DOI 10.12694/scpe.v14i4.927
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 14, Number 4, pp. 188–200. http://www.scpe.org

AN OSGI MIDDLEWARE TO ENABLE THE SENSOR AS A SERVICE PARADIGM

GIUSEPPE DI MODICA, FRANCESCO PANTANO, ORAZIO TOMARCHIO *

Abstract. The Internet of Things vision has recently stimulated many research efforts in different communities. The collection of diverse information produced by the proliferation of inter-connected sensing entities is one of the biggest challenge that should be adequately addressed. The huge amounts of raw data produced by IoT devices need to be structured, stored, analysed, correlated and mined in a reliable and scalable way. The size of the produced data, and the high rate at which data are being produced, suggest that we need new solutions that combine tools for data management and services capable of promptly structuring, aggregating and mining data even just at the time they are produced. In this paper we propose a middleware, to be deployed on top of physical sensors and sensor networks, capable of abstracting sensor devices from their proprietary interfaces, and offering them to third party applications in an as-a-Service fashion for prompt and universal use. The middleware also offers tool to elaborate real-time measurements produced by sensors. A prototype of the middleware has been implemented. In the paper a real use case scenario is also discussed.

1. Introduction. The current role of the Internet of Thing (IoT) is no more limited only to electronic identification of objects but it is perceived as a way to act, measure, or provide services based on real-world entities [20]. Advancements in networking technologies and sensor/actuator capabilities provide a large number of physical world objects with communication and computation capabilities to interact with their surrounding environment.

However, the emerging IoT platforms are currently hard to deploy and operate, requiring except in the most trivial cases the intervention of domain experts to interpret the sensor data and, eventually, to come up with actuation commands. This approach is clearly too expensive and time-consuming, and simply does not scale with the increasing number of IoT devices. In addition, the development of IoT application is highly scenario and technology dependent, due to the heterogeneity of devices. That is why powerful middleware solutions are required to integrate heterogeneous devices and dynamic services for building complex systems for the IoT.

We believe that the service-oriented approach [14, 24] provides adequate abstractions for application developers, and that it is a good approach to integrate heterogeneous sensors and different sensor network technologies with Cloud platforms through the Internet, by paving the way for new IoT applications.

In this paper, extending the work presented in [8] we propose an OSGi-based middleware, called *Sensor Node Plug-in System (SNPS)*, where sensors are no longer low-level devices producing raw measurement data, but can be seen as "services" to be used and composed over the Internet in a simple and standardized way in order to build even complex and sophisticated applications.

The remainder of the paper is structured in the following way. Section 2 presents a review of the literature. Sect. 3 introduces the architecture of the proposed solution. In Sect. 4 we discuss and motivate the choice of the data model implemented in the middleware. Section 5 provides some details on the sensor composition process. In Sect. 6 a use case scenario is discussed. We conclude our work in Sect. 7.

2. Related work. The most notably effort in providing standard definition of Web service interfaces and data encodings to make sensors discoverable and accessible on the Web is the work done by the Open Geospatial Consortium (OGC) within the Sensor Web Enablement (SWE) initiative [4, 23]. The role of the SWE group is to develop common standards to determine sensors capabilities, to discover sensor systems, and to access sensors' observations. The principal services offered by SWE include:

- Sensor Model Language (SensorML): provides a high level description of sensors and observation processes using an XML schema methodology
- Sensor Observation Service (SOS): used to retrieve sensors data.
- Sensor Planning Service (SPS): used to determine if an observation request can be achieved, determine the status of an existing request, cancel a previous request, and obtain information about other OGC web services
- Web Processing Service (WPS): used to perform a calculation on sensor data.

*Department of Electrical, Electronic and Computer Engineering, University of Catania, Catania, Italy (firstname.lastname@dieei.unict.it).

A common misconception of the adoption of SWE standards is that, rather than encapsulating sensor information at the application level, they were originally designed to operate directly at the hardware level. Of course, supporting interoperable access at the hardware level has some advantages as it copes well with the "plug and play" concept. Currently, some sensor systems such as weather stations and observation cameras already offer access to data resources through integrated web servers. However, besides contradicting the view of OGC's SWE of uncoupling sensor information from sensor systems, the downside of this approach becomes clearer when dealing with a high number of specialized and heterogeneous sensor systems, and in particular in resource-limited scenario where communication and data transportation operations must be highly optimized. Even a relatively powerful sensor gateway has not the basic requirement to act as a web server in many cases, it is networked via a low-bandwidth network and is powered by a battery, so it has neither the energy nor bandwidth resources required to run a web service interface.

The need for an intermediate software layer (middleware) derives from the gap between the high-level requirements of pervasive computing applications and the complexity of the operations involved in the underlying Wireless Sensor Networks (WSNs). A WSN is characterized by constrained resources, a dynamic network topology, and low level embedded OS APIs, while the application requirements include high flexibility, re-usability, and reliability to cite a few. In general, WSN middlewares help the programmer develop applications by providing appropriate system abstractions, reusable code services and data services, flexible network infrastructure management and efficient resource management services.

Some research efforts have surveyed the different approaches of middleware and programming paradigms for WSN: in [11] middleware challenges and approaches for WSN were analysed, while [26] and [22] analysed programming models for sensor networks. In the following we focus on two programming models which in the past years have been taken in great consideration by researchers: the *message-oriented* and the *service-oriented* approach.

Message oriented middlewares. Message oriented middlewares (MOMs) provide distributed communication among participants on the basis of the *asynchronous* interaction model. Basically, in a MOM a participant (sender) who need to communicate to another participant (receiver) produces a message which is managed by a software queue which, in its turn, takes care of routing that message to the final destination. Two types of interaction models can be implemented in a MOM: point-to-point, by which a single message produced by a sender is delivered only once to only one receiver, and publish/subscribe, which allows a single producer to send a message to or potentially hundreds of thousands of receivers.

Many sensor frameworks calls on MOM as asynchronous communication can guarantee very high energy saving in event-driven WSN applications. Most of them opted for the publish/subscribe mechanism to implement message exchange between sensors and nodes. Mate [17] uses a Virtual Machine approach as level of abstraction. It deals with several issues such as limited bandwidth and energy consumption for large networks. Claimed features are fault tolerance, dynamism, flexibility and reconfigurability components. The weak point of this middleware is, however, energy expenditure, mainly due to generated communication overhead; therefore is not suitable for complex applications. Magnet [3] is a Virtual Machine based middleware as well. It works as an abstraction layer built on top of MagneOS, an adaptive operative system sensor oriented that allows to detect and report any moving object. The weak point is that the entire network is based on a single Java Virtual Machine running static components, so the performance is acceptable only under very stringent constraints. Impala [19] is an event-driven middleware which manages requests and responses in a completely asynchronous mode. It adapt itself to many application scenarios and can automatically set parameters according to the presented scenarios. It introduces a small overhead in transmission phase. Milan [12] allows to integrate applications on different networks. It lets applications specify QoS requirements and adjust network characteristics to improve the life cycle management. These goals are achieved by collection data from sensors aggregated on the network, with particular care for energy consumption and bandwidth usage. Cougar [7] is a database oriented middleware that stores sensor and data in a relational fashion. The management operations of WSNs are implemented in the form of SQL queries. Sina [25] presents an architecture which adapts to the physical environment. In fact, the network is modelled as a distributed system of objects, queried according the SQL language. Through the hierarchical clustering mechanism, Sina caters for scalability, as the routing of requests takes place in a hierarchical manner as a function of the root node. DsWare [18] is a middleware that uses a database approach

and is also network event-driven. Its architecture is based on modules that deal with the management of data, group management, detection of events. The publish/subscribe mechanism is used for asynchronous, topic-based communication. As Cougar, DsWare uses SQL for the registration and cancellation of events.

Service oriented middlewares. In a service oriented approach participants of a distributed system communicate through the *synchronous* interaction model. Service, rather than message, is the focus of service oriented middlewares (SOMs). This approach greatly facilitates the design task (service-centric): adding advanced features simply means implementing new services. If compared to the MOM, SOMs propose a more straightforward approach to messaging, but suffer from inflexibility and tight coupling (potential geometric growth of interfaces) between the communicating participants.

Recently, the service-oriented approach has been applied to sensor environments [14, 21]. The common idea of these approaches is that, in a sensor application, there are several common functionalities that are generally irrelevant to the main application. For example, most services will have to support service registries and discovery mechanisms and they will also need to provide some level of abstraction to hide the underlying environments and implementation details. Furthermore, all applications need to support some levels of reliability, performance, security, and QoS. All of these can be supported and made available through a common middleware platform instead of having to incorporate them into each and every service and application developed.

The SStreaMWare [10] middleware exploits a query-based system in order to access the data collected. Its service oriented nature solves the problem of software heterogeneity, facilitating the integration and guaranteeing compatibility on the one end, and allowing a weak coupling between the user application and the sensors on the other end. Useme [5] provides a high-level programming language to develop applications for wireless sensor network. The service oriented approach ensures scalability, interoperability and efficiency. Oasis [16] is an ambient-aware, service oriented programming model. The object-oriented paradigm provides an abstraction that focuses on the monitored physical phenomenon, bypassing the complexity of the network topology. MiSense [15] is a service-oriented middleware that provides a publish-subscribe mechanism based on well-defined contents. It is able to reduce the complexity through a structure above the component layers by imposing restrictions on the modularity and offering a well defined and specific interface to the rest of the system. UbiSOAP [6] is designed on the SOAP protocol. Its aim is to provide services on a pervasive sensors network. It's composed two layers, a multi-radio network layer and a communication-oriented Web Services layer. A SOAP Transport service is used by the client and the service: the Transport service interacts with the multi-radio network, sending and receiving messages over the network. TinySOA [2] is a service-oriented architecture that provides developers with simple API to implement applications on sensor networks, using the same programming language in which they were originally written. The main advantage of TinySOA is to abstract away the developer from details of the WSN hardware and the communication layer.

The service oriented approach is the one we adopt in the work being presented. In particular, we leverage on OSGi [1] as the key paradigm enabling the *service* technology in the context of sensor networks. Furthermore, the choice of the service oriented paradigm allows for a more smooth integration with Cloud platforms and for advanced discovery mechanisms also employing semantic technologies [9]. The work we propose aims at reaching a worthy compromise between WSN heterogeneity, system scalability and interoperability.

The middleware we propose on the one hand embeds service oriented features in that functionalities are provided end exposed through services; on the other one the message oriented paradigm is used to manage events produced by sensors. This way advantages of both approaches are then caught. Further, the sensor aggregation feature offered by the middleware contributes to enhances the flexibility of the sensor programming model. By means of this novel service, third party applications are exposed a new way to design and implement data manipulation schemes which relieves them from the burden of gathering, putting together and elaborate on all data coming from their sensing campaigns.

3. The SNPS middleware. This section presents the proposal for a middleware devised to lay on the physical layer of wireless sensors, abstract away the sensors' specific features, and turn sensors into smart and composable services accessible through the Internet in an easy and standardized way. The middleware was designed to follow the basic principles of the IoT paradigm [20]. Sensors are not just sources of raw data, but are seen like smart objects capable of providing services like filtering, combining, manipulating and delivering information that can be readily consumed by any other entity over the Internet according to well-known and

standardized techniques.

Primary goal of the middleware, which we called *Sensor Node Plug-in System (SNPS)*, is to bring any physical sensor/actuator on an abstraction level that allows for easier and standardized management tasks (switch on/off, sampling), in a way that is independent of the proprietary sensor's specification. By the time a sensor is "plugged" into the middleware, it will constitute a resource/service capable of interacting with other resources (be them other sensors plugged into the middleware or third party services) in order to compose high-value services to be accessed in SOA-fashion. The middleware also offers a set of complimentary services and tools to support the management of the entire life cycle of sensors and to sustain the overall QoS provided by them.

Basically, the SNPS can be said to belong to the category of the service-oriented middlewares [24]. In fact, the provided functionality are exposed through a service-oriented interface which grants for universal access and high interoperability. Yet, all data and information gathered by sensors are stored in a database that is made publicly accessible and can be queried by third party applications. Further, the SNPS also support asynchronous communication by implementing the exchange of messages among entities (sensors, components, triggers, external services). All these features makes the middleware flexible to any application's need in any execution environment.

At design time it was decided not to implement the entire middleware from scratch. A scouting was carried out in order to identify the software framework that best supported, in a native way, all the characteristics of flexibility and modularity required by the project. Eventually, the OSGi framework [1] was chosen. The OSGi framework natively supports the component's life cycle management, the distribution of components over remote locations, the seamless management of components' inter-dependencies, and an asynchronous (event-based) communication paradigm.

The SNPS middleware is then organized into several components, each of which is implemented as a software module (or "bundle") within the OSGi framework. Figure 3.1 shows the architecture of the middleware and its main components.

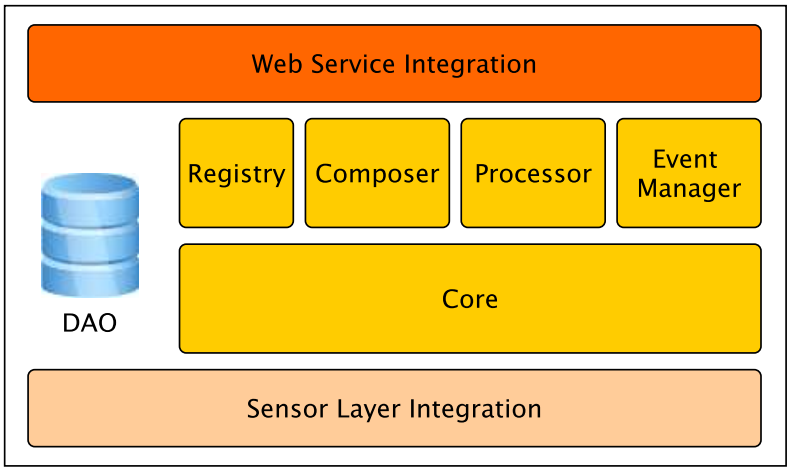

FIG. 3.1. *SNPS architecture*

The overall architecture can be broken down into three macro-blocks:

- Sensor Layer Integration
- Core and related Components
- Web Service Integration

In the following we provide a description of each macro-block.

3.1. Core and related Components. The components we are about to discuss are charged the responsibility of providing most of the middleware's functionality. In Figure 3.2 the connections among the components

are shown.

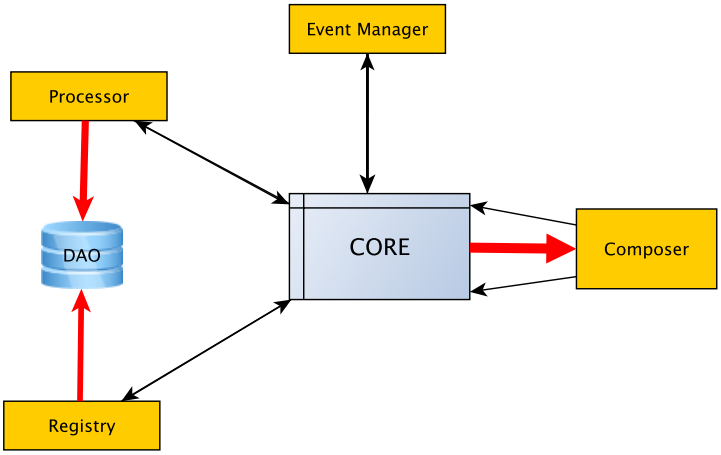

FIG. 3.2. *Core and related Components*

Core. It is where the business logic of the Middleware resides. The Core acts as an orchestrator that coordinates the middleware's activities. Data and commands flowing forth and back from the web service layer to the sensor layer are dispatched by the Core to the appropriate component. This component is responsible for the virtualization of physical sensors, i.e., conforming the features and capabilities of physical sensors under a common representation. In addition, the core is able to define the working policies of the middleware, identifying points of failure and performing action in order to recover components.

Registry. It is the component where all information about sensors, middleware's components and provided services are stored and indexed for search purpose. As for the sensors, data regarding the geographic position and the topology of the managed wireless sensor networks are stored in the Registry. Also, each working component needs to signal its presence and functionality to the Registry, which will have to make this information public and available so that it can be discovered by any other component/service in the middleware.

Composer. It represents the component which implements the sensors' composition service. Virtualized sensors have a uniform representation which allows for "aggregating" multiple sensors into just one sensor that will eventually be exposed to applications. An insight and practical examples about the aggregation service is provided in Section 5.

Processor. It is the component responsible for the manipulation of the data coming from the sensors. It enforces the sensors' aggregation schemes defined through the Composer. In particular, when data come from multiple sensors composing an aggregate, this component applies the directives of the related aggregation mechanism.

Event Manager. It is one of the most important components of the middleware. It provides a publish/-subscribe mechanism which can be exploited by every middleware's component to implement asynchronous communication. Components can either be producers (publishers) or consumers (subscribers) of every kind of information that is managed by the middleware. This way, data flows, alerts, commands are wrapped into "events" that are organized into topics and are dispatched to any entity which has expressed interest in them.

DAO. It represents the persistence layer of the middleware. It exposes APIs that allow service requests to be easily mapped onto storage or search calls to the database.

3.2. Sensor Layer Integration. The Sensor Layer Integration (SLI) represents the gateway connecting the middleware to the physical sensors. It implements a *bidirectional communication channel* supporting commands to flow both from the middleware to the sensors and from the sensors to the middleware as well and a *data channel* (for data that are sampled by sensors and need to go up to the middleware).

The addressed scenario is that of wireless sensor networks implemented through so called Base Stations (BS) to which multiple sensors are "attached". A BS implements the logic for locally managing its attached sensors. Sensors can be wiredly or wirelessly attached to a BS, forming a network which is managed according to specific communication protocols, which are out of our scope. The SLI will then interact just with the BS, which will only expose its attached sensors hiding away the issues related to the networking.

The integration is realized by means of two symmetrical bundles, which are named respectively *Middleware Gateway bundle (iMdmBundle)* and *WSN Gateway Bundle (iWsnBundle)*. The former lives in the middleware's runtime context, and was thought to behave as a gateway for both commands and data coming from the BSs and directed to the middleware; the latter lives (runs) in the BS's runtime context, and forwards commands generated by the middleware to the BSs. Since the middleware and the BSs may be attached to different physical networks, the communication between the two bundles is implemented through R-OSGi, which is a specific OSGi's bundle offering a remote communication service which other bundles living in different runtime contexts can use to communicate to each other. In Figure 3.3 the two bundles and their respective runtime contexts are shown.

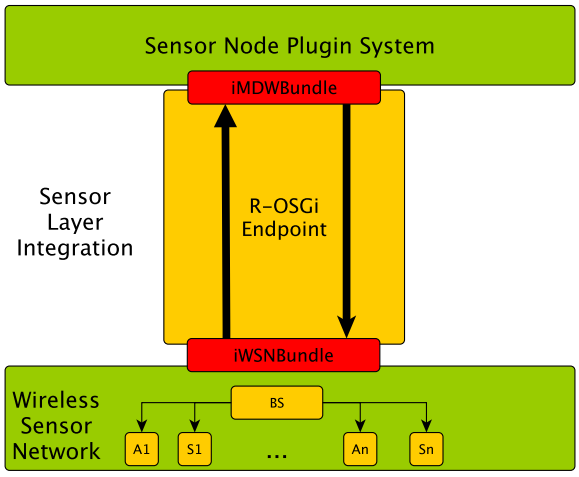

FIG. 3.3. *OSGi bundles implementing the Sensor Layer Integration*

The SLI was designed to work with any kind of BS, independently of the peculiarity of the sensors it manages, with the aim of abstracting and making uniform the access to sensors' functionality. Making uniform the management of the sensors' life cycle does not mean giving up the specific capabilities of sensors. Physical sensors will maintain the way they work and their peculiar features (in terms, for instance, of maximum sampling rate, sampling precision, etc.). But, in order for sensors (read base stations) to be pluggable into the middleware and be compliant to its management logic, a minimal set of requirements must be satisfied: the *iWsnBundle* to be deployed on the specific BS will have to interface to the local BS' logic and implement the functionality imposed by the SNPS middleware (switch on/off sensors, sample data, run sampling plan) by invoking the proprietary base station's API.

3.3. Web Service Integration. As it is shown in Fig. 3.4, the *OSGi bundle Wrapper* exports the functionality of the SNPS middleware to a Web Service context.

A modular system bundle-based enables applications that run on different platforms to communicate with each other.

This result has been achieved by exploiting the OSGi communication model; different can communicate not only importing and exporting services, but using SOA strategies Web services based.

In this case, Web services have been built using Apache CXF, which is an open source services framework that helps you build and develop services using front-end programming APIs, like JAX-WS and JAX-RS.

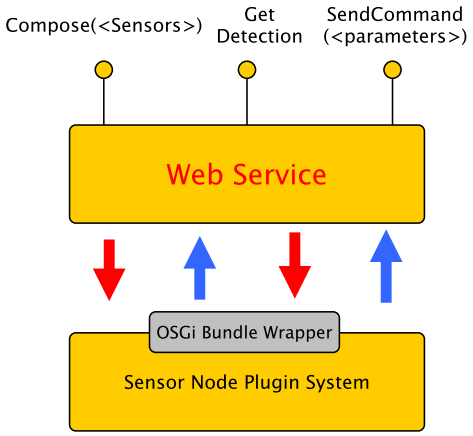

FIG. 3.4. *Wrapping and exposing SPNS as a Web Service*

These services can speak a variety of protocols such as SOAP, XML/HTTP, RESTful HTTP or CORBA, and in this case we select the SOAP protocol, exporting to clients a range of services to access middleware resources. By doing so, you can execute query for a specific sensor and its associated data, according to certain search criteria. In this way clients are able to communicate directly with sensors virtual images by sending commands, including:

- sensor activation
- sensor deactivation
- sending sampling plans
- request for sensor detection
- sensor composition

In particular, the deactivation command does not perform a proper device shutdown, but allows to change it in a power-saving state, in order to save resources that can be used at a later date.

4. SNPS data model. The SNPS data model is one of the most interesting features of the middleware. Goals like integration, scalability, interoperability are the keys that drove the definition of the model at design time. The objective was then to devise a data model to structure both sensors' features (or capabilities) and data produced by sensors. The model had to be rich enough to satisfy the multiple needs of the middleware's business logic, but at the same time had to be light and flexible to serve the objectives of performance and scalability. We surveyed the literature in order to look for any proposal that might fit the middleware's requirement. Specification like SensorML and O&M [4] seam to be broadly accepted and widely employed in many international projects and initiatives. SensorML is an XML-based language which can be used to describe, in a relatively simple manner, sensors capabilities in terms of phenomena they are able to offer and other features of the specific observation they are able to implement. O&M is a specification for describing data produced by sensors, and is XML-based as well. XML-based languages are known to be hard to treat, and in many cases the burden for the management of XML-based data overcomes the advantage of using rigorous and well-structured languages. We therefore opted for a solution that calls on a reduced set of terms of the SensorML specification to describe the sensor capabilities, and makes use of a much lighter JSON [13] format to structure the data produced by sensors. An excerpt of what a description of sensor capabilities look like is shown in Fig. 4.1.

This is the basic information that must be attached to any sensor before it is plugged into the middleware. Among others, it carries data regarding the phenomena being observed, the sampling capabilities, and the absolute geographic position. When the sensor wakes up, it sends this information to the middleware, which will register the sensor to the Registry bundle, and produce its *virtualized image*, i.e., a software alter-ego of the physical sensor which lives inside the middleware run-time. The virtual sensor has a direct connection with the physical sensor. Each interaction involving the virtual sensor will produce effects on the physical sensor too. It

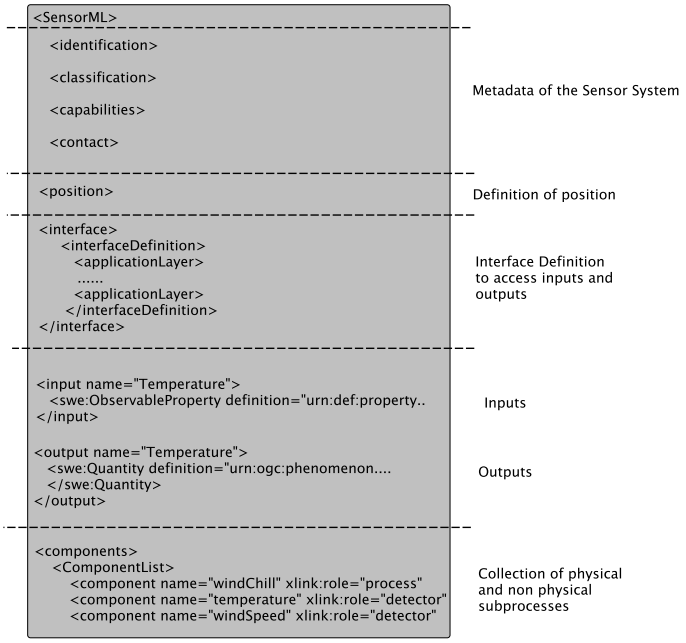

FIG. 4.1. *Description of sensor capabilities in SensorML*

is important to point out that all virtual sensors are treated uniformly by the middleware's business logic.

Furthermore, SensorML is by its nature a process-oriented language. Starting from the atomic process, it is possible to build the so-called *process chain*. We exploited this feature to implement one of the main service provided by the SNPS, i.e., the sensors' composition service (see Sect. 5 for more details). This service, in fact, makes use of this feature to elaborate on measurements gathered by multiple sensors.

As regards the definition of the structure for sensor data, JSON was chosen because it ensures easier and lighter management tasks. The middleware is designed to handle (sample, transfer, store, retrieve) huge amounts of data, with the ambitious goal to also satisfy the requirements of real-time applications. XML-based structures are known to cause overhead in communication, storage and processing tasks, and therefore they do not absolutely fit our purpose. Another strong point of JSON is the ease of writing and analyzing data, which greatly facilitates the developer's task. A data sampled by a sensor will then be put in the following form:

```
Sensor_Measure:
{
    ''SensorId'':''value'',
    ''data'':''value'',
    ''type'':''value'',
    ''timestamp'':''value''
}
```

5. Building and composing Virtual Sensors. Sensor Composition is the most important feature of the SNPS middleware. Simply said, it allows to get complex measurements starting from the samples of individual sensors. This composition service is provided by the Composer bundle (cf. Fig 3.1).

An important prerequisite of the composition is the sensor "virtualization", which is a procedure performed by the Core component when a sensor is plugged into the SNPS middleware (see Sect. 3.2). Aggregates of sensors can be built starting from their software images (virtual sensors) that live inside the SNPS middleware. Therefore, in order to create a new composition (or aggregate) of sensors, the individual virtual sensors to be combined need to be first identified. Secondly, the operation that is to be applied to sensor's measurements

must be specified. This is done by defining the so-called *Operator*, which is a function that defines the expected input and output formats of the operation being performed. The final composition is obtained by just applying the Operator to the earlier chosen virtual sensors.

The operator for aggregating sensors can be defined using MathML [27]. In the operator schema, each sensor input is bound to an item of the formula to be executed. A check is then performed in order to verify the compatibility of the unit of measures of the data that are being aggregated by the operator. This feature is very useful in the case of heterogeneous sensors' composition, while in the case of homogeneous sensors pre-defined patterns are offered.

Once the operator has been created, a new virtual sensor (the aggregate) is available in the system and exposed for use by third party applications. Figure 5.1 shows the structure of an aggregate of sensors.

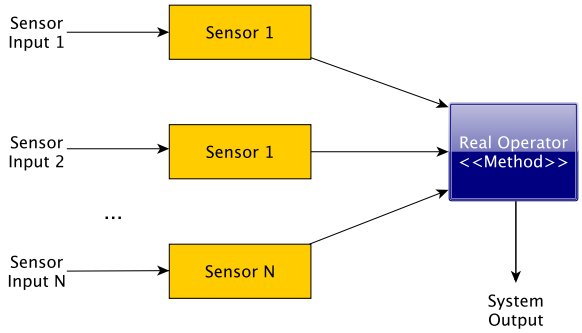

FIG. 5.1. *Sensors composition*

Let us figure out a practical use case of sensor composition. Imagine that there are four temperature sensors available in four different rooms of an apartment. An application would like to know about the instant average temperature of the apartment. A new sensor can be built starting from the four temperature sensors by just applying an *average* operator, as shown in Fig. 5.2.

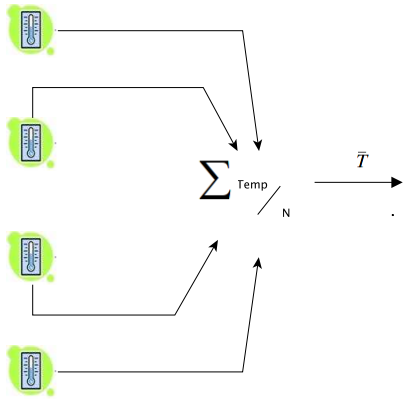

FIG. 5.2. *Average operator*

In this specific case, the input sensors are homogeneous. As stressed earlier, the middleware also provides for the composition of heterogeneous sensors (e.g., temperature, humidity, pressure, proximity), provided that the operator's I/O scheme is adequately designed to be compatible with the sensors' measurement types.

6. Use case scenario. A prototype of the middleware has been implemented and its functionality have been tested. In this section we provide some insight on a real use case that we set up in order to prove the

effectiveness of the implemented mechanisms. In particular, here we focus on what we believe is the most important middleware's provided service, which is the sensors composition service.

The sensors composition process puts emphasis on the semantics of the operation, rather than relying on the simple measure. In this regard, it has been developed a use case, in order to emphasize the power and importance of the aggregation of sensors. Let us recall the average temperature example, and try to describe which are the execution steps triggered in that specific use case.

FIG. 6.1. *Apartment scenario*

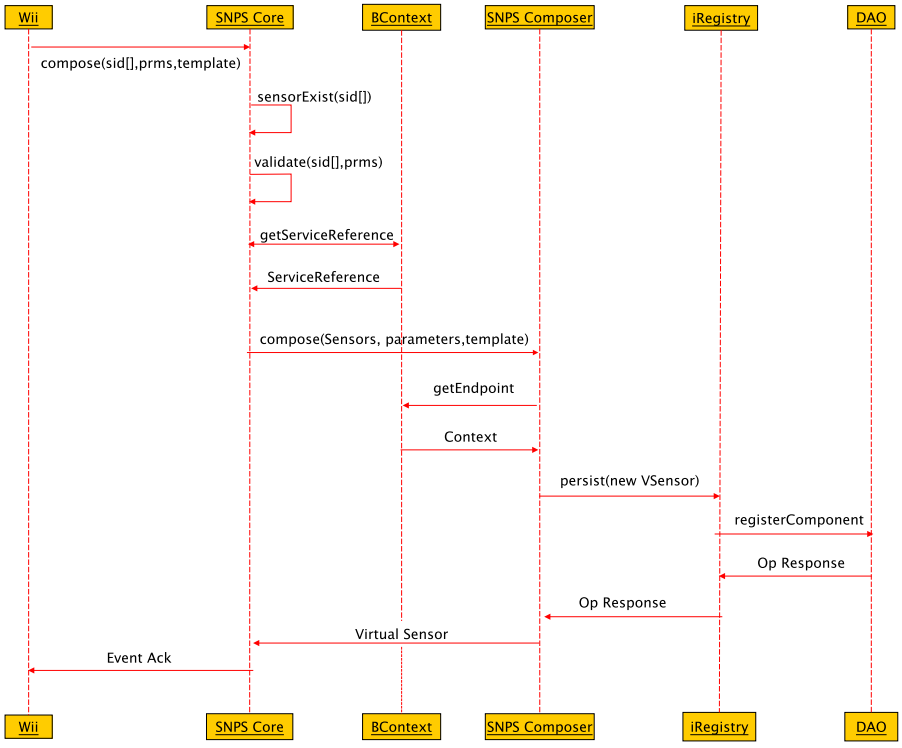

FIG. 6.2. *Sequence diagram for the sensor composition phase*

We will consider the use case as divided into two distinct phases: 1)sensors' composition and 2)aggregate sensor inquiry.

Phase 1: Sensors' composition.. In the first stage, we are going to consider the following actors:

- Web Integration Interface (Wii) Component. It represents the entity generating the composition request;
- Composer. It generates the new virtual sensor from simple temperature sensors;
- Registry. It registers the sensor;
- Core. It orchestrates the composition task among the middleware components.

The operations carried out in the scenario, shown in Fig. 6.2, are the following:

1. The Wii propagates the request to the Core of the platform.
2. The Core retrieves the images of the selected sensors and perform a two-steps validation:
   - Verification of the existence of the sensor images in memory;
   - Validation of the operator to be applied to the sensors;
3. The Core invokes the composition service provided by the Composer;
4. The Composer generates the new (virtual) aggregate sensor;
5. The Registry registers the new sensor;
6. The Core generates a "registration" event for the new sensor, according to the Publish/Subscribe paradigm;

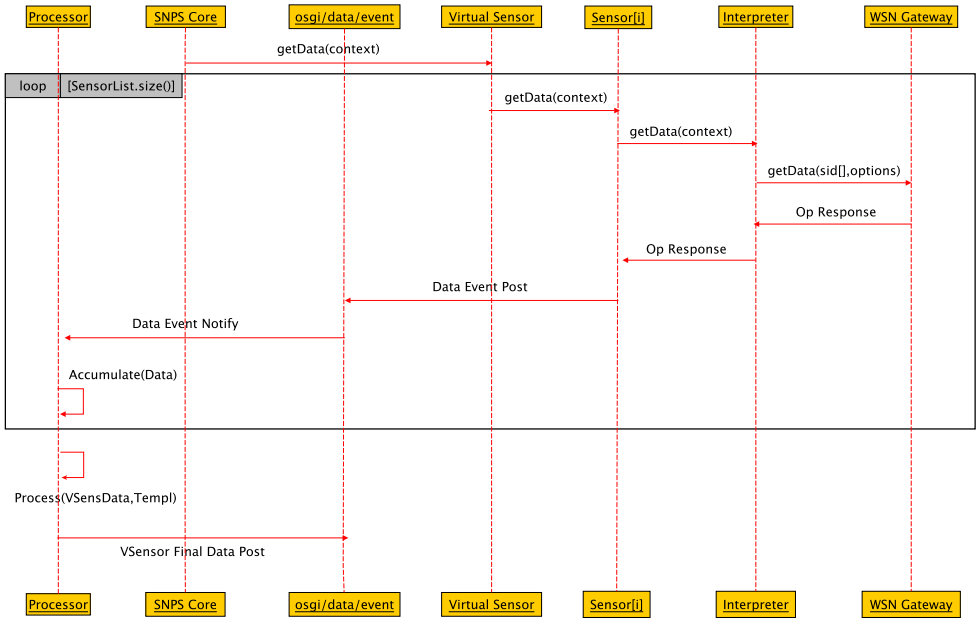

FIG. 6.3. *Sequence diagram for the aggregate sensor inquiry*

Aggregate sensor inquiry.. The steps made in this phase, described in Fig. 6.3, are the following ones:

1. The Wii propagates the request to the Core;
2. The Core, after selecting the aggregate sensor, invokes the get-data operation;
3. The virtual sensor image invokes, for each composing sensor, a service provided by the Sensor Layer Integration (SLI);
4. The SLI propagates the request to the gateway (at WSN Level), which is able to interact directly with the Base Station, which maps the command into a direct command to each physical sensor;
5. After getting the data, the SLI generates a Data response event, which the aggregate sensor is able to collect;
6. The aggregate sensor, finally, applies the operator to the previously collected data, and generates an event on the topic of interest;
8. The Processor records the measurement.

7. Conclusion and future work. The size of data produced by sensors and sensor networks deployed worldwide is growing at a rate that current data analysis tools are not able to follow. Sources of data are multiplying on the Internet (think about smart devices equipped with photo/video cameras). There is a plethora of sensor devices producing information of any kind, at very high rates and according to proprietary specification. This complicates a lot the task of data analysis and manipulation. In this paper we have proposed a solution that aims to ease these tasks. What we propose is not just an early-stage idea but a concrete middleware that implements a mechanism useful to abstract sensors away from their proprietary interfaces and structure, which also offers tool to aggregate and expose sensors and sensor data in the form of services to be accessed in SOA fashion. A prototype of the middleware has been implemented and tested in a small testbed. In the future we are going to conduct extensive experiments to test the scalability and the performance of the middleware in distributed (even geographic) contexts.

Acknowledgment. This work has been partially funded by the Italian project "Sensori" (Industria 2015 - Bando Nuove Tecnologie per il Made in Italy) - Grant agreement n. 00029MI01/2011.

REFERENCES

[1] O. Alliance, *Open Service Gateway initiative (OSGi)*, 2013. Available at http://www.osgi.org/.
[2] E. Aviles-Lopez, and J. Antonio, Garcia-Macias, J. Antonio, *Tinysoa: a service-oriented architecture for wireless sensor networks*, Service Oriented Computing and Applications, (2009).
[3] R. Barr, J. C. Bicket, D. S. Dantas, B. Du, T. W. D. Kim, B. Zhou, and E. G. Sirer, *On the need for system-level support for ad hoc and sensor networks*, SIGOPS Oper. Syst. Rev., (2002), pp. 1–5.
[4] M. Botts, G. Percivall, C. Reed, and J. Davidson, *Ogc sensor web enablement: Overview and high level architecture*, in GeoSensor Networks, S. Nittel, A. Labrinidis, and A. Stefanidis, eds., vol. 4540 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2008, pp. 175–190.
[5] E. Caete, J. Chen, M. Diaz, L. Llopis, and B. Rubio, *Useme: A service-oriented framework for wireless sensor and actor networks*, in Applications and Services in Wireless Networks, 2008. ASWN '08. Eighth International Workshop on, 2008.
[6] M. Caporuscio, P.-G. Raverdy, and V. Issarny, *ubisoap: A service-oriented middleware for ubiquitous networking*, Services Computing, IEEE Transactions on, (2012).
[7] A. Demers, J. Gehrke, R. Rajaraman, N. Trigoni, and Y. Yao, *The cougar project: a work-in-progress report*, SIGMOD Rec., (2003).
[8] G. Di Modica, F. Pantano, and O. Tomarchio, *SNPS: an OSGi-based middleware for Wireless Sensor Networks*, in Advances in Service-Oriented and Cloud Computing, Communications in Computer and Information Science, 393 (2013), pp. 1–12.
[9] G. Di Modica, O. Tomarchio, and L. Vita, *A P2P based architecture for Semantic Web Service discovery*, International Journal of Software Engineering and Knowledge Engineering, 21 (2011), pp. 1013–1035.
[10] L. Gurgen, C. Roncancio, C. Labbé, A. Bottaro, and V. Olive, *Sstreamware: a service oriented middleware for heterogeneous sensor data management*, in Proceedings of the 5th international conference on Pervasive services, ACM, 2008.
[11] S. Hadim and N. Mohamed, *Middleware: Middleware challenges and approaches for wireless sensor networks*, IEEE Distributed Systems Online, 7 (2006), pp. 1–.
[12] W. Heinzelman, A. Murphy, H. Carvalho, and M. Perillo, *Middleware to support sensor network applications*, Network, IEEE, (2004).
[13] IEEE Network Working Group, *JavaScript Object Notation (JSON)*, 2006.
[14] V. Issarny, N. Georgantas, S. Hachem, A. Zarras, P. Vassiliadist, M. Autili, M. A. Gerosa, and A. B. Hamida, *Service-oriented middleware for the Future Internet: state of the art and research directions*, Journal of Internet Services and Applications, 2 (2011), pp. 23–45.
[15] K. Khedo and R. K. Subramanian, *Meeca: Misense energy efficient clustering algorithm*, in Wireless Communication and Sensor Networks, 2007. WCSN '07. Third International Conference on, 2007.
[16] X. Koutsoukos, M. Kushwaha, I. Amundson, S. Neema, and J. Sztipanovits, *Oasis: A service-oriented architecture for ambient-aware sensor networks*, in Composition of Embedded Systems. Scientific and Industrial Issues, F. Kordon and O. Sokolsky, eds., Springer Berlin Heidelberg, 2007.
[17] P. Levis and D. Culler, *Mate: a tiny virtual machine for sensor networks*, in Proceedings of the 10th international conference on Architectural support for programming languages and operating systems, ACM, 2002, pp. 85–95.
[18] S. Li, S. Son, and J. Stankovic, *Event detection services using data service middleware in distributed sensor networks*, in Information Processing in Sensor Networks, F. Zhao and L. Guibas, eds., Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2003.
[19] T. Liu and M. Martonosi, *Impala: a middleware system for managing autonomic, parallel sensor systems*, SIGPLAN Not., (2003).
[20] D. Miorandi, S. Sicari, F. D. Pellegrini, and I. Chlamtac, *Internet of things: Vision, applications and research challenges*, Ad Hoc Networks, 10 (2012), pp. 1497 – 1516.

[21] N. Mohamed and J. Al-Jaroodi, *A survey on service-oriented middleware for wireless sensor networks*, Service Oriented Computing and Applications, 5 (2011), pp. 71–85.
[22] L. Mottola and G. P. Picco, *Programming wireless sensor networks: Fundamental concepts and state of the art*, ACM Comput. Surv., 43 (2011), pp. 19:1–19:51.
[23] OGC, *Sensor Web Enablement (SWE)*, 2013. Available at http://www.opengeospatial.org/ogc/markets-technologies/swe/.
[24] M. P. Papazoglou and W.-J. van den Heuvel, *Service Oriented Architectures: approaches, technologies and research issues*, VLDB Journal, 16 (2007), pp. 389–415.
[25] C. Srisathapornphat, C. Jaikaeo, and C.-C. Shen, *Sensor information networking architecture*, in Parallel Processing, 2000. Proceedings. 2000 International Workshops on, 2000, pp. 23–30.
[26] R. Sugihara and R. K. Gupta, *Programming models for sensor networks: A survey*, ACM Trans. Sen. Netw., 4 (2008), pp. 8:1–8:29.
[27] MathML, *MathML(W3C)*, 2013. Available at http://www.w3.org/Math/.

Edited by: Maria Fazio and Nik Bessis
Received: Nov 2, 2013
Accepted: Jan 10, 2014