SNPS: An OSGi-Based Middleware for Wireless Sensor Networks

Giuseppe Di Modica, Francesco Pantano, and Orazio Tomarchio

Department of Electric, Electronic and Computer Engineering
University of Catania
Catania, Italy
`firstname.lastname@dieei.unict.it`

Abstract. We are witnessing a widespread deployment of sensors and sensor networks in any application domain. These sensors produce huge amounts of raw data that need to be structured, stored, analyzed, correlated and mined in a reliable and scalable way. Some application environments also add real-time requirements which make things even harder to manage. 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 when they are produced. In this paper we propose a middleware, to be deployed on top of physical sensors and sensor networks, capable of abstracting sensors 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.

1 Introduction

Nowadays, sensors are everywhere. You may find them in your smartphone, in your house, in your car, in streets, and so on. They measure various phenomena, and can be used for very different purposes: monitoring, surveillance, prediction, controlling. Their number is actually increasing day by day, as foreseen by the Internet of Things (IoT) vision [9]. A huge amount of data is generated each second but we are far from taking advantage of all this “potential” knowledge.

There are several reasons for that: heterogeneous sensor networks are usually disconnected among them, and often are still not connected to a globally accessible information network. Even in the case they are connected, we do not know how to search for those sensors which may be of help for our purpose. Moreover, when we find a potentially interesting sensor, often we are not able to get data from it due to its proprietary data interface or, if we succeed to get data, we need to correctly interpret its meaning. In addition, when developing an application that has to use a sensor network for monitoring a certain phenomena, the application programmer should only concentrate on application-level issues and ideally use the programming languages, tools and methodologies that he is accustomed to.

We believe that the service-oriented approach [7,13] 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 we present an OSGi-based middleware, called *Sensor Node Plug-in System (SNPS)*, where sensors are no longer low-level devices producing raw measurement data, but are seen as "services" able 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. In Section 3 the architecture of the proposed solution is introduced. In Section 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. We conclude our work in Section 6.

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 initiative [2,12]. 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.

A common misconception of the adoption of SWE standards is that they, instead of encapsulating sensor information on application level, were originally designed to operate directly on a hardware level. Of course, supporting interoperable access on the hardware level has some advantages and comes very close to 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 SWE of uncoupling sensor information from sensor systems, the downside of this approach arise when dealing with a high number of specialized and heterogeneous sensor systems, and in resource-limited scenario (as typical WSNs) where communication and data transportation operations have to be highly optimized. Even a relatively powerful sensor gateway is not necessarily suitable as a web server in many cases it may typically be networked via a low-bandwidth network and powered by a battery and so it has neither the energy or bandwidth resources required to provide a web service interface.

The need for an intermediate software layer (middleware) derives from the gap between the high-level requirements from pervasive computing applications and the complexity of the operations in the underlying WSNs. The complexity of the operations within a WSN is characterized by constrained resources, 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 middleware helps the programmer develop applications in several ways: it provides appropriate system abstractions, reusable code services and data services. It helps the programmer in network infrastructure management and providing efficient resource services.

Some research efforts have been done on surveying the different aspects of middleware and programming paradigms for WSN. For example, [5] analyzed different middleware challenges and approaches for WSN, while [14] and [11] analyzed programming models for sensor networks.

As an example of different approaches, we cite here TinyDB [8], a query processing system based on SQL-like queries that are submitted by the user at a base station where the application intelligence resides. Enabling dynamic reconfiguration is one of the main motivations for component-based designs like the RUNES middleware [3]. Finally, operating systems for WSNs are typically simple, providing basic mechanisms to schedule concurrent tasks and access the hardware. In this respect, a representative example is TinyOS [15] and the accompanying nesC language.

Very recently, to provide high flexibility and for adding new and advanced functions to WSN middleware, the service-oriented approach has been applied to sensor environments [7,10]. 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.

In this context, the OSGi technology [1] defines a standardized, component/service oriented, computing environment for networked services. Enabling a networked device with an OSGi framework adds the capability to manage the life cycle of the software components in the device from anywhere in the network without ever having to disrupt the operation of the device. In addition, the service oriented paradigm allows for a more smooth integration with Cloud platforms and for advanced discovery mechanisms also employing semantic technologies [4].

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 [9]. 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 [13]. 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 implements a component-oriented model, which natively supports the component's life cycle management, the distribution of components over remote locations, the seamless management of components' inter-dependencies, and the asynchronous communication paradigm.

The SNPS middleware was then organized into several components, and each component was later implemented as a software module (or "bundle") within the OSGi framework. Figure 1 depicts the architecture of the middleware and its main components.

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.

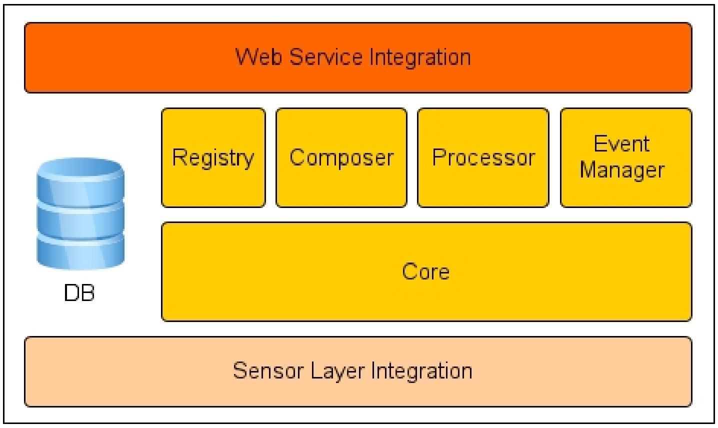


Fig. 1. SNPS architecture

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 2 the connections among the components are depicted.

Core. It is where the business logic of the Middleware resides. The Core acts as an orchestrator who 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.

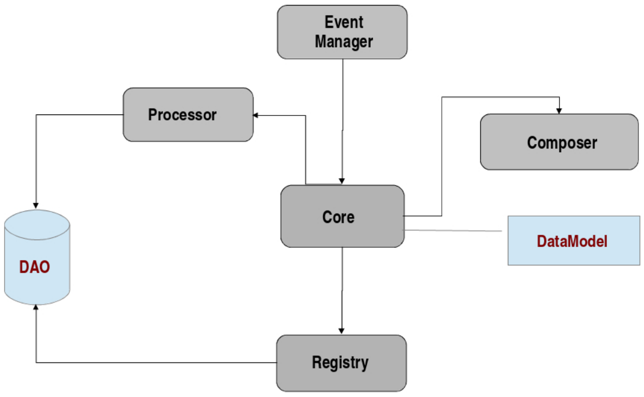


Fig. 2. Core and related Components

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.

Processor. It is the component responsible for the manipulation of the data flow coming from the sensors. In particular, it provides a service to set and enforce a sampling plan on a single sensor or on an aggregate of sensors. Also, this component can be instructed to process data according to specific processing templates.

Composer. It represents the component which implements the sensors' composition service. Physical sensors can be "virtualized" and are given a uniform representation which allows for "aggregating" multiple virtualized sensors into one sensor that will eventually be exposed to applications. An insight and practical examples about this functionality are provided in Section 5.

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 a remote "OSGI Context", which is a specific OSGi's features allowing bundles living in different runtime contexts to communicate to each other's. In Figure 3 the two bundles and their respective runtime contexts are shown.

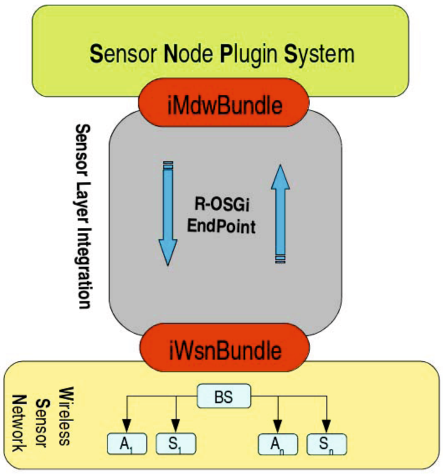


Fig. 3. OSGi bundles implementing the Sensor Layer Integration

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 uniforming the access to sensors' functionality. Uniforming 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 depicted in Figure 4, the *OSGi bundle Wrapper* exports the functionality of the SNPS middleware to a Web Service context.

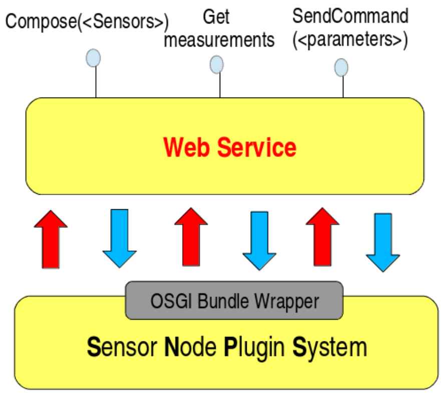


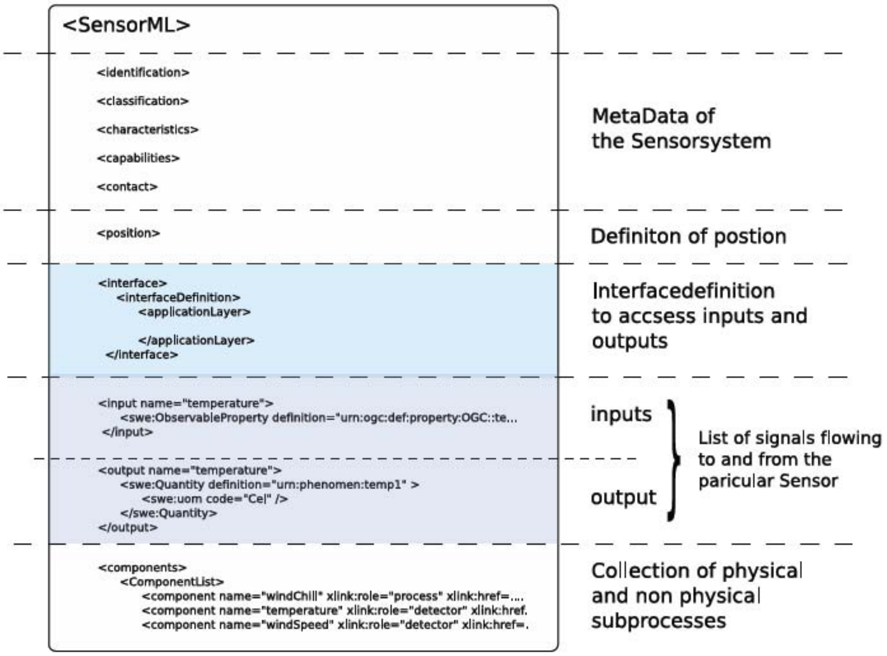
Fig. 4. Wrapping and exposing SPNS as a Web Service

SNPS services can be invoked from any OSGi compliant context. On the other hand, making the SNPS accessible as a plain Web Service will make its services profitable for a great number of applications in several domains. The functionality implemented by the SNPS' bundles have been packaged into the following categories of services:

– Search for sensors;
– Retrieve sensors capabilities and sensors data;
– Compose sensors;
– Send commands to sensors (enable/disable, set a sampling plan).

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 [2] 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[6] format to structure the data produced by sensors. An excerpt of what a description of sensor capabilities look like is depicted in Figure 5.



```
<SensorML>
    <identification>
    <classification>
    <characteristics>
    <capabilities>
    <contact>

    <position>

    <interface>
        <interfaceDefinition>
            <applicationLayer>

            </applicationLayer>
     </interface>

    <input name="temperature">
        <swe:ObservableProperty definition="urn:ogc:def:property:OGC::te...
     </input>

    <output name="temperature">
        <swe:Quantity definition="urn:phenomen:temp1" >
            <swe:uom code="Cel" />
        </swe:Quantity>
    </output>

    <components>
        <ComponentList>
            <component name="windChill" xlink:role="process" xlink:href=....
            <component name="temperature" xlink:role="detector" xlink:href.
            <component name="windSpeed" xlink:role="detector" xlink:href=.
```

MetaData of the Sensorsystem
Definiton of postion
Interfacedefinition to accsess inputs and outputs
inputs
output
List of signals flowing to and from the paricular Sensor
Collection of physical and non physical subprocesses

Fig. 5. Description of sensor capabilities in SensorML

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 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 Section 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 (see Figure 1).

An important prerequisite of the composition is the sensor "virtualization", which is a procedure performed when a sensor is plugged into the SNPS middleware (see Section 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 selected. 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. By that time, a new virtual sensor (the aggregate) is available in the system, and is exposed as a new sensor by the middleware.

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 applying the average operator, as depicted in Figure 6.

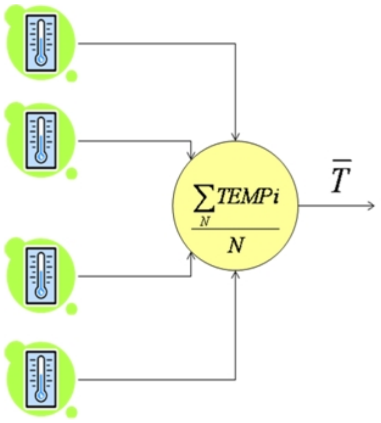


Fig. 6. Average operator

In this specific case, the input sensors are homogeneous. 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 Conclusion

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 have proposed is not just an early-stage idea but a concrete middleware that implements a mechanism to abstract sensors away from their proprietary interfaces and structure, and 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. 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.

Acknowledgments. 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. OSGi Alliance: Open Service Gateway initiative, OSGi (2013), `http://www.osgi.org/`
2. Botts, M., Percivall, G., Reed, C., Davidson, J.: OGC Sensor Web Enablement: Overview and high level architecture. In: Nittel, S., Labrinidis, A., Stefanidis, A. (eds.) GSN 2006. LNCS, vol. 4540, pp. 175–190. Springer, Heidelberg (2008)
3. Costa, P., Coulson, G., Gold, R., Lad, M., Mascolo, C., Mottola, L., Picco, G.P., Sivaharan, T., Weerasinghe, N., Zachariadis, S.: The runes middleware for networked embedded systems and its application in a disaster management scenario. In: Fifth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom 2007), pp. 69–78. IEEE Computer Society (2007)
4. Di Modica, G., Tomarchio, O., Vita, L.: A P2P based architecture for Semantic Web Service discovery. International Journal of Software Engineering and Knowledge Engineering 21(7), 1013–1035 (2011)
5. Hadim, S., Mohamed, N.: Middleware: Middleware challenges and approaches for wireless sensor networks. IEEE Distributed Systems Online 7(3), 1 (2006)
6. IEEE Network Working Group: JavaScript Object Notation, JSON (2006), `http://www.ietf.org/rfc/rfc4627.txt?number=4627`
7. Issarny, V., Georgantas, N., Hachem, S., Zarras, A., Vassiliadist, P., Autili, M., Gerosa, M.A., Hamida, A.B.: Service-oriented middleware for the Future Internet: state of the art and research directions. Journal of Internet Services and Applications 2(1), 23–45 (2011)
8. Madden, S.R., Franklin, M.J., Hellerstein, J.M., Hong, W.: Tinydb: an acquisitional query processing system for sensor networks. ACM Trans. Database Syst. 30(1), 122–173 (2005)
9. Miorandi, D., Sicari, S., Pellegrini, F.D., Chlamtac, I.: Internet of things: Vision, applications and research challenges. Ad Hoc Networks 10(7), 1497–1516 (2012)
10. Mohamed, N., Al-Jaroodi, J.: A survey on service-oriented middleware for wireless sensor networks. Service Oriented Computing and Applications 5(2), 71–85 (2011)
11. Mottola, L., Picco, G.P.: Programming wireless sensor networks: Fundamental concepts and state of the art. ACM Comput. Surv. 43(3), 19:1–19:51 (2011)
12. OGC: Sensor Web Enablement, SWE (2013), `http://www.opengeospatial.org/ogc/markets-technologies/swe/`
13. Papazoglou, M.P., van den Heuvel, W.J.: Service Oriented Architectures: approaches, technologies and research issues. VLDB Journal 16(3), 389–415 (2007)
14. Sugihara, R., Gupta, R.K.: Programming models for sensor networks: A survey. ACM Trans. Sen. Netw. 4(2), 8:1–8:29 (2008)
15. TinyOS community: TinyOS (2013), `http://www.tinyos.net/`