GoThings
An Application-layer Gateway Architecture for the Internet of Things

Wagner Luís de A. M. Macêdo, Tarcísio da Rocha and Edward David Moreno
Federal University of Sergipe, São Cristóvão, Sergipe, Brazil

Keywords: Constrained Devices, Internet of Things, Messaging Protocols.

Abstract: With the Internet of Things (IoT), it is predicted that the number of connected devices will reach 50 billion by 2020. Many of these devices often adopt, at application layer, mutually incompatible messaging protocols. A possible solution to this problem is to use a same messaging protocol among all devices. However, a single protocol is not always suitable for both constrained and unconstrained devices. Several solutions to the interoperability issue in the IoT have been proposed, but they neither provide transparent interoperation nor are extensible and configurable enough. Meanwhile, this paper proposes GoThings, a preliminary gateway architecture which can enable interconnectivity between different messaging protocols. GoThings is focused on extensibility, configurability and generality, in the context of IoT problems.

1 INTRODUCTION

With the Internet of Things (IoT), it is predicted that the number of connected devices will reach 50 billion by 2020[1]. IoT is a vision of a world where all physical objects (things) have some form of Internet connectivity. Another explanation is that IoT enables Internet to be used to connect people to things and things to things.

Due to this expected number of devices the envisioned environment for the IoT is extremely heterogeneous, so that different devices may be unable to communicate to each other. Heterogeneity happens in all network layers, specifically in the application layer is emerging a number of mutually incompatible messaging protocols[2]. However, the interoperability issue is even higher because messaging protocols may adopt different message exchange patterns, such as *request-response* and *publish-subscribe*.

A possible solution to this problem is to use a same messaging protocol among all devices. In the Web of Things (WoT) approach (Guinard et al., 2010), for instance, HTTP was chosen to integrate all devices. However, HTTP is not always suitable for many participants of IoT, such as sensor devices, since they are not able to work with any protocol due to strict constraints in terms of memory, processing and energy (Dargie and Poellabauer, 2010). The reverse is also true, i.e. a protocol for constrained devices is usually not adequate to unconstrained ones.

Some constrained devices are battery-limited. In order to to save power, they are commonly configured to frequently sleep and being active when strictly necessary. This put some barriers, since devices are not always active to answer.

Proposals have already been presented to solve the interoperability issue in the IoT but we could not find one solution allowing a bidirectional transparent interoperation between constrained devices with different messaging protocols that is extensible for any kind of protocol regardless of message exchange pattern. When we say transparent, we mean a solution that provides a communication which is virtually direct, i.e. sender is not aware of an intermediary.

The main goal of this paper is to propose an architecture for gateways that are able to interconnect different messaging protocols for the Internet of Things that meets the following requirements:

- Multi protocol support by an extensible framework via plugins which reduces complexity of adding new protocols.
- Support to the message exchange patterns request-response and publish-subscribe.
- Configurability, through a structure that lets you easily add or change behaviors of the gateway depending on the context.

[1]https://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf. Access in November 2, 2014.

[2]http://www.prismtech.com/download-documents/1561. Access in November 2, 2014.

Luís de A. M. Macêdo W., da Rocha T. and David Moreno E..
GoThings - An Application-layer Gateway Architecture for the Internet of Things.
DOI: 10.5220/0005493701350140
In *Proceedings of the 11th International Conference on Web Information Systems and Technologies* (WEBIST-2015), pages 135-140
ISBN: 978-989-758-106-9

- Caching support to avoid unnecessary accesses to constrained networks and to enable requests to suspended devices.

The rest of the paper is organized as following. Section 2 discusses about common approaches to interoperability. Section 3 presents a brief review of related works. In Section 4 we show our proposed solution. Section 5 concludes the paper.

2 INTEROPERABILITY APPROACHES

Interoperability may refer to the ability for one or more systems or domains to connect, understand and exchange data with one another for a given purpose (Blair et al., 2011). The goal in providing interoperability between different protocols is to allow communication between different domains keeping the transferred original message unchanged or without losing original semantics.

To have interoperability, it is required to map the different behaviors of the domains. This mapping is usually called a bridge. In the literature, there are two main approaches to interoperability between different domains: direct bridges and indirect bridges (Issarny et al., 2011).

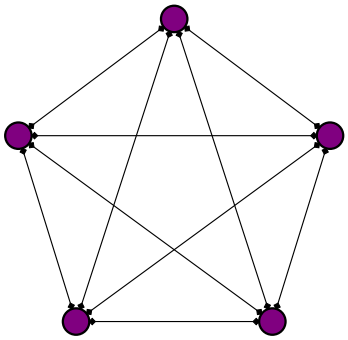

Figure 1: Direct bridges approach.

Direct bridges (see Figure 1) consist of a direct translation of behaviors between domains, so a translator is required for each pair of involved domains. The advantage of a direct bridge is to require only one translation operation, but the needed number of translators grows quadratically to the number n of domains, as stated in equation 1.

$$t(n) = \frac{n(n-1)}{2} \quad (1)$$

Contrasting, in indirect bridges an intermediate format is used, so the behaviors of each domain are mapped by only one translator to an intermediate domain (see Figure 2). Although this approach has the disadvantage of requiring two translation operations, since every message is first translated into intermediate format before being translated to target domain format, the needed number of translators is linear to the number of domains, what indeed is a major advantage of using indirect bridges.

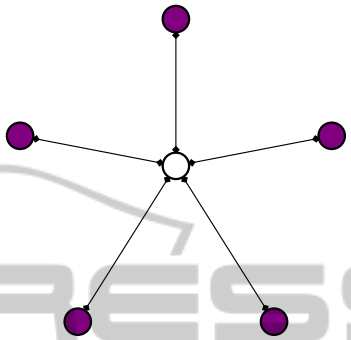

Figure 2: Indirect bridges approach.

We adopt in this paper the indirect bridge approach. The reasoning related with this decision is explained in Section 4.

3 RELATED WORK

The great number of different messaging protocols follows the wide range of heterogeneous devices ready to the IoT. So, in this section, we will first review some of the most relevant protocols and then we will discuss some related proposals to this paper.

In the context of IoT protocols, some more robust devices, like IP cameras, are often equipped with more than one protocol to access the images (HTTP and RTSP are the most common). However, most IoT devices are so restricted that cannot operate with heavyweight protocols such as HTTP.

In order to meet demands for a lightweight protocol compatible with the popular HTTP, the Internet Engineering Task Force (IETF) designed the CoAP protocol (RFC 7252), that keeps the same semantics of HTTP, but, instead of TCP, it is implemented over UDP. CoAP has some important features to constrained devices like multicast communication and the *observe* request, a kind of request where the server responds whenever the resource is changed. Contiki[3], one of the major operating systems for the IoT, adopts CoAP as its primary application protocol.

[3]http://contiki-os.org/

Also important to the IoT is MQTT[4], a publish-subscribe protocol created by IBM and designed to be simple, lightweight and suitable for constrained environments. The MQTT implementation is over TCP, inhibiting adoption in extremely constrained devices such as some wireless sensors, which led to the development of MQTT for Wireless Sensor Networks (Hunkeler et al., 2008).

Coming from chats, i.e. human-to-human communication (H2H), it is remarkable the presence of XMPP (RFC 6120) in the list of protocols for IoT. But in the interesting work of (Hornsby and Bail, 2009) and (Klauck and Kirsche, 2012), XMPP is proposed for human-to-machine communication (H2M) in a scheme that allows users to add sensors as friends for receiving sensor updates via instant messaging. A negative factor of XMPP is that it relies in XML data format, considered heavy for constrained devices, so its use in the IoT is still not wide.

Several approaches have been proposed to solve the interoperability problem of messaging protocols in the IoT. The Web of Things (Guinard et al., 2010) is an implementation of IoT that makes use of web technologies, such as REST (Fielding and Taylor, 2002), as a common paradigm among people and devices. But when devices cannot run a HTTP server, integration takes place using intermediaries called Smart Gateways which is a kind of web proxy that abstracts heterogeneous communication through a RESTful API. In addition, the WoT approach supports publish-subscribe interactions via web feeds. Our proposal is directly related with the WoT Smart Gateway, but it is not limited to a REST-like interface.

(Castellani et al., 2012) discuss about problems and solutions in mapping between HTTP and CoAP protocols regarding the implementation of a *cross proxy* between these protocols. This kind of proxy can transparently interconnect protocols maintaining most of semantics, but it has the drawbacks of being locked to only two and limited to very closer protocols.

(Collina et al., 2012) proposed a publish-subscribe topic-based broker where access to topics is done via multiple protocols, regardless of the communication pattern. The authors validated the proposal by implementing a broker reached via HTTP (request-response) and MQTT (publish-subscribe). Despite this proposal provides interoperability, the broker only allows communication between device and broker, not a direct communication between devices. In other words, transparent interoperability is not supported.

(Bromberg et al., 2009) proposed z2z, a sophisticated generative approach to build generic gateways. It is composed by a domain-specific language (DSL) also called z2z, a runtime system and a compiler, and it is focused on easier development and efficiency. The gateways are statically generated to C code from specifications written in z2z. Although z2z provides a very customized approach to gateway development, it generates gateways for only pairs of protocols (by direct translation) and request-response is the only supported message exchange pattern.

Facing problems of existing solutions in the context of IoT-driven gateways, next section presents our proposal which is expected to solve their limitations.

[4]http://mqtt.org/

4 GATEWAY ARCHITECTURE

This paper proposes GoThings, an architecture for building application-layer gateways for the IoT. The proposal aims at the following:

Interoperability. The architecture should make possible to develop a gateway which intermediates devices with different application-layer protocols.

Extensibility. The architecture should allow addition of new protocols to the gateway without modification of its internal structures.

Generality. The architecture should be generic enough to permit its use in various IoT scenarios.

Configurability. The architecture should provide mechanisms to modify the system behavior at a granular level.

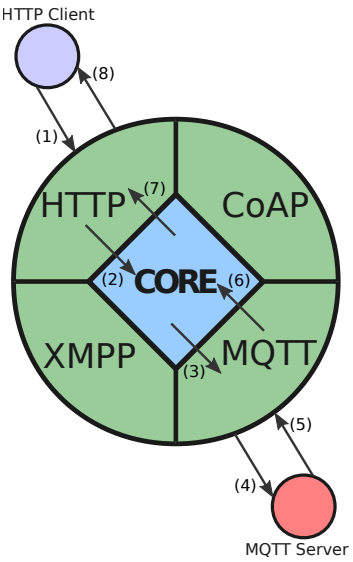

Figure 3: The flow throughout the gateway of a HTTP-MQTT request.

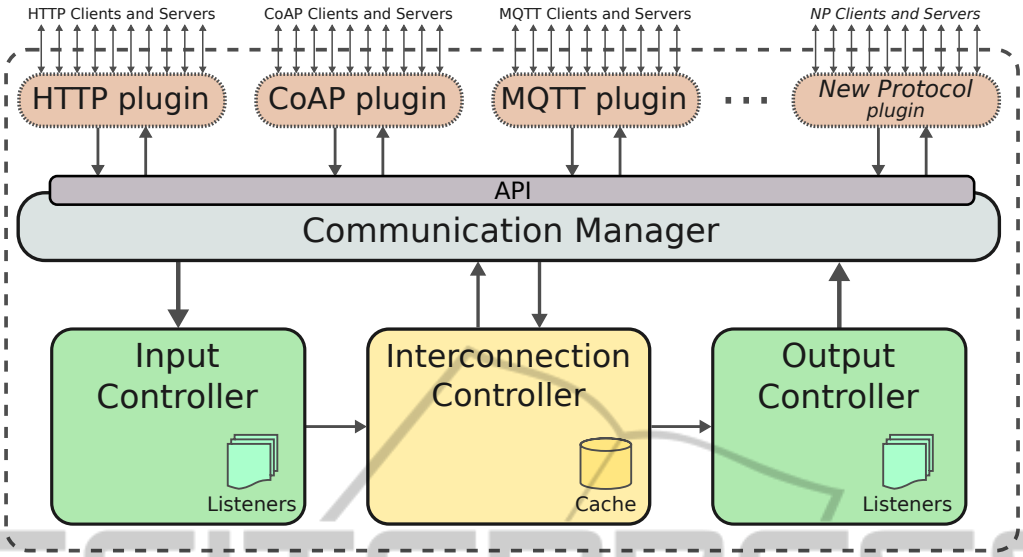

Figure 4: The GoThings architecture overview.

Justified by the need of a gateway easily extensible to other protocols we followed in GoThings architecture the indirect bridge approach. Indirect bridges make necessary an intermediate format as stated in Section 2. For that we set as intermediate format a message model carrying common fields and options from an intersection of IoT protocols and message exchange patterns.

A known problem in indirect translations is about missing protocol semantics of one or both sides. We do not think GoThings architecture will suffer of this because we focus our attention on IoT interaction patterns that are usually very simple and specific.

Figure 3 shows at high level the flow passing through the gateway of a successful request from a HTTP client to a MQTT server. The flow begins when a HTTP client requests to gateway through the HTTP plugin (step 1), which forwards to the core (step 2). After request is parsed, the gateway knows to what plugin the message will be forwarded, which in this case is the MQTT one (step 3). Accordingly, MQTT plugin makes a request to MQTT server and receives a response (steps 4 and 5), that is returned to the gateway core (step 6), which returns to HTTP client (steps 7 and 8).

It is important to see how a HTTP request is recognized by the gateway as a request to a MQTT server. For that we used URI as the element that carries routing information between protocols. Thus, assuming the gateway is at `gw.me.com` and the server at `bk.you.org`, the following URI

```
http://gw.me.com/mqtt/bk.you.org/moisture
```

indicates the request must be forwarded to the MQTT plugin (note `mqtt` in the address) which is able to request `/moisture` from `bk.you.org`.

The GoThings architecture overview can be seen in Figure 4. The visible components are briefly explained below:

- **Communication manager** intermediates the communication between plugins and controllers. It enables the gateway extensibility and abstracts each protocol ways through the message model. The plugins talk to communication manager using an API to facilitate plugin development.
- **Interconnection controller** is the component that forwards requests to the target plugin and maintains request states until receives an answer. The cache is an important element of this component.
- **Input and output controllers** are components that handle, respectively, requests and responses. The listeners are actions triggered by events from both input and output controllers. A listener could be either a simple logging action or a complex action that may change the message.
- **Plugin** is both a client and a server implementation of a specific protocol. The plugins cannot communicate directly with each other, but only via communication manager.

4.1 Gateway Operation

When a client makes a request to the gateway, it makes through a plugin. But until the client receives a response, some effort is required. An outline of these effort can be seen in Figure 5 that shows a collab-

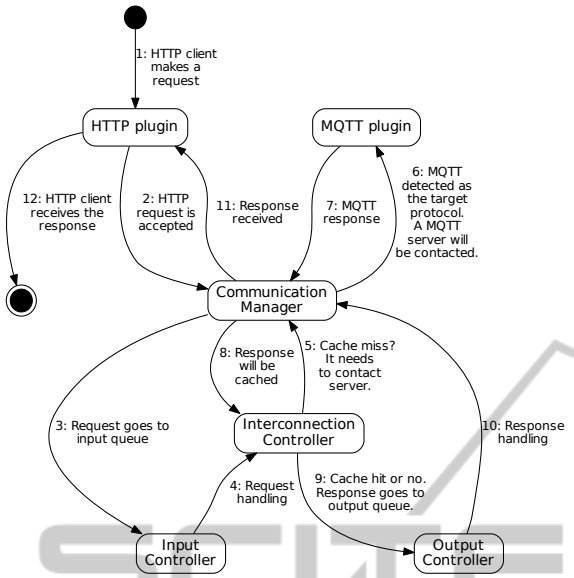

Figure 5: Collaboration diagram for a HTTP-MQTT request.

oration diagram for the same HTTP-MQTT request touched upon before. We explain the sequence below:

1: The client makes a HTTP request to the gateway exactly like he would request to a normal HTTP server.

2–3: HTTP plugin receives the request and sends it to Communication Manager (CM), which in turn forwards to Input Controller (IC).

4: IC sends the request message to every registered input listeners and then forwards to Interconnection Controller (ICC).

5–8: This sequence is performed only if ICC finds the asked resource in cache (*cache hit*). On contrary, if nothing found (*cache miss*), an actual interaction with a MQTT server is required. This is done after ICC passes the request to CM which forwards it to target plugin (MQTT in this case). The plugin makes a request to server, and when response arrives the plugin passes it to CM that sends it to ICC which, finally, stores it in cache.

9: Whether the response was retrieved from cache or not, it is passed to Output Controller (OC).

10: Similar to IC, the OC sends the response message to every registered output listeners and then forwards to CM.

11–12: CM sends the response to the source protocol plugin (HTTP in this case) which finalizes the request by sending the response to client.

4.2 Interoperability

Each protocol talks a particular language in a particular way. This makes heterogeneous devices unable to interoperate unless every protocol needed for the communication is installed.

Constrained devices, however, are not able to have many protocols available at the same time due to memory limitations. Furthermore, the complexity of maintaining multi protocol applications is higher than single protocol ones. Our gateway architecture was designed to be able to deal with these issues.

4.3 Extensibility

As stated before, GoThings makes use of the indirect bridge approach because we focus on easiness of plugging in any number of previously unknown protocols. That said, our proposed gateway architecture was designed to be easy for interconnecting protocols not thought before, relying on plugin subsystem. In fact, enabling a new protocol into the gateway is only a matter of develop and install a new plugin.

A plugin is a software component that has to implement some interfaces to send and receive messages, and interact with communication manager. The plugin should have both a client and a server of a given protocol to a complete functionality, but a partial functionality is possible. One point here is that many protocols have ready-to-use libraries that can be used to accelerate development of plugins.

4.4 Configurability and Generality

The listeners inside input and output controllers are the key to the configurability. They allow a more granular gateway configuration. With them, as an example, we could instruct gateway to assure payload of response is in JSON format if request comes from a constrained device. Or we could program one or more listeners to add semantic interoperability features to the gateway.

By providing such mechanisms that can alter the gateway behavior, the configurability supports the generality. Additionally, the message model is a vital element to make the architecture be generic enough. In this sense, it is important to remember that the generality concept of this paper is in the context of IoT problems.

4.5 Suitability for Constrained Devices

The cache in the interconnection controller is the key element that makes GoThings apt to constrained de-

vices. A cache allows the gateway to reduce forwarding of requests and to ensure that a value is always returned, even when requested devices are sleeping.

In Section 4.1 we could see how cache may reduce the communication overhead. In general, by avoiding requests made to constrained devices, the gateway becomes suitable for them.

5 CONCLUSIONS

This paper proposed GoThings, a preliminary architecture for creating gateways to interconnect application-layer messaging protocols focused on Internet of Things, and suitable for constrained devices. In regard to message exchange pattern, the architecture support both request-response and publish-subscribe.

The main aspects of the architecture are interoperability, extensibility, generality and configurability. GoThings allows transparent interoperability in a way that devices can use its own protocol to have a heterogeneous communication. GoThings was designed to be extensible: enabling a new protocol is only a matter to install a new plugin. GoThings is generic enough in the context of IoT problems, and the users are able to change gateway behaviors through the use of listeners.

Finally, there are some ongoing work. We are looking for the best ways to specify the message model in a manner that existing and possibly future IoT protocols can be interconnected without losing their semantics. When the message model is properly specified we can work on plugin subsystem which includes the specification of communication manager API. We plan to deploy our proof-of-concept gateway implementation in a Raspberry Pi[5]. We will compare performance with the achieved of native protocol implementations. We expect to finish these work until January 2016.

REFERENCES

Blair, G., Paolucci, M., Grace, P., and Georgantas, N. (2011). Interoperability in Complex Distributed Systems. In *Formal Methods for Eternal Networked Software Systems*. Springer Berlin Heidelberg.

Bromberg, Y.-D., Réveillère, L., Lawall, J., and Muller, G. (2009). Automatic Generation of Network Protocol Gateways. In *ACM/IFIP/USENIX, 10th International Middleware Conference*.

[5]http://www.raspberrypi.org/

Castellani, A., Fossati, T., and Loreto, S. (2012). HTTP-CoAP cross protocol proxy: an implementation viewpoint. In *IEEE 9th International Conference on Mobile Ad-Hoc and Sensor Systems*.

Collina, M., Corazza, G. E., and Vanelli-Coralli, A. (2012). Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST. In *IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications*.

Dargie, W. and Poellabauer, C. (2010). *Fundamentals of wireless sensor networks: theory and practice*. John Wiley & Sons Ltd., United Kingdom.

Fielding, R. T. and Taylor, R. N. (2002). Principled design of the modern Web architecture. *ACM Transactions on Internet Technology*, 2:115–150.

Guinard, D., Trifa, V., and Wilde, E. (2010). A resource oriented architecture for the Web of Things. In *Internet of Things (IOT)*, pages 1–8, Tokyo. IEEE.

Hornsby, A. and Bail, E. (2009). μXMPP: Lightweight implementation for low power operating system Contiki. In *IEEE International Conference on Ultra Modern Telecommunications & Workshops*.

Hunkeler, U., Truong, H. L., and Stanford-Clark, A. (2008). MQTT-S – A publish/subscribe protocol for Wireless Sensor Networks. In *3rd IEEE International Conference on Communication Systems Software and Middleware and Workshops*.

Issarny, V., Bennaceur, A., and Bromberg, Y.-D. (2011). Middleware-Layer Connector Synthesis: Beyond State of the Art in Middleware Interoperability. In *Formal Methods for Eternal Networked Software Systems*. Springer Berlin Heidelberg.

Klauck, R. and Kirsche, M. (2012). Chatty things – Making the Internet of Things readily usable for the masses with XMPP. In *8th International Conference on Collaborative Computing*, Pittsburgh, PA.