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

D4.1 - BUTLER SmartServer Platform and Enabling Technologies

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:
TIL	V1.2

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 (GTO)
Shadi Atalla, Federico Rizzo, Antonio Simone, Francesco Sottile (ISMB)
François Nacabal (MAYA)
Cristina Frà, Massimo Valla (TIL)
Maria Fernanda Salazar, Miguel-Angel Monjas (ERC)
Alexey Andrushevich, Rolf Krisler (IHL)
Arun Kishore Ramakrishnan, Davy Preuveneers (KUL)
Fabrice Clari (INNO)

Thanks to Bertrand Copigneaux (INNO), Luca Lamorte (TIL) and Miguel-Angel Monjas (ERC) for their deliverable 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 secure and context aware services, interconnected with the IoT.
The SmartServer Platform is the result of task T4.1 activities and is described in this document. It is composed of a set of SmartServers to provide common horizontal functionalities needed by several BUTLER applications, used in BUTLER Proof of Concepts or in BUTLER trials. Examples of such functionalities are: authentication and authorization, user profile, context-awareness, localization and user behavior. This document is intended as an “accompanying report” to describe the SmartServer software that has been developed and deployed either locally in a target application environment/scenario, or in the cloud, to be accessed remotely by other servers, mobile applications, or third parties.

In summary this document is the SmartServer and API guide for developers interested in developing BUTLER applications and re-using common functionalities offered by the “open” BUTLER horizontal platform.
In addition to this document the BUTLER API Catalogue, available online, is the reference point to provide a quick overview on available SmartServers in BUTLER and where developers are able to test the APIs offered by the BUTLER horizontal platform.

Keywords :

Platform, Horizontality, Servers, API, API Catalogue, Developer Support, Documentation, Open System, Common Functionalities, REST, Architecture, Integration, Authorization, User Profile, Context-awareness, Complex Event Processing, Location, User Behavior, Multimedia Server, Energy Forecasting, Energy Data, SmartServer

Disclaimer

Revision History

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

Revision	Date	Description	Author (Organisation)
V0.1	30 May 2013	Initial TOC and assignments of sections to partners	TIL
V0.2	4 Sept 2013	Revised and shared TOC, first contributions integrated	TIL
V0.3	16 Oct 2013	User Profile SmartServer (TIL) integrated (updated version)	TIL
V0.4	23 Oct 2013	Integrated Context Manager (TIL), Localization Manager (ISMB), User Behavior (KUL), Multimedia (MAYA, CEA), Mobile App Directory (INNO), Authorization (GTO), DDX (ERC), Energy Forecasting (ERC) SmartServers	TIL, ISMB, KUL, MAYA, CEA, INNO, GTO, ERC
V0.5	30 Oct 2013	Updated descriptions from MAYA, KUL, ISMB	TIL, MAYA, KUL, ISMB
V0.6	31 Oct 2013	Chapter 1 and chapter 2. Acronyms. Checked references. Updated descriptions for: Developers' Support, DDX, Energy Forecasting, Energy Awareness Data, Authorization chapters	TIL, ISMB, ERC, GEMALTO, IHL
V0.7	31 Oct 2013	Abstract, author list, Conclusions. Final formatting and editorial review	TIL
V0.8	31 Oct 2013	Ready for internal review	TIL
V0.9	31 Oct 2013	Changes according to reviewers comments	TIL
V1.0	31 Oct 2013	Final document to be submitted to EC	INNO
V1.1	19 Nov 2013	Additional internal review, references corrected	ERC, TIL
V1.2	24 Nov 2013	Changes according to additional review	TIL, INNO

Table of Content

List of Figures

List of Tables

Acronyms

Acronym	Defined as
AES	Advanced Encryption Standard
API	Application Programming Interface
AS	Authorization Server
BRMS	Business Rules Management System
BSON	Binary JSON
CC	Context Consumer
CEP	Complex Event Processing
CMP	Container Managed Persistence
CoAP	Constrained Application Protocol
CP	Context Provider
CRUD	Create Read Update Delete
CS	Context Source
DB	Data Base
DDX	Device Data eXchange
EAR	Enterprise Application Archive (from J2EE specification)
GUI	Graphical User Interface
GW	GateWay
HTTP	Hyper-Text Transfer Protocol
HTTPS	Hyper-Text Transfer Protocol Secured
IoT	Internet of Things
IP	Internet Protocol
J2EE	Java 2 Enterprise Edition
JAR	Java ARchive
JAX-RS	Java API for RESTful Web Services
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
JSP	Java Server Pages
LAN	Local Area Network
LM	Localization Manager
M2M	Machine-to-Machine
MQTT	MQ Telemetry Transport
NAT	Network Address Translation
NFC	Near Field Communication
NoSQL	Not only SQL (database)
OAUTH 2.0	Open standard for authorization 2.0
OMA NGSI	Open Mobile Alliance - Next Generation Service Interface
ORM	Object-Relational Mapping
REST	Representational State Transfer
RSSI	Received Signal Strength Indicator
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TLS	Transport Layer Security
ToA	Time of Arrival
TDoA	Time Difference of Arrival
URL	Uniform Resource Locator USB Universal Serial Bus

UWB	Ultra-Wide Band
WAR	Web application ARchive
WEKA	Waikato Environment for Knowledge Analysis
XML	eXtensible Markup Language

Table 1 - Acronyms and Definitions

1. Executive Summary

The **SmartServer Platform** described in this document comprises a set of SmartServer instances, as implemented until M24, providing horizontal functionalities needed by several BUTLER applications, used in BUTLER Proof of Concepts or in BUTLER trials. This document is intended as an "accompanying report" to describe the software developed and deployed as BUTLER SmartServers, either locally in a target application environment/scenario, or in the cloud to be accessed remotely by other servers, mobile applications, or third parties.

In summary this document is the SmartServer and API guide for developers interested in developing a BUTLER application and re-using common functionalities offered by the "open" BUTLER horizontal platform.

In the first version of the SmartServer Platform until M24 the following SmartServer instances have been implemented and made available to BUTLER developers:

- Device Data eXchange (DDX) SmartServer
- Localization Manager SmartServer
- Renewable Energies Forecasting SmartServer
- Energy Awareness Data SmartServer
- User Profile SmartServer
- User Behavior SmartServer
- Context Manager SmartServer
- Multi Media SmartServer
- Authorization SmartServer

In addition an Application Repository has been developed to provide a way to deploy and manage BUTLER mobile applications, and -in case of HTML-5 applications-, to host them on the repository so that BUTLER SmartMobile clients can access and install them on the mobile framework on the device.

The general SmartServer approach chosen, together with a detailed list of the available SmartServers and the responsible partners is provided in the first chapter. Integration between platforms is also described, along with the general API approach used by most of SmartServers, which follows a common RESTful API design.

Following the common approach section, a chapter is dedicated to each SmartServer, providing to developer an explanation of the functionalities offered, the implemented software architecture, and the enabling technologies form WP2 that have been used to develop each software module.

Since each SmartServer provides to developers a set of API, this document also provides enough details to understand and use the provided API and integrate common horizontal functionalities in BUTLER applications. Examples of such functionalities are: authentication and authorization, user profile management, context-awareness, localization and user behavior.

In addition to the API description that can be found in this document, developers have also another important source of information. The BUTLER API Catalogue is the reference point to have a quick view on available SmartServers in BUTLER and the details of the offered APIs.

The BUTLER API Catalogue tool also offers "Live Testing" functionality, where the developer is able, after proper authorization, to learn about API syntax, parameters used and make sample requests to the API endpoint listed in the catalogue, checking the answer received by the endpoint. This is a fast and

convenient way for developers to learn about BUTLER platform APIs and test them in a sort of “BUTLER developer workbench”. A chapter is dedicated to describe these developer tools and the approach used.

2. SmartServer Platform

As defined in D2.4 [1], in BUTLER a Smart Server is *"a set of software components that provide functionalities through a set of APIs to applications, SmartObjects/Gateways and SmartMobile"*. The role of SmartServers is to provide an interface to the world of digital information, processing and mediating information that is relevant to the user and the BUTLER applications he is using.

The **SmartServer Platform** described in this document is composed of a set of SmartServer instances, as implemented until M24, to provide horizontal functionalities needed by several BUTLER applications, used in BUTLER Proof of Concepts or in the BUTLER trials.

BUTLER application developers can find in the BUTLER SmartServer platform common required functionalities that could be used by several applications: by accessing the corresponding SmartServer API, developers can integrate the functionality offered by a SmartServer into the application business logic, simplifying its complexity and reducing the time to integrate the application in the BUTLER eco-system.

Since the SmartServer Platform is made of several SmartServers, all SmartServers API documentation is collected in the **BUTLER API Catalogue** [7], where developers will be able to find all the relevant documentation for the offered APIs and also a testing environment where is possible, in a guided way, to compose API requests, set invocation parameter and check retuned responses.

Smart Servers can be deployed in different ways:

Remote or Cloud Smart Servers
These are Smart Servers components that provide general functionalities and access to information available from different Internet. The main role of Remote Smart Servers is to connect Smart Objects/Gateways and Smart Mobile with 3rd party applications or web data sources and to offer general centralized functionalities.

Local Smart Server
Smart Server can also be deployed locally in an environment such as a household, an office building or a train. In this case server side functionalities will be offered again thought a set of APIs that are directly accessible within the local network. Service discovery and security are offered through the Local Smart Server. The Local Smart Server can also work as an application level proxy for functionalities available remotely on the cloud.

While developing the BUTLER SmartServer platform a uniform programming interface (API) model to develop Smart Server functionalities has been proposed, applications can invoke functionalities and APIs on a Local or Remote Smart Server without any difference.

2.1. SmartServer functionalities

Figure 1 illustrates the BUTLER SmartServer Platform as the first set of developed SmartServers. The servers were chosen according to the most important, common and horizontal functionalities needed by the first set of BUTLER applications and proof of concepts.

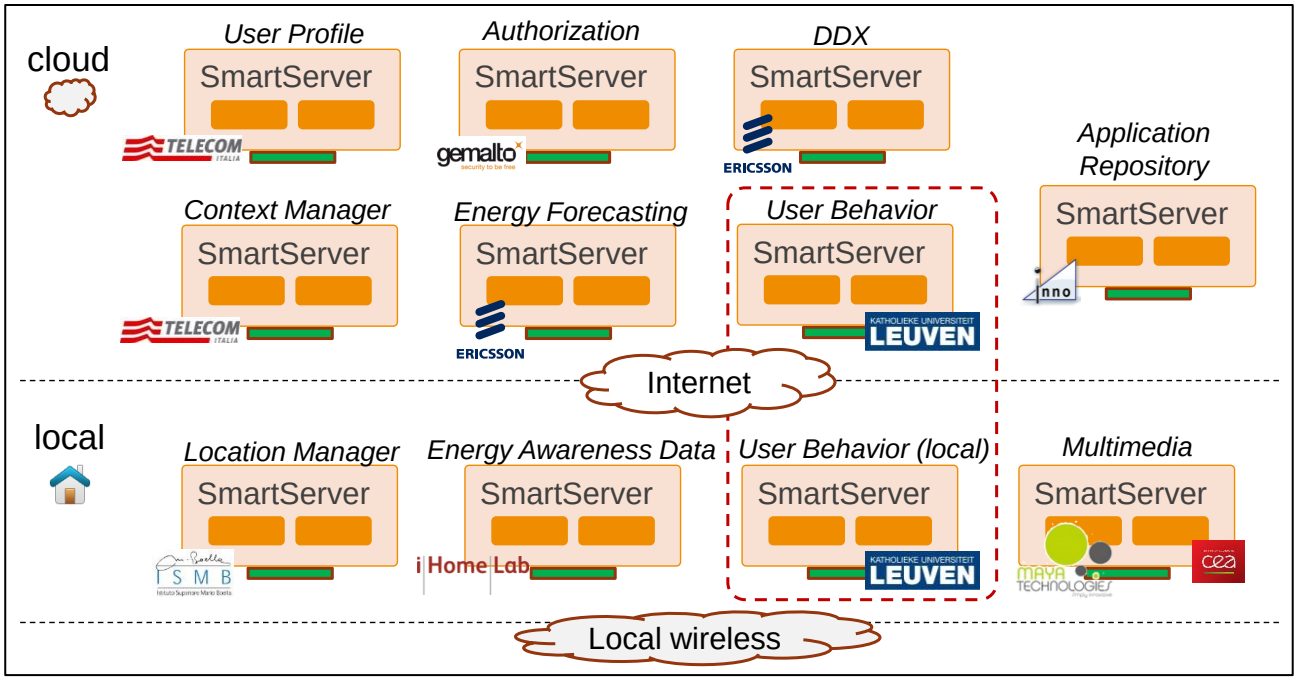


Figure 1: The BUTLER SmartServers available in the SmartServer Platform at M24.

Some of the SmartServers are designed to be deployed locally to the environment, others are provided as public internet endpoints in the "BUTLER cloud", to provide common services or integrate functionalities provided by other systems. For example the Localization Manager SmartServer is deployed locally to collect location ranging measure and provide to other applications or server integrated information about location of a user or device. The Context Manager, on the other side, is deployed remotely on the cloud to aggregate context information that is collected from several sources, devices or users.

The following table summarizes the list of current SmartServer instances that are offered in the first version of the BUTLER SmartServer platform. For each SmartServer a brief description of the offered functionality is provided, together with the responsible consortium partner.

BUTLER SmartServer	Responsible Partner	Functionalities offered	Local/Remote (Cloud)
Device Data eXchange (DDX)	Ericsson España S.A. (ERC)	Data Marketplace, Resource and Context Exposition, Complex Event Processing, Permanent Storage	**Remote**
Localization Manager	Istituto Superiore Mario Boella (ISMB)	Estimated positions of SmartObjects in real time, last updated positions and related history	**Local**
Renewable Energies	Ericsson España S.A.	Energy consumption and renewable energy generation predictions based on three different	**Remote**

Forecasting	(ERC)	inputs: current consumption rates, current weather conditions and weather forecast	
Energy Awareness Data	iHomeLab (IHL), Telecom Italia (TIL)	Access to energy consumption information about local devices, their status and device control	**Local**
User Profile	Telecom Italia (TIL)	User profile management: registration, retrieval and update of storage of user profile data and application preferences. Association of user profile to his/her devices	**Remote**
User Behavior	Katholieke Universiteit Leuven (KUL)	Activity recognition and behavior modeling: feature extraction, information fusion, domain knowledge, probabilistic correlations	**Local/Remote**
Context Manager	Telecom Italia (TIL)	Context management for users and objects, context aggregation, context history storage, generation of higher-level context and notification	**Remote**
Multi Media	Maya (MAYA), Commissariat à l'Energie Atomique (CEA)	Common and coherent view of multimedia contents available for playing, multimedia content indexing. Content may be stored locally, or streamed from a network location	**Local**
Authorization Server	Gemalto SA (GTO)	Authorization management, authentication	**Remote**
Application Repository	Inno AG (INNO)	BUTLER Applications life-cycle management; hosting of BUTLER HTML-5 applications	**Remote**

Table 2 - Summary of SmartServers part of the first release of the BUTLER SmartServer Platform

2.2. Integration with other BUTLER platforms

In general SmartServers are integrated with other BUTLER components using IP communication; in particular when communicating with a SmartMobile or Cloud SmartServer the HTTP protocol is used to provide easy accessibility even in conditions where other protocols are not allowed, for example when the communication is behind a NAT, Firewall or HTTP proxy. In a local environment where there is usually direct IP connectivity (being it Wi-Fi or LAN), other communication protocols can be chosen. Another deployment possibility is also that both SmartServer and SmartObject Gateway are installed on the same hardware device, for example a gateway board or a local PC. In this case communication can also be performed over direct Java integration through Java APIs.

In figure below the integration and communication between the three platforms (SmartServer, Smart Object Gateway and SmartMobile) is represented.

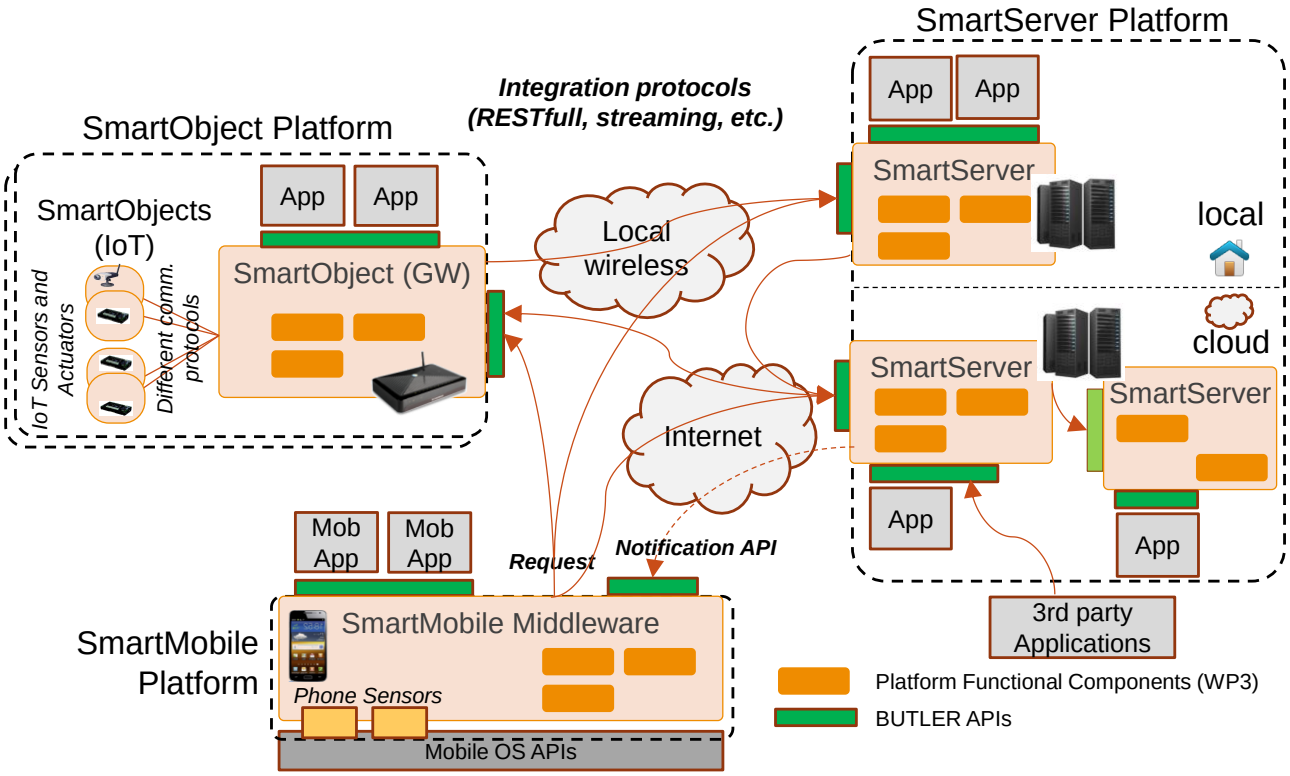


Figure 2: Integration of the three platforms in BUTLER.

In particular it is important to note the following:

- SmartMobile
  - can communicate directly with SmartObject GW as well as by passing through a SmartServer, which can act as a proxy between SmartMobile and SmartObject GW
  - can communicate with a SmartServer either through local wireless or via public internet
  - do not offer any external API, except for a Notification API to receive notification from SmartServers
- SmartObject GW
  - can be contacted by a SmartMobile or SmartServer
  - can send request to local or remote SmartServer, for example for authorization check
- SmartServers

- can communicate between each other, for example a local SmartServer can use APIs and functionalities offered by a remote (cloud) SmartServer, or two SmartServer deployed in the cloud can use each other's APIs to perform specific business logics
- applications can run on a local or remote SmartServer
- third party applications are integrated by accessing SmartServer APIs

2.3. Common API Design

2.3.1. URL General Format

In general, each SmartServer should support a common URL format that identifies: the functionality offered by the SmartServer, the version of the API and the specific API functionality, followed by the REST resource tree to address the specific resource. The general URL format for the BUTLER SmartServer is thus the following:

```
https://<smart-server-host>:<port>/<specific-functionality-path>/<API-version>/<resource-Path>
```

Authorization mechanisms, as described in section 4.4.3 of the BUTLER deliverable D3.2 [3], require also the use of proper HTTP headers to provide authorization data in the request, as the standard HTTP requires:

```
Authorization: Bearer <authorization-token>
```

For example the following URL can be used to request for current location of an entity '25957' returned by the Localization Manager SmartServer:

```
GET  http://localization.iot-butler.eu/api/v1/objects/25957/localization/..
```

where the API version is v1, and the resource path starts with: /objects/25957/localization/..

2.3.2. Restful API best practices for BUTLER

This section provides an overview of best practices commonly adopted when designing REST APIs [8]: most of the BUTLER SmartServers web APIs described in this document are designed according to these conventions; those that currently do not follow these convention exactly will be aligned in future versions of the platform.

The key principles of REST involve separating the APIs into logical resources. These resources are manipulated using HTTP requests where each method (GET, POST, PUT, DELETE) has specific meaning. Once the resources have been defined, developers need to identify which actions apply to them and how those map to the APIs. RESTful principles provide strategies to handle **CRUD** actions using HTTP methods mapped for example as follows:

- GET /books - Retrieves a list of books
- GET /books/23 - Retrieves a specific book
- POST /books - Creates a new book
- PUT /books/23 - Updates book #23
- DELETE /books/23 - Deletes book #23

A main advantage of REST is that leveraging on existing HTTP methods it is possible to implement significant functionality on just a single /books endpoint. There are no method naming conventions to follow and the URL structure is clean and clear. To keep things simple, as REST paradigm suggests, the endpoint names should always be nouns (not verbs) and plural. The API provider has not to deal with odd pluralization and can manage the implementation in an easier way (e.g. it can handle /books and /books/23 APIs under a common controller).

Sometimes the actions do not fit into the world of CRUD operations and there is no way to map them to a sensible RESTful structure. For example, a multi-resource search does not really make sense to be applied

to a specific resource's endpoint. In this case for example it is possible to use, enclosing the proper documentation to avoid confusion, the endpoint /search even if it is not a noun.

2.3.1. Data serialization

In BUTLER JSON is preferable to XML or any other serialization format for both request and response bodies. It is preferable to return data directly in API responses, like in the following examples:

Returning an object:

```
{
    "id" : "ye475t”,
    "name" : "Anna"
}
```

Returning a list:

```
["item1", "item2"]
```

2.3.1. BUTLER error message formats

A well-designed REST API should provide a useful error message in a known consumable format. The representation of an error should be no different than the representation of any resource, just with its own set of fields. The API should always return sensible HTTP status codes and typically they should come with consumable JSON error representation.

A JSON error body should provide to the developer at least a useful error message, a unique error code (that can be looked up for more details in the API’s documentation) and possibly a detailed description. JSON output representation would look like:

```
{
  "code" : P27,
  "message" : "Error message",
  "description" : "More details about the error"
}
```

HTTP defines a bunch of meaningful status codes that can be returned from the API. These can be leveraged to help the API consumers route their responses accordingly. Well-designed REST APIs should at least rely (when possible) on the following:

- 200 OK - Response to a successful GET, PUT or DELETE. Can also be used for a POST that does not result in a creation.
- 201 Created - Response to a POST that results in a creation. Should be combined with a Location header pointing to the location of the new resource
- 204 No Content - Response to a successful request that will not be returning a body (like a DELETE request)
- 304 Not Modified - Used when HTTP caching headers are in play
- 400 Bad Request - The request is malformed, such as if the body does not parse
- 401 Unauthorized - When no or invalid authentication details are provided
- 403 Forbidden - When authentication succeeded but authenticated user does not have access to the resource
- 404 Not Found - When a non-existent resource is requested

- 405 Method Not Allowed - When an HTTP method is being requested that is not allowed for the authenticated user
- 410 Gone - Indicates that the resource at this end point is no longer available. Useful as a blanket response for old API versions
- 415 Unsupported Media Type - If incorrect content type was provided as part of the request.

3. DDX SmartServer

DDX stands for Device Data eXchange and works as a BUTLER SmartServer instance implementing or using some of the functional components defined in the BUTLER Architecture, namely the Data Marketplace, the Resource and Context Exposition, the Complex Event Processing engine and the Permanent Storage (see sections 4.4.6 and 4.4.7 in D3.2 [3]). This multi-sided platform:

- Provides Organized, real-time and ready-to-use data from devices and sensors
- Enables the creation of a M2M ecosystem
- Brings together Data Owners and App Developers
- Resulting in new revenue streams

As can be seen in figure below, BUTLER DDX platform comprises two main functional modules: the Real-Time Data Marketplace –the front end of the solution composed by the Data Owner Portal (DOP), and the Dynamic Data Mall (DDM)–, and the DDX Data Broker –the data communication and processing module–.

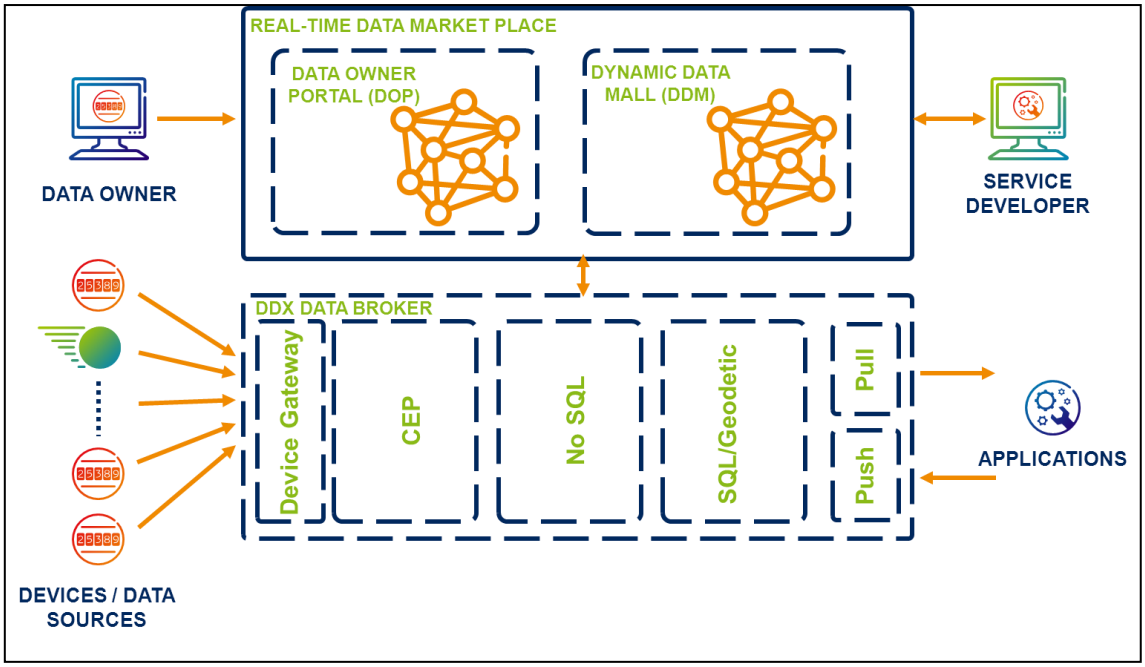


Figure 3: BUTLER DDX High level architecture.

3.1. Functionalities

One of the functionalities provided by BUTLER is the possibility of accessing exposed data generated by BUTLER SmartObjects. This exposed data can be roughly described as a continuous flow of values measured by SmartObjects. Such functionality enables third-party developers to create new applications on top of BUTLER. BUTLER addresses the exposition and purchase of BUTLER SmartObjects data by said third-parties through DDX, which mediates between SmartObjects and applications. DDX aims to enable Service Developers to discover and use data from SmartObjects through a single discovery point. It also addresses the management of simple events that Service Developers can define taking SmartObject data as input. That way, Service Developers can define a set of conditions that take SmartObject data as input and trigger actions whenever the conditions are met.

Two main roles are thus defined: the aforementioned Service Developer and the so-called Device Owner:

- **Device Owners** are the actual owners (or administrators) of SmartObjects. They are the ones that make a decision on sharing with third-parties the information coming from their SmartObjects.
- Said third parties, **Service Developers**, are in turn those wishing to use data coming from SmartObjects to create new services and applications.

Functionalities are classified according to the user role:

3.1.1. Device Owner

- Online registration of Data Owners.
- Data sources registration: The Data Owner is provided with an intuitive graphical interface in order to add and connect his/her data sources to the DDX Data Broker.
- Map-based data source management interface: The Data Owner will be provided a map-based interface to manage individual data sources and entities.
- Real-time data monitoring: Real-time charts are provided on the data sources the Data Owner has added to the platform.
- Individual or batch upload of data sources descriptions: The Data Owner will be able to manually register data sources in the BUTLER DDX platform. S/he will be also able to make bulk uploads of data source descriptions by using specially formatted XML documents.
- Data Brokerage towards service developers: Data from the Data Owner's data sources can be distributed to third parties.
- Flexible definition of data offerings: The Data Owner will be provided with the needed tools to define and manage his/her data offering by adding one or more individual data sources to the so-called packages.
- Bulk storage (condition-based or not): Data items from the Data Owner's data sources can be stored for subsequent use by the Data Owner.

3.1.2. Service Developer

- Online registration of Service Developers: Service Developer can register at any time, even without any previous commercial relationship to the DDX operator (focused on B2C scenarios).
- Data Owner-defined Data Offerings: The Service Developer is able to purchase the Data Offerings already defined by the Data Owner.
- Map-based discovery interface: The Service Developer can use a map-based interface to search for entities and individual data sources (Data Offerings needn't have a geographical representation)

- Search by data owner, tag and offering name: When searching for data offerings (also called packages), the aforementioned arguments should be available. An individual data offering can have the following features:
    - Data Owner: the business entity selling data
    - Tag: keyword(s) assigned by the Data Owner that describes the content of the data source included into the package
    - Name: the name the Data Owner gave to the package
- Service Developer-defined Business Events: The Service Developer is able to define business events taking as input selected data sources s/he has purchased.
- Real-time data monitoring: Dynamic charts are provided over the Data Offerings the Service Developer has purchased.
- Pull (over cached data) and push (publication/subscription) interfaces: Two different mechanisms for data access are defined:
    - Publication/subscription: a dynamic mechanism for the Service Developer application to receive a data stream
    - Read: a static mechanism for the Service Developer application to query both static data sets and stored values of dynamic data sources
- JSON-based interfaces: Available interfaces will be based on JSON (or other simple interfaces)

3.2. Software Architecture

This section provides a brief description of the different modules that are part of the DDX Data Broker platform and the related interfaces that allows the connection between the Market Place and the DDX Data Broker.

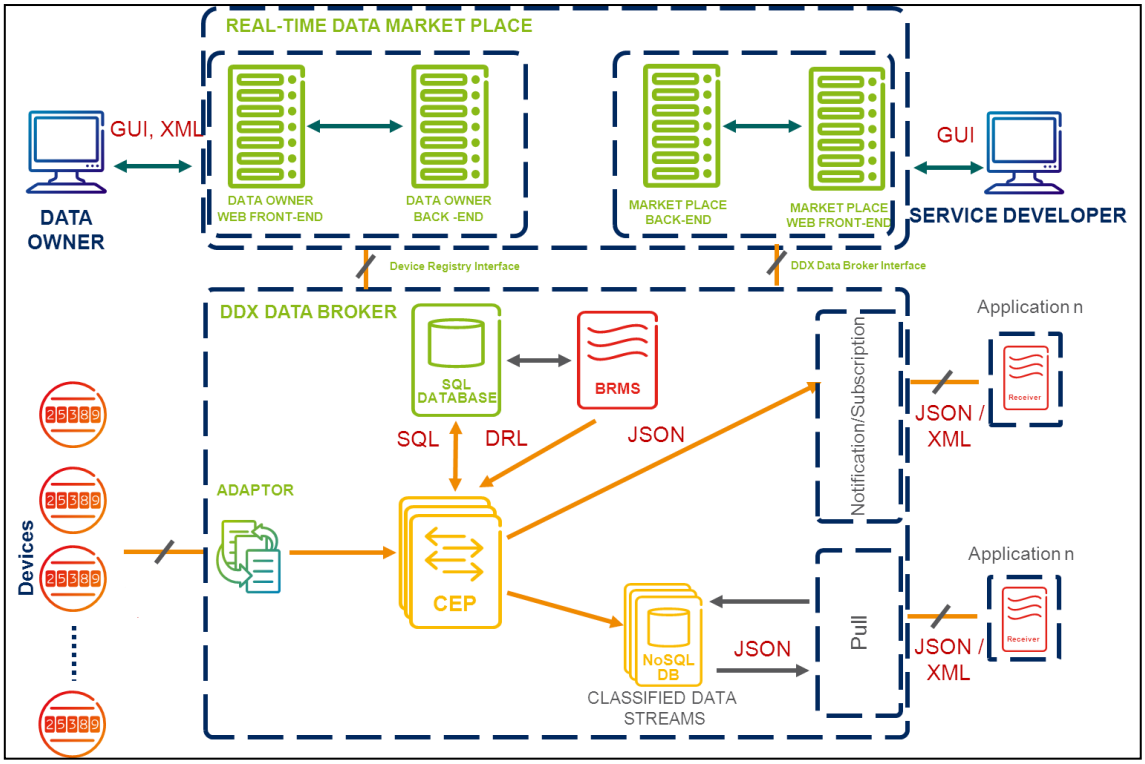


Figure 4: BUTLER DDX Low level architecture.

The set of horizontal back-end functional components included in the DDX Data Broker:

- Device Gateway consists of:
  - Connection Socket: IP address and port to connect the device's stream to the platform.
- A Complex Event Processing (CEP) engine:
  - Event processing functionality. Defines how the CEP engine processes the real-time information and how it becomes organized into packages ready to be offered to developers through the market place.
  - A Business Rule Management System (BRMS) integrated with the CEP engine and connected to the management interface. It enables the organization of the real-time data flows into packages; it also is used to define the business rules that will be used to define the real time data flows that will be stored in the NoSQL database.
- A No-SQL Database to cache relevant information coming from devices enabling subsequent delivery to customers.
- A SQL/Geodetic Database. It handles the storage of the information related to the location of the sensors and the different virtual entities that contains said sensors.
- A Notification/Subscription system to deliver information generated by the platform to external applications.

The following integration interfaces are defined in the Market Place:

- Device Registry. This interface enables the registration of new devices in the platform. The devices are registered in the Data Owner Portal and later on offered in the Dynamic Data Mall within packages.
- DDX Data Broker.
    - Provisioning:
        - It allows Developers to start receiving real time data.
        - It allows Developers to start consulting offline (stored) data.
    - Rules ingestion. It allows Developers to enrich the BUTLER DDX's knowledge base. A rule is composed of:
        - Identifier
        - A set of data sources
        - A set of conditions
        - A set of actions

3.3. Enabling Technologies

The figure below shows a list of the BUTLER DDX enabling technologies:

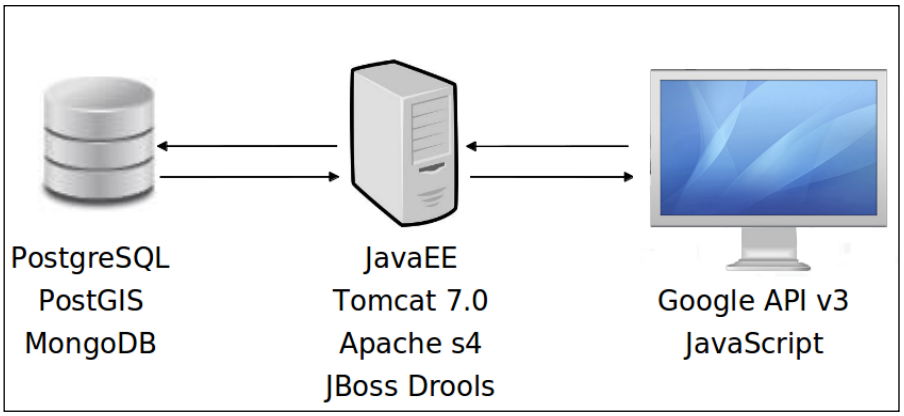


Figure 5: BUTLER DDX Tools Layer.

Next, a detailed list of the software tools and components is shown:

ELEMENT	SOLUTION	COMMENTS
Development Environment	Java J2EE JDK6 Apache Maven 3.0.5 Spring JQuery	Java J2EE JDK6 project built with Apache Maven for easy distribution of dependencies, and Spring for security and DB access. Maven is also used to define the platform structure and separate different functionalities of the platform in several loosely coupled modules. JQuery is used for the front-end to communicate asynchronously with Java Servlets deployed in Tomcat 7.
Databases	PostgreSQL PostGIS 2.03	For relational data storage, PostgreSQL is used. PostGIS for spatial and geographic support for the SQL database is used as well.
Complex Event Processing Engine	Apache S4 0.5.0 MongoDB 2.4.4 JBoss Drools Expert 5.5.0.Final Mosquitto 1.1.3	The CEP engine uses Apache S4 and JBoss Drools Expert to process data source's items, MongoDB to store them, and Mosquitto to control their dispatching to developers.
Web Server	Tomcat 7	There is a server machine running Tomcat 24/7 in the current development environment.
REST Services	Jersey	Open Source JAX-RS API framework used to develop the client part of the REST web services used to communicate the CEP engine with the rest of the platform.

Table 3 - BUTLER DDX Software tools and components list.

3.4. API Specification

BUTLER DDX provides five different APIs. Three of them are classic APIs, i.e. they are accessible programmatically. The remaining two differ from the three previous ones because of their graphical user interface availability. In order to access them it is mandatory to do it through the Real-Time Data Market Place.

3.4.1. No-GUI APIs

The first interface is an input interface through which the information sent from different BUTLER SmartObjects is received by the BUTLER DDX. Second and third ones are designed as output interfaces, the former, called Push interface, implements a pub/sub system, while the latter, called Pull interface, allows a sort of query system. Said interfaces enable applications to query information from SmartObjects or to receive information pushed by them.

3.4.1.1. Input interface

BUTLER DDX relies on a Complex Event Processing engine as main element for the brokerage of SmartObject data towards the applications. Every piece of data received from SmartObjects is injected into it in order to be analyzed.

BUTLER DDX exposes a public socket and is ready to receive every measure sent by any data source registered into the BUTLER DDX platform. The information must be sent in the following stream format:

```
DeviceId:SensorId Data
```

where `DeviceId` is the Device Identifier (an integer number that uniquely identifies each device generated when a new device is registered by the Device Owner); `SensorId` is the Sensor Identifier (an integer number that uniquely identifies each sensor generated when a new sensor is registered by the Device Owner); and `Data` is the integer or decimal number representing the information the sensor sends. Note that between `SensorId` and `Data` there is a space.

Device Owners are expected to implement a layer between their devices and the BUTLER DDX entry point to format the data their devices provide. They can know the device and sensor identifiers because the Real-Time Data Market Place displays that information in the details page.

3.4.1.2. Output (Push) interface

This output interface follows a Publication/Subscription paradigm in order to deliver to the applications those notifications they have been subscribed to. Two types of notifications are available:

- Notifications carrying information from sensors belonging to data offerings (packaged) already purchased by Service Developers;
- Notifications generated by rules created by the Service Developer.

The tool chosen for this task is Mosquitto, a MQTT message broker [24]. From the DDX Marketplace it is possible to download the Mosquitto Java client, in JAR format, in order to receive the notifications (com.ericsson.tbi.ddx.client-v1.0.jar) the Service Developer applications are entitled to. Since its usage is not trivial, a sample application with its sources is included as well. There is also a Developer Guide.

Here a sample application source code is available:

```
import com.ericsson.tbi.ddx.cep.base.Action;
import com.ericsson.tbi.ddx.cep.base.Measure;
import com.ericsson.tbi.ddx.client.ClientException;
import com.ericsson.tbi.ddx.client.INotificationListener;
import com.ericsson.tbi.ddx.client.Subscriptor;

public class Sample {

    /**
     * @param args
     */
   public static void main(String[] args) throws ClientException {
       Subscriptor subs = new Subscriptor(userId, "topicFile");
       subs.addNotificationListener(new NotificationListener());
    }

    private static class NotificationListener implements INotificationListener {

        public NotificationListener() {
        }

        @Override
    public void handleNotification(Action action) {
System.out.println(action);
//TO DO
        }

        @Override
    public void handleNotification(Measure msr) {
       System.out.println(msr);
//TO DO
        }
    }
}
```

First of all, the library com.ericsson.tbi.ddx.client-v1.0.jar must be added to the classpath. Then, the userId and the topic file are necessary. They are available in the BUTLER DDX web portal, at the Download client page. The userId is an integer number that identifies uniquely each Service Developer. The topic file is a binary file that contains the credentials needed to perform the subscription to the notifications the user has to receive.

In the moment the above parameters are configured, the application is ready to work. After connecting to the DDX Mosquitto instance, it will display the content of each notification coming from a data source included into a package (measure) and from Complex Event Processing (CEP) events (actions). The code can be altered in order to implement more complex functionalities.

3.4.1.3. Output (Pull) interface

This second output interface provides a query mechanism to retrieve data stored in a NoSQL database. This database stores the measures sent by the sensors in BSON format [25]. Example:

```
{
       "id" : NumberLong(28),
       "location" : {
              "name" : null,
              "latitude" : 40.39028,
              "longitude" : 3.628211,
              "altitude" : 0
       },
       "timestamp" : NumberLong("1367319143888"),
       "data" : 25.8,
```

```
        "units" : "%"
}
```

If the user has a Device Owner role, s/he is able to define storage rules in order to avoid an indiscriminate storage of his/her sensor's measures. When it comes to Service Developers, storage rules definition is not allowed, so this type of user would have to query the whole collection of data, with a predefined set of available operations.

In a similar way to that of the Push interface, a Java client has to be downloaded (com.ericsson.tbi.ddx.pull-client-vX.Y.jar), a sample application and a Developer Guide are available at the BUTLER DDX Market Place.

To select all the stored data belonging to a single sensor/data source:

```
public ResultSet selectAll(long id)
```

To select those stored data belonging to a single sensor/data source that matches the condition of being less than a given value.

```
public ResultSet selectLessThan(long id,float value)
```

To select those stored data belonging to a single sensor/data source that matches the condition of being greater than a given value.

```
public ResultSet selectGreaterThan(long id,float value)
```

To select those stored data belonging to a single sensor/data source that matches the condition of being less than a given value and greater than another given value.

```
public ResultSet selectBetween(long id,float value1,float value2)
```

To select those stored data belonging to a single sensor/data source that matches the condition of being equals to a given value.

```
public ResultSet selectEquals(long id,float value)
```

To select all the stored data from every sensor/data source located within a circle defined by a central point (coordinates) and its radius.

```
public ResultSet selectAllCloseTo(float latitude,float longitude,float radius)
```

To select all the stored data belonging to a single sensor/data source, received during a given time period.

```
public ResultSet selectBetweenTime(long id,long start,long end)
```

Here a sample application source code is available:

```
import com.ericsson.tbi.ddx.pull-client.NoSQLaccess;
```

```
import com.ericsson.tbi.ddx.pull-client.ClientException;
import com.ericsson.tbi.ddx.pull-client.ResultSet;

public class Sample {

    /**
     * @param args
     */
   public static void main(String[] args) throws ClientException {
        NoSQLaccess acc = new NoSQLaccess();
        ResultSet res = acc.selectAll("100");
        //TO DO
    }
}
```

First of all, it is necessary to add the library com.ericsson.tbi.ddx.pull-client-vX.Y.jar to the classpath. Then, a NoSQLaccess object must be instantiated. All the select functions can be called from that instance.

The object ResultSet can be accessed through an iterator, just like regular SQL ResultSet Java objects.

3.4.2. GUI APIs

3.4.2.1. Permanent storage rules

The GUI that manages this API is available at the Data Owner Portal, only visible for Data Owner users.By default the established rules store every single data item coming from the data sources, so this feature allow Data Owners to alter this behavior.

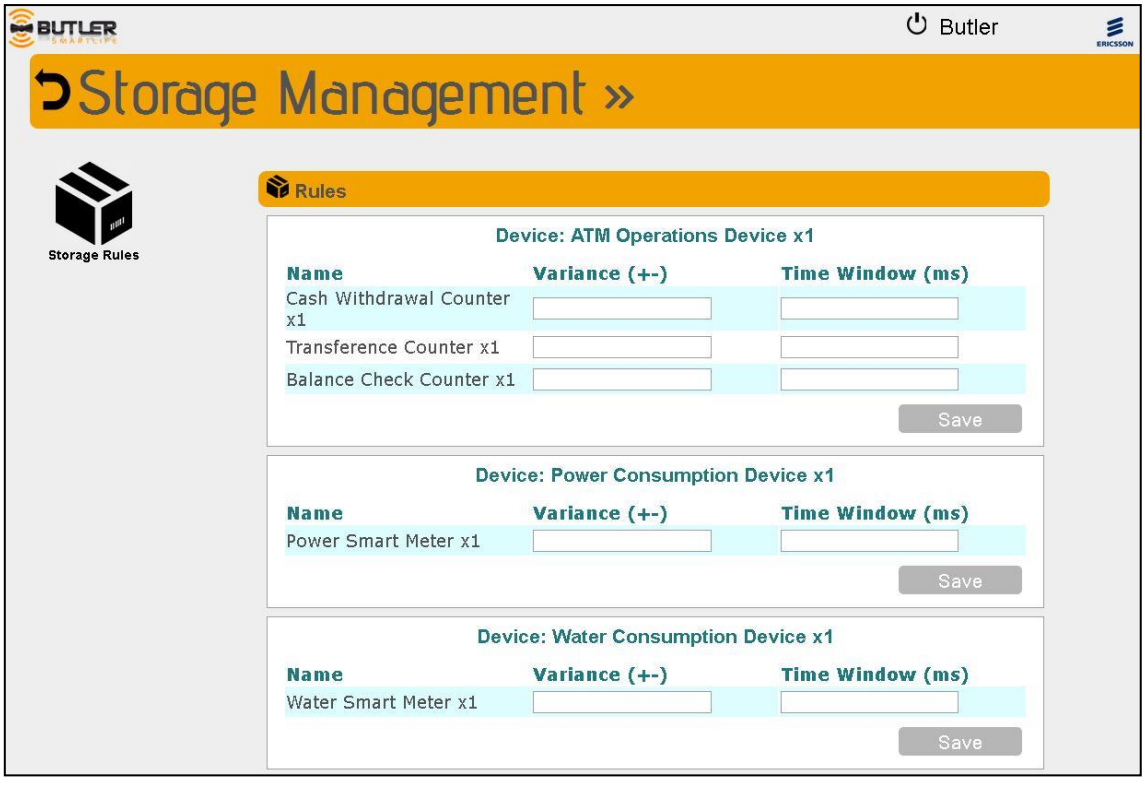


Figure 6: BUTLER DDX Storage Management Screen.

Two fields are displayed per each data source the Service Developer has acquired: Variance and Time Window. Variance is the value that acts as threshold between a data item and its predecessor to trigger the storage rule. Time Window is the time period (milliseconds) the data items will be stored. In order to clarify these concepts, we will explain all the possibilities using examples:

- Fill in the Variance field only: if we set 5, a data item will be stored only if its value is 5 or more units higher or lower than the previous data item's value received. This field works with absolute values, so negative numbers are not allowed.
- Fill in the Time Window field only: if we set 10000, the rate of data item storage is 10 seconds.
    - In the case the data source update rate is 5 seconds, only half of the data items will be stored. Two data items will arrive per time window and only the first one is stored.
    - In the case the data source update rate is 15 seconds; all the data items will be stored. Only one data item arrives per time window as maximum, and some time windows even receive any.
- Fill in both fields: Both rules will work together. If we combine both previous examples, Variance 5 and Time Window 10000, a data item will be stored if its value is 5 or more units higher or lower than the previous one or it is the first data item received within a 10 seconds time window.

3.4.2.2. Business Rules Management System

The GUI that manages this API is available at the Dynamic Data Mall, only visible for Service Developer users. The functionalities provided by this feature are supported by Apache S4 as Complex Event Processing engine and JBoss Drools Expert as rule engine.

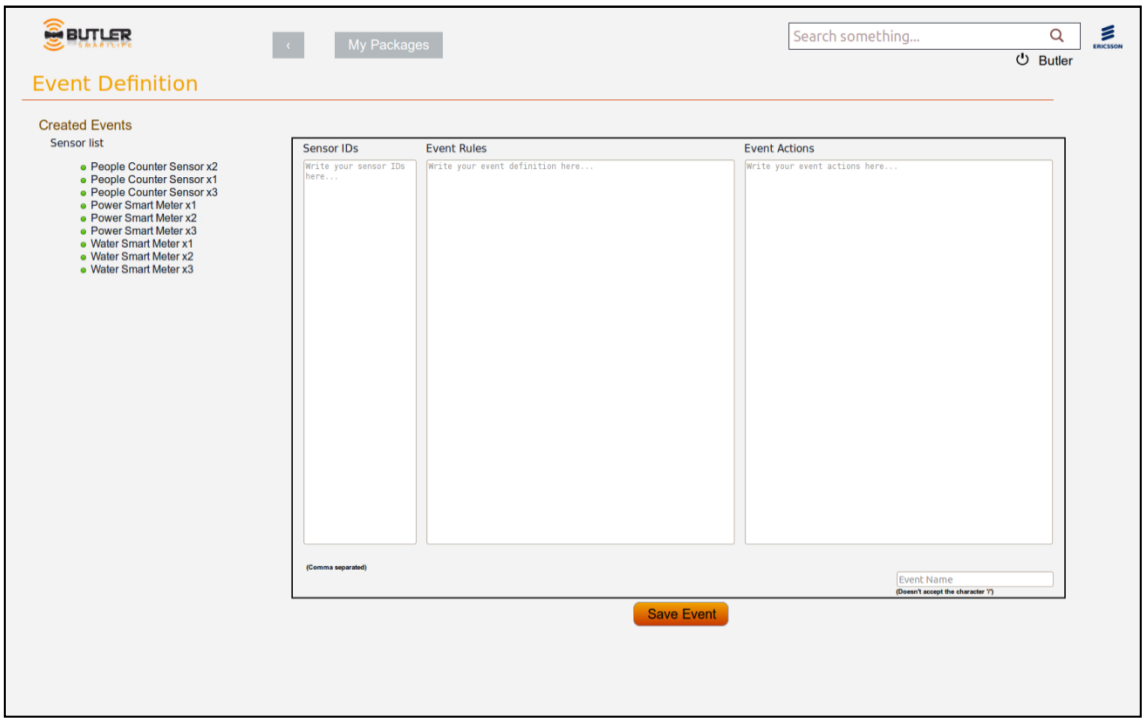


Figure 7: BUTLER DDX Event Definition Screen.

BUTLER DDX Business Events allow users to define sets of actions that will be triggered when a set of conditions are matched by the data items sent by the data sources the Service Developer is subscribed to. The GUI has four text boxes. The user must fill in all of them in order to define a business event.

- Sensors IDs

  This box must contain the IDs of the sensors aimed to participate in the event. Left screen pane shows a list of every data source's name the Service Developer is subscribed to. By clicking on them, their ids will appear into the box. For instance:

```
143:21, 162:31, 163:32
```

- Event Rules

  This box must contain the conditions to match. The format is the following:
  (variable_id : **Measure** ((condition_id) && (condition_data)))
  Within a condition a Boolean variable must be declared (its id can contain alphanumeric characters, but never start with a number) which takes the value of everything at the right side of the colons. Measure is a constant that represent the field of a data item that contains its value. Inside Measure we will work with two conditions, one for the id field and another for the data field. For example, a value for the condition_id field may be (id == 162:31) and for the condition_data field (data> 30). Note that both of them represent constants.
  Finally, operators could be: >, <, >=, <=, &&, ||, +, -, *, /
  Below there is an example of a rule consisting of two conditions. It is recommended to use this as a reference for generating new rules. The id values of the sensors ("167:231" and "167:232"), the conditions for the data field ("> 40" and "> 1000"), and the name of the variables ("temp" and "alt") shall be changed, of course.

```
(temp : Measure ( (id == 167:231) && (data > 40) ) )
&&
(alt : Measure ( (id == 167:232) && (data > 1000) ) )
```

- Event Actions

  In this field the user must enter the actions to be executed when the conditions introduced in section 2 are met. Currently, the platform only allows sending a notification to the user. Therefore, this field will only contain the description sent in the notification. An example would be:

```
Test1 event triggered with 'temp' & 'alt'
sensors
```

- Event Name

  This field defines the name for the event. The name must be comprised of alphanumeric characters. An example would be the following:

```
Test1
```

  Finally, if the rule was defined correctly, when the Save Event button is pressed, a message will prompt with the following text: “Event created successfully”. Otherwise, it will display: “Error. Could not store Event”.

For a deeper knowledge about Drools Expert programming, please check its official documentation [26].

3.4.3. Deployment

Every component described in this section can be found/accessed through the following URLs:

- Data Owner Portal: http://195.53.58.184:8080/blueberryButler
- Dynamic Data Mall: http://195.53.58.184:8080/marketplaceButler
- DDX Data Broker Input: 195.53.58.184:54444

The Real-Time Data Market Place server may request a pre-authentication step. The credentials are the following:

- User: ddx
- Password: ddxbutler

4. Localization Manager SmartServer

4.1. Functionalities

The main functionality of the Localization Manager (LM) is to estimate the positions of SmartObjects in real time as well as to provide these data to the other BUTLER components. This functional module is used to enable several BUTLER context-aware applications in different IoT domains, such us smart home, smart health, smart shopping and so on. For instance, within the smart home domain, the *multimedia follow-me* application uses the LM to play a selected multimedia content inside the room in which user is localized at that moment.

As showed in the block scheme depicted in Figure 8, the LM continuously (e.g., every 1 second) takes as input ranging measurements (e.g., RSSI, ToA, TDoA) from SmartObjects through the SmartObject Gateway and computes, in a centralized manner, the positions of the SmartObjects. As soon as the position of a SmartObject is updated, the LM publishes the estimated position as context data to a configured Context Manager endpoint. Moreover, the LM internally stores the positions history and makes them available for other BUTLER functional modules.

The LM guarantees a service that seamlessly operates across different vertical domains. In fact, the LM server ensures the interoperability across different communication technologies (e.g. UWB, ZigBee, NFC, CoAP, etc.), where the different protocols and communication standards are hidden by using as input a common ranging data format.

To sum up, the LM provides two main APIs based on RESTful HTTP:

- **Ranging API**: used to receive ranging measurements from the SmartObject Gateway
- **Localization API:** used to provide estimated positions to other BUTLER components

More details about these two APIs are provided online in the BUTLER API Catalogue [7].

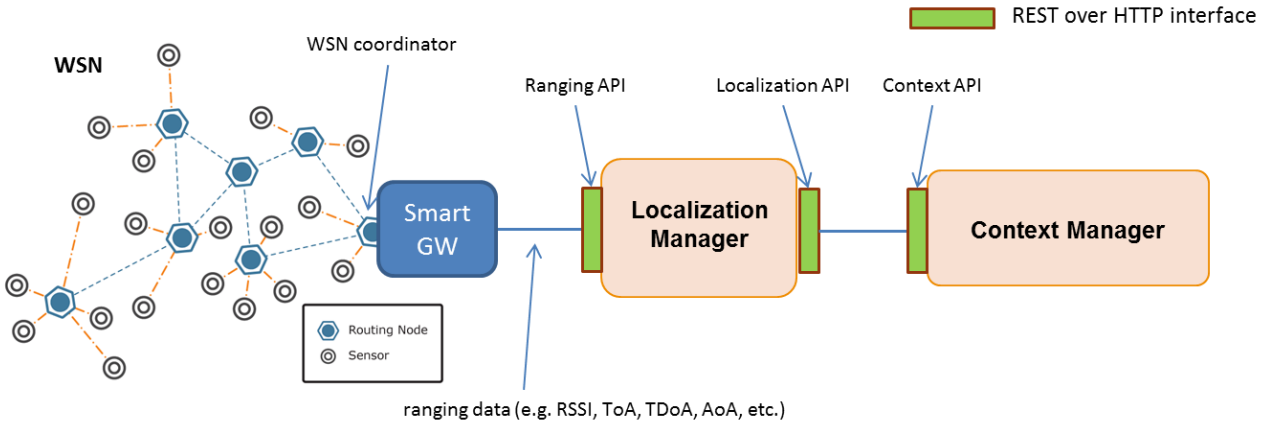


Figure 8: Block scheme of the Localization Manager.

4.2. Software Architecture

Figure 9 shows the architecture of the LM which is composed of four main components presented as follows.

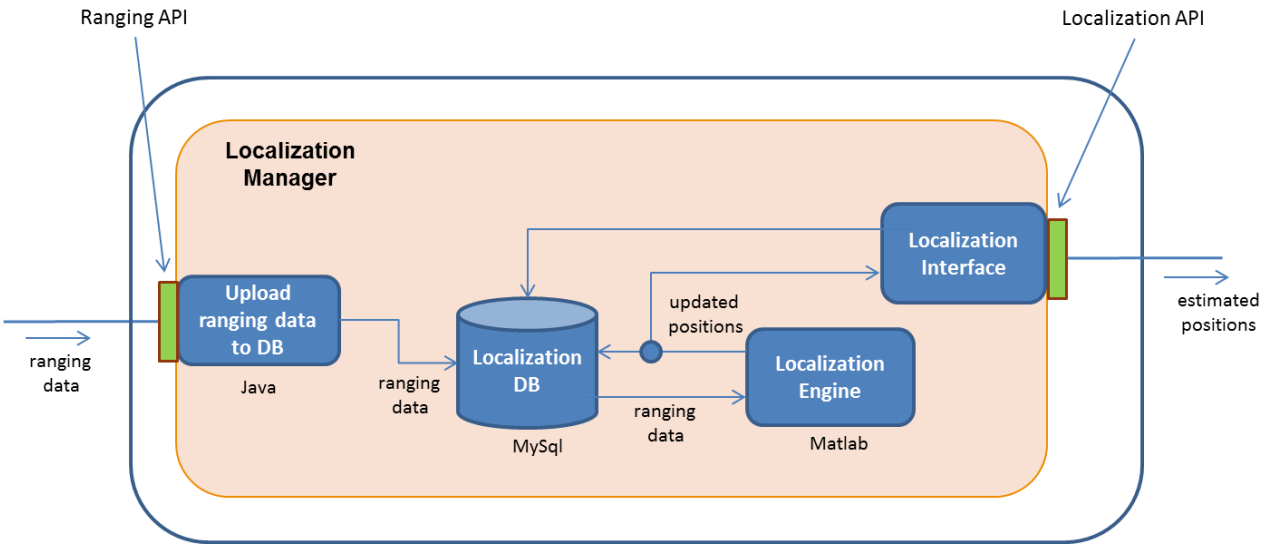


Figure 9: Localization Manager software architecture.

Upload ranging data to DB (URD2DB): this component, which is developed in Java, implements the interface between the LM and the SmartObject Gateway. It implements a HTTP REST interface (JSON parser) through which it receives ranging measurements from the SmartObject Gateway. The main role of the URD2DB is to convert HTTP REST commands into SQL commands compliant to the LM database scheme. Moreover, the URD2DB ensures the validity of the ranging data before being further processed by the other internal modules. Furthermore, the URD2DB is responsible of acknowledgement of all incoming HTTP REST commands either by successful ACK or error ACK.

Localization DB: this component is a relational database that is used to store the following data: SmartObjects information (e.g., SmartObject identifiers and coordinates of SmartObject anchors), ranging measurements and estimated positions. Note that both ranging and positioning data are time-stamped according to the UTC time reference.

Localization Engine (LE): this is the core component of the LM that is responsible to estimate the positions of the SmartObjects. The localization algorithm, which is implemented in MatLab, uses as input ranging measurements and anchors' coordinates that are stored in the database. This module estimates the positions with a periodicity that can be set according to the mobility of the SmartObject (e.g., 1 or 2 seconds). As soon as the position of a specific SmartObject is updated, the LE stores the position in the DB and sends it to the Localization Interface component. Note that positions are expressed according to a relative reference system (x, y, z) and then converted according to the absolute one (lat, long, height).

Localization Interface: this component, which is developed in Java, implements the interface between the LM and other BUTLER components that need as input estimated positions of SmartObjects. This module implements HTTP REST interface (JSON parser) through which it exposes estimated positions. In particular, it interacts with the localization DB to retrieve localization information about specific SmartObject in a given time. Moreover, as mentioned above, this module receives updated positions from the LE and forwards them to the Context Manager as context data.

4.3. Enabling Technologies

As presented in the deliverable D2.4 [1], in order to meet the horizontal requirements, different localization algorithms and services are seamlessly required from a combined geo and temporal information (smart transport) to accurate indoor positioning (smart shopping) and even to binary classification (e.g. in smart home, inside or outside a specific room, etc.). In order to accomplish such different, simultaneous and ubiquitous tasks, and at the same time to ensure the above mentioned features, the LE implements different localization algorithms. Within the LE there is a mechanism that selects the most suitable localization algorithm that achieves the required performance and self-adapts by tuning some parameters according to: the current available connectivity, type of ranging and associated accuracy, which are time variant and depend on the environment. So far, within the LE, it has been implemented the 'Energy Efficient Tracking' (EET) algorithm presented in detail in [2] [5], based on the Extended Kalman Filter (EKF) approach, and the interval scaling (I-SCAL) algorithm presented in [2] [6], designed to solve static positioning problem in centralized manner.

4.4. API Specification

The LM catalogue includes a set of three REST APIs interfaces. The first API is related to the interaction between the SmartObject Gateway and the LM. It is used by the GW to send ranging measurements to the LM. The second API concerns the interaction between the LM and other BUTLER components. It is used to provide absolute or relative estimated positions to other BUTLER components. The Third API is used to configure SmartObject anchors' coordinates in the LM. This API is used by an administrator to set the coordinates of the SmartObject anchors and the ID of all SmartObjects (anchors and mobiles). The first two APIs are presented in the following subsections while the third one is only presented in the BUTLER Catalogue [7] in order to limit the length of this deliverable.

4.4.1. Ranging API

It is used to receive ranging data from the BUTLER SmartObject Gateway.

4.4.1.1. Authentication

This method is not authenticated yet.

4.4.1.2. Signature

```
PUT    /localization/api/v1/{resourceType}/{smartObjectID}/ranging
```

4.4.1.3. Parameters

PARAMETERS	VALUE	DESCRIPTION	TYPE
resourceType	objects	Resource type	String
smartObjectID	104	Smart Object Device ID	String

4.4.1.4. Example

REQUEST

```
PUT /localization/api/v1/objects/104/ranging HTTP/1.1
Content-Type: application/json;charset=UTF-8

{
    "typeRanging": "RSSI",
    "smartGw_time_stamp": 8715446875,
    "rangeMeas": [
        {
            "Id_smartObject_B": "10",
            "meas_AB": -50.4,
            "meas_AB_accuracy": null,
```

```
            "meas_BA": -51.7,
            "meas_BA_accuracy": null
        },
        {
            "Id_smartObject_B": "14",
            "meas_AB": -64.3,
            "meas_AB_accuracy": null,
            "meas_BA": null,
            "meas_BA_accuracy": null
        },
        {
            "Id_smartObject_B": "19",
            "meas_AB": null,
            "meas_AB_accuracy": null,
            "meas_BA": -81.7,
            "meas_BA_accuracy": null
        }
    ]
}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
{
    "response": {
        "meta": {
            "v": "1.0.0",
            "status": "OK",
            "code": 200,
            "timestamp": "2013-10-20T08:47:12+02:00",
            "method": "sendRangingData"
        }
    }
}
```

4.4.2. Localization API

It returns the last estimated position of a SmartObject given the timestamp.

4.4.2.1. Authentication

This method is not authenticated yet.

4.4.2.2. Signature

```
GET  /localization/api/v1/{resourceType}/{smartObjectID}/localization/{typeLoc}/q?timeIn={timeIn}
```

4.4.2.3. Parameters

PARAMETERS	VALUE	DESCRIPTION	TYPE
resourceType	objects	Resource type	String
smartObjectID	104	Smart Object Device ID	String
typeLoc	abs or rel	It means relative or absolute estimated position	String
timeIn	1374850358	It means when the position was estimated	Double

4.4.2.4. Example

REQUEST

```
PUT    /localization/api/v1/objects/25957/localization/rel/q?timeIn=1374850358   HTTP/1.1
Content-Type: application/json;charset=UTF-8
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
{
    "smartObjectId": "25957",
    "timePosEst": 1374850357.852,
    "x_local_coord": 5.534,
    "y_local_coord": 2.428,
    "z_local_coord": 0,
    "accuracyHoriz": 0,
    "accuracyHeight": 0
}
```

4.4.3. Error codes

The following table shows the most common errors.

HTTP STATUS	CODE	DESCRIPTION
400 Bad Request	A00	The request is not valid
400 Bad Request	A01	Invalid date format
400 Bad Request	A02	Error in JSON validation
401 Unauthorized	B02	Could not get data
404 Not Found	D03	Resource not found
500 Internal Server Error	F00	Generic server error
500 Internal Server Error	F01	Error on inserting data

5. Renewables Energies Forecasting SmartServer

The Renewable Energies Forecasting SmartServer aims to compute both energy consumption and renewable energy generation predictions based on three different inputs: current consumption rates, current weather conditions and weather forecast. By gathering that information it is able to forecast the best and worst instants to connect to the electrical grid, considering the estimated price.

5.1. Functionalities

This SmartServer comprises five parts:

- Weather forecast data collection
- Smart meter data collection
- Data aggregation
- Energy consumption and generation forecast
- Results exposition

5.1.1. Weather forecast data collection

Every half an hour, a JSON file with a 10-day hourly weather forecast for the desired locations is downloaded from Weather Underground.[1] Predicted weather conditions are dumped into the relational database. Those files also contain current weather conditions, which are sent to the Complex Event Processing engine as events.

5.1.2. Smart meter data collection

Several groups of virtual smart meters are sending simulated energy consumption measures regularly from each location to the Complex Event Processing engine as events. It will be also possible to send detailed energy consumption information classified into several device types in the future.

5.1.3. Data aggregation

The Complex Event Processing engine recognizes two kinds of events: energy consumption and weather conditions. Its recurrent behavior works as follows:

- It opens a 30 minutes time window and analyzes every single event that is received during its length.
- If a weather condition event is received, the number of wind mills / solar panels installed in the location the weather conditions belong to is queried, and the power generation for that single moment is computed and stored into the relational database.
- If an energy consumption event is received, it is buffered until the time window is closed.
- Once the time window is closed, all the energy consumption events are aggregated considering their origin location, added and the result is stored into the database. Current weather conditions are also included.
- Then a new time window is opened.

5.1.4. Energy consumption and generation forecast

Using as input the stored historical energy consumptions, a process generates forecasted consumptions based on linear and multi-linear regressions prediction models. The independent values are the weather conditions, both past and forecasted.

[1] Weather forecasts are downloaded from Weather Underground, through their so-called Anvil plan for developers (no cost while the number of downloads per day does not get over 500). See <http://www.wunderground.com/weather/api>

On the other hand, it also computes forecasted energy generation. It takes as input the weather forecast stored into the relational database and the number of wind mills / solar panels installed in each location and computes the power generation.

Both results, consumption and generation, are stored into the relational database as well.

5.1.5. Results exposition

The service can be accessed through a web service that offers three functions per renewable energy type (wind and solar). Its logic is based in the assumption that a high renewable power production decreases the power price as well as a high power demand increases it.

- The first one computes the estimated time for maximum energy price within a period, expressed as an amount of hours from the present moment, for a specific location. It queries consumption and generation forecast for every hour within that period and, after comparing them, returns the moment when the difference between both values is the largest one (consumption(t) – generation(t)).
- The second one computes the estimated time for minimum energy price within a period, expressed as an amount of hours from the present moment, for a specific location. It queries consumption and generation forecast for every hour within that period and, after comparing them, returns the moment when the difference between both values is the smallest one (consumption(t) – generation(t)).
- The third function gauges the accuracy of the energy generation forecast. It compares past real-time generation data with past forecasted generation data for a past period in a specific location and computes the mean absolute percentage error.

5.2. Software Architecture

The following figure depicts the Renewable Energies Forecast SmartServer architecture:

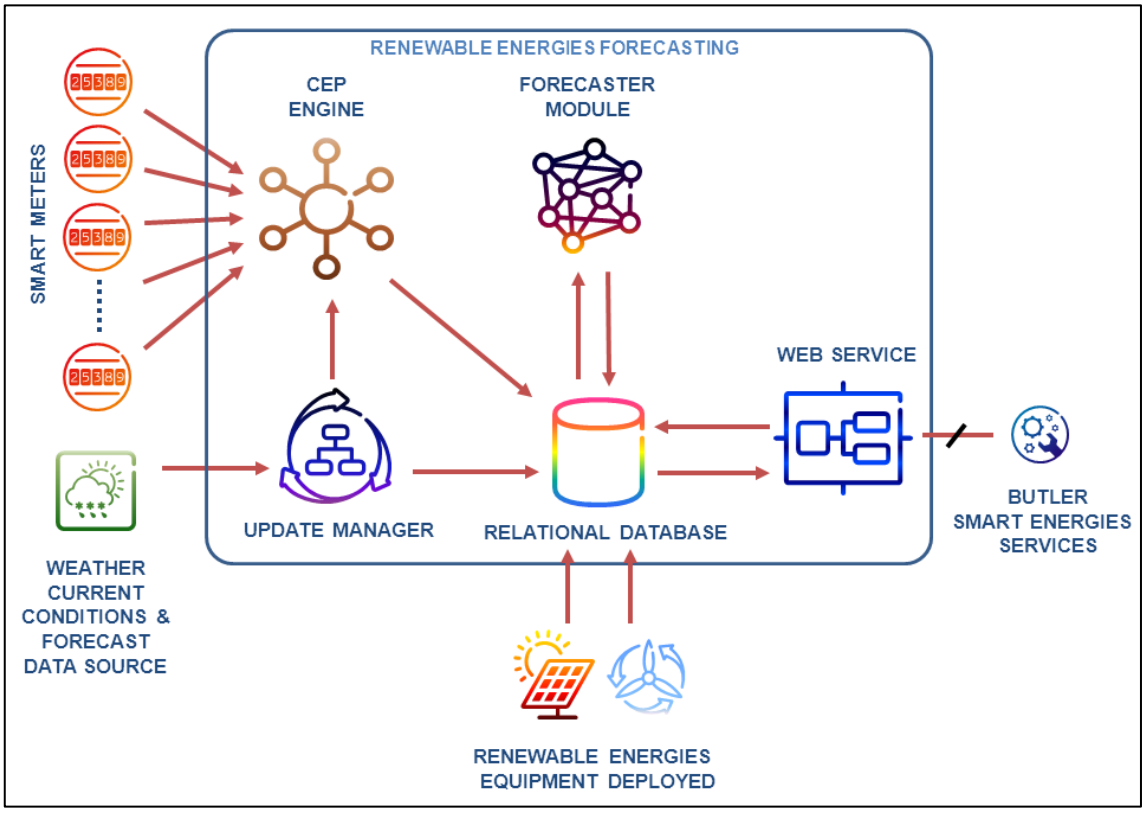


Figure 10: Renewable Energy Forecast architecture and workflow.

It consists of the following modules:

- Smart Meter simulators
  - Generate power consumption measures based on predefined patterns
  - Send them to the CEP engine
- Complex Event Processing engine
  - Receives events through a socket connection
  - Aggregates power consumption events
  - Performs simple power generation calculations
- Update Manager: it manages three threads
  - The first one downloads periodically current weather conditions and weather forecast JSON files
  - The second one generates events containing current weather conditions and sends them to the CEP engine
  - The third one dumps weather forecast data into the relational database
- Forecaster Module
  - Executes prediction models based on linear and multilinear regressions periodically
- Relational (SQL) database: it stores different kinds of data
  - Weather forecasts
  - Renewable energies equipment deployed

  - Real power consumptions[2]
  - Forecasted power consumptions[3]
  - Real power generations[4]
  - Forecasted power generations[5]
  - Smart meters' locations
- SOAP Web Service
  - Represents the interface through which it is possible to query results

5.3. Enabling Technologies

The technologies involved are: Complex Event Processing, relational database storage and Java SE/EE.

- Smart Meter simulators
  - Java SE
    - Threads
    - Sockets
- Complex Event Processing engine
  - Esper 4.6.0 [17]
  - Java SE
    - Sockets
    - JDBC
- Update Manager: it manages three threads
  - Java SE
    - Threads
    - JSON
    - JDBC
- Forecaster Module
  - OpenForecast 0.5.0 [27]
  - Java SE
    - Threads
- Relational (SQL) database: it stores different kinds of data
  - PostgreSQL 9.2 [28]
- Web Service
  - Java EE
  - Apache Axis web service [29]. It is deployed on an Apache Tomcat 7 server.

[2] Smart Meters are being simulated and their measures are so
[3] With respect to the so-called real power consumption data
[4] Both renewable energies equipment deployed and power generation algorithms are not based on real data
[5] With respect to the so-called real power generation data

5.4. API Specification

The API is implemented through an Apache AXIS SOAP Web Service. There are three available functions for each kind of renewable energy:

5.4.1. Wind Energy

- **getTimeOfMaxWindEnergyPrice**
  It returns the time for maximal €-price in the interval of X hours from the current time for the energy produced by the wind farms in a specific location.

5.4.1.1. Authentication

This method is not authenticated yet.

5.4.1.2. Signature

getTimeOfMaxWindEnergyPrice(int hours, String location)

5.4.1.3. Parameters

PARAMETERS	DESCRIPTION	TYPE
Hours	Length of the forecasting time window in hours (max 48h)	Integer
Location	Desired geographical location (city)	String

5.4.1.4. Example

REQUEST
Since this is a SOAP Web Service, it must be invoked from Java code

```
getTimeOfMaxWindEnergyPrice(24,“Zurich”);
```

RESPONSE
Date/time Java object.

```
2013 November 14, 20:30:00
```

(after being parsed to string)

5.4.1.5. Error codes

The date 1970 January 1, 00:00:00 means the calculation could not be performed.

- **getTimeOfMinWindEnergyPrice**
  It returns the time for minimal €-price in the interval of X hours from the current time for the energy produced by the wind farms in a specific location.

5.4.1.6. Authentication

This method is not authenticated yet.

5.4.1.7. Signature

getTimeOfMinWindEnergyPrice(int hours, String location)

5.4.1.8. Parameters

PARAMETERS	DESCRIPTION	TYPE
Hours	Length of the forecasting time window in hours (max 48h)	Integer
Location	Desired geographical location (city)	String

5.4.1.9. Example

REQUEST
Since this is a SOAP Web Service, it must be invoked from Java code.
```
getTimeOfMinWindEnergyPrice(24,“Zurich”);
```

RESPONSE
Date/time Java object.
```
2013 November 15, 05:30:00
```
(after being parsed to string)

5.4.1.10. Error codes

The date 1970 January 1, 00:00:00 means the calculation could not be performed.

- **getWindForecastConfidence**
  It returns the forecasting %-confidence of energy price produced by the wind farms in a specific location based in the data stored into the database during the last X hours.

5.4.1.11. Authentication

This method is not authenticated yet.

5.4.1.12. Signature

getWindForecastConfidence(int hours, String location)

5.4.1.13. Parameters

PARAMETERS	DESCRIPTION	TYPE
Hours	Length of the backwards time window in hours	Integer
Location	Desired geographical location (city)	String

5.4.1.14. Example

REQUEST
Since this is a SOAP Web Service, it must be invoked from Java code.
```
getWindForecastConfidence(24,“Zurich”);
```

RESPONSE
Double object
```
50.25
```

5.4.1.15. Error codes

0.0 and Infinity values mean the calculation could not be performed.

5.4.2. Solar Energy

- **getTimeOfMaxSolarEnergyPrice**
  It returns the time for maximal €-price in the interval of X hours from the current time for the energy produced by the solar farms in a specific location. This method is not available yet.

5.4.2.1. Signature

getTimeOfMaxSolarEnergyPrice(int hours, String location)

5.4.2.2. Parameters

PARAMETERS	DESCRIPTION	TYPE
Hours	Length of the forecasting time window in hours (max 48h)	Integer
Location	Desired geographical location (city)	String

5.4.2.3. Example

REQUEST
Since this is a SOAP Web Service, it must be invoked from Java code.

```
getTimeOfMaxSolarEnergyPrice(24,“Zurich”);
```

RESPONSE
Date/time Java object.

```
2013 November 15, 14:00:00
```

(after being parsed to string)

5.4.2.4. Error codes

The date 1970 January 1, 00:00:00 means the calculation could not be performed.

- **getTimeOfMinSolarEnergyPrice**
  It returns the time for minimal €-price in the interval of X hours from the current time for the energy produced by the solar farms in a specific location. This method is not available yet.

5.4.2.5. Signature

getTimeOfMinSolarEnergyPrice(int hours, String location)

5.4.2.6. Parameters

PARAMETERS	DESCRIPTION	TYPE
Hours	Length of the forecasting time window in hours (max 48h)	Integer
Location	Desired geographical location (city)	String

5.4.2.7. Example

REQUEST
Since this is a SOAP Web Service, it must be invoked from Java code.

```
getTimeOfMinSolarEnergyPrice(24,“Zurich”);
```

RESPONSE
Date/time Java object.

```
2013 November 12, 06:30:00
```

(after being parsed to string)

5.4.2.8. Error codes

The date 1970 January 1, 00:00:00 means the calculation could not be performed.

- **getSolarForecastConfidence**
  It returns the forecasting %-confidence of energy price produced by the solar farms in a specific location based in the data stored into the database during the last X hours. If the result is 0, try again with a smaller amount of hours. This method is not available yet.

5.4.2.9. Signature

getSolarForecastConfidence(int hours, String location)

5.4.2.10. Parameters

PARAMETERS	DESCRIPTION	TYPE
Hours	Length of the backwards time window in hours	Integer
Location	Desired geographical location (city)	String

5.4.2.11. Example

REQUEST

Since this is a SOAP Web Service, it must be invoked from Java code.

```
getSolarForecastConfidence(24,“Zurich”);
```

RESPONSE

Double object

```
40.75
```

5.4.2.12. Error codes

0.0 and Infinity values mean the calculation could not be performed.

5.5. Deployment

Currently only the Wind Energy Forecasting part is deployed. Solar Energy Forecasting features are still under development. The Wind Energy Forecasting SOAP web service is published in the URL http://195.53.58.184:8080/WindEnergyWebService/services/WindEnergy

The file needed in order to develop a client for this SOAP web service can be found at http://195.53.58.184:8080/WindEnergyWebService/wsdl/WindEnergy.wsdl

6. Energy Awareness Data SmartServer

6.1. Functionalities

Applications and services aiming to raise energy-awareness of their service users need a component that enables supports and simplifies energy data management. Energy Awareness Data Smart Server is a software application that, offering standard APIs, is in charge of the management of energy data.

Energy Awareness Data Smart Server exposes a set of RESTful APIs to allow applications and other platform components to get the details on device consumption based on energy data or even set it on their own. Users are also capable to manually control the actors (devices) through the offered API. Figure 11 shows Energy Awareness Data SmartServer high level architecture.

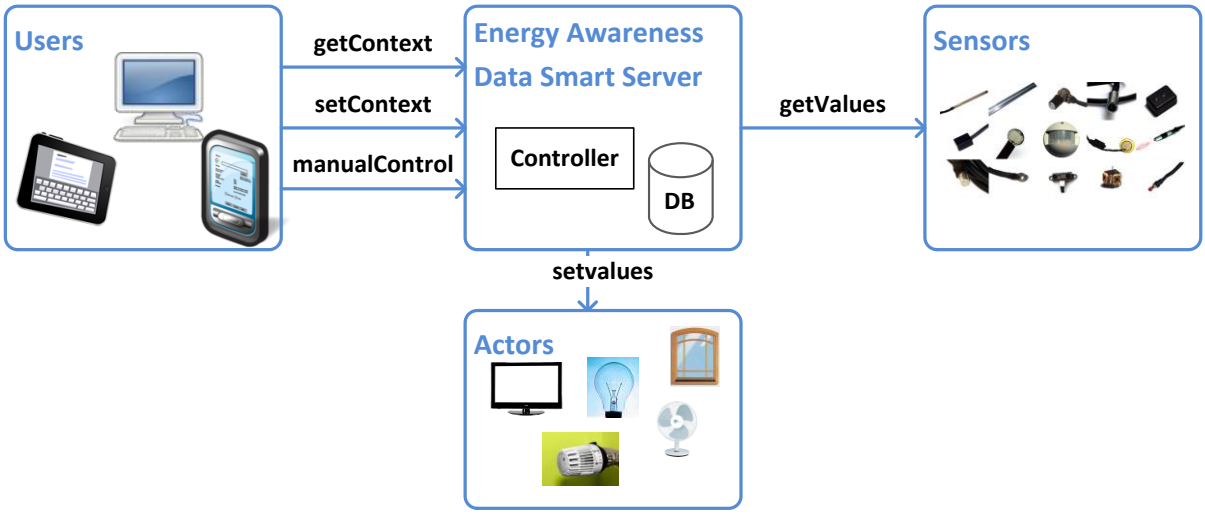


Figure 11: Energy Awareness Data high level architecture

All the functionalities provided by Energy Awareness Data SmartServer are protected according to BUTLER authentication framework, based on OAuth 2.0 [31].

6.2. Software Architecture

The Energy Awareness Data Smart Server is basically structured in a five-layer architecture. Including the User Interface and the sensors as a layer as well it is build up on a five-layer and three-tier architecture. The top layer of the Energy Awareness Data Smart Server is the controller of the graphical user interface called **UiController**. An interface is provided for adobe flash and HTML5 visualizations, where Flash communicates with an XML protocol and HTML5 with a Jackson [30]. In the second layer is the controller of our application. This layer only contains one software package called **EnergyController**. It collects data and calculates based on that a possible context for the user. The whole business logic is implemented in this single package.

On our third layer is a **Config** package. This basically controls the start-up behavior of the EnergyController and provides some behavior information, like which algorithm the controller has to use or which hardware is connected. This information is used to improve the performance of the application and to control the behavior in different environments. The **DataBase** collects all the states of sensors and the behavior of the user to provide different algorithm as much data as they need to do their job. In the package **Communication** is responsible for connecting all the different kind of data sources together. One of this data sources are **Web services**. They use the internet for collecting data. Examples of web services are

weather information or some sensor values from different locations. On the other end of a web services a **Webserver** is used. **Sensors** can also be plugged more or less direct to our server. We support an interface for the IEEE 802.15.4 standard, Ethernet, I2C or also very basic AD converter and digital input. The same kinds of interfaces are used in the package **Actors**. We are able to control different kind of loads. See Figure 12 for a schematic of our software architecture.

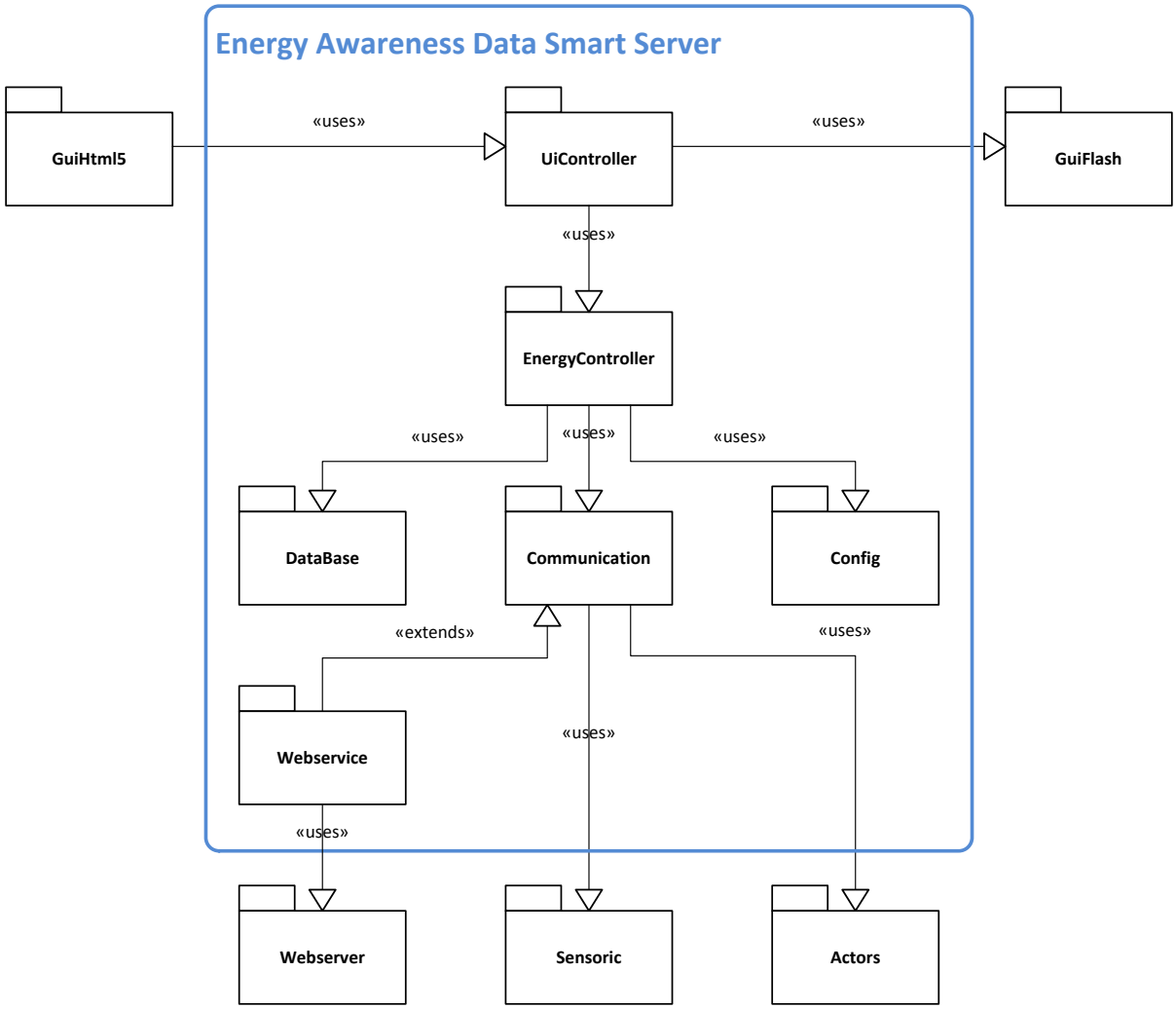


Figure 12: Energy Awareness Data software architecture

6.3. Enabling Technologies

The Energy Awareness Data Smart Server is a Java SE application. This environment has been selected to exploit its functionalities in terms of services, implementations and protocols and its characteristics of portability and scalability.

6.4. API Specification

This section provides signatures and some examples of the REST APIs offered by the Energy Data SmartServer.

6.4.1. AppliancesCatalogue

This method is used by an application to retrieve the complete catalogue of available appliances and their descriptions for the requested user.

6.4.1.1. Authentication

This method is protected according to OAuth 2.0 [31] protocol flows.

6.4.1.2. Signature

```
GET /energydata/{api_version}/catalogue
Content-Type: application/json;charset=UTF-8
```

6.4.1.3. Parameters

PARAMETERS	VALUE	DESCRIPTION	TYPE
api_version	v1	API Version	String

6.4.1.4. Example

REQUEST

```
GET    /energydata/v1/catalogue HTTP/1.1
Authorization: Bearer laskjdqweiunclsd19823eadhsjd
Content-Type: application/json;charset=UTF-8
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Body:
[
    {
        "name": "MyFan",
        "id": "a01",
        "state": "off",
        "category": "heating:fan",
        "icon": "../images/fan.png",
        "model": "XY-Mh1",
        "maxPower": 500,
        "avData": [
            "state",
            "power"
        ],
        "room": "home.studio",
        "brand": "Braun",
        "pdfURL": "../pdf/manualPhilips.pdf"
    },
    {
        "name": "MyLamp",
        "id": "a00",
        "state": "on",
        "category": "lighting:lamp",
        "icon": "../images/lamp.png",
        "model": "AVG-123",
        "maxPower": 50,
        "avData": [
            "state",
            "power"
        ],
```

```
        "room": "home.studio",
        "brand": "Philips",
        "pdfURL": "../pdf/manualPhilips.pdf"
    },
    {
        "name": "MyHeater",
        "id": "a02",
        "state": "on",
        "category": "heating:heater",
        "icon": "../images/heater.jpg",
        "model": "HH M 4",
        "maxPower": 1300,
        "avData": [
            "state",
            "power"
        ],
        "room": "chalet",
        "brand": "Samsung",
        "pdfURL": "../pdf/manualPhilips.pdf"
    }
```

6.4.2. GetAppliance

This method is used by an application to retrieve the complete description of an appliance of the requested user.

6.4.2.1. Authentication

This method is protected according to OAuth 2.0 protocol flows.

6.4.2.2. Signature

```
GET    /energydata/{api_version}/appliance/{appliance_id}
Content-Type: application/json;charset=UTF-8
```

6.4.2.3. Parameters

PARAMETERS	VALUE	DESCRIPTION	TYPE
api_version	v1	API Version	String
appliance_id	a02	Id of the requested appliance	String

6.4.2.4. Example

REQUEST

```
GET /energydata/v1/appliance/a02 HTTP/1.1
Authorization: Bearer laskjdqweiunclsd19823eadhsjd
Content-Type: application/json;charset=UTF-8
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Body:
{
    "id": "a02",
    "state": "on",
    "current": 0,
    "power": 1222.875764090471,
    "voltage": 0,
    "temp": 0
}
```

6.4.3. UpdateAppliance

This operation allows performing actions on an appliance updating one or more of its attributes.

6.4.3.1. Authentication

This method is protected according to OAuth 2.0 protocol flows.

6.4.3.2. Signature

```
PUT     /energydata/{api_version}/appliance/{appliance_id}
Content-Type: application/json
Body: see example in 6.4.1.4
```

6.4.3.3. Parameters

PARAMETERS	VALUE	DESCRIPTION	TYPE
api_version	v21f	API Version	String
appliance_id	a02	Id of the requested appliance	String

6.4.3.4. Example

This example shows how to switch off the appliance identified by id “a02” updating its “state” attribute.

REQUEST

```
POST /people/v1/@me/@self HTTP/1.1
Authorization: Bearer laskjdqweiunclsd19823eadhsjd
Content-Type: application/json;charset=UTF-8
Body:
{
    "state": "off"
}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Body – updated Appliance Resource:
{
    "id": "a02",
    "state": "off",
    "current": 0,
    "power": 0,
    "voltage": 0,
    "temp": 0
}
```

7. User Profile SmartServer

7.1. Functionalities

Applications that offer customized and personalized services to their users need a component that enables, supports and simplifies users' management. User Profile SmartServer is a software module that, offering standard APIs, is in charge of the management of users' profiles.

User Profile SmartServer exposes a set of RESTful APIs [1], [9] that are compliant with OpenSocial [10] specification and allows applications and other platform components to register, retrieve, update and delete user's profile data. Figure 13 shows User Profile SmartServer high level architecture.

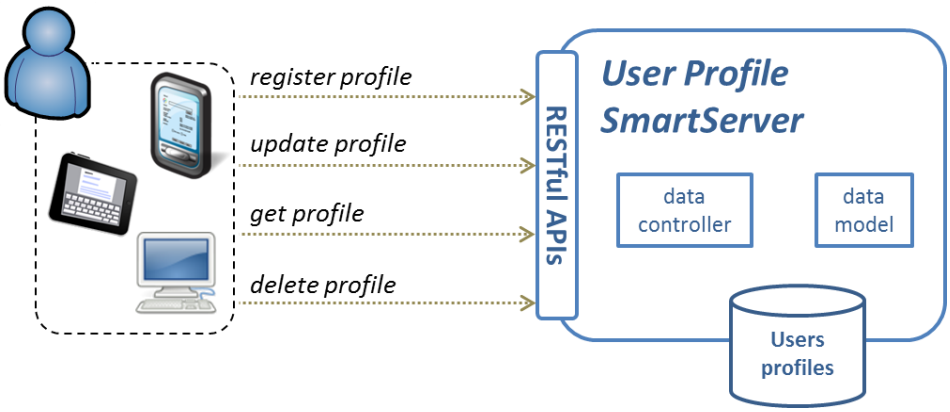


Figure 13: User Profile SmartServer high level architecture

All the functionalities provided by User Profile SmartServer are protected according to BUTLER authentication framework, based on OAuth 2.0 [31]. Using properly authorized tokens, applications can create new users profiles and read or update existing ones. The set of attributes that compose the profile of a user is extensible to adapt to different BUTLER applications needs.

The User Profile SmartServer is also in charge of maintaining the association between a user and his devices or smart objects, considering them as a "special" aspect of the profiling of a user even not actually part of his profile.

7.2. Software Architecture

The User Profile SmartServer is an enterprise application composed by several modules described in Figure 14. The component results in a single EAR, containing all modules, each one having a specific role in the system.

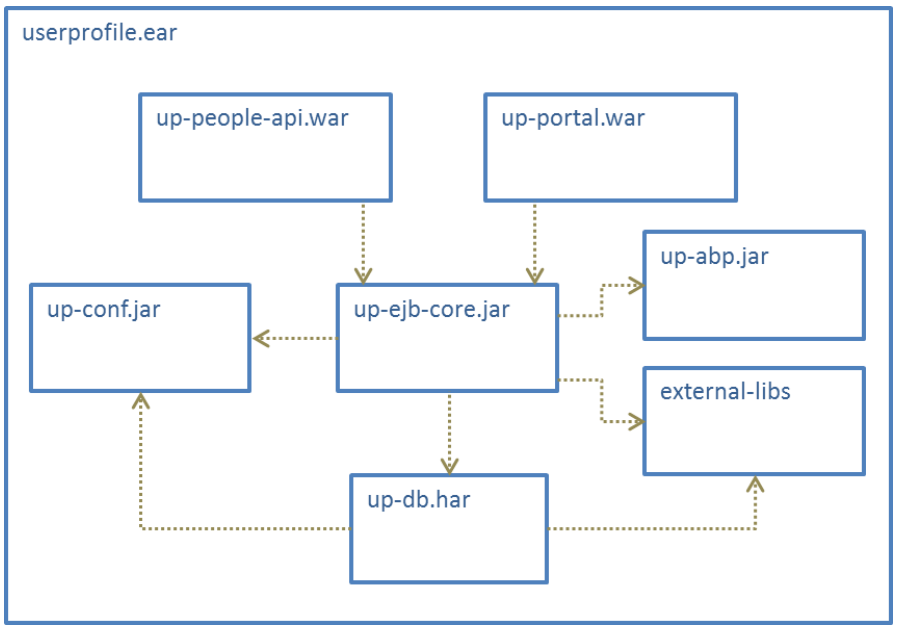


Figure 14: User Profile SmartServer software architecture

Following list details the software modules of User Profile SmartServer:

- *up-people-api.war* contains the web application archive of the User Profile SmartServer. It includes servlets to expose REST APIs according to OpenSocial data model, static pages to document the main functionalities and a set of JSPs to test them. This component is also in charge of the response format serialization (JSON vs. XML, etc.)
- *up-portal.war* is a support web application useful to test and debug backend functionalities
- *up-conf.jar* includes a set of configuration files
- *up-abp.jar* is an experimental component to manage user's contacts and relations
- *external-libs* is represented as a generic box containing all the external libraries used by the User Profile SmartServer
- *up-db.har* includes binaries and resources used to access and manage the underlying database
- *up-ejb-core.jar* is the core of the component. It includes User Profile SmartServer business logic implemented using enterprise beans, their classes (home, remote, local, and implementation), all the utility classes, and the deployment descriptors

7.3. Enabling Technologies

The User Profile SmartServer is a Java EE application [11]. This environment has been selected to exploit its functionalities in terms of services, implementations and protocols and its characteristics of portability and scalability. The chosen Java environment enabled also the usage of JAX-RS [12] as support in creating web services according to the REST architectural pattern. JAX-RS uses annotations, introduced in Java SE 5, to simplify the development and deployment of web service clients and endpoints.

The Data controller component is delegated to functions inside EJBs since there is a consistent part of the system that uses stateless session EJBs [13]. This allows decoupling data from controller logic and delegating to applications server critical functions (i.e. transaction managing and persistence, as persistence is Container Managed Persistence - CMP).

Data persistence relies on a relational database management system implemented on a MySQL engine [14]. User Profile SmartServer database consists of more than 25 tables; most important ones are those used to store users' profiles and relation between users and devices.

The mapping between an object-oriented domain model and the relational database is done using Hibernate [15], an object-relational mapping (ORM) library for the Java language. Its main feature is the mapping from Java classes to database tables (and from Java data types to SQL data types), also providing data query and retrieval facilities.

Finally, the User Profile SmartServer component is deployed on the JBoss Application Server [16].

7.4. API Specification

This section provides the signatures and some examples of REST APIs offered by the User Profile SmartServer.

7.4.1. RegisterProfile

This method is used by an application to create a new profile for an existing user.

7.4.1.1. Authentication

This method is protected according to OAuth 2.0 protocol flows.

7.4.1.2. Signature

```
POST   /people/{api_version}/@me/@self
Content-Type: application/json
Body: Person object as defined in OpenSocial Data Model [10]
```

7.4.1.3. Parameters

PARAMETERS	VALUE	DESCRIPTION	TYPE
api_version	v1	API Version	String

7.4.1.4. Example

REQUEST

```
POST /people/v1/@me/@self HTTP/1.1
Authorization: Bearer laskjdqweiunclsd19823eadhsjd
Content-Type: application/json;charset=UTF-8
Body:
{
    "id": "",
    "displayName": "Donald",
    "name": {
```

```
        "familyName": "Smith",
        "givenName": "Donald"
    },
    "phoneNumbers": [
        {
            "value": "+393331112222",
            "type": "mobile",
            "primary": true
        }
    ],
    "emails": [
        {
            "value": "myemail@test.com",
            "type": "business",
            "primary": true
        }
    ],
    "thumbnailUrl": " http://myhost.com/image.jpg"
}
```

RESPONSE

```
HTTP/1.1 201 Created
Content-Type: application/json;charset=UTF-8
Body - Person object as defined in OpenSocial Data Model [10] with a just generated “id”
attribute:
{
    "id": "dg24teg287346",
    "displayName": "Donald",
    "name": {
        "familyName": "Smith",
        "givenName": "Donald"
    },
    "phoneNumbers": [
        {
            "value": "+393331112222",
            "type": "mobile",
            "primary": true
        }
    ],
    "emails": [
        {
            "value": "myemail@test.com",
            "type": "business",
            "primary": true
        }
    ],
    "thumbnailUrl": " http://myhost.com/image.jpg"
}
```

7.4.2. GetProfile

This method is used by an application to retrieve an existing user profile.

7.4.2.1. Authentication

This method is protected is protected according to OAuth 2.0 protocol flows.

7.4.2.2. Signature

```
GET    /people/{api_version}/@me/@self
Content-Type: application/json
```

7.4.2.3. Parameters

PARAMETERS	VALUE	DESCRIPTION	TYPE

api_version	v1	API Version	String

7.4.2.4. Example

REQUEST

```
GET /people/v1/@me/@self HTTP/1.1
Authorization: Bearer laskjdqweiunclsd19823eadhsjd
Content-Type: application/json;charset=UTF-8
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Body - Person object as defined in OpenSocial Data Model [10]:
{
    "id": "dg24teg287346",
    "displayName": "Donald",
    "name": {
        "familyName": "Smith",
        "givenName": "Donald"
    },
    "phoneNumbers": [
        {
            "value": "+393331112222",
            "type": "mobile",
            "primary": true
        }
    ],
    "emails": [
        {
            "value": "myemail@test.com",
            "type": "business",
            "primary": true
        }
    ],
    "thumbnailUrl": " http://myhost.com/image.jpg"
}
```

7.4.3. UpdateProfile

This operation allows the update of an account profile. Profile data, both maintained and to be updated, is sent in the request body inside an OpenSocial Person object [10].

7.4.3.1. Authentication

This method is protected according to OAuth 2.0 protocol flows.

7.4.3.2. Signature

```
PUT    /people/{api_version}/@me/@self
Content-Type: application/json
Body: Person object as defined in OpenSocial Data Model [10]
```

7.4.3.3. Parameters

PARAMETERS	VALUE	DESCRIPTION	TYPE
api_version	v1	API Version	String

7.4.3.4. Example

This example shows the update of a profile changing the mobile phone number and adding a second personal mail address.

REQUEST

```
PUT /people/v1/@me/@self HTTP/1.1
Authorization: Bearer laskjdqweiunclsd19823eadhsjd
Content-Type: application/json;charset=UTF-8
Body:
{
    "id": " dg24teg287346",
    "displayName": "Donald",
    "name": {
        "familyName": "Smith",
        "givenName": "Donald"
    },
    "phoneNumbers": [
        {
            "value": "+393331114444",
            "type": "mobile",
            "primary": true
        }
    ],
    "emails": [
        {
            "value": "myemail@test.com",
            "type": "business",
            "primary": true
        },
        {
            "value": "mysecondemail@test.com",
            "type": "personal",
            "primary": false
        }
    ],
    "thumbnailUrl": " http://myhost.com/image.jpg"
}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Body – updated Person object as defined in OpenSocial Data Model [10]:
{
    "id": " dg24teg287346",
    "displayName": "Donald",
    "name": {
        "familyName": "Smith",
        "givenName": "Donald"
    },
    "phoneNumbers": [
        {
            "value": "+393331114444",
            "type": "mobile",
            "primary": true
        }
    ],
    "emails": [
        {
            "value": "myemail@test.com",
            "type": "business",
            "primary": true
        },
        {
            "value": "mysecondemail@test.com",
            "type": "personal",
            "primary": false
        }
    ],
    "thumbnailUrl": " http://myhost.com/image.jpg"
}
```

7.4.4. DeleteProfile

This operation allows deleting an existing user profile.

7.4.4.1. Authentication

This method is protected is protected according to OAuth 2.0 protocol flows.

7.4.4.2. Signature

```
DELETE /people/{api_version}/@me/@self
```

7.4.4.3. Parameters

PARAMETERS	VALUE	DESCRIPTION	TYPE
api_version	v1	API Version	String

7.4.4.4. Example

REQUEST

```
DELETE /people/v1/@me/@self HTTP/1.1
Authorization: Bearer laskjdqweiunclsd19823eadhsjd
```

RESPONSE

```
HTTP/1.1 200 OK
```

7.4.5. Error codes

HTTP STATUS	CODE	DESCRIPTION
400 Bad Request		The request is not valid
404 Not Found	P03	User profile not found
401 Unauthorized	P04	User profile is not active
401 Unauthorized	P05	User profile already exists
403 Forbidden	P10	The operation is forbidden for the current authenticated user
500 Internal Server Error	P99	Internal Server Error

8. User Behavior SmartServer

8.1. Functionalities

In BUTLER, we support activity recognition and behavior modeling through SAMURAI [21], an event-based stream mining SmartServer that integrates and exposes well-known software building blocks for complex event processing, machine learning and knowledge representation as RESTful services. The user behavior SmartServer (SAMURAI) provides the following key functionalities:

- **Feature extraction:** Convert raw data into features that are more meaningful for comparison (e.g. extracting high-level features such as walking, running, number of steps from raw accelerometer data)
- **Information fusion:** Aggregate data from different sources to increase the confidence in the quality of the inferred information (e.g. the current activity)
- **Domain knowledge:** Leverage background information to narrow down likely activities (e.g. semantically linking locations with different activity types)
- **Probabilistic correlations:** Identify frequent co-occurrences in event streams to derive event patterns of interest (e.g. spatio-temporal patterns)

The User Behavior SmartServer accepts inputs from the User Profile SmartServer and Localization Manager SmartServer, recognize user activities and updates the behavior information in the Context Manager SmartServer for future reference. In the context of BUTLER, the user behavior SmartServer is integrated into the diabetes SmartHealth Proof-of-Concept. For the diabetes application, the user behavior SmartServer continuously monitors a body-worn accelerometer stream for physical activity of the user and integrates it with other continuous context information such as user location stream from the Localization Manager SmartServer to infer the energy expenditure of the individual. The advantage of this approach is that the system offers personalized assistance, not only leveraging the results of the blood glucose sensor itself, but also basing its advice on the current context and anticipating what is likely going to happen next.

8.2. Software Architecture

Our event-based streaming architecture uses Jersey [20] to expose well-known software libraries for complex event processing (ESPER) [17], machine learning (Weka) [18] and knowledge representation (Parliament) [19] as RESTful services. These are illustrated in Figure 15. The basic primitive for all interaction with the framework are events. An event is represented as a set of typed key-value pairs that can be easily serialized into the JSON format. Our SmartServer offers high-level APIs that build upon the other components to implement activity and behavior awareness. Internally, these interfaces rely on a model to correlate time, locations and activities of an individual. Due to performance reasons and the streaming nature of the system, these learned models reside in memory only.

The architecture offers publish/subscribe capabilities to have clients (applications or subsystems) notified when particular (patterns of) events occur with push notifications implemented as REST callbacks (see details in following section). The architecture has two basic components to hold events:

- **Cache**: This simple general purpose component serves as a volatile in-memory container of events for use by other components or applications.
- **Event Queue**: Clients that do not support push notifications through REST callbacks can register a queue to hold events and poll that instead.

Their RESTful APIs follow the CRUD mapping on HTTP methods to create (POST), read (GET), update (UPDATE) or delete (DELETE) events.

Our architecture builds upon the Esper5 library for complex event processing (CEP). Esper usually relies on Java POJOs [32] to represent events at compile time. However, in our IoT ecosystem new event types can be created anytime. We therefore expose a RESTful API to dynamically register new event types at runtime. Similarly, building blocks for other intelligent functionalities (Weka for classification/clustering and Parliament for semantic localization).

The system does not offer any generic persistence features for past events or activities (these will be stored in the context exposition functional component). The actual component is a WAR file deployed on an Apache Tomcat 7.0 application server. These APIs and corresponding REST interfaces are described in the following sub-sections. For more details on the architecture refer to the official page of SAMURAI [21].

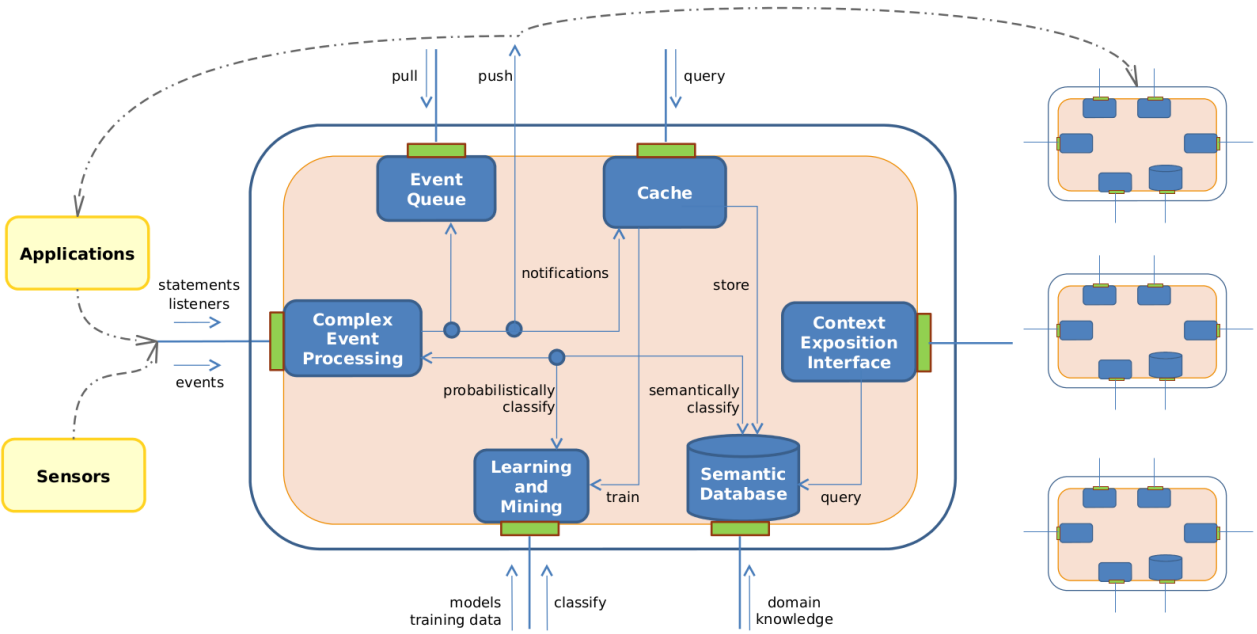


Figure 15: A conceptual overview of the user behavior SmartServer architecture

8.3. Enabling Technologies

ESPER:

The complex event processing is supported by a component which exposes ESPER library as RESTful interfaces. It processes event streams on the fly and we use it for two purposes:

- Feature extraction from low-level events (e.g. from accelerometer to steps)
- Publish/subscribe interaction with applications or SAMURAI subsystems.

WEKA:

Weka is a collection of machine learning algorithms implemented in Java and is integrated in the SAMURAI SmartServer to provide support for classification through machine learning that goes beyond semantic classification which relies on predefined knowledge and ontologies.

Parliament:

A version of semantic localization component is implemented using Parliament: a BBN Technologies built triple store and reasoner for semantic web. It supports GeoSPARQL, the newly adopted OGC standard for geospatial semantic data. A proof-of-concept implementation of a smart home environment for qualitative spatial reasoning about the location of the user with in a flat and possible set of activities in that location has been developed using the framework.

Jersey:

All the enabling technologies described above are integrated and exposed as a single RESTful interface using Jersey. Jersey RESTful Web Services framework is open source, production quality framework for

developing RESTful Web Services in Java that provides support for JAX-RS APIs and serves as a JAX-RS (JSR 311 & JSR 339) Reference Implementation [20].

8.4. API Specifications

8.4.1. Get Activity

This allows the user or application to specify a known user id, his or her location information, a timestamp and a threshold for the minimum confidence in the prediction of the activity and get a list of activities.

8.4.1.1. Authentication

This method is not authenticated yet.

8.4.1.2. Signature

```
GET    /activities/activity?userid={userid}&location={location}&threshold={threshold}
```

8.4.1.3. Parameters

PARAMETERS	VALUE	DESCRIPTION	TYPE
userid	davypreuveneers	Id of the user	String
location	kitchen	Known/predefined semantic location description, e.g. Kitchen	String
threshold	0.15	Confidence value for prediction	Number
time	2013-10-28T12:15:23+01:00	Time of the day in the Java simple date format: "yyyy-MM-dd'T'HH:mm:ss+00:00"	String

8.4.2. Get Activity Confidence

This allows the user or application to specify a known user id, his or her location, a timestamp and the anticipated user activity and get the confidence in the prediction of the activity being performed by the user.

8.4.2.1. Authentication

This method is not authenticated yet.

8.4.2.2. Signature

```
GET    /activities/activityconfidence?userid={userid}&activity={activity}&location={location}
```

8.4.2.3. Parameters

PARAMETERS	VALUE	DESCRIPTION	TYPE
userid	davypreuveneers	Id of the user	String
location	Kitchen	Known/predefined semantic location description, e.g. Kitchen	String
activity	Clean	Activity of the user which is of interest	String
time	2013-10-28T12:15:23+01:00	Time of the day in the Java simple date format: "yyyy-MM-dd'T'HH:mm:ss+00:00"	String

8.4.3. Get Semantic Location

This offers a way to translate geo-location information of a user into semantically more meaningful information.

8.4.3.1. Authentication

This method is not authenticated yet.

8.4.3.2. Signature

```
GET /locations/semanticlocation?relativeRefSystem={relativeRefSystem}&x={x}&y={y}
```

8.4.3.3. Parameters

PARAMETERS	VALUE	DESCRIPTION	TYPE
relativeRefSystem	Home_davypreuveneers	Name of the map for reference	String
x	3.5	X coordinates in the map	Number
y	4.5	Y coordinates in the map	Number
accuracy	0.2	The accuracy of the location estimation by the respective localization systems	Number

8.4.4. Get Steps

This allows the user or application to specify a user id, the date and time period when the number of steps is required

8.4.4.1. Authentication

This method is protected according to OAuth 2.0 protocol flows.

8.4.4.2. Signature

```
GET /activities/steps?userid={userid}
```

8.4.4.3. Parameters

PARAMETERS	VALUE	DESCRIPTION	TYPE
userid	davypreuveneers	Id of the user	String
date	2013-10-16	Date of interest (YYYY-MM-DD)	String
period	morning	period of interest: earlymorning 04-06 hrs morning 06-12 hrs noon 12-13 hrs afternoon 13-17 hrs evening 17-20 hrs night 20-24 hrs latenight 24-04 hrs	String

8.4.5. Get Calorie Expenditure

This allows the user or application to specify a user id, the date and time period when the number of calories spent information is required by the user/application.

8.4.5.1. Authentication

This method is not authenticated yet.

8.4.5.2. Signature

```
GET /activities/calorieexpenditure?userid={userid}
```

8.4.5.3. Parameters

PARAMETERS	VALUE	DESCRIPTION	TYPE
userid	davypreuveneers	Id of the user	String
date	2013-10-16	Date of interest (YYYY-MM-DD)	String
period	morning	period of interest: earlymorning 04-06 hrs morning 06-12 hrs noon 12-13 hrs afternoon 13-17 hrs evening 17-20 hrs night 20-24 hrs latenight 24-04 hrs	String

8.4.6. Error Codes

HTTP STATUS	CODE	DESCRIPTION
200		Successful HTTP request
500		Unsuccessful HTTP request with corresponding error message
404		Requested statement (operation) not found
409		Requested listener/model already present with corresponding error message

9. Context Manager SmartServer

9.1. Functionalities

The Context Manager SmartServer is the component in charge of the management of context information about users and objects. It mainly works as:

- an aggregator of raw context data coming from several *Context Sources (CS)*;
- a generator of higher level context information obtained invoking specific external *Context Providers (CP)*;
- a producer of aggregated context data, related both to users and other entities in the space (e.g. various smart objects), made available to external *Context Consumers (CC)*.

Figure 16 shows a high level architecture of the Context Manager SmartServer, where an exposure layer offers to Context Sources and to Context Consumers a set of RESTful APIs over HTTP protocol. Behind this layer, the core of the Context Manager SmartServer is implemented by a set of internal sub-components that also interacting with external systems, contribute to properly serve the requests.

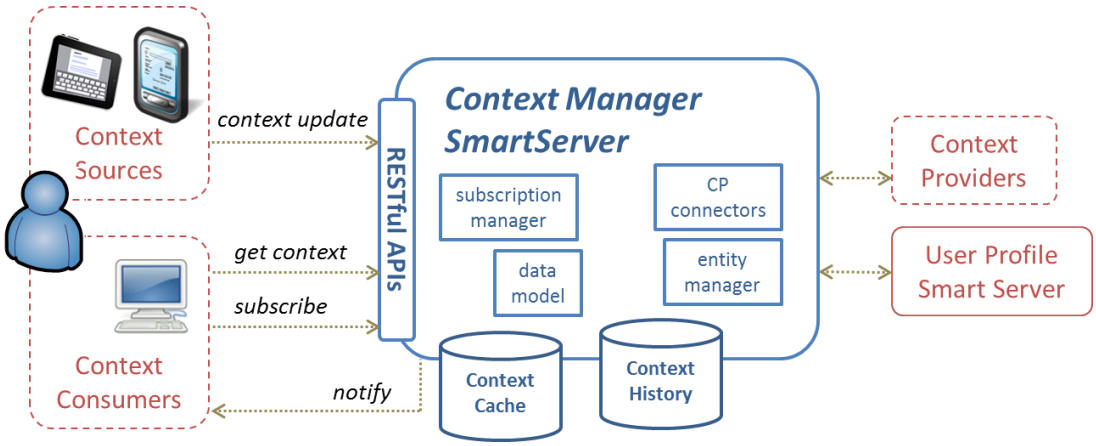


Figure 16: Context Manager SmartServer high level architecture

More in detail, a Context Source is a component that pushes context information about one or more context domains, in asynchronous mode. A CS sends context data according to its internal logic and never due to an explicit invocation from another component. A Context Provider is a module that provides context information in synchronous mode; it never sends data to other components in asynchronous mode. The Context Manager SmartServer can invoke it to obtain new context information the CP is able to provide computing input context data already available. A Context Consumer is an entity (e.g. a context based application) that needs context data; it can obtain it sending explicit requests related to specific context domains to the Context Manager SmartServer or even subscribing itself to specific events on context information.

Context information managed by Context Manager SmartServer always refers to an entity (the subject the context refers to, e.g. a user or an object) and to a domain (a partition of context data composed by a set of meaningful attributes, e.g. position domain is composed by latitude, longitude and accuracy attributes). A specific context domain has a validity after which data expire. Until the expiration date, data are stored in a Context Cache and are available to be used both as input for Context Providers and as response for Context Consumers requests. After the expiration, context data are moved in another storage - a Context History - where they are kept for a configurable amount of time (i.e. 30 days) and retrievable to reply to requests on past context information.

Connections between different entity types are managed exploiting the support of the User Profile SmartServer that stores the relations between users and their devices. As example consider a user identified by nick “donald” that has configured on User Profile two smart objects (obj1 and obj2) as his devices. These objects, as Context Sources, send position context data related respectively to entity obj1 and obj2 to the Context Manager SmartServer while a Context Consumer is interested in donald position. The Context Manager SmartServer, helped by User Profile SmartServer, deals with these relations, managing if necessary conflicting information, and providing to the interested consumer donald context regardless the sources that actually published it.

9.2. Software Architecture

The Context Manager SmartServer is an enterprise application composed by several modules described in Figure 17. The component results in a single EAR, containing all modules, each one having a specific role in the system.

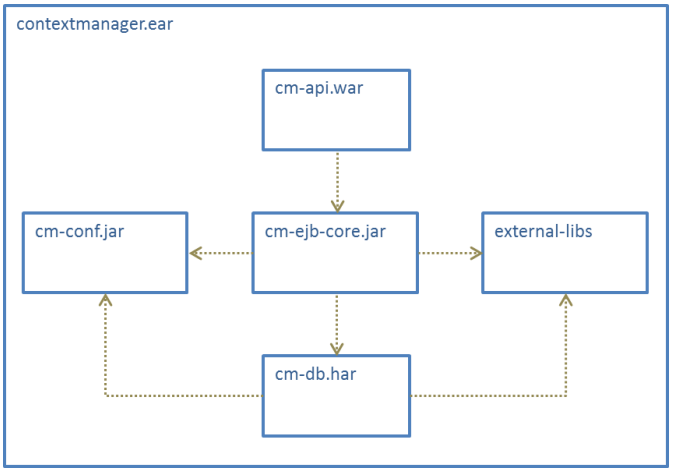


Figure 17: Context Manager SmartServer software architecture

Following list details the software modules of Context Manager SmartServer:

- *cm-api.war* contains the web application archive of the Context Manager SmartServer. It includes servlets to expose REST APIs, static pages to document the main functionalities and a set of JSPs to test them. This component is also in charge of the response format serialization (JSON vs. XML, etc.)
- *cm-conf.jar* includes a set of configuration files
- *external-libs* is represented as a generic box containing all the external libraries used by the Context Manager SmartServer
- *cm-db.har* includes binaries and resources used to access and manage the underlying database
- *cm-ejb-core.jar* is the core of the component. It includes Context Manager SmartServer business logic implemented using enterprise beans, their classes (home, remote, local, and implementation), all the utility classes, and the deployment descriptors

9.3. Enabling Technologies

The Context Manager SmartServer is a Java EE application [11]. This environment has been selected to exploit its functionalities in terms of services, implementations and protocols and its characteristics of portability and scalability. The chosen Java environment enabled also the usage of JAX-RS [11] as support in creating web services according to the Representational State Transfer (REST) architectural pattern. JAX-RS uses annotations, introduced in Java SE 5, to simplify the development and deployment of web service clients and endpoints.

Data persistence is guaranteed by a relational database management system implemented on a MySQL engine [14]. Context Manager SmartServer database consists of 14 tables; most important ones are those used to store current and historical context data and their schema looks like the following:

```
tb_cache (
  `ID` bigint(20) NOT NULL auto_increment,
  `ENTITY` varchar(255) collate utf8_bin NOT NULL,
  `DOMAIN` bigint(20) NOT NULL,
  `TS` bigint(20) NOT NULL,
  `EXP` bigint(20) NOT NULL,
  `DATA` mediumtext collate utf8_bin NOT NULL
)
```

Other main tables are used to manage subscriptions to context events and relations between supported entities.

The mapping between an object-oriented domain model and the relational database is done using Hibernate [15], an object-relational mapping (ORM) library for the Java language. Its main feature is the mapping from Java classes to database tables (and from Java data types to SQL data types), also providing data query and retrieval facilities.

The Context Manager SmartServer is deployed on JBoss Application Server [16].

9.4. API Specification

This section provides the signatures and some examples of REST APIs offered by the Context Manager SmartServer.

9.4.1. ContextUpdate

This method is used by a Context Source to publish context data belonging to a specific domain and related to a specific entity.

9.4.1.1. Authentication

This method is protected according to OAuth 2.0 protocol flows.

9.4.1.2. Signature

```
POST /context/{api_version}/{entityType}/{entityId}/{domain}
Content-Type: application/json
Body: see example in 9.4.1.4.
```

9.4.1.3. Parameters

PARAMETERS	VALUE	DESCRIPTION	TYPE
api_version	v21f	API Version	String
entityType	user	Entity type (“user”|“imei”)	String
entityId	donald	Entity identifier (username, object id, etc.)	String

domain	position	Domain of context data	String

9.4.1.4. Example

REQUEST

```
POST   /context/v1/user/donald/position HTTP/1.1
Authorization: Bearer laskjdqweiunclsd19823eadhsjd
Content-Type: application/json;charset=UTF-8
Body:
{ "data": {
        "ctxEls": [
            {
                "ctxEl": [
                    {
                        "scope": "position",
                        "entity": {
                            "id": "donald",
                            "type": "username"
                        },
                        "contextProvider": {
                            "id": "LocalizationManager",
                            "v": "1.0"
                        },
                        "timestamp": "2013-10-08T12:12:36+02:00",
                        "expires": "2013-11-08T19:00:00+02:00",
                        "dataPart": {
                            "longitude": 3.2316884682,
                            "latitude": 5.913486456
} } ] } ] } }
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Body:
{ "response": {
        "meta": {
            "v": "1.0.0",
            "status": "OK",
            "code": 200,
            "timestamp": "2013-10-22T14:53:49+02:00",
            "method": "contextUpdate"
} } }
```

9.4.2. GetContext

This method is used by a Context Consumer to retrieve context data belonging to a specific domain and related to a specific entity.

9.4.2.1. Authentication

This method is protected according to OAuth 2.0 protocol flows.

9.4.2.2. Signature

```
GET   /context/{api_version}/{entityType}/{entityId}/{domain}?since={since}&until={until}&at={at}
Content-Type: application/json
```

9.4.2.3. Parameters

PARAMETERS	VALUE	DESCRIPTION	TYPE
api_version	v21f	API Version	String
entityType	user	Entity type (“user”|“imei”)	String
entityId	donald	Entity identifier (username, object id, etc.)	String

domain	position	Domain of context data	String
since	1w	Initial timestamp of period of interest (only for query on past context). Optional: it can be xs:DateTime or expressed in number of weeks (*n*w) or days (*n*d)	String
until	-1d	Final timestamp of period of interest (only for query on past context). Optional: it can be xs:DateTime or expressed in number of weeks (*n*w) or days (*n*d). Default is now.	String
at	2013-10-22T15:03:37+02:00	Specific moment in the past the context data has to refer to	String

9.4.2.4. Example

REQUEST

```
GET /context/v1/user/donald/position HTTP/1.1
Authorization: Bearer laskjdqweiunclsd19823eadhsjd
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Body:
{ "response": {
        "meta": {
            "v": "1.0.0",
            "status": "OK",
            "code": 200,
            "timeRef": "2013-10-22T15:03:37+02:00"
        },
        "result": {
            "ctxEls": [
                {
                    "ctxEl": [
                        {
                            "contextProvider": {
                                "id": "Localization Manager",
                                "v": "1.0"
                            },
                            "entity": {
                                "type": "username",
                                "id": "donald"
                            },
                            "scope": "position",
                            "timestamp": "2013-10-22T14:53:49+02:00",
                            "expires": "2013-11-22T21:41:13+02:00",
                            "dataPart": {
                                "longitude": 3.2316884682,
                                "latitude": 5.913486456
} } ] } ] } } }
```

10. MultiMedia SmartServer

10.1. Functionalities

Smart Objects like televisions, that can play audio and video streams, need a common and coherent view of multimedia contents available for playing.

The MultiMedia SmartServer aims at indexing every multimedia content, making it available to any BUTLER-enabled device (typically a television). The content itself may be either stored locally in the SmartServer, or streamed from a network location (being internet or LAN).

10.1.1. Top-level architecture

Figure 18 below shows the high-level architecture of the MultiMedia SmartServer component.

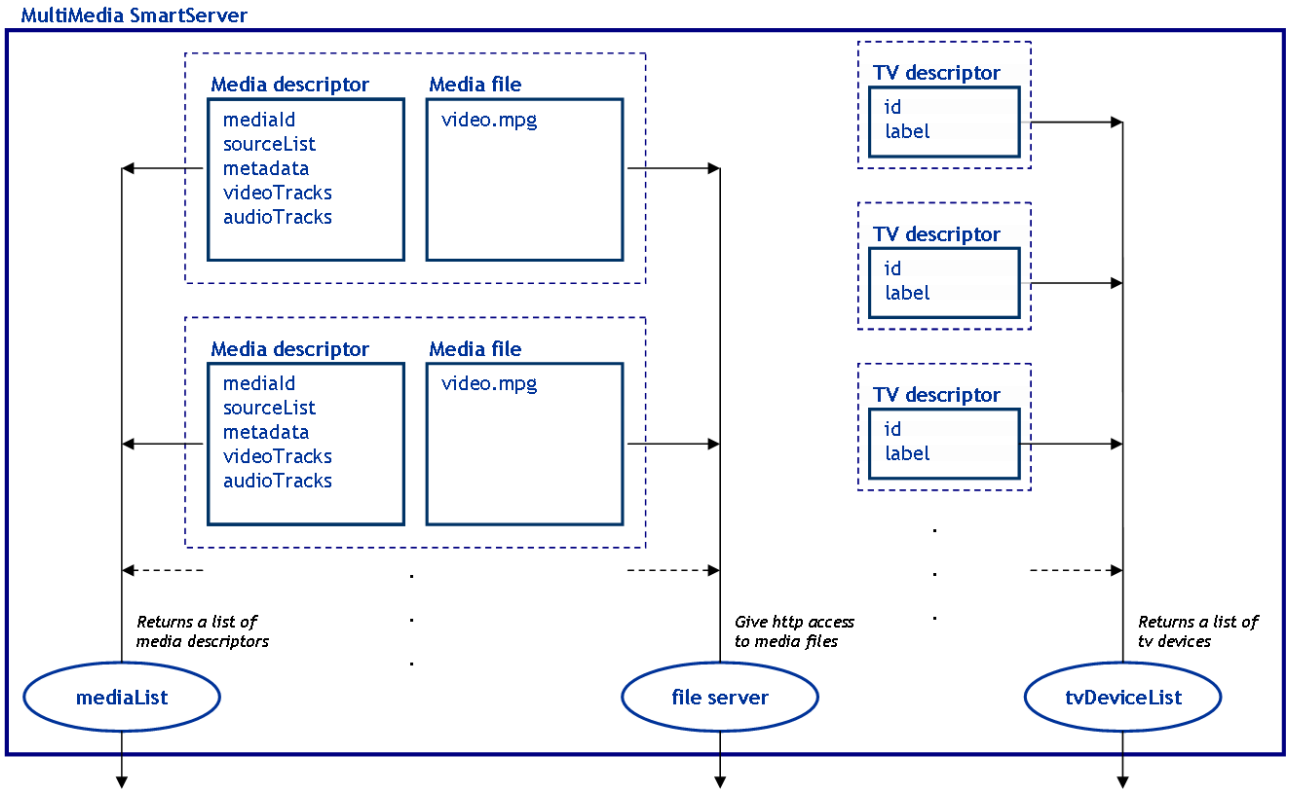


Figure 18: MultiMedia SmartServer high-level architecture

The most important service provided by the MultiMedia SmartServer is **mediaList**. It returns a list of media item descriptors containing all necessary information (access means, metadata and tracks). The following section describes in more details the content of these media descriptors.

In case of locally stored multimedia content, the media item itself (i.e. audio or video file) is made available through a **file server**, being a built-in http server or a stand-alone implementation such as Apache httpd.
Additionally, the MultiMedia SmartServer provides a list of multimedia device descriptors that can be actually used for multimedia rendering, through the **tvDeviceList** service. Again, details on multimedia device descriptors are given in the following section.

10.1.2. Data models

This section describes in details the data models as used in MultiMedia SmartServer services.

Figure 19 shows the architecture of the Media data model used as a single descriptor for any multimedia content.

- The MediaModel model itself contains two scalar fields: **mediaId** and **sourceList**.
- The **mediaId** field is an identifier, unique for a given SmartServer. It allows to identify a given media independently from the access method used (URL, server path, etc.).
- The **sourceList** is a list of URLs giving all access methods valid for this media.
- The **metadata** field is a map of Metadata models. This underlying model groups various metadata information like the title, the duration, etc.
- The language in which the metadata are given is coded according to the standard ISO 639-3 definition.
- The last three fields, **videoTracks**, **audioTracks** and **subtitleTracks**, are of TrackModel. They group the different types of tracks found in the media file. Usually there is only one video track, and several audio and subtitle tracks for multi-language movies.

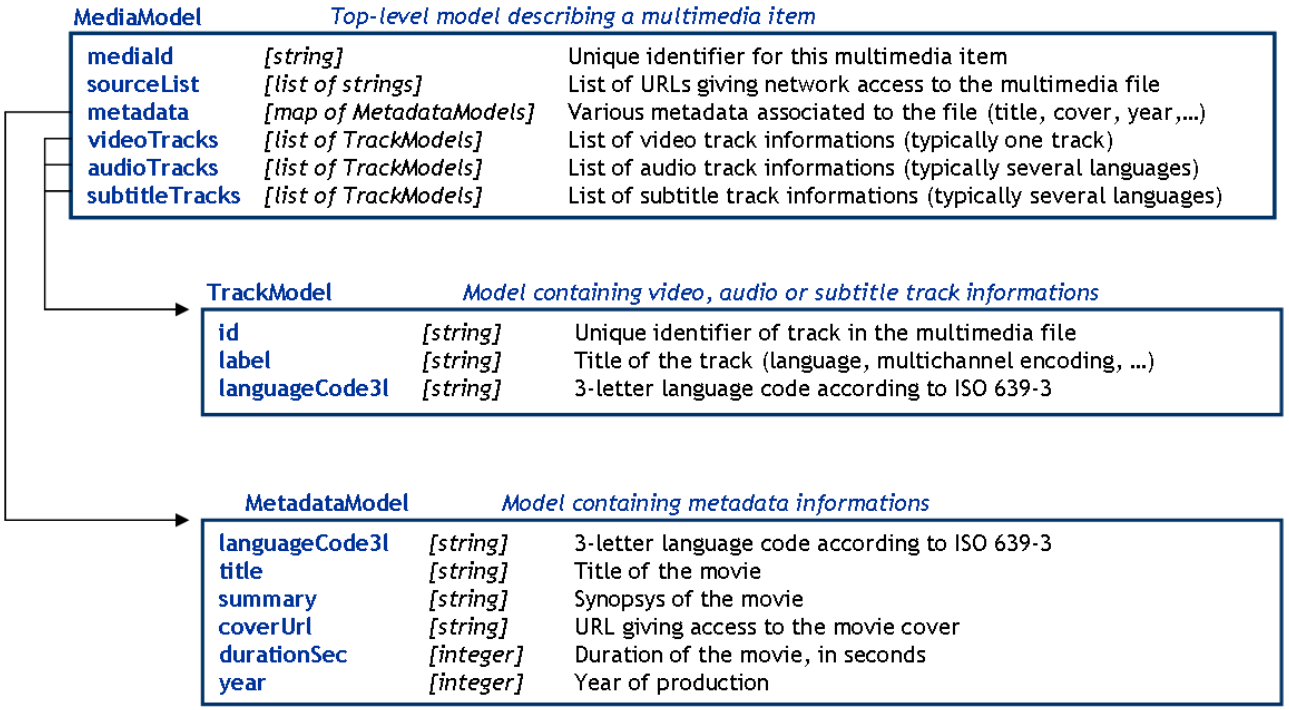


Figure 19: Media data model

Figure 20 describes the content of the TV device model, as returned by the **tvDeviceList** service.

- The first field, **id**, is an identifier unique in the local SmartServer domain (usually a house or a building). It is built and returned by the television itself, in the form of a serial number.
- The second field, **label**, is a human-readable name describing for instance the television location in the house (room name, or any other arbitrary string).

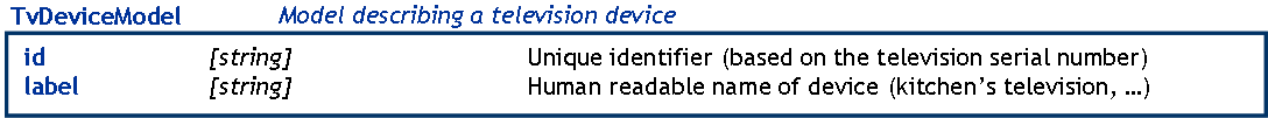


Figure 20: TV device data model

10.2. Software Architecture

Multimedia SmartServer is available as an OSGi [22] bundle, **MultiMediaProvider.jar**. This bundle only depends on an additional utility bundle, **MayaCommons.jar**.

The file server, giving access to media files, is implemented as a stand-alone http server (**Apache** httpd).

10.3. Enabling Technologies

The media descriptor side of the MultiMedia SmartServer is implemented in Java, encapsulated in an OSGi framework (Apache Felix [32]). The reason for this technological choice is that the MultiMedia SmartServer must be runnable both on a SmartGateway device (for a local SmartServer implementation) and a regular enterprise-grade SmartServer.

The file server side of the MultiMedia SmartServer is implemented as a standalone Apache Server [23].

10.4. API Specification

MultiMedia SmartServer services are available through a JSON-RPC API. For better readability, the actual JSON-RPC method is named with the get prefix.

10.4.1. getMediaList

The getMediaList method returns a list of MediaModel items (serialized in JSON).

10.4.1.1. Authentication

This method is not authenticated yet.

10.4.1.2. Signature

```
  { "method": "getMediaList",
    "params": [],
    "id": 1
  }
```

10.4.1.3. Parameters

This method does not require parameters.

10.4.1.1. Example

REQUEST

```
{ "method" : "getMediaList",
  "params" : [],
  "id" : 1
}
```

RESPONSE

```
{"result":[ {
"mediaId":"movie_Big_Buck_Bunny",
"sourceList":["http://192.168.0.3/multimedia/buck_bunny/bunny_movie.mp4"],
"metadataMap":{"eng":{
        "languageCode3l":"eng",
        "title":"Big Buck Bunny",
        "summary":"",
        "coverUrl":"http://192.168.0.3/multimedia/buck_bunny/bunny_movie.jpg",
        "durationSec":596,
        "year":2008}},
"videoTracks":[{"id":"0","label":"","languageCode3l":""}],
"audioTracks":[{"id":"0","label":"","languageCode3l":""}],
"subtitleTracks":[]
```

```
} ],
"id":1
}
```

10.4.2. getTvDeviceList

The getTvDeviceList method returns a list of available television devices.

10.4.2.1. Authentication

This method is not authenticated yet.

10.4.2.2. Signature

```
  { "method": "getTvDeviceList",
    "params": [],
    "id": 1
  }
```

10.4.2.3. Parameters

This method does not require parameters.

10.4.2.4. Example

REQUEST

```
{ "method" : "getTvDeviceList",
  "params" : [],
  "id" : 1
}
```

RESPONSE

```
{ "result" :
[
  { "id" : "TV-SAMSUNG-UE40ES6400-SH3ZDUD436AQ",
    "label" : "Samsung 40' Living Room",
  },
  { "id" : "TV-SAMSUNG-UE32ES5500-JDJG6SHD83DD",
    "label" : "Samsung 432 Kitchen",
  }
],
  "id" : 1
}
```

10.4.3. Error codes

No error codes yet.

11. Authorization SmartServer

11.1. Functionalities

The Authorization SmartServer enables separation of setup of the trust and the security of the data transfer between resource consumer (application) and resource provider. The trust enabler is not involved in data transfer; this separation allows implementation of privacy requirements. Here it is up to the Resource Consumer to support privacy requirements following current regulation concerning data storage and data usage.
In BUTLER Gemalto has defined the security framework including all message exchange; the security protocol allows end-to-end security between resource consumer (application) and resource provider (server, gateway and object). The application accesses resources on behalf of a user.

The authorization server provides authentication and authorization WEB services and WEB portal both for server administrators and final users. Final users can manage their resources, give access permissions to others users (acquaintance) to access their resources, manage access tokens generated for specific application and resource.

The security protocol is described in section 4.4.3 section from Deliverable D3.2 [3].

Recall of the Security Roles
The BUTLER security framework defines the exchange of messages between high level abstract security roles. Each security role is strictly defined and can be implemented by SmartObject, SmartObject Gateway, SmartMobile and SmartServer.

ROLE	DEFINITION
User	User entity granting access to an abstract 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 or acting on a resource on behalf of a user. Such user must be authorized to access the resource.
Authorization Server	The Authorization Server is the responsible for user authentication and for authorizing Resource Consumers getting or acting on a resource by issuing a resource-related *access-token.* Optionally, it may delegate the user authentication to an external Authentication Server. The Authorization Server is then responsible for implementing access-control management with regard to resources. It also plays the role of resource directory as access to resources is only possible when the Authorization Server issues an *access-token*.
Authentication Server	This **optional** role can be used by Authorization Server to rely on an authentication protocol not natively implemented in the Authorization Server. It means that the Authorization Server and Authentication Server shall be able to federate user identities.

Table 4 - High Level Security Roles

High Level Interactions
The figure below presents the interactions between the different Security Roles defined in BUTLER.

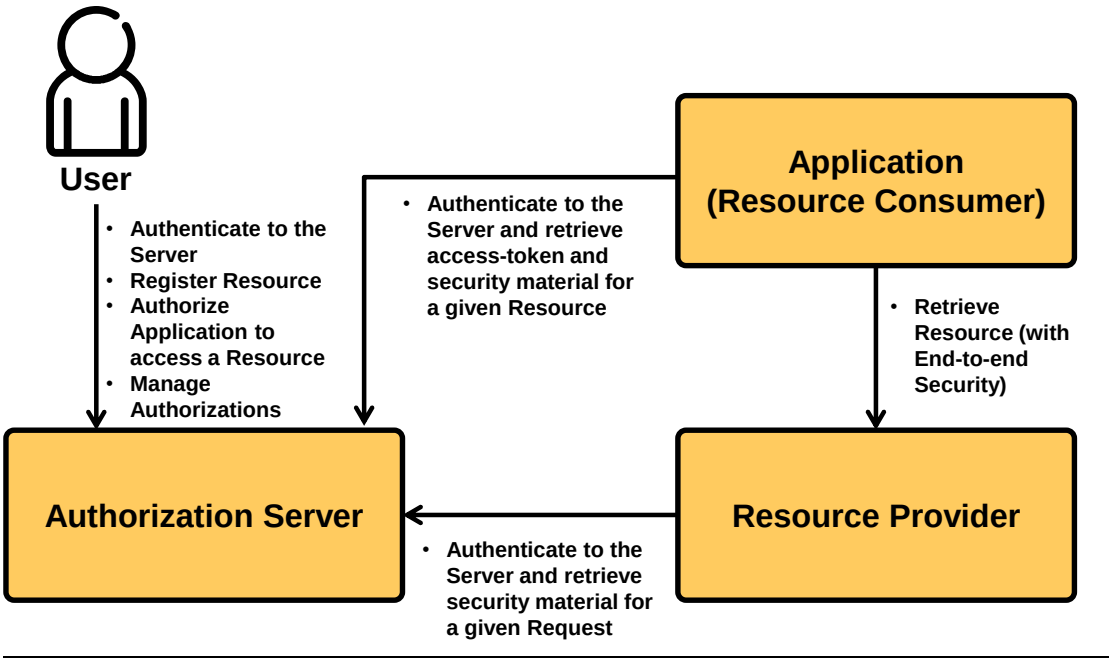


Figure 21: High-level Interactions between Security Roles

An overall use case can be represented as follow:

1. A User authenticates towards the Authorization Server
2. The User registers resource metadata to the Authorization Server.
3. Later, an application accesses the Authorization Server to retrieve an access-token for a specific resource always on behalf of an authenticated user – if the user is not already authenticated to the Authorization Server, it shall authenticate. Once authenticated, the user shall grant the application to retrieve the access-token.
4. Using the access token, the application can securely access the resource by giving the resource access-token.

The BUTLER Security Services include:

- Secure Transport of messages between any device and Authorization Server.
- Retrieval of the Access Token.
- User authentication to Authorization Server.
- Client application authentication to Authorization Server.
- Resource Registration
- Resource Authentication to Authorization Server.
- Object Key Management.
- End to End security between (application) Resource Consumer and Resource Provider.

11.2. Software architecture

The figure below presents the high level server architecture.

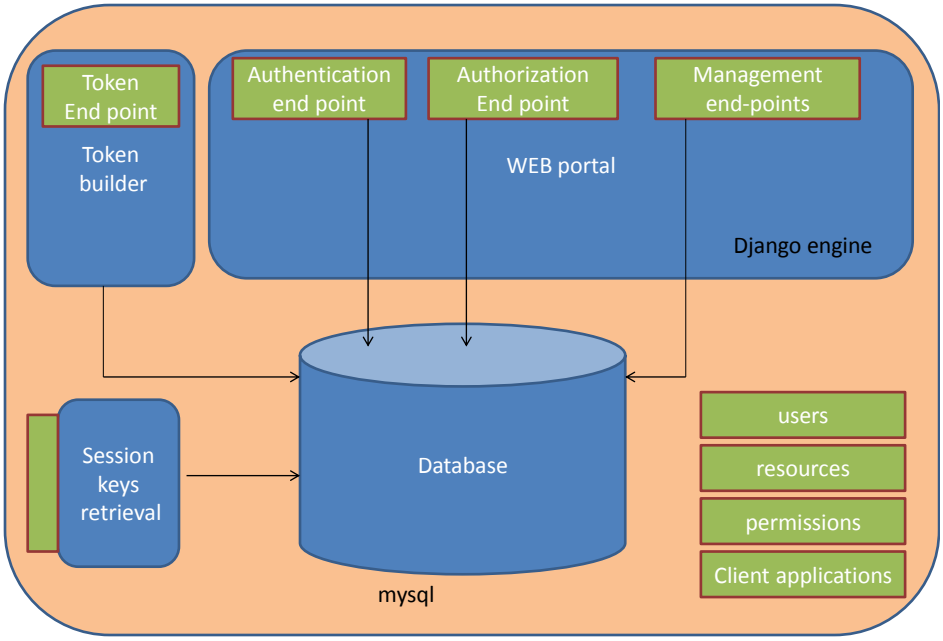


Figure 22: Authorization SmartServer high-level architecture.

WEB Portal. This component is developed using Django (open source) python framework [33]. It provides user interface for user authentication, authorization process and management user interface for administrator, and final users.

- Authentication end-point. The default authentication protocol uses the login/password paradigm. If called via an application, the authentication end-point first performs the user authentication and redirects to the application with an opaque handle representing the authenticated user for such application.

- Authorization end-point. Through this end-point the user is prompted to grant the calling application retrieving an access token to access a specific resource. On return, the application is provided with a grant-code to be used at token end-point.

- Management end-points. This set of end-points implements the management of the server database – see database description. The Management end-points provide user interface for:

    - Server administrator
        - User profile registration.
        - Client application registration.
    - Final user
        - User profile update. The user can update its profile including list of acquaintances to manage access Permissions.
        - Resource registration.
        - Permission management. Thru this interface, the resource owner can add/remove permission to one of its acquaintance for receiving an access token.
        - Token management. Thru this interface, the user can invalidate an access token already provided to an application.

WEB services. This component does not perform user interface. It provides required data to the calling application or resource provider.

- Token end-point. This end-point authenticates the calling application and returns the token (and security material) associated to the grant-code input parameter. Such grant-code is returned by the Authorization end-point to the Client redirection URL.
- Session keys end-points. This end-point authenticates the calling resource provider and returns the session keys associated to the keys-identifier input parameter. Such key-identifier is retrieved by the resource provided form the encrypted access token.

DATABASE. The database component uses **MySQL** [14]**.**

The server registers persistent data in SQL database. This database can be located anywhere on the network. For now, it is located on the same machine than the Authorization Server.

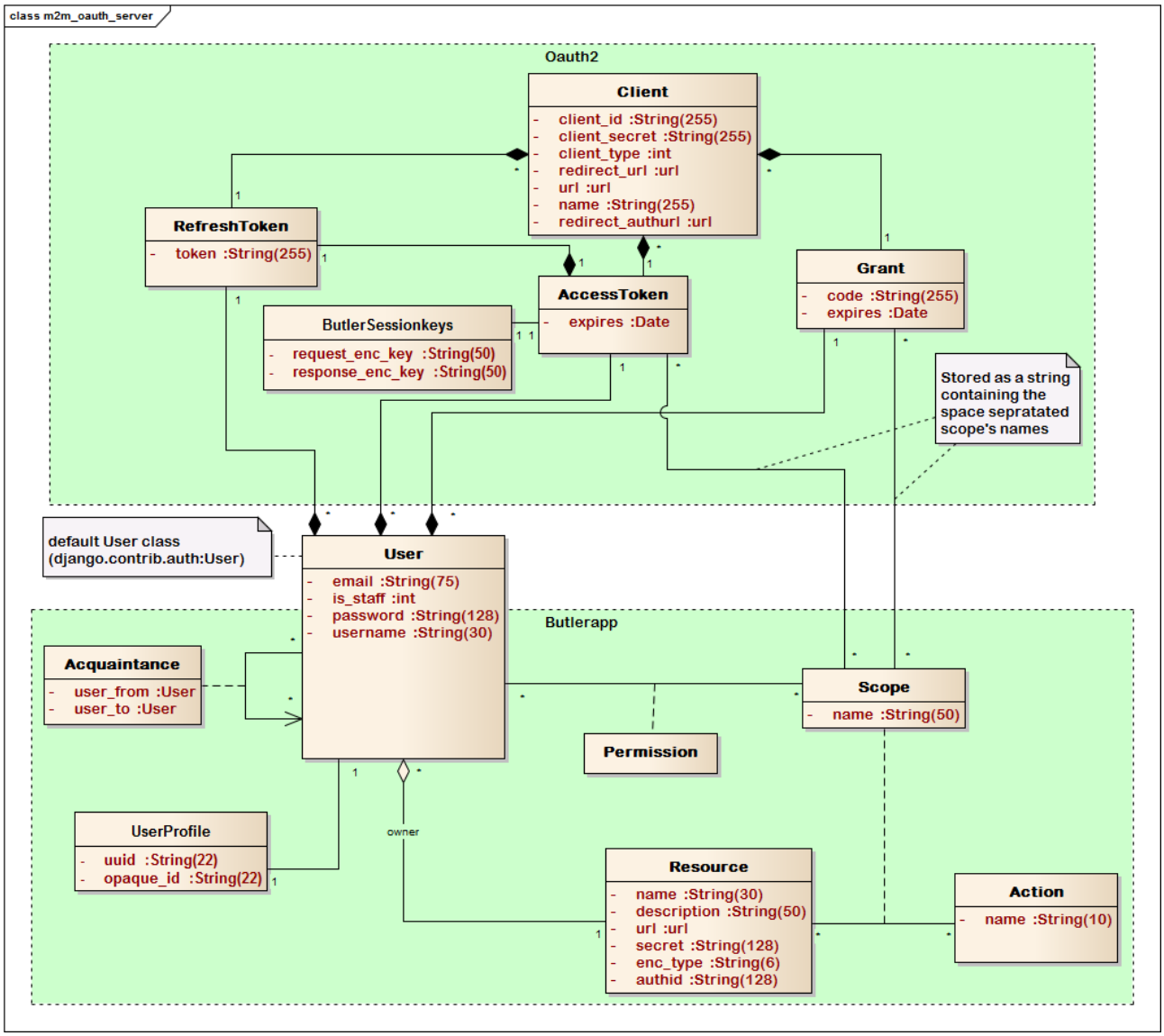


Figure 23: Authorization SmartServer database schema.

The main tables are User, Client and Resource, are described in the following paragraphs.

User.
The user table is managed by the server administrator.

DATA	DESCRIPTION
User - name	The name of the user.
User - password	The user authentication credential
User - UUID	Server generated long identifier. The user may know another user. If the user wants to give permissions to another user (let's say its friend) to access its own resources, the user shall add its friends to its Acquaintance list by adding the UUID of its friend. The UUID of the friend shall be given by the friend using an out-of-band channel (for instance by mail).
User - email	RFU

Client application.
The client application table is managed by the server administrator.
The clients are registered in the authorization server. They have some parameters that will be used in the OAuth process:

DATA	DESCRIPTION
Client Identifier	Represented by 20 hexadecimal digits, set when adding the client to the server See Client Authentication in
Client Secret	Represented by 40 hexadecimal digits, set when adding the client to the server
Redirection URL end point	Used by the authorization server to return response containing authorization credentials to the client via the user agent. See 11.4.4.
Redirection Auth URL end-point	Used by the authorization server to return response containing the user opaque identifier of the authenticated user. This end-point is called by the authentication end-point via the user agent. See 11.4.3.

Resource.
The resource is managed by the owner of the resource. Once authenticated the user can register a new resource in the database, the user is automatically the owner of this new resource.

DATA	DESCRIPTION
Common name	The usual name of the resource. This name helps the user to identify the resource but is no unique in the system.
Description	(optional) more detailed information
Resource URL	The end-point address of the resource. Ex: http://mychalet.fr/temperature.
Actions	The list of available actions for this resource
Key Material	The key material used by the server to compute the key encrypting the server generated access tokens. For now, the "Symmetric Encryption (AES 128 bits) key is supported. This value shall be also registered in related Resource Provider - see Deliverable 4.3 – Security Service.
Resource authentication code.	This code (credential) will be used by the Resource to authenticate to the authorization server and retrieve the request-encryption-key and the response-encryption-key for securing data exchange between the resource consumer and the resource provider.

	This value shall be also registered in related Resource Provider - see Deliverable 4.3 – Security Service. See also Session Key end-point 11.4.6.
Encryption Type	This type defines the algorithm to be used to encrypt the access-token using the key material. For now, the "Symmetric Encryption (AES 128 bits) key is supported

Permission (joint table) and **Acquaintance** (joint) table

This table allows giving permissions to a friend of the resource owner to retrieve an access token to such resource. The Acquaintance table registers the list of user acquaintance.

11.3. Enabling technologies

The Authorization SmartServer generates access-token and security material according the new security protocol has been defined and implemented in WP2 / Task 2.2 – security at application level. It is described in in section 4.4.3 section from Deliverable D3.2 [3].

The protocol is designed to be easily implemented at low cost devices. For now, the implementation does not request any specific secure element at device side. The deployment of the security keys is done manually. The user defines – by himself - a resource provider secret and resource provider authentication code and sets such credentials in both resource provider software and Authorization SmartServer database using the management end-points. This way, the security can be perform using objects operated by the user and not by specific operator that may enhance the initial key provisioning by providing software protected security keys or (better) by providing security using embedded secure element. Anyway, in this case, the object shall be linked to the Authorization SmartServer – this constraint allows security enhancement but could be irrelevant for real deployment of small objects.

11.4. API specifications

11.4.1. Scope definition

The scope is a list of space-separated actions. An action is the form of "<url>_<action name>". The URL is the URL where the resource is exposed by the Resource Provider and should be base64 encoded.

To generate the token, scopes must relate to the same resource. Otherwise, the request will be denied.

Example

For the resource accessible at URL "http://mychalet.org/gateway/temp1" and actions "SET" and "GET", the scope will be:

```
aHR0cDovL215Y2hhbGV0Lm9yZy9nYXRld2F5L3RlbXAx_GET
aHR0cDovL215Y2hhbGV0Lm9yZy9nYXRld2F5L3RlbXAx_SET
```

11.4.2. Client authentication

When required by the end-point the client application must use the HTTP Basic authentication scheme as defined in RFC2617 [34] to authenticate with the authorization server when requesting a token. The client_id and client_secret will we used.

Basic Authentication example

If the user agent wishes to send the client_id "37adc08f6963cecc59b6" and client_secret "f9c98a457f2b5e55d86575d2f435e34dc4c42157", it would use the following HTTP header field (base64 encoding of "<client_id>:<client_secret>"):

```
Authorization: Basic
MzdhZGMwOGY2OTYzY2VjYzU5YjY6ZjljOThhNDU3ZjJiNWU1NWQ4NjU3NWQyZjQzNWUzNGRjNGM0MjE1Nw==
```

11.4.3. User Authentication Request end-point

MESSAGE: Resource Consumer - Authorization Server

The client application may request the Authorization Server to authenticate the user. Once the user is authenticated, the client application will be provided with an opaque handle representing the authenticated user for this application. The application does not receive any information about the user but knows that a specific opaque handle is mapped to a single user. On next authentication, the application will receive the same value for the same user. This way, the user does not expose personal data to external application. Thanks to the opaque handle for preserving user privacy by avoiding sending user identifier.

11.4.3.1. Authentication

Not protected - requires only the client identifier.
For security reason, the password is not passed because this method is called thru the user agent and the application password is never outside the client application.

11.4.3.2. Signature

```
GET    /api/auth?client_id={client_id}
```

11.4.3.3. Parameters

PARAMETERS	VALUE	DESCRIPTION	TYPE
client_id	12345	the client identifier	String

11.4.3.4. Example

For example, the client directs the user-agent to make the following HTTP request using TLS:

REQUEST

```
GET /api/auth?client_id=37adc08f6963cecc59b6 HTTP/1.1
Host: <m2m-oauth-server>
```

RESPONSE

The server redirects the response to the client application *redirect_authurl.* The client authentication redirection endpoint is defined in the client repository.

The authentication issues an opaque handle and delivers it to the client application.

11.4.3.4.1. Authentication

Not protected.

11.4.3.4.2. Signature

```
GET    <client-authentication-redirection-endpoint>?userapp_opaque_id={userapp_opaque_id}
```

11.4.3.4.3. Parameters

PARAMETERS	VALUE	DESCRIPTION	TYPE
userapp_opaque_id	534534	40 Hexa Decimal Digits representing the opaque identifier of the user for the client application. This identifier is always the same for a couple (application/user), but the application cannot reverse the identifier to the user identifier in the authorization server.	String

11.4.3.4.4. Example

The authorization server redirects the user-agent by sending the following HTTP response:

```
HTTP/1.1 302 Found
Location: <client-authentication-redirection-
endpoint>?userapp_opaque_id=a6be7cfae3b79e0ebc2c4e1d1c6d747b11a5c074e120cc1bd27713f506032747
```

11.4.4. Authorization end-point

MESSAGE: Resource Consumer - Authorization Server

The Authorization end-point is used by the client application to get an access-token for a specific resource and action – together so called scope. For this purpose, the server provides two schemas specified in RFC 6749 [31]:

- Authorization Code Grant. With this schema, the user shall be authenticated to the Authorization Server and shall grant the client application to get the access token. In this schema the user credential is not given to the client application.
- Password Grant. With this schema, the user fully trusts the client application by giving its personal credential. The application now can retrieve the access token without any I user interaction. In this schema, the user always grants the application to retrieve access token for its resources.

11.4.4.1. Authorization Code Grant

Follow the scheme described in RFC 6749, section 4.1 [31].

11.4.4.1.1. Authentication

Not protected - requires only the client identifier.
For security reason, the password is not passed because this method is called thru the user agent and the application password is never outside the client application.

11.4.4.1.2. Signature

```
GET
/api/oauth2/authorize?response_type={response_type}&client_id={client_id}&scope={scope}&state={st
ate}
```

11.4.4.1.3. Parameters

PARAMETERS	VALUE	DESCRIPTION	TYPE
response_type	code	REQUIRED. For “Authorization Code Grant” schema, the value MUST be set to "code".	String

client_id	xxxx	REQUIRED The client identifier.	String
scope	-----	REQUIRED The scope of the access request as defined in 11.4.1.	String
state	----	RECOMMENDED. An opaque value used by the client to maintain state between the request and callback.	String.

11.4.4.1.4. Example

REQUEST
The client directs the user-agent to make the following HTTP request using TLS:

```
GET
/api/oauth2/authorize?response_type=code&client_id=37adc08f6963cecc59b6&scope=aHR0cDovL215Y2hhbGV
0Lm9yZy9nYXRld2F5L3RlbXAx_GET&state=363057b4 HTTP/1.1
Host: <m2m-oauth-server>
```

RESPONSE
The server redirects the response to the client application at *redirect_url.*

If the resource owner grants the access request, the authorization issues an authorization code and delivers it to the client by adding the following parameters to the redirection URI using the "application/x-www-form-urlencoded" format.

11.4.4.1.4.1. Authentication

Not protected.

11.4.4.1.4.2. Signature

```
GET    <client-redirection-endpoint>?code={code}&state={state}
```

11.4.4.1.4.3. Parameters

PARAMETERS	VALUE	DESCRIPTION	TYPE
code	----	REQUIRED. Represented by 40 hexadecimal digits. The authorization code will expire shortly after it is issued to mitigate the risk of leaks (OAUTH_EXPIRE_CODE_DELTA parameter in Server configuration). The client MUST NOT use the authorization code more than once. If an authorization code is used more than once, the authorization server will deny the request. The authorization code is bound to the client identifier.	String
state	--	Defined by the client is Authorization Request	String

11.4.4.1.4.4. Example

The authorization server redirects the user-agent by sending the following HTTP response:

```
HTTP/1.1 302 Found
Location: https://<client-redirection-
endpoint>?code=62ec27e2437e8d457ea5657dc4bb2fa38f17ea8b&state=363057b4
```

11.4.5. Token end-point

MESSAGE: Resource Consumer - Authorization Server

11.4.5.1. Authorization Code Grant

Follow the scheme described in RFC 6749, section 4.1 [31].

11.4.5.1.1. Authentication

The client application must authenticate using authentication protocol defined in 11.4.1.

11.4.5.1.2. Signature

```
POST /api/oauth2/access_token/
```

11.4.5.1.3. Parameters

PARAMETERS	VALUE	DESCRIPTION	TYPE
grant_type	authorization_code	The type of the next code	String
code	----	The authorization code received from the authorization server.	String
hash_app_private_id	----	**NEW BUTLER parameter – not in OAUTH 2.0** The application shall define a private identifier (preferably a random value) and gives the hash of such value to the token end-point. When building the request-authentication-data to access the resource provider, the client application shall have to use this private identifier.	String

11.4.5.1.4. Example

Token request example - the client makes the following HTTP request over TLS:

REQUEST

```
POST /api/oauth2/access_token/ HTTP/1.1
Host <m2m-oauth-server>
Authorization Basic
MzdhZGMwOGY2OTYzY2VjYzU5YjY6ZjljOThhNDU3ZjJiNWU1NWQ4NjU3NWQyZjQzNWUzNGRjNGM0MjE1Nw==
Content-Type application/x-www-form-urlencoded

grant_type=authorization_code&code=62ec27e2437e8d457ea5657dc4bb2fa38f17ea8b&hash_app_private_id=7
1b4e190fc7a0aa86f24cb18d88c09bfd8a45292f1ae434fac3c0351f4d838d3
```

RESPONSE

If the access token request is valid and authorized, the authorization server issues an access token, a refresh token and session keys. It constructs the response by adding the following parameters (using the "application/json" media type) to the entity-body of the HTTP response with a 200 (OK) status code.

PARAMETERS	VALUE	DESCRIPTION	TYPE
access_token	---	The Token for accessing the resource provider.	String
scope	----	The scope of the access request as defined in 11.4.1.	String
expires_in	----	The lifetime in seconds of the access token.	Number
refresh_token	----	The refresh token, which can be used to obtain new access tokens as described in 11.4.5.3.	String
request_authentication_key	----	The encryption key that shall be used by the client to encrypt the request authentication data - base64 encoded.	Number
request_encryption_key	---	Base 64 encryption key. It will be by the client to encrypt the request parameters	String
response_encryption_key	---	Base 64 encryption key. The key will be used by the resource to encrypt the result of the request.	String.
userapp_opaque_id	---	40 Hexa Decimal Digits representing the opaque identifier of the user for the client application. This identifier is always the same for a couple (application/user), but the application cannot reverse the identifier to the user identifier in the authorization server.	String

An example successful response (with indented JSON):

```
HTTP/1.0 200 OK
Content-Type: application/json;
Pragma: no-cache
Cache-Control: no-store

{
   "access_token":
"Z/J6by7ou7oCK95tf3KsLGMpgqAyqFbW9/EmkIL3p4y4NpdoXfW+lmDII244PguOlzQi5/S8J+5CspQvoBzDyNgtT56ERd3T
YgdAJEt7kGoujW+hn5iwBekFMUaUREJep/HwZF3Ace3JpAwDkRZEGITFmHRJPysCw1e2BH/AErk=",
   "scope": "aHR0cDovL215Y2hhbGV0Lm9yZy9nYXRld2F5L3RlbXAx_GET",
   "expires_in": 31535999,
   "refresh_token": "e29487f5845852784be9f8f4f4030e57b4cd00e7",
   "request_authentication_key": "rM2UVeb8T/8YM6O+6qLvuQ==",
   "request_response_keys_identifier": 8,
   "request_encryption_key": "bqj+hrycw1B0K1RGJFKTPQ==",
   "response_encryption_key": "4UEU9cU+tCFBJ04Ak+Mxaw==",
   “userapp_opaque_id”:  “a6be7cfae3b79e0ebc2c4e1d1c6d747b11a5c074e120cc1bd27713f506032747”
}
```

11.4.5.2. Password Grant

Follow the scheme described in RFC 6749, section 4.3 [31]

In this schema, the user fully trusts the client application; it provided its credentials to the client application and the user does not have any control to grant access to its resources.

11.4.5.2.1. Authentication

The client application must authenticate using authentication protocol defined in 11.4.1.

11.4.5.2.2. Signature

```
POST    /api/oauth2/access_token/
```

11.4.5.2.3. Parameters

PARAMETERS	VALUE	DESCRIPTION	TYPE
grant_type	"password"	The type of the next code	String
username	----	The name of the user	String
password	-----	The password of the user	String
scope	---	The scope of the access request as defined in 11.4.1.	String
hash_app_private_id	----	**NEW BUTLER parameter - not in OAUTH 2.0** The application shall define a private identifier (preferably a random value) and gives the hash of such value to the token end-point. When building the request-authentication-data to access the resource provider, the client application shall have to use this private identifier.	String

11.4.5.2.4. Example

The client makes the following HTTP request using TLS:

```
POST /api/oauth2/access_token/ HTTP/1.1
Host <m2m-oauth-server>
Authorization Basic
MzdhZGMwOGY2OTYzY2VjYzU5YjY6ZjljOThhNDU3ZjJiNWU1NWQ4NjU3NWQyZjQzNWUzNGRjNGM0MjE1Nw==
Content-Type application/x-www-form-urlencoded

grant_type=password&username=john&password=mysecret&scope=aHR0cDovL215Y2hhbGV0Lm9yZy9nYXRld2F5L3R
lbXAx_GET&hash_app_private_id=71b4e190fc7a0aa86f24cb18d88c09bfd8a45292f1ae434fac3c0351f4d838d3
```

RESPONSE
The same as described in 11.4.5.1

11.4.5.3. Refreshing a (delivered) Access Token

MESSAGE: Resource Consumer - Authorization Server.

11.4.5.3.1. Authentication

The client application must authenticate using authentication protocol defined in 11.4.1.

11.4.5.3.2. Signature

```
POST    /api/oauth2/access_token/
```

11.4.5.3.3. Parameters

PARAMETERS	VALUE	DESCRIPTION	TYPE
grant_type	"refresh_token"	The type of the next code	String
refresh_token	----	The refresh token issued to the client.	String
password	-----	The password of the user	String
scope	---	OPTIONAL. The scope of the access request as defined in 11.4.1.	String
hash_app_private_id	----	NEW Gemalto parameter - not in OAUTH 2.0 The application shall define a private identifier (preferably a random value) and gives the hash of such value to the token end-point. When building the request-authentication-data to access the resource provider, the client application shall have to use this private identifier.	String

11.4.5.3.4. Example

REQUEST

The client makes the following HTTP request using TLS:

```
POST /api/oauth2/access_token/ HTTP/1.1
Host <m2m-oauth-server>
Authorization Basic
MzdhZGMwOGY2OTYzY2VjYzU5YjY6ZjljOThhNDU3ZjJiNWU1NWQ4NjU3NWQyZjQzNWUzNGRjNGM0MjE1Nw==
Content-Type application/x-www-form-urlencoded

grant_type=refresh_token&refresh_token=20f6e265ddf73ddcd06e7a936be502266e3efc52&hash_app_private_
id=71b4e190fc7a0aa86f24cb18d88c09bfd8a45292f1ae434fac3c0351f4d838d3
```

RESPONSE

The same as described in 11.4.5.1

11.4.6. Session keys end-point

MESSAGE: Resource Provider - Authorization Server.

This message is used by the Resource Provider API to get the session keys associated to a given key identifier. The Resource Provider shall authenticate to the Authorization using its authentication code defined in the resource data of the Authorization Server.

11.4.6.1. Authentication

The resource authenticates using the **authcode** parameter – see parameters.

11.4.6.2. Signature

```
POST   /api/session_keys/
```

11.4.6.3. Parameters

PARAMETERS	VALUE	DESCRIPTION	TYPE
request_response_keys_identifier	--	The keys identifier. The id of the key set, given by the application	Number

		when attempting a resource with a token	
authcode	----	The resource auth code provided when registering the resource	String

11.4.6.4. Example

REQUEST

The client makes the following HTTP request using TLS:

```
POST /api/session_keys/ HTTP/1.1
Host <m2m-oauth-server>
Content-Type application/x-www-form-urlencoded

request_response_keys_identifier=8&authcode=myauthcode
```

RESPONSE

If the request is valid and if the identified keys are already enabled, the authorization server returns the session keys, constructs the response by adding the following parameters (using the "application/json" media type) to the entity-body of the HTTP response with a 200 (OK) status code.

PARAMETERS	VALUE	DESCRIPTION	TYPE
request_response_keys_identifier	--	The request keys identifier. The id of the key set, given by the application when attempting a resource with a token	Number
request_encryption_key	----	Base64 request encryption key	String
response_encryption_key	---	Base64 response encryption key	String

An example successful response (with indented JSON):

```
HTTP/1.0 200 OK
Content-Type: application/json;

{
   "request_response_keys_identifier": 8,
   "request_encryption_key": "bqj+hrycw1B0K1RGJFKTPQ==",
   "response_encryption_key": "4UEU9cU+tCFBJ04Ak+Mxaw=="
}
```

12. Developers' Support

12.1. BUTLER Application Repository

12.1.1. Overview

As described in deliverable D4.2 [4], a SmartMobile application can be either a hybrid app or a native one. While designing the deployment workflow, it clearly appeared that a common repository space was required in order to deploy BUTLER services and applications. Thus, the application repository would act as:

- A regular app repository, where users could browse a list of available applications.
- A web server that would make available apps through mobile web browsers, in HTML5.

The figure below shows a high level view of the involved roles and interactions between the SmartMobile platform and the BUTLER Application Repository, covering both the client side (SmartMobile app, web browser) and the BUTLER services repository.

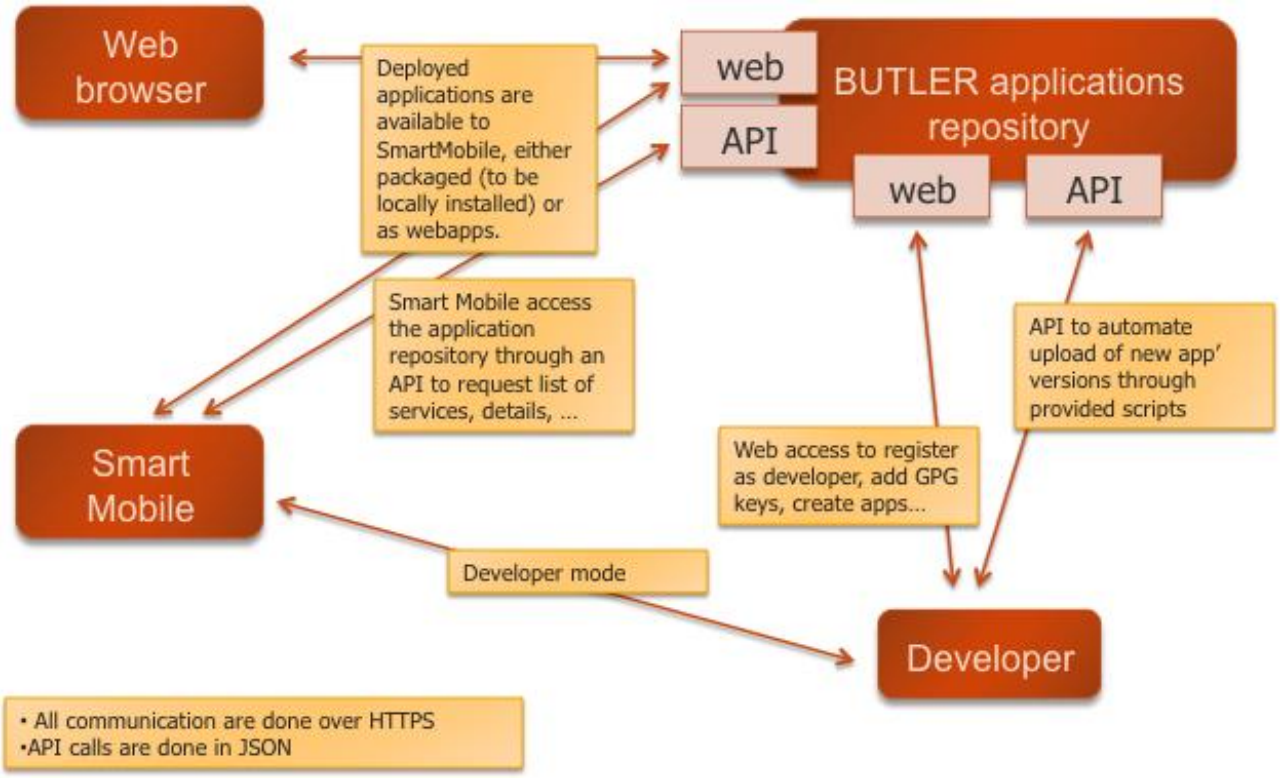


Figure 24: Roles and interaction between SmartMobile BUTLER repository

The following steps describe the workflow that an application developer should follow to publish a new BUTLER application:

1. The developer builds his/her application using the BUTLER SmartMobile development tools
2. When the application is finished and ready to be deployed, the developer uses a script to submit the application to the BUTLER Application Repository. This script signs the code and uploads it to the BUTLER Application Repository
3. Once on the BUTLER Application Repository, the server-side component checks the signature, and if the file is validated, makes the application available to deployment to BUTLER SmartMobile clients

4. Then, clients (e.g. BUTLER SmartMobile client or web browsers) can access the application: while the BUTLER SmartMobile client will use the BUTLER Applications Repository API to get the list of files to be downloaded, the web browser will request files as done in regular web applications.

12.1.2. Security Mechanisms

In order to ensure that applications are securely deployed, a secure workflow has been created. A GPG[6] key is needed to sign an application to be deployed.

Figure 25 shows what a developer must do before proceeding with his/her first deployment: in order to be able to deploy, the developer must first generate a GPG key. GPG needs to be installed; to create a key:

```
> gpg -gen-key
```

To get a public key:

```
> gpg --armor --export
```

To get the list of keys' id (useful if multiple keys available):

```
> gpg --list-keys
```

The BUTLER Wiki[7] provides more technical detailed on how to setup the development environment and guides the developers through an example. Moreover, additional details on that matter are given in D4.2.

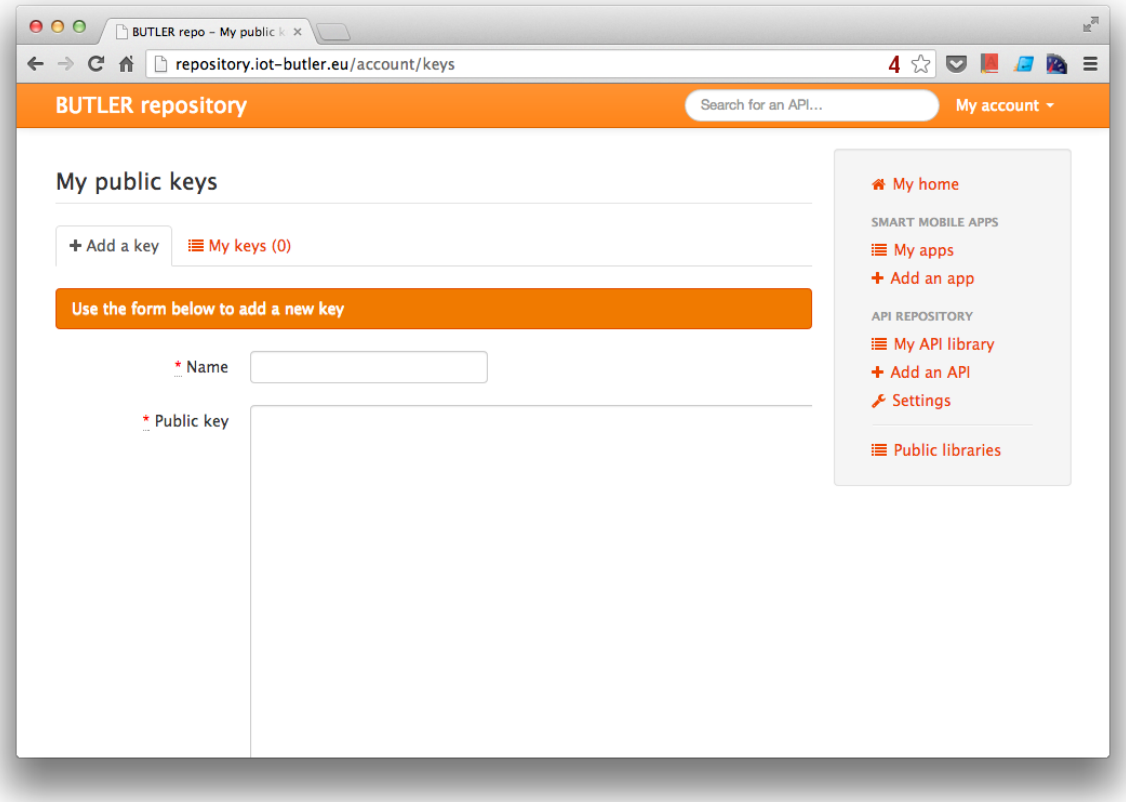


Figure 25: Repository keys configuration.

[6] http://www.gnupg.org/

[7] http://ctu-tools.inno-projects.net. Access available upon request.

12.1.1. Application Repository API

Please note that this section describes the API related to the application repository. It does not cover the API Catalogue (which is available at the same URL) that is used to document BUTLER's API.

The repository exposes a few API endpoints that are used by:

- The deployment tool to upload new versions of application
- SmartMobile apps:
  - To get list of available applications
  - To get list of files for a specific application

This API is described in the BUTLER Application Repository API as shown on the following screenshot.

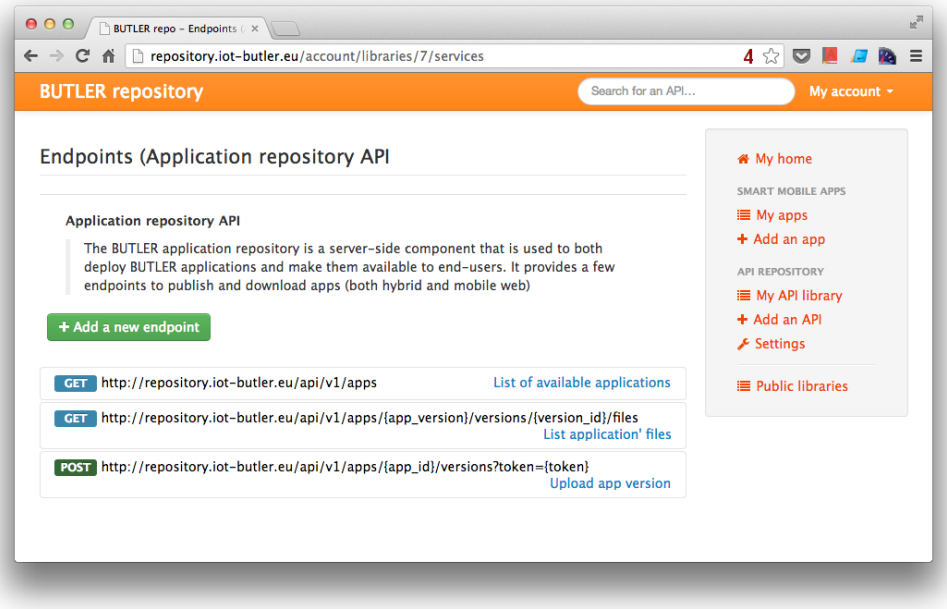


Figure 26: API Catalogue.

12.2. BUTLER API Catalogue

12.2.1. Introduction

In order to facilitate the process of documenting API developed during the project, a dedicated BUTLER API Catalogue (web application) has been developed and available to both BUTLER partners and external developers. The BUTLER API Catalogue is published at the following URL:

http://repository.iot-butler.eu/libraries

Its main objectives are to:

- Document each Smart Server API
- Provides a mean to export as PDF registered APIs
- Allow live testing of APIs in order to ease development processes

The following screenshots give an overview of the tool.

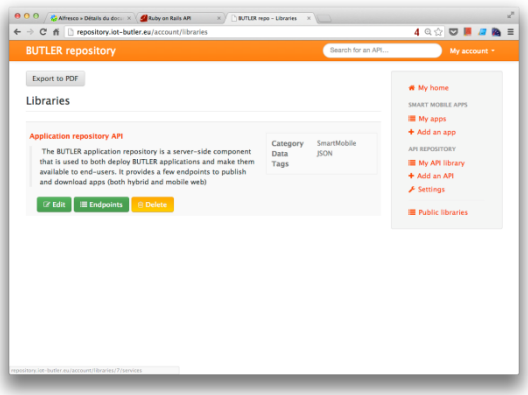


Figure 27: API Catalogue - my libraries

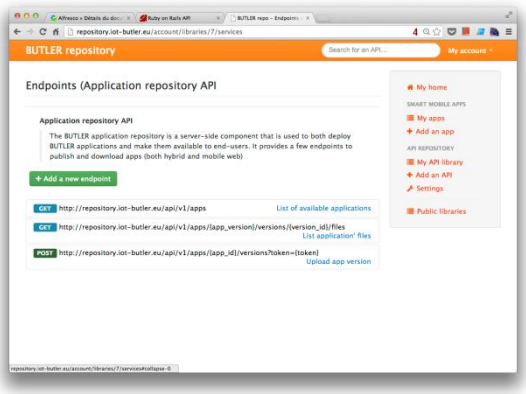


Figure 28: API Catalogue - endpoints

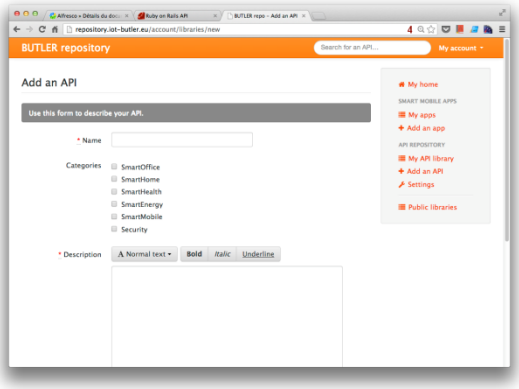


Figure 29: API Catalogue - add an API

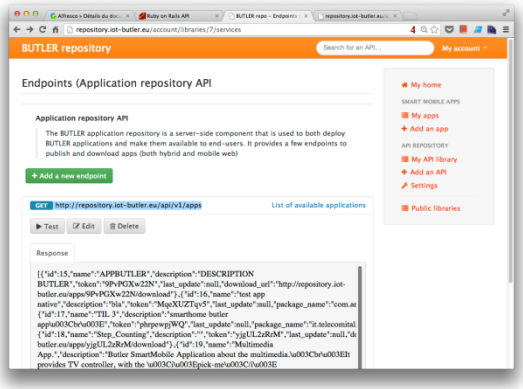


Figure 30: API Catalogue - live testing

12.2.1. Live testing

As shown on Figure 30, developers get through the BUTLER API Catalogue the possibility to get APIs by sending custom requests.

When a BUTLER API developer documents online his/her API, he/she indicates all details needed to send requests: endpoint, HTTP methods used (GET, POST, …), parameters (name, format, default values), request body (in case of POST requests), data format, etc. Holding this information, the API catalogue then provides to developers willing to integrate a BUTLER API a mean to send test requests before doing their integration work.

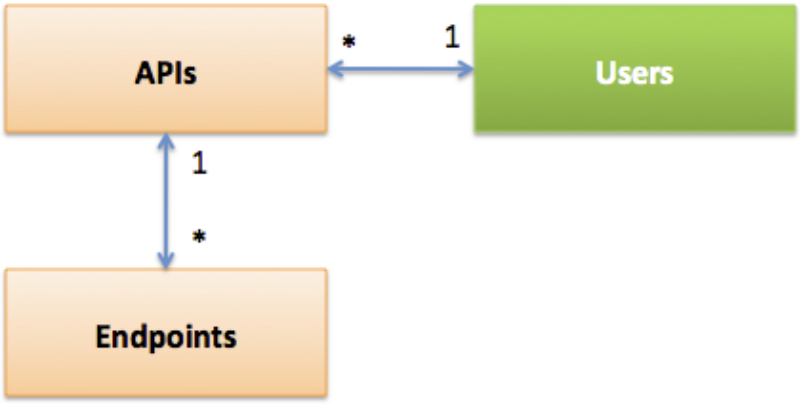
To test an API, a developer has to provide parameter values and click the “test” button. The API Catalogue will then send the request to the related SmartServer and will display the server’s response on a text area.

Technically, in order to avoid cross-origin policy limitations, it works through an intermediary server. Indeed, when a developer requests a test, all details related to the API to be tested (mainly endpoints and parameters) are sent to the API catalogue backend which in turn sends the request to the corresponding server (in that case, the API catalogue backend simply acts as a proxy). Upon reception of the response, it is sent back to the client (e.g. the web browser).

12.2.2. Architecture

The API Catalogue is a web application, relying on a web server and a database to store API-related data.

The figure below shows the data model: a table holds APIs description and another holds endpoints (e.g. methods in the sense of object oriented programming). Multiple endpoints can be associated to an API. Within the endpoints table, each endpoint technical description (e.g. parameters and URL) is stored within a textual field containing JSON data. An example is given on the figure below.




```
[{"name":"app_version","value":"9PvPGXw22N",
  "desc":"Version ID (token, string[10])","type":"String"},
 {"name":"version_id","value":"15",
  "desc":"Application ID","type":"Number"}]
```

Figure 31: BUTLER API Catalogue data model and endpoint data in JSON.

This application is built using the following tools:

- On the server side: Apache HTTPd [23], Ruby on Rails [36] along with the Ruby programming language, MySQL database server [14]
- On the client side: any HTML5 compliant web browser, JavaScript (JQuery [35]), JSON.

13. Conclusions

This document provides the description of the SmartServer Platform in BUTLER, as implemented until month 24.

The platform has been described server by server, for each server a description of offered functionalities, its internal software architecture and the detailed API description is given. For each SmartServer application developers will find in this document a description of the horizontal component to be integrated in their BUTLER applications.

Server APIs have been described as they are currently developed in the first version of the horizontal platform. Together with the documentation provided in this document, application developers will also find an online version of the API description, called the BUTLER API Catalogue, where most of the APIs can also be tested live to learn the specific signature and required parameters, as well as to check the returned data from each API request performed towards the SmartServer.

The following steps for the SmartServer Platform to be carried out in Year 3 are:

- add additional SmartServer to the platform, according to new requirements coming from trial applications and improved versions of existing ones
- improve the functionalities offered by each SmartServer, according to feedback received from applications developers
- produce a second version of each offered API, taking into account suggestions coming from developers
- improve documentation of the current APIs
- improve error handling and relative documentation
- complete the API Catalogue server to offer additional functionalities like integrated authentication (developer's authentication)
- perform a thorough evaluation of system performances according to indicators to be decided and to the service trials over which the platform will be measured
- provide additional developer tools to enhance the usability of the platform from a development perspective and it ability to attract new developers and external applications that want to benefit from the BUTLER platform
- integration of security mechanisms defined in BUTLER to all deployed SmartServers

These improvements of the platform will lead to a second version of the software described here planned for M30, which will similarly be described in Deliverables D5.1 and D5.3.

14. References

[1] Butler Consortium, D2.4 – Selected technologies for the Butler platform (March 2012).

[2] Butler Consortium, D2.2 – Requirements, Specifications, Localization and Context-acquisition for IoT Context-aware Networks (October 2012).

[3] Butler Consortium, D3.2 – Integrated System Architecture and Initial Pervasive BUTLER proof of concept (October 2013)

[4] Butler Consortium, D4.2 – BUTLER SmartMobile platform and enabling technologies (November 2013)

[5] Dai MingBo, F. Sottile, M. A. Spirito, R. Garello. “An Energy Efficient Tracking Algorithm in UWB-based Sensor Networks”. 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob). October 2012, pp. 173-178.

[6] J. Saloranta, S. Severi, D. Macagnano, G. Abreu. “Sensor Localization with Algebraic Confidence”, 46th Annual Conference on Signal, Systems and Computers. November 2012.

[7] BUTLER API Catalogue. Available online at http://repository.iot-butler.eu/libraries

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

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

[10] OpenSocial Data Model. Available at http://opensocial-resources.googlecode.com/svn/spec/3.0/

[11] Java EE at a Glance. Available at http://www.oracle.com/technetwork/java/javaee/overview/index.html

[12] JSR 339: JAX-RS 2.0: The Java API for RESTful Web Services. Available at http://jcp.org/en/jsr/detail?id=339

[13] EJB 3.0 Specs http://jcp.org/aboutJava/communityprocess/final/jsr220/

[14] MySQL Database system. Web page at http://www.mysql.com/

[15] Hibernate – JBoss Community. Web page at www.hibernate.org

[16] JBoss Application Server. Web pate at http://www.jboss.org/overview/

[17] Complex event processing with ESPER, http://esper.codehaus.org/

[18] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten (2009); The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1, http://www.cs.waikato.ac.nz/ml/weka/citing.html

[19] Parliament™: A High-Performance Triple Store, SPARQL Endpoint, and Reasoner, Available at http://parliament.semwebcentral.org/

[20] Jersey RESTful Web Services framework. Available at https://jersey.java.net/

[21] SAMURAI: A Streaming Architecture for Mobile and Ubiquitous RESTful Analysis and Intelligence. Available at https://monte.cs.kuleuven.be/samurai/

[22] OSGi Service Platform Release 4. Web page at http://www.osgi.org/Release4/HomePage

[23] Apache Server. Web page at http://httpd.apache.org/

[24] Mosquitto. An Open Source MQTT v3.1 Broker. Web page at http://mosquitto.org/

[25] BSON. Web page at http://bsonspec.org/

[26] Drools Expert User Guide. Available at http://docs.jboss.org/drools/release/5.5.0.Final/drools-expert-docs/html/index.html

[27] OpenForecast. Web page at http://www.stevengould.org/software/openforecast/index.shtml

[28] PostgreSQL. Web page at http://www.postgresql.org/

[29] Apache AXIS. Web page at http://axis.apache.org/axis/

[30] Jackson JSON Processor. Web page at http://wiki.fasterxml.com/JacksonHome

[31] [RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012. Available at http://www.ietf.org/rfc/rfc6749.txt

[32] Apache Felix iPOJO Service Component Model. Web page at http://felix.apache.org/site/apache-felix-ipojo.html

[33] Django project. Web page at https://www.djangoproject.com/

[34] [RFC2617] Franks, J. et al., "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999. Available at http://www.ietf.org/rfc/rfc2617.txt

[35] jQuery. Web site at http://jquery.com/

[36] Ruby on Rails. Web site at http://rubyonrails.org/