SEVENTH FRAMEWORK PROGRAMME OF THE EUROPEAN COMMUNITY
(EC GRANT AGREEMENT N° 26088)

ICT PLATFORM FOR HOLISTIC ENERGY EFFICIENCY SIMULATION AND LIFECYCLE MANAGEMENT OF PUBLIC USE FACILITIES

Deliverable D4.2:

Full prototype of ICT system integration and intelligent access services

Responsible Authors:
Joern Ploennigs, Henrik Dibowski, André Röder, Klaus Kabitzsch, Burkhard Hensel

Co-Authors:
Ken Baumgärtel, Peter Katranuschkov

Due date: 31.08.2012
Issue date: 31.08.2012
Nature: Prototype

Start date of project: 01.09.2010 **Duration:** 36 months

Organisation name of lead contractor for this deliverable:
Technische Universität Dresden, Chair for Techincal Information Systems (TUD-TIS)

History:

Version	Description	Lead Author	Date
0.9	Full version	Joern Ploennigs (TUD-TIS)	28.08.12
1.0	**Checked and approved final version**	TUD-CIB	31.08.12

Citation

Ploennigs, J., Dibowski, H., Röder, A., Kabitzsch, K., Hensel B., Baumgärtel, K. and Katranuschkov P. (2012): HESMOS Deliverable D4.2: Full prototype of ICT system integration and intelligent access services, © HESMOS Consortium, Brussels.

Acknowledgements

The work presented in this document has been conducted in the context of the seventh framework programme of the European community project HESMOS (n° 26088). HESMOS is a 36 month project that started in September 2010 and is funded by the European Commission as well as by the industrial partners. Their support is gratefully appreciated.
The partners in the project are Technische Universität Dresden (Germany), NEMETSCHEK Slovensko, S.R.O. (Slovakia), Insinooritoimisto Olof Granlund OY (Finland), Royal BAM Group NV (The Netherlands), Obermeyer Planen + Beraten (Germany) and AEC3 LTD. This report owes to a collaborative effort of the above organizations.

Project of SEVENTH FRAMEWORK PROGRAMME OF THE EUROPEAN COMMUNITY		
Dissemination Level		
PU	Public	**x**
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)	

TABLE OF CONTENTS

Executive Summary

The **objective** of WP 4 "Model-based ICT system integration and intelligent access services" is to close the gap between building information models (BIM) and building automation systems (BAS). As a bridge between these two worlds, an ontology is used which is the basis for the software prototype described in this document. Other goals of WP4 are (1) development of intelligent access services for easy energy-relevant evaluation of monitored data, (2) development of an engineering query language enabling user-friendly formulation of energy-related tasks and (3) definition of usage scenarios with regard to BAS.

This deliverable comprises a software prototype and this supporting report. It reflects all **tasks** of WP 4:

- T4.1 Principal concepts for information synthesis and interoperability with eeBIM
- T4.2 Ontology for integrated ICT system management
- T4.3 Intelligent access services to sensing data and other ICT-based sub-systems
- T4.4 Engineering query language and usage scenarios

The deliverable report is structured into six chapters:

In the **first chapter,** the motivation for the software prototype is given.

In the **second and third chapters**, an overview of the developed intelligent access services and the engineering query language is presented. The work on these topics will be continued and refined in the beginning of the next project period in conjunction with WP 8.

The **fourth and fifth chapters** provide details about the web service functionality, architecture and interface as well as the management software tools which have been developed to administrate the provided data.

Details on tasks T4.3 and T4.4 are presented in Deliverable D 8.2 "Integrated Interoperability Methods" (Laine et al., 2012), in order to keep the specification of all platform integration and model/tool access services in one document. In particular, this covers beside the BAS support services model management, simulation and post-processing support as well as a set of general-purpose methods for session, transaction and data management.

The following **partners** were involved in the RTD work:

- **TUD-TIS**: Lead, building automation expertise
- **TUD-CIB, TUD-IBK**: Web service interfaces, harmonisation of the output data schemas, primary testing of the developed services
- **NEM**: eeBIM knowledge, link to WP3
- **BAM**: End user (Tasks 4.1 and 4.4)
- **AEC3**: IFC/BIM expertise (Tasks 4.2).

1. Introduction

One main goal of HESMOS is to compare the simulated energy consumption of a building to the data monitored during operation. There are many **advantages** of doing that:

1. Inaccurate estimations of building properties in the **design phase** can be recognized. Thus, the model parameters can be updated to **improve simulations** for future designs.
2. In the operational phase, simulations can be improved to be **more realistic** by matching model parameters to measured data.
3. The possibility to **automate processes** and to control centrally important parts of the building automation system (BAS) is provided.
4. Facility managers are supported to **detect drawbacks of the current operation scheme**, regarding single rooms, the main plants, and the whole building.
5. Measurements help to **plan refurbishments**: Critical points can be recognized and money is not wasted for improving well-running systems.

To be able to compare simulations with measured data it is in first place necessary to **provide** the measured data. However, since building management software (BMS) and building automation systems (BAS) are available in a wide variety, it is important to develop a unified access method to the data. In Deliverable D.1 (Ploennigs et al. 2011) of the HESMOS project, an **ontology** has been described which is able to support the access to all common types of building automation systems and also with the possibility to integrate new types of building automation systems. This ontology, called **BASont**, is permanently in development; its current state is presented in (Ploennigs et al. 2012).

For this deliverable, a **software prototype**, more precisely a **web service**, has been developed which provides the functionality given by the BASont. In addition, new software for **administration** of the web service functionality has been programmed.

The BASont is not the only new concept which is used to provide the measured data. **Intelligent access services** provide the interface to client software like facility management software (Forns-Samso et al. 2011) or the nD Navigator (Zellner et al. 2011). An **engineering query language** is in development which allows simple automated queries to the web service on different levels of detail according to the knowledge and requirements of the user.

Figure 1 shows the components of the HESMOS architecture which are focus of this deliverable.

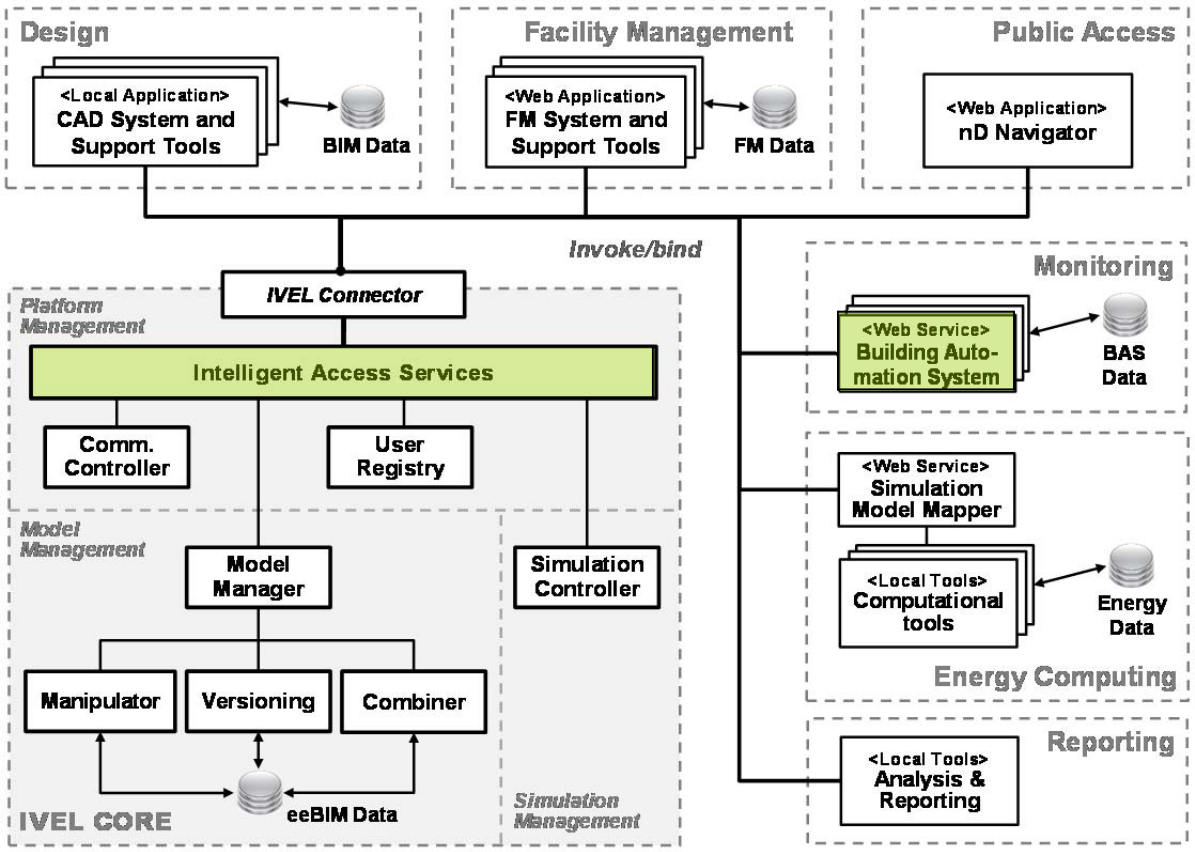

Figure 1: Focused components for this deliverable in the HESMOS software architecture (marked in light green)

2. Intelligent Access Services

The Intelligent Access Services (IAS) is a service layer in the IVEL Core that creates the connection to the BAS data web service to provide the measured data to other web applications and web services. Furthermore, it combines the workflows of starting simulations, managing simulations (start/cancel and list of running simulations), managing general information (e.g. user management), and managing the required measurement data. While it provides the management functionalities it also is responsible for storing requested data and saving the runtime changes of input parameters. Therefore the workflows are based on a Link Model where relations to input parameters and results are defined and can be queried easily.

The IAS analyses user requests from the web applications which are connected to it, e.g. the nD Navigator. It is possible that a user can compare real measured data with a simulation of one or more locations in a specific time period. When such a user request is issued, the IAS creates multiple more detailed requests and sends them to the appropriate web services like the energy computing service and the BAS web service. The requests to the BAS are leaned on the methodology described in Chapter 5 below. Hence, the IAS call multiple methods until the user request is fulfilled. The analysis of the user requests involves the distinction of the calling applications. In the case of the nD Navigator a file (e.g. JSON) for visualizing the results is requested. In the case of facility management it needs a more structured file (e.g. XML). This post-processing is one of the important services in the IAS.

Another important function is the storage of the measured data. By analysing a user request the IAS will look into a registry of already stored BAS data files. If the ordered data is already available, the IAS does not have to call the BAS web service and simply can process the stored file. This increases the performance of the overall workflows and decreases the network load. More information is provided in (Laine et al. 2012a).

As the IAS covers also services related to user management, session/transaction management, model management, simulation management and the post-processing of results, their full specification is provided in the HESMOS Deliverable D 8.2 (Laine et al. 2012b). The part regarding BAS is not reproduced here for conciseness.

3. Engineering Query Language

The purpose of the Engineering Query Language (EQL) is to provide easy-to-understand and use access to the BAS data to all end users of the HESMOS platform (facility managers, building operators, owners) who have no deep knowledge of the BAS and the related software services, schemas and protocols. Originally, it was planned to provide the EQL using SPARQL to address the BASont and enhancing it by a user-friendly vocabulary on the basis of experience from the EU project ISTforCE (Katranuschkov & Gehre, 2002). However, initial work and early end-user feedback showed that:

a) The grammar of the EQL becomes fairly complex and is therefore not readily understandable and intuitive
b) Various concepts are easier to describe semi-graphically, via a GUI, instead by a fully lexical approach.

Therefore, a hybrid approach was decided. Accordingly, the IAS functions and filtering functionality from the developed BIMfit toolset at the TU Dresden in conjunction with HESMOS and the BuildingSMART initiative are used as background. A lightweight grammar, using logical connectors such as AND, OR, XOR, NOT, NOR is used to specify more complex queries on top of the IAS. The resulting queries are presented to the end-user as dynamically defined dropdown boxes with a finite number of ready-made choices and/or as dynamically created input forms in a similar manner as in popular tools like MS Excel. This allows e.g. to select all dividing walls in a building having width of 12 cm and apply the same construction or material template to them in a single user request, to simultaneously query the BAS sensors for temperature data in all rooms with a specific room usage on a certain storey of the building and so on. In this way, a flexible GUI is achieved which is well acceptable to the end user. This approach is applied also in WP 5, e.g. for the selection of various eeTemplates to generate missing eeBIM data in the architectural IFC input. However, EQL development is still going on and will be demonstrated in the final platform prototype (WP 8).

4. BAS data web service

This chapter will look into the details of the developed web service and its administration software. Section 4.1 presents the basic architecture. Section 4.2 defines the methodology how to use the web service. The details of the interface are given in section 4.3.

4.1 Architecture

Seen from the end users point of view who wants to get measured data, a rough version of the architecture is shown in Figure 2. The "end users" are the nD Navigator and the facility management software. They have a direct connection to the IVEL core using the intelligent access services and the engineering query language, see chapters 2 and 3, respectively. The IVEL core uses the newly developed web service to get data from the proprietary BMS of the building in question in a unified format. Data aggregation methods are used to pre-process the rough data and extract energy key performance indicators, eKPIs (Bort et al. 2011). These results are returned to the end users.

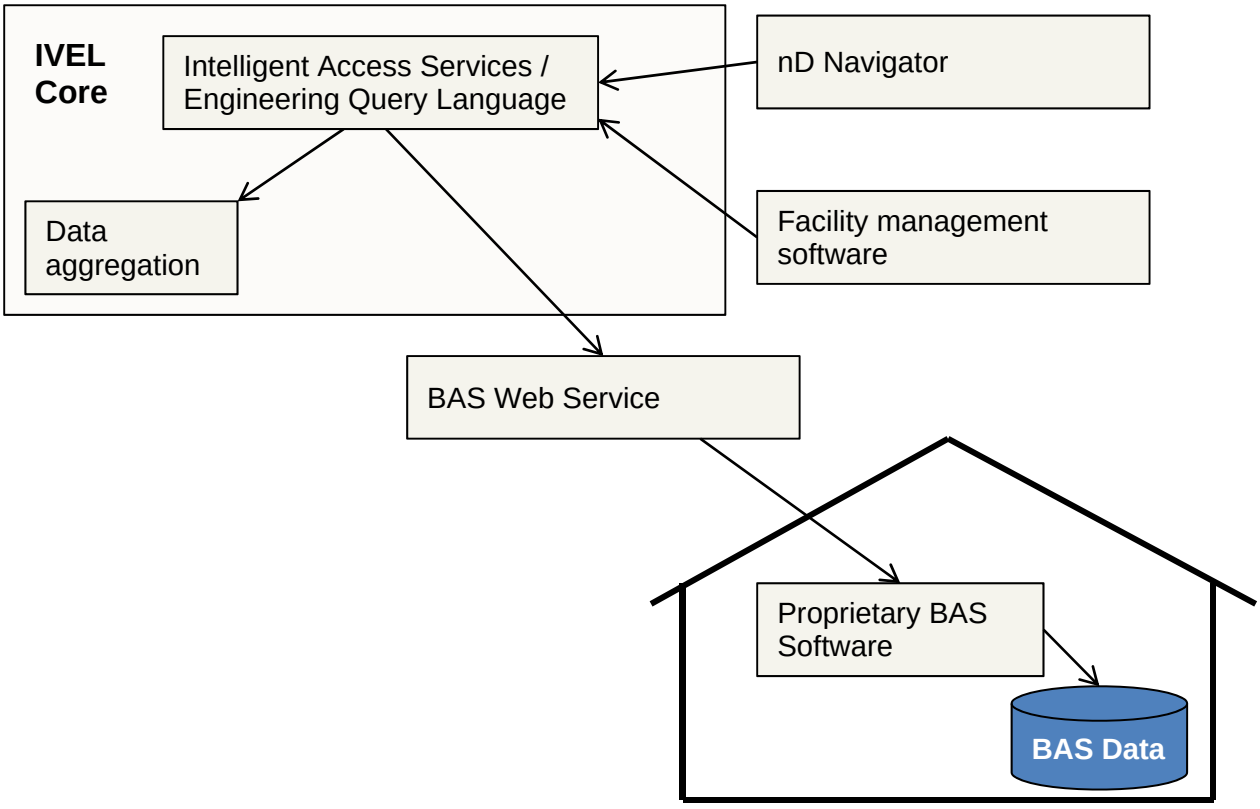

Figure 2: Rough scheme of the BAS web service architecture

The web service itself is composed of several parts, which are shown in Figure 3. The parts are described in the following table.

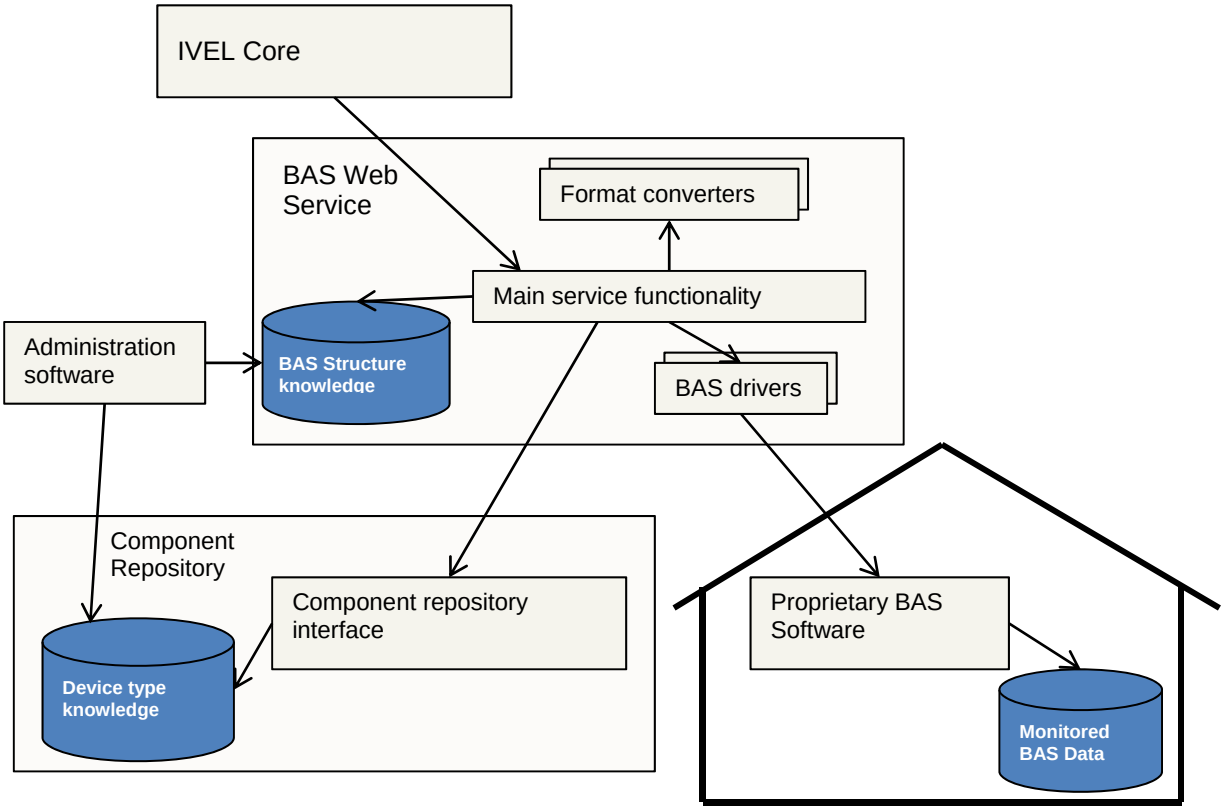

Figure 3: Architecture of the BAS web service

Table 1: Web Service components

IVEL Core	The IVEL Core contains the main **management** elements of the IVEL, e.g. simulation controller, model management and intelligent access services. It is the **interface to all client software**. The core also contains **data aggregation** methods for evaluating the rough measurement data provided by the BAS web service to find eKPIs and statistical values like minima, maxima, and mean values.
Component Repository	The component repository is **independent of concrete buildings**. It contains a huge amount of BAS **device types** with their provided functionality. In the HESMOS context, the component repository is used to find all device types which are able to deliver a specific quantity. E.g. it can be used to find all device types which are able to measure room temperature or air duct pressure. It is loosely coupled with the main BAS web service, because it can be used by other types of software, especially for BAS design of new buildings (Ploennigs et al. 2012).
Proprietary BAS Software	This part of the web service is usually located **directly in the building**. It is not created by IVEL experts but by usual BMS providers who currently do not know anything of the IVEL. This is important for integrating a **wide variety** of BAS, also older ones. The interface to such software can be very different. Examples are web services, file systems, and HTML web servers.
BAS Driver	Since proprietary BAS software can have many kinds of interfaces as described above, BAS drivers are used to **transform** each kind of provided data into a **BASont-compatible format**. These drivers can be developed by experts of the BAS web service, but also—if the IVEL gets a successful standard in practice—by the BMS manufacturers.
Main service functionality	The core of the BAS web service fulfils the **main mapping tasks**, i.e. it has the task to find the measurement data of a given building which depends to a given room and quantity. E.g. if the end user wants to get the room humidity of room A212, the web service must determine which devices are in room A212, use the component repository to find the device which is able to measure the room humidity, poll the measured data from the building using the BAS driver, and return the desired data in a unified format.
Format converters	Some clients need the measured data in a specific format. One example is the Granlund facility management software in the HESMOS context. The best solution to cope with this fact would be the integration of a converter in the client software. However, it is not always possible to make changes to the client software, or it can be very complicated. Because of that, the BAS web service architecture allows format converters which **produce a specific data format** already at the "producer' side". So, no changes of the client software are necessary. Another important use case is the TUD-IBK Delphin format which is used in the IVEL core to extract eKPIs from measured or simulated data. Also this format can be generated from the web service, reducing effort in the IVEL itself. A further advantage can be the reduced data volume to be transmitted in comparison to the relatively rich SOAP XML format. Of course, the data reduction depends on the format.

Administration software	The knowledge about the BAS device types in the component repository and the BAS device instances in the building have to be edited once to be used in the whole life cycle of the building afterwards. Administration software has been developed to support these **editing tasks**. The administration software does not use the web service directly, but it has direct access to the databases or file systems of each component to store changed information. In practice there are several programs which are used by different stakeholders, e.g. building operators and repository managers.

4.2 Access scheme

The provided BAS and IAS functions should be called in a specific order to get meaningful results. Figure 4 presents the order graphically. The first method to be called is always **getSessionID**. This is the login function. After the login, **getAvailableBuilding** should be called to get the list of recently used buildings. If a building in this list is not further needed, it can be removed from this list using **removeBuilding**. If the building whose information is desired is in the list, information about this building can be called using the **getBuildingInformation** function. Otherwise, the new building can be added to the list by the **addBuilding** method which also returns the building information. Having the building information the method **getMeasurementPoints** can be called for finding measurement points of a building according to a specification, e.g. all measurement points delivering room humidity in one given room. Using this list, monitored data can be fetched from the server by the **getLatestMeasurement**, **getMeasurementData** and **getMeasurementDataFormatted** method. Setpoints can be changed by using the **changeSetpoint** function. The last called function should always be **finishSession** which performs the logout.

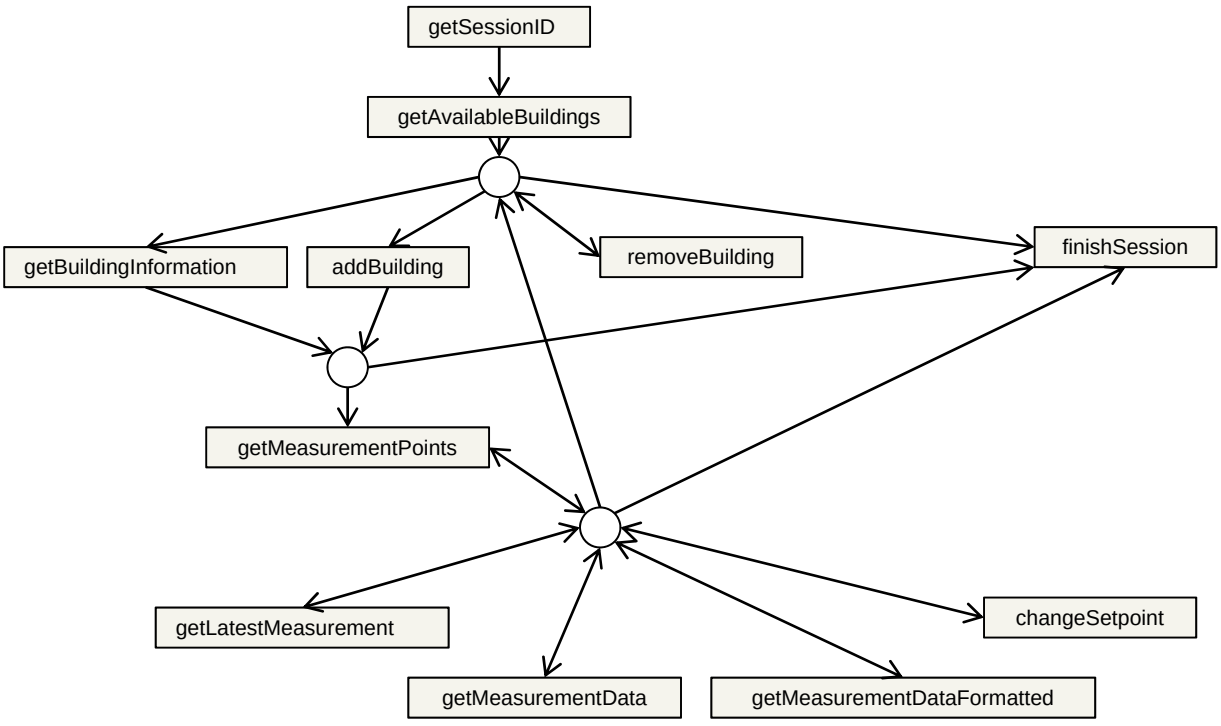

Figure 4: Recommended order of function calls

4.3 Interface

This section presents the necessary details about all functions of the web service which can be called from the IVEL core.

The BAS web service has been realized using the **SOAP** protocol as this is the most common web service protocol in practice (Pagni et al. 2008). Additionally, SOAP allows simpler code generation for clients with less probability of implementation mistakes, e.g. compared to the simpler, HTML-based REST protocol.

The web service has been developed using the Java programming language (Arnold, Gosling und Holmes 2005).

The web service is **not** designed to deliver any kind of **eKPI**. It only delivers **raw measured data**, i.e. time series of monitored measurement points. The computation of eKPIs is done by the data aggregation component in the IVEL core.

Java syntax is used to describe the method signatures. Nevertheless, clients can be programmed in any programming language which allows developing web service clients, e.g. C/C++ and C#.

Method name	**getSessionID**
Type signature	`String getSessionID(String username, String password)`
Description	This is always the first function that must be called. Based on a username and password, a session key is returned which must be used for authentication in every following method call.
Parameters	`Username` – a valid user name `Password` – the password fittig to the username
Returns	a session key; a SOAP Fault if username or password are invalid

Method name	**finishSession**
Type signature	`boolean finishSession(String sessionID)`
Description	With this method a session can be finished. After calling this method, the session ID becomes invalid and cannot further be used in the other methods. It is not mandatory to finish a session because there is always the possibility that logout can fail due to network problems. But, for security reasons, it is highly recommended to call this method after each session.
Parameters	`sessionID` – the sessionID returned by the **getSessionID** method
Returns	true, if the logout was successful; a SOAP Fault otherwise (e.g. the session ID is not valid or the method is called the second time with the same session ID).

Method name	**getAvailableBuildings**
Type signature	`String[] getAvailableBuildings(String sessionID)`
Description	Gets the list of building "tags" which have been used before. This method can be used to show the user the list of previously used buildings.
Parameters	`sessionID` – the sessionID returned by the **getSessionID** method
Returns	An array of building "tags". A "tag" is usually the combination of a "nickname" with a unique URI of the building's BAS interface/driver. Returns a SOAP Fault if the session ID is invalid or no buildings have been added yet.

Method name	**addBuilding**
Type signature	`BuildingInformation addBuilding(String sessionID, String uri, String nickname, String username, String password)`
Description	Adds a new building to the list of previously used buildings and returns information about this building. This function does not permit that the username and password can later be left out, because the web service is usually used by many people who are not allowed to get information from every building but only some for which they know a login. Also it is possible that for one building several logins with different rights (writing, reading, only some rooms) exist.
Parameters	`sessionID` – the sessionID returned by the **getSessionID** method `uri` – the unique URI of the building's BAS interface/driver `nickname` – an arbitrary name for the building what can easily be interpreted by typical users, e.g. "Head quarter in London City" `username` – the username which is required to access data from this building. This is not the username of the method **getSessionID** which is used for the BAS web service only, but specific for each building. `password` – the password which is required to access data from this building. This is not the password of the method **getSessionID** which is used for the BAS web service only, but specific for each building.
Returns	A **BuildingInformation** structure (see below), containing all available rooms and devices of the building; a SOAP Fault if the URI, the session ID, the user name or password is invalid or no connection to the building could be created.

Method name	**removeBuilding**
Type signature	`boolean removeBuilding(String sessionID, String buildingTag)`
Description	Removes a building from the list of previously used buildings.
Parameters	`sessionID` – the sessionID returned by the **getSessionID** method `buildingTag` – the building "tag", i.e. the entry in the list of available buildings which has been got using the **getAvailableBuildings** method.
Returns	True, if the building has been removed successfully; a SOAP Fault otherwise (i.e. session ID or building tag is invalid)

Method name	**getBuildingInformation**
Type signature	`BuildingInformation getBuildingInformation (String sessionID, String buildingTag, String username, String password)`
Description	Makes a connection to the building's driver, fetches its **BuildingInformation** structure, stores it locally and returns it to the user.
Parameters	`sessionID` – the sessionID returned by the **getSessionID** method `buildingTag` – the building "tag", i.e. the entry in the list of available buildings which has been got using the **getAvailableBuildings** method. `username` – the username which is required to access data from this building. This is not the username of the method **getSessionID** which is used for the BAS web service only, but specific for each building. `password` – the password which is required to access data from this building. This is not the password of the method **getSessionID** which is used for the BAS web service only, but specific for each building.
Returns	a **BuildingInformation** structure (see below), containing all available rooms and devices of the building; a SOAP Fault if the building tag, the session ID, the username or password is invalid or no connection to the building could be created.

Method name	**getMeasurementPoints**
Type signature	`MeasurementPoint[] getMeasurementPoints (String sessionID, String buildingName, String room, String deviceID, String function)`
Description	Returns all measurement points in the building which match to the parameters, i.e. which are in a given room, depend to a given device and provide a given function. If one of these three parameters is null, this parameter is ignored. E.g. all measurement points in a given room are got by setting `deviceId` and `function` to null. If all three parameters are null, all measurement points of the building are returned. The **getBuildingInformation** (or alternatively **addBuilding**) method must be called (once) before calling this method, because the building's **BuildingInformation** structure is needed to evaluate the query.
Parameters	`sessionID` – the sessionID returned by the **getSessionID** method `buildingName` – the name of the building. This is not the "tag" but the "nickname" which can be found in the **BuildingInformation** structure which is got by calling the **getBuildingInformation** method. `room` – the name of a room. Only measurement points depending to this room are returned. If this parameter is null, the room is ignored when interpreting this query. `deviceID` – the ID of a device instance. Only measurement points depending to this device instance are returned. If this parameter is null, the device ID is ignored when interpreting this query. `function` – a function (quantity). Only measurement points with this function are returned. The range of functions is not limited, but it is

	recommended to use the following functions. At the time of writing this document, the following functions are valid: “Room Temperature”, “Room Humidity”, “CO2 Level”, “Core temperature”, “Water Temperature”, “Duct Air Temperature”, “Duct Air Pressure”, “Electricity Meter”, “Heat Meter”, “Volumetric Flow Rate”, “Water Volume Meter”, “Illuminance”, “Outdoor Temperature”, “Outdoor Humidity”, “Velocity”, “Temperature Setpoint”, “Other”. If this parameter is null, the function is ignored when evaluating the query.
Returns	An array of **MeasurementPoint** structures (see below) which depend to the given room, device ID, and function; a SOAP Fault if the session ID or buildingName is invalid or if no measurement point fits to the parameters.

Method name	**getMeasurementData**
Type signature	`MeasurementCollection[] getMeasurementData (String sessionID, String buildingName, String username, String password, String[] measurementPoints, Date startTime, Date endTime)`
Description	Returns the measured data for a given time range and for a given list of measurement points.
Parameters	`sessionID` – the sessionID returned by the **getSessionID** method `buildingName` – the name of the building. This is not the “tag” but the “nickname” which can be found in the **BuildingInformation** structure which is got by calling the **getBuildingInformation** method. `username` – the username which is required to access data from this building. This is not the username of the method **getSessionID** which is used for the BAS web service only, but specific for each building. `password` – the password which is required to access data from this building. This is not the password of the method **getSessionID** which is used for the BAS web service only, but specific for each building. `measurementPoints` – a list of measurement point IDs for which the monitored data is to be requested. This will usually be the result of a **getMeasurementPoints** method. `startTime` – the earliest date and time for which monitored data is desired. `endTime` – the latest date and time for which monitored data is desired.
Returns	An array of **MeasurementCollection** structures (see below) each containing the measured data for one measurement point; a SOAP Fault if session ID, building name, username, or password is invalid or if no connection to the building’s driver was possible.

Method name	**getLatestMeasurement**
Type signature	`MeasurementCollection[] getLatestMeasurement(String sessionID, String buildingName, String user, String password, String[] measurementPoints)`
Description	Returns the latest measurement for each measurement point in the measurementPoints array. Depending on the building's driver it is also possible that the current value is polled directly from the BAS.
Parameters	`sessionID` – the sessionID returned by the **getSessionID** method `buildingName` – the name of the building. This is not the "tag" but the "nickname" which can be found in the **BuildingInformation** structure which is got by calling the **getBuildingInformation** method. `username` – the username which is required to access data from this building. This is not the username of the method **getSessionID** which is used for the BAS web service only, but specific for each building. `password` – the password which is required to access data from this building. This is not the password of the method **getSessionID** which is used for the BAS web service only, but specific for each building. `measurementPoints` – a list of measurement point IDs for which the monitored data is to be requested. This will usually be the result of a **getMeasurementPoints** method.
Returns	An array of **MeasurementCollection** structures (see below) each containing the measured data for one measurement point; a SOAP Fault if session ID, building name, username, or password is invalid or if no connection to the building's driver was possible.

Method name	**changeSetpoint**
Type signature	`boolean changeSetpoint(String sessionID, String buildingName, String username, String password, String measurementPoint, String value)`
Description	Changes the value of a "measurement point". The function of the measurement point will usually be (but is not limited to) "Temperature Setpoint".
Parameters	`sessionID` – the sessionID returned by the **getSessionID** method `buildingName` – the name of the building. This is not the "tag" but the "nickname" which can be found in the **BuildingInformation** structure which is got by calling the **getBuildingInformation** method. `username` – the username which is required to access data from this building. This is not the username of the method **getSessionID** which is used for the BAS web service only, but specific for each building. `password` – the password which is required to access data from this building. This is not the password of the method **getSessionID** which is used for the BAS web service only, but specific for each building. `measurementPoint` – the id of the measurement point to be changed. `value` – the new value for this measurement point.
Returns	True, if the change was successful; a SOAP Fault if session ID, building name, username, password, or measurementPoint are invalid, if no connection to the building's driver was possible or if the change was not possible due to lacking rights of the user or an invalid new value.

Method name	**getMeasurementDataFormatted**
Type signature	`String getMeasurementDataInDelphinFormat(String sessionID, String building, String user, String password, String measurementPoint, Date startTime, Date endTime, String format)`
Description	Calls the **getMeasurementData** method with the given parameters and converts the result to the format specified by the last parameters.
Parameters	See **getMeasurementData** method `format` – a string representing the desired file format. When writing this document, the formats "IBK Delphin 6" and "Granlund XML" are supported.
Returns	A file content in the given format; a SOAP Fault if session ID, building name, username, or password is invalid or if no connection to the building's driver was possible or if the requested format is unknown.

Structure name	**BuildingInformation**
Description	A structure containing information about a building. This can be used to support the selection of rooms and devices by the user.
Members	`String name` – the nickname of the building. This is set in the **addBuilding** method and must be used in the methods **getMeasurementPoints**, **getMeasurementData**, **getCurrentMeasurement**, **changeSetpoint**, and **getMeasurementDataFormatted**. `String uri` – a unique URI defining the building's driver. This is set in the **addBuilding** method and used to connect to the building's driver. `Room[] rooms` – an array of **Room** structures (see below) containing all rooms of the building. This can be used to allow a well-directed selection by the user before calling the **getMeasurementPoints** method. `DeviceInstance[] devices` – an array of **DeviceInstance** structures (see below) containing all devices installed in the building. This can be used to allow a well-directed selection by an expert before calling the **getMeasurementPoints** method.

Structure name	**Room**
Description	A structure containing information about a room. This can be used to support the selection of a room by the user.
Members	`String name` – the name of the room, e.g. "A105" `String description` – a description for the room, e.g. "CEO office" or "seminar room"

Structure name	**DeviceInstance**
Description	A structure containing information about a device instance. This can be used by experts to identify a special device in a building.

Members	`String deviceID` – the unique ID of the device instance `String room` – the name of the room the device is located in `String deviceType` – the unique ID of the device type `String info` – a textual description of the device instance. `String locationGUID` – the IFC guid of an IFC object to which this device instance is mapped `double coordinateX` – the x coordinate of the device's location relative to the IFC object referenced by locationGUID `double coordinateY` – the y coordinate of the device's location relative to the IFC object referenced by locationGUID `double coordinateZ` – the z coordinate of the device's location relative to the IFC object referenced by locationGUID

Structure name	**MeasurementPoint**
Description	A structure containing information about a measurement point. Also setpoints are regarded as measurement points – they have the possibility to change them using the **changeSetpoint** method. The details can be shown to the user for identifying a searched measurement point more easily
Members	`String id` – a unique ID of the measurement point `String device` – the ID of the device instance the measurement point depends to `String datapoint` – the data point of the device which is represented by this measurement point `String info` – a textual description of the measurement point `String quantity` – the quantity type according to the TUD-IBK Delphin format `String unit` – the unit in which the quantity is measured

Structure name	**MeasurementCollection**
Description	A structure containing measured data (time series) and related information.
Members	`MeasurementPoint measurement` – a **MeasurementPoint** structure containing general information about this measurement collection `Measurement[] measurements` – an array of **Measurement** structures each containing the measurement for one time instance (a time series)

Structure name	**Measurement**
Description	A structure containing information about one single measurement.
Members	`Date time` – the date and time at which this measurement has been made `String value` – the value of the quantity at the given time instance `String info` – additional information, e.g. "O.K.", "sensor breakdown", "low battery", "network error" etc.

5. Files and Software Tools

Around the new web service a variety of files and software tools are used to administrate and test the web service. These results of the HESMOS project are described in this chapter.

5.1 Example client

The web service will later be used by **facility management software** and the **nD Navigator**. However, as these clients are not available yet, for **testing** the web service as soon as possible, an example client with graphical user interface has been developed. This example client is presented in this section.

Figure 5 shows the main GUI of the test client. In the upper area, the user can manage the list of available buildings. After connecting to a building, the lower area is used for getting measurement data or setting reference values (set points). For example, if the user chooses room “A105” and wants to get the “Room temperature”, two devices and two measurement points can be selected, because there are two devices in this room which are able to measure the temperature, it is a relatively large room. After selecting one of these measurement points, data can be fetched by pressing the buttons below, see an example in Figure 6.

This client is not intended to be used by end users but only by the software developers for quickly testing the web service and to demonstrate its functionality. The end user will have a simpler front end.

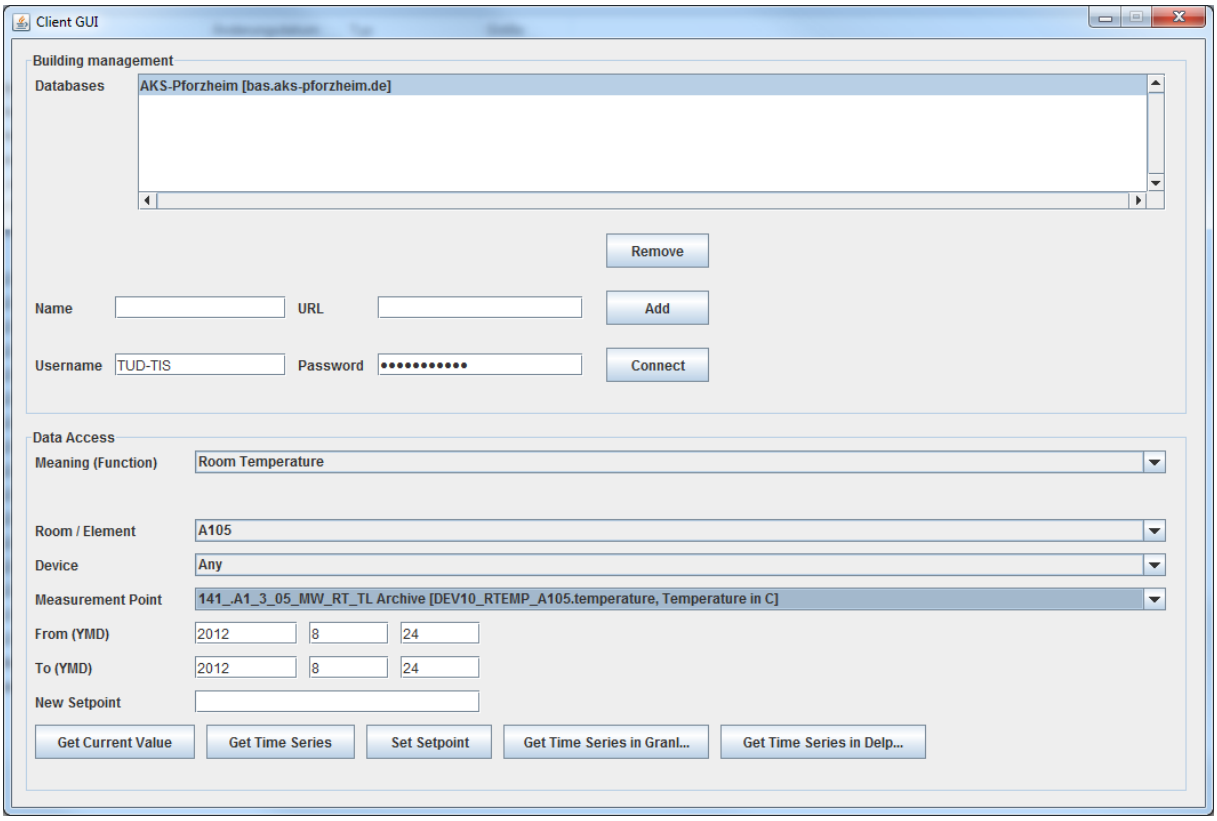

Figure 5: GUI of the example client.

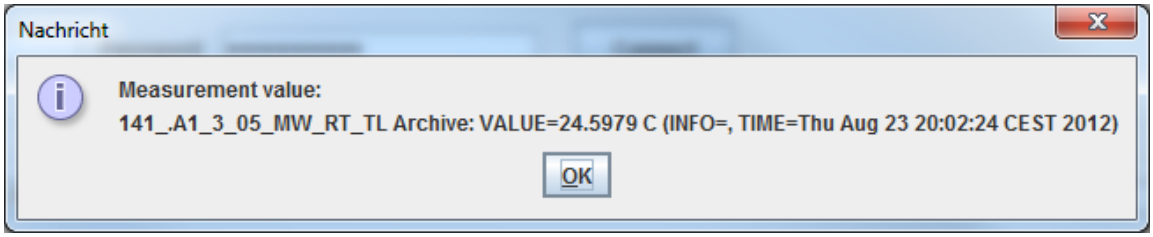

Figure 6: Result of clicking on "Get current value" in the previous figure.

5.2 Building Description file

Although an ontology repository is the basis of all information which is needed for the web service, an **XML** file type has been defined which allows the simple description of BAS systems of concrete buildings. This description of a single BAS could be stored as an XML file in the IVEL, e.g. for documentation. Also the editors generate this file as the basis for the ontology database. The room tag contains three main sub-tags, describing rooms, devices, and measurement points. A short example of the XML structure is shown below, the explanation follows.

```
<?xml version="1.0"?>
<building name="AKS Pforzheim" uri="bas.aks-pforzheim.de">
<roomlist>
      <room name="A001" info="" />
      <room name="A006" info="Laboratory" />
      <room name="A007" info="Server room" />
      <room name="A008" info="" />
      <room name="D131" info="" />
      <room name="HEAT" info="heating plant" />
      <room name="O" info="outdoor area" />
</roomlist>
<devices>
      <device id="DEV1_RTEMP_A001" room="A001" type="HDEV RT model 2300"
      info="room temperature sensor in room A001" />
      <device id="DEV423_RTEMP_RHUM_D131" room="D131"
      type="roomtemperature_roomhumidity_generic" info="multisensor of
      unknown vendor in room D131" locationGUID="48db81ai389184gs87kp33"
      coordinateX="4" coordinateY="2.5" coordinateZ="0.4" />
</devices>
<measurementpoints>
      <measurementpoint id="131_.A1_2_01_MW_RT_TL" device="DEV1_RTEMP_A001"
      functionalProfile="ANY" operationMode="ANY"  datapoint="temperature"
      quantity="Temperature" unit="C" info="" />
      <measurementpoint id="184_.B_2_29_MW_RT_TL"
      device="DEV426_RTEMP_RHUM_D131" functionalProfile="ANY"
      operationMode="ANY"  datapoint="temperature" quantity="Temperature"
      unit="C" info="" />
      <measurementpoint id="184_.B_2_29_MW_RF_TL"
      device="DEV426_RTEMP_RHUM_D131" functionalProfile="ANY"
      operationMode="ANY"  datapoint="humidity" quantity="RelativeHumidity"
      unit="%" info="" />
</measurementpoints>
</building>
```

Explanation

As attributes of the building tag, the building name and a unique ID (URI) are stored.

The first sub-tag is the list of rooms, using names and descriptions. This list can usually be taken from the IFC structure. It is copied to this file because generating the list from the IFC file each time it is needed would take too much time. Also “virtual rooms” (e.g. outside area, heating plant) can be generated for grouping devices which do not directly depend to a concrete room.

The devices list contains all physical devices in the building. The first entry in the example shows a room temperature sensor of the type “HDEV RT model 2300”. The second is a multisensor, measuring room temperature and room humidity, coming from an unknown vendor (thus called “roomtemperature_roomhumidity_generic”). This option is useful, because in many cases the building operator does not know the exact model type but the semantics of the data it delivers. Using these “generic” types allows including these devices with lowest effort. While the first device is only mapped to a room, for the second device exact coordinates are given, relative to the IFCObject with GUID “48db81ai389184gs87kp33”.

The third sub-tag includes all measurement points. Each measurement point has a unique ID and depends to a device in the previously explained device list. Also further information for unique access is given, e.g. the functional profile, operation mode, and data point of the device which is used as well as the unit.

5.3 Management software

Management software is necessary to edit all relevant information of the ontology. These tools have been developed in the HESMOS project and have research prototype character. That means that the software works correctly and provides all main functionality but should be improved for industrial use.

5.3.1 Simple component repository editor

A simple GUI for editing the component repository has been developed. This software does not yet support all possibilities of the BAS ontology, but all functionality which is needed to use the web service in the HESMOS project, especially with the pilot project “Pforzheim school”. It is therefore limited to the device type information which is necessary for HESMOS. It is planned that the software will be improved after making practical experiences in the rest of the HESMOS project. Figure 7 shows an example of the GUI.

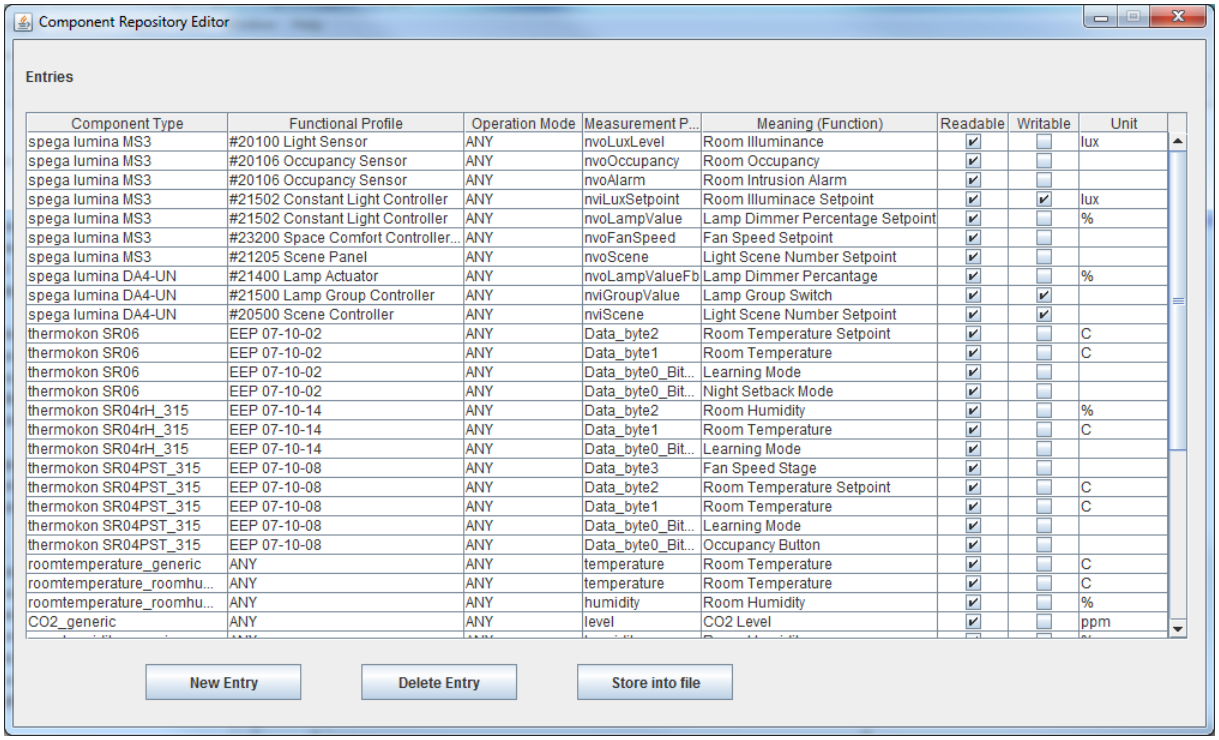

Figure 7: GUI of the component repository editor prototype

5.3.2 Complex component repository editor

Another ontology editing software can be used for defining more complex information about device types. It has been developed outside of HESMOS based on an older version of the BAS ontology; see (Dibowski & Kabitzsch, 2010). Many properties of devices can be specified there. This software is also only a research prototype for exploring the possibilities of the ontology. The knowledge of the editing person must be much higher than for the simple editor software above. So, for practical reasons, the simple editing software is more suited for the HESMOS project.

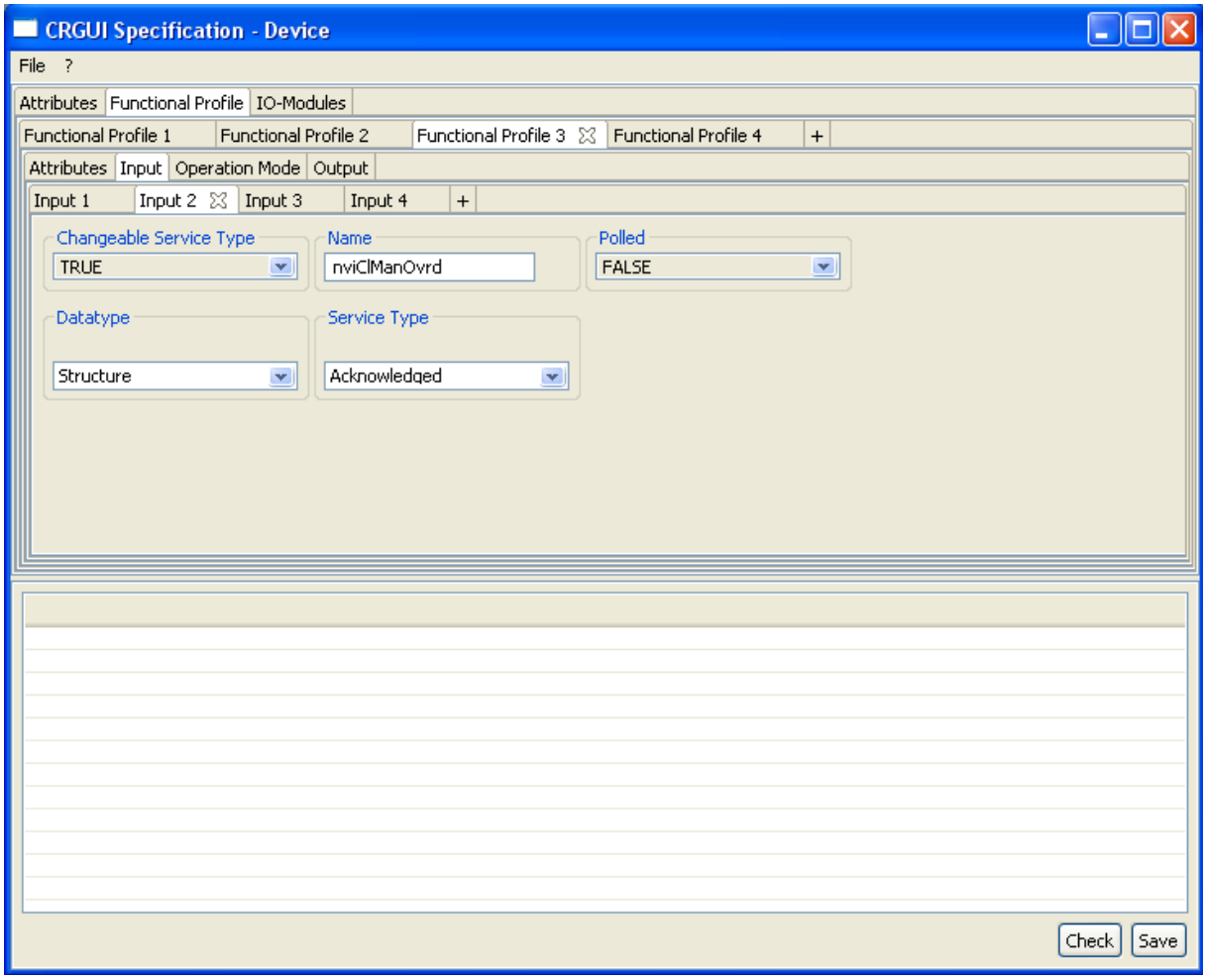

Figure 8: Complex component repository editor

5.3.3 Building data editor

For editing the ontology data of a concrete building, a software tool has been developed. It will be used by a building operator who knows the building and its measurement points. The edited data is finally stored in the XML file structure shown in section 5.2.

On the GUI presented in Figure 9 the rights of different users can be adjusted. For example, facility managers can be allowed to change setpoints while architects who use the data for planning refurbishments are only allowed to read the measured data. After pressing “Edit user” the password of the user can be changed.

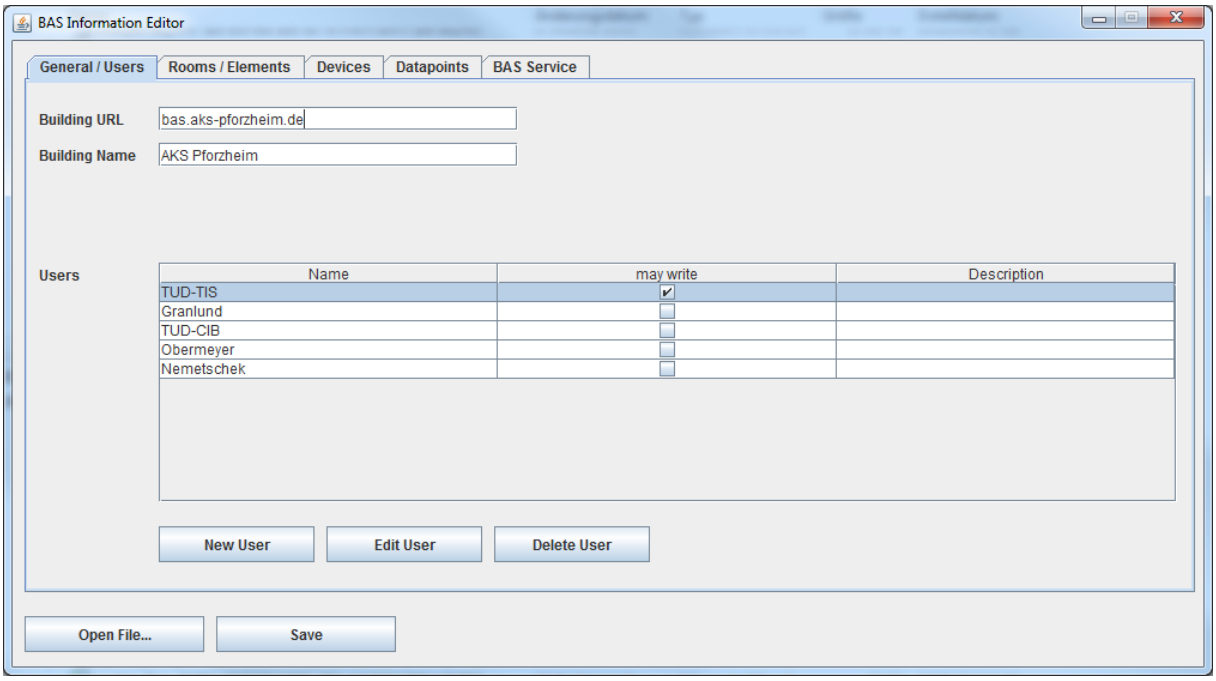

Figure 9: GUI for editing end user rights for BAS data access

Figure 10 presents the GUI for editing the list of installed BAS devices in a building. All properties (ID, type, location etc.) can be directly edited by clicking in the field and writing new information. After each change the new information is validated. E.g. it is checked whether the device ID is unique. If an unknown room name is typed, a warning is shown with the possibility to automatically add the new room to the room list. This is especially comfortable, if room names are used which do not match to the IFC structure, e.g. “outdoor area” or “heating plant”.

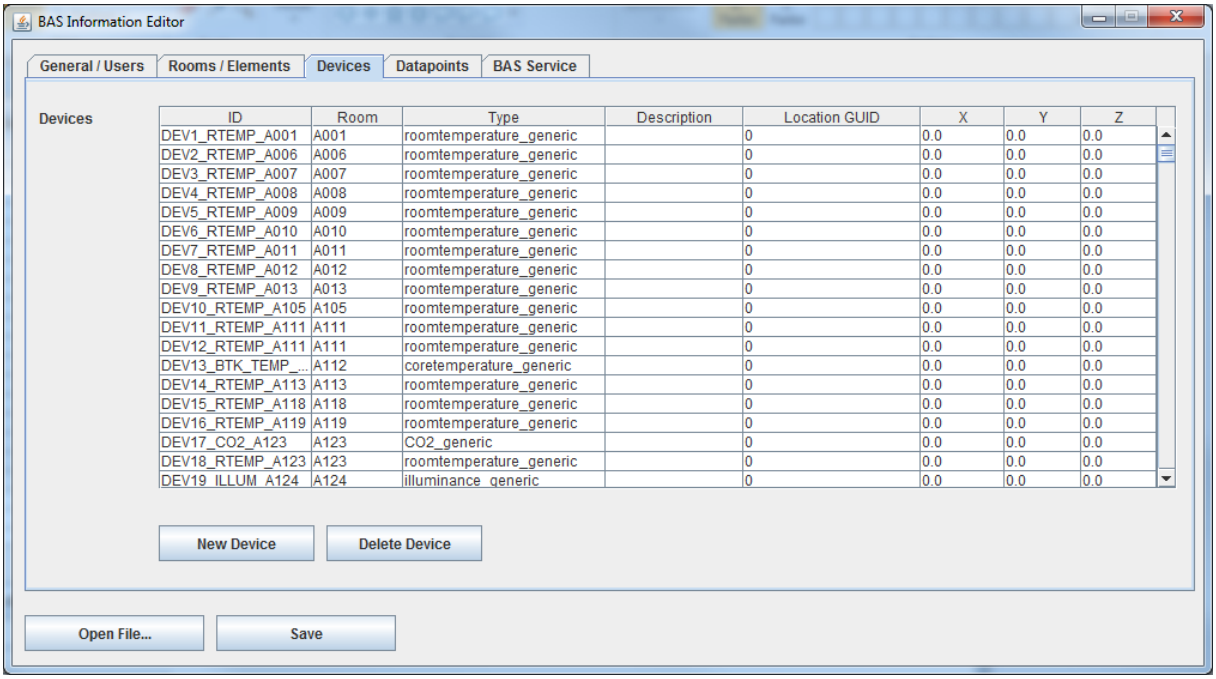

Figure 10: GUI for editing BAS device instances of a building

The measurement points (data points) of the devices can be edited in the GUI shown in Figure 11. Also there, all changed information is validated to ensure that each ID is unique and only existing devices are specified.

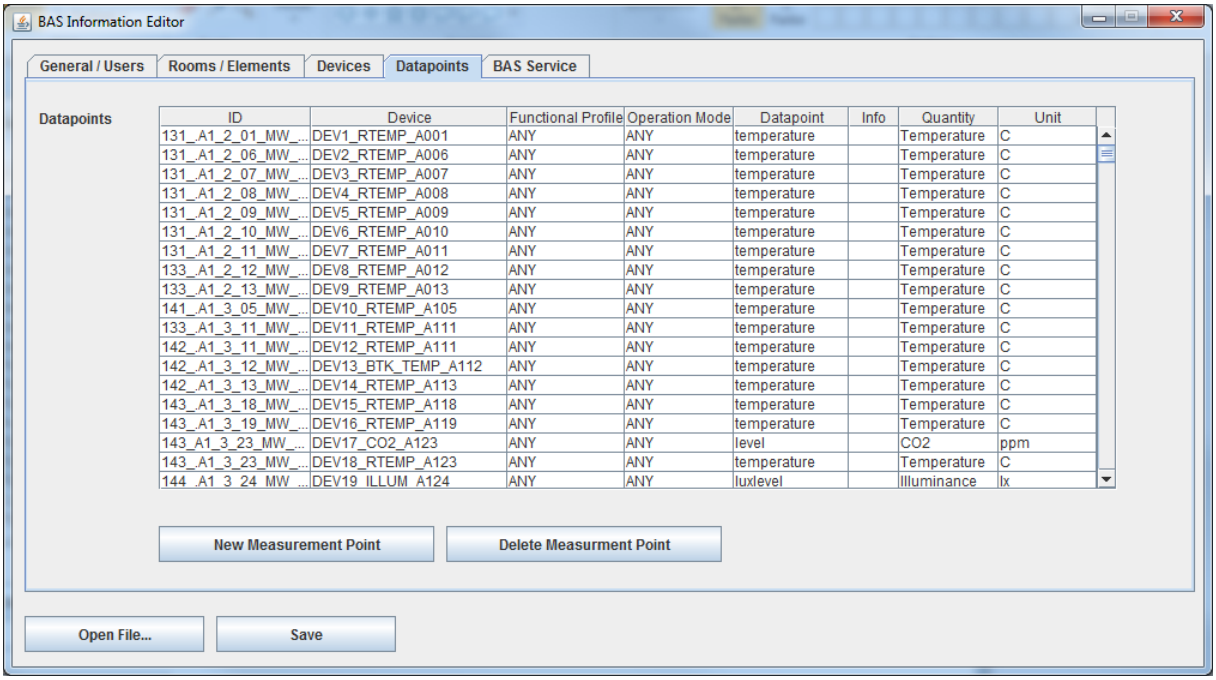

Figure 11: GUI for editing data points of the BAS devices

5.3.4 Mapping creator

Although all necessary information about a concrete building can be edited in the software presented in section 5.3.3, it may be a time-consuming and error-prone task to write all IDs of measurement points and devices by hand. For most buildings a list of available measurement point IDs and rooms can be obtained in the form of a CSV file from the BAS installers (CSV is an often used, text-based table format which also can be exported from Excel tables).

A tool has been developed which can import such CSV files. Using these files, the user can generate the building description file by simply clicking on a measurement point ID, the appropriate room, and the function realized by this measurement point, see Figure 12. Afterwards, the resulting building description file can be imported in the software of section 5.3.3 to edit details like descriptions, coordinates of important devices etc. Besides, since the file structure is simple XML, any XML editor can be used by experts. This can be very useful to do complex copy or replace operations.

In several tests it has been found that using this tool, in only one hour up to 1000 measurement points can be mapped to their appropriate rooms and functions.

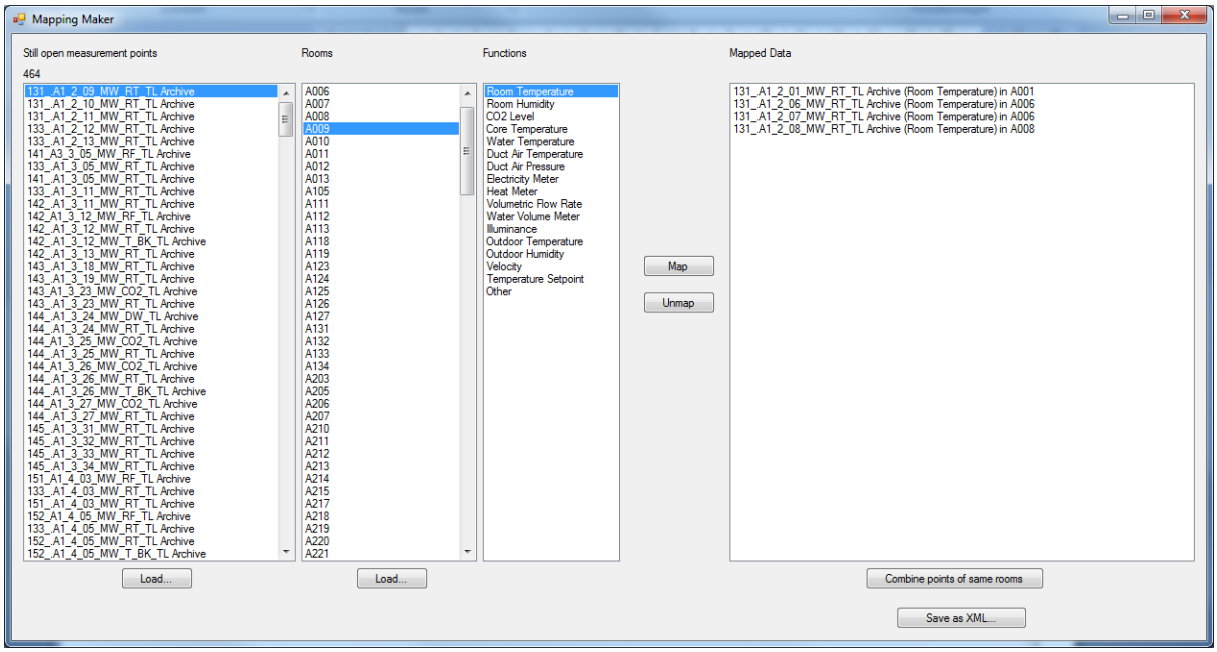

Figure 12: Mapping maker software

6. Conclusions

In this deliverable report, the prototype of the building automation systems web service has been described which is used together with the intelligent access services to provide the targeted functionality. Using a newly developed example client, the correct functionality could be validated.

All necessary software components for work package 4 have been developed and tested. Not only "dummy" tests using invented data have been performed, but also tests with the real information about the pilot project AKS Pforzheim. Thus, remaining mistakes and all main performance problems could be solved. The BAS data of the pilot project Pforzheim can successfully be accessed via the web service. Nevertheless, the tools have prototype character and should be enriched by wizards and help functions for industrial use.

Because of the current state of the project, no "end user test" could be made yet. For that purpose more software is needed which is still in development in other work packages with later deadlines. Hence, further work is planned towards the BAS web service prototype, including:

- The engineering query language
- More comfortable administration and editing software
- Enhancement of the BAS ontology
- Performance optimization.

These planned improvements do not influence other deadlines of the HESMOS project since all necessary functionalities for WP 4 are available already now and finalisation of the mentioned additional issues shall be done, as planned, in the frames of WP 8.

Thus, it can be concluded that with this deliverable D 4.2 the objectives of WP 4 are fulfilled.

Literature Sources

Arnold, Ken, James Gosling, und David Holmes. *The Java Programming Language (4th Edition).* Prentice Hall, 2005.

Bort, B., T. Caruana, und M.-C. Geißler. *HESMOS Deliverable 9.1: Requirement Synthesis and Energy-Related Key Performance Indicators.* Brussels: HESMOS Consortium, 2011.

Dibowski, H., und K. Kabitzsch. „Generic Specification Toolchain for Ontology Based Device Descriptions.“ *Proceedings of the 15th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2010).* Bilbao, Spain, 2010.

Dibowski, Henrik, und Klaus Kabitzsch. „Ontology-Based Device Descriptions and Device Repository for Building Automation Devices.“ *EURASIP Journal on Embedded Systems*, 2011.

Forns-Samso, F., M.-C. Geißler, F. Jonas, A. Karola, und T. Laine. *HESMOS Deliverable D6.1: Enhancement of the energy-related tools for the lifecycle use of eeBIM.* Brussels: HESMOS Consortium, 2011.

Laine, T., K. Baumgärtel, R. Hänninen, R. Zellner, und P. Katranuschkov. *HESMOS Deliverable 8.1: Configuration and Deployment of the Developed Basic SOA System.* Brussels: HESMOS Consortium, 2012.

Laine, Tuomas, et al. *HESMOS Deliverable 8.2: Integrated Interoperability Methods.* Brussels: HESMOS Consortium, 2012.

Pagni, Marco, Jörg Hau, und Heinz Stockinger. „A Multi-Protocol Bioinformatics Web Service: Use SOAP, Take a REST or GoWith HTML.“ *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid.* 2008. 728-734.

Ploennigs, J., B. Hensel, H Dibowski, und K. Kabitzsch. „BASont - A modular, adaptive Building Automation System Ontology.“ *Proceedings of the 38th Annual Conference of the IEEE Industrial Electronics Society (IECON 2012).* Montreal, Canada, 2012.

Ploennigs, J., H. Dibowski, A. Röder, K. Kabitzsch, und B. Hensel. *HESMOS Deliverable D4.2: Ontology specification for model-based ICT system integration.* Brussels: HESMOS Consortium, 2011.

Zellner, R., et al. *HESMOS Deliverable 7.1: Concept and Architecture of the nD-Navigator.* Brussels: HESMOS Consortium, 2011.

Appendix I: Acronyms

BACnet	Data Communication Protocol for **B**uilding **A**utomation and **C**ontrol **Net**works
BAS	**B**uilding **A**utomation **S**ystems
BIM	**B**uilding **I**nformation **M**odel
BMS	**B**uilding **M**anagement **S**ystem
CSV	**C**omma-**S**eparated **V**alues
eeBIM	**E**nergy **e**nhanced **B**uilding **I**nformation **M**odel
eKPI	**E**nergy-related **K**ey **P**erformance **I**ndicators
EQL	**E**ngineering **Q**uery **L**anguage
IAS	**I**ntelligent **A**ccess **S**ervices
ICT	**I**nformation **C**ommunication **T**echnology
IFC	**I**ndustry **F**oundation **C**lasses
IVEL	**I**ntegrated **V**irtual **E**nergy **L**aboratory
LON	**L**ocal **O**perating **N**etwork
SPARQL	**S**imple **P**rotocol **a**nd **R**DF (Resource Description Framework) **Q**uery **L**anguage
URI	**U**niform **R**esource **I**dentifier
XML	e**X**tended **M**arkup **L**anguage