INFORMATYKA EKONOMICZNA BUSINESS INFORMATICS 3(33) • 2014
ISSN 1507-3858

Daniel Wilusz, Jarogniew Rykowski
Poznań University of Economics
e-mails: {wilusz; rykowski}@kti.ue.poznan.pl

ORCHESTRATION OF DISTRIBUTED HETEROGENEOUS SENSOR NETWORKS AND INTERNET OF THINGS[1]

Abstract: This paper focuses on management of sensor networks and Internet of Things. Two alternative architectures for service management in IoT and sensor networks are analyzed. The first one is based on Open Service Gateway (OSGi) framework and Remote Services for OSGi (R-OSGi) bundle. The second one extends Representational State Transfer (REST) paradigm. The analysis confirms that these two architectures meet the requirements of both sensor networks and IoT management. However, to bypass the disadvantages of these architectures, the new one is proposed being a combination of the analyzed architectures. In our proposition, the OSGi framework is applied to manage local sensor networks and REST-based architecture allows for managing complex services in Internet of Things. The RESTlet OSGi module integrates the OSGi management system for sensor networks with REST-based management of Internet of Things.

Keywords: Internet of Things, Future Internet, sensor network, REST, OSGi.

DOI: 10.15611/ie.2014.3.09

1. Introduction

The Internet evolves continuously offering new services and possibilities. Nowadays we face the era of Future Internet, where among such hot topics as cloud computing, big data analysis or social media, the Internet of Things (IoT) appears. The IoT is significantly influencing business and society on a global scale. We observe a continuous IoT evolution. Primarily, IoT was defined as an intelligent network connecting devices, information and people to enable remote coordination of resources by humans and machines [Brock 2001]. However, nowadays IoT is perceived as a cutting edge phenomenon, no longer limited to the electronic

[1] Selected parts of this article were published under nonexclusive copyright in Proceedings of the Federated Conference on Computer Science and Information Systems FedCSIS 2014 (see [Rykowski, Wilusz 2014]). This work was supported by the GOLIATH project jointly funded by the Poland NCBR and Luxembourg FNR Lead Agency agreement, under NCBR grant number POLLUX-II/1/2014 and Luxembourg National Research Fund grant number INTER/POLLUX/13/6335765.

identification of objects, but defined as a technology integrating devices with information network, where these devices act as active participants in business processes [Haller et al. 2009].

Plenty of application areas of IoT were identified in such economy areas as manufacturing, supply chains, energy, healthcare, automotive industry, insurance, financial services or research laboratories, to mention a few [Haller et al. 2009; Wilusz, Rykowski 2013; Wilusz et al. 2013]. It is expected that Internet of Things will expand outside the internal infrastructures of companies. The development of Internet of Things requires proper architecture, which will meet the requirements of dynamic and heterogeneous IoT environment. In this paper two alternative architectures for service management in IoT and sensor networks are discussed. After consideration of drawbacks of these architectures the new one is proposed, being a smart mixture of previously analyzed ones.

The remainder of the paper is organized as follows. Section 2 describes essential characteristics of sensor networks and Internet of Things, which allows for identification of basic requirements of these networks. Section 3 consists of description of The Open Service Gateway (OSGi) framework and presentation of OSGi-based architecture for IoT management. Section 4 introduces basics of Representational State Transfer (REST) and describes an architecture based on extended REST services. Next, the compound OSGi and REST based architecture is proposed in Section 5. Finally, Section 6 concludes the paper.

2. Characteristics of sensor networks and Internet of Things

Sensor networks and Internet of Things may be perceived as similar concepts as both are composed of small hardware nodes with limited resources, such as memory size or CPU computational power. Moreover, both are physically distributed over certain location and communicate by means of standard network protocol. However, this similarity is apparent. This section presents a comparison of sensor network and Internet of Things and enumerates generic requirements for data acquisition and architecture design.

Usually, a sensor network is composed of many nodes having similar purpose and fixed functionality. Typically, one single point of an interaction (gateway) is available, to contact the network as a whole. Additionally, several solutions are applied for energy saving and optimization of information routing, as the nodes are mainly compact hardware devices, battery-operated and with limited computing possibilities.

Sensor network acts as a single group, which is usually controlled in the centralized manner. The devices are not accessed directly from outside, but rather the sensor network provides a set of certain functions taking into account user access rights. Because of no need for individual addressing of devices and their functions, external access and control of individual nodes is usually blocked, and the network

is self-manageable. For example, for energy-saving reasons, some nodes are temporary deactivated and gathered information is pre-processed to minimize network transfer, etc. [Waltenegus, Poellabauer 2010].

On the contrary, a typical IoT network consists of heterogeneous nodes of different purpose and functionality. There is no strict goal defined for the IoT network aside from abstract ones such as user comfort or energy savings. Every IoT device acts as a "good servant" [Weiser 1991] by operating in a useful but unnoticeable way in cooperation with other devices.

IoT networks are composed on ad-hoc basis and interaction with people is incidental and even unnoticed. The interaction with humans depends on local and global context such as a geo-location or environment features.

In IoT networks, the portability is the key issue, and the efficient ad-hoc human-to-machine interactions should be welcomed. These interactions need to be as discrete as possible, by abstracting from communication protocols, device addresses, data format, etc. To provide such abstraction, an efficient searching/filtering mechanism should be available. This mechanism should allow searching and choosing (possibly automatically) an appropriate device to serve given request. In more detail, the human interactions with IoT network should be characterized by the following properties:

- individual addressing of network functions abstracting the implementation of these functions – in particular direct device addressing,
- searching for accurate devices to serve particular request,
- filtering the best (optimal) devices for fulfilling user's request,
- enabling many searching modes, e.g.: any device, at least one device, every device capable of executing request,
- monitoring total activity and accessibility of devices,
- portability,
- scalability in terms of the number of devices as well as the number of requests.

The above-enumerated properties of human interactions with IoT network shows that a global management system for controlling devices is required. Such a mechanism should be centralized, similarly to the management of sensor networks. On the one hand, this centralization may seem to be contradictory with the requirements of IoT devices, in particular the autonomy and their ad-hoc composition and exploitation. On the other hand, only centralized management system will allow grouping devices into bigger conglomerates, orchestrate them and multiply their functionality. Moreover, the centralization of device management will allow searching for optimal devices to serve requests and balance the system load, energy consumption, network traffic, etc. Under the above-mentioned conditions, the optimal approach is to implement local catalogue of device functions and extend this catalogue by statistical analysis of such operations as device load, accessibility or unavailability, last activation time, number of served requests, efficiency of serving requests, etc. Additionally, the catalogue may act as the request broker, mapping

syntax and semantics of any request to the syntax and semantics of heterogeneous devices communication protocol.

In order to implement the above-mentioned catalogue, two frameworks may be taken into account. The first is based on OSGi platform supporting dynamic management of modules and providing out-of-the-box service registry. The second framework is based on an extension of REST services for centralized management of distributed REST resources and servers. In the next chapters we are going to describe and evaluate these frameworks, basing on the operations listed below:

- a registration of a device and its functions,
- an interaction and communication mapping via individual proxy,
- monitoring real-device state and providing information about its accessibility,
- searching for appropriate devices to handle specified request,
- a selection of an optimal device to serve given request based on the context and statistical information,
- an orchestration of device functions based on local context,
- an administration of devices via graphical user interface.

3. Managing IoT and sensor networks by the means of OSGi

OSGi (Open Service Gateway) is a Java based framework which provides specification for dynamic module system [OSGi Alliance 2014b]. The aim of OSGi is to provide a framework for the efficient management of software modules lifecycle, by providing dynamical installation, start, stop and removal of bundles (modules). Bundles are building block of OSGi-based dynamic modular systems. One of characteristics of OSGi bundles is mutual interaction which consists in following actions:

- sharing Java packages,
- registering and calling services,
- managing the lifecycle of other bundles,
- sending and handling events to trigger specified actions.

The functions offered by bundles are controlled by out-of-the-box registry provided by OSGi framework. The OSGi registry identifies services based on their interfaces and metadata. In result, no strict reference is required to invoke a service. In case no service of given interface or meeting additional constraints were found in registry, null value is returned. When multiple services of specified interface were found, and the caller needs only one service, the OSGi registry returns the one with the highest rank (specified by user) and the lowest identifier (the oldest service registered) [Flotyński et al. 2013; Hall et al. 2011].

The OSGi framework offers a built-in, event-based communication. It allows for dealing with data changes triggered by dynamic behavior of the bundles or OSGi framework. Bundles may be programmed to react on specified events, e.g., to start a particular bundle after installation. Moreover, the bundle developers are able to

define the events and handle these events by themselves. The event mechanism enables a specification of events' topics and additional metadata which may be processed by predefined event handling services [Flotyński et al. 2013; OSGi Alliance 2014a; Castro Alves de 2011].

On the one hand, the support for dynamic modules, service registry and event-based communication look like a complete solution for Internet of Things network. On the other hand, OSGi framework needs to be extended in order to meet all the requirement for IoT management system, as discussed in previous section. The extended architecture of OSGi-based IoT management system is presented in Figure 1 and described below.

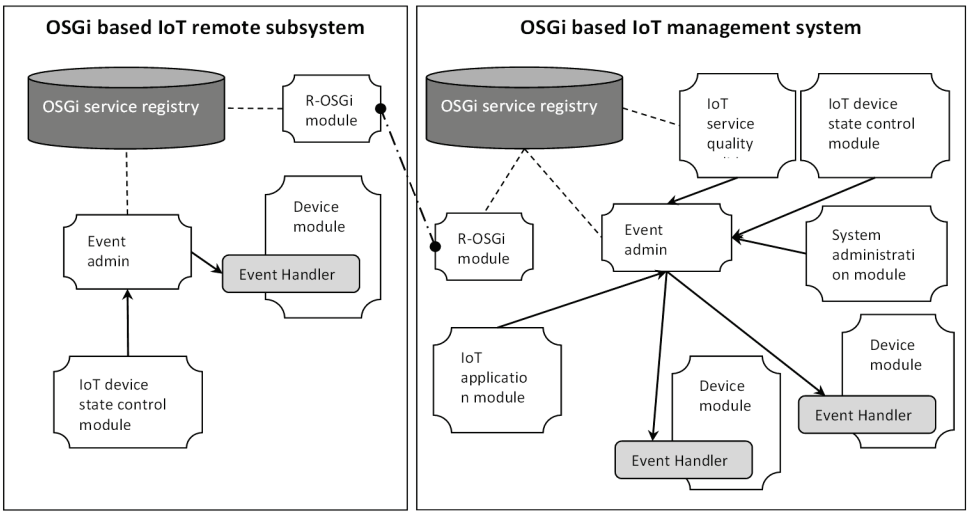

Figure 1. OSGi – based architecture of management system for IoT network

Source: own elaboration.

The significant drawback of OSGi platform is the lack of support for remote services. This problem was solved by an introduction of R-OSGi (Remote Services for OSGi) module provided by ETH Zürich [Rellermayer et al. 2007]. There are two significant functions of R-OSGi. The first one is related to a generation of dynamic proxies for remote invocation of services. The second one enables a registration of remote services from distributed registries in a local-service registry.

The reflection of current device state needs dynamical changes in service metadata. To enable such a functionality, an implementation of IoT device-state control module is required. This module will be responsible for sending auditing events and getting responses from dependent services, which need to be notified of the device status. Although OSGi allows for a specification of service metadata during service registration, still an universal semantics for handling device modules

is missing. However, a proposition of a complete semantics for IoT network is beyond the scope of this paper.

The IoT management system should enable the user to invoke the most suitable service for the realization of their request. However, the capabilities of OSGi to provide this function are limited, and this is the reason for additional implementation of IoT statistics and validation module. This module would enable the validation of the service properties and a delivery of proper statistics on service performance by measuring and logging such service properties as: execution time, last invocation time, number of invocations, number of generated exceptions and errors, etc.

To enable manual administration of devices in IoT network, the OSGi console may be extended by a predefined set of functions. The system administration module is responsible for device discovery based on service metadata and enabling revision or modification of the device state.

Several advantages and disadvantages of OSGi-based management system may be enumerated. The following drawbacks of OSGi-based architecture were identified. First disadvantage is a lack of built-in support for distributed services, as OSGi was produced to foster implementation of modular software running on one device, and distributed environments were not taken into account. Even if the R-OSGi initiative partially solves the problem, still the control over remote services is limited and there is no control over distributed bundles. Second, the service registry has limited capabilities, as the complete information about the services may be provided only in the form of java.util.Dictionary metadata. Such a solution limits the way of providing some additional information (concerning service quality or device state). Next, there is no shared semantics. A proposition of standardized semantics to foster unequivocal communication among heterogeneous systems and the services is required. Finally, the modules are Java-dependent. The OSGi framework was developed for Java platform and requires extension software to be implemented in Java. Since a typical IoT network consists of heterogeneous devices, the support for other programming languages should be assured.

Even if we enumerated evident disadvantages of the OSGi platform, numerous advantages of this framework may be listed as well. First, OSGi allows for code sharing and encapsulation. The code may be shared among OSGi bundles, which provides a possibility to limit memory overhead. Moreover, it is possible to provide encapsulation (separating service interfaces from their implementations) by specifying the code to be shared within bundle properties. Second, OSGi offers dynamic modules. The bundle lifecycle management enables changing the system functionality in runtime. This OSGi function is especially beneficial, as IoT network is very dynamic with devices use to appear and disappear. Third, the support for event based communication enables to separate service invocation from its implementation. What is more, the events may transmit natural-language orders, which are interpreted by event handlers. Such approach supersedes direct method invocation. Additionally, requesters need not to be aware of the service location,

because event administrator broadcasts the events to all event handling services. Next, it is possible to provide composed services based on simpler ones. Such orchestration may be performed by addressing specific features, extended by efficient event-handling mechanism to identify the most appropriate services to serve given request and context. Finally, an interface for manual administration may be easily implemented, as the OSGi management console may be extended by user-defined commands. Such a possibility enables a fast implementation of IoT device-administration console.

4. Managing IoT and sensor networks by the means of REST

The Representational State Transfer (REST) style is an abstraction of the architectural elements within a distributed hypermedia system [Fielding, Taylor 2002]. REST ignores the details of component implementation and protocol syntax in order to focus on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements.

A resource is a basic REST entity, standing for any named piece of information, being a target of a hypertext link. Each resource is identified by a unique Uniform Resource Locator (URL address), to fetch a value from a server: either static (if the resource is a file, piece of text, an image, etc.), or dynamic, being a result of an invocation of a piece of program code. In the latter case, the resource is treated as a "black box" from the point of view of its caller.

REST resources are frequently used to access the functionality of IoT devices, as this lightweight technology is well suited to limited hardware and software resources of the devices. Also, REST servers are used to proxy such access for very limited and non-standard devices [Rykowski et al. 2010]. Due to their advantages, especially the efficiency and simplicity of implementation, REST resources seem to be a better base than classical SOAP-based SOA services. Some disadvantages of REST approach, such as limited security due to open-form of URL addressing, are not crucial for closed and specialized environments, such as most of IoT networks.

Although REST is a very useful proposition for the implementation of IoT-based framework, this technology must be substantially extended to meet the requirements presented in Section II. Here we propose such extensions as a generic REST-based architecture for the Internet-of-Things environment, namely self-management of REST servers, and efficient searching method based on device functionality and context.

The key requirement for the IoT system is an efficient method for the selection of device(s) for the incoming request in a given context. To this goal, the system must (1) know the devices' possibilities and characteristics, including their geo-locations (range of impact), (2) be able to search for the optimum device(s) to fulfill given request, and (3) activate the just-searched device in order to provide certain functionality. As already mentioned, these are basic tasks for a centralized catalogue.

However, such catalogue is not a part of REST technology. Thus, we propose a uniform REST-based architecture with a dedicated catalogue (also REST-based) as the base for IoT system (Figure 2).

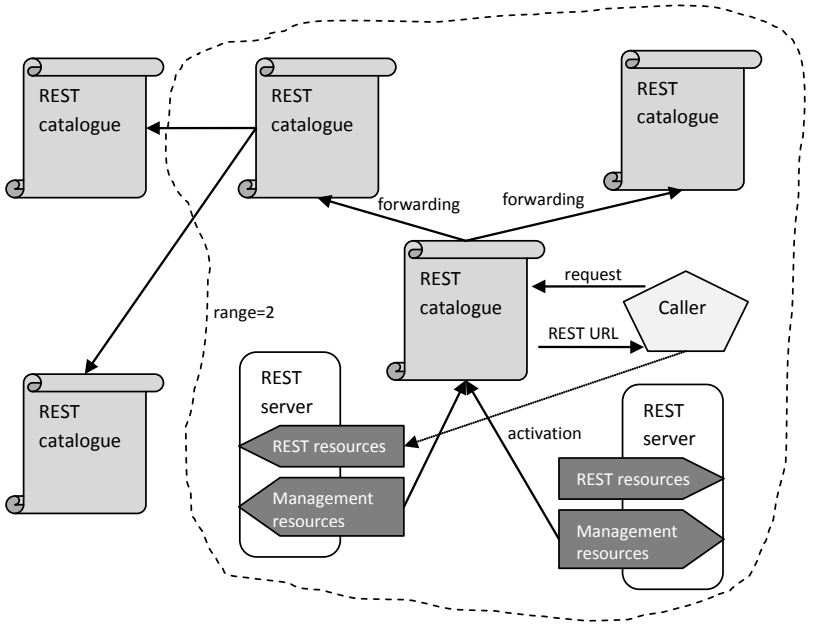

Figure 2. Basic architecture scheme for IoT REST-based environment

Source: own elaboration.

The architecture is based on two principles: extending REST servers by some management resources, observed as additional REST resources by the catalogue(s), and providing mapping of semantically-expressed requests to URLs of REST resources. To implement the first goal, we assume that each REST server is equipped with certain (predefined) resources to control and supervise the server, including:

- "management" resource to provide an access to some basic commands such as quit/suspend/resume/restart,
- "statistics" resource to provide some information about server usage (both divided to particular resources related with this server, as well as the server as the whole), for example: average service timings, number of requests served, number and description of invocation errors, etc.; this resource also provides an information about the real device (if any), connected to the resource and using this resource as its own proxy; in such a way, the catalogue may be informed whether the device is accessible or temporary unavailable,
- "functionality" resource to provide some knowledge for the functions possibly served by this server (namely, by its resources); such function descriptions are

defined according to common semantics for the whole system (cf. interpretation of the request below).

Each REST server is obliged to register itself in any catalogue, providing its own URL locator. The locator (more precisely – the resources mentioned above) may be periodically inspected by the catalogue to collect up-to-date information about server state. This information is used to search for ready-to-use devices (cf. a description of request serving later on). Once the internal state of the REST resource is changed, e.g., according to respective change of the state of real device connected to this resource, the server re-registers to the catalogue with the updated information. The catalogue may be arbitrary chosen by the server, however, usually it is the "closest" one (i.e., the one operating in the same sub-network).

Imagine one wants to activate certain function of the system. To this goal, he/she must address the catalogue with the semantic description of the desired action. This description is compared with the possibilities of the devices (however, only those declaring their state as currently accessible), and a device is chosen to meet the criteria. The caller obtains the locator of the resource linked with desired activity/device, to directly address respective REST service and, indirectly, the device. Note that the called URL was not known by the caller in advance, as it was (possibly dynamically) generated and sent by the catalogue. Once the situation is changed, some other resource/device may be activated according to similar request. Note also that the semantics of the URL locator of the resource to activate is not known to the caller, thus the details of the activation may be hidden towards the users of devices' functionality. This approach greatly improves portability of the system usage, on condition the semantics of the requests is common for all the callers and catalogues.

If given request is to be possibly served by several devices, the catalogue may choose one device based on context and statistical information collected from the management/statistical resources of the corresponding REST service. For example, one may address the less-overload resource, last-activated or most-unused device, the one with the shortest response time, etc. Surely, all the devices-in-range may be activated in parallel, e.g., "close window" command will result in closing all the windows, as all the available devices related with windows will be identified by the request and possibly activated one by one.

We may also propose, instead of accessing a single resource for a single request, activating a set of somehow inter-related resources – i.e., an orchestration of resources. To this goal, a mechanism is needed to map the request semantics to some program code, in turn responsible for the pipelining of the resources. The final result is provided as if all the activated resources are a single resource, thus the whole orchestration mechanism is transparent to the caller. The orchestration process is based on a specialized language being an extension of XML standard and OWL ontologies [Rykowski et al. 2010] – Ontology Scripting Language (OSL). Basic statements of OSL language are similar to those used for shell programming in a Unix-based system (choice statement IF-ELSE, looping – WHILE, compound

statement – sequence interpretation, calling a subroutine – CALL, defining and accessing a variable, etc.). As may be seen, orchestrating devices by means of OSL is related to imperative rather than declarative mode of programming, although dynamic searching for "the best" device goes beyond this mode – a programmer never knows which device (and if any) will be able to fulfill the request.

In most of the applications, connecting all the devices to a single host is not possible, due to (1) limited number of external connectors (such as USB), and (2) natural need for the distribution of the devices across a wider area. Thus, it is desirable to distribute not only the devices and hosts (proxies), but also the parts of the controlling framework. So far we assumed that there is only one central point for the control of the whole network – REST catalogue. However, due to unrestricted distribution of REST resources it is possible to part this centralized point to a hierarchy of interconnected sub-parts (sub-catalogue), each one providing the control over certain network part. This distribution of REST-catalogues is depicted in upper part of Figure 2. To keep the control on the network as a whole, we propose to connect the sub-controllers as a graph and to span the controlling in the same manner as it is used to synchronize the resources of any peer-to-peer (P2P) network, with arbitrary restricted nesting level.

A device may choose any of the sub-controllers to register with. Then, this device is manageable locally by this sub-controller in the direct way described above. Similarly, all the local requests served by this sub-controller towards all its devices are processed locally. However, if there is a need to access remote (from the point of view of this controller) devices, then the sub-controller forwards the request to all its neighborhoods. In turn, if a neighbor is not able to process the request, forwards it to all its neighbors except the one which initiate the request, and so on. For each forwarding step, nesting level of the request (a range) is increased. While this level reaches certain value, the forwarding is stopped. Returning to Figure 2, the request marked as "level=2" is forwarded only to three sub-controllers, while two "far" ones remain untouched. The stopping value is declared for each sub-controller by its administrator, and for the request by the initiator of this request – each time smaller of these two values is taken into consideration.

All the responses are collected by the forwarder of the request, and, if the request level is still greater than one, they are sent as a common response to the caller. Finally, the originating node collects all the responses of all its neighbors, acting as a "global" response to the initial request.

To limit the possible cycles in the forwarding of the request, each request is identified, and the past-request identifiers are collected for some time in each of the sub-controllers. Once a newly coming request was already served in the past, this call is disregarded and no more forwarded. Thus, even if the graph of interconnected sub-controllers contains cycles, these cycles are detected and never block the system.

In the same way we may obtain some global information about the network (statistics, information for certain-device of function availability, etc.), more

precisely – about the local neighborhood ("no longer than N connections from the selected node"). The more global is the request, the longer we must wait for the response, similar to typical P2P behavior.

As a typical IoT environment usually covers rather small geographical area (such as a room, building or a public place –market, shop, museum, etc.) – by restricting the level of spreading the requests to reasonable value one also limits the overall network traffic to the reasonable level, and the response delay is counting in parts of the second. We may also imagine restricting the bigger levels to those with special access rights, such as system administrators – for most of the requests, these will be addressed to local devices (level equal to 1). Then, both the possible delays and increased network traffic are not a sharp problem.

We may enumerate both the advantages and disadvantages of the proposed REST-based architecture, which are presented in this paragraph. The disadvantages are mainly related to two global observations – independence of the program code for the REST servers, and the need for shared semantics. First, there is no possibility to share the code among REST servers even if identical parts of the program (i.e., the libraries) are used by several devices. This feature results in large memory usage and may be relaxed, within the same computer, by some shared-code techniques such as Dynamic Library Linking (DLL). Second, all the resources must know in advance the address of the catalogue, to register with. This restriction may be relaxed with non-standard usage of DHCP broadcasting, UDP cyclic broadcasting, or, in case of personal mobile equipment operated directly by humans, at-the-place registration by means of NFC/QR-code identification tags, etc. Third, additional network traffic is observed to monitor device availability in real-time. However, with reasonable polling interval (for most of the systems such timing as 5–10 seconds is completely enough) this restriction may be bypassed. Fourth, one must provide strict definition of the semantics of requests and device activities (functions), shared for the whole system. This restriction is related to the need for the strict format of URL locators for additional REST resources (management/statistics). However, as this traffic is not observed by the end-users, this is a problem only for system designers. And last but not least, the activations of devices' functions are based on URL locators of the corresponding REST resources. Thus, only limited parameterization of such calls is possible – all the details must be coded and thus somehow hidden as URL locator parts. However, this is also mainly the problem of system designers, as end-users have no knowledge about the semantics of the device activations. Moreover, sometimes such information should be intentionally hidden for the end-users to limit the direct access to the devices (i.e., the access not controlled by the catalogue).

The advantages of REST-based approach outweigh the above-presented restrictions and problems. First, it is a uniform approach – the whole traffic is realized as REST-compliant calls, which strongly facilitates system implementation. Second, small resources and servers dominate across the system, thus the consumption of computer resources such as CPU time is reasonably small. Our experiments showed

that even hundreds of such servers executed on a single PC cause no problems, as a single resource typically consumes less than a few percent of computer resources. Third, there is no problem of the synchronization of some resources "shared" among many REST servers, for example real devices distributed across a larger area, communication gateways, etc. Such synchronization is needed only for a single REST server, however, as this server is programmed as a single entity and probably by a single programmer or small group of designers, such synchronization is usually quite easy to achieve. Fourth, all the system parts (both REST resources/servers and IoT devices) may be arbitrary distributed in a local- or even wide-area network – on condition the distributed servers know and may access the catalogue host. Fifth, REST servers may operate even with very limited hardware, also built-in to the networked devices. Sixth, there is theoretically unlimited possibility of the orchestration of devices – providing "virtual devices" acting as real ones, however, possibly much more complex and powerful, at the same time – more easy to utilize as "black boxes" of certain functionality. Seventh, the catalogue represents up-to-date information for device availability – continuous monitoring is undertaken not only for device state, but also some statistics for its usage. The system updates the information, e.g., periods of device (un)availability, every few seconds, which is usually sufficient for a typical IoT network. And finally, single- and group-based management for devices and their corresponding resources is possibly achieved, including server start, quit, restart, suspending/resuming, also GUI-based individual administration.

5. Coupled architecture for IoT and sensor network management

Both OSGi and REST-based architectures are suitable for the management of IoT and sensor networks. However, both REST and OSGi-based approaches require significant extensions. As the OSGi has no built-in support for handling distributed services, this framework seems to be better suited for sensor networks, where rather homogeneous sensors are connected with gateways. Moreover, the possibility to dynamically start and run services or bundles and to share code libraries fits sensor networks management as well. As for REST protocol, running a separate REST server for each device without the possibility to share code libraries causes memory overhead, which is an undesirable feature for sensor network management system. However, for heterogeneous devices in a typical IoT network, the code sharing possibility is not crucial.

After the analysis of advantages and disadvantages of both OSGi- and REST-based approach, a smart mixture of these two frameworks seems to be natural solution to bypass the disadvantages of previously discussed solutions. We think that an OSGi-based framework may effectively manage local, homogeneous sensor networks. To operate in a distributed environment, local OSGi-served networks may be shared with REST services, with the services of each sensor network provided

outside these networks via RESTlet server module integrated with OSGi environment. Then, the services of sensor network may be accessed by REST clients from REST-based management system. The coupled REST- and OSGi-based architecture for the management of IoT networks is presented in Figure 3 and described below.

In the proposed architecture, there are two layers of abstraction: the lower one, which is OSGi – based and the higher one, which is REST – based. Figure 4 depicts the layers hierarchy and associated frameworks.

Within the lower level layer, the sensor network gateways are connected to the local server. The OSGi mechanism running on this server allows for dynamic management of devices: starting, stopping, suspending, installing, uninstalling, etc. The code libraries required to interact with sensor network elements are shared among bundles. Plugging of any additional gateway for the sensor network or a device will require at most an installation and starting an additional bundle, while the system remains continuously operational.

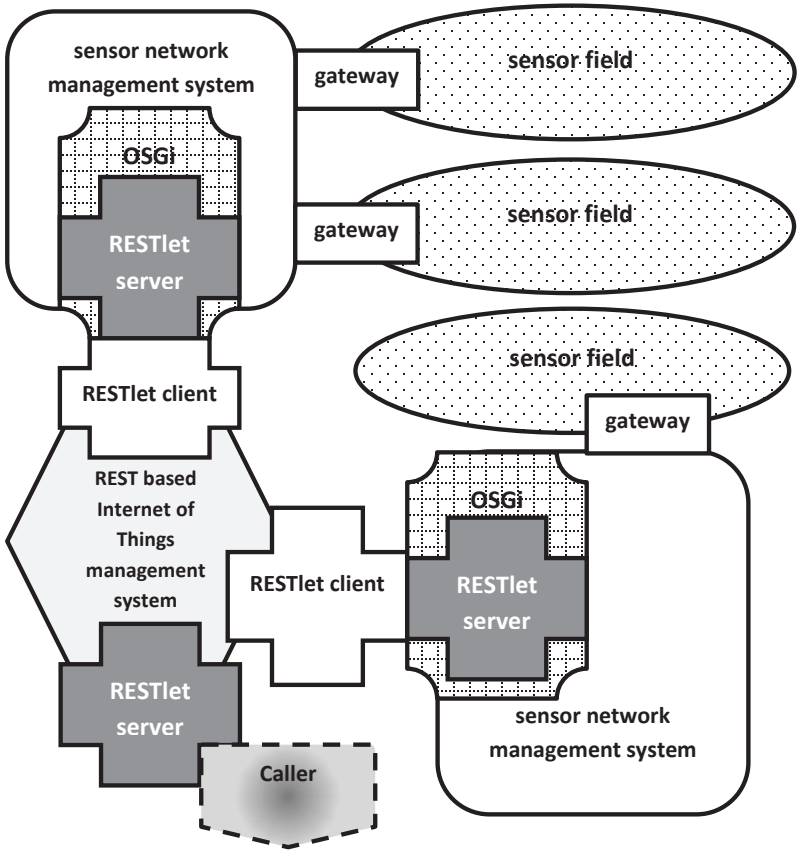

Figure 3. Combined OSGi and REST – based architecture of management system for IoT networks

Source: own elaboration.

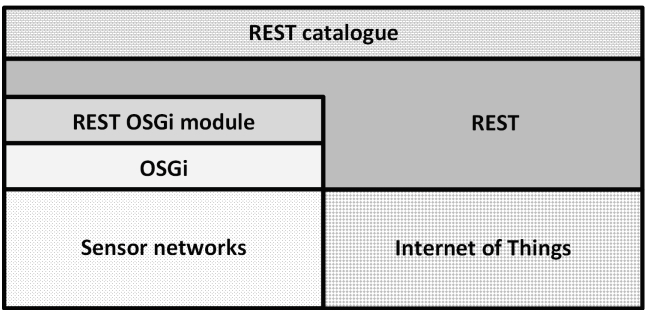

Figure 4. IoT management system layers

Source: own elaboration.

The RESTlet module will act as a proxy abstracting functions offered by sensor networks. As the services of sensor networks are rather basic ones, the built-in registry and *Dictionary* objects are sufficient to orchestrate these networks. For local device-administration purposes, the OSGi console provides sufficient functionality. However, there is still a need to implement service-quality monitoring module, which is the task of upper-level REST-based management services.

The proposed solution allows for a remote access to the sensor network services by means of the REST protocol. In this approach, higher level services provided in the IoT layer are orchestrated by REST catalogue(s). The mechanism described in details in Section 4 is responsible for IoT services discovery, composition and orchestration. However, each local sensor network is easily managed by means of specialized OSGi framework.

6. Conclusions

The analysis of OSGi- and REST-based systems described in Sections 3 and 4 showed that similar amount of work is required to implement an Internet-of-Things middleware with these systems as the basis. However, for both architectures substantial extensions must be proposed to adapt these environments to the specific requirements of IoT networks. We have noticed that OSGi-based approach is better suited for sensor networks (the applications covering homogeneous devices and fixed, predefined system functionality, usually related to a single location and local network), and the REST-based framework is more useful in *ad-hoc*, dynamic environment achieving heterogeneous devices and services, also in a distributed environment. In order to bypass the disadvantages of both approaches, the mixed OSGi- and REST-based architecture was proposed. There, the OSGi framework is responsible for the management of devices at sensor-network level, while REST-based system provides proxies and orchestration for distributed, heterogeneous devices, to provide complex services required by an IoT network.

References

Brock D.L., 2001, *The Electronic Product Code (EPC) A Naming Scheme for Physical Objects*, Auto-ID Center, http://www.autoidlabs.org/uploads/media/MIT-AUTOID-WH-002.pdf.

Castro Alves de A., 2011, *OSGi in Depth*, Manning Publications, Greenwich.

Fielding R.T., Taylor R.N., 2002, *Principled design of the modern Web architecture*, ACM Transactions on Internet Technology, vol. 2, no. 2, ACM, New York, pp. 115–150.

Flotyński J., Krysztofiak K., Wilusz D., 2013, *Building modular middlewares for the Internet of Things with OSGi*, [in:] Galis A., Gavras A. (eds.), *The Future Internet*, LNCS, vol. 7858, Springer-Verlag, Berlin, Heidelberg, pp. 200–213.

Hall R.S., Paulus K., McCulloch S., Savade D., 2011, *OSGI in Action: Creating Modular Applications in Java*, Manning Publications, Greenwich.

Haller S., Karnouskos S., Schroth Ch., 2009, *The Internet of Things in an enterprise context*, [in:] Domingue J., Fensel D., Traverso P. (eds.), *Future Internet – FIS 2008*, LNCS, vol. 5468, Springer-Verlag, Berlin, Heidelberg, pp. 14–28.

OSGi Alliance, 2014a, *OSGi™ Service Platform Release 4 Version 4.2*, http://www.osgi.org/javadoc/r4v42/.

OSGi Alliance, 2014b, *Technology/HomePage*, http://www.osgi.org/Technology/HomePage.

Rellermayer J.S., Alonso G., Roscoe T., 2007, *R-Osgi: Distributed Applications through Software Modularization*, [in:] Cerqueira R., Campbell R. H. (eds.), *Middleware 2007*, LNCS, vol. 4834, Springer-Verlag, Berlin, Heidelberg, pp. 1–20.

Rykowski J., Hanicki P., Stawniak M., 2010, *Ontology scripting language to represent and interpret conglomerates of IoT devices accessed by SOA services*, [in:] Ambroszkiewicz S., Brzeziński J., Cellary W., Grzech A., Zieliński K. (eds.), *SOA Infrastructure Tools: Concepts and Methods*, Wydawnictwo Uniwersytetu Ekonomicznego w Poznaniu, Poznań, pp. 235–262.

Rykowski J., Wilusz D., 2014, *Comparison of architectures of service management in IoT and sensor networks by means of OSGi and REST services*, [in:] Ganzha M., Maciaszek L., Paprzycki M. (eds.), *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems*. *Annals of Computer Science and Information Systems*, vol. 2, Polskie Towarzystwo Informatyczne, Warsaw, Institute of Electrical and Electronics Engineers, New York City, pp. 1207––1214.

Waltenegus D., Poellabauer Ch., 2010, *Fundamentals of Wireless Sensor Networks: Theory and Practice*, John Wiley & Sons, Chichester.

Weiser M., 1991, *The computer for the 21st century*, Scientific American, vol. 265, pp. 94–104.

Wilusz D., Flotyński J., Sielicka M., 2013, *Supporting experimentation in a food research laboratory with the Internet of Things*, PhD Interdisciplinary Journal no. 3/2013, Gdańsk University of Technology, Gdańsk, pp. 113–119.

Wilusz D., Rykowski J., 2013, *The Architecture of Coupon-based, Semi off-line, Anonymous Micropayment System for Internet of Things*, [in:] Camarinha-Matos L.M., Tomic S., Graça P. (eds.), *Technological Innovation for the Internet of Things IFIP AICT*, vol. 394, Springer-Verlag, Berlin–Heidelberg, pp. 125–132.

ORKIESTRACJA ROZPROSZONYCH, HETEROGENICZNYCH ŚRODOWISK W SIECIACH SENSOROWYCH I INTERNECIE RZECZY

Streszczenie: Artykuł skupia się na zarządzaniu sieciami sensorowymi oraz środowiskiem Internetu Rzeczy. Dwie alternatywne architektury systemów zarządzania usługami w sieciach sensorowych oraz Internecie Rzeczy zostały poddane analizie. Pierwsza architekura bazuje na platformie OSGi (*Open Service Gateway*) oraz module *Remote Services for OSGi*, rozszerzającym OSGi o możliwość komunikacji z rozproszonymi usługami. Druga architektura rozszerza wzorzec REST (*Representational State Transfer*). Analiza powyższych architektur pokazuje, że oba rozwiązania spełniają wymagania stawiane przez środowiska sieci sensorowych oraz Internetu Rzeczy. Jednak w celu zlikwidowania wad występujących w obu podejściach proponujemy nową architekturę, w której platforma OSGi jest wykorzystana do zarządzania urządzeniami w lokalnych sieciach sensorowych, natomiast wzorzec REST wykorzystany jest w celu zarządzania złożonymi usługami Internetu Rzeczy, także w środowisku rozproszonym. W celu integracji platformy OSGi z architekturą bazującą na wzorcu REST zastosowano specjalizowany moduł RESTlet dedykowany dla platformy OSGi.

Słowa kluczowe: Internet Rzeczy, Internet Przyszłości, sieci sensorowe, REST, OSGi.