Dynamic Monitoring Dashboards Through Composition of Web and Visualization Services

Sofie Van Hoecke(✉), Cynric Huys, Olivier Janssens, Ruben Verborgh, and Rik Van de Walle

Data Science Lab, ELIS, Ghent University-iMinds, Ghent, Belgium
sofie.vanhoecke@ugent.be
http://datasciencelab.ugent.be

Abstract. In order to present and communicate the condition of monitored environments to supervising experts, a dashboard is needed to present the status of all sensors. The heterogeneity and vast amount of sensors, as well as the difficulty of creating interesting sensor data combinations, hinder the deployment of fixed structure dashboards as they are unable to cope with the accordingly vast amount of required mappings. Therefore, in this paper, the development of a dynamic dashboard is presented, able to visualize any particular and user defined data and sensor composition. By implementing the heterogeneous sensors as semantically annotated Web APIs, a dynamic sensor composition and visualization is enabled. The resulting condition monitoring dashboard provides a clear overview of the system KPIs in acceptable timing and provides helpful tools to detect anomalies in system behaviour.

Keywords: Web APIs · Semantic annotation · Monitoring environments · Dynamic composition and visualization · Dashboards

1 Introduction

Multi-sensor architectures, consisting of heterogeneous sensors, are becoming increasingly popular, despite the many challenges they face in terms of data heterogeneity and proprietary data representation standards and communication protocols. The sensors communicate their results through different protocols and represent their data in different formats, resulting in a huge heterogeneity in terms of sensor data representation. Nevertheless, monitoring applications expect a near real-time flow of up-to-date sensor data.

By adopting the Internet of Things vision and implementing the sensors as web-connected devices, sensors have a uniform Web API [1], solving most of the challenges listed above. By using RESTdesc to semantically describe the sensors, advanced sensor compositions and mash-ups can be dynamically generated with existing Semantic Web reasoners [1], enabling the detection of complex events that previously would have remained undetected.

In order to visualize the condition of monitored environments to supervising experts, a monitoring dashboard is needed to present the status of all sensors.

B. Mandler et al. (Eds.): IoT 360° 2015, Part II, LNICST 170, pp. 465–474, 2016.
DOI: 10.1007/978-3-319-47075-7_50

Both the heterogeneity and amount of sensors, as well as the difficulty of creating interesting sensor data combinations, hinder the deployment of fixed structure dashboards as they are unable to cope with the accordingly vast amount of required mappings. Therefore, the development of a dynamic dashboard is required, able to visualize any particular and user defined data composition.

This paper introduces a system architecture in which dynamic sensor compositions and dynamic visualizations of these compositions are supported. The resulting visualization instantiation can be rendered in a widget on the dashboard. Both composition and visualization processes are feasible by semantically annotating the data and visualization services, and by providing these descriptions together with additional logic to a semantic reasoner.

The platform for dynamic visualization of multi-sensor architectures is implemented within an offshore wind farm use case where each of the wind turbines has several sensors by which the current condition of the turbine can be observed.

The remainder of this paper is as follows. Section 2 provides an overview on the current state-of-the-art of dynamic visualization techniques. Next, Sect. 3 describes the proposed platform architecture. In Sect. 4, an accompanying case study is performed, i.e. the monitoring of a (virtual) offshore wind farm. After presenting screenshots of the resulting dynamic dashboard, Sect. 5 evaluates the proposed platform in terms of usability, reliability and general performance. Finally, Sect. 6 formulates the major conclusions.

2 Related Work

Hoang et al. [3] propose a hypergraph-based Query-by-Example approach for developing a dashboard that satisfies a user query according to the current user requirements. Using this hypergraph, knowledge is mapped from heterogeneous and disparate data sources onto homogeneous ontological clusters. The hypergraph-guided data linkage supports interactive exploration, contextualization and aggregation of the data. Gitanjali et al. [2] adopt this Dashboard-by-Example framework to develop a dynamic dashboard that allows to identify semantically equivalent data in warehouses by modeling complex data as components and permitting users to link this data to detect structural dependencies.

Leida et al. [4] annotate both the data and the visualization services using a label and chart ontology using OWL DL and SWRL. The visualization process is initialized by a SPARQL query and generic types (labels) are assigned to each of its variables. A third ontology holds the instantiations of concepts that were defined in the two former ontologies. The semantic Pellet reasoner [5] is responsible for the actual creation of the visualizations.

In order to optimize and integrate global production processes, Mazumdar et al. [6] adopt Semantic Web and information extraction mechanisms to collect, structure and visualize document collections. The proposed methodology is domain independent, abstract and based on ontologies. The actual visualization process occurs in a similar way as the Dashboard-by-Example approach.

3 Dynamic Visualization of Multi-sensor Architectures

The vast amount and heterogeneity of data sources are two major obstacles for the dynamic and efficient composition of this data. In monitoring environments the data sources are sensors, having limited resources and storage. The automatic – and meaningful – composing of sensor data is essential in order to detect more complex events. It allows an early intervention and prevents permanent component damage. Furthermore, by linking sensor data with relevant ad hoc mash-up data, nontrivial associations can be exposed. From an analytical point of view, it is also crucial to have access to the evolution and history of the sensor data, enabling future failure detection and prevention. Sensors are typically not able to keep track of their previous values, so external storage is required.

Sensor and data compositions need to be dynamically visualized, hereby limiting the user input to selecting the preferred visualization method from a system-generated list of meaningful options, taking into account the preferences and characteristics of the current user profile. This results in a personalized and dynamically built monitoring dashboard that allows to correctly analyse and interpret the most critical functions of the monitored environment. Figure 1 presents the proposed system architecture, in which four major components can be distinguished. The central broker component contains the core functionality and connects (compositions of) data streams to an appropriate visualization service, enabling meaningful real-time visualization instantiations to be created and displayed on the dashboard widgets. The components are described below.

3.1 Data Services

Sensors can be implemented as Web APIs to cope with the heterogeneous (and possibly proprietary) data representation standards and communication protocols. By semantically annotating the obtained Web services and means of inference, a higher level coupling can be achieved without any additional configuration required. By considering each individual sensor as an abstract REST resource, the system will be less application specific. A resource requesting component needs to have access to the included semantic description, so (by means of *content negotiation*) sensor data can be correctly interpreted.

3.2 Visualization Services

The main responsibility of the visualization services is to visualize the submitted sensor data. Many satisfying and well performing visualization libraries exist, and it suffices to enclose one of them in a Web service, making it accessible by means of an API. This way, application specific functionality can be inserted. Data is submitted using query string parameters. The actual mapping of sensor data to these parameters is carried out in the central broker component, provided that data and visualization services are consistently and compatibly annotated. The visualization service interprets the submitted data using the query parameters, and subsequently processes it. Source code that visualizes the

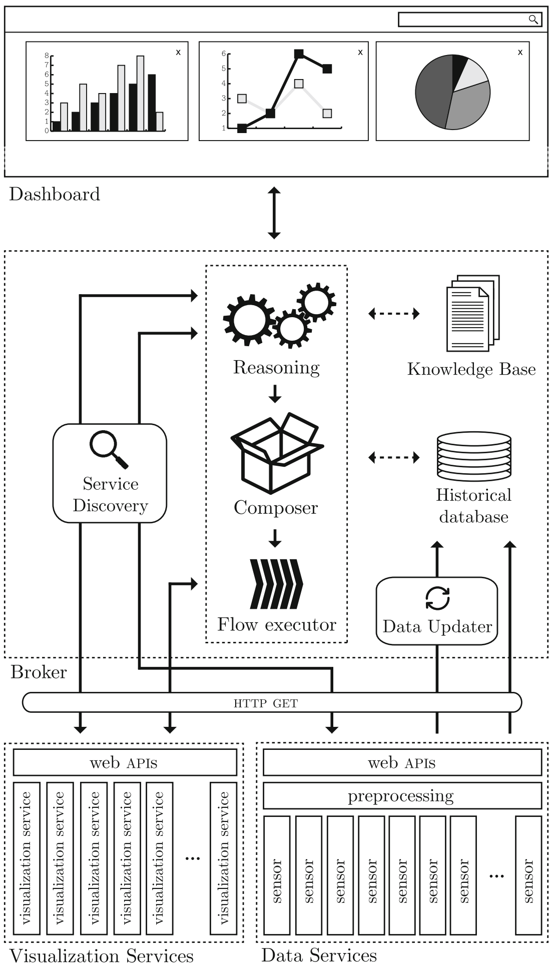

Fig. 1. Platform for dynamic visualization of multi-sensor architectures

data is generated, as well as an update function to which real-time data can be provided. This update function is called periodically, with a frequency specified by the current user. The ad hoc connection between visualization services and Web APIs allows the creation of a personalized monitoring dashboard, exposing formerly unknown relations between events and/or component behaviour.

3.3 Broker Component

Among the main tasks of the central broker component are (i) the discovery and management of data and visualization services, (ii) the execution of the three-phased reasoning process, (iii) the periodic retrieval and storage of up-to-date sensor data, and (iv) the presentation and updating of visualization instantiations on the dashboard. The broker component needs to have access to the data and visualization service URIs in order to involve the services in the reasoning process. Hard-coding these locations is not flexible. Therefore, the central component obtains the URIs autonomously using a discovery mechanism. This mechanism also actively monitors the statuses of discovered sensors by the use of pinging. Alternatively, sensor discovery can also be done passively by listening to a sensor's heartbeat. A sudden disconnection may indicate a component failure, so in this case all dashboard users are notified.

The semantic reasoner within the central broker component is responsible for, given a sensor combination, (i) suggesting appropriate visualization services, (ii) binding the sensor data with the selected visualization service (i.e. define a mapping between the sensor data and the query parameters) and (iii) proposing relevant sensors or sensor types that could be added to the composition. The execution times of each of these reasoning phases needs to be reasonable. The knowledge base (KB) contains the central logic that – along with semantic descriptions of the data and visualization services – is indispensable for the reasoning processes as it is the link between both service types.

The central component also maintains a historical database (HDB) that is responsible for the storage of historical data, as sensors generally have no or limited storage. The Semantic Web data model is closely connected to the relational database (RDB) model [7]. When the current visualization service is able to display historical data, the HDB has to be considered as a primary (pseudo) API. The *10 min mean* norm is a commonly applied measure for the frequency by which up-to-date data is fetched. When the historical data is extended with recent data that is collected with a higher frequency, the system allows the detection of particular patterns in the evolution. Moreover, by providing the evolution of the sensor values, future events can be predicted. The visualization service may assist the user by e.g. actively comparing data ranges and marking similarities or unexpected inconsistencies, hereby enabling anomaly detection.

Once the reasoner connects a given sensor (composition) with a selected visualization service, the composer fills in the data in the created HTTP GET request, along with historical data (if requested) and an identifier, by which the resulting visualization instantiation can be recognized. The flow executor executes this request, upon which the visualizer service generates the desired source code. This subcomponent also deals with potential visualization errors.

Once the connection between data and visualization services is established, it can be exploited unaltered in order to facilitate the stream of up-to-date sensor data. Accordingly, no reasoning is required for the update process.

As sensors and visualization services are already implemented as Web APIs and a platform independent – divergent user profiles usually come with divergent devices – application is desired, a web application implementation as dashboard is preferred. The dashboard in its most elementary shape has a search bar, allowing the user to browse the available entities and select the required sensors. Requested visualization instantiations are allocated to widgets on the dashboard.

4 Case Study: An Offshore Wind Farm

The monitoring of offshore wind farms is a complex task. Each of the turbines has multiple sensors by which the condition of the turbine can be observed. In order to correctly and promptly detect (partial) failures, the communicated data within this monitoring environment has to be reliable (i.e. both correct and timely). The main focus of this proof of concept case study is on composing meaningful sensor compositions, connecting them with a suited visualization service and identifying inconsistencies by exploiting the knowledge provided in the KB and acquired in the HDB.

All components are implemented using the Ruby on Rails (RoR) MVC framework. A RoR application offers a full web application stack, and follows a RESTful, resource-oriented approach. A proper web server is embedded, so apart from a working Ruby installation, no extra software is required. RoR provides a database by default and encourages the use of web standards (such as JSON) for data transfer and HTML, CSS and JavaScript for user interfacing.

4.1 Data Services

Because of the sensitivity of data from existing wind farms, a virtual offshore wind farm with 15 turbines (each turbine containing 9 sensors) is simulated. Three major sensor categories are created, each but one producing fictitious sensor values. In order to truthfully simulate the wind farm, the values are restricted and depend on the values of related sensors.

By semantically annotating the data these sensors produce, a machine is able to interpret it, hereby enabling spontaneous connections on a higher level, without requiring any additional configuration. JSON-LD semantically annotates the sensor data and offers mappings from JSON to the RDF model using the *context* concept, linking JSON object properties with ontology concepts [8].

4.2 Visualization Services

All visualization services employ the enclosed HighCharts JS library to visualize the submitted data. When required, additional parameters (restrictions) can be included to mark e.g. extreme values. The proof of concept contains eight different visualization services.

4.3 Broker Component

The reasoning phases are the most complex subtasks of the central broker component. Rather than a traditional service description, RESTdesc is used to describe the service functionality, enabling automated users and machines to use the service. RESTdesc is designed for compactness and elegance with modern Semantic Web technologies and tools, so existing vocabularies can be reused.

The KB contains all required logics (described in either RESTdesc or N3) to perform a successful sensor composition and visualization. In order to avoid duplicated source code and centralize knowledge, a hierarchy of data and visualization services is constructed. Using this KB and the annotated sensor data, the semantic EYE reasoner is able to perform the three-phased reasoning process.

As the HDB can expand very quickly, in future work, it can be advantageous to construct multiple databases, per sensor type, to reduce the search space reasonably without causing much overhead. The active monitoring of the statuses of the sensors is integrated in the periodic (*10 min mean*) data updating process – the sensors are approached by the latter process anyway. This way, the sensors are implicitly pinged when requesting their current value. Appropriate actions are taken according to the delivered HTTP status codes. The possible applications of the HDB are diverse and broader than covered. The HDB is *the* tool of choice when performing trend analysis and trend (or value) prediction.

The composer and flow executor interact with the visualization services and bundle, execute and process the composed requests. The resulting visualization instantiation is visualized on a dashboard widget.

As stated above, no reasoning is required when updating existing visualization instantiations. To update these instantiations, new data objects in which newly retrieved values are encapsulated, can be passed to their respective update function. The update frequency can be set for each widget individually. Users are often interested in some specific aspects of the monitored wind farm, requiring several minutes to build the corresponding dashboard. To solve this, the system provides *profiles* – predefined widget clusters that swiftly make specific functionality accessible – that can be instantiated in several seconds.

The connecting logic in the KB is located at the central broker component. In order to achieve a completely loosely coupled system, data and visualization services may only depend on their individual descriptions, which can also be stored in the KB. The location of the KB and whether or not it should be distributed is worth a discussion, and has to aim for maximized usability, maximized decoupling and knowledge centralization.

4.4 Resulting Dynamic Dashboard

Figure 2 presents a possible dashboard configuration for monitoring offshore wind farms. Widgets can be easily added and/or removed. Every widget holds the descriptions of the containing sensors and has a menu to adjust the visualisation and according options. Thanks to the semantic descriptions of all sensors and visualization services, related sensor data can also be easily selected from these widgets and added to the visualization.

5 Evaluation

The limited adaptability and static nature of traditional dashboards, as well as the automated detection of complex events and creation of advanced compositions, are the driving forces behind the design of the proposed platform. There

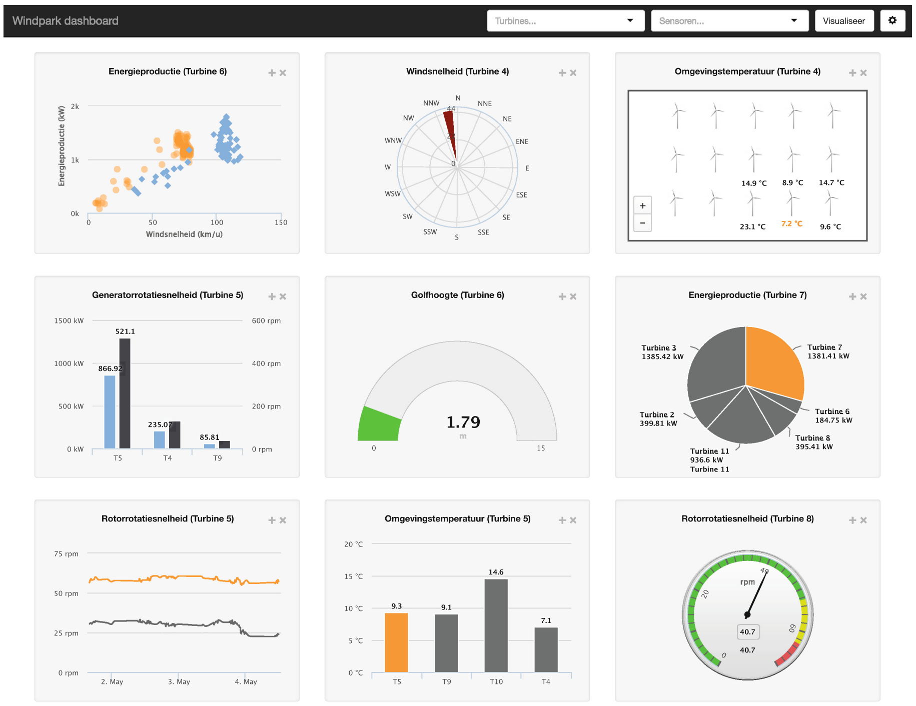

Fig. 2. Dynamic dashboard for monitoring offshore wind farms

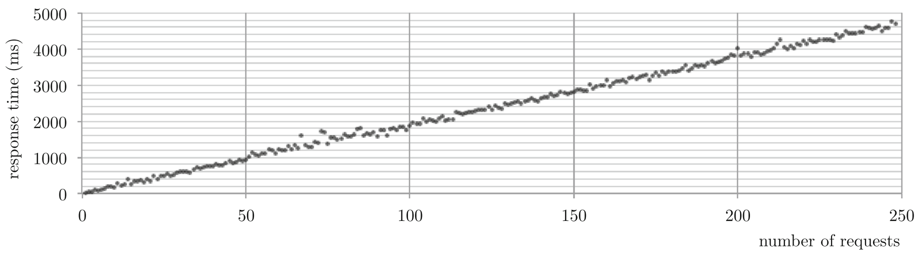

Fig. 3. Sensor data retrieval times

is a need for dashboards that satisfy current user needs, in a user-friendly way and with minimal configuration required.

New widgets are added in a straightforward way and the profile concept offers quick and easy access to vital system aspects. The interactive HighCharts library offers neat, accurate, well performing and user-friendly data charts.

Data needs to be fetched and visualized in *soft* real-time, so users can get no false sense of security. The reliable HTTP protocol transfers sensor data to the central component. Note that individual sensor requests are independent, and therefore are able to be executed in parallel. The prototype however executes

Table 1. Execution times of the three-phased reasoning process (ms)

Reasoning	Number of visualization services involved	Number of sensors involved									
		1	5	10	20	30	40	50	75	100	125
Phase 1	1	1	2	5	8	14	20	26	49	70	106
	2	2	3	6	14	17	22	32	54	81	114
	3	3	5	8	14	22	33	40	74	105	147
	4	3	4	8	17	24	34	43	69	105	150
	5	3	6	11	18	30	36	46	81	116	161
	6	3	6	11	19	27	41	54	88	132	186
	7	3	6	12	19	29	43	53	91	128	166
	8	3	6	9	17	28	41	60	86	126	170
Phase 2	1	26	962	3.719	13.263	27.269					
Phase 3		1	2	2	6	10	14	12	18	24	30

the update requests sequentially as the overhead would be too large considering the initial intention of the prototype (i.e. proving the concept). In this sequential case, retrieval times grow linearly with the number of sensors, but remain acceptable even for a larger number of sensors (see Fig. 3). As long as the total retrieval time does not exceed the user defined update interval, the user will not experience noticeable delays.

The three-phased reasoning process is the most time consuming. There is an acceptable initialization and networking overhead (in which the required semantic descriptions are collected), but the actual reasoning process, of which the execution times are presented in Table 1, is the most time-intensive. Although the reasoning procedures in the first and third phase generally have acceptable execution times, the second phase (in which the sensor data is mapped onto query parameters) is much more time consuming. Because of the (combinatorial) increasing number of possible mappings in this phase, the EYE reasoner may have a hard time connecting a visualizer service with multiple sensors.

Execution times could be reduced by providing the reasoner with additional directives and less generic linking logic. However, this undermines the dynamic nature of the system, which of course is its absolute strength. The creation times of *average* widget configurations are nevertheless acceptable, and creating new widgets only happens occasionally.

6 Conclusion

Dashboards present and communicate the condition of monitored environments to supervising experts. Contrary to current static solutions for monitoring dashboards, the proposed platform enables dynamic data visualization.

By adopting the Internet of Things vision and implementing sensors as Web connected devices, semantically annotated using RESTdesc, the presented platform allows to precisely visualize the data produced by sensors in multi-sensor environments by dynamically generating meaningful service compositions. Such a dynamic dashboard application, combined with advanced failure detection mechanisms – the HDB and reasoning component are excellent tools for this – proves to be a very powerful monitoring tool for complex, hardly accessible and/or critical environments. It enables its users to correctly monitor the condition of the environment and moreover, as a result of the meaningful sensor composition process, to detect complex events that used to remain undetected.

The platform only consists of existing technologies and Semantic Web concepts. Both the composition and visualization processes are fully dynamic, application independent and complete within acceptable time. Up-to-date data values – crucial for a monitoring application – are delivered in soft real-time.

Acknowledgement. This work was partly funded by the IWT VIS O&M Excellence project, and performed in the framework of Offshore Wind Infrastructure Application Lab.

References

1. Van Hoecke, S., Verborgh, R., Van Deursen, D., Van de Walle, R.: SAMuS: service-oriented architecture for multisensor surveillance in smart homes. Sci. World J. **2014**, 9 p. (2014). Article ID 150696, doi:10.1155/2014/150696
2. Gitanjali, J., Kuriakose, M., Kuruba, R.: Ontology and hyper graph based dashboards in data warehousing systems. Asian J. Inf. Technol. **13**(8), 412–415 (2014)
3. Duong, T.A.H., Thanh, B.N., Tjoa, A.M.: Dashboard by-example: a hypergraph-based approach to on-demand data warehousing systems. In: Proceedings of International Conference on Systems, Man, and Cybernetics (2012)
4. Leida, M., Du, X., Taylor, P., Majeed, B.: Toward automatic generation of SPARQL result set visualizations: a use case in service monitoring. In: Proceedings of the International Conference on e-Business (ICE-B), July 2011
5. Parsia, B., Sirin, E.: Pellet: an OWL DL reasoner. In: Third International Semantic Web Conference-Poster, vol. 18 (2004)
6. Mazumdar, S., Varga, A., Lanfranchi, V., Petrelli, D., Ciravegna, F.: A knowledge dashboard for manufacturing industries. In: García-Castro, R., Fensel, D., Antoniou, G. (eds.) ESWC 2011. LNCS, vol. 7117, pp. 112–124. Springer, Heidelberg (2012)
7. Berners-Lee, T.: Web Design Issues; What the Semantic Web can Represent. http://www.w3.org/DesignIssues/RDFnot.html
8. Lanthaler, M., Gütl, C.: On using JSON-LD to create evolvable restful services. In: Proceedings of the Third International Workshop on RESTful Design, pp. 25–32 (2012)