Available online at www.sciencedirect.com

ScienceDirect

Procedia Engineering 100 (2015) 1215 – 1223

Procedia Engineering

www.elsevier.com/locate/procedia

25th DAAAM International Symposium on Intelligent Manufacturing and Automation, DAAAM 2014

An Integrated Approach to Common Problems in the Internet of Things

Sergey Efremov*, Nikolay Pilipenko, Leonid Voskov

Higher School of Economics, 20 Myasnitskaya str., 101000 Moscow, Russia

Abstract

The recent advances in technology enabled transition to the Internet of Things (IoT), in which physical objects around us become an integral part of the global information system. A major technical challenge however is to make these numerous objects interact seamlessly with each other. The latest works related to concepts, such as Web of Things or Social Web of Things, partly address the issue. In our paper we further investigate the topic and point out several problems that need to be efficiently solved for the Internet of Things to work on large scale numbers. One of the main tasks is to make devices easily discoverable. Thus, an efficient way to handle and store their metadata is required. Another problem is connected with providing different models of inter-device communication, asynchronous being the most important, as many of today's widely used web standards were not designed for it. Finally, we propose a general cloud-based IoT architecture aimed at solving the above-described problems.

Peer-review under responsibility of DAAAM International Vienna

Keywords: Internet of Things; Web of Things; Device discovery; Communication models; Cloud WoT platform

1. Introduction

The concept of the Internet of Things (IoT) [1] started to gain momentum around 1999 when perspectives of broader device-to-device communication became a reality. The original idea was to give each physical object a tag, which would serve as object's identifier in a global network. Logistics was the target application at the time.

* Corresponding author. Tel.: +7-910-421-72-64
E-mail address: sefremov@hse.ru

Peer-review under responsibility of DAAAM International Vienna
doi:10.1016/j.proeng.2015.01.486

As technologies developed over the years, a broader range of possibilities has emerged. Low-power microcontrollers, efficient batteries along with energy-saving wireless technologies enabled to enrich capabilities of individual devices as well as to extend their lifetime. In the last 10 years a whole eco-system of so-called smart devices has been created, which now find their application in almost all possible areas, from personal gadgets to complete systems of smart cities. According to the Cisco research groups [2], in 2008-2009 the number of internet-connected devices became larger than the number of the people on Earth, which, as many suppose, is the actual transition to the Internet-of-Things.

As it often happens with novel network technologies, the first stage of their development marked appearance of multiple proprietary protocols, which vendors used in their equipment. As a result, individual solutions were completely non-compatible with each other. Gradually integration of the Internet of Things and the World Wide Web started, which created the term "Web of Things (WoT)" [6].

Below are the main characteristics of the modern Web of Things:

- HTTP is used as the main protocol for communication between devices
- Device interfaces are made open
- Smart devices expose their functionality through a REST interface [3], thus any device is represented as a service

In this paper we outline several problems of IoT and WoT, which, to our best knowledge, still have not been fully solved.

The first problem arises when smart devices operate in a dynamic environment. For example, a smart car driving along the road and acquiring information about road conditions or traffic jams from other cars or sensors along its way. In this type of scenario links between smart devices cannot be established in advance, as it would have been possible in a mostly static environment, e.g. smart home. Thus, a mechanism for discovering available devices is required.

The second problem is specific for Web of Things and it regards asynchronous communication patterns. Many internet-connected devices use a very simple data transfer strategy: when a new piece of information is available, it is sent to a web-service. This pattern perfectly fits original web standards. However, when data needs to be pushed asynchronously to a remote device, some other schemes are required. In section 2 we discuss possible options to solve this problem.

Our contributions are the following. In section 2 we give a compiled set of requirements for a device discovery service in a global IoT architecture. We also propose a prototype of a query language to extend its search capabilities. Section 3 contains our general idea of an adaptive protocol scheme for asynchronous communication in the Web of Things. Finally in section 5 we describe an architecture of our own cloud-based WoT platform.

2. Device discovery

Device discovery is one of the main tasks in the Internet of Things. The increase in the number of internet-oriented objects along with the development of the open systems concept enabled building global dynamic scenarios that cover hundreds and thousands of autonomous devices interacting with each other for collaborative purposes.

The main goal of search in IoT is to find a relevant source of information or an actuator device within multiple heterogeneous distributed systems. Search criteria may include device type, geolocation, functional capabilities and other characteristics.

As mentioned before, at the first stage of IoT development manufacturers used their own proprietary protocols and technologies for inter-device and inter-system communication. In contrast, in the Web of things the task of finding a device is reduced to finding a web-service representing a device in the global network. As it is shown in section 3, these services may operate on devices themselves or on special middleware data platforms. Any device gets its virtual online representation just by specifying an open interface (RESTful API). However, in general, a device may have no advance knowledge of other devices or systems that will interact with it.

In a global network two levels of services exist (fig. 1). The device service level is formed by all devices with their open web-interfaces. The aggregation service level consists of different mashup applications, device connectors, discovery services and others.

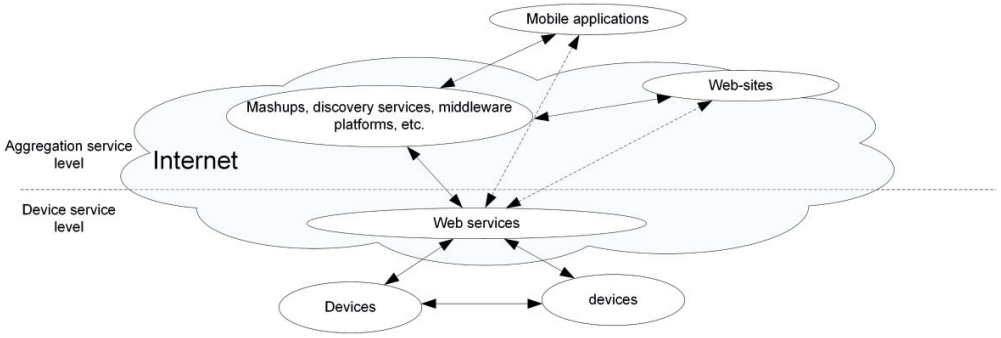

Fig. 1. IoT services.

A number of solutions for discovery of objects in IoT have been already suggested. In [12] a detailed comparison is provided, below we summarize the main ideas:

- EPCglobal standards [15] contain a high-level description of a discovery service. At the moment of writing this paper, the standards were partly adopted. The standard suggests the following components: a unique EPC code for each device, EPCIS (EPC Information Service) – a service providing data for devices with a specific EPC, local search services storing references to all available EPCIS and a central EPCglobal service.
- BRIDGE provides a high-level description and analysis of approaches to implementation of a discovery service. The key idea of BRIDGE is a directory of services. Each service registers in the DS blocks of information about availability of data related to a particular device. The discovery service also alerts connected clients in case of updated information.
- The Afilias service developed by the eponymous company solves a number of problems: object identification, providing backward compatibility with existing identification schemes, ensuring usability, extensibility and solution openness. To support multiple independent schemes ESDS (Extensible Supply-chain Discovery Service) is used. At the moment Afilias is available as a prototype with the underlying infrastructure, technical support and development tools. The solution is based on DNS and compatible with EPCglobal.

There are number of other projects aimed at solving search and discovery tasks in IoT: ID@URI, DHT-P2P, WWAI and UDDI. It should be noted that all of the described discovery systems for IoT are currently at the stage of discussions, standardization or test projects.

To work on a global scale a search schema has to be standardized. Due to the fact that it is practically impossible to implement dynamic search globally, specialized discovery services (DS), capable to integrate device information, are required. Below are the basic requirements for a DS, which we believe are the most important:

1. Flexible object identifiers (OID)
2. Request by OID should return a list of all servers having information about the related device
3. It should be possible to find devices not only by complete OIDs, but by their components, such as product series, if an identification scheme supports them.
4. A DS should support requests querying presence of specific attribute
5. Search by the value of a specific attribute should be supported
6. Latest data from a device may be duplicated inside a DS for simplifying and accelerating the search process.
7. A full set of functions for service usage must be supported: device registration, authentication and authorization.

EPCglobal standards [15] contain a basic description of a global IoT architecture. However, they are oriented on finding a service based on an EPC code known in advance and do not support any other options of device search. The basic discovery scenario includes the following steps:

1. Acquiring an EPC of a physical device, e.g. by using RFID
2. Requesting EPCglobal for the service address (EPCIS)
3. Further communication with the service to obtain data or control

We propose our own solution based on this general architecture, with the primary focus on a query language. Adding a query language may significantly extend the search functionality. It will allow to discover objects that are beyond visibility of a particular device as well as to find services that do not have a physical representation.

To enable querying a discovery service needs to contain a directory with meta-description and address of each registered resource.

When initiating search, a client sends a HTTP GET request to the search service with additional parameters. The service returns a list of devices conforming to the required parameters. A display format is defined in the request header. A service should support all major formats, such as XML, JSON, CSV and others.

We have made an analysis of several existing protocols and solutions and came to the conclusion that the filtering and search system of the OData V4 protocol [16] may be taken as the base for the query functionality. Below we outline the main functions that a discovery service needs to implement.

A general request to a discovery service has the following structure: *GET search?<filtering parameters>*. For example, the following request will return all registered devices having a temperature attribute equal to 30:

GET search?$filter=Temperature eq 30

Filtering parameters should support comparison operators, logical functions, simple mathematics and a number of additional functions. The latter may include the following set:

- Functions on strings (*contains, endswith, startswith, length, indexof, substring, tolower, toupper, trim, concat*)
- Functions on dates (*year, month, day, hour, minute, second, date, time, now*)
- Mathematical functions (*round, floor, ceiling*)
- Special functions – *hasproperty* (checks whether a device has a certain property), *hasaction* (finds if a device supports a specified action), *distance* (calculates physical distance between two objects) and others.

On a global scale a discovery request includes the following sequence of actions:

1. Select all devices conforming to the search request
2. Send requests to selected devices to check if their state is synchronized
3. Return a list of devices to the client

To minimize communication between devices and the discovery service and to reduce search time cached device data may be stored inside the DS and devices themselves must specify a TTL value for their data.

3. Inter-device communication in the Web of Things

In this section we first give an overview of the three main models of device-to-device communication in the Web of Things and then discuss asynchronous patterns.

3.1. Direct access

In this scenario one of the interacting devices acts as a web-server. It is possible that in the near future most of embedded systems will support a complete TCP/IP protocol stack to allow each device to have its own IP-address and provide a RESTful API. However, it requires high processing capabilities and cannot be implemented on every device.

There are a number of solutions on the market that implement the described approach. For example, in the Sun SPOT project [4] each sensor (luminance, temperature, accelerometer and others) has an integrated server providing RESTful interface. The devices connect to a global network through a specialized proxy server.

3.2. Communication through a gateway

Providing direct network access is a complicated task for embedded systems. There is a significant number of technologies that focus on maximizing energy-efficiency of individual devices. As a result, their processing capabilities are very limited and it becomes impossible to implement a complete network stack on a microcontroller. A good example is wireless sensor networks. Usually a sensor node only implements 2 or 3 bottom OSI layers, which in most cases are not directly compatible with TCP/IP (protocols like IEEE 802.15.4, Bluetooth, ZigBee).

The problem can be solved by using internet gateways, which on the one hand use HTTP for communication with each other and on the other hand support specialized protocols for interacting with devices. A gateway has a separate IP-address in the global network and acts as a web-server.

For example, the Ploggs system [4] consists of intellectual wireless sockets measuring energy consumption and transmitting data through Bluetooth and a central gateway with a web-server installed, which allows users to gain remote access to devices.

3.3. Middleware data platforms

The primary goal of such platforms is to provide a centralized point to store, search, control as well as to visualize data from smart things. A server or a group of servers is maintained to store data from all connected devices. Different information systems can be built on top of such platforms, e.g. for monitoring equipment or human health.

In case of an open platform a toolkit for developing custom scenarios of interaction between smart devices can be provided.

A number of data platforms are available on the market today. Some of them are strictly proprietary solutions and not designed for public use, some others were developed as open systems [18].

3.4. Asynchronous Web of Things

Providing a downlink to a smart device in WoT, so that it can quickly get updates or commands, and organizing interaction of autonomous devices in multi-agent systems [21] are not at all easy tasks. In general it does not depend on which of the three schemes discussed above an individual device uses.

Historically web standards were oriented towards synchronous communication – a client sends requests to a server, which in turn responds with the data required. Transition to Web 2.0 with its dynamic content lead to appearance of multiple asynchronous schemes built on top of the standard HTTP.

There are several asynchronous technologies grouped by the name of COMET [8]: polling, long polling, forever frame and server sent events.

The most straightforward technique is polling. When using polling, a client sends regular requests to server checking whether any data is ready for it. The obvious disadvantage is large overhead required for message transmission even though no data may be ready for the client. The technique can be optimized by making polling interval dynamic, depending on what events are expected from the server.

When using long polling a client sends a request to a server, which then keeps the connection open until data is ready to be sent back. The approach is very similar to regular polling except that intervals between client requests are dynamic and depend on data update frequency on the server. The drawback of long polling is an increase in the number of open connections on the server.

The forever frame technique utilizes a hidden iframe integrated in a web-page. This iframe is sent as a chunked block, which enables dynamic transferring of new data while connection is active. When an event is ready to be sent to a client, the server creates a new <script> block, which is sent through the iframe connection. As soon as the web

browser receives the whole block it executes javascript code that is inside. Although this is a very convenient approach supported by all browsers, it cannot be easily implemented in resource-constrained devices.

For Server-Sent Events (SSE), which are part of the HTML5 standard, a client subscribes to server events through a similar approach with a chunked block. The connection is kept open after the initial request and the server immediately sends new events to the client through this connection. At the present time this technique is not supported by all browsers.

The WebSockets technology [9, 18], which is also a part of HTML5, is seen by many as the best solution proposed so far for asynchronous communication in the web. It is built on top of the HTTP protocol and provides functionality of classic sockets with duplex communication channels. At the moment of writing the present paper only drafts of the standard are available and the standard is not supported by all browsers and proxy-servers.

However, none of these schemes can be directly applied to the Web of Things for a number of reasons. Firstly, devices in WoT are often resource limited, thus it is not possible to implement protocols that require parsing and execution of Javascript code on a microcontroller.

Secondly, in traditional web a server is assumed to be ubiquitous in terms of available resources and supported protocols. In the WoT a service implementing RESTful API can operate on an end device. This device may also have its own preferences concerning the choice of particular communication protocol. As a result, a two-side agreement on a protocol is required.

Finally, smart devices often have changeable requirements to the quality of service and hence the protocol. For example, a battery-powered alarm sensor will have different preferences for a regular beacon message and a message signalling an alarm. The first one may be transmitted with energy conservation in mind, while the second one has to be passed as quickly as possible.

As it is stated in [19], different protocols have different characteristics in terms of delay, overload and energy consumption.

Considering all these factors, we propose to use an adaptive scheme of protocol choice, which has a handshaking procedure before the actual data exchange takes place. This handshaking procedure takes into account preferences of both sides and chooses the best protocol for a particular moment in time, according to a number of criteria.

As our current paper focuses on the idea in general, we omit the exact description of this protocol. We plan to devote one of our future works specifically to this implementation. In section 5 we give a brief overview of the current implementation that is being tested now.

4. Related works

Over the last 10 years the number of research activities on the topic of Internet of Things has been constantly increasing, as perspectives of the technology become more and more obvious. Below we give an overview of the works, which are most relevant to the topic of this paper.

A number of solutions have been considered as base protocols for the Internet of Things: JINI, UPnP, JXTA [6]. Appearance of WS-* web-services lead to several researches, which focused on their application to embedded systems and sensor networks [5]. Several lightweight WS-* implementations, such as DPWS (Device Profile for Web Services) and DNLA were later proposed.

The first works on the REST architecture in IoT addressed the problem of reusing existing web standards and technologies for integrating physical devices on the web-level and providing interactions between them [3, 4, 7]. For example, in [7], a set of basic components for the Web of Things is given, which solves the tasks of intercommunication, discovery, access control and others. Additional tools for web applications like COMET-technologies [8] and HTML5 [9] were also analyzed in relation to the Web of Things.

The recent concept of a Social Web of Things (SWoT) extends capabilities of the classic Web of Things, as it views physical devices as elements of social interactions. As it is shown in [10, 11], devices can communicate with each other, establish relationships, provide partial or full access and other features common to social networks.

A number of studies focus on discovery services. Many of them are devoted to the EPCglobal standards [12-14]. Beier et al. [13] proposed one of the first implementations of a simple extensible architecture for a discovery service based on EPCglobal. In [15] the authors review several disadvantages of the EPCglobal service, mainly concerning security issues, and propose their own advanced solution.

5. Testbed platform

We have developed our own middleware data platform [20] for the Web of Things, which includes the following main components:

1. Products. A product describes a set of devices through a meta-description. Each product has a unique identifier, a text description and a set of attributes/properties that devices possess. A product version mechanism is also implemented, which allows adding or removing properties inside a product, while keeping backward compatibility with older devices.
2. Things/Devices. Each thing has a unique identifier for external access, namely a URI, e.g. http://thinger.ru/api/things/mymeteo. Inside the system any device contains two blocks of data: static information (identifier, name, text description and others) and dynamic information – values of the properties described in a corresponding product. For example, information from a meteo station can be represented in JSON format as follows: {"Name":"Temperature","Value":"23","Updated":"2013-04-06T12:46:47.307"}
3. Data providers, which allow to integrate off-the-shelf devices from different manufacturers into the system. External devices are required to have a RESTful API and to use open authorization protocols like OAuth.
4. Data handlers enable building dynamic scenarios of data processing and inter-device communication using the JavaScript language. These scenarios are executed immediately after a device property changes.
5. Social providers are used for integration with existing social networks.
6. Widgets serve as graphical representations of devices. Widgets may be read-only (data visualization) and read-write. In the latter case data and commands can be send to the devices through a widget. A device can have multiple widgets inside the system, and at the same time, a widget may represent several devices simultaneously.
7. Access subsystem, which is used to manage access to devices. Users registered in the system can share their own devices with other users.

The unified platform described above allows to simplify implementation of discovery algorithms as communication with external systems is not required. With its architecture processing a search request on the server includes the following steps:

1. Find products and/or product versions that satisfy the request
2. Find devices with related products / product versions
3. Filter devices conforming to search query parameters
4. Return the resulting list of devices

In the near future we plan to make queries not only to a local database on the server, but also to external devices and external systems, which will allow us to obtain more up-to-date information.

We have deployed a test version of a Web of Things cloud platform (fig. 2), developed with ASP.NET, which at the current moment includes the following parts:

- RESTful Web-service, which provides a single interface to device data
- Web-site used for data visualization and an extended set of tools.

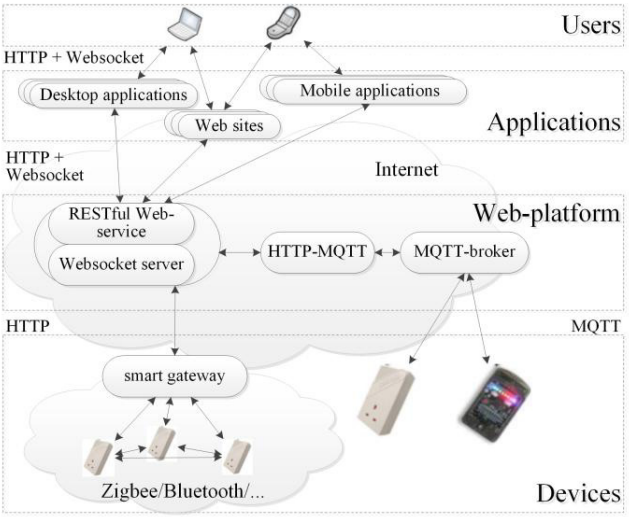

Fig. 2. Architecture of the developed WoT platform.

At the present moment the adaptive protocol, described in section 3, is implemented with the standard SignalR library [17]. It provides tools of automatic transport choice depending on client and server preferences. SignalR has predefined priorities for asynchronous communication strategies with WebSockets always being the first option and Long Polling - the last one.

At the next stage of platform development we plan to further extend functionality of dynamic protocol choice. This will include a more sophisticated handshaking procedure, which should take into account dynamic preferences of both parties.

6. Conclusion and future work

In this paper we have addressed two important problems that exist in modern Internet of Things and Web of Things. When smart devices are not preprogrammed or pre-set with links to each other, they require a discovery mechanism. When the actual data exchange starts, asynchronous communication schemes may be required. Using an adaptive protocol will satisfy dynamic requirements of devices.

The ideas presented in this paper can be further developed in several directions. First of all, none of the mentioned solutions were discussed in terms of security issues. Discovery services need to provide mechanisms to allow access only to authorized resources. End devices also need a way to check the identity of a remote party, thus authentication, authorization services as well as message signing algorithms are required.

Secondly, the proposed solution for device discovery requires a more detailed specification and testing on a global scale.

The adaptive protocol choice requires a mathematical model that would link main Quality of Service criteria with characteristics of particular protocols. The handshaking procedure also has to be specified in detail.

Acknowledgements

This research is supported by the Russian National Research University “Higher School of Economics” (Grant No 14-05-0064).

References

[1] L. Atzori, A. Iera, and G. Morabito. The Internet of Things: A survey. Computer Networks 54 (15), 2010, pp. 2787-2805.

[2] D. Evans. The Internet of Things: How the Next Evolution of the Internet Is Changing Everything. White Paper, Cisco Internet Business Solutions Group, April, 2011.

[3] D. Guinard, V. Trifa, E. Wilde. A resource oriented architecture for the Web of Things. Internet of Things (IoT), 2010 (Tokyo, Japan).
[4] D. Guinard, V. Trifa, T. Pham, O. Liechti. Towards physical mashups in the Web of Things. Proc. of the Sixth International Conference on Networked Sensing Systems, (INSS), 2009, pp. 1–4.
[5] N. B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao. Tiny web services: design and implementation of interoperable and evolvable sensor networks. In Proc. of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys). New York, NY, USA: ACM, 2008, pp. 253–266.
[6] B. Traversat, M. Abdelaziz, D. Doolin, M. Duigou, J. Hugly, and E. Pouyoul. Project JXTA-C: Enabling a Web of Things. In Proceedings of the 36th Annual Hawaii International Conference on System Sciences, 2003, pp. 282–290.
[7] D. Guinard. A Web of Things Application Architecture - Integrating the Real-World into the Web [dissertation]. ETH Zurich, 2011.
[8] S. Duquennoy, G. Grimaud, J.-J. Vandewalle. The web of things: interconnecting devices with high usability and performance. In Proceedings of ICESS '09, HangZhou, Zhejiang, China, May 2009, pp. 323-330.
[9] Si-Ho Cha, Yoemun Yun. HTML5 Standards and Open API Mashups for the Web of Things. Computer Applications for Web, Human Computer Interaction, Signal and Image Processing, and Pattern Recognition Communications in Computer and Information Science, 342, 2012, pp. 189-194.
[10] D. Guinard, M. Fischer, V. Trifa. Sharing Using Social Networks in a Composable Web of Things. Proceedings of the 1st IEEE International Workshop on the Web of Things (WoT 2010) at IEEE PerCom, Mannheim, Germany. pp. 702-707.
[11] L. Atzori, A. Iera, G. Morabito, M. Nitti. The Social Internet of Things (SIoT) – When social networks meet the Internet of Things: Concept, architecture and network characterization. Computer Networks 56 (16), Nov. 2012, pp. 3594–3608.
[12] S. Evdokimov, B. Fabian, S. Kunz, and N. Schoenemann. Comparison of discovery service architectures for the internet of things. Proceedings of IEEE Conf. Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC'10), Newport Beach, CA, 2010, pp. 237–244.
[13] S. Beier, T. Grandison, K. Kailing, R. Rantzau. Discovery Services – Enabling RFID Traceability in EPCglobal Networks. In: 13th International Conference on Management of Data, 2006, pp. 214–217.
[14] C. Kurschner, C. Condea, O. Kasten, and F. Thiesse. Discovery Service Design in the EPCglobal Network – Towards Full Supply Chain Visibility, in Proceedings Internet of Things (IOT 2008), Zurich, Switzerland, 2008, ser. LNCS 4952. Springer-Verlag, Berlin-Heidelberg, 2008, pp. 19–34.
[15] EPCglobal standards. http://www.gs1.org/gsmp/kc/epcglobal
[16] OData protocol specification. http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part1-protocol.html
[17] SignalR library. http://signalr.net
[18] Balamuralidhara, P., Misra, P. and Pal, A. (2013) Software Platforms for Internet of Things and M2M. Journal of the Indian Institute of Science, 93, 487-498.
[19] E. Estep. Mobile html5: efficiency and performance of websockets and server-sent events [Master's thesis]. School of Science Double Degree Programme NordSecMob, Aalto University, June 28, 2013.
[20] N. Pilipenko, L. Voskov. THINGER: Web-oriented platform for interaction between smart things. Proceedings of the 17th International Conference DCCN 2013, Moscow, Russia, October 2013, pp. 289-294.
[21] B. Gastermann, M. Stopper, B. Katalinic. Outline of an Autonomous Control Concept for Continuous Flow Production using RFID and Agent-Based Services, Annals of DAAAM for 2010 & Proceedings of the 21st International DAAAM Symposium, 20-23rd October 2010, Zadar, Croatia, Katalinic, B. (Ed.), pp. 0863-0864, Published by DAAAM International Vienna, Vienna.