MANAGING, SEARCHING, AND ACCESSING IOT DEVICES VIA REST RESOURCES

Jarogniew Rykowski[1]

[1]Department of Information Technologies, The Poznan University of Economics
Niepodległości 10, 61-875 Poznań, Poland

ABSTRACT

In this paper a new method is proposed for management of REST-based services acting as proxies for Internet-of-Things devices. The method is based on a novel way of monitoring REST resources by hierarchical set of directories, with the possibility of smart searching for "the best" device according to at-the-place devices' availability and functionality, overall context (including geo-location), and personal preferences. The system is resistant to changes of network addresses of the devices and their services, as well as core system points such as directories. Thus, we successfully deal with the problem of (dis)connectivity and mobility of network nodes, and the problem of a "newcomer" device trying to connect to the network at an incidental place/time.

Main novelty of the approach is a summary of three basic achievements. Firstly, the system introduces unifying tools for efficient monitoring. On one hand, we may control an availability and load (statistics) of devices/services. On the other hand, we are able to search for "the best" device/service with different criteria, also formulated ad-hoc and personalized. Secondly, the system is resistant to sudden changes of network topology and connections (basically IP addressing), and frequent disconnections of any system element, including core nodes such as central directories. As a result, we may have a common view to the whole system at any time/place and with respect to its current state, even if the elements of the system are distributed across a wider area. Thirdly, any element of the system, from simple devices to global directories, is able to self-adjust to evolving parameters of the environment (including other devices as a part of this environment). In particular, it is possible for a mobile "newcomer" device to interact with the system at any place and time without a need for prior installation, re-programming, determination of actual parameters, etc. The presented approach is a coherent all-in-one solution to basic problems related with efficient usage of IoT devices and services, well suited to the hardware- and software-restricted world of Internet of Things and Services. Fully implemented, the system is now being applied for an "intelligent" home and workplace with user-centric e-comfort management.

KEYWORDS

Internet of Things, Representational State Transfer, directories, orchestration, complex and virtual devices

1.INTRODUCTION

Recently we observe a boom of small computer devices, accompanying us at every step of our everyday life. There are more and more complex devices around us, and more and more "intelligence" is trying to help us at home and work. We may notice also some attempts to support outside activities, such as travelling, thus moving towards incidental and ad-hoc usage of at-the-place and ready-to-serve, miniaturized devices. This concept if widely known as Internet of Things [1] and Internet of Services phenomenon, with small and thus almost imperceptible

devices/service acting according to Mark Weiser's rule of "good servant" – useful, but not disturbing and not requiring to pay attention [2].

Located anywhere, invisible IoT devices raised some technical and organizational problems. The technical difficulties are mainly related to the interoperability (especially for mobile and personal devices), scalability, addressability and accessibility (precisely – an availability of a device to be contacted from a local network by another device or with a local user interface), and management (also the problems related with device malfunction and disconnection) [3]. The organizational (including psychological and societal aspects [4]) problems are basically related with the fear for privacy protection [5]. Even if the latter problems seem to be important, it looks like we should concentrate, during the first stage of development, on the technical ones. And among all of these problems, the problem of searching, addressing, and finally accessing the devices in the scope of incidental interactions is the crucial one.

Namely, what we would like to observe for an IoT-based system is:

- efficient method for device/service identification, also in ad-hoc mode,
- efficient searching for given functionality (offered in turn by the devices/services), also contextual and provided/required ad-hoc at given place/time,
- efficient orchestration of devices towards simple and complex services, also personalized and working according to ad-hoc requirements,
- self-management, including group- and functionality-based control and dynamic catalogues of IoT services based on devices' availability and offered/declared functionality,
- at-the-place incidental discovery of devices/services based on their availability and functionality, resistant to device malfunction and temporal disconnection.

The five above-mentioned requirements are mutually dependent and strongly correlated, thus a target IoT system should be able to manage all of them at once, in a consistent manner. Therefore we see a need for a single tool/framework to control globally all the above-mentioned aspects (e.g., searching for and activating only the currently accessible, local devices). We should deal with a set of individual installations, possibly linked with common knowledge/user interface, to be contacted in the same way regardless the place and time of operation.

In this paper we propose such a complex and consistent framework for the Internet of Things and Services. As the technical base, we chose REST architecture [6], with REST resources being proxies/servers to IoT devices and services (including orchestrated virtual devices/services) [7]. Thus, our approach relates to a method for managing, searching for, and sharing REST resources. The method is especially useful in ad-hoc data exchange among networked devices/services. Additionally, the present invention may be used for creating complex and personalized systems of Internet of Things/Services, in particular to be used incidentally at a random place/time according to BYOD ("Bring Your Own Device") idea [8].

We assume that IoT devices are very restricted according to both hardware and software, not to say about the communication link. Thus we deal with two basic modes of REST implementation: either built-in into devices, or working as a remote proxy to a limited device/connectivity (e.g., USB-connected hardware unit) with the help of an underlying computer or specialized controller. Typical REST framework practically offers no support for searching, group operations, service discovery etc. In stable environments (i.e., with fixed set of devices and services), this restriction may be bypassed by a parametrization and specialized external software. However, for ad-hoc interaction, these activities must be built-in into the local framework to efficiently utilize offered services. Thus a need for substantial enrichment of REST services arose, however, still keeping

the simplicity of the software/hardware of IoT devices. To this goal, we propose a REST-based complex architecture with dedicated layer above the services offered by IoT devices to control them both individually and in the group mode, with local centralized management and repository.

The reminder of the paper is organized as follows. In Section 2 we present some basic features and limitations of REST methodology. In Section 3 we present in details our approach to extend a REST-based framework to convert it into a complex management system. We address such problems as efficient searching, addressing, and activating methods for local IoT devices and services, to be used in ad-hoc (incidental) mode. In Section 4 we compare our approach with similar work, while Section 5 concludes the paper and points some directions for future work.

2. REST METHODOLOGY

Representational State Transfer REST methodology was proposed in 2000 by Roy Fielding, one of the main researches related to HTTP protocol [9]. REST defines the architecture and the rules for building a networking system that allows for very high scalability. This model is based on a resource identified by Uniform Resource Identifier (URI). Since each REST resource is represented by a separate address, it is not necessary that all the resources are in the same place, and each resource (or any subset) may be located on a different host. REST, in addition to the resources identified by URIs, is also a matter of representation - a resource may be interpreted as XML, JSON, text, image, etc. In contrast to other networked systems, especially those based on SOAP/WSDL/UDDI [10], REST implementations are very light, thus may be provided for limited hardware and software. In the matter of fact, REST resources, apart their own functionality, need only an implementation of HTTP protocol to operate. REST-based systems are scalable and may be freely distributed.

The most important element of each REST-based system is a resource. REST resource, within the spirit of the present technical concept, is any given, individually identified element of a system. Preferably, there is a basic rule stating that a single address must be equivalent to a single resource. A unified resource identifier (URI) may consist of a part comprising parameters (part of the address after the '?' character). The parameters do not influence resource identification but may change its behaviour. For example, a parameter value `"?language=de"` means that the language for the resource presentation is German. A resource may be also linked with a physical device, to which access will be realized by means of a REST server address that acts as an interface for data exchange with this device.

Each REST server comprises a given set of resources, meaning the server has associated number of unique addresses of which each identifies a given resource. The resources in REST addressing are grouped hierarchically, whereas a hierarchy is identified by successive sections of the URL address. For example a resource identified with `http://192.168.1.1/home/kitchen/door` represents access to a device associated with a door in the kitchen at home. The resources may be also addressed in groups. For example, an address of `http://192.168.1.1/home/kitchen/` will select all resources present in the kitchen, e.g., door, window, table, etc. Such group addressing is a kind of multilevel directory.

Each REST resource (as mentioned previously) has a defined name, which may not be necessarily standardized, and is periodically tested/monitored by dedicated REST servers – directories. Data gathered, by each server of a directory, may be used for statistical purposes and for searching for other servers having required features.

3. MANAGEMENT OF REST RESOURCES

In case of using servers, operating according to the REST paradigm, an application of standard methods for the management of a server and its content is not possible. The REST servers, which by definition are light servers, in contrast to typical WWW servers do not have built-in mechanisms for tracking, and so called log. The creators of the REST method have assumed that such servers have to be as minimal as possible with respect to the amount of programming code and resources used. Therefore, all mechanisms for controlling server's behaviour (or more precisely REST resources, since each resource shall be treated individually as an equivalent of WWW mini-server) shall be implemented by a designer of the end-system, if so required.

Due to the fact that all REST resources are treated independently, complex management of a group of REST servers (resources) of common use but different location/implementation hard to maintain. Additionally, in order to define a number and types of REST resources accessible in a given context (location), one needs to equip the target system with an appropriate directory, the location of which is publicly known to all potential users of the system. The creation and use of such directory is also not standardized. Therefore, a system implemented according to these principles has a very limited portability and does not meet the requirements posed in ad-hoc data exchange and other incidental methods of the interaction among electronic devices.

In general, the REST methodology has a number of limitations for ad-hoc device interaction. It is apparent that the REST servers, as currently known, do not meet the requirements posed by ad-hoc interaction. In particular, they do not meet the requirements of efficient administration of resources or their groups in multiple locations, efficient monitoring of resources state (for example availability, requests statistics or resources usage), automatic propagation of changes and contextual access to information from a directory. Moreover, there is not any system mechanism for finding a REST server within a predefined group of servers, which would be based on ad-hoc generated characteristics (for example resource usage, type, location etc.).

Therefore, as can be readily seen, enhancements of methods for managing, searching and sharing of REST resources would be very beneficial. Such enhancements shall be preferably useful in ad-hoc interaction.

The aim of our proposal is to alleviate the aforementioned drawbacks of the prior art and propose a method for managing, searching and sharing of REST resources. The object of the present invention is a method for managing representational state transfer (REST) resources, the method comprising the following steps:

- starting a REST server, and determining at least one of the available servers as a closest directory server, to register the server in this directory,
- accessing from the directory all the information about the environment, as necessary; this step comprises setting the addresses of key system elements (c.f., next sections),
- periodically exchanging statistical data between the directory and the server, in order to monitor current server’s state and availability,
- periodically updating directory state in the server (including address, load, etc.); this step may involve a change of the current directory, when a better one becomes available,
- broadcasting the information obtained from the directory to other servers, once requested.

Although the above-mentioned steps at the very first view look similar to any directory access and distributed-system monitoring, in case of REST environment they are not trivial in implementation and usage. For example, the second step is usually hard to achieve without prior knowledge of an exact address of a directory. To clarify the idea, in the next sections we unfold the steps and precise the activities that are undertaken to achieve all the proposed goals.

3.1. Basic assumptions for system implementation

Before we describe overall system architecture and functionality, we have to enumerate some basic assumptions we have made for system implementation and usage, these are shortly presented below.

- Each device (both real and virtual) has an associated REST resource. This resource acts as a proxy, i.e., it is the only representative of this device for the whole system. The resource and its corresponding REST server may be installed and executed at any host, provided that it is accessible in the local network via an IP address. The device itself may play the role of its own proxy, if it is equipped with built-in REST server.
- Each REST server representing any device has some dedicated REST resources for server management. These resources, with well-defined (and common for the whole system) names/parameters are addressable from the outside by any other server. These resources are the only standardized elements of the system. As for the other resources related with an access to devices, the designers are free to use any syntax/parameter list/domain values, on condition the access is realized according to the REST rules.
- Each server subscribes to a dedicated service called a directory, being in turn a REST resource. Typically, a single directory manages the servers from a local network; however, this is only a reasonable administrational restriction, not a system requirement. A directory is responsible for accessing, storing and distributing information about current state/parameters of the servers and, indirectly, their resources and related devices. To this goal, the directory uses above-mentioned dedicated resources for server management. Directories may be hierarchically connected, i.e., a lower-level directory may be maintained by a higher-level one, similar to the way any other server is maintained. Each directory, during its monitoring process, transmits to all monitored servers its statistical data and updates of its state (for example its current location). Based on these data, each REST server is capable of informing any other server: explicitly about its own state or directory's state, in which it is registered, and indirectly about current state of any given server, after consulting the directory.
- In our approach, information content, shared by the monitored REST servers, is standardized and transmitted preferably as an XML document.

Certain REST servers play specified roles in the system, implementing basic system services. The list below depicts typical servers (the content of this list is open for possible enhancements of system functionality):

- SMS (System Management Service) stands for above-mentioned directory for maintaining other REST servers of any type and purpose, including lower-level SMS services as well;
- DNS (Domain Name Service) plays a similar role to well-known Internet DNS service, i.e., is able to convert user-friendly addresses of the resources/servers to current IP addresses, despite the fact the addresses are fixed or temporal (i.e., obtained from a DHCP service);
- SNG (Serial Number Generator) is responsible for providing unique identifiers, needed, among others, for the registration of new devices and their declared capabilities [11]. Usually, a single instance of this service is provided for the whole local system, to avoid miss-

generation of identical identifiers. The service never generates twice the same identifier, and the identifiers are not reusable, even if their corresponding devices/servers are no more in use;
- OSL (Ontology Scripting Language) is an activation-point service, to declare and use capabilities of the devices (this process is depicted in more details in [11, 12]);
- LRS (Light Resource Server) repository is used for storing and accessing simple textual and graphical files, according to REST rules. Typically, LRS servers contain such information as configuration data, icons and similar small graphics, internationalized messages, etc. Although this information could be accessed by means of a typical WWW server or even a local file, using LRS services implies that all the information across the system is accessible according to REST rules – thus, both the device functionality and this information is accessed in exactly the same way, which usually makes the implementation much more clear for the designers and users. And last but not least, LRS servers may be maintained by an SMS directory, being the manageable part of the environment.

Detailed roles of some of the above-mentioned services will be described later in the text.

3.2. REST management resources

As already mentioned, we assume that for each REST server being a part of the environment has built-in, additional management resources. Other words, we replaced the generic "REST application" class by its extended version. Towards this goal, we developed a dedicated Java library, to be used instead of generic RESTful one. From the programmer's point of view, it is enough to derive applications for REST services and resources from the extended classes, backward-compatible. In addition, if the extended REST server is about to provide some statistical information about its usage, one has to include certain, well-defined calls to predefined methods of the extended class at the very beginning and very end of each procedure serving HTTP GET/DELETE/PUT/POST requests, to measure the period of serving a request.

There are three management resources being the core of the extended REST application class, these are enumerated below.

1/ Resource named "management", accessible via URL address "url/management" or "url/server" ("url" stands for an address of an OSL/REST server). This resource is responsible for server management and some statistics propagation (as described further in the text). Full range of the commands related with this resource, as well as response syntax and interpretation, is given in the next sections of this document. Most of all, this information is to be processed and disseminated by an SMS service of the framework (c.f., previous section).
2/ Resource named "default", accessible via URL address "url/default". This resource is a resource providing values (if any) of some predefined server parameters (so called "default" values), to be used by any software that depends on the server, e.g., graphical user interface. Each parameter is stored in a typical form ("name=value"), and the value is always a text (non-empty string of characters). Once a parameter is registered, its nonempty value is returned at request, via a parameterized call to the "default" resource. Requests for non-registered parameters are served as empty values with no error signalization. Similar to the previous resource, the commands related with this resource are explained further in the text.
2/ Resource named "configuration", accessible via URL address "url/configuration" or "url/environment". This resource is used for system management, i.e., to propagate a list of URL addresses of the monitored servers, in particular key servers such as SNG, OSL, and SMS services. Similar to the previous resources, the commands related with this resource are explained further.

If, for some reasons, it is not possible to extend a REST (or any other) server by the above-mentioned resources, one has to use a predefined wrapper, being a part of our extended REST library. The wrapper, given by a list of REST resources (i.e., URL addresses), is able to act as a proxy to these resources (or any external server), providing full compatibility with the environment. Note that the detailed functionality of a given wrapper instance depends only on the functionality of the corresponding REST resource. In such case, however, each access to a device always invokes two HTTP transfers, using the wrapper as a single point of redirection.

3.3. Overall system architecture

The way of exchanging messages among devices and hosts, as well as the way of addressing devices' functionality is determined by the main assumption of REST methodology – to include all the details about a request, and all its parameters, in parts of a URL address of a resource to be addressed by this request. As the IoT devices usually exchange small amounts of information, we assumed that HTTP GET method [13] will stand for the basic communication utility. This method is also well suited for human inspection, for three reasons. Firstly, both URL address and a response (in our case – mainly an XML document) are human-readable. Secondly, to perform a communication, a standard browser may be used (and similar, standard Java libraries). As for the latter, one may easily perform some manual tests just entering certain addresses in a browser and seeing the results at a screen. This fact greatly improves the way of preparing and testing some functionality. And last but not least, the response sent (as above-mentioned – preferably an XML document) may contain some additional information not necessary stating the called function itself, but also (among others) current device status and load, timings and other statistical data, current addresses of crucial system utilities, etc.

Surely, HTTP PUT, DELETE and POST methods [13] also apply. POST method is used in case there is a need to upload more complex information to a server, and DELETE – for a notification to remove some information from a server. However, these commands cannot be accompanied with a text response, thus no additional information may be sent (just error notification, if needed). It is up to a system designer to choose basic communication method for each device (and its corresponding REST resource) individually. Note once again that such choice does not affect any other part of the system – this is only a declared way of accessing the device, to be used solely while communicating with this device. Note also that for some very restricted devices it is much more easy to choose GET method for the communication, providing really minimum functionality and thus saving hardware and software resources.

As already mentioned, our approach is directory oriented, and SMS service stands for a directory (a catalogue) of all available at-the-moment and at-the-place devices and their services. Each SMS directory takes control over certain (local) area, with a possibility to link all such servers into a single hierarchy. In such way:

- if a server communicates with its local directory, a direct link is used (Fig. 1a),
- if a server tries to communicate with any other server (with not known address) except a local directory, two messages are to be exchanged: the first one to the directory, to the "configuration" resource, to find current address of the other server, and the second one to this server directly, using just-obtained address; this is a one-redirection-point traffic (Fig. 1b); note that next messages between these two servers are to be exchanged directly (Fig. 1a),
- if a server wants to locate a directory, and it knows only a location of any other server, the "configuration" resource of the other server may be used as the redirection point (Fig. 1c). Note that (1) a chain of such connected servers may be longer that two hosts, and (2) next messages are to be exchanged directly (Fig. 1a) as long as the addresses remain unchanged.

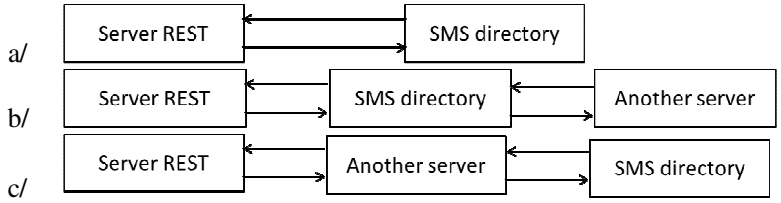

Figure 1. Exchanging information in the system: a/ server REST accesses a directory directly, b/ server REST accesses indirectly another server via a directory, c/ server REST accesses a directory indirectly via any other server

To realize the last communication method, we have to assume that each server already connected to the system knows a location of at least one SMS directory. In such case, it is enough to know just a single node from the network to possibly locate and access any other node, fetching first directory's address from such node, and then target address from just-located directory. Later on we will relax this requirement, in the section devoted to solving the "newcomer problem".

We perfectly realize that above-presented solution may affect overall system security, as breaking-through a single node provides an access to the whole network. However, (1) we bypass here safety&security discussion, as beyond the scope of this paper, (2) for our implementation we applied cryptography and standard authorization tools (such as PKI-based solutions) to improve the security, and (3) we assume our system will be executed mainly in a private network, with limited ways of entering the network from outside. Note also that majority of current IoT devices and systems, as well as network communication, are not secured at all.

Summarizing:

- the network is hierarchical, with sub-networks controlled by local SMS directories;
- each server chooses its direct SMS directory, usually based on a place of connection;
- any server may be a single-point of redirection to a local directory and local system services, and two-point redirection to any other server (such redirection is necessary only for the first message to be exchanged with this server, however).

There are three basic steps of an association of a server (and indirectly – its corresponding device) to SMS directory: registration (single step), update (continuous step) and searching/addressing (incidental step).

To register in an SMS directory, a REST server performs the following tasks:

- a determination of an address of the nearest SMS directory (according to schema from Fig. 1b or 1c),
- a transmission of an XML document stating server address and some additional parameters of its work (such as a unique system identifier, human-readable comment, etc.),
- a creation of an entry in SMS repository representing this server, indexed by its serial number,
- a determination of an availability-inspection period for the server (10 seconds by default).

Once registered, the server is searchable for any other server in the environment, including its SMS directory itself. However, a server may be occasionally disconnected, may be intentionally switched off for some time or may change its location – all these actions should be reflected in the SMS repository. To this goal, a monitoring mechanism is used, working in two modes: polling and pushing mode. In the polling mode, SMS directory periodically sends a message to

the "management" resource of each registered server and analyses the response. As already mentioned, the polling access is by default performed every 10 seconds (this parameter is configurable by system designer/administrator independently for each monitored server), which is a reasonable trade-off between data accuracy and network/device occupancy. If no response is obtained, the server is marked as "unavailable" – such servers are not searchable for any other server except the directory and its polling procedure. Otherwise, the response (an XML document, as already mentioned) is fetched and analysed. The response, except basic information about current server state (similar to "I'm alive" messages for well-known PING utility) contains also some statistical information about server usage (number of executions, average execution period, load, period of activity since last restart, etc.), to be used by other servers as parameters for the searching (i.e., "locate the fastest server of type X").

Executing polling procedure for a bigger repository and larger number of monitored servers would probably periodically provoke high load of the directory host. To avoid this situation, we assume that each monitored server sends its status to the "management" resource of the directory a moment before the planned polling activity. Such action – so called pushing mode – makes the polling activity inactive till the next planned moment, while all the information about the server is updated in the repository. Even if the number of transmissions is similar for the polling and pushing mode, the latter is better suited for a random distribution of updates in time, and for the limited hardware/software of the monitored servers – they may apply GET method for pushing, minimizing the number of bytes sent, and choosing the moments in time they are activated.

Despite the mode used, the directory informs the servers about its own state, basically – about current IP address. For polling mode, such information is included in URL (GET method) or sent as message body (POST), while for pushing mode sent as a response to each transmission initiated by a server. As a consequence, every few seconds the whole environment is updated with the information about current directory location.

Note that the monitoring is resistant even to on-line changes of an address of any server, including a directory, due to the complementary activities of polling and pushing modes. Imagine a directory changes its location. Each pushing message is lost. However, as the new directory has no information from the servers, it connect them using polling mode and thus informing them about the new address. Next time the servers push their status to the directory, they will use correct address, thus eliminating the need for further polling.

Imagine a monitored server changes its address. As this server still is able to communicate with the directory, it simply sends the new address together with next pushing notification, thus correctly updating the information in the directory.

The above-discussed combination of pushing and polling communication is not resistant to the global change of the address of the whole network, i.e., a simultaneous change of all the addresses of all the servers (including the directory) at once. However, we solved this problem with an additional communication mechanism, depicted later on in the section devoted to the "newcomer problem".

As for the latter activity – searching for the devices, this task is trivial. SMS directory, while requested, may provide a list of registered addresses of all the known and active at-the-moment servers, also filtered by certain features: type, name, comment, load, average execution period, moment of last activation, etc. Note once again that the system allows its users for accessing only these devices, for which there is a certainty that they are active at a given time.

Apart the monitoring task, a directory may play a role of a central point for server management. By means of "management" resource of any server, the directory may perform some administrational tasks, such as: suspending and resuming, stopping, and restarting. However, there is no way to start a server with such command – each command must be addressed to a running server via REST interface. Starting a REST server is a task for Proxy Manager module depicted later in the text.

To facilitate the tasks with server monitoring and management, specialized GUI is provided. With this utility, it is possible to identify a server based on its user-declared type, name, comment, as well as some statistical data (load, executions, etc.). Once a server (or a group of servers) is identified, it is possible to control the behaviour of this server by executing administrational commands, in particular suspending and resuming its activity, and to edit user-declared parameters, such as a comment or a human-readable name.

The advantages of the present invention include improved usability in ad-hoc interaction wherein there is not any possibility of applying static addressing and searching for servers and REST resources. In the presented solution it is enough that a server is aware of at least one address of another server in the network (i.e., this server can directly address the another one, this restriction will be relaxed in the next section of this document by means of UDP broadcast) and by the other server's inter-mediation the server may retrieve information regarding current state of any other server within the network. The initial server may also search for servers comprising selected features (e.g., statistical information about access time, load, etc.) or being of a given type. Further, the initial server may compare servers by means of various criteria including for example usage/load statistical information, timings, etc.

The system according to the present invention is especially beneficial in these circumstances where there are frequent configuration changes, and there are many servers of various characteristics and purpose whereas interaction with these servers is mainly effected by means of ad-hoc communication.

From a user's perspective the system allows for the unification of access to functions offered by servers without a need for generating a separate implementation for different locations. This opens a possibility for incidental use, especially in public places (under a condition of the standardization of types and methods of calling the REST servers), such as offices, museums, public-transport locations and hospitals.

An important feature of the system is that all of its elements are fully complaint with the REST architecture, including the directories, and all the information is gathered and shared as standardized XML documents. Such uniform approach allows for quick and efficient implementation of the system for any environment utilizing REST servers.

At the same time, the network traffic responsible for monitoring is minimized to the necessary minimum level.

3.4. Hierarchical structure of system management and accessibility

It is hard to imagine that all the IoT devices will be connected to and served by a single PC host only. Such a solution is good for a small installation, e.g., restricted to a single room or small house, but becomes impractical if the system scales up. On the other hand, there should be a mechanism to address some devices at a larger area. For example, imagine a building with several offices. Usually, the doors and windows in each office are to be controlled separately and

independently by at-the-place incidental users. However, in case of emergency, all the doors and windows should be opened with a single command, for the whole building. Even if probably used quite rarely, such possibility should be taken into consideration while preparing system architecture and overall organization.

As the basic mode of interaction is related with local interfacing, and the "remote" and "global" requests are rare, we propose to extend the single-directory architecture depicted in the previous sections by typical peer-to-peer (P2P) rules and distributed addressing [14]. To this goal, we propose to link several directories into a hierarchical graph. To address the directories within this hierarchy, we propose to apply well known time-to-live (TTL) rule/parameter [15]. In this approach, all local calls have TTL parameter equal to 1. This means that the request is restricted to a single node of the graph (single directory server). For global calls, one sets this parameter to certain value greater than 1. Each server, while processing the requests, distributes this request locally and decreases the TTL value by 1. If this value is still greater than 1, then this server propagates this request to any server directly connected, with the just-decreased TTL value, except the server that originally propagated the request (if any). In such way, the higher TTL value is, the wider is the range of the request, and the bigger is the possible number of the servers/devices to be addressed.

All the responses to the broadcasted request are collected by the forwarder of the request, and sent as a common response to the caller. Finally, the originating node collects all the responses of all its neighbours, acting as a "global" response to the initial request.

To limit the possible cycles in the forwarding of a request (the directory graph is not checked against internal cycles), each request is identified, and the past-request identifiers are collected for some time in each of the sub-directories. Once a newly coming request was already served in the past, this call is disregarded and no more forwarded. Thus, even if the graph of interconnected sub-directories contains cycles, these cycles are detected and never block the system.

In the same way we may obtain some global information about the network (statistics, information for the availability of certain-device or function, etc.), more precisely – about the local neighbourhood ("no longer than N connections from the selected node"). The more global is the request, the longer we must wait for the response, similar to typical P2P behaviour.

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

3.5. "Newcomer" problem and solution

So far we assumed that to contact the system at any place one must know in advance at least one address of any REST server already connected to the system. This server is then used to redirect the communication to the nearest directory. This is a problem for a typical IoT environment with ad-hoc users coming from the outside – such users have no information about any parameter/address of a local network which is incidentally near-by. Thus, as we already mentioned in the text, we clearly see that this restriction should be relaxed. To this goal, we

propose to apply UDP broadcast [16], with the information about the system disseminated at certain local port.

The solution is based on periodical transmission of the current (and local) IP address of the directory by the directory itself. Typically, a broadcasted message is sent every second. As the communication details of this broadcast transmission are publicly known (certain URL and port number), it is enough for a device to listen to this broadcast to achieve all the details about directory addressing within a second (maximum). Note that UDP-broadcasted traffic is usually stopped by the routers and gateways, so this is a local transmission only – each site may provide its own information, and this information will not interfere with the other messages depicting other localizations, even if very closed to each other. In such case no information about the local system is to be distributed outside the system, and the newly coming devices are not forced to remember in advance any local address. Everything that should be standardized is the broadcasting address/port – the parameters common for any local implementation of the system.

Apart efficient notification for the newcomers, broadcasting also enables on-the-fly change of the address of any server connected to the system, including the directory. Once the address of a server is changed, and this change is detected by this server, this server may try to connect with the directory, stating its new address. If this transmission succeeded, it means that the address of the directory is not changed; however, the directory knows the new address of the server since the moment of the transmission. If the transmission fails, this means that all the addresses of the local network are changed. In such case the server simply waits (no more than a second) to receive the new broadcast information, and, obtaining the new address for the directory, it is able to register its new address there. As a result, the whole network is resistant to the changes of any IP addresses, including total change for all the servers of the local network. The new addresses are known to all the servers in a part of a second, and the system is able to sustain and continue its work without the need for the restart of any node.

Note also that broadcasting makes it possible to replace directories on-the-fly. If a directory is switched off, it is enough to switch on a new directory for this local network (this task may be performed manually or automatically, as a result of detecting the fact of losing the communication link with the previous directory), and the information about current structure (and capabilities) of the system will be known to anybody (i.e., any device and any human using any personal device) within few seconds. And last but not least – unfortunately, broadcast transmission is sometimes not supported for limited hardware/software. In case only traditional HTTP-based transmission (TCP) is possible, one should rely on previously depicted addressing methods. Thus all the methods (asking any known server, pushing the changes – based on TCP traffic, and UDP broadcasting) should be used in parallel.

3.6. Grouped access and orchestration of devices/services

So far we abstracted of the way of serving the requests to activate IoT devices, linking a request with a given device. Surely, this way of the activation of the devices is not enough. Thus we propose three possible ways of the activation, namely:

- single-device request, to be served by a single IoT device independently of other devices; the device may be chosen by some parameters of the request (i.e., “open doors” request will open the doors at given geo-location), or by the context (i.e., “less loaded” device from a group, “any device” – random choice, “fastest” – the choice based on the statistics of usage collected by the directory, etc.);

- group request, to be served in parallel by the devices of the same purpose (e.g., "open the window" request addressed to all the windows); all these devices are contacted and activated in parallel and unconditionally, however, without mutual synchronization (the devices are not informed about the fact of an activation of any other device, within the same request);
- orchestration – cooperation of many different devices to achieve a common goal, e.g., "open the window on condition the outside temperature is not under 15°C" – involving one device for opening the window and one device for the measurement of the outside temperature; note that the first device is activated only for some cases, based on the value read from the "thermometer" device (which in turn is activated always).

The first two addressing modes were already presented in the previous sections – they are based on the addressing mechanism provided by the directory. Thus, only the third mode needs some more explanations here.

The problem of efficient linking of the devices into bigger and more powerful groups has been addressed many times, also at the level of web services (i.e., with device functionality observed as services). However, for most of the proposals, once orchestrated, a conglomerate of devices is still continuously treated as yet-another, however virtual, device. Complex devices are modelled according to their services/interfaces, abstracting their role in the real world. The complexes are hard-linked with their components, and as such hardly to re-use, e.g., at a different place.

We propose to model device conglomerates that abstract of hard links with devices (and other conglomerates). A conglomerate is a complex set of invocation definitions, with a possibility to control data flow among these invocations by means of typical control statements similar to an imperative programming language, such as a condition (IF-THEN-ELSE), a loop (DO-WHILE), variable definition (VAR, variables are not typed), sequence, etc. The computational power of the language to control this data flow is similar to the one of a typical operating-system shell – detailed syntax of the language is based on UNIX shells, such as Korn Shell [17] and BaSh.

Within a conglomerate, the ontologies are used to dynamically search for and filter out the devices according to the situation/request. In such way, a conglomerate abstracts of real devices, concentrating on high-level description of virtual functionality, mostly related with real-world object and situations. For example, a conglomerate for "opening doors" should dynamically detect a geo-location of the request, and further, based on ontology describing the functionality of in-range devices, search for and activate the exact device to open the doors. In any case, the conglomerate contains no hard links to any devices.

A conglomerate may be executed within given context, being a set of "name=value" textual parameters. A named connection of context parameters and a conglomerate is called an activity. Activity is the basic unit of programming – this is a counterpart of a typical procedure/method of an imperative language. Such as a procedure, an activity is identified by unique name, has a set of formal parameters declared with possibly initial values, and may be called (by name) within certain context (i.e., actual values of all formal parameters, as well as some additional parameters). Activities may be invoked from external (incoming requests), or as a result of an invocation from any other activity. In any case, the invocation is realized with a set of formal/actual parameters, some of them being a part of the context (as previously mentioned).

To represent IoT conglomerates and activities, we developed a dedicated XML-based language called Ontology Scripting Language (OSL) and a graphical tool called Activity Manager (AM), equipped with an interpreter and step-by-step debugger [12]. Computational power of this language is similar to a typical scripting language. We intended to facilitate a declaration (and

further usage) of the conglomerates/activities, thus our goal was to provide only the graphical way of programming of conglomerates' structure and behaviour. Fig. 2 presents a screenshot of AM utility, emphasising pull-down contextual menu for the manipulation of program code.

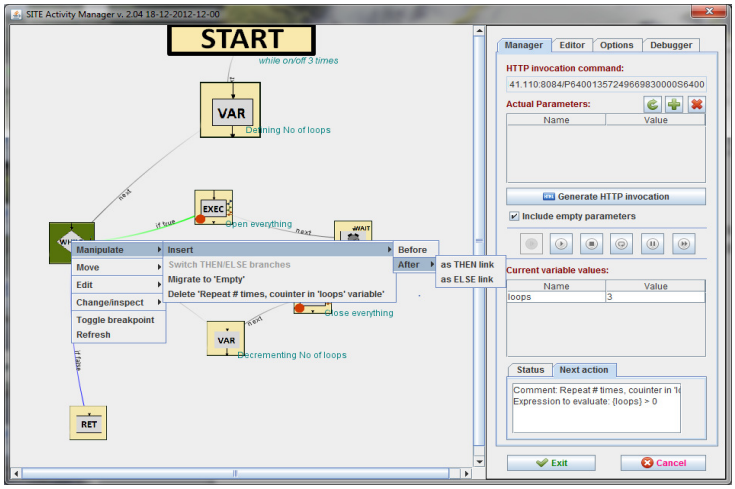

Figure 2. Programming and debugging a conglomerate/activity with Activity Manager

Surely, the above-presented figures and descriptions provide only a first impression on OSL and Activity Manager. However, a detailed presentation of these elements is not the core for this publication. Similar to any imperative programming language and corresponding tools and utilities, preparing the code of the conglomerates and further activating them is quite complex and strongly application-dependent. Here would like only to prove that our proposal effectively deals not only with simple, but also with complex, "virtual" devices and services.

4. SIMILAR WORK

After reading the text one may ask a simple question – why to search for a new tool for server management, and not to apply one of the plenty of already existing solutions? There are several proposals, both commercial and from the public domain, to efficiently manage any set of Web servers (such as management utilities built-in into any Apache/Tomcat implementation), and to control the content of Web pages (CMS frameworks, such as WordPress, Drupal [18] or Joomla). The latter systems are also used for dynamic generation of Web pages (which is in the matter of fact useless for IoT applications, as the responses are generated in majority by the devices and services themselves, with limited external control). However, all of these solutions have two basic disadvantages, which completely disqualify them for the incidental usage in the Internet of Things and Services. Firstly, none of them (well, almost none – some partial proposals will be discussed later in this section) supports REST servers, not to say about individual REST resources. This restriction comes from the fact that the REST methodology keeps in target maximum simplicity of the system, not allowing for any complicated solution at the server-side. Second, none of the solutions is resistant to frequent and incidental changes in the network structure, which is common for IoT networks and ad-hoc usage. The proposed monitoring systems are not ready for frequent disconnections of the services, for temporary problems related with device activity and availability (due to extremely simplified hardware and software, some devices cannot serve many requests in parallel, forcing some calls to wait for a long time), they do not distinguish operation mode of the server and its corresponding device (e.g., proxy server is

on, but its device is not operational at the moment – classically, such problems must be detected and further analysed manually). Moreover, even sophisticated searching tools cannot provide efficient group addressing (based on different parameters, including geo-location) and the orchestration. As a result, all of the already-proposed solutions addressed to the Web are useless for IoT applications. To our best knowledge, up to present time, there has not been provided a generic solution for the complex problem of gathering, processing and sharing administrative data regarding REST resources addressed to a dynamic application area such as Internet of Things and Services.

Similar problem – efficient searching for a service in the network based on its description, was also addressed many times. There are two basic methods of the implementation of searching procedure – based on an external directory, with central point of management, and directory-less (freely distributed). We already discussed the most popular and known solutions in [11], stating that Web- and SOA-oriented directory-based solutions are not flexible enough to deal with dynamic changes and ad-hoc cooperation with IoT devices/services, and directory-less methods cannot be applied to very hardware- and software-restricted world of small IoT devices, mainly due to limitations in network traffic and limited possibilities for temporary caching the information and meta-data in the network nodes. Thus, we clearly see a need for an efficient and coherent mechanism for (1) self-discovering how to access an unknown-in-advance network in ad-hoc mode, (2) searching for "the best" REST service related to IoT device, based on some criteria, including service functionality and some contextual information, such as usage statistics, network timings, etc., (3) addressing the service, also in highly dynamic mode, and finally (4) activating the service. Our solution is a proposal towards such mechanism.

We may also enumerate some proposals towards efficient monitoring of REST services, these are discussed below. Although this is only a part of our proposal, the knowledge about the state of the services is crucial for efficient usage of them. Note only, that in our case we monitor independently a state of a REST service, acting as a proxy to an IoT device, and the state of the device itself, while for the other solutions what is monitored is only the service state. Thus, even if some of the below-presented systems and tools at the first view are interesting, in the matter of fact they are hardly applicable to the Internet of Things. Anyway, we briefly discuss some proposals to find out common points and basic differences, in comparison with our solution.

As previously explained, the developers of the REST method envisaged the use of an external directory for tracking state and availability of REST resources. In general, presently available solutions have limited applicability (for example monitoring of specific resources with static addresses to be a'priori known) and require manual subscription of the monitored resource to a directory. Such an approach rules out the use of a directory in ad-hoc interaction wherein neither location of the directory nor potentially useful resources are defined in advance and shall be dynamically defined taking into account the context (for example a geo-location).

According to directory-centric idea, the work of Burke et al. [19] discloses a data structure which, given an identifier for a REST resource, can rapidly yield a configured target and can simultaneously yield all configured pattern based rules and constraints for the target. The disclosed data structure is a tree structure including nodes for URL portions. Each node is associated with a hash tree specifically grown in a manner that ensures collision occurrences are non-existent. The tree structure is effectively two or more superimposed trees; one for URL pattern matching to determine a target, another for determining constraints. A single tree traversal, which can be based on a progressive hash, can be used to concurrently determine a target and a set of constraints, which represents improved performance over conventional implementations that require multiple, distinct query/response operations to produce equivalent

results. The solution is a kind of a map making it possible to re-construct an address of a REST resource based on XML descriptors and resource characteristics. This is an interesting proposal, however, cannot be applied to any dynamic application and is not able to monitor current device/service state while rebuilding the address.

Burke's proposal is in the matter of fact a method for mapping URL addresses to REST requests (address of a REST server that is responsible for a given document or a group of documents). It is not a directory-driven method and does not allow for searching for a REST resource. It only allows for switching one static address to another static address. The solution does not allow changes in network structure, is not sufficiently resilient to errors and requires the knowledge of server's URL address that shall be called. This solution does not fulfil the needs of an ad-hoc driven system in which the number of nodes and their locations rapidly change. There are no provided adequate monitoring capabilities that could make the system highly dynamic. The system has a static network structure and that structure is known to devices addressing the available REST resources. The structure of the network is constant, thereby inadequate for dynamic adaptation of addresses to changing operational circumstances.

Algermissen depicted a smart method of using standard DNS service to search for and contact REST resources [20]. The idea is based on a typical DNS map, however extended in such way the map points to full URLs rather than simple host names/addresses. Thus, it was possible to apply DNS for multi-service hosts. Even if interesting at the very first view, the idea completely abstracts of the DNS implementation, in particular, timings and caching. Thus, it may be applied only for a local network, with a fixed structure. It is not possible to extend this idea towards dynamic DNS, monitoring of server/resource state, mobile and frequently disconnected nodes, etc. Moreover, any change in any address (including DNS-like name) is to be propagated after server restart, which makes the approach practically useless for any dynamic application.

We should also mention here some techniques for remote testing of REST services, such as AlertFox [21] (for which REST is only one of the technologies to be applied) and AlertSite [22], based on (1) sending a sample request, and (2) comparing the obtained result with a desired pattern. In addition, the testing service is able to estimate target-service performance, based on timings of the call. Similarly, specialized REST services are used to monitor the state of their corresponding Web servers, such as for GlassFish solution by Oracle [23]. Oracle also implemented WebLogic tools to monitor usage statistics for REST servers and even individual REST resources [24], but this solution is restricted only to Oracle-based software.

Hydra [25] is an example of a complex system dealing with the problem of heterogeneity of IoT devices, with a solution based on SOA mechanisms. The approach is quite complex, however, the solution is not optimized towards light REST services – it rather comes towards a huge centralized system for connecting every device all around. As only standard searching strategies are used, the solution poorly addresses the problem of device unavailability, disconnections, migration, etc. Moreover, the problem of the "newcomer" is not solved. The latter problem was partially addressed by Guttman et al. [26] as a part of Service Discovery Location SDP. The problem was solved by multicast messaging and DHCP-based broadcast. However, this solution was devoted to quite fixed networks, with stable addressing and location of the nodes. The goal was to advertise the services in the network, rather than monitor their addressability and availability. Our proposal goes much further, providing the monitoring almost in real-time, as well as enhancing the capabilities of a directory by the measurement of service load/delays.

Similarly, Garcia et al. [27] proposed to apply SOA-based modelling and service discovery to the Internet of Things. Searching for the devices is based on modelling the functionality by means of

WSDL, BPEL and BPMN descriptions, with the communication based on SOAP protocol. Indeed, the approach makes it possible to address the services in more abstract way. However, this idea is poorly linked with the very limited world of REST resources and devices. Moreover, similar to the previously presented solutions, the approach is not resistant to frequent changes in network configuration (which is certainly not a part of the service configuration, as the authors claim), in particular IP addresses of the nodes, and the mobility of devices. In the matter of fact, in this approach REST resources are used as "slaves" for the upper-level SOA services, with limited control and monitoring of their state. Similar, in [28] a smart connection is proposed of REST- and SOAP-accessible devices and the information cloud. However, this proposal lacks efficient searching mechanism and does not address the problem of a newcomer.

Guinard's work [29], perhaps the most similar to our approach, goes towards a complex directory-like service, involving device registration at a server, including memorizing device's capabilities, "advertising" of device possibilities to other devices, loosely declaring the real-life objects related with devices, and pairing the requests with the capabilities to find a device to fulfil the request. However, this is only the very first stage of the work, and the ontologies are freely mixed with some semantics hidden in e.g., descriptions and user-declared names of the devices, such as a "printer", "lighting", etc. No implementation details are given for this work, and the proposal abstract of complex devices and device orchestration. The Authors also further discuss the idea of applying some SOA-based standards to IoT monitoring [30], stating that the REST-based approaches are better suited for the restricted world of Internet of Things; however, they lack security and standardization.

The problem of geo-searching for devices (localization problem) was also addressed several times, a good survey on this topic may be found in [31]. However, most localization services are quite static (as related to wireless-sensor networks) and do not address the newcomer problem.
In general, all these monitoring tools generate huge network traffic, are hardly programmable towards light REST services (they are mainly optimized for monitoring Web servers), and are not linked with a complex searching tool. Moreover, what is monitored is the availability of a service rather its internal state, and no context (such as geo-location) is taken into consideration. Thus, it is not a trivial task to obtain meaningful information about all the IoT devices in-range and available at-the-place.

In [7] we compared our approach with its well-known and widely used competitor – OSGi framework, as well as a distributed extension of this system – OSGi-R. We stated that OSGi-based approach is better suited for sensor networks (the applications covering homogeneous devices and fixed, predefined system functionality, usually related to a single location and local network, with wide code re-use), and a REST-based framework is more useful in ad-hoc, dynamic environment achieving heterogeneous devices and services, also in a distributed environment. Our proposal also provides proxies and efficient orchestration for distributed, heterogeneous devices, to provide complex services required by an IoT network, while this is not the case of OSGi-based solution.
An interesting OSGi-based solution towards searching for and orchestrating IoT devices in a small environment such as single home is given in [32]. This is typical platform-centric approach, with OSGi playing the role of core system point, and some enhancements towards SOA-compliant services over this core, playing with internal OSGi resources. Similar to the above-mentioned generic discussion about OSGi, this system is however hardly distributed, thus the solution is restricted to a small local network. Moreover, the XML descriptors used to identify the services are an informal semantic description, however, not standardized. Also, very few implementation details are given, and neither an interface with humans and IoT devices nor context-aware usage is discussed.

For the purposes of verifying an availability of a REST service, some programmers also applied a trick called "long polling". With a dedicated IP socket one opens a connection to a service, but never finishes it. Open connection means "I'm still alive", and once the service is down, the connection is closed. Surely using this approach one is not able to determine service characteristics (such as load and timings) and compare the results (as there is no traffic coming from the monitored service). This approach generates small network traffic, however, is hardly applied to non-parallel implementations of small IoT devices (an open connection usually blocks any other connection with the same network node). Open connections also forces the devices to be always active, thus consuming the energy all the time.

Summarizing state-of-the-art discussion, we may say that there are plenty of monitoring tools for WWW servers and files, however, almost none for distributed REST servers, and there are some solutions for REST-based SOA services, however, these are developed for rather complex systems and not suited for limited hardware and software of IoT devices. However, no solution addresses the problem of sustainability after the change of IP address (all are working with a set of fixed, manually provided addresses to monitor). There is also no support for frequent disconnections of REST-accessible devices and for monitoring the state of devices rather than their corresponding REST services, and no solution addressing the problem of a newcomer. So, it looks like there is not a single complex solution dealing with all the problems mentioned in this paper, and our proposal is a significant step in this direction.

5. CONCLUSIONS AND FUTURE WORK

In this paper we propose a new uniform approach for accessing IoT devices and services by means of REST servers and resources. The approach is a complex solution to the following problems:

- device/service identification, based on described/required functionality and some contextual information (such as geo-location and at-the-place availability),
- efficient searching for "the best" (from the point of view of an end-user) device/service,
- group addressing and activation of several devices, also in the orchestration (programmable and conditional) mode,
- self-management of the devices including reporting about their availability and internal state, with a central directory as the main dissemination point,
- resistance to frequent changes in network topology and neighbourhood, disconnections, state changes, etc., connected with a possibility of ad-hoc incidental interaction at given place and time,
- massive usage of mobile personal devices, also at unknown-in-advance places and situations (so called "newcomer" problem).

Main novelty of the approach is a summary of three basic achievements. Firstly, the system introduces unifying tools for efficient monitoring. On one hand, we may control an availability and load (statistics) of devices/services. On the other hand, we are able to search for "the best" device/service with different criteria, also formulated ad-hoc and personalized. Secondly, the system is resistant to sudden changes of network topology and connections (basically IP addressing), and frequent disconnections of any system element, including core nodes such as central directories. As a result, we may have a common view to the whole system at any time/place and with respect to its current state, even if the elements of the system are distributed across a wider area. To deal with efficient addressing within a distributed installation, we apply

P2P-based solutions such as Time-to-Live TTL range. And last but not least, any element of the system, from simple nodes related with particular devices to global directories, is able to self-adjust to evolving parameters of the environment (including other devices as a part of this environment). In particular, it is possible for a mobile "newcomer" device to interact with the system at any place and time (incidental access) without a need for prior installation, re-programming, etc.

The presented approach is a coherent all-in-one solution to basic problems related with efficient usage of IoT devices and services, well suited to the hardware- and software-restricted world of Internet of Things and Services. Fully implemented within the scope of SITE (Semantic Internet of Things Environment) framework, the system is now being applied for an "intelligent" home and workplace with user-centric, contextual [33] e-comfort management [34]. The SITE implementation was the base for two EU patent applications, the one devoted to ontology-based service discovery [35], and the second one being the base for this paper [36].

ACKNOWLEDGEMENTS

This work was supported by the GOLIATH project jointly funded by the Poland NCBR and Luxembourg FNR Lead Agency agreement, under NCBR grant number POLLUX-II/1/2014 and Luxembourg National Research Fund grant number INTER/POLLUX/13/6335765.

REFERENCES

[1] K. Ashton, That 'Internet of Things' Thing, RFID Journal, 22 July 2009
[2] M. Weiser, "The computer for the 21st century", in: Scientific American vol. 265, 1991, pp. 94-104, doi: 10.1038/scientificamerican0991-94
[3] Internet of Things – Architecture IoT-A, Project Deliverable D2.5 - Adaptive, fault tolerant orchestration of distributed IoT service interactions, http://www.iot-a.eu/public/public-documents/documents-1/1/1/D2.5/at_download/file
[4] A. Meijer, B. Koops, W. Pieterson., S. Overman, S. ten Tije, Government 2.0: Key Challenges to Its Realization, Electronic Journal of e-Government Volume 10 Issue 1 2012, pp. 59 -69)
[5] J. Scholl, Five trends that matter: Challenges to 21st century electronic government, Information Policy Volume 17 Issue 3 2012, pp. 317-327
[6] Representational State Transfer REST, based on Wikipedia, http://en.wikipedia.org/wiki/Representational_State_Transfer
[7] J. Rykowski, D. Wilusz, Comparison of architectures for service management in IoT and sensor networks by means of OSGi and REST services, Annals of Computer Science and Information Systems, ISBN 978-83-60810-58-3, 2014
[8] "Bring Your Own Device BYOD idea", Gartner, Inc., from http://www.gartner.com/technology/topics/byod.jsp, last access March 2014
[9] R. T. Fielding, R. N. Taylor, "Principled design of the modern Web architecture", in: ACM Transactions on Internet Technology vol. 2 issue 2, New York: ACM, 2002, pp. 115-150, doi: 10.1145/514183.514185
[10] Service-Oriented Architecture (SOA) Definition, http://www.service-architecture.com/articles/web-services/service-oriented_architecture_soa_definition.html
[11] J. Rykowski, Addressing IoT devices by means of semantic descriptions of their functions, submitted for IEEE Internet of Things Journal (currently under review)
[12] J. Rykowski, P. Hanicki, and M. Stawniak, Ontology Scripting Language to Represent and Interpret Conglomerates of IoT Devices Accessed by SOA Services , in: SOA Infrastructure Tools - Concepts and Methods, ed. Ambroszkiewicz, S., J. Brzeziński, W. Cellary, A. Grzech, and K. Zieliński, UEP Press, Poznań, 2010, pp. 235-262, ISBN 978-83-7417-544-9

[13] Hypertext Transfer Protocol -- HTTP/1.1, draft-lafon-rfc2616bis-03, http://www.w3.org/Protocols/HTTP/1.1/rfc2616bis/draft-lafon-rfc2616bis-03.html
[14] R. Rodrigues, P. Druschel, Peer-to-Peer Systems, Communications of the ACM, Vol. 53 No. 10, Pages 72-82
[15] Time to Live (TTL) parameter for TCP/IP, http://www.maxi-pedia.com/time+to+live
[16] Understanding TCP and UDP discovery methods, http://c353616.r16.cf1.rackcdn.com/UnderstandingTCPUDP.pdf
[17] Home Page For The KornShell Command And Programming Language, http://www.kornshell.com/
[18] Drupal – open source CMS, project homepage, https://www.drupal.org/
[19] T. C. Burke, A. S. Reddy, A. Srinivasan "Technique for finding rest resources using an n-ary tree structure navigated using a collision free progressive hash", US patent application 20090164485, 2009, http://www.google.com/patents/US20090164485
[20] J. Algermissen, Using DNS for REST service discovery, May 2010, form http://www.infoq.com/articles/rest-discovery-dns
[21] Home page for AlertFox service by IPSwitch, http://alertfox.com/help/
[22] Home page for AlertSite UXM service, http://smartbear.com/product/alertsite-uxm/overview/
[23] GlassFish solution by Oracle, http://docs.oracle.com/cd/E18930_01/html/821-2416/gjipx.html
[24] Monitoring RESTful Web Services Using WLST, part of Oracle WebLogic Server documentation, http://docs.oracle.com/cd/E24329_01/web.1211/e24983/monitor.htm#RESTF196
[25] M. Eisenhauer, P. Rosengren, and P. Antolin, Hydra: A development platform for integrating wireless devices and sensors into ambient intelligence systems," The Internet of Things, pp. 367-373, 2010
[26] E. Guttman, C. Perkins, J. Veizades, , M. Madison, Service Location Protocol, v.2., IEEE Internet Computing, July/August 1999, pp.71-80, from http://www.academia.edu/3454281/Service_location_protocol_Automatic_discovery_of_IP_network_services
[27] A. G. García, M. Á. Álvarez, J. P. Espada, O. S. Martínez, J. M. C. Lovelle, C. P. G-Bustelo, Introduction to Devices Orchestration in Internet of Things Using SBPMN, International Journal on Interactive Multimedia and Artificial Intelligence, ISSN 1989-1660, Special Issue on Computer Science and Software Engineering, Volume 1, Number 4, 2011, pp. 16-22
[28] R. Piyare and S. R. Lee, Towards Internet of Things (IOTS): Integration of Wireless Sensor Network to Cloud Services for Data Collection and Sharing, International Journal of Computer Networks & Communications (IJCNC) Vol.5, No.5, September 2013
[29] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, Interacting with the SOA-Based internet of things: Discovery, query, selection, and on-demand provisioning of Web Services, EEE transactions on Services Computing, vol. 3, no. 3, pp. 223-235, 2010, ISSN: 1939-1374
[30] D. Guinard, I. Ion, S. Mayer, "In Search of an Internet of Things Service Architecture: REST or WS-*? A Developers' Perspective", in: Mobile and Ubiquitous Systems: Computing, Networking, and Services, A. Puiatti, T. Gu, Eds. Berlin Heidelberg: Springer-Verlag, 2012, pp. 326-337, doi: 10.1007/978-3-642-30973-1_32
[31] A. Mesmoudi, M. Feham, N. Labraoui, Wireless Sensor Networks Localization Algorithms: A Comprehensive Survey, International Journal of Computer Networks & Communications (IJCNC) Vol.5, No.6, November 2013
[32] A. Bottaro, A. Gerodolle, P. Lalanda, "Pervasive Service Composition in the Home Network," 21st International Conference on Advanced Information Networking and Applications, 2007. AINA '07, 21-23 May 2007, Page(s): 596 – 603, ISSN : 1550-445X, ISBN: 0-7695-2846-5, DOI: 10.1109/AINA.2007.112.Iotmanagement
[33] Y. Naudet, Reconciling context, observations and sensors in ontologies for pervasive computing. In Sixth International Workshop on Semantic Media Adaptation and Personalization (SMAP), pp. 3–8, 2011
[34] B. Gateau, J. Rykowski, Personal e-comfort modelling and management based on Multi-Agent System and Internet of Things network, Proceedings of 5th International Joint Conference on Pervasive and Embedded Computing and Communication System PECCS'15, Angers, France, February 2015
[35] EU Application No/Patent No: 12461550.1-1525, 12.12.2012, Title: Method for devices addressing within a network

[36] EU Application No/Patent No: 12461551.9-2211, 12.12.2012, Title: Method for managing, searching and sharing of representational state transfer (REST) resources

Author

JAROGNIEW RYKOWSKI received the M.Sc. degree in Computer Science from the Technical University of Poznan, Poland in 1986 and the Ph.D. degree in Computer Science from the Technical University of Gdansk, Poland in 1995. In 2008 he received habilitation degree from the Institute of Computer Science, Polish Academy of Science (Warsaw, Poland).

From 1986 to 1992 he was with the Institute of Computing Science at the Technical University of Poznan. From 1992 to 1995 he worked as an Assistant in the Franco-Polish School of New Information and Communication Technologies in Poznan. In 1995 he became an Associate Professor in the School. Since 1996 he has been with the Poznan University of Economics, working as an Assistant Professor in the Department of Information Technology.

He participated in several industrial projects concerning operating systems, networks, programming language compilers (assemblers, LISP), multimedia databases and distributed systems for e commerce.

His research interests include software agents, with special emphasis put on personalized access to WWW servers by means of mobile and personal devices, and telecommunication networks. His recent interests have gone towards applications of Internet of Things and calm-computing devices, including "intelligent buildings and workplaces", semantic support for IoT systems, telematics, ad-hoc and multi-hop networking, and similar systems. He is the author and co-author of 3 books, over 45 papers in journals and conference proceedings and 2 patents.