NRG4CAST
FP7-2012-NMP-ENV-ENERGY-ICT-EeB
Contract no.: 600074
www.nrg4cast.org

NRG4CAST

Deliverable D2.3

Data cleaning and data fusion final prototype

Editor:	Klemen Kenda, JSI
Author(s):	Jasna Skrbec, Klemen Kenda, JSI; Simon Mokorel, Envigence; Martin Birkmeier, FIR; Tatsiana Hubina, CSI; Rosie Christodoulaki, NTUA;
Reviewers:	Giannis Hamodrakas, SL; Steffen Nienke, FIR;
Deliverable Nature:	Prototype (P)
Dissemination Level: (Confidentiality)[1]	Public (PU)
Contractual Delivery Date:	November 2013
Actual Delivery Date:	November 2013
Suggested Readers:	Developers creating software components to be integrated into final tool for different users, developers creating models, prediction tools, etc., developers of final data cleaning and data fusion prototype.
Version:	0.8
Keywords:	Data cleaning, data fusion, data analysis, Kalman filter, data aggregation

[1] Please indicate the dissemination level using one of the following codes:
• **PU =** Public • **PP =** Restricted to other programme participants (including the Commission Services) • **RE =** Restricted to a group specified by the consortium (including the Commission Services) • **CO =** Confidential, only for members of the consortium (including the Commission Services) • **Restreint UE =** Classified with the classification level "Restreint UE" according to Commission Decision 2001/844 and amendments • **Confidentiel UE =** Classified with the mention of the classification level "Confidentiel UE" according to Commission Decision 2001/844 and amendments • **Secret UE =** Classified with the mention of the classification level "Secret UE" according to Commission Decision 2001/844 and amendments

Disclaimer

This document contains material, which is the copyright of certain NRG4CAST consortium parties, and may not be reproduced or copied without permission.

In case of Public (PU):
All NRG4CAST consortium parties have agreed to full publication of this document.

In case of Restricted to Programme (PP):
All NRG4CAST consortium parties have agreed to make this document available on request to other framework programme participants.

In case of Restricted to Group (RE):
The information contained in this document is the proprietary confidential information of the NRG4CAST consortium and may not be disclosed except in accordance with the consortium agreement. However, all NRG4CAST consortium parties have agreed to make this document available to <group> / <purpose>.

In case of Consortium confidential (CO):
The information contained in this document is the proprietary confidential information of the NRG4CAST consortium and may not be disclosed except in accordance with the consortium agreement.

The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the NRG4CAST consortium as a whole, nor a certain party of the NRG4CAST consortium warrant that the information contained in this document is capable of use, or that use of the information is free from risk, and accept no liability for loss or damage suffered by any person using this information.

Executive Summary

Data cleaning and data fusion are the most important part of pre-processing the data. If we get corrupt or inaccurate data in the main phase of working on data, it can destroy an otherwise very good architecture or system. Especially in our case in which data will be an input for monitoring and prediction algorithms, it is important to have clean and properly fused data in order to have good results. Cleaning is a very sensitive part of the pre-processing phase as well. We do not want to clean too many data, we still want to keep part of the data stream with special events shown as maximum or minimum. Therefore we need to pay special attention to keep good data and clean the bad ones.

The work reported in this deliverable is the continuation of the work, which was already reported in the NRG4CAST D2.2. We took results from first deliverable and further developed them into the prototype, which is described here in details. The prototype architecture consists of three layers, basis is the analytical platform, which includes APIs for incoming and outgoing data, data layer, aggregation, cleaning and JS engine, middle layer takes care for CMS and safety, and GUI as a top layer. QMiner analytical platform incorporates data model with four main stores typical for input data from sensors: Subject, Node, Sensor and Measurement. In order to find the proper method for data cleaning, we compare some methods and we are defending the Kalman filter as the most adequate for our purposes and future work on the data.

Data cleaning can start when measurements are sent or more accurately when they are collected on the server or on the node site. Each case study has its own process of sending, collecting data and later saving it for historical purposes. In Miren pilot case messages sent from light poles are checked for cases of corrupt, incomplete, irrelevant, missed or duplicated messages. Sensors in electric cars has a lot of chances that something will go wrong during their way to the server. In FIR pilot case data is cleaned already in the data box, which collects data inside an electric car, also some precautions are included into the architecture of the whole system (e.g. standardisation of information send around). On CSI pilot site the data cleaning is included in the data pre-processing, where data comes together and is put into data base. Combining data from all case study sites we need to know what happened with data before it comes to the conjunctive spot, so we can do additional cleaning properly.

Data fusion presents second part of this deliverable and is also continuation of work previously done on NRG4CAST D2.2. In NRG4Cast project we have a lot of different data (sensor, external, textual) and we need to bring all these sources to align them together in a right way and with no loss of information. Aggregates were implemented on sensor sources, which brings also transparency of the data. Examples of external data fusion are shown on the energy price and SensorFeed data. Data fusion and data cleaning prototype is described in detail in the final chapters.

Table of Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
CET	Central European Time
DB	Database
GUI	Graphical User Interface
JS	JavaScript
KF	Kalman Filter
OGSA-DAI	Open Grid Services Architecture – Data Access and Integration
SVM	Support Vector Machine
QMAP	QMiner Analytical Platform

1 Introduction

This deliverable builds on the findings of the prior deliverable NRG4CAST D2.2 – Data Cleaning and Data Fusion Initial Prototype.

Significant part of D2.2 was dedicated to the analysis of the data sources. Reasons for data cleaning and data fusion were pointed out and described why they are bothering data scientists. Methods and approaches were gathered, categorized into a section for data cleaning and a separate section for data fusion. Methods and approaches for sensor data were specially emphasised, since they are of great importance for our project. Data sources have been put together and preliminary data schemas have been created.

Static data sources (non-streaming data) have been analysed in depth. An overview diagram of all data sources was made, allowing everyone involved with data on the project to get a major picture of data and an easier illustration of ideas on work to be done on this data. On the provided static data from CSI sources deeper analysis was done; it consisted of exposing what kind of metadata we can get. For each assembly further analysis was made in order to see the quality of this data. For example too many missing values would make pretty poor quality of the data. We also proceed with quality of the content of the data. Almost same flow of analysis was made for NTUA and FIR case studies' historical data that were made available for analysis. Processing all the static data revealed that no further data cleaning is needed for this kind of data for now. There was no sign of mistakes or holes of missing data, which would influence data as whole set-up. However in later phases of data fusion there is still a possibility that the need for additional approaches will arise. On the provided stream data a cleaning process was set up and demonstrated; this was Kalman filtering as the most appropriate method in our case.

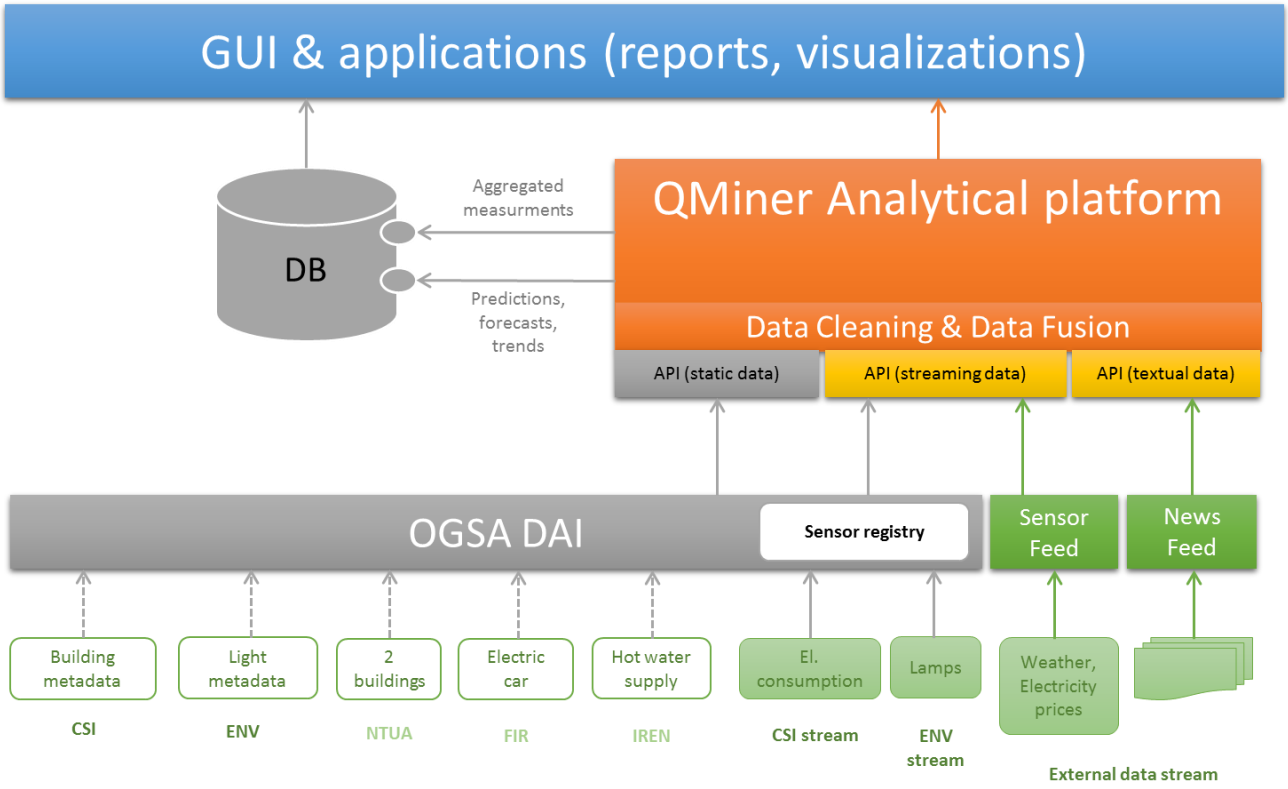

Figure 1: Architecture of the system.

The Architecture of the system is depicted in Figure 1. This picture is relevant also for deliverables NRG4CAST D2.4 and NRG4CAST D6.3. Depicted in colour (non-grayscale) are the components that are of relevance for this deliverable and where at least a part of them is described in detail. Main component that is described in this deliverable is the Data Cleaning & Data Fusion module within the QMiner Analytical

platform. We have also described part of the data gathering infrastructure that is relevant for Data Fusion: SensorFeed (developed within work for this deliverable) and NewsFeed.

Data sources are described with respect to the Data Cleaning that is in a fair share done already at the data providers' level.

With the Data Cleaning we have mainly focused to the streaming data as most of the data reaches analytical platform as a stream or at least as a simulated stream. We have taken into account characteristics of the sensor data and demands for data cleaning module, tested a few different methods and have chosen the most suitable one to be used, which we determined was Kalman filtering. Kalman filter and its usage for Data Cleaning are described in a bit more detail.

In the Data Fusion part we have focused on an efficient implementation of time alignment and geo-location querying capabilities of the system. We have also imported data from external sources (energy prices/consumption, weather open data) and focused on the infrastructure for processing such data.

In Section 2 we describe theoretical aspects of the methods used for Data Cleaning and Data Fusion, we also describe the architecture of the system, review the data model used in the analytical platform. Then we present results of multiple methods used for detecting outliers in the incoming data and focus on the Kalman filter, which was chosen among them. We propose a method for data cleaning that can be generalised to any prediction method, not only Kalman Filter.

In Section 3 and 4 we focus on the Data Cleaning and Data Fusion that has been done on the available data sources. In the Data Fusion part we focus mostly on the aggregates, that are aligned by their timestamp and therefore suitable for a number of data mining/machine learning methods, also geographical indexing of the sensor data is presented.

Section 5 is more technical and describes software, used in the prototype. There are also brief instructions for the usage of the prototype[2]. We conclude in Section 6, which also exposes a few open issues and suggestions for the future work.

[2] Prototype is accessible at: http://demo.nrg4cast.org/en/d2.3.html

2 Methods and Approaches

2.1 Prototype Architecture

In this section, the prototype architecture is to be presented in more detail. Architecture is depicted in Figure 2. It consists of the QMiner Analytical Platform and a GUI on top of it. Part of the middle layer is partially used also as an additional safety layer at the southbound API.

The GUI is powered by an additional middle layer, which puts another safety level above the QMiner Analytical Platform. The middle layer is also used as a CMS for the prototype and other potential prototypes, hosted on the same platform.

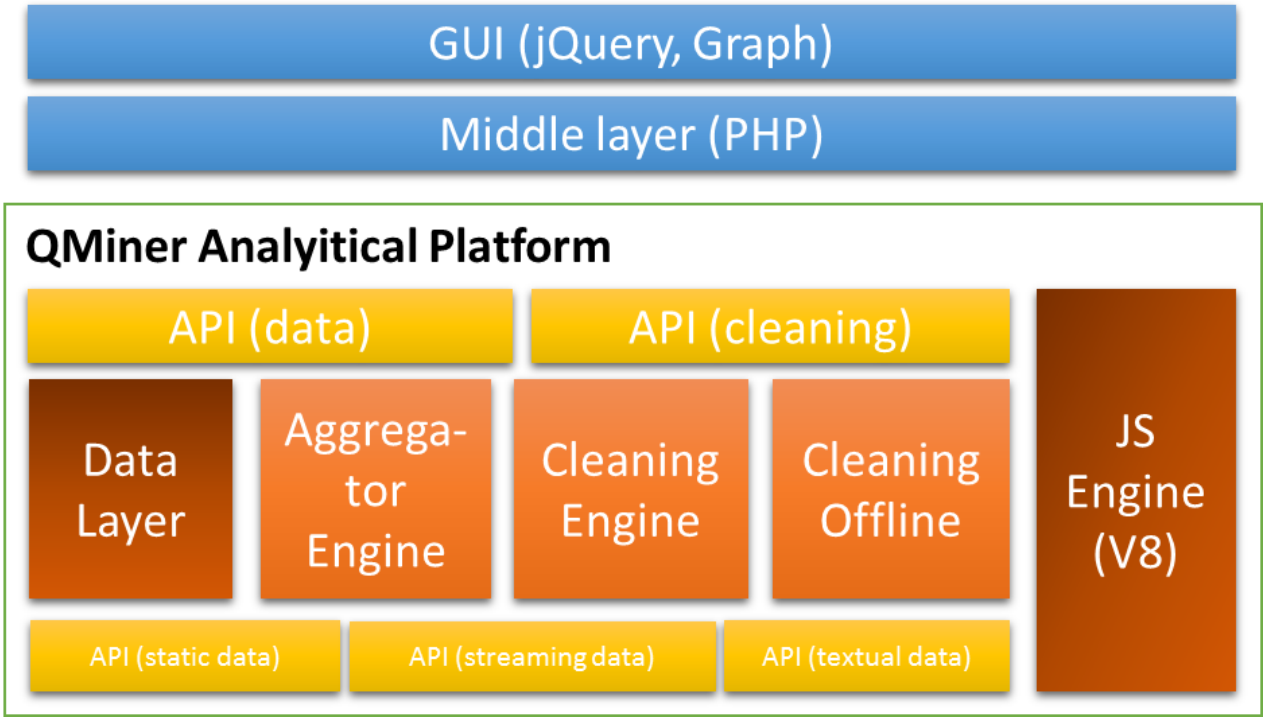

Figure 2: Prototype architecture.

QMAP has southbound (incoming) and northbound (outgoing) API for data exchange. There are 3 different southbound APIs that are described in a NRG4CAST D2.4 [36]. The API for static data is basically a data adapter for SOAP services at the OGSA-DAI level. Northbound APIs are only divided in the logical sense as they are implemented on the same principles. Part of the API is dedicated to data retrieval (measurements, aggregates and meta-data) and part of it is dedicated to cleaning. Northbound API is discussed in detail in Section 5.1.1.

There are two pillars that enable all the modules within QMAP. Those are the JS engine (Google V8) which exposes QMiner C++ functionality and the Data Layer, which is accessible from anywhere within the QMAP. Those two pillars support the three modules, implemented for this deliverable. Those are Aggregator Engine, Cleaning Engine (online) and Offline Cleaning Engine. All the engines are described in detail in Section 3.

2.2 Data Model

The described data model has been configured in QMiner. It contains 4 main stores[3] (Subject, Node, Sensor and Measurement), 3 description code books (SubType, Property and Type) which contain additional

[3] reader might be more familiar with the term “table”, which is an SQL *equivalent* of the store

information to describe certain subjects and sensors. There is one more store with derived data. This is the store of Aggregates which is in more detail described in section 4.1.1.

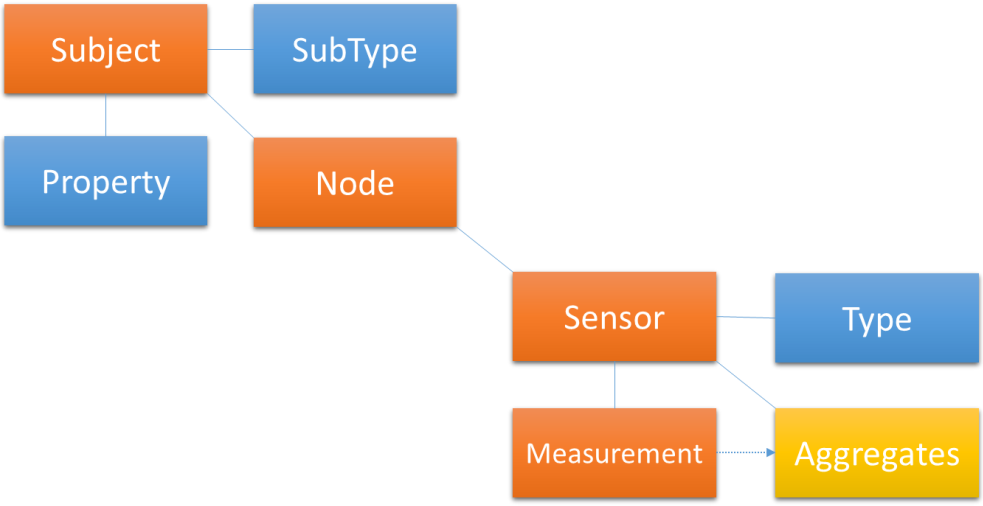

Figure 3: Data Layer schema in the QMiner.

This data model represents the data layer of the QMiner component. Data Layer is tightly integrated with the layer with data mining and machine learning algorithms.

2.3 Comparison of Methods for Data Cleaning

Methods for Data Cleaning just about the data tier should not be processor-power demanding (great load of data), they should be independent (meaning that they should not rely on any other data but the sensor measurements themselves and some initial configuration data). At this level there is no semantic enrichment of sensor measurements and the corresponding metadata. The following methods have been chosen:

- last measurement
- moving average
- moving average with trend
- Kalman Filter in the following configuration
    - 0^{th} order polynomial model (constant)
    - 1^{st} order polynomial model
    - 2^{nd} order polynomial model (dynamic model)

We have prepared two datasets – real augmented (with added typical failures) dataset and an artificial dataset. Artificial dataset has been generated with a periodic smooth function with a period of 1 day (sampled at 15 minutes). Noise has been added to the measurement and 4 kinds of outliers have been randomly put into the sample.

The following function has been used to generate the base function:

$$f(x) = A\sin(kx)$$

The following parameters have been used: A = 1, k =1/86400, where x represents a timestamp in seconds and k represents a one day period in those units. Gaussian noise has been added to the measurements with an amplitude of 0.02.

The four kinds of outliers were added. All of the outliers have of course an amplitude that is higher than the amplitude of noise in the sample. Distribution of the outliers was Gaussian and also the amplitude of an outlier has been chosen with Gaussian random function. There was however a hard barrier of 0.5, which

means that outliers with less than 0.5 x Amplitude have not been inserted into the dataset. Table of the amplitudes and probabilities was as follows:

Probability	**Amplitude**
0.01	0.05
0.01	0.1
0.01	1
0.005	5

Table 1: Probabilities and amplitudes of outliers in the artificial test dataset.

The measures, used to evaluate the different method, would be precision, recall and F1 score. They are defined as:

$$Precision = \frac{tp}{tp + fp}$$

$$Recall = \frac{tp}{tp + fn}$$

$$F1 = 2\,\frac{precision \times recall}{precision + recall}$$

In the equations above the *tp, fp, tn and fn define true positives, false positives, true negatives and false negatives respectively.*

Results of testing, where F1 score has been optimized, are shown in Table 2 and Table 3.

Method	**Precision**	**Recall**	**F1**
Last measurement	0,508	0,738	0,602
Moving average	1,000	0,929	0,962
Moving avg. with trend	1,000	1,000	1,000
KF – 0^{th} degree	0,508	0,738	0,602
KF – 1^{st} degree	0,889	0,762	0,821
KF – 2^{nd} degree	1,000	1,000	1,000

Table 2: Test results on the real augmented dataset.

Method	**Precision**	**Recall**	**F1**
Last measurement	0,493	0,678	0,571
Moving average	0,940	0,659	0,775
Moving avg. with trend	0,977	0,669	0,794
KF – 0^{th} degree	0,493	0,678	0,571
KF – 1^{st} degree	0,934	0,582	0,718
KF – 2^{nd} degree	0,981	0,675	0,800

Table 3: Results on the artificial dataset.

As a short term prediction also different regression methods could be used. Both methods, Kalman Filter and linear regression can be used for future prediction (in the causal case). However, the critical difference is the implementation, which might be significant. In the case of linear regression the entire process needs to get repeated, which is computationally expensive. On the other hand Kalman filter is inherently recursive. Using it for prediction on only the past data, which is the task with the data cleaning scenario, is very efficient [10]. Of course it is possible to use on-line learning methods. In this setting and with the premise that noise in measurements is Gaussian Kalman filter ensures the optimal (in the sense of minimising the mean square error) prediction of the future states [11].

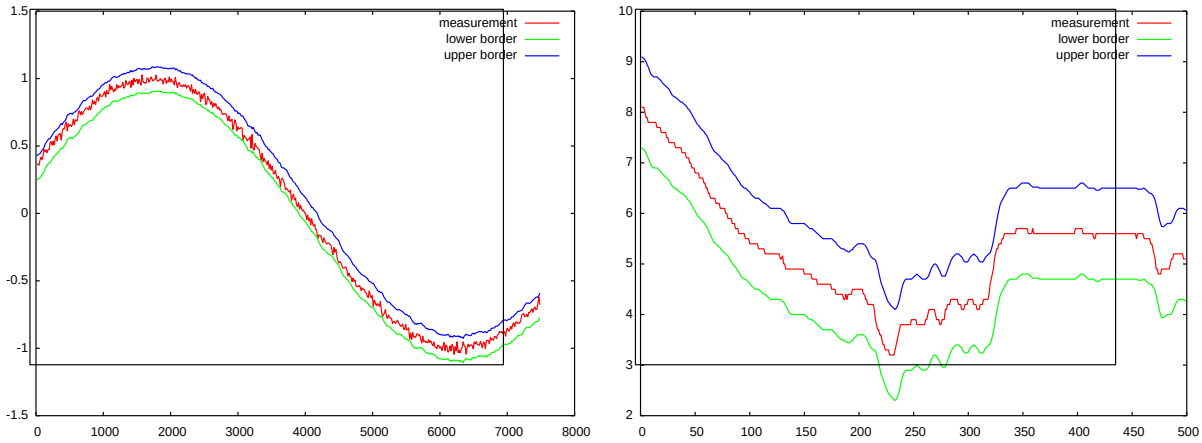

Figure 4: Testing of the moving average algorithm on artificial and real dataset.

According to the results and discussion the method of choice is Kalman Filter with a 2nd degree model (dynamic).

2.4 Kalman Filter

The Kalman Filter has already been explained in NRG4CAST D2.2 [2], but for completeness reasons this updated and partially also simplified (according to additional work done in [1]) section is included also in this deliverable.

The Kalman filter is a method for solving the discrete-data linear problem. The filter consists of a set of mathematical equations that can estimate the underlying (hidden) state of a process in a way that the mean of the squared error is minimized. The filter supports estimation of past, present and even future states [1][2][8].

Underlying process to be modelled is a Gauss-Markov process (see Figure 1). This means that any subsequent state is only dependent on the previous state of the system. Figure depicts observations (in our case sensor measurements) x_j and underlying hidden states θ_j (vectors of a real value of the measured phenomena and its first and second temporal derivative). The arrows in the figure depict the Gaussian processes and point from underlying state θ_j to the next state θ_{j+1} (transitional equation) and also from the underlying state θ_j to the observable state of the system x_j (observational equation) [1][2].

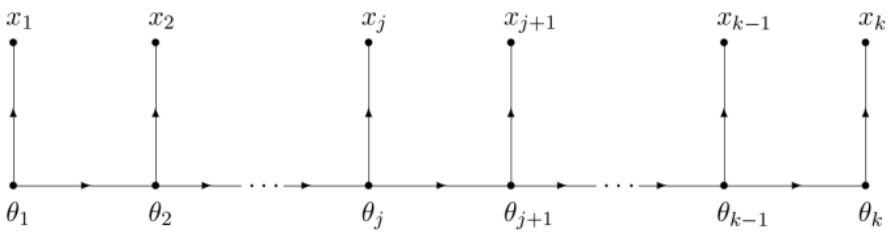

Figure 5: Diagram of the Gauss-Markov process [1][2].

The result of solving such a problem is a set of equations that include prediction and correction phase depicted in Figure 2 [1][2]. Comprehensive explanation and derivation of the filter equations can be found in the literature [6][7].

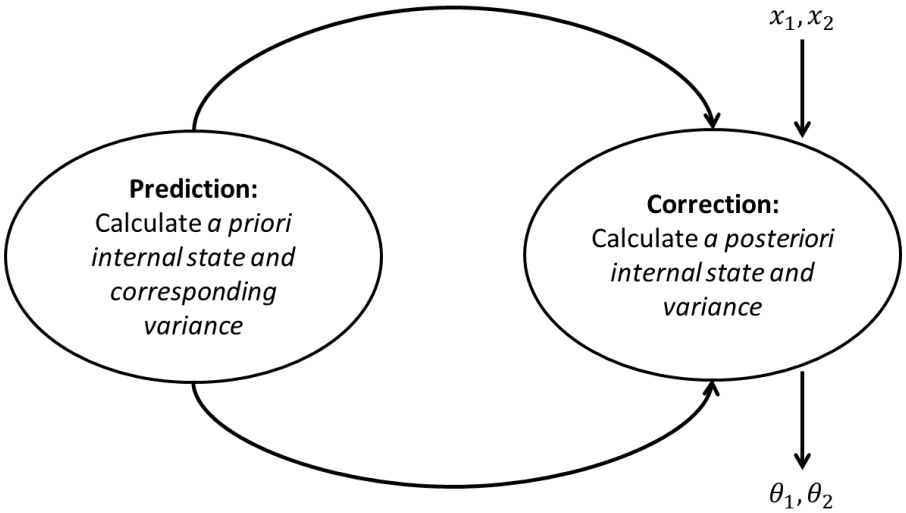

Figure 6: Kalman Filter schema – prediction and correction phase [1].

The Kalman filter loop consists of two phases. Correction and prediction phase. The correction phase takes a measurement (x_j) and corrects the prediction (or the initial state) of the system. The correction phase also updates the underlying model parameters of the filter, which makes the filter adaptive. This means that the model is changing according to the measurements in an on-line manner. The prediction phase relies on the model-nature of the algorithm and is therefore able to project the current underlying state of the system into the future.

To illustrate the prediction phase of the Kalman filter only one equation from the whole set needs to be understood – the transitional equation below.

$$\theta_{k+1}^{-} = \Phi_{\mathrm{k}} \theta_k$$

The new *a priori* state (prediction) θ_{k+1}^{-} is obtained by simply multiplying the transitional matrix Φ_k with the *a posteriori* state vector (previous correction). In our case, each row in the transitional matrix describes how each element of the state vector is transformed.

2.4.1 Kalman Filter for Data Cleaning of Sensor Data

In our experience, the characteristics of the sensor data are as follows:

- streaming (on-line)
- high frequency (i.e. sensor readings are much more frequent than big changes of the property they are measuring)
- measured property is continuous and is changing smoothly (no big sudden jumps are expected with most of the properties; exceptions should be handled in the phase of semi-supervised initialization of the data cleaning filter)
- there are either only vague or too complex models for modeling the physical phenomena being measured

Considering the features above, we conclude that Kalman filtering is a suitable method for detecting outliers in sensor data. With its prediction features, one can also be able to replace the incorrect or missing data in the sensor data stream.

We propose a second-degree model (also suggested in [8]) that takes into account first and second order temporal derivatives of the measured properties. The equalities in Figure 3 define the state vector and the model to be used with a Kalman filter for data cleaning. A denotes the physical phenomena the sensor is measuring, and t the time. The state vector θ_k is 3-dimensional and includes the actual value of the physical phenomena in the first component, its first temporal derivative in the second component and its second temporal derivative in the third component.

$$\theta_k = (A, dA/dt, d^2A/dt^2)$$

$$\Phi_k = \begin{pmatrix} 1 & \Delta t & \frac{1}{2}\Delta t^2 \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{pmatrix}$$

Figure 7: The state vector and the transitional matrix in a dynamic linear model.

The concept of usage is straightforward. We exploit the prediction phase of the Kalman filter to predict the value of the measured phenomena for the timestamp of the measurement. Based on the comparison of the prediction with the actual value, the system decides whether the new measurement is correct or if it should be classified as an outlier. The difference between prediction and measurement is interpreted in terms of variance.

Figure 8 shows two examples for assessing whether a new measurement is an outlier or not. The principle can be generalized to any method using the prediction, not only Kalman filtering. In the first case prediction lies within the defined gap and in the second case the measurement lies outside the gap and is therefore discarded. The gap could be learned with a semi-supervised method, where the user would assist the algorithm by manually annotating the good measurements and outliers in the training set. The set defines a hard border for the gap: every measurement that falls outside the gap is considered as an outlier to the algorithm.

Stochastic invalid or missing values can be replaced with prediction values of the Kalman filter. These errors occur at random and usually represent single, isolated events. The second class of invalid/missing values represents those which are a consequence of a system failure (device or network), last longer and cannot be compensated.

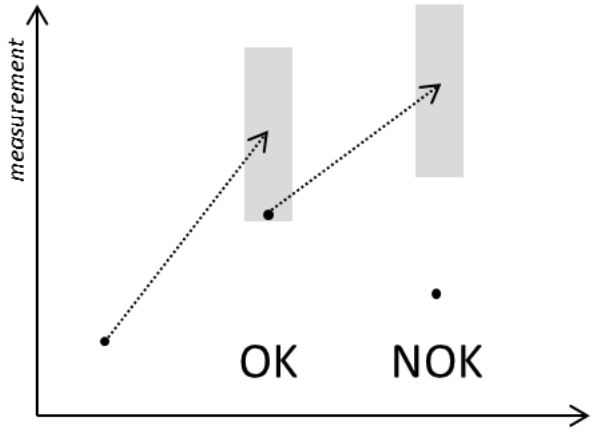

Figure 8: Evaluation of new measurements with Kalman filter prediction and learned threshold.

2.4.2 Illustration of the Principle

Principle is illustrated on the outside temperature dataset with one sensor reading per 15 minutes. The dataset includes measurements from July to August 2013 with occasional stochastic failures in the form of 0.0˚C readings.

Figure 9 shows basic principle of the algorithm. The Kalman filter in its prediction phase returns two relevant values: prediction for the value of the temperature and its variance. After configuration stage, an expert user has determined proper upper and lower bound interval (and other parameters of the filter like process noise covariance, measurement covariance and post error covariance), which was 5σ (only observing the first dimension of state vector).

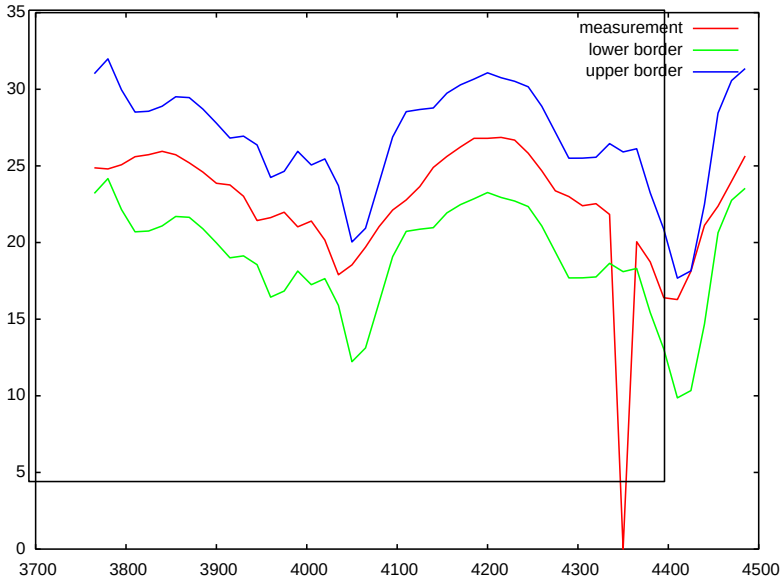

Figure 9: Identifying an outlier with the Kalman filter.

Figure shows the upper and lower boundary calculated from prediction and its variance and the actual measurements. Where a measurement lies outside the band delimited by the lower and upper boundary an outlier is detected.

With a good choice of model parameters, we have been able to achieve the results that included all the true positives and none of the false negatives. However, errors in the dataset have been specific and easily identifiable in the summer time (with high temperatures). The most difficult problem for the algorithm was a sudden change in temperature – as expected.

The Kalman filter is not a complex algorithm, but it can be difficult to adjust all the required initial conditions and parameters, which demand either an expert user or a statistical method to adjust them. With optimal tuning of the parameters very good results can be achieved [1].

2.4.3 Solving the Instability of the principle

The principle had proven to be unstable and the Kalman filter prediction had diverged in some cases. If the filter encounters a false negative it relies on the prediction model, which can then move the lower/upper boundary so that no measurement ever again fits the criteria (see Figure 10).

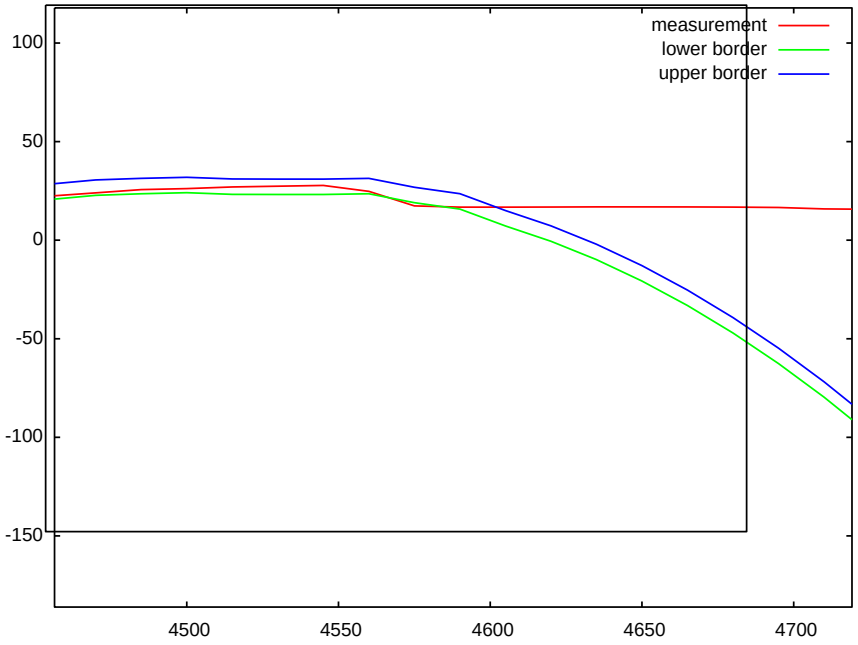

Figure 10: Instability of the algorithm when detecting a false negative.

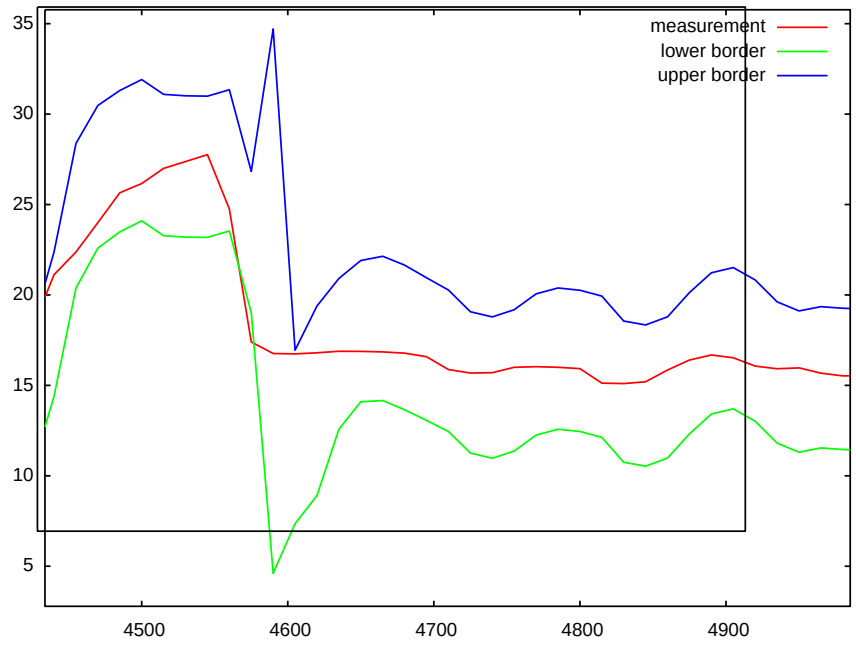

Figure 11: Instability workaround.

A possible workaround includes an artificial increase of *a posteriori* variance (see band enlargement after false negative in the Figure 11). Theoretically, changing variance is mathematically incorrect, but in practice it has proven to be efficient. Besides, variance in the case of slowly changing values of sensor data soon converges to the vicinity of the correct values. Further research on the proposed approach is needed [1].

3 Data Cleaning on the Pilot Data

3.1 Data Preprocessing and Cleaning at the Data Providers' Side

3.1.1 NTUA Pilot Case

The data sources in the NTUA pilot case are the electricity sensors, the thermal comfort level sensors and the natural gas bills.

Data on electricity consumption and thermal comfort level will be obtained automatically from the sensors. Data on natural gas consumption will be available from the utility bills (so manually inserted).

The data taken from the 23 substations is stored in a central data handling unit (PC) in the premises of NTUA Laboratory.

3.1.2 Miren Pilot Case

Each light sends a message with measurements to the server every 15 minutes. Every message has the unique ID. Server controls incoming messages and checks a type, a regularity and the accuracy of the messages. If a message is corrupt (unknown message) the server stores this message in the database and we use this data for analytics purposes. If the message is inaccurate or incorrect we still use this message and consider something is wrong with the light. Incomplete messages can be used and we store them in database for future analysis. Irrelevant messages are indicators that something is wrong with the light. Duplicated messages are detected and stored, but only one message is used for further processing. We detect the missing messages at the gateway and at the server side. If a message from a certain light pole is not received two times in a row as scheduled the system needs to determine, whether the lights have been turned off or whether there is a failure at the light. Here a simple inference system has been implemented that can determine between two typical case. According to topology of the local light network the system can determine, whether the light unresponsiveness is an isolated case (in that case the reason might be lamp malfunction) or are there some neighbouring lights unresponsive too, which means the probable cause of the event is that the lights have been turned off in that particular area.

3.1.3 FIR Pilot Case

For a reliable data acquisition it is crucial to perform a data cleaning. In Case of the FIR pilot case the data is mainly perceived by an electric vehicle. Therefore, not the direct data stream perceived by the sensors is monitored but the data interface of the car. Consequently, most of the data received from the sensors is already merged and two iterations of data clearing had been performed within the data acquisition. The first one is usually done by the sensor itself and the second one is performed by the data box that collects the whole sensor information of the car. The data box is using the cloud-based interface for data acquisition that is provided by the research project "Oscar". Therefore, the first data access of NRG4Cast project can be provided, once the data is stored in the cloud. While accessing and working on the stored data, it has to be kept in mind, that the "Oscar" platform only updates the data if there is a change within the monitored parameters.

To ensure a maximum reliability, additional cleaning steps are performed to reduce the data errors. In a first step, the data is standardised. Thus, the data is converted into a consisted format that ensures the comparability. For example, the one standardised measurement point of the mileage of the electric vehicle consists of a number describing the kilometre and one part representing a timestamp. Once the data is received in this predefined format, the second step, the actual cleaning can be performed. Therefore, once a measurement is received, the first step is to determine the completeness of every data point. Corrupted data can therefore be found and excluded in further analyses. Furthermore, the timestamps can be compared to identify and remove duplicates. Before one of the data sets can be removed, both need to be verified to find the incorrect measurement. This verification can be performed by using a verisimilitude check that determines how likely a value is in the context of its predecessor and successor. For instance, a mileage measurement can only be valid if it is in the corridor of the previous and following value. If both

values are feasible, the first measurement is used. The verisimilitude check is additionally used to verify the measurements regarding their credibility. For example, the mileage value is unlikely to perform rapid changes within a very limited time period.

Using the described mechanisms in addition to the two existing data cleaning methods provided by the sensors themself and the data collection box within the electric vehicle, the provided data should provide a maximum in reliability.

3.1.4 Data pre-processing for CSI site

In the CSI building, real time data on electric energy consumption are monitored by 41 appliances installed on the switchboards of Data Centres, within the Power Centre of the substations and within the Power Centre of Data Centre. The measurements are managed through the software of Schneider Electric (PowerLogic ION Enterprice) which stock data into the SQL ION database. Moreover, the cooling energy generator is supported by 4 refrigeration units. Schneider Electric Software allows the monitoring of refrigerator electrical consumption as well. There are no data on thermal energy consumption.

We will provide ION DataBase backup for the years 2011 and 2012 with a record frequency of 15 minutes. On the other hand, from the year 2013 we will transfer records on real time from ION to the relational database provided by an existing web-based service QRS (the web based tool for energy consumption control and energy saving planning), with a record frequency of 15 minutes as well.

The QRS is a system which allows municipalities for automatic loading and elaboration of energy bills, realisation of reports and graphs on consumption monitoring, gives suggestion for interventions, savings and makes energy consumption prediction. It is based on operational databases in Oracle 10g and Postgress 9.0.4, and specific applications as front-end developed in PHP. User Technical Administrator, through ETL database process, will transfer proprietary internal data recorded by CSI building Energy Monitoring System into the relational database QRS.

For both the cases mentioned above, the ION DBs will present the same structure, and migrated data will be not manipulated in order to review data and correcting extreme values. We selected five parameters in order to estimate the electrical power supply for all the offices of the CSI building (see Table 4) where:

Total Electrical Power Supply (Offices & CED) = ION table (PILLER B_cab_bt_B_ARRIVO_TRAFO1> kWtot) + ION table (PILLER B_cab_bt_B_ARRIVO_TRAFO2> kWtot)

Electrical Power Supply for CED = ION table (NAME: PILLER B_cab_bt_B_Gruppi_Frigo1_2> Real Power Tot) + ION table (NAME: PILLER B_cab_bt_B_Alimentazione_Rete> Real Power Tot) + ION table (NAME: CABINA_CED_QSM_B> Real Power Total)

Estimation of Electrical Power Supply for Offices of CSI-Piemonte Building		
a	b	a - b
Total Electrical Power Supply (CED & Offices)	Electrical Power Supply for CED	Electrical Power Supply for Offices

Table 4: Estimation of Electrical Power Supply for Offices of CSI-Piemonte Building.

For all the five parameters, if the current transformator is turned off, the input of the data logger will be 0. On the other hand, if a record anomaly occurs, the input of the data logger will be an empty cell.

In the following section, we present a mapping template for Data Migration of tables and fields from the SQL ION DataBase of the CSI building Energy Monitoring System into the Postgres QRS DataBase of the web based tool for energy consumption control and energy saving planning.

All the alphanumeric values related to the column DATALOG_STAMP_ID of the table DATALOG, corresponding to the attribute QUANTITY =193 of the ION DataBase need to be migrated in pre-set tables of the QRS database, with a frequency of five minutes and without any data elaboration or correction.

The mapping for data extraction is schematized as follow:

ID NUMBER:	NAME
31	PILLER B_cab_bt_B_ARRIVO_TRAFO1>kWtot
32	PILLER B_cab_bt_B_ARRIVO_TRAFO2>kWtot
33	PILLER B_cab_bt_B_Gruppi_Frigo1_2>Real Power Tot
28	PILLER B_cab_bt_B_Alimentazione_Rete>Real Power Tot
24	CABINA_CED_QSM_B>Real Power Total

Table 5: Selected parameters in the table "SOURCE".

Other provided systems and data migration for the NRG4Cast are listed as follow:

- Data gathered from new devices and sensors (e.g. in-door condition and energy consumption) which will be installed in different areas of the CSI building, as well as gas-heating, tele-heating and electricity consumption data attributed to preselected building of the City of Turin, will be transferred into the QRS Data Sink.
- User Technical Administrator, through a ETL database process, will transfer proprietary internal data recorded by CSI building Energy Monitoring System into a common relational database provided by an existing web-based service QRS (the web based tool for energy consumption control and energy saving planning). Moreover, data gathered from new devices and sensors (e.g. in-door condition and energy consumption) which will be installed in different areas of the CSI building, as well as gas-heating, tele-heating and electricity consumption data attributed to preselected building of the City of Turin, will be transferred into the QRS Data Sink.
- User Technical Administrator will transfer alphanumeric and geometric data gathered from by the systems SICEE (Information System for the Energy Performance Certificate of a Building) and SIGIT (Heating Installations Management Information System) into the ENERCAD 3D (Energy 3D Cadastre for the City of Turin) Data Sink, which will be connect, through a ETL database process, to the NRG4Cast OGSA-DAI middleware.
- User Technical Administrator, through a ETL database process, will transfer data from QRS to the Web Services WSO2 Middleware Platform, Enterprise Service Bus, which enable the connection of NRG4Cast OGSA-DAI middleware with the relational database, since CSI data access is just enabled by a SOA approach.
- User Technical Administrator, through a Web Services Description Language, will connect data gathered from the system MeteoData (Open Data from Weather Station in Turin of Regional Environmental protection Agency) to the NRG4Cast OGSA-DAI middleware.

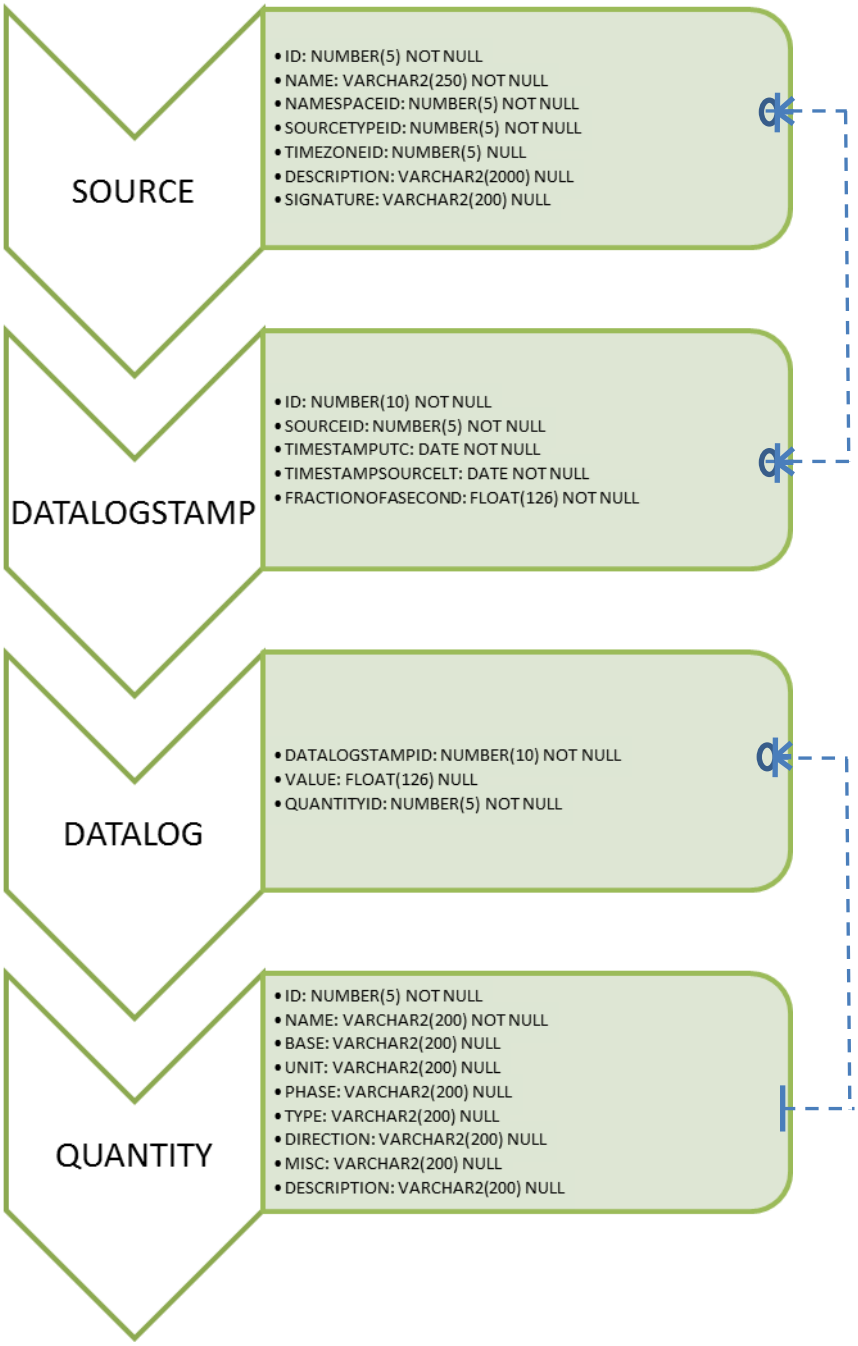

Figure 12: CSI SQL database schema.

3.1.5 Other Use Cases

Other use cases reported they do not perform any data cleaning techniques on their data.

3.2 OGSA-DAI Data Cleaning

OGSA-DAI has the capacity to perform data pre-processing (e.g. replace missing values by mean or by last value). The functionality isn't provided out-of-the box, but a separate implementation is needed. In the current setting, this functionality is not used.

3.3 Implementation

Kalman Filter with a second degree dynamic model has been chosen as the preferred short-term predictor for usage within the Data Cleaning Scenario. Kalman Filter has been implemented within Glib, a C++ general purpose library that powers QMiner. During the testing phase also Extended Kalman Filter has been implemented, which enables the usage of non-linear models.

3.3.1 Kalman Filter Setup

Initial phase of the data cleaning process is done by an expert user. This user can take advantage of the upper right part of the GUI (see Figure 14) to perform configuration of the Kalman Filter for a certain sensor. Links in the GUI invoke different actions. User can select the sensor which he wants to configure. He can prepare the data (first part of the measurements in the data layer) and set up the configuration parameters of the KF as well as the width of the gap of how much the next measurement can differ from prediction that is still not classified as an outlier. The “Execute and show result” command performs the Data Cleaning batch process on the initial dataset and returns the results in the form of a chart. An example for CSI building cooling is shown in Figure 13.

Expert user can use interactive chart capabilities (zooming x-axis, y-axis, moving, examining the data point) to analyse the results and determine whether the KF initialization parameters are optimal. Each sensor/dataset has its own properties which influence the initialization KF parameters so every sensor parameters should be set with great caution. The expert user needs to determine the confidence he should have in the learned model, he should estimate the noise in the incoming data. After those parameters are set, he should tend to minimize the tolerance gap, which will make the KF able to find as many outliers as possible and at the same time not detect too many false negatives. In the figure there are two data points which have been probably misclassified as an outlier by the algorithm. Expert user should enlarge the gap or set up such parameters that the model would be more susceptible to trend change in the data.

Downsides of KF approach and possible enhancements are discussed in the Conclusions of this deliverable.

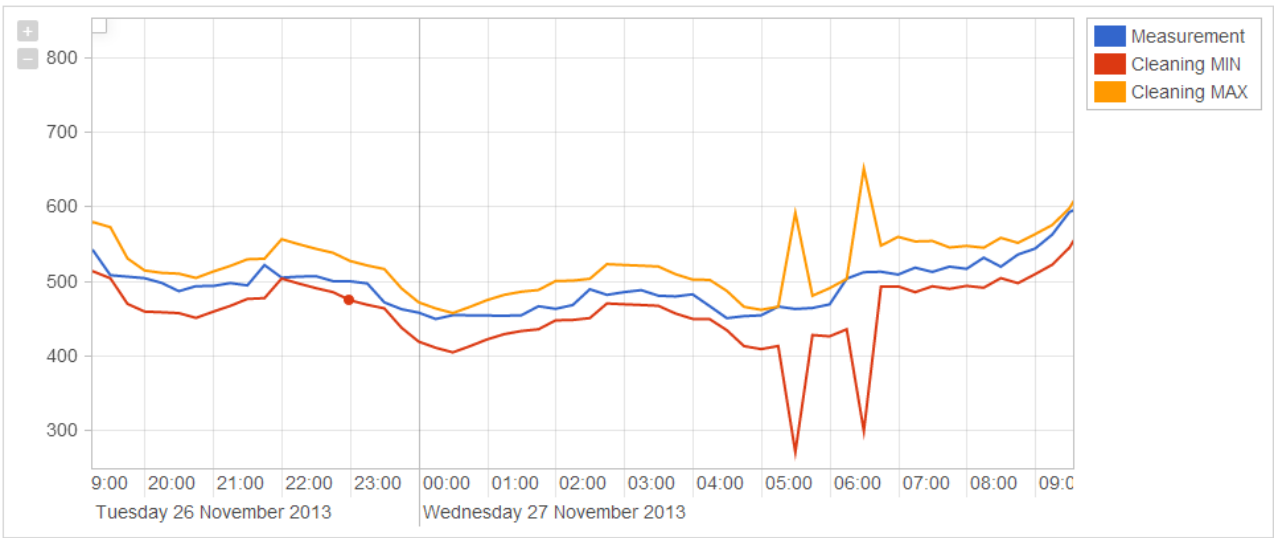

Figure 13: Example of the Kalman Filter configuration phase.

After the expert user is satisfied with the results he can store the configuration. If the expert user determines that KF approach to Data Cleaning would do more harm than good for a certain sensor, he can simply reset it (remove from the configuration) and the on-line data cleaning component will ignore this sensor.

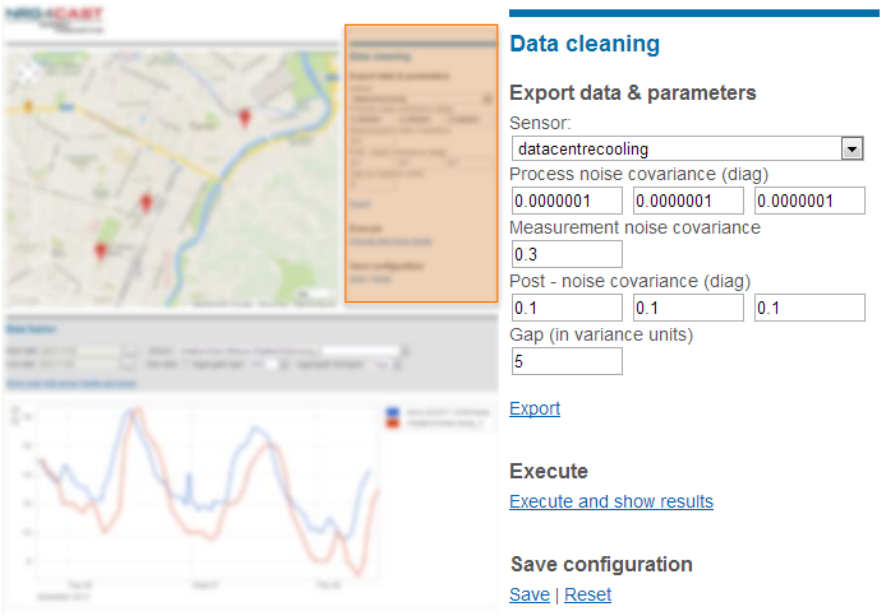

Figure 14: Part of GUI dedicated to Data Cleaning.

3.3.2 On-line Data Cleaning

Once a sensor has been configured, the configuration of the sensor can be loaded into the on-line data cleaning engine. A separate KF object is initiated for every sensor in the configuration file. When new measurement arrives it is compared to the filter prediction. If the measurement lies within the expert user defined gap it is forwarded to QMAP data layer, otherwise it is replaced with KF prediction value.

4 Data Fusion on the Pilot Data

Data fusion is a wide and not precisely defined sub-field of research. It includes a variety of techniques that can achieve many different results. In the context of this deliverable, we have focused mainly on the data fusion techniques that support machine learning and further usage of the data. This means that we have implemented data fusion techniques that can be implemented at the lower level and that demand a minimum knowledge of the domain. The techniques we have addressed are: time alignment of the measurements and preparation of data aggregates (such as average, count of samples, sum, minimum, maximum and standard deviation for different predefined time windows), preparation of geo-location based querying.

Data fusion is sometimes understood as a process of combining different sensor sources that result in new knowledge. An instance of such an approach might be for example a Kalman Filter that would combine GPS latitude and longitude data with the speed and accelerometer data, which would result in a smooth and more precise trajectory data. Such data fusion is in our opinion a subject of future work in the project.

4.1.1 Implementation of Aggregates

Aggregates have been completely re-implemented in the final prototype. More detail about re-implementation of the whole system is available in the next chapter. Data layer in the QMiner has been designed as a generic engine without the need of implementation on the C++ level. The generic engine is initialized via a JSON definition of the data tables (which are called stores). Generic engine includes all the standard DB systems functionality like defining fields with different types, keys and primary keys, joins and indexes. JSON definition of the aggregate store is shown in Figure 15.

```
{
  "name":"Aggregate",
  "id":4
  "fields":[
     {
        "name":"Timestamp", "primary":true, "type":"datetime", "featureType":"datetime",
        "aggregationType":"timeline", "displayType":"text", "store":"cache",
        "codebook":false, "shortstring":true
     },
     {
        "name":"Val", "primary":false, "type":"float", "featureType":"none",
        "aggregationType":"none", "displayType":"none", "store":"cache",
        "codebook":false, "shortstring":true
     },
     {
        "name":"AggrType", "primary":false, "type":"string", "featureType":"none",
        "aggregationType":"none", "displayType":"text", "store":"cache",
        "codebook":false, "shortstring":true
     },
     {
        "name":"WindowLen", "primary":false, "type":"string", "featureType":"none",
        "aggregationType":"none", "displayType":"text", "store":"cache",
        "codebook":false, "shortstring":true
     }
  ],
  "joins":[
     { "name":"Sensor", "type":"field", "store":"Sensor", "inverse":"hasMeasurement" },
     { "name":"Type", "type":"field", "store":"Type" },
     { "name":"Node", "type":"field", "store":"Node" }
  ],
  "keys":[
     { "field":"Timestamp", "type":"datetime" },
     { "field":"AggrType", "type":"value" },
     { "field":"WindowLen", "type":"value" }
  ]
}
```

Figure 15: Definition of Aggregate store in QMiner.

JSON definition includes 5 main categories: name and id of the store, definitions of fields, joins and keys. The aggregate store has 4 fields, which are timestamp and value of the aggregate, additional description is provided by AggrType field, which includes type of the aggregate (average, count, min, max, stdev or sum) and WindowLen field, which describes the length of the aggregate window in minutes. Aggregates are joined with Sensor, Type and Node field, which means that queries can be performed over those stores to

retrieve appropriate aggregates. Additional keys are defined over the fields, which make the store possible to be queried by them. Aggregates can be retrieved according to their type, timestamp and window length.

It is important to point out the token **"store":**"cache" from the fields' definition. Aggregate store is expected to be a huge one, containing the data for the whole history of the use cases. This data can therefore not be stored in memory. The *cache* data structure enables the data to be stored on a hard drive and loaded (and kept there for a while when used) into memory on demand.

Data stream is being pushed to the Measurement store. An aggregate manager class is registered to the OnAdd Event of the Measurement store. This means that the manager is invoked on every addition of a measurement. Aggregate manager receives the event with the corresponding (measurement) data and checks the sensor, the data is coming from. If an appropriate aggregator is already registered to this sensor, the manager forwards the measurement to it (TStreamAggrNum instance), otherwise it creates a new aggregator. When time window closes for a certain aggregate, all the data is stored into the Aggregate store. The whole process is depicted in Figure 16.

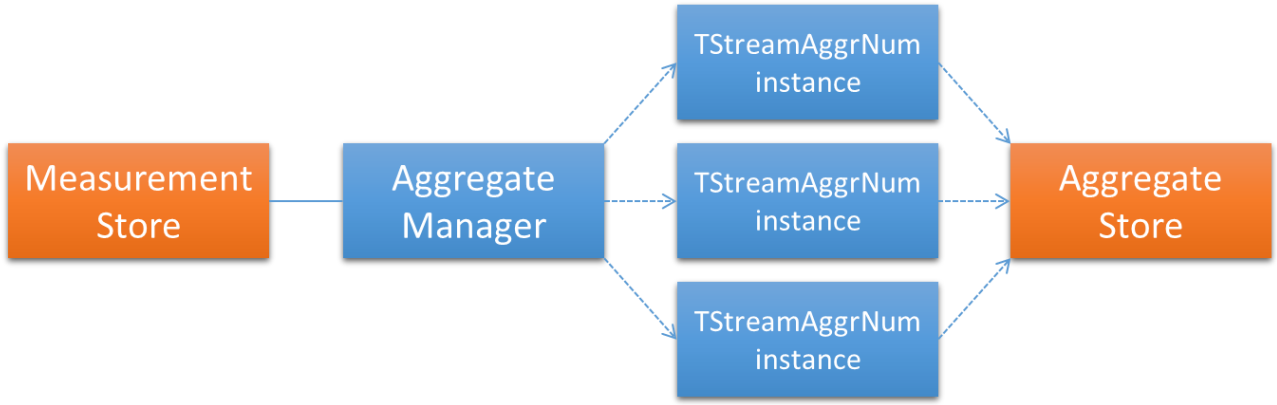

Figure 16: Schema of the aggregate calculation process.

What the figure above is not showing is that there is a separate Aggregate Manager for each time window. The following time windows have been identified as relevant: 15 minutes, 1 hour, 1 day, 1 week, 1 month. Those are implemented at the moment.

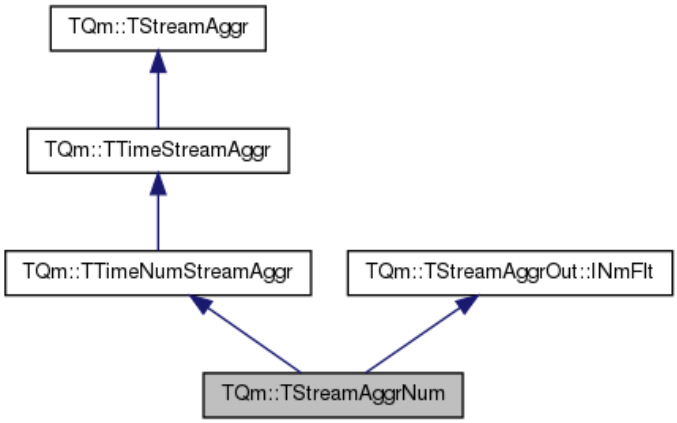

Figure 17: Inheritance diagram of TStreamAggrNum.

QMiner implements a generic engine for different types of stream aggregates, hence the complex inheritance diagram in Figure 17. TStreamAggrNum is a numeric aggregate, that is based on a stream aggregate (TStreamAggr) with addition of a timestamp data (TTimeStreamAggr) and value of the new measurement (TTimeNumStreamAggr). It enables multiple outputs with virtual methods provided by TStreamAggrOut. Outputs are depicted in Figure 18.

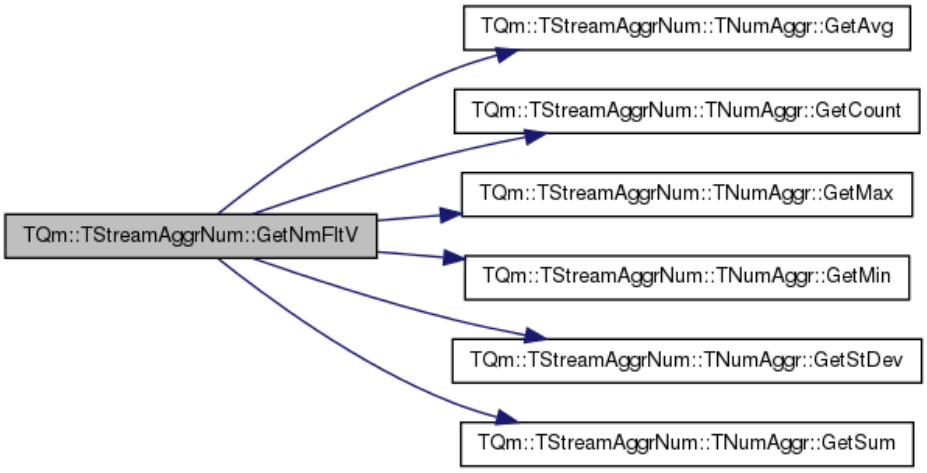

Figure 18: Aggregate retrieval functions of TStreamAggrNum.

The numeric aggregates that are calculated, are average, count (number of measurement received during the calculation of the aggregates in this batch), max, min, standard deviation and sum.

4.1.2 Geo-location based querying

Geo location data in QMiner is stored with the pairs of floats as shown in the Node store definition in Figure 19. Unique property of this field in the QMiner is that it supports querying according the distance on the ellipsoid. In practice this means that we can find all the nodes within some geo location with a single, optimized query.

```
{
   "name":"Node",
   "id":0
   "fields":[
      {
         "name":"Name", "primary":true, "type":"string", "featureType":"none",
         "aggregationType":"none", "displayType":"text", "store":"memory",
         "codebook":false, "shortstring":true
      },
      {
         "name":"Position", "primary":false, "type":"float_pair", "featureType":"none",
         "aggregationType":"none", "displayType":"map", "store":"memory",
         "codebook":false, "shortstring":true
      }
   ],
   "joins":[
      { "name":"hasSensor", "type":"index", "store":"Sensor", "inverse":"Node" }
   ],
   "keys":[
      { "field":"Name", "type":"value" }
   ]
}
```

Figure 19: Definition of the Node store in QMiner.

For this reason a separate class TGeoIndex has been implemented. This class contains a SearchRange function that yields a vector of records within a certain range of a centre location.

4.2 External Sensor Data

4.2.1 Sensor Data on Energy Prices

The acquired data sample is consisting of spot market prices, since this is one of the usually used benchmarks for electricity prices. The data used are spot market[4] prices and quantities—meaning that delivery of cash and commodity has to be done within two working days after the trade date—for all hours of the day, crawled from the European power exchange[5] web site for each day from 31 December 2009 to 23 September 2013. All the data is from German auction.

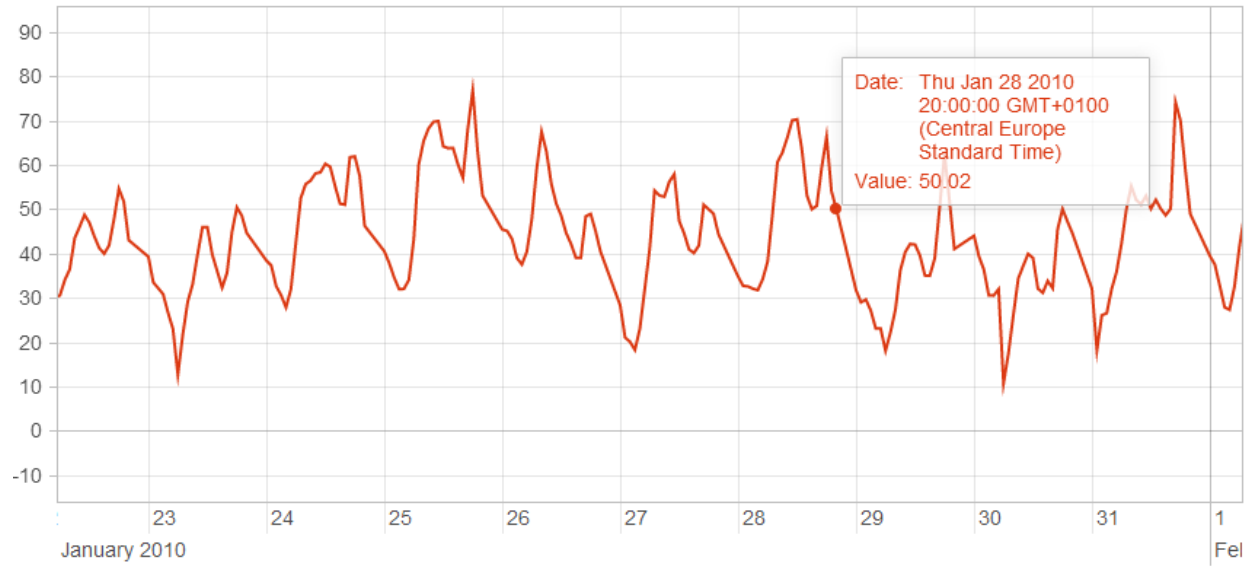

Figure 20: Visualization of energy price data.

The aforementioned data was acquired from conventional relation database format in two tables and exported into JSON format and fed to script that parsed and sent the data to QMiner instance via HTTP API. Visualization of the data is shown in Figure 20 and Figure 21.

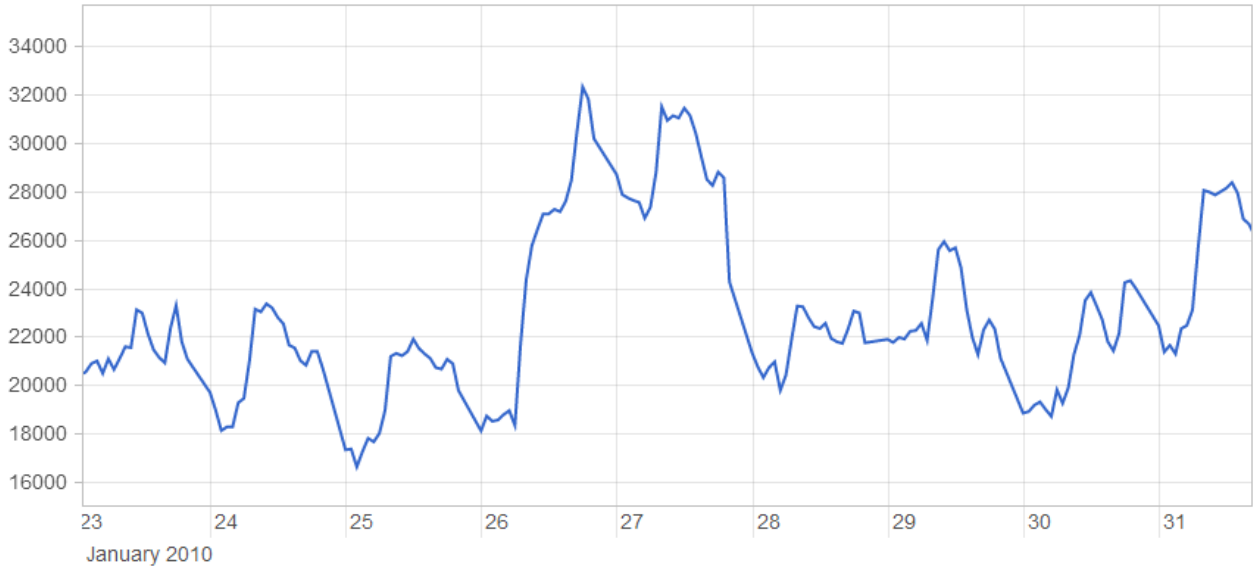

Figure 21: Visualization of energy consumption data.

[4] http://en.wikipedia.org/wiki/Spot_market
[5] http://www.epexspot.com

4.2.2 SensorFeed

As a by-product of the data fusion initiative of the NRG4Cast project we have laid a foundation for a service that is able to scan external sensor data, store it and forward it (or multiplex) to the NRG4Cast (or any other) platform. The platform implements the reduced data layer schema, which is depicted in Figure 22. We only use data about the Node (physical location with name and geolocation), Type (of the sensor with units of measurement and phenomena they are measuring), Sensor (which links Node and Type) and Measurement (which includes timestamp and value). On the top layer there is a table that takes care about all the registered feeds (URL, data about scheduling, success of the parsing job etc.) and there is of course also the Feed Type (which tells the scheduler which parser to involve with a certain sensor feed).

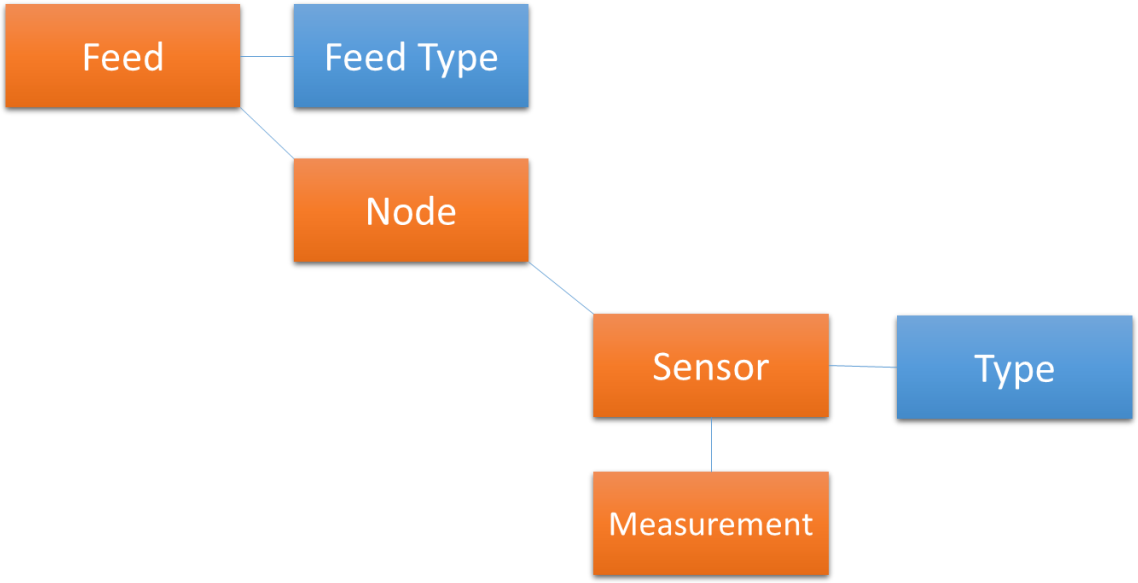

Figure 22: Reduced Data Layer Schema for SensorFeed.

The software consists of 4 main components as depicted in Figure 23: database (MySQL is used), parser engine(s) (PHP is used) and scheduler (internal Windows scheduler is used at this initial stage). Multiplexer is used to push the data to any subscribed services, at this point this would be the QMiner HTTP API engine for data streams.

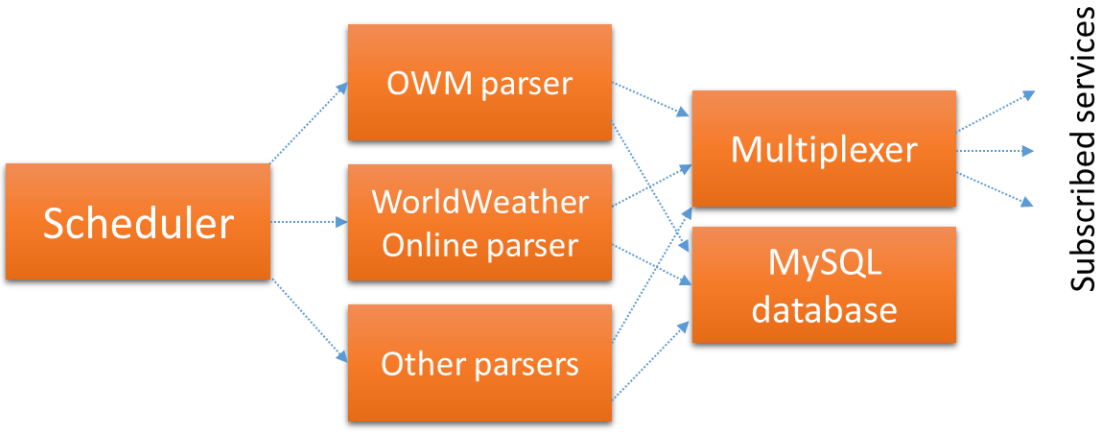

Figure 23: SensorFeed architecture.

Scheduler is taking care of invoking parsers to check for new data at the data sources. As at the moment only 2 data sources are present (Open Weather Map and WorldWeatherOnline) there is no need for special scheduling and so this part is quite simple. It will be part of a future work to implement a scheduler that will be based on a more advanced software solution (like in NewsFeed).

There are two types of parsers implemented with the SensorFeed at the moment. Those are parsers that can retrieve weather condition and forecast data from some open weather data sources on the internet. We have chosen a couple of sources to parse. Those are WorldWeatherOnline (with the initial qualitative analysis we have concluded that this source has finer granularity of measurement but smaller accuracy) and OpenWeatherMap (which is completely open source, offers more data, has better quality of the data, but the measurements are sparser). Parser retrieves the data from the data source, analyses it, compares it to the old data in the database and – in case the measurements are new – pushes the measurements to the database and to the multiplexer engine, which takes care of pushing the data to any of the clients.

In the NRG4Cast scenario we are querying both weather data sources for the data, usable for NRG4Cast pilot cases (Miren-Kostanjevica, Turin, Athens). Example is provided in Figure 24.

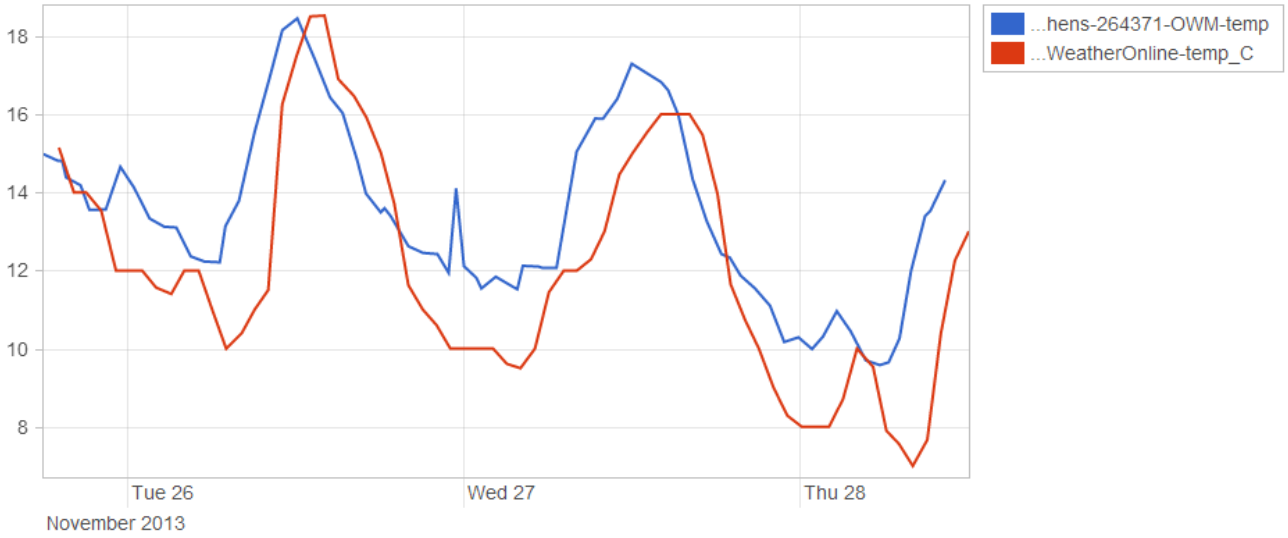

Figure 24: Temperature for Athens from 2 different open data providers.

4.3 Usage of Newsfeed in NRG4Cast

There is a lot of news around the world, caring huge piece of information with themselves. There are some tools gathering them, but none was appropriate and adaptable in such way that it would be really useful for research purposes. Therefore JSI developed its own tool for aggregating new articles from World Wide Web, called Newsfeed. During development and later, between upgrades made under research work on some EU FP7 projects (RENDER, X-like, PlanetData, MetaNet), some new ideas originated from different research areas. First task was to clean these articles and semantically enrich them; they do not help much to machines, since computers cannot read and understand human language by themselves.
The pipeline is set into several steps, as is shown in Figure 25. Before downloading and start working with news, we need to first obtain the text of the news. For this purpose the software is included in the pipeline, which goes periodically through a list of RSS feeds and a subset of Google news and tries to find new sources with news. Having all this sources, Newsfeed then gains articles. In each article, again new sources of news are searched in order to expand the source's database.

Figure 25: Newsfeed pipeline of working with news.

Texts of articles are then send through a cleaning process. During this process, useful parts of the text are find and the primary language of the article is determined. If language of an article is English then the Enrycher web service takes over and extracts more useful information from it. The enrichment part also categorizes article into a DMOZ topic hierarchy. The QMiner storage tool stores both versions, after cleaning and after enrichment process. Enrycher results are in xml format, which are very useful for future

processing of stored articles. Articles, which also have known geographical positions, are then shown on the map of the world and user can click on the article's tag and see short paragraph of it. News are also shown ordered by their timestamp as a live stream on site of the map application.
News are stored with QMiner storage tool, which brings order to saving gathered articles. Processed articles can be indexed in various ways. Most logical is indexing by timestamp of the creation of the article and by their geo-location. Articles, which are also clustered, are indexed in another useful way, by concepts, which can be found very useful when we need a specific type of articles.

Newsfeed is a tool for variety of use with its coverage of around two thousand web sites with news all over the world. For NRG4Cast project it is a perfect tool to be situated as a base for further development. We need information about different indexes (economic, social, etc.), situations connected with electricity and natural gas prices, future and past events, that might have influenced the consumption of electricity, and others. Very important part of newsfeed articles is also that they are geo-tagged, therefore we can gather just article for specific region or city for which we need to know additional information to analyse and later predict behaviour of consumption and of prices for electricity, natural gas and/or hot water. Similar advantages are present since all articles are indexed with timestamp; we can just grab those that happened during time interval when consumption was increased without any obvious reason (obvious reasons are that consumption of e.g. hot water is increased during winter period, etc.). The small subgroup of articles can be then easier connected with other data provided in the project and then processed further than big messy list of articles, from which majority of them has no influence to other data's behaviour.

5 Prototype Description

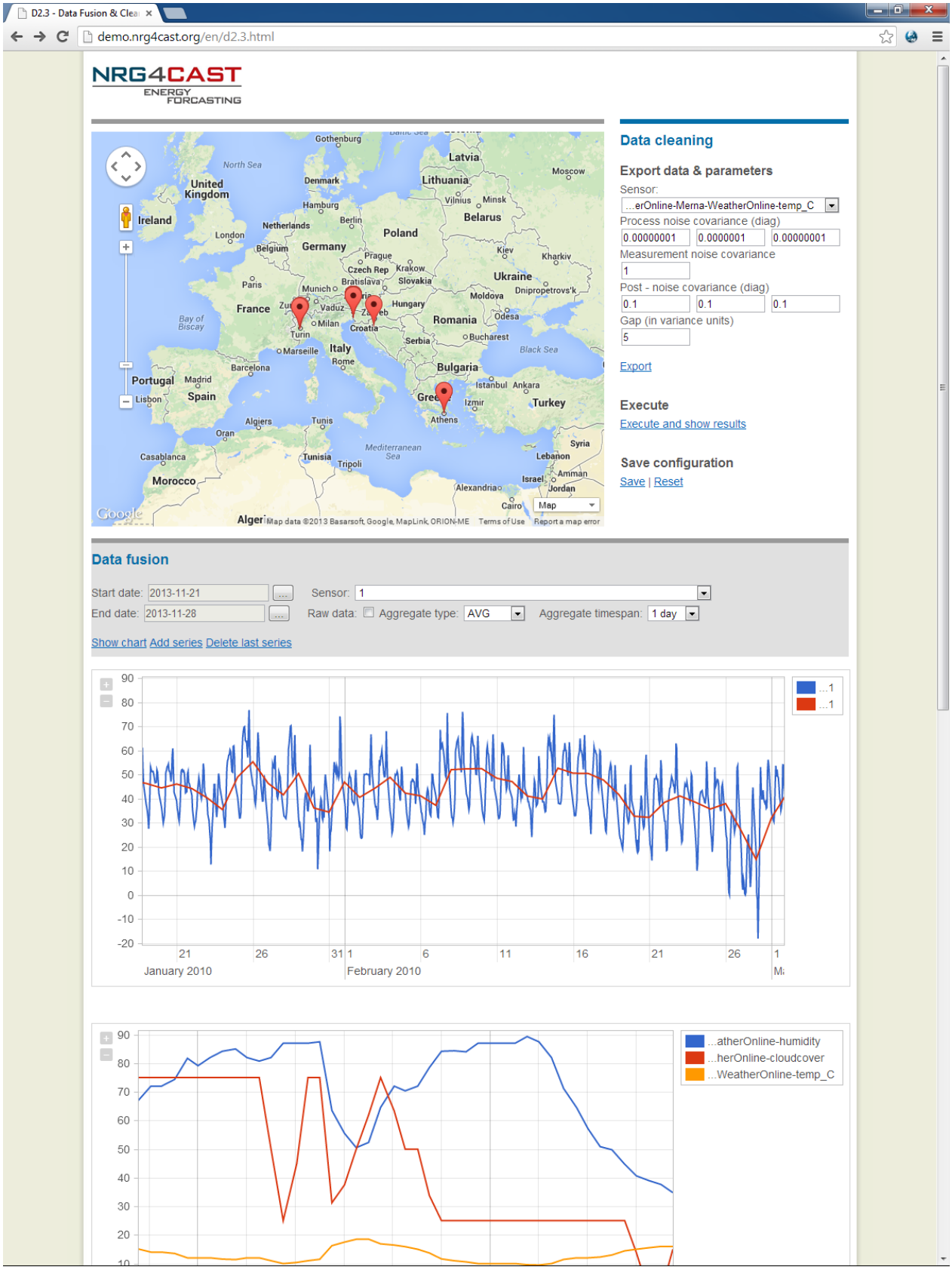

Figure 26: Prototype for D2.3 – Data Fusion and Data Cleaning graphical user interface.

Prototype GUI is depicted in Figure 26. The GUI is composed of 3 parts. First part consists of the map, based on Google Maps widget. This part of the GUI demonstrates that the data layer enables storing of

geographical location, which can be used for querying and also for a data expert to visualize the sensor nodes. Second part is dedicated to the data cleaning and is located in the upper right part of the screen. This part enables the user to select a sensor, export the related data which is needed to perform the data cleaning calibration. User interface for entering model parameters is available and as the result the user can review the results of the model and either refine the model parameters or save (or even discard) the configuration for a certain sensor. Below there is the third part of the prototype, which is dedicated to demonstration of aggregates and time sync capabilities of the system. User can create charts with multiple data series (adding and deleting is implemented). Series can either contain raw measurement data or calculated aggregates (average, count, sum, min, max or standard deviation). The charts are interactive and enable the user to manipulate the scales and offsets of the displayed data.

5.1 Implementation Details

GUI prototype uses jQuery[6] for data manipulation and user interface effects, for the charts the the Graph[7] library is used. Middleware layer that provides interface to different external sources and is also an end-point for other tools (like SensorFeed, OGSA-DAI push mechanism) is implemented in PHP/MySQL setting that is running on Apache 2.0. On the bottom layer there is a tweaked QMiner engine (based on C++) with custom JS libraries, called EnStreaM.

5.1.1 Northbound API

QMAP northbound API is dedicated to be accessed by the GUI components. It provides a simple HTTP REST-like web service with a number of functions described below.

Function:	enstream/get-nodes
Description:	*Lists all the nodes in the Nodes store.*
Input:	/
Output:	JSON array with pairs of Name and Position (which is a pair of floats).

Function:	qm_wordvoc
Description:	*Lists all the sensors in the Sensor Name index.*
Input:	keyid=2 (ID of the key in the data layer)
Output:	JSON array with pairs of *str (Name)* and *fq (frequency)*. *str* parameter represents sensor name from the Sensor store.

Function:	enstream/get-measurements
Description:	*Lists all measurements that satisfy the query parameters.*
Input:	sensorid – unique ID of the sensor (sensor name) startdate – date in MySQL format for the start of the interval enddate – date in MySQL format for the end of the interval
Output:	JSON array with pairs of Timestamp and Val.

Function:	enstream/get-aggregates
Description:	*Lists all the aggregates that satisfy the query parameters.*
Input:	sensorid – unique ID of the sensor (sensor name) aggrtype – type of the aggregate (average, count, min, max, stdev or sum) windowlen – length of aggregate window in minutes startdate – date in MySQL format for the start of the interval

[6] http://jquery.com/
[7] http://almende.github.io/chap-links-library/graph.html

	enddate – date in MySQL format for the end of the interval
Output:	JSON array with pairs of Timestamp and Val.

Function:	enstream/get-cleaning-sample
Description:	*Returns the sample of the data for data cleaning.*
Input:	sensorid – unique ID of the sensor (sensor name)
Output:	JSON array with pairs of Timestamp and Val.

Function:	enstream/get-aggregates
Description:	*Lists all the aggregates that satisfy the query parameters.*
Input:	sensorid – unique ID of the sensor (sensor name) aggrtype – type of the aggregate (average, count, min, max, stdev or sum) windowlen – length of aggregate window in minutes startdate – date in MySQL format for the start of the interval enddate – date in MySQL format for the end of the interval
Output:	JSON array with pairs of Timestamp and Val.

Table 6: Northbound API description at the QMiner instance.

Function:	save-data-cleaning
Description:	*Saves parameters for data cleaning.*
Input:	sensorid – unique ID of the sensor (sensor name) parameters – comma delimited parameters for KF (process noise paramers, measurements noise, post covariance) gap – gap size
Output:	Returns “OK” or description of the error.

Function:	reset-data-cleaning
Description:	*Removes configuration for certain sensor.*
Input:	sensorid – unique ID of the sensor (sensor name)
Output:	Returns “OK” or description of the error.

Function:	execute-data-cleaning
Description:	*Executes off-line data cleaning module with the exported data.*
Input:	none
Output:	Returns results of the offline data cleaning in a CSV format (timetamp;measurement;min;max).

Table 7: Northbound API description at the middle ware instance.

QMiner instance is accessible at http://localhost:9889/ (only locally from the machine where the instance is installed).
Middleware entry point is accessible at http://demo.nrg4cast.org/api/ and is accessible from everywhere.

5.1.2 Visualization of Aggregates and Measurements

Control panel for the data cleaning has been described in detail in Section 3. In this section we will describe the visualization of measurements and aggregates through the GUI. Position of the controls for visualization is depicted in Figure 27 and the control panel itself is shown in Figure 28.

One can visualize raw measurement (keep the Raw data checked) or aggregates in the opposite case. User chooses start and end date of the time interval, he is interested in. Datapicker widget (jQuery UI) helps user choose the dates. Next he chooses sensor from the dropdown and in case, he is interested in aggregates,

he chooses type of the aggregate (AVG, MIN, MAX, CNT, SUM, ST. DEV). Last dropdown lets him choose aggregate time window (15 minutes, 1 hour, 1 day, 1 week).

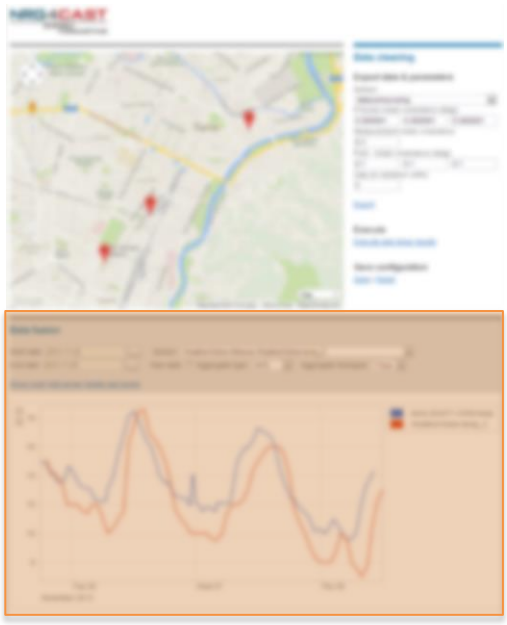

Figure 27: Position of controls for visualisation of aggregates and measurements in the GUI.

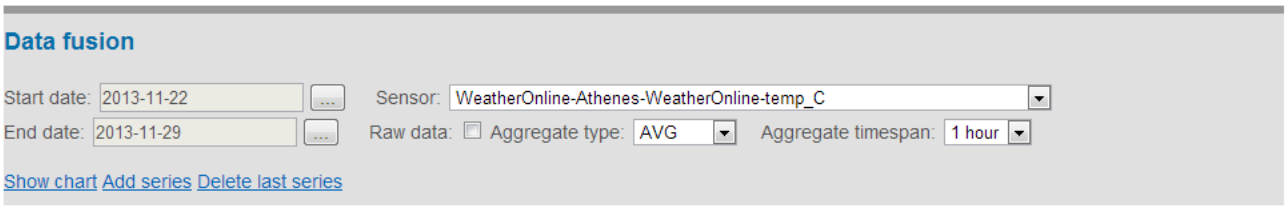

Figure 28: Controls for visualization of aggregates and measurements.

User can then create a new chart with “Show chart” command. If there a chart already exists, he can add or delete new data series from it. He can for example:

- visualize the raw measurements and the corresponding aggregates
- visualize same phenomenon measured at different sensors
- visualize different aggregates of the same sensor
- visualize different sensor data and get some intuition about the interconnection of both measured phenomena
- etc.

6 Conclusions and Future Work

This document describes the theoretical foundations and implementation details of Data Fusion and Data Cleaning final prototype.

For data cleaning we have proposed a methodology that in our opinion best suits the nature of most of the sensor streaming data nowadays. The proposed algorithm (Kalman Filter with second degree dynamical model) is self-sufficient and does not require any additional domain knowledge. The used feature vectors are only generated from the sensor data itself and the model only relies on the configuration data provided by an expert user.

Algorithm and model have been chosen based on the analysis of a few methods/models. Analysis included performance on the real and artificially generated dataset. Experiments also revealed an instability of the proposed method. The approach has been successfully addressed with a modification of the algorithm.

Many interesting ideas have been identified for the future work. A very big problem when using the Kalman filter is initialization of the filter. One needs to optimize the behaviour of the filter to a large number of parameters (approx. 20 for a 3-dimensional model). It would be interesting to investigate an optimization with gradient descent or other efficient methods (Levenberg-Marquardt), where the measure to minimize would be the χ^2 measure.

We have developed a mechanism for time alignment and enrichment of the sensor data. On the basis of measurements we calculate different numeric aggregates with different time windows. Aggregates can be used by data mining/machine learning techniques. We have also provided geographical indexing for the positions of the sensors.

Laboratory installation of the prototype is available at http://demo.nrg4cast.org/en/d2.3.html.

References

[1] K. Kenda, J. Škrbec and M. Škrjanc. Usage of the Kalman Filter for Data Cleaning of Sensor Data. In proceedings of IS (Information Sociery) 2013, Ljubljana, September 2013.

[2] J. Škrbec, K. Kenda, NRG4CAST D2.3 – Data Cleaning and Data Fusion – Initial Prototype. NRG4CAST, May 2013.

[3] S. Jeffery, G. Alonso, M. Franklin, W. Hong, and J. Widom. A pipelined framework for online cleaning of sensor data streams. In ICDE, page 140, 2006.

[4] B. Khaleghi, A. Khamis, F. O. Karray, S. N. Razavi. Multisensor data fusion: A review of the state-of-the-art, Information Fusion, Volume 14, Issue 1, January 2013, Pages 28-44.

[5] R. E. Kalman. A new approach to linear filtering and prediction problem. Journal of basic Engineering, 82(1):35-45, 1960.

[6] R. G. Brown, P. Y. C. Hwang. Introduction to random signal and applied Kalman filtering. John Wiley, New York, 1996.

[7] P. S. Maybeck. Stohastic models, estimation, and control, volume 141 of Mathematics in Science and Engineering. Academic Press, 1979.

[8] N. N. Vijayakumar, B. Plale. Missing Event Prediction in Sensor Data Streams Using Kalman Filters, Knowledge Discovery from Sensor Data, 2009.

[9] Y. Hamodrakas et al., NRG4CAST D2.4 – Data Distribution Prototype. NRG4CAST, November 2013.

[10] H. W. Sorenson. Least-squares estimation: from Gauss to Kalman. IEEE Spectrum, vol. 7, pp. 63-68, July 1970.

[11] Dan Simon. Kalman Filtering. Embedded Systems Programming, pp. 72, June 2001.

