Towards an air pollution health study data management system - A case study from a smoky Swiss railway

Evangelia Papoutsoglou*, Argyrios Samourkasidis*, Ming-Yi Tsai†, Mark Davey†,
Alex Ineichen†, Marloes Eeftens† and Ioannis N. Athanasiadis*
*Democritus University of Thrace
Xanthi, Greece
Email:(papoutsoglou.e@gmail.com, argysamo@gmail.com, ioannis@athanasiadis.info)
† Swiss Tropical and Public Health Institute
Basel, Switzerland
Email:(m.tsai,mark.davey,alex.ineichen,m.eeftens@unibas.ch)

***Abstract*—In air pollution health studies, measurements are conducted intensively but only periodically at numerous locations in a variety of environments (indoors, outdoors, personal). Often a variety of instruments are used to measure various pollutants ranging from gases (eg, CO, NO2, O3, VOCs, PAHs) to particulate matter (eg, particles smaller than 2.5um: PM2.5, PM10, ultrafine particles: UFP), and including other environmental parameters such as temperature, relative humidity, GPS position. As a result it is always a significant challenge for researchers to effectively QA/QC, combine, and archive these data so as to reliably assess people's exposure to poor air quality. With the CEDAR system presented here we aim to provide a solution to this problem by employing a platform using templates for easily reading custom formatted files, apply rules for filtering and quality checking measurements, and ultimately publishing them as services on the web. The system is demonstrated for the case an air quality project conducted in a Swiss railway station where smoking is allowed.**

I. Introduction

Environmental data collection and analysis are an integral part of many scientific pursuits. Environmental data management tasks exhibit extreme variation for several reasons. The nature of each project or study significantly impacts the data management techniques employed (if any at all). Each discipline has its own preferred methods, including nomenclature and standards. Many details are domain-specific, hence well known, and several decisions are ad-hoc. Last but not least, the scope of each project is typically limited to the problem at hand, which narrows down the intended scope of data management tasks to the very immediate and short-term actions. As a result, there are few tools available for efficient environmental data management, curation and sharing.

In this paper, we present our developments towards a system that efficiently manages sensor-borne data, called CEDAR. CEDAR (stands for Customizable Environmental Data Archive) aims to be domain-independent and customizable, so that it accommodates the needs of different domains and studies, while at the same time provides simple templates for common sensor and data management tasks. CEDAR focuses on time series operations, as they are integral to sensor-borne data. The CEDAR platform is designed so that sensor and data semantics are explicitly declared, along with processes accounting for the nature of different measurements. Built on top of rich semantics, CEDAR is capable of providing standardized services, for data archival, curation and sharing. Mechanisms for intra- and inter-unit operations are set in place, to respond to user-defined queries. There are also features for common tasks, such as extracting derivative quantities, performing quality checks (QC) and quality assurance (QA) tasks.

From an architectural point of view, CEDAR is built on templates, so that general solutions can be re-applied easily and dynamically, i.e. without the need to recompile the software or append the database. Our assumption is that sensor data sets become available in custom file formats, which are stored by the system "as is". Each sensor output file is accompanied with template files, which enable CEDAR to not only read the data, but also to *understand it*, from its semantic metadata. Templates contain instructions on how to parse the data as well as annotations of the observed quantities and phenomena. In this way, data sets are stored as measurements of certain phenomena in the common database, and extra functions are enabled for quality checking, extracting derivative quantities, and summaries. These are also enabled through templates. Last, but not least, environmental datasets are offered on the web as services, which allow for end user applications to query and retrieve them through a graphical user interface (GUI).

In this paper, we outline the CEDAR framework and present its design and implementation. Our first experimentations with real world data to demonstrate the system operation are in the domain of air pollution health studies. The main driver for this choice is the variety of pollutant monitoring equipment employed in such studies. In this paper, we present our experiences with managing data from a study of a Swiss railway station, where smoking is still allowed. Scientists collected data using both fixed stations and moveable equipment. The variety of data is being assembled via templates into an archive and presents the challenges of working with such data and what the next steps in this process could be.

II. Related Work

In environmental sciences, sensors are tools that produce measurements, which serve as ground truth observations for subsequent studies. In principle, sensors can be of several kinds, each

 ISBN: 978-87-7903-712-0

addressing different needs and concerns, and, correspondingly, their output can also vary. Combining many of them for a single study, and analyzing the collected information in an organized manner often poses a new challenge: not of discipline or of science, but merely one of appropriate management. Effective management is a prerequisite for certain aspects of data analysis [1].

The aggregation and collective processing of many heterogeneous datasets, which nevertheless refer to the same core physical quantities for the same purpose, are very challenging to implement. An approach in which semantic annotations are not only directly tied with the actual data, but also considered for every operation, can circumvent these challenges and lay the groundwork for a unified and semantically-aware management system [2], [3]. Lee *at al*. [4] delve further into these challenges and present the Concinnity platform to addresses concerns about trustworthiness.

Software tools as EQuIS [5] and TerraBase [6] software, provide a wide range of operations. Both accept datasets, subject them to arduous QA/QC processes, and store them in their respective underlying databases. The database management systems are also complemented with data visualization modules, with specific emphasis on geological parameters.

A. Air pollution health studies data

In air pollution health studies, measurements are conducted intensively but only periodically at numerous locations in a variety of environments (indoors, outdoors, personal). Often a variety of instruments is used to measure various pollutants, ranging from gases (e.g., CO, NO_2, O_3, VOCs, PAHs) to particulate matter (e.g., particles smaller than 2.5μm: PM2.5, PM10, ultrafine particles: UFP), and including other environmental parameters such as temperature, relative humidity, GPS position. As a result, it is always a significant challenge for researchers to effectively QA/QC, combine, and archive these data so as to reliably assess people's exposure to poor air quality. Now, with the advent of cheap real-time sensors and the growing internet of things, there is an increasing need for well-designed approaches to archiving these data, so that they can be flexibly accessed with ease by researchers as well as the general public.

In our demonstrator, we deployed our framework for the case study of an air quality project conducted in a Swiss railway station where smoking is permitted. The pollutants monitored were PM2.5, UFP, black carbon, nicotine, temperature, relative humidity (RH). There was one fixed site, and seven other train station micro-environments surveyed by field staff carrying real-time instruments.

B. Time series data management and fusion

Management systems that can adequately process time series data have been in the spotlight of the inter-disciplinary research community. A lot of efforts have been put into developing management systems, specifically designed either for real time adaptation and processing of time series data, or for archiving them and providing a query service over them.

In GALILEO [7], database implementation is of great importance, because the system is optimized for storing streaming data. In order to cope with node failures, which would lead to data loss, they replicate data storage nodes. Thus, GALILEO needs a lot of computational power to function.

TSMS (Time Series Management System) is a specialized object-oriented Database Management Systems for te banking industry [8]. The system they envisioned should be able to interact with and facilitate the functionality of external programs (e.g. modelling platforms, decision support systems), and apply filters on existing time series. In addition to that, the aforementioned system should cope with data reliably, assigning tags on each measurement.

Mason et al [9] introduce the Virtual Observatory and Ecological Informatics System (VOEIS) Data Hub, a centralized data management system which curates environmental observations through data life cycle. Metadata annotate every measurement and templates are utilized for the data input process. While templating facilitates input process, VOEIS software implements it only for specific types of files (csv, xls, xlsx), which renders our input approach more robust and promising.

In AiRCHIVE [10], the authors implemented an autonomous environmental data archival system, which received input from sensors attached to it. While efforts were concentrated on incorporating web protocols which enhance interoperability, little efforts were allocated on input and data storage methods.

Time series data often comes in very high volumes, so using a document-based system like MongoDB is a common solution. User activity monitoring [11], server logs and real system records are all common applications; however, the systems implemented for those tasks rarely need to address needs other than data storage and very elementary operations. In fact, distributed processing techniques are often pursued for the purpose of analysis [12]. Additionally, efforts have been made to resolve such issues not by implementing new management systems, but by developing interfaces for aggregated information retrieval from different source databases [13], [14].

Ghanem et al [15] discuss data integration techniques on a grid-based collective system, but ultimately propose the design of different APIs for each specific workflow. They also mention the potential in semantic tagging for such information, reinforcing our belief that our system truly has potential.

C. Terminology

Concerning the vocabulary used in the following sections, and in order to avoid any ambiguity, some terms have to be defined first. A *measurement* refers to a set of values recorded *simultaneously* by a single sensor. The number of values recored is entirely dependent on the sensor used, but it is at least one.[1] Note the

[1] As an example consider a GPS sensor device that yields two or three values: latitude, longitude and altitude (optional).

constraint: each measurement must state its time of collection. Furthermore, each measurement may carry several tags associated with it. A *time series* consists of several measurements, i.e. pairs of time-stamped values. It refers to a single physical quantity observed over a time interval. Time series are regular when measurements are taken at constant time intervals. They may contain missing or erroneous values, as they represent the raw output of the sensor instrument, without other processing.

III. System specifications

The main objective of CEDAR is to provide a means for easier data management and analysis for sensor-borne data. Air quality health studies data comprises numerous measurements, heterogeneous in both meaning and format, as they are collected by different equipment. Our goal is to organize all measurement datasets and make them available via interoperable, queryable services. In the following sections we present the key features of our system.

A. Requirements

A CEDAR project involves the curation, archival and processing of several time series, using one or more sensors, each observing different phenomena. Each sensor produces a custom text file that contains that results of each measurement for a certain period of operation, and all these files are made available to CEDAR.

The first step is to make explicit the semantics of each observation process occurring. This inevitably entails the identification of the physical meaning and unit of the measurements produced by each sensor, which can be associated with other factors of interest (e.g. substance, medium, etc.) or spatiotemporal references. Undoubtedly this depends on the specific needs of each project. Domain scientists need to be involved in this step to make such information available, using the appropriate metadata templates. Templates depend on the type of equipment, as each instrument typically produces its own file format. The system may also extract information from the underlying folder structure, as scientists tend to encode valuable information there, which needs to be extracted and processed. Metadata templates serve as the source of some primary QC statements: minimum and maximum allowed values for each measured quantity can first be stated there, and then used to fix those not complying.

Secondly, CEDAR is able to execute common time series operations. These may occur at predefined time spans (e.g. hourly, weekly, monthly, etc.) or at user-specified ones. The system provides built-in support for the most common operations, such as calculation of maximum, minimum, rolling mean, average and percentile values. Additionally, the user is able to specify more functions via an extension mechanism.

Thirdly, CEDAR combines existing measurements into derivative user-defined quantities, and appropriately tag them. This includes operations across different units (and unit conversions), as well as other transformations. Newly calculated information, based on a user query, can be optionally stored into the archive for future reference. Such operations include quality test and assurance procedures, which can be done in two stages: Either when data is initially imported into the archive, according to existing rules given by the user, or later by revisiting datasets and expanding them with QA/QC tags.

The system may also offer additional services: missing or suspect measurements are detected and tagged as such, and basic value correction functions (e.g. due to calibration errors or offsets) may also implemented via templates.

Last, but not least, data archived by CEDAR and most of its capabilities must be offered as services over the web. A RESTful implementation was deemed preferable, to enhance transparency and promote scalability, among other advantages. A GUI was developed in order to provide users with the means to submit queries and visualize data, effortlessly. Data dissemination in a interoperable way, an essential feature for a sensor data management system, was addressed by adopting Sensor Observation Service (SOS). SOS is an Open Geospatial Consortium (OGC) standard, which allows querying observations and sensor metadata, employing a set of predefined and well documented requests.

B. Abstract architectural design

The CEDAR platform operates in four stages, shown in Figure 1. Each input needs to be accompanied by two template files. The input template, and the metadata auxiliary file. They allow the template reader component to successfully identify and tag the information in each input data file. The **template reader** goes through each input data file, and parses its text according to the instructions in the input template. The metadata auxiliary file provides with additional semantics for constructing a timeseries of measurements from the data. It may also provide with additional information and optionally filters suggested to generate derived timeseries or perform common tasks as QA/QC. The semantic auxiliary file may also associate the quantity measured with one or more corresponding ontologies, which subsequently allow for easier collaboration between different people and teams, in compliance with the semantic web vision.

In the second stage, the newly formatted and semantically-tagged time series are imported into the **database**, which constitutes the innermost layer of the application. For reasons explained below, the document-based database MongoDB was deemed suitable for this implementation. At the top level, time series are categorized according to the study they are part of, the sensor that produced them, and optionally by a more sophisticated structure, suitable for each project. CEDAR always preserves the raw data, as initially imported, along with the results of the QA/QC process.

The database component is connected with the **processing layer**. This component intercepts all data from the database, and translates each time series into a set of objects for further processing. It is also here where the more complex operations are executed, as they become significantly easier with the object-oriented structures. The object-oriented application layer can also accept custom user filters and use them in its operations.

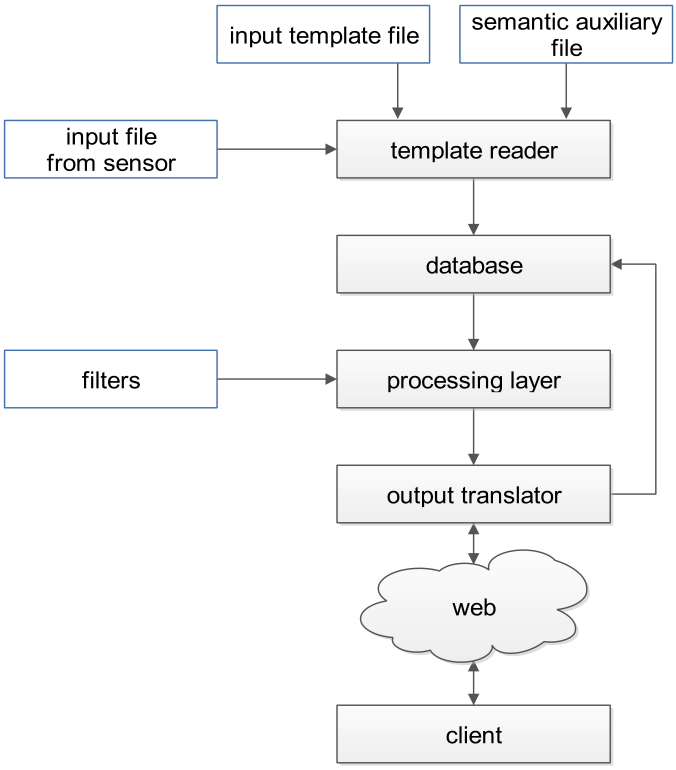

Fig. 1. CEDAR abstract architectural design

The **output layer** is the module in direct contact with the end-user via the web. It accepts custom queries from the graphical user interface, and subsequently translates them in a way the object-oriented layer can interpret. Furthermore, this component is responsible for resolving user queries, as well as for preparing the responses to them.

An additional component is the client interface. An interactive GUI is preferred, as it simplifies the querying process, and is able to present the user with additional features, executed on the browser. These includes on the fly visualizations and transformations.

IV. Detailed design

In this section we provide an in-depth description of each component and its function and implementation details.

A. Template reader

The template reader is the component responsible for the initial extraction and composition of each time series from an input file. As mentioned above, two additional files are necessary for this process. We have developed the template reader from scratch and it was written in Python. The input template file is a text file that mimics the structure of its corresponding input file. Usually these input files consist of a header section, with details pertaining to the entire measurement set, and a table-like section containing the measurements themselves. Both sections can include valuable data that we want to acquire. It is imperative that the structure of the template file and the input data file be identical, as anything else would result in complications. This includes whitespace, tabs or other non-visible characters, which is very important for the detection of irregularities (in particular, missing values). The template reader is able to handle any text file as input, ranging from the easily-parsable CSV files, to the less strictly-formatted text files. Our current implementation uses regular expressions, which are derived from the template file.

We sought a simple way to compose input templates, and concluded that a pattern similar to popular output template libraries as Cheetah [16], or Mako [17] would be sufficient. In these, constant text is simply represented as such, and variables follow a specific pattern. Mimicking that, we use the `$` identifier to mark variables, followed by the variable name surrounded in curly brackets`{ }`. The result of this format is of the type: `${variable_name}`.

Variables may be found in both header and tabular sections. Typically in headers we have some kind of simple key-value assignments of values to variables. Some of them could be optional. In the tabular section we need more constructs to fully describe the content, as we need to associate values with references (temporal or spatial coordinates) and possibly an unknown number of occurrences.

To signify that a set of rows should be processed according to a common template line we introduce the `<%title>` tag, inserted into the line before the template line in question. The line following the `<%title>` becomes the common pattern, and is used for all lines following the subsequent <%process> tag. These tags are always expected to appear together, and mark a tabular portion of a file.

An assumption made should be noted here: we expect that each measurement is fully contained in a single line, and that each one of them includes a time stamp. Template files ignore consecutive whitespace characters, as these can be inserted by various sensors liberally to enhance the readability of their output files. Unfortunately, this also implies that spaces cannot be reliably used as delimiters in the measurement section, as missing measurement values would complicate the proper parsing of a file.

This way, given one piece of an input file, we can effortlessly create a fitting template by only editing it, and thus specifying the variables we wish to be read by the framework, as shown in the code segments below.

Code Segment 1: Example input file. Note that (·) stands for a single space and (→) signifies the tab character.

```
Filename:·4A14Q401.TXT
Averaging·Period:·1·sec
Date·and·Time:·2015.02.03·09:30:00
Offsets:··-0.23→··-0.41

Time→Batt→Temp
0→·8.32→17.7
1→·8.32→17.5
```

Code Segment 2: Corresponding input template. A template file is expected to be found with the `.tmpl` file extension.

```
Filename:·${filename}.TXT
Averaging·Period:·${avPer}·sec
Date·and·Time:·${datestamp}·${timestamp}
Offsets:··${offset1}→··${offset1}

<%title>
${Time}→${Batt}→${Temp}
<%process>
```

The metadata auxiliary file contains all relevant semantic meta-information for each sensor type. Each variable defined in the input template has a dedicated section here, which includes all metadata and tags that need to be indexed and archived in the database. Semantic awareness relies on these files, so this is a very crucial component. Most commonly, each variable declares its physical quantity, units, spatiotemporal reference and custom properties may be added as necessary. A short example follows:

```
timestamp:
  quantity: time
  units: hours:minutes:seconds
  format: HH:MM:SS
Time:
  physical quantity: offset
  units: second
Temp:
  quantity: temperature
  units: Celsius
  medium: Ambient Air
  sml-identifier-shortName: DHT22
  sml-identifier-sensorID: AnalogTempSensor
```

Names are case-sensitive, and variable names should always be unique in each template. As mentioned earlier, all variables are expected to have the first two fields, namely `quantity` and `units`. Other predefined fields, such as `format`, can be used to indicate the format of the variable to be read. Furthermore, a user can define own fields (e.g. `medium`) to specify more properties to be indexed for that variable, which are subsequently available to the processing layer. The metadata auxiliary file is formatted as a common `.yaml` file [18].

B. Database

The database selected for this project is the NoSQL document-oriented database MongoDB [19]. As opposed to its traditional relational counterparts, a document database presents the following redeeming features for our given scenario:

1. More efficient and easier handling and management of the potentially huge "tables" which hold the set of each kind of measurements, as regards table sparsity.
2. No obligation of adherence to a predefined database schema; increased adaptability to new formats of sensor inputs, domains of application, etc.
3. Excellent horizontal scalability (scaling out), as project datasets (or even different time series, on a lower level) are split and saved as separate documents.
4. Connected information, such as elements of the same time series and their direct derivatives (e.g. averages), can all be easily retrieved from the storage in relatively contiguous read operations, as documents display less fragmentation.

The above is directly derived from the nature of document-based databases: as the name implies, their elementary storage and basic structural units are documents, which directly correspond to files. Each of them can follow a different schema, and allows for further nested documents inside it, as well as references to other documents. Document size is generally on the scale of a few megabytes up to a maximum of 16MB, which is appropriate for even exceedingly long time series data.

In our case, each time series is stored in a single document. Splitting a set of measurements with numerous variables into as many independent time series provides certain benefits: document size can be kept to a minimum, and queries requesting information about a single variable need not be subject to the overhead involved in reading all data collected in that specific session from the sensor. It is also desirable to avoid over-fragmentation of data in many document files, so it is conceivable that homogenous measurements from consecutive measurement sessions –potentially recorded from the sensor into separate input files– can all be included in the same document. Another benefit of splitting the data into individual time series is the possibility of incorporating more data, in the form of e.g. tags for each measurement as a result of a user query, into the same file more easily.

Semantic fields and tags, as drawn from the semantic information file in the templating stage, are common to all associated measurements, and are placed at the root of the document. The measurements themselves is structured in the way described below, similar to the technique seen at [20]. Date and time is expressed by means of nested documents at varying depths, and creating in this way multiple sub-documents. Year information is placed at the top of the hierarchy, followed by month and then day in their respective subdocuments. At this point we can also separate each day into hours or even minutes or seconds, depending on the granularity of the data provided. A side advantage of this is also found in keeping subdocument volumes low. The lowermost level of the hierarchy consists of key-value pairs; time information is expressed in the former, while the latter is another document containing all the information associated with a single measurement. This includes the actual values measured at that specific point in time, as well as custom tags as provided by the user or calculated by the system.

This way of separating date and time information into successive subdocuments benefits the type of operations we would like to perform on each time series. Calculations of quantities such as averages, maxima and minima can be easily executed, stored and retrieved for each of the subdivisions. Other valuable information, such as the number of measurements with specific tags over easily defined time periods (e.g. over a month) can also be explicitly stored at a corresponding document depth.

C. Processing layer

This layer consists of two sub-components associated with core data processing operations. The first sub-component connects to the database and creates measurement objects. The second is able to apply operations on those objects by applying filters (operations) in order to generate derived measurements or other results. In the processing layer, each object represents a measurement, or a set of measurements combined in a meaningful manner. Variables store the IDs of each element of a measurement, their corresponding time stamps, units, and other tags. The output layer reads each part of a user's query, and determines which time series should be read from the database. Each element of that time series is transformed into an object holding all of its data, and these objects are then assembled into one primary time series object. The time series object also draws from the database any information pertaining to all objects-elements it encloses, which are at the very least annotations for the physical meaning of their values as well as their units. Furthermore, the same time series object is instructed to mark the methods that will produce from it a secondary time series object with all necessary properties and values, as a step towards answering the user's query. Finally, one or more time series objects are passed to the output layer, as they contain the final response information.

D. Output layer

The output layer acts as the front end of the system to the outer world. It receives queries from the web, either from end users via the GUI or from other machines via REST-full protocols, and responds to them appropriately by calling the appropriate objects and activate filters in the processing layer. As regards the output, this component simply transforms the objects produced in the processing layer into the appropriate format. This can range from simple serialization into text (e.g. XML, JSON, YAML formats) to commands that would insert the information contained in the objects and queried for by the user back into the database itself for future use. Output templates can also be used at this stage, so the results of processing can be presented to the user in the form desired. The templating engine Mako [17] was used in this project for this purpose. In order for this feature to work, the output translator must also be able to connect the variable names inside a template file with their associated query results, since these engines generally function with text substitution.

Following the submission of a user query to the system, the input translator's first task is to formulate a query toward the database and then pass it to the object-oriented layer, where it will eventually be submitted to the database. Simple queries, such as the presentation of all measurements in a time series, only involve a direct request to the database and appropriate formatting of the data returned. More complex ones would necessitate more object structures and potentially more queries to the database, so all relevant information can be retrieved. The most important operation of this translator lies in defining these object structures as would best suit the given task, the relationships between these structures, and queries to draw their data from the database. Among the data structures provided there has to be one that will suit the result. Especially considering that this result might be stored back into the database, regardless of what the output presentation to the user entails, the resulting object has to contain all data necessary to uniquely identify its sources and be stored with references to them, if not enclosed in its parent time series. Finally, it also provides the logic to answer a user's query, expressed in terms of the objects that the object-oriented layer will construct under the guidance of the input translator.

E. Client interface

This component is responsible for presenting the user with a practical graphical interface, and transmitting the choices made by the user to the server hosting the service. The GUI draws from the database information concerning:

- available physical quantities, as expressed in the semantic information file
- tags accompanying these physical quantities
- sensor information
- available time information.

With the above the user can select their desired inputs, apply filters among a collection of predefined and user-defined ones, and specify the desired output format. Time granularity is also an important parameter which can affect the presentation of the output.

Inputs can be processed individually, i.e. depend on a single physical quantity time series to produce the output. For instance, this would be the case for an averaging operation. More inputs can be combined for more complex processing, where accounting for more variables over certain time periods is preferable.

F. Sensor Observation Service interface

SOS component is currently under development, but when available, it will provides an interface to make CEDAR's sensors and sensor data archives, accessible via an interoperable web based interface. *Core* profile, which is defined in the SOS specification offers three operations. The corresponding *requests* to these operations are:

- `GetCapabilities`, responds with a self-description of the service. It includes detailed information regarding the hosted data and the available operations.
- `DescribeSensor`, responds with metadata about the requested sensors and sensor systems (`procedure` key).
- `GetObservation`, responds with measurements of a selected quantity (`ObservedProperty` key), measured by a specific sensor (`procedure` key) at a specific time (`eventTime` key), in a requested response format (`responseFormat` key).

V. Implementation

A. Case study

Since May 1st 2010, Switzerland has had a federal law against environmental tobacco smoke (ETS) in enclosed spaces that are publicly accessible or where workers are present; however, smoking is allowed in separate smoking rooms, open spaces and in private homes. In terms of public spaces, there has been much controversy around smoking in hospitality venues: e.g., restaurants, cafes, bars, but minimal discussion about smoking in quasi indoor public environments such as train stations with the exception of several newspaper articles about ETS in Zürich's main station during Oktoberfest (www.tagesanzeiger.ch, 2014). Switzerland has the densest train network in Europe and railway travel is the main mode of transportation in Switzerland with Zurich, Bern, and Basel having >400'000, >200'000, and >100'000 passengers per day, respectively. However, despite general public reliance on this travel mode, smoking in train stations is hardly restricted. [21]

In the interest of assessing if the air quality in a Swiss train station presented a potential public health concern, we conducted a pilot measurements study in Basel's SBB main train station in the Fall of 2014 [22]. The pilot consisted of taking measurements with a backpack of direct-reading instruments from mid-afternoon to early evening. The team made four circuits/loops around the train station stopping at 7 pre-selected locations where 6-minute measurements were made (~1 hour/circuit). Another team operated a reference site at one location on the Passerelle (main elevated area of the station where most shops, food stalls, and restaurants are located with access to nearly all tracks). Additionally, nicotine measurements at each of the 7 locations were made in the 4-day period leading up to and including the afternoon of sampling. The pollutants measured were particulate matter of size 2.5um and less (PM2.5), ultrafine particles (UFP, between 10 and 300nm), and black carbon (BC) all at one second resolution. The PM2.5 mass measurement represents the finer fraction of particles that can penetrate to the deeper airways of the lung; UFP particles measured in number of particles per cubic centimeter are sub-micrometer particles that are almost entirely generated by combustion processes; and black carbon is an indication of the darkness of the aerosol and, in the ambient environment, is often indicative of diesel sources. At the reference site, in addition to UFP, BC, two aerosol size spectrometers for the nanometer and micrometer ranges were deployed [22].

The main measurements from the circuits provided data for 7 distinct locations from one set of instruments. This affordable approach, however, results in each location being only characterized for short periods of time and non-simultaneously. Nevertheless, the patterns between locations were consistently similar from one circuit/loop to the next. The reference site provided a continuous picture of the temporal evolution of pollutant levels at one location. Reliable data management and fusion is critical for the effective analysis of these data.

Devices used in these types of measurements are not integrated into one data-logging system. Instead, each device has its own output file, some of which need to be post-processed using proprietary software, where, e.g., a calibration is applied. The format of these data is expectedly heterogeneous (headers, date/time stamps, measurement units); furthermore, depending on the device and the aerosol sampled, researchers may apply an additional correction factor. Other integrated data (non-real-time data such as multi-day passive nicotine samplers) that only provide a single average for their sampling period need to also be included. These data will be compared to World Health Organization's (WHO) 24-hr air quality guidelines as well as to other ETS studies.

This case study provides one example of the complexity of air pollution exposure science monitoring data. For other studies, with different study designs, such data can be collected in patently different ways and over much longer time periods. A methodology that can flexibly accommodate data is increasingly important as data generation explodes with technological developments.

B. Deployment

CEDAR components were designed to function not only for the purposes of this specific project, but be generally applicable for a wider scope of work. To demonstrate this case study we created templates for reading and annotating the different kinds of sensors involved in this study. The input template is read first by CEDAR, and regular expressions are formed based on it. Comparing each line of the data file with a regular expression isolates the variable values, which can then be treated as regular strings. At the same time, the metadata auxiliary file specifies the semantics of these variables, which can be converted and processed accordingly in CEDAR. Time and date information in the measurement section is tagged as such in the yaml file along with its format, and is combined to form the time stamp for all elements of that measurement. Measurements are all recomposed into their respective time series, marking missing values as such with a tag, and incorporating the header variables at the top level of each time series.

Simple operation to be applied without any further user action can easily been incorporated in a filter file. These filters should be one-to-one relationships, i.e. simple checks for each element of a time series, and are especially useful for selective initial tagging. Example uses include annotation of out-of-limits values and warning labels that would contribute to easier understanding of the data. These filters are only read during the input stage, and actually applied after all data has been imported into the database. They are therefore treated as user queries and executed automatically. The database component is fairly straightforward, as MongoDB document structure can be fully replicated with Python dictionaries, which can be accepted without further processing.

The remaining components are simpler to approach in a backwards fashion, starting with the graphical user interface. A complex user request is broken down by the user into its elementary parts. This not only aids readability, but is also directly translatable into object terms. Each part is given a `label` by the user, and can be referred

to by other parts of the query. For example, consider a sample query as: "*For every hour check if the average concentration of substance A exceeds value X, and the average particle size of substance A was less than Y, then return the two quantities*". This can be broken down as follows:

```
- label: substance_average
  quantity: concentration
  tags: [substance: A, sensor: S], units: #/m3
  filter: average, averaging_span: 1 hour
  time_frame: start_date to end_date
  store_in_database: true

- label: average_check
  filter: condition
  expression: substance_average > A
  store_in_database: false

- label: size_avg
  quantity: size
  tags: [substance: A, sensor: S], units: nm
  filter: average, averaging span: 1 hour
  time_frame: same_as substance_average
  store_in_database: true

-   label: size_check
  filter: condition
  expression: size_avg < Y
  store in database: false

-   output: substance_average, size_avg
  output condition: average_check AND size_check
  output tags: [alert]
  output format: json
  store in database : false
```

This query would ultimately yield two partial time series as its result, with time stamps accompanying the value averages fulfilling the given conditions. The timestamp storing technique described earlier is a good way to improve the efficiency of time frame lookups, which is a conceivably common operation for time series.

The graphical over-the-web user interface is presented to the user incrementally, depending on their choices. A label, for which a field is always present in each input panel, is given to each quantity, so that the users can refer to them in other sections of the same query. Once a label has been filled in, a dropdown list prompts the user to select the physical quantity of the subject time series. The list is retrieved from the database. A filter and required tags, as well as unit conversions, can then be selected for that physical quantity, with valid options presented again in a list. Parameters to each filter are also defined here, as well as time constraints or granularity for the time series, when applicable. Finally, the user has to confirm the output time series and condition as well as the preferred output format in the output panel.

A query can be broken down into as many sections as necessary. Each of them conveys a simple transformation, and they collectively build a request of arbitrary complexity. The graphical user interface is presented incrementally to the user depending on their previous choices, and facilitates the definition of each query section by presenting all available options at each step, as well as prohibiting entry of non-applicable filters and transformations (e.g., requesting an average of a status code time series would not be allowed). This process involves communication with the database at various stages, but provides the user with a simple, robust, and easy to use query builder. Results can also be imported back into the database, as suggested by the "store in database" field for future reference.

Each field of this query is transmitted over the net to the server with an HTTP GET request. The input-output translator produces the queries that retrieve from the database the elementary quantities, i.e. concentration and size for this particular case. Time series objects are built from each, carrying all relevant information. However, they also contain the query information to transform them as required. In this case, the object that holds the concentration time series also includes the average filter, with its parameters as set by the user, the units, and the *store in database* value. A concept for this particular example can be seen on Figure 2.

These objects undergo the transformations now specified by their internal parameters, and continue through the instructions the input translator has provided. Semantic information is never discarded. Finally, two objects are returned to the translator, which now formats them and outputs them in two channels. First, a `json` file is provided to the user as the output format suggests, and secondly the two time series are stored back into the database. If they are results of simple processing, they can be stored in the same document as their original time series. Simple processing is indicated by common origin of the tags and semantic information for all measurements in the time series. Otherwise, separate documents are formed, and all relevant tags are carried over.

This type of querying covers a very wide range of operations. Users can define custom filters as they wish, adhering to specific limitations, to provide additional functions. A filter is provided as Python code and is placed in the appropriate folder. Built-in filters are present in three relationship variations: one-to-one, one-to-many, many-to-one.

- A filter expressing a one-to-one relationship is the numerical transformation filter, which implements simple arithmetic on the given quantities, e.g. temperature + 273.
- A filter expressing a one-to-many relationship is the interpolation filter, which could be used to increase the granularity of a time-wise sparse series.
- Finally, a filter expressing a many-to-one relationship is the averaging filter, which derives a single value from many more.

Currently present filters range in their function from filters offering simple numerical operations and logic checks, to filters that calculate minimum and maximum values, averages, and rolling means.

As far as quality checks are concerned, CEDAR provides some elementary functionality. The first stage of quality controlling

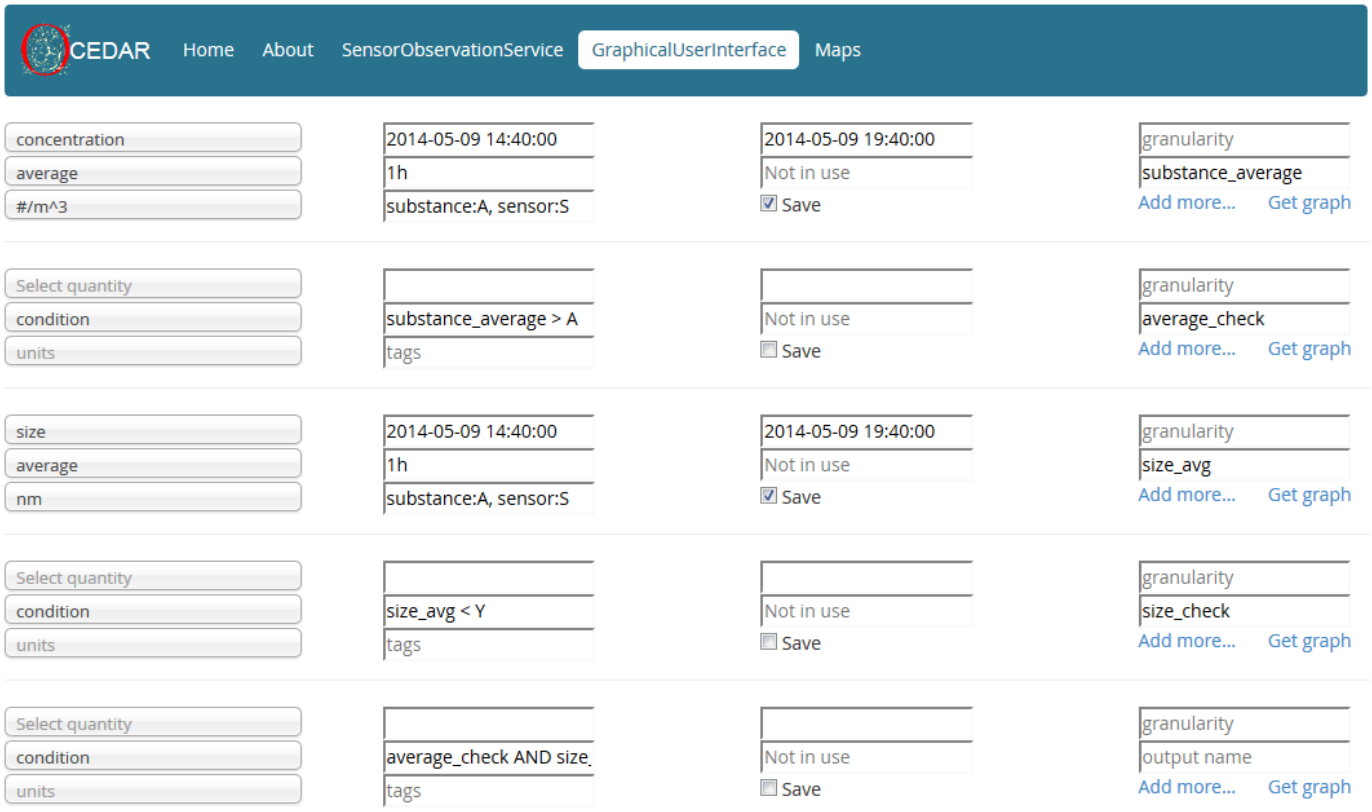

Fig. 2. Screenshot of the GUI

consists of basic comparisons against the higher and lower limits provided along with the metadata for each value. Missing values are explicitly tagged as such, but never removed from the database. This also represents CEDAR's general philosophy: in cases where new values are calculated to provide a quality-checked replacement for their old counterparts, those remain in the database, tagged as such. Information loss and one-way transforms are thus discouraged.

In this particular study, no measures were developed to account for the time drifting of sensors. In some cases the sensors are automatically or manually re-calibrated, but these processes are not reflected in the datasets provided to CEDAR. Beyond that, overly long sessions of data-logging were avoided in this instance, so any residual effects of time drifting are considered to be negligible. The system —partly owing to the range of its desirable flexibility— has no way of countering false semantics. We are operating under the assumption such issues this can be mitigated on the end users' side, by careful metadata input.

VI. Discussion and Future work

CEDAR software platform was motivated by AiRCHIVE's initiative [10], resulting to a more mature implementation and a fresh system design. CEDAR counts as a sensor data management system, which can take input from any file, utilizing a sophisticated template engine. On the other hand, AiRCHIVE serves real-time data publisher incorporated in a sensor system. Also, the document-based Mongo DB, used in CEDAR, facilitates a lot of complex operations on data, unlike AiRCHIVE's relational database which lacks of flexibility.

Although still under development, this system's potential is significant. The case study with air quality monitoring data from a smoky Swiss railway is a solid test to validate the system's functionality. We aspire to evaluate it against different case studies, involving meteorological and hydrological time series data. We also aim to develop more variant built-in filters, which users could apply on data, without developing them. Finally, the backend performance optimization is of great importance to us, so it would respond to queries and operate robustly and time- and energy-efficiently.

References

[1] R. P. Mount *et al.*, "The office of science data-management challenge," Stanford Linear Accelerator Center (SLAC), Tech. Rep., 2005.

[2] A. E. Rizzoli, M. Donatelli, I. N. Athanasiadis, F. Villa, and D. Huber, "Semantic links in integrated modelling frameworks," *Mathematics and Computers in Simulation*, vol. 78, no. 2-3, pp. 412–423, Jul 2008.

[3] F. Villa, I. N. Athanasiadis, and A. E. Rizzoli, "Modelling with knowledge: a review of emerging semantic approaches to environmental modelling," *Environmental Modelling and Software*, vol. 24, no. 5, pp. 577–587, May 2009.

[4] C.-H. Lee, D. Birch, C. Wu, D. Silva, O. Tsinalis, Y. Li, S. Yan, M. Ghanem, and Y. Guo, "Building a generic platform for big sensor data application," in *Big Data, 2013 IEEE International Conference on*. IEEE, 2013, pp. 94–102.

[5] EarthSoft, "Environmental quality information system: EQuIS," Online at http://www.earthsoft.com, 2002. [Online]. Available: https://books.google.gr/books?id=DNcbtwAACAAJ

[6] Terrabase. website. [Online]. Available: http://terrabase.com/
[7] A. Albano, L. Cardelli, and R. Orsini, "Galileo: A strongly-typed, interactive conceptual language," *ACM Transactions on Database Systems (TODS)*, vol. 10, no. 2, pp. 230–260, 1985.
[8] W. Dreyer, A. K. Dittrich, and D. Schmidt, "An object-oriented data model for a time series management system," in *Scientific and Statistical Database Management, 1994. Proceedings., Seventh International Working Conference on*. IEEE, 1994, pp. 186–195.
[9] S. J. Mason, S. B. Cleveland, P. Llovet, C. Izurieta, and G. C. Poole, "A centralized tool for managing, archiving, and serving point-in-time data in ecological research laboratories," *Environmental Modelling & Software*, vol. 51, pp. 59–69, 2014.
[10] A. Samourkasidis and I. N. Athanasiadis, "Towards a low-cost, full-service air quality data archival system," in *Proc. 7th Intl. Congress on Environmental Modelling and Software, International Environmental Modelling and Software Society (iEMSs)*, 2014.
[11] P. J. Guo and M. Seltzer, "Burrito: Wrapping your lab notebook in computational infrastructure." in *TaPP*, 2012.
[12] E. Dede, M. Govindaraju, D. Gunter, R. S. Canon, and L. Ramakrishnan, "Performance evaluation of a mongodb and hadoop platform for scientific data analysis," in *Proceedings of the 4th ACM workshop on Scientific cloud computing*. ACM, 2013, pp. 13–20.
[13] G. Ball, V. Kuznetsov, D. Evans, and S. Metson, "Data aggregation system-a system for information retrieval on demand over relational and non-relational distributed data sources," in *Journal of Physics: Conference Series*, vol. 331, no. 4. IOP Publishing, 2011, p. 042029.
[14] A. Vaikuntam and V. K. Perumal, "Evaluation of contemporary graph databases," in *Proceedings of the 7th ACM India Computing Conference*. ACM, 2014, p. 6.
[15] M. Ghanem, Y. Guo, J. Hassard, M. Osmond, and M. Richards, "Grid-based data analysis of air pollution data," in *The Fourth International Workshop on Environmental Applications of Machine Learning EAML 2004*, 2004, p. 25.
[16] T. Rudd, M. Orr, I. Bicking, and C. Esterbrook, "Cheetah-the python-powered template engine," in *10th International Python Conference.—2002. http://www. python. org/workshops/2002-02/papers/08/index. htm*, 2007.
[17] Mako contributors, "Mako: Hyperfast and lightweight templating for the python platform," 2013. [Online]. Available: http://www.makotemplates.org/
[18] O. Ben-Kiki, C. Evans, and I. döt Net, "YAML Ain't Markup Language, version 1.2," http://www.yaml.org/spec/1.2/spec.html, 2009.
[19] K. Chodorow, *MongoDB: the definitive guide*. O'Reilly Media, 2013.
[20] S. Parikh. (2014, October) Schema design for time series data in mongodb. [Online]. Available: http://blog.mongodb.org/post/65517193370/schema-design-for-time-series-data-in-mongodb
[21] jan 2015. [Online]. Available: http://www.tagesanzeiger.ch/zuerich/stadt/Raucherhoehle-Hauptbahnhof/story/10581808
[22] M. Tsai, A. Ineichen, B. Flückiger, and M. Davey, "AT-Tagung - résultats préliminaires des mesures de qualité de l'air réalisées en gare de Bâle CFF," Arbeitsgemeinschaft Tabakprävention Schweiz, Tech. Rep., nov 2014.