BaaS

Building as a Service

FP7-ICT-2011-6: ICT Systems for Energy Efficiency
Small or Medium-scale Focused Research Project
Grant Agreement No. 288409

Deliverable 2.3:

BIM repository and associated methods and tools

Deliverable Version:	2.3, v.1.5
Document Identifier:	baas_wp2_d2.3_bimrepository_1.5.docx
Preparation Date:	October 23, 2013
Document Status:	Final
Author(s):	César Valmaseda, Miguel A. García, José L. Hernández, Daniel Garcia; CARTIF. Dimitrios Rovas, Giorgos Kontes; TUC. JaeSeung Song, Martin Floeck; NEC
Dissemination Level:	PU - Public

SEVENTH FRAMEWORK PROGRAMME

Project funded by the European Community in the 7th Framework Programme

ICT for Sustainable Growth

European Commission Information Society and Media

Deliverable Summary Sheet

Deliverable Details

Type of Document:	Deliverable
Document Reference #:	2.3
Title:	BIM repository and associated methods and tools
Version Number:	1.5
Preparation Date:	October 23, 2013
Delivery Date:	October 23, 2013
Author(s):	César Valmaseda, Miguel A. García, José L. Hernández, Daniel Garcia; CARTIF. Dimitrios Rovas, Giorgos Kontes; TUC. JaeSeung Song, Martin Floeck; NEC
Document Identifier:	baas_wp2_d2.3_bimrepository_1.5.docx
Document Status:	Final
Dissemination Level:	PU - Public

Project Details

Project Acronym:	BaaS
Project Title:	Building as a Service
Project Number:	288409
Call Identifier:	FP7-ICT-2011-6
Call Theme:	ICT Systems for Energy Efficiency
Project Coordinator:	Fundacion Cartif (CARTIF)
Participating Partners:	Fundation Cartif (CARTIF, ES); NEC Europe Ltd. (NEC, UK); Honeywell, SPOL, S.R.O (HON, CZ); Fraunhofer-Gesellschaft zur Förderung der Angewandten Forschung e.V. (Fraunhofer, DE); Technical University of Crete (TUC, GR); University College Cork, National University of Ireland, Cork (UCC-IRU, IE) Dalkia Energia y Servicios (DALKIA, ES)
Instrument:	STREP
Contract Start Date:	May 1, 2012
Duration:	36 Months

Deliverable 2.3: Short Description

BaaS Deliverable 2.3 presents the background, methodologies and technical guidelines to deploy a IFC4-based Server containing all the information of each all of the buildings involved in the project, as well as the corresponding clients aimed at connecting to those servers and interchange the building information stored according to the IFC4 data model (ISO 16739:2013).

Keywords: BaaS; BIM; BIMServer; Server; Client; Interface, Connector; Query; Model View Definition; IFC; APO; Data Model.

Deliverable 2.3: Revision History

Version:	Date:	Status:	Comments
0.1	15/03/2013	Draft	CARTIF: deliverable template (ToC and skeleton)
0.2	18/03/2013	Draft	CARTIF: Adaptation of contents of older versions to the new skeleton
0.3	28/04/2013	Draft	CARTIF: New sections and content after agreement with all the partners involved. (TUC&NEC)
0.4	2/05/2013	Draft	CARTIF: New content in Sections 1,3 & 4 TUC: New content in Sections 2 & 5
0.5	9/05/2013	Draft	NEC: Contribution to section 4.2 CARTIF, TUC: Contribution to all the sections of the Deliverable as well as its structure and content.
0.6	14/05/2013	Draft	CARTIF: New content in Sections 1 & 4, Executive Summary, TUC: New content in Sections 2.3, 2.4, 4.4 and Appendix B.
0.7	17/05/2013	Draft	CARTIF: Contribution to Section 4.2, revision of the document format and common sections, Update of the Figure 13. TUC: Contribution to Sections 1.2, 2.4 and 4.3. NEC: Contribution to Sections 1.2 and 4.2
0.8	24/05/2013	Draft	CARTIF: Revision of Sections 4.2, 4.3 and Appendix B. Goals for the final version added in Executive Summary and Introduction.
0.9	7/06/2013	Draft	CARTIF: Final version of Section 4.2 and 4.3 TUC: Contributions to the Introduction, Executive Summary and additional information added to the Appendix B.
1.0	07/06/2013	Intermediate report	CARTIF: Final version of the Intermediate Report
1.1	24/09/2013	Draft	CARTIF: Corrections in the Sections 4.2, 4.3.1 and 5.2.

1.2	25/09/2013	Draft	CARTIF: Inclusion of the definition of the MVD class for the queries performance and the deployment scheme of the BIM Servers.
1.3	08/10/2013	Draft	TUC: Sections 4.1.1, 4.1.2 and 4.3.2 reviewed with the new version of the BIM Server. Added a new example of query in the new Server.
1.4	11/10/2013	Under review	TUC, CARTIF: Review of the document with some corrections.
1.5	23/10/2013	Final	TUC, CARTIF: Including the comments from the review of the partners.

Table of Contents

List of Figures

List of Tables

Abbreviations and Acronyms

AC	Air Conditioning
AEC Industry	Architecture, Engineering and Construction Industry
API	Application Programming Interface
APO	Assess, Predict, Optimize
BaaS	Building as a Service
BACN	Building Automation and Control Network
BAN	Building Automation Network
BIM	Building Information Modelling
BIMSie	BIM Service interface exchange
BMS	Building Management System
CLL	Communication Logic Layer
CRUD	Create, Read, Update and Delete
DACM	Data Access and Control Manager
DWH	Domestic Hot Water
EMF	Eclipse Modelling Framework
FDD	Fault Detection and Diagnosis
FLOSS	Free/Libre Open Source Software
gbXML	Green Building XML
IaaS	Infrastructure as a Service
ICT	Information and Communication Technologies
IFC	Industry Foundation Classes
ISO	International Organization for Standardization
JAR	JAVA Archive
JSON	JavaScript Object Notation
MEP	Mechanical electrical and plumbing
MVD	Model View Definition
OOP	Object-oriented programming
OSGi	Open Services Gateway initiative
PaaS	Platform as a Service
PAS	Publicly Available Specifications
PB	Protocol Buffers
RC	Release Candidate
SaaS	Software as a Service

SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
UML	Unified Modelling Language
WAR	Web Application Archive
WP	Work Package
WSDL	Web Service Definition Language
XML	Extensible Mark-up Language

Executive Summary

The Deliverable 2.3 contains the definition and specification of the architecture, features and communication guidelines of the BIM Repositories of the BaaS Project, both the servers to be installed in the demo sites and containing all the building information, and the required connectors/clients to extract that information using the standard data model IFC4.

The following areas are covered in detail in this draft Deliverable:

- Detailed review of the different IFC data model versions with special emphasis on the last release of the standard, IFC4 (ISO 16739:2013).
- List of requirements of the BaaS Project with regard to the BIM Server, carefully analysing how the proposed solution meets them.
- Proposed solution by the BaaS Team, consisting on the use of TNO's BIM Server 1.2 adapted to be able to manage IFC4 models as part of the activities of the Task 2.3.
- Definition of MVDs for the information exchange required by the different activities of the BaaS Project.
- Examples of functionality of all the elements of the system: server, client, MVDs and queries' performing.

Contributors of this Deliverable include CARTIF, NEC and TUC.

IMPORTANT NOTICE: As a consequence of the activities carried out in the context of the Deliverables 2.1 [1] and 3.1 [3] as well as the conclusions obtained from them, the activities of the Task 2.3 and this Deliverable have been significantly altered from the general approach foreseen in the proposal phase to another approach more defined and focused on the detailed analysis and commissioning of the models and systems proposed in those documents. Consequently, the team involved in the development of this document has considered appropriate to change the name of this Deliverable from the original one "Standardized data editors using SOTA of ifcXML" to "BIM repository and associated methods and tools", in order to perfectly reflect the real scope of the document.

1 Introduction and objectives

1.1 Purpose and target group

The WP2 is in charge of the building's data interoperability and standardization. Specifically, the work package 2 objective is to collect, aggregate, integrate and use the existing buildings' data, such as geometrical and structural information, as well as readings from the BMS, and data models. Therefore, the activities carried out in this work package are aimed at specifying and developing an extended data warehouse and a Building Information Model (BIM) based on standardized data model and functions.

Particularly, the Task 2.3 is in charge of implementing the BIM specification from Task 2.1 using an ISO standard data model (Industry Foundation Classes IFC - ISO 16739:2013) to facilitate seamless interoperability of the BIM with upstream activities launched by the Assess, Predict, Optimize - APO - services.

According to the basis settled in the Deliverables 2.1 [1] and 3.1 [3], where the BaaS Project's main architecture and interfaces are detailed as well as the IFC data model is specified as the standard to share building information among the different elements of the BaaS Project architecture, this document aims at specifying a sort of guideline, both methodological and technical, to design and deploy an IFC4-based BIM repository to store the information of the buildings in IFC format as well as serving those data under request by a number of clients expected to be connected to each of the servers deployed in the context of this project.

Additionally, the state of the art included in the previously mentioned deliverables will be extended with all the specific features of the IFC4 data model in order to introduce the required details to understand the basic elements of a building information repository and to understand the mechanisms involved in the server-client information exchange.

Finally, the complete list of elements of the IFC ecosystem, as well as the proper configuration and development methodologies required to set the repository up will be detailed in this document; together with all the guidelines to develop a client able to connect and interchange the required information related to both configuration and data access purposes will be detailed and documented with their adequate examples.

1.2 Contribution of partners

The partners involved in this task, CARTIF TUC and NEC, worked together to analyse, adapt and update the existing solutions in order to meet the requirements of the BaaS project's global system. The specific contribution of each partner to the technical and organizational activities of this Task and Deliverable are in this section.

1.2.1 CARTIF

Administrative and Organizational tasks:

- Coordination and tracking of the deliverable's development.
- Organization of meetings and conference calls to track the progress of the activities and ensure the objectives of the document are reached.

Technical tasks:

- Development of the source code of CARTIF's client aimed at being connected to the BIMServer located in CARTIF's building (BIMServer 1.2 RC5) and interchange data server-client in IFC2x3 format.
- Modification of the TUC's client to connect it to a second BIMServer located in CARTIF's building (BIMServer 1.2 RC1 adapted to IFC4 by TUC) and interchange data server-client in IFC2x3 format.
- Contribution in the development of the sections 1, 2, 3, 4 and Appendix A of the deliverable, as well as the non-technical sections.

1.2.2 TUC

Administrative and Organizational tasks:

- Active participation in the meetings and conference calls.

Technical tasks:

- Adaptation of BIM Server (v1.2) to IFC4 data model and development of a client-side application for validation and server management.
- Development of a client-side application to perform administration tasks and query requests to an IFC4 enabled BIM Server (v1.2) for TUC building.
- Contribution in the development of Sections 1, 2, 4, 5 and Appendix B of the deliverable.

1.2.3 NEC

Administrative and Organizational tasks:

- Active participation in the meetings and conference calls.

Technical tasks:

- Definition of the interface (NEC) – Section 4.2
- Contributions in the development of the State of the art section.

1.3 Relationship with other work packages

This deliverable responds to why IFC4 is selected as BaaS data model, how the BIM server developed by TNO is selected and adapted to BaaS as BIM repository, how BaaS system is connecting to the BIM server to upload and download IFC projects, as well as how BaaS implements queries to collect some specific piece of information from the IFC server.

Each service or application defined in BaaS could have some building information requirements, so the majority of these services and applications could demand building information and so, have the necessity to request such information by means of a set of queries to the BIM server. Any request to the BIM server from other BaaS component or entity is translated in the communication layer (middleware) in a set of queries to the BIM Server collecting the required data.

To better understand which building information requires a service or an entity in the BaaS system, BaaS applies the concept of MVD. A MVD represents the data requirement specification for the implementation of an IFC interface to satisfy the exchange requirements of some application. Thus, the exchange requirements for each APO service or other BaaS entity are defined and made publicly available using a MVD. Within this context, if two software

components are to interact, they need to exchange sufficient information, meeting all the exchange requirements, so that this communication is complete are defined in the MVD.

This deliverable presents the guidelines to fulfil the process through which the communication layer will be able to implement the necessary set of queries to the BIM server to request the necessary building information exchange requirements of any APO services or BaaS entity.

Each APO service or BaaS Entity with building information exchange requirements have to, based on the guidelines shown in this deliverable, define the MVD that represents their exchange requirements.

Once the exchange requirements are defined using a MVD within the corresponding task or WP associated to the service or entity (mainly WP5), WP3 has to deploy and use the BIM Server Client presented in this deliverable to implement the set of necessary queries to collect the defined exchange requirements for each MVD.

In summary, this deliverable presents the tools needed to hold building information and support request as well as the guidelines to use such tools, also shows the BIM server and the software component which as client implements how to do queries to the building information, and how the APO services and BaaS entities should map and share to other WPs their building information requirements.

This deliverable presents one example where the whole process (from the exchange requirement identification using an MVD to the implementation of the corresponding set of queries to satisfy the data requirements) is shown to an APO service.

However, due to the fact that this deliverable closes in month 16, it is not going to either create the MVD for each application or implement the set of queries for each service, as this is the responsibility of the WP-Task in charge of such application and WP3, respectively.

The specific responsibilities and dependencies of this deliverable 2.3 with other WPs are highlighted in the list below:

- [Draft version month 12] – the activities developed in the context of this deliverable and the task 2.3 trigger the development of the connector I-7 while the communication with the upper layers belongs to WP3.
- [Draft version month 12] – the results of this deliverable 2.3 enable to WP5 to assess the APO service requiring static information from the BIM repository.
- [Draft version month 12] – the results of this deliverable 2.3 enable to WP4 to collect all the static information from the server (e.g. about walls, windows, geometry, etc.) required to perform the buildings' modelling and simulations.
- [Draft version month 12] – the outcomes of this deliverable offer the other WPs the MVD-based methodology to exchange information among BaaS system's components and services.
- [Draft version month 12] – the BIMServer included as a result in this version is functional and adapted to IFC, ready to host the IFC models of the BaaS project's buildings.
- [Final version month 16] – the BIM Connector/Client, developed in close collaboration with the WP3, is completely functional and integrated in the BaaS system.

1.4 Terminology

Test Cases	A test case in software engineering is a set of conditions or variables under which a tester will determine whether an application or

	software system is working correctly or not. The mechanism for determining whether a software program or system has passed or failed such a test is known as a test oracle. In some settings, an oracle could be a requirement or use case, while in others it could be a heuristic. It may take many test cases to determine that a software program or system is considered sufficiently scrutinized to be released. Test cases are often referred to as test scripts, particularly when written. Written test cases are usually collected into test suites.
Test Suite	In software development, a test suite, less commonly known as a validation suite, is a collection of test cases that are intended to be used to test a software program to show that it has some specified set of behaviours. A test suite often contains detailed instructions or goals for each collection of test cases and information on the system configuration to be used during testing. A group of test cases may also contain prerequisite states or steps, and descriptions of the following tests. Collections of test cases are sometimes incorrectly termed a test plan, a test script, or even a test scenario.
Release Candidate	A release candidate (RC) is a beta version with potential to be a final product, which is ready to release unless significant bugs emerge. In this stage of product stabilization, all product features have been designed, coded and tested through one or more beta cycles with no known showstopper-class bug. A release is called ‘code complete’ when the development team agrees that no entirely new source code will be added to this release.
Requirement	A requirement is a description and specification of the functionality desired for a system. It can also collect performance features of the system, such as availability, scalability, security, etc. From the end user point of view, a requirement represents a constraint imposed by the client on the development of a software product.
Class	In software engineering, a class is a set or category of things having some property or attribute in common and differentiated from others by kind, type, or quality. In object-oriented programming, a class is a construct that is used to create instances of it – referred to as class instances, class objects, instance objects or simply objects. A class usually represents a noun, such as a person, place or thing, or something nominalized.
Object	Objects in "object-oriented programming" are essentially data structures together with their associated processing routines. For example, a file is an object: a collection of data and the associated read and write routines. Objects are considered instances of classes.
Class Diagram	A class diagram is an illustration of the relationships and source code dependencies among classes in the Unified Modelling Language (UML) where the classes are arranged in groups that share common

	characteristics and for the interrelation of objects. Class diagrams are useful in all forms of object-oriented programming (OOP).
Connector	A software connector is an architectural building block that manages component interactions, normally application-independent.
Interface	An interface is a concept that is meant to facilitate and standardize inter-process and inter-component communication and interaction. One of the main characteristics of interfaces is that the component or entity which implements the interface can be treated as a “black box”, i.e. as long as the interface is properly implemented, the outside world does not need to know anything of the internal procedures or functioning. Interfaces are crucial elements for loosely coupled systems.
Cloud computing	Cloud computing is a general term for anything that involves delivering hosted services over the Internet. These services are broadly divided into three categories: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS). The name cloud computing was inspired by the cloud symbol that's often used to represent the Internet in flowcharts and diagrams.
Data model	A data model in software engineering is an abstract model that describes how data are represented and accessed. Data models formally define data elements and relationships among data elements for a domain of interest. Data models describe structured data for storage in data management systems such as relational databases.
Specification	Complete description of the behaviour of a system to be developed and may include a set of use cases that describe interactions the users will have with the software. In addition it also contains non-functional requirements. Non-functional requirements impose constraints on the design or implementation.
Standard	An established norm or requirement about technical systems instituted for compatibility and interoperability between software, systems, platforms and devices.
Building information Modelling	Building information modelling (BIM) is a process involving the generation and management of digital representations of physical and functional characteristics of a facility. The resulting building information models become shared knowledge resources to support decision-making about a facility from earliest conceptual stages, through design and construction, through its operational life and eventual demolition.
Entity	An entity is an external system that interworks with the main system under development and is necessary for the correct and complete behaviour of it. For example, regarding the BIM System architecture,

	an entity can be a determined server, a client, a connection, etc.
Query	Queries are the primary mechanism for retrieving information from a database and consist of questions presented to the database in a predefined format.
Service	A set of related software functionalities that can be reused for different purposes, together with the policies that should control its usage. OASIS[1] defines service as "*a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description.*"
APO Services	An APO Service is a high-level element developed to provide integrated assessment, prediction and optimization (APO) services that guarantee harmonious and parsimonious use of the available resources.

For further information about these concepts and other concepts related to software engineering and Building Information Modelling, please refer to [6] and [7].

[1] https://www.oasis-open.org/

2 State of the art

2.1 Comparison between to open solutions: IFC4 and gbXML.

As briefly introduced in the section 2.2.6 of the Deliverable D2.1 [1], besides IFC data model, one of the most developed and extended schemas to transfer building information among software tools is gbXML data model. Both schemas are widely supported by vendors and manufactures in the AEC Industry and; therefore, both of them were considered as possible solutions to meet the BaaS system requirements.

BaaS supported services require different types of data which are related with different building industries, such as the architectural, engineering and mechanical electrical plumbing (MEP) industries and involve associated tasks such as thermal and energy performance simulations and control operations. Various data schemes have been developed in order to support these data types, which are designed according to their format and the needs of the industries and the tasks they are involved in. Prominent examples include the IFC data standard developed by buildingSMART and the gbXML schema developed by Green Building Studio and supported by Autodesk, Bentley and Graphisoft.

Within BaaS framework building thermal and energy simulation tasks as well as control and fault detection operations are planned to be performed. Such operations require access and bidirectional communication between certain data structures referring to specific parts of the building model. When a control task is to be performed for example, these structures may contain data from sensor measurements and controls of devices of specific rooms of a building. Also, in case of a whole building thermal simulation, where the whole building is treated as a single zone only the geometry and material characteristics of the building envelope are required. These examples indicate that BaaS services need to access only certain partitions the building model depending on the task being performed. Such requirement can be satisfied efficiently only by the IFC schema via the model view definitions (MVD) and not by gbXML. (In gbXML such a requirement would involve loading and repartitioning the whole building model for each of the above processes, an operation that requires time and memory resources).

Generally, in IFC a MVD defines a subset of the IFC schema used in order to cover specific requirements of the architecture engineering and construction (AEC) industry. Furthermore in MVDs rules for exchanging data among the specific subset of IFC and a related application can also be defined rendering IFC ideal for Baas purposes. Such rules are not present in the gbXML schema, a fact which adds more credit towards selecting IFC.

BaaS tasks require the involved data sources to update their contents according to the changes performed in the building. Such adaptability is offered only by the IFC database, as it supports interoperability in the sense that different applications may alter its contents at any time. For example any installation of a new device can be followed by the update of the IFC data base using an IFC-compatible application and affect the results of a thermal simulation or a control operation being performed by another IFC-compatible application. This capability of IFC alludes towards a more dynamic follow up of the building life cycle and is aligned according to BaaS objectives.

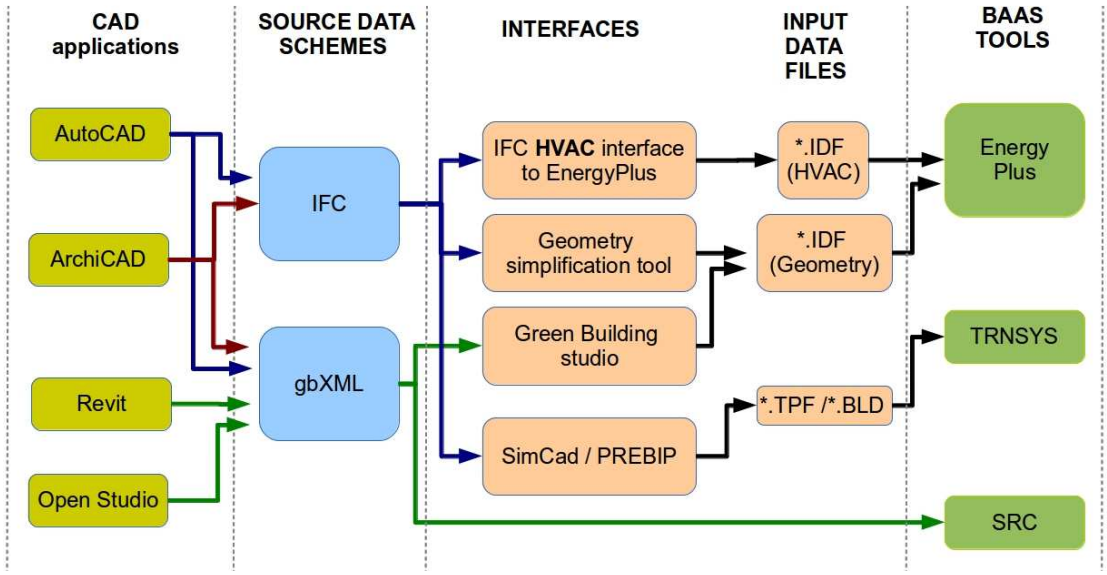

Figure 1 Existing interfaces between IFC and gbXML source data schemes and BaaS Simulation Tools

The present status of existing AEC industry applications and relative data schemes are characterized by four major components as Figure 1 summarizes. The first contains the CAD applications that provide the data schemes of the building models. The second major component consists of the data schemes of the building models, which are organized differently containing different class and object definitions. The third component is the developed interface programs, which connect the source data schemes with simulation and other applications within the BaaS framework (EnergyPlus, TRNSYS, SRC). The last component consists of the applications within BaaS scope, which use these data schemes and have different input data requirements as well as input file formats. The plethora of existing interface programs and respective file formats lead to merging attempts under a common platform. Examples include the SimModel described in [9]. The above pluralism is indicative of a shifting trend towards supporting IFC data scheme as opposed to the gbXML scheme demonstrated by certain missing links.

More precisely, concerning building devices, and as Figure 1 indicates, a direct link between the IFC data structures and the HVAC component of the IDF input files of EnergyPlus has been established since 2003 [10]. Such data link is missing from gbXML, a fact that demonstrates that IFC is more preferable than gbXML. Additionally, another reason advocating IFC popularity as opposed to gbXML, is the missing link between gbXML and TRNSYS, as opposed to programs SimCad and PREBIB, which when used as a sequence can provide tpf and bld TRNSYS input files from IFC data (see Figure 1).

Inherently, the two most popular source building data schemes (IFC and gbXML) were designed to serve different purposes. On the one hand, IFC was initially designed to support information used in architectural, structural and engineering domains, on the other hand gbXML was designed in order to facilitate the data export between leading CAD software such as Autocad Revit and ArchiCad and building thermal and energy simulation applications. As these data schemes were designed for different purposes, their data structures and their interrelations differ substantially [11]. Content-wise IFC is richer compared with gbXML, which lacks information required by BaaS services.

As far as building geometrical information is concerned, IFC contains geometrical data organized in a completely different and more efficient manner than gbXML. This difference is

highlighted by the fact that IFC classes describing the geometry of building entities use non-approximated solid definitions as opposed to the descriptions adopted by gbXML which contain only point coordinates. This becomes apparent in the description of curved solid constructions such as in cases of cylindrical walls or domes, which are described in IFC by a centre point and a radius, as opposed to multiple-point approximations used to describe the same entities in gbXML. Consequently, IFC's solid geometrical description of the building constructions, allows a variable degree of approximation accuracy as opposed to a fixed degree used by gbXML. As the degree of approximation of the geometrical representations of the building entities has an analogous impact in the accuracy of the thermal simulation results and is inversely proportional to the simulation execution speed, adopting a data scheme which allows a variable degree of geometrical approximation of the building model, such as IFC, enables the BaaS services requiring thermal simulations to be versatile, ranging from fast less accurate ones to slow and more accurate ones.

Furthermore the building construction surfaces containing openings are defined differently in IFC compared to gbXML. In gbXML constructions containing openings are defined by multiple partitions in two dimensions, as opposed to IFC where they are defined as a single three-dimensional entity (see Figure 2 example referring to a wall with two windows). This fact adds an additional difficulty when attempting to interface gbXML with simulation tools like EnergyPlus where construction with openings are defined as single entities, as it requires geometric operations in order to transform multiple partitions into single entities. As a result, in such cases IFC appears a more suitable data source for EnergyPlus and consequently for Baas services.

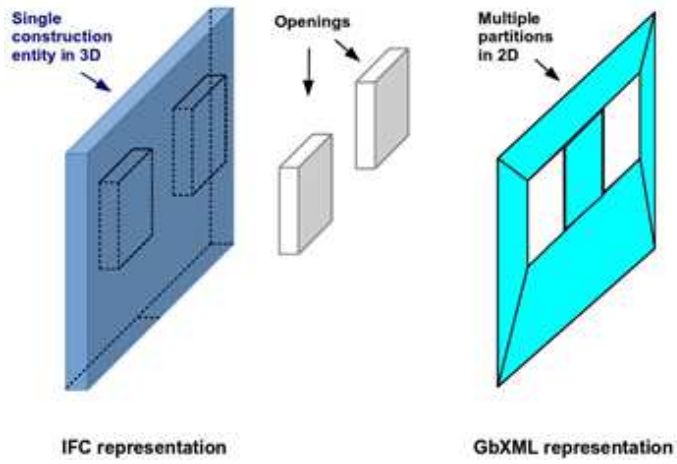

Figure 2 IFC and gbXML representations of a building construction with openings

Although gbXML is designed for building thermal simulations, it cannot provide second level b-type space boundary information as defined in [4]. Second level b-type boundaries appear in tilted and regular walls at the places where wall thicknesses intersect with space volumes. The impact of these space boundary types in the thermal simulation results increases with the wall thickness. As the geometrical representations of building constructions are defined in IFC in three dimensions compared with the two dimensional respective definitions of gbXML (see Figure 2), second level space boundaries of type-b can be extracted from IFC files, adding more credit towards selecting IFC for Baas purposes.

2.2 IFC Data Model. (ISO 16739:2013)

The deliverable 2.1 of BaaS project [1] defines the IFC (Industry Foundation Classes) data model as the standard selected to share and interchange building information in the BaaS Project system architecture over other existing solutions, especially chosen for its openness, interoperability and capacity to cover all the phases of the building life cycle.

IFC is a data representation standard primarily for architectural and construction product data (ISO 1994). The information model was defined by the International Alliance for Interoperability (IAI), nowadays known as buildingSMART Organization[2]. One of main goals of IFC is to provide information exchange between model-based tools in different industry domains, such as the construction and facility management.

After the first release of the IFC released in January 1997, it has been extended to contain information from various domains, such as building controls, plumbing and fire detection, structural elements, structural analysis, heating, ventilation, and air conditioning systems, electrical systems, architecture, construction management, and facilities management. At the early stage, the information models were only provided as schemas represented in the EXPRESS data definition languages, which is a data modelling language defined in ISO 10303-11, however nowadays there exist multiple ways of describing the IFC data, such as ifcXML.

Figure 3 shows the IFC release history. At the time of this writing, IFC4 (also known as IFC2x4) is the latest version and has been officially released in March 2013.

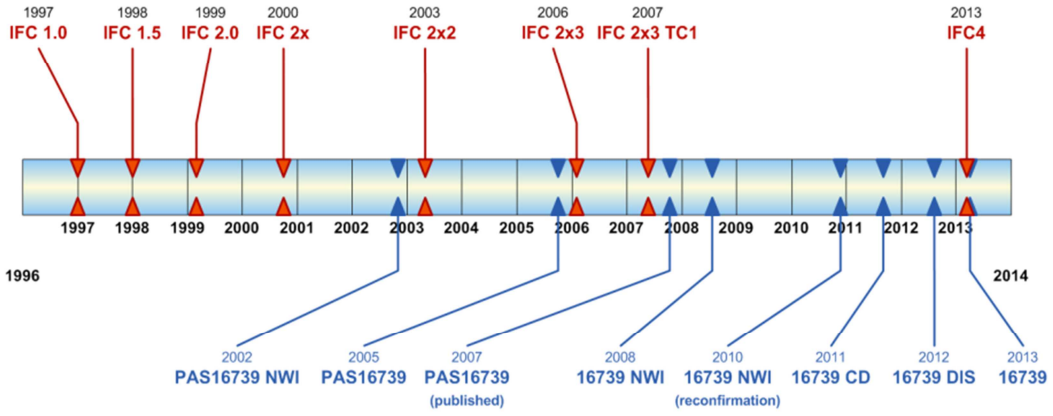

Figure 3: History of IFC releases [8]

In addition to the IFC specification written using the EXPRESS, an ifcXML specification is also published as well since the IFC2x release. IfcXML is an XML format defined by ISO 10303-28, part 28 and contains the contents of IFC data. IfcXML enables to exchange IFC data in XML formats. Each ifcXML release is usually published as an XML Schema Definition, XSD. XSD is derived from the IFC EXPRESS model, and a method mapping the IFC EXPRESS model into the ifcXML XSD follows a configuration file that controls the specifics of the translation process. For each version of IFC schema, the corresponding configuration files are standardized and published.

For example, the ifcXML2x3 was released in 2007 with an effort to establish rules and policies for managing and developing XML schemata. This version is for IFC2x3, which mainly improves the quality of IFC2x2, adds some features to the previous version [8]. In particular, the

[2] http://www.buildingsmart.org/

ifcXML schema for IFC2x3 consists of two parts -- ***ex.xsd*** and ***IFC2x3.xsd***. ***ex.xsd*** is the common schema for all translated EXPRESS models containing the definitions for the header section and the general data types that result from the translation of the EXPRESS data types. ***IFC2X3.xsd*** is the IFC2x3 specific unit of serialization that contains the XSD definitions of all IFC specific classes, relationships, attributes and data types.

The IFC4 data model integrates a number of features and it is intended to be used as the next basis for IFC enabled interoperability of Building Information Models. The IFC4 includes (1) extensions in building, building service and structural areas (2) enhancements of geometry and other resource components (3) numerous quality improvements, and (4) a new documentation format. The following shows main extensions and improvements of IFC4:

- enhancement and completion of the IFC object classification;
- ability to map to external classifications to and from;
- further improvement and optimization of definitions for the process and cost element;

Details of new features of the IFC4 can be found in the link[3].

Additionally, Table 1 shows a brief description of each IFC release and its corresponding ifcXML release.

IFC Release	Year	ifcXML Release	Features
IFC1.0	1997	None	core model, resource models and four initial domain extensions (architecture, building services, construction management, and facilities management)
IFC2x	2000	ifcXML2x	The first IFC platform release. The concept of a core model and domain extensions was introduced.
IFC2x2	2003	-	No official ifcXML release. Introduced many extensions, e.g. it contains the first IFC sub model, extensions for building control definitions.
IFC2x2 Add 1	2004	ifcXML2x2	Small addendum of IFC2x to fix issues. This release has been used for IFC2x2 implementation and certification.
IFC2x3	2006	ifcXML2x3	The third release of the IFC2x platform. This release includes mainly quality improvement of IFC2x2.
IFC4	2013	ifcXML4	Combines a number of features increase with some major rework and improvements of the existing IFC specification.

Table 1: IFC releases and their corresponding ifcXML

[3] http://www.buildingsmart-tech.org/ifc/IFC2x4/rc1/html/change/IFC2x4-rc1_whats_new.htm

Based on the Industry Foundation Classes, buildingSMART defined multiple file formats for various encodings of the same underlying data (IFC-SPF as text format defined by ISO 10303-21, IFC-XML as XML format defined by ISO 10303-28, IFC-ZIP as ZIP compressed format consisting of an embedded IFC-SPF file). At the time of writing, for the most recent standard ifcXML2x4, the corresponding ifcXML definition is still pending.

Generally speaking ifcXML - being based on XML concepts and mechanisms - is suitable for open interfaces and human and machine readable protocols. Further it allows manipulation by commonly available XML editors. That said, an ifcXML based interface to the BIMserver would conceptually allow for a free choice of technology on both client and server side. However, as BaaS aims to reuse available tools and software for the BIM Server component where possible and given that the TNO BIMserver provides already a ServerClientLibrary (see section 3.4.2), it is prudent to rely on this rather than implementing an ifcXML based interface from scratch. Depending on the development of the TNO BIMserver and its interface library, ifcXML2x4 may or may not be used in this project.

2.3 IFC4 Data model definition.

IFC4 data model is the enhancement of IFC2X3 schema and has been accepted as an ISO standard for “data sharing in the construction and facility management industries”, thus constituting a widely-used open BIM standard. The new data schema features the following properties[4]:

- Enables a plethora of BIM workflows, like BIM to GIS and 4D and 5D model exchanges.
- Allows extension of IFC to infrastructure and other parts of the building.
- Is fully compatible and integrated with mvdXML technology for Model View Definitions, thus enabling model validation processes.

In order to be able to provide this functionality, IFC4 features a layered architecture[5], containing the following layers:

- Core data schemas: these schemas formulate the most general IFC layer, providing the common concepts and basic relationships for further specializations and contain the following:
  - The IfcKernel schema, which defines the core part of IFC, featuring general constructs.
  - The IfcControlExtension schema, which declares base classes for control objects.
  - The IfcProcessExtensions schema, which defines classes allowing the mapping of processes in logical sequences of “tasks”.
  - The IFcProductExtension schema, which provides classes for further specializing the properties of a physical product.
- Shared data schemas: implement intermediate specializations of the entities defined in the core data schemas, shared by multiple domains:

[4] http://www.buildingsmart-tech.org/specifications/ifc-releases/ifc4-release/buildingSMART_IFC4_Whatisnew.pdf

[5] http://www.buildingsmart-tech.org/ifc/IFC4/final/html/index.htm

- IfcSharedBldgElements schema, containing the main components of the raw building (e.g. walls, roof, etc.).
- IfcSharedBldgServiceElements schema, containing concepts required for the interoperability of Building Service domain extensions, such as basic type and occurrence definitions for flow and distribution systems, among others.
- IfcSharedComponentsElements schema, providing the ability to represent small parts, such as accessories and fasteners.
- IfcSharedFacilitiesElements schema, which provides basic concepts for the facilities management process.
- IfcSharedMgmtElements schema, providing a set of concepts to elaborate the management process throughout the building lifecycle.

- Domain specific data schemas: IFC4 features a collection of domain specific data schemas, consisting of self-contained entities that manage to organize definitions according to the respective industry disciplines. These data schemas are the following:
  - The IfcArchitectureDomain schema, which defines basic concepts used in the architectural domain that have not been generalized or pushed lower in the model, such as door and window lining and panel parameters.
  - The IfcBuildingControlsDomain schema, which defines concepts of building control, building automation, instrumentation and alarm and supports occurrences of sensors, controllers, etc. The specific schema offers the capability to capture real-time device data, sensor and actuator properties and define control entities that monitor and utilize specific sensor inputs to produce control outputs among others.
  - The IfcConstructionMgmtDomain schema, which defines resource concepts in the construction management domain, such as resources used in construction process (including material, labour and equipment resources), resource time information to support allocations and levelling, resource productivity calculation to determine work, usage and duration of tasks, time-phased data to indicate scheduled and actual work, etc.
  - The IfcElectricalDomain schema, which defines concepts of cabled systems where the cabling carries various forms of cable transmission, such as electrical supply, data and telephone signals, along with a collection of devices connected by cabling.
  - The IfcHVACDomain schema, which defines concepts required for the Heating, Ventilation and Air Conditioning (HVAC) domain, including taxonomy of systems typically used in buildings, such as boilers, chillers, fans, along with terminal and flow control devices, such as air vents, valves and dumpers.
  - IfcPlumbingFireProtectionDomain schema, which defines concepts of plumbing and fire protection. For plumbing, the scope includes services external to the building up to the final manhole connecting to the public drainage/sewage service provision, while for fire protection it includes all services from the point at which a fire authority service is connected up to the point at which the public connection is terminated to the building.
  - The IfcStructuralAnalysisDomain and the IfcStructuralElementsDomain schemas. The IfcStructuralAnalysisDomain aims at integrating the structural engineering domain by associating structural assumptions to the existing building element and spatial structures, thus defining the planar and spatial structure analysis models which can be used by structural analysis applications. In close relation, the IfcStructuralElementsDomain schema provides the ability

to represent structural-related building elements and building element parts, like footings, piles and reinforcement parts.

- Resource definition data schemas: these schemas consist of supporting data structures which do not exist independently but are referenced by one or more entities deriving from the root definition (IfcRoot entity) of the kernel defined in the core data schemas layer.
- Fundamental concepts and assumptions: here concept templates are defined, indicating use of data types for particular scenarios. Each template defines a set of entities and attributes, featuring specific constraints for particular attributes. This collection of concepts also forms the basis of Model View Definitions.

2.4 IFC Models Management

2.4.1 IFC Editors

2.4.1.1 Autodesk® RevitTM

This 3D CAD software tool has been specifically designed for supporting Building Information Modelling (BIM). It is probably the most extended tool in the AEC industry since it includes different functionalities for architectural design, MEP systems' design (mechanical, electrical and plumbing), construction and structural design, etc. in order to coordinate the whole project through a parametric engine.

In the last release of the tool (Revit 2013), all the existing tools in the previous versions (Revit Architecture, Revit MEP and Revit Structure) were united in a single tool, in the Autodesk Building Design Suite software package.

The way of modelling is through the use of components (named as families) parameterized in order to allow the inclusion of the information relative to its dimension, materials and other properties, and also allowing and easy way for modifying them.

Regarding the interoperability, it is possible to export and import in IFC, but in the last version of the tool (Revit 2013), only IFC 2x3 is supported. So, all the elements included in the IFC 4 scheme (i.e. sensors, etc.) are not supported. Thus, it is necessary to use another tools, such as Constructivity, that do support.

Revit also has an integrated tool for pre-calculating the heating and cooling loads of the building, according to the introduced location, pre-defined construction systems and pre-defined HVAC systems. So, this functionality only gives an approach for this calculation, since it does not use the real modelled systems. In this sense, another included functionality is the possibility of exporting the model in gbXML format, and use web-based energy analysis software named Autodesk® Green Building Studio®, which can calculate also in an approached way the energy performance of the building. The advantage of this tool is that it allows making some general design alternatives in order to improve the energy performance or the calculation of some LEED® credits (such as the daylight credit).

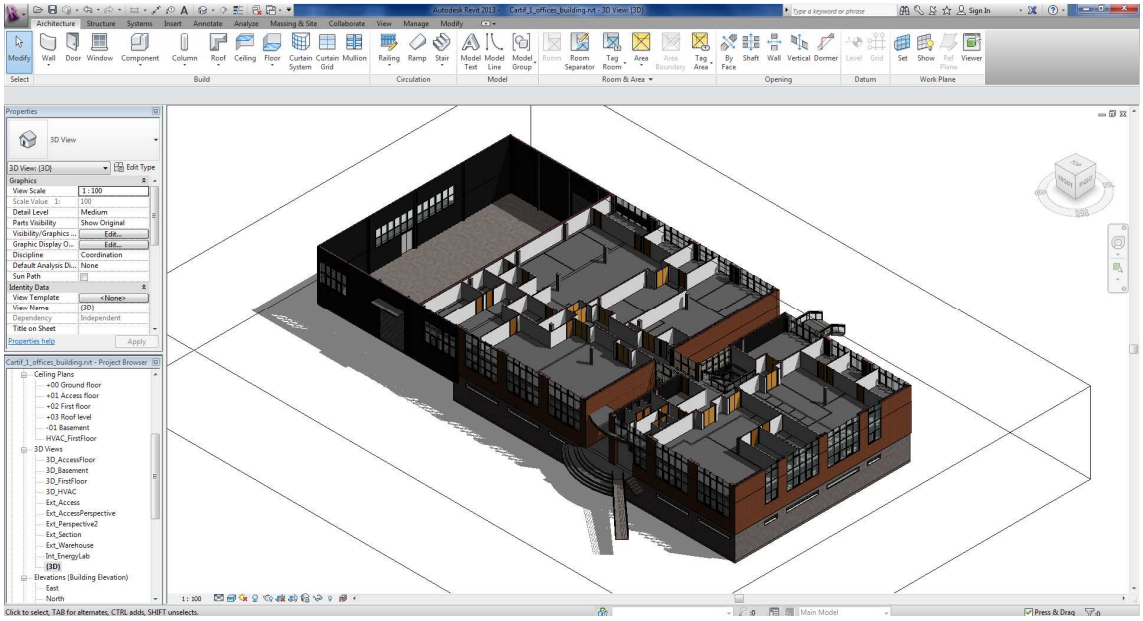

Figure 4: Model of Cartif building in Autodesk® Revit™ 2013

2.4.1.2 Constructivity

Constructivity One is an all-in-one software suite for building information modelling, featuring an effort to provide an integrated environment that supports the following domains/functionalities:

- Architecture
- Structural Design
- HVAC Design
- Electrical Design
- Plumbing Design
- Scheduling
- Construction Management
- Facilities Management
- Building Automation

The functionality of the software suite is achieved through three distinct software modules: the Constructivity Model Server, the Constructivity Model Editor and the Constructivity Model Viewer.

Constructivity Server serves as a repository supporting automatic merging and conflict resolving services, and allowing all (design) team members to create, update and view projects. Of course, due to its limited functionality (with respect to BaaS requirements), it cannot substitute TNO BIMServer.

Constructivity Model Editor, although supporting 3D model editing, features a different design approach compared to other model editors (like Revit), since it is IFC-oriented, i.e. all elements and interrelationships between the elements of the model are correlated to the respective IFC objects. This property, although might hinder users migrating to Constructivity from other modelling software and evoke steepest learning curve, enables faster and more informative IFC design process. In addition, the fact that Constructivity Editor supports both IFC2X3 and IFC4

(RC4)[6] data models necessitates its adoption within BaaS. Figure 5 shows the 3D editing interface, while Figure 6 shows the IFC4 entities list, corresponding to the same building.

Constructivity Model Viewer is the third component of Constructivity One suite, using the same engine as the Constructivity Editor and allowing browsing and 3D visualization of both IFC2X3 and IFC4 objects, including among others[7]:

- Building models
- Product types
- Structural elements
- Building systems
- Materials

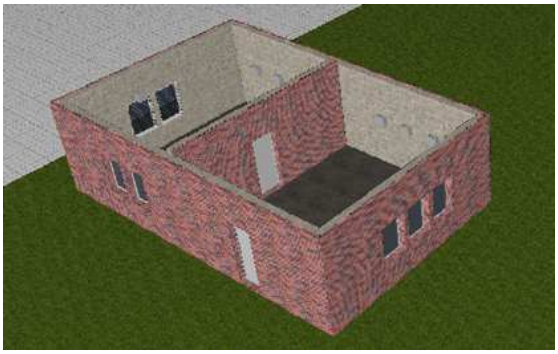

Figure 5 Intermideate stage of a simple building design process, using Constructivity Model Editor

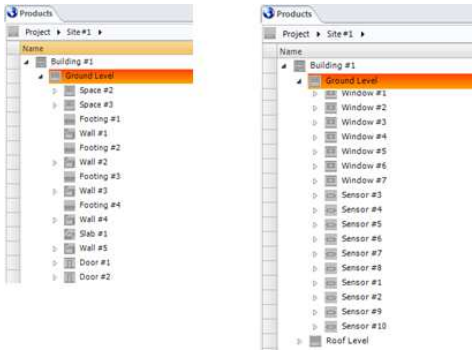

Figure 6 IFC4 editable properties of the building using Constructivity Model Editor

2.4.2 IFC Viewers

2.4.2.1 Solibri Model Viewer[8]

Solibri Model Viewer is free-of-charge software tool built for viewing Open Standard IFC files and Solibri Model Checker files. Solibri Model Viewer brings BIM files from all IFC compatible software products available, being able to import IFC2.0, IFC2x, IFC2x2, and IFC2x3 models. This software tool works on Windows and MacOS platforms and allows sharing IFC models among the different stakeholders involved in a specific project.

[6] In fact it is the only software capable of graphically editing IFC4 entities

[7] http://www.constructivity.com/cmviewer.htm

[8] http://www.solibri.com/solibri-model-viewer.html

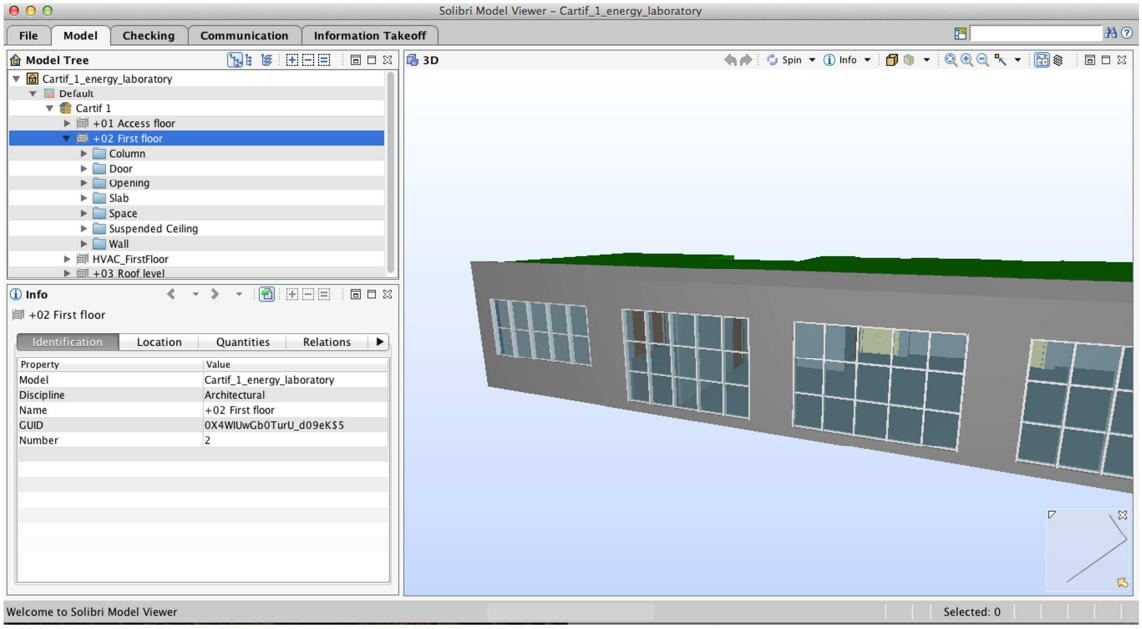

Figure 7: Model of Cartif's Energy Department in Solibri Model Viewer.

2.4.2.2 Tekla BIMsight[9]

Tekla BIMsight is a software application for building information model-based construction project collaboration. It can import models from other BIM applications using the Industry Foundation Classes (IFC) format. With BIMsight, users can perform spatial co-ordination and visual checks for design and constructability issues, automate clash detection and mark-up the model with notes and redlines.

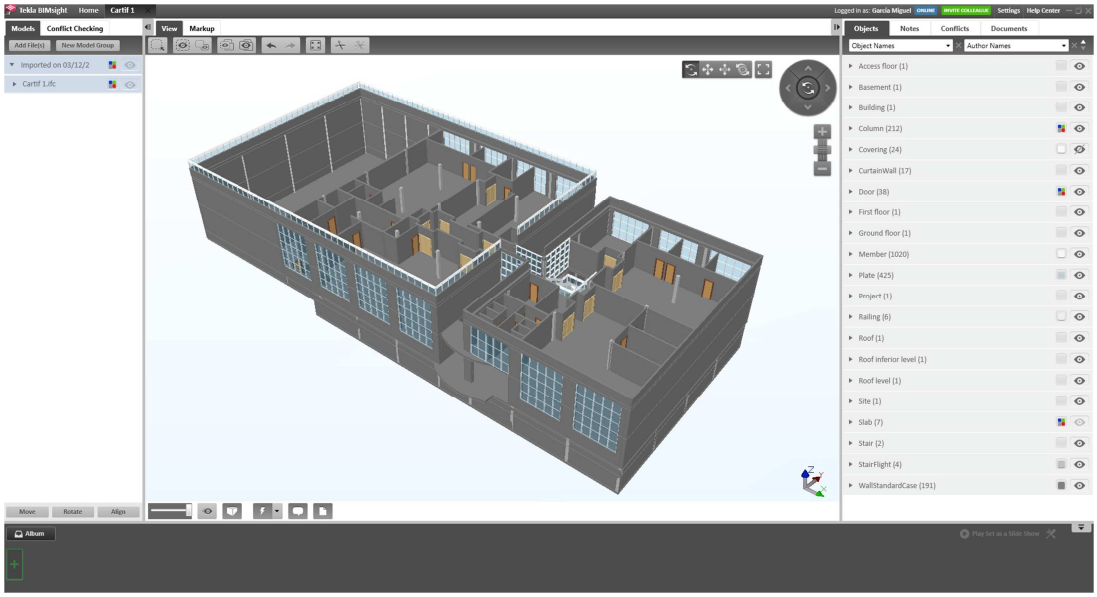

Figure 8: Model of Cartif's Energy Department in Tekla BIMsight.

[9] http://www.teklabimsight.com/getStarted.jsp

2.4.2.3 BIMsurfer

BIM Surfer10 is an open source web-based viewer for the visualization of IFC/BIM models based on WebGL, implemented in JavaScript. Although the collection of features it offers as a viewer is comparable to the capabilities of existing solutions, it supports a relationship to the TNO BIMServer that can prove advantageous in many cases, even though it does not support IFC4 entities. In fact, BIM Surfer is the only IFC model viewer (among the collection of viewers utilized and reviewed within BaaS) that features a direct connection to TNO BIMServer.

Figure 9: BIM Surfer connection to TNO BIMServer

Figure 10: TNO BIMServer projects available through BIM Surfer

Under this perspective, BIM Surfer supports two IFC import methods:

- SceneJS File: Users can import a local SceneJS file (JSON format), which is then processed by the Surfer to visualize the model. Since TNO BIMServer provides the capability to convert IFC files to JSON format through the SceneJS serializer plugin, the collaboration between the two modules is smooth, without necessitating any extra effort.
- Connection to a server: The second method allows the user to directly connect to any TNO BIMServer and select a project as an input to the Surfer. Thus, an initial screen requiring the server address and user credentials is provided (Figure 9) and after login the set of projects visible by the user are available for selection (Figure 10). Subsequently, using the provided TNO plugins for the serialization and download tasks, the 3D representation of the model is shown (Figure 11). Note here that this method is hindered by the security restrictions of several web-browsers, thus BIM Surfer is planned to interact as external service to the BIMServer through the BIM Service interface exchange (BIMSie) standard.

[10] http://bimsurfer.org/

Figure 11 TUC Building preview in BIM Surfer

3 Requirements for the BIM Server

The Appendix C of the Deliverable 1.1 [3] collects all the requirements for the BaaS system regarding both the WP2 and the remaining WPs. Therefore, in this chapter it is collected, analysed and extracted the specific and harmonized list of technical requirements about the BIM repository and its interface. This list will have to be considered as the unique list of requirements used to test all WP2 outcomes in relation to BIM system and its interface.

Name	**FR-02.2:** Interoperability
WPs affected	WP 2
Description	The system should interwork in heterogeneous networks. • The BaaS system should guarantee an appropriate interconnection among all their internal pieces of software (APO services, modules, components) as well as with external data sources and tools (BMS/BACN (Building Automation and Control Network), BIM server, DWH, external systems and services, and external tools). The whole distributed "eco"-system (regardless of being deployed locally, in a cloud, or a mix thereof) should communicate transparently and maintain coherence and consistency of data transferred. • The BaaS system should be able to communicate (read & write access) with existing building information model (BIM) repositor(y/ies). The interface (connector) implementing the protocols provided by the BIM repositories should be developed. In case that more than one BIM repository will be used by the BaaS system, the BaaS system should be able to communicate in a homogeneous and coherent way with all of them. The BaaS system should maintain the coherency of data. • The BaaS system should be able to communicate (read & write access) with the APO Services, providing this layer with all the data needed from the Data Layer (BMS, BIM, DWH, etc.). • The BaaS system should be cloud-enabled. The BaaS system should implement (or use from external servers/providers) those *Platform*, *Infrastructure*, and *Software as a Service* (PaaS, IaaS, SaaS) models needed in a "cloud environment" in order to guarantee the interoperability among all the components which make up the BaaS system. • The BIM server(s) should be provided at least one open or standardized protocol to establish the communication with the BaaS system.
Importance	Critical
Rationale	The communication amongst all the components is necessary for the proper behaviour of the whole system.

Table 2: System Management Requirements: Interoperability (functional)

In the Table 2, the interoperability requirement is collected. This requirement specifies the need for the communication of the entities in the whole system including the BIM repository. For that

purpose, there is a need for the retrieval of data from the BIM because other modules will demand the information of the building. Thus, it is required to integrate a connector through a well-known interface with the BIM data in order to offer this information to other entities. Moreover, the BIM server could be presented in a cloud system, which means several entities of the BIM could be deployed in different servers so that the information would be accessible from multiples sources. This fact raises the scalability of the system because any other additional BIM could be added to the cloud and it could work together the current ones.

Name	FR-02.3: Openness
WPs affected	WP 2 & WP 3
Description	The system should work with open systems where possible based on SOA (Service Oriented Architecture). • Solutions based on FLOSS (free/libre open source software) should be used. Use of FLOSS components should be encouraged and promoted (e.g. OpenBIM Server; LON- or BACnet- based BMSs; open and relational DWHs; M-BUS based meters). If the use of FLOSS is impossible, then the BaaS platform should use proprietary software (proprietary BIM server; proprietary BMS, etc.). • The BaaS system should implement open or standardized protocols for the communication with the BIM Server. The system should be able to query the BIM Server using different kind of filters (site, building, storey, room, system/subsystem objects, object types, object properties). • The format of the BIM repository should conform to an Open Data Formats (model) (better if standardized) representation, such as the IFC (latest implementation).
Importance	Standard
Rationale	BaaS activities should foster openness and the adoption and use open standards

Table 3: System Management Requirements: Openness (functional)

This second requirement, presented in the Table 3, is related to the use of open-source libraries for the implementation of the interface with the BIM Server. To fulfil this requirement, the connector should use free license libraries or modules in order to communicate the BIM repository whenever is possible.

Besides the open-source libraries, the usage of standard protocols and/or standard information representation (data models) makes the chance of standardising the software developments and the easy-handle of the system.

Name	**FR-03:** Data Management
WPs affected	WP 2
Description	The system should be able to maintain data consistency and to ensure high

	availability of the data • The system should be able to securely backup data and restore it if needed. Multi-level incremental backups are preferred. • The system should be able to keep historical records / logs of access, modification, deletion, etc. of data. • The BaaS system should be able to read and write information from/to the BIM repository, including: a) the entire BIM model, b) specific information (object properties, list of sensors, etc.) of the BIM repository. • The system should be able to write, update, or delete information into/from the BIM repository. For instance, sensors/actuators malfunctions could be detected by the BaaS system (*fault detection and diagnostics service*), so this new state of the sensor/actuator should be able to be updated in the BIM repository; as well as new sensors could be commissioned in the BMS/BACN system, so this new object should be added in the BIM repository.
Importance	Critical
Rationale	Good data management is crucial for the resilience and fault tolerance of the entire system.

Table 4: Data Management Requirements (functional)

Finally, Table 4 represents the Data Management requirement, which is a very important constraint for the whole system. The consistency and coherency of the data is very significant in any software system. In that way, BaaS has to ensure the information is compliant with the rest of the entities of the system such as the BMS sensors. Thus, the system should be able to read/write/update the information in the BIM so that the consistency and coherency would be assured.

On the other hand, periodical backups avoid the loss of data when an error is suddenly thrown. Therefore, the maintenance of backups for recovering lost information is a task to bear in mind during the development process.

It has to be noted, there are some additional requirements in the aforementioned tables, which are more specific for the BIM system but they are represented in a more general way in the Deliverable 1.1. Because of the needs for the BIM, it is required to specify these conditions in that sense in this deliverable. Besides this mandatory list of requirements, it could be established optional ones, which are not collected in the global list because the first approach only includes the mandatory needs for the system. Thus, WP2 has detected one possible additional requirement with regard to the BIM:

- BIM Server could provide one easy-to-use graphical user interface.

With all the requirements collected, the design of the BIM server connector must be thought and adapted to the constraints presented. Thus, this data source stores the static information of the systems, more specifically, the building information. On the other side, the DWH saves the dynamic data which jointly the BIM information set up the extended BIM concept (combination of DWH and BIM resources). Both the static and dynamic data should be provided by the WP2 to the middleware in order to manage the information of the system. Thus, the task 2.3 covers the communication with the BIM server whereas the middleware is the responsible of the

connectivity from the upper layers to the data sources. Summarizing, the task 2.3 interfaces the BIM server for connecting and querying the BIM server whereas the Communication Logic Layer is in charge of routing the requests from upper layers.

3.1 Meeting the BaaS Requirements with an IFC-based BIM Server

The following table illustrates the features of the IFC ecosystem and how the BaaS project requirements, presented in the previous section, can be addressed by the adoption of an IFC-based solution:

BaaS System Requirements	IFC Model Solutions
Interoperability: Communication among the pieces of software	IFC Data Model eases the communications in the same way among all the entities by establishing the communication rules to be followed by all the software entities.
Interoperability: Communication with BIM Server repositories	BIM Server was developed based on the IFC data model specifications and, consequently, it is IFC based and compatible. The use of this standard data model (ISO 16739:2013) enhances the compatibility of the BaaS project system with different data repositories.
Interoperability: Cloud deployment	IFC enables the deployment in a cloud using this data model in order to standardize the communication in the cloud.
Openness: Free license software, open or standard data formats and protocols	IFC Data model is the registered standard ISO 16739:2013, specifying a conceptual data schema and an exchange file format for Building Information Model (BIM) data.
Data Management: Read&Write data information	Since BIM Server is using IFC data model as a common language, there is no need for translation of data formats.

Table 5: BaaS Project Requirements vs. IFC features

4 BIM Server specification

4.1 Selection of software

4.1.1 BIM Server v1.2 by TNO

The BIM server offers a software system architecture designed to ease the communication with external entities and allowing centralizing the information of any building. The BIMServer's core is based on the IFC standard and therefore it is perfectly able to handle those IFC data. Figure 12 shows the level layers implemented by the BIM Server from TNO.

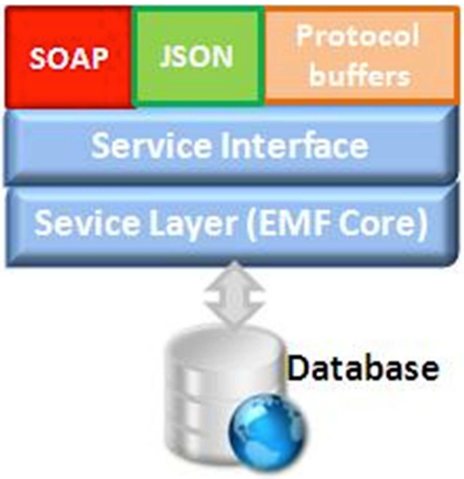

Figure 12: BIM Server system architecture

The lowest level layer is the database, which stores the entries as key-value pairs accessible through the KeyValueStore Interface. The implementation of this database is Oracle Berkeley DB Java Edition [12], an open source, embeddable, transactional storage engine written entirely in Java and running in the Java Virtual Machine without the need of a remote server. In difference to a relational database, this one stores object graphs, objects in collections, or simple binary key/value data directly in a B-tree on disk. This kind of implementation reduces the complexity in the communication between objects and relational databases. In this way, an object which is annotated as persistent is directly stored.

The upper layer provides the core of the BIM Server and it is called Service Layer (EMF Core). The core manages the behaviour of the BIM Server through all the functionalities allowed. It receives the requests from the upper layers and manages them in order to work with the objects and models so that the information could be exchanged among all the entities involved, including the database and other external entities.

The last internal layer of the BIM Server is the Service Interface that is the Java interface of the BIM Server and all the activity and communication with the core and the main functionalities are done through this layer. This interface provides all the methods for external applications to access and manage the data stored in the BIMServer. That means the Service Interface is the link among the core, the functionalities of the BIM Server and the communication with the external tools.

For all these reasons and based on the BaaS BIM server requirements listed in the Section 3, TNO BIMServer has been selected for the task, featuring a collection of properties that cover

the posed interoperability (Table 2), openness (Table 3) and data management (Table 4) requirements, among others.

4.1.1.1 Openness

As previously stated, TNO BIMServer supports the IFC data model standard, thus enabling smooth interaction with the plethora of external software that support the new standard. Even though other data model representations exist (like gbXML), the capability of TNO BIMServer to support the only validated data model is sufficient for BaaS purposes.

In addition, the TNO BIMServer is an open platform, providing the ability to acquire the code from repositories and customize the server. This way, the server can be adapted to other IFC versions (e.g. IFC4 – see Section 4.1.2), can be re-compiled using custom properties and supporting newer Java versions or several parts supporting specific tasks can be adopted to BaaS needs.

The server customization is also enabled by the modular architecture of the server, based on plugin development. In this approach, the vital tasks supported by the server are developed as plugins, available to the user. This way, more than one plugin for the same task can be available, addressing different requirements, and users can develop custom solutions suitable for their needs, and incorporate them into the server through the plugin interface, without modifying the server core. The basic plugins, necessary for BaaS are the following:

- Serializer plugin: a serializer converts an object model stored in the server to a stream of data. This way, any available model can be exported by the server to all available (supported) formats, such as IFC, IfcXML, cityGML, etc.
- Deserializer plugin: the deserializer performs the opposite task to the serializer; i.e. converts a stream of data (e.g. an IFC file) to an object model to be stored in the server.
- QueryEngine plugin: the query engine allows users to query any model stored in the BIM server.
- Service plugin: it can extend the functionality of the server, by registering new services to the server functionality.

Note here, that even though the provided plugins manage to support a plethora of tasks, Model View Definition (MVD) support is unavailable (see Section 4.3.2 for more information). Thus, within BaaS a new plugin will be developed, able to identify whether a provided model is MVD compliant and able to process the model in order to satisfy the requirements posed by the MVD if possible.

4.1.1.2 Data management

TNO BIMServer utilizes the Eclipse Modelling Framework (EMF) to provide a modular and object-oriented representation of the IFC schema. Here, the IFC STEP/EXPRESS file is parsed and the included classes (more than one thousand) and their interrelationships are converted to an EMF Core (ECore) file. Subsequently, the EMF framework extracts the information from the ECore file and generates Java classes, passed to the database layer.

In the database layer, functionality similar to version control software (like subversion) is provided and the versioning system is governed by the following principles[11]:

[11] https://code.google.com/p/bimserver/wiki/Database_internals

- Models are stored in projects, while a new model version is a new project revision.
- All project revisions are accessible and cannot be altered

This way, any model update (full or partial) is supported as a new project revision, while the availability of the revision history allows for reverting to previous versions automatically. Finally, even though there is no automatic database backup mechanism available, manual backup is an option, by copying the server workspace ("home" directory) to the backup, without losing any information.

4.1.1.3 Interoperability

TNO BIMServer is written in java, thus providing the capability to be deployed as an executable JAR or WAR file in any operating system supporting Java. In addition to this, the capability of embedding the server in another application is provided, thus allowing server hosting in a large variety of platforms, including the BaaS infrastructure.

Once the server has been deployed, external services (like a BaaS fault detection module) require access to the models to query information preferably through well-established and open protocols. In order to achieve that, TNO BIMServer has adopted and supports a variety of connection options, as shown in Figure 13, including Soap [14], Protocol Buffers (PB) [15] and JSON messages [16]. On top of that, a JavaScript API is also provided.

All external services requiring information from the server can use any of the available protocols to communicate to the "ServiceInterface". This is a java interface containing all the available methods an external service can call[12]:

- Basic calls: login, create projects, check-in/checkout revisions, query models, manage users, etc.
- Administrative calls: setup servers, check logs, manage database migrations, etc.
- Settings calls: view and edit server settings.

Note here that the Service Interface is implemented both utilizing BIMserver-specific calls and using BIMsie standard[13] implementations.

TNO BIMServer features two properties further elaborating external services access:

- Automatic generation of description files: suppose that some additional methods are required by an external service; these methods will be implemented by the user and added to the Service Interface of the server. From there, instead of manually updating all the necessary components of the server, a provided script generates the new WSDL and Proto files, which define the available calls for the Soap and PB protocols.
- BimServerClient Library: to facilitate the connection to the server, a java library is provided. Here, two scripts parse the created WSDL and Proto files and generate a SoapClient and a ProtocolBuffersClient respectively, utilized by the provided library to accommodate communication to the server in a transparent way. Note here, that the BimServerClientLib also provides access to the plugins and the EMF core client-side, thus providing server-side capabilities to the client.

[12] https://code.google.com/p/bimserver/wiki/Interfaces

[13] http://buildingsmart.github.io/BIMSie/

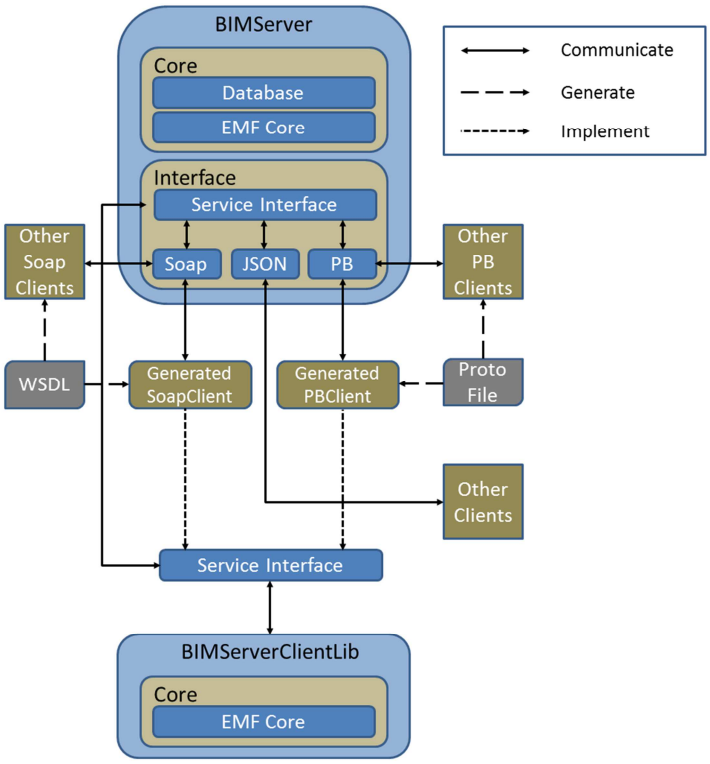

Figure 13: TNO BIMServer communication scheme

Overall, the provided communication architecture and philosophy establishes TNO BIMServer as a generic framework with enhanced interoperability and extendibility properties, thus rendering the selected server suitable for BaaS requirements.

4.1.2 Adaptation to IFC4

Since within BaaS the use of IFC4 and TNO BIMServer has been adopted, a server version capable of supporting IFC4 was vital. Unfortunately, due to the transitional phase from IFC2X3 to IFC4 all available server versions support IFC2X3, thus it was decided to manually adapt the TNO BIMServer to IFC4 by using the publicly available code.

In order to accomplish that, a decision had to be made on the server version to be adapted. Several trials on BIMServer 1.1 indicated that this version was not suitable for the task, since it included an earlier version of the embedded database holding the IFC schema that could not be re-configured. On the other hand, during the first year of the project, BIMServer 1.2 was under development, with constant bug fixes and enhancements. In addition, the IFC4 schema version had not been finalized yet (IFC4RC4) Thus, initially, we decided to utilize an early version of 1.2 (nightly build – 26/09/2012) that provided the necessary enhancements for the task at hand, while exhibiting the robustness and stability of version 1.1[14]. Thus, this version of the server was adapted to support the current IFC schema (IFC4RC4).

Although the server was adapted successfully, the utilization of Soap as a connection option to the BIMServer led to incompatibility with the OSGi framework setup of the BaaS System, due

[14] In fact, progressive interaction with newer versions of BIMServer 1.2 (RC2-6) revealed the unstable version of these release-candidates and supported the initial decision.

to different versions of required software bundles. In addition, the final IFC4 release facilitated a substantial number of variations compared to the previous IFC version (IFC4RC4).

In order to address the aforementioned limitations, the final stable version of BIMServer 1.2 (released at July 6th, 2013), including the more suitable for BaaS requirements JSON interface, has been adopted to the final IFC4 version. Here, the available IFC schema is represented internally in the server using the Eclipse Modeling Framework (EMF), which in turn is used to generate the proper model files to be used by the server. More analytically, the update process facilitated the following steps:

- The latest IFC4 description file (IFC4.exp) was downloaded by the buildingSMART alliance website;
- Using the downloaded IFC4 step-file and the buildingSMART plugin of the TNO BIMServer, an EMF ECore file was generated, including all the IFC4 entities and interrelationships, represented using the EMF;
- The internal database, as well as the packages imported in all other files of the server, was adapted manually to be able to support the transition from IFC2X3 to IFC4;
- The code was migrated to IFC4 using functionality provided by the server and the new java files including the IFC4 model were generated using the EMF plugin of Eclipse;
- Finally, all resulting compilation errors due to the incompatibilities between IFC2X3 and IFC4 were manually edited and corrected.

After successfully completing the aforementioned steps, a stable BIMServer version capable of supporting IFC4 files manipulation was available. The server was tested using two IFC4 simple example files provided by the buildingSMART alliance (example_ifc4_wall.ifc and example_ifc4_house.ifc) as well as a custom two-room office building designed using Constructivity Model Editor[15]. The first two files were imported “as-is”, while the IFC file generated from Constructivity necessitated some minor alterations on the windows and doors styles for the import process to be successful. Finally, a number of test queries on all files/projects indicated the conservation of the correct properties throughout the overall process.

4.2 Definition of the interface with other layers

In the BaaS system, as shown in Figure 14, four interfaces are associated between Data Layer and Communication Logic Layer (CLL) as follows:

- I-5: provides direct access to the BMS of an asset for collection of dynamic building data and actuation or to additional external ICT systems, which are not integrated with the BMS.
- I-6: links the Communication Logic to the Data Warehouse for storing and retrieving historical data, e.g. dynamic building data, optimization or prediction results, etc.
- I-7: is used for accessing the building information model holding static information on the building.
- I-8: connects to external services, which provide additional data required for optimization and prediction, such as weather data.

All the communications required to exchange data between the BIMServer and CLL (implemented in the BIM connector) are provided by the I-7 interface. In order to follow the openness requirement of the BaaS system and to be in line with the chosen Server (BIMServer

[15] http://www.constructivity.com/

by TNO), this interface will be implemented in JAVA. Additionally, all the communications performed over every entity of the BIMserver will be managed in the server side and, as previously stated; those communications will be carried out using either SOAP (Simple Object Access Protocol) or JSON (JavaScript Object Notation) or protocol buffers.

As part of the activities of the Task 2.3, the connection with the different entities of the BIM Server has been defined and specified, as well as the management of all those communications. To that end, after analysing all the possibilities available, the BaaS connector will perform the communication between the connector and the BIMServer through JSON Services, which is a text-based open standard designed for human-readable data interchange and derived from the JavaScript scripting language. The data represented by JSON are called objects. The BIM Server includes useful libraries in order to communicate any client through the Web without having to be concerned about the implementation of the interface. Finally, and with regard to the specific queries to manage the information store, the BIMServer provides the necessary libraries and methods (Appendix A:) to manage the geometric models, user information, project objects and specifications of architectural elements.

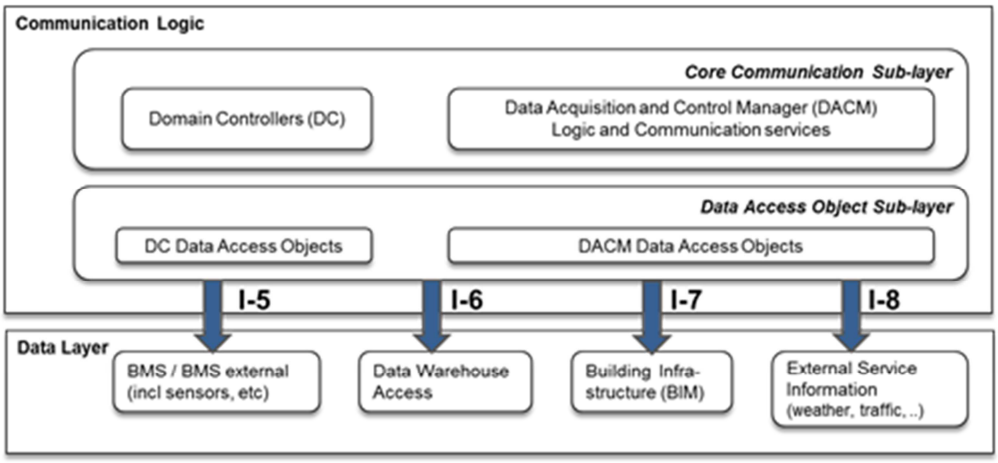

Figure 14: Interface I7 client into DAO sub layer [3].

4.2.1 BIM Server deployment scheme

With the aim of meeting the cloud development requirements stated in the Description of Work, the number and exact location of the BIMServer(s) must be defined, together with the strategy to manage all the connections between them and the CLL. Therefore, in order to both reduce the process charge of the computers running the BIMServers and facilitate the maintenance tasks, the project team decided to deploy two BIMServer entities, one of them for the Spanish case studies hosted by Cartif and another one for German and Greek ones, such as it is shown in the Figure 15, which eases the maintenance of the entities of the BIM Server.

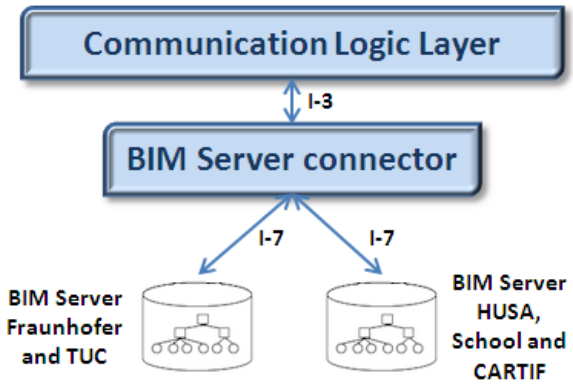

Figure 15: Deployment scheme for the BIM Server

Additionally, with respect to the strategy to manage the connection with them, the connector must be able to identify each of the operative BIM Server entities and, to that end, and considering that only two BIM Servers are running, the IP address, the port number, user and password (to be shared internally among partners) are required to establish a link with a particular BIM Server entity and model.

With regard to the information among all the entities in the system, the exchange of these properties gets involved all the layers in the architecture because all should know how to communicate any single BIM Server or BIM model. Nevertheless, the only component that must know this information is the connector, because sharing IPs, users and passwords is not secure. Therefore, another mechanism is more useful than the sharing of all the information. Thus, a set of identifiers has been defined in order to boil down the BIM model within the specific BIM Server that the connector should query. Table 6 displays the map between the building (every single entity of the BIM Server) and the identifier for the communication, which it has been decided to be the IFC building identifier from the building IFC model. Thus, when the connector receives this parameter in the event, it is able to filter the suitable connection.

Building	Building IFC code	IP Address	Port
CARTIF	0X4WIUwGb0TurU_d3sNheR	193.146.230.54	8082
Fraunhofer	1q80AVNS16bBczA_J1Snt3	147.27.11.33	8080
TU Crete	3SnGK6xX9DWO4W9hsuDeMF	147.27.11.33	8080
HUSA Chamartin	0AuePbfAfFTx$LFOL9pMXy	193.146.230.54	8082
Santa Elvira School	3iN1BqPX5DGe$YmokcIg7V	193.146.230.54	8082

Table 6: Map of the BIM Server and the code

The interoperability requirements are not only limited to the communication with the BIMServer but also with the upper layers, particularly with the DACM layer of the CLL. Since the BaaS project represents each of the data layers of the architecture as an independent service, this communication will necessarily imply services mechanisms and the implementation of a Service Oriented Architecture (SOA). In this case, the OSGi framework has been selected as the SOA-based approach to facilitate the interworking among services and plug-ins, and the communication will be based in events detection as it is specified in the WP3 (please refer to the Task 3.3 for more detailed information).

4.2.1.1 Test environment for running queries

Within BaaS project the BIM Server v1.2 has been used as basis of the container for the BIM models of the different buildings. However, as aforementioned, this BIM Server is compliant with IFC2x3, meanwhile, for BaaS purposes, the BIM Server has been adapted to IFC4. As new development, in beta version, it needs to be tested. In order to develop, check and run queries for every single building, the proposed environment keeps one instance of the BIM Server for each building in the project being five the number of entities during the test phase. In the test environment, there is no communication between the BIM Server and other layers or entities in the BaaS system, but it is running as a stand-alone application. The connectivity and tests will be performed through the client Java code in the development environment.

4.2.1.2 From the test environment to the final deployment

Once the tests prove the stability of the BIM Server, running queries, keeping the service active as much as possible and giving a relative low time response in the results of the queries, the BIM Server is going to be deployed into the final version. Yet, before the final implementation, the communication between the service and the Communication Logic Layer has to be tested. For that purpose, TUC as developer of the "adapted" BIM Server will host the server in the TUC facilities in order to control and manage the possible exceptions and errors during the lifecycle of the systems in the communication tests. Finally, with the feedback of the two tests environments and the possible changes in the behaviour of the BIM Server, the final deployment shown in the Figure 15 will be carried out.

4.3 Clients' Development guidelines

4.3.1 BIM connector class diagram

As previously mentioned, for compatibility and openness reasons, the connector will be developed and programmed in JAVA language. This section provides the class diagram to be followed for the development of the connector between the CLL and the BIMServer. This diagram, shown in the Figure 16, is a first approach, and the main schema can be divided into three parts: OSGi, handler and communication.

As it can be observed in the diagram, the OSGi part is composed by the BundleActivator, the ServiceReference and the Activator classes, all of them responsible for publishing the connector as a bundle in the OSGi framework context. Thus, the connector could be treated as any other plug-in in the system and it is aligned with the CLL framework in order to ease the communications. In addition to this, this part subscribes the OSGi events [17] that have to be received/sent by the connector. A brief description of each of the classes can be found below:

- BundleActivator: It is the interface from the OSGi framework, which publishes the bundle as an OSGi plug-in in the BaaS system. Therefore, the connector must implement this interface in order to activate the BIM server connector as an OSGi plug-in.
- ServiceReference: This interface, belonging to the OSGi framework, is in charge of adding, modifying and deleting services. Not only the connector should be an OSGi plug-in, but also it should add the services for making use of the operations offered. Therefore, the Java development must implement this interface where the bundle is subscribed to the events and it also adds the events to be launched as services.

- Activator: It is the class destined for implementing the OSGi interfaces and their methods for starting up the component as an OSGi bundle and adding the events as OSGi services.

The second area of the diagram is composed by the handler classes, the EventHandler and the BIMHandler:

- EventHandler: This interface, part of the OSGi framework, must be implemented in the connector in order to handle the events published in the framework, allowing the management of them.
- BIMHandler: This is the class implementing the EventHandler interface from OSGi in the BaaS system. Thus, when an event whose topic (name) is the expected by the bundle could be managed by this class. This means, when such an event is received, the BIMHandler class checks the type of event and assigns an operation code for filtering the tasks to be fulfilled and the queries associated. With this identifier, the handler gives the control to the BIMManager in order to complete the operations.

Finally, the third part is the communication one which will be in charge of the connection between the BIMServer and the rest of the BaaS System. This part is composed by a BIM Manager for handling the communication between the upper layers and the data resources, the BIM Communication that manages the communication directly to the BIM Server and the queries and persistent classes (MVDDefinition and Sensor), which represent the basis for the queries to be carried out. The in-depth explanation for each class is below:

- BIMManager: The BIMManager class receives the operation from the handler as a function of the OSGi event received and at that point, two possible operations have been identified: storing and retrieving data. This manager redirects the requests from the OSGi framework to the communicator side of the connector acting as the interface between the class for the communication with the Communication Logic Layer (BIMHandler) and the data source connectivity (BIMCommunicator).
- BIMCommunicator: This is the class for the direct communication with the BIM Server data source in order to establish the JSON connection and query the information. This class receives the request from the Manager with the operation code for that purpose. Thus, with the building ID, the connector determines the server and the model which should be queried. Then, if the operation is the retrieval of data, the communicator reads the information requested in the MVDDefinition which is described below. Once it knows this information, it is able to query the specific information in the BIM Server, performing the appropriate queries. In the case of storage, the communicator reads the sensor information object and queries the BIM Server with the new data to be updated.
- MVDDefinition: This class represents the information required by the system and those data which should be queried from the BIM Server. Thus, this class identifies the MVDs with the operation code in order to perform the queries associated to the MVD. As an example to illustrate this, supposing the operation number is “1”, and it is associated to the MVD for the FDD module, the MVD class will return all the queries required to fulfil that MVD requirement in terms of IFC building information. Thus, the class contains a set of XML files based on the mvdXML files provided by the IFCdoc software. In this way, the pre-loading of the files in the component eases the communication emitting less information in the request. For parsing these files, any software is needed for generating the available queries, which is included inside the BIMCommunicator code.
- Sensor: This class represents the attributes from a sensor, actuator or facility able to be modified in the BIM Server when an inconsistency or incoherency is detected. Thus,

when the operation is “to store”, the communicator makes use of this information. That is the only “semi-static” information identified and susceptible to be changed in the BIM Server at the moment. If further information should be updated, the mapping class would be added. Finally, the buildingId property is useful for filtering the building where the query has to be run.

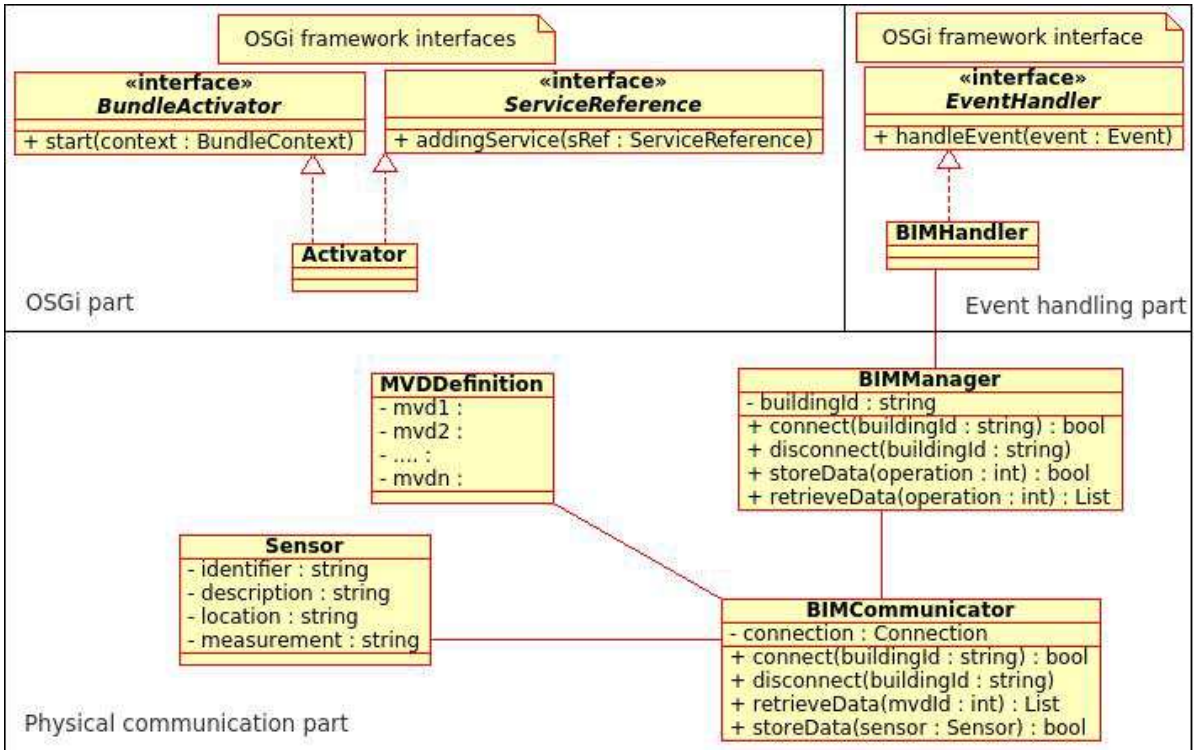

Figure 16: BIM connector class diagram

In summary, the behaviour of the connector and the class interaction is summarised in the following steps:

1. The connector receives an OSGi event that should be properly handled by the BIMHander for filtering the operation to be fulfilled.
2. The BIMManager connects to the BIMServer/model specified in the parameter buildingId that is reached in the OSGi Event.
3. The BIMHandler and BIMManager are in communication depending on the event so as to filter the operation to be run both storage and retrieval of information.
4. The BIMCommunicator class runs the operation.
    a. In case of storage operation, the BIMCommunicator updates the information stored in the BIMServer, either changing the information in the model or the objects related to the Sensor objects (the unique objects able to be changed as yet).
    b. In case of retrieval operation, the BIMCommunicator retrieves the keywords in the MVD indicated by the operation number. After that, it builds the specific query to the server.

5. The connector makes up the object type with the information and sends an event to the upper layer in order to inform about the data or the update process.

Finally, it is important to be pointed out that the class diagram could be divided into two parts from the development point of view. First of all, the communication with the BIM Server is responsibility of the WP2, which involves the classes BIMManager, BIMCommunicator, MVDDefinition and Sensor. On the other hand, the WP3 is the responsible of the connectivity among the components of the BaaS System and; therefore, the WP3 makes use of the connector in order to integrate the OSGi communication, developing the OSGi classes: BIMHandler and Activator.

4.3.2 Queries' implementation

4.3.2.1 Understanding the queries

TNO BIMserver will serve as the base software framework, which will be expanded according to the specific needs of BaaS project. Since a great number of BaaS services will necessitate a vast amount of information from the BIM, a detailed investigation on the TNO-provided query libraries is essential, in order to determine useful existing functionalities, as well as to identify necessary extensions.

First of all, TNO BIMServer provides two available query engines along with the ability to define custom query engines as plugins to the server. The first engine is called Building Information Model Query Language (BimQL[16])[18]. BimQL follows the SQL language definitions and is capable of providing create, read, update and delete (CRUD) functionality. The motivation behind the development of BimQL is the necessity to provide the ability to query the complex IFC data model using a set of user-friendly and intuitive commands. Although BimQL provides an intuitive and straightforward way of querying the models uploaded to the server, the majority of the java classes implementing the functionality are developed manually, using the java classes generated by the EMF as guidelines. This implies that automatic adaptation to IFC4 (required by BaaS) is not possible, but extra effort is necessitated to adapt the classes to the new schema. In addition, since IFC4 contains additional functionality compared to IFC2X3 supported by the current BimQL version, a large amount of information will not be available for querying.

Due to the BimQL shortcomings (with respect to BaaS needs), the only viable option is the JavaQueryEngine plugin. Here, the java classes created by the EMF that contain the entities and their interrelationships define the available queries, thus query support is automatically provided when migrating to IFC4. On the other hand, due to this formalism, even simple queries like the one selecting all the building doors defined before require complex code, as shown in Appendix B:.

Moving forward, after selecting the suitable query engine for the task, a decision has to be made on whether the queries will be executed client- or server-side, since TNO BIMServer provides two possibilities:

- Download the whole model: The whole IFC model is downloaded client-side, stored in a proper object and the queries are executed locally;

[16] http://bimql.org/

- Download the query result. The query is sent to the server (as a stream), where it is compiled, executed and only the result is sent back to the client.

4.3.2.2 Defining and performing the queries

Although downloading the whole model as described above might seem as an unviable option within BaaS – since downloading the whole model multiple times can yield performance degradation during multiple simultaneous calls – the second option can lead to the same inefficiencies even if a smaller part of the model is requested for the same reason.

Due to that, within BaaS the full model of the building is downloaded once and a local copy is maintained in the proper object type (IfcModelInterface), available for queries. If the model is updated on the server, an event is generated, notifying on the necessity to update the local copy also. This way, the robustness of the BaaS system is enhanced, since the queries to the model are not hindered by the quality of the network or the installation properties of the BIM server.

5 Model view definition

5.1 MVD Concept

The use of BIM and IFC tools allows for semi-automatic deployment and operation of APO services in all buildings at hand, regardless of variations on the building types, construction, location and available systems. On the other hand, the use of BIM and IFC alone inserts more complexity to the problem, rather than simplifying the task, since requiring by all software components to provide support for the entire IFC schema is not a viable solution [19]. Due to this fact, the concept of Model View Definition (MVD) has been adopted.

According to BuildingSMART alliance[17], an MVD "defines a legal subset of the IFC complete schema and provides implementation guidance for all IFC concepts (classes, attributes, relationships, property sets, quantity definitions, etc.) used within this subset. It thereby represents the software requirement specification for the implementation of an IFC interface to satisfy the Exchange Requirements". Thus, the exchange requirements for each APO service are defined and made publicly available. Within this context, if two software components have to interact they need to exchange sufficient information – all the exchange requirements so that this communication is complete are defined in the MVD. So the "sending" component (let's call it the writer), should create all the information to be sent (in conformance to the MVD), and the "receiving" component (let's call it the reader), should know how to use the information (which comes in conformance to the MVD), to perform some useful task. So both the "reader" and the "writer" should be designed to satisfy the requirements posed by the MVD (i.e. understand the MVD).

Now, it is conceivable that there are many "writer" components, like CAD tools or GUI interfaces that populate aspects of the data model. BIM acts as the aggregator of such information, and the provider to clients (via available interfaces) of the requested information. Moreover, the availability of the BIM and the MVD description allows the generation of queries to the BIM based on the MVD, since the MVD actually determines which queries are supported, i.e. we can expect some meaningful data in the response. This way a "library" of queries for each exchange requirement (FDD, CDO, etc.) can be generated, and it will be automatically supported by all IFC files compliant to the MVD. Finally, following this approach, the APO service modules are equipped with auto-configuration capabilities, while new modules can be imported to the system through a trivial process, as long as they ensure compatibility with the exchange requirements of the respective service, i.e. they are MVD-compliant.

The most implemented MVD is the Coordination View developed by BuildingSMART, targeted at elaborating information sharing between the architectural, structural and mechanical engineering principles during the design phase of the building. Although the Coordination View is incorporated in IFC4, it provides minimum support to BaaS objectives, thus the BaaS MVD will be developed, defining and supporting the Fault Detection, Control and Thermal Simulation Exchange Requirements.

[17] http://www.buildingsmart-tech.org/specifications/mvd-overview

5.2 Development guidelines

The development of a new MVD is a complex process, since the resulting view has to be able to encapsulate all the necessary information to satisfy the Exchange Requirements, while at the same time being implemented in a structured way, easily expanded and utilized by MVD design processes for similar domains. To that extent, the notion of reusable data exchanged modules is introduced. These modules are called "Concepts" and IFC4 definition is based on their utilization.

Moving to practical implementation, although the process of defining and implementing MVDs has been mostly standardized [22], the availability and widespread use of the buildingSMART mvdXML standard, along with the accompanying MVD design tool (ifcDoc), lead to the selection of the specific tools. Here, the utilization of mvdXML is based on a set of concept templates, which are the reusable building-blocks for defining MVDs and are incorporated in the IFC4 schema. Note here that defining new IFC entities or new concept templates should be avoided, since it can take several years for the proposed enhancements to be adopted by the next IFC version, while in the meantime the BIM and external software should be customized to support them.

Within BaaS, the defined MVDs will be implemented using the ifcDoc tool and the resulting definitions will be exported in mvdXML files, which in turn will be used for compliance checking of the provided IFC files and for transforming data according to the defined MVDs.

However, from the connector point of view this concept slightly deviates because of performance issues. In the case of software, an MVD could be too large for reading and seeking the parameters to be queried in the BIM Server. It is for that reason that the connector is using simplified MVDs in which the parameters missed are the only information represented through a set of keywords as explained before. Thus, the communicator only reads the parameters in order to run the query.

5.3 MVD usage example

The necessity of MVDs will become obvious through a use-case scenario. Consider the two-room test building shown in Figure 17, equipped with the following sensors (Figure 18): one humidity, temperature and luminance sensor in each room; two window contact sensors in each room; and two contact sensors in each door.

Figure 17: The outside view of the test building designed in Constructivity Model Editor

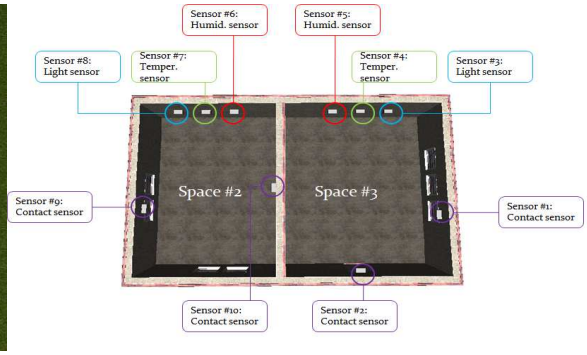

Figure 18: The test building floorplan along with the available sensors

Furthermore, assume that each room is served by an AC unit, covering both the cooling and heating demands of each space and that the following APO services are active on the building:

- FDD: every 2 minutes an FDD module verifies that the AC unit in each room is not operating while the window is open.
- CD: every 15 minutes the AC set-points are defined as the linear combination of the available sensor measurements of each room, including the window contact sensor (see for example the implementation of [20]).
- Thermal Simulation: every day a whole building simulation is initiated, using actual schedules from the building (occupancy, door and window opening, etc.), to estimate the total energy consumption.

In all three above modules, the information on which contact sensor belongs to each room is vital: the FDD module needs to know which window is open and if in the same room the AC is operating; the CD module needs to design the window opening strategy based on the sensor measurements of the specific room the window belongs to; and the Thermal Simulation module needs to map the logged opening schedules to the respective windows in order to assimilate the thermal behaviour of the building.

In order to acquire the necessary information, the IFC model of the building is uploaded to the specific TNO BIMServer adapted to support IFC4 files described earlier. From there, using the BimServerClientLib available calls, a set of queries can be initiated to request the information.

Thus, an initial query is deployed, supporting the following steps:

1. Get all IfcSpace objects of the building, and
2. Populate the IfcSensor objects that belong to each IfcSpace.

Upon completion, this query manages to discover only the temperature, humidity and luminance sensors, while fails to identify that the building is equipped with contact sensors. This is due to the approach followed during the design phase in Constructivity; there the temperature, humidity and luminance sensors were assigned to the respective IfcSpace object, while the contact sensors were assigned to the respective objects (windows-doors) they serve.

To overcome this problem, a different query is designed and deployed, facilitating the following steps:

1. Get all IfcSensor objects, and
2. Discover in which IFC object they are assigned to.

This query manages to correctly discover all building sensors, but a new query is required to determine in which IfcSpace each window equipped with a contact sensor belongs to. Moreover, a potential problem arises if an IfcSensor object is not attached to any room element (for example a sensor is attached to the roof of the building).

In order to provide a generic solution to the specific problem, an MVD is defined, *requiring all IfcSensor objects to be assigned to the respective IfcSpace they serve*. The impact of the specific MVD to the designed IFC files is twofold: not only each IfcSensor object necessitates an additional relationship to the respective IfcSpace, but also the definition of IfcSpaces is obligatory to ensure compliance with the defined MVD.

References

[1] BaaS Project Team, Deliverable D2.1: Data Warehouse Requirements and extended BIM

[2] Specification, BaaS Project, 2012.

[3] BaaS Project Team, Deliverable D3.1: High-Level Architecture, Interfaces Definitions, Data Models Extension Description, Specification, BaaS Project, 2012.

[4] BaaS Project Team, Deliverable D1.1: Theoretical Case Studies, BaaS Project, 2012.

[5] BaaS Project Team, Deliverable D3.2: Functional Architecture Specification, BaaS Project, 2012.

[6] B. Bruegge and A. H. Dutoit, *Object-Oriented Software Engineering Using UML, Patterns, and Java*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009.

[7] I. Jacobson, The Unified Software Development Process, ser. Object technology series. Pearson Education, 1999.

[8] BUILDINGSmart, IFC4 – the new buildingSMART Standard, BUILDINGSmart, viewed on 15th March 2013, <http://www.buildingsmart-tech.org/specifications/ifc-releases/ifc4-release/buildingSMART_IFC4_Whatisnew.pdf>

[9] J. O'Donnell et. al. SimModel: A domain data model for whole building energy simulation. *12th International IBPSA Conference. 2011*.

[10] V. Bazjanac and T. Maile. IFC HVAC interface to EnergyPlus-A case of expanded interoperability for energy simulation, *Lawrence Berkeley National Laboratory*.

[11] B. Dong et. al. A comparative study of the IFC and gbXML informational infrastructures for data exchange in computational design support environments. *10th International IBPSA Conference. 2007.*

[12] ORACLE, *Oracle Berkeley DB JAVA Edition*, ORACLE, viewed on 6th May 2013, <http://www.oracle.com/technetwork/database/berkeleydb/overview/index-093405.html>

[13] Bimserver, *Open Source Building Information Modelserver*, bimserver Wiki, viewed on 10th May 2013, < https://code.google.com/p/bimserver/wiki/features>

[14] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi and S. Weerawarana. Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI. *Internet Computing, IEEE, 2002, 6(2), 86-9.*

[15] Protocol Buffers. Google's Data Interchange Format. (2011).

[16] D. Crockford. The application/json media type for javascript object notation (json). 2006.

[17] Hall R., Pauls K., McCulloch S. and Savage D., *OSGi in action – Creating modular applications in Java*, Manning Publications, April 2011.

[18] W. Mazairac and J. Beetz. Towards a Framework for a Domain Specific Open Query Language for Building Information Models. In P. Geyer, A. Borrmann, Y. Rafiq and P. de Wilde (Eds.), Proceeding of the International Workshop: Intelligent Computing in Engineering, München: Technische Universität München

[19] V. Bazjanac. 2007. Impact of the US national building information model standard (NBIMS) on building energy performance simulation. *Presentation at the Building Simulation 2007 conference, Beijing*.

[20] G.D. Kontes, G.I. Giannakis, E.B. Kosmatopoulos and D.V. Rovas. Adaptive-fine tuning of building energy management systems using co-simulation. *2012 IEEE International Conference on Control Applications (CCA).* pp. 1664-1669, IEEE.

[21] M. Weise, et. al. Implementation guide: Space boundaries for energy analysis. US General Services Administration (GSA) and Open Geospatial Consortium (OGC), 2011.

[22] C.M. Eastman, I. Panushev, R. Sacks, M. Venugopal, V. Aram and R. See. A guide for development and preparation of a national BIM exchange standard. *buildingSMART Report*.

Appendix A: Guidelines to develop a JAVA client for BIM Server 1.2

1. Running the BIMServer 1.2

In order to run the BIMServer in stand-alone mode, the main requirements are:

- a Java Virtual Machine (JVM) running in the computer.
- a .JAR file containing all the required libraries corresponding to the selected version of the BIMServer. (RCX)

Meeting these basic requirements, the .JAR file can be executed over the JVM and a pop-up window will appear containing the main interface of the BIMServer and allowing the user to start the BIMServer after the configuration of few parameters as well as launching the Web browser to configure the Server's login parameters. This BIMServer Starter screen can be observed in the Figure 19.

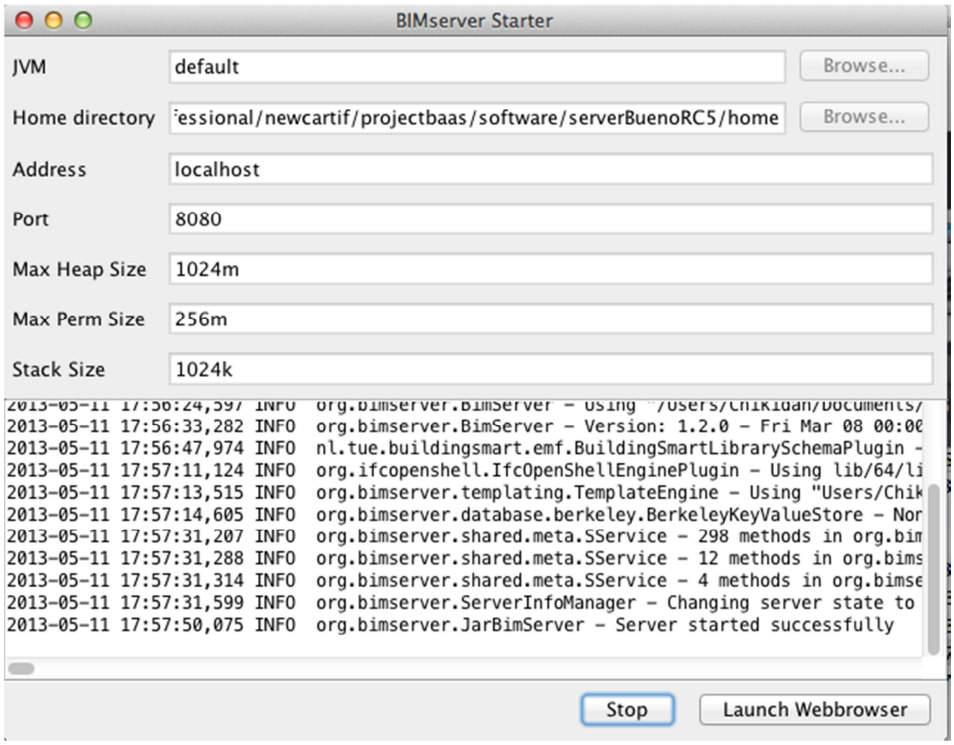

Figure 19: BIMServer 1.2 RCX Starter Interface.

Once the BIMServer is running, it can be accessed either by clicking on the "Launch Webbrowser" button available in the BIMServer Starter interface or by accessing to any Web browser and indicating the Configured URL and the corresponding PORT. As an example, in order to access a BIMServer from the same computer it is running and considering the 8080 as the default PORT, the proper URL would be: *http://localhost: 8080/*. The Figure 20 shows the login parameters interface, available in the first connection to the server.

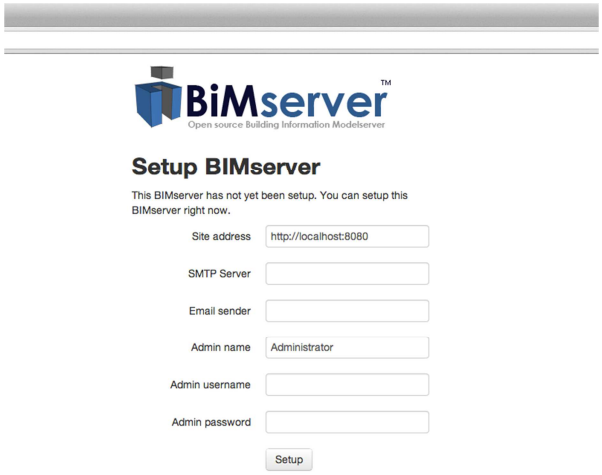

Figure 20: BIMServer 1.2 RCx login parameters configuration.

2. Running the Client:

Methods to connect with the BIMServer:

SoapBimServerClientFactory()	Method to create a SOAP connection/link with a specific BIMServer and perform a number of operations such as creating a SOAP Client or adding a determined service.
UsernamePasswordAuthenticationInfo()	Method to create the login information in the right format to be interpreted by the BIMServer.
SoapBimServerClientFactory. create()	Method to create the SOAP Client, using the SOAP factory (SOAP link) and the authentication information previously created. After the execution of this method, the Client is created and different operations can be performed between the client and the BIMServer, all of them using the SOAP protocol.
bimServerClient getServiceInterface()	Method to obtain the service interface of the BIMServer with which our Client is connected. The service interface indicates which services and operations are available in the server and allow performing different operations between the client and the server, regarding users, projects and models.
bimServerClient.disconnect()	Method to disconnect our client from the BIMServer it is connected. It will be used when all the operations are finished or in order to connect our client to a different BIMServer.

Table 7: JAVA Methods to connect the Client with the BIMServer.

Example of the JAVA code to connect a client with a BIMServer already running in local:

```
private static void connect() throws ServiceException {
//connection to the SOAP server on the BIM Server
ServicesMap servicesMap = new ServicesMap();
SoapBimServerClientFactory soapFactory = new SoapBimServerClientFactory("http://localhost:8082",servicesMap);
UsernamePasswordAuthenticationInfo authenticationInfo = new UsernamePasswordAuthenticationInfo("admin@example.org", "admin");
try {
        bimServerClient = soapFactory.create(authenticationInfo);
}catch (ChannelConnectionException e) {
        System.out.println("Connection failed...\n");
        e.printStackTrace();
}
System.out.println("Connecting...\n");
serviceInterface = bimServerClient.getServiceInterface();
bConnected = true;
}
```

Methods to manage projects:

In order to manage projects in the BIMServer, it is mandatory to use its service interface as indicated in the Connection section. The main methods available in the service interface to manage projects are listed in the table below:

getAllReadableProjects()	By using this method, the client can obtain all the projects available in the BIMserver (created in previous connections).
addProject()	This method allows creating a new project to work with, the only information required is a name in string format.
deleteProject()	Method to delete a project by name.
getProjectsByName()	This method is useful to obtain a project in a proper format adapted to BIM. As in previous cases, the only information required is the project name.

Table 8: JAVA Methods to manage projects in the BIMServer.

Example of the JAVA code to create a new project in the BIMServer:

```
SProject_list = serviceInterface.getAllReadableProjects();
for(SProject proj : SProject_list){
```

```
        System.out.println("Projects in the system: "+proj.getName());
}
        System.out.println("Write the name of the project: ");
        sc = new Scanner(System.in);
        String sName = sc.nextLine();
        if(serviceInterface.getProjectsByName(sName) == null)
                serviceInterface.addProject(sName);
else if (SProject_list.contains(serviceInterface.getProjectsByName(sName).get(0)))
                System.out.println("Project already exists");
```

Methods to manage Users:

As well as in the previous case, the User's management is carried out through the service interface. Some of the most useful methods to manage users in the BIM server are listed below:

getAllUsers()	Method that returns a list of all the current authorized users in the BIMServer. This method is very useful before creating a new user to check if the new users had previously signed up.
user.getName()	Method to filter the user information and obtain the name of the selected user.
getUserByUserName()	Method to obtain the information of a particular user by introducing the name.
addUser()	Method to add a new user to the BIMServer
deleteUser()	Method to delete an existing user from the BIMServer.
getAllAuthorizedUsersOfProject()	Method that returns a list containing all the users who are authorized to manage a particular project.
autorishedUsers.add()	Method to authorize an existing user to operate the selected project.

Table 9: JAVA Methods to manage users in the BIMServer.

Example of the JAVA code to add a user to a particular project:

```
SProject project = serviceInterface.getProjectsByName(sProjectSelected).get(0);
List<SUser> allUsers = serviceInterface.getAllUsers();
for(SUser user : allUsers){
        System.out.println("User in the system: "+user.getName());
}
```

```
List<Long> autorishedUsers = new ArrayList<Long>();
List<SUser> usersProject = serviceInterface.getAllAuthorizedUsersOfProject(project.getOid());
if(usersProject.size() > 0){
        for(SUser user : usersProject){
                System.out.println("User in the project: "+user.getName());
                autorishedUsers.add(user.getOid());
        }
}
else System.out.println("This project does not have users");
System.out.println("Write the name of the user from the system to add in the project ");
sc = new Scanner(System.in);
String sUser = sc.nextLine();
if(allUsers.contains(serviceInterface.getUserByUserName(sUser))                    &&
!usersProject.contains(serviceInterface.getUserByUserName(sUser))){
autorishedUsers.add(serviceInterface.getUserByUserName(sUser).getOid());
        project.setHasAuthorizedUsers(autorishedUsers);
}
else System.out.println("The user in not in the system or it is already as project user");

}
}
```

Methods to manage IFC models:

Finally, the model's management methods will allow us to upload different revisions of an IFC model to a particular project created in the BIMServer, as well as perform a number of operations over them. It must be indicated that, as in the previous cases, this methods are part of the service interface. The list below shows some of the pivotal methods:

Checkin()	Method to upload an IFC model to the BIMServer, particularly to one of the projects preciously created and selected in the current session. Further information such as the Project ID or the file path is required to execute this method.
Download()	Method to download a specific revision of a model. Therefore, the project ID, the required revision and the destination path must be indicated to successfully run this method.
checkoutLastRevision()	Method with a parallel functionality with the previous one. In this case, the revision is not required since the

	last revision of the models uploaded to a particular project is the one to be downloaded.
getAllSerializers()	Method that returns all the serializers available in the BIMServer to process the model. It a pivotal method to be executed before downloading a model in order to know if the required serializer is supported by the current version of the BIMServer.
getAllDeserializers()	Method that returns all the options available in the BIMServer to deserialize a selected model. It a pivotal method to be executed before uploading a model in order to know if the required deserializer is supported by the current version of the BIMServer.
getSerializerByName()	Method to obtain the ID of the serializer selected to process the IFC model when checking in, using as a parameter its name.
getDeserializerByName()	Method that returns the ID of the deserializer selected to process the IFC model when downloading, using as a parameter its name.

Table 10: JAVA Methods to manage IFC models in the BIMServer.

Example of the JAVA code to both check in and download a particular version of an IFC model:

```
//Check in a model in a particular project:


SProject project = serviceInterface.getProjectsByName(sProjectSelected).get(0);
System.out.println("Write the absolute path of the file: ");
sc = new Scanner(System.in);
String sFile = sc.nextLine();
File ifcFile = new File(sFile);
DataHandler ifcDataHandler = new DataHandler(new FileDataSource(ifcFile));
serviceInterface.checkin(project.getOid(), sProjectSelected + " IFC Project",
serviceInterface.getDeserializerByName("IfcStepDeserializer").getOid(), ifcFile.length(),
ifcFile.getName(), ifcDataHandler, false, true);


//Download a model
for(SSerializerPluginConfiguration ser : serviceInterface.getAllSerializers(true)){
System.out.println("Serializer: "+ser.getName()+ " " + ser.getDescription());
}
serializerOid = bimServerClient.getServiceInterface().getSerializerByName("Ifc2x3").getOid();
```

```
long id = serviceInterface.download(serviceInterface.getProjectsByName(sProjectSelected).get(0).getLastRevisionId(), serializerOid, false, true);

SDownloadResult sResult = serviceInterface.getDownloadData(id);

if(sResult != null){

IOUtils.copy(sResult.getFile().getInputStream(), new FileOutputStream(new File("/Documents/downloadedmodel.ifc")));

System.out.println("IFC Downloaded: "+sResult.getFile().getInputStream().toString());

}
```

Appendix B: Examples of functionality

1. Comparison of a sample query code between BimQL and the JavaQueryEngine of the TNO BIMServer

Sample query in BimQl, selecting all the doors of a building:

```
Select ?Var1
Where  ?Var1.EntityType="IfcDoor"
```

The same query using the JavaQueryEngine:

```
package org.bimserver.jqep;
import java.io.PrintWriter;
import org.bimserver.plugins.ModelHelper;
import org.bimserver.plugins.Reporter;
import org.bimserver.emf.IfcModelInterface;
import org.bimserver.emf.IfcModelInterfaceException;
import java.util.*;
import org.bimserver.models.ifc2x4rc4;

public class Query implements QueryInterface {
   private IfcModelInterface model;
   @Override
   public void query(IfcModelInterface source, IfcModelInterface dest, Reporter reporter, ModelHelper modelHelper) throws IfcModelInterfaceException {
      reporter.info("Running doors example");
      List<IfcDoor> doors = source.getAll(IfcDoor.class);
      for (IfcDoor door : doors) {
         reporter.info("Name: "+ifcDoor.getName()+"||| GUID: " + ifcDoor.getGlobalId().getWrappedValue());
         modelHelper.copy(ifcDoor, dest);
      }
   }
}
```

In both cases, the response will be the same, as shown below for TUC building of Figure 11:

```
Get all TUC Building doors:
```

```
Name: M_Single-Flush:0813 x 2134mm:212612 ||| GUID: 26aBjFPH5FjffGmwmxF2xV
Name: M_Single-Flush:0864 x 2134mm:302761 ||| GUID: 0YCISfbPf2TfUl934Q6ciM
Name: M_Single-Flush:0864 x 2032mm:212356 ||| GUID: 26aBjFPH5FjffGmwmxF2tV
Name: M_Single-Flush:0864 x 2134mm:265927 ||| GUID: 2L4odM3DX7rOo0crbwalSh
Name: M_Single-Flush:0864 x 2032mm:212311 ||| GUID: 26aBjFPH5FjffGmwmxF2qC
Name: M_Single-Flush:0864 x 2134mm:231026 ||| GUID: 3qx2bgwgfDOx0rVR8dtkrH
Name: M_Single-Flush:0864 x 2032mm:425371 ||| GUID: 1GIZh$dl93WR8JZtlvk23$
Name: M_Single-Flush:0813 x 2134mm:212631 ||| GUID: 26aBjFPH5FjffGmwmxF2xC
Name: M_Single-Flush:0864 x 2134mm:424266 ||| GUID: 1GIZh$dl93WR8JZtlvk2Gk
```

2. Example: Query for elements not included in building storeys.

In some cases, the project site includes additional structures besides the main building. For example, consider the FJK-House[18] sample building shown in Figure 21, where a garage is also included to the site. Here, a direct query on the main building storeys will not be able to discover the construction elements (walls, windows, etc.) of the garage, since they are not contained to the main building. In order to be able to acquire these elements, a query on all the buildings/structures of the site is required, as shown below:

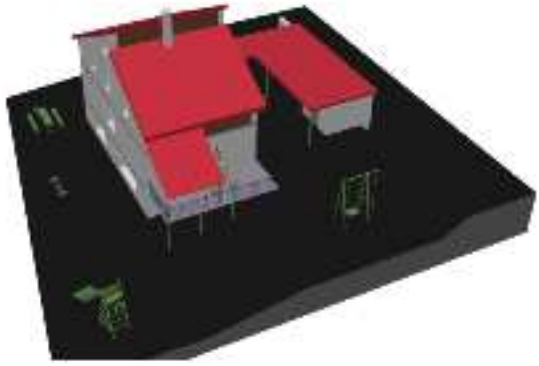

Figure 21: FJK-House example

```
// Get Building's elements which aren't contained in storeys
if(buildingList.get(b).isSetContainsElements()){
   for (IfcRelContainedInSpatialStructure rel : buildingList.get(b).getContainsElements()) {
      getElements(rel);
   }
```

[18] http://www.iai.fzk.de/www-extern/index.php?id=1135&L=1

```
}
if(buildingList.get(b).isSetIsDecomposedBy()){
   relDecomposes = null;
   relDecomposes = buildingList.get(b).getIsDecomposedBy();
   relAggregates = null;
   relAggregates = (IfcRelAggregates) relDecomposes.get(0);
   // Get products from Building Storeys
   for (IfcObjectDefinition ifcObjectDefinition2 : relAggregates.getRelatedObjects()) {
      // Get the floor
      floor = (IfcBuildingStorey) ifcObjectDefinition2;
      // Save floor number to parse it in classes
      elevation++;
      // Get the spaces of each floor
      if (ifcObjectDefinition2.isSetIsDecomposedBy()) {
         relDecomposes = null;
         relDecomposes = ifcObjectDefinition2.getIsDecomposedBy();
         relAggregates = null;
         relAggregates = (IfcRelAggregates) relDecomposes.get(0);
         for (IfcObjectDefinition ifcObjectDefinition : relAggregates.getRelatedObjects()) {
            ifcSpace = (IfcSpace) ifcObjectDefinition;
            // Call methods to add the Space elements
            getDefinitionShape((IfcProductDefinitionShape) ifcSpace.getRepresentation());
            LocalPlacementFactorial(ifcSpace, 1);
            ifcSpace = null;
         }
      }
      // Get contained elements of each floor
      for (IfcRelContainedInSpatialStructure rel : floor.getContainsElements()) {
         getElements(rel);
      }
      floor = null;
   }
   elevation = 0;
}
```

```
// else no storeys
else{
   getElements(buildingList.get(b).getContainsElements().get(0));
}
```

The response of the above query (after sorting out only the IfcWall entities) will provide the Global Unique IDs of the garage walls:

```
Garage Wall GUID: 1fzq1C9xfB7xfJf2k7oRrB
Garage Wall GUID: 1do6PA5CXDpuCoLdhvz0Ms
Garage Wall GUID: 3jGgI6df15ZBYKzN3p2ndc
Garage Wall GUID: 3PUma4QWP3GuM4TFLrDQoX
Garage Wall GUID: 0U2ipRBOj3BPpz3TxOg6vu
Garage Wall GUID: 1LmvHEFI13vwIyE2kIPt2t
Garage Wall GUID: 3GdfwgvJb2jhizfZiI8Rb1
Garage Wall GUID: 1Rf0aNMXjDbAruciA2WeNR
Garage Wall GUID: 3vI9DGiy115eAaKpDBBIF6
Garage Wall GUID: 0JQXU6F9HA9g3bRu57KyUe
Garage Wall GUID: 2Y4tzAJivFuBiqfgc7BS2A
Garage Wall GUID: 2dVtmaOMf4pR0QtAWg3waO
Garage Wall GUID: 3mzi8hSRP4b8uGilkYnb_z
Garage Wall GUID: 1fzMnZwB56mu$xywOAOufU
Garage Wall GUID: 14SAanBLP0xuA3vq$24dxI
Garage Wall GUID: 0jPzjd0BjCkhEJJng4mwp9
Garage Wall GUID: 0ofwsmS7j4YOip56sfvYLr
Garage Wall GUID: 3Vq0yreJz2euSuMho1daCY
Garage Wall GUID: 3Ny7qSRMjC5OmKL1Q0wRB3
Garage Wall GUID: 3JJvySzk59LPGKu5_WiEnr
Garage Wall GUID: 1p2w$0RQb0gwFCaG9KsMcY
Garage Wall GUID: 3pYdFEBPvA1fufVQD3ySi8
Garage Wall GUID: 1agoKIfkf4OvvKEHHbaEtK
Garage Wall GUID: 3zyOVsHF916htCgc9vxOtx
```

3. MVD usage example

Consider the example of Section 5.3, where the necessity of MVDs becomes apparent through a simple setup. The test building has been implemented in IFC4 using Constructivity Model Editor, while its properties are made available using the modified TNO BIMServer. Apart from

illustrating the value of MVDs within BaaS project, this first complete test case allowed an evaluation the compatibility of Constructivity with IFC4, as well as the ability of the adapted server to support the new schema. The present Section aims at illustrating the necessary steps followed to setup the test case along with the lessons learnt during the process.

To start, the two-room test building, shown in Figure 22 was modeled in Constructivity Model Editor. Although the editor allowed fast and accurate development of the specific model, further investigation revealed an incompatibility to IFC4 for curved surfaces. This means that all curved surfaces of a sample building have to be approximated manually using polygons – a laborious and time-consuming task for large buildings.

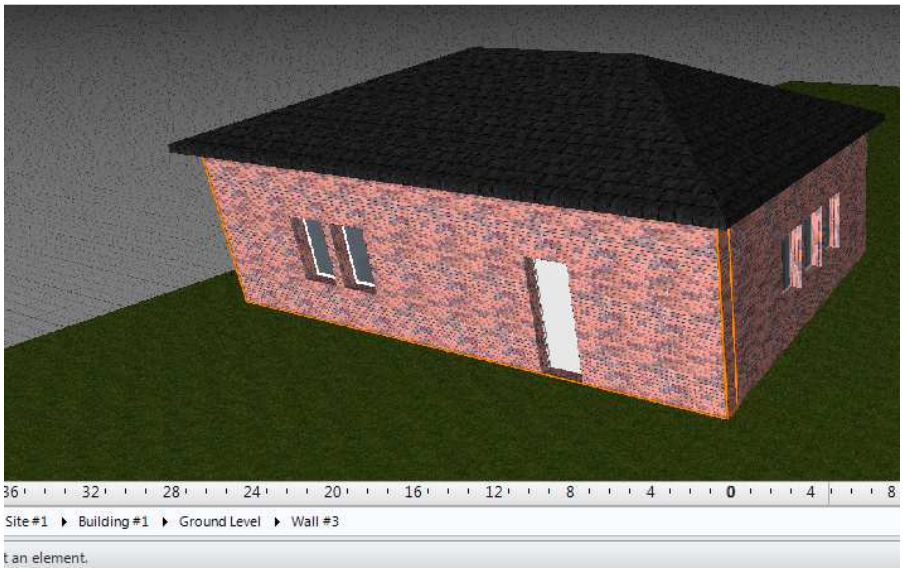

Figure 22: The two-room test building

Subsequently, the resulting IFC file was uploaded to the BIMServer after manually adapting some minor incompatibilities (mostly textual information) between the properties of the file and the server representation. Note that in this section, the commands necessary to complete the task for both IFC4-adopted versions of the BIMserver (1.2RC1 and 1.2Final) are presented – when different – for comparison. Of course, the results of the queries are identical for both server versions.

To start, in order to be able to upload the file, initially we have to login to the available server, using the following commands:

Table 11 BIMServer 1.2RC1 Connection

```
ServicesMap servicesMap = new org.bimserver.shared.meta.ServicesMap()

SoapBimServerClientFactory factory = new
SoapBimServerClientFactory("http://localhost:8080",servicesMap)

UsernamePasswordAuthenticationInfo authenticationInfo = new
UsernamePasswordAuthenticationInfo("admin@bimserver.org", "admin")

bimServerClient = factory.create(authenticationInfo, "http://localhost:8080/soap")
```

Here, an empty ServicesMap object is created (indicating that all services will be supported upon connection) and passed to the Factory method, along with the server address. Finally, a

new connection to the server is established through the Soap protocol and using the proper user credentials.

Table 12 BIMServer 1.2Final Connection

```
JsonBimServerClientFactory factory = new
JsonBimServerClientFactory("http://serverLocation:8080");

UsernamePasswordAuthenticationInfo authenticationInfo = new
UsernamePasswordAuthenticationInfo("uname@bimserver.org", "pwd");

try {
   bimServerClient = factory.create(authenticationInfo);
} catch (ChannelConnectionException e) {
   System.out.println("Connection failed...");
   e.printStackTrace();
}
```

In the new BIMServer version on the other hand, the specific declaration of the ServicesMap is not required, while connecting through JSON is preferred over using Soap.

Moving forward, the IFC file containing the building description is uploaded to the server by the user, through the following code:

Table 13 BIMServer 1.2RC1 Project Checkin

```
String projectName="TEST_BUILDING";
String fileName="test_building.ifc";
bimServerClient.getServiceInterface().addProject(projectName);
SProject_list=bimServerClient.getServiceInterface().getProjectsByName(projectName);
SProject_test= SProject_list.get(0);
long poid =SProject_test.getOid();
String comment="Test Building";
File file = new File(fileName);
long fileSize=file.length();
DataHandler ifcFile = new DataHandler(new FileDataSource(file));
SDeserializerPluginConfiguration dc =
bimServerClient.getServiceInterface().getDeserializerByName("IfcStepDeserializer");
long deserializerOid=dc.getOid();
long sCheckinResult=bimServerClient.getServiceInterface().checkin(poid, comment,
deserializerOid, fileSize, fileName, ifcFile, false, true);
```

Here, a new project is created in the server to host the new test building to be uploaded, and the name of the IFC file is provided. Once the project has been created, a data stream containing the

file contents is uploaded to the server, processed by the proper deserializer and stored to the database, thus being available to external software.

BIMServer 1.2Final Project Checkin

```
String projectName="TEST_BUILDING";
String fileName="test_building.ifc";
bimServerClient.getBimsie1ServiceInterface().addProject(projectName);
SProject_list=bimServerClient.getBimsie1ServiceInterface().getProjectsByName(projectName);
SProject_test= SProject_list.get(0);
long poid =SProject_test.getOid();
String comment="Project for testing purposes";
File file = new File(fileName);
long fileSize=file.length();
DataHandler ifcFile = new DataHandler(new FileDataSource(file));
SDeserializerPluginConfiguration dc=bimServerClient.getBimsie1ServiceInterface().getSuggestedDeserializerForExtension("ifc");
long deserializerOid=dc.getOid();
long sCheckinResult=bimServerClient.getBimsie1ServiceInterface().checkin(poid, comment, deserializerOid, fileSize, fileName, ifcFile, true);
```

For the final version of BIMServer 1.2, the calls remain the same, with two exceptions:

- All requests to Service Interface use the BIMsie interface definitions (getBimsie1ServiceInterface method);
- The decerializer for the IFC file is not declared explicitly, but the default decerializer for such file types is used.

Moving forward, a query is applied to the uploaded project, requesting the discovery of all IfcSpace objects of the building and their including IfcSensor objects (using the “IsDecomposedBy” relationship) through the following code:

```
System.out.println("Get which sensors each IfcSpace contains...");
List<IfcSpace> spaces=model.getAll(IfcSpace.class);
for (IfcSpace ifcSpace:spaces){
   System.out.println(ifcSpace.getName());
   EList<IfcRelAggregates> isDecomposedBy = ifcSpace.getIsDecomposedBy();
   if (isDecomposedBy != null && !isDecomposedBy.isEmpty()) {
      for (IfcRelAggregates dcmp : isDecomposedBy) {
         EList<IfcObjectDefinition> relatedObjects = dcmp.getRelatedObjects();
```

```
        for (IfcObjectDefinition relatedObject : relatedObjects) {
            if(relatedObject instanceof IfcSensor){
                IfcSensor ifcSensor=(IfcSensor) relatedObject;
                System.out.println("Contained Sensor ---" + " Name: " + ifcSensor.getName() + " ||| GUID: " + ifcSensor.getGlobalId().getWrappedValue() + " ||| Type: " + ifcSensor.getPredefinedType());
            }
        }
    }
  }
}
```

Here, a list of all IfcSpace objects is constructed and iterated. For each IfcSpace, the IFC objects it includes are extracted and their properties are outputted if they belong to the IfcSensor concept. This query outputs the following results:

```
Space #2
Contained Sensor --- Name: Sensor #6 ||| GUID: 0mnOgsCz15hvyjFv8pQjEN ||| Type: LIGHTSENSOR
Contained Sensor --- Name: Sensor #7 ||| GUID: 3SPMQ3OT16uB32WdNVpUnE ||| Type: TEMPERATURESENSOR
Contained Sensor --- Name: Sensor #8 ||| GUID: 3iJjf3OvXCF9U06SCCy_Q4 ||| Type: HUMIDITYSENSOR
Space #3
Contained Sensor --- Name: Sensor #3 ||| GUID: 2Z0n6P6ZnEag9NNLVEd6Zq ||| Type: LIGHTSENSOR
Contained Sensor --- Name: Sensor #4 ||| GUID: 2lWaYPui1AxxiWinuEGFk6 ||| Type: TEMPERATURESENSOR
Contained Sensor --- Name: Sensor #5 ||| GUID: 1BZhGaSQn7afwluHauLKWo ||| Type: HUMIDITYSENSOR
```

It is obvious (as stated in Section 5.3) that the contact sensors are not included to the resulting list of sensors, since they are not attached to the IfcSpace objects. So, a new query is formed, which extracts all IfcSensor objects first and then outputs the IFC object they are attached to, as shown below:

```
System.out.println("Get all sensors of the building and map to parent object...");
List<IfcSensor> sensors=model.getAll(IfcSensor.class);
for (IfcSensor ifcSensor:sensors){
   System.out.println("Name: " + ifcSensor.getName() + " ||| GUID: " + ifcSensor.getGlobalId().getWrappedValue() + " ||| Type: " + ifcSensor.getPredefinedType());
```

```
    EList<IfcRelAggregates> dcmp=ifcSensor.getDecomposes();
    for (IfcRelAggregates decomposes : dcmp) {
        IfcObjectDefinition relatedObject = decomposes.getRelatingObject();
        System.out.println("Sensor Belongs to: " + relatedObject.getName());
        System.out.println(" ");
    }
}
```

Here, a list of all IfcSensor objects is constructed and iterated and the IFC object each sensor is attached to is discovered using the “Decomposes” relationship. This leads to the complete list of sensors, as shown below:

```
Name: Sensor #4 ||| GUID: 2lWaYPui1AxxiWinuEGFk6 ||| Type: TEMPERATURESENSOR
Sensor Belongs to: Space #3
Name: Sensor #7 ||| GUID: 3SPMQ3OT16uB32WdNVpUnE ||| Type: TEMPERATURESENSOR
Sensor Belongs to: Space #2
Name: Sensor #5 ||| GUID: 1BZhGaSQn7afwluHauLKWo ||| Type: HUMIDITYSENSOR
Sensor Belongs to: Space #3
Name: Sensor #8 ||| GUID: 3iJjf3OvXCF9U06SCCy_Q4 ||| Type: HUMIDITYSENSOR
Sensor Belongs to: Space #2
Name: Sensor #3 ||| GUID: 2Z0n6P6ZnEag9NNLVEd6Zq ||| Type: LIGHTSENSOR
Sensor Belongs to: Space #3
Name: Sensor #6 ||| GUID: 0mnOgsCz15hvyjFv8pQjEN ||| Type: LIGHTSENSOR
Sensor Belongs to: Space #2
Name: Sensor #2 ||| GUID: 0DC4pKbIDF4xMQUlI_2xyj ||| Type: CONTACTSENSOR
Sensor Belongs to: Door #1
Name: Sensor #10 ||| GUID: 3nuf74XBHEb8B215CTgE5A ||| Type: CONTACTSENSOR
Sensor Belongs to: Window #6
Name: Sensor #9 ||| GUID: 3Pyba$fBj439rpMWcA7H48 ||| Type: CONTACTSENSOR
Sensor Belongs to: Door #2
Name: Sensor #1 ||| GUID: 2bJRWdcLr8c94cSsO5yq_e ||| Type: CONTACTSENSOR
Sensor Belongs to: Window #3
```