Chapter 6

The Role of Middleware in Distributed Energy Systems Integrated in the Smart Grid

Jesús Rodríguez-Molina

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/62870

Abstract

Middleware architectures have proven to be of major importance when dealing with distributed systems, as they are able to abstract the inevitable heterogeneity of the hardware devices present in a deployment with the aim of offering a collection of interfaces and resources of homogeneous appearance to the upper, application-oriented layers. In an energy-based distributed system as the Smart Grid, this role is replicated, as the hardware devices that are found, while essentially related to the power grid and the functionalities that can be extracted from it (advanced metering infrastructure, remote terminal units, renewable energy resources, etc.), still present the same challenges that other distributed systems are expected to deal with, such as heterogeneous features, different information formats, diversity of their performance procedures, or integration and interconnectivity issues. Therefore, a middleware architecture is still of major usability in the power grid. This chapter offers information about the common features that are present in a middleware architecture that works under the requirements and use cases typical of the Smart Grid, as well as offers examples on how middleware integrates legacy, proprietary, and newly developed pieces of equipment within the same distributed energy grid.

Keywords: middleware, distributed systems, software architecture, data management, service integration

1. Introduction

Interconnecting hardware devices of different features has been a challenge faced repeated times in different areas of knowledge. It started as a task to be done at the network layer, where different proprietary systems struggled to communicate with each other when equipment from differ-

ent manufacturers had to cooperate in retrieving and exchanging information. Some of the most prominent organizations regarding standardization (ISO, CCITT) started working back in the late 1970s [1] in a layered solution that would allow interoperability among different pieces of equipment while (open system interconnection (OSI) model), around the same time, another layered architecture would impose its own criteria in order to solve interconnectivity challenges (transmission control protocol/Internet protocol (TCP/IP) model). While those solutions—especially the latter one, which became widely adopted in the following years—offered reasonably accurate patterns on what could be obtained from each of the layers (implementation details were left to each of the manufacturers, which only had to guarantee that their products would be compliant with the specifications described in each of the layers), few holistic implementations were offered upper than the network layer. This fact implied that despite the network and lower layers had standardized protocols that would accomplish the expected functionalities for connection-oriented and connectionless communications, solving the challenges associated with the different formats for data representation in upper levels (as session or presentation in the OSI model, and most of the application layer in TCP/IP) would still be a major challenge, as increasingly differing devices were being used in distributed systems. As far as the Smart Grid is concerned, each of the microgrids that are involved in the larger deployment might count with appliances that have little to do with the ones that are found in another one. For example, advanced metering infrastructure may provide different services and be developed by a different company in one microgrid if compared to another one; even the same one is likely to have pieces of equipment from different vendors. Advanced end users or developers with enough know-how are willing to create their own smart meters by integrating hardware and software solutions reliant on open source technologies to provide services, thus adding a degree on unpredictability in the information formats that are used throughout the Smart Grid. If some other devices and/or resources are taken into account (RTUs, DERs, RESs, etc.), the differences among the infrastructures and hardware components grow exponentially. Clearly, a solution that goes beyond purely network-based communications, which are oblivious to the meaning of the information that is interchanged (only network datagrams are considered, thus leaving the data unprocessed and without knowledge inference), is required for upper layers.

Fortunately, middleware can be used to solve these issues. Middleware can be defined as a software-based layer located between the lower, network- and hardware-based layers and the higher, application-based ones, which has as its main functionality the abstraction of the underlying heterogeneity associated with the different hardware devices participating in a distributed deployment, in a way that a collection of homogeneous-looking interfaces (more specifically, a set of application programming interfaces or APIs) will be offered to the end users of the application layer. The main functionality of middleware becomes evident in **Figure 1**: despite the different data formats used by the devices located in different levels, the middleware layer still homogenizes the collected data from lower ones, withholding the underlying heterogeneity of the whole system.

When all is said and done, middleware architectures are used in heterogeneous, noncentralized systems where data collection is scattered throughout a collection of different pieces of

http://dx.doi.org/10.5772/62870

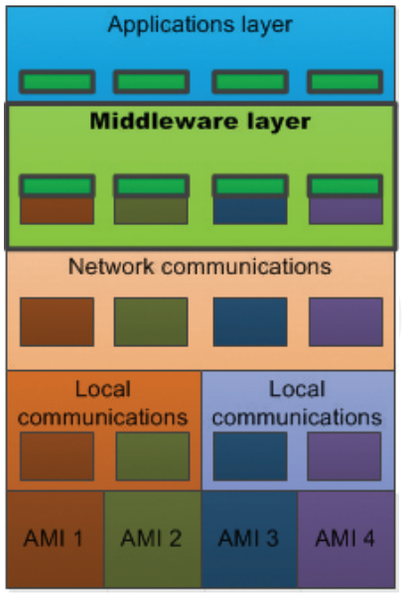

Figure 1. Generic middleware architecture for the Smart Grid.

equipment responsible for harvesting data, transferring them and either taking decisions or executing commands made by human beings. When considering how energy production, transport, and consumption can be managed, it becomes clear that if it is done in a distributed manner and using the infrastructure of a smart grid, middleware can be very convenient for a plethora of tasks:

1. A middleware architecture in a distributed system effectively acts as the "glue" that puts together many different hardware and software entities. It processes the data in a way that it will be offered to users and applications (human operators, mobile apps, etc.) oblivious to the underlying complexity and different appliances used in the distributed energy generation systems.

2. A middleware architecture can be used to enhance the available services developed for the system. If, for example, load balance, context awareness, real-time pricing, security, semantic capabilities or data aggregation functionalities are desired, they can be added as part of the middleware architecture in case they cannot be installed either as part of the hardware or the applications. This is a situation that happens more often than not, as applications may be too lightweight to be able to perform calculations expected from different features (statistic information, big data management) and hardware devices might be either not powerful enough to perform those calculations, or may be proprietary and/or legacy devices that cannot be reprogrammed, or else they would be regarded as "hacked" by their vendors.

3. A middleware architecture allows the creation or enhancement of business models, as it can be used under different kinds of service paradigms, such as Software-as-a-Service or Platform-as-a-Service. In addition to that, application developers can have a very easy time connecting to the middleware architecture their own pieces of work, as they only need to care about the upper level interfaces that the middleware architecture offers them.

4. Prosumers can manage the amount of energy in a more flexible manner, as they are able to use the services that are in the middleware to decide how much or when to pour their produced electricity into the power grid, they can coordinate with other users to offer a better price for the energy, etc.

There are several parameters that must be considered when a middleware architecture is to be developed for a system:

a. How transferred information is going to be homogenized: the data transferred by different devices located at the last mile of the distributed system may be made by different manufacturers that use their own proprietary data format. Due to that, it becomes even more evident that middleware is useful to solve this kind of challenges. However, it has to be figured out the way that data formats will be the same for all the existing devices. One of the ways that can be used to solve that issue is by using semantic resources as common information model or CIM [2]. CIM is a standard somewhat resembling unified modeling language that can be used as a way to design software in a smart grid-based system. In addition to that, CIM can be extended by additions done in a microgrid, so they will be added to the overall available libraries. An architecture based on CIM relies heavily on semantics, which can also be used as a way to create or extend existing ontologies that will not only expand the collection of terms and associations among those terms, but also add more capabilities to the middleware using them, as the capacity of inferring knowledge from information or even raw data will be of major importance for taking decisions.

b. Functional and nonfunctional requirements that are to be born in mind when designing the middleware architecture: functional requirements will be turned into use cases that represent the functionalities expected from a distributed, energy-based system. These requirements are conceived to improve the existing state of the art at micro- and macro-services level: on the one hand, already present services must be improved in terms of efficiency (doing the same task faster) and complexity (more complicated tasks can be performed as well). On the other hand, new services can be included in the middleware architecture that can be based on purely software capabilities (context awareness, semantic capabilities, publish/subscribe paradigm, security) or more involved in distributed energy generation (load balance, demand side management, tariff calculation).

c. Present and already functioning models must be studied as well. One of the most widely known is the smart grid architecture model or SGAM [3], where different layers (component, communication, information, function, business) are responsible for different attributes that model a whole Smart Grid system with a holistic point of view. While there is a considerable room for improvement and innovation, this and more models and proposals can be used as a starting point for further studies and developments.

Overall, research activities done to develop the middleware architecture must consider the current state of the art to have a good grasp of the status of the existing solutions (especially regarding the open issues and challenges that remain to be dealt with), improving it with functional and nonfunctional requirements, either being inferred from the state of the art or motivated by a research project, and obtaining use cases from those requirements, which will

http://dx.doi.org/10.5772/62870

result in software modules that satisfy them. Special care has also to be taken to how information will become standardized in the system, as well as the possible constrains that are introduced by nonfunctional requirements. The resulting output will be a middleware architecture that will be compliant with previously formulated objectives: it will be distributed, enhanced with several software modules and capable to manage energy generation and distribution. Typically, it will result in a layered architecture that will handle hardware and application information at the lower and higher borders, whereas there will be a core area with all the services required for data management and distributed energy functionalities.

This chapter is devoted to the definition of middleware architectures for distributed systems and how they can be used in an environment related to distributed generation of energy, such as the smart grid. The main features of middleware will be displayed, as well as a methodology to do research on this topic, and a collection of services design that can be used as a templated for future implementation works.

2. Methodology for middleware design in the Smart Grid

When considering the requirements to build a middleware architecture focused on the functionalities to be made use of in the Smart Grid, there are some facts that must be observed. Since research activities should be used here as a tool to build a commercial, usable solution that will effectively improve the already existing power grid, three lines of work will be followed, that is to say, the state of the art regarding existing developments, functional and nonfunctional requirements and the homogenization procedures that will be carried out for the main functionalities of the middleware. Following each of these, three lines will have different impacts on the methodology, as each of them deals with a different side of the

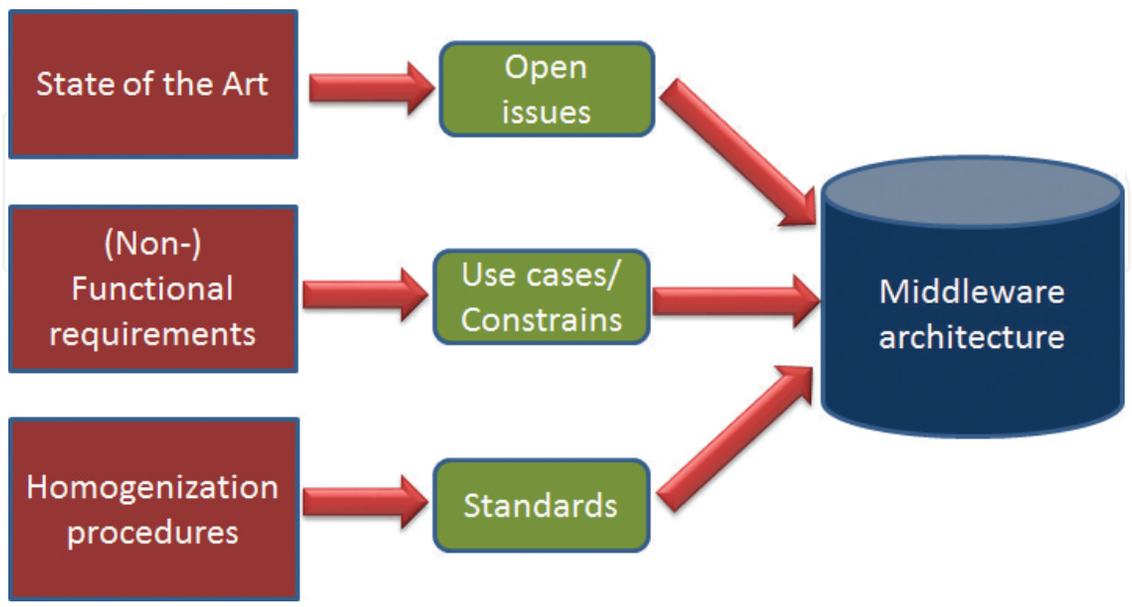

Figure 2. Middleware location and main functionality.

challenges for designing the architecture. For example, the study on the state of the art will result in acquiring knowledge not only regarding the existing solutions, but also their most usual strengths and weaknesses. Similarly, functional and nonfunctional requirements will imply the elaboration of a plethora of use cases that will result in a sequence of actions that will have their usability established by the nonfunctional requirements defining the system (more specifically, nonfunctional requirements will describe how the middleware architecture present in the Smart Grid will be and functional requirements will depict what that middleware architecture is capable of doing). Finally, homogenization solutions define what kind of data models are going to be used for information representation once the processed data are handled to the upper layers of the deployment. The synergy of these three lines of research will result in a middleware architecture suitable for the Smart Grid, as depicted in **Figure 2**.

The different software components and features will be a matter to be described in the next chapter, once the procedures to design a middleware architecture for the Smart Grid are fully understood.

2.1. Study of the state of the art in middleware architectures

The study of the already available solutions regarding middleware architectures for the Smart Grid should come as the first step to be done when developing a new one. The logic behind this reasoning is getting a grasp of the features and possibilities of the already developed works, as it is more cost-efficient in terms of monetary and time resources counting with some previous point to start the actual development, rather than having to "reinvent the wheel" every time a new architecture has to be designed and implemented. Depending on the needs of one deployment, the capabilities of one solution might be enough for the requirements of the project, albeit it should be considered as a symptom of having not so original work or research objectives. Besides, it has to be taken into account that some of the solutions might have not been disseminated enough, and in fact they could be only partial implementations of theoretical proposals, with little to no code available to be reused or at least be employed as a template for a development done from scratch. The tools that can be used to get a grasp on the available solutions are as follows:

- Scientific papers: these are one of the most common know-how resources that can be used to have an idea of the work carried out in a particular area of knowledge. Typically, it is best to use a scientific-themed search engine in order to look for scientific papers, so that a collection of them will be composed and they can be consulted in search for relevant information. Relevant search engines with scientific papers to show are Google Scholar [4] or the IEEE Xplorer Digital Library [5]. It is not rare that some of these search engines have their full load of content only accessible under a subscription, where a monthly fee has to be paid to access to all the data. Nevertheless, some institutions (research centers, universities, etc.) pay for that service and offer it in a transparent manner to their own members (research staff, students, etc.). Several academic publishing companies may also have a scientific search engine in their websites. As far as information collection from other purposes is concerned, there are two kind of scientific papers that can be taken into account, which can be categorized as survey-themed and innovation-themed papers. Usually,

http://dx.doi.org/10.5772/62870

survey-themed papers have already collected the state of the art of a particular topic that could match with a varying degree of accuracy the one that the researchers are aiming their efforts to. They can be expected to contain a list of solutions already published and tested depending on the location of their area of interest (e.g., theoretical proposals may have been tested in a laboratory or in a research institution, whereas project-related ones might have to prove the exploitability of their concepts from a more business-oriented point of view, or have to be deployed during a long period of time). Innovation-themed papers are more focused on describing a novel idea that implies a step forward in the boundaries of knowledge on a certain topic. These latter ones often contain a Related Works section where available solutions (at the time of having the innovation-themed paper published) are reviewed to an extent. Experienced readers of scientific papers are usually capable of distinguishing which content or kind of paper is more interesting for their goals.

- Online and offline literature: One of the most obvious ways to obtain information from a topic is consulting books on it. These resources might be found both as online (eBooks, online journals) and offline resources (hardcopy books and journals). Academic publishers and journals are one of the longest existing sources of relevant information in research for different areas of knowledge. It is not uncommon to find out that some of them started publishing scientific works during the nineteenth century [6]. As far as middleware is concerned, *The Complete Book of Middleware* [7] or *Middleware and Enterprise Application Integration* [8] could be regarded as some of the most popular books due to the accuracy and detail of their content. The Smart Grid also has its own share of advisable literature, with journals as IEEE [9] offering profuse data about how the power grid can be made smart. Special issues from scientific journals can be used as a source of knowledge as well, especially if they match the topic of interest that is being researched. It must be taken into account, though, that while there are journals devoted to very specific areas of knowledge there is no one, to the best of the author's knowledge, which is fully focused on middleware architectures. Furthermore, white papers (that can be defined as reports of short length that attempt to define clear features about a specific issue regarding Information and Communication Technologies) can also be used as a source of knowledge. Both the Smart Grid [10] and some commercial distributions of middleware [11] have several white papers providing knowledge, which provide a very immediate source of information that can be used by technicians or staff with a lower degree of interest in research.
- Research Projects: Research projects funded all around the world by public national or supranational authorities [12, 13], semiprivate or private research institutions or universities usually offer some public resources, as part of the tasks that have to be done regarding dissemination and documentation of the very project that is being funded. The easiest way to obtain material from research projects is checking their own websites [14, 15], although members of the consortium made by all the partners participating in the project can be contacted as well, provided that they offer contact details. The available online materials that can be downloaded from the websites of those projects can be very diverse; the most usual ones include public deliverables (documents with features of different parts or tasks from the project that describe the development tasks that have been carried out), an index

of the publications associated with the project (showing the name of the publication, its authors and the journal where it was accepted) and the partners that are included in the consortium responsible for the execution of the project.

- Repositories and hosting services: as far as implementation works are concerned, a possible starting point to create a new software solution can be obtaining feedback from the online available code in order to get an idea on how to create a software development, design software functionalities or even debug some already developed code. The most widely known repository of these characteristics is GitHub [16], where code that is willingly uploaded by a large number of developers can be obtained, improved and shared again, as long as the person interested in obtaining and sharing data creates an account in the site. Although repositories can be a useful place to get some immediate knowledge on a specific task, the code that is shown is usually provided "as is," without any guarantee of it working flawlessly (in the end, it makes sense that is done like that, as it is obtained cost-free) or completely fulfilling the tasks it was expected to perform.
- Social networks: There are some scientific research social networks that can also be used as a way to collect information from the current state of the art in one area of knowledge, as the members of those networks can be scientists with a high level of expertise [17] able to provide some background information. As it happens with repositories and hosting services, scientific social networks must be approached with a degree of caution, since reliability of social networks as a way to obtain feedback is usually way lower than the one obtained from more orthodox sources, as the quantity and quality of the answers relies on quantity and quality of the peers willing to provide their know-how in one topic. It is not uncommon to obtain misleading or plainly wrong answers to a question that will hinder the undergoing research. Social networks should be used in close cooperation—and subordination—to other more scientifically sound sources rather than being taken as the only source of research material.

2.2. Election of functional and nonfunctional requirements

From the resource usage point of view, the requirements for a Smart Grid middleware architecture may vary from one deployment to another, but in the end, they result similar in most of the cases. Functional requirements are strongly related to the functionalities that are expected from a system of these features. They involve a certain number of capabilities that the system (by system, it is meant the microgrid that is deployed as part of a wider Smart Grid) offers to some actors, who may or may not be human beings (machinery located outside the Smart Grid, e.g., may make use of services offered by it, such as tariff policy elaboration and modification in real time).

On the other hand, nonfunctional requirements are very specific features that are imposed on the system, which will have an impact in system performance (by setting a minimum level of certain parameters or limiting the capabilities of the system). Rather than describing or being related to system functionalities, they set the boundaries for them, so that the functional requirements will be fulfilled with success within the borders set by the nonfunctional ones. The reasons for having strict nonfunctional requirements are varied and are usually due to

http://dx.doi.org/10.5772/62870

technical (hardware capabilities of the devices used in the microgrid, open or proprietary hardware, location of the required appliances in a deployment) or economical (budget limits, Service Level Agreements, reutilization of already existing premises and/or equipment) motivations.

2.2.1. Functional requirements

When actors become engaged to the functional requirements the latter start being referred to as use cases. According to the principles of software engineering, use cases can be defined as a group of events and actions performed within the boundaries of a system for the benefit of some actors out of it, but using the system. In this context, functional requirements define those events and actions to be made by the power grid once it has been enhanced with a middleware architecture, so the latter is pivotal to obtain the desired performance from the former. A few of the most common use cases are as follows:

- **Device registration:** When one new device is added to the microgrid, its existence must be acknowledged by the whole system in order to obtain and provide data used to perform the functionalities described by the functional requirements. The most usual new equipment to be added are smart meters, as the last mile infrastructure of a microgrid integrated in the Smart Grid is more likely to change than the core of equipment and technologies (due to client rotation, more frequent smart meter substitution due to low prices, etc.).
- **Data request:** As in any distributed system, data will be offered as a service for the end users at the peripheral zone of the system. For them, data will be of critical importance in order to make decisions regarding their energy consumption or the services they would like to obtain from Smart Grid stakeholders such as the transmission system operator (TSO) or even the distribution system operator (DSO), in case they count with their own supply of renewable energy systems (RESs). Data can be requested with end user intervention (information for end-user mobile apps or web applications in a more general) or without it (bidirectional data transfers between advanced metering infrastructure and the core of the Smart Grid).
- **Alarm triggering:** Alarms might be triggered in the microgrid if an unforeseen event takes place; they can be related to some kind of economic feature (tariff limits, overspending, waste of energy) or a physical one (damage in one piece of the installed equipment, software failures). As it can be guessed, alarms are usually triggered by either the software installed in the distributed pieces of equipment or by the equipment themselves, rather than by any end user (unless they have been given the option to do so by means of an application).

These use cases can be expressed in a more formalized manner by means of unified modeling language (UML). As it is implied by its name, UML is a software modeling language that is profusely used for software design as a previous step to implementation works [18]. Since one of its main goals is describing in a graphical and accurate manner the features of one system, as well as the relationships among its different components, it comes in handy for the representation of the uses cases that will be studied at the middleware architecture level. UML makes use of many different diagrams to explain the different functionalities and viewpoints

of one system; for example, computational analysis can be used to have a reference of the different subsystems that are used in the overall middleware architecture, along with the relationships that are held among them. Thus, subsystem diagrams are used in order to depict those relations and subsystems. Additionally, functional analysis is used to describe some other features of the system related with the functional requirements: it is here where the use case diagrams that show the defined use cases, along with their links to the actor taking part of the system as external stakeholders, are represented. Use case diagrams are employed to describe elements of the Smart Grid (as it is shown in the **Figure 3**), display the actors involved in the system, the use cases that have been contemplated, the boundaries of the system containing those use cases and the relations between use cases and actors. While the content of the use cases might differ greatly from one are of knowledge from other (a microgrid is very different from a wireless sensor network or a group of cooperating robots), their representation varies little among them.

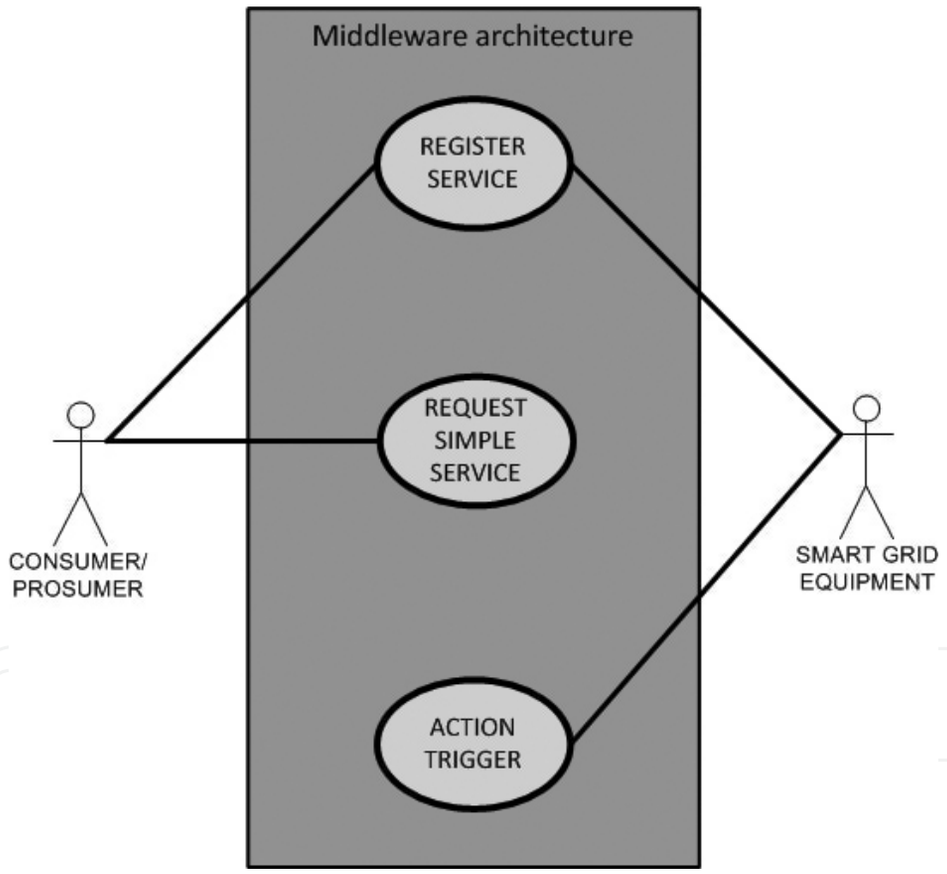

Figure 3. Use case diagram for common use cases.

At the same time, component diagrams depict the different software elements that make a subsystem and how they obtain resources one from the other by means of component interfaces that transfer the information among the existing components. Some other diagrams are advised to be used for functional analysis; for example, sequence diagrams show the set of

http://dx.doi.org/10.5772/62870

steps that are made for a use case to be completed. Each of those steps is a software instance that is needed to get through to complete the flow of actions that will end up providing the service, as each of them provides a significant inner action for the fulfillment of the requirement defined by the use case. Sequence diagrams must be interpreted in a bi-dimensional fashion: the horizontal axis represents the direction of the information exchange (which uses the software instances and the name given to the actions that communicate one instance with the other in both directions), whereas the vertical one represents the time that is used by each of the software entities in the flow of the represented use case. Last but not least, those software entities and action names are better represented by the usage of class diagrams, which gather the different actions to be taken as methods or functions within the software instances that were used in the sequence diagram as instances. Class diagrams are usually very close to the implementation stage, which is typically carried out after the classes and methods defined by this diagram become definitive. An example of sequence diagrams for the already studied use cases is shown in **Figure 4**.

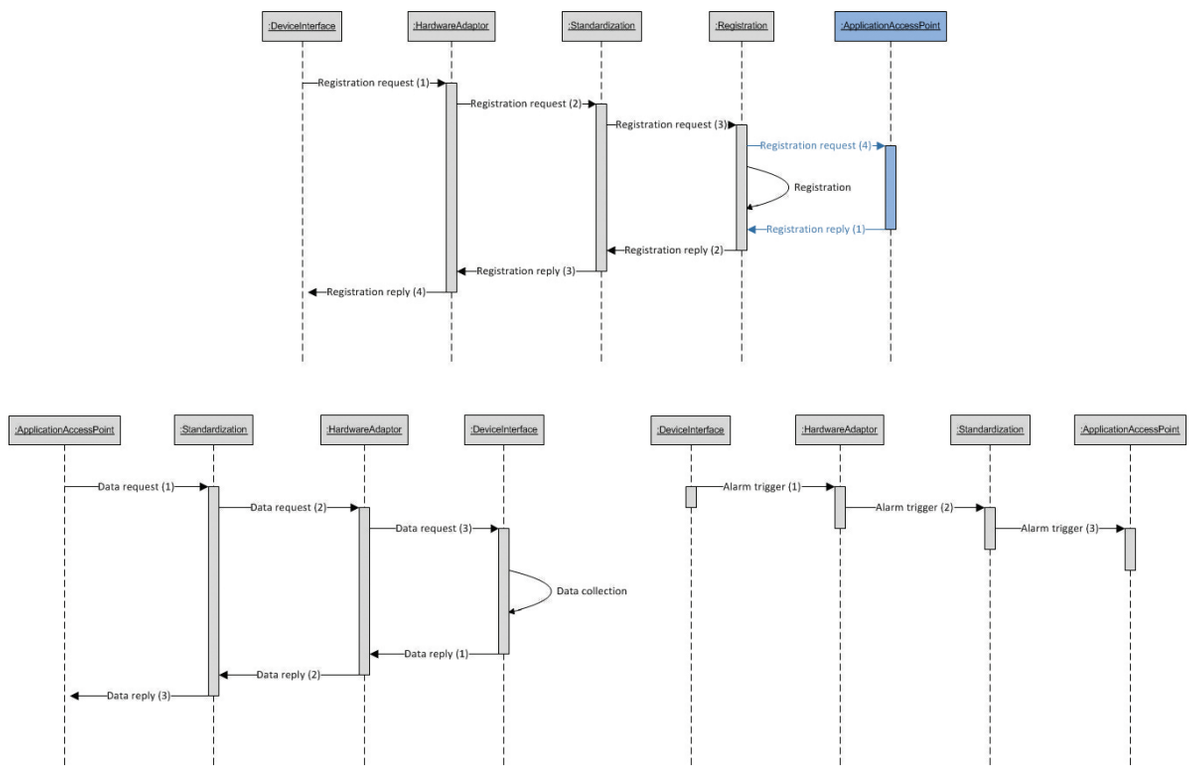

Figure 4. Sequence diagrams for the previously shown use cases.

Here, it can be seen in the first sequence diagram how a typical device registration procedure would take place: the interface of the device that is going to be registered keeps the hardware in touch with the communication services enabled in the Smart Grid. By means of this interface, a first request message is sent. This message is transferred throughout the communications system (which will typically involve IP communications at the network level and TCP/UDP at the transport one) until it reaches the middleware component responsible for receiving

request messages from devices (hardware abstractor). This component will send the received message to the software module in charge of formatting the messages delivered from levels located lower than the middleware layer (standardization). This is the message, formatted according to what has been defined for the middleware (and containing all the information regarding the services that can be offered by the device that is going to be registered) that will be used for registration. Commonly, a registration acknowledgement message will be sent as a reply to the device that requested the registration and data transfer will be ready to be done, although depending on the degree of control that is given to the end user, the registration request may be sent to the application access point to be evaluated by a human operator. The other two sequence diagrams deal with data interchanges and alarm triggering. Data are requested from an application connected to the middleware via the application access point and is sent though the standardization module in order to have it sent according to the format that is understandable by the hardware device that is being requested. Hardware abstractor sends the request through the network layer until it is received by the interface at the very device capable of providing the service. Once the data are collected, they can be sent back to the middleware architecture by means of the communications network until they are received again at the application access point, which will interface with the application to deliver the data. Alarm triggering is a process used to inform the end user about an abnormal situation that has happened in the grid. Depending on the nature of the alarm, it might be sent by either the hardware component that has detected the issue or other system entities, such as a middleware module. When the issue is detected by a hardware device (as depicted in **Figure 4**), a message containing information about it will be sent through the communication system to the middleware components previously depicted (hardware abstractor, standardization) until it reaches the application access point, which will send it the application as an alarm that has been push from the underlying system.

As far as the Smart Grid is concerned, the boundaries of those use cases are frequently defined by the most hardware-based elements of the deployment (power grid appliances, smart meters, etc.) on one side, and the closest software entity possible to an end user (mobile app, website, etc.). Hence, the sequence diagrams are representing the relations among those border elements by means of action interchanges among some other, more internal software instances.

Another very accurate representation of the different stages used to design a middleware architecture can be found in [19]. Here, it is explained how the different functionalities and software modules are conceived, related and located in the middleware system.

2.2.2. Nonfunctional requirements

The other kind of requirements that was presented is the nonfunctional ones. As it happened with the former, they are present in almost any imaginable distributed system, although they can appear in any other system that is not distributed, as long as some boundaries are imposed to have a system working under realistic circumstances. Nonfunctional requirements are a group of features that strictly define the main features that establish what a system is. This is something that must be born in mind, as sometimes they can be loosely interpreted as the

http://dx.doi.org/10.5772/62870

boundaries of one system. Nonfunctional requirements have been used for quite a while in the design of both hardware and software products. For example, nonfunctional requirements established during the design stages of the popular Citroën 2CV where used to devise *"a design brief for a low-priced, rugged "umbrella on four wheels" that would enable four small farmers / peasants to drive 50 kg (110 lb) of farm goods to market at 50 km/h (31 mph)"* [20]. By defining what that car would be, the boundaries of that design were implicitly mentioned as well. It is easy to regard nonfunctional requirements as constrains; however, they usually describe features that are necessary to understand the system that is being design, rather than imposing mutilating conditions to the system.

As for the middleware architectures for the Smart Grid, nonfunctional requirements are often focused on the properties that both distributed systems and the power grid have: on the one hand, features as scalability, resilience or data security impact the system from a distributed software development point of view. On the other hand, electricity distribution characteristics as power consumption or load peak limits have their own impact in the design of the overall system too.

2.3. Semantics and middleware architectures

As previously stated, the main reason for middleware architecture to exist is its capacity to homogenize the different information representation formats that are received from lower layer elements. Among the several software tools that can be used to carry out this functionality, semantics is one of the most effective. In information and communication technologies, semantics can be defined as the capability of a system to infer knowledge form gathered data so that the meaning of the transmitted data will be apprehended by it and the system will be able to use it in the future to optimize its overall performance. The core idea around semantics is that raw data are not only meant to be transmitted from one remote location to another but also processed in a way that will make the system able to infer the actual meaning of the information that is being transmitted throughout the distributed system. For example, should raw data be transmitted about temperature in an equipment, a semantically enhanced system will be able to both transfer the data and assess them, so that an educated decision can be taken with regards of the information transmitted. In a nutshell, a middleware architecture using semantics will have learnt, or inferred, that a very higher than usual temperature means that there is some kind of trouble with the appliance or the geographical area the temperature reading was retrieved from.

Semantics become implemented in a more tangible way by means of ontologies. An ontology is a collection of terms and definitions of wide use in a software system (which may be distributed or not, but it is assumed here that it will be so, as the rules of ontologies also apply to them), along with the relationships between those terms, to the point that it should be considered more as a graph with interrelated elements rather than a dictionary [21]. In addition to that, an ontology can be updated to incorporate new elements or even drop deprecated ones,

so the knowledge that is acquired by the system can be refreshed as it works during its usual lifespan, thus becoming "wiser" as time goes on.

As described in [19], a semantic module usually consists of several parts:

- An action collector that will retrieve the actions that are being carried out by the system. The pace for collecting actions data can be adapted to the needs of the system, ranging from several times during the day to an almost real-time pace. The smaller of that time span the work strain is put to the whole module (and by proxy, the middleware architecture), though.
- An inference managing entity that infers knowledge for the system, based on the actions that have been collected, the rules fixed for the system and another repository for stored facts.
- A rules repository that stores the norms that have been fixed for the system. They are of major importance to decide whether an action should be triggered or not.
- A facts repository where the past actions that were gathered by the action collector are kept.
- An action trigger component that will be used to send commands whenever the semantic module rules it is necessary. This component comes especially in handy for managing alarms and events.

The most popular semantics element used in the Smart Grid is the CIM [2], which defines its own ontology for most of the pieces of equipment that can be found in a power grid. The representation of CIM is identical to what can be expected from a UML one; this can be deemed as a token of the increasing involvement of software engineering in the power grid.

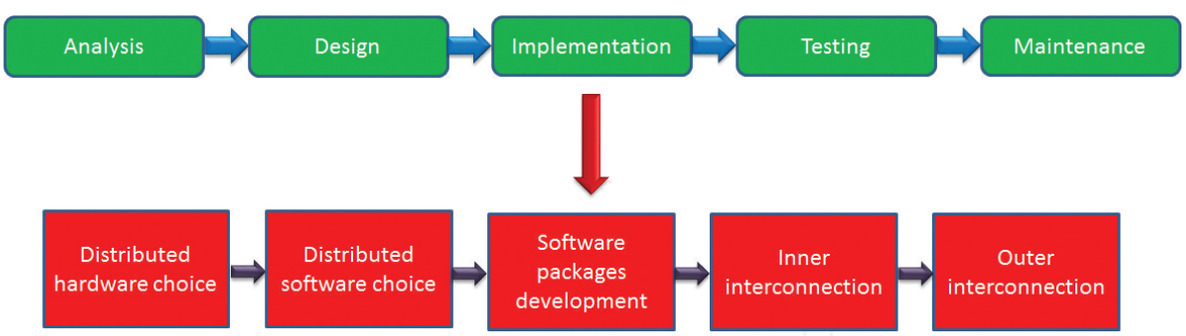

Figure 5. Waterfall model for software design and development tasks for middleware.

Overall, the methodology used to design a middleware for the Smart Grid may be flexible to an extent. For example, the traditional waterfall model can be used to complete the intermediation layer until all its expected functionalities have been performed (see **Figure 5**), even though it is advisable to have some feedback in each of the stages in the very likely case that any bug or deviation from the scheduled functionalities has to be dealt with. Usually, when

http://dx.doi.org/10.5772/62870

tackling the implementation stage, there are five steps that must be taken into account to successfully complete the development works of the middleware architecture:

- A decision must be made regarding which hardware devices present in the Smart Grid deployment will be used to have the middleware installed. There might be some appliances that are proprietary solutions without any possibility to have new software installed.
- The software platform that is going to be used to have all the software components installed must also be chosen. It should be capable to work in a distributed way, sending messages from one part of the platform to other ones located remotely. Enterprise service bus architectures (ESBs) work in a suitable manner to solve this challenge, and they can send messages among different software components embedded in them.
- The software packages that are developed to implement the services that are going to be located in the middleware architecture must be codified once the hardware devices and the software platform have been defined.
- Connectivity among the software modules that have been implemented must be guaranteed in order to have them all able to send and receive information whenever there are data to be transmitted through the middleware architecture, either toward the application layer being used by the last mile clients or the network layer interconnecting the deployed appliances.
- Connectivity among the different elements outside the middleware must be accomplished so that there will be a seamless integration of all the subsystems that conform the Smart Grid (or at least, the microgrid where they have been deployed) and data can be transferred from/to the applications developed on top of the middleware architecture to/from the hardware devices present in the system deployment done.

3. Services and components in a layered middleware architecture

The deployments where middleware architectures are used have some important similarities, as it has already been hinted in several sections. These can be easily obtained by taking into account the most usual functional requirements that are expected from those deployments, that is, (a) hardware abstraction for communications between the intermediate software layer and the pieces of equipment in the Smart Grid, (b) semantic capabilities to make the overall system smarter, (c) upper-level access points to the applications that are interconnected to the middleware, (d) registration of the detected devices, (e) other services as context awareness, security, etc. Arguably, wherever a middleware architecture becomes deployed, those services are taken for granted by the end user, who will count on them without noticing which software or hardware entity is providing them. Taking that fact into account, a generic middleware layer can be defined that will be matching most of the requirements that have been imposed to the system. The appearance of the architecture is displayed in **Figure 6**:

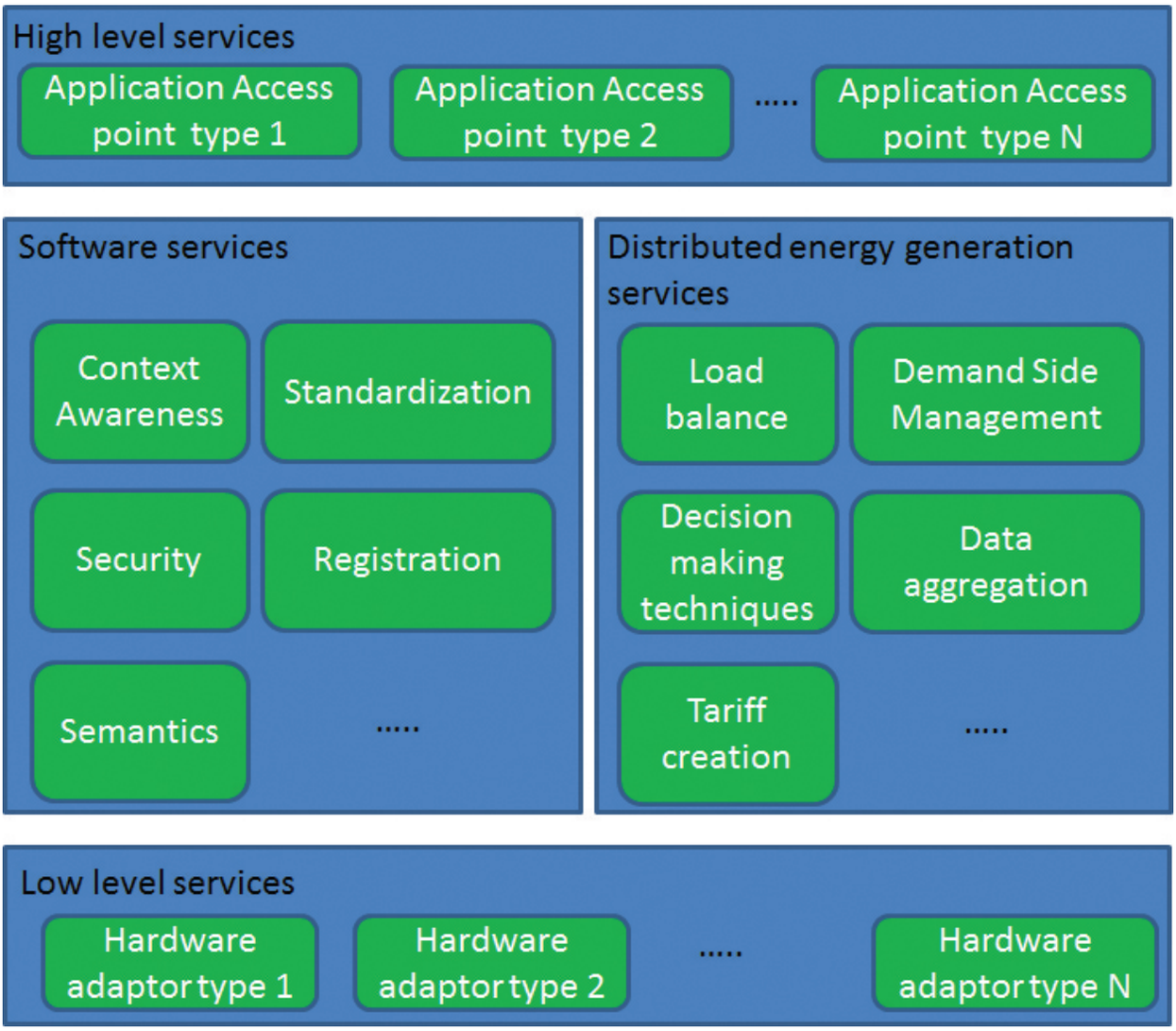

Figure 6. Waterfall model for software design and development tasks for middleware.

The services that are going to be offered have been divided into four different kinds, each of them composing groups of services that are located in different places according to their characteristics. They are as follows:

- **Low-level services:** These are the ones that are more closely interacting with the pieces of equipment where the middleware architecture has been deployed. Basically, they are responsible for receiving the data that is transferred from the interfaces of the appliances that are used in a microgrid. Consequently, definition of the behavior of the low-level interfaces between middleware and hardware is critical for the correct delivery of the information. There are several protocols that can be used for that matter. For example, Java Message Service (JMS) [22] can be used between entities capable of understanding that programming language. Another protocol suitable for this task is advanced message queuing protocol (AMQP) [23] which is able to communicate devices by means of message queuing. The information interchange will be performed by using hardware adaptor

modules, which will change their codification depending on the nature of the devise that are interfaced.

- **Distributed energy generation services:** These are the services that are most closely related to the Smart Grid capabilities. Furthermore, this is the module that is most prone to changes, as the services that are required from one microgrid deployment to another one (let alone a completely different distributed system) may differ greatly. The most usual services that can be found are for load balance (defining the shared amount of resources that each of the elements present in the grid is going to cooperate with to deliver energy), making decisions (regarding both technical and management issues), creating tariffs (with the purpose of offering them in a dynamic way, as a mobile phone contract or ASDL services), aggregating data (in order to collect statistical information about the transferred information that will optimize other services, such as load balance or processes involved in making decisions) or offering demand side management (as a way to shave the power peaks that happen during certain time spans).
- **Software services:** They offer a significant amount of focus in the distributed system nature of the deployment. They can be found in almost any environment that implies a middleware architecture used as an intermediation layer among devices. The most important ones are context awareness (in order to notice any change in the deployed elements and having the middleware reacting to those changes in cooperation with the semantic operations), standardization of the transferred information, security (which might be offered as encryption services in this group of services or as secure web interfaces in the high-level services), registration (so as to have a collection of the available devices and the services they can provide to the end users or other elements of the Smart Grid) and semantics (working closely with standardization to use a common data model throughout the whole middleware architecture).
- **High-level services:** The weight of these services in the architecture may vary from one deployment to the other: in some cases, they behave as if they were applications, and in some other, they just offer an access point to the actual applications. Among the technologies that can be used to implement the interfaces for information exchange from/to the applications are constrained application protocol (CoAP) [24], useful for lightweight requirements, and representational state transfer (REST) interfaces [25], which offer an easy way to establish communications via web services.

The most usual way for them to be using each of the functionalities is expressed in **Figure 7**. As it can be seen, the subsystems need functionalities from each other, namely, the high-level services subsystem is using the software and distributed energy subsystems to obtain the services contained in each of those parts of the middleware. At the same time, those two subsystems are interacting with the low-level services to collect the information fetched from outer, lower layers. Interestingly enough, software services are often used by the services that are more typical of the smart grid, but they are use more as a way to offer support than providing an actual service to the end user, so software services rarely employ the distributed energy services (hence the unidirectional arrow in **Figure 7**). In any case, depending on the use case that is being dealt with, some components of the middleware architecture may be

required and some others may not, in the different ways that were described by the sequence diagrams. The services that have been described here are the ones that are most popular and useful for a system of the features presented here (distributed systems that enhance the regular power grid). Nevertheless, there might be some other ones that are more important for punctual environments or developments.

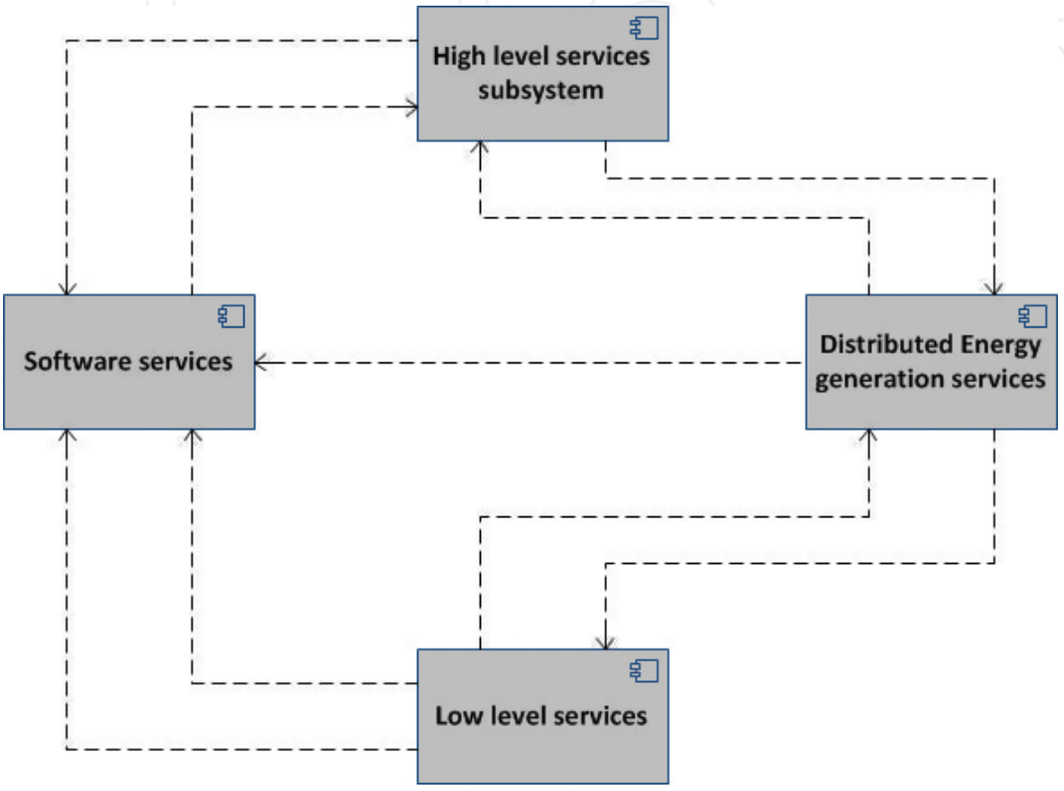

Figure 7. Software subsystems within the generic middleware architecture for the Smart Grid.

4. A real case study for middleware in the smart grid

The ideas described in this chapter have been put into practice in several application domains. One of them is regarding a European project called e-GOTHAM [26] that involved building a Smart Grid where several pieces of advanced metering infrastructure and power grid-related appliances were integrated by means of a semantic middleware architecture. The designed semantic middleware architecture was published as a scientific research work [27] and an implementation of it, with all the necessary software components for that specific iteration, was used in the Finnish city of Ylivieska [28]. The overall structure of the power grid network (with the middleware architecture installed in a machine) was also reported as a standardization and dissemination activity [29] and is presented in **Figure 8** in a high-level manner.

As described, there are several buildings located in the city of Ylivieska (Concert Hall, schools, a power plant, etc.) using several hardware parts (such as wireless sensor networks) that can

http://dx.doi.org/10.5772/62870

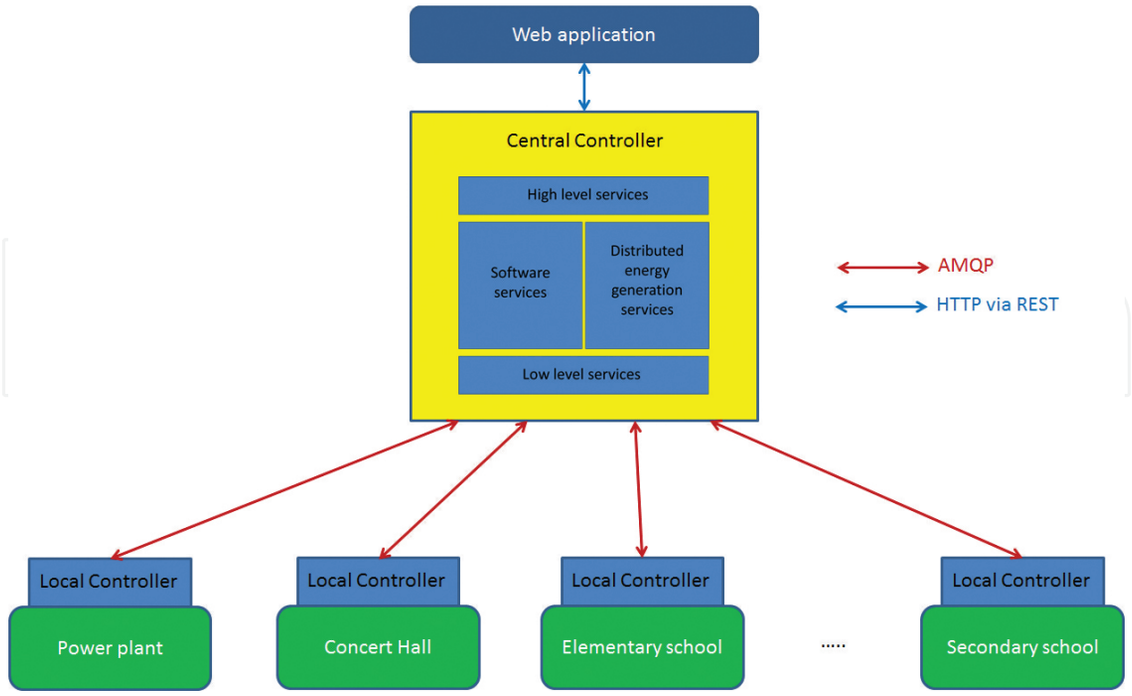

Figure 8. Microgrid deployment with the middleware architecture.

be regarded as AMI. That infrastructure is interconnected by means of local controllers that, depending on their capabilities or whether they are proprietary or open solutions, either can have middleware modules installed, or will rely on the middleware components installed in the central controller. It is this central controller where most of the software modules of the middleware are present, in accordance with the referred architecture (device registration, semantic capabilities, access points to the application layer, etc.). The framework that has been used to have all the software components contained was an open source Enterprise Service Bus, currently known as JBoss Fuse [30]. Collection of data was made from those buildings and presented in a web application with some ancillary functions added (e.g., energy consumption forecast). Other projects, such as I3RES [31], have used similar ideas in order to build middleware architectures for the Smart Grid that have been used in living labs such as Steinkjer, a Norwegian city that often participates in this kind of purposes [32].

5. Conclusions

The usage of middleware architectures for the Smart Grid offer clear benefits on the deployments where they are installed: they guarantee that many different pieces of hardware can cooperate seamlessly (regardless of the manufacturer that makes them or the data representation formats that they use), offer different services when applications of devices cannot deliver them, upgrade the system so that it will be able to infer information from the transferred data to be used in the future and improve the services that can be used by the power grid, turning them into a smarter entity capable of providing more information and making smarter decisions. In addition to that there are several middleware modules (hardware abstraction

modules, context awareness, data aggregation, decision making modules) that are almost mandatory if the full potential of middleware is to be obtained. Fortunately, its widespread usage makes possible that, by collecting information from other proposals or studying the state of the art, can either have some previously developed modules adapted to the need of the middleware architecture in one particular system, or develop new ones with a reduced amount of work, as some guidelines have been offer in this chapter to do so.

The key contribution of this chapter is the summarization of prominent knowledge that has been noticed throughout distributed systems and software architectures research to come to the conclusion that, since there is a plethora of services that will almost always be required, a generic middleware architecture, with a very well defined set of functionalities, can be used whenever middleware is required. This idea can be used for successive deployments that might improve the Smart Grid even further, shortening development times and making a more cost-efficient use of economic resources.

Author details

Jesús Rodríguez-Molina*

Address all correspondence to: jesus.rodriguezm@upm.es

Research Center on Software Technologies and Multimedia Systems for Sustainability (CITSEM - Centro de Investigación en Tecnologías Software y Sistemas Multimedia para la Sostenibilidad), Technical University of Madrid (UPM), Spain

References

[1] Alani, M.M., "Guide to OSI and TCP/IP Models". SpringerBriefs in Computer Science. 233 Spring Street New York, New York 10013–1578. USA. ISBN 978-3-319-05152-9. 2014.

[2] Simmins, J.J., "The impact of PAP 8 on the Common Information Model (CIM)," in Power Systems Conference and Exposition (PSCE), Phoenix, Arizona 2011 IEEE/PES, pp. 1–2, 20–23 March 2011. doi: 10.1109/PSCE.2011.5772503.

[3] CEN-CENELEC-ETSI Smart Grid Coordination Group. "Smart Grid Reference Architecture".http://ec.europa.eu/energy/sites/ener/files/documents/xpert_group1_reference_architecture.pdf. (last accessed November 2012).

[4] Google scholar, https://scholar.google.es/. (last accessed February 2016).

[5] IEEE Xplore, http://ieeexplore.ieee.org/Xplore/home.jsp. (last accessed February 2016).

[6] Elsevier, company history, https://www.elsevier.com/about/company-information/history. (last accessed February 2016).

[7] Myerson, J.M., "The Complete Book of Middleware". Auerbach Publications. 6000 Broken Sound Pkwy NW, Florida. USA. ISBN 978-0849312724. 2002.

[8] Serain, D., "Middleware and Enterprise Application Integration". Springer. Schwedenstraße 9, 13359 Berlin, Germany. ISBN 978-1852335700. 2002.

[9] IEEE, "IEEE Transactions on Smart Grid", http://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=5165411. (last accessed February 2016).

[10] Association of Home Appliance Manufacturers, "Smart Grid White Paper: The Home Appliance Industry's Principles & Requirements for Achieving a Widely Accepted Smart Grid". https://www.aham.org/index.php?ht=a/GetDocumentAction/i/44191. (last accessed December 2009).

[11] Oracle Corporation. "Oracle Fusion Middleware and Microsoft Interoperability. Addressing Enterprise-wide Needs". http://www.oracle.com/technetwork/middleware/fusion-middleware-microsoft-interop-1-131446.pdf. (last accessed February 2006).

[12] Horizon 2020 –European Commission. https://ec.europa.eu/programmes/horizon2020/. (last accessed February 2016).

[13] Electronic Components and Systems for European Leadership – Joint Undertaking. http://www.ecsel-ju.eu/web/index.php. (last accessed February 2016).

[14] Sensing, monitoring and actuating on the UNderwater world through a federated Research InfraStructure Extending the Future Internet. http://fp7-sunrise.eu/. (last accessed February 2016).

[15] ICT-Based Intelligent Management of Integrated RES for the Smart Grid Optimal Operation. http://www.i3res.eu/v1/. (last accessed February 2016).

[16] GitHub. https://github.com/. (last accessed February 2016).

[17] ResearchGate. https://www.researchgate.net/. (last accessed February 2016).

[18] Object Management Group. http://www.omg.org/spec/UML/2.5/. (last accessed February 2016).

[19] Rodríguez-Molina, J.; Martínez, J-F.; Castillejo, P., and de Diego, R. "SMArc: A Proposal for a Smart, Semantic Middleware Architecture Focused on Smart City Energy Management," International Journal of Distributed Sensor Networks, vol. 2013, Article ID 560418, 17 pages, 2013. doi:10.1155/2013/560418

[20] Bellu, R., "Toutes les Citroën, des origines à nos jours". Jean-Pierre Delville éditeur, p. 250, Paris, France. ISBN 2-85922-014-3. 1979.

[21] Díaz, V. H.; Martínez, J-F.; Cuerva, A.; Rodríguez-Molina, J.; Rubio, G.; Jara, A. Chapter 9: "Semantic as an Interoperability Enabler in the Internet of Things" from "Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems

(River Publishers' Series in Information Science and Technology)". ISBN 978-8792982735. June 2013. Niels Jernes Vej 10, 9220 Aalborg Denmark

[22] Chapter 47: "Java Message Service Concepts" from "The Java EE 6 Tutorial", 500 Oracle Parkway, Redwood City, California 94065. USA. https://docs.oracle.com/javaee/6/tutorial/doc/bncdq.html. (last accessed February 2016).

[23] AMQP, Advanced Message Queuing Protocol. https://www.amqp.org/. (last accessed February 2016).

[24] CoAP RFC 7252, Constrained Application Protocol. http://coap.technology/. (last accessed February 2016).

[25] Fielding, R.T. and Taylor, R.N. "Principled design of the modern Web architecture". Proceedings of the 22nd international conference on Software engineering (ICSE '00). ACM, New York, NY, USA, pp. 407–416. DOI=http://dx.doi.org/10.1145/337180.337228. June 2000.

[26] e-GOTHAM ("Sustainable—Smart Grid Open System for the Aggregated Control, Monitoring and Management of Energy"). http://www.artemis-ju.eu/project/index/view?project=39 Last accessed date: 20th May 2016.

[27] de Diego, R.; Martínez, J.-F.; Rodríguez-Molina, J.; Cuerva, A. "A Semantic Middleware Architecture Focused on Data and Heterogeneity Management within the Smart Grid," Energies, vol. 7, pp. 5953–5994, 2014.

[28] Ylivieska town web site (Finnish). http://www.ylivieska.fi/. Information from tourist guide: http://www.ylivieska.fi/instancedata/prime_product_julkaisu/ylivieska/embeds/ylivieskawwwstructure/18035_75-Ylivieska-brochure-new_tourist_guide.pdf Last accessed date: 6th April 2016.

[29] e-GOTHAM Validation report for prototype 2 in the three pilots (deliverable D8.4). https://drive.google.com/file/d/0B0xf2wnfLaObV0VrS0Y4aUFGenc/view Last accessed date: 20th May 2016.

[30] Red Hat JBoss Fuse, Download for Development Use. http://www.jboss.org/products/fuse/download/ Last accessed date: 6th April 2016.

[31] I3RES – ICT based Intelligent management of Integrated RES for the Smart Grid optimaloperation.https://www.citsem.upm.es/index.php/en/projects-en?view=project&task=show&id=37 Last accessed date: 6th April 2016.

[32] Demo Steinkjer Living Lab (Norwegian). https://www.demosteinkjer.no/ Last accessed date: 20th May 2016.