Implementation Experiences of Semantic Interoperability for RESTful Gateway Management

IoT Semantic Interoperability Workshop 2016, Position Paper[1]

Bill Silverajan
Tampere University of Technology
bilhanan.silverajan@tut.fi

Mert Ocak
Ericsson
mert.ocak@ericsson.com

Jaime Jiménez
Ericsson
jaime.jimenez@ericsson.com

I. INTRODUCTION

Proper management of gateways and access points is one of the most important aspects for ensuring the success of the Internet of Things (IoT). Traditionally, network management of gateways and routers has focused on fault detection, improving network efficiency and application performance in controlled, trusted environments such as an enterprise. However, undertaking gateway management in the IoT presents a vastly different challenge, owing partly to the heterogeneity, number and diversity of the gateways and the kinds of networks they are connected to.

The IoT itself can be perceived as networks of networks. Significant numbers of IoT gateways and routers would be used as relays or proxies with many kinds of non-IP networks as well, as the vast majority of today's IoT constrained devices are not natively IP-enabled. Many such gateways can themselves be resource constrained or be battery powered, and can reside in environments that operate for great lengths of time without human intervention.

An IoT gateway can be characterised according to its functionality:

- In its most basic configuration, the gateway simply supplies Internet connectivity to IP-enabled devices and sensors by bridging two kinds of radio networks. The most common example is a smartphone or a cellular WAN gateway acting as a WiFi access point or a low power PAN relay.
- It proxies network communication between non-IP LAN nodes and the Internet, using the device-to-gateway communication pattern described in RFC 7452 [1]. Typical examples include a Bluetooth to IP gateway, or a vehicular gateway connected to the controller area network (CAN bus)-based vehicular network.
- It facilitates endpoint reachability for management operations in the presence of firewalls and NATs using packet encapsulation or tunnelling techniques.
- It provides architectural interoperability among existing management tools and systems. Examples are intermediary gateways between an operator's SNMP-based management framework and its own management system optimised for lightweight communication using protocols such as CoAP or MQTT.

Other ways to categorise IoT gateways are possible too, in terms of how the gateway is powered, whether it intermittently powers down its radio interfaces upon a lack of activity to save energy, whether it is mobile, and so on. These categories and properties inevitably aid in the design of data models and the semantics used for managing these gateways.

In network management, the concepts of an Information Model (IM) and a Data Model (DM) are explained in RFC 3444 [2]: An IM models managed objects at a conceptual level independent of underlying protocols or implementation details. It can be flexibly defined in many ways, ranging from formal methods to informal descriptions using a natural language. On the other hand, a DM is defined at a lower level and it includes all the protocol and implementation specific details and how to map managed objects onto protocol constructs. To date, most models standardised are DMs. These definitions render it possible to define multiple different DMs using an IM, with the IM maintaining the coherence and preserving semantics between the derived DMs.

In this paper, we address some of the challenges of achieving better semantic interoperability with existing IMs and DMs when undertaking REST-based management of IoT gateways. This is based on our own initial experiences in implementing and managing various gateways using the Lightweight M2M (LWM2M) protocol [3] from the Open Mobile Alliance (OMA) atop CoAP, and additionally employing the IP Smart Object (IPSO) Application Objects as a Data Model. Object models defined by OMA and IPSO directly map and expose the capabilities of CoAP endpoints as URI-based resource representations, as using CoAP for application as well as management data is a significant advantage for constrained nodes.

Gateways implementations were based on open linux platforms such as OpenWRT and Raspbian, as well as Android. Nevertheless, the hurdles we encountered as well as our findings, are generic enough to be applicable for many kinds of open as well as closed or proprietary IoT gateway platforms, and indeed endpoints themselves.

We identify and describe three main challenges in the following sections, as well as solutions and possible workarounds. These consider how the gateway plays an important part as an integrator between multiple platform specific data models, how hypermedia driven interaction with a gateway can reduce the complexity of developing new data models for gateways as well as how gateways can aid with end point proxying and caching.

[1] *Collaboration for this position paper was made possible by the EIT Digital ACTIVE project.*

II. INTEGRATION ROLE OF GATEWAY BETWEEN DIFFERENT DATA MODELS

As discussed in section 2.3 of RFC 7452 , a gateway needs to be introduced into the communication architecture that bridges between especially IP network with semantic data models and non-IP short range radio technologies with proprietary data models. Integrating such proprietary data models to the network requires the gateway to translate between the data models. This translation is done using proprietary methods in most of the current gateway implementations and hence, creates silos between different gateway manufacturers. As an example, Bluetooth SIG publishes GAP and GATT REST API white papers [4,5] to standardize the APIs defined on Bluetooth gateways but common data models to seamlessly integrate the Web and e.g. BLE data models are needed to provide interoperability between these two technologies.

Defining semantic object models for proprietary data models exposes those resources to the RESTful devices without the need of a separate API definition on the gateway. This approach would unambiguously provide semantic interoperability between different communication architectures. Such object models can be defined by mapping the proprietary data model to e.g. IPSO object which would enable the integration of BLE, ZigBee devices to CoAP and LWM2M networks.

A separate example here is the gateway semantic object models defined by mapping an OpenWRT gateway's native UCI configuration system [6] to IPSO objects and resources. UCI configuration files are defined in a structured way for each gateway configuration and each section in the config file can be mapped to an IPSO object. Options under each config file can also be mapped to resources in the IPSO object. Some of the basic set of UCI configuration files are the following:

Basic	
/etc/config/firewall	NAT, packet filter, port forwarding, etc.
/etc/config/network	Switch, interface and route configuration
/etc/config/system	Misc. system settings
/etc/config/wireless	Wireless settings and wifi network definition

Table 1. Some basic UCI configuration files

These configuration files can be translated to separate IPSO objects e.g. /etc/config/wireless file can be mapped to "Gateway Wireless Interface" IPSO object. Wireless interface configuration file have default options and these options can be represented as resources under "Gateway Wireless Interface" object, which results in the following IPSO Object and some of the possible resources under the object are also given:

Object	Object ID	Object URN	Multiple Instances?	Description
IPSO Gateway Wireless Interface	1025	urn:oma:lwm2m:ext:1025	Yes	Basic settings for the wireless interface.

Table 2. IPSO Gateway Wireless Interface object

Name	Type	Description
ifname	string	Specifies a custom name for the wifi interface, which is otherwise automatically named.
mode	string	Selects the operation mode of the wireless network interface controller.
disabled	boolean	Disables the radio adapter if set to 1.
ssid	string	The broadcasted SSID of the wireless network.
macaddr	MAC address	Identifies the underlying radio adapter associated to this section.
txpower	integer	Specifies the *transmission power in dBm*
network	string	Specifies the network interface to attach the wireless to.

Table 3. Excerpts from UCI wireless config file options

Resource Name	Resource ID	Access Type
Ifname	0	R,W
Mode	1	R,W
Disabled	2	R,W
SSID	3	R,W
MAC Address	4	R,W
Transmission Power	6	R,W
Network	7	R,W

Table 4. IPSO Gateway Wireless Interface resources

The case for IoT Gateways as integrators between platforms and ecosystems is built on the premises that gateway often have higher computing power and are permanently connected to the Internet. It is often the case that APIs are defined as static definitions that represent the functionality of a device. When the definition is expanded or changes, the API is broken. Providing that there is no alternative mechanism to add metadata and other information that could help prevent APIs from breaking, gateways can mitigate the effect on interoperability between devices.

Surprisingly, many of the organizations are creating similar application semantics than, in practice, only differ on the vocabulary used. Just like a gateway can translate REST-based commands into UCI OpenWRT ones, similar translations could be enabled for other schemas from different organizations (e.g. LWM2M, OCF, IPSO, W3C…).

III. ENABLING HYPERMEDIA-DRIVEN GATEWAY INTERACTION

The approach of object models directly mapping to endpoint capabilities works well for constrained endpoints such as sensors. For more sophisticated devices, such as a gateway, exposing properties and data as managed objects and resources can easily result in large and unsustainable object models and resource lists. IPSO has addressed this to a large extent, by deliberately defining objects as simple building blocks. These can be then aggregated by developers to construct composite objects using a Web Linking Framework, providing a practical approach for resource interactions with composite objects.

While the LWM2M data and operation models support read, write and execute operations on a resource (effectively a CoAP GET, PUT or POST operation), a gateway's proprietary or native configuration API often needs to be invoked upon a change in a resource's representation. This is effectively an implicit process not exposed to the management server as the object model does not attach any semantics to the kinds of native operations that could or should be invoked upon actions on LWM2M resources. When applied to gateway management this can become an issue, as the object model does not accurately capture the kinds of operations and configuration changes a gateway may need to perform in the background, to accurately reflect the resource state in the object model.

As an example, consider a packet filtering mechanism on an access point, for which an example LWM2M or IPSO object model comprises of firewall objects. Each firewall object represents a single rule, while each resource represents an option, such as a name, source or destination address, incoming interface, connection state. Each action on the object to be taken needs to be modelled as a resource, in addition to an additional final "commit" resource, in order to first properly construct and then trigger the firewall rule on the gateway by the server, as well as to subsequently monitor its status. Without any additional semantics, similar kinds of object and resource models need to be developed for other policy objects, system configuration and the gateway's network interfaces. While this is a workable solution, it is far from elegant. A different gateway implementation may also choose to perform its runtime firewall management in a different manner, and incorporate additional resources to reflect these operations, leading to fragmented semantic models that may not interoperate well.

One possible way to reduce this complexity is to adopt a hypermedia driven approach by embedding a REST constraint called Hypermedia As The Engine of Application State (HATEOAS). Using HATEOAS, an object only exposes resources directly necessary for the current context, while returning possible operations on the object or resource as hyperlinks and relation types implementing standard REST verbs.

In other words, a firewall object can then be simplified so that the gateway's native API invocations can be exposed to a management server over a REST interface when necessary. A "commit" hyperlink could be exposed under a "name" resource if and only if the firewall object has yet to be deployed, for example. While using HATEOAS increases the messages exchanged between a management server and the gateway, it allows the responses from the gateway to the management server to be more transparent.

Similarly, instead of modelling the status of a network interface in the table from Section II with a "Disabled" resource, HATEOAS allows an "Enable" or "Disable" action to be presented to the management server based on whether the network interface is active or not, while simplifying the object model of the network interface itself. This leads to less fragmentation of the semantic model, as changes to the management API need not be mapped to the resource model, but could be achieved by changing the necessary hyperlinks which allows less adaptation both at the management server as well at the gateway.

One way to represent the Gateway state information is with a dedicated State Object, composed of as many resources as required to define the Gateway state. An Instance of the State Object could be linked using the "ObjectLink" to a specific Gateway Object or to a Resource that requires state description within the Object.

As seen on Table 5, the new description of a "State Resource" is just a link to the actual Object, which can be added using standard IPSO linking for Composite Objects.

Table 6 shows how the actual Object representation could be. It takes into account all states of the Gateway and possible transitions between states. For simplicity reasons we use two states (ON/OFF, Enabled/Disabled) meaning that the device is on/off. The transitional state is necessary when changes between states are not immediate.

The advantage of representing device states in this manner is that they can be queried using the same REST operations (GET,PUT,DELETE…) as any other resource. Applications can therefore also represent and observe the state of the device.

Resource Name	Access Type	Type	Multiple Instances?	Description
State	R,W	ObjLink	No	Object defining the device state

Table 5. New State Resource on the Gateway Object linking to the State Object Instance.

Resource Name	Resource ID	Access Type	Description
Disabled	0	R,W	The Gateway is off and the interfaces are disabled.
Waiting	1	R,W	The Gateway is in a transitional state (1000 ms).
Enabled	2	R,W	The Gateway is on and the interfaces are enabled.

Table 6. Example representation of a State Object Instance for a Gateway using IPSO data models.

IV. SEPARATING REACHABILITY FROM RESOURCE INTERACTION IN DATA MODELS

A distinguishing feature of an IoT gateway is its ability to communicate and serve its resources over two or more connection endpoints. We earlier discussed semantic models for IoT gateways possessing multiple network interfaces acting as proxies between entirely different protocol stacks and radio technologies. However, gateways can also perform proxying between two disparate transports for the same management or application protocol.

As an example, the LWM2M protocol relies on using CoAP for obtaining management and resource information, with CoAP itself has been standardised to use UDP and DTLS. However LWM2M is also specifying TCP and TLS as viable bearer transports for cellular networks, as network management functionality in IoT can often be hampered owing to the existence of middleboxes such as firewalls and NATs between the management server and the managed nodes. Additionally, the use of Websockets has also been proposed for consideration as an alternative transport for CoAP, for similar reasons.

A LWM2M gateway thus can function as a proxy for LWM2M/CoAP clients over UDP in its LAN, while using a TCP or Websocket endpoint in its WAN for communication with a LWM2M server. However, proxying management operations with LWM2M is not really addressed presently. Therefore, with the currently available data model, the gateway needs to masquerade the objects and resources of each endpoint on its LAN as if they were its own, to an external management server.

Ideally, discovering and using proxy functionality should be incorporated into future data models which unambiguously assert the location and type of managed endpoints residing behind such a gateway. Although the CoAP protocol supports a Proxy-URI option, how this can be achieved and defined in a Data Model remains an open issue. However, in doing so, the semantic model becomes expressive enough to cleanly separate end-to-end resource retrieval and interaction at runtime, from any underlying transport-specific reachability issues. As the gateway resides in the path between a management server and managed endpoints, a similar principle can also be applied, in which the gateway functions as a translator for very resource-constrained endpoints, providing semantic interoperability for connected end devices.

In terms of caching and validation, an added benefit of this role is that the gateway then possesses and provides a cached representation of the schema itself, and is able to locally validate the translation of objects and resources exposed by a very constrained end-point. Additionally a gateway acting as a CoAP proxy may be requested to retrieve the same resource from a CoAP endpoint over multiple transports. This could be possible if such an endpoint exposes its resources over both UDP and DTLS (or UDP and TCP), for example. Instead of retrieving the same requested resource representation multiple times, the gateway can ideally returned an already cached, valid representation if one exists. Although some work exists in understanding retrieval of CoAP-based resources over multiple transports from an endpoint, it is still in its infancy [7].

REFERENCES

[1] Tschofenig, H., Arkko, J., Thaler, D., and D. McPherson, "Architectural Considerations in Smart Object Networking", RFC 7452, DOI 10.17487/RFC7452, March 2015

[2] Pras, A. and J. Schoenwaelder, "On the Difference between Information Models and Data Models", RFC 3444, DOI 10.17487/RFC3444, January 2003

[3] Open Moblie Alliance, “Lightweight Machine-to-Machine Technical Specification v1.0, Candidate Enabler,” December 2015

[4] Bluetooth SIG, “GAP REST API White Paper”, April 2014

[5] Bluetooth SIG, “GATT REST API White Paper”, April 2014

[6] OpenWrt UCI System, https://wiki.openwrt.org/doc/uci, Accessed February 2016

[7] Silverajan, B, “CoAP Protocol Negotiation”, ID-silverajan-core-coap-protocol-negotiation, September 2015