Grant agreement no.: *248605*

D9 – Interface modules

Editor: **Marc BOURDEAU** **CSTB**

Contributor(s): **Borja TELLADO** **TECNALIA**
Mauro DRAGONE **UCD**
Toon Van CRAENENDONCK **PCL**
Vincent GAY **THALES**
Marc BOURDEAU **CSTB**
Alexis BOISSONNAT **CSTB**

Issue Date	30 September 2011
Deliverable Number	D9
WP	WP5: System Integration
Status	☐Draft ☐Working ☐Released ☑Delivered to EC ☐Approved by EC

Dissemination level	
X	**PU** = Public
	PP = Restricted to other programme participants (including the Commission Services)
	RE = Restricted to a group specified by the consortium (including the Commission Services)
	CO = Confidential, only for members of the consortium (including the Commission Services)

Document history			
V	***Date***	***Author***	***Description***
1.0	***26/09/2011***	***M. Bourdeau***	***Deliverable for internal review by the Project Coordinator***
1.1	***30/09/2011***	***M. Bourdeau***	***Final version to be delivered to the EC***

Disclaimer

Summary

This deliverable, produced at month 18 of the project, results from the work achieved in task T5.1 “Interfaces development” which is the first task of WP5 dealing with system integration.

The main goal of this Work Package is the integration of the main components of the FIEMSER system (Monitoring & Control System Manager, Intelligent Control System, User Interface, and FIEMSER Data Base). It is in charge of updating the specification of the communication between components drafted in D4 (System Architecture), developing the interfaces that allow their integration, designing testing protocols and validating the integration.

The main objective of task T5.1 is to refine the design of the architecture developed in task T1.3, and specify the interfaces between components, by considering the latest development of the FIEMSER components achieved in WP2, WP3 and WP4.

The work addressed in this task consisted in:

- Refine the FIEMSER network topology in case of a multi-dwelling building
- Detail the communication mechanisms within the FIEMSER system (Web Service-based synchronous communication, and OSGi-based asynchronous communication of events)
- Describe the interactions between FIEMSER components, in terms of required and provided services, as well as published and subscribed events.

The specifications resulting from this task T5.1 will be directly used by the following activities of WP5 devoted to the development of the interfaces to achieve the system integration.

Contents

SUMMARY **3**
ABBREVIATIONS **5**
FIGURES **7**
TABLES **8**
1. INTRODUCTION **9**
1.1 DELIVERABLE CONTENT 9
1.2 RELATION TO OTHER TASKS 9
2. UPDATE OF ARCHITECTURAL APPROACH **11**
2.1 NETWORK TOPOLOGY 11
2.2 SYSTEM ARCHITECTURE & COMMUNICATION BETWEEN COMPONENTS 12
2.2.1 SOA, Web Services, OSGi *12*
2.2.1.1 Service-Oriented Architecture (SOA) 12
2.2.1.2 Web Services 13
2.2.1.3 OSGi 14
2.2.2 Principles of communication between FIEMSER components *14*
2.2.2.1 WS-based synchronous communication 14
2.2.2.2 OSGi-based asynchronous communication 15
2.2.3 XML schema for http/REST communication *16*
2.2.4 Modelling of events / Events API *18*
3. INTERACTIONS BETWEEN COMPONENTS **22**
3.1 BMCN (BUILDING MONITORING & CONTROL NETWORK) 23
3.2 ICS (INTELLIGENT CONTROL SYSTEM) 23
3.3 UI CSS (USER INTERFACE CORE SERVICES SERVER) 28
3.3.1 For the Facility Manager UI *29*
3.3.2 For the End-User UI *32*
3.4 DATA BASE 37
4. DEVELOPMENT ENVIRONMENT **38**
5. CONCLUSION **39**
ACKNOWLEDGEMENTS **40**
REFERENCES **41**

Abbreviations

API	Application Programming Interface
BMCN	Building Monitoring and Control Network
CORBA	Common Object Request Broker Architecture
CRUD	Create/Read/Update/Delete
CXF	Apache CXF is an open source services framework
DB	Data Base
D-OSGi	Distributed OSGi
FIEMSER	Friendly Intelligent Energy Management System for Existing Residential Buildings
H&S	Hardware & Software
HTTP	HyperText Transfer Protocol
ICS	Intelligent Control System
IP	Internet Protocol
JSON	JavaScript Object Notation
LAN	Local Area Network
oBIX	open Building Information eXchange
OSGi	Open Service Gateway initiative
REST	REpresentational State Transfer
R-OSGi	Remote OSGi
RPC	Remote Procedure Call
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol (originally)

SVN	Apache Subversion
UI CSS	User Interface Core Services Server
UML	Unified Modelling Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WLAN	Wireless Local Area Network
WS	Web Service
WSDL	Web Service Definition Language
XML	eXtended Markup Language
YAML	YAML Ain't Markup Language

Figures

FIGURE 1 - INTERDEPENDENCIES AMONG WORK PACKAGES 9
FIGURE 2 - INTERDEPENDENCIES AMONG WP5 TASKS 10
FIGURE 3 - FIEMSER NETWORK TOPOLOGY 12
FIGURE 4 - EVENT COMMUNICATION ARCHITECTURE 14
FIGURE 5 - EVENT COMMUNICATION BETWEEN "REMOTE" OR "DISTRIBUTED" OSGI PLATFORMS 15
FIGURE 6 - EXAMPLE OF HTTP/REST COMMUNICATION WITH THE FIEMSER DATABASE 17
FIGURE 7 - COMMON XML SCHEMA FOR HTTP/REST COMMUNICATION IN FIEMSER (TEMPORARY VERSION)... 18
FIGURE 8 - BACKUS-NAUR FORM FOR THE BMCN EVENT'S TOPIC 19

Tables

TABLE 1 - BMCN'S EVENT DESCRIPTION 20
TABLE 2 - CHOSEN PLATFORMS / FRAMEWORKS FOR FIEMSER COMPONENTS 38

1. Introduction

1.1 Deliverable content

This deliverable, produced at month 18 of the project, results from the work achieved in task T5.1 “Interfaces development” which is the first task of WP5 dealing with system integration.

The main goal of this Work Package is the integration of the main components of the FIEMSER system (Monitoring & Control System Manager, Intelligent Control System, User Interface, and FIEMSER Data Base). It is in charge of updating the specification of the communication between components drafted in D4 (System Architecture), developing the interfaces that allow their integration, designing testing protocols and validating the integration.

The main objective of task T5.1 is to refine the design of the architecture developed in task T1.3 (WP1), and specify the interfaces between components, by considering the latest development of the FIEMSER components achieved in WP2, WP3 and WP4 (see below “relation to other tasks”). This is a key task because a successful integration between system components requires a detailed definition of the various communication mechanisms and associated rules (protocol, syntax…). Besides, it also requires a common understanding of the semantic information passed between system components. This is achieved thanks to the FIEMSER Data Model shared between all modules.

Due to the work progress between the delivery of D4 (System Architecture) and D5 (FIEMSER Data Model), and this deliverable, substantial changes have been made in the FIEMSER Data Model, at level of both object properties and associated methods. These changes have been reported in the UML description of the Data Model, which is an internal project resource shared between partners, also used for the generation of Java objects.

Additionally, a common project structure has been fixed in order to avoid integration problems between software components that are developed by different partners.

1.2 Relation to other tasks

The interaction of WP5 with the rest of work packages is summarized by the following figure:

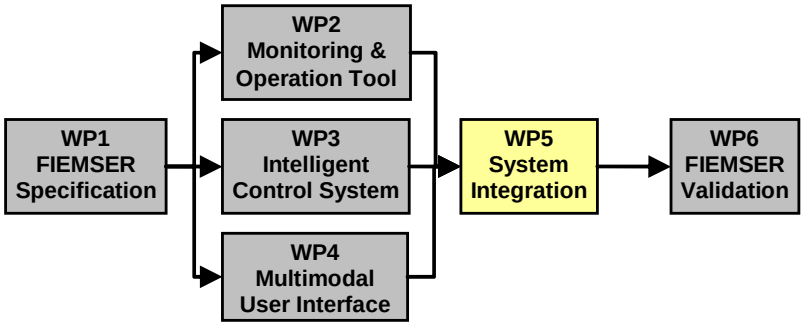

Figure 1 - Interdependencies among work packages

WP1 defined the FIEMSER technical specification, which is used as input by WP2, WP3 and WP4, which run in parallel and develop the main components of the FIEMSER system: the monitoring & operation network and the data base (WP2), the intelligent control system (WP3) and the multimodal GUI (WP4).

The results of these parallel work packages are integrated in WP5 to provide the FIEMSER system that will be validated in WP6.

This deliverable (D9) is the first one to be produced by WP5 and takes as main inputs two deliverables that were developed in WP1 (*D4 - System Architecture* and *D5 - FIEMSER data model*) and the interaction with WP2, WP3 and WP4 development activities.

"D4 - System Architecture" defines the overall FIEMSER reference architecture, specifies the interface of the GUI with the rest of FIEMSER modules and defines system development and operation environment (operating system, programming languages, etc.) and "***D5 - FIEMSER Data Model"*** complements it with the detailed definition of FIEMSER data model.

WP2, WP3 and WP4 development activities provide a more detailed analysis about the implementation and APIs of each FIEMSER module.

D9 deliverable is the main output of "Task 5.1 - Interfaces development" and will fix the FIEMSER module interfaces. This deliverable will be the main input, together with the FIEMSER modules implementation, to "Task 5.2 - Pair integration and testing of components". The following figure shows the interdependencies among WP5 tasks:

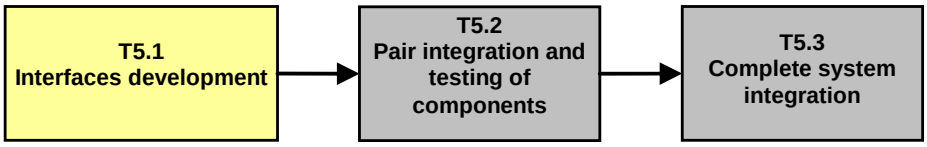

Figure 2 - Interdependencies among WP5 tasks

2. Update of architectural approach

2.1 Network topology

In WP1, two scenarios were defined: “Isolated Home Scenario” and “Multiple Dwelling Scenario”. In Task 5.1 we have selected the “Multiple Dwelling Scenario” as the reference one because it includes and extends the “Isolated Home Scenario” requirements. In this scenario, we anticipate that a FIEMSER system is deployed within a building for the energy monitoring and management of several apartments. In such a scenario, we depict hereafter the main FIEMSER components which are deployed, as well as the communication channels between them:

- FIEMSER components (H&S)
  - 1 BMCN per dwelling: the Building Monitoring and Control Network (BMCN) consists of a Box which bridges between the multiple sensors and actuators deployed in the dwelling on the one hand, and the FIEMSER server on the other hand.
  - 1 centralized Database, ICS and UI CSS which are embedded on a single server machine

    Note: the FIEMSER Server can be located in the building and connected on a private IP LAN with all Boxes. An alternative is that the FIEMSER Server is in charge of multiple dwellings situated in multiple buildings. In that case, the FIEMSER Server is located in a backbone infrastructure and connects through the WLAN/Internet and a Proxy to the Boxes located in each of the buildings.

- Communication channels
  - Between Box and FIEMSER Server: this is done through synchronous/asynchronous communications over the IP/Ethernet LAN, using respectively HTTP-REST or OSGi event interfaces.
  - Between Box and Sensors/Actuators: this is done through embedded drivers for each of the underlying communication protocols, e.g., KNX, Modbus, Zigbee, or any third-party protocol.
  - Internally within FIEMSER Server between ICS, CSS, and DB: this is done through synchronous/asynchronous communications using respectively HTTP-REST or OSGi event interfaces.

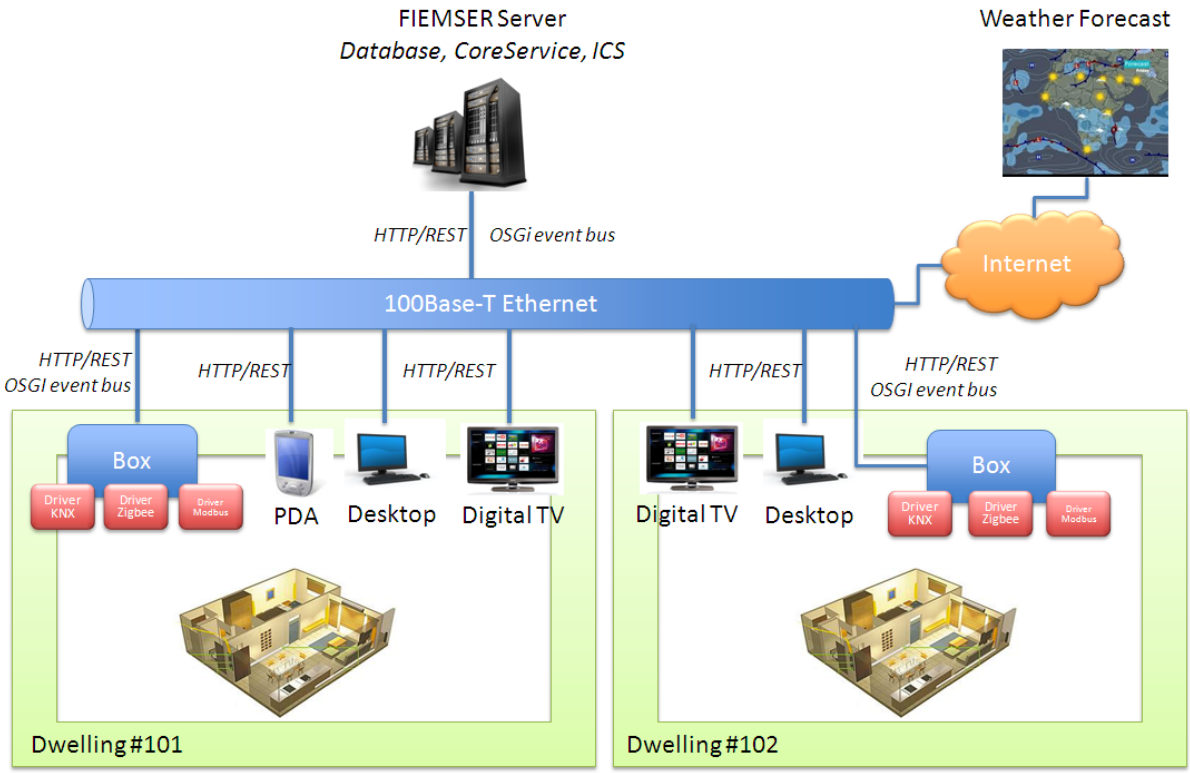

Figure 3 - FIEMSER network topology

2.2 System architecture & Communication between components

2.2.1 SOA, Web Services, OSGi

2.2.1.1 Service-Oriented Architecture (SOA)

As stated in deliverable D4 “FIEMSER System Architecture”, the FIEMSER system relies on the Service-Oriented Architecture (SOA) paradigm with the definition of modular service interfaces. Service Oriented Architecture (SOA) is a distributed computing paradigm in which business functionality is provided by autonomous systems called services, which are exposed in a network infrastructure through well-defined interfaces. This allows building complex yet flexible systems as well as reusing application logic through the composition of services. The key advantage of a SOA approach is to offer modularity, isolation, flexibility, loose-coupling, and interoperability, among a large-scale of heterogeneous networked devices.

SOA is a concept which is not tied to a particular technology. However, Web Services are currently the preferred communication method to deliver interoperable SOA. This is why we chose this technology for FIEMSER.

Additionally, we also chose the OSGi framework as an integration platform, since it is well adapted to embedded and mobile devices (see below). Besides, OSGi contributes to our global SOA approach in the sense that it uses a modular and service-oriented model.

2.2.1.2 Web Services

Web services constitute application programming interfaces (API) or Web APIs that are accessed via Hypertext Transfer Protocol (HTTP) and executed on a remote system hosting the requested services.

Web Services can be used in different ways. Three main types of Web Services are generally considered, based on respectively: XML-RPC, SOAP, and REST.

1. XML-RPC and SOAP

These two protocols can be grouped since they follow the same architectural "philosophy".

XML-RPC is a remote procedure call (RPC) protocol which uses XML to encode its calls and HTTP as a transport mechanism.

SOAP (originally Simple Object Access Protocol) is the traditional, standards-based approach for developing Web Services. It specifies a protocol (over HTTP) for exchanging structured information (XML-based messages) in the implementation of Web Services.

The principle of both protocols is to call remote business methods by using application-specific vocabulary, which tends to say that there are not generic protocols. In SOAP, these methods are defined in the WSDL (Web Services Description Language) contract.

2. REST

In reaction to the more heavy-weight SOAP-based standards, modern Web Services are moving from SOAP-based services towards Representational State Transfer (REST) based communications.

REST is not a protocol but an architecture style for designing networked applications. The idea is that, rather than using complex mechanisms such as CORBA, RPC or SOAP to connect between machines, simple HTTP is used to make calls between machines (point-to-point communication). Contrary to SOAP, which follows a method-oriented model, REST is resource-oriented. The focus is on interacting with resources, rather than messages or operations.

A RESTful Web Service is a simple Web Service implemented using HTTP and the principles of REST. It is a collection of resources, with three defined aspects:

- The base URI for the Web Service, such as http://example.com/resources/
- The internet media type of the data supported by the web service. This is often JSON, XML or YAML but can be any other valid internet media type.
- The set of operations supported by the Web Service. RESTful applications use simple HTTP requests to post data (create and/or update), read data (e.g., make queries), and delete data (CRUD=Create/Read/Update/Delete).

So, unlike SOAP-based Web Services, RESTful Web Services do not require WSDL service-API definitions.

2.2.1.3 OSGi

As presented in deliverable D4, the FIEMSER platform is based on the OSGi (Open Services Gateway initiative) framework. Applications are modularized in the form of bundles that can be remotely installed, started, stopped, updated and uninstalled without requiring a reboot. These bundles are developed by using the JAVA language.

The OSGi Alliance has specified many services. Amongst them, one of particular interest for FIEMSER is the **Event Admin** Service, which provides an inter-bundle communication mechanism based on a publish-and-subscribe model to handle events.

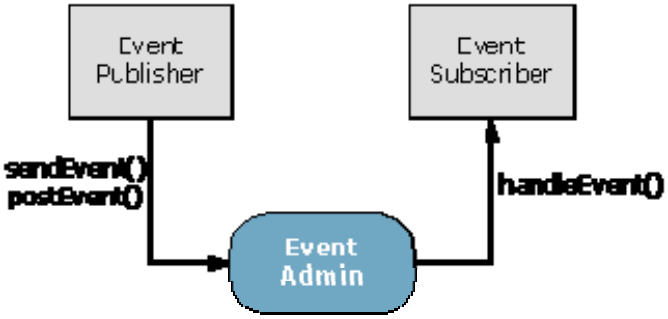

Figure 4 - Event Communication Architecture

2.2.2 Principles of communication between FIEMSER components

To support use cases, two types of communication between FIEMSER components should be considered:

- Synchronous communication: this will be achieved through a Web Services layer developed for each main FIEMSER component.
- Asynchronous communication: this will be achieved through OSGi publish/subscribe mechanism allowing notification of events to other components.

2.2.2.1 WS-based synchronous communication

For WS-based synchronous communication, any of the above-mentioned architectures can be chosen for each FIEMSER module. However, the REST architecture will be preferably implemented because it is a lighter and more extensible solution. No WSDL-like description of the services is required for REST. What is needed is to define the XML schema used for exchanging data (see “2.2.3 XML schema for http/REST communication”).

2.2.2.2 OSGi-based asynchronous communication

For asynchronous communication, the following schema illustrates how OSGi events publish/subscribe mechanism will be implemented for FIEMSER. Since the FIEMSER architecture is distributed on several computers (boxes + main server), we need some means to pass events between remote OSGi platforms. This will be achieved through the distributed OSGi extension R-OSGi[1].

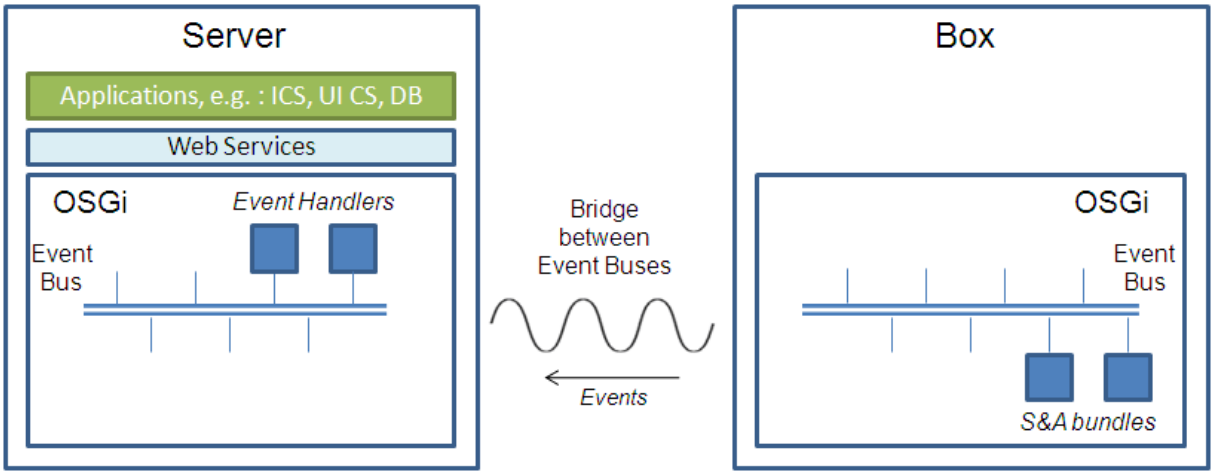

Figure 5 - Event communication between "remote" or "distributed" OSGi platforms

R-OSGi provides a transparent way to access services on remote OSGi platforms. All that a service provider framework has to do is registering a service for remote access. Subsequently, other peers can connect to the service provider peer and get access to the service. Remote services are accessed in an entirely transparent way. For every remote service, a local proxy bundle is generated that registers the same service. Local service clients can hence access the remote service in the same way and without regarding distribution.

Additionally, R-OSGi provides a bridge that allows seamless communication of events between distant computers. As a consequence, all events that are published on one machine are seen on the remote one as if they have been produced locally. On each box side (BMCN module), OSGi bundles will be in charge of publishing the events that occur on the network (e.g. a light is switched on). These events will be automatically notified on the Event Bus of the remote OSGi framework (server side). Application-specific bundles (Event Handlers) will be in charge of calling the relevant web services in response to the events which they would

[1] The other distributed extension of OSGi, D-OSGi, specifically targets Web Services and poses a much bigger overhead - 10MB - compared to R-OSGi Apache CXF implementation, which is 230 KB. R-OSGi can be deployed in minimal OSGi implementations, such as Concierge, targeting computationally constrained devices.

have subscribed to. There will be typically one Event Handler per FIEMSER component (i.e. one for the Data Base, one for the ICS, and another one for the UI CSS).

Even though R-OSGi is a sophisticated middleware for OSGi frameworks, it uses a very efficient network protocol and has a small footprint. This makes it ideal for small and embedded devices with limited memory and network bandwidth. The service runs on every OSGi-compliant environment. R-OSGi has been tested with Eclipse Equinox, Knopflerfish, and Oscar / Apache Felix.

2.2.3 XML schema for http/REST communication

We chose to use the XML language for the description of resources data returned by the RESTful Web Services developed in FIEMSER (see "2.2.1.2 Web Services").

In order to ensure an easy use of these Web Services, and allow the processing (parsing) of the returned XML streams, it is needed to define an XML schema[2] for each HTTP/Rest call. In fact, since many calls are similar in the sense that the returned information embeds more or less the same structure of data, it is possible to define a common XML schema to address, even not all communications with Web Services, at least a good part of them. Only the most complex Web Services would then request the definition of specific XML schemas.

The following figure gives an example of http/REST communication between a FIEMSER component (e.g. the ICS) and the FIEMSER Database. In this example, the user (represented by the ICS module) wants to get the details of the building corresponding to the id '123'.

[2] XML Schema as a recommendation published by the W3C in May 2001 is a language for describing XML document format that defines the structure and the content type of an XML document. This definition can include checking the validity of this document.

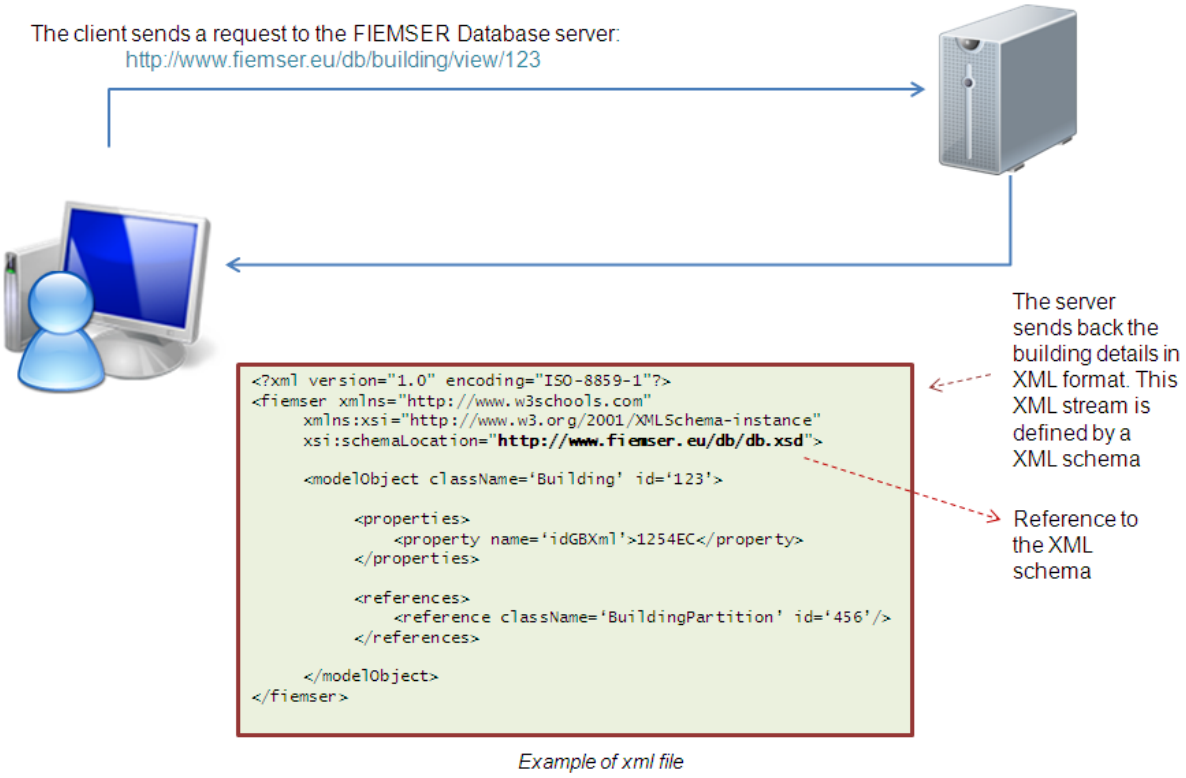

Figure 6 - Example of http/REST communication with the FIEMSER database

A provisory version of the common XML Schema is given hereafter. This schema will have to be validated against the set of Web Services to be provided by the FIEMSER components, and the Database in the first place.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.w3schools.com"
      xmlns="http://www.w3schools.com">

   <xs:element name="fiemser">
      <xs:complexType>
         <xs:sequence>

            <!-- Define ModelObject (top level) -->
            <xs:element name="modelObject">
               <xs:attribute name="className" type="xs:string" use="required"/>
               <xs:attribute name="id" type="xs:integer" use="required"/>

               <!-- ModelObject contains some basics properties-->
               <xs:element name="properties">
                  <xs:sequence>
```

```
                <xs:element name="property">
                  <xs:attribute name="name" type="xs:string" use="required"/>
                </xs:element>
              </xs:sequence>
            </xs:element>

            <!-- And reference some complexes objects -->
            <xs:element name="references">
              <xs:sequence>
                <xs:element name="reference">
                  <xs:attribute name="classNam" type="xs:string" use="required"/>
                  <xs:attribute name="id" type="xs:integer" use="required"/>
                </xs:element>
              </xs:sequence>
            </xs:element>
          </xs:element>


        </xs:sequence>
      </xs:complexType>
    </xs:element>

</xs:schema>
```

Figure 7 - Common XML Schema for http/REST communication in FIEMSER (temporary version)

2.2.4 Modelling of events / Events API

The BMCN will use the Event Admin OSGi service to broadcast the events generated by any sensor and actuator network administered by the BMCN.

The Event Admin service provides an inter-bundle communication mechanism based on an event *publish* and *subscribe* model. An Event object has a topic, defining the *type* of the event, and a set of properties associated to the event. The topic is intended to serve as a first-level filter for determining which handlers should receive the event and also optimize the dispatching of the events to the handlers. Topics are arranged in a hierarchical namespace. Each level is defined by a token and levels are separated by slashes. Topics should be designed to become more specific when going from left to right.

Event handlers must be registered as services with the OSGi framework under the object class *org.osgi.service.event.EventHandler*. Event handlers should be registered with a property (constant from the EventConstants class) EVENT_TOPIC. Handlers can provide a prefix (which can terminate with the wildcard token ('*'\u002A) to specify to which topics or sub-group of topics they want to subscribe. Each Event Handler is notified for any event which belongs to the topics the handler has expressed an interest in. In addition, event handlers can

be registered with a service property named EVENT_FILTER. The value of this property must be a string containing a LDAP filter. Any of the event's properties can be used in the filter expression. For example, the filter '(&(temperature<18)(temperature>=22))' filters all temperature updates where the temperature is less than 18 degrees and more than or equal 22.

Figure 6 represents the description in Backus-Naur[3] form of the topics used by the BMCN to implement the notification service specified in D4. The first token of the topic ("Fiemser") is used to identify all the events on the OSGi bus addressed to FIEMSER components. Other values may be reserved in the future to handle non-application events, for instance, used for debugging and testing purpose. The second token identifies the instance of the BMCN (in the case of a multi-dwelling scenario there will be multiple instances connected to the R-OSGi). The third token specifies the nature of the events, i.e. if the event is an update on the status of a device, the characteristics of a newly installed device, or a data update. The remaining tokens in the topic helps addressing the source of the event at increasing levels of resolution, respectively (i) all the devices connected to the BMCN, (ii) a specific building area, (iii) the devices of a specific type, (iv) one specific device. Ctrl Devices may include both sensors and actuators.

```
<Topic> ::= "/" "fiemser" "/" <BMCNID> "/" <EventType> "/" <Address>
<EventType> := "status" | "discovery" | "data" | "behaviour"
<Address> :=  ("*" | <BuildingLevelAddress>)
<BuildingLevelAddress> :=  <BuildingAreaID> "/" (* | <CtrlDeviceTypeLevelAddress>)
<CtrlDeviceTypeLevelAddress > :=  <CtrlDeviceTypeID> "/"(*|<CtrlDeviceLevelAddress>)
<CtrlDeviceLevelAddress > :=  <CtrlDeviceID>
<CtrlDeviceTypeID> := <ActuatorType> | <SensorType>
<ActuatorType> := "Switch" | "Dim" | "HVAC" | "Thermostat" ...
<SensorType> := "Humidity" | "Temperature" | "Light | "Energy" |

<BMCNID> := Numeric     {1..numbers of BMCN}
<BuildingAreaID> := <Text>
<CtrlDeviceID> := <Text>
```

Figure 8 - Backus-Naur form for the BMCN event's topic

Examples:

The following are some examples of BMCN event topics.

1) Topic = /fiemser/bmcn1/*, to subscribe to all type of notifications originated by the BMCN in the first apartment.

[3] A notation technique for context-free grammars, often used to describe the syntax of languages used in computing, see http://en.wikipedia.org/wiki/Backus-Naur_Form.html .

2) Topic = /fiemser/bmcn2/status/*, to subscribe to all status update events originated by the bmcn in the second apartment.

3) Topic = /fiemser/bmcn1/discovery/kitchen/*, to subscribe to all discovery updates originated from the kitchen of the first apartment.

4) Topic = /fiemser/bmcn1/data/kitchen/temperature/* to receive all the temperature updates from any temperature sensor installed in the kitchen of the first apartment[4].

5) Topic = /fiemser/bmcn1/status/kitchen/temperature/TOven1, to subscribe to all the status update events originated by a thermometer installed near the oven in kitchen of the first apartment.

6) Topic = /fiemser/bmcn1/behaviour/kitchen/*, to subscribe to all the behaviour notification originated from the kitchen of the first apartment.

Table 1 lists the properties used to describe each event for all event types supported by the FIEMSER's BMCN components. The second column lists all the properties associated to each event and describes their possible values with a Backus-Naur formula. In addition, each event object contains an Address property specifying a building level address used to identify the source of the event.

Table 1 - BMCN's Event Description

Event Types	Description and Properties
Status	Raised every time the status of an existing device changes Address::= <BuildingLevelAddress>//specifies building area and device Status::= "DISABLED" | "ACTIVE" | "FAILURE" ErrorCode::= <Numeric>
Discovery	Raised when a new device is connected to the BMCN for the first time Address::==<BuildingLevelAddress>//specifies building area and device + device dependent properties (e.g. unit of measure, accuracy, measurement range, etc.)
Data	Raised whenever a device generated new sensor data Address:==<BuildingLevelAddress>//specifies building area and device Value ::= <Numeric>
Behaviour	Raised when the value (e.g. on/off) of an existing device changes after the user has triggered the activation of a behaviour specified at the BMCN level. Address:==<BuildingLevelAddress>//specifies building area and device affected by the behaviour Status::="NEW" | "ACCEPTED" | "REJECTED" // specifies if this is a

[4] A client would need to subscribe to this topic with the LDAP filter '(temperature<18) ' to be notified only when the temperature drops below 18 degree Celsius.

	new activation of a behaviour, or if it is subsequent notification of user acceptance/rejection of the ICS's override. Value ::= <Numeric> // specifies the new value set for the device InstanceBehaviourID ::= <Numeric> // the id of the behaviour instance. It can be used to provide a configuration to override the one that was automatically set by behaviour

3. Interactions between components

This section describes, for each FIEMSER component:

- Interfaces/Services (required and/or provided by this component), with reference to UML methods and associated http/REST calls. Including a short definition.
- Events (published or subscribed). Including a short definition. In case of subscribed events, also including the http/REST call used in the Event Handler to process the event.

Following templates are used to report this information.

Provided services

UML method	http/REST call

Required services

Targeted component	E.g. Database
UML method	**http/REST call**

Published events

Event message

Subscribed events

Event message	http/REST call

3.1 BMCN (Building Monitoring & Control Network)

Besides the event services described in Section 2.2.4, the BMCN provides the following service via both R-OSGi and HTTP/REST.

Provided Services

UML method	HTTP/REST call
Configure (in:String, in:String, in:ConfigurationInfo) Definition: Configures a given device, following a *ConfigurationInfo* structure that describes the required configuration**.** I#1: deviceId I#2: behaviourInstanceId I#3: ConfigurationInfo object If the behaviourInstanceId is -1, this request is a standard request. If the behaviourInstanceId is > 0, this request is used to override what was set automatically by the BMCN after a behaviour was triggered by the user. Upon reception of this request, the BMCN will ask the user if he/she wants to accept/reject the override. In the case of acceptance, the BMCM will cancel the effect of the behaviour and set the configuration specified in the override request. In case of rejection, the BMCN will ignore the override request. In both cases, acceptance/rejection will be notified via the Event service described in Section 2.2.4.	http://www.fiemser.eu/bmcnX/configuration?deviceId=123&behaviourInstanceId=123&configurationType=...&parameter1=...

3.2 ICS (Intelligent Control System)

The ICS module as well as other modules of the FIEMSER platform exposes two kinds of interfaces: WS (Web Service) interface and OSGi events interface.

- **WS interface**: The ICS WS interface is developed using Jetty embedded web application server.

The approach of using a embedded server inside the ICS module allows to follow a loosely coupled approach during the FIEMSER platform development. For WS architecture different approaches have been studied but the HTTP REST approach has been chosen. The choice of HTTP REST as WS development approach has been supported by two main reasons.

1. ***Simple to develop and upgrade***: Based on a standardized way of service declaration (URI naming) and information exchange methods, operations are implicitly defined in the calls, makes easy to upgrade software platforms.

2. ***Performance:*** Due to its low resource requirement is especially valid for environments in with light components are required. It also facilitates very low coupled architectures.

- **OSGi interface:** The OSGi interface acts as gateway between the ICS module and the R-OSGi framework. The OSGi interface, developed as a bundle, mainly has the role of forwarding S&A module events to the ICS module. In addition, it also implements the mechanism of publishing commands targeting the S&A module.

Next tables describe the functions that compose the above introduced interfaces.

Provided Web Services

UML method	HTTP/REST call
setDeviceValue (in:String) Definition: Used to update the ICS values with the published device Events. It can be used by different components to update ICS runtime values.	http://www.fiemser.eu/ics/eventInfo?idEventSource='XX'&date='yyyy-mm-ddT24hh:mm:ss'&value='XX'

Required Web Services

Targeted component	CSS
UML method	**HTTP/REST call**
notifyNewAdvice (in:String, in:Advice) Definition: Used to inform the end-user of a new advice. I#1: User Id I#2: Advice object O: none	http://www.fiemser.eu/css/notify_advice?userID=123&advice=Advice

Targeted component	Database
UML Method	**HTTP/REST call**
getBuilding () Definition: This function retrieves the required information to build the "Building" object. Building is the main information container object. It has the description of the building environment where the FIEMSER platform will be deployed	http://www.fiemser.eu/db/building/view/123
getBuildingProgrammedSchedule (in:String, in:Date, out:BuildingPartitionSchedule) Definition: For certain building partition id retrieves and reference date returns its building partition schedule. I#1: Building partition id. I#2: Date to be used for schedule fetching O: BuildingPartitionSchedule object	http://www.fiemser.eu/db/buildingProgrammedSchedule/view?buildingPartitionId=123&date=20110101

getScheduleByType (in:String, in:Date, out:List<Schedule>) Definition: For certain device type and date it retrieves their Schedule list. I#1: Device type for which the schedule is wanted I#2: Date to be used for schedule fetching O: List of Schedule objects that belong to that type of devices	http://www.fiemser.eu/db/schedule/list?typeId=123&date=20110101
getBuildingZones(in:String, out:List<BuildingZone>) Definition: For certain building id retrieves its BuildingZones I#1: Building partition id. O: List of BuildingZone objects that belong to the specified building partition	http://www.fiemser.eu/db/buildingZone/list?buildingId=123
getBuildingSpaces (in:String out:List<BuildingSpace>) Definition: For certain building id its spaces are retrieved I#1: Building zone id. O: List of BuildingSpace objects that belong to the specified building zone	http://www.fiemser.eu/db/buildingSpace/list?buildingZoneId=123
getDeviceList (in:String, out:List<Device>) Definition: Function used to fetch the device list associated to certain building space. I#1: Building space id. O: List of Device objects that belong to the specified building space	http://www.fiemser.eu/db/device/list?buildingSpaceId=123
getDeviceListByType (in:String, in:String, out:List<Device>) Definition: This function is used to retrieve Device list that belong to a certain building space and also to a certain device type I#1: Building space id. I#2: Device type identification. O: List of Device objects that belong to the specified building space	http://www.fiemser.eu/db/device/list?typeId=123&buildingSpace=123
getHomeUsageProfile (in:String, in:Date out:HomeUsageProfile) Definition: This function retrieves the HomeUsageProfile associated to certain building g partition id and date. I#1: Building partition identification. I#2: Date used as reference to fetch the HomeUsageProfile. O: HomeUsageProfile object that belongs to the specified building partition and date.	http://www.fiemser.eu/db/homeUsageProfile/view?buildingPartitionId=123&date=20110101
getZoneUsageProfile (in:String, out:List<ZoneUsageProfile>) Definition: Function that for a certain home usage profile id retrieves its ZoneUsageProlile list. I#1: Home usage profile identification. O: List of ZoneUsageProfile objects that belong to the specified HomeUsageProfile.	http://www.fiemser.eu/db/zoneUsageProfile/list?homeUsageId=123
getApplianceUsage (in:String, in:String, out:ApplianceUsage) Definition: Function that having as inputs the scene id and the appliance id retrieves its usage I#1: Scene identification. I#2: Appliance identification O: ApplianceUsage object that belongs to the specified scene and appliance	http://www.fiemser.eu/db/applianceUsage/view?sceneId=123&applianceId=123

getComfortSettings (in:String, out:ComfortSettings) Definition: Function that of a scene id gets its associated comfort settings I#1: Scene identification. O: ComfortSettings object that belongs to the specified scene and appliance	http://www.fiemser.eu/db/comfortSettings/view?sceneId=123
getControlRules (in:String, out:ControlRule) Definition: For certain building partition id retrieves its control rules I#1: Building partition identification. O: ControlRule object that belongs to the specified building partition	http://www.fiemser.eu/db/controlRule/view?buildingPartitionId=123
getUserInformation (in:string out: List<User>) Definition: Function that for a certain BuildingPartition retrieves the Users associated to it I#1: Building partition identification. O: List of User objects linked to certain building partition. In addition user roles and granted privileges are also retrieved	http://www.fiemser.eu/db/permissions/list?buildingPartitionId=123
getDataLog (in:String, in:Date, out:DataLog) Definition: Function that for certain device id and date retrieves its DataLog I#1: Device identification. I#2: Date used to fetch device DataLog information O: DataLog object that belongs to the specified device and date	http://www.fiemser.eu/db/dataLogs/list?deviceId=123&date=20110101
getOperationMode (in:String, out:OperationMode) Definition: Function that for certain device id gets its OperationMode I#1: Device identification. O: OperationMode object that belongs to the specified device	http://www.fiemser.eu/db/operationMode/view?deviceId=123
getWeatherForecast (in:Date, out:WeatherForecast) Definition: Function that for certain date retrieves its weather forecast. If the date is the current day the weather forecast returned will be the available last update. I#1: Date to use as reference to fetch the weather forecast O: WeatherForecast object that belongs to that date	http://www.fiemser.eu/db/weatherForecast/view?date=20110101
getScenes (in: String, out:List<Scene>) Definition: Function that retrieves the scenes related to the specified building zone. I#1: Building zone identification O: List of Scene objects that apply to the specified building zone	http://www.fiemser.eu/db/scene/list?buildingZoneId=123
getDayAheadPrices (in:Date, out:DayAheadPrices) Definition: Function that for a specified date retrieves the applicable energy prices. I#1: Date used as reference to fetch the energy prices. If the current date is used the prices applicable for day are returned O: DayAheadPrices related to the used date.	http://www.fiemser.eu/db/dayAheadPrces/view?date=20110101
getHomeLoads (in:String, out:List<HomeLoad>) Definition: Function that for a specified building space retrieves its home load list. I#1: Building space identification O: List of HomeLoad objects associated to a building space	http://www.fiemser.eu/db/homeLoad/list?buildingSpaceId=123

getDeviceById (in:String, out:Device) Definition: Function that retrieves the Device object related to a certain device id. I#1: Device identification O: Device object related to the specified device identification	http://www.fiemser.eu/db/device/view/123
getBuildingPartitions (in:String, out:List<BuildingPartition>) Definition: Retrieves the Building partitions of the specified Building I#1: ID building O: List of Building partitions (objects) linked to the specified building	http://www.fiemser.eu/db/buildingPartition/list?buildingId=123
getAllHomeUsageProfiles (in:String, out:List<HomeUsageProfile>) Definition: Retrieves the set of Home usage profiles defined for the specified Building partition I#1: Building partition Id O: List of HomeUsageProfiles linked to the specified Building partition	http://www.fiemser.eu/db/homeUsageProfile/list?buildingPartitionId=123
getLoadsforScene (in:String, out:ApplianceUsage) Definition: Based on certain scene identification its ApplianceUsage is retrieved; this object stores the load usage. I#1: Scene Id O: ApplianceUsage object that stores the loads usage.	http://www.fiemser.eu/db/view/Scene/123
storeSchedule (in:String, in:ResourceSchedule) Definition: This function is used to store in the DB the calculated resource schedule for certain device or resource. For example this function will be used to store the output of the E+ related with the renewable energy generation sources schedule. I#1: Building partition Id I#2: ResourceSchedule structure to be formatted and sent to the DB	http://www.fiemser.eu/db/schedule/new?buildingPartitionId=123&resourceScheduleId=123
addAdvice (in:Advice) Definition: creates a new advice to be stored I#1: Advice O: none	http://www.fiemser.eu/db/advice/add?advice=Advice

Published events

Event message
Topic= /fiemser/<Address>/data/<BuildingAreaID>/Actuator Type/<CtrlDeviceID> This topic is used by the ICS when it is necessary to send a order or command to one or certain devices.

Subscribed events

Event message	http/REST call
Topic = /fiemser/bmcnX/discovery/* The ICS uses this topic to subscribe to events related with new device connections. This information will be used by the ICS to fetch the database looking for device specific data such as ConsumptionProfile, new scene configurations, etc.	Associated http/REST call to ICS will create a new device in the internal ICS memory space. This new device will read its configuration or running parameter from the DB.
Topic=/fiemser/bmcnX/behaviour/* Using this topic the ICS subscribes to all the behaviour status changes that may happen in any of the apartments (BuildingPartition) Other magnitudes as temperature or energy meter values will be periodically polled from the DB.	Associated http/REST call to ICS will trigger processes focused on finding a better, more energy efficient, building configurations. In response the ICS will notify building actuators reconfiguration options. This topic will be also used to inform to the ICS that the suggested action has been rejected, or not, by the "Behaviour" component. Possible values to be notified by this event are: Status::==NEW|ACCEPTED|REJECTED

3.3 UI CSS (User Interface Core Services Server)

In this chapter, we list the interfaces required and provided by the Core Services Server (CSS).

WS interface: description of the methods provided / required by the UI CSS

OSGi-based events: description of the events that need to be subscribed by the UI CSS, and the methods called for handling these events.

The CSS module as well as other modules of the FIEMSER platform exposes two kinds of interfaces: WS (Web Service) interface and OSGi events interface.

- ***WS interface****: The ICS WS interface is developed using Jetty embedded web application server.*

- ***OSGi interface:*** *The OSGi interface acts as gateway between the CSS module and the R-OSGi framework. The OSGi interface, developed as a bundle, mainly has the role of forwarding S&A module events to the CSS module. In addition, it also implements the mechanism of publishing commands targeting the S&A module.*

3.3.1 For the Facility Manager UI

Required services

Related to user definition

The CSS will use methods offered by the Database in order to:

- Get authorized users per building partition
- Get details on a user
- Create/delete new user

Next table describes the interfaces described above.

Targeted component	Database
UML Method	**HTTP/REST call**
getUserByPartition (in:String out:List<User>) Definition: Function that for a certain BuildingPartition retrieves the Users associated to it I#1: Building partition identification. O: List of User objects linked to certain building partition. In addition user roles and granted privileges are also retrieved	http://www.fiemser.eu/db/user/list?buildingPartitionId=123
getUserByUserName (in:String, out:User Definition: Function that for a certain UserName retrieves the User Information associated to it I#1: User name O: User object	http://www.fiemser.eu/db/user/view?userName=John
addUser (in:User) Definition: Function that registers a new user I#1: User object O: none	http://www.fiemser.eu/db/user/add?userName=john&password=xHdJKT&buildingPartitionId=123
setUserByPartition (in:String, in:String) Definition: Function that registers a user associated to a building partition I#1: User name I#2: Building partition Id	http://www.fiemser.eu/db/buildingPartition/edit/123?userName=john

deleteUser (in:String) Definition: Function that removes a user from the database I#1: User name	http://www.fiemser.eu/db/user/delete?userName=john

Related to configuration

The CSS will use methods offered by the Database in order to:

- Get details on a device
- Get all equipments in one partition room
- Parameter details on an equipment
- Get/edit FIEMSER system parameters of the building or specific apartment

Next table describes the interfaces described above.

Targeted component	Database
UML Method	**HTTP/REST call**
getDeviceInformation (in:String, out:Device) Definition: Function that for a certain Device retrieves the Device Information associated (location, type, value, status) I#1: Device Id O: Device object	http://www.fiemser.eu/db/device/view/123
getDeviceByPlace (in:String, out:List<Device>) Definition: Function that for a certain Building partition retrieves the Devices associated to it I#1: Building partition identification. O: List of Device objects linked to the Building partition	http://www.fiemser.eu/db/device/list?buildingPartitionId=123
getEventByPlaceAndPeriod (in:String, in:Period, out:List<Event>) Definition: Function that returns all the events/measures for each device within a given building space over a given period of time I#1: Building Space identification. I#2: Period of observation O: List of Events objects	http://www.fiemser.eu/db/event/list?buildingSpaceId=123&Period=123
setDeviceConfiguration (in:String, in:ConfigurationInfo) Definition: Function that sets a specific configuration for a device I#1: Device Id I#2: ConfigurationInfo object	http://www.fiemser.eu/db/device/edit/123?configurationType=...&parameter1=...
getWeatherForecastAddress (out:String) Definition: Function that retrieves the weather forecast address I: none O: Weather forecast website IP address	http://www.fiemser.eu/db/weatherforecastservice/view

getDailyEnergyConsumptionReference (in:String, out:Float) Definition: Function that retrieves the daily energy consumption reference per apartment (i.e. building partition) I#1: Building partition Id O: Daily Energy Consumption Reference	http://www.fiemser.eu/db/buildingPartition/view/123

Related to operation

The CSS will use methods offered by the Database in order to:

- Get the operational view of the overall building or per apartment
  - Weather condition, detailed energy consumption, power generation, stored energy, energy flow
- Get the schedule for the whole building or per apartment
- Get daily performances/advices/control rules/logs in a building partition

Next table describes the interfaces described above.

Targeted component	**Database**
UML Method	**HTTP/REST call**
getWeatherForecast (in:Date) Definition: Function that returns the weather forecast for the specified date. If no date, returns by default for the current day. I#1: Date O: WeatherForecast object	http://www.fiemser.eu/db/weatherForecast/view?date=20110101
getSchedule (in:Date, in:String, out:BuildingProgrammeSchedule) Definition: Function that returns the calculated home schedule for the specified date and building partition. I#1: Date I#2: Building partition Id O: BuildingProgrammeSchedule object	http://www.fiemser.eu/db/schedule/view?date=20110101&buildingPartitionId=123
getHomePerformanceForDay (in:Date, in:String, out:HomeDailyMeasurementLog) Definition: Function that returns the EE performance for the specified date and building partition I#1: Date I#2: Building partition identification. O: HomeDailyMeasurementLog object	http://www.fiemser.eu/db/performances/view?date=20110101&buildingPartitionId=123
getBuildingPerformanceForDay (in:Date, out:BuildingDailyPerformance) Definition: Function that returns the EE performance of the whole building for the specified date I#1: Date O: BuildingDailyPerformance object	http://www.fiemser.eu/db/performances/view?date=20110101

getControlRules (in:String, out:List<ControlRule>) Definition: Function that returns the control rules chosen for the specified home usage profile I#1: HomeUsageProfile Id O: List of ControlRule objects	http://www.fiemser.eu/db/controlRules/list?homeUsageProfileId=123
getEventsByDeviceAndPeriod (in:String, in:Date, in:Date, out:List<Event>) Definition: Function that returns all the events for the specified control device during the given period I#1: Device Id I#2: Start date of the period I#3: End date of the period O: List of events	http://www.fiemser.eu/db/events?deviceId=123&start=20110101&end=20110101

Provided services

Related to configuration

The CSS will offer a method in order to:

- Push/display alarms on S&As that require maintenance

UML method	HTTP/REST call
notifyMaintenanceAlarm (in:Alarm) Definition: Used to inform the Facility Manager that an equipment requires maintenance. I: Alarm object O: none	http://www.fiemser.eu/css/notify_maintenance_alarm?alarm=123

3.3.2 For the End-User UI

Required services

Related to Home Screen

The CSS will use methods offered by the Database in order to:

- Get weather condition
- Get current energy usage (in Euro or Watt)
- Get daily advice
- Get money saved so far

Targeted component	Database
UML Method	**HTTP/REST call**
getCurrentWeather (out:List <Sensor,Value>) Definition: to display the weather information (temperature and icon on the home screen) I#1: none O: List of <Sensor, value>	http://www.fiemser.eu/db/weather/view
getHomePerformanceForPeriod (in:Date, in:Date, in:String, out:List<HomeDailyMeasurementLog>) Definition: Function that returns the EE performance for the specified period (e.g. energy usage in euro and/or watt, money saved) I#1: Init Date I#2: End Date I#3: Building Partition Id O: List of HomeDailyMeasurementLog objects	http://www.fiemser.eu/db/performances/view?date=20110101&buildingPartitionId=123&format=WattAndEuro
getAdvice (in:String, in:Date, out:List<Advice>) Definition: function that returns the advices created on the specified date (e.g. today), including the status of their activation I#1: User Id I#2: Date O: List of Advice object (including their status)	http://www.fiemser.eu/db/advice/view?userId=123&date=20110101
modifyAdviceStatus (in:String, in:Boolean) Definition: Function that modifies the status of an advice (accepted, rejected) I#1: Advice Id I#2: Status O: None	http://www.fiemser.eu/db/advice/edit/123?status=activate

Related to My House Now screen

The CSS will use methods offered by the Database in order to:

- Get overview of current room temperatures
- Get overview of set-point temperatures
- Specify individual room temperature (set comfort temperature)
- Get overview of appliances in a specific room

Targeted component	Database
UML Method	**HTTP/REST call**
getEventByPlaceAndDate (in:String, in:Date, out:List<Events>) Definition: returns all events (including temperature measurements) for each device located in this building space for a date (incl. the current date) I#1: Building space Id I#2: Date O: List of Events	http://www.fiemser.eu/db/event/list?buildingSpaceId=123&date=20110101
setControlRule (in:String, in:Boolean) Definition: used to enforce a control rule (e.g., set a comfort temperature within a room) I#1: Control Rule Id I#2: Boolean (activate, deactivate) O: none	http://www.fiemser.eu/db/control/edit/123?instruction=123
getControlRules (in:HomeUsageProfile, out:List<ControlRule>) Definition: returns all control rules (incl. comfort temperature instruction) associated to a home usage profile I#1: HomeUsageProfile Id O: List of ControlRule objects	http://www.fiemser.eu/db/control/list?homeUsageProfileId=123
getDevicesByPlace (in:String, out:List<Device>) Definition: returns all devices present within one building space I#1: Building space Id O: List of Device objects (incl. their status)	http://www.fiemser.eu/db/device/list?buildingSpaceId=123
getBuildingPartitions (in:String, out:List<BuildingPartition>) Definition: Retrieves the Building partitions of the specified Building, used to derive the floor map of an apartment (e.g., surface, opening, etc.). I#1: ID building O: List of Building partitions (objects) linked to the specified building	http://www.fiemser.eu/db/buildingPartition/list?buildingId=123

Related to Scheduler screen

The CSS will use methods offered by the Database in order to:

- Set day type mode manually for one day or for over a period(i.e., home usage profile)
- Get the list of all day type profiles
- Add new day type
- Modify one part of an existing day type

Targeted component	Database
UML Method	**HTTP/REST call**
getAllHomeUsageProfiles (in:String, out:List<HomeUsageProfile>) Definition: returns all defined home usage profiles, that is day types, with associated scenes for each day type, for the specified building partition I#1: Building partition identification O: List of HomeUsageProfile objects	http://www.fiemser.eu/db/homeUsageProfile/list?buildingPartitionId=123
setHomeUsage (in:String, in: String, in:Date) Definition: assigns an home usage profile to a specific day or for a complete period, for a given building partition I#1: Building partition Id I#2:Home Usage Profile Id I#2: Date O: none	http://www.fiemser.eu/db/calendar/edit/123?buildingPartition=123&homeUsageProfileId=123&date=20110101
addHomeUsageProfile (in:HomeUsageProfileID, in:BuildingPartitionID, in:List<Scene>) Definition: Function that defines a whole day type mode (that is a Home Usage Profile) I#1: Home Usage Profile Id I#2: Building Partition Id I#3: List of Scene objects O: none	http://www.fiemser.eu/db/homeUsageProfile/add?HomeUsageProfileId=123&scene=…&scene=…
addScene (in:String, in:Time, in:Time, in: List<ApplianceUsage>, in:List<ComfortSetting>) Definition: Function that defines a Scene for a given Home Usage Profile, that is a subpart of a day type profile for a dwelling, including the involved appliances, their periods of operation and operating modes, and the comfort settings I#1: Home Usage Profile Id I#2: Init time I#3: End time I#4: List of ApplianceUsage objects I#5: List of ComfortSetting objects	http://www.fiemser.eu/db/homeUsageProfile/edit/123?sceneproperty1=…

Related to Advices

The CSS will use methods offered by the Database in order to:

- Get the list of advices
- Modify the status of an advice (accept/reject)

Targeted component	Database
UML Method	**HTTP/REST call**
getAdvice (in:String, in:Date, out:List<Advice>) Definition: function that returns the advices created on the specified date, including the status of their activation I#1: User Id I#2: Date O: List of Advice objects (including their status)	http://www.fiemser.eu/db/advice/list?userId=123&date=20110101
modifyAdviceStatus (in:String, in:Boolean) Definition: Function that modifies the status of an advice (accepted, rejected) I#1: Advice Id I#2: Status O: None	http://www.fiemser.eu/db/advice/set?adviceId=123&status=activate

Related to the Settings screen

The CSS will use methods offered by the Database in order to:

- Get and modify season comfort temperature for each room

Targeted component	Database
UML Method	**HTTP/REST call**
getComfortSettings (in: String, out:List<ComfortSetting>) Definition: returns comfort settings for the specified apartment (e.g., temperature per each room, per each season) I#1: Scene Id O: List of ComfortSetting objects	http://www.fiemser.eu/db/comfortSetting/list?sceneId=123
setComfortSettings (in:String, in:List<ComfortSetting>) Definition: sets comfort settings for the specified apartment I#1: Scene Id I#2: List of ComfortSetting objects	http://www.fiemser.eu/db/scene/edit/123?settings=…

Provided services

Related to Advices

The CSS will offer a method in order to:

- Push/display advices with time label

Next table describe the functions that compose the above introduced interfaces.

UML Method	HTTP/REST call
notifyNewAdvice (in:String, in:Advice) Definition: Used to inform the end-user of a new advice I#1: User Id I#2: Advice object O: none	http://www.fiemser.eu/css/notify_advice?userID=123&advice=Advice

3.4 Data Base

WS interface

The FIEMSER Data Base is a particular component in the sense that the provided services are those required by the other FIEMSER components. Therefore these services are already listed in the previous sections (see required services from targeted component "Database" for the BMCN, ICS, and UI CSS components), with their corresponding http/REST calls to the Data Base Web Services.

OSGi-based Events Handler

Practically any event occurring in the S&A network needs to be subscribed by the Data Base component in order to fill in the related information in the Data Base.

The following table summarizes the events subscribed by the Data Base.

Event message	http/REST call
Topic = /fiemser/bmcnX/status/* This is to subscribe to all status update events originated from the BMNC in the apartment number X.	Associated http/REST call to the DB will update device status by using event properties: address::=<BuildingLevelAddress> status ::= "DISABLED" | "ACTIVE" | "FAILURE" Example: http://www.fiemser.eu/db/device/edit/123?status=DISABLED
Topic = /fiemser/bmcnX/discovery/* This is to subscribe to all discovery update events originated from the BMNC in the apartment number X.	Associated http/REST call to the DB will create a new device (sensor or actuator) by using event properties: address::=<BuildingLevelAddress> + device dependent properties Example: http://www.fiemser.eu/db/device/new?prop1=...
Topic = /fiemser/bmcnX/data/* This is to subscribe to all data update events originated from any device of the BMNC in the apartment number X.	Associated http/REST call to the DB will update device properties (i.e. values) by using event properties: address::=<BuildingLevelAddress> value Example: http://www.fiemser.eu/db/device/addLog/123?logDate=20110101&logValue=24...

4. Development Environment

Following the SOA architecture, each FIEMSER component may rely on its own application server to be as much independent as possible. But, for practical implementation reasons, those components installed on the same computer and developed over the same programming platform, may obviously share the same application server.

The table below summarizes the platform chosen for each FIEMSER component.

Table 2 - Chosen platforms / frameworks for FIEMSER components

FIEMSER Component	Platform / Framework
Data Base	Java/JEE (Java Enterprise Edition), Groovy, Hibernate, Stripes, Canoo
ICS	Java, Jetty, OSGi
BMCN (boxes)	Java, OSGi, oX (oBIX server)
UI Core Services	Python, Django

Besides, a common project structure is needed to have an easy access to the resources of each development. For this purpose, we chose to use the Maven framework which is the current standard.

Maven is a free software tool for management and production automation of projects written in Java (and JEE in particular), as well as some other languages. The objective is comparable to the Unix Make utility: produce software from source, optimizing the tasks performed to this end and ensuring the efficient production order.

Besides, Maven standardizes the location of the resources in this way:

Project/	*The project name*			
	src/	*Contains all the project resources provided by the developers (source code, configuration files, etc.)*		
		main/	*Contains the main resources of the project*	
			java/	*Java source code of the project (all the .java files)*
			resources/	*Resources of the project*
		test/	*Contains the tests of the project*	
			java/	*If tests are written in Java, the Java source codes are there*
			resources/	*Resources used for the tests (and only for the tests)*
	target/	*Contain the executables and all the resources generated by Maven*		
	pom.xml	*The Maven project description*		

5. Conclusion

This deliverable is the first one produced from WP5 activities related to the FIEMSER system integration. It details the communication principles (both synchronous and asynchronous), and the interfacing between FIEMSER modules in the general context of a SOA approach. The REST Web Services provided and required by each module are defined consistently with the modules specification which is done in parallel in WP2, WP3 & WP4. The asynchronous communication of events between the BMCN (boxes) and the other FIEMSER modules is also addressed through the modelling of events and the description of events published and/or subscribed by each module.

Next step in WP5 will consist in the implementation of these interfaces, to allow a progressive integration of all FIEMSER modules, accompanied by necessary tests between pairs of components (T5.2) and for the whole system (T5.3).

Acknowledgements

The FIEMSER Consortium would like to acknowledge the financial support of the European Commission under the IST programme.

References

Previous FIEMSER deliverables:

D4 – System Architecture

D5 – FIEMSER Data Model

D7 – User interface Architectural Design

OSGi Event Admin Service specification:
http://www.dynamicjava.org/articles/osgi-compendium/event-admin-service

RESTful Web services - The basics:
http://www.ibm.com/developerworks/webservices/library/ws-restful/

Maven framework:
http://maven.apache.org/