner, compared to other analytical frameworks.

Finally, since QMiner will be used as a building block, wrapped in the Node.js project, the approach will offer a large variety of different modules to connect to different streaming data sources (using Kafka broker as a loosely coupled integration). A combination of architectural design and loosely coupled components offers an effective analytical infrastructure for fast prototyping and testing, as well as robust operation.

4.2 Installation of QMiner

QMiner is platform independent and can therefore run on various platforms: Windows, Linux, iOS, Raspbian, etc. Since it is built as a Node.js module, we installed it by using the npm[3] package manager.

Detailed instructions for installation of QMiner can be found on the official website *(https://QMiner.github.io/setup/)* or GitHub page *(https://github.com/QMiner/QMiner)*.

Prerequisites:

node.js v9.x, v8.x, v7.x, v6.x, v5.x, v4.x and npm 5.3 or higher

To test that your node.js version is correct, run node --version and npm --version. The node.js version should be more than v0.12.

Windows: Visual C++ Redistributable Packages for Visual Studio 2015[4] is required. Download **vcredist_x64.exe** for node.js x64 or **vcredist_x86.exe** for node.js x86.

Installation:

```
npm install QMiner
```

Test installation:

```
node -e "require('QMiner'); console.log('OK')"
```

4.3 Data analysis framework

QMiner is used at the heart of our stream data analytics platform. Figure 4.1 represents the inclusion of QMiner into the analytical framework of this project. The design includes all the important components (from database to machine learning algorithms) to design the entire and fully functional application. A detailed description with examples of usage of the most important components is available in this section.

[3] https://www.npmjs.com/
[4] https://www.microsoft.com/en-us/download/details.aspx?id=48145

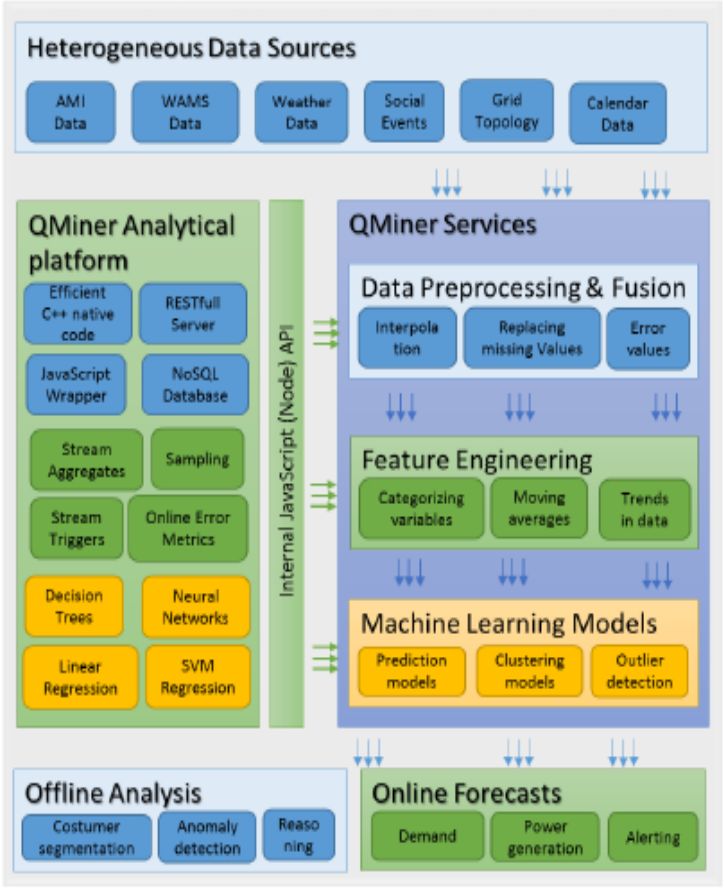

Figure 4.1: Modelling architecture with QMiner framework

Additionally, Figure 4.2 presents the conceptual analytical workflow, showing the main functionalities and tasks of each subsection within the workflow. This figure explains how the overall QMiner framework fits into the broader data analytics workflow.

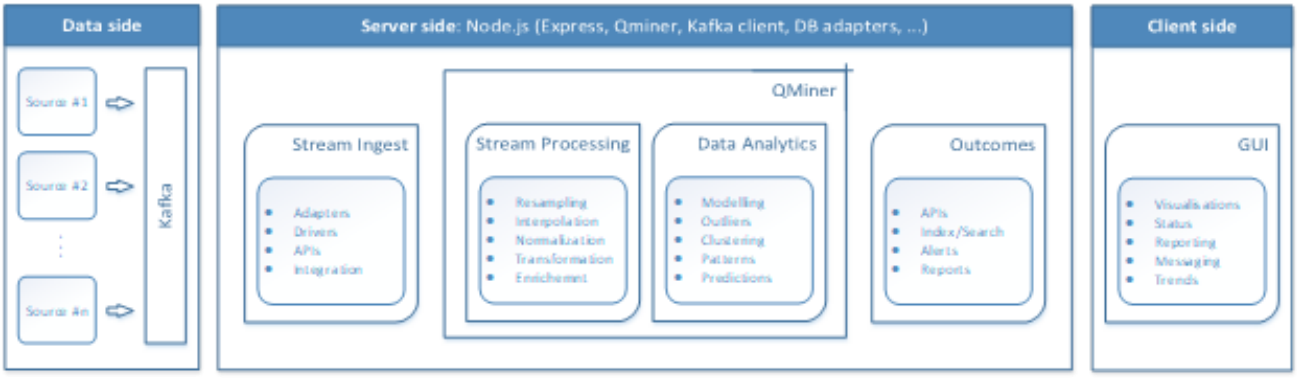

Figure 4.2: Full stack analytical workflow including QMiner as a Node.js package

4.3.1 Data processing

In order to process data streams in an online manner in QMiner, stream aggregates are used, which are stateful and typically do not interact with large sets of records - the streaming scenario entails operating at the end of the data stream (possibly with a short buffer). Their implementations reflect that, for example, the exponential moving average filter is implemented by using update formulas, with constant cost of updates - every record induces a fixed computational burden that does not change with time.

The main approach in utilising stream aggregates is as triggers, which can be added for store (when a new record is added), or to another stream aggregates. A stream pipeline can therefore be created, by adding triggers to many stream aggregates, where each depends on a previous (chaining stream aggregates). Combining stream aggregate filters and sets enable complex pipelines that contain many levels of data processing as well as modelling. Most commonly used stream aggregate functions in QMiner are presented in the following sections.

4.3.1.1 Base and Store

Stream aggregates can be managed by various objects in QMiner. Base[5] is the main object in QMiner that represents the database and monitors what stores exist and what aggregates are subscribed to what stores. Stores[6] are containers of records and are created through a base, either during initialisation of the base by using schema, or by using the createStore[7] function of the base.

Every time a new record is added to a store, the store triggers all its stream aggregates so that their states are updated. We can use store to actually store and manage records or we can use it only for schema (as a ghost store, which does not actually contain any records), which is used by the stream aggregates to create processing pipelines (aggregate set).

EXAMPLE: Creating a store with schema in base constructor.

```
// import qm module
var qm = require('QMiner');
// using the base constructor
var base = new qm.Base({
  mode: "createClean",
  schema: [{
    name: "Class",
    fields: [
      { name: "Name", type: "string" },
      { name: "StudyGroup", type: "string" }
    ]
  }]
});
base.close();
```

[5] https://rawgit.com/QMiner/QMiner/master/nodedoc/module-qm.Base.html
[6] https://rawgit.com/QMiner/QMiner/master/nodedoc/module-qm.Store.html
[7] https://rawgit.com/QMiner/QMiner/master/nodedoc/module-qm.Base.html#createStore

EXAMPLE: Creating a store with the createStore function.

```
// import qm module
var qm = require('QMiner');
// factory based construction using base.createStore
var base = new qm.Base({ mode: 'createClean' });
base.createStore([{
  name: "People",
  fields: [
    { name: "Name", type: "string", primary: true },
    { name: "Gender", type: "string", shortstring: true },
    { name: "Age", type: "int" }
  ],
  joins: [
    { name: "ActedIn", type: "index", store: "Movies", inverse: "Actor" },
    { name: "Directed", type: "index", store: "Movies", inverse: "Director" }
  ],
  keys: [
    { field: "Name", type: "text" },
    { field: "Gender", type: "value" }
  ]
}]);
base.close();
```

4.3.1.2 Windowing (moving window)

Time series window aggregator represents the window buffer for the computation of time series data aggregates. The window aggregator stores the values inside a moving window time frame and implements all the stream aggregate methods except getFloat() and getTimestamp(), which are implemented in signal processing aggregates presented in the next section. The time series window vector aggregate represents the values read from a time series window buffer.

EXAMPLE: Time series window aggregate example[8]. It is usually used in combination with other stream aggregates, such as moving averages.

```
var aggr = {
  name: 'TimeSeriesAggr',
  type: 'timeSeriesWinBuf',
  store: 'Heat',
  timestamp: 'Time',
  value: 'Celsius',
  winsize: 2000
};
base.store("Heat").addStreamAggr(aggr);
```

[8] https://rawgit.com/QMiner/QMiner/master/nodedoc/module-qm.StreamAggr.html#reset

4.3.1.3 Signal processing

Signal processing aggregate calculates the value of a function on a window of stream measurements and implements at least two methods: getFloat() method; exposes the value, and getTimestamp() method: offers the timestamp of the newest record in the aggregate's window buffer. In QMiner, the following list of signal processing aggregates are available:

- Time series tick aggregator: a data adapter that connects storage (record fields) and other stream aggregates as it exposes the numeric fields as a time series stream aggregate;
- Min aggregator: monitors the minimal value in the connected stream aggregator;
- Max aggregator: monitors the maximal value in the connected stream aggregator;
- Histogram aggregator: represents an online histogram. The aggregate defines an ordered set of points p(0),..., p(n) that defines n bins;
- Moving Average (MA): calculates the moving average value of the connected stream aggregator values;
- Exponential Moving Average (EMA): calculates the weighted moving average of the values in the connected stream aggregator, where the weights are exponentially decreasing;
- Moving correlation aggregator: calculates the moving correlation of the stream aggregators, that it is connected to;
- Moving covariance aggregator: calculates the moving covariance of the stream aggregators, that it is connected to;
- Moving variance: calculates the moving variance of the stream aggregator that it is connected to.

EXAMPLE: Moving average aggregator example [9] using the windows buffer (timeSeriesWinBuf) aggregate to compute the moving average.

```
// create a base with a simple store
var base = new qm.Base({
  mode: "createClean",
  schema: [
  {
     name: "Heat",
     fields: [
        { name: "Celsius", type: "float" },
        { name: "Time", type: "datetime" }
     ]
  }]
});

// create a new time series stream aggregator for the 'Heat' store, that takes the values from the 'Celsius' field
// and the timestamp from the 'Time' field. The size of the window is 1 day.
var timeser = {
  name: 'TimeSeriesAggr',
  type: 'timeSeriesWinBuf',
  store: 'Heat',
  timestamp: 'Time',
  value: 'Celsius',
  winsize: 86400000 // 1 day in miliseconds
};
var timeSeries = base.store("Heat").addStreamAggr(timeser);

// add a min aggregator, that is connected with the 'TimeSeriesAggr' aggregator
var ma = {
  name: 'movingAgerageAggr',
  type: 'ma',
  store: 'Heat',
  inAggr: 'TimeSeriesAggr'
};

var movingAverage = base.store("Heat").addStreamAggr(ma);
```

4.3.1.4 Resampler

Resampler is a stream aggregator that creates new values that are interpolated by using the values from an existing store. Type of interpolation can be either “linear”, “previous” or “next”.

[9] https://rawgit.com/QMiner/QMiner/master/nodedoc/module-qm.StreamAggr.html#reset

EXAMPLE: The example [10] creates a new resampler stream aggregator for the 'Heat' store that takes the values from the 'Celsius' field and the timestamp from the 'Time' field. The interpolated values are stored in the 'interpolatedValues' store. The interpolation should be linear and the interval should be 2 seconds.

```
var res = {
  name: 'resamplerAggr',
  type: 'resampler',
  store: 'Heat',
  outStore: 'interpolatedValues',
  timestamp: 'Time',
  fields: [{
    name: 'Celsius',
    interpolator: 'linear'
  }],
  createStore: false,
  interval: 2000
};
var resampler = base.store("Heat").addStreamAggr(res);
```

4.3.1.5 Merger

The main function of a Merger stream aggregator is to merge records from two or more stores into a new store depending on the timestamp, where type of interpolation can be either “linear”, “previous” or “next”.

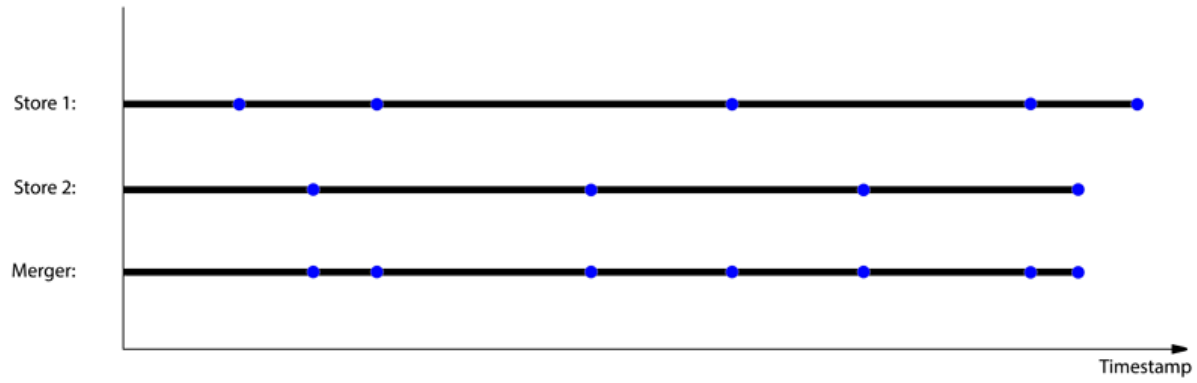

Figure 4.3: Visualisation of merger output, using previous value interpolator

[10] https://rawgit.com/QMiner/QMiner/master/nodedoc/module-qm.html#~StreamAggrResampler

EXAMPLE: The example[11] creates a new merger stream aggregator that mergers the records of the 'Cars' and 'Temperature' stores. The records are interpolated linearly and stored in the 'Merged' store.

```
var mer = {
  name: 'MergerAggr',
  type: 'merger',
  outStore: 'Merged',
  createStore: false,
  timestamp: 'Time',
  fields: [
    { source: 'Cars', inField: 'NumberOfCars', outField: 'NumberOfCars', interpolation: 'linear', timestamp: 'Time' },
    { source: 'Temperature', inField: 'Celsius', outField: 'Celsius', interpolation: 'linear', timestamp: 'Time' }
  ]
};
var merger = new qm.StreamAggr(base, mer);
```

4.3.1.6 Custom made JavaScript aggregates

Data pre-processing presents an essential step in the process of online data driven ~~data~~ model development. The main pre-processing operations in the modelling pipeline are: handling missing data, corrupt data handling, outlier detection, data segmentation, data normalisation …

QMiner enables to easily create a custom-made aggregate for a specific pre-processing task. Creating a custom-made aggregate in QMiner is demonstrated on a case of handling corrupt data, which are a cause of different errors that can occur during the measurement or transmission. For example, measurements can have physically non-valid values, such as negative consumption. The example of creating custom aggregate for corrupt data is presented below.

EXAMPLE: Custom made JS cleaning aggregate. The example from a prototype developed in MobiS project [12]demonstrates a simple custom-made JavaScript aggregate, specific for traffic prediction domain. In this specific example, traffic counter sensor reported speed of 999km/h, when there were no cars in the specific measuring intervals. By using custom made streaming aggregate, we can easily detect when the sensor counter value is 0, and change the speed to speed limit.

```
// If there is no cars, set speed to speed limit
trafficStore.addStreamAggr({
   name: "fixSpeedWhenNoCars",
   onAdd: function (rec) {
      if (rec.NumOfCars === 0) {
         rec["Speed"] = rec.measuredBy.MaxSpeed;
         rec["TrafficStatus"] = 1
      }
   },
   saveJson: function () { return {} }
})
```

[11] https://rawgit.com/QMiner/QMiner/master/nodedoc/module-qm.html#~StreamAggrMerger
[12] http://www.mobis-euproject.eu/

4.3.1.7 *Evaluation metrics*

Evaluation or Error metrics are used to evaluate and compare the fit and accuracy of forecasting models. Error metrics are essential tools for model evaluation and estimation as a metric for comparison of different options developed. Evaluation results therefore serve as a basic metrics for final model selection.

For choosing the appropriate error measure, there are several regressions (e.g. mean absolute error, mean absolute percentage error, mean error, mean square error, R2 score, root mean square error) as well as classification metrics (e.g. precision, accuracy, recall, f1) built in QMiner's library[13]. Most are implemented recursively, meaning that we can use them both in classical offline mode, or as online streaming aggregates where the metric is updated with each new streaming record (prediction).

EXAMPLE: Online regression error metrics as streaming aggregates. Example from MobiS project[14].

```
// create online regression metric instances
var mae = new analytics.metrics.MeanAbsoluteError();
var mse = new analytics.metrics.MeanSquareError();
var rmse = new analytics.metrics.RootMeanSquareError();
var r2 = new analytics.metrics.R2Score();

// create custom aggregate for updating evaluation models
trafficPredictionsStore.addStreamAggr({
   name: "evaluation",
   onAdd: function (rec) {
      mae.push(rec["true"], rec["predicted"]);
      mse.push(rec["true"], rec["predicted"]);
      rmse.push(rec["true"], rec["predicted"]);
      r2.push(rec["true"], rec["predicted"]);
   },
   saveJson: function () { return {} }
})

// access errors
mae.getError();
mse.getError();
rmse.getError();
r2.getError();
```

[13] https://rawgit.com/QMiner/QMiner/master/nodedoc/module-analytics-metrics.html
[14] http://www.mobis-euproject.eu/home

EXAMPLE: Example of online classification error metrics as streaming aggregates[15].

```
// create classification predictionCurve instance
var classificationScore = new analytics.metrics.ClassificationScore();

// create custom aggregate for updating evaluation models
eventsPredictionsStore.addStreamAggr({
   name: "evaluation",
   onAdd: function (rec) {
      classificationScore.push(rec["true"], rec["predicted"])
   },
   saveJson: function () { return {} }
})

// access errors (accuracy, precission, recall, f1)
classificationScore.accuracy();
classificationScore.precision();
classificationScore.recall();
classificationScore.f1();
```

4.3.2 Forecast modelling

4.3.2.1 Feature extractors

To represent the data into machine readable format, we have to map data records into linear algebra vectors or matrices applicable for machine learning algorithms. Algorithms that transform records into vectors (or matrices) are called feature extractors[16] in QMiner. Several feature extractors can be combined into objects referred to as feature spaces[17]. A feature space object extracts a larger vector from a record by concatenating feature vectors obtained from all its feature extractors. Its main obligation is bookkeeping of global vector and names of individual features. The list of feature extractors in QMiner can be found in

[15] http://www.mobis-euproject.eu/home
[16] https://rawgit.com/QMiner/QMiner/master/nodedoc/module-qm.html#~FeatureExtractor
[17] https://rawgit.com/QMiner/QMiner/master/nodedoc/module-qm.FeatureSpace.html

Name	Type	Description
constant	module:qm~FeatureExtractorConstant	The constant type. Adds a constant value as a feature.
random	module:qm~FeatureExtractorRandom	The random type. Adds a random value as a feature.
numeric	module:qm~FeatureExtractorNumeric	The numeric type. Adds the numeric value as a feature.
categorical	module:qm~FeatureExtractorCategorical	The categorical type.
multinomial	module:qm~FeatureExtractorMultinomial	The multinomial type.
text	module:qm~FeatureExtractorText	The text type. Creates the bag-of-words text representation.
join	module:qm~FeatureExtractorJoin	The join type.
pair	module:qm~FeatureExtractorPair	The pair type.
jsfunc	module:qm~FeatureExtractorJsfunc	The jsfunc type. Allows creating a custom feature extractor.
dateWindow	module:qm~FeatureExtractorDateWindow	The date window type.
sparseVector	module:qm~FeatureExtractorSparseVector	The sparse vector type.

Table 4.1: QMiner primary feature extractors

EXAMPLE: In order to be able to do modelling and prediction, feature vectors have to be constructed first. Which features will the model use, depends strongly on the data source. This is solved in a way, that for each data workflow we define a different set of features, attached to different extractors. An example for the traffic loop sensor from the MobiS project data sources can be seen below.

```
// define feature space
var ftrSpace = analytics.newFeatureSpace([
  { type: "numeric", source: testStoreResampled.name, field: "Speed" },
  { type: "numeric", source: testStoreResampled.name, field: "Gap" },
  { type: "numeric", source: testStoreResampled.name, field: "Occupancy" },
  { type: "numeric", source: testStoreResampled.name, field: "TrafficStatus" },
  { type: "numeric", source: testStoreResampled.name, field: "Ema1" },
  { type: "numeric", source: testStoreResampled.name, field: "Ema2" },
  { type: "multinomial", source: testStoreResampled.name, field: "DateTime", datetime: true }
]);
```

4.3.2.2 Linear Regression

Linear regression is a well-established approach in machine learning. It does not demand large computational resources and usually yields to surprisingly good results (with regard to model complexity). This stream aggregator computes a simple linear regression given two streams and represents variates (input) and covariates (output).

In QMiner, recursive linear regression [18] is implemented, which is the online (incremental) equivalent of the basic version and allows us to update the model at each new record. By updating a model with each new

[18] https://rawgit.com/QMiner/QMiner/master/nodedoc/module-analytics.RecLinReg.html

record, we can avoid known problems related to streaming data, such as concept drift, where there is a significant change in the system, which traditional offline model could not adapt to.

Pros: Since the method is light weight and is capable of incremental learning, it is very suitable to the cases requiring many models to be computed or when low computational capabilities are available (such as microcontrollers), which is very useful for decentralised systems and edge processing. Another advantage is that (almost) no parameters need to be optimised for the model to produce good results.

Cons: Since it is a linear model, it can perform a bit worse than other non-linear models in highly dynamic processes. In such cases, we need to consider the trade-off between the model accuracy and the model complexity, which is related to computational speed. By its nature, linear regression only looks at linear relationships between dependent and independent variables and assumes that the data are independent. If the data are not independent, the model can sometimes yield inaccurate results. Another drawback of the linear regression method is that outliers can have huge effects on the regression.

EXAMPLE: Traditional (offline) usage of recursive linear regression method in QMiner.

```
// Linear regression constructor
var linreg = analytics.newRecLinReg({ "dim": ftrSpace.dim, "forgetFact":1.0 });
// Learning phase
linreg.learn(ftrSpace.ftrVec(Training[trainRecId]), val.Value);}
// Prediction phase
var prediction = linreg.predict(ftrSpace.ftrVec(val));
```

EXAMPLE: For online usage, in QMiner we can create a stream aggregate which listens to all new records (trigger) and updates the model in case of new data added to the data store.

```
Training.addTrigger({
        onAdd: function (val) {
                // test prediction and remember it for later
                var prediction = linreg.predict(ftrSpace.ftrVec(val));
                Training.add({ $id: val.$id, Prediction: prediction });
                // update model
                var trainRecId = Training.getStreamAggr("delay").first;
                if (trainRecId > 0) { linreg.learn(ftrSpace.ftrVec(Training[trainRecId]), val.Value); }
});
```

4.3.2.3 Support Vector Regression (SVR)

In machine learning, support vector machines (SVMs, also support vector network) are supervised learning models with associated learning algorithms that analyse data used for classification and regression analysis. An SVM model is a representation of the examples as points in space, mapped so that the examples of the

separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall[19].

The Support Vector Regression (SVR) uses the same principles as the SVM for classification, with only a few minor differences. The model produced by support vector classification (as described above) depends only on a subset of the training data, because the cost function for building the model does not care about training points that lie beyond the margin. Analogously, the model produced by SVR depends only on a subset of the training data, because the cost function for building the model ignores any training data close to the model prediction. However, the main idea is always the same: to minimise error, determining the hyper plane which maximises the margin, keeping in mind that part of the error is tolerated[20].

Pros: Capable of modelling non-linear and dynamic processes well. Online model, capable of incremental learning. They are also appropriate for large volume of multivariate streaming data, but enough data have to be provided.

Cons: Computationally intense (slow prediction computation). There are usually also several parameters that have to be carefully optimised for a good prediction performance.

EXAMPLE: SVM model usage in QMiner[21].

```
//SVM training set
trainSet = svm.newTrainSet();
for (var i = 0; i < recs.length; i++) {
  trainSet.add(ftrSpace, recs[i], recs[i].value, true);
}
//SVM parameters
var svmParam = { c = 1.0, j = 1.0, eps= 0.1 };
//Train Model (learn)
var svmRegModel = svm.trainRegression(trainSet, svmParam);
```

[19] https://en.wikipedia.org/wiki/Support_vector_machine
[20] http://www.saedsayad.com/support_vector_machine_reg.htm
[21] https://rawgit.com/QMiner/QMiner/master/nodedoc/module-analytics.SVR.html

4.3.2.4 K nearest neighbours

K nearest neighbours (k-NN) is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until request. The k-NN algorithm is among the simplest of all machine learning algorithms, used both for classification and regression. The result is a collection of k sets of records, which are grouped by the similarity of their features. KMeans for regression problems is based on k-nearest neighbours, where the target is predicted by local interpolation of the targets associated to the nearest neighbours in the training set[22].

Pros: It is a relatively simple and light weight method that is usually better than baselines, but not better than more complex methods. It is a non-linear method that can find some patterns in historical data that linear models cannot. It is very robust to noise (especially if we use inverse square of weighted distance for computing distance).

Cons: One major drawback for our use case is that k-NN method is not online, meaning that it has to iterate trough a collection of records every time we want to make a prediction and can become slow with time. Therefore, we use a buffer with a certain number of the last records, to avoid this problem. It is still a relatively simple method that can perform with good results, but more complex methods usually find more accurate results.

4.3.2.5 Random Forests and Incremental Decision Tree

Random forests or random decision forests are a system of learning methods for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees[23].

By default, the Random forest algorithm is not an online algorithm and can take a lot of time to train and update the model. There are some variations that can reduce learning computational time by optimising input data, such as Very Fast Decision Trees learner (VFDT). The VFDT algorithm uses only a small sample of the available data instances when choosing the split attribute. This makes it suitable for processing data streams where the whole data set is not available or is too large to be stored in memory. To determine the number of examples needed to split a node, the algorithm uses the Heoffding or Chernov bound which guarantees that, with certain confidence, the attribute it has chosen is the correct one.

In QMiner, an improved version of this algorithm is in development, which will enable incremental learning (online model). This algorithm will be used and tested in the future implementations and tests of prediction

[22] https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
[23] https://en.wikipedia.org/wiki/Random_forest

workflows. For the exploratory analysis phase, we will use a standard version of the random forest model from Scikit-learn Python library [24][25].

Pros: Since Random Forests models have very few parameters to tune and can be used quite efficiently with default parameter settings (i.e. they are effectively non-parametric) they are a good choice to use as a first cut when the underlying model is not known, or a decent model needs to be produced. It usually produces very good results, but it is slower than simpler linear regression models (trade-off between model complexity and speed has to be considered).

Cons: The main limitation of the Random Forests algorithm is that a large number of trees may slow down the algorithm for real-time prediction. But the number of trees can be optimised as an input parameter. This is a known trade-off between accuracy and speed. We mentioned that the standard version of the Random Forests algorithm does not support incremental learning, but variations for processing streams exist.

4.3.2.6 Stay Point Detection (SPD)

Stay-point detector (SPD) is a special case of stream aggregate, working on a stream of geo-location data (raw GPS coordinates). The raw stream is aggregated into points with higher density for a given time (staypoint), and a set of points with lower spatial density (path). To be able to analyse locations (places) of the users, we first need to understand their location visits and their past mobility patterns. To some extent, raw GPS data contains this information through the distribution and density of points over time. However, the data are noisy and does not explicitly specify where (and how long) the user had stayed. For these reasons, we first employ a staypoint detection (SPD) algorithm, which is able to filter out GPS errors, pinpointing the location and accurately tracking the residence time at each location[26].

The results of our two-pass SPD algorithm are a list of cleaned (as opposed to raw GPS data) staypoints and trajectories of user movement (geo-activities). Besides the location, the descriptions of these activities also contain residence time (start time at the location and duration). These geo-activities represent the input data for the higher level analysis and modelling such as prediction of next location.

[24] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
[25] scikit-learn.org
[26] http://ieeexplore.ieee.org/document/8089732/

EXAMPLE: Example[27] of SPD stream aggregate usage in QMiner

```
// used only for schema
// will not be used to hold records (push will not be called)
var aggr = new qm.StreamAggr(base, {
   type: "stayPointDetector",
   store: store,
   userField: "User",
   timeField: "Time",
   locationField: "Location",
   accuracyField: "Accuracy",
   activitiesField: "Activities",
   params: { dT: 50, tT: 300 }
});

//test
var ts = Date.now();
for (var i = 0; i < 100; i++) {
   // create QMiner wrapped record from JSON
   var rec = store.newRecord({
      Time: ts + i,
      Location: [Math.random(), Math.random()],
      Activities: [20,15,22,23,50],
      Accuracy: 1
   });
   // calls onAdd on all stream aggregates registered on store
   store.triggerOnAddCallbacks(rec);
   var result = aggr.saveJson();
   console.log(result);
   console.log(new Date(result.lastTimestamp));
}
```

4.3.3 Power system modelling and fault detection

Real data are expensive to acquire and may be limited to some specific use cases due to other operational limitations (safety, planning …). The objective of task 6.3 is to fill in missing data, such as voltage and current which cannot be evaluated based on statistical patterns. Two methods are evaluated:

- At train level, compressive sensing techniques coupled with stochastic multi-objective optimisation algorithms are excellent candidates to model those parameters without costly equipment on-board
- At line level, power system simulation techniques, with linear optimal power flow equations to support the energy assets, such as generators, storage, loads and environmental constraints

Those enriched data sets will be provided as inputs to task 6.2 to evaluate their effectiveness on the prediction accuracy. The power system modelling techniques also enable the detection of faults and power quality disturbances. A typical example includes detection of system imbalances through voltage sags. Based on the voltage sags signatures the fault type, the fault location, the system grounding and the

[27] https://github.com/QMiner/QMiner/blob/master/src/third_party/geospatial/example/staypoint.js

connection of rolling stock can be estimated. Once these information become available, fast and accurate estimation of possible abnormal operational patterns can be achieved leading to significant reduction in LCC.

4.3.4 Operation optimisation

The huge volume of data that will be collected and stored in the data management platform will be effectively used within task 6.4 to address the energy efficient railway operation problem in long, mid and short-term basis. This planning can involve the identification of daily time-tables, type of rolling stock to be used, number of units/carriages needed and energy aware driving profile strategies adopting scalable optimisation methods. The ultimate objective is the development of solutions that are in line with the "Intelligent train" paradigm enabling the operation of the rolling stock in an autonomous mode (driverless trains). The positioning solution based on low cost Light Detection and Ranging technologies (LIDARS) will be also exploited for the development of in-tunnel cartography scheme that will assist in optimising the operation of the railway system.

5 Analytics and visualisation application

5.1 Structure

The Reims tram is equipped with many meters allowing the measurement of different physical quantities. These quantities are used in the project to model and forecast the energy consumption of the tram. A visualisation tool is therefore needed to easily access the measured data as well as the forecasted data. A web-based application will be developed to present the data themselves (their evolution with time and their correlations) and the results of the forecasting and of the modelling. Moreover, this tool will allow to easily visualise the accuracy of the models, with a visual comparison of the measured and of the forecasted/modelled data, in addition to displaying the results of the different evaluation metrics.

The different types of data described in section 2.1 will be displayed:

- the pre-processed data that will be received through a Kafka flux;
- the forecasted data, the forecast accuracy and the model's parameters received through a Kafka flux;
- the historical data received through direct requests via a secured API.

The application is based on Node.js, an open-source JavaScript environment. The communications between the web-server and the client side are ensured by Socket.IO, a JavaScript library for real-time communications and analytics. This application will be deployed on the same machine where the global infrastructure detailed in section 2.3 is located.

5.2 Description of the screens

The application is constituted by several configurable screens. The first one is a dashboard displaying the real-time data with a geolocation map and a synchronised graphic (Figure 5.1).

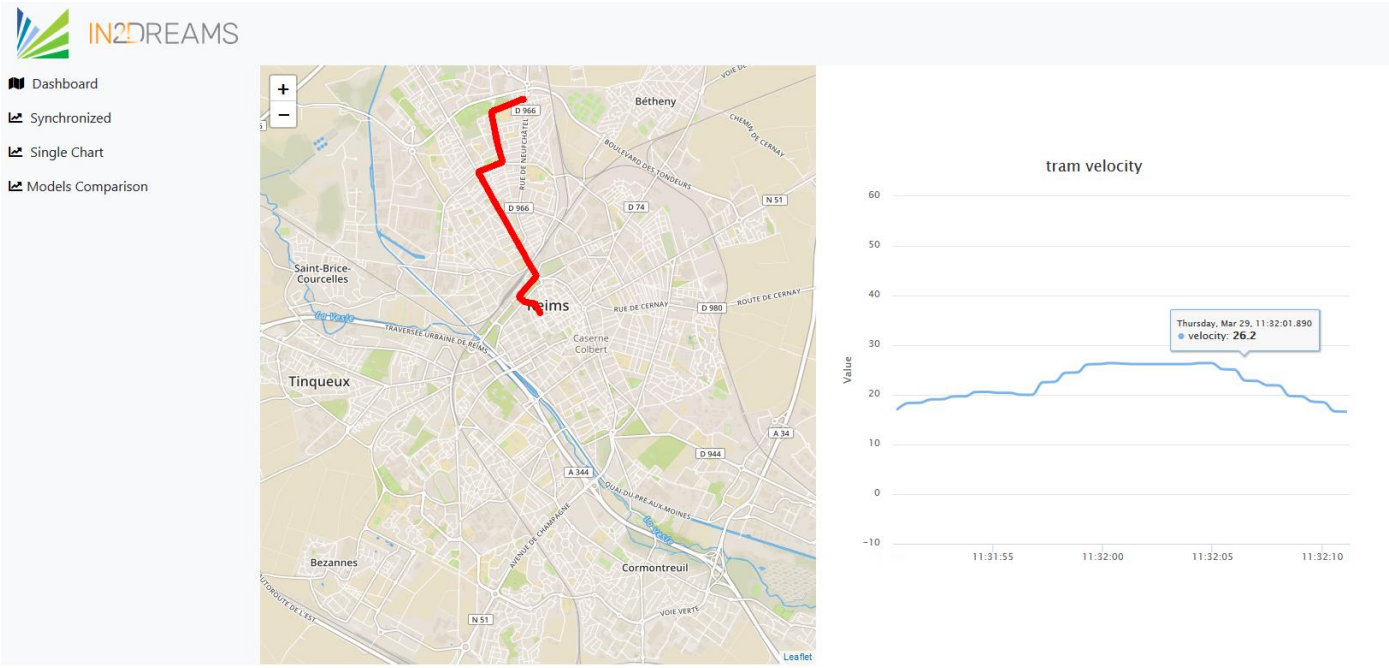

Figure 5.1: Dashboard with real-time data

The second screen displays the historical data and includes a configurable panel as well, which enables the user to select the physical quantity he/she wants to display and/or the time interval. Three types of graphics are available in this screen: a time series graphics to visualise the evolution of the selected measurement with time (line chart, Figure 5.2), a scatter chart to study correlations between different physical quantities (Figure 5.3), and synchronised graphics to display mutual dependencies between physical quantities as a function of time (Figure 5.4).

Figure 5.2: Time evolution of the selected measurement (line chart)

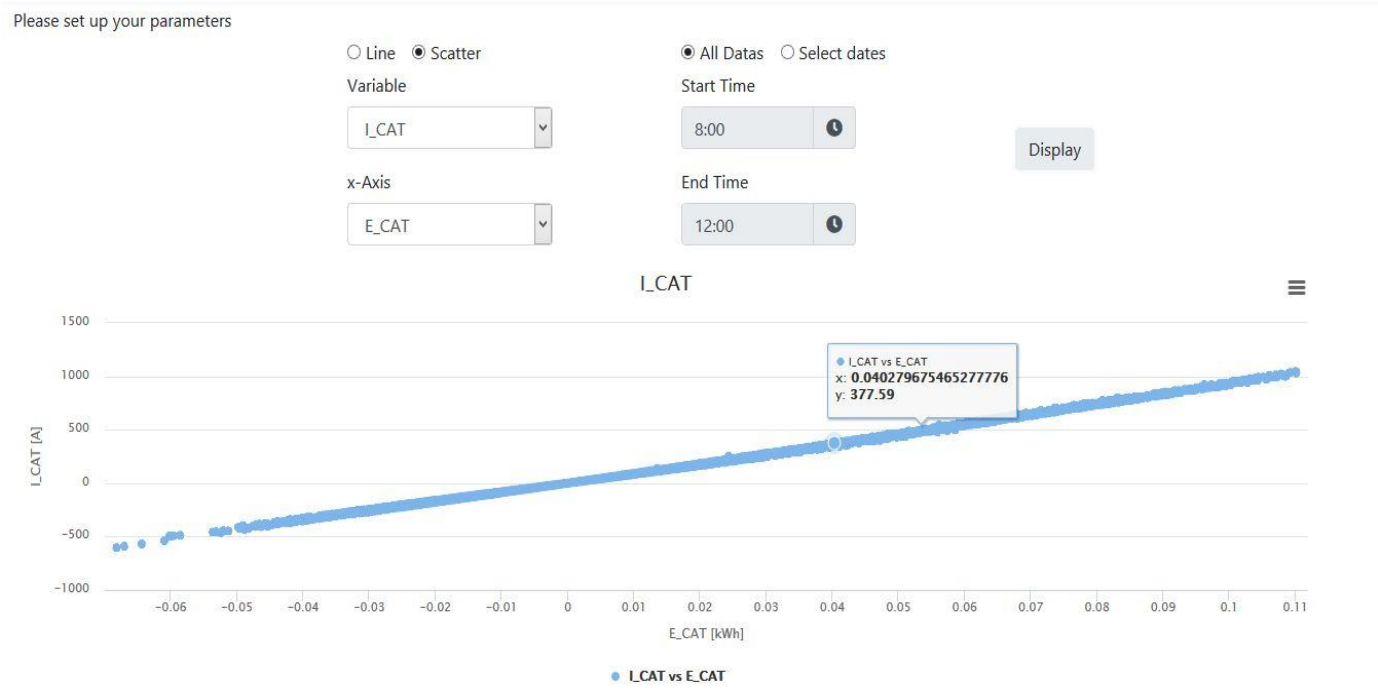

Figure 5.3: Scatter chart

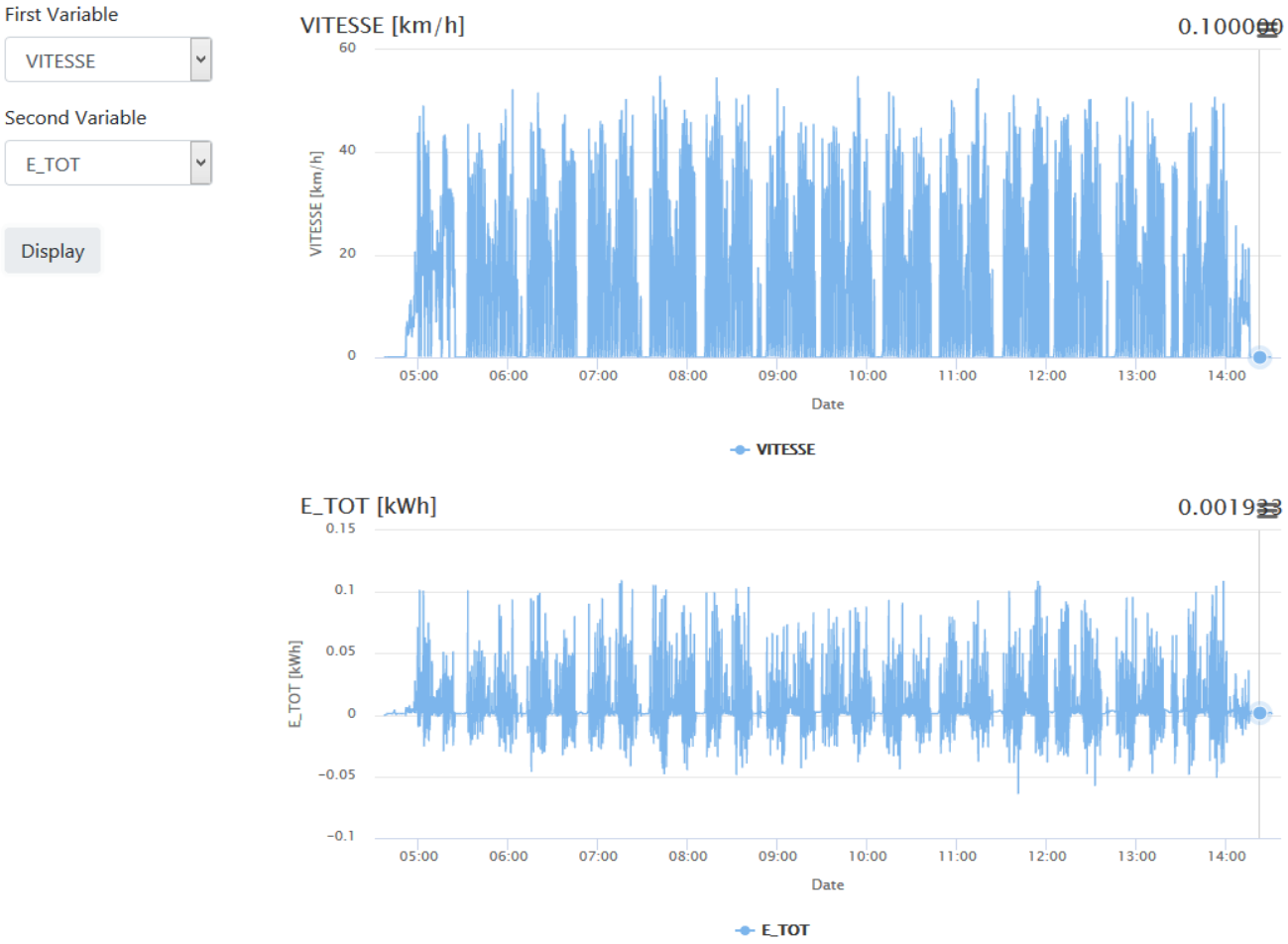

Figure 5.4: Synchronised charts

After the tasks 6.2 and 6.3 will be completed, the application will also provide the results of the forecasting modelling (task 6.2) and of the power system and fault detection modelling (task 6.3).

6 Conclusions

In conclusion, QMiner is best-suited for the objectives related to IN2DREAMS WP6 as it is designed for scaling to millions of instances on high-end commodity hardware, providing efficient storage, retrieval and analytics mechanisms with real-time response. Moreover, as the framework is written in C++ for lightweight data processing, it enables ubiquitous deployment (such as on microcontroller on on-board units, etc.) for various use case scenarios.

The powerful software architecture integrates communication with the ODM platform developed within the IN2RAIL project, analysis inside QMiner, data bases and visualisation application. As the component is open source and integrated using a message broker, a high level of flexibility and scalability can be achieved. Designed architecture and selected components therefore offer modular structure with loosely coupled

components, to achieve a high level of analytical infrastructure flexibility and scalability. In such way, the analytical infrastructure will be able to support a plethora of use cases and the complexity of the designed project in the following phases.

The installation of QMiner within the data analytics platform has been successful. The software will be used for analytics (tasks 6.2 and 6.3) and application use case (task 6.4).

A use case with real data has been selected in order to define how the data analytics platform will be used to support a real use case scenario and to gain insight into the main required functionalities which will be used and developed in the upcoming tasks. Moreover, a visualisation application is available within this platform for an easier access and understanding of the data and of the model results.