Adaptive Runtime Middleware: Everything as a Service

Achilleas P. Achilleos[1], Kyriaki Georgiou[2], Christos Markides[1], Andreas Konstantinidis[1] and George A. Papadopoulos[2]
[1] Frederick University, 7 Y. Frederickou Str., Nicosia, Cyprus
{com.aa, com.mc, com.ca}@frederick.ac.cy
[2] University of Cyprus, 1 University Avenue, Nicosia, Cyprus
{george, kgeorg04}@cs.ucy.ac.cy

Abstract. The Internet of Things applies and has a large impact on a multitude of application domains, such as assistive technologies and smart transportation, by bringing together the physical and virtual worlds. Due to the large scale, the extreme heterogeneity and the dynamics of the IoT there are huge challenges for leveraging the IoT within software applications. The management of devices and the interactions with software services poses, if not, the greatest challenge in IoT, so as to support the development of distributed applications. This paper addresses this challenge by applying the service-oriented architecture paradigm for the dynamic management of IoT devices and for supporting the development of distributed applications. A service-oriented approach is a natural fit for both communication and management of IoT devices, and can be combined logically with software services, since it is currently the paradigm that excels and dominates the virtual domain. Building on our past and ongoing work on middleware platforms, this work reviews middleware solutions and proposes a service-oriented middleware platform to face IoT heterogeneity, the interactive functionality of IoT and promote modular-based development to scale as well as provide flexibility in the development of IoT-based distributed applications.

Keywords: middleware, IoT, services, mobile devices, distributed applications.

1 Introduction

During the last decade, key trends have been observed in the world of embedded devices, which refer mainly to miniaturization, increased computation, cheaper hardware and the shift of software approaches towards service-oriented integration in the Internet of Things (IoT). On the basis of the stated-by-many vision for the IoT, the majority of the devices will soon have communication and computation capabilities that they will use to connect, interact, and cooperate with their surrounding environment [1], [2], including other devices and services. Business-oriented complex distributed applications are being developed on the basis of composition and collaboration among diverse services, in many cases across different vendors.

The Internet of Services (IoS) vision [3] assumes this on a large scale, where services reside in different layers of the enterprise, IT networks, or even running directly on devices and machines of the company [4]. As the Internet proved its merit both for

content and services, we are now facing a trend where service-based information systems blur the border between the physical and virtual worlds, offering a fertile ground for a new breed of real-world aware distributed applications. Therefore, for the success of the IoT, future research, vision and business ecosystems require a merge between cloud computing and the IoT, by enabling a model of "Everything as a Service".

Such a model will deliver an IoT paradigm, where applications rely on the cooperation between heterogeneous devices and software applications, all of which are offered as dynamic web services. Thus, functions such as dynamic discovery, query, selection, and provisioning of web services will be needed for facilitating access and interaction between real-world objects (i.e. devices) and virtual objects (i.e. software services) [4]. The future Internet will provide the capability for embedded heterogeneous real-world entities, similar to virtual entities, to offer their functionalities (e.g. provisioning of sensor data) as RESTful/Web APIs [6]. This will enable virtual entities (i.e. enterprise services) to interact with real-world entities since both will be offered as services in a realisation of the "Everything as a Service" model.

The added value brought by real-world services, (i.e. services) provided by embedded systems that are linked to the physical world, is the increased efficiency of the decision making process due to the fact that they offer real-time data about the world. Thus, the critical issue about such a model is that embedded heterogeneous devices will be able to offer their functionality as web services, which can be used by applications, other services, or even other devices. In this case, device drivers will not be needed anymore and a new level of efficiency will be achieved as web service clients can be generated dynamically at runtime [4]. This will result in a mashup of services where horizontal collaboration between devices will be possible, as well as vertical collaboration of devices with software services and enterprise applications that provide correspondingly interaction capabilities with people [5].

In related work [4], [7], the key challenges continue to be open and need to be addressed [8], such as providing topology dynamics, high scalability and overcoming heterogeneity in such a dynamic IoT environment. In fact, such a highly dynamic environment is also further augmented by the fact that peoples' needs evolve over time, so a scalable and reliable IoT environment needs to be designed with inherent built-in modularity, flexibility and a variety of components in order to meet diverse individual situations and to remain attractive to end-users over time. The following list outlines the key unresolved IoT challenges [8]:

- **Heterogeneity:** Sensors and actuators are the main actors in an IoT environment, where due to the highly heterogeneous nature of IoT devices, enabling interoperability is a complex task.
- **Scalability:** To accurately represent the real world, a sensing/actuating task will more often require the cooperation and coordination of numerous things.
- **Flexibility:** Different configurations may be required for different situations.

A service-oriented approach that follows the concepts of the Internet can provide a solution to the above mentioned challenges. Web technologies provide the base on which a service-oriented approach for the IoT can be formulated to properly address these restrictions. This enables different devices and software components to work together by ex-posing their functionalities to others as web services. Services are the

entities that enable users to access the capabilities through pre-defined interfaces in accordance with the policies and constraints, which are part of the description of that service [10]. Web services are platform-independent and can be accessed through the Internet. The original contribution of the web was as a content-provisioning medium. Today, the key role of the web is to act as a facilitator in service outsourcing [2]. This role enables businesses to collaborate dynamically, thus reducing overheads. Therefore, the service deployment model can be applied to any component, physical or virtual, so as to make it available as a service [2], [9].

In the IoT, there is huge heterogeneity in both the communication technologies, and the system level technologies. Therefore, apart from service-oriented computing, a middleware system can support heterogeneity of both communication and system-level technologies that are diverse and many in the world of the IoT. In general, a middleware abstracts the complexities of the system or hardware, allowing the application developer to focus all his effort on the task to be solved [10]. In fact, a middleware system offers a software layer between applications, the operating system and the network communications layers. Based on the above, it is evident that a middleware system can offer an abstraction layer for handling the complexities of the IoT and addressing the challenges faced in developing such applications.

This work aims to support the development of dynamic, flexible and distributed IoT applications, by combining service-oriented computing and middleware technologies. The proposed service-oriented system, coined Adaptive Runtime Middleware (ARM), allows addressing heterogeneity, scalability and flexibility issues. The RESTful architectural pattern is adopted, which enables upon deployment of smart devices (e.g. sensors, actuators), to discover these devices, detect their capabilities, regenerate and redeploy the middleware injecting new RESTful service interfaces (i.e. APIs).

ARM's key capability is the annotation-driven runtime code generation that drives dynamic injection of device capabilities in the form of new service interfaces. The ARM's self-adaptive nature enables interoperability amongst heterogeneous devices, automates device discovery and management, and exposes these devices as services. This offers simplicity for the developer, without introducing additional technologies that increase the already profound complexity in the IoT. In fact, the developer will only need to base its client application implementation on the generated documentation, which describes the generated services that enable management of the IoT devices.

2 Related Work

2.1 Context-Aware Middleware

Several approaches and research work has been performed for the development of middleware systems in different research domains. The Cooltown project [11] supports wireless mobile devices in interacting with a web-enabled environment by assigning URLs to devices, people and things as a web-presence identifier – providing therefore a "rich" interface to the entity. Middleware systems include the Gaia [12] that aims to provide a distributed functionality similar to an operating system, the MiddleWhere

[13], which provides enhanced and enriched location information to applications by utilizing a number of location sensing techniques based on a location model, and the MobiPADS [14] that targets mobile environments and its services are provided through various migrated entities from different MobiPADS environments.

A context-aware middleware is also developed in the MUSIC EU project [15], which is a comprehensive open-source software development framework. MUSIC is an ubiquitous OSGi-based context-management middleware system for developing adaptive applications and services for ubiquitous environments.

2.2 Middleware for the Internet of Things

Several middleware IoT architectures and frameworks have been proposed, aiming for a more usable connection among, often, complex and already existing applications that were not originally designed to be connected. The essence of the IoT is making it possible for just about anything to be connected and communicate data over a network, where the middleware framework is part of the architecture thus enabling that connectivity among heterogeneous devices and software services.

An example of a scalable and modular architecture that integrates various components and technologies is openHAB [16]. OpenHAB is an open-source, agnostic automation software with an active community, which encompasses different home automation systems and technologies under the same umbrella of a single solution, enabling the user to define the interaction of systems and devices through automation rules and uniform user interfaces. It is also OSGi-based, and provides APIs for integration with other systems, where REST API is used for remote communication.

OpenIoT is an open-source middleware for connecting cloud sensors and collecting information, extending the IoT solution and exploring efficient ways to use and manage cloud environments [17]. Through an adaptive middleware framework, which is deployed on the basis of one or more distributed nodes, data are collected, filtered, combined and semantically annotated from virtual sensors or physical devices. The proposed middleware does not support though access via service interfaces.

2.3 Service-Oriented Middleware

The service-oriented design paradigm deals with the implementation of software or applications in the form of services by following the concepts and ideas of service-oriented computing (SOC). SOC benefits, such as technology neutrality, loose coupling, service reusability, service composability, and service discoverability [18], can be also beneficial to IoT applications. However, IoT's heterogeneity, scalability and flexibility make service discovery, deployment and composition challenging.

The Hydra EU research project set out to develop a middleware for Networked Embedded Systems. The Hydra middleware allows developers to incorporate heterogeneous physical devices into their applications by offering easy-to-use web service interfaces for controlling any type of physical device irrespective of its network technology such as Bluetooth, RF, ZigBee, RFID, WiFi, etc. As stated in [19, 20], the software middleware is based on Service-Oriented Architecture (SOA), which means that the

communication occurs transparently between the lower layers. The aim of the middleware, coined LinkSmart, was to support diverse and heterogeneous connected devices, which enable developers to implement applications that depend on and adapt to context information. *Services are defined statically* in the proposed middleware.

CHOReOS [21] is a service-oriented middleware that enables large scale choreographies of adaptable and heterogeneous services in IoT. It aims to address scalability, interoperability, and adaptability issues via *static service interfaces*. The SenseWrap service-oriented middleware combines Zeroconf protocols with hardware abstraction using virtual sensors [22]. A virtual sensor provides transparent discovery of resources, through the use of Zeroconf protocols, which applications can use to discover sensor-hosted services. SenseWrap also provides a standardized communication interface to hide the sensor-specific details from the applications.

3 The Adaptive Runtime Middleware (ARM)

3.1 Our Contribution

This paper builds on our research work on middleware systems and in research projects such as MUSIC, AsTeRICS and Prosperity4All, so as to design and develop an adaptive middleware system for the IoT. Such a system will have the ability when a smart module (e.g., sensor, actuator) is installed, to discover it, detect its capabilities, regenerate and re-configure the middleware. The middleware supports annotation-driven runtime code generation [23] of device capabilities in the form of *dynamic services*. The key aspect is simplicity for the developer, without introducing additional technologies, IDEs and platforms. The developer can use the generated service interfaces to manage devices, and thus create cross-platform distributed applications (e.g. Android, iOS, HTML5).

3.2 ARM Architecture

The proposed middleware takes advantage of the principles of RESTful architectural pattern and exposes devices as services. The functionalities of each device (e.g., Smart Light – turn light on, dim light) are implemented as annotated Java functions available within each device-specific OSGi component. The key idea is that OSGi components correspond to IoT devices, which can be accessed and managed using RESTful interfaces. In addition, there are two main services implemented as OSGi components, which refer to the middleware core functionality and the REST server that hosts the device resources. The middleware core functionality detects the device capabilities via the annotations in the OSGi component that is installed and allows generating the service interfaces that the REST server component exposes, which correspond to the functionalities of the installed device.

The smallest components in the architecture are the individual OSGi bundles. Each one of these bundles implements the device capabilities, which could be as simple as turning a light on/off, or could be as complex as the interactions between multiple

actuators and sensors. The middleware core functionality detects the capabilities of newly installed bundles, thus generating the RESTful interfaces that expose and enable access to these capabilities.

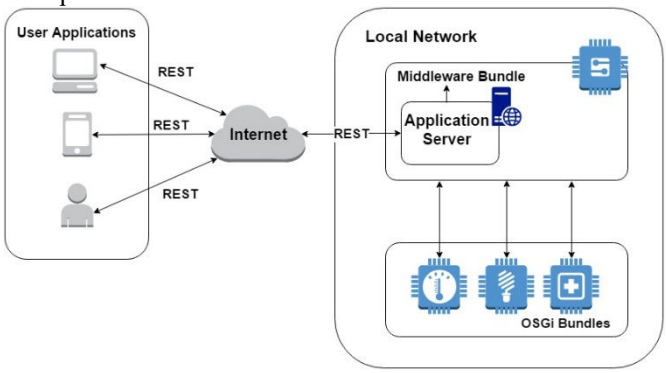

Fig. 1. ARM middleware architecture.

The communication between the components of the proposed adaptive runtime middleware system is illustrated in Figure 1, where the architecture of the middleware system. The middleware bundle contains the application server that enables communication with the installed bundles via the generated service interfaces. Furthermore, the REST-based architecture enables to access devices over the network in distributed end-user locations (e.g., home, office). The developer is able to develop client applications that make use and even allow interaction between devices in distributed locations, since the service interfaces can be accessed seamlessly via the middleware system available in these locations.

3.3 ARM Implementation

The dynamic middleware is realized as an OSGi bundle, which utilizes the benefits of the OSGi specification for enabling the modularity and scalability of the system. The implemented middleware is built on top of the OSGi Equinox framework, used also in the Eclipse IDE, which is actually an implementation of the OSGi specification. In addition, the REST architecture satisfies and offers solutions in terms of the flexibility needed in an IoT environment. For the implementation of the REST OSGi bundle, the Java API for RESTful Web Services (JAX-RS) specification and its analogous Jersey implementation were used. The OSGi-JAX-RS Connector (i.e., Staudacher) was used, since it packages the Jersey implementation in the form of a bundle and thus integrating consistently the Jersey and OSGi frameworks.

Apart from the core bundles, the middleware and REST server, each device or software service can be defined in a separate OSGi bundle. For instance, a WiFi smart socket can be implemented as an OSGi bundle. This approach offers many advantages since it enables above all flexibility, heterogeneity and scalability as new devices and software services can be supported. The requirement is that developers create a new bundle that enables communication to the device or software service.

Java annotations are syntactic metadata that can be added in the code. Hence, when a new bundle is installed, the middleware detects and starts the component, parses the annotations of public methods and generates the service interfaces that enable direct access to the new resource. The middleware will also generate the documentation for the service, based on the annotations of each public method defined in the bundle. These annotations define the functionality of the bundle, the signature of public methods including the input and output parameters of the method. This mechanism is exploited to enable runtime code generation of the service interfaces.

Descriptor and Annotations

The developer of each IoT device bundle needs to follow a set of guidelines, in order to utilize the adaptive runtime functionality of the middleware. Each bundle can be implemented and exported as a JAR file, which contains a descriptor (i.e. XML file) and the implementation classes. The descriptor defines only the full name of the implementation class for the bundle. This refers to the package followed by the symbolic name for the bundle as defined in the component manifest, in the form of: *"<Exported-Package>.<Bundle-Symbolic-Name>"*. For instance, if the package is *phillipshue* and the symbolic name is *SmartLight*, then the full name will be *phillipshue.SmartLight*. The developer should use Java annotations on top of the public methods for documenting the functionalities provided by, e.g. the SmartLight, which are used to generate the service interfaces as defined next.

Service Interfaces

The generated service interfaces need to be consistent and adhere to a simple resource path definition logic, which enables developers of client applications to easily access, and learn how to invoke and thus make use of device functionalities. Table 1 presents the generic definitions for the service interfaces that provide access to device or software services. These refer to the resource paths automatically generated by the middleware.

Table 1. Middleware path hierarchy for generated service interfaces.

Path	Description
<baseURL>	Lists information on the available bundles, including description and interface definition. The *baseURL* represents the service interface of the middleware.
<baseURL>/<BundleName>	Provides information on a specific bundle. *BundleName* is the middleware name for the bundle. The *BundleName* can be retrieved by invoking the *baseURL*.
<baseURL>/<BundleName>/<MethodName>	Invokes functionality by the *MethodName*, which is appended after the *BundleName* (no parameters).
<baseURL>/<BundleName>/<MethodName>/<Parameter-Data>	Invokes functionality by the *MethodName*, which is appended after the *BundleName* (accepts parameters).
<baseURL>/<BundleName>/<MethodName>/def	Presents information related to the functionality offered by this method of the specified bundle.

4 Smart Light Use Case Demonstrator

The use case demonstrator introduced in this section presents the installation of the bundle, as well as accessing, communicating and controlling the Philips Hue smart light. In this use case scenario, an HTML5 client is implemented and used for demonstrating the middleware capabilities. The bundle implementation offers access to four device capabilities: 1) turn light on, 2) turn light off, 3) dim light and 4) set light level. First the bundle descriptor needs to be defined by the bundle developer as follows:

```
<bundle-definition>
  <fullname>
    osgi_SmartPhilipsHUELight.SmartPhilipsHUELight
  </fullname>
</bundle-definition>
```

Figure 2, presents a fraction of the code that showcases how annotations are defined for the “turn light on” method implementation of the Phillips Hue. The next step involves exporting the bundle. The middleware will then parse the descriptor containing the bundle’s full name. If the bundle is not already installed, the middleware will automatically install and start it. Using reflection, the middleware detects all device capabilities, re-configures the middleware via runtime code generation and injects/publishes the discovered functionalities as RESTful service interfaces.

```
1 package osgi_SmartPhilipsHUELight;
2
3 import osgi_annotations.interfaces.*;
4
5 @ClassDescription(value = "SmartPhilipsHUELight is an OSGi bundle for communicating with a Philips HUE Light. "
6         + "This OSGi bundle implements four basic functionalities for interacting with the smart light: "
7         + "\t* Turn Light On\t* Turn Light Off\t* Dim Light\t* Set Light Level")
8
9 public class SmartPhilipsHUELight {
10
11     @MethodDescription(value = "TurnLightOn method turns the light On. "
12         + "Returns true if the light is turned On, otherwise false.")
13
14     public boolean TurnLightOn() {
15         boolean lightIsOn = InternalOperationTurnLightOn();
16         return lightIsOn;
17     }
18 }
```

Fig. 2. Code snippet of the Phillips Hue implementation class.

Figure 3 showcases the resource paths for invoking the device capabilities. The developer of the client application is now able to invoke the base URL, which will return the description of the bundles currently installed and the paths for retrieving details on how to invoke each device capabilities. Figure 4 presents the currently installed Phillips Hue bundle and exposes the description and paths for invoking the implemented functionalities of the device. An HTML5 client has been implemented, which allows showcasing the use of the dynamically generated service interfaces that enable accessing and controlling the Phillips Hue Smart Light device (demo video[1]).

[1] Available at: https://www.youtube.com/watch?v=NQ0tzv5Ob48&sns=em

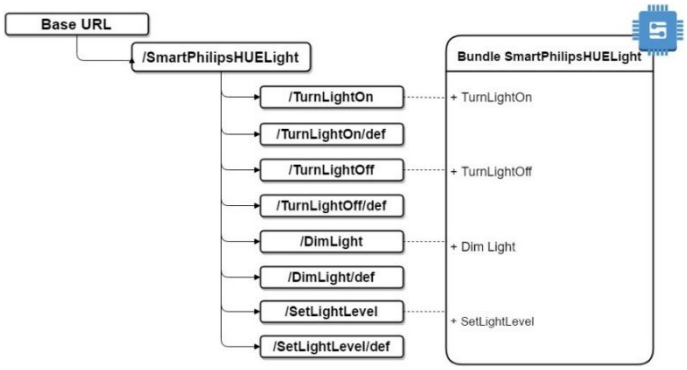

Fig. 3. Generated RESTful service interfaces for the Phillips Hue Smart Light.

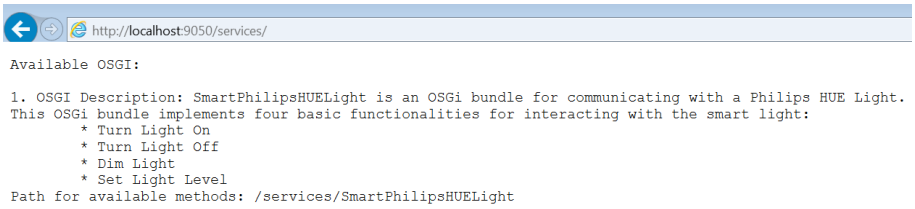

Fig. 4. Generated RESTful service interfaces for the Phillips Hue Smart Light.

5 Conclusions and Future Work

The research work presented in this paper aims to provide an Adaptive Runtime Middleware (ARM). The proposed middleware allows utilizing the benefits of the OSGi framework, the Java reflection mechanism and the RESTful architectural pattern in order to provide solutions to the IoT challenges of scalability, heterogeneity and flexibility. The presented use case scenario demonstrates the adaptive capabilities of the proposed ARM. The architecture of ARM enables to address the aforesaid IoT issues, since for each IoT device or Cloud service a corresponding bundle can be developed following the guidelines presented in this work. The bundle can be then exported and the ARM - can install, start and parse the bundle so as to generate at runtime the required service interfaces. Future research work aims to extend the middleware capabilities so as enable dynamic generation of Server-Sent Events (SSE) for handling sensor devices data as soon as they become available. Finally, a rules engine will be implemented for defining dependencies between device and/or software services, for example motion detected → turn on camera.

References

1. E. Fleisch and F. Mattern (2005) Das Internet der Dinge: Ubiquitous Computing und RFID in der Praxis: VTAH. Springer-Verlag.

2. F. Zeshan, et al. (2014). Service Discovery Framework for Distributed Embedded Real-Time Systems. In I. Ghani, W. Kadir, & M. Ahmad (Eds.), Handbook of Research on Emerging Advancements and Technologies in Software Engineering.
3. D. Lizcano et al. (2008). Leveraging the Upcoming Internet of Services through an Open User-Service Front-End Framework. In: Mähönen P., Pohl K., Priol T. (eds) Towards a Service-Based Internet. ServiceWave 2008. LNCS, vol 5377. Springer, Berlin, Heidelberg.
4. D. Guinard, et al. (2010) Interacting with the SOA-Based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services. IEEE Trans. on Services Computing, vol. 3, no. 3.
5. V. Issarny, et. al. (2016) Revisiting Service-Oriented Architecture for the IoT: A Middleware Perspective. In: Sheng Q., Stroulia E., Tata S., Bhiri S. (eds) Service-Oriented Computing. ICSOC 2016. Lecture Notes in Computer Science, vol 9936. Springer, Cham.
6. D. Guinard and V. Trifa, (2009). Towards the Web of Things: Web Mashups for Embedded Devices, In: Proc. Workshop Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM'09).
7. T. Teixeira, et al. (2011). Service Oriented Middleware for the Internet of Things: A Perspective. In: Towards a Service-Based Internet, LNCS, vol. 6994, pp. 220–229. Springer Berlin Heidelberg.
8. F. Paganelli, et al. (2012). A DHT-based discovery service for the Internet of Things. In: Journal of Computer Networks and Communications.
9. Y. Zhu, Xiao-hua M. (2010). A Framework for Service Discovery in Pervasive Computing. 2nd Int. Conference on Information Engineering and Computer Science (ICIECS).
10. S. Neely, et al. (2006). Adaptive middleware for autonomic systems, In: Ann. Técommun., vol. 61, no. 9–10, pp. 1099-1118.
11. J. Barton and T. Kindberg. (2001). The Cooltown user experience.
12. M. Román, et al. (2002). Gaia: A Middleware Infrastructure to Enable Active Spaces. IEEE Pervasive Computing , pp. 74-83.
13. A. Ranganathan, et al. (2004). Middlewhere: A middleware for location awareness in ubiquitous computing applications. Springer.
14. A. Chan and S.-N. Chuang. (2003/12). Mobipads: a reflective middleware for context-aware mobile computing. IEEE Trans. on Software Engineering , 29 (12), 1072-1085.
15. N. Paspallis, et al. (2009). Developing Self-Adaptive Mobile Applications and Services with Separation-of-Concerns', At Your Service: Service-Oriented Computing from an EU Perspective, MIT Press, chapter 6, pp. 129-158.
16. openHAB (2017). Retrieved January 24, 2017, from http://www.openhab.org/
17. J. Soldatos, et al. (2015). OpenIoT: Open Source Internet-of-Things in the Cloud. Interoperability and Open-Source Solutions for the Internet of Things, 9001, 13-25.
18. OpenIoT Consortium. (2016). OpenIoT - Open Source cloud solution for the Internet of Things. Retrieved January 24, 2017, from http://www.openiot.eu/
19. M. Papazoglou. (2003). "Service-oriented computing: Concepts, characteristics and directions," in Proc. 4th Int. Conf. Web Inf. Syst. Eng. (WISE'03), pp. 3–12.
20. M. Sarnovsky, et al. (2008). First demonstrator of hydra middleware architecture for building automation, Václav Snášel. pp. 204–214, FIIT STU Bratislava.
21. A. M. C. Souza et al. (2013). A Novel Smart Home Application Using an Internet of Things Middleware, Smart SysTech 2013,Erlangen/Nuremberg, Germany, pp. 1-7.
22. A. B. Hamida, Fabio Kon, et al. (2013). Integrated CHOReOS middleware - Enabling large-scale, QoS-aware adaptive choreographies.
23. A. Azzara, et al., Middleware solutions in WSN: The IoT oriented approach in the ICSI project, Int. Conf. SoftCOM, 2013.