Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2013, Article ID 560418, 17 pages
http://dx.doi.org/10.1155/2013/560418

Research Article

SMArc: A Proposal for a Smart, Semantic Middleware Architecture Focused on Smart City Energy Management

Jesús Rodríguez-Molina, José-Fernán Martínez, Pedro Castillejo, and Rubén de Diego

Centro de Investigación en Tecnologías Software y Sistemas Multimedia para la Sostenibilidad (CITSEM), Edificio La Arboleda, Campus Sur UPM, Carretera de Valencia, Km 7, 28031 Madrid, Spain

Correspondence should be addressed to Jesús Rodríguez-Molina; jrodmolina@diatel.upm.es

Received 5 July 2013; Revised 6 October 2013; Accepted 20 November 2013

Academic Editor: Yuan He

Among the main features that are intended to become part of what can be expected from the Smart City, one of them should be an improved energy management system, in order to benefit from a healthier relation with the environment, minimize energy expenses, and offer dynamic market opportunities. A Smart Grid seems like a very suitable infrastructure for this objective, as it guarantees a two-way information flow that will provide the means for energy management enhancement. However, to obtain all the required information, another entity must care about all the devices required to gather the data. What is more, this entity must consider the lifespan of the devices within the Smart Grid—when they are turned on and off or when new appliances are added—along with the services that devices are able to provide. This paper puts forward SMArc—an acronym for semantic middleware architecture—as a middleware proposal for the Smart Grid, so as to process the collected data and use it to insulate applications from the complexity of the metering facilities and guarantee that any change that may happen at these lower levels will be updated for future actions in the system.

1. Introduction

Energy constrains and consumption issues are increasingly putting a strain on the development of human settlements and, more notoriously, medium and large cities. Considering that in the following years more people than ever will live in towns and suburban areas, there are certain challenges that must be faced at a scale hardly ever seen before in terms of mobility, energy resources, or pollution. It is here when the concept of Smart City comes up. As cited by Kehua et al. in [1], a Smart City will use information and communication technologies focused on sensing, analyzing, and integrating data of critical importance obtained from city core systems. According to the authors, a Smart City will be integrating smart planning ideas, smart construction modes, smart development approaches, and smart management methods. Among the aspects the Smart City may deal with—as waste treatment or transport, with intense research being made involving the latter [2]—this paper deals with the issues related to power distribution, delivery and consumption, and the software architectures that can be used to manage and improve the overall performance of these entities. Unsurprisingly, the Smart Grid will play a key role in these improvements.

1.1. The Need of a Smart Grid. The Smart Grid has been defined by several authors in fairly different manners. For example, in [3] it is claimed to be "*the use of sensors, communications, computational ability and control in some form to enhance the overall functionality of the electric power delivery system. A dumb system becomes smart by sensing, communicating, applying intelligence, exercising control and through feedback, continually adjusting.*" In spite of the variety, there are several advantages from the Smart Grid that are recognized and shared: sensing devices are used for monitoring and controlling other hardware entities, automation is used for several purposes, and intelligence is used to enhance its capabilities in terms of forecasting and data collecting. Expectations on what the Smart Grid is capable of doing

to improve energy management are high, and the literature about how it can be used is widespread; electric vehicles can benefit from its usage [4] or reduce the carbon footprint produced by massive energy consumption during peak hours [5]. However, there is very little concern about what the Smart Grid should be made of. In our opinion, the Smart Grid is not different from other systems that are bound to a layered architecture model, although instead of having a monolithic, fixed variety of similar components at the hardware level it consists of a wide range of heterogeneous meters, smart meters, or Supervisory Control and Data Acquisition systems (SCADAs), each of them performing tasks related to data harvesting or infrastructure monitoring involving different environments (factories, dwellings, department stores, etc.). As for the other levels, they are the ones that can be expected: an operating system, either different in each of the working devices or a single one managing a wide area of the Smart City, will interact with the differing capabilities of the hardware elements and will send any requested or required data to the communication devices present at its upper level, that is to say, the network layer.

1.2. The Need of a Middleware Architecture. It must be taken into account that, as the information is obtained from appliances of very different properties, chances are that it is depicted in varying representation formats (little endian, big endian, etc.) that will collide with the expected one, so elements within the Smart Grid must come to an agreement on how data must be transferred. Since having many of the metering and/or monitoring devices removed and changed with a single model could be a challenging, expensive, and time-consuming task (as part of the Smart City, the Smart Grid is supposed to be equipped with thousands of these pieces of equipment), the agreement must be based on adapting the different data representations into a single one that is able to offer easily accessible services at the application layer. This is the moment when a middleware layer has to be considered, for its most important functionalities will be dealing with problems as scalability, interoperability, or device heterogeneity, aiming to shield the underlying complexity of a system and the heterogeneity typical of different operating systems, computer architectures, or networking protocols [6]. The global architecture of the Smart Grid ought to look as displayed in Figure 1.

Therefore, it is only natural that middleware architectures have been used for interconnectivity and interoperability issues for quite a time. Solutions like Remote Method Invocation or RMI, that is to say, a Java-based middleware architecture made up by three different layers (Stub/Skeleton, Remote Reference Layer, and Transport Layer) used to invoke a Java method on a remote machine [7] and Common Object Request Broker Architecture (CORBA), which can be regarded as a language for independent object buses that interact via a client-server architecture employing Object Requests Brokers [8], tend to fit in well with distributed systems that are deployed on regular equipment like personal computers, laptops, or even smartphones, for they have an abundant amount of computational and energetic resources.

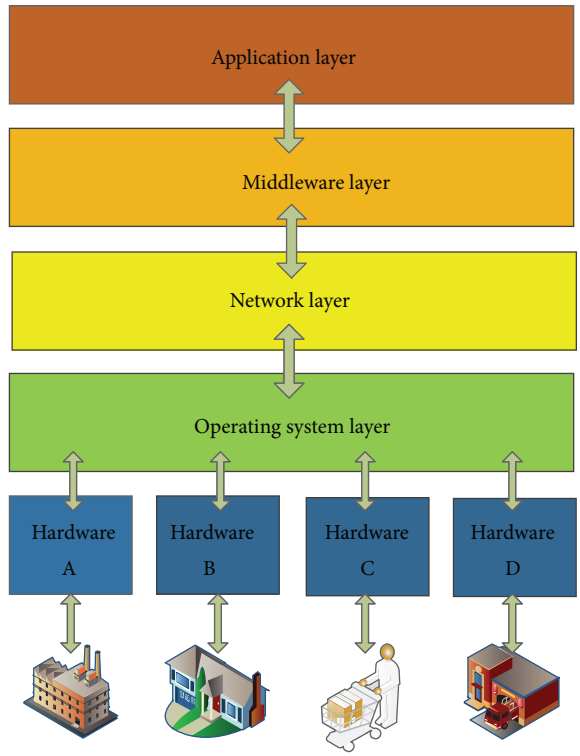

Figure 1: Location of the middleware layer.

1.3. The Need of a Semantic Middleware Architecture. There are several other questions that must be dealt with when middleware architectures are designed for Smart City-related environments, though. Unlike conventional network architectures, where there are minor changes in quantity and capabilities of the devices that get connected every day, Smart Cities are prone to have a more dynamic behaviour in terms of services and applications: there may be services that become unusable due to several reasons (battery depletion, security quarantines, node unavailability or damage, etc.) or, on the other hand, brand new services may become available (new devices are installed; former ones are augmented or replaced by more powerful appliances, etc.) by attaching new equipment that may or may not have been previously taken into account for communications and data transfers. This poses obvious challenges in terms of interoperability and interconnectivity, especially considering that the Smart City is likely to have millions of sensors and actuators once a city-wide infrastructure is deployed. Furthermore, sensors and actuators do not have hardware capabilities as abundant as the devices that were usual in former middleware platforms, as noted by Yufei et al. [9] or Bicen et al. [10]. Clearly, traditional middleware solutions do not seem to match the requirements and challenges that have to be faced in the Smart Grid or the Smart City.

Fortunately, there are ways that can be exploited instead in order to solve the presented issues. Among the available options, the usage of semantics seems like an option to be considered. When talking about the semantics in the context of middleware architectures, it is referred to the possibility

of extracting information about the meaning of a message, rather than just transmitting it from one source to one destination. Thus, if information about device capabilities can be obtained, regardless of their differing hardware characteristics (data storage, ROM, RAM, etc.) or software (operating system, plug-ins, etc.) in a format that will be defined equally for all the system components, then the middleware architecture, and by proxy the Smart Grid and the Smart City, will become way more flexible in terms of what devices it is able to incorporate or dismiss. Additionally, if the middleware architecture is made dynamic enough, interconnectivity and interoperability issues can be reduced to a minimal extent. The tool that will be used to obtain this "domain standardization" as it will be involving just the system it is used in is an ontology model. In the context of information technologies, an ontology model is a representation not only of the different entities that belong to a field of knowledge but also of the relationships among them. For example, Albarrak and Sibley [11] mention how the development of an information model represented by ontology is a compelling task that requires many different skills: relationships, restrictions, concepts, and identification of properties. In addition to that, a good grasp on ontology modeling and ontology languages is a must have. Consequently, the different elements that belong to a system, along with their capabilities and/or constrains, can be accurately defined by the use of semantics, and any semantic architecture will employ an information model according to several concepts kept in one or several ontologies.

Commonly, an ontology model will work in a semantically augmented system as follows: anytime a new device is added to the whole system, it will request in a mutually intelligible format, as an application layer language like eXtensible Markup Language (XML) or JavaScript Object Notation (JSON), how information will be retrieved (e.g., if temperature format is requested, the query will contain whether Fahrenheit, Kelvin, or Celsius degrees should be used, how many digits should the answer provide, etc.). This request will be pointed at any repository that is containing the ontology, which keeps the information representation format expected from the data interchange. Once the format is given by the ontology and collected by the new device, any ulterior data transfer will be done taking the ontology model into account, as expressed in Figure 2.

Note that this consulting process is necessary because the information content and format are relevant for the final datum meaning (i.e., its semantics); if content and format are not observed, then datum semantic cannot be apprehended and queries will not be attended correctly. When the three most important stages of Figure 2 are considered, it becomes obvious that data format, in terms or hierarchy and information, becomes a critical part of the whole system. An example of that information representation flow is described in Figure 3. To begin with, the request that will be sent from the device to the ontology repository will be devoid of the data format required (Stage 1). Nevertheless, since the repository is aware of the nature of the data that is queried, it will send back a reply containing how the data format is expected to be (Stage 2). From that moment on, whenever there is a data transmission it will be done so according to an established data hierarchy (Stage 3 and Interactions).

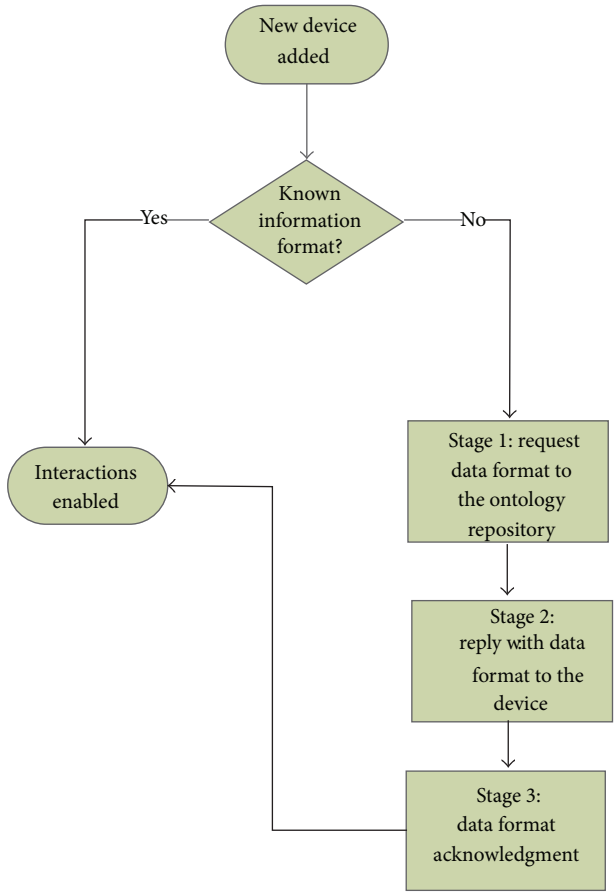

Figure 2: Flowchart on the process of data format retrieval.

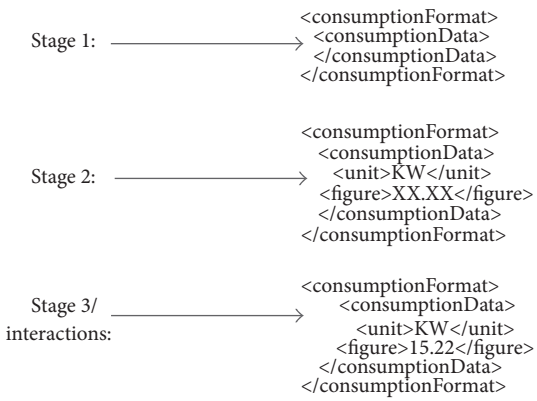

Figure 3: XML-based data representation format.

This paper is structured as follows: an introduction about the most important topics that this document is going to be about has already been presented. Next section will offer the related works in the field of middleware. Section 3 will present the main characteristics of SMArc (semantic middleware architecture) by showing both computational and functional analyses. Once our proposal has been put

forward and described, conclusions will be presented, along with future work lines pointing at the development in Smart Grid middleware.

2. Related Works

There are many ways in which middleware can be classified, and according to the feature of choice, several surveys have been done. The ones that have been included here are the surveys regarding middleware architectures that seem to be most likely to fit within the scope of this paper. Should a perspective of cloud computing and Mobile Social Networks be obtained? surveys on cloud computing by Dey [12] and on Mobile Social Networks made by A. Karam and N. Mohamed [13] are advised to be checked.

2.1. Study on Existing Middleware Proposals. Aamna Saeed and Tabinda Waheed put forward a survey focused on middleware architectures enabled with context awareness [14], which they define as systems made up by distributed components as sensors, actuators, context information processors, and context information stores among other elements. Context awareness has been defined as the information used to characterise the situation of an entity [12], and in a holistic manner it can be referred to as the capability of operating in different ways depending on the information retrieved from the environment an entity is located in. Therefore, there are several architectures described by the authors that attempt to operate taking their surroundings into account. For example, *Aura* works by providing an "aura" to a user that acts as a proxy for services, and whenever a new environment is entered the "aura" will readapt itself. *Cooltown* is another middleware architecture considered in this survey as a way of establishing communications between mobile devices wirelessly connected and a web-based environment by portraying present devices as URLs. None of these or the other middleware conceptions, though, have strong semantic capabilities or have been designed taking the Smart Grid or the Smart City into account.

Perera et al. [13] have also made an extensive paper on the different middleware architectures that have been developed in a decade (2001–2011) with a strong component in context awareness. Their criticism is focused on the issue that there are very few middleware architectures that are truly context-aware; Hydra, UBIWARE, UBIROAD, and SMEPP are the only four quoted as the ones having that feature. The most thoroughly described architecture, *Hydra*, attempts to integrate in an IoT-related architecture (ambient intelligence systems) wireless devices and sensors, and, additionally, it makes use of a Context Aware Framework or CAF so as to provide context awareness. However, it does not take into account the changing nature of the Smart Grid.

Additionally, Knappmeyer et al. have made a survey on context provisioning middleware [15]. JCAF and again SOCAM are put forward as examples of implementations based on RMI. Middleware is classified according to several features that go beyond context awareness (layered, object oriented or event-based, direct sensor coupling, central server, central server with distributed components, or peer-to-peer architecture), but as far as the Smart Grid or the Smart City are concerned, presented middleware architectures either do not match the challenges that have been stated (semantics, low capability devices) or are dealing with them only partially (security and privacy).

Wang et al. survey middleware architectures based on events rather than context awareness [16]. Apart from describing already known solutions as CORBA, there are three more architectures that are described: JEDI, Siena, and Hermes. *JEDI* is introduced as an object-oriented infrastructure conceived for operation and development within event-based systems. At the same time, *Siena* is an event notification service claimed to have been deployed at an Internet scale. Finally, *Hermes* is claimed to be a distributed event-based middleware platform bent on employing a peer-to-peer infrastructure. However, there are several features that must be added in the context of a Smart Grid; security is not mentioned under any condition, and low capability devices are not explicitly taken into account.

Other technologies strongly related to ubiquitous computing or the Internet of Things are put to a use as well. Among these, RFID is one of the most popular, so research has been conducted by Syta et al. on a *RFID Authentication Middleware* for Mobile Devices [17]. Although this development seems to have quite an impact on the area of mobile services, it is simply too far away from the concept of the Smart Grid and has not been conceived for its usage in that context.

Another set of proposals is given in the survey undertaken by Fatos Xhafa et al., conceiving middleware somewhat from a perspective of grid and peer-to-peer collaboration [18], although the authors make clear that peer-to-peer is not understood as a way to share files but to access computational data. The grid middleware architectures proposed by this latter survey are as follows: gLite, NetSolve/GridSolve, Globus and Globus-based middleware, MPI-based middleware, and Java RMI-based middleware. *Globus and Globus-based* middleware are regarded by the authors of the survey as the most popular middleware architecture related to their idea of grid middleware. In fact, it has several features that are of great usefulness, as coping with security and heterogeneity issues and providing resource discovery. Plus, some other services as resource allocation, authentication and security, system monitoring, process management, or remote access are also available. What is more, Globus has become integrated as part of the NetSolve project. Sadly, this collection of solutions has not been designed bearing the Smart Grid in mind, and its mapping and implementation in an environment like this remain a challenge.

Other proposals try to recreate facilities that are present in more regular environments. For example, security is the main topic of the survey done by Sain et al. [19]. The authors claim that security must be considered as a major concern as many activities have their start from mobile (banking, personal healthcare, etc.) or Wireless Sensor Networks (which are increasingly being enhanced with security-related features as in [20]). Thus, security applications extend to a wide area (healthcare, databases, security middleware, and etc.) and

software is done bearing security in mind. The studied middleware architectures (which have been gathered as Message-Oriented, Database, Reflective, Tuple Space, Event Based, and Service Discovery Middleware) are qualified as low with regard to security implementation [19]. Although these architectures cannot be considered compatible with what is required for a Smart Grid, security-related functionalities are a feature that is considered in our proposal.

Middleware for Wireless Sensor Networks is also presented. Mohamed and Al-Jaroodi have made a survey on Service-Oriented Middleware (SOM) approaches for Wireless Sensor Networks [21]. In this paper, SStreaMWare, USEME, SensorWeb 2.0, OASiS, B-VIS, MiSense, SOMDM $(SI)^2$, SOA-MM, and ubiSOAP are described. One of the most interesting is *SensorWeb 2.0*, a research project that, among other issues, tackles how to create a SOM able to support data collection and manipulation in sensor networks of heterogeneous nature and how to use sensor data through extended periods of time and support contiguous sensor data access. At the same time, *SOMDM* is portrayed as a service-oriented, message-driven middleware that tries to keep the overhead of received and transmitted messages to a minimum by using Service Oriented Architecture facilities. Finally, *ubiSOAP* is a ubiquitous, SOAP-based middleware architecture aiming to provide seamless connection with web services and even legacy SOAP systems. These middleware architectures seem to be the best fitting ones for a Smart Grid environment; however, they make very little use of semantic features and do not strongly enforce important mechanisms for scalability or interoperability.

2.2. Main Challenges. When comparing the already presented work with the one that is described here, several challenges spring up: although there are a huge number of described middleware developments, none of them are successfully tackling all the issues that can be expected from an environment as the ones within the scope of this paper; while there are middleware architectures that deal with low capability devices in a Wireless Sensor Network and offer security or fare efficiently as part of distributed environments, there is not a proposal that combines all of these features into a single architecture. This is mostly due to the fact that the most prominent middleware architectures presented here have not been designed with the idea of adapting them to a Smart Grid or a Smart City.

What is more, other important features as semantics are severely missing in most proposals. This is an important issue, for the scalability and interoperability of the system cannot be easily improved in the long term, as different devices of varying capabilities may become trendy in the future and the middleware infrastructure could end up outdated or, in the worst case scenario, unusable when new devices had to be added.

Finally, none of the proposals suggests a way to make decisions or infer actions depending on the operations that have been carried out previously. The lack of this feature diminishes the operability and intelligence of the systems and should be any add-on aiming to provide semantic capabilities; it would not be completely exploited with inference elements missing.

Table 1 is showing the main strengths and weaknesses of the studied architectures. As it can be seen, they are usually lacking several features that would make them a killer application in terms of middleware for the Smart Grid. Specifically, semantic features and inference engines are missing in almost all of them, and, at the same time, none of them has been specifically designed to take into account all constrains and needs of the Smart Grid.

3. SMArc as a Semantic Middleware Proposal

SMArc (Semantic Middleware Architecture) provides several contributions not found on any other middleware proposals for the Smart City or the Smart Grid, either by offering them as part of one single architecture (low capability devices, work as a distributed system, etc.) or by being new to this environment (semantics, inference engine).

(i) *SMArc takes into account low capability devices,* critical for data collection in a Smart Grid. In fact, it has been implemented with low capability devices, as it will be explained later.

(ii) *SMArc offers security features so as to give privacy to the harvested information.* This is a characteristic hard to find in other architectures that may conceive security as a mere afterthought

(iii) *SMArc works as a distributed system,* for it will require the information of scattered elements through a system as extended and widespread as the power network.

(iv) *SMArc has been specifically designed for the Smart Grid.* Unlike any other proposals of the multiple ones studied, SMArc is taking into account the common issues that middleware may present under this environment: addition of new devices of dissimilar nature, composed service requests, or event triggering when a particular situation requires it.

(v) *SMArc takes into account semantic features.* A light ontology and data representation under a specific format have been used in the implementation in order to guarantee that the interchanged data uses a representation format which will be common in the system, and therefore information will be easier to extract.

(vi) *SMArc makes use of an inference engine.* While all the other proposals are used to retrieve data, SMArc uses semantic features to its advantage to extract information and if it is required, it will trigger an action based on it.

Table 1: Related works comparison between SMArc and the studied proposals.

Proposal	Low capability devices	Security	Distributed	Smart Grid	Semantic	Inference engine
Aura	−	−	−	−	−	−
Cooltown	−	−	−	−	−	−
SOCAM	−	−	+	−	−	−
Hydra	+	−	−	−	+	+
JEDI	−	−	+	−	−	−
Siena	−	−	+	−	−	−
Hermes	−	−	+	−	+	−
RFID Auth. Middleware	+	+	−	−	−	−
gLite	−	−	+	−	−	−
NetSolve/GridSolve	−	−	+	−	−	−
Globus	−	+	+	−	−	−
MPI-based middleware	−	−	−	−	−	−
MAgNet	+	−	+	−	−	−
SStreaMWare	+	−	+	−	+	−
USEME	+	−	+	−	+	−
SensorWeb 2.0	+	−	+	−	−	−
OASiS	+	−	+	−	−	−
B-VIS	+	−	+	−	−	−
MiSense	+	−	+	−	−	−
SOMDM	+	−	+	−	−	−
$(SI)^2$	+	−	+	−	−	−
SOA-MM	+	−	+	−	−	−
ubiSOAP	+	−	+	−	−	−
SMArc	+	+	+	+	+	+

SMArc Consists of Several Modules

Ontology Module. It will integrate any new vocabulary related to new services that may spring up as the result of adding new devices (smart meters, home loads, etc.) to the Smart Grid.

Repository Module. Its functionalities will be primarily involving semantic data storage that will be required for different tasks while the middleware architecture is put to a use.

Services Module. As it can be inferred from its name, this module deals with the services that are present in the middleware architecture. It must be scalable enough to add any new ones.

Resources Module. This module is in charge of managing all the hardware infrastructures that are required to harvest data from the context. As the available hardware may be varying in terms of numbers and features, it will also need to be flexible enough.

Inference Engine Module. Working closely with the ontology module, the inference engine will provide the semantics required to treat the information flowing through the middleware layer.

All in all, SMArc will be a semantic middleware architecture placed between the application layer and the communications layer, as depicted in Figure 4.

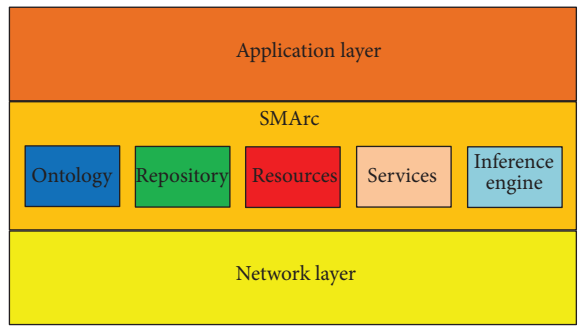

Figure 4: SMArc placement and composing blocks.

Next subsections describe the functionalities of each of the modules by including them in a computational and functional analysis.

3.1. Computational Analysis. The inner relations of the subsystems are better shown in Figure 5, where modules are already depicted as subsystems.

Each of the subsystems is modelled by several components. To begin with, the Resource subsystem is divided into three components involving Resources hardware, that is to say, the most relevant features (as CPU, RAM or ROM memory) from the devices that are used for data collection

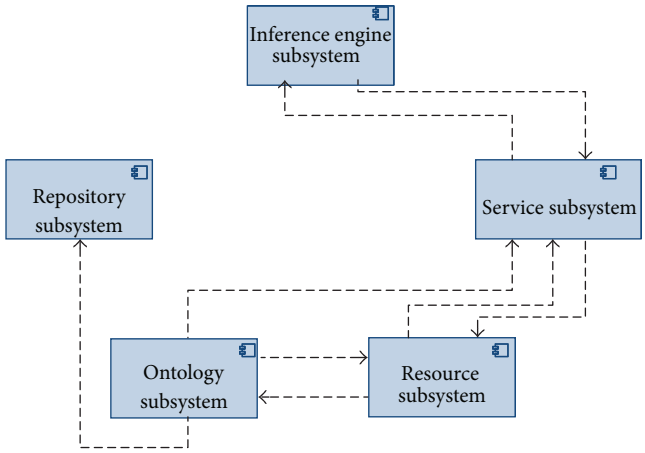

Figure 5: SMArc subsystem diagram.

from the environment, the different sensors, and actuators present in the deployment with their own characteristics (humidity or luminosity readers, LEDs or switches as actuators, etc.) and device input/output operations necessary to retrieve the information and data format used by the middleware. If the devices that this subsystem communicates with are capable of equipping security features, SMArc will establish communications using them. Resource subsystem components, and how they are interlinked, are displayed in Figure 6.

The Inference Engine subsystem displayed in Figure 7 requires components that will be critical in semantics and decision-making mechanisms. There are two of them that the system must be enabled with: a facts repository, where all the actions that are taking place within the semantic middleware architecture are stored, and a rules repository that establishes the regulations and actions to be taken in the system. There are three other components participating in this subsystem: an Action Collector that will register any relevant new actions happening in the system, an Inference Manager interweaved between the two former repositories which will work in a way that, if a match between the information provided by the facts repository and the rules repository triggers an action (e.g., measuring humidity from 9 p.m. on at the rules repository and time of the day becoming 9 p.m. at the facts one), the Inference Manager will send an order, and an Action Trigger component which will execute any action delivered from the Inference Manager.

The ontology subsystem must be taken into account at this point. There are two main functionalities that are expected from the ontology: providing the data representation that should be used and updating any change that may be taking place in the Smart Grid. Therefore, this subsystem uses two inner components: one is a Formatter, which is used to specify how data communications will look like as mentioned in the first section of this paper, while the other one is an Updater that will collect any change in the measuring capabilities of the Smart Grid, either when a new device is added or when it is lost. The way they are interrelating to each other can be observed at Figure 8.

Similarly, Services subsystem is also using repositories, although being of slightly different nature, as this subsystem will store services that can be retrieved from the system rather than rules or facts, hence the decision of naming it container instead of repository. There are two main components involved here. One is a Factory that is used for design and storage of the services that can be retrieved from the system. The other component is a Request Manager that will handle the requests done in the system, along with the responses, in terms of storage or delivery. Overall, this subsystem is heavily dependent on the Ontology subsystem in order to obtain its different functionalities, which at the same time are being provided to the Inference Engine subsystem. Service subsystem has been depicted in Figure 9.

Last but not least, Repository subsystem is way simpler than the others because its main functionalities are just related to the ontology repository, and only one component is required for information manager—OntologyIO, which is handling the update of the repository where the ontology is contained. Its overall appearance is presented in Figure 10.

3.2. Functional Analysis. After presenting the components of SMArc, the behaviour of the system will be introduced as well in a functional analysis. In order to offer the most extensive information, several diagrams are going to be presented in this section. As requirements have to be considered in the system, the first step has been portraying them in a use case diagram, as depicted in Figure 11. As far as the different stakeholders taking part in the system are concerned, there are two different kinds of entities: actors–external devices that are used for measurement and service provisioning and four use cases that are going to be satisfied: register services (used for newly discovered services), request a simple service (regular data retrieval), request a composed service (data retrieval from two different kinds of meters), and action triggering (based on the inference engine rules and facts).

In order to offer a more detailed view on how these four use cases are managed, each of them will be described by means of sequence diagrams depicted in the following sections of this paper.

3.2.1. Service Registration. If services are to be offered to the application layer, they must be registered before they can be exploited. There are some actions that are mandatory if services are going to be registered, though, as data format must be acknowledged so as to have services described with a structure that is recognizable when they are requested. The process to register a service to make it usable is presented in Figure 12.

Typically, when a new device appears in the Smart Grid, it will be willing to publish its services. As the device must be aware of how services are being stored, it will request the storage format (1) to the class able to communicate the request. This request will be sent to the class with the suitable format (2) and once the request has reached it the format will be obtained (3). Afterwards, it will be sent back to the

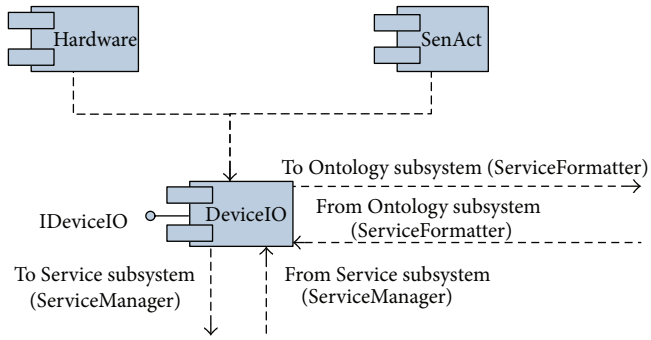

Figure 6: Resource subsystem components.

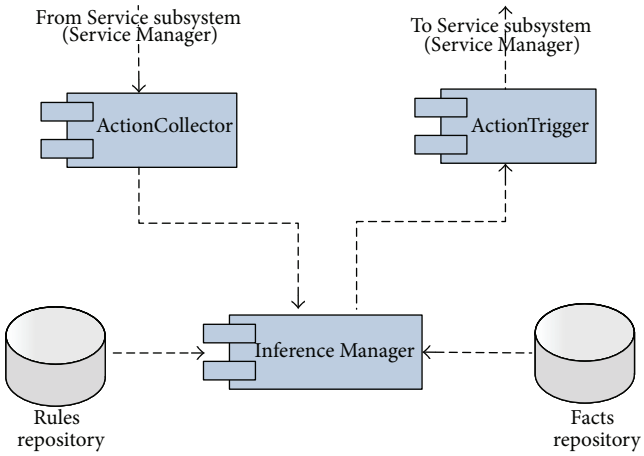

Figure 7: Inference Engine subsystem components.

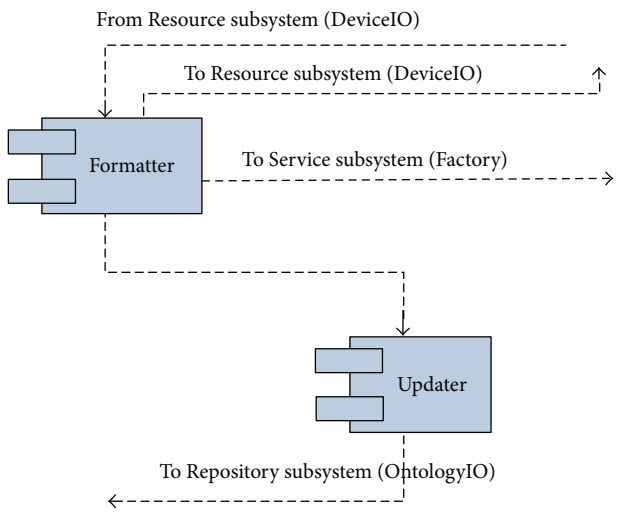

Figure 8: Ontology subsystem components.

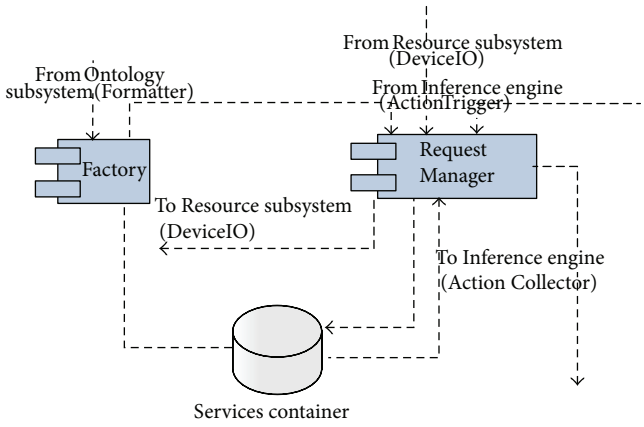

Figure 9: Services subsystem components.

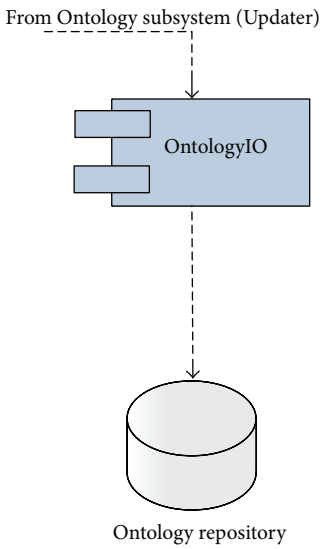

Figure 10: Repository subsystem components.

source device, jumping from one class (4) to another (5). As a result, the device will obtain the specific format that should be used to send the data to the ontology (6), which will be sent to a class that is part of the Ontology subsystem (7). As the format provided by the device is the expected one at the latter subsystem, sensing (8) and actuating (9) capabilities can be extracted and sent (10, 11) to a factory that will determine specific devices that can be invoked (12). Additionally, the

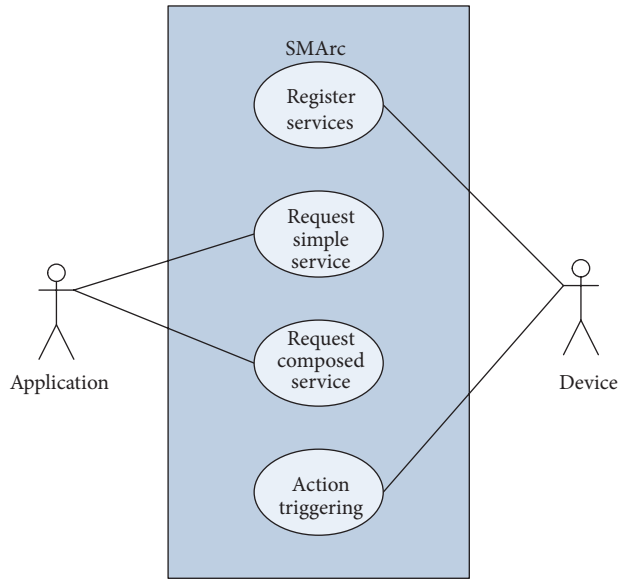

Figure 11: Use case diagram of the proposed semantic middleware architecture.

services will be stored for future updates in case they have to be changed or removed (13).

3.2.2. Simple Service Request. As it has been shown in Figure 11, there are two different kinds of services that can be retrieved: simple and composed. Firstly, it will be explained how simple services are retrieved. When a service request is made to a device, an element from the Service subsystem that will behave as the service manager will check the request in order to make sure that it has been done under the format expected by the system (1). As any request that is done may be used by the Inference Engine subsystem, it will be stored (2) and notified to the component taking care of semantic inferences (3). Once it has been checked that the request—or, at this point, the action—has no previous record (4), it will be stored (5).

While all these checking and storing operations are being made, the request will be treated as well: after sending the data request to the Inference Engine subsystem, the request will be sent to the class that is used for input and output data (6) and the one that is used for data retrieval from the device (7). Responses (8, 9) will be sent back to the service manager (10) and it will process them in a way that will be similar to the one used with requests. In this case, responses will be sent for storage (11) and notified for data inference (12). As it happened before, responses will be checked so as to find out whether they are already present or not (13) and stored if necessary (14).

Simple services are requested in a manner that is explained in the sequence diagram present in Figure 13.

3.2.3. Composed Service Request. Simple services may not be the only ones that can be obtained; in fact, it is usual to expect from a Smart Grid other services that are obtained as a result from merging the properties of one simple service with the properties of another one or even from several capabilities of the same device. For example, if an *energy daily expenses* service is wanted to be implemented, there are several pieces of data that are required from devices of very different nature (energy, time period, and electricity cost) that are unlikely to be built in the same appliance. Therefore, data from different meters must be retrieved instead of collecting it from a single meter as it was exposed in the previous case. The actions that have to be taken are shown at the sequence diagram of Figure 14.

The steps taken to successfully deliver the service are done very similarly. A request for a composed service is checked in order to be sure that it is fulfilling the required request format (1) and afterwards is sent to the Inference Engine subsystem (2).

From here, the request will be sent to the action collector that is used for semantic inferences (3) and, as it was done before, this latter element will be checked (4) and saved if it has to be done so (5). At the same time, the request is sent to the suitable class that will deal with input/output operations (6) and, right afterwards, the needed requests are done (7). What is different in this case from the former one, where simple services were requested, is that not only several services requests are needed in order to gather all the data that the composed services need to be provided with but also one more operation must be performed in order to process the obtained parameters to compose the service (8). After this processing, the response will follow a way very similar to the one followed by a simple service request. The response will be sent back to the class in charge of managing the input and output operations (9) and to the service manager (10). The received response will be checked as well (11) and sent again to the Inference Engine (12), where actions must be taken or not (13) depending on the received data. Finally, the action will be checked so as to find its meaning (14) and store it if it is appropriate (15).

3.2.4. Action Trigger. In the cases previously exposed the Inference Engine subsystem has played a reactive role, collecting requests regarded as actions and storing them in case they are new. Nevertheless, as it has been described before, actions can be triggered as the result of comparing actions that have just happened with rules that establish some other actions to be performed. Therefore, a sequence diagram has been depicted as Figure 15 so as to better describe the different stages that will be taken to successfully trigger an action in SMArc.

Unlike previous examples, where the action at the use cases was started by a user as part of the Smart Grid (and, by proxy, a member of the Smart City), whenever an action must be triggered according to the set of rules defined at the Inference Engine, the starting point will be assumed by SMArc and the element that is responsible for action inferred in particular (1). This element will send a message to the already mentioned service manager requesting that a particular device takes some action (2). Afterwards, since what is being done is requesting a service, the actions taken are similar to the ones explained in Sections 3.2.2 and 3.2.3

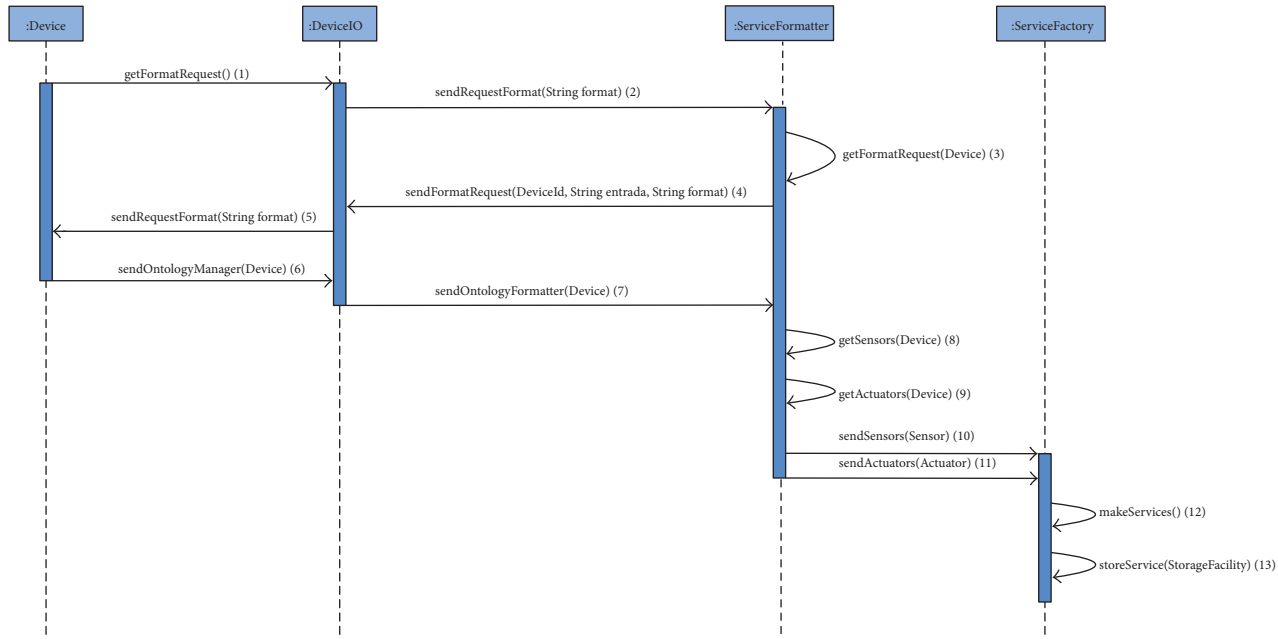

Figure 12: Service registry sequence diagram.

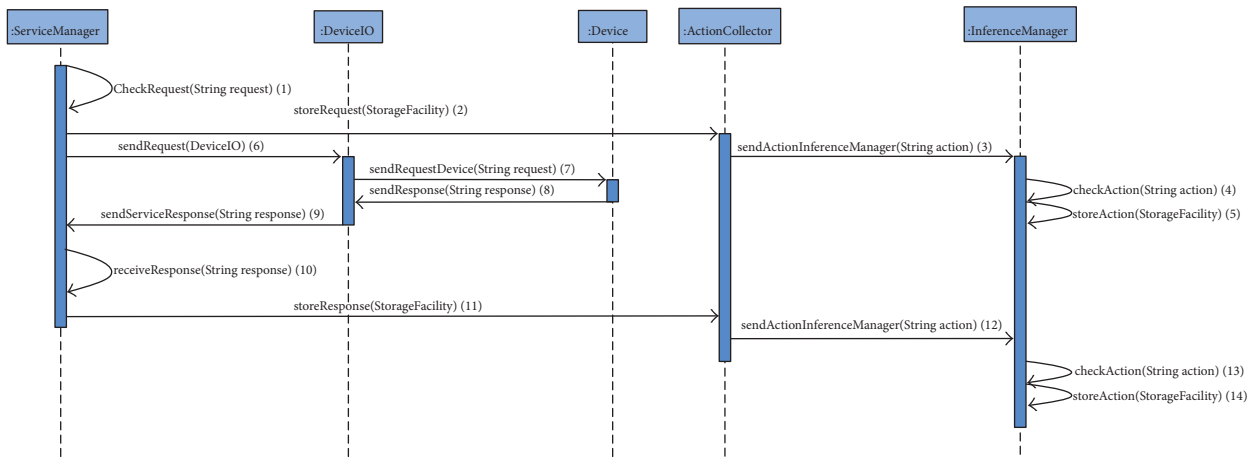

Figure 13: Simple service request/response sequence diagram.

(although those related to the inference manager actions will not take place here). The action request will be checked in search for any sort of failure (3) and will be sent for its storage (4); the request is stored as usual because it is considered that it may come in handy for future services related to historical data. Thus, the action collector sends back the request as an action to the inference manager (5) that will check (6) and store the action (7) if it is a new triggered one.

Meanwhile, the data request is satisfied as it is usually done: the request will be sent to the device class that is controlling input and output data (8) and the query is sent again to the device class bound to the device (9) which will send back the response (10) until it reaches the service manager (11) where it will be checked again (12). Another action is sent from the action collector as the response is obtained (13), checked (14), and stored (15) as it was in former cases. The service response obtained will be finally saved (16). Note that in this case a simple service has been considered as the triggered action. Although composed services would be triggered alike, an extra stage to merge all the obtained data would be needed.

The previous sequence diagrams depict how the different requirements that have been established can be solved. However, further data matching the information interchange shown is required so as to have a better grasp on each of the use cases. Therefore, class diagrams are offered as part of this subsection. These class diagrams are divided into the different components of the subsystems that have been shown before. To begin with, class diagram regarding Resource subsystem is depicted as a part of Figure 16. It is the most complex

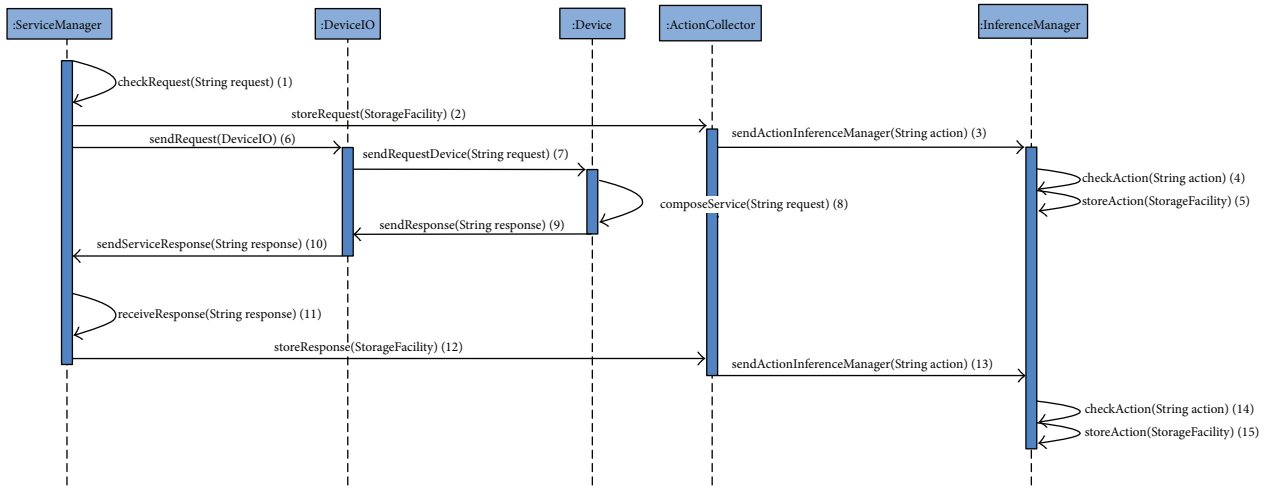

Figure 14: Composed service request/response sequence.

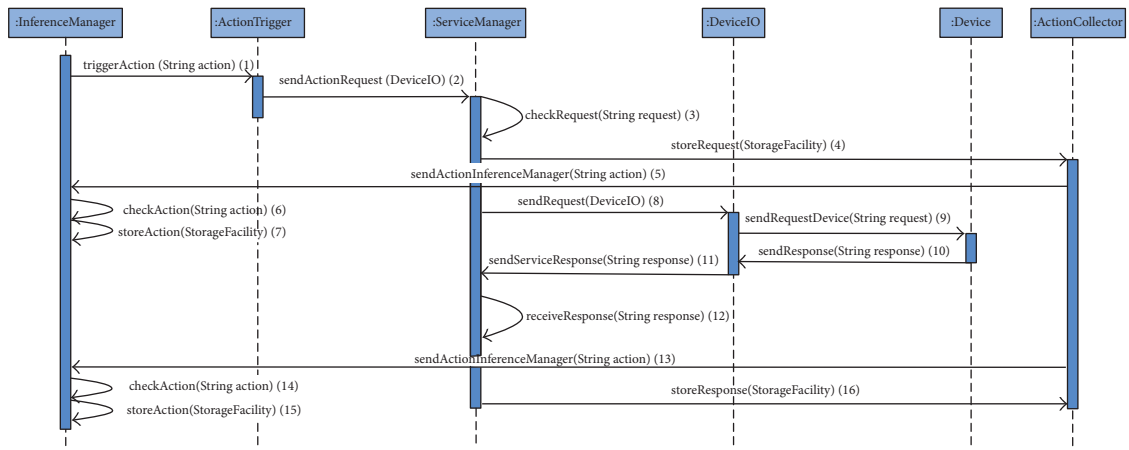

Figure 15: Action trigger sequence diagram.

one, as it has to deal with hardware components that may be very different ones from the others. As the subsystem it is representing, the class diagram also shows how to tackle the input and output operations among mounted devices and their hardware capabilities. All the methods and operations that are executed against a device that is not part of the middleware but an external entity will be done adding security functionalities; for example, if the device asking for the data format is using security mechanisms (data encryption is the most likely one) and/or is able to deal with encrypted data, then the format request will be deencrypted when it enters SMArc—by means of the getFormatRequest () method—and reencrypted whenever it leaves the semantic middleware architecture—once the sendResponse (String) method is used. Figure 16 is also showing the class diagram used for the Ontology subsystem; as its main functionalities are adapting the information flow and updating the ontology contents, a class has been created for each of the functionalities: on the one hand, ServiceFormatter is containing methods that will be used for data formatting, as getFormatRequest (Device), or capabilities transfer—sendSensors(Sensors) and sendActuators(Actuators). On the other hand, an Updater class is present in order to adapt the ontology to any new element of the Smart Grid.

Figure 17, on the other hand, is illustrating class diagrams from three different subsystems: the Inference Engine subsystem, the Service subsystem, and the Repository one. The class diagram which involves Inference Engine has been designed with all their expected attributes and operations. An ActionCollecter class has been enabled to collect all the actions (by actions it is meant both requests and responses) and an ActionTrigger class to execute them. The execution of new actions will be conditioned by the decision-making processes at the InferenceManager class, based on the feedback that is received from the collected actions and the rules that the system must be compliant with, so that actions will be checked (a checkAction (String) method has been created to treat that task) for any foreseeable event.

Service subsystem classes are dependent on the difference between simple and composed services that have been

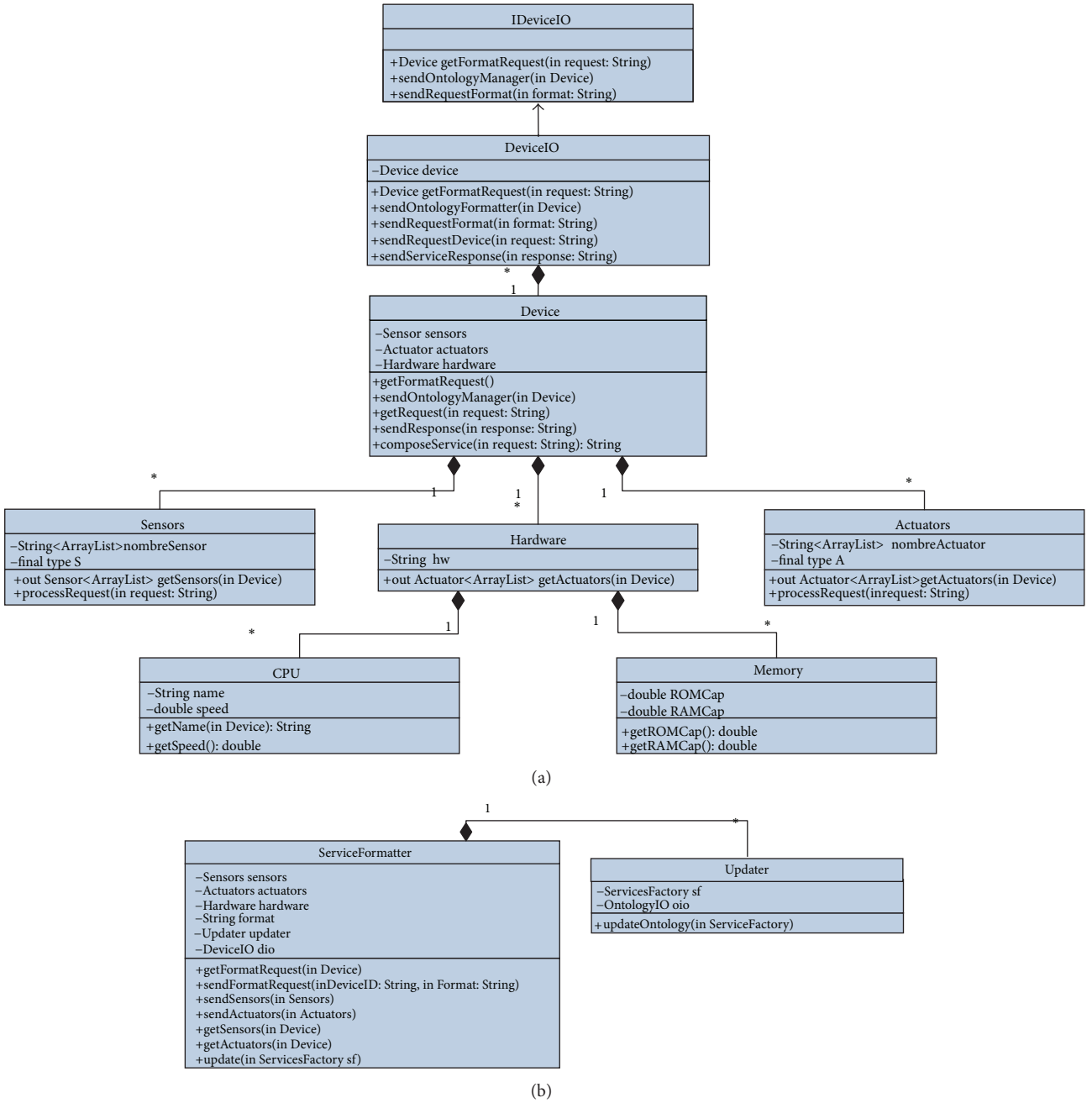

Figure 16: Resource (a) and Ontology (b) subsystems class diagrams.

established. Consequently, there are two very similar classes used to assemble each kind of service (SimpleServicesFactory and ComposedServicesFactory) with the only difference that composed services cannot be retrieved from the environment or a device but are composed within SMArc (they are obtained by gathering simple services data from different Smart grid elements), so no method prepared for composed service retrieval has been conceived. A ServiceManager is also used for information check, transfer, and storage. Finally, the Repository subsystem will take care of updating the contents of the database kept for the ontology repository, obviously taking into account the received information from the Ontology subsystem.

4. Performance Tests on SMArc

In order to have a certain idea of the performance of this proposal, it has been tested to retrieve values dealing with the services that it is expected to fulfill. Some of the classes and methods introduced in the previous description have been left with a very small implementation, since this deployment is a specification used when low capability devices are

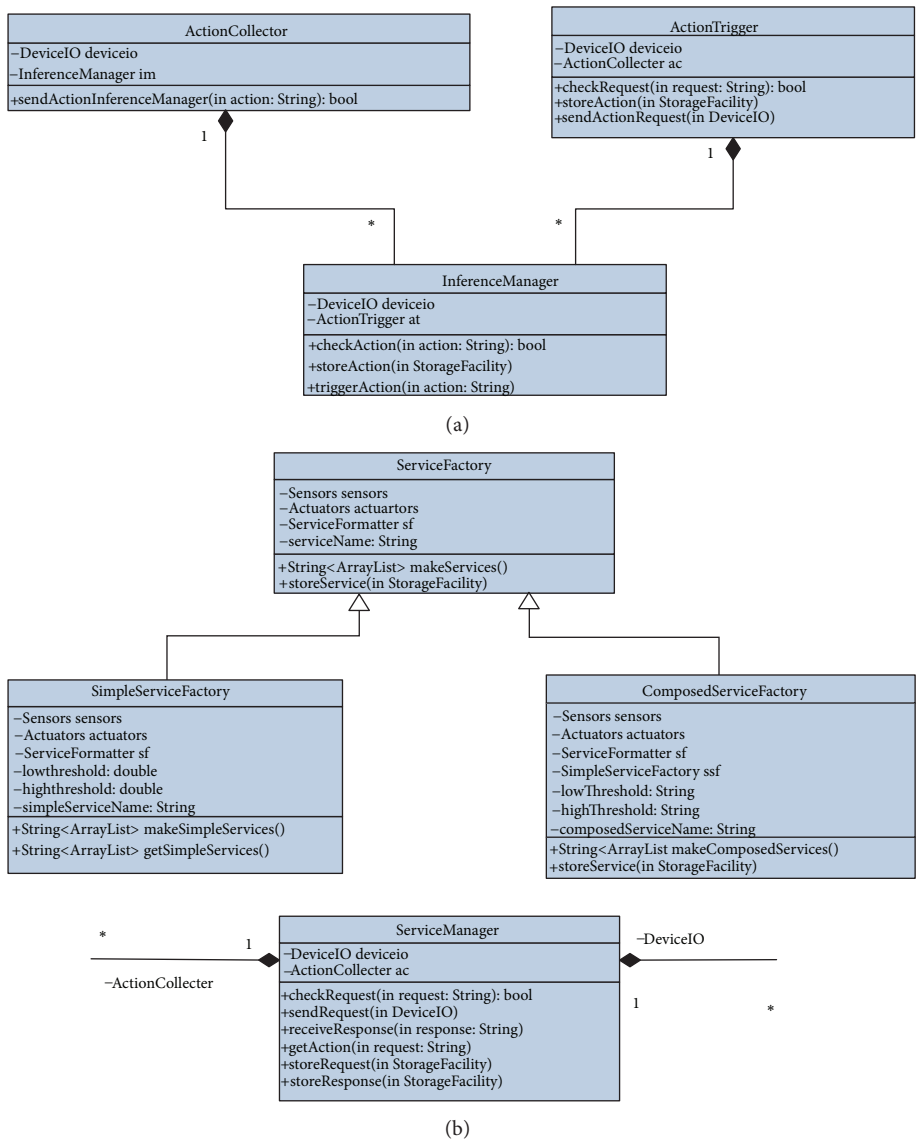

Figure 17: Inference Engine (a), Repository (b), and Service (c) subsystems class.

employed. The idea of the proposal is that it will be able to be escalated to many other devices apart from the ones that are being used for this particular implementation, thus providing a ground specified enough to be used as an advanced template but open enough to implement inner methods depending on the capabilities of the system.

4.1. Implementation Description. The implementation and deployment have been made by using SunSPOT motes. There are several reasons for using them in this context: as the Smart Grid deployed in a Smart City will have to collect data from places where wired equipment would be quite a nuisance (power stations, local or central controllers, home infrastructures, etc.) it looks like a more suitable option to use wireless electronic devices. Thus, data is likely to be gathered from a Wireless Sensor Network, and, in this situation, equipment able to communicate via IEEE 802.15.4 standard, which is the most usual for WSNs nodes, will get the upper hand under many circumstances. Additionally, SunSPOT motes offer good enough capabilities for a node; the latest distribution is equipped with 8 Mb of Flash memory and 1 Mb of RAM [22]. Although they look like very limited parameters, they are more than enough to act as sensor-enabled devices that will obtain useful information from the context for consumers and staff. In this way, SunSPOT motes ensure that data collection or storage will not become an issue. SunSPOT motes can be used as regular nodes placed anywhere, sending and receiving information that will become accessible via a base station plugged to any electronic appliance equipped with a USB connection. Their appearance is displayed in Figure 18.

For practical purposes, storage has been made on separated files that are kept in a PC running an Ubuntu

Figure 18: SunSPOT regular mote, without its sunroof (left) and SunSPOT base station (right).

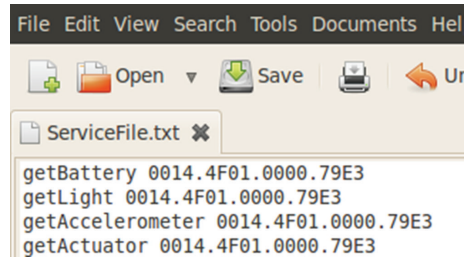

Figure 19: Service storage appearance.

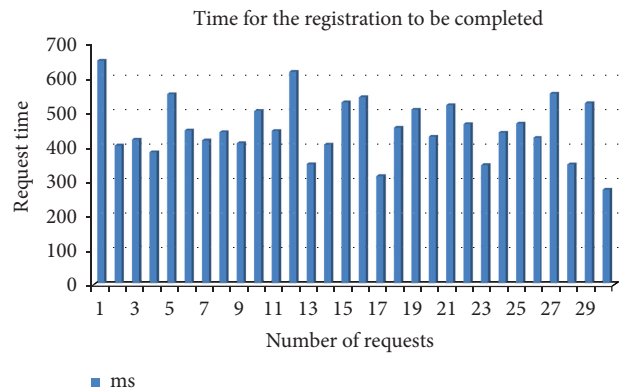

Figure 20: Service registration performance.

Table 2: Prominent figures of service registration.

Datum	Figure (ms)
Average	453.4
Median	444
Minimum	274
Maximum	651

distribution; apart from drastically shrinking the costs and complexity of the system, it lets the Sun SPOT base station access the files without any other requirements with regard to WSNs. It has to be noted that, while utilities are implemented in J2ME at the nodes, the base station will use J2SE for any application, so it will be able to be executed from a device even if it has no J2ME versions installed. Five different files have been used: one for service storage, another one for ontology storage (based on OWL), another one to keep the data format that would be used (based on XML), and two more related to functionalities of the Inference Engine (a Rules file and a Facts file). Since communications between the different elements of the system that has been deployed will depend on data interchange among IEEE 802.15.4 interfaces, the results obtained will be strongly influenced by the performance capabilities of the IEEE standard.

4.2. Service Registration. The first test was done so as to measure how long it takes the services present in the different nodes of the Wireless Sensor Network to be registered at the PC that is communicating with the WSN by means of the SunSPOT base station. The information regarding the system was saved according to a specific format: the name of the service was kept at the left side of a row that on the right side would contain the MAC address of the mote that was providing it (get ⟨Servicename⟩⟨Mote MAC address⟩). It must be born in mind that although the services may be namely similar, the data that they are collecting is not the same (as services are deployed in different nodes), so the device where they are deployed must be learnt as well. The appearance of the saved services has been shown in Figure 19.

In order to reliably test this feature, 30 attempts at establishing a connection between the motes and the base station were made. The results can be seen from the graph of Figure 20.

Along with the data presented in the graph, another table with the most prominent values obtained has been depicted as Table 2.

Although the average time is almost negligible for a human being, there is a plethora of different, heterogeneous values (minimum value is less than half the maximum), despite the fact that this case is the most homogeneous of the tests provided. Those differences may be due to the processing of the XML-like format used to process the data, which was done following both the appearance shown in Figure 3 and the procedure of the sequence diagram presented in Figure 12. In any case, that heterogeneity is not that distorting in these tests, as the median and the average values are very close one to the other.

4.3. Simple Service Request. As implied by the former sections of this paper, simple services are obtained from a single sensor of a single device. The results of the tests are presented in Figure 21.

Here, it was decided to request light readings from the SunSPOT mote, as it is a value that is usually involved in energy consumption on the Smart Grid (the more the sunlight there is in a room in a flat, a factory, or an office, the less the electricity is consumed). However, there is very little difference in terms of performance if any other parameters from the mote are measured, as they are all obtained from the same sensor board that is installed in all the motes that

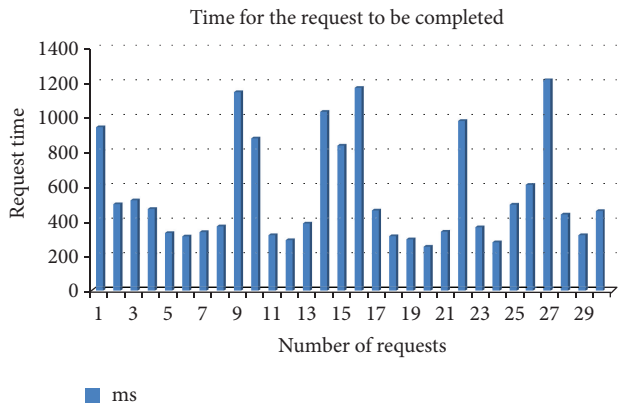

Figure 21: Simple service request (light) performance.

Table 3: Prominent figures of simple service performance.

Datum	Figure (ms)
Average	561.2
Median	454
Minimum	257
Maximum	1227

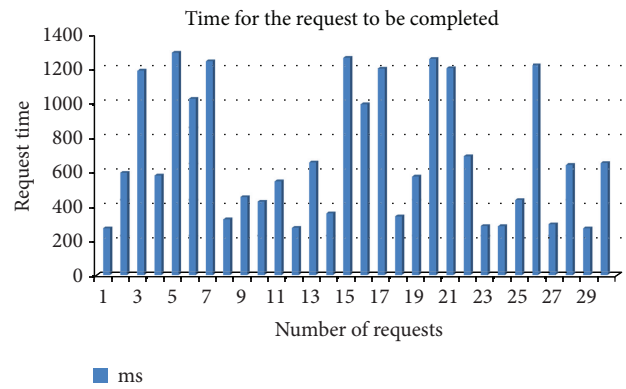

Figure 22: Composed service request performance.

Table 4: Prominent figures of composed service performance.

Datum	Figure (ms)
Average	661.1
Median	558
Minimum	259
Maximum	1230

are nodes of the Wireless Sensor Network collecting data for the Smart Grid. As far as the SunSPOT motes are concerned, they are all called by using the same interface `EDemoBoard`.

A simple glance at the graph can tell that the obtained values are way more differing in this test. Nevertheless, Table 3, containing the most prominent of them, has been added as well.

In the light of the obtained figures, it can be told that results show a higher disparity than before; despite having a median almost equal as when registering the services from a mote (444 milliseconds in the former case and 454 milliseconds here), the average value is different and greater than before; clearly, higher values obtained from the tests are distorting the overall impression on the service request. These differences with the other cases can be explained by bearing in mind that here, although the procedure to be followed (described in the sequence diagram of Figure 13) has almost the same number of steps than before, those steps are dependent on IEEE 802.15.4 communications rather than local data processing, as when storing information, so data transfers may pose challenges in terms of transmission that were simply not present before.

4.4. Composed Service Request. This request performs in a very similar way as the one described in the former section, but it will demand two simple services to be delivered. It has been chosen to create a "comfort service" that requires temperature and luminosity parameters; typically, the value of the parameters will be evaluated jointly so as to determine the overall level of comfort in a room. Should any of those two parameters be too low or too high, according to lower and higher fixed thresholds, then a value will be given at the piece of equipment the base station is plugged to, pointing out that comfort level is too low due to worse parameter. If the values are within the limits defined by the thresholds for each of them but are *close* to one of the thresholds (that closeness has been defined as a value standing inside a buffer zone that has as higher/lower limit as the higher/lower thresholds), then the given result will be Medium Comfort. Any other information will result in a High Comfort level. The graph showing the results from the 30 attempts used to test this characteristic has been displayed in Figure 22.

As done before, Table 4 is showing the most important figures of the performed tests.

In this case, it is notorious to mention that while the minimum and maximum values for the measured parameters are almost the same as the ones obtained before, the average and median values are greater. This implies that the communication issues present before procedures are present here in an almost equal manner; since the simple services are being obtained from the same mote (as it seems reasonable to believe that comfort will be evaluated in the same room, there is little point in having different motes sensing the same data from a room), the communication interchanges are done the same way than before. If something, the board containing the sensors of the mote must be requested two times instead of one to obtain the required data, and some more time will be spent on processing the retrieved information into a message that must make measurable comfort level understandable in a natural language.

4.5. Comparison with Other Proposals. Other proposals presented here as related works have been tested as well with differing results. Depending on the objectives of the authors, tests were made so as to check different kinds of parameters that may or may not match the ones made on our proposal. For example, simulator tests that run on event-based Hermes

middleware have not a clear correspondence with the registry and services provided by our proposal [23]. The most notorious time-based measured parameter is routing efficiency that takes into account latency between different elements of a network. In this case, it is shown that an increasing number of subscribers will affect the performance of the routing process by requiring more time to finish the task. In all situations Hermes proves to require a way higher average time than SMArc in registering a service or tackling a service request (the best latency possible is around 2 s, while SMArc typical time required is around half a second).

Syta et al.'s proposal on a RFID authentication middleware for mobile devices has been tested with a fully functional prototype. A continuous authentication implementation was tried using several different encryption mechanisms. The results obtained with RFID authentication middleware are around the ones achieved with SMArc, albeit requiring longer times in the former than in the latter (around 600 milliseconds for a 10 Kb file transmission).

ubiSOAP performance is evaluated by the authors of the proposal too [24]. An echo web service was tested under different network configurations. In all the tests performed the obtained time samples were always higher than the average time required for SMArc (again, if the best case scenario is considered, a local transmission will require slightly less than one second).

From all these results, it can be inferred that SMArc has proven to be competitive enough when compared to other architectures. However, it should be noted that the main purpose of SMArc is creating a semantic middleware that is adding several concepts that were not present before in a single architecture. What is more, since SMArc is flexible enough to be implemented in other different devices, with very different hardware capabilities, the results may differ greatly from one implementation to another.

5. Conclusions and Future Works

A proposal for a semantic middleware architecture has been put forward in this paper that has been named SMArc (semantic middleware architecture). It can be successfully encased as a layer part of a Smart Grid, already proven to be an element of major importance for future developments as part of the Smart City. A comparison of the different, most relevant middleware architectures has been added as well in order to both have a survey on the availability of middleware options and point out the originality and particularities that have been used to design SMArc. What is more, computational and functional analyses have been provided too, so as to illustrate the behaviour of the system and what it is made of and have a clear idea of what can be expected from it. Our proposal is offering a series of advantages unseen in any other developments, at least in a single one: as far as the Smart Grid and the Smart City are concerned, SMArc has been designed bearing it in mind from scratch, so it does not have to be ported or adapted from other contexts. Unlike most of the other architectures, semantics is incorporated in the design, thus guaranteeing that scalability and interoperability are going to be easy and optimized for future device addition. Finally, an inference engine has also been added as the mechanism that is going to be used for decision making and action triggering, which are features that are not frequently found in middleware architectures for low capability and distributed systems. The overall design is a more suitable option than any other studied solutions for the scope of the Smart Grid, as it makes several contributions to the state of the art: some of them were already present in other architectures but in a rather scattered fashion—usage of low capability devices, security features, distributed system performance, and Smart Grid specific design—while some others can be considered quite a novelty—semantics, inference engines.

There are some future enhancements that can be considered in this presented proposal. Further upgrading can be done if information is expected to be shown in a device capable of displaying a Graphical User Interface; further classes can be designed for its implementation in case a Java Swing or an Android GUI is required. Plus, a study on the optimization of the code that has been developed can be made too, in order to obtain a better performance in ulterior developments.

Acknowledgments

This paper has been done as part of the research efforts made at the GRyS (Grupo de Redes y Servicios de próxima generación, Next Generation Networks and Services Group) Research Group belonging to the Technical University of Madrid (UPM). Alexandra Cuerva must be mentioned as a researcher consulted for the design of this proposal. Additionally, this paper is part of the work being currently made in the I3RES (ICT-based intelligent management of integrated RES for the Smart Grid optimal operation) Research Project, an FP7 initiative (reference no. 318184) that targets the seamless integration of renewable energy sources and development of management tools for the Smart Grid [25].

References

[1] S. Kehua, L. Jie, and F. Hongbo, "Smart city and the applications," in *Proceedings of the International Conference on Electronics, Communications and Control (ICECC '11)*, pp. 1028–1031, September 2011.

[2] J. Zhu, Y. Feng, and B. Liu, "PASS: a parking-lot-assisted carpool method over vehicular ad hoc networks," *International Journal of Distributed Sensor Networks*, vol. 2013, Article ID 491756, 9 pages, 2013.

[3] C. W. Gellings, *The Smart Grid: Enabling Energy Efficiency and Demand Response*, The Fairmont Press, 2009.

[4] S. Bohn, J. Dargay, D. Gately et al., "An ICT architecture for managed charging of electric vehicles in smart grid environments," *Journal of Engineering*, vol. 2013, Article ID 989421, 11 pages, 2013.

[5] M. Erol-Kantarci and H. T. Mouftah, "The impact of smart grid residential energy management schemes on the carbon footprint of the household electricity consumption," in *Proceedings*

of the 4th IEEE Electrical Power and Energy Conference (EPEC '10), IEEE, August 2010.

[6] Q. Li and M. Zhou, "The state of the art in middleware," in *Proceedings of the International Forum on Information Technology and Applications (IFITA '10)*, pp. 83–85, July 2010.

[7] S. K. Meesala, "Parallel processing implementation on clusters of terminals using Java RMI," in *Proceedings of the International Conference on Computing, Communication and Applications (ICCCA '12)*, February 2012.

[8] S. Artemio, B. Leonardo, M. J. Carlos et al., "Evaluation of CORBA and Web services in distributed applications," in *Proceedings of the 22nd International Conference on Electrical Communications and Computers (CONIELECOMP '12)*, February 2012.

[9] W. Yufei, L. Weimin, and Z. Tao, "Study on security of wireless sensor networks in smart grid," in *Proceedings of the International Conference on Power System Technology (POWERCON '10)*, October 2010.

[10] A. O. Bicen, O. B. Akan, and V. C. Gungor, "Spectrum-aware and cognitive sensor networks for smart grid applications," *IEEE Communications Magazine*, vol. 50, no. 5, pp. 158–165, 2012.

[11] K. M. Albarrak and E. H. Sibley, "A survey of methods that transform data models into Ontology models," in *Proceedings of the 12th IEEE International Conference on Information Reuse and Integration (IRI '11)*, pp. 58–65, August 2011.

[12] A. K. Dey, "Understanding and using context," *Personal and Ubiquitous Computing*, vol. 5, no. 1, pp. 4–7, 2001.

[13] A. Karam and N. Mohamed, "Middleware for mobile social networks: a survey," in *Proceedings of the 45th Hawaii International Conference on System Science (HICSS '12)*, 2012.

[14] A. Saeed and T. Waheed, "An extensive survey of context-aware middleware architectures," in *Proceedings of the IEEE International Conference on Electro/Information Technology (EIT '10)*, May 2010.

[15] M. Knappmeyer, S. Liaquat, E. Reetz et al., "Survey of context provisioning middleware," *IEEE Communications Surveys and Tutorials*, vol. 15, no. 3, pp. 1492–1519, 2013.

[16] T. Wang, W. Xu, J. He, R. Chen, and W. Gu, "A brief survey of event-based middleware," in *Proceedings of the 2nd International Conference on Computer Engineering and Technology (ICCET '10)*, pp. V1461–V1464, April 2010.

[17] E. Syta, S. Kurkovsky, and B. Casano, "RFID-based authentication middleware for mobile devices," in *Proceedings of the 43rd Annual Hawaii International Conference on System Sciences (HICSS '10)*, January 2010.

[18] F. Xhafa, S. Pllana, and L. Barolli, "Grid and P2P middleware for scientific computing systems," in *Proceedings of the 4th International Conference on Complex, Intelligent and Software Intensive Systems (CISIS '10)*, pp. 409–414, February 2010.

[19] M. Sain, P. Kumar, Y. Lee, and H. J. Lee, "A survey of middleware and security approaches for Wireless Sensor Networks," in *Proceedings of the 6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT '11)*, 2011.

[20] W. Xin-Sheng, Z. Yong-Zhao, and W. Liang-Min, "Load-balanced secure routing protocol for wireless sensor networks," *International Journal of Distributed Sensor Networks*, vol. 2013, Article ID 596352, 13 pages, 2013.

[21] N. Mohamed and J. Al-Jaroodi, "Service-oriented middleware approaches for wireless sensor networks," in *Proceedings of the 44th Hawaii International Conference on System Sciences (HICSS '11)*, January 2011.

[22] Sun SPOT hardware specifications, http://www.sunspotworld.com/products/.

[23] P. R. Pietzuch "Hermes: a scalable event-based middleware," Tech. Rep., 2004, http://citeseerx.ist.psu.edu/view-doc/summary?doi=10.1.1.129.9934.

[24] M. Caporuscio, P. Raverdy, and V. Issarny, "UbiSOAP: a service-oriented middleware for ubiquitous networking," *IEEE Transactions on Services Computing*, vol. 5, no. 1, pp. 86–98, 2012.

[25] "ICT-Based Intelligent Management of Integrated RES for the Smart Grid Optimal Operation," 2013, http://cordis.europa.eu/projects/rcn/106338_en.html.