Project Deliverable

Project Number:	Project Acronym:	Project Title:
287901	BUTLER	uBiquitous, secUre inTernet-of-things with Location and contExt-awaReness

Instrument:	Thematic Priority
Integrated Project	Internet of things

Title

D4.3 - Smart Object GW Platform Functional Specification

Contractual Delivery Date:	Actual Delivery Date:
September 2013	October 2013

Start date of project:	Duration:
October, 1st 2011	36 months

Organization name of lead contractor for this deliverable:	Document version:
ST-I	V1.13

Dissemination level (Project co-funded by the European Commission within the Seventh Framework Programme)		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission	
RE	Restricted to a group defined by the consortium (including the Commission)	
CO	Confidential, only for members of the consortium (including the Commission)	

Authors (organizations) :

Yazid Benazzouz, Levent Gurgen, Christine Hennebert, Diana Moreno Garcia, Christophe Munilla (CEA)
Julien Delsuc, Philippe Smadja (Gemalto)
Shadi Atalla, Federico Rizzo, Antonio Simone, Francesco Sottile (ISMB)
François Nacabal (Maya)
Fabien Castanier, Stefano Pascali, Antonio Vilei (ST-I)
Cristina Frà, Massimo Valla (TIL)
Juan Sancho (TST)
Anup Shrestha (Zigpos)

Thanks to Alberto Martinez Cantera, Javier Arcas Ruiz-Ruano (Tecnalia) and Juan Rico (TST) for their review and very useful remarks and suggestions.

Abstract :

The Work Package 4 of the BUTLER project deals with the specification and the implementation of the smart platforms, namely Smart Server (T4.1), Smart Mobile (T4.2) and Smart Objects (T4.3). In the BUTLER architecture, the various platforms are interconnected to provide users with context aware services. Standing between the objects and the other BUTLER platforms, an important functional component called a gateway is needed. The latter, referred to as the BUTLER SmartObject Gateway in the context of this project, enables connectivity between objects located in short range area networks (e.g. Personal, Body or Home Area Networks) and other entities such as mobile devices and servers. Another important feature of the BUTLER SmartObject Gateway is exposing the services implemented by the Smart Objects and allowing access to their resources in a way that is independent of device specific technology. To do so, the gateway implements a coherent and exhaustive set of APIs enabling BUTLER application developers to focus on the service implementation and not on the issues related to the underlying connectivity/protocol or on how the objects are physically implemented and deployed.

This report describes in details the functional implementation of the BUTLER SmartObject Gateway.

Keywords :

Gateway, Objects, Resources, Protocols, Discovery, Management, Network Monitoring, Security Services, Architecture, Adapters, Bridges, API

Disclaimer

Revision History

The following table describes the main changes done in the document since it was created.

Revision	Date	Description	Editors (Organisation)
V1.0	July 2013	Creation	S. Pascali (STMicroelectronics)
V1.1	September 30, 2013	Major Update	A. Vilei (STMicroelectronics)
V1.2	October 14. 2013	Integration of contributions from TST, ISMB.	F. Castanier, A. Vilei (STMicroelectronics)
V1.3	October 16, 2013	Integration of contributions from CEA	F. Castanier, A. Vilei (STMicroelectronics)
V1.4	October 17, 2013	Reviewed section about Gateway Management	F. Castanier, A. Vilei (STMicroelectronics)
V1.5	October 17, 2013	Integration of contributions from Gemalto, Maya, Zigpos	F. Castanier, A. Vilei (STMicroelectronics)
V1.6	October 18, 2013	Preparation of Abstract	F. Castanier, A. Vilei (STMicroelectronics)
V1.7	October 21, 2013	Integration of contributions from CEA, ISMB	F. Castanier, A. Vilei (STMicroelectronics)
V1.8	October 21, 2013	Updated References and integration of new contributions from CEA, TIL, Zigpos, Gemalto	F. Castanier, A. Vilei (STMicroelectronics)
V1.9	October 21, 2013	Updated contributions from CEA	D. S. Moreno Garcia (CEA)
V1.10	October 23, 2013	Updated document based on feedback from internal reviews	F. Castanier, A. Vilei (STMicroelectronics)
V1.11	October 28, 2013	Editorial improvements	F. Castanier, A. Vilei (STMicroelectronics)
V1.12	October 29, 2013	Pre-release	F. Castanier, A. Vilei (STMicroelectronics)
V1.13	October 30, 2013	Final review	B. Copigneaux (inno)

Table of Content

1. EXECUTIVE SUMMARY 8

2. ACRONYMS 10

3. SMARTOBJECT GATEWAY ARCHITECTURE 12

4. DEVICE PROTOCOL ADAPTER 14

4.1. Overview 14

4.2. Protocol Stacks 15

4.2.1. CoAP/6LoWPAN 15
4.2.2. ZigBee 18
4.2.3. NFC 19
4.2.4. Other Protocols 20

4.3. Protocol Bridges 21

4.3.1. CoAP/6LoWPAN Protocol Bridge 21
4.3.2. ZigBee Protocol Bridge 22
4.3.3. NFC Protocol Bridge 23
4.3.4. Other Bridges 27

4.4. Device Access API 28

5. SMART OBJECT ACCESS AND CONTROL 36

5.1. Overview 36

5.2. Resource Manager 38

5.2.2. Discovery process 38

5.3. Resource Directory 40

5.4. Connectivity Handler 41

5.5. Data Manager 42

5.5.1. Freshness Model 42

5.6. Data Storage 43

5.7. Message Cache 44

5.8. Security Services 45

5.8.1. SmartObject Gateway Security services requirements 45
5.8.2. Resource Registration 46
5.8.3. Resource Key Management 46
5.8.4. Security Application Protocol – example of the HTTP Binding 49
5.8.5. SmartObject Gateway Security Service API 50

5.9. Network Monitoring 53

5.9.1. Classification of Network Monitoring Techniques 53
5.9.2. BUTLER Network Monitoring Strategy 55
5.9.3. Network Monitoring Design Guidelines 55
5.9.4. BUTLER Network Monitoring Architecture 57
5.9.5. Network Monitoring Statistics 57

5.10. Consumer API 59

5.10.1. Consumer API Request and Response protocol 59
5.10.2. Security Management at Consumer API 61

6. GATEWAY MANAGEMENT 63

6.1. Overview 63

6.2. Device Manager 64

6.2.1. Device Management Overview 64
6.2.2. Device Management API 65

6.3. Gateway Manager 66

6.3.1. Gateway Manager Overview 66
6.3.2. Service Management API 69
6.3.3. Manager API 70

7. CONSUMER PROTOCOL ADAPTER 71

7.1. Overview 71

7.2. JSON-RPC Bridge 72

7.3. REST API 75

7.3.1. NFC Use Case Example using REST API 80

8. REFERENCES 85

List of Figures

Figure 1–1: Interactions among BUTLER Platforms 8
Figure 1–2: Smart Object Gateway: High level architecture 8
Figure 3–1: SmartObject Gateway: APIs and Functional Components 12
Figure 4–1: Device Protocol Adapter functional group 14
Figure 4–2: Web services vs IoT 15
Figure 4–3: 6LoWPAN network topology 16
Figure 4–4: ZigBee 2007 Network Stack 18
Figure 4–5: ZigBee IP Network Stack 18
Figure 4–6: NFC 19
Figure 4–7: ZigBee Protocol Bridge Architecture 22
Figure 4–8: NFC Protocol Bridge Architecture 23
Figure 4–9: Device implementations 33
Figure 4–10: Service Implementations 33
Figure 4–11: Ranging measurements with respect to neighbouring nodes 34
Figure 4–12: Ranging Data API description 35
Figure 5–1: Smart Object Access and Control functional group 36
Figure 5–2: Service and Resource Model 38
Figure 5–3: Device Discovery 39
Figure 5–4: Service and resource registration 40
Figure 5–5: Theoretical data model for Message Cache 44
Figure 5–6: Registration of key material (set by the owner) 47
Figure 5–7: Retrieval of Access Token sequence diagram 47
Figure 5–8: End to end security sequence diagram 48
Figure 6–1: Gateway Management functional group 63
Figure 6–2: System/Device Management interface 64
Figure 6–3: OSGi Web Console (view 1) 66
Figure 6–4: OSGi Web Console (view 2) 66
Figure 6–5: Bundle lifecycle 68
Figure 6–6: iPOJO instance lifecycle 68
Figure 7–1: Consumer Protocol Adapter functional group 71

List of Tables

Table 4–1: Basic NFC functionalities 24
Table 4–2: Extended NFC functionalities 25
Table 4–3: Addition parameters for NFC API calls [] 25
Table 4–4: Device Methods 28
Table 4–5: Service Methods 29
Table 4–6: Resource Methods 30
Table 4–7: ActionResource Types 30
Table 4–8: DataResource Type 31
Table 4–9: ModifiablePropertyResource Type 31
Table 4–10: StateVariableResource Type 31
Table 4–11: SensorDataResource Type 31
Table 4–12: ActionResource Type 32
Table 4–13: Observable Interface 32
Table 4–14: SendRangingData API description 35
Table 5–1: BUTLER Security Roles 45
Table 5–2: Resource Description at the Authorization Server 46
Table 5–3: HTTP M2MS Protocol 49
Table 5–4: Example Request and Response Data 49
Table 5–5: Example for /WEB-INF/web.xml 51
Table 5–6: Security Material Configuration file for “temperature” 52
Table 5–7: Exploration Methods 59
Table 5–8: Access Methods 60
Table 5–9: Request and Response 61
Table 5–10: Internal Security API 62
Table 6–1: Device Management API 65
Table 6–2: Service Management API 69
Table 6–3: Manager API 70
Table 7–1: Example – Retrieve the list of installed TV sets 73
Table 7–2: Example – Play a video on a given TV 74
Table 7–3: Example – Get the list of objects 76
Table 7–4: Example – Representation of a light device with two LEDs 76
Table 7–5: Example – Retrieve the model information from an object 77
Table 7–6: Example – Dim LED light 77
Table 7–7: Example – Turn on LED light 77
Table 7–8: Example – Get the list of actions supported by an object 77
Table 7–9: Example – HTTP PUT method 77
Table 7–10: Example – Subscription to state change for LED light 78
Table 7–11: Example – Subscription to value change for temperature sensor 78
Table 7–12: Example – Error response 79

1. Executive Summary

The aim of this document is to define the functional specification of the BUTLER Smart Object Gateway. The latter is a component that allows interconnection of different networks to achieve access and communication among embedded devices, servers (the Smart Server Platform in the BUTLER case) and mobile terminals (the Smart Mobile Platform in the BUTLER case).

Figure 1–1 shows the interactions of the Smart Object Gateway with the other BUTLER platforms.

Figure 1–1: Interactions among BUTLER Platforms

Figure 1–2 presents an overview of high level architecture of the Smart Object Gateway.

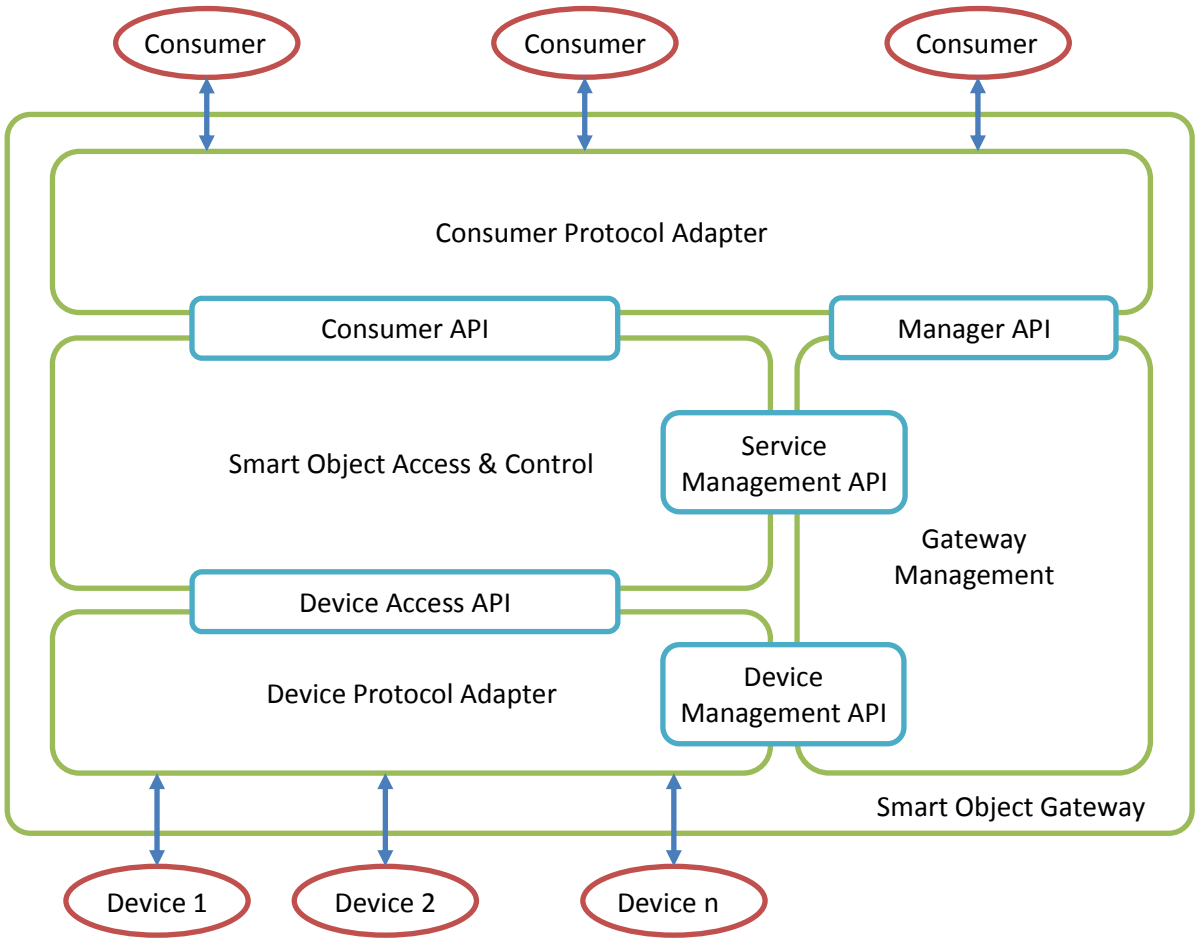

Figure 1–2: Smart Object Gateway: High level architecture

The Smart Object Gateway is composed of four **functional groups** and their relative interfaces:

- Device Protocol Adapter
- Smart Object Access and Control
- Gateway Management
- Consumer Protocol Adapter

The **Device Protocol Adapter** functional group allows abstracting the specific connectivity technology of different wireless sensor networks. It is made of several bridges associated to each protocol stack. All the bridges comply with a generic Device Access API used to interact with northbound services of the SmartObject Gateway.

The **Smart Object Access and Control** functional group implements the core functionalities of the SmartObject Gateway like discovering devices and resources and securing communication among devices and consumers of their services. It also performs caching of data for battery operated sensors and actuators to save power. The services of the Smart Object Access and Control functional group are exposed to Consumers by means of a protocol agnostic **Consumer API**.

The **Gateway Management** functional group includes all the components needed to ease management of devices connected to the SmartObject Gateway, regardless of their underlying technologies. A **Device Management API** is used for this purpose. This functional group also contains the components managing cache, resource directory and security services. These management features are exposed by means of the **Manager API**.

The **Consumer Protocol Adapter** functional group consists of a set of protocol bridges. Each of them translates the protocol agnostic **Consumer API** and **Manager API** interfaces into specific application protocols (e.g. JSON RPC, RESTful HTTP, etc.).

In the BUTLER Security Framework, the Authorization Server is the single point for security management. All actors shall delegate authorization management and user management to the Authorization Server.
Using these implementation principles, the Resource Provider which in this case is the SmartObject Gateway, does not encompass any sensitive information about calling applications. BUTLER Security Protocol allows secure authentication of the application requesting a resource and implements end-to-end security between the Consumer and the SmartObject Gateway.

2. Acronyms

Acronym	Defined as
6H	6LoWPAN Host
6LoWPAN	IPv6 over Low Power Wireless Area Network
6LBR	6LoWPAN Border Router
6LR	6LoWPAN Router
AoA	Angle of Arrival
API	Application Program Interface
APS	Application Support Sublayer
BS	Base Station
CoAP	Constrained Application Protocol
CoRE	Constrained RESTful Environments
CE	Card Emulation
CEPT	European Conference of Postal and Telecommunications Administrations
CH	Connectivity Handler
CNM	Centralised Network Monitoring
CRUD	Create Read Update Delete
DNM	Distributed Network Monitoring
ECMA	European Computer Manufacturers Association
FC	Functional Component
FG	Functional Group
I2C	Inter Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IoT	Internet of Things
JSON	JavaScript Object Notation
HTML	Hyper Text Mark-up Language
HTTP	Hyper Text Transfer Protocol
LDAP	Lightweight Directory Access Protocol
LE	Localisation Engine
M2M	Machine to Machine
MAC	Medium Access Control
MAN	M2M local Area Network
NFC	Near Field Communication
NWK	Network layer
OSGi	Open Services Gateway Initiative
OSI	Open Systems Interconnection
PAN	Personal Area Network
P2P	Peer To Peer
RA	Router Advertisement

RDF	Resource Description Framework
REST, RESTful	REpresentational State Transfer
RPC	Remote Procedure Call
RPL	Routing Protocol for Low Power and Lossy Networks
RS	Router Solicitations
RSSI	Received Signal Strength Indicator
RTT	Round Trip Time
R/W	Read/Write
SLIP	Serial Line Internet Protocol
SPI	Serial Peripheral Interface
TCP	Transmission Control Protocol
ToA	Time of Arrival
UDP	User Datagram Protocol
URI	Unified Resource Identifier
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
WSN	Wireless Sensors Network
ZCL	ZigBee Cluster Library
ZDO	ZigBee Device Object

3. SmartObject Gateway Architecture

This chapter expands the simplified architecture of the SmartObject Gateway presented earlier in the executive summary section. Compared to Figure 1–2, indeed, Figure 3–1 provides a more detailed view of the internal components making up each of the four SmartObject Gateway functional groups.

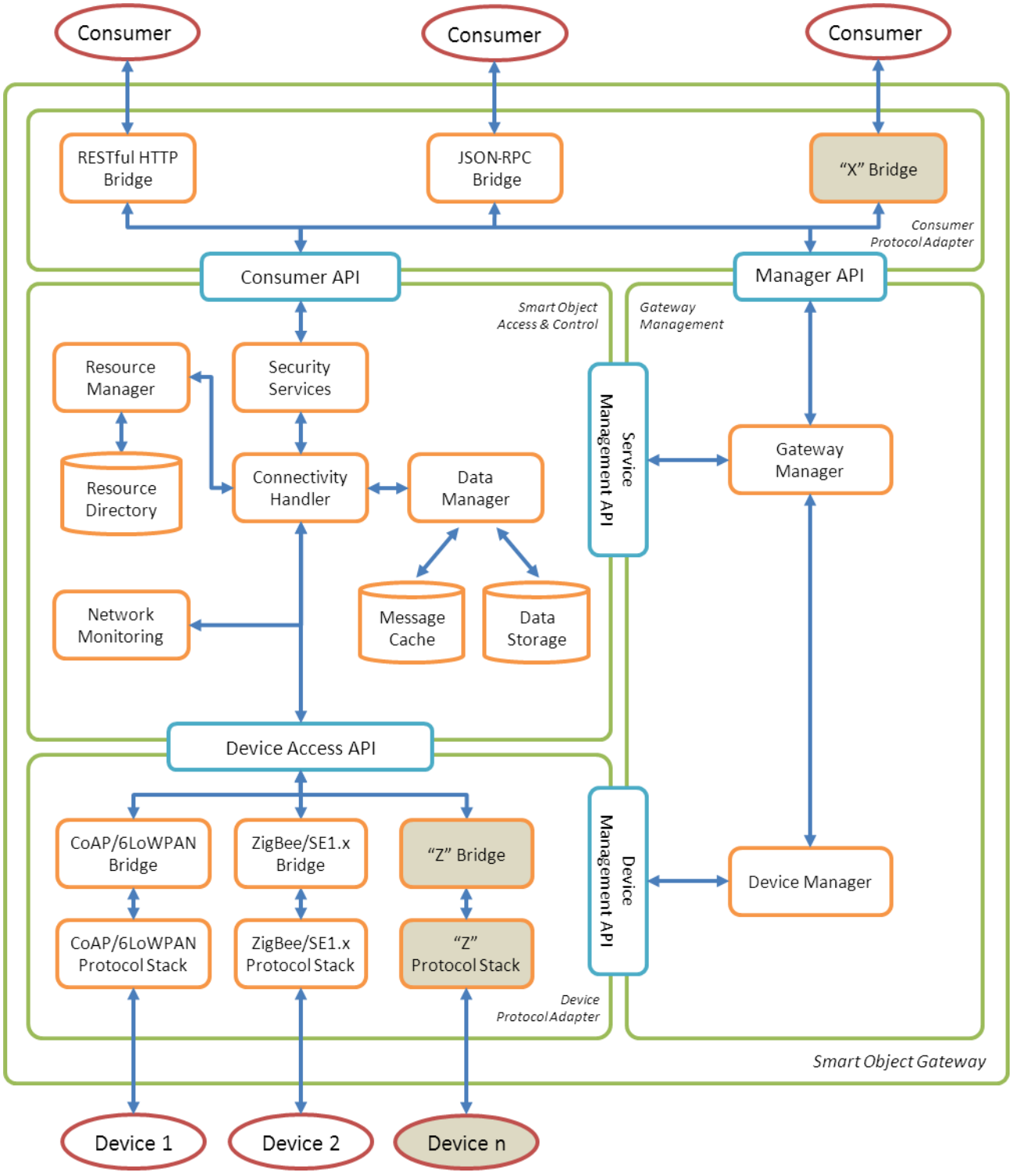

Figure 3–1: SmartObject Gateway: APIs and Functional Components

The Device Protocol Adapter functional group is composed of a set of bridges related to specific connectivity protocols as described in subsequent Chapter 4.

The Smart Object Access and Control functional group is made up of the following components:

- Resource Manager
- Resource Directory
- Connectivity Handler
- Data Manager
- Data Storage
- Message Cache
- Security Services
- Network Monitoring

Details about the above functional components are presented in Chapter 5.

The Gateway Management functional group, instead, is described in Chapter 6.

Finally, the Consumer Protocol Adapter functional group consists of a set of bridges related to specific consumer protocols as defined in Chapter 7.

4. Device Protocol Adapter

4.1. Overview

The main role of the Device Protocol Adapter functional group (Figure 4–1) is to allow connectivity to sensors and actuators networks, regardless of their connectivity protocols.
In order to provide this functionality, for each of the supported protocols (e.g. CoAP/6LoWPAN, ZigBee, NFC, etc.) the SmartObject Gateway must have a dedicated physical interface, compatible with the physical layer of the specific sensor network. Obviously the physical layer is only the starting point. A dedicated protocol stack implementation is required too, coupled with a dedicated bridge towards the Device Access API that will connect this Device Protocol Adapter to the Smart Object Access and Control functional group.

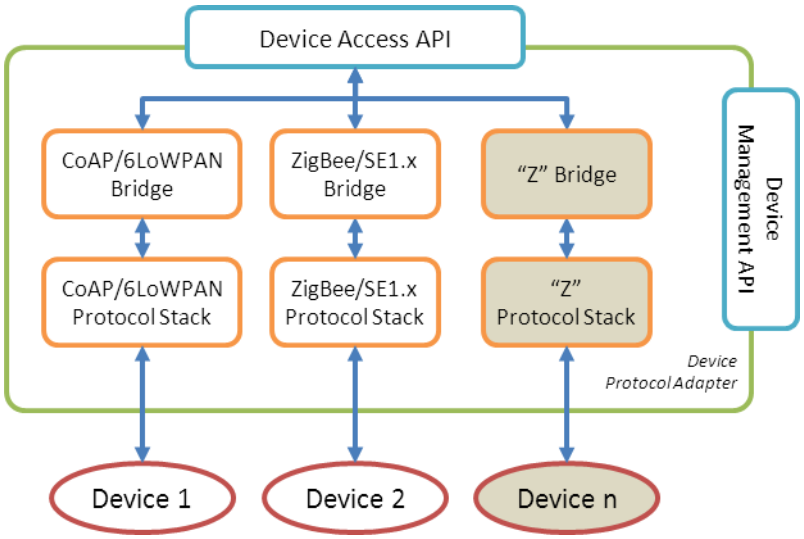

Figure 4–1: Device Protocol Adapter functional group

4.2. Protocol Stacks

A Protocol Stack component represents the implementation of a specific communication protocol. It handles device discovery and access to sensor and actuator resources in a protocol specific way.

This Functional Component communicates with the same protocol technology devices (southbound) and with the relative protocol bridge (northbound).

4.2.1. CoAP/6LoWPAN

6LoWPAN (IPv6 over Low Power Wireless Area Network) [4-1] represents an adaptive layer that defines encapsulation and header compression mechanisms in order to allow IPv6 packets to be exchanged over IEEE 802.15.4 networks. Moreover, 6LoWPAN aims to adapt the IP protocol also for low-power and constrained devices.

As an example, Figure 4–2 shows a network architecture in which the 6LoWPAN provides an IPv6 adaptation layer with respect to the IEEE 802.15.4 [4-2] while on top of it the following protocols and standards are typically used: RPL [4-3], UDP and CoAP [4-4]. In particular, IEEE 802.15.4 is a standard that specifies the physical and the MAC layer for smallest and constrained devices in Wireless Personal Sensor Networks. RPL is a network layer protocol specifically designed for constrained network. At transport layer, UDP protocol is usually used since it is a protocol that does not provide retransmission and with a reduced header (lighter than TCP). At application level, CoAP protocol brings the REST architecture within the constrained networks.

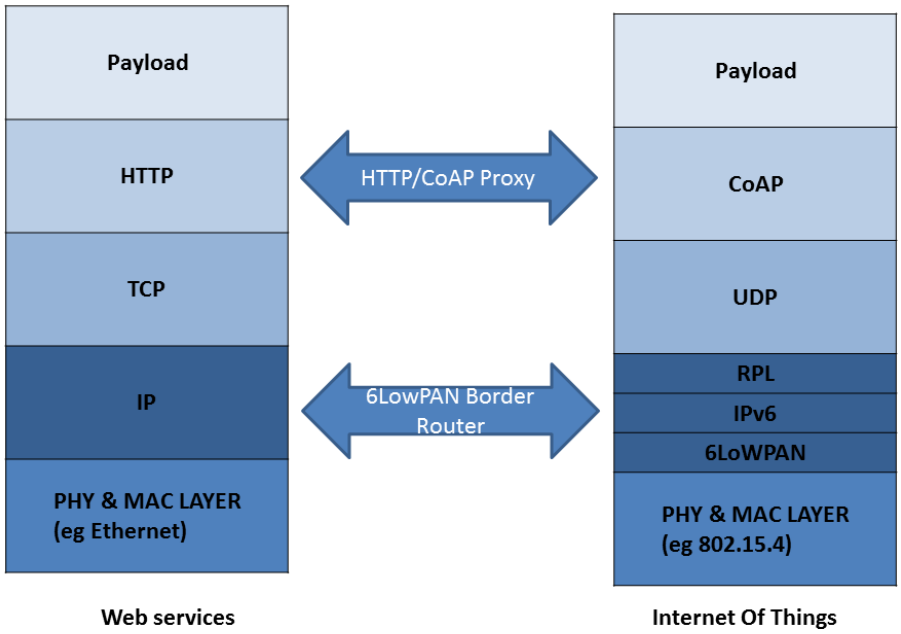

Figure 4–2: Web services vs IoT

Network topology updates and device discovery

The 6LoWPAN network topology provides the presence of three main elements (see Figure 4–3):

- 6LoWPAN host (6H)
- 6LoWPAN router (6LR)
- 6LoWPAN border router (6LBR)

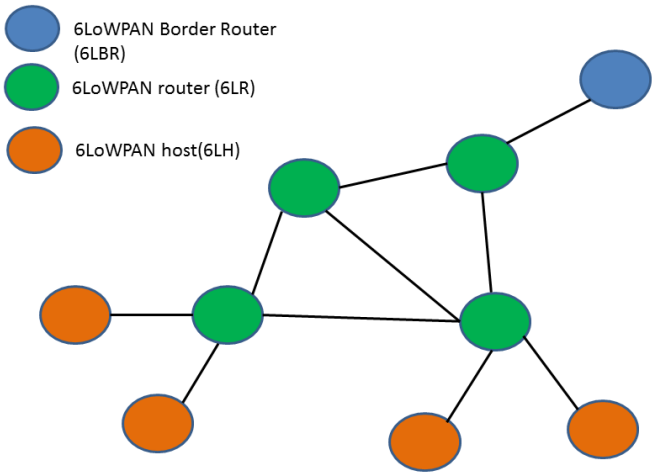

Figure 4–3: 6LoWPAN network topology

The border router is able to handle all the 6LoWPAN sensor nodes within the network area. In particular, it maintains a list of all sensors IPv6 addresses, i.e. a neighbour cache containing the hosts and 6LRs which is updated periodically.

A 6LBR is responsible for connecting Smart Object devices to the SmartObject Gateway and it is also responsible for handling traffic to and from the 802.15.4 and IPv6 interface. The 6LoWPAN protocol stack provides more than one approach to implement the device discovery mechanisms. One of these foresees that the sink node (either the 6LR or the 6LBR) sends out periodic announcement messages (Router Advertisements, RA); once a node joining the network has received a router advertisement, it can start the registration procedure. In this way, the router is able to know how many nodes are in the network area. Another approach is the following; every node starts the registration process by sending requests to the router (Router Solicitations, RS). The response from the routers is a Router Advertisement with the expected information (e.g. the IPv6 prefix and network parameters). When a new device is added or an old one is removed from the neighbour cache (the area network topology is changed), the bridge relies on the border router for this updates. As mentioned above, the border router maintains a list of all the devices IPv6 addresses. Moreover, it exposes this list via embedded web server. The bridge gets the list of all network devices at new running cycle (based on time out mechanism) by querying the border router HTTP server over a SLIP tunnel. By obtaining the list of IPv6 addresses, the bridge compares the new list with the old one, deciding if a new device is added or an old one was removed from the 6LoWPAN network.

Resource directory access and discovery in 6LoWPAN application

Apart from device discovery procedures, it is necessary to have some mechanisms to access the device resources. In this sense, CoAP provides a lightweight alternative to HTTP for constrained environments and nodes, useful within networks characterized by limited resources. CoAP is a transfer protocol used by M2M applications (which runs on constrained devices such as sensors and actuators) to create web services and to allow the interaction between the various devices within a Wireless Sensor Network. It uses a subset of the HTTP methods (GET, PUT, POST and DELETE) to allow the interaction among the end-points, providing features such as resource-discover and including key concepts such as web URIs (Uniform Resource Identifier) and content-types.

The Internet Engineering Task Force (IETF) Constrained RESTful Environments (CoRE) Working Group has done the standardization work for the CoAP protocol. As for the web, the RESTful architecture is well-suited to most types of M2M applications; this approach allows to address resources by URI and to exchange data associated with the resources via different representations (e.g. via HTML or RDF – Resource Description Framework) through an HTTP/CoAP content negotiation. The RESTful architecture is based on the client-server paradigm. The server contains information that can be saved or retrieved by clients. In this instance, a sensor or an actuator can be viewed as a server and the application or the gateway is treated as the client.

4.2.2. ZigBee

According to the Zigbee Standards Organization 2007, the Zigbee stack architecture is made up of blocks called layers. The IEEE 802.15.4-2003 standard defines the two lower layers: the physical (PHY) layer and the medium access control (MAC) sub-layer. The ZigBee Alliance builds on this foundation by providing the network (NWK) layer and the framework for the application layer. The application layer framework consists of the application support sub-layer (APS) and the ZigBee device objects (ZDO). Manufacturer-defined application objects use the framework and share APS and security services with the ZDO.

<table>
<tr><td>Application Framework</td><td>ZigBee Device Object (ZDO)</td></tr>
<tr><td colspan="2">Application Support Sublayer (APS)</td></tr>
<tr><td colspan="2">Network Layer (NWK)</td></tr>
<tr><td colspan="2">Medium Access Control (MAC) Layer</td></tr>
<tr><td colspan="2">Physical (PHY) Layer</td></tr>
</table>

Figure 4–4: ZigBee 2007 Network Stack

The ZigBee network using this protocol stack should have a ZigBee gateway to get the IP connectivity.

The ZigBee IP specification February 2013 defines the following protocol stack for the IP enabled Zigbee device:

<table>
<tr><td>Applications</td></tr>
<tr><td>Transport layer (TCP / UDP)</td></tr>
<tr><td>Network Layer (NWK)
(IPV6, ICMPv6,6LP-ND)</td></tr>
<tr><td>6LoWPAN adaptation layer</td></tr>
<tr><td>Medium Access Control (MAC) Layer</td></tr>
<tr><td>Physical (PHY) Layer</td></tr>
</table>

Figure 4–5: ZigBee IP Network Stack

Using the ZigBee IP devices following this protocol stack, the ZigBee IP router itself can provide the internet connectivity.

4.2.3. NFC

NFC stands for Near Field Communication, a very short range communication technology intended for low power, low data rate communications. The NFCIP-1 (Near Field Communication Interface and Protocol 1) and later on the NFCIP-2 specifies NFC basic capabilities:

- Frequency operation band: 13.56 Mhz
- Transfer rates: 106, 212 and 424 kbps
- Operation range: up to 20 cm, being 8 cm a regular utilization range
- Bit encoding and modulation: Modified Miller 100% ASK and Manchester 10% ASK
- Communication modes: Active and Passive
- Role assignment: Initiator and Destination
- Transport protocol: Activation, Data Exchange and Deactivation

The following picture depicts how NFC messages are handled by different Tag types; these ones, mapped into existing standardized protocols, are finally managed by the different existing NFC versions.

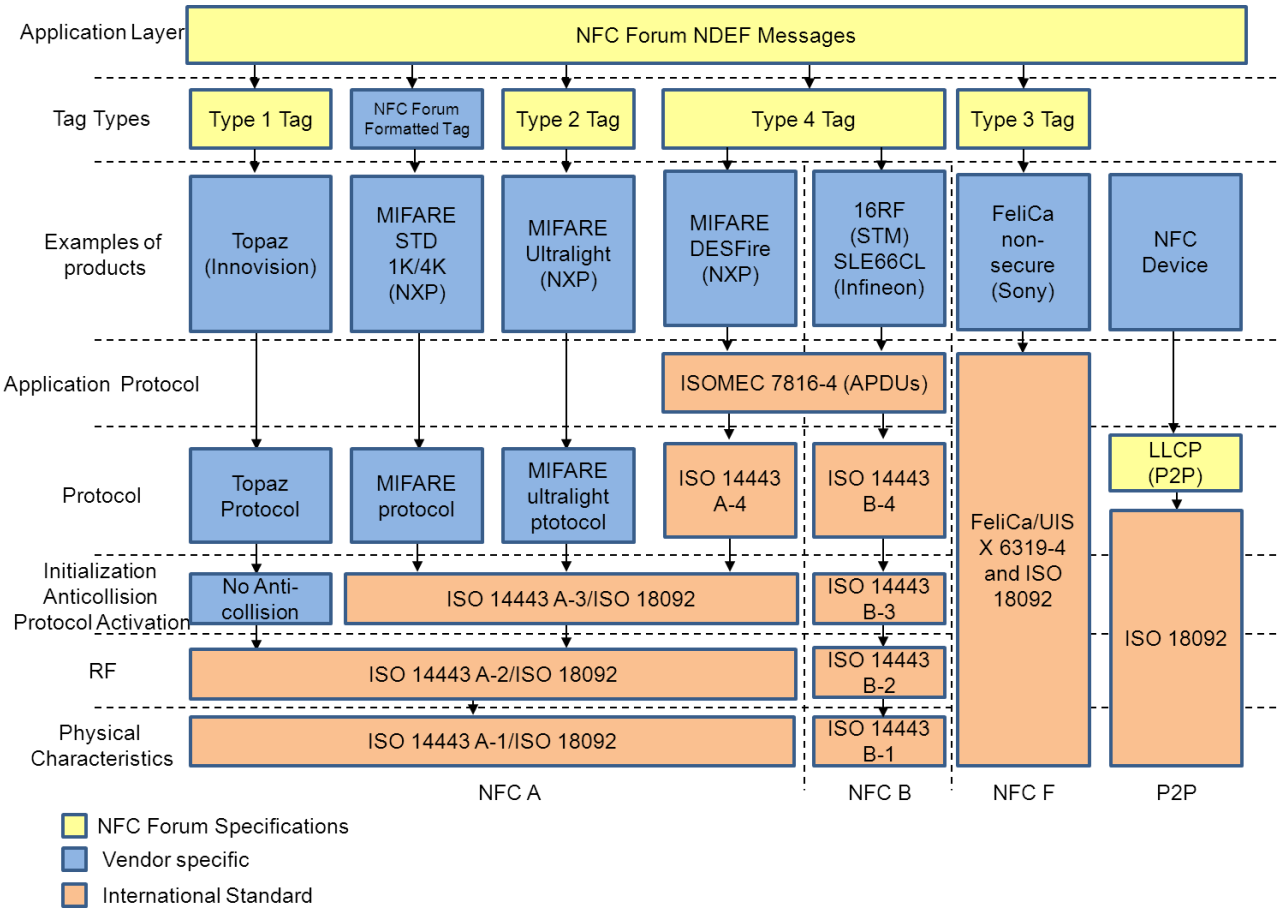

Figure 4–6: NFC

The NFC technology specification has been fully described in Section 4.3.1.2.2 of [D3.1].

4.2.4. Other Protocols

The BUTLER SmartObject Gateway architecture is flexible enough to allow support for devices using additional protocols. Such new devices and protocols are generically represented as “Z” *Device* and “Z” *Protocol Stack* respectively in Figure 4–1.

4.3. Protocol Bridges

For each specific connectivity technology supported by BUTLER there must be a Protocol Bridge that allows low level access to the physical object and provides integration with the Device Access API. The latter is a generic Java API that allows exporting object data to the rest of BUTLER in a technology independent fashion.

Bridges provide the following functionalities:

- Abstract details about underlying technology
- Discover Devices
- Track events occurred on them
- Execute message to actuate them

Moreover, bridges are responsible for device's accessibility, by discovering and exposing device resources. Once a device is discovered and notified to the relative bridge, an identifier is assigned to it. The discovery message is then sent by the bridge to the Device Access API.

The following subsections add more details for the specific bridges supported by the SmartObject Gateway.

4.3.1. CoAP/6LoWPAN Protocol Bridge

Mainly this functional component shall act as a bridge that allows interaction between CoAP devices and the Device Access API.

After the discovery phase, this Protocol Bridge registers the device as a service (BUTLER Smart Object Service), in order to represent the device in the BUTLER world. In this phase the GW still knows very little about the device. Thus, in order to learn more about it and its capabilities, or to interact with the device itself, the bridge must retrieve the device's description from the URL provided by the device in the discovery message. Hence, it uses the IPv6 to send a CoAP discover message with a GET to the "/.well-known/core" URI. Then the bridge parses the response payload to build the resource tree. It looks for the first level resource and recognizes only "sensor" and "actuator" string. The string "sensor" will be mapped as sensor application(s) and "actuator" will be mapped as actuator application(s).

4.3.2. ZigBee Protocol Bridge

The purpose of the ZigBee Protocol Bridge is to connect the ZigBee devices into the BUTLER network via Device Access API. The bridge should act as a data server, providing ZigBee-related data from PAN network measurements, an interface to control the PAN nodes and discovery and registry mechanism in the SmartObject Gateway. The bridge works as a PAN coordinator, so it has to process and maintain essential information for the SmartObject Gateway such as network addresses of the nodes, their state and capabilities, and should be visible via Device access API.

The Bridge should translate the Device access API into a ZCL (ZigBee Cluster Library) API and should provide connectivity between the ZigBee PAN coordinator and the SmartObject Gateway.

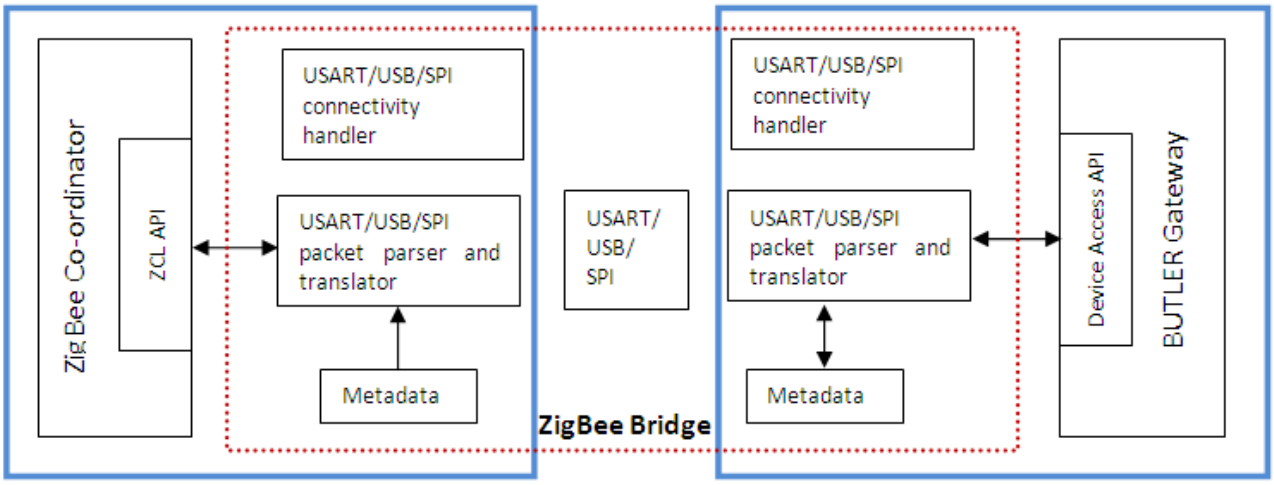

Figure 4–7: ZigBee Protocol Bridge Architecture

The bridge, as shown in Figure 4–7, is made of two modules: the first one, connected to or running in the ZigBee PAN coordinator, provides an interface via a physical medium like USB or USART or SPI; the other one, coupled or embedded in the SmartObject Gateway, has to parse the Device Access API commands into an intermediate request/response code to the ZigBee PAN as specified by the metadata, which in turn might depend on the particular application scenario like Home Automation or Smart Energy profile or any proprietary extension of IEEE 802.15.4 PAN. The metadata also contains information about the PAN, the resources and services provided and thus, the SmartObject Gateway uses it for the discovery and registration of the ZigBee PAN. The module coupled in the ZigBee coordinator provides this metadata. For proprietary ZigBee extensions, the vendor should provide the libraries equivalent to the ZCL.

4.3.3. NFC Protocol Bridge

As proposed in Section 4 of [D3.2], an NFC Service is made available for very short range communications. This subchapter deepens into the interfaces between the actors involved in the NFC scenario, from the Users to the Cloud Applications running in Smart Servers, passing through the NFC Modules, Smart Object Devices and SmartObject Gateway. Going from top-down on the following picture:

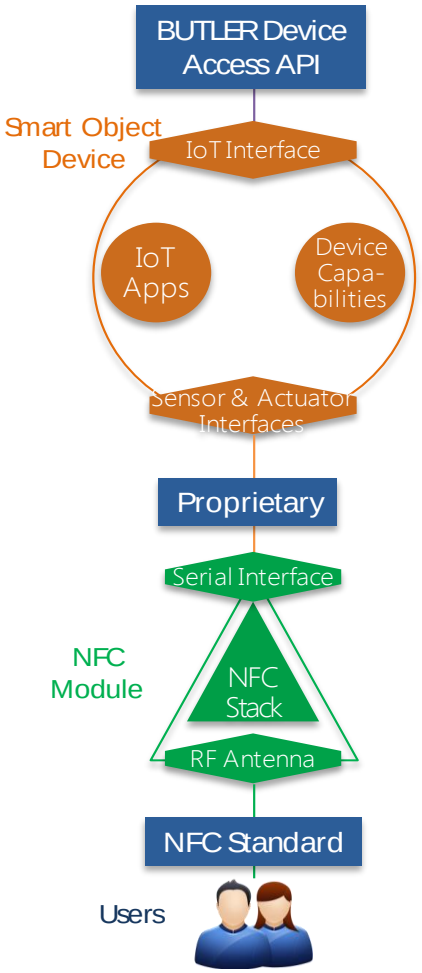

Figure 4–8: NFC Protocol Bridge Architecture

- Communication between *Smart Object Devices* and Sensors & Actuators attached to them, in this case *NFC Modules*, is done by means of a proprietary protocol via a Serial Interface such as I2C, Modbus, SPI, and the like.

- Finally, the *Users* of this particular NFC Service communicate with the *NFC Modules* using their NFC-enabled smartphone or SmartCard, both operations under the umbrella of the **NFC Standard**.

NFC Service API

Due to the fact that each NFC Module behaves according to its own set of commands, the BUTLER NFC Service does not define how Smart Object Devices operate over NFC Modules. However, it has been studied that the great majority of NFC Devices concur in a minimum list of NFC operations, which are shown in
Table 4–1.

Scope	Functionality	Parameters	Comments
Configuration (Conf)	SetId	Id	NFC Module identifier
	SetAuthKey	Key	Definition of a secure key
	SetMode	RW, P2P, CE	NFC operational mode
Read/Write (R/W)	ReadTag	Data	Obtains value of predefined parameter
	WriteTag	Data	Alters the value of a given parameter
Peer-to-Peer (P2P)	SendData	Data	Transmits information to an NFC Device
	ReceiveData	Data	Receives information from an NFC Device
Card Emulation (CE)	SetData	Data	Impersonates an NFC Card

Table 4–1: Basic NFC functionalities

Taking advantage of the previous functionalities, the NFC Service builds additional features empowering end-user applications, such as historic reports of scanned tags, creation of alarms, group management and many more.

In the following table there is a categorization of the available features.

Scope	Resource	Function	Comments
NFC API	NFC Tag	GetAll	Returns all scanned NFC Tags
		GetNew	Returns new NFC Tags scanned since last request
		GetRange	Returns NFC Tags scanned between a range of dates
		GetFrom	Returns NFC Tags scanned by a given NFC Module
		SmartTag	Sets attributes of the NFC Tag (Card Emulation mode)
	NFC Module	Get	Lists NFC Modules
		Create	Registers a new NFC Module in the system
		Delete	Deregisters a given NFC Module in the system
		Info	Returns attributes of a given NFC Module
		Configure	Sets attributes of a given NFC Module
	NFC Group	Get	Lists NFC Groups
		Create	Creates a new NFC Group
		Delete	Deletes a given NFC Group
		Info	Returns attributes of a given NFC Group
		Configure	Sets attributes of a given NFC Group
	NFC Alarm	Get	Lists NFC Alarms
		Create	Creates a new NFC Alarm
		Delete	Deletes a given NFC Alarm
		Info	Returns attributes of a given NFC Alarm
		Configure	Sets attributes of a given NFC Alarm

Table 4–2: Extended NFC functionalities

Some of the API calls described above can use additional parameters to filter the query:

Parameter	Usage	Accepted values
limit	Number of results to return	Positive integer
offset	Used to page through results	Positive integer
order_by	Order results according to this parameter	Id, -Id, Timetag, -Timetag
gt	Retrieve events which occurred after the specified date. Use with order_by=timetag	Datetime, ISO 8601 formatted (e.g. 2012-10-01T08:00+01) [1]
lt	Retrieve events which occurred before the specified date. Use with order_by=timetag	Datetime, ISO 8601 formatted (e.g. 2012-10-01T08:00+01)
exact	Exact value matching, use with order_by	datetime if order_by timetag, id_number if order_by id

Table 4–3: Addition parameters for NFC API calls [2]

[1] https://en.wikipedia.org/wiki/ISO_8601
[2] http://open.sen.se/dev/

Resource Attributes

This section outlines the Attributes and Descriptions for each of the NFC API Resources.

1. **NFC Tags** scanned by NFC Modules have the following attributes:

ID	The ID of the NFC Tag
Key	The secret key in use by the NFC Tag
Data	The information payload contained in the NFC Tag
Timetag	Datetime in which the NFC Tag was read

2. **NFC Modules** must be registered at least in one group in order to be accessible from a *Cloud Application*. They have the following attributes:

ID	The ID of the NFC Module
Name	The Name of the NFC Module
Description	The Description of the NFC Module
Key	The secret key in use by the NFC Module
Mode	The operational mode of the NFC Module
Timetag	Datetime of the last reading
Group	A list of groups the NFC Module belongs to

3. **NFC Groups** are abstract resources encompassing a number of NFC Modules. They are assigned to certain *Cloud Applications* and granted with certain access rights.

ID	The ID of the NFC Group
Name	The Name of the NFC Group
Description	The Description of the NFC Group
Members	List of NFC Modules registered in this NFC Group
Permissions	List of permissions for executing commands on this group

4. **NFC Notifications** trigger notifications from *Smart Object Devices* or *SmartObject Gateways* to *Cloud Applications*.

ID	The ID of the NFC Alarm
Name	The Name of the NFC Alarm
Description	The Description of the NFC Alarm
Command	The command to execute
Trigger	The event trigger

4.3.4. Other Bridges

As mentioned in Section 4.2.4, the BUTLER SmartObject Gateway architecture is flexible enough to allow integrating new kind of devices and protocols. Support for such additional protocols can be added by implementing a new bridge, generically represented as “Z” *Bridge* in Figure 4–1.

4.4. Device Access API

The SmartObject Gateway provides an abstract and homogeneous way of representing and accessing different devices and their resources and services through the **Device Access API**.

The Device Access API allows building easily horizontal applications from various domains using the available devices and their provided services and resources. Actually, using the Device Access API, device clients do not need to worry about devices' technologies and communication protocols. The Device Access API is defined by the Service and Resource model introduced in Section 5.2.1.

In the Service and Resource model, services are classified as:

- *SmartObjectServices* which represent services exposed by devices, such as a player service exposed by a TV device; and
- *SmartServices* which represent logical services (not exposed by devices), such as a weather service. Services provide resources that can be then accessed and manipulated by applications and other SmartServices too.

The Device Access API defines methods for accessing and manipulating the resources exposed by services. The main methods of the three core Device Access API interfaces (Device, Service and Resource) are presented next.

4.4.1. Device Interface

The Device interface defines methods for accessing the information provided by a device, including its provided services. Such methods are as follows:

Modifier and Type	Method and Description
java.lang.String	getLocation() Gets the location of the device.
java.lang.String	getSerialNumber() Gets the serial number of the device.
Service	getService(java.lang.String service) Gets the service by its name.
java.util.Set<java.lang.String>	getServiceNames() Gets the names of the device exposed services.
java.util.Set<Service>	getServices() Gets the set of services (smart object services) exposed by the device.
java.lang.String	getStatus() Gets the status of the device.

Table 4–4: Device Methods

4.4.2. Service Interface

The Service interface represents Smart Object Services (provided by devices) and Smart Services. It mainly defines the methods for accessing service exposed resources (i.e. properties, sensor data, state variables and actions): GET, SET, ACT, SUBSCRIBE and UNSUBSCRIBE. Table 4–5 details the main Service interface methods.

Modifier and Type	Method and Description
Resource	getResource(java.lang.String name) Gets the resource identified by its name.
java.util.Set<Resource>	getResources() Gets all the exposed resources.
Data	get(java.lang.String resource) Gets the data of the given resource.
Data	set(java.lang.String resource, java.lang.Object value, java.util.Set<Metadata> metadata) Sets the given data value and set of metadata to the given resource.
Response	act(java.lang.String resource, java.lang.Object... parameters) Executes the given resource with the given input parameters.
Response	subscribe(java.lang.String resource,ResourceListener subscriber, java.util.Set<Condition> conditions, Period period) Subscribes the service client to the given resource.
Response	subscribe(java.lang.String resource, java.lang.String attribute,ResourceListener subscriberListener) Subscribes the service client to the given attribute of the given resource.
Response	unsubscribe(java.lang.String resource, java.lang.String subscriptionID) Cancels the subscription, identified by subscriptionID, to the given resource.

Table 4–5: Service Methods

4.4.3. Resource Interface

This interface represents resources exposed by services (smart object services and smart services). It defines methods related to resources, including methods for accessing directly the resource attributes.

Table 4–6 details the main Resource interface methods.

Modifier and Type	Method and Description
Data	get(java.lang.String attribute) Gets the data of the given attribute.
java.util.Set<Data>	get(java.lang.String[] attributes) Gets the set of data of the given set of attributes.
java.util.Set<Attribute>	getAttributes() Gets the resource attributes.
Method	getAccessMethod(java.lang.String type) Gets the resource access method of the given type.
java.util.Set<Method>	getAccessMethods() Gets the resource access methods.

Table 4–6: Resource Methods

Resources are classified into Data resources (i.e., Property –static or modifiable-, SensorData and StateVariable) and Action resources.

Resource Type		Description
Data	Property	Represents a property exposed by a service. This is information is likely to be static (e.g., model, vendor, serial number, etc.).
	ModifiableProperty	Represents a modifiable property exposed by a service (e.g., owner, price, etc.).
	StateVariable	Represents a variable state property exposed by a service. State variables are mainly modified by actions (for example, turning on the light modifies the light state, opening the door changes the door state).
	SensorData	Represents sensed data exposed by a service. This information is provided mainly by physical devices sensing environment data.
Action		Represents a functionality exposed by a service. Actions can be performed on the physical environment via actuator devices (e.g., turn on the light, open the door), and also on the virtual environment (e.g., make a parking reservation).

Table 4–7: ActionResource Types

Each resource type defines methods specific to the type of resource. For example, the DataResource type defines the *get* method that allows getting the resource data. The specialized StateVariableResource type defines *set* methods that allow modifying the data value of a state variable.

The following tables resume the main methods defined by the different resource types.

Type	Method and Description
Data	get() Gets the resource data.

Table 4–8: DataResource Type

PropertyResource type has the same methods as DataResource (Table 4–8).

Type	Method and Description
Data	set(java.lang.Object value, java.util.Set<Metadata> metadata) Sets the given value to the data resource and adds or updates the given set of metadata.

Table 4–9: ModifiablePropertyResource Type

Type	Method and Description
Data	set(java.lang.Object value, java.util.Set<Metadata> metadata) Sets the given value to the data resource and adds or updates the given set of metadata.
Data	set(java.lang.Object value, java.util.Set<Metadata> metadata, java.lang.String modifier) Sets the given value to the state variable resource for the given action resource modifier and adds or updates the given set of metadata.
java.util.Set<java.lang.String>	getModifiers()

Table 4–10: StateVariableResource Type

Type	Method and Description
Data	getLast() Gets the last sensor data.

Table 4–11: SensorDataResource Type

Type	Method and Description
ActionResponse	act() Executes the action resource.
ActionResponse	act(java.lang.Object... parameters) Execute the resource with the given objects corresponding to the defined input parameters.
void	addParameters(Parameter... parameters) Add the parameters to the act method.
void	removeAccessWithoutParameters() Removes the access without parameters to the act method.

Table 4–12: ActionResource Type

All resource types are observable, meaning that clients can subscribe and unsubscribe to them. The following methods are part of the Observable interface.

Modifier and Type	Method and Description
Response	subscribe(java.lang.String attribute, ResourceListener subscriberListener, java.util.Set<Condition> conditions, Period period) Subscribes the service client to the given resource attribute with the given set of conditions and time period.
Response	unsubscribe(java.lang.String subscriptionID) Cancels the subscription identified by subscriptionID.

Table 4–13: Observable Interface

The provided classes must mainly be extended when developing new types of devices to be integrated into the SmartObject Gateway. Devices must provide the BUTLER Device interface by extending the *AbstractDevice* abstract class. Services must implement the *BUTLER Service* interface by extending either the *SmartObjectService* or *SmartService* abstract class (see figure below). Resources must implement the BUTLER Resource interface by extending one of the specialized resource types: *PropertyResource*, *StateVariableResource*, *SensorDataResource*, *ActionResource*.

Figure 4–9 and Figure 4–10 show this extensibility principle when developing devices and services respectively.

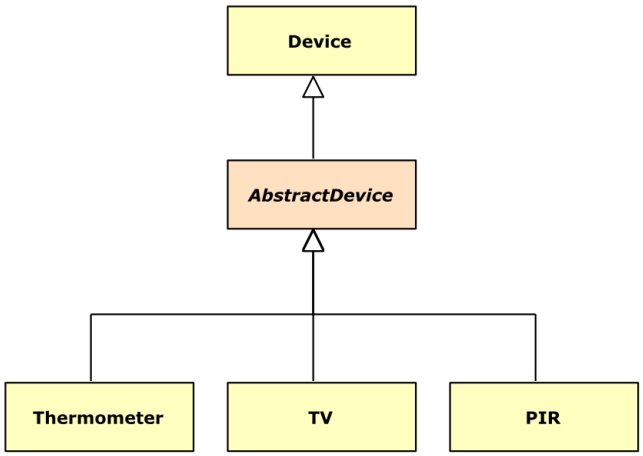

Figure 4–9: Device implementations

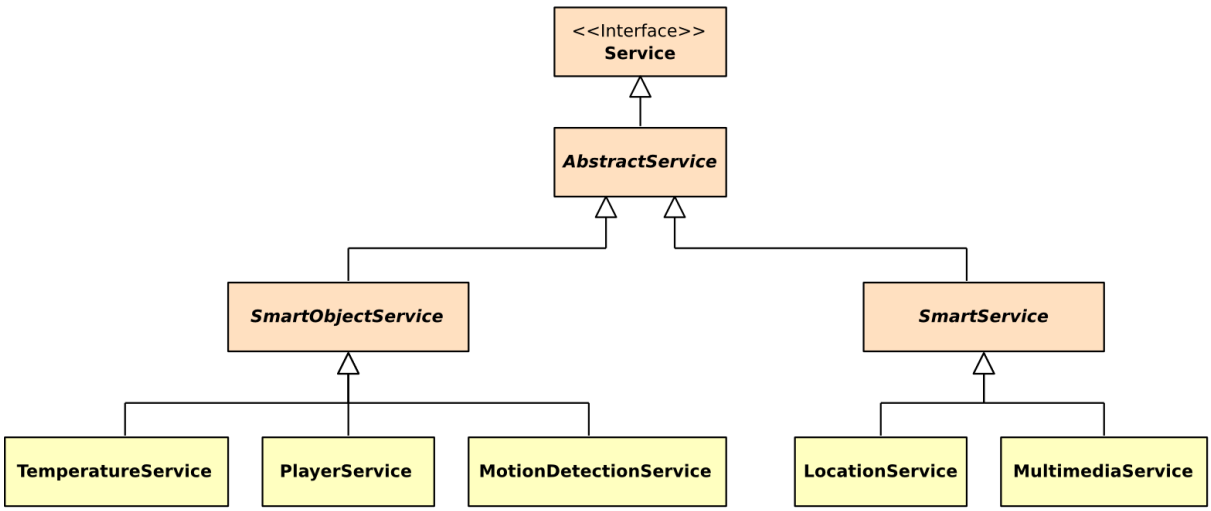

Figure 4–10: Service Implementations

4.4.4. Ranging Data API

Each Smart Object that needs to be localized has to implement a ranging functional module that allows the smart object to perform ranging measurements with respect to its 1-hop neighbours and send the resulting ranging message to the SmartObject Gateway. The task of the GW is just to receive the ranging messages from unknown Smart Objects and forward them to the localization server.

According to the localization architecture, presented in [D3.2], unknown smart objects continuously perform range measurements (e.g., ToA, AoA, RSSI) among them and send these data to the SmartObject Gateway. In turn, the latter sends these raw ranging data to the Smart Server where the Localization Engine (LE) module is executed. Thus, the LE periodically estimates the positions of unknown Smart Objects on the basis of the received ranging data and anchor nodes' positions (anchors are nodes whose coordinates are a priori known).

More in details, as depicted in Figure 4–11, each Smart Object device builds a range data message (represented by the blue arrow) containing range measurements with respect to its 1-hop anchor nodes as well as neighbouring smart objects to allow cooperative localization. Since Smart Objects are mobile nodes and their positions change in time, it is important that ranging measurements are immediately sent to the SmartObject Gateway and in turn forwarded to the Smart Server as soon as they are available, otherwise a potential large delay will be translated into positioning errors.

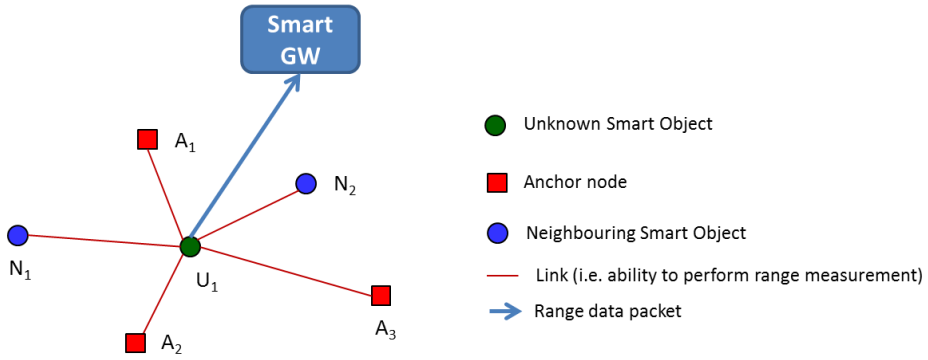

Figure 4–11: Ranging measurements with respect to neighbouring nodes

The ranging data defined in the devices access API is showed in Figure 4–12
.

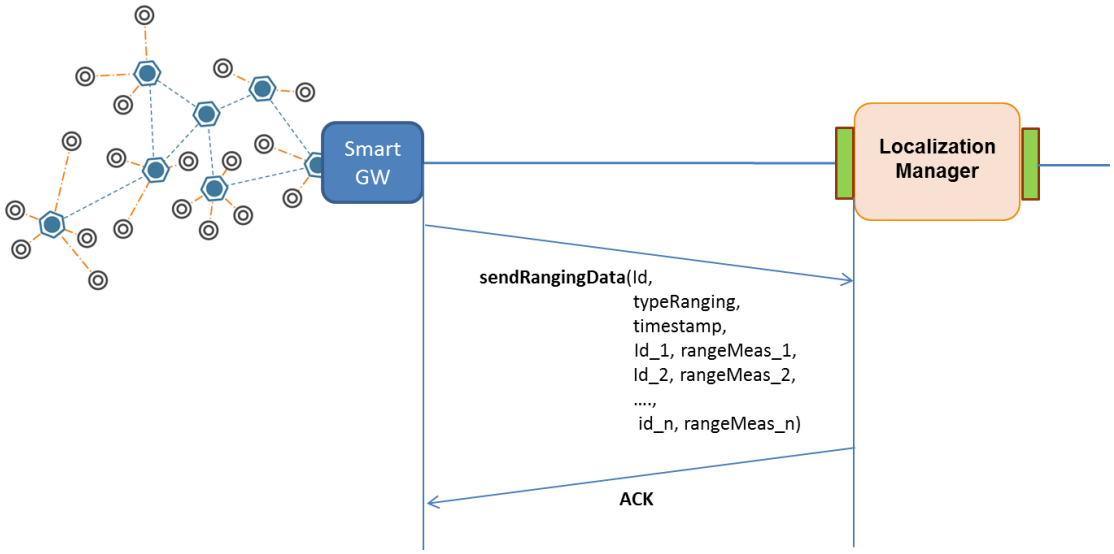

Figure 4–12: Ranging Data API description

In particular, the range data message that goes through the device access API contains the following fields: the Smart Object ID, type of ranging (e.g. RSSI, ToA or AoA) and an array of pairs where each pair consists of the neighbour object ID and the corresponding range measurement.

More details about the ranging API are listed in Table 4–14.

Argument Name	Measurement Unit	Data Type	Explanation
Id		Integer	Identifier of the Smart Object that performs range measurements with respect to its neighbours.
typeRanging		String	It identifies the type of ranging measurement (e.g. RSSI, ToA or AoA)
timestamp	seconds	Double	Time at which the 'range data packet' arrives at the smart GW expressed according the UTC time.
rangeMeas	RSSI [dBm] ToA [m] AoA [degree]	<Integer> <Double>	Data structure containing pairs where each pair consists of the neighbor object Id and the corresponding range measurement.

Table 4–14: SendRangingData API description

5. Smart Object Access and Control

5.1. Overview

The Smart Object Access and Control functional group includes a large number of the SmartObject Gateway functionalities:

- It handles the communication with the Consumer Protocol Adapter (REST API, JSON RPC, etc.) and IoT (and non-IoT) devices, providing URI mapping, incoming data/messages translation in an internal format and outgoing data/messages translation in Consumer format.
- Proxy service (caching) for battery operated device
- It manages the subscription/notification phases towards the Consumer
- It supports Devices and Resource Discovery and Resource Management capabilities, to keep track of IoT Resource descriptions that reflect those resources that are reachable via the SmartObject Gateway. These can be both IoT Resources, or resources hosted by legacy devices that are exposed as abstracted IoT Resources. Moreover, resources can be hosted on the SmartObject Gateway itself. The Resource Management functionality makes it possible to publish resources in the SmartObject Gateway, and also for the Consumer to discover what resources are actually available from the SmartObject Gateway.
- It also provides the security services needed to enforce the BUTLER security policy.

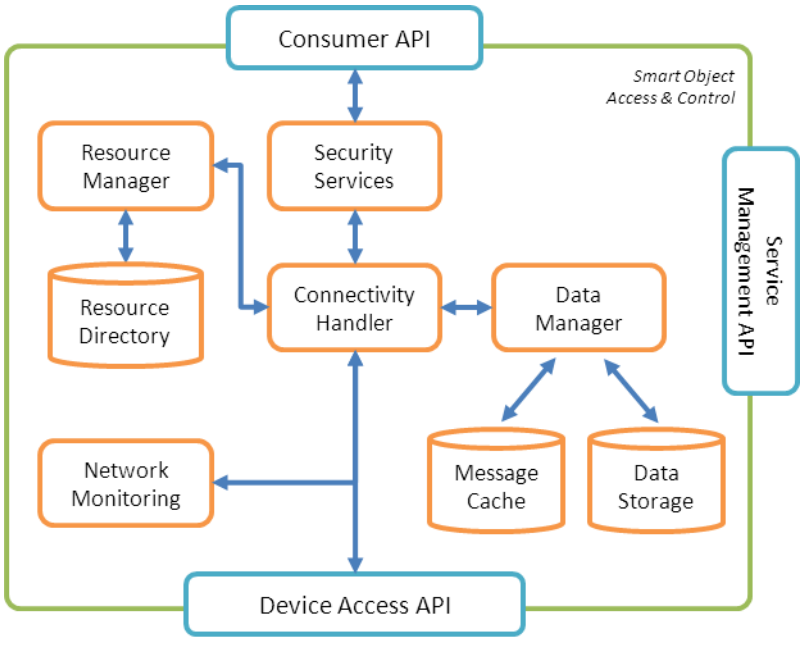

Figure 5–1: Smart Object Access and Control functional group

The Smart Object Access and Control functional group, depicted in Figure 5–1, is made up of the following functional components:

- Resource Manager
- Resource Directory
- Connectivity Handler
- Data Manager
- Data Storage
- Message Cache
- Security Services
- Network Monitoring

In the following sections, each of these functional components is described in more details.

5.2. Resource Manager

The main functions of this component are:

- Handle information about services and resources hosted by connected devices
- Handle information about services and resources hosted on the GW.
- Publish resource descriptors for external usage.

5.2.1. BUTLER Service and Resource Model

BUTLER Service and Resource model allows exposing the resources provided by an individual service. The latter, characterized by a service identifier, represents a concrete physical device (e.g., a temperature sensor, a TV, a motion detector, a parking space detector) or a logical entity not directly bound to any device (e.g. a weather service, parking space service, etc.). Each BUTLER service exposes resources and could use resources provided by other services.
Figure 5–2 depicts the BUTLER Service and Resource model (further details can be found in section 3.4.2.3 – BUTLER Service and Resource Model of [D3.2]):

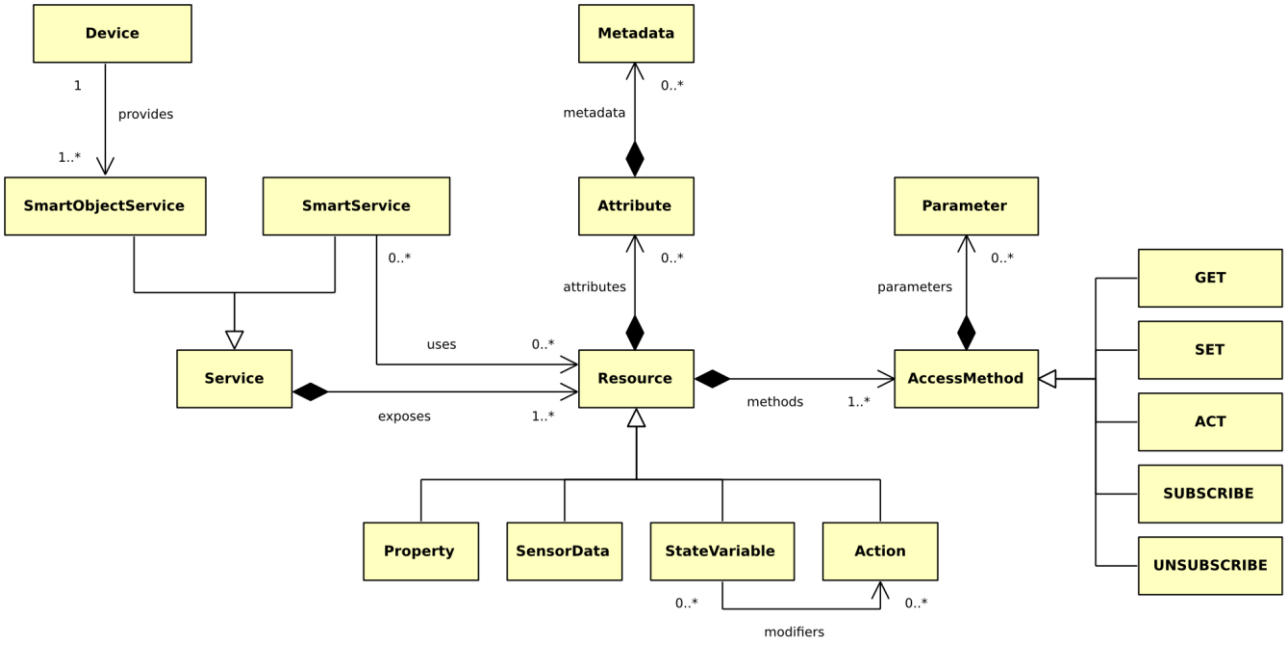

Figure 5–2: Service and Resource Model

5.2.2. Discovery process

Being able to identify, discover and manage IoT resources is not an add-on but a basic need. However, the devices linked to these resources that have to be connected are per se very heterogeneous in terms of used technology, protocols, capabilities and interaction patterns. Currently, each technology (i.e. ZigBee, 6LoWPAN, etc.) provides some of these functions in different ways. BUTLER integrates heterogeneous identification, naming and addressing technologies using unequivocal identifiers. Scalable discovery and look-up makes IoT resources available to all type of applications considering important real-world aspects like

location, time, availability, capabilities, quality, etc. It also provides the necessary functionality to monitor dynamic links between IoT resources and things. Finally, BUTLER enables IoT resources to become citizens of the Internet by providing scalable global schemes for deployment, operation, maintenance and fully remote management of these resources, including abstract models for the resources, common resource management interfaces, status monitoring, and fault handling.

Figure 5–3 describes the device discovery process (please refer to section 3.5.1.3 – Device Discovery of [D3.2] for more information) and the creation of the corresponding smart object services and their registration into the Service/Resource repository.

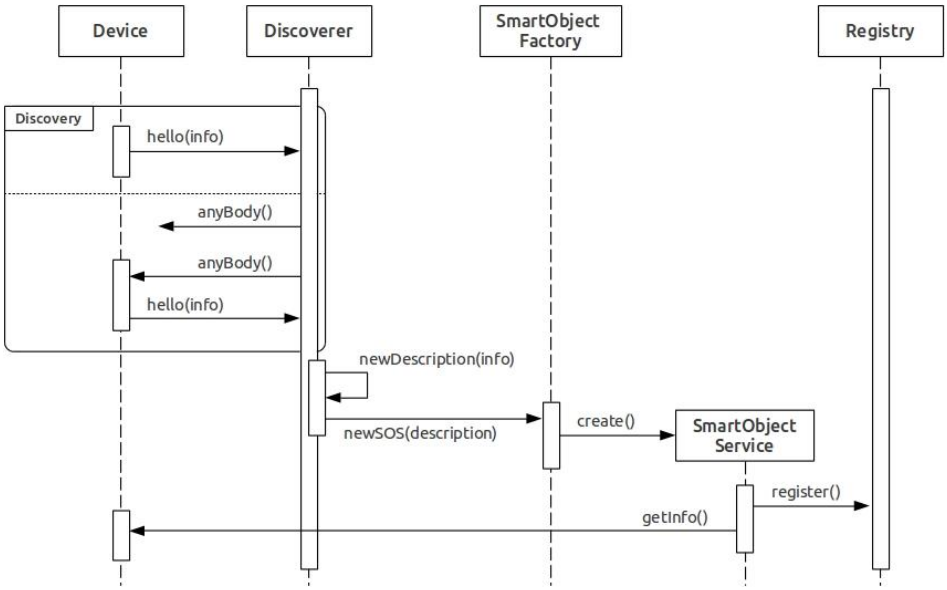

Figure 5–3: Device Discovery

5.3. Resource Directory

The Resource Directory allows storing information, i.e. resource descriptions, about the resources provided by individual devices connected to the SmartObject Gateway. Resource descriptions typically contain data such as:

- Info about endpoints that host the resources (address, port, etc.)
- Type of resource (i.e. temperature)
- Contextual data (i.e. position)

The Resource Directory supports looking up resource descriptions, as well as publishing, updating and removing resource descriptions to it (SQL interface).

Discovering and using resources exposed by BUTLER Services is a favoured approach for avoiding using static service interfaces and then increase interoperability. Therefore, BUTLER Services and their exposed resources are registered into Service/Resource repositories. The BUTLER SmartObject Gateway uses the OSGi service registry as Service/Resource repository, where resources are registered as service properties (see Figure 5–4). Clients ask the Service/Resource repository for resources fulfilling a set of specified properties (defined by LDAP filters). In response, the Service/Resource repository sends clients the list of service references that expose the requested resources and for which clients have authorizations (processed by the authorization server). Clients can then access/manipulate the resources exposed by their selected service objects.

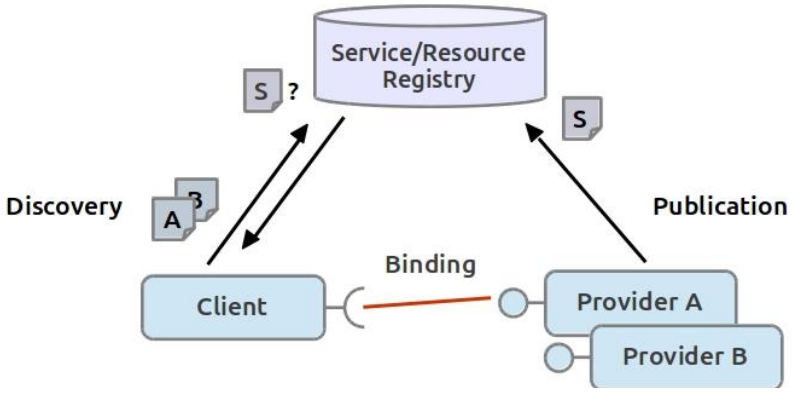

Figure 5–4: Service and resource registration

Registered resources and services are available to local clients and accessed using Java method invocations. Moreover, resources and services can be exposed for remote discovery and access using different communication protocols, such as HTTP REST, JSON-RPC, etc., and advanced features may also be supported (as semantic-based lookup).

5.4. Connectivity Handler

The main function of Connectivity Handler (CH) is to manage the connectivity between the resource hosting device and the resource requestor. Please refer to Figure 5–1 to see the interactions among CH and the other functional components.

Whenever a Consumer tries to access a resource via Consumer API, it forwards the requested URI, received from Consumer API, to the Resource Manager in order to check if a specific resource descriptor exists or not inside the Resource Directory and eventually to verify the related connectivity status (online | offline). If resource descriptor doesn't exist, the CH will arrange a message response with error code for the Consumer API. Otherwise (the resource is managed by the SmartObject Gateway), the CH will forward resource request to the Data Manager, regardless of the just retrieved resource connectivity status. Then, the Data Manager will check if a fresh resource representation is available inside Data Cache. If yes, it will be used to satisfy the request without contacting the origin server on the device, thereby improving efficiency. On the contrary, if a fresh representation of the requested resource cannot be found inside Data Cache and resource connectivity status is off-line, the Data Manager will take care of storing message into a Message Cache. The message is forwarded as soon as endpoint (the resource hosting device) is reachable. On the other hand, if communication endpoint is available, the Connectivity Handler directly forwards the information to the right interface..

The message request will reach the Device Access API if and only if Data Cache has not a fresh representation of the requested resource and the hosting Device is available.

At the same time whenever response is originated form IoT device (or abstract IoT device), it will be also forwarded to the Data Cache (by the mean of Data Manager) in order to create a resource representation inside the cache, or to update it, if it already exists.

5.5. Data Manager

The Data Manager functional component provides support for hosting devices that for various reasons cannot be continuously online. The main reason is that some devices work under resource constraints such as being battery operated. This typically requires asynchronous communication with the IoT resources and for this purpose, local storage is required. To this aim, the Message Cache and Data Storage components provide two support functions. The first one stores cached IoT resource requests from the consumer for devices that are temporarily offline. The second one locally stores (caches) data from IoT resources for subsequent retrieval by consumer applications. The goal of caching is to reuse a prior response message to satisfy a current request. In some cases, a stored response can be reused without the need for a new network request, reducing latency and network round-trips; a "freshness" mechanism is needed for this purpose (see 5.5.1).
In particular, for a given request, a stored response shall not be used unless:

- the request method and that used to obtain the stored response match,
- all options match between those in the request and those of the request used to obtain the stored response
- the stored response is either fresh or successfully validated as defined below.

The set of request options that is used for matching the cache entry is also collectively referred to as the "Cache-Key".

5.5.1. Freshness Model

When a response is "fresh" in the cache, it can be used to satisfy subsequent requests without contacting the origin server, thereby improving efficiency.
The mechanism for determining freshness is for an origin server to provide an explicit validity time in the response message. The latter is not fresh after the validity time interval has elapsed.
If an origin server wants to prevent caching, it MUST assign a null explicit validity time.
Generally, origin servers will assign non-null validity times when the representation is not likely to change in a significant way within the specified time period.
Default validity time is 60 seconds.

5.6. Data Storage

The Data Storage functional component implements a local cache for cacheable resource representations provided by servers (real and/or abstracted) in response to consumer requests. Each entry inside Data Storage is associated to a single resource representation and includes all the necessary information to understand if it can be used as fresh data. The following information, along with the data to be cached, are maintained in the Data Storage component:

- the requesting method;
- the expiration time, defined as: *expiration_time = last_served_time + max_age*

  where *last_served_time* is the timestamp of the last time the response has been served or validated, and *max_age* is the value of the maximum validity time included into response message. If not specified, default value is 60 seconds.

5.7. Message Cache

It provides a local storage for request messages addressed to off-line resources. To avoid vulnerability request messages are removed whenever storage capability is over some threshold, starting from the oldest one.

This cache is organised as a relational database, where resource and consumer are the entities and request messages are the relationship between them. See diagram below for detail.

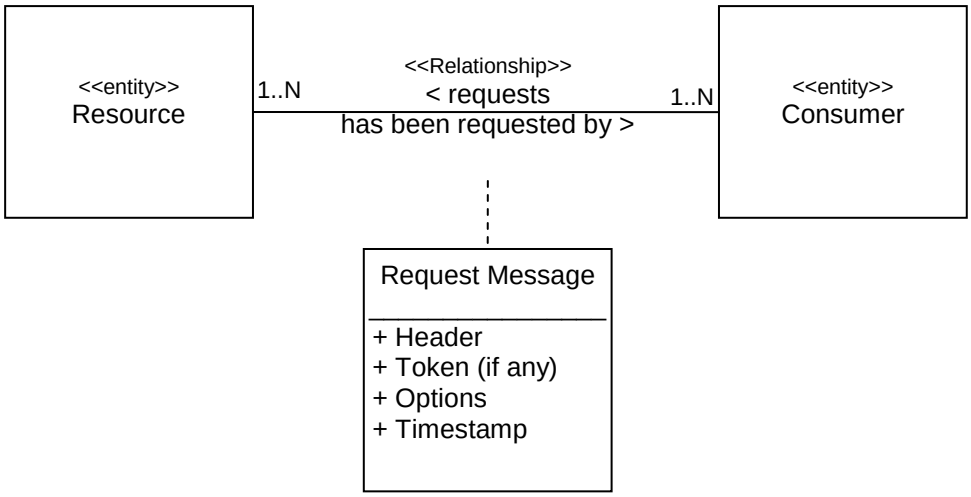

Figure 5–5: Theoretical data model for Message Cache

However, in order to not duplicate data inside Message Cache, the requested resource will be just referenced by mean of its unique identifier adopted by Resource Directory (RD). Moreover, a timestamp is also included in order to manage the cancellation of obsoleted request.

Whenever a resource becomes available it sends an update message toward Resource Directory in order to update its Connectivity Status to be on-line. This update fires a trigger that checks Message Cache for request messages involving the resource. If so, the Data Manager component interacts with the Connectivity Handler in order to build the response messages and delete request in the cache.

5.8. Security Services

BUTLER Security Services in the SmartObject Gateway enforce the security policy on the usage of resources managed by the gateway, providing the following functionalities:

- Consumers Authorization and Authentication
- Resources access control
- Data confidentiality and integrity

BUTLER Security Services implementation is directly linked to the Connectivity Handler. The overall Security Framework is described in D3.2 in chapter “BUTLER Security Services”.
This section describes the security services available on the SmartObject Gateway.

Table 5–1 lists the security roles defined for BUTLER.

Security Role	Description
User	User entity granting access to a resource. Generally, the user refers to a person, but can also be an application. The user shall be authorized to access the resource by the owner of the resource.
Resource Provider	Entity providing (and optionally updating) a resource. The Resource Provider shall check an *access-token* to provide/update the resource. Resource Metadata shall be registered in Authorization Server (AS)
Resource Consumer	Client application getting and consuming resource on behalf of a user. Such user must be authorized to access the resource.
Authorization Server	The Authorization Server plays the role of Resource Directory. It implements access controls management. The Authorization Server authenticates the user and authorizes resource-consumer getting resource by issuing a resource related *access-token.* Optionally, it may delegate the user authentication to an External Authentication Server
(optional) Authentication Server	*This optional role can be used by Authorization Server to rely on authentication protocol not natively implemented in the Authorization Server. It means that the Authorization Server and Authentication Server shall federate some user identities.*

Table 5–1: BUTLER Security Roles

5.8.1. SmartObject Gateway Security services requirements

In the BUTLER security framework, the Authorization Server is the BUTLER security enabler which is the single point for security management. All actors shall delegate authorization management and user management to Authorization Server.
Using these implementation principles, the Resource Provider (in this case the SmartObject Gateway) does not encompass any sensitive information about calling applications and does not encompass any user information. These principles are very useful to allow new application requesting a resource without managing such application at the gateway and to allow unknown user (from the gateway perspective) accessing the resource – for instance a friend of the resource owner.

At the SmartObject Gateway side, the BUTLER security protocol allows secure authentication of the application requesting a resource and implements end-to-end security between the calling application (the resource consumer) and the SmartObject Gateway (the resource provider). The security protocol does not transport any user information.

5.8.2. Resource Registration

A Resource is an Entity that can be addressed by the external world. With regard to the authorization process and, specifically for the Authorization Server, the Resource address is described using a URL schema. Each URL will be used by the Application (Resource Consumer) to reference the resource and perform allowed actions on it.

At the Authorization Server, the resource is described using the following elements.

Name	Resource name.
Owner	Owner of the resource.
Semantic	Semantic description: for instance: temperature,
Resource URL	The URL of the resource for instance http://mychalet.com/room1 (shall be unique). The resource URL uniquely identifies the resource.
Available actions: GET/SET/…	The actions a resource “consumer” can perform on the resource.
Key Material	The key material required to build the access-token for end-to-end security
Authentication Credentials	Credential used by the resource to access security material associated to an access-token.

Table 5–2: Resource Description at the Authorization Server

5.8.3. Resource Key Management

The main issue in security management is the setting of the initial security material. It is not a real problem when the objects are operated by well know entities like Mobile Operator and Banks. Such operators own the object (for instance the SIM card) which supports security features. In the BUTLER context we assume the object is owned by the user. The user will have to setup the security material.

User defined shared key schema

The security framework allows the user setting the shared key of its objects. In this case, the SmartObject Gateway shall propose a local interface to setup the security material. Another solution is to provide the user with the security material which can be generated dynamically by the SmartObject Gateway and provided using user interface. In the following example, the user sets the security material.

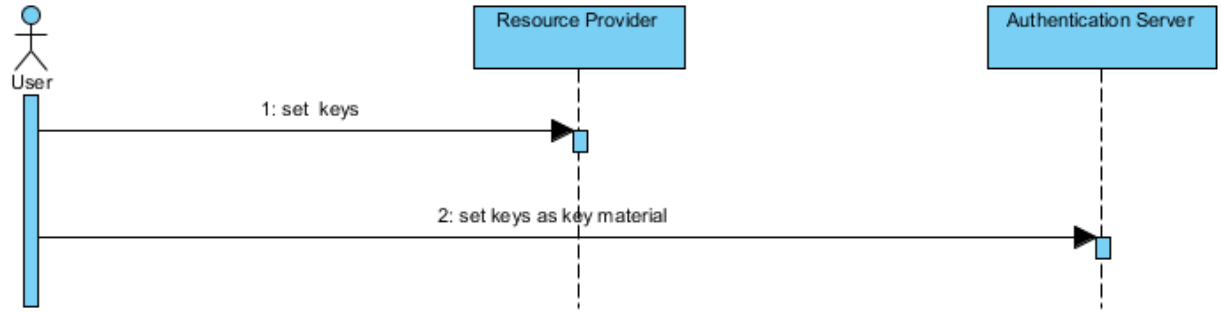

Figure 5–6: Registration of key material (set by the owner)

End-to-end Security

The end-to-end security involves both the Resource Consumer and Resource Provider. The end-to-end security is implemented at application level. The overall security protocol is defined in [D3.2] in chapter concerning Security Services.
The application shall retrieve token and security material to the Authorization Server.
The following schema presents the overall message exchange to retrieve an access-token (and security material) on behalf of a user:

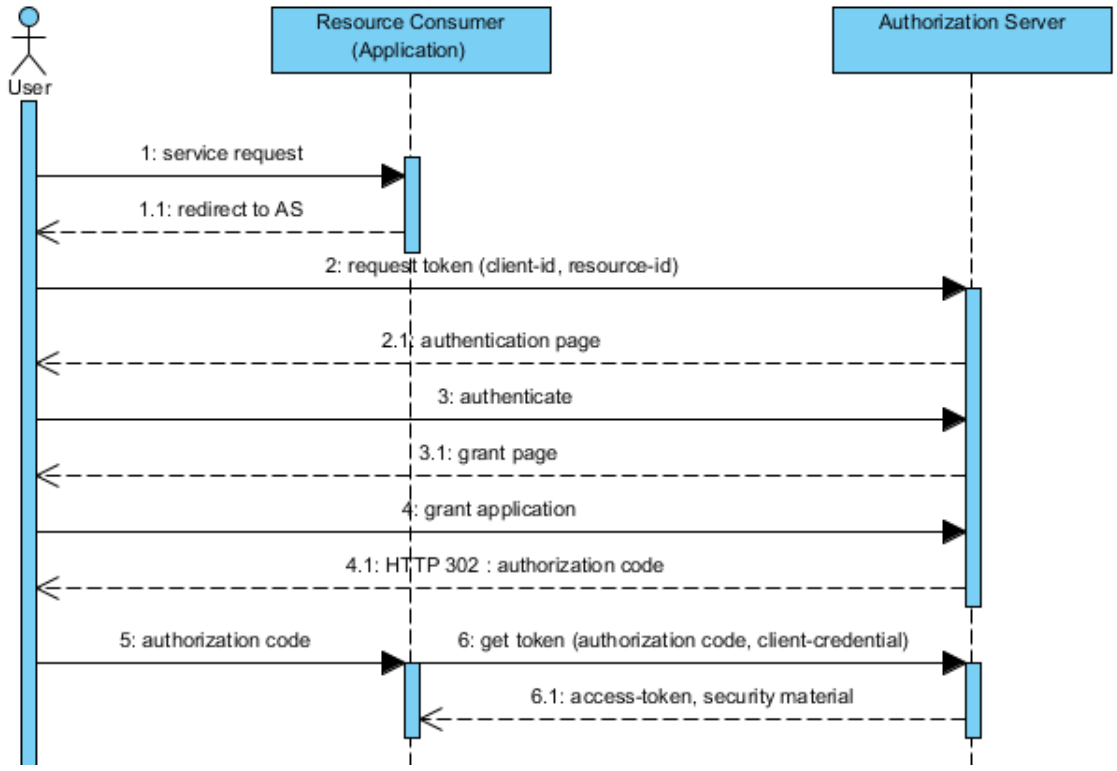

Figure 5–7: Retrieval of Access Token sequence diagram

The application shall first provide its identification; next, user shall authenticate and select required resources. Once authenticated, the application is provided with an authorization code to get the access-token; next, the application authenticates to the Authorization Server to retrieve the access token and security material.

The Security Material has been randomly generated by the Authorization Server. It consists of:

- Authentication-key
- Request-encryption-key
- Response-encryption-key

Accessing the resource

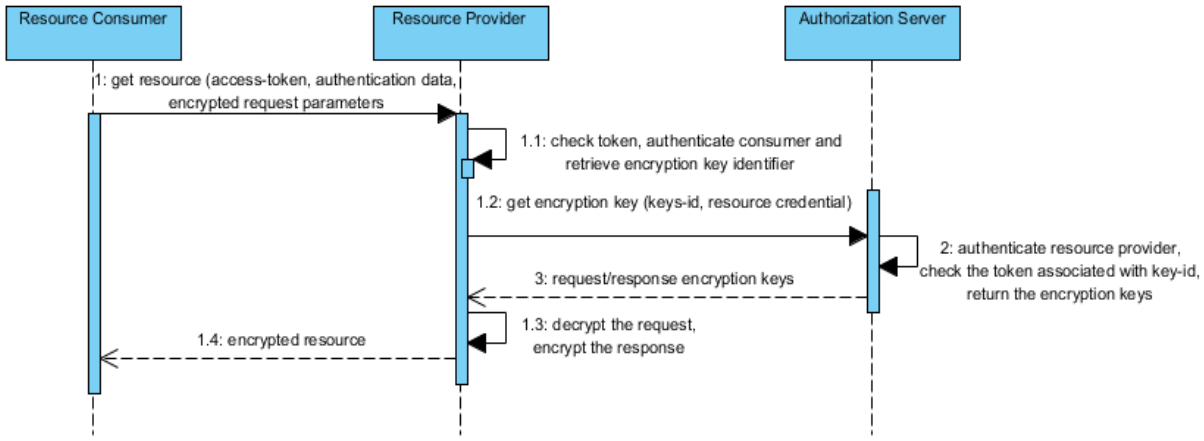

Figure 5–8: End to end security sequence diagram

To access the resource, the client side l builds the *authentication-data* and encrypts the *request-payload.* The SmartObject Gateway checks the *authentication-data,* decrypts the *request-payload* and encrypts the *response-payload.*

5.8.4. Security Application Protocol – example of the HTTP Binding

The client shall send an HTTP request using POST method. The SmartObject Gateway responds with the JSON data. The protocol is simple and it is described here through an example (presented in Table 5–3 and Table 5–4).

```
// Resource Consumer request ……….
POST /ResourceProvider/temperature HTTP/1.1
User-Agent: curl/7.32.0-DEV
Host: localhost:8080
Accept: */*
Content-Encoding: base64
Authorization: M2MS
KaEsZJ80fkHuF7mGpbQm6yIq/Gz/ZhxDnmI/u7IOTZ7d0QtuwkZd8tyDWI945p4BY2GYysr7dn2ykpZBZbb9568SWjC
ljuK8XT2kHUV2UvWXborhWJ3B9pDbV/7uWZMHCXAcgWzFKrE4cf+Xsm8frtQzQ6Mq6Y9T8IGET/G0T1o=:hGXenLjCY
lBtQCjQ9P5rdBnvxKugZdz6Tf6CZCSQdhxeiDlCJ+Wt8pYsTxQKbrS8
Content-Length: 44
Content-Type: application/x-www-form-urlencoded

H9ioezdEmBVuofP76FoA+p1O3Eqx2vpcHyOC/yuRj5k=

//  Resource Provider response ……..
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/m2ms application/json
Content-Length: 126
Date: Thu, 03 Oct 2013 09:18:31 GMT
{"request-identifier": "123456", "m2ms-application-
data":"ufZ+6YysbY39ltE0GsFjGKeWLddAJQa2DWGDWuJUZaFe+gI6A23zsA8Vedw/qFKO" }
```

Table 5–3: HTTP M2MS Protocol

Request Data	HEADER.Authorization.ProtocolName: M2MS
	HEADER.Authorization.Access-token: KaEsZJ80fkHuF7mGpbQm6yIq/Gz/ZhxDnmI/u7IOTZ7d0QtuwkZd8tyDWI945p4BY2GYysr7dn2ykpZBZbb9568SWjCljuK8XT2kHUV2UvWXborhWJ3B9pDbV/7uWZMHCXAcgWzFKrE4cf+Xsm8frtQzQ6Mq6Y9T8IGET/G0T1o=
	HEADER.Authorization.Request-authentication-data hGXenLjCYlBtQCjQ9P5rdBnvxKugZdz6Tf6CZCSQdhxeiDlCJ+Wt8pYsTxQKbrS8
	Body: Encrypted-payload H9ioezdEmBVuofP76FoA+p1O3Eqx2vpcHyOC/yuRj5k=
Response Data	Request-identifier: 123456 retrieved from Request-Authentication-Data
	M2ms-application-data: ufZ+6YysbY39ltE0GsFjGKeWLddAJQa2DWGDWuJUZaFe+gI6A23zsA8Vedw/qFKO

Table 5–4: Example Request and Response Data

5.8.5. SmartObject Gateway Security Service API

In order to facilitate the integration work, the WP2 work package on Task 2.2 has developed a library that shall be integrated in the SmartObject Gateway's Consumer API Layer for security purpose. The API library provides the functionalities which shall be called by the Consumer API component using its received encrypted data argument to retrieve user data and potential methods' parameters (subscription consistency management – cf. 5.10. Section).

Securing an API – HTTP Binding example

To use the M2M Gemalto security technology, the SmartObject Gateway application developer has to define an extended javax.servlet.Filter implementation class which calls the Consumer API component by supplying it the encoded request body, the resource security material according to the last field of the requested URI, and the authentication server's URL *as d*efined in the web container configuration file (web.xml). The M2M Gemalto security library, called by the Consumer API component with all those parameters, checks the *access-token* and the *authentication-data* and decrypts the *request* body. The decrypted data are then used to retrieve wanted information, and/or to actuate a targeted actuator. On completion, the library is used to encrypt the resource response.

The SmartObject Gateway administrator simply configures the file WEB-INF/web.xml and setups resource security material in configuration file. The administrator shall configure the application container to enable Security Filtering. He/she has to update the configuration file /WEB-INF/web.xml (an example is provided in Table 5–5).

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schema Location="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
   version="2.4">

<!-- FILTER -->
<filter>
   <filter-name>SecurityFilter</filter-name>
   <filter-class>gemalto.security.example.MyFilter</filter-class>
    <init-param>
      <param-name>authServerURL</param-name>
      <param-value>http://trustmanager.gemalto.iot-butler.eu/api/session_keys/
             </param-value>
    </init-param>
    <init-param>
       <param-name>proxyHost</param-name>
       <param-value> the proxy of your organization (or empty) </param-value>
     </init-param>
     <init-param>
       <param-name>proxyPort</param-name>
       <param-value> the port to be used (or empty)</param-value>
     </init-param>
     <init-param>
       <param-name>logger</param-name>
       <!-- possible value:  "container", "console", "none"
            default = container -->
       <param-value>container</param-value>
     </init-param>
</filter>

<!-- filtering temperature servlet -->
<filter-mapping>
   <filter-name>SecurityFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>

</web-app>
```

Table 5–5: Example for /WEB-INF/web.xml

Configuring the Resource Security Material

The Security Filter retrieves the resource security material from a resource provider configuration file. The security materials are registered in the file /WEB-INF/resourcedb/<resourcename>

Where <resourcename> := last field of the request URI.

Example:

http://mychalet.com/ResourceProvider/temperature

The Resource Provider shall define the /WEB-INF/resourcedb/temperature file as shown in Table 5–6.

```
secret:<secret>
authcode:<authentication code>
```

Table 5–6: Security Material Configuration file for "temperature"

<secret> and <authcode> shall be equal to the values registered in the Authorization Server when registering the resource. On the Authorization Server, the resource owner shall configure the resource including the above *secret* and the *authcode.* This way, the Authorization Server and the SmartObject Gateway share the same security credentials, see Resource Registration in 5.8.2.

5.9. Network Monitoring

The Network Monitoring (NM) module provides availability, utilization and overall performance of underlying M2M local area networks (MANs). In particular, NM aims to provide aggregated network status statistics per smart object basis. Thus, in real time fashion, NM intercepts and analyses network traffic in order to estimate real time statistics. In turn, these statistics could be used to ensure that user service level objectives are met while the network resources are utilized in a cost-efficient way. NM exposes these statistics to the Smart Server through BUTLER consumer APIs. NM is crucial for real time statistics about the network where indeed will help network administrator meet the network service level agreement and have better planning for your network future improvement. Moreover, NM acts as a proxy between the SmartObject Gateway manager and its bridges.

5.9.1. Classification of Network Monitoring Techniques

In this section, NM approaches are classified into two different types. We first look into the distinction between Smart Object Device and SmartObject Gateway oriented monitoring. Then, we classify NM approaches based on whether they use active monitoring or passive monitoring strategies.

Smart Object Device and Smart Object Gateway Oriented Monitoring

The underlying Smart Object Devices network consists mainly of the SmartObject Gateway plus one or more Smart Object Devices. The Smart Object Devices are spread into groups. Objects in the same group share MAN resources i.e. a group of objects 6LowPAN, ZigBee to cite a few. Each group is connected to the gateway by its own bridge. This bridge is part and running in the gateway. The gateway facilitates connecting the Smart Objects to the larger BUTLER network or the internet. This hierarchical grouping and interconnection of Smart Object Devices easily classify the objects network traffic into two types. The first type is the traffic between two Smart Object Devices in the same or different groups and it is named *local traffic*. The second type is the traffic between the SmartObject Gateway and a Smart Object Device and it is named *global traffic*.

An important issue that emerges when considering Network Monitoring is related to the traffic monitoring coverage. We consider two main alternatives:

1) *Smart Object Device NWK Monitoring* is appropriate for maintainers that require a fine-grained statistics status of the network links among Smart Object Devices themselves and the Smart Object Devices and Gateway.

2) *SmartObject Gateway NWK Monitoring* provides a view of network links between the gateway and the Smart Object Devices only, this approach lowers the computation overhead on the Smart Object Devices since most of Network Monitoring functionalities are hosted and run within the gateway.

The two approaches are further detailed as follows:

- *Smart Object Device NWK Monitoring*: the network traffic sourced or sunk by a Smart Object can be monitored. So, more detailed statistics are generated. Hence, a better availability, utilization and overall performance are presented by NM. However, there are some issues to be considered with the Smart Object approach. First, each Smart Object is often a black box containing proprietary software; there may be no way to

add more functional logic for monitoring purposes, or even the addition of new logic could break one or more capacity constraints of the Smart Object. Second, deriving a time correlated performance metric within the Smart Object requires critical step: to reconstruct time synchronization throughout the network. In order to sort out these issues, either dedicated Smart Object devices can be used that implement NM features or introduce specific NM functionalities in the devices.

- *SmartObject Gateway NWK Monitoring*: Only global traffic that is sourced or sunk by the SmartObject Gateway can be monitored. So, the coverage of status statistic is less than the previous case. NM functionality is required only in the gateway. A few modification or configuration is needed for the Smart Objects. No time synchronization throughout the network is required.

Passive versus Active Monitoring

Another classification scheme, which is often used when dealing with Network Monitoring, distinguishes between *active* and *passive* monitoring techniques. For this work, we adopt the following classification criterion: a monitoring tool is classified as active if it induces traffic into the network; otherwise it is classified as passive. Passive monitoring is more appropriate for monitoring gross connectivity metrics like link throughput; it is also needed for accounting purposes. Passive Network Monitoring techniques analyse network traffic by capturing and examining individual packets passing through the monitored link, allowing for fine-grained operations, such as deep packet inspection. The main benefit of passive monitoring approaches, compared to active monitoring, is its non-intrusive nature. Active Network Monitoring techniques incur an unavoidable network overhead due to the injected probe packets, which compete with user traffic. In contrast, passive Network Monitoring techniques passively observe the current traffic of the monitored link, without introducing any network overhead.

- Active monitoring is more effective for observing the network sanity and is suitable for application oriented observations, such as jitter, when related to multimedia applications. On the other side, this approach implies an unavoidable network overhead due to the injected probe packets which compete with user traffic. To incorporate active Network Monitoring in BUTLER there are issues to be considered.
    - NM functionality is spread on both the SmartObject Gateway and Smart Object side.
    - Specific dedicated messages are used to update link information.
        - **BUTLER Ping message**. In order to update NWK statistics, the smart GW can send to a specific smart object an echo request message and waits to the corresponding echo reply message. More details are provided as follows.
            - **Echo request message:** it is sent by the SmartObject Gateway to a specific Smart Object. It contains the following fields: Request sequence number and timestamp. The first field is incremented by one each time the gateway sends an Echo Request. The timestamp field holds an instance of current system time at which the gateway generates this request. This field remains unmodified in the Smart Object side.
            - **Echo reply message:** it is sent from Smart Objects back to the SmartObject Gateway. It contains the following fields: Reply sequence number and timestamp. The first field. is copied as it is from the sequence number of the corresponding Echo request

message. The timestamp field is alos copied as it is from the corresponding Echo request message. Node Id is the Smart Object IPv6 address.

- **BUTLER Traceroute message**:
  - This message, which is sent by the smart object GW to a specific smart object device, is used to obtain the list of nodes used as multi-hop along the path between the GW and the specific device.

- The application data packet structure is modified by adding a Seq. no field. This field is used to uniquely identify the current message among other messages corresponding to the same Smart Object. The SmartObject Gateway inserts this field in the message and the Smart Object copies it as it is in the corresponding reply.

To sum up, passive monitoring provides network's performance with a non-intrusive approach while the active approach provides a more reactive response. Thus, as a general rule, an effective Network Monitoring should exploit both techniques as presented in the following.

5.9.2. BUTLER Network Monitoring Strategy

BUTLER Smart Object Devices can be based on different heterogeneous technologies and applications. This requires that the SmartObject Gateway should be able to handle all attached heterogeneous MAN, such as 6LoWPAN, ZigBee, RFID, etc. Due to aforementioned heterogeneity, any supposed Active or Smart Object Device oriented NM approach requires to adapt new logic to handle the NM functionalities.

For BUTLER, a hybrid passive-active and SmartObject Gateway oriented procedures are combined to form BUTLER Network Monitoring. This way imposes a specific requirement and configurations within the Smart Object devices and the SmartObject Gateway. And handling most of the NM functionality is delegated to the SmartObject Gateway.

5.9.3. Network Monitoring Design Guidelines

Network statistics strongly depend on the protocol stack layer (in the OSI Module stack) where NM observes the data packet. For instance, if NM computes the latency as the propagation delay at the physical layer, then this delay is lower and more accurate in terms of precision than any observed delay at the higher protocol layers like MAC or network. Another good example, which emphasizes the effect of where the observation occurs, is when a data packet is lost at any lower layer and a second retransmission succeeded to deliver this data packet to its destination. In this case, all upper layers will never know that this packet has been lost. Eventually, this will be revealed as an increase in the delay values.

For BUTLER purposes, NM is attached at the application layer of OSI module. At this layer, data packets can be expressed as BUTLER messages. Large majority of these messages consist of a request from the SmartObject Gateway and a response from the Smart Object.

Another important issue to be considered is that BUTLER messages are further classified by the Connectivity Handler (CH) based on the availability of the addressed Smart Object Device. So, whenever Consumer tries to access a resource, the CH checks if it is connected.

If endpoint is unavailable (device that hosts Resource is sleeping or Consumer is disconnected), it forwards the message to the Data Handler component that takes care to store message into a Connectivity Cache. The message is forwarded as soon as endpoint is reachable. On the other hand, if communication endpoint is available the request CH directly forwards the information to the right interface. Due to this behaviour of CH, the delay related statistics firmly vary on the basis of the attachment point of NM with the reference CH.

To sum up, when NM is attached to the Northbound of Connectivity Handler, the measured delay value includes the sleep time of the intended smart object. While, if NM is connected Southbound of Connectivity Handler. Then the delay value does not include the sleep delay time.

According to the above mentioned considerations, our suggestion is to provide two design architectures and guidelines for NM.

1. *Centralized* module which provides the entire Network Monitoring functionalities, hereafter it is called CNM (Centralized Network Monitoring). CNM has the following properties and capabilities:
   - It is part of the main SmartObject Gateway components where no in-depth knowledge of any attached adaptor or Smart Objects is required.
   - It must receive, analyse and forward all BUTLER messages (request and response pair).
   - For every observed Smart Object ID it holds instances of the network statistics.
   - For each received message it updates the network statistics (more details are provided later).
   - Identifies, replies, and terminates any BUTLER message concerns gaining knowledge of the network status and monitoring (basically these messages are not any sensing or actuating messages).
2. *Distributed* architecture is called DNM (Distributed Network Monitoring). DNM consists of many modules. All modules are similar in capabilities and also in functionalities. A single DNM has the following properties and capabilities:
   - It is part of a bridge, now the bridge is also acting as a manager for this module.
   - For each attached Smart Object to the bridge, it analyses all BUTLER messages heading to the Smart Object and it observes all the network statistics (more details are provided later).
   - For each Smart Object ID it holds instances of the network statistics.
   - Identify and reply any BUTLER message concerns on gaining knowledge of the network status and monitoring (basically these messages are not any sensing or actuating messages).
   - All the statistics are treated similarly as if they were sensing variable. For instance, the network latency becomes similar to the Smart Object temperature value.

5.9.4. BUTLER Network Monitoring Architecture

Overall NM accomplishes its aims through three main functionalities reported as follows:

1. Passive Part: traffic sniffing is performed by capturing BUTLER messages while traversing the internal data bus (or just configuring the Connectivity Handler to send both parts, request and response, of all BUTLER messages to NM) within the SmartObject Gateway. Then, it processes these messages to figure out the useful information such as command direction (is it request or response?), the intended Smart Object Device ID by BUTLER message, target URI, system timestamp (this is the current SmartObject Gateway time), the total message request size in bytes, matching response to the corresponding request and then do calculation after updating the internal data structure.
2. Active Part: acts as proxy with the Bridges. NM forwards BUTLER Ping and Traceroute messages to its intended Bridge within the SmartObject Gateway. Then, it forwards back the response.
3. Production stage which compromises interacting with other components in SmartObject Gateway to facilitate exposing network statistics through BUTLER external API.

5.9.5. Network Monitoring Statistics

MN provides the following output statistics and messages.

1. The link utilization is expressed by the following variables:
   - **Number of transmitted messages** (nTx) is the total number of BUTLER messages per Smart Object. Note that, nTx with a given time interval allows to compute nTx per time unit like (nTx/minute) or (nTx/Hour) etc. For instance, throughput can be computed as the number of messages successfully delivered per unit time.
   - **Number of application bytes** (nBytes) is the total number of BUTLER message bytes per Smart Object. A single value is:
     nBytes = number of bytes of message header + number of bytes of message payload.

2. The link quality is expressed by the following variables:
   - **Latency** is a measure of Round Trip Time (RTT) expressed in milliseconds that is experienced by successfully performing a BUTLER message and it is a good measure to check the response time from the Smart Object against a request. Actually, Network Monitoring always exposes the average of all measured latency values. In communications, the lower limit of latency is determined by the medium being used for communications. In reliable two-way communication systems, latency limits the maximum rate that information can be transmitted, as there is often a limit on the amount of information that is "in-flight" at any one moment. This delay includes all kind of other associated delays, i.e. for ZigBee protocol the delay between the SmartObject Gateway and the Coordinator and the delay between the Coordinator and the Smart Object.
   - **Packet Loss ratio** is an indicator of message loss that occurs when a BUTLER message fails. The following formula is used to calculate this ratio:
     cmdLossRatio = Number of lost messages / (Number of lost messages + nTx).

3. SmartObject Gateway Manager messages:
   a. **BUTLER Ping**: is a BUTLER message similar to well-known Ping computer network administration utility used. BUTLER Ping is used to test the reachability of a Smart

Object device and to measure the round-trip time for messages sent from the originating SmartObject Gateway to a destination Smart Object device.

b. **BUTLER Traceroute**: is a BUTLER message similar to well-known traceroute. It is used for displaying the route (path) from the SmartObject Gateway to a specific Smart Object.

All variables are attached with a timestamp corresponding to the time at which the variable value was modified.

5.10. Consumer API

The Consumer API is a generic java based API that allows exporting information from the SmartObject Gateway and its associated home area network objects. Its services are used by the Protocol Adapters. The consumer API allows interaction between the upper layer protocols (as depicted in Figure 5–1) and the security checking management, which controls access to resources by consumer applications.

5.10.1. Consumer API Request and Response protocol

The <SECURITY-DATA> notation used below for method's signatures gathers authorization data, authentication data (cf. Figure 5-5: End to end security sequence diagram), the resource security material, and the authentication server's URL. If argument data potentially contains encrypted parameters the <PARAMETERIZED-SECURITY-DATA> notation is used and the specific parameters definition is specified.

- In the case of a direct access to the SmartObject Gateway, the consumer API provides two exploration methods:

Method	Signature
LIST	LIST(<SECURITY-DATA>) : <DATA>
RESOURCE	RESOURCE (<PARAMETERIZED-SECURITY-DATA>) : <DATA> *Parameter(s) := <RESOURCE-ID>*

Table 5–7: Exploration Methods

  - The *LIST method* returns the set of accessible resources for the requiring client according to the defined security policy. The resource description is developed below:
    - ✓ the identifier of the resource: its common name as it has been recorded in the authorization server;
    - ✓ its type, as defined in the resource model: StateVariable, SensorData, or Property (cf. Section 0);
    - ✓ a list of associated attributes and their meta-data;
    - ✓ a list of available methods as defined in the resource model: GET, SET, SUSCRIBE, UNSUSCRIBE, ACT. An important point is that the list of available methods depends on the security policy defined for both the client and the resource, and so can be different for two clients accessing the same resource.
  - The *RESOURCE method*, which takes the common name of a targeted resource as parameter and returns its description as defined above.

- In the case of a targeted known resource, the consumer API offers methods to directly access it using its endpoint. These methods are quite similar to the ones proposed by

the Device Access API (cf. 4.4. Section), excepted by the fact that they include an access token parameter in their signature.

Method	Signature
GET	GET (<PARAMETERIZED-SECURITY-DATA>) : <DATA> *Parameter(s) := [<ATTRIBUTE-NAME>,]**
SET	SET (<PARAMETERIZED-SECURITY-DATA>) : BOOLEAN *Parameter(s) := [<ATTRIBUTE-NAME>,]? <VALUE>*
SUBSCRIBE	SUBSCRIBE(<PARAMETERIZED-SECURITY-DATA>) : SUSCRIPTION-ID *Parameter(s) := <CONSUMER-SERVICE-ENDPOINT>*
UNSUBSCRIBE	UNSUBSCRIBE (<PARAMETERIZED-SECURITY-DATA>) : BOOLEAN *Parameter(s) := <SUSCRIPTION-ID>*
ACT	ACT (<PARAMETERIZED-SECURITY-DATA>) : <DATA> *Parameter(s) := [<PARAMETER>,]**

Table 5–8: Access Methods

- The *GET method*, by default (i.e. without parameter) returns the set of meta-data (i.e. <name, type, value> triplet) associated to the “VALUE” attribute of the resource. Otherwise, the result grows rich of an additional set of data for every name of attribute passed as parameter;
- The *SET method* allows by default defining the value meta-data of the “VALUE” attribute. If a name of attribute is passed as parameter the value meta-data of this last one is updated. If the new value has been set properly the method returns a TRUE boolean value, otherwise it returns a FALSE boolean value;
- The *SUBSCRIBE method* allows registering a change-event callback endpoint. It is important to notice that for security purpose the subscription stays active as long as the access-token of the requiring client is. If the callback endpoint registration has been done properly the method return a subscription identifier, a unique numeric value on the SmartObject Gateway, otherwise it returns a NULL value;
- The *UNSUBSCRIBE method* allows unregistering a previously made subscription using its identifier as parameter;
- The *ACT method* allows triggering of actuators and potentially returns a data related to actuation (dependent of the resource).

In addition to its intermediary role between communication bridges and security manager, the consumer API has to handle the subscription callbacks consistency through resources/subscriptions and access-tokens/endpoints mappings and validation using the security manager.

5.10.2. Security Management at Consumer API

The request and response shall follow the security protocol which is transported by the north-bridge protocol.

Message	Data
REQUEST	<CMD> | <PARAMETRIZED-SECURITY-DATA> or <CMD> | <SECURITY-DATA> with <SECURITY-DATA> := <access-token> | <request-authentication-data>
RESPONSE	<DATA> := <request-identifier> | <encrypted-response>

Table 5–9: Request and Response

The Consumer API uses the Internal Security API to handle the request and build the response. The Consumer API returns the response that shall be transported over the north-bridge protocol.

Internal Security API

The internal security consists of Java methods to check and decrypt input data and encrypt the output response as shown in Table 5–10.

```
public class com.gemalto.m2ms.security.SecurityProviderAPI {

// REQUEST
public static SecurityRequestResponseMaterial
      checkTokenAndRetrieveSecurityRequestResponseMaterial(
      String request_url,
      String access_token_base64,
      String request_authentication_data_base64,
      String resource_key_material)
      throws InvalidKeyException, NoSuchAlgorithmException,
             NoSuchPaddingException,
             IllegalBlockSizeException,
             InvalidAlgorithmParameterException,
             InvalidKeySpecException;

public static byte[] decryptRequestPayload(
      // from protocol
      byte[] encrypted_payload,
      // from SecurityRequestResponseMaterial.
      String request_encryption_key)
      throws InvalidKeyException, NoSuchAlgorithmException,
             NoSuchPaddingException,
             IllegalBlockSizeException,
             InvalidAlgorithmParameterException

// RESPONSE.
public static byte[]  encryptResponsePayload(
      // the response
      byte[] response,
      // from SecurityRequestResponseMaterial.
      String response_encryption_key)

      throws InvalidKeyException, NoSuchAlgorithmException,
             NoSuchPaddingException,
             IllegalBlockSizeException ;
}

public class com.gemalto.m2ms.security.SecurityRequestResponseMaterial {
   // an instance of this class is returned by the
   // SecurityProviderAPI.
   // checkTokenAndRetrieveSecurityRequestResponseMaterial()

   // caller shall set the connectivity parameter
   // to be able to retrieve the request-encryption-key and
   // the response-encryption-key from the authorization server.
    public void setAuthenticationCode (String code);

    public void setAuthServerURL (URL url);

    public String getRequestIdentifier();

    public String getRequestEncryptionKey() throws
           MalformedURLException,
           IOException,
           ProtocolException

    public String getResponseEncryptionKey() throws
           MalformedURLException,
            IOException,
           ProtocolException;

}
```

Table 5–10: Internal Security API

6. Gateway Management

6.1. Overview

The Gateway Management FG includes the necessary functionalities for management and maintenance of Smart Objects and services, such as configuration, software management, performance management, diagnostics, accounting and security.

The main functional components are the Gateway Manager and the Device Manager.

Figure 6–1 shows the Gateway Management FG architecture.

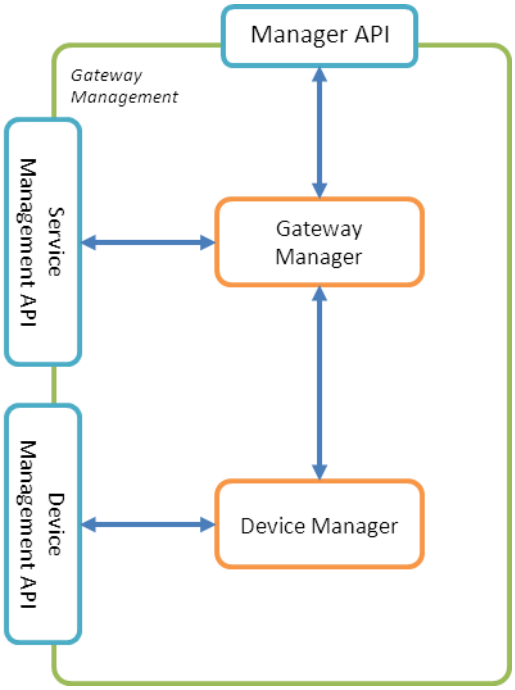

Figure 6–1: Gateway Management functional group

6.2. Device Manager

The Device Manager FG defines the Device Management API to interact with the Device Protocol Adapter FG.

6.2.1. Device Management Overview

BUTLER system includes Device Management functionality for large scale deployments as described in section 3.5.4 of [D3.2]. Management consists of configuration, monitoring and administration of entities such as network elements, system resources, applications or services in order to increase their efficiency.

Figure 6–2 illustrates the overall device management concept.

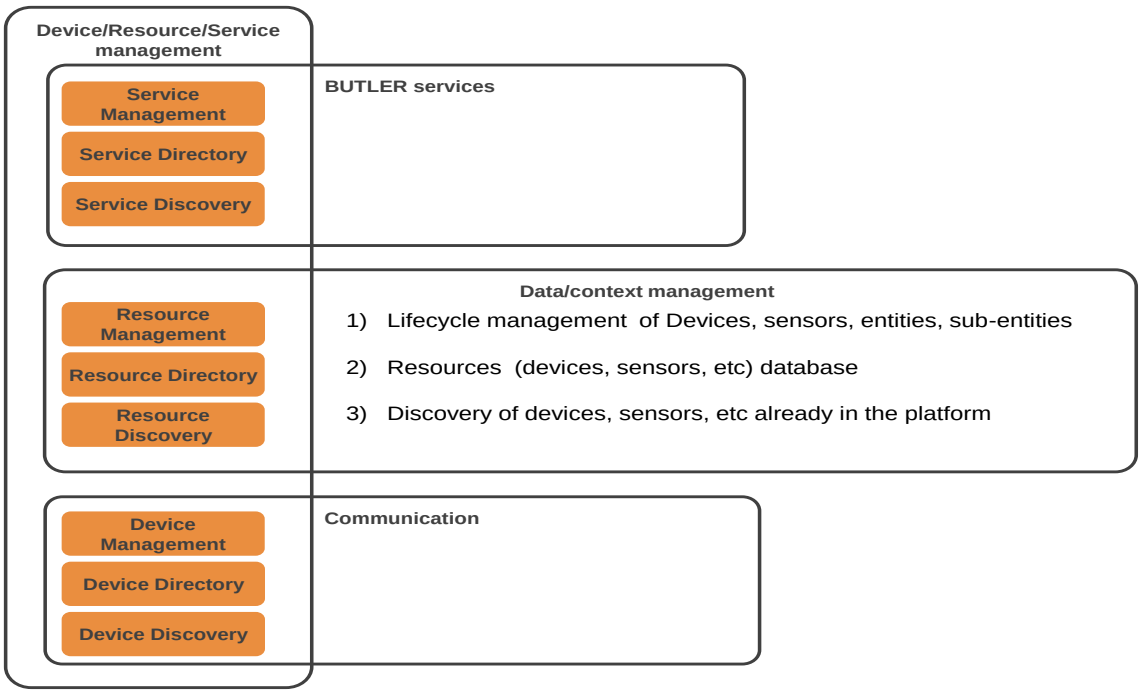

Figure 6–2: System/Device Management interface

6.2.2. Device Management API

The Device Management API allows the interaction with sensors and actuators for management purpose. Several administration parameters can be modified remotely on the device platforms. However, take care that after a reboot, the device node will come back to its initial configuration. The generic management operations, applicable to different Smart Objects are described below.

Method	Signature	Description
SET_PERIOD	Set_Period (String deviceID, long period)	All the sensors connected to a node are scanned with a period of time that can be changed remotely. This period define also the frequency the sensor events will be notified to the user if he subscribed to periodic notifications.
SET_LOCATION	SET_LOCATION (String deviceID, string location)	The registered location of the node can be changed remotely.
SLEEP	SLEEP (String deviceID, long Duration, UNIT unit)	Sleep operation asks the devices to enter its Sleep Mode to save some battery. It takes as input the sleep duration of the device. While sleeping, devices stop polling its sensors and can't receive or send data.

Table 6–1: Device Management API

6.3. Gateway Manager

The Gateway Manager FG defines the Service Management API to interact with the Smart Object Access and Control FG and the high level Manager API to interface with the Consumer Protocol Adapter FG.

6.3.1. Gateway Manager Overview

BUTLER SmartObject Gateway is based on the Open Services Gateway initiative (OSGi) [6-1] that simplifies management of the overall gateway system. OSGi allows representing sensor and actuator device capabilities in abstract terms. iPOJO [6-2] has been used to describe services components and to simplify OSGi services development. Service components represent SmartObjects' functional and management capabilities, while separating access to their interfaces.

Figure 6–3 and Figure 6–4 provide an example of an OSGi web-based console for managing the gateway services. It allows administration tasks and secured access to the BUTLER SmartObject Gateway environment. It also allows overseeing and configuring services and events at runtime. In addition, the OSGi remote management functionality can be used to detect and fix problems with the platform and to install or update new SmartObjects services.

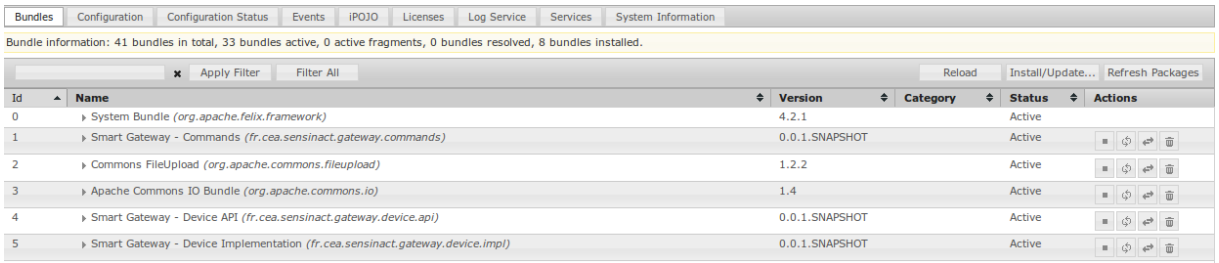

Figure 6–3: OSGi Web Console (view 1)

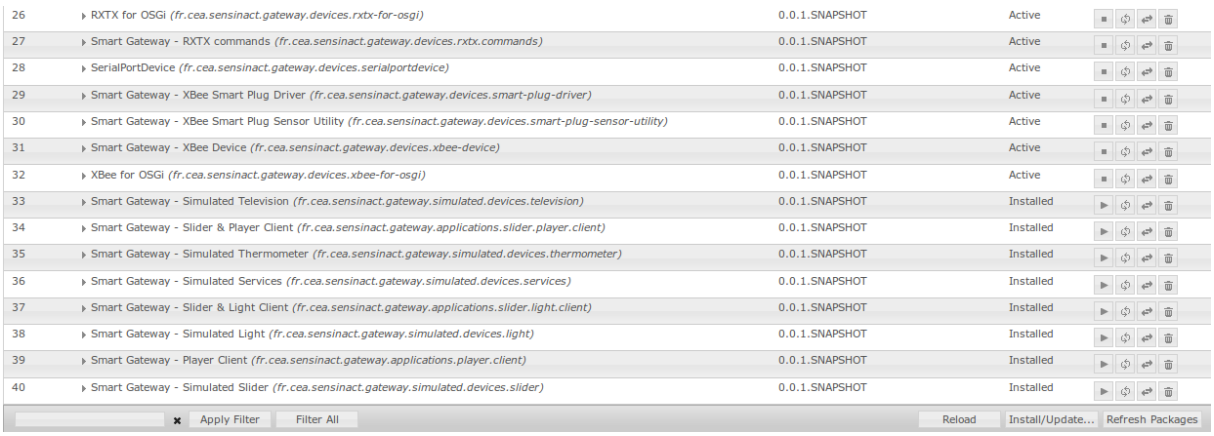

Figure 6–4: OSGi Web Console (view 2)

Lifecycle Management

From design to runtime, SmartObject has a lifecycle of five phases: modelling, development, integration, deployment and execution. All these phases are defined with the goal of increasing the reusability of software components and reducing development efforts that can be important due to the heterogeneity and the dynamic nature of IoT environments.

- **Modelling** phase refers to representing any IoT device as BUTLER SmartObjects. Each device provides then one or several services each one exposing a set of resources. BUTLER services and resources conform to the BUTLER service and resource model introduced in section 5.2.1.

- **Development** phase consists of implementing the services and resources, defined in the modelling phase, provided by devices. Some BUTLER-related developments could be done on the device (even if it is not required); however the main development tasks will be performed at the service layer. Each BUTLER SmartObjectService should implement the BUTLER API in order to allow accessing its resources in a homogeneous way.

- **Integration** makes all development components working together by identifying appropriate communication interfaces. Bridges provide the interface between the BUTLER platform and device specific communication protocols, operating systems, data models, etc. BUTLER already supports devices from various providers.

- **Deployment** phase consists of deploying implemented services, necessary libraries and technical services on the platform. Some deployment features allowing automatic charging of dependent services and libraries are implemented.

- **Execution** defines the phase where SmartObject services are running and can be reused by other BUTLER services or applications. Their execution can be monitored and controlled by management applications. With a service oriented approach, services are loosely coupled to their implementation allowing thus reconfiguration of applications at runtime.

The BUTLER SmartObject Gateway relies on the OSGi specification which provides a service model and platform for dynamic service deployment and execution. OSGi services are described by Java interfaces and properties. OSGi defines service packaging and deployment units named *bundles*. A bundle contains Java classes implementing the service functionalities and other resources such as configuration files, native libraries, images, etc. Moreover, a bundle specifies its provided and required Java packages in order to share and use code from other bundles: applications are composed by interconnecting bundles. Figure 6–5 shows the different states of a bundle lifecycle (installed, resolved, starting, active, stopping and uninstalled) and the existing transitions between them.

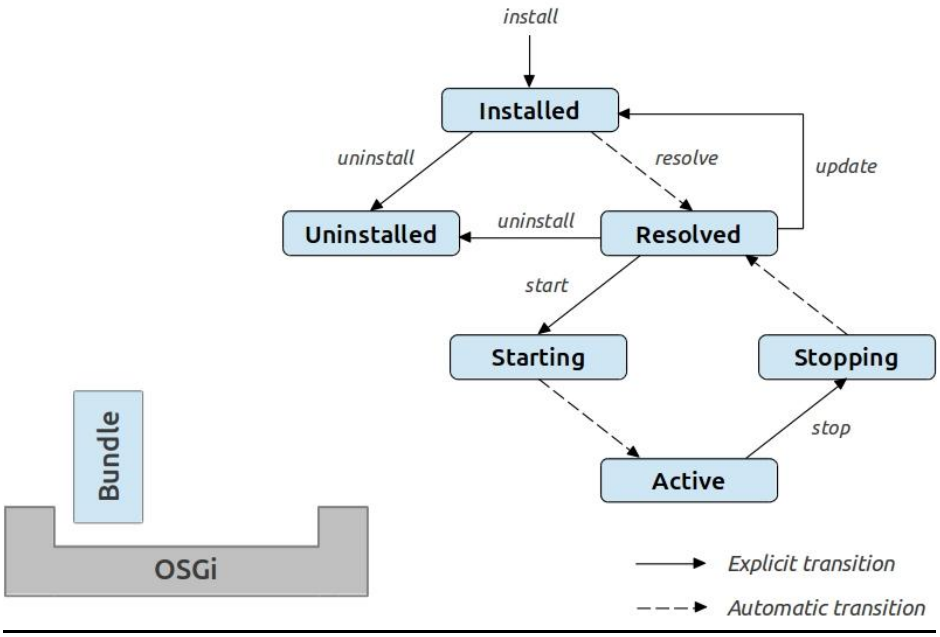

Figure 6–5: Bundle lifecycle

BUTLER Services are thus OSGi services packaged and deployed via bundles. At runtime, a SmartObjectService instance is configured and created based on the device discovery, and disposed once the device is no more available. Then, the runtime lifecycle of SmartObjectServices depends on the appearance and removal of devices. Based on the iPOJO model[3], the runtime lifecycle of a SmartService instance depends on the state of its service dependencies: a SmartService is valid if all its mandatory dependencies are satisfied, otherwise it is invalid and its resources are not exposed. SmartServices can then be reconfigured and managed dynamically. Figure 6–6 shows the lifecycle of iPOJO instances.

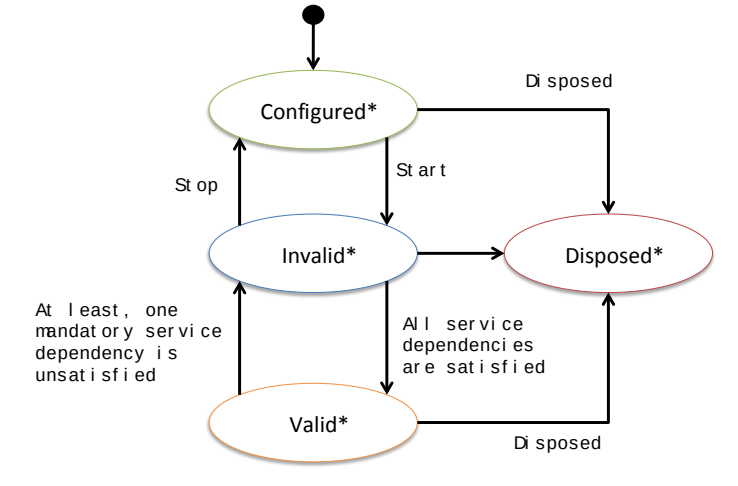

Figure 6–6: iPOJO instance lifecycle

[3] iPOJO is a service component runtime aiming to simplify the development of OSGi applications. It allows building applications exhibiting modularity and requiring runtime adaptation and autonomic behaviour.

6.3.2. Service Management API

The Service Management API provides basic management functionality for service monitoring and update. It also defines access policy to different related SmartObjects services and resources such as, permission granting and group management. This API also provides diagnostics and monitoring statistics.

Method	Signature	Description
Registration	ErrorCode Register (URL url)	Register a new service pointed by the given URL. It Returns a code error.
Lifecycle monitoring	ErrorCode Stop(String serviceID) ErrorCode Start (String ServiceID) ErrorCode Update(String serviceID, URL url) ErrorCode Uninstall (String serviceID)	Stop/start/update/uninstall a service, it also provides a code error.
Authentication	ErrorCode adduser(List<String> servicesList, SECURE_USER_Data secureUserData) ErrorCode removeUser()	Give authorization to the given user to use a list of services. Remove a user, he cannot use any service
Overseeing services and events	Log getLog() List<String> getServices(OPTION option)	Provides log events Provides the list of running services, option take different value: ALL, device (only service which are device service)

Table 6–2: Service Management API

6.3.3. Manager API

The Manager API is a generic Java based API which extends the Consumer API (defined in 5.10.) to export the management functionalities provided by the Device Management API and by the Service management API. Therefore, this API follows the security requirements adopted by the consumer API which are <SECURITY-DATA>, <PARAMETERIZED-SECURITY-DATA>. This design makes the support of multiple protocols more efficient and easier when providing dedicated protocol management access via Consumer Protocol Adapters. Basic methods of the Manager API are described in the following table.

Method	Signature
REGISTRATION	REGISTER (<PARAMETERIZED-SECURITY-DATA>) : < ERRORCODE> *Parameter(s) :=* <URL>
LIFECYCLE MONITORING	STOP(<PARAMETERIZED-SECURITY-DATA>) : < ERRORCODE> *Parameter(s) :=* <*SERVICE_ID*> START (<PARAMETERIZED-SECURITY-DATA>) : < ERRORCODE> *Parameter(s) :=* <SERVICE_ID> UPDATE(<PARAMETERIZED-SECURITY-DATA>) : < ERRORCODE> *Parameter(s) :=* <*SERVICE_ID,* URL> UNINSTALL(<PARAMETERIZED-SECURITY-DATA>) : < ERRORCODE> *Parameter(s) :=* <*SERVICE_ID*>
OVERSEEING SERVICES AND EVENTS	GETLOG(<SECURITY-DATA>) : < LOG> GETSERVICES(<PARAMETERIZED-SECURITY-DATA>) : <SERVICES_LIST> *Parameter(s) :=* <*OPTION*>
SET_PERIOD	SET_PERIOD (<PARAMETERIZED-SECURITY-DATA>) Parameter(s) := <DEVICE_ID, PERIOD>
SET_LOCATION	SET_LOCATION (<PARAMETERIZED-SECURITY-DATA>) Parameter(s) := <DEVICE_ID, LOCATION>
SLEEP	SLEEP (<PARAMETERIZED-SECURITY-DATA>) Parameter(s) := <DEVICE_ID, DURATION, UNIT>

Table 6–3: Manager API

7. Consumer Protocol Adapter

7.1. Overview

The Consumer Protocol Adapter functional group, shown in Figure 7–1, provides the Smart Server and the Smart Mobile platforms with multiple ways of interacting with the SmartObject Gateway. Each of its internal bridges (i.e., protocol adapters) relies on the Java based Consumer API and Manager API defined respectively by the Smart Object Access and Control and by the Gateway Management functional groups.

For the first implementation there will be bridges for JSON-RPC and REST API. Other adapters can be developed in the future according to the specific needs.

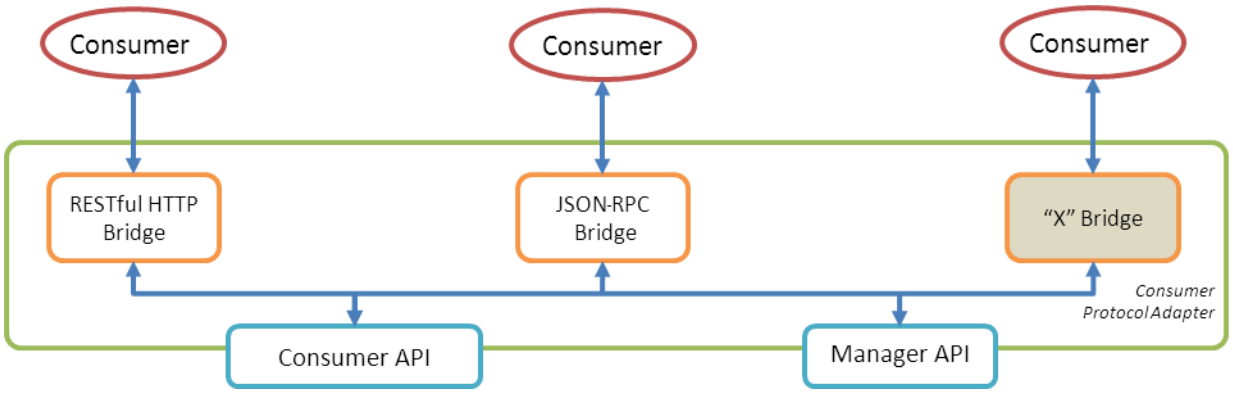

Figure 7–1: Consumer Protocol Adapter functional group

7.2. JSON-RPC Bridge

The JSON-RPC consumer protocol adapter provides an implementation of the JSON-RPC specification [7-1]. JSON-RPC is a light-weight protocol for remote procedure calls. It is built upon the ECMA-standardized JSON [7-2], a popular data interchange format, to which it adds a semantic of procedure calls (input parameters and output results).

JSON is a text format for interchanging structured data, with no particular programming language in mind. Syntax is inspired from ECMA Javascript, but can be easily parsed and generated in C, Java, Python, PHP, etc. Data interchanged in JSON are expected to be represented as textual strings, like texts and numbers. Binary data are not directly supported but can be managed through a binary encoding format such as base64.

JSON-RPC is a mechanism that can be implemented over http protocol or raw TCP/IP sockets. It only relies on a bi-directional data connection, during which each peer may invoke methods (procedures) provided by the other one. The method invocation consists in building a single serialized request, composed of three fields, respectively method, params and id:

- the 'method' field is a string giving the name of the method to be invoked,
- the 'params' field contains an JSON array of objects, representing the arguments of the method,
- the 'id' field is an arbitrary string or a number, used to match the response with the request.

When one side receives a request, it is required to answer with a response composed of the following fields:

- the 'result' is a JSON object, that can only be null in case of an error during the method invocation,
- the 'error' field is a JSON object containing any description of an error, if any,
- 'id' field must match the id given in the request.

A notification is a special request with a null id, and for which no response is expected.

The BUTLER consumer protocol adapter complies with JSON RPC 1.0. Bi-directional communication over a bi-directional stream socket connection.

For example, Table 7–1 shows how a BUTLER application can retrieve the list of installed TV sets.

REQUEST	<pre>{
	"method": "getTvDeviceList",
	"params": [],
	"id": 1
	}</pre>

RESPONSE	<pre>{
	"result": [
	{
	"id": "TV-SAMSUNG-UE32ES5500-SHCGYQLKF5JTM",
	"label": "Samsung 32' Living Room",
	},
	{
	"id": "TV-SAMSUNG-UE32ES5500-AABBCCDDEEFFG",
	"label": "Samsung 32' Kitchen",
	}
	],
	"id": 1
	}</pre>

Table 7–1: Example – Retrieve the list of installed TV sets

The next example (Table 7–2) makes use of JSON-RPC to play a video on a given TV set. The video is described by a media descriptor containing information such as the source URL of the video and the movie title. The TV set is identified with its built-in serial number.

Please notice that the request shown in Table 7–2 is a JSON-RPC notification so there is no response to it.

REQUEST	{ "method": "playToDevice", "params": ["TV-SAMSUNG-UE32ES5500-SHCGYQLKF5JTM", { "mediaId": "http://172.20.255.107:8081/multimedia/bunny_movie.mp4", "metadataMap": { "fra": { "coverUrl": "http://172.20.255.107:8081/multimedia/bunny_movie.jpg", "title": "Big Buck Bunny", "languageCode3l": "fra", "durationSec": 596, "year": 0 } }, "sourceList": ["http://172.20.255.107:8081/multimedia/bunny_movie.mp4"], "audioTracks": [], "subtitleTracks": [], "videoTracks": [] }, 0] }

Table 7–2: Example – Play a video on a given TV

7.3. REST API

The BUTLER REST Consumer Protocol Adapter provides a RESTful access to SmartObjects data and functionalities connected to a BUTLER SmartObject Gateway.

REST (REpresentational State Transfer) is an architectural style defined in 2000 by Dr. Roy Fielding [7-3] [7-4]. The basic principles behind this architectural style are that the system must follow the client-server paradigm and that the architectural components interact via requests and responses and must be accessible through uniform interfaces. Another key notion is the *resource* concept: it is everything which is accessible; its state can be transferred, and can be univocally identified and addressed by a Uniform Resource Identifier (URI).

For example an object resource can be identified by the following URI:

http://gateway.iot-butler.eu/objects/1234

In the example above, '1234' is the resource identifier.

A REST system performs its functions trough the CRUD (Create, Read, Update and Delete) operations on resources, using regular HTTP methods. In particular CRUD operations act on a resource in the following way:

- Create: mapped into the HTTP POST in order to create (or add) a new resource
- Read: it is mapped into the HTTP GET in order to access the resource
- Update: it is mapped into the HTTP PUT in order to modify the resource
- Delete: it is mapped into the HTTP DELETE in order to destroy the resource

These operations act on the same way on a single resource (e.g. http://gateway.iot-butler.eu /objects/1234) as well as in a collection of resources (e.g. http://gateway.iot-butler.eu/objects).

The BUTLER REST Consumer Protocol Adapter implements REST APIs, which are designed based on the IPSO Application Framework. [7-5]. The IPSO Framework is designed for a simple client-server interaction model on atomic resources, in order to minimize complexity. An IP SmartObject runs a simple web server (HTTP or CoAP) and exposes resources that conform to this framework. This approach is very useful for the scope of the BUTLER project, because it is very suitable for constrained devices.

It is important to remind that the IPSO Application Framework is a draft and it should be considered as work in progress. So it needs to be extended by using specific solutions for being able to manage BUTLER SmartObjects.

According to basic concepts defined by IPSO Alliance, each attribute of SmartObject Devices connected to the SmartObject Gateway is seen as a **resource** accessible via regular HTTP methods. In the following, a non-exhaustive set of examples of APIs signatures and their responses is provided.

In BUTLER SmartObject Gateway, each SmartObject is modeled as a resource and is identified by an id.

In the following examples only the relative path is considered, supposing that the SmartObject Gateway is exposing the REST interface on http/s at an endpoint like: http://gateway.iot-butler.eu

Invoking the resource /objects it's possible to obtain the list of objects ids connected to it:

Address	GET /objects
Response	200 OK ["/12345", "/agstd", "/6dt3d", "/jdj72"]

Table 7–3: Example – Get the list of objects

All the object attributes are also modeled as resources and, given the object id, the list of its available resources is retrievable accessing the endpoint

/objects/<object_id>

For example the following table represents a light device with two LEDs (identified by ids led0 and led1).

Address	GET /objects/12345
Response	200 OK ["/dev/mng", "/dev/n", "/dev/mdl", "/dev/mdl/hw", "/dev/mdl/sw", "/lt/led0", "/lt/led0/dim", "/lt/led0/on", "/lt/led1", "/lt/led1/on"]

Table 7–4: Example – Representation of a light device with two LEDs

Each resource is readable by a GET invocation to its URL. For example, the following example allows retrieving the model of the object identified by "12345":

Address	GET /objects/12345/dev/mdl

Response	200 OK
	{
	"mdl": "STM32W MB851 REV C"
	}

Table 7–5: Example – Retrieve the model information from an object

Client applications can act on objects and their resources in two ways. The operations that act on specific object's resources can be performed using HTTP PUT method. The request body must contain only the resources which values have to be changed. For example, the following tables represent respectively the operation of changing led0 dim value to 67% and the action of switching on led1:

Address	`PUT /objects/12345/lt/led0`

Body	`{`
	`  "dim": 67`
	`}`

Response	`200 OK`

Table 7–6: Example – Dim LED light

Address	`PUT /objects/12345/lt/led1`

Body	`{`
	`  "on": true`
	`}`

Response	`200 OK`

Table 7–7: Example – Turn on LED light

More complex actions (e.g. actions that need additional information to be performed) are also possible. Invoking the endpoint

/objects/<object_id>/actions

the list of actions supported by an object is returned. For example:

Address	`GET /objects/12345/actions`

Response	`200 OK`
	`[`
	`  "/action1",`
	`  "/action2"`
	`]`

Table 7–8: Example – Get the list of actions supported by an object

According to the response of the previous API and to the SmartObject Gateway documentation that describes additional parameters (if any) to be used on each supported action, the operation can be performed using HTTP PUT method. The response will contain the resource itself, with attributes values resulting from the action. For example:

Address	`PUT /objects/12345/actions/action1?par1=xyz`

Response	`200 OK`
	`{`
	`  "res1": "abc",`
	`  "res2": 5234`
	`}`

Table 7–9: Example – HTTP PUT method

The REST Consumer Protocol Adapter supports also subscriptions to the SmartObject Gateway. Client applications can send subscription requests, eventually specifying some filtering criteria, to the API described in the following example. The client has to keep the HTTP/S connection to the SmartObject Gateway open; when the requested condition is verified, the resource itself will be sent as notification.

To send a subscription to the SmartObject Gateway, a client application must send an HTTP GET request to the endpoint
/objects/<object_id>/subscriptions/<resource_id>?<filter_criteria>

The filter criteria parameter is optional and can be omitted if the client is interested in all changes of the resource value. Next examples show respectively a subscription to state change (on/off) of led1 and a subscription to changes in at least one degree read by a temperature sensor:

Address	`GET /objects/12345/subscriptions/led1/on`

Response (each time the notification condition occurs)	`200 OK`
	
	`{`
	`  "on": true/false`
	`}`

Table 7–10: Example – Subscription to state change for LED light

Address	`GET /objects/12345/subscriptions/sen/temp?st=1`

Response (each time the notification condition occurs)	`200 OK`
	
	`{`
	`  "temp": <degrees>`
	`}`

Table 7–11: Example – Subscription to value change for temperature sensor

Currently defined filter criteria are:

- st – unit step
- eq – equals
- gt – greater than
- lt – lower than

Error messages returned by the APIs described in this section are modeled as detailed in the following table:

Example error response	xxx HTTP Status Code
	{
	"code": <application_error_code>,
	"msg": "<error message>",
	"descr": "<higher level error description>"
	}

Table 7–12: Example – Error response

7.3.1. NFC Parking Use Case Example using REST API

In the following section we are going to use the Smart Parking Use Case ([D1-2], Section 5.5) as a *How-To* example for building a simple NFC Payment Service using the REST API. The use case conceives in each street one *Smart Object Device* controlling several *Vehicle Detection Sensors* and *Actuators* (namely the *Lighting System*). Now, NFC Modules are going to be attached to each *Smart Object Device* enabling NFC transactions.

First Step

The first step is discovering the NFC Modules by querying the *NFC General Group*, in which developers only have "read" privilege (as in UNIX systems, **r** is reading, **w** is writing and **x** is executing).

Address	`http://api.iot-butler.eu/v1/NFC/Groups/<NFC_General_Group_ID>`
Body	
Method	`GET`

Response	`200 OK`
	`{`
	`"ID": "ABCDEF123456",`
	`"Name": "NFC General Group",`
	`"Description": "This group exposes all NFC Module resources",`
	`"Members": [`
	`{"ID": "AAA"}, {"ID": "BBB"}, {"ID": "CCC"},`
	`{"ID": "DDD"}, {"ID": "EEE"}, {"ID": "FFF"}`
	`],`
	`"Permissions": "r--"`
	`}`

Example of possible error response	`404 Not Found`
	`{`
	`"Status": 404,`
	`"Message": "The resource based on the specified URI does not exist",`
	`"Info": "http://api.iot-butler.eu/docs/errors/404"`
	`}`

As a developer, you can now create your own NFC Group by calling the POST method. Inside the response you get the ID of the group you just created or an error code in case of failure. Being the owner of the group grants you with all privileges.

Address	http://api.iot-butler.eu/v1/NFC/Groups

Body	{
	"Name": "My Parking Group",
	"Description": "A Collection of my NFC Modules"
	}

Method	POST

Response	201 CREATED
	{
	"ID": "A1B2C3D4E5F6",
	"Name": "My Parking Group",
	"Description": "A Collection of my NFC Modules",
	"Members": null,
	"Permissions": "rwx"
	}

Example of possible error response	401 Unauthorized
	{
	"Status": 401,
	"Message": "Authentication credentials are required to access the resource",
	"Info": "http://api.iot-butler.eu/docs/errors/401"
	}

Finally add the desired NFC Modules you want to operate inside the group created, based on the members list obtained in the previous call.

Address	http://api.iot-butler.eu/v1/NFC/Groups/A1B2C3D4E5F6

Body	{
	"Members": [
	{"ID": "AAA", "Name": "SmartParking NFC Module A"},
	{"ID": "BBB", "Name": "SmartParking NFC Module B"},
	{"ID": "CCC", "Name": "SmartParking NFC Module C"},
	]
	}

Method	PUT

Response	200 OK
	{
	"ID": "A1B2C3D4E5F6",
	"Name": "My Parking Group",
	"Description": "A Collection of my NFC Modules",
	"Members": [
	{"ID": "AAA", "Name": "SmartParking NFC Module A"},
	{"ID": "BBB", "Name": "SmartParking NFC Module B"},
	{"ID": "CCC", "Name": "SmartParking NFC Module C"},
	],
	"Permissions": "rwx"
	}

Example of possible error response	403 Forbidden
	{
	"Status": 403,
	"Message": "The supplied authentication credentials are not sufficient to access the resource",
	"Info": "http://api.iot-butler.eu/docs/errors/403"
	}

Second Step

In this step an NFC Module is going to be configured for measuring readings (thus, Read/Write mode). First update its settings using a PUT call:

Address	http://api.iot-butler.eu/v1/NFC/Modules/AAA

Body	{
	"Description": "This NFC Module will be reading NFC Tags",
	"Key": "111222333444555666AAABBBCCCDDDEEEFFF",
	"Mode": "RW"
	}

Method	PUT

Response	200 OK
	{
	"ID": "AAA",
	"Name": "SmartParking NFC Module A",
	"Description": "This NFC Module will be reading NFC Tags",
	"Key": "111222333444555666AAABBBCCCDDDEEEFFF",
	"Mode": "RW",
	"Timestamp": null,
	"Groups": ["ID": "ABCDEF123456", "ID": "A1B2C3D4E5F56"]
	}

Example of possible error response	400 Bad request
	{
	"Status": 400,
	"Message": "A required query parameter was not specified for this request",
	"Info": "http://api.iot-butler.eu/docs/errors/400"
	}

The chosen NFC Module is well configured using the BUTLER Consumer API, so now it is time to the underlying BUTLER Device Access API to act upon the SmartObject, what is explain with further details in Section 4.4.

Third Step

Following with the example, retrieve all the NFC Tags that have been read so far.

Address	http://api.iot-butler.eu/v1/NFC/Tags/

Body	{
	"Get": "All",
	"GetFrom": "AAA"
	}

Method	GET

Response	200 OK
	[
	{
	"ID": "1",
	"Key": "12345",
	"Data": "Hello world!",
	"Timetag": "2013-07-08T13:40+01"
	},
	{
	"ID": "2",
	"Key": "12345",
	"Data": "Hola mundo!",
	"Timetag": "2013-07-08T13:43+01"
	},
	{
	"ID": "3",
	"Key": "56789",
	"Data": "Bonjour monde!",
	"Timetag": "2013-07-08T13:45+01"
	}
	]

Example of possible error response	400 Bad Request
	{
	"Status": 400,
	"Message": "The value provided for one of the parameters was not in the correct format",
	"Info": "http://api.iot-butler.eu/docs/errors/400"
	}

After a while, request only new NFC Tags read, i.e. excluding the ones returned in previous calls.

Address	http://api.iot-butler.eu/v1/NFC/Tags/

Body	{
	"Get": "New",
	"GetFrom": "AAA"
	}

Method	GET

Response	200 OK
	[
	{
	"ID": "4",
	"Key": "56789",
	"Data": "Ciao mondo!",
	"Timetag": "2013-07-08T14:20+01"
	},
	{
	"ID": "5",
	"Key": "12345",
	"Data": "Hallo welt!",
	"Timetag": "2013-07-08T14:31+01"
	}
	]

Example of possible error response	413 Request Entity Too Large
	{
	"Status": 413,
	"Message": "The size of the request body exceeds the maximum size permitted",
	"Info": "http://api.iot-butler.eu/docs/errors/413"
	}

8. References

[4-1] G. Montenegro et al, Transmission of IPv6 Packets over IEEE 802.15.4 Networks, RFC 4944, IETF, September 2007

[4-2] IEEE 802.15.4-2011 http://standards.ieee.org/findstds/standard/802.15.4-2011.html

[4-3] T. Winter et al, RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks, RFC 6550, IETF, March 2012

[4-4] Z. Shelby et al, Constrained Application Protocol (CoAP), IETF Draft, June 2013 http://datatracker.ietf.org/doc/draft-ietf-core-coap/

[6-1] OSGi Framework Specification http://www.osgi.org

[6-2] The iPOJO Service Component Runtime http://felix.apache.org/site/apache-felix-ipojo.html

[7-1] JSON-RPC 1.0 Specification http://www.simple-is-better.org/json-rpc/jsonrpc10.html

[7-2] The JSON Data Interchange, ECMA, Standard 404, First Edition, October 2013 http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

[7-3] Roy Fielding, Chapter 5 of Ph.D. dissertation: "Representational State Transfer (REST)" http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[7-4] M. Lanthaler, C. Gutl. Towards a RESTful service ecosystem. 4th IEEE International Conference on Digital Ecosystems and Technologies (DEST). April 2010, pp. 209-214

[7-5] The IPSO Application Framework, IPSO Alliance Working Document http://www.ipso-alliance.org/wp-content/media/draft-ipso-app-framework-04.pdf

References to other BUTLER Deliverables:

[D1-2] Deliverable D1.2 - Refined Proof of Concept and Field Trial Specification. May 2013

[D3-1] Deliverable D3.1 - Architectures of BUTLER Platforms and Initial Proofs of Concept. October 2012

[D3-2] Deliverable D3.2 - Integrated System Architecture and Initial Pervasive BUTLER proof of concept. October 2013