From a Smart House to a Connected City: Connecting Devices Services Everywhere

João Paulo Cardoso de Lima[1], Leandro Buss Becker[2], Frank Siqueira[3], Analucia Schiaffino Morales[1], Gustavo Medeiros de Araujo[1]

[1]Department of Computing
Federal University of Santa Catarina (UFSC)
Araranguá, SC – Brazil

[2]Department of Automation and System
Federal University of Santa Catarina (UFSC)
Florianópolis, SC – Brazil.

[3]Department of Informatic and Statistics
Federal University of Santa Catarina (UFSC)
Florianópolis, SC – Brazil
joao.pcl@grad.ufsc.br,
{leandro.becker, frank.siqueira, analucia.morales, gustavo.araujo}@ufsc.br

Abstract. *The growing development of smart devices makes it possible to create new distributed applications targeted for smart spaces. The design of intelligent spaces assumes that there is an infrastructure to support the applications requirements. Many academic works have proposed middlewares that provide an abstraction for the use of network services. The network services of an smart space, such as an automated home, can have different communications interfaces. Accordingly, we developed a middleware called UDP4US (Universal Device Pipe for Ubiquitous Services) which was designed to abstract different patterns of communication, keeping the discovery of devices on a local network services. In this paper, we present a new UDP4US architecture component that aims to expose the local network devices services to the Internet. The new component was developed with the REST technology, thus the devices services can be discovered and accessed over the Internet. The new component was exhaustively tested in order to find the limits of its effectiveness. The evaluation of the new component was performed by measuring its discovery and execution times plus the success rate of the services execution exposed over the Internet. The results from the present work are important to guide a better design of distributed applications for smart places.*

1. Introduction

Ubiquitous computing requires binding inexpensive computing device to larger computers and others resources using, preferably, wireless medium. For example, a smart house controlled with smart devices shall have remote control of home lighting, integrated entertainment systems, messaging services, and should be able to monitor the health conditions of the house inhabitants. Mark Weiser in [Weiser 1991] proposed the idea of injecting computation capacity into the physical world with high spatial density and invisibility, by having nodes and collectives of nodes operating autonomously, i.e., the technology

should be part of everyday life until they are indistinguishable from it. In general, the ubiquitous computing devices are composed of hardware platforms, operating systems, network protocols, interaction substrates, applications, privacy, and computational methods. The main concern in this area has been the integration of the digital artifacts with the physical world [Kumar 2009].

In the context of ubiquitous computing, devices interaction has been traditionally addressed within a local network [Gubbi et al. 2013]. However, nowadays, this concept is extended in a way that devices can be reached through the Internet. This new concept is known as the Internet-of-Things (IoT) [Atzori et al. 2010]. A wide variety of devices is available as a consequence, a great amount of different communication interfaces are under use. This is, in fact, a barrier to provide proper interaction between different devices. To solve this problem, efforts have been made to describe protocols that allow different classes of devices to operate together.

Some protocols have been employed in the ubiquitous computing, such as DPWS (Device Profile for Web Service)[1] which is designed to simplify the integration between web services and device-provided services. DPWS brought SOA concepts to the device level [Cândido et al. 2010], allowing devices to expose their services. This enables a wide applicability, ranging from industrial to home automation [Sleman and Moeller 2008]. The DLNA (Digital Living Network Alliance)[2] is a collaborative standard organization that defines interoperability guidelines to enable devices to communicate in a home network. The DLNA aims the sharing of multimedia content between producer and consumer devices. However, the DLNA is not limited to sharing multimedia content, as it can be used with the same purpose as DPWS. Given that both DPWS and DLNA use UPnP as underlying protocol, they can be combined in order to allow interoperation between heterogeneous devices and the development of several application scenarios.

Additionally, the REST (REpresentational State Transfer) technology has been widely used to integrate systems. The REST is an architectural style [Fielding 2000], which is being adopted in the context of Web-based services and applications as a strategy to provide interoperation system. The REST technology can also be useful in the context of ubiquitous systems, due to its lightweight and stateless nature, REST can be seen as a suitable solution for device integration [Guinard et al. 2010, Riedel et al. 2010, Moritz et al. 2010].

Firstly, this paper presents the Universal Device Pipe for Providing Ubiquitous Services (UDP4US) [Felisberto et al. 2015] emphasizing its new component. The new component is the REST module, which is the main contribution presented in this paper. The REST component is responsible for exposing and execution device's services over the Internet. The remainder of this paper is structured as follows. Section 2 outlines some issues about REST technologies. Section 3 presents the most relevant related work in this field. Section 4 provides an overview of the proposed UDP4US and its components. Section 5 outlines the implementation of UDP4US REST module and the experimental testbed employed. Finally, Section 6 presents the conclusions and the future work ideas.

[1]http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01
[2]http://www.dlna.org/

2. REST Technology

The REST was proposed by [Fielding 2000] aiming to simplify the Hypermedia Distributed Architecture. His work introduces the term REST, which stands for Representational State Transfer. REST is an architectural style that became an alternative to the standard Web Service technology. The main difference between traditional Web Services and REST is that the former is based on the SOAP protocol, which demands a more complex protocol stack than REST. REST is based just on HTTP operations, which scale better because of the independence from the SOAP protocol. The simplicity of REST makes it an ideal candidate to build an "universal" API for embedded devices. This concept is often referred to as "Web of things" [Guinard et al. 2010]. The main features of REST are the use of URIs and of resource entities. Four HTTP methods can be employed to provide services, as shown in figure 1. These methods are described as follows:

- *GET*: Requests data from a specific resource. It is defined as a safe method and should not be used to trigger an action.
- *POST*: The information sent in the request body is used to create a new resource instance.
- *DELETE*: Remove a resource instance. Should return the 204 status if there is no resource associated with the specified URI.
- *PUT*: Updates a resource at the specified URI. If the resource does not exist, it does create one. The main difference between POST and PUT is that the former can deal not only with resources, but can also process information.

RESTful services can exchange messages using several data formats, such as XML, JSON and HTML. The use of the HTTP protocol allows it to easily cross network boundaries. Since RESTful services have a stateless behavior, the overall system scalability is highly improved.

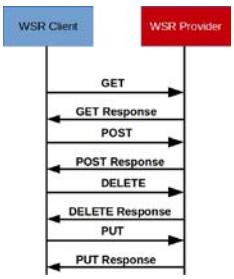

Figure 1. Message Exchanges in REST

3. Related Works

Some related works address devices integration at the protocol level, using DPWS or DLNA as a part of the proposed solution. For instance, in [Samaras et al. 2013] it is proposed a DPWS modified stack to be applied to 6LoWPAN-Based Wireless Sensor Networks (WSN). Therefore, the DPWS implementation was enhanced in order to consume less memory and decrease the processing overhead. The work shows the advantages to add SOA concepts into WSN by applying DPWS, in spite of increasing the message overhead.

A different integration approach was presented in [Dohndorf et al. 2010]. The proposal was to integrate DPWS with the OSGi platform. Therefore, all DPWS devices could be seen and controlled over the Internet, instead of being managed just in a local network. The authors intention was to place devices into the internet of things.

Some works have shown the role that REST plays on the integration of IoT systems that uses DPWS standard [Han et al. 2015, Guinard et al. 2010, Riedel et al. 2010, Moritz et al. 2010]. For instance, in [Han et al. 2015] it is presented an extension for DPWS using a REST proxy for IoT applications. Although REST or HTTP-based implementations cannot support event-driven models as DPWS, the work has shown improvements in latency and overhead, as well as simplifying the global topology.

In [Riedel et al. 2010] it is presented an approach for handling the task of integrating multiple concurrent IoT systems using automatic generation of DPWS gateways for sensor nodes. The study has shown that Web service gateways have great scalability in local area networks, besides being flexible enough to support Industrial applications.

Since SOA and REST are not conflicting, combining them, provides features to meet requirements of single applications (LAN) and IoT scenarios (WAN) for smart cooperating objects. Therefore, in [Moritz et al. 2010] it is discussed the drawbacks of using DPWS along with REST, which may require more resources and implementation efforts to carry out eventing and discovery features.

4. Universal Device Pipe for Providing Ubiquitous Services

Considering an ubiquitous environment, one can imagine several services being provided. However, making service composition using current SOA facilities is quite complicated for several reasons. The most difficult is the wide variety of technologies available to describe, publish, and compose services. Depending on the target application, different protocols and mechanisms should be used to provide the required services.

In order to illustrate this problem, let us suppose an application scenario that adopts three different technologies, such as: DPWS, DLNA, and REST. Consider a person located in a public transportation vehicle (bus, train, etc). To make good use of its time, the person can watch a movie or listen to the radio using his/her tablet or smartphone along the journey. A given software running in the tablet or smartphone could keep updating the person location to the smart home system (e.g. by REST) so that it could forecast the person arrival time in order to properly maintain a comfortable temperature in the house. When the person arrives at home, he/she might like to continue watching the video on the TV (without having to make setups). At the moment that the smart home system realizes that the person is located in the living room it would transfer the video from the tablet to the TV in a synchronized manner (e.g. by DLNA). The identification of the person location inside the house could be done, for example, using a DPWS enabled RFID reader.

However, performing such operations using current technologies is a difficult task (not to say impossible). These protocols are incompatible with each other, making it hard to perform communication between heterogeneous devices. In order to provide a contribution for solving this problem, we have analyzed which would be the requirements necessary to allow the integration of heterogeneous devices, as follows:

- The first requirement is to expose the device and its services on the network. Each device has a specific metadata which describes its features, such as manufacturer, model, version, and details about its services. For instance, a sensor node that senses ambient temperature could provide the service *getCurrentTemperature*. This service should also be detailed, with all the necessary parameters, to allow the interaction with the client.
- The second requirement is to discover the device and its services. The clients should easily request any device on the network, regardless if the device is within a local or a remote network. Likewise, the devices should reply to discovery requests sent by clients.
- The third requirement is to perform service requests. With the metadata device, the clients must be able to run the service without the knowledge of the specific communication interface from the device.

In order to provide a solution for accomplishing the mentioned requirements, we propose UDP4US (Universal Device Pipe for Ubiquitous Services). The UDP4US was designed to be an integration channel at application level. The UDP4US works as a pipe that connects the different technological solutions, such as DPWS, DLNA, and REST. The details about UDP4US architecture and its operations are described next.

4.1. UDP4US Architecture

The UDP4US was designed to abstract services provided in different network protocols. It plays the role of a *middleware* on top of the supported protocols, allowing to provide abstract services and properly transporting messages. UDP4US provides a specific communication interface for each supported protocol, as illustrated in figure 2. This is provided by the *Service Broker* component, which in fact is a service with a published operation. The Service Broker hides the complexity to deal with incoming requests from different protocols. It listens the requests and translates them through the pipe. Currently, the Service Broker can handle requests from DPWS, DLNA, and RESTful clients.

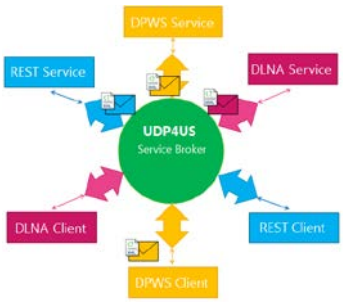

Figure 2. The Service Broker

The UDP4US stack protocol is presented in figure 3. From top-down, the topmost layer is the Service Broker Integrator (SBI). The SBI is responsible for implementing interfaces for each related protocol, in order to translate the requests. The SBI has a device for each supported protocol. For instance, the SBI has a device for DPWS that can be reached by any DPWS enable device, but cannot be directly reached by DLNA or REST enabled devices. Therefore, if a DLNA-enabled device needs to discover a

device or perform a service that is not DLNA-enabled, but that is a DPWS-enable device, the SBI has a DLNA-enabled device that listens to requests and translates to the DPWS protocol. The devices implemented in the SBI are in fact virtual devices that simply listen to incoming requests.

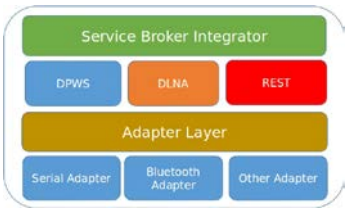

Figure 3. Proposed Protocol Stack

Below SBI there are the supported application protocols: DPWS, DLNA, and REST. The DPWS and DLNA engines are responsible for discovering and processing their requests independently. The REST interface plays the role of exposing DPWS and DLNA devices over the Internet.

The next layer is the Adapter Layer, which can be seen as a driver for the operating system. It acts like any off-the-shelf device that must implement a specific device-driver to support a given operating system. For devices that are neither DPWS nor DLNA enabled, different interfaces can be used. Currently it provides interfaces for serial communication and Bluetooth. Such requests are later translated for DPWS or DLNA. Our proposal was designed to be extended to others Adapter Layers, such as 802.15.4.

4.2. UDP4US Operation

The UDP4US can act as a Client Device and a Provider Device at the same time. For instance, if a Service Provider is not reachable using a given protocol, such request is translated to one of the other supported protocols. For example, if a DLNA client does not get an answer for a request, UDP4US translates the request to DPWS or REST in order to try to find a service. Furthermore, if the device provider is neither DLNA nor DPWS enabled, the Adapter Layer plays the role of a request translator. It is important to highlight that the requests could be a discovery or a service invocation. For example, if a Bluetooth device needs to interact with a serial device, i.e., both are not DPWS- nor DLNA-enabled devices, the requests must be translated to both technologies.

5. Experimental Evaluation

In order to validate the new component from UDP4US project, some experiments were performed to measure the bounds of the REST component. The experimental setup is illustrated in figure 4. From the right side, the DPWS-enabled client is requesting a service to its local network. However, the requested service was not presented in the same local network. Then the UDP4US listening the requests, translates to REST component and it dispatches over the internet. On the left side, a Service Provider DPWS enabled was listening incoming request from its local network. All incoming requests from internet were managed by UDP4US middleware. The incoming requests from the internet are listened by REST component, which gets the request and translate to UDP4US format.

The Service Broker component will translate the request to the specific format, which was in this case to DPWS metadata. Finally, the Service Provide DPWS enabled can process the request and reply to the DPWS-enabled client.

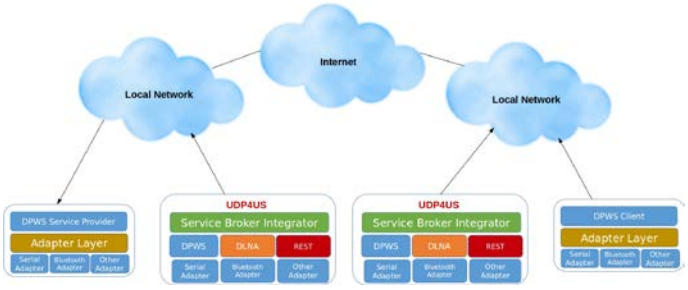

Figure 4. Experimental Setup Over the Internet

The main goal of the experiments were to find the boundaries of this new UDP4US component. Finding the limits of our approach might help to better guide the applications design, with a clear understanding of the viability the applying the concept of IoT. We performed the experiments with a high number of clients requesting one device's service. The number of clients ranged from 50 to 500, increasing with a step of 50, which means that UDP4US must handle with a high number of requests. The client and service provider were hosted in different networks. In order to reduce the statistical bias, each obtained result refers to the average of 10 executions. Furthermore, all experiments were carried out with clients performing discovery and execution of a service. The service execution performed by means of two-way requesting, such as a system requesting a value from a sensor. In the last experiment, the goal was to measure the cost of the solution over the internet, that way we have a sense of performance on IoT solutions. To oversee the performance analysis from all protocols, three metrics were chosen: i) number of successfully executed requests, ii) service discovery time, and iii) service execution time.

5.1. Implementation

The UDP4US prototype implementation was built on top of tree frameworks, the WS4D-JMEDS[3], Cling[4], and REST services, which were hosted by an Apache Server. All frameworks were developed using the Java SE platform. All source code generated in this project is publicly available in the github repository[5]. The equipment used in all experiments are described as follow:

- Service Provider Side:
  - *Service Provider* A notebook with Intel(R)Pentium(R) CPU P6100 @ 2.00 GHz, 2 GB of RAM and Ubuntu 13.10 Operating System hosting the service provider. An Arduino UNO R3 was attached to it by USB.
  - *UDP4US* A notebook with Intel(R) Core(TM) i3-M330 @ 2.13GHz, RAM 4 GB of RAM and Windows 10 hosting the REST Service Provider.
- Client Side:

[3]http://ws4d.e-technik.uni-rostock.de/, DPWS Implementation
[4]http://4thline.org/projects/cling/, DLNA Implementation
[5]https://github.com/UFSCAraSistemasUbiquos

– *UDP4US* A notebook with Intel(R) Core(TM) i5-2410M CPU @ 2.30 GHz, 6 GB of RAM and MS Windows 7 OS hosting the REST Client.
– *Client* A notebook Intel(R)Core(R) CPU i3-370M @ 2.53 GHz, 4 GB of RAM and Arch Linux Operating System hosting the client DPWS enabled.

5.2. Obtained Results

This section presents the results obtained in the performance experiments, where a set of REST Clients requested a virtual device hosted on DPWS through UDP4US. Figure 5 illustrates the Number of Requests successfully issued over. We can observe that the amount of successful requests during 500 concurrent requests was nearly 84%. Although the tests have been executed in extreme conditions, one can say that UDP4US can deal with 500 concurrent requests over the Internet.

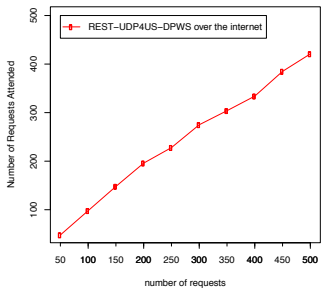

Figure 5. Number of Requests Attended Over the Internet

In figure 6 it is shown the Service Execution Time over the Internet. As expected, the response time was higher compared with requests on the local network [Felisberto et al. 2015]. This due to the fact that the Internet has a higher latency than the local network. The Service Discovery Time is also shown in figure 6. The Service Discovery Time follows a similar behavior of the Service Execution Time, also due to the high traffic over the Internet.

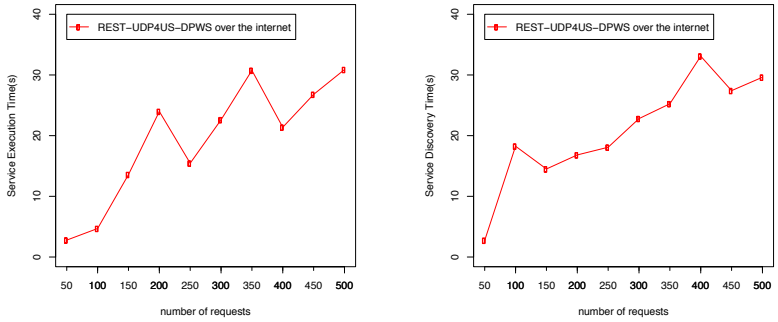

Figure 6. Service Discovery and Execution Times Over Internet

The first observation we pointed out is that, for all experiments, as the number of clients increases, the execution and discovery times also increases. In addition, the number of successfully issued requests decreases. The behavior was a bit irregular, because

the requests do not have any QoS schema in order to prioritize network traffic. Then, all requests compete for the access to the network. In spite of the high response time of the requests, the experiments have clarified the boundaries of response time and the number of possible concurrent requests for IoT applications using REST and DPWS technologies.

6. Conclusions and Future Works

A lot of research has been made to provide a baseline protocol to connect heterogeneous devices. Such protocols can be applied in different market segments such as home automation, industries, health care and many others. Most of automation segment aims the use of a local network to expose device's services. Our previous work complies with the requirements to integrate multiple protocols such as DPWS, DLNA, and Bluetooth. In this paper, we presented the REST module for the UDP4US architecture. The REST module is responsible to expose the local network services to the internet. Now the UDP4US architecture brought new possibilities to distributed applications for smart spaces. A smart space such as a house can be connect to other internet services, such as public transport services. The new opportunities for IoT applications can extend from a smart house to a connected city.

To make this new trend possible, it is important to establish the limits of the protocols in order to better guide the implementation of each application scenario. The presented work shows that the performance of the proposed solution has been satisfactory, since the tests were performed with the intent of applying the maximum stress. Besides, the integration was successful and allowed the different devices attached to incompatibles protocols to reach each other and exchanged messages over the internet.

In a near future, we intend to extend the UDP4US architecture to provide a QoS schema for client requests. The QoS will be provided by a M,K-Firm scheduler, that will prioritize the requests that will be near to dynamic failure [Hamdaoui and Ramanathan 1995]. Furthermore, others adapter layers can also be extended, such as an adapter for IEEE 802.15.4 technology.

References

Atzori, L., Iera, A., and Morabito, G. (2010). The internet of things: A survey. *Computer networks*, 54(15):2787–2805.

Cândido, G., Jammes, F., de Oliveira, J. B., and Colombo, A. W. (2010). Soa at device level in the industrial domain: Assessment of opc ua and dpws specifications. In *Industrial Informatics (INDIN), 2010 8th IEEE International Conference on*, pages 598–603. IEEE.

Dohndorf, O., Kruger, J., Krumm, H., Fiehe, C., Litvina, A., Luck, I., and Stewing, F.-J. (2010). Towards the web of things: Using dpws to bridge isolated osgi platforms. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, pages 720–725.

Felisberto, T. Z., Tramontin, E. D., da Cunha dos Santos, F., Morales, A. S., Siqueira, F., and de Araújo, G. M. (2015). Universal device pipe for ubiquitous services. In *SBESC Brazilian Symposium on Computing Systems Engineering*. IEEE.

Fielding, R. (2000). Representational state transfer. *Architectural Styles and the Design of Netowork-based Software Architecture*, pages 76–85.

Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660.

Guinard, D., Trifa, V., Karnouskos, S., Spiess, P., and Savio, D. (2010). Interacting with the SOA-based internet of things: Discovery, query, selection, and on-demand provisioning of web services. *IEEE Transactions on Services Computing*, 3(3):223–235.

Hamdaoui, M. and Ramanathan, P. (1995). A dynamic priority assignment technique for streams with (m, k)-firm deadlines. *Computers, IEEE Transactions on*, 44(12):1443–1451.

Han, S. N., Park, S., Lee, G. M., and Crespi, N. (2015). Extending the Devices Profile for Web Services Standard Using a REST Proxy. *Internet Computing, IEEE*, 19(1):10–17.

Kumar, S. (2009). Challenges for ubiquitous computing. In *Networking and Services, 2009. ICNS '09. Fifth International Conference on*, pages 526–535.

Moritz, G., Zeeb, E., Prüter, S., Golatowski, F., Timmermann, D., and Stoll, R. (2010). Devices profile for web services and the REST. *IEEE International Conference on Industrial Informatics (INDIN)*, pages 584–591.

Riedel, T., Fantana, N., Genaid, A., Yordanov, D., Schmidtke, H. V., and Beigl, M. (2010). Using web service gateways and code generation for sustainable iot system development. In *Internet of Things (IOT), 2010*, pages 1–8. IEEE.

Samaras, I., Hassapis, G., and Gialelis, J. (2013). A modified dpws protocol stack for 6lowpan-based wireless sensor networks. *Industrial Informatics, IEEE Transactions on*, 9(1):209–217.

Sleman, A. and Moeller, R. (2008). Integration of wireless sensor network services into other home and industrial networks; using device profile for web services (dpws). In *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on*, pages 1–5. IEEE.

Weiser, M. (1991). The computer for the 21st century. *Scientific american*, 265(3):94–104.