Adaptive Filtering as a Service for Smart City Applications

Riccardo Petrolo, Valeria Loscri, Nathalie Mitton, Elisa Herrmann

Adaptive Filtering as a Service for Smart City Applications

Riccardo Petrolo*, Valeria Loscri†, Nathalie Mitton†, and Elisa Herrmann‡
*Rice University, USA - †Inria Lille - Nord Europe, France - ‡ATOS, Spain

***Abstract*—Smart cities are a key factor in the consumption of materials and resources. As populations grow and resources become scarcer, the efficient usage of these limited goods becomes more important. Building on and integrating with a huge amount of data, the cities of the future are becoming a realization today. There are millions of sensors in place already, monitoring various things in metropolises. In the near future, these sensors will multiply until they can monitor everything from streetlights and trashcans to road conditions and energy consumption.**

In this context, effective strategies or solutions for refining data sets can play a key role. Based on these premises, we propose intelligent and adaptive filtering mechanisms as a service (FIIAAS) integrated in the VITAL-OS middleware and will show their feasibility and their effectiveness in the smart city context.

***Index Terms*—Smart Cities, Internet of Things, Cloud of Things, Filtering.**

I. Introduction

With the growth of our population and the advent of concepts such as cloud computing and the Internet of Things (IoT), the natural step cities will take is to become more interconnected. There are millions of sensors in place already, monitoring various things in metropolises. In the near future, these sensors will multiply until they can monitor everything from trashcans to streetlights and to road conditions and energy consumption [**?**].

These smart cities will allow us lowering energy consumption, optimizing resources exploitation, and building efficient cities.

The European Commission has predicted that by 2020, there will be 50 to 100 billion devices connected to the Internet [**?**]. When large numbers of sensors are deployed and start collecting data, traditional application-based approaches become infeasible. Therefore, researchers have introduced a significant amount of middleware solutions.

In the literature, there are numerous descriptions of middleware to support wireless sensor networks in a broad spectrum of activities. IoT middleware solutions help users retrieve data from sensors and feed them into applications easily by acting as mediators between (remote) hardware and application(s). Moreover, several filtering mechanisms have been proposed in different contexts ranging from Wireless Sensor Networks to RFID. In the IoT context, there is a real need of effective filtering mechanisms.

This work is partially supported by CPER FEDER DATA and by the European Community in the framework of the VITAL FP7 project.

In this paper, we propose an adaptive filtering mechanism as a service, able to reduce and refine the initial dataset on the basis of the parameters defined by the user. The mechanism is integrated in the VITAL platform in order to show its effectiveness.

The rest of the paper is organized as follows. In Section II, we give a general overview of the VITAL architecture and its components. In Section III, we detail the filtering module and specify the two types of filtering mechanisms implemented as static filtering. Section IV describes the continuous mechanism filtering as adaptive approach. Section V overviews the broad set of filtering problems and solutions. Finally, we conclude our paper in Section VI.

II. VITAL

The main objective of VITAL is to integrate different IoT platforms and ecosystems. To this end, a key factor is the virtualization of interfaces in combination with cross-context tools that enable the access and management of heterogeneous objects supported by different platforms and managed by different administrative stakeholders.

Figure 1 shows the VITAL architecture organized in three main layers: *IoT Platforms and Data Sources*, *Platform Agnostic Management, Monitoring and Governance*, and *Smart City Applications and Tools*. Below we present the features of fundamental modules:

- **IoT Platforms and Data Sources**. At the bottom of the architecture different data-sources stand. In order to be virtualized and integrated into VITAL, those systems have to expose a well defined PPI (Platform Provider Interface).
- **Platforms Access and Data Acquisition (PADA)**. It has the objective to access the low-level capabilities of the IoT Systems (through PPI) and to transform the acquired data and meta-data into a common data model (i.e., VITAL ontology).
- **Data Management Services (DMS)**. It provides cloud-based functionality for managing data and meta-data. The offered services include data and meta-data persistence, creation of new data, and more. The DMS communicates, via REST interfaces, directly with PADA, Added Value Services and the VUAIs (Virtualized Unified Access Interfaces).
- **ICOs and Services Discovery (SD)**. VITAL provides the means for discovering ICOs in the scope of horizontal

integrated IoT applications spanning multiple platforms and business contexts. This module interacts with the DMS in order to discover the "appropriate" resources for a particular business context [**?**].

- **Added Value Functionalities**. This module involves a set of complete services and tools:
  - *Filtering*. This module is in charge for reducing the information associated with individual data streams persisted in the platform agnostic data management layer. Therefore, it reduces unwanted information, thereby optimizing processing performance and economizing on network bandwidth.
  - *Complex Event Processing (CEP)*. It enables the processing of data-streams for multiple sources in order to identify patterns and/or infer events.
  - *Orchestration*. Its goal is to combine and manage multiple services from the above-listed modules, in order to deliver new added-value services. The combination of the various services is based on a workflow of service oriented components and interactions, which may be specified on the basis of rules.
- **Smart City Applications and Tools**. VITAL supports the development, integration, deployment, and operation of Smart City applications. This goes for instance with a complete environment to assist in the easy deployment and development of Smart City application [**?**].

III. Intelligent mechanisms of Filtering

The Filtering module can be accessed through RESTful web service. The communication standard used to exchange data is JSON(-LD), the format in which this information will arrive as input to the filtering module. The received parameters will then be used to drive the filtering process itself. In order to retrieve information, the Filtering module relies on the query endpoint provided by the Data Management Service (DMS). To this end, queries are defined using MongoDB[1] Query standard. According to the standard, query parameters are exchanged using a JSON object containing specific criteria, or conditions, to be used for the selection of required results. The VITAL Filtering module supports two different functions, threshold and resampling.

A. Threshold Filtering

Threshold filtering is a function that allows users and applications to retrieve observations based on the comparison of the measured value of a property and a threshold value defined in the request. Figure 2 shows an example of the interactions between the Filtering module and a generic requester. In this example, a requester (i.e., the Orchestrator module) directly activates the Filtering with the objective to filter observation for a specific ICO. The function receives as input a JSON object containing a set of keys like ICO identifier and threshold numerical value. The request object is enriched with other options in order to configure the filtering process to operate on a wider set of parameters.

In the following, we give some more details about the available options in the filtering request and how they can be configured to change the filtering logic. The key position has been included to allow the filtering in a specific spatial region. The aforementioned key is associated to a JSON object with the triple latitude, longitude and radius. Even though position is optional, the triple is mandatory. If a user wants to restrict the filtering process over a specific area, all the elements of the triple must be provided. If one of the element is missing the request is considered as malformed and a status code 400 (Bad Request) is returned. The keys ICO and position are meant to be used alternatively. If the request object contains the ICO key, the filtering is performed only over the observations measured by the selected internet connected object. If, instead, the request is characterized by the definition of position option, the filtering process is performed over the observations measured by all the internet connected object that are registered in the geographical region specified in the request. The option inequality defines the relationship between the threshold value and the observed value.

Keys *from* and *to* define a time interval over which the filtering is performed and expect input values in the XSD format. The limitation over a time interval is optional, and the filtering behavior changes according to the received configuration. If both time boundaries are defined the filtering is limited on the observations within the selected range. If only *from* is provided, to is automatically set to the time value at the moment of the request. If no time range is defined the filtering process is performed over all the observation in the system. For visually illustrating the filtering mechanism, we can consider all available observations stored in the DMS, for a specific ICO, regarding speed measurement. Such observations can be plot as illustrated in Figure 3.

B. Resampling

In the digital signal processing theory, a signal can be sampled on a specific sample rate, measuring a physical value on regular time intervals. Once the signal have been sampled, the need to change this time interval can occur. This change can be indicated as upsampling when the number of samples is increased and downsampling when the number of sampling is decreased. Upsampling operations are usually executed using an interpolation process, whereas the downsampling operations are conducted through a decimation process. In the case where both operations are available the resultant is known as resampling. For the filtering functionalities we wanted to extend this concept also to data measured by an internet connected object. Through the use of the resampling function, observations stored in the DMS can be used as data source to create a set of observation, which are separated in time by a constant value. To provide an example, suppose that an application need to calculate the speed measured by an ICO with an interval of 15 minutes, while data are measured with a interval than can vary from 8 to 12 minutes. To this end

[1]https://www.mongodb.com

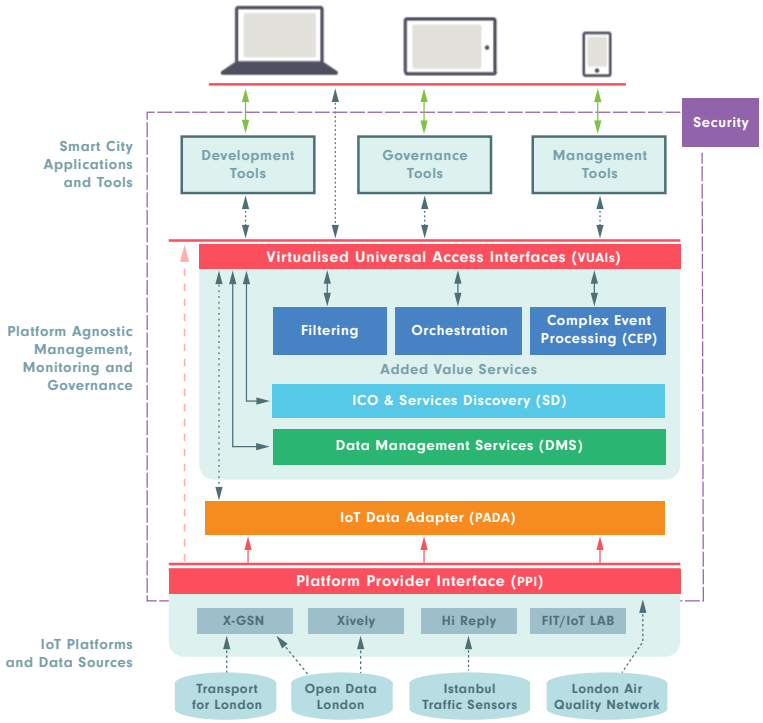


Fig. 1: The VITAL architecture.

the publish/subscribe paradigm offers a selective and flexible acquisition and filtering mechanism of sensed data on mobile devices, taking into account users preferences expressed as subscription.

IV. Adaptive Filtering for IoT Data-Driven Processing

Adaptive filtering provides the capability of creating customized filters that support complex filtering pattern by means of a Complex Event Processing. The filter takes streams of incoming events, in the form of input observations and evaluates whether those events meets the rules specified by the user and then publishes the resulting events to observers. The VITAL CEP components for the different adaptive filters are implemented based on the ATOS reference architecture for distributed, scalable and cloudified complex event processing [**?**]. An event stream can have an infinite number of events. Windows provide a means to select event subsets for complex event detection. Out of many possible ways to select events, two of the most basic mechanisms include:

- Time Windows: In some specific cases, it is not sufficient to detect a complex event when certain values are detected. There is a need to take time range into account as well. For example, detecting events when certain values exceed or fall below some threshold within a specified time period. These time periods are usually represented by fixed time windows or sliding time windows. For example, sliding time window can collect all sensor readings during the last two hours and fixed time window collects readings once every second hour;
- Tuple Windows: Instead of measuring elapsed time, tuple windows select events based on number of occurrences of particular events within an input stream. A typical example of tuple window is collection of last twenty sensor readings for further evaluation.

The typical transient event lasts for infinite small periods. However real world scenarios require support for so called long lasting events. The duration of such events is for a fixed amount of time or until another event arrives. Specifically, in the context of VITAL the idea of join events is to match events coming from different input streams and produce new complex event stream. A join between two data streams necessarily involves at least one window. To perform a join, it is usually necessary to wait for events on the corresponding stream or to perform aggregation on selected events.

The most complex event processing implementations support window to window joins, outer joins or stream to database joins. The joins are used for example to correlate information across different security devices for sophisticated intrusion detection mechanism and response.

A. Continuous Filtering

The CEP reference architecture is composed of four functional blocks as listed below:

- Coordinator
- Event Collector

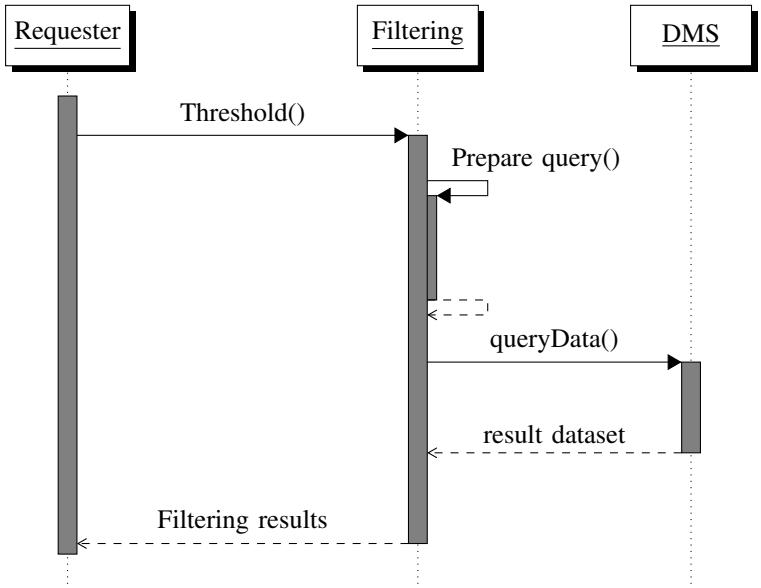


Fig. 2: Filtering threshold - Example of interactions.

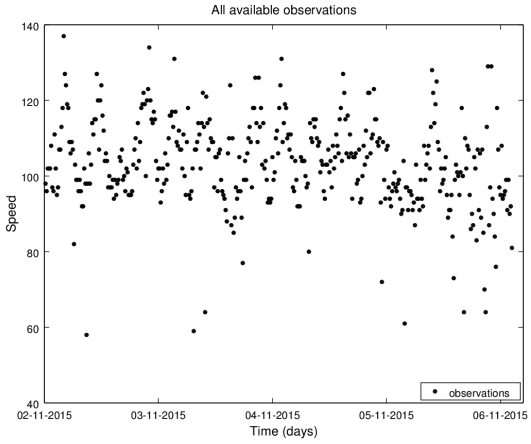
- Complex Event Detector
- Complex Event Publisher

Continuous filtering is the capability of attaching filtering to a data stream and receiving filtered results by listening to observations of different sources collected in the DMS till a delete filter request tells the filtering mechanism to stop. This filtering mechanism will create a CEP as a data stream endpoint, and will publish the results as observations continuously to the DMS until the filter is no longer needed and is stopped by the user application. The components of the Continuous Filter are depicted in 4. The Filter Administrator for the continuous filter has been redefined to create, read, update, and delete continuous filters as virtual sensors.

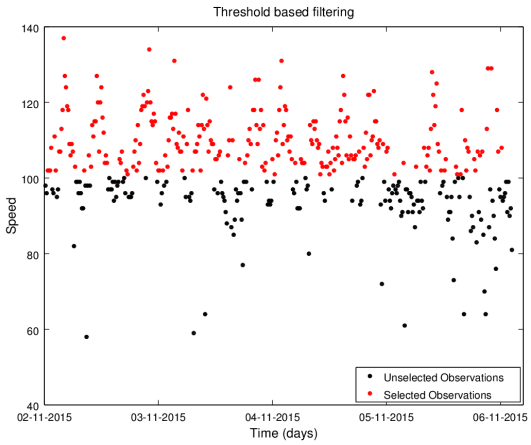
In this case, the Event Collector the Listener component is totally redefined for subscribing observations sources from the DMS allowing the collection of historical and real time observations to be filtered. The listener process has been implemented for pulling observations from the source sensors until the user sends a request to stop the filter. Since the filter is going to last longer than the creation request, the Event Collector and the Event Publisher are redefined to run in different threads from the Filtering Administrator component as they are going to be needed until the filter is stopped. Once the CEP instance is created and the components are running, the response to the user application is the identification of this new virtual sensor in order to query the filtered observations or to manage the continuous filter. Due to the modularity of the Atos CEP architecture, it is possible to implement many different kinds of adaptive filters providing different implementations of the functional blocks of the architecture. For example, filters that compare events to events in same or different stream, or compare events to some aggregated values. A typical example of event filtering is capturing sensor readings where values average fall outside of expected range. In this case, the Event Collector the Listener component is totally redefined for subscribing observations sources from the DMS allowing to gather historical and real time observations to be filtered. The listener process has been implemented for pulling observations from the source sensors until the user send a request to stop the filter. The VITAL Event Filtering functionality is exposed through set of RESTFul Services through the Filtering administrator for managing the different type of filters it provides.

V. Related Work

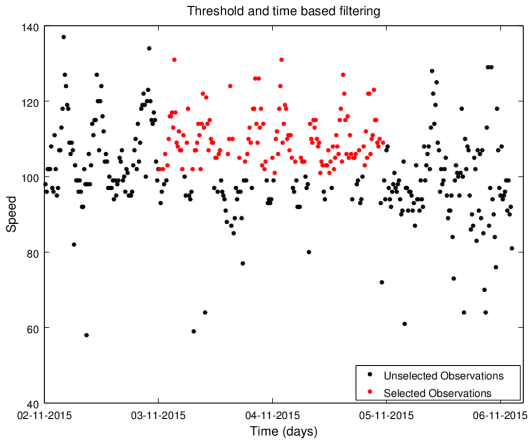
In literature there are numerous descriptions of middleware that implement filtering functionality. IoT middleware solutions help users retrieve data from sensors and feed them into applications easily by acting as mediators between (remote) hardware and application(s). One example is Middleware Linking Applications and Networks (MiLAN) [**?**], which describes a complex middleware aimed to assist the communication over WSN for specific purposes. MiLAN receives information about the applications and about the sensors and resources available in the network. It exploits such information to adapt the network configuration and, at the same time, meet the application's needs while extending the network lifetime. One of the filtering functions provided by MiLAN is mainly related to the selection of nodes actively involved. As an example, there may be scenarios where multiple sensors have overlapping coverage areas, hence producing redundant information. The ability to enable only a subset of nodes in the area can meet the application requirements and at the same time save energy and extend the network lifetime. To this purpose the system takes into account the power costs of using every node in the network, which include the power to run the device, the power to transmit its own and other nodes data and the power needed to maintain a specific role in the



(a) Speed observations of an ICO



(b) Selection Threshold



(c) Selection by threshold and time interval

Fig. 3: Filtering mechanisms.

network. The resources available in the network can be used to introduce Virtual Sensors obtained by mixing different data sources. Such functions can be exploited in the VITAL system.

Another example of proposed middleware is Garnet [**?**], where the architecture is mainly focused in data stream management. In the model depicted in Garnet, many sensors transmit their data to a fixed network infrastructure via a wireless medium. The access to this network is granted through receivers. The relationship between sensor nodes and receivers is many to many, thus arriving data undergoes a filtering process. Such a process reconstructs the data stream by eliminating duplicates generated by the reception of the same information by multiple receivers. Moreover the broker is capable to filter sensor data according to the global needs. So the acquired raw data are pre-processed, filtered and enriched with semantic information. Another important aspect is that those data should be sent to the interested receivers in near real time. To this end the publish/subscribe paradigm offers an appropriate solution. It offers a selective and flexible acquisition and filtering mechanism of sensed data on mobile devices. The dynamism and flexibility offered is because it takes user's preferences expressed as subscription into account. Furthermore, communication between publishers and subscribers is asynchronous, thus a device can be registered and disconnected at the same time. All the data that suits device's subscriptions can be sent as soon as it gets reconnected.

Liu and Martonosi in [**?**] propose a middleware architecture that enables application modularity, adaptability, and reparability in wireless sensor networks. The proposed Impala allows software updates to be received via the nodes wireless transceiver and to be applied to the running system dynamically. The filtering capabilities in this case are used in order to dispatch events to the above system units and initiate chains of processing. Within the context of RFID, different filtering mechanisms have been introduced in literature; for example the EPCglobal standard specifies filtering mechanisms as part of the ALE (Application Level Event) layer of the architecture [**?**]. As specified by the EPCglobal standard in [**?**], the role of the ALE interface within the EPCglobal Network Architecture is to provide independence among the infrastructure components that acquire the raw EPC data, the architectural component(s) that filter & count that data, and the applications that use the data. In detail, the ALE interface:

- Provides a means for clients to specify, in a high-level, declarative way, what EPC data they are interested in, without dictating an implementation.
- Provides a standardized format for reporting accumulated, filtered EPC data that is largely independent of where the EPC data originated or how it was processed.
- Abstracts the sources of EPC data into a higher-level notation of "logical reads", often synonymous with "location", hiding from clients the details of exactly what physical devices were used to gather EPC data relevant to a particular logical location.

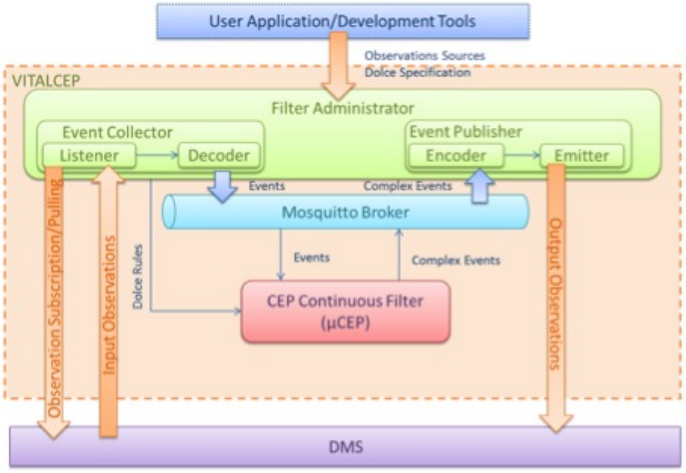


Fig. 4: CEP Continuous Filter Architecture.

Authors in [**?**] introduce a middleware implementation of ALE mechanisms. The proposed AspireRFID[2] extends the filtering function to any kind of data such as active sensor data, MAC addresses, phone numbers, etc. The Cougar project[3] aims to tasking sensor networks through declarative queries. Given a user query, a query optimizer generates an efficient query plan for in-network query processing, which can vastly reduce resource usage and thus extend the lifetime of a sensor network. In this case, the filtering capabilities of the system are used in order to filter the sensors interested by the query. The main aspect that we addressed with this literature analysis was to identify a broad set of filtering problems and solutions. Unfortunately, due to the position of our filtering functions, some of those problems cannot be properly addressed. This is because some of those functionalities are strictly connected with the sensing devices. This means that they have to be handled by entities themselves, which reside inside the individual silos, before data is sent to the VITAL platform, like the filtering of duplicated values. Some other aspects are instead too specific to applications or users thus such operations should be left to the applications, which lie in an upper layer.

VI. Conclusion

In this paper, we have proposed different filtering mechanisms as a service and we have shown their feasibility and their effectiveness when integrated in the context of VITAL-OS framework, a semantic-based framework able to manage several data sources in the context of the Internet of Thing paradigm with an horizontal vision of the different platforms. Specifically, we have shown a viable solution to shrink the data set based on the parameters defined by a user. Continuous and adaptive filtering give the possibility to compare events to events that belong to the same or to different stream, or compare events to some aggregated values. Filtering mechanisms represent a very valuable added functionality for IoT platforms.

[2]http://wiki.aspire.ow2.org

[3]https://www.cs.cornell.edu/boom/2004sp/ProjectArch/Cougar/index.html