Cooperative Middleware Platform as a Service for Internet of Things Applications

Leonardo Albernaz Amaral, Ramão Tiago Tiburski, Everton de Matos, Fabiano Hessel
Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre - RS - Brazil
(leonardo.amaral, ramao.tiburski, everton.matos.001)@acad.pucrs.br, fabiano.hessel@pucrs.br

ABSTRACT

The proliferation of the Internet of Things (IoT) technology in several business domains requires a well-defined and distributed infrastructure of system services that provides IoT device and data management and supports a proper application development. To address these issues we have developed COMPaaS, a web-based middleware platform to enable an easy and loosely coupled cooperation between applications and IoT devices, and also to facilitate the development of IoT applications. Some tests were performed to validate the middleware component of the platform and the results revealed that the middleware is able to process up to 1000 applications requests simultaneously and also notify the responses within an acceptable time for the applications. It indicates that not only the performance of the middleware is acceptable for most IoT applications, but also that the cooperative approach imposed by the platform allows a satisfactory integration between the system modules and aggregates important value for the development of IoT applications.

General Terms

Management, Performance, Design and Experimentation.

Keywords

IoT, Middleware Platform, Cooperative Systems and Applications.

1. INTRODUCTION

Middleware systems are enabling technologies that facilitate the development of distributed applications. These systems have evolved from hiding network details from applications into more sophisticated systems to handle many important application requirements, providing support for distribution, heterogeneity, mobility, interoperability, and data management, just to name a few. The evolution of middleware technologies has been influenced by numerous developments and standard efforts, and the Internet of Things (IoT) is an important field that dictates the increasing evolution of middleware in terms of both worldwide scientific research and industrial development.

SAC'15, April 13-17, 2015, Salamanca, Spain.
Copyright 2015 ACM 978-1-4503-3196-8/15/04…$15.00.
http://dx.doi.org/10.1145/2695664.2695799

IoT has been stating a vision of a new computing paradigm that goes beyond to plug physical devices into the Internet in order to link physical and virtual objects globally. IoT must cause new experiences to users, providing smart and cooperative services according to users behaviors and needs. Thus, the main enabling factor of the IoT is the integration of several technologies in an environment where things can automatically communicate to computer systems, people and each other providing services for the benefit of humankind.

Many applications in the IoT start with single purpose systems designed for homogeneous physical device platforms using specific network protocols. The hardware of the device has a fixed set of sensors, and the applications cannot be easily ported to other platforms. There is very little reuse from one project to another, and the system architecture is thoroughly tight coupled. The separation of design abstractions between the low-level hardware and high-level applications is not easy in these systems when compared with server-based systems, not to mention making them cooperative, adaptable and evolvable to new services and new environments.

There exist some open issues for better middleware support to facilitate the development of IoT applications [1][2][5]. Middleware must be able to support application developers and users avoiding them to be constrained by which and how physical devices are implemented or deployed in a target environment. Middleware must provide standard abstractions for both the device registration and interoperability, as well as for the provision of high-level and cooperative services to applications, making the IoT programming as platform independent as possible using simple, web-based and high-level primitives instead of the node-centric programming. The automatic mapping of the application requirements in specific tasks in the middleware, allows the developer to simply requests its desires in terms of high-level events, rather than be concerned with which kind and how devices must be used by the application.

This paper aims to present COMPaaS, a Cooperative Middleware Platform as a Service to assist in the development of IoT applications. COMPaaS extends the specifications of the EPCglobal regarding RFID middleware interfaces for high-level services provision. Further, it provides a lightweight system architecture based on the ETSI specifications for M2M (Machine-to-Machine) services platform, as well as web-based application services for physical devices integration (CoAP project) [8]. The main functions of the middleware platform range from data management to device integration and address the provision of high-level and cooperative services to applications.

The remainder of this paper is organized as follow. Section 2 presents the overall structure of the COMPaaS middleware platform as well as its main features. Section 3 shows some validation tests, while Section 4 reviews existing approaches in terms of middleware for IoT and platforms for M2M communication. Section 5 provides a discussion about the previous sections. Section 6 provides some concluding comments and suggests further activities.

2. COMPAAS – COOPERATIVE MIDDLEWARE PLATFORM AS A SERVICE

COMPaaS middleware platform is a software system that provides to users a simple and well-defined infrastructure of services (see Figures 1 and 2). Behind the services provided there is a set of software elements (or system modules) that deal with the users and applications requirements, for example, request and notification of data, discovery and management of physical devices, communication issues, and rule-based data management.

The goals of the COMPaaS can be summarized as follow:

- Abstract the integration and interoperability with physical devices (physical resources) through the provision of hierarchical services according to the profile of the device.
- Abstract the collection and management of the data provided by physical devices through the provision of application level services.
- Provide high-level services to facilitate the development and integration of IoT applications.
- Provide well-defined software architecture based on IoT/M2M and WoT (Web of Things) standards.

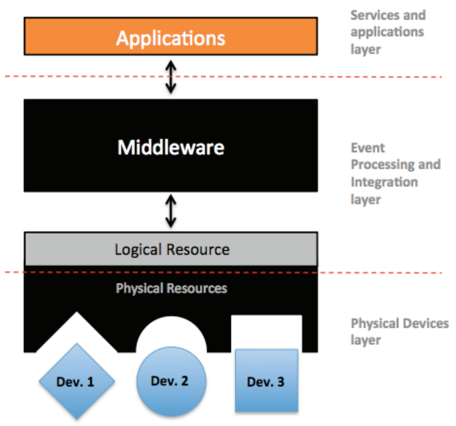

Figure 1. COMPaaS architecture overview.

2.1 Features of the Platform

COMPaaS is composed of two main cooperating systems (see Figure 1), Middleware and Logical Resource. Middleware is the system responsible to abstract the interactions between applications and physical devices and hides all the complexity involved in these activities. On the other hand, Logical Resource is the system (or systems) responsible to hide all the complexity of physical devices and abstracts the functionalities of these devices. Both systems and their abstraction layers are explained in the next topics.

2.1.1 Services Abstraction and Cooperation Model

In terms of design standardization and cooperation model the architecture of the COMPaaS extends the patterns defined by the EPCglobal regarding the specifications of RFID Middleware systems, and provides three services layers that are illustrated in Figure 2 (from top to bottom): Application Level Events (ALE), Middleware Level Events (MLE), and Device Level Events (DLE). Both services abstract specific software layers and always provide services addressed to the next upper layer of the architecture.

According to Figure 2, ALE is the service layer responsible to handle the desires (requests) of the applications. It receives requests from the applications and triggers the related actions in the middleware. Middleware system not only acts as service provider providing the ALE services to applications, but it also acts as service consumer consuming cooperative services from one or more MLE interfaces. Middleware level events (MLE) are the events produced by the logical resources (abstraction for physical devices) that are requested and consumed by the middleware. Therefore, the MLE uses the DLE interface to abstract the native API of the physical device, and consequently, also to control the features of the device in order to provide a well-defined service interface on top of the logical resource.

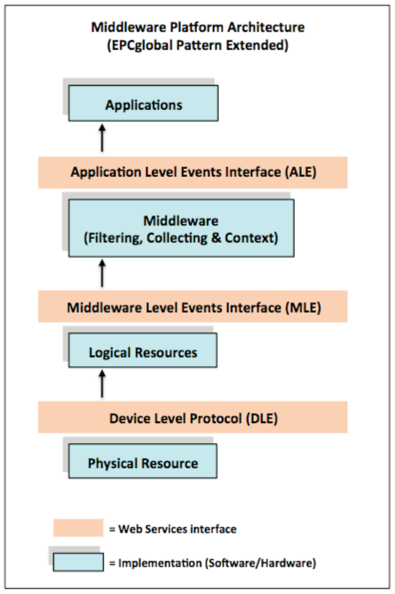
Middleware Platform Architecture (EPCglobal Pattern Extended)
Applications
Application Level Events Interface (ALE)
Middleware (Filtering, Collecting & Context)
Middleware Level Events Interface (MLE)
Logical Resources
Device Level Protocol (DLE)
Physical Resource
= Web Services interface
= Implementation (Software/Hardware)

Figure 2. Layers of Services Abstraction.

2.1.2 Logical Resource

Logical resource is the system abstraction for the physical devices that are relevant to the applications and that must be accessible to provide some benefit. A logical resource can be represented as a “basic resource” or a “composite resource”. A basic resource is an extension of a logical resource. A composite resource can be composed of one or more basic resources. A good example of a composite resource is a sensor network composed of several sensor and actuator nodes with each one node acting as a basic resource generating its on data.

Logical resources are described through system profiles. Each system profile contains attributes to characterize the physical device, such as: name, manufacturer, function, model, data type, URI, etc. These attributes are used by applications to find the desired resources. Besides the profile of the resource, the basic resource contains two more system elements: communication module and service module. The communication module is not only responsible for the publication of the resource (the registration of the resource in the middleware), but also to provide the features for data communication and the notification of the operational status of the resource to the middleware. On the other hand, the service module is responsible to expose the interfaces and features of the resource to the middleware.

2.1.3 *Resource-Oriented middleware*

The main functions of the middleware range from data management to device integration and address the provision of high-level services to applications.

Figure 3 presents the organization of the modules of the middleware. All "services" are part of the middleware API available to applications, except the "communication service", which is used for both applications and logical resources. The rest of the modules (Resource Manager, Resource Handler, and Event Handler) allow the integration with logical resources.

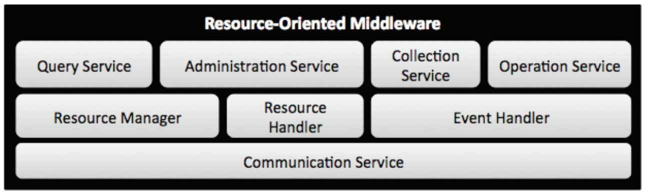

Figure 3. Modules of the Middleware System.

The Communication Service receives the information of the profile from the logical resource and notifies the Resource Manager to realize the registration of the profile. The Resource Manager module is responsible to deal with the configuration of the resources allowing the composition of the infrastructure of resources in a logical structure according to the application requirements. Besides the configuration of the resources, the Resource Manager also deals with the persistence and acquisition of the resources.

The Resource Handler module is responsible for the coordination of the activities of the resources as well as the verification of their operational status being able to report any possible error. The resource handler uses a rule-based event-processing engine (Drools), which facilitates not only the inclusion of new logical resources in the middleware, but also the entire management of these resources as well as their data.

The Event Handler module is responsible for the management of the data and the operations that are requested by both the Collection and Operation Services. The collection cycle and the operation cycle are the tasks responsible to realize this management. The collection cycle is the period of time in which the data is collected by the middleware from one or more logical resources, which happens with the support of the Resource Handler module. After the collection phase, the data are processed (filtering and aggregation of data) in order to select only the data that are relevant to the application. At the end of each collection cycle, a report is sent to the application reporting not only the requested data, but also a list of possible errors.

A collection cycle is defined through the specification of a collection report. This specification report specifies: the resources to be used in the data collection cycle, the duration time and the restrictions of the cycle, the method to be used to process the collected data, and the desired time interval between the requested reports.

An operation cycle is the period of time in which some operations are executed in the resources according to the application request. The Resource Handler is responsible for the execution of the desired operations in the resources, whereas the operation cycle manages the restrictions and controls the execution time. At the end of each operation cycle, a report is sent to the application with the list of successful and unsuccessful operations.

2.2 Technical overview of the platform

Figure 4 presents and overview of the middleware platform and highlights some technical details around the integration between application, middleware and logical resources, as well as the main flow of information. The idea is to present the communication interfaces, communication patterns, and informations that are exchanged between the cooperating systems.

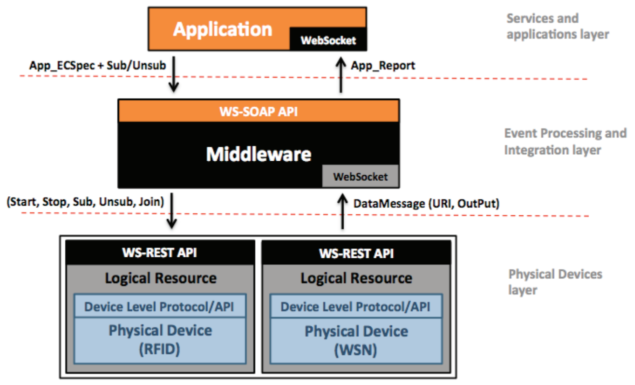

Figure 4. Systems integration and information flow.

To help the description of the systems we use a simple use case regarding "a generic application requesting data from both an RFID reader and a sensor network". First we describe the main characteristics of the environment and some technical details of the systems. Further, we present the basic interactions and the data exchanged.

- The operational environment is composed of the application, middleware, and two logical resources, one that encapsulates and controls an RFID reader, and another one that encapsulates and controls a sensor network.
- Each logical resource provides its own services to the middleware through a standard RESTful API (HTTP protocol for subscription). These services abstract the main methods of each device and allow middleware to send commands like start, stop, subscribe, unsubscribe and join. Each logical resource must be implemented in order to define its "profile" and to "encapsulate the low-level API of the device (device driver)". Thus, the

native API of the device can be mapped into the RESTful API.

- Logical resources use a WebSocket channel (a data notification service provide by the middleware) for asynchronous responses (TCP-based protocol for notification of data). We use WebSocket not only to avoid the lack of performance of the HTTP (request-response style), but also to allow that physical devices can notify the middleware without a synchronous request.
- We also use a "DataMessage object" as the default data packet between logical resources and middleware. DataMessage is an object that represents the data generated by the logical resources. It is a generic object and must be created in the design time according to profile of the resource. It demands a proper and pre-defined CEP-based rule in the middleware to be processed.
- Middleware provides its on services to the application through Web Service SOAP. We use SOAP as the communication pattern between applications and middleware because SOAP is a standard and fits well in scenarios with "medium-performance" in terms of real-time communication requirements. Furthermore, SOAP is "neutral" in terms of transport protocol. It runs over any transport protocol (HTTP, TCP, UDP) and also allows independence of any client-programming model (loosely-coupled distributed communication pattern) what is an important requirement for the interoperability required by the middleware project.
- We also use WebSocket for asynchronous responses from middleware to applications (TCP-based protocol for notification of data). We use WebSocket to allow that the middleware can notify the application without a synchronous request.
- Both systems architectures (Middleware and Logical Resource) are based on "Subscribe/Notify" and "Observer" communication pattern. "RESTful + WebSocket" for the Logical Resource, and "SOAP + WebSocket" for the middleware.

The interactions and the exchanging of data are described below.

- The application uses the ALE interface of the middleware to define its request (a "collection specification" to gather data from both the RFID and the sensor network) and to subscribe the middleware.
- Middleware interprets the application request and creates the respects "collection cycles".
- Middleware uses the MLE interface of the logical resources and subscribes the devices involved (each Logical Resource) in the "collection cycles".
- Middleware starts the devices and collects data from them. Each logical resource provides the requested data according to its generation data pattern and encapsulates it in a DataMessage object that is notified to the middleware.
- Middleware processes the data according to the collection specification and, at the end of each cycle, stops and unsubscribes the devices, and also prepares the proper reports to be sent to the application.
- Middleware notifies the requested reports to the application.
- The application unsubscribes the middleware and uses the data collected.

3. PLATFORM VALIDATION

To perform the tests of the middleware component we have used an infrastructure of test composed of two PCs – one with two virtualized machines, and another one with no virtualization. Two virtual machines were hosted on a PC with Ubuntu 14.04.1, Intel Xeon E5-2620 2.00GHz, 32GB of Ram, and with hyper-threading, enabling it to reach up to 24 cores. One virtual machine has Ubuntu 14.04.1, 16 cores, and 8GB of Ram and was dedicated to host the middleware. The other virtual machine has Ubuntu 14.04.1, 8 cores, and 4GB of Ram and was dedicated to simulate the devices. The other PC used has Ubuntu 14.04.1, Intel Core i5-3230M CPU 2.60GHz, and 8GB of Ram and is responsible to simulate the applications requesting data to the middleware.

The goals of the tests are twofold: (1) Evaluate the middleware system's ability to process simultaneous requests from applications without affect their response time requirement, returning the results in an acceptable time; And (2) perform a functional verification of the platform, analyzing the management of devices, the processing of data, and the provision of services.

To perform the tests we have defined a test plan with different scales of time and specification reports. The logical resources were simulated in 15 devices (abstractions for 15 medical devices), being 10 basic resources, and 5 composite resources. Each composite has two basic resources resulting in a total of 20 simulated resources. The time interval for data generation of each device was defined in 1 second.

About the specification reports of the applications, each one has been configured to request data from all the registered devices during 5 seconds (colleting time), with 1.5 second as a repeat period, and grouping the data per resource type. As the applications requests were generated and managed by an application engine, the interval time between sending each specification report is 100 milliseconds.

Several tests were performed and the results are presented in Table 1. The first column shows the amounts of specification reports used in the test, which represent the applications requests. The first row describes the total time of each request (in seconds). After this time each requester finishes its execution. The remaining data are the average processing time (also in seconds).

Table 2. Processing capability of the middleware (s)

Specs/Time (s)	300	900
100	5.30	5.53
500	6.26	6.49
1000	6.91	7.13

The tests reveal that the middleware is able to process up to 1000 applications requests simultaneously and also to notify them within an acceptable time (1000 applications requests, around 70.000 report notifications for each request, and an average of 80 reports notified per second). One of the parameters informed by the application in the specification request is that "each collection cycle must last 5 seconds". In other words, the middleware must collect data from the devices for 5 seconds, and the time remaining is used in the processing of the data. In this sense, the time considered acceptable by the application is nearer than 5 seconds, since 1000 specifications are treated at the same time.

The increase in the requesting time from 300 seconds to 900 seconds does not cause practically any impact, since the number of specification reports processed concurrently remains the same. The only difference is in terms of the number of reports notified by the middleware, which increases according to the increasing in the requesting time. Another important aspect to be considered is that the variation between each amount of requests also does not affect the performance of the middleware, since it keeps the processing activity within an acceptable time. This indicates that 1000 applications may request the middleware services concurrently and all of them will receive responses within an acceptable time, which seems to be satisfactory for most of the IoT applications.

Regarding the second objective of the tests, we have noticed that the communication patterns of the platform worked well, allowing a decoupled and asynchronous web-based cooperation between the systems. The middleware was able to manage the devices through their interfaces (RESTful interface), interacting with them when necessary. The module responsible for the data processing also behaved as expected, treating the data of all the devices and correctly assembling the report that was sent to the applications. The services provided by the middleware (SOAP services) also responded well, since they were used for hundreds of applications to create, subscribe and unsubscribe the specification reports.

4. RELATED WORKS

As we mentioned at the beginning of this paper, IoT applications need to be supported by middleware solutions. The functionalities required by IoT middleware systems are explained in detail in [2], [3], and [4]. In addition, challenges in developing middleware solutions for IoT are discussed in [5]. According to [4], the majority of the IoT middleware solutions do not provide context-awareness, data management and interoperability functionalities. In contrast, almost all the solutions are highly focused on device management, which involves connecting sensors to the IoT middleware.

According to [5], other important issue for IoT middleware is scalability, which can range from devices to applications requests. The middleware systems proposed in [6] and [7] have some similarities to COMPaaS, however, they do not present scalability results in terms of processing capability, which difficult any comparative process.

Another perspective in terms of existing systems and platforms for IoT can be created through an analysis of the existing M2M platforms for IoT. In terms of commercial platforms, The European Telecommunication Standards Institute (ETSI) Technical Committee M2M (TC M2M) provides IoT/M2M architecture with a generic set of capabilities for IoT/M2M services. ETSI M2M defines the Service Capability Layer (SCL), which provides functions that are shared by different applications, and the reference points between M2M devices, gateways, and the network. Many commercial M2M platforms are compliant with the ETSI M2M standard.

Current commercial M2M platforms include Cosm [9], ThingSpeak [10], Nimbits [11], EVERYTHNG [12], Sensinode [13], OnePlatform [14], Axeda [15], SensorCloud [16], NeuAer [17], and iDigi [18]. The main characteristics of these platforms can be summarized as follows:

- Application protocol and interface: Existing platforms use REST architecture or Simple Object Access Protocol (SOAP) for interfaces and they operate on Hypertext Transfer Protocol (HTTP), Constrained Application Protocol (CoAP) [8], or HTTPS protocols. Most platforms use RESTful architecture based on the HTTP protocol. Some use lightweight CoAP, and others use HTTPS for security.
- Registration: Devices and users have to be registered to be available for the platform or to use the platform.
- Data processing: Platform can process and analyze the information collected from devices, such as the maximum, minimum, and average values of the data or perform other customized function (as filtering and aggregation).
- Account: The platforms provide differentiated accounts according to privacy, storage, service type (SMS, web service, cloud), and the number of available devices.
- M2M network support: Some platforms allow gateways to connect devices that cannot directly connect to the Internet.

In terms of research platforms, INOX [19] is an example of M2M service architecture for IoT. This platform provides the registry and discovery of things, monitoring, virtualization of objects, networking and computational resources, service enablers and self-functionalities, and orchestration capabilities for controlling and managing services.

The authors in [20] designed an M2M platform that bridges business applications and machines based on the Java 2 Enterprise Edition (J2EE) framework and service oriented architecture (SOA). Web service is used to communicate with the external components. This can reduce their dependence on one another and allows for quick responses to changes in the industry chain. Internal design uses J2EE, which consists of a presentation layer, a business layer, and a data access layer.

5. DISCUSSION

Networking alone does not make the IoT useful. IoT applications today depend on Web architectures, using HTTP to access information and to perform updates. HTTP is based on Representational State Transfer (REST), an architectural style that makes information available as resources identified by URIs. Thus applications communicate by exchanging representations of these resources using HTTP as the transfer protocol.

Web architectures are a powerful and extensible paradigm that is quickly spreading to IoT applications, letting developers of web-based applications continue using their existing skills. However, not only the applications are benefiting from the web-based patterns, but also the IoT and M2M platforms are using these patterns to provide standard services to applications. COMPaaS is an example of platform that benefits the applications through the use of web patterns. The systems into the COMPaaS (subsection 2.1) (Middleware and Logical Resources) are based on "Subscribe/Notify" communication pattern. This architectural pattern is an important step towards the separation of concerns, since it decouples the services abstractions between the low-level hardware and high-level applications requirements.

Another important aspect to be discussed in the web-based technology is the "Observer" pattern. In HTTP, transactions are always client-initiated, and the client must perform GET operations again and again (polling) if it wants to stay up to date about a resource's status. This pull model becomes expensive in an environment with limited power, limited network resources, and nodes that sleep most of the time. In this way, COMPaaS also uses an asynchronous approach to support pushing information from servers to clients (the notification of data). In a GET request, a client can indicate its interest in further updates from a resource by specifying the "Observe" option. If the server accepts this option, the client becomes an observer of this resource and receives an asynchronous notification message each time it changes. Each such notification message can be identical in structure to the response to the initial GET request.

Several standards are dealing with these web-based issues for IoT. In CoAP project [8], instead of trying to create another complex publish–subscribe architecture, CoAP effectively provides a minimal enhancement to the REST model, just adding the well-known observer design pattern. The CoRE technology has already become widespread in both open source communities and industry applications, with implementations of CoAP and related specifications available in several programming languages along with Firefox and Wireshark support. Several other standards activities are using IETF CoRE standards as part of more complete M2M systems. The ETSI M2M Technical Committee has specified a service-layer architecture for M2M that includes bindings for both HTTP and CoAP.

6. CONCLUSION

In this paper we present the COMPaaS a middleware platform to support the development of applications for the IoT. This platform is composed of layers that abstract the device and data management as well as the interoperability of the systems involved. COMPaaS provides well-defined software architecture based on extensions of standards from the EPCglobal, IoT, WoT (Web of Things), and M2M, which help in the decoupling of the services abstractions between the low-level hardware and high-level applications requirements.

Some tests were performed to validate the middleware component. The results revealed that the middleware is able to process up to 1000 applications requests simultaneously and also notify the requests within an acceptable time. It indicates that not only the performance of the middleware is acceptable for most IoT applications, but also that the cooperative approach imposed by the platform allows a satisfactory integration between the system modules and aggregates important value for the development of IoT applications.

The next steps with the COMPaaS project are focused on new tests based on real deployment environments, as well as some new features in order to add context-awareness capability, information services provisioning, and security services support.

7. ACKNOWLEDGMENTS

Our thanks to CAPES/CNPQ for the funding within the scope of PNPD project number 058792/2010.

8. REFERENCES

[1] Perera, C., Zazlaviky, A., Christen, P., Georgakopoulos, D., "Context Aware Computing for the Internet of Things: A Survey", IEEE Communication Surveys and Tutorials, vol. 16-1, 414-454, 2014.

[2] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelffle, "Vision and challenges for realizing the Internet of things", Tech. Rep., March 2010, www.internet-of-things-research.eu/pdf/IoT_Clusterbook_March_2010.pdf [Accessed on: 2014-12-09].

[3] Atzori, L.; Iera, A.; Morabito, G. "The Internet of Things: A Survey", Computer Networks, vol. 54–15, pp. 2787-2805, 2010.

[4] Bandyopadhyay, S.; Sengupta, M.; Maiti, S.; Dutta, S. "Role of middleware for internet of things: A study", International Journal of Computer Science and Engineering Survey, vol. 2–3, pp. 94–105, 2011.

[5] Chaqfeh, M. and Mohamed, N. "Challenges in middleware solutions for the internet of things". In: International Conference on Collaboration Technologies and Systems (CTS), pp. 21–26, 2012.

[6] Kim, M.; Lee, J. W.; Lee, Y. J.; Ryou, J.-C. "Cosmos: A middleware for integrated data processing over heterogeneous sensor networks", ETRI Journal, vol. 30–5, pp. 696–706, 2008.

[7] Valente, B.; Martins, F. "A middleware framework for the internet of things". In: Third International Conference on Advances in Future Internet, pp. 139–144, 2011.

[8] Kim, Jaewoo., Lee, J., Kim, J., yun, J., "M2M Service Platforms: Survey, Issues, and Enabling Technologies", IEEE Communication Surveys and Tutorials, vol. 16-1, pp. 61-76, 2014.

[9] Cosm: https://cosm.com/

[10] ThingSpeak: https://www.thingspeak.com/

[11] Nimbits: www.nimbits.com/

[12] Evrythng: http://evrythng.com/

[13] Sensinode: http://www.sensinode.com/

[14] Oneplatform by Exosite: http://exosite.com/

[15] Axeda Platform: http://www.axeda.com/

[16] SensorCloud: http://www.sensorcloud.com/

[17] NeuAer: http://www.neuaer.com/

[18] iDigi Device Cloud: http://www.digi.com/

[19] S. Clayman and A. Gali, "INOX: a managed service platform for inter- connected smart objects," *Proc. of the workshop on Internet of Things and Service Platforms*, pp. 1-8, 2011.

[20] S. Zhang, J. Zhang, and W. Li, "Design of M2M Platform Based on J2EE and SOA", International Conference on E-Business and E- Government, pp. 2029-2032, *2010.*