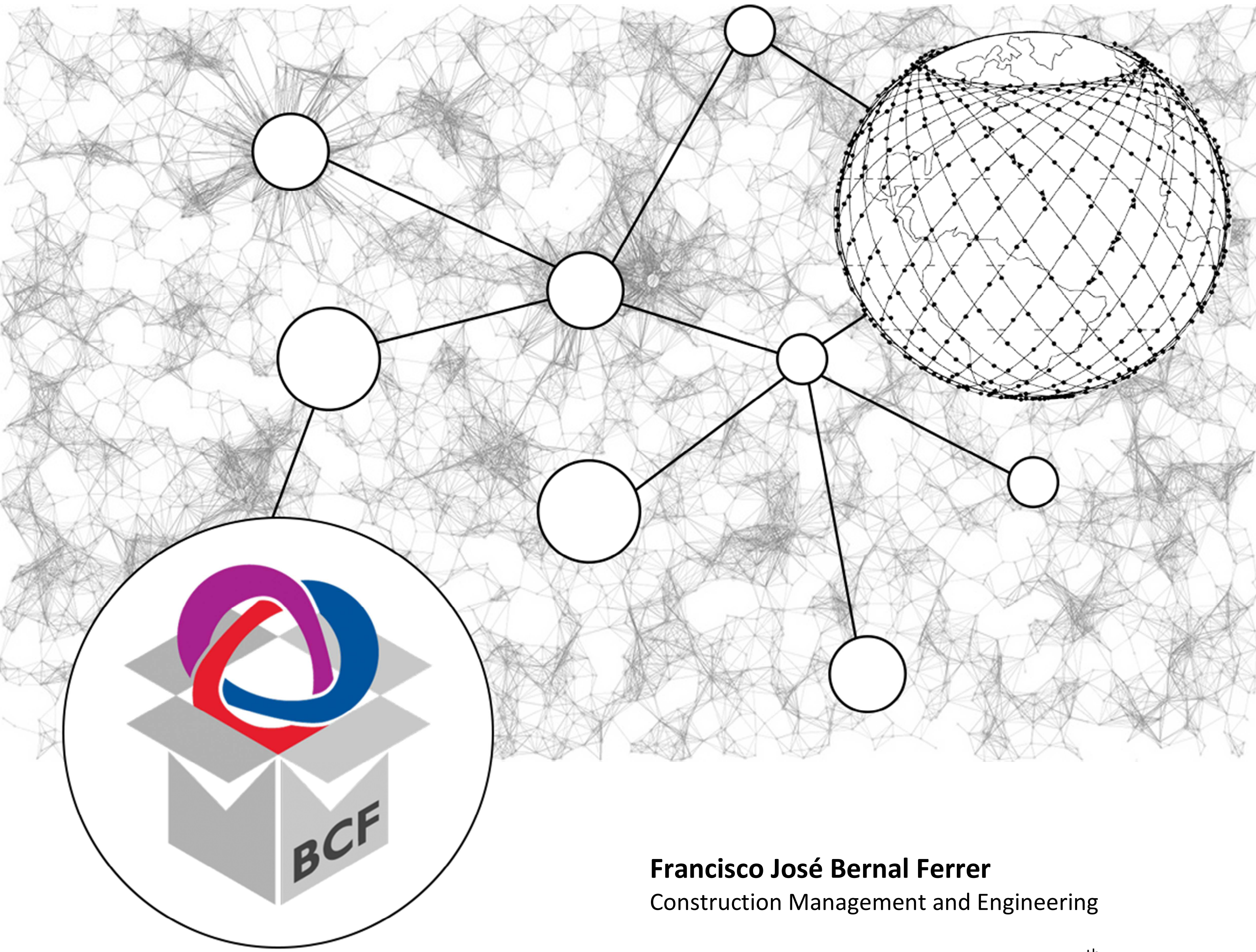
Internal design validation attestation in BIM models

A combination of Semantic Web technologies and BIM Collaboration Format



Francisco José Bernal Ferrer
Construction Management and Engineering

September 27th, 2017

Colophon

Internal design validation attestation in BIM models

A combination of Semantic Web technologies and BIM Collaboration Format

Student

Author:	Francisco José Bernal Ferrer
Student number:	0978452
Email:	f.j.bernal.ferrer@student.tue.nl
Master track:	Construction Management and Engineering
University:	Eindhoven University of Technology

Graduation Committee

Chairman (TU/e):	prof.dr.ir. B. (Bauke) de Vries
1st supervisor (TU/e):	dr. Dipl.-Ing. J. (Jakob) Beetz
2nd supervisor (TU/e):	T.F. (Thomas) Krijnen
3rd supervisor (V&L):	L. (Leon) Leenders

TU/e Technische Universiteit **Eindhoven** University of Technology

Preface

This thesis is presented as the result of my master's degree studies in Construction Management and Engineering at the Eindhoven University of Technology (TU/e). Both research and development have been carried out with the help and collaboration of academics and construction professionals in the Architecture, Engineering and Construction (AEC) industry. And in the next lines I intend to show my gratitude to all the people who have contributed in one way or another to the development of this thesis.

First of all, I want to thank all the experts that I had the chance to talk with before and during the elaboration of this thesis. Their willingness to collaborate and to share their knowledge in the AEC industry was as helpful as inspiring to me; and for that, I am really grateful.

Secondly, I would like to thank my supervisors for their instructions and counsel. To Jakob Beetz for his excellent guidance and thoughtful advice which helped me to challenge myself and always look at the big picture. Special thanks to Thomas Krijnen because his technical assistance and patient explanations have considerably improved the obtained prototype. And, of course, thanks to Leon Leenders whose broad perspective, motivation and provision of resources have extended significantly the value of the thesis I am presenting today.

Last but not least, I would like to thank my family and friends. To my family because their constant support has made everything easier; and to my friends (and enemy) because their interest and opinions served me as motivation.

To all of you, thank you.

Table of contents

Summary

The implementation of Building Information Modelling (BIM) in the Architecture, Engineering and Construction (AEC) industry represents the development and use of computer-generated n-dimensional (n-D) models to simulate the planning, design, construction and operation of a facility. Specifically for design and construction phases, the creation of BIM models helps architects, engineers and constructors to visualise what is to be built in simulated environments and to identify potential design, construction or operational problems that before were only noticeable in the execution phase (Azhar, Hein, & Sketo, 2014; Grilo & Jardim-Goncalves, 2010; Xu, Ma, & Ding, 2014). As a result, the creation of BIM models involves a collaborative environment where the exchange of information is carried out repeatedly amongst engineering departments in order to meet the requirements that have been previously defined for a given project. In addition, collaboration becomes more complicated when the file format that each engineering department uses is not the same because of different software tools.

To overcome this limitation, the development of the BIM Collaboration Format (BCF) and, later on, the RESTful web service "bcfAPI" enable both informal communications and the exchange of BCF data between BIM software tools in BIM workflows (BuildingSMART, 2017). Regarding BCF, research on its implementation has shown that besides model checking purposes (clash detection) or coordination issues (Asier Mediavilla, José Luis Izkara, & Iñaki Prieto, 2015; Jakob Beetz, Berlo, Laat, & Bonsma, 2011), the exchange of BCF data across different software applications in a BIM workflow could be extended to other process-related activities such as task, requirement, cost and risk management (Treldal, Parsianfar, & Karlshøj, 2016). However, this extension would require interoperability of the IFC model not only with the BCF data but with other data sets.

Currently, linking the BIM model to external data sets through the implementation of Semantic Web technologies is gaining scientific interest within the AEC industry (Curry et al., 2013; Pieter Pauwels & Terkaj, 2016; Vanlande, Nicolle, & Cruz, 2008). Since the early 2000s, the semantic approach has been introduced in the building industry as an opportunity to enhance the interoperability between building object's information and the IFC model, improving the efficiency of information exchange process as well (Jakob Beetz, van Leeuwen, & de Vries, 2009; Hans A. Schevers, John Mitchell, Paul Akhurst, & Chris Linning, 2007; Pieter Pauwels & Davy Van Deursen, 2012). In such way, the combination of BCF and Semantic Web technologies could increase the interoperability between the BCF data and other domains.

This thesis studies the combination of the informal communication provided by the BCF with the Semantic Web technologies for the purpose of internal design validation attestation. To understand what the internal design validation implies, the methodology starts with the research on three actual construction projects (case studies) provided by V&L in order to study and gain insight not only into the organization and management of project requirements, but also the exchange of information processes happening during the design phase of these projects. As a result of the research at V&L, two outcomes are discussed: (1) a System Breakdown Structure (SBS)/Requirements Breakdown Structure (RBS) matrix and (2) a process map of the exchange of information workflow.

The SBS/RBS matrix gathers the most common building objects and requirements that have been identified within the three different case studies. By inspecting project documentation (mainly calculations and advisory reports) it was possible to compare which similarities and differences existed between the same building elements and how they were calculated and checked against requirements. Moreover, the feedback coming from the personnel in V&L was really valuable in order to understand which requirements were relevant for the matrix as well as to which building elements they were applicable to. On the other hand, a better understanding of how parties involved in the design process collaborate and share information has been obtained from the elaboration of a process map for one of the case studies. In fact, it was possible to identify the actors and information involved in the internal validation process, and what is more important, the flaws of the current performance as well as the functionalities that the practical approach should maintain.

The practical approach develops a prototype that on the one hand combines BCF with the Semantic Web technologies, and on the other hand, is able to provide and/or improve the identified functionalities for the purpose of internal validation attestation. To that end, the current BCF specification, which is defined in an XSD schema, is represented as an ontology that the prototype uses as a template when modelling BCF data as RDF triples. These triples containing BCF data are later queried through the prototype by predefining SPARQL queries that retrieve only the necessary information depending on which functionality it is considered. The prototype is later validated in terms of correctness and completeness of the generated data and data visualization. Moreover, the identified prototype's limitations show that there is room for improvement concerning practical purposes such as the creation of multiple BCF issues per building object, the creation of multiple BCF at once and the attachment of visual information like viewpoints and/or snapshots.The findings of the development are discussed as well so benefits like the interoperability of the BCF data and its connection to the BIM model is pointed out, and solutions for the mentioned limitations are described.

Finally, conclusions present the prototype as the tangible proof of how the informal communication provided by BCF can be combined with the Semantic Web technologies. Moreover, the developed workflow examples capture the interaction of engineers with the prototype and show how it could be integrated into the design process. The combination is possible by modelling BCF information as RDF triples based on a predefined ontology that serves as a template for the prototype when modelling BCF data. The ontology's structure, although it is the equivalent of the BCF specification (XSD schema), could be simplified by omitting the hierarchical relationships between the markup and the sub-elements with no multiple instances per markup. Moreover, the BCF data is stored as triples in an RDF repository, to be later retrieved using SPARQL queries through the prototype.

Concerning the management of requirements, BCF is capable of dealing with internal requirements that have to be with design changes in the structural BIM model. The most common requirements that apply to Building Structure projects have been gathered in a matrix as a result of the research and interviews performed in V&L. On the other hand, the implementation of SE management tools in the Building Structures projects in comparison to the Civil projects differ mainly in that SE is demanded as a process in the Civil projects and for that reason, its implementation is more developed than in Building Structures projects. In fact, the requirements that are handled are also different, being the most common in the Building Structures projects the ones gathered in the SBS/RBS matrix.

Abstract

This thesis studies the combination of the informal communication provided by the BCF with the Semantic Web technologies for the purpose of internal design validation attestation. In order to understand what requirements and information an internal design validation imply, the methodology studies the exchange of information processes and requirements management and organization based on three different actual construction projects provided by the structural engineering company Verhoeven en Leenders. The research carried out in this company as well as the interviews with the personnel helped to set the functionalities and the different use cases that the practical approach aims to provide for the purpose of internal validation attestation. In addition, the research also helped to identify and gather together an initial compilation of the most common building elements and requirements in Building Structures projects as well as their applicability relationships. As a result of the thesis, the prototype successfully combines BCF and Semantic Web technologies by modelling BCF data as RDF triples that are stored in an RDF repository to be retrieved using SPARQL queries. When modelling BCF data as RDF, it is only necessary to maintain the existing hierarchical relationship between the markup and other sub-elements in the XSD schema for the elements without multiple instances per markup.

On the other hand, concerning management of requirements, BCF is capable of dealing with internal requirements that have to be with design changes in the structural BIM model. The most common requirements that apply to Building Structure projects have been gathered in a matrix as a result of the research and interviews performed in V&L. The implementation of SE management tools in the Building Structures projects in comparison to the Civil projects differ mainly in that SE is demanded as a process in the Civil projects and for that reason, its implementation is more developed than in Building Structures projects. In fact, the requirements that are handled are also different, being the most common in the Building Structures projects the ones gathered in the SBS/RBS matrix.

List of Abbreviations/Glossary

BIM: Building Information Modeling

MEP: Mechanical, Electrical and Plumbing

HVAC: Heating Ventilation Air Conditioning

AEC: Architecture, Engineering and Construction

BCF: BIM Collaboration Format

XML: Extensible Markup Language

IDM: Information Delivery Manual

SMC: Solibri Model Checker

OTL: Object Type Library

SE: Systems Engineering

CAD: Computer Aided Design

ICT: Information and Communication Technology

POP: Product, Organization and Process

XC: Extreme Collaboration

VDC: Virtual Design and Construction

MVD: Model View Definitions

OWL: Web Ontology Language

RDF: Resource Description Framework

RDFS: Resource Description Framework Schema

SPARQL: SPARQL Protocol And RDF Query Language

TTL: Turtle

URI: Unique Resource Identifier

List of figures

List of tables

1. Introduction

BIM represents the development and use of computer-generated n-dimensional (n-D) models to simulate the planning, design, construction and operation of a facility. It helps architects, engineers and constructors to visualise what is to be built in simulated environments and to identify potential design, construction or operational problems (Azhar et al., 2014).

Collaborative implementations of BIM in building projects work on a Shared Data Model (Figure 1) where architects, contractors, MEP engineers and other actors work together with the BIM manager for the elaboration of an IFC model that contains all the necessary building information.

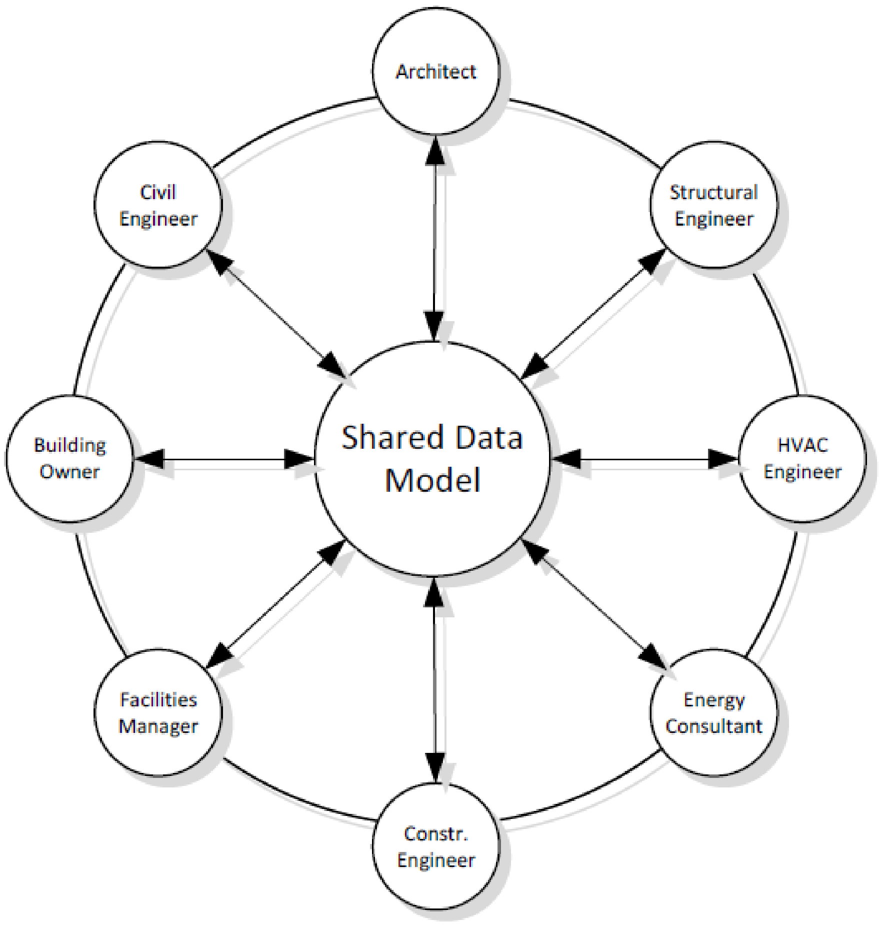


Figure 1. Process-based models (Source: https://www.slideshare.net/berlotti/centraal-bim-model, TNO, Leon van Berlo, 2015)

This collaborative implementation has influence in the life cycle of a building project, specifically, in the pre-visualization of design problems during the "Design phase" that before were usually noticeable in the "Execution phase". As a result, costs of design changes are reduced according to the MacLeamy curve of BIM intentions (Figure 2).

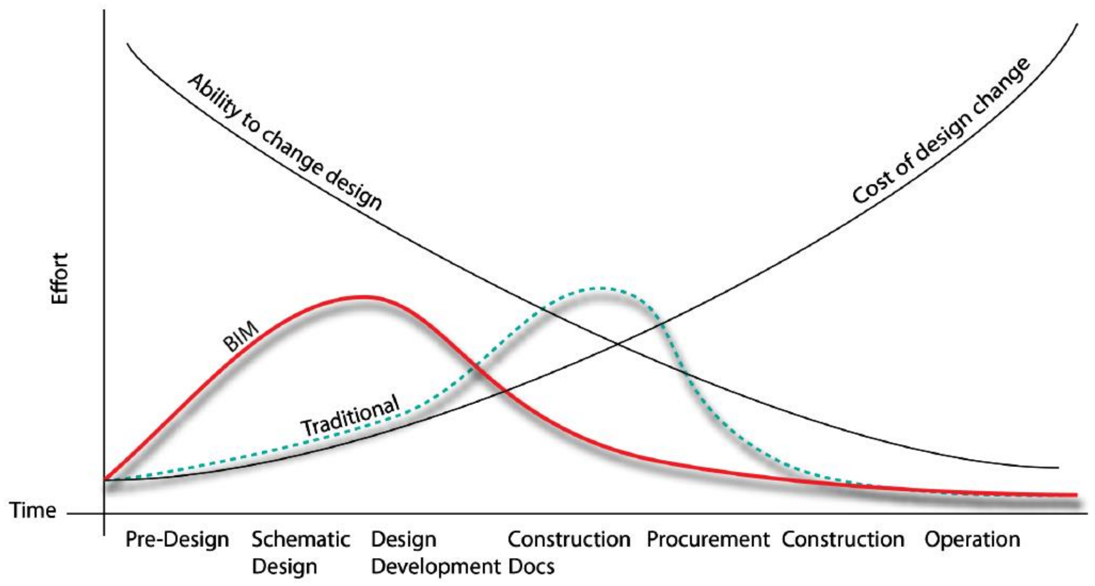


Figure 2. MacLeamy curve of BIM intentions (Source: book *"BIM and Construction Management: Proven Tools, Methods, and Workflows"* by Brad Hardin, Dave McCool, 2015)

The truth is that it is far more effective to coordinate these building systems using a visual approach with a 3D model, so that the location and relationships of all the components (architecture, structure, HVAC, electrical, plumbing, etc.) and their potential conflicts can be resolved while still in the project's planning phases (Grilo & Jardim-Goncalves, 2010; Xu et al., 2014).

On the other hand, as a parallel consequence, the design phase of any building project involves a collaborative environment where the exchange of information is carried out repeatedly amongst engineering departments in order to meet the requirements that have been previously defined for a given project. In addition, collaboration becomes more complicated when the file format that each engineering department uses is not the same because of different software tools.

Therefore, when project participants iteratively exchange model based design data, they have to evaluate the data within the framework of a programmed procedure in which a receiver confirms whether the building information model is satisfied both syntactically and semantically and whether the updated building design conforms to the agreed upon specifications and requirements. Otherwise, exchanges frequently do not realise intended geometric transformations, project requirements, and required syntactic and semantic conditions in building model data, exacerbating the problem of model integrity and resulting in expensive changes during the construction and operation phases (Y.-C. Lee, Eastman, & Lee, 2015).

The development of the BIM Collaboration Format (BCF) includes both, an XML file format as well as a RESTful web service for the purpose of communicating BIM-related tasks. The file format enables informal communications between BIM software tools while the RESTful web service "bcfAPI" enables software applications to exchange BCF data seamlessly in BIM workflows (BuildingSMART, 2017). Basically, a message that complies to a XML-based schema is encoded containing BIM-topics (e.g. issues, proposals, change requests ...) and it is assigned to building elements in order to exchange written comments and screenshots for a better collaboration between parties. By using the IFC *Global ID*, which is the attribute that identifies and differentiates every element in the IFC model, issues are assigned to building elements.

To adequately manage issues within a BCF environment, it is important to set up an effective set of values for the fields that describe the topic/issue, tailored to the processes of the project partners (L. van Berlo & Krijnen, 2014). Many forms of communication amongst different softwares around the design process can be then unified by implementing BCF as the common language between engineering departments.

On the other hand, Systems Engineering (SE) management tools such as Relatics are being implemented in civil works for the management of project information, providing project managers with a defined framework for the control of data. Relatics is essentially a web-based platform used by managers in projects to customise a Systems Engineering and management of requirements application completely to their needs (Relatics, 2017). With it, project managers are able to organise information in a coherent network of requirements together with other project information (stakeholders, design components…).

On the contrary to BCF, Relatics avoids the file exchange of information with a web-based centralized relational database, which provides a more efficient and synchronised management of data, preventing the possible exchange and storage of outdated information and/or modifications. In fact, the “bcfAPI” was born to overcome the manual exchange of files by email attachments and automate it via a standardized RESTful API.

Systems engineering is an interdisciplinary approach and means to enable the realisation of successful systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, and then proceeding with design synthesis and system validation while considering the complete problem (INCOSE, 2017). SE differs from traditional disciplines in that (1) it is focused on the system as a whole; (2) it is concerned with customer needs and operational environment; (3) it leads system conceptual design; and (4) it bridges traditional engineering disciplines and gaps between specialties (Alexander Kossiakoff, William N. Sweet, Samuel J. Seymour, & Steven M. Biemer, 2011).

Concerning management as a whole, it is also relevant to count on a less manual validation framework since for effective collaboration a process, checking these requirements in an automated and unambiguous way is of crucial importance (Krijnen & van Berlo, 2016; Y.-C. Lee et al., 2015). A manual validation framework involves a repetitive process carried out by a manager whose risk of missing or committing an error increases proportionally with the number of times that he has to validate a task on the same day. For that reason, since nowadays we can count on machines or rule-checking engines that are able to read requirements, the next step could be to integrate them as a new part of BCF and extending this technology not only to issue but also requirements management.

1.1 Problem definition/objective of the thesis

Research was carried out at the structural engineering company Verhoeven en Leenders (V&L) where both the Building Structures and Civil Division were analysed concerning management of requirements and exchange of information during project design processes.

It was observed that contrary to the Civil Division, where management of requirements is carried out by using a systems engineering management tool (Relatics) due to the strict framework demanded by some clients, the Building Structures Division at Verhoeven en Leenders does not rely on a well-structured process for the implementation of systems engineering. The current situation at the company is that engineers only give face-to-face feedback on the things that went wrong and/or attach comments or references to specific checks (e.g. 2D drawings) indicating that something failed without a fixed action protocol.

The behaviour from engineers in V&L has been also noticed by several authors (L. van Berlo, Derks, Pennavaire, & Bos, 2015; Gu & London, 2010; W. Shen et al., 2010) despite the fact that Building Information Modeling (BIM) technologies are becoming more and more common in the AEC field. Eventually, this reluctance to BIM adoption could be counter-productive in a collaborative environment that requires data exchanges and communication among different domains and applications. In addition, as the information processing becomes more and more automated, standardized and qualified data is necessary for efficient working processes (Zhang & Beetz, 2014).

Regarding BCF, research on its implementation has shown that besides model checking purposes (such as clash detection) or coordination issues (Asier Mediavilla et al., 2015; Jakob Beetz et al., 2011), the exchange of BCF data across different software applications in a BIM workflow could be extended to other process-related activities such as task, requirements, cost and risk management (Treldal et al., 2016). However, this extension requires interoperability of the IFC model not only with the BCF data but with other data sets.

Currently, using web-based BCF management tools (bcfAPI) that synchronize BCF issues without having to export BCF ZIP files ("BIMcollab," 2017; L. van Berlo & Krijnen, 2014) improves interoperability between the IFC model and the BCF data in the field of process information exchange. On the other hand, linking the BIM model to external data sets through the implementation of semantic web technologies is gaining scientific interest within the AEC industry (Curry et al., 2013; Pieter Pauwels & Terkaj, 2016; Vanlande et al., 2008).

Since the early 2000s, the semantic approach has been introduced in the building industry as an opportunity to enhance the interoperability between building object's information and the IFC model, improving the efficiency of information exchange process as well (Jakob Beetz et al., 2009; Hans A. Schevers et al., 2007; Pieter Pauwels & Davy Van Deursen, 2012). In such way, the combination of BCF and Semantic Web technologies could increase the interoperability between the BCF data and other domains.

The thesis' scope is part of a project focused on researching the application of linked data technologies for the purpose of internal and external design validation (Figure 3). For that purpose, the objective is divided into (1) supplementation of an IfcOpenShell based desktop viewer with the possibility of visualising objects' requirements that have been previously linked to the elements through semantics and attaching external validation documentation (red tasks in Figure 3); and (2) allowing the communication between engineers through BCF (in combination with Semantic Web technologies) for the purpose of internal design validation attestation (green tasks in Figure 3).

The first objective is addressed in Miryana Stancheva's thesis (Stancheva, 2017). On the other hand, it is the second objective that is addressed within the scope of this thesis and, specifically, its goal is to provide the IfcOpenShell based desktop viewer with an option to create BCF issues so engineers can communicate when there are problems with building elements meeting the project requirements in the BIM model. The creation of BCFs will be done within an RDF environment that will prevent the creation of BCF zip files so that all the issues created will be stored in a SPARQL endpoint. Moreover, this stored information should be also retrievable directly from the desktop viewer providing the user with a direct connection to information (like BCF object status) for all the elements in the model.

Backwards and external compatibility with other BCF management tools (such as KUBUS BCF Manager) will be also included within the scope of this thesis. In this way, it will be possible not only to generate BCF issues within an RDF environment but also to import BCF information (zip files) created/modified with other BCF management tools.

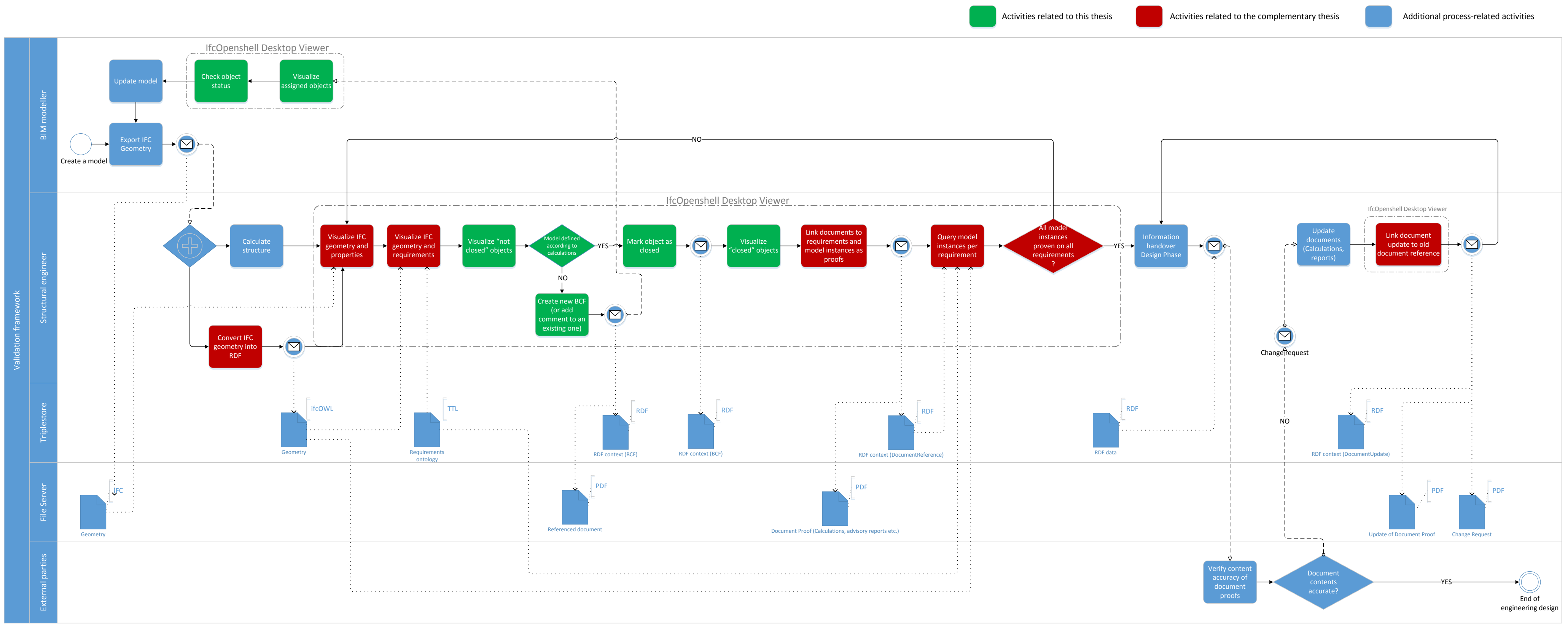

Activities related to this thesis
Activities related to the complementary thesis
Additional process-related activities
Validation framework
BIM modeller
Structural engineer
Triplestore
File Server
External parties
IfcOpenshell Desktop Viewer
Update model
Check object status
Visualize assigned objects
Create a model
Export IFC Geometry
NO
IfcOpenshell Desktop Viewer
Calculate structure
Visualize IFC geometry and properties
Visualize IFC geometry and requirements
Visualize "not closed" objects
Model defined according to calculations
YES
NO
Create new BCF (or add comment to an existing one)
Mark object as closed
Visualize "closed" objects
Link documents to requirements and model instances as proofs
Query model instances per requirement
All model instances proven on all requirements ?
YES
Information handover Design Phase
Update documents (Calculations, reports)
IfcOpenshell Desktop Viewer
Link document update to old document reference
Change request
Convert IFC geometry into RDF
ifcOWL
Geometry
TTL
Requirements ontology
RDF
RDF context (BCF)
RDF
RDF context (BCF)
RDF
RDF context (DocumentReference)
RDF
RDF data
NO
RDF
RDF context (DocumentUpdate)
IFC
Geometry
PDF
Referenced document
PDF
Document Proof (Calculations, advisory reports etc.)
PDF
Update of Document Proof
PDF
Change Request
Verify content accuracy of document proofs
Document contents accurate?
YES
End of engineering design

Figure 3. Thesis' scope[1]

[1] An A3 version of the schema can be found in the Appendix, 7.9 Appendix IX: Thesis' scope

1.2 Research question(s)

The main research questions would be:

1. **Can Semantic Web technologies be integrated into a building structures design process for internal validation attestation using BCF?**
2. **Is BCF a capable tool for the management of requirements?**

The already existent experience of the personnel within the Civil Division of the company with systems engineering management tools such as Relatics make this engine an opportunity for managing the requirements. So a sub-question could be:

3. **What differences exist between the implementation of systems engineering management tools (Relatics) in building structure projects and civil projects?**

Moreover, requirements coming from stakeholders, clients or co-workers within the company can take different shapes, for instance: you have to add/remove these elements into/from the model or you have to modify the material class of this element. So sub-questions to be answered could be:

4. **What requirements apply specifically to projects in the Building Structures sector and which requirements have a general scope? And how will these requirements look like in a structural engineering context?**

Regarding the communication framework through BCF within an RDF environment and storage (SPARQL endpoint), further analysis could be carried out regarding the next sub-questions:

5. **How can BCF information be structured and stored using Semantic Web technologies (RDF)? Can this information be accessed and used for internal validation attestation of requirements (SPARQL endpoint)?**

1.3 Methodological justification

The chosen methodology for the research approach of this thesis involves three main stages: research, development and validation. The combination of these stages complements each other and consequently the elaboration of the thesis.

First of all, before starting any development project, a research is needed to gain basic insight into the status and level of development of current tools related to the topic. What is more, developing a validation framework requires an understanding of every piece of information exchanged in it. Process models are used in IFC specification development projects as the means to discover and capture the information content of a business process and how that information is to be exchanged between participants in the process (IDM Technical Team, 2007). Therefore, a process model of the exchange of information processes that occur during the whole design phase is needed to have a global view of the actors, type of information that is exchanged between departments and responsibilities; and, specifically, to be able to identify the necessary actors and information for internal design validation attestation.

Interviews with the personnel at Verhoeven en Leenders are performed as well since the best source of information for assessing this process model and the exchanged information is the one coming from the actors that are involved in it. Moreover, since the Civil Division has experience with systems engineering management tools (Relatics) regarding the project requirements, it is reasonable that feedback coming from this division of the company is as necessary as valuable for the research.

Secondly, based on the findings obtained from the initial research and interviews, a tool is developed to address the needs that were identified while studying the implementation of Semantic Web technologies as well. The features that this application offers to the user are designed according to indications and recommendations coming from employees and supervisors apart from the insight obtained from the first stage. Once the tool is ready, its value is proved on the validation stage.

The third and last stage consists of use cases generated to show not only the value of the features designed in the developed tool but to find possible issues or improvable characteristics. The point of this stage is to show that the application works, deals with current BCF implementations and, what is more important, brings additional possibilities to the design process in the building environment.

1.4 Research design

The research model that has been followed in the elaboration of this thesis is shown in Figure 4. Basically, it consists of three stages (research, development and validation) and the final documentation stage where the conclusions obtained from the each stage are written and discussed in the graduation report.

During the first stage called research, a study of the existing validation framework at both Building Structures and Civil Division is carried out by reading the available company documentation (like BIM or SE protocols) and interviewing several employees. Moreover, requirements management, its organisation and the information exchange workflow are studied within several actual company projects in order to identify general and project specific requirements and elaborate a System Breakdown Structure of building elements and project requirements. In addition, a process map of the exchange of information workflow during a project is carried out as well.

Scientific papers regarding BCF, Semantic Web technologies (RDF and SPARQL) and management of requirements and project information are investigated aiming to gain insight into the existing implementations, tools and experiences. In addition, how requirements are managed with systems engineering management tools (Relatics) is also analysed directly at the company by inspecting actual civil projects.

The second stage involves a practical approach that aims to develop a management tool that is able to combine BCF and Semantic web technologies for the purpose of internal design validation and attestation. The developed tool allows users to create BCF issues within an RDF environment that are uploaded and retrieved from a SPARQL endpoint. This tool is based on the IfcOpenShell desktop viewer and the final code, which is written in Python, reuses parts from already graduated students (Riet, 2016; Ven, 2017) that already work with the IfcOpenShell viewer.

Conclusions from the second stage connect with the third stage where the tool is validated through several use cases in order to prove whether BCF is an appropriate tool for requirements management, and if benefits are achieved through the combination of BCF and semantic web technologies. These use cases are identified and designed according to what existing BCF management tools can and cannot do nowadays as well as the indications and advice coming from employees and the supervisor at the company. In this way, the value of the developed tool can be proved not only by substituting BCF Zip files with RDF contexts but also it will contain functionalities based on actual user demands. An analysis looking for possible issues or improvable features of the developed tool is carried out as well so that recommendations about them and for future research or tool extensions are proposed.

Finally, all three stages and their relevant conclusions are documented with the elaboration of the graduation report.

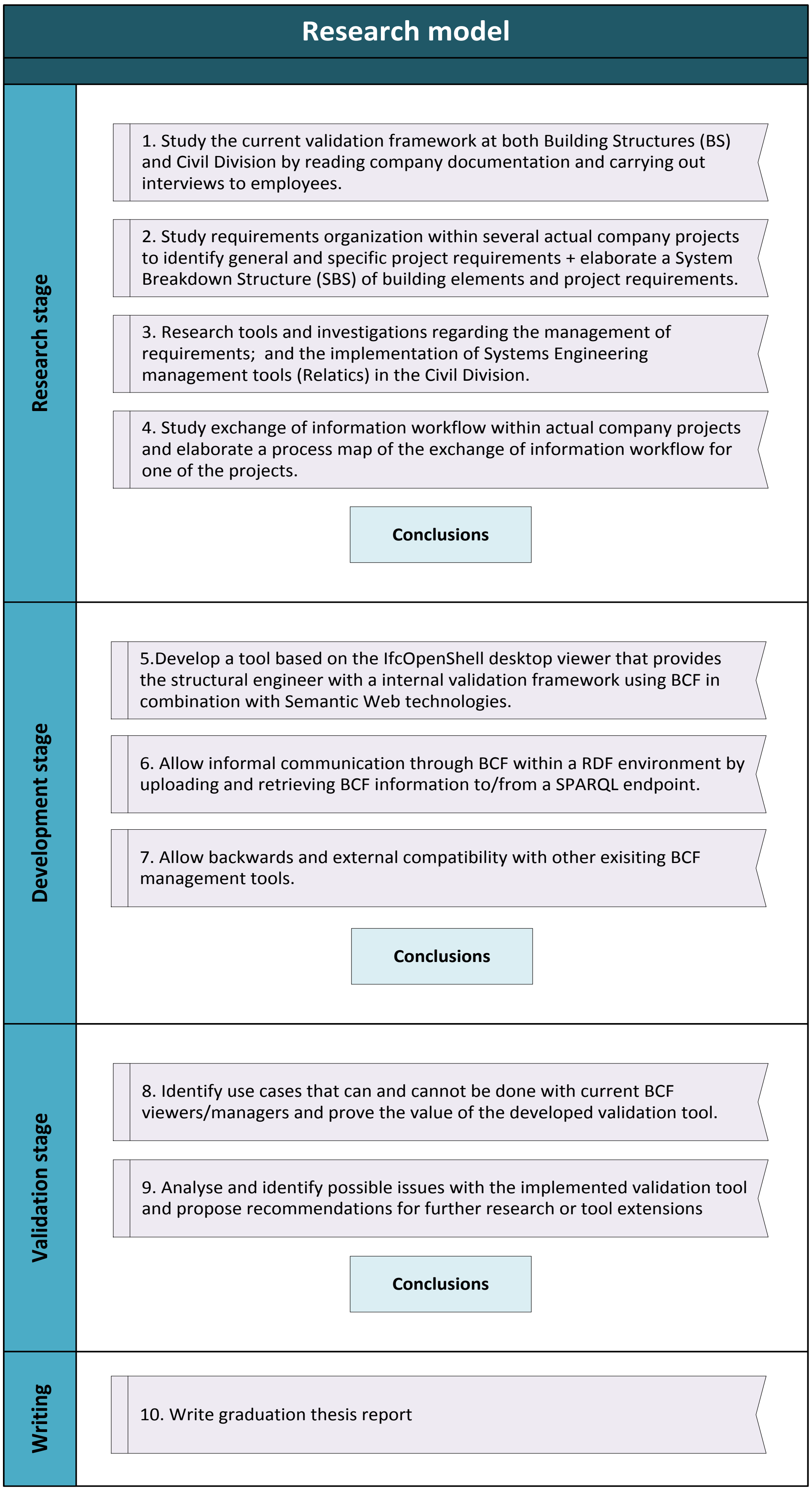

Research model
Research stage
1. Study the current validation framework at both Building Structures (BS) and Civil Division by reading company documentation and carrying out interviews to employees.
2. Study requirements organization within several actual company projects to identify general and specific project requirements + elaborate a System Breakdown Structure (SBS) of building elements and project requirements.
3. Research tools and investigations regarding the management of requirements; and the implementation of Systems Engineering management tools (Relatics) in the Civil Division.
4. Study exchange of information workflow within actual company projects and elaborate a process map of the exchange of information workflow for one of the projects.
Conclusions
Development stage
5.Develop a tool based on the IfcOpenShell desktop viewer that provides the structural engineer with a internal validation framework using BCF in combination with Semantic Web technologies.
6. Allow informal communication through BCF within a RDF environment by uploading and retrieving BCF information to/from a SPARQL endpoint.
7. Allow backwards and external compatibility with other exisiting BCF management tools.
Conclusions
Validation stage
8. Identify use cases that can and cannot be done with current BCF viewers/managers and prove the value of the developed validation tool.
9. Analyse and identify possible issues with the implemented validation tool and propose recommendations for further research or tool extensions
Conclusions
Writing
10. Write graduation thesis report

Figure 4. Research model

1.5 The practical/social and/or theoretical/scientific importance of the thesis

The importance and usefulness of this thesis can be justified within a practical and scientific environment.

Concerning the scientific importance, exchange of BCF issues by using regular email attachments hinders the efficiency and performance of engineering departments concerning the validation of BIM models. In addition, it becomes limited when thousands of issues generated from a single construction project need to be managed. Even with web-based tools like BIM Collab ("BIMcollab," 2017), which allows the management of BCF data without the need to exchange BCF zip files, it still does not use RDF as data exchange. For that reason, a combination of BCF with semantic web technologies (RDF) provides the AEC industry with an approach that is not only able to substitute the creation and exchange of zip files, but also it makes the BCF information machine-processable and retrievable. What is more, it creates opportunities to combine it with data sets from other domains that use RDF as a main data format in order to create a network of interconnected knowledge.

Regarding the practical importance, because of the upcoming Dutch law "Wet kwaliteitsborging voor het bouwen" construction companies will have to document and prove every aspect or decision of the construction design. For that reason, counting on an internal validation framework that is able to keep track of every structural change and decision made in the structural model represents not only an advance into the automation and digitalization of the construction design process but also a great advantage for the company who counts on it, in comparison with competitors who don't, regarding the tendering of new projects.

1.6 Reading guide (the organisation of the thesis)

This section will shortly introduce the contents of every chapter of the thesis, which consists of a total of seven chapters:

Chapter 1 called "Introduction" will present the reader the main topics that the thesis is related to, starting with the problem definition and objective of the thesis, continuing with the research questions, methodological justification, research design and followed by the practical/scientific importance of the thesis.

Chapter 2 called "Literature review" introduces the reader to the current status and level of development of the topics of the thesis by explaining and highlighting the most relevant findings that other researchers have investigated and developed about them.

Chapter 3 called "Methodology" brings in the findings and results obtained after carrying out a research of the documentation and actual projects at the structural engineering company Verhoeven en Leenders. The research concerns the identification and management of project requirements; and the capture of the information exchange during the design process of actual construction projects.

Chapter 4 called “Tool development” presents the practical approach of the thesis. Through different sections, the developed tool and its functionalities are explained, analysed, validated and discussed.

Chapter 5 contains the conclusions that will sum up all the relevant findings of this thesis. To that end, the initial research questions presented in chapter one are answered based on those findings and recommendations for further research are discussed as well.

Chapter 6 lists the references that have been consulted for the elaboration of this thesis.

Chapter 7 gathers the different documentation that has been consulted or developed during the elaboration of thesis and, on the whole, has an important influence in the thesis.

2. Literature review

2.1 Introduction

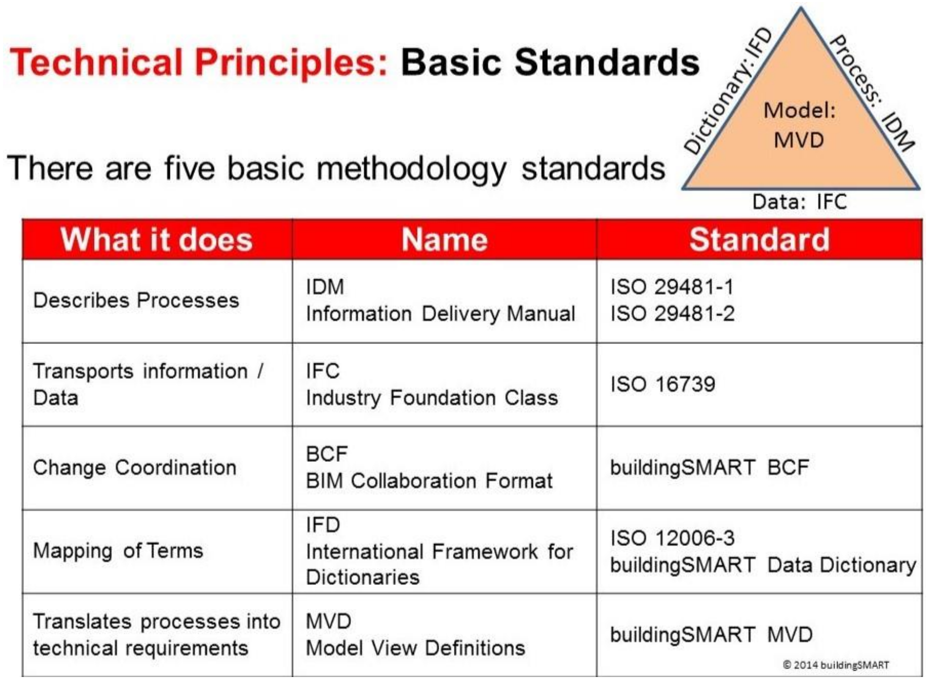
This chapter contains the literature review that shows the current situation and/or status of topics like BIM, IFC interoperability, BCF, requirements management, RDF and SPARQL.

A list of the articles addressed in this literature review can be found in the concept matrix in section 7.1 Appendix I: Concept matrix, where they have been classified according to their main topic.

2.2 Building Information Modeling (BIM): open standards, interoperability, implementation and adoption

Building Information Modeling (BIM) is an intelligent 3D model-based process that equips architecture, engineering, and construction professionals with the insight and tools to more efficiently plan, design, construct and manage buildings and infrastructure (Autodesk, 2017). It is expected to increase inter-organizational and disciplinary collaboration in the construction industry and to improve the productivity and quality of the design, construction, and maintenance of buildings (Miettinen & Paavola, 2014).

Interoperability between BIM tools is provided through the Industry Foundation Class (IFC) open standard. The IFC is a language for transferring information between BIM applications while maintaining the meaning of different pieces of information in the transfer. It represents architectural and construction-related CAD graphic data as 3D real-world objects; and it is one of the five basic open standards (Figure 5) which exist to perform different functions in the delivery and support assets in the built environment (BuildingSMART, 2017).


Dictionary: IFD
Process: IDM
Model: MVD
Data: IFC
© 2014 buildingSMART

Technical Principles: Basic Standards

There are five basic methodology standards

What it does	Name	Standard
Describes Processes	IDM Information Delivery Manual	ISO 29481-1 ISO 29481-2
Transports information / Data	IFC Industry Foundation Class	ISO 16739
Change Coordination	BCF BIM Collaboration Format	buildingSMART BCF
Mapping of Terms	IFD International Framework for Dictionaries	ISO 12006-3 buildingSMART Data Dictionary
Translates processes into technical requirements	MVD Model View Definitions	buildingSMART MVD

Figure 5. BuildingSMART basic standards (Source: http://buildingsmart.org/standards/technical-vision/open-standards-101/)

Other open standards included within the basic BuildingSMART standards, and which are relevant for the development of this thesis, are the Information Delivery Manual (IDM) and the BIM Collaboration Format (BCF). IDM captures business processes and provides detailed specifications of the information that a user involved in the process will have to provide at a certain point during the mentioned process. On the other hand, BCF is more oriented towards the coordination of any communication workflow in the process. It encodes messages (in an XML schema format) to enable workflow communication between different BIM software tools.

For several years now, open BIM data standards have been tested to check whether they meet the needs of the Architecture, Engineering and Construction (AEC) industry. For example, Leon van Berlo compares (L. A. H. M. van Berlo, Beetz, Bos, Hendriks, & Tongeren, 2012) the efficiency of the collaboration process in two different cases: first using a central data repository where all actors use the same software tool, and second, using open standards like IFC to exchange data because all actors used different software tools.

Amongst the conclusions, three points can be highlighted: (1) engineers should be free to choose their own software tools in order to achieve a higher performance in the execution of their engineering tasks; (2) synchronization in a central data repository of models should be done on a weekly basis instead of in real time. The idea is that IFC data of other users is used as a reference to create your own native data. Finally, (3) the amount of available information for BIM users should be that it contains only information that they need at a given time and not all of it. In fact, interviewed respondents stated that the IDM and MVD concept (BuildingSMART 2011) could solve this issue.

The features and possible implementations of IFC in the AEC industry are discussed by Thomas Froese (Thomas Froese, 2003). According to the author, the IFC’s scope includes both product (building systems and elements, their geometry, design properties…) and non-product (costs, schedules, people, organisations, resources, documents…) information. However, it is argued that most systems implement this scope only in one direction to use product information as an input to non-product applications, like input geometry into an energy simulation application (Grilo & Jardim-Goncalves, 2010); and very few implement it backwards to use non-product information back into IFC files to exchange non-product data.

In addition, it presents that current implementations only consider this open BIM standard for the exchange of IFC files between team members, which is simple and effective, but limited when it comes to managing a shared project with many users and large amounts of data; and the same applies for BCF zip files. The problem is that IFC model only offers a standardisation of the information format, but it does not create any exchange framework. Subsequently, the paper supports the formalisation of data exchange protocols to support IFC-based transactions. These transactions, as mentioned in Leon van Berlo’s paper (L. A. H. M. van Berlo et al., 2012), could be managed through the implementation of IDM.

Later on, van Berlo, Derks, Pennavaire and Bos aimed to standardize a workflow for collaborative engineering with IFC (L. van Berlo et al., 2015). Through observations and interviews to users in the AEC industry, it is concluded that IFC is the most effective technology to support a standardised collaboration process with BIM. Solibri Model Checker (SMC) and Tekla BIMsight are examples of software applications that work on IFC data and, what is more, exchange data through BCF. Nevertheless, it is also interesting that some interviewed users find that collaboration is not about standardising a data flow or process, but about connecting to other team members by knowing which data is needed during any giving time in the process; which again, corresponds to the definition of IDM's objective.

An actual implementation of open BIM standards, object type libraries (OTL), systems engineering (SE) and an Information Delivery Manual (IDM) is described in Hans Hoeber's paper (Hans Hoeber & Daan Alsem, 2016). The explained approach shows IDM as the methodology to define the required timing, frequency, content and format of information exchange, COINS as the information exchange format and VISI as the formalisation of the communication process between the actors in a construction project. In other words, IDM and VISI manage the actual information transactions between partners addressing the information requirements of actors involved over the life-cycle; and COINS manages the content of the message by providing interoperability.

Despite the fact that this approach supports and concludes that information can be managed through different open BIM standards in a project life-cycle, it holds that the standardization of the communication process is an important principle to provide a suitable communication framework able to meet the business needs; which disagrees with the opinion of interviewed users by Leon van Berlo (L. van Berlo et al., 2015) and his findings.

On the contrary to Miettinen's point about increasing collaboration in the construction industry (Miettinen & Paavola, 2014), the documented case study from Neff (Neff, Fiore-Silfvast, & Dossick, 2010) argues that BIM and digital models amplify the disciplinary representation of the building by architects, engineers and buildings, pointing out the organisational and cultural differences between project participants.

"With paper, meeting participants had some openness to infer what drawings could represent or how they might function in three-dimensional space. With digital models, this interpretative flexibility was taken away and was replaced with explicitness about what was possible for the building"(Neff, Fiore-Silfvast, & Dossick, 2010).

It holds that the cognitive distinctions of what the building is and what it will become cannot be simultaneously represented in BIM due to its explicitness and concludes that digital models do not have the interpretative flexibility to maintain negotiations across knowledge boundaries. Nevertheless, this can be argued by Leon van Berlo (L. A. H. M. van Berlo et al., 2012), who discuss that the users do not aim at creating a perfect BIM model, but focus on doing a good engineering task by using other's IFC data as a reference. In fact, according to its findings, the user never works in or from data in a central repository but only with their own database or model instance and taking responsibility for their own data.

The reluctance of some users within the AEC industry to adopt open BIM standards and IT technologies has been also mentioned in other articles (L. van Berlo et al., 2015; W. Shen et al., 2010). For instance, Gu and London (Gu & London, 2010) explained that collaboration is still primarily based on the exchange of 2D drawings although most disciplines are now working in a 3D environment due to the lack of trust on completeness and accuracy of 3D models from the users, which differs from other authors (Grilo & Jardim-Goncalves, 2010; Krijnen & van Berlo, 2016) who present BIM as the transition from 2D CAD drawings to semantically rich information models that allow visualization, understanding and construction to take place in 3D dimensions; increasing productivity and efficiency of the construction processes as a result.

Gu and London (Gu & London, 2010) add that the registration of communication and information exchange is something that is not generally captured in the BIM model and users demand. As a conclusion, they state that the BIM approach is a relevant factor for an industry where projects are characterised by multidisciplinary and multi-organizational teams. However, like it has been mentioned by other authors (L. A. H. M. van Berlo et al., 2012; L. van Berlo et al., 2015; Thomas Froese, 2003), efforts should be focused on creating a Collaborative BIM Decision Network able to assign responsibilities and activities interdependencies, conduct design reviews and validations in order to facilitate BIM adoption.

According to Grilo and Jardim-Goncalves (Grilo & Jardim-Goncalves, 2010), BIM adoption requires interoperability to support its implementation within the AEC industry. And overcoming the interoperability it is not only a matter Information and Communication Technology (ICT) problems but also addressing business processes along with new management relationships, culture, employees, values and management of contractual issues between interacting parties. In fact, governments are developing requirements documents to define and limit the scope of information to a right level of detail and project phase when exchanging building models (Krijnen & van Berlo, 2016), which can be partially solved through IDM initiative (Y.-C. Lee et al., 2015).

2.3 Information Delivery Manual (IDM)

The importance of creating a framework for information management and collaboration between parties in order to ensure that information is reusable and retrievable has been pointed out in several articles during the previous chapter. Specifically the ISO 29481-1:2010 “Building information modelling - Information delivery manual - Part 1: Methodology and format” standard developed by buildingSMART has been mentioned by several authors (L. A. H. M. van Berlo et al., 2012; Hans Hoeber & Daan Alsem, 2016; Y.-C. Lee et al., 2015) as the methodology to capture and specify processes and information flow during the lifecycle of a facility.

Due to the fragmented nature of the construction industry and because construction projects usually bring together different companies, engineering and architectural departments; it is necessary to establish an organisation to know which and when different kinds of information have to be communicated in order to increase the efficiency of the collaboration process (BuildingSMART, 2011). In fact, according to Kvan (Kvan, 2000), in order to be successful in a collaborative project, teams and activities involved in the process must be defined and their outcomes and interdependencies identified.

To that purpose, the main objective of IDM methodology is to have a well-defined communication process between the parties so that the necessary data are communicated to all parties involved in a certain process at the right time and in the right format (Asier Mediavilla et al., 2015). In other words, it indicates where the most important processes in the whole captured process are located, which the exchanged and needed information for the process realisation is, how the information should be delivered, and who the actors (using the information) involved in those processes are (Figure 6).

Its implementation brings benefits not only to BIM users but also to software solution providers. On the one hand, BIM users are given an easy to understand language description of building construction processes, information requirements and expected results. On the other hand, it gives BIM software developers a functional breakdown of the processes and the IFC capabilities that need to be supported for each functional part (Jeff Wix & Jan Karlshøj, 2010).

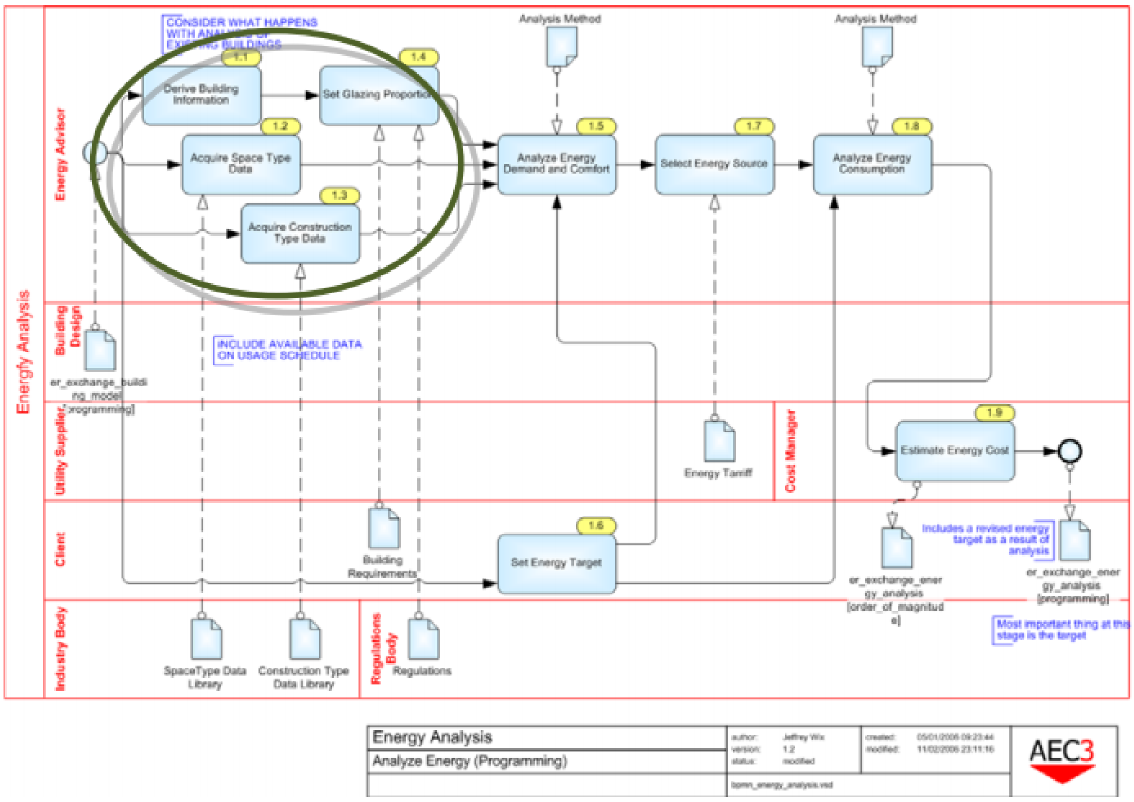

Energy Analysis
Energy Advisor
Building Design
Utility Supplier
Client
Industry Body
Regulations Body
Cost Manager
CONSIDER WHAT HAPPENS WITH ANALYSIS OF EXISTING BUILDINGS
1.1
Derive Building Information
1.4
Set Glazing Proportion
1.2
Acquire Space Type Data
1.3
Acquire Construction Type Data
Analysis Method
1.5
Analyze Energy Demand and Comfort
1.7
Select Energy Source
Analysis Method
1.8
Analyze Energy Consumption
er_exchange_building_model [programming]
INCLUDE AVAILABLE DATA ON USAGE SCHEDULE
Energy Tariff
1.9
Estimate Energy Cost
Building Requirements
1.6
Set Energy Target
Includes a revised energy target as a result of analysis
er_exchange_energy_analysis [order_of_magnitude]
er_exchange_energy_analysis [programming]
Most important thing at this stage is the target
SpaceType Data Library
Construction Type Data Library
Regulations
Energy Analysis
Analyze Energy (Programming)
author: Jeffrey Wix
version: 1.2
status: modified
created: 05/01/2006 09:23:44
modified: 11/02/2006 23:11:16
bpmn_energy_analysis.vsd
AEC3

Figure 6. BPMN diagram (Source: IDMC 004, BuildingSMART)

It is mentioned by van Berlo, Bomhof and Korpershoek (L.A.H.M. van Berlo, F. Bomhof, & G. Korpershoek, 2014) the existence of a BIM protocol generator (still in beta testing) for addressing IDM's goals and objectives. It captures the BIM working methods in a project, identifies the preferred working method from project partners and adds them to a BIM project protocol. To get this information, project participants are sent a questionnaire through which the needed information and some basic capabilities of each project partner are studied and captured.

"When partner A with software X needs information from partner B with software X, the text in the generated BIM protocol will probably state that they will exchange data with software X. When one of them uses other software but both know how to handle IFC the BIM protocol generator will plot a text that information between A and B will be exchanged with IFC. This way the whole collaboration network will become clear and it will show where problems might occur" ("BIM protocol generator," 2014).

Once generated, the first BIM protocol draft should not be considered as final. As it is explained in its description, this generator facilitates the creation of an official document, saving hours of interviews and meetings to identify working methods. However, it is recommendable to participants to meet after its creation to revise and supplement it in order to define the final collaboration framework.

Other past studies are more focused on the organisation of project objectives in a computing environment. Hitchcock (Hitchcock, 1995), for instance, presents a simple and flexible hierarchical organisation of the project requirements and their associated data that could help BIM software developers to visualise how project information should be able to flow in both forward and backwards directions between all the phases of the project lifecycle. As it is explained, such organisation of project objectives shows the relation between design intent elements and its evolution over time too.

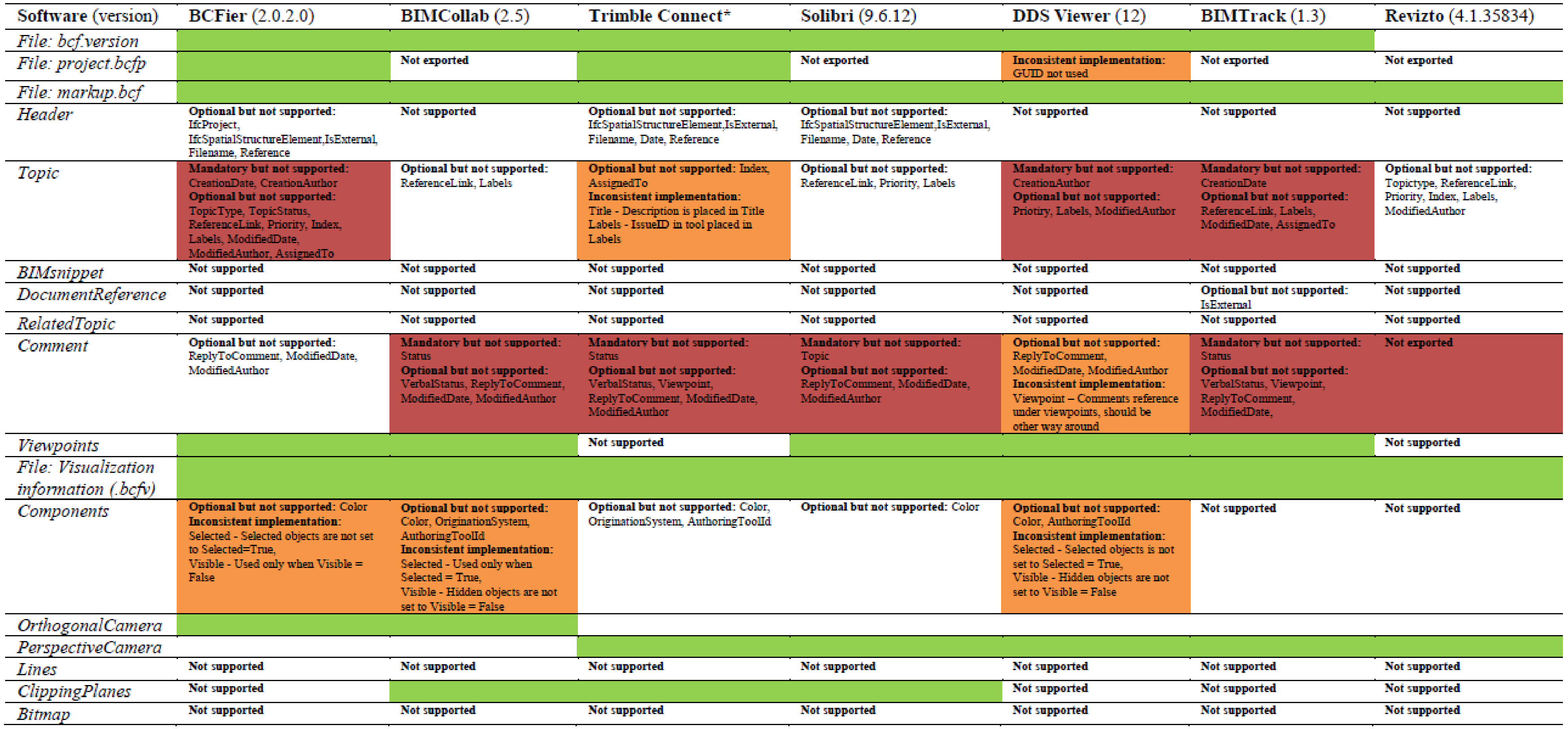
"Making project objectives explicit and representable in a consistent format, much of the currently implicit (or even hidden, misrepresented, misunderstood) intent behind design, construction, and operation decisions can be more clearly communicated, and the overall building process can be more efficiently and effectively performed and managed"(Hitchcock, 1995).

On the whole, several papers (Grilo & Jardim-Goncalves, 2010; Miettinen & Paavola, 2014; Xu et al., 2014) coincide in that BIM-enabled projects with interactive feedback on design decision consequences that are able to keep track and meet project objectives could generate greater savings and more efficient processes.

2.4 BIM Collaboration Format (BCF)

BIM Collaboration Format is an open standard that defines a XML-based schema to exchange written comments related to a specific IFC file. The implementation of BCF for the description of views on particular building elements and verbalization of related issues is one of the potential directions for its application (Jakob Beetz et al., 2011).

When used for management and communication of issues, a BCF file holds a description of the issue, a status, links to a BIM model and objects, a picture of the issue and a camera orientation (bcf XML v1); then such BCF file is packed in a ZIP file that will be later transferred by regular file sharing means. In 2014, the bcfXML v2 included the option to append documents (document reference) and elements of a data model (BIM snippet) as well as the option to include more viewpoints and snapshots. Nevertheless, according to the review carried out by Treldal, Parsianfar and Karlshøj (Treldal et al., 2016), the current implementation of BCF in tools that support tasks' management shows that no tool is able to export a BCF file that contains all specifications defined in the current bcfXML v2 (Figure 7). Amongst the tools they analysed, we can find: BCFier, KUBUS BIMCollab, Trimble Connect, Solibri Model Checker, DDS Viewer, BIMTrack and Revitzo.



Software (version)	BCFier (2.0.2.0)	BIMCollab (2.5)	Trimble Connect*	Solibri (9.6.12)	DDS Viewer (12)	BIMTrack (1.3)	Revizto (4.1.35834)
File: bcf.version	🟩	🟩	🟩	🟩	🟩	🟩	
File: project.bcfp	🟩	**Not exported**	🟩	**Not exported**	🟧 **Inconsistent implementation:** GUID not used	**Not exported**	**Not exported**
File: markup.bcf	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Header	**Optional but not supported:** IfcProject, IfcSpatialStructureElement,IsExternal, Filename, Reference	**Not supported**	**Optional but not supported:** IfcSpatialStructureElement,IsExternal, Filename, Date, Reference	**Optional but not supported:** IfcSpatialStructureElement,IsExternal, Filename, Date, Reference	**Not supported**	**Not supported**	**Not supported**
Topic	🟥 **Mandatory but not supported:** CreationDate, CreationAuthor **Optional but not supported:** TopicType, TopicStatus, ReferenceLink, Priority, Index, Labels, ModifiedDate, ModifiedAuthor, AssignedTo	**Optional but not supported:** ReferenceLink, Labels	🟧 **Optional but not supported:** Index, AssignedTo **Inconsistent implementation:** Title - Description is placed in Title Labels - IssueID in tool placed in Labels	**Optional but not supported:** ReferenceLink, Priority, Labels	🟥 **Mandatory but not supported:** CreationAuthor **Optional but not supported:** Priotiry, Labels, ModifiedAuthor	🟥 **Mandatory but not supported:** CreationDate **Optional but not supported:** ReferenceLink, Labels, ModifiedDate, AssignedTo	**Optional but not supported:** Topictype, ReferenceLink, Priority, Index, Labels, ModifiedAuthor
BIMsnippet	**Not supported**	**Not supported**	**Not supported**	**Not supported**	**Not supported**	**Not supported**	**Not supported**
DocumentReference	**Not supported**	**Not supported**	**Not supported**	**Not supported**	**Not supported**	**Optional but not supported:** IsExternal	**Not supported**
RelatedTopic	**Not supported**	**Not supported**	**Not supported**	**Not supported**	**Not supported**	**Not supported**	**Not supported**
Comment	**Optional but not supported:** ReplyToComment, ModifiedDate, ModifiedAuthor	🟥 **Mandatory but not supported:** Status **Optional but not supported:** VerbalStatus, ReplyToComment, ModifiedDate, ModifiedAuthor	🟥 **Mandatory but not supported:** Status **Optional but not supported:** VerbalStatus, Viewpoint, ReplyToComment, ModifiedDate, ModifiedAuthor	🟥 **Mandatory but not supported:** Topic **Optional but not supported:** ReplyToComment, ModifiedDate, ModifiedAuthor	🟧 **Optional but not supported:** ReplyToComment, ModifiedDate, ModifiedAuthor **Inconsistent implementation:** Viewpoint – Comments reference under viewpoints, should be other way around	🟥 **Mandatory but not supported:** Status **Optional but not supported:** VerbalStatus, Viewpoint, ReplyToComment, ModifiedDate,	🟥 **Not exported**
Viewpoints	🟩	🟩	**Not supported**	🟩	🟩	🟩	**Not supported**
File: Visualization information (.bcfv)	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Components	🟧 **Optional but not supported:** Color **Inconsistent implementation:** Selected - Selected objects are not set to Selected=True, Visible - Used only when Visible = False	🟧 **Optional but not supported:** Color, OriginationSystem, AuthoringToolId **Inconsistent implementation:** Selected - Used only when Selected = True, Visible - Hidden objects are not set to Visible = False	**Optional but not supported:** Color, OriginationSystem, AuthoringToolId	**Optional but not supported:** Color	🟧 **Optional but not supported:** Color, AuthoringToolId **Inconsistent implementation:** Selected - Selected objects is not set to Selected = True, Visible - Hidden objects are not set to Visible = False	**Not supported**	**Not supported**
OrthogonalCamera	🟩	🟩					
PerspectiveCamera			🟩	🟩	🟩	🟩	🟩
Lines	**Not supported**	**Not supported**	**Not supported**	**Not supported**	**Not supported**	**Not supported**	**Not supported**
ClippingPlanes	**Not supported**	🟩	🟩	🟩	**Not supported**	**Not supported**	**Not supported**
Bitmap	**Not supported**	**Not supported**	**Not supported**	**Not supported**	**Not supported**	**Not supported**	**Not supported**

Figure 7. Summary of results from bcfXML v2 exports – displaying missing, incorrect, supported and not supported content (Source: Paper *"Using BCF as a mediator for task management in building design"*, 2016)

Because during a real project the amount of BCF files generated could be really big, along with the release of bcfXML v2 a RESTful API was developed to allow a BCF server to automatically synchronise BCF tasks and avoid file exchange (Linhard, K. & Steinmann, R., 2015). In fact, an extension to BCF is presented (L. van Berlo & Krijnen, 2014) where project partners were able to create issues, manage them online and assess them in the BIM model through a BCF server; so that an overview of the status of both project and individual objects is created. Amongst the most important findings, it agrees with other authors (L. A. H. M. van Berlo et al., 2012; Hitchcock, 1995) on that is difficult to make project members work on the same software platform due to the fragmented nature of the industry, and therefore working with different aspect models can be more effective.

A comparison between the specifications of BCF and IDM Part 2 is also analysed by Treldal, Parsianfar and Karlshøj (Treldal et al., 2016) concluding that both standards have many similarities but neither standard is superior to the other. For that reason, it is believed that a harmonisation of the two standards is better than selecting one of them for tasks' and/or requirements management: *"IDM Part 2 is based on organising transactions and messages in predefined or at least hierarchical order. This can be valuable in documenting agreements, but it runs the risk of making everyday tasks overly complicated to define. In this light, it is better to use the BCF specification as the starting point for a harmonisation, and to use the principles from IDM Part 2 to add additional methodology and attributes"(Treldal et al., 2016)*.

Expanding the scope of the BCF specification to include methodologies from IDM Part 2 will make the specification more complex according to Treldal, Parsianfar and Karlshøj (Treldal et al., 2016), for that reason it proposes to follow the same expansion structure from IFC, dividing the BCF data schema architecture in different modules.

The idea of using a task management server is aligned with the previously described reasons for developing the BCF API web service to avoid BCF file exchange. Although managing confidential information from different companies on a central server could be a challenge. Investigations (L. A. H. M. van Berlo et al., 2012) have shown that a decentralised solution is more appropriate with model serves used only as reference models to one another. It suggests that tools like BCF could be used to communicate with their dedicated server and, where a tool needs information on tasks from other servers, they could use BCF to query other servers for the information required. For that purpose, it is preferable if the BCF specification used URI's instead of only GUIDs to identify not only the unique tasks but also the location of the task.

On the other hand, the literature regarding the implementation of BCF for anything different that issue management is scarce. In fact, as it has been mentioned before by Leon van Berlo (L.A.H.M. van Berlo et al., 2014), there is not a lot of experience with BIM information in the requirements stage of the projects. It is true that the original purpose of BCF was informal communication during the design phases (Asier Mediavilla et al., 2015), however, an added value for the BCF application range would be to study tasks' management as well as validation and documentation of project requirements (Treldal et al., 2016).

2.5 Requirements' management

Clients' requirements cover aspects that go from the overall goals, activities and spatial needs to very detailed material and condition requirements. This documentation is the starting point of any design process, but the design is by nature an iterative process; therefore changes are made and client requirements evolve. However, usually, the requirements documentation is not updated accordingly to these changes, which can lead to an end result which is really different from the original documented requirements. And if requirements are not satisfied, the quality of the design and construction process is questionable.

Concerning the management of requirements, some research has been done so far. For instance, a project ontology that explicitly represents the functional requirements, designed forms and predicted and observed behaviours of the product, organisation and process (POP) of the project has been introduced by García, Kiviniemi and Ekstrom (García, Kiviniemi, & Ekstrom, 2003).

Through the Extreme Collaboration (XC) and Virtual Design and Construction (VDC) project development methods, the shared visualizations of the POP models allowed designers to see the impact of their design choices on both their own and related disciplines as well as give the opportunity for all designers to review the design choices of other disciplines. As a result, *"capturing, representing and sharing the desired, predicted and observed project information allowed the project team to check how the evolving design satisfied specifications, predicted potential conflicts as well as learn from successful and not successful cases"* (García et al., 2003).

Following the same reasoning, a requirement management interface is suggested later on (Arto Kiviniemi & Martin Fischer, 2004) by linking a unique requirements model to multiple design alternatives in product model based design tools (Figure 8 and Figure 9).

The practical importance of this requirements interface is that it represents a solution for the management of complex project information that, together with the duration and changing actors in different project phases, makes it impossible for participants in a design process to remember every relevant requirement and, what is more, the relationship and influence of each in the design solutions.

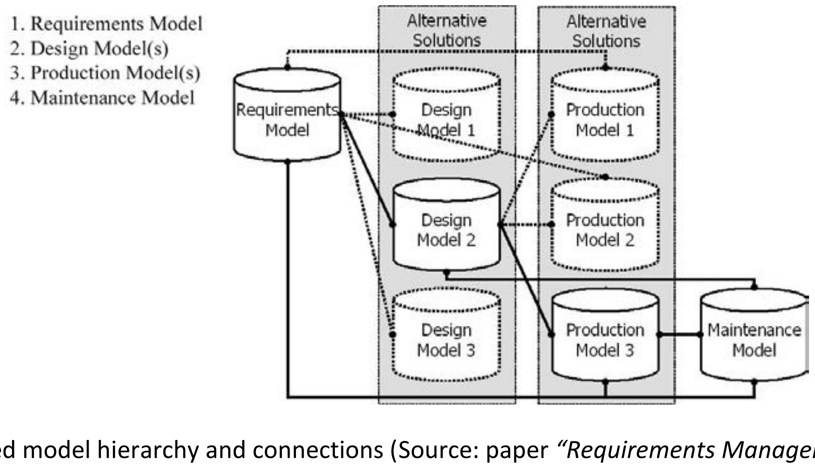


Figure 8. Proposed model hierarchy and connections (Source: paper *"Requirements Management Interface to Building Product Models"*, 2004)

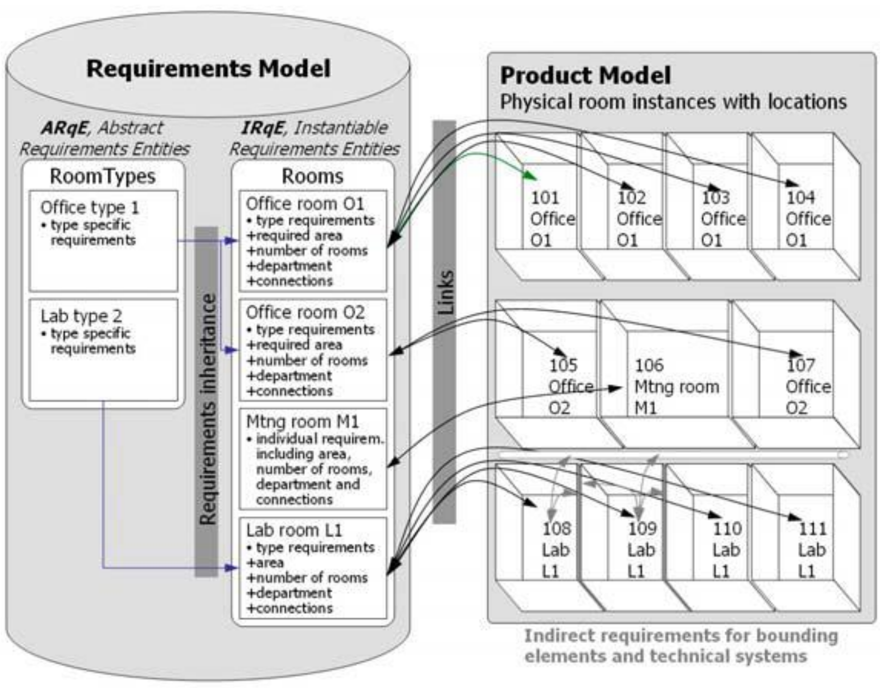


Figure 9. Proposed concept to link detailed spatial requirements to a product model (Source: paper *"Requirements Management Interface to Building Product Models"*, 2004)

In previous chapters, Gu and London (Gu & London, 2010) mentioned that current design tools do not support recording of client's requirements and its evolution is usually not communicated to the whole project team. Moreover, it is believed (Arto Kiviniemi & Martin Fischer, 2004) that there is a lack of theory to link the requirements to the product model based design systems, due to the fact that there is no connection between requirements and design documents and because requirements are not updated coherently and in an easily accessible format. In addition, it also agrees with Leon van Berlo (L. A. H. M. van Berlo et al., 2012) on that a good solution to the availability of information and requirements management includes the verification of "fuzzy" requirements (like human interpretable descriptions) and an appropriate level of detail; which means to find exactly the relevant information for the ongoing design task from the project data.

Another important aspect of a comprehensive solution regarding the management of requirements is their traceability. The requirements history and their evolution during the design process are addressed through the development of a prototype system called DesingTrack (Ozkaya & Akin, 2007). However, the findings that arose during their research into the requirements – design relationships in architectural design are, for future research, as important as the development of the prototype itself.

First of all, establishing a requirements structure was one of the faced problems during their research because relationships between requirements and design are observed during design sessions rather than during data modelling. It is asserted that the dependencies between requirements and designs often are more complex than that can be captured with a predefined model, they are instance-based. For that reason, it is important to think whether the integrated design environment approach is compatible with the natural flow of the design and what it takes for the users to adjust to such a mental model; which brings back the point of Grilo and Jardim-Gocalves (Grilo & Jardim-Goncalves, 2010) about the fact that introducing a management framework into the process is not only a ICT matter but also cultural.

Secondly, requirements management should not be considered a front-end task during the design process or as an activity that is addressed marginally; but it should be in correlation with form exploration. Any architectural design project begins with a set of given requirements which can be as well defined as a pre-prepared program, or as fuzzy as a list of spaces and soft expectations. According to Ozkaya and Akin (Ozkaya & Akin, 2007), understanding and capturing the repeating requirements patterns and used relationships between them is the first step for future research, which requires the generation of a lot of different examples.

The approach from Kiviniemi and Fischer (Arto Kiviniemi & Martin Fischer, 2004) is discussed by Ozkaya and Akin (Ozkaya & Akin, 2007) too. It holds that extending the building information model with another model that gathers all possible requirements attributes ignores the fact that in architectural design semantic difference occur very often in representing design attributes, while extending DesingTrack to enable the use of IFC as a classification superset makes building information model transparent to the user allowing its use with other models.

As it has been mentioned by different authors, requirements management involves more than an ICT approach, it requires a systems integration that needs to fit into the working environment so it adapts to the collaborating workflow. Collaboration (including communication, document management and interoperability) is claimed (W. Shen et al., 2010) to be the most frequent issue in the construction industry due to its complex and multidisciplinary collaborating teams, the construction project lifecycle, and the use of heterogeneous software systems/tools.

When it comes to BIM and IFC, current validation tools, like SMC, use a similar methodology (import IFC file, parse, store, and retrieve IFC instance) and for that reason it is believed (Y.-C. Lee et al., 2015) that there is a possible integrated approach for validation of BIM data. However, still an automated rule checking process for the validation of project design and requirements requires that efforts are focused on reinterpreting requirements that are provided in a natural language into something formally understood (Krijnen & van Berlo, 2016), which has been shown to be a laborious task (Ozkaya & Akin, 2007).

2.6 Semantic Web

According to Beetz, Leeuwen and de Vries (Jakob Beetz, Jos P. van Leeuwen, & Bauke de Vries, 2006): *"One of the most elemental and agreed upon widely accepted visions of the Semantic Web initiative is to create a network of interconnected knowledge resources rather than loosely coupled documents that are merely compatible on a syntactical level. The goal is to make these resources discoverable, retrievable, interpretable and processable for pieces of software that act on behalf of a user"*.

During the early 2000s, researchers started introducing the Semantic Web technology in the AEC industry as an opportunity to enhance the machine readability and interpretability of distributed information and as a way of standardising the information exchange between softwares (Jakob Beetz et al., 2009). First investigations (T. Elghamrawy & Boukamp, 2008; Pan, Anumba, & Ren, 2004) pointed out the improvements that using semantic web technologies could bring in terms of information exchange in the construction field.

In the next years and considering the introduction of IFC in the AEC domain as well, the use of semantic web technologies represented the possibility to address interoperability issues by allowing linking diverse information models. For instance, Pauwels and van Deursen (Pieter Pauwels & Davy Van Deursen, 2012) discussed how IFC models can be made available as RDF graphs in the semantic web and how they can be linked to other information; concluding with the linked data strategy as a valid approach for addressing interoperability issues.

This tendency of using semantic web technologies is embraced in the technical roadmap of BuldingSMART as well. Moreover, there are several papers (Abanda, Tah, & Keivani, 2013; Edward Corry et al., 2014; Tarek Elghamrawy & Boukamp, 2010; Hans A. Schevers et al., 2007; Pieter Pauwels, Zhang, & Lee, 2017; L. Shen & Chua, 2011) that discuss and support how Semantic Web technology can ease the collection of information by providing a network of connected data and preventing building designers from having to spend time on data preparation and manual entry into applications, like for instance, energy analysis (Niknam & Karshenas, 2015). In addition, construction cost estimating is also included in the range of expected benefits. By linking building information models to and from external data sources provided by material suppliers the data can be accessed, combined and shared in a straightforward manner in a machine – processable representation (Pieter Pauwels, Seppo Törmä, Jakob Beetz, & T. Liebich, 2015).

Another example is the one in the article *“Connecting building component catalogues with BIM models using semantic technologies: An application for precast concrete components”* (Gonçal Costa & Leandro Madrazo, 2015), where it is aimed to assist the design team in the assembly and dimensioning of structural components during the project phase by combining both types of information: building component catalogues with BIM models.

The truth is that current implementations of BIM are limited regarding the collaboration between actors in the design process when it comes to building information exchange. So the necessity of moving from modelling approaches based information to innovative representation systems based on knowledge exists (Rezgui, Boddy, Wetherill, & Cooper, 2011; Simeone & Cursi, 2016; Tamer E. El-Diraby, 2013). In other words, the point is to change from a model based on information (which offers a lot of information but it is not manageable) to a model based on knowledge (so actors only receive information that is needed).

By modelling concepts and relationships among the technological components that are contained in a BIM model, it is possible to manage the semantic level of representation in order to make sure that every actor in the building design process access only the information that makes him understand design choices and, therefore, collaborate in a more efficient way. These concepts and relationships are represented in an ontology, which contains entities (domain objects), relations between them, the attributes (properties) of these entities and their values (Svetel & Pejanović, 2010). In order to formalise an ontology, specific technologies and standards like the Resource Description Framework (RDF) and the

Ontology Web Language (OWL) are used; so when software applications communicate using, OWL, for example, relations can be made between different ontologies to improve interoperability.

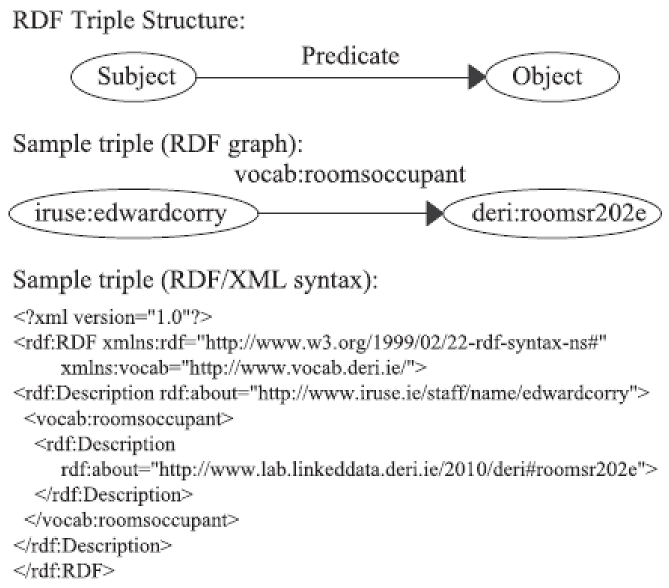
2.6.1 Resource Description Framework (RDF) and SPARQL

The meaning of concepts could be expressed by using RDF as the data model for representing information about entities.

"RDF is, first and foremost, a system for modelling data. It gives up in compactness what it gains in flexibility. Every relationship between any two data elements is explicitly represented, allowing for a very simple model of merging data. A relationship (expressed in a familiar form of subject/predicate/object) is either present or it is not. Merging data is thus reduced to a simple matter of considering all such statements from all sources, together in a single place" (Allemang & Hendler, 2011a).

Information is represented as directed labelled graphs (RDF graphs) where each node represents a concept or object in the world, and it is identified with a Unique Resource Identifier (URI) (Pieter Pauwels et al., 2017). URIs have a global scope, associating a URI with a resource means that anyone can link to it, refer to it, or retrieve a representation of it (Shadbolt, Berners-Lee, & Hall, 2006). What is more, URIs ensure that concepts are not just bare terms devised by someone, but are connected to unique definitions on the Web (and that is the strength of this technique).

An RDF graph can be serialised using different syntaxes like RDF/XML (Figure 10), N-triples, Turtle (TTL) and Notation-3. The structure of an RDF graph can be improved by using RDF vocabularies or ontologies. The most basic elements are contained in the RDF Schema (RDFS) vocabulary (specifications of classes, subclasses, comments and data types) and more expressive elements are also available within OWL vocabulary.



```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:vocab="http://www.vocab.deri.ie/">
<rdf:Description rdf:about="http://www.iruse.ie/staff/name/edwardcorry">
  <vocab:roomsoccupant>
    <rdf:Description
        rdf:about="http://www.lab.linkeddata.deri.ie/2010/deri#roomsr202e">
    </rdf:Description>
  </vocab:roomsoccupant>
</rdf:Description>
</rdf:RDF>
```

Figure 10. Example of RDF triple structure (Source: paper *"Using semantic web technologies to access soft AEC data"*, 2014)

According to Beetz (J. Beetz, 2014), the advantage of using RDF for modelling the information is the capability to store an arbitrary number of states into a graph. In this way, a single GUID could identify a whole cluster of concepts, their values and relationships. Moreover, it is believed (Jakob Beetz et al., 2009) that using graph query languages to operate on large ontologies like average IFC models is less complex than the use of complete rule and reasoning engines.

Currently, several initiatives about representing building information as RDF graphs or using semantic web technologies can be found. For instance, an initial study is carried out to show how diverse streams of information can be captured and linked with other building data in order to broaden the range of data silos available for building performance optimisations (Edward Corry et al., 2014). As it is explained, their work is ongoing and is focused on converting data silos to RDF and developing a performance framework (Figure 11) capable of capturing and interpreting this data.

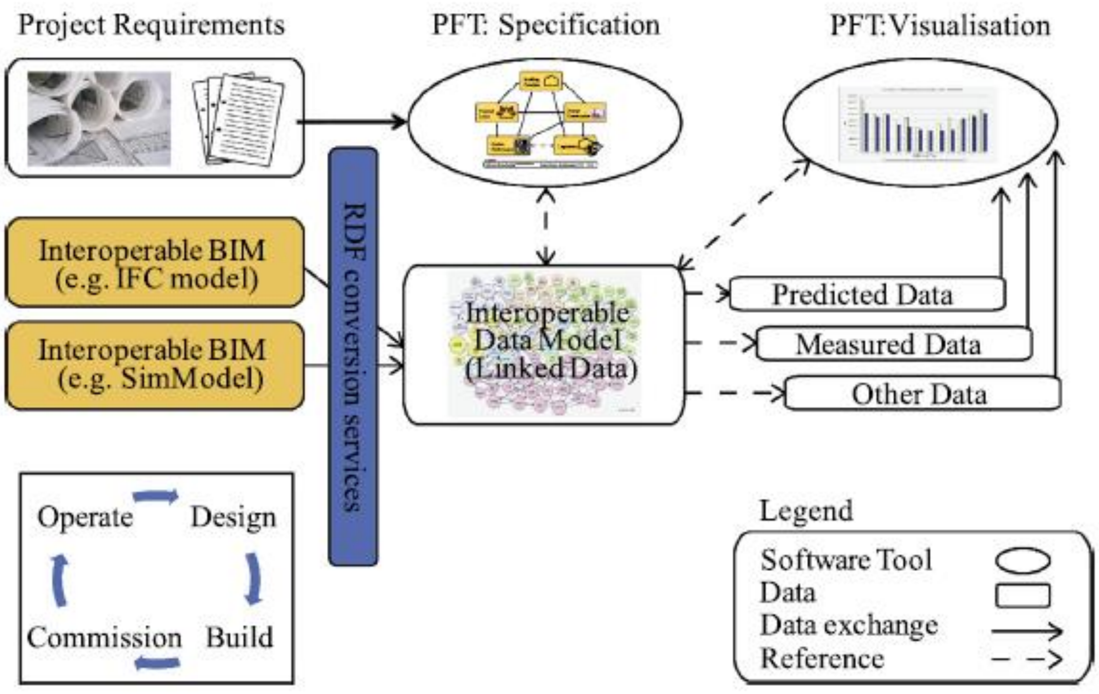


Figure 11. Semantic web based building performance assessment platform (Source: paper *"Using semantic web technologies to access soft AEC data"*, 2014)

Another example (Hans A. Schevers et al., 2007) shows how a digital facility model of the Sydney Opera House was aimed to be developed by using RDF and OWL. This model enables IFC objects to be connected to OWL objects, creating a service oriented software environment.

Moreover, an ontology for monitoring and controlling energy consumption is developed and extended through a collection of different papers (H. Wicaksono, Rogalski, & Kusnady, 2010; Hendro Wicaksono, Dobreva, Häfner, & Rogalski, 2013). This ontology is carried out by building an RDF representation of a building model and using an OWL ontology for the building information.

Nevertheless, representation of data is useless without means to access it. In order to do it, the standard for retrieving the data uses a query language called SPARQL (SPARQL Protocol and RDF Query Language). It provides means for querying information from an RDF graph or to transform a graph into a new form (Allemang & Hendler, 2011b; J. Beetz, Coebergh, Botter, Zlatanova, & De Laat, 2014). Basically, queries are matched against data graphs (Figure 12) by specifying on their (queries) graph patterns what information is requested from the data graph. The structure of the query follows the graph pattern including resources and variables, indicating how entities that match variables are related to one another.

```
SELECT ?director
WHERE {:JamesDean :playedIn ?movie .
       ?movie :director ?director .}
```

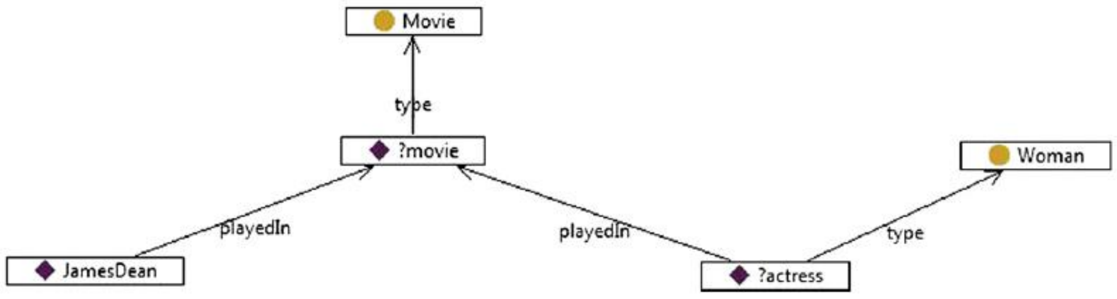


Figure 12. Example query structure against data graph

SPARQL implementations are documented in several papers together with the use of semantic web technologies in the AEC industry. In fact, the original ontology developed by Wicaksono (H. Wicaksono et al., 2010) is extended by including another OWL ontology inspired by IFC, while a SPARQL endpoint is built on top of the used rule engine in order to query the results of the rules. Similar approaches (Bouzidi, Fies, Faron-Zucker, Zarli, & Thanh, 2012; Zarli, A.Yurchyshyna, Le Thanh, & Faron Zucker, 2008) rely entirely on SPARQL SELECT and CONSTRUCT queries for rule checking.

On the other hand, Pauwels (P. Pauwels et al., 2011) points out that SPARQL it is useful for a one-by-one querying but it does not allow an equally automated rule checking process as a combination of a dedicated rule language and a rule engine can provide. In this approach, an information description language and a rule language stemming from the semantic web field are investigated as a possible enhancement of IFC for building performance checking.

Amongst other examples, SPARQL queries are proposed (Jakob Beetz et al., 2006) to convert on demand IFC geometry into alternative geometric representations. In addition, in other papers (Jakob Beetz, Bauke de Vries, & Jos van Leeuwen, 2007; Matthias Weise & Pieter Pauwels, 2015) SPARQL queries are used as well to generate partial model views from RDF-encoded IFC models.

Concerning cost estimation, it was proposed by Lee, Kim and Yu (S.-K. Lee, Kim, & Yu, 2014) to extract information from an ifcXML file and parse the information as RDF instances of two small ontologies: a work item ontology and a work condition ontology. SPARQL queries are used later for retrieving the information from the resulting graph that is held in a SPARQL endpoint.

"When an expert inputs work item queries as a form of SPARQL via the SPARQL endpoint, the query layer submits them to the ARQ engine, which subsequently retrieves the relevant work item from the knowledge base. Results of the SPARQL queries are returned in XML format, which is easy to use in a cost estimating application" (S.-K. Lee et al., 2014).

Nikham and Karshenas (Niknam & Karshenas, 2017) present a shared ontology approach to the semantic representation of building information where SPARQL queries are used to integrate data from various AEC-FM domain knowledge bases. As it is justified in the paper, no discrepancies were observed when comparing the results coming from different SPARQL queries for inter-domain information access with similar information that was manually extracted from design, cost and schedule documents.

Finally, concerning SPARQL and the query language, the existence of the web application SPARKLIS developed by Sebastian Ferré should be mentioned. *"Sparklis is a query builder in natural language that allows people to explore and query SPARQL endpoints with all the power of SPARQL and without any knowledge of SPARQL."*("Sparklis | DBpedia," 2017).

By using this web application, no previous knowledge of the RDFS schema or the vocabulary is needed in order to discover and consult the contents of any endpoint. In addition, at the same time the user explores the information stored in the SPARQL endpoint, queries are generated according to filters that the user has implemented by using natural language.

3. Methodology

3.1 Introduction

This chapter of the thesis contains the description of the methodology that was carried out during the research stage at the company Verhoeven en Leenders (V&L). The approach studies the existing validation framework, at both the Building Structures and the Civil Division of V&L, by reading the available company documentation and interviewing several employees regarding three actual projects provided by the company. These projects have been used as a source of information for the study of requirements management and exchange of information workflow. As a result, a System Breakdown Structure (SBS)/Requirements Breakdown Structure (RBS) matrix and a process map of the exchange of information workflow have been elaborated.

The SBS/RBS matrix and process maps are explained and discussed in sections 3.5 Results and 3.6 Discussion of this chapter.

3.2 Building Structures vs. Civil Division

Verhoeven en Leenders is a structural engineering company which specialises in the design, calculation and modelling of construction projects in the fields of building structures and infrastructure. Therefore, the company is divided into two work divisions and namely, Building Structures Division and Civil Division.

Considering the scope of this thesis, the initial understanding of these divisions was that they differ on the management of requirements coming from the client at the beginning of the design phase. According to experts within the company, the strict demands for project management and organisation in civil projects coming from the clients (e.g. RWS) make it compulsory to handle all of the project's requirements through system engineering management tools (Relatics). In this way, the client can rely on a management tool for the future maintenance of the project after its completion. In order to gain a better insight into how requirements are structured and managed by the use of Systems Engineering management tools throughout the design and construction of a project, one of the civil projects of the company, where both are applied, was investigated and namely, the bicycle park Vijfhoek.

In the SE schema (Figure 13) a flowchart displaying the sequencing of tasks during the design process, as well as the input and output information for each one of the steps, is presented.

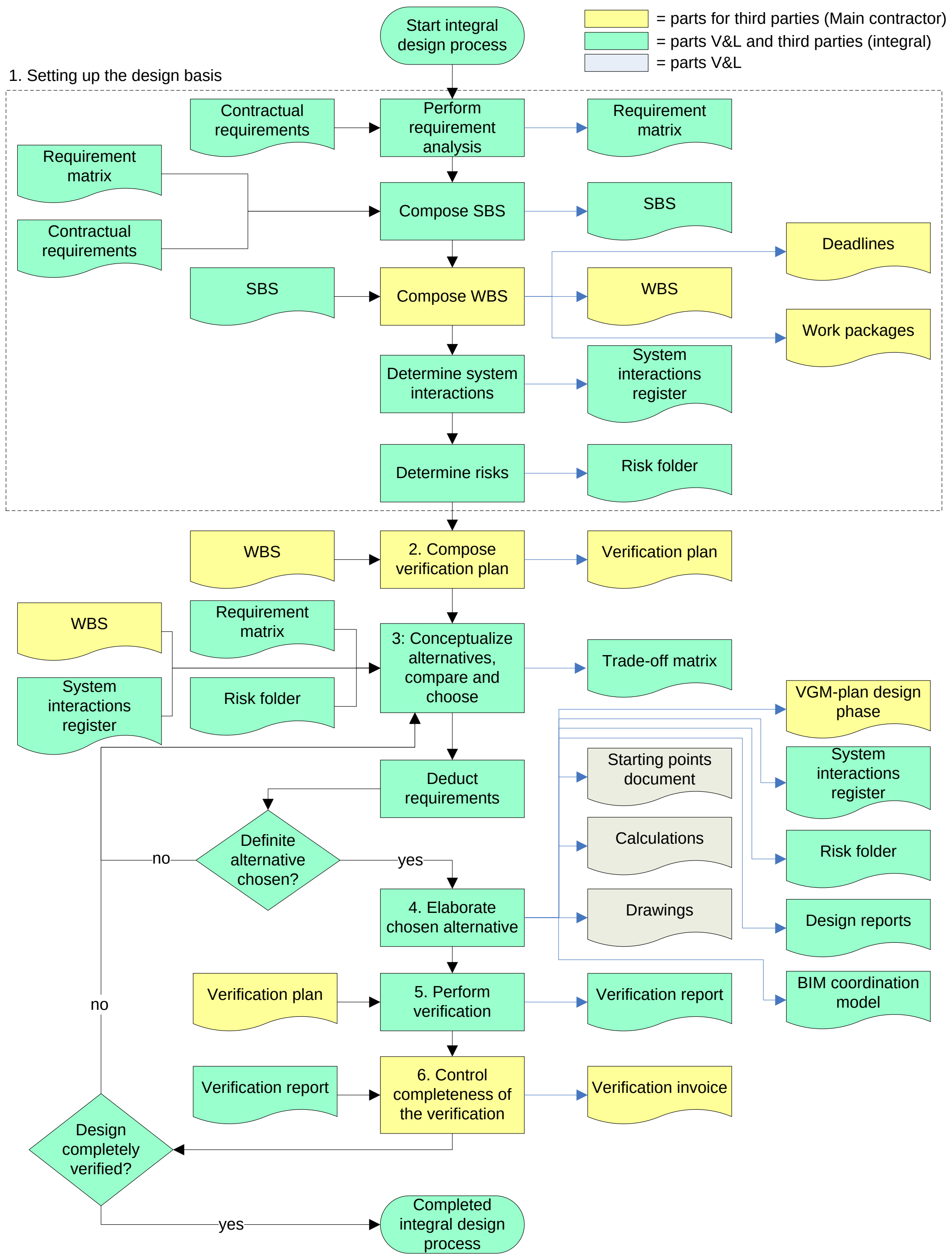

Start integral design process
= parts for third parties (Main contractor)
= parts V&L and third parties (integral)
= parts V&L
1. Setting up the design basis
Contractual requirements
Perform requirement analysis
Requirement matrix
Requirement matrix
Contractual requirements
Compose SBS
SBS
Deadlines
SBS
Compose WBS
WBS
Work packages
Determine system interactions
System interactions register
Determine risks
Risk folder
WBS
2. Compose verification plan
Verification plan
WBS
Requirement matrix
3: Conceptualize alternatives, compare and choose
Trade-off matrix
System interactions register
Risk folder
VGM-plan design phase
Starting points document
System interactions register
Deduct requirements
Calculations
Risk folder
no
Definite alternative chosen?
yes
4. Elaborate chosen alternative
Drawings
Design reports
BIM coordination model
no
Verification plan
5. Perform verification
Verification report
Verification report
6. Control completeness of the verification
Verification invoice
Design completely verified?
yes
Completed integral design process

Figure 13. Systems engineering framework (Source: Systems Engineering Handbook V&L)

When we compare the well-structured workflow in the design of a civil project (Figure 13) with the workflow within the Building Structures Division, where no framework for the sequencing of tasks and the information exchange between stakeholders has been established, it becomes clear that project requirements are being poorly managed. The lack of a systematical management framework results in the loss of valuable project information due to the fact that the decision-making process, the design iterations and the tasks performed by the engineer are not explicitly documented. This issue also results in the absence of a traceable record of requirements during the different project phases. Another aspect, which additionally complicates the matter, is the fact that when it comes to building structures projects, it is the architect who is more in contact with the client and carries out the first drafts of the project. In other words, the architect manages the demands of the client by creating the first architectural model that will be later sent to the structural engineer and modeller, who will cooperate in the creation of the structural design. Therefore, the requirements associated with the structure are a derivative of the architectural design.

If we compare both situations, we can conclude that the influence on the requirements' management for the engineers in V&L is greater in civil projects due to the fact that client requirements are delivered directly to them; while in a building structures project, where the architect has the lead, requirements are already managed and defined by him/her in the architectural model. In fact, requirements that concern a structural engineer are also different from the ones that are relevant for an architect. What is more, as it was pointed out by experts in V&L, engineers can count on a more developed framework for Systems Engineering (SE) management within civil projects because SE is demanded as a process. On the other hand, SE is less developed within building structures projects because it is not demanded yet.

When we look at the collaboration workflow between the structural engineer and the BIM modeller for the development of the structural model, BIM Collaboration Format (BCF) is being used between them for the exchange of information; specifically, clash detection/revision. However, information shared between these two parties through BCF only contains comments or indications regarding flaws or decisions in the design made by the draftsman that will interfere with the compliance of the client's requirements. The primary form of communication between the two parties, however, still remains either in the form of 2D sketches, remarks marked in colour directly on the 2D drawings (floor plans, sections) of the project or simply face-to-face feedback. These practices further confirm the inconsistent nature of information management within the division. For that reason, it is interesting to check whether it is possible to take a step further with BCF and extend the exchange of comments into requirements verification.

3.3 Verhoeven en Leenders: organisation and projects

The project's design phases, the level of development (LOD) levels associated with them, as well as all actors involved in them and their responsibilities are defined in the Quality Manual (*Dutch: Kwaliteitshandboek*) of the company. This manual clarifies how each design phase should be carried out and which information/documents should be transferred between the collaborating parties. The manual also distinguishes between Building Structures and Civil projects.

The different project's design phases in which V&L are usually involved in are the following:

- **VO = Preliminary Design *(Dutch: Voorlopig Ontwerp)* = LOD200**
- **DO = Final Design *(Dutch: Definitief Ontwerp*) = LOD300**
- **UO = Execution design – *(Dutch: Uitvoering Ontwerp)***

- **UO-PV = Design Execution *(Dutch: Uitvoering Ontwerp – Productie Voorbereiding)* = LOD350**
- **UO-WT = Implementation Design *(Dutch: Uitvoering Ontwerp – Werktekening)* = LOD400**

The corresponding levels of detail (LOD) for the different design phases of a Building Structures project in the company are also explained in the table below.

Table 1. Levels of detail (LOD) in V&L (Source: Systems Engineering Handbook V&L)

LOD 200 (VO)	
In this phase, the starting points for the structural design are defined.	
Modeler	The modeller makes 2D drawings of the primary supporting structure based on the drawings (floor plans, sections) and 3D model received from the architect.
Engineer	The structural engineer makes basic calculations of the loads for each floor, based on the 3D model and drawings (floor plans, sections) of the architect and also in consideration of the room categories the building.
LOD 300 (DO)	
In this phase, the structural design and the corresponding calculations are prepared.	
Modeler	In this phase, the structural model for the primary supporting structure, as well as the drawings of the construction details are carried out with the help of Revit or Tekla Structures.
Engineer	The engineer prepares the official design calculations, consisting of calculations of individual elements and/or systems with their corresponding forces and reactions, as well as determines the profiles and material quality of the elements.
LOD 350 (UO-PV)	
In this phase, the structural design is prepared for further development by third parties. Data for this purpose shall be provided by other parties (contractor, architect, MEP consultant/engineer etc.).	
Modeler	The modeller applies the necessary changes/additions to the model including the deviations related to the openings.
Engineer	The engineer must calculate the openings and recesses in the structure which are classified into three categories and namely, structural, architectural and installation openings.
LOD 400 (UO-WT)	
In this phase, concrete works for the in-situ concrete are elaborated on for the execution of the structural design.	
Modeler	If possible, the modeller adds reinforcement data in the 3D model in Revit/Tekla. Otherwise, the same information is displayed in the 2D structural drawings.
Engineer	The drawings of the formwork including measurements are prepared for the concrete components poured in situ and eventually, a plan of the piles is also prepared.

3.3.1 Case studies

Three actual projects were studied during the research stage of this thesis in order to find similarities and differences not only in terms of requirements' management but also in regards to the contents of the calculation documents, depending on the level of detail and the design phase. The first two projects belong to the Building Structures division and the third one is a part of the Civil division (Figure 14). The names of the projects are:

1. **Academy Vanderlande te Veghel (Academy)**
2. **Nieuwbouw MAVO Schravenlant XL te Schiedam (School)**
3. **Fietsparkeergarage Vijfhoek (Bike parking)**

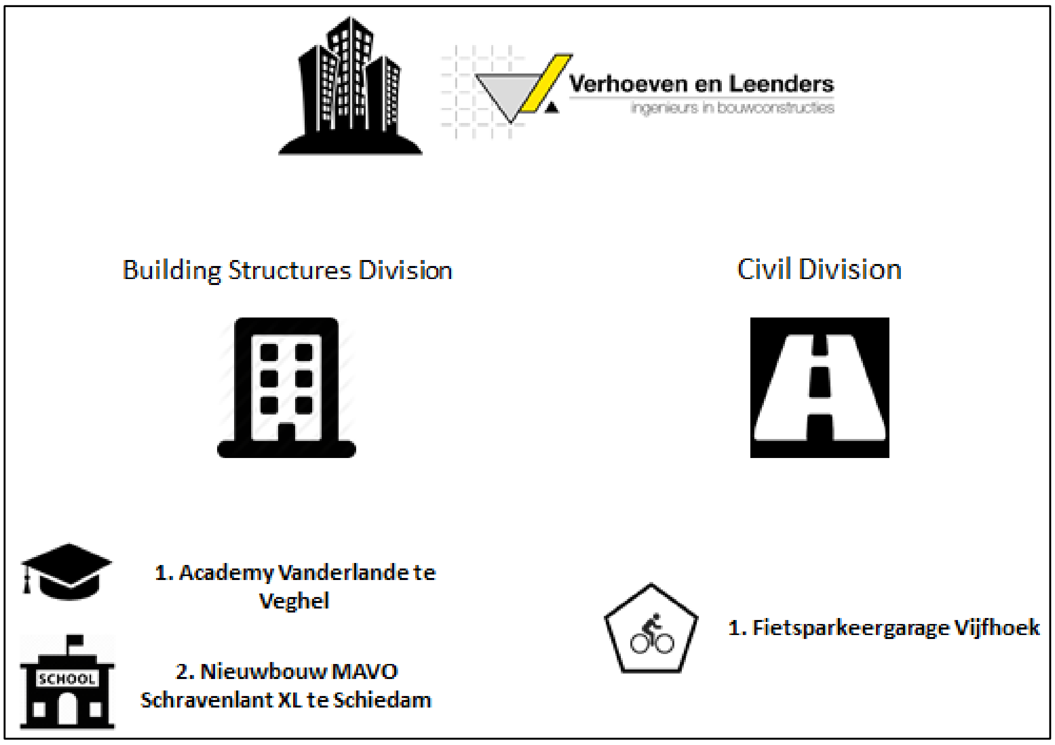

Verhoeven en Leenders
ingenieurs in bouwconstructies
Building Structures Division
Civil Division
1. Academy Vanderlande te Veghel
2. Nieuwbouw MAVO Schravenlant XL te Schiedam
SCHOOL
1. Fietsparkeergarage Vijfhoek

Figure 14. Case studies from V&L

Regarding the civil project (Fietsparkeegarage), SE was implemented through a systems engineering management tool (Relatics) since the beginning of the project for the management of requirements. As a result, the engineers working on the project used an SBS and an RBS for the organisation of the work packages, for clash detection and for keeping an updated record of the different tasks performed during the different design phases.

Upon taking a look at the building structures projects, only the school project (Nieuwbouw MAVO Schravenlant XL te Schiedam) counted on Systems Engineering principles for the tracking of project requirements. It consisted of an Excel sheet, provided by the company in charge of the preliminary design of the project and was thus a very simplified attempt at using SE. Regarding the academy project, no SE or any other type of tool for the capturing and managing of information was used.

The civil project was used as an initial source of information or as an example of the successful implementation of SE in a construction project. The Relatics organisation and different tree structures were useful to gain insight about how requirements are organised and how they relate to physical objects; their level of detail, and the way these requirements evolve during the different design phases of the project.

It should be pointed out that the design phase at which V&L started working on the project is different for each of the building structures projects. In other words, V&L was in charge of the project design since the preliminary design in the academy while for the school, it was another company who made the preliminary design of the project and V&L continued with the final design. This fact has additionally impacted the way requirements are organised in both projects because for the school the continuity between the preliminary design phase and final design phase had to be assured.

In addition, the legal structure of both projects is also different due to the difference in contract choice (Figure 15). The contract type for the academy project is a UAV contract, which presupposes a traditional way of working (Figure 16). The school project, however, has a UAV-GC contract where the way of working between the different project's stakeholders is integrated and thus, collaborative (Figure 17).

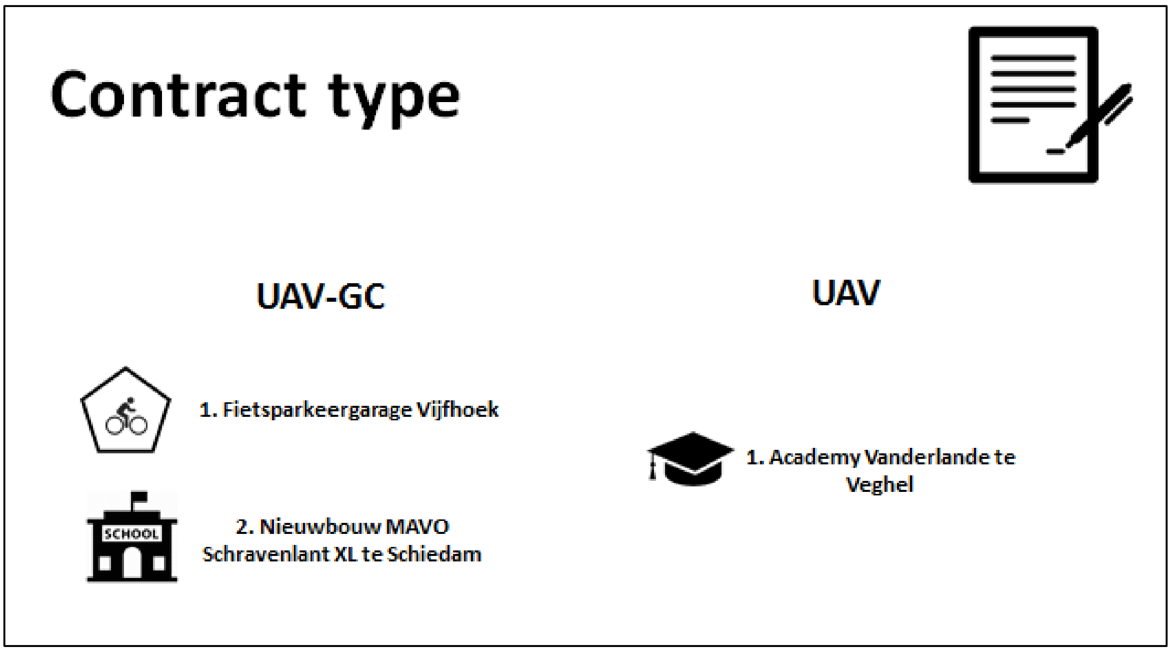

Contract type
UAV-GC
UAV
1. Fietsparkeergarage Vijfhoek
2. Nieuwbouw MAVO Schravenlant XL te Schiedam
SCHOOL
1. Academy Vanderlande te Veghel

Figure 15. Case studies classified by contract type

The collaboration workflow for the MAVO project (Figure 17) is thoroughly analysed and documented as process maps (communication and input/output workflow) in the results of this methodology/approach.

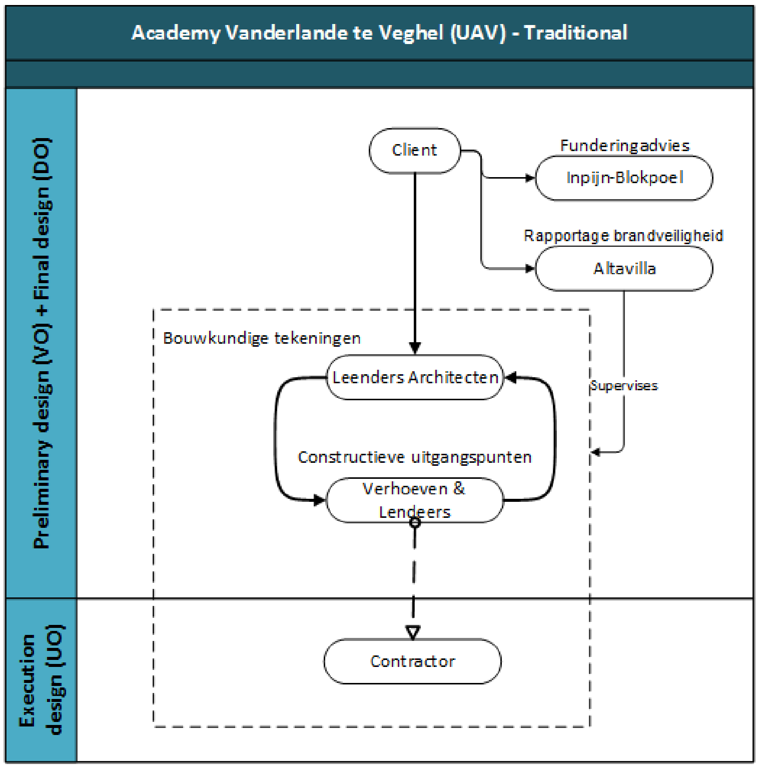


Figure 16. Parties involved in the workflow for UAV contract (traditional)

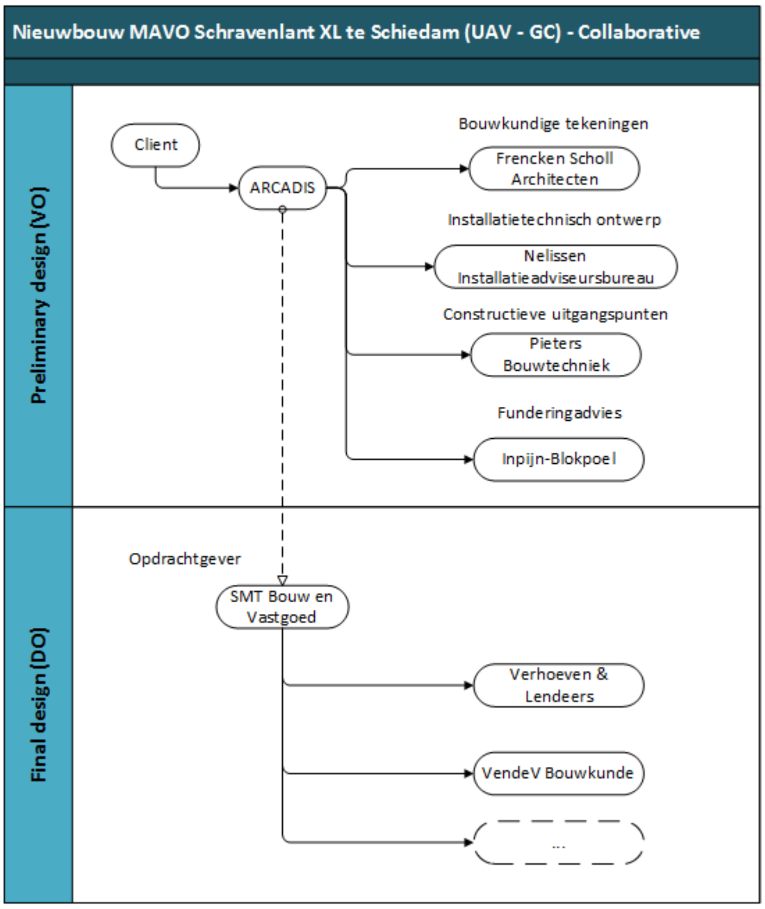


Figure 17. Parties involved in the workflow for UAV-GC contract (collaborative)

3.4 Verhoeven en Leenders: systems engineering

V&L counts on a Systems Engineering (SE) manual that provides the guidelines for implementing SE during the design phase of a construction project (Figure 13). The main benefit of Systems Engineering for the AEC domain is the possibility to capture and organise vast quantities of information while making the information easily traceable and accessible through the implementation of a tree structure. Although there can be various types of tree structures related to a construction project (e.g. process, requirements, objects…), the ones that are useful for this thesis and therefore, will be used in this investigation, are the System Breakdown Structure (SBS) and the Requirement Breakdown Structure (RBS).

According to the SE Handbook from V&L, the SBS is a hierarchical description of the physical parts composing a certain project, while the RBS is a summary of all the identified requirements in a construction project which should be directly related to the components from the SBS object tree. The SBS divides an entire system into different parts until an individual system element is obtained and the system cannot be further divided into components. Each component should obtain a unique number, which serves as an identifier and in addition, it should be assigned to a discipline responsible for its completion (e.g. structural engineering, MEP engineering). When talking about requirements, however, a distinction needs to be made between the different types of requirements that exist in a construction project:

- **Functional requirements *(Dutch: Functie-eisen)***
  Requirements relating to the functions which need to be realised; they indicate 'what the system should do'.
- **Aspect requirements *(Dutch: Aspecteisen)***
  These are requirements relating to supporting functions or aspects of the system, for example, requirements regarding management and maintenance, design, the stability of the system.
- **Object requirements *(Dutch: Objecteisen)***
  Requirements related to objects that have an impact on, for example, the shape, colour, strength, and dimensions. These requirements arise as a result of the design choices of client and contractor.
- **System Interaction requirements *(Dutch: Raakvlakeisen)***
  Requirements which come as a result of relations between the system and the system's environment (external requirements) as well as from interactions between different components of the system (internal interactions/clashes).
  For example, external interactions could be the nuisance caused to the neighbours of a construction site; an internal interaction could be the clash between physical objects in the model.
- **Process requirements *(Dutch: Proceseisen)***
  Requirements for activities which are necessary to be performed in order to successfully and timely achieve an objective (e.g. piling may take place from ... to ... hours).

To each requirement, a unique identification number is assigned, as well as a requirement description, parent requirement, responsible person and connection to an object (or objects) to which the requirement should be applied. Each of these types of requirements, except the Process requirements, is later subdivided according to the hierarchy in Figure 18, depending on the level of detail of the construction project.

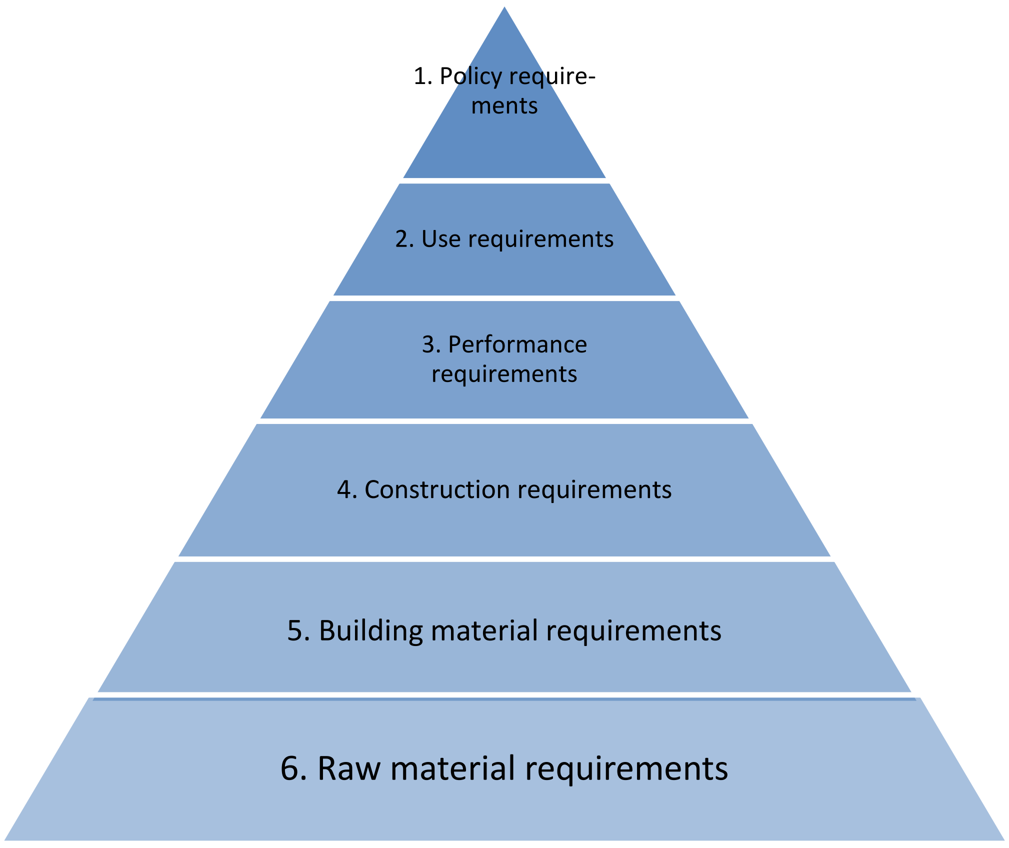

1. Policy require-ments
2. Use requirements
3. Performance requirements
4. Construction requirements
5. Building material requirements
6. Raw material requirements

Figure 18.Hierarchy of requirements (Source: systems engineering handbook from V&L)

1. **Policy requirements**
   These are requirements regarding amongst others, capacity and social security. They are intended for planners, urban designers, etc.
2. **Use requirements**
   They relate to the functioning of a building structure. Examples for this are variations in movement, comfort level or safety. These requirements are inputs for architects, traffic engineering designers, etc.
3. **Performance requirements**
   They provide information on the expected performance of a structure. For example, they concern the embankment of pavements and are the basis for the structural engineer.
4. **Construction requirements**
   They relate to the behaviour of the structure, its sustainability, strength and stiffness, distortion, and are also part of the input for the designer.
5. **Building material requirements**
   They determine the choice of materials. These requirements apply to the planning engineer and the contractor.

6. **Raw material requirements**
   They relate to the raw materials comprising the various building materials. They are described in terms of tensile strength, maximum elongation or particle-size distribution.

3.5 Results

The results consist of a System Breakdown Structure (SBS)/Requirements Breakdown Structure (RBS) matrix and two types of process map elaborated through the consultation of documents related to the three case studies projects described previously in this chapter.

The SBS/RBS matrix is based on the information coming from all three case studies. On the other hand, the process maps are elaborated based only on the MAVO project (School). The reason to use this project is that from the building structure's projects, this is the only one that systems engineering was applied partially.

The SBS/RBS matrix is presented as a template for building structure projects that contains the possible building objects and requirements that are present in this type construction projects; and the applicability relation between requirements and objects. On the other hand, the resulting process maps are divided into two types: communication workflow and input/output workflow. Although both types contain the same information, the communication workflow is focused on which tasks are carried out by whom and when these tasks are happening during the design process. On the other hand, the input/output workflow shows thoroughly which documents are needed as an input for every task contained in the communication process map as well as which documents are the expected output coming from every task.

3.5.1 Requirements' System Breakdown Structure

The SBS/RBS matrix can be found in section 7.2 Appendix II: Requirements' System Breakdown Structure. The resulting schema after analyzing project elements and requirements in the defined case studies is a matrix that combines the tree structure of common building objects (SBS) and requirements (RBS) identified for Building Structure's projects. The matrix goes from a level of detail LOD200 to LOD300. Moreover, the "dots" in the matrix are used to show which requirements apply to which building objects.

Concerning the building objects (SBS), in the LOD 200 the "Building Structure" is defined as the main top level, which is divided into "1.Primary structure", "2.Secondary structure" and "3.Temporary structure". This level (LOD200) contains building objects that are normally connected with each other in the building structure and/or act like a system, such as the "1.1Building levels", the "1.2Façade" or the "1.3Staircase". For that reason, with regard to requirements, the requirement that applies to the building objects in the LOD200 is the distributed load bearing (F1), which is a functional requirement and could be represented as dead load, live load, snow load, wind load or an additional load. Specifically, all loads apply to all building objects in this level except for the wind load, which does not apply to "1.1Building levels" and the "1.3Staircase"; and snow load which does not apply for the "1.3Staircase" as well. This tree structure defined for the "Primary structure" applies for the "Secondary structure" and the "Temporary Structure" too.

As the level of detail increases, the LOD300 contains more building objects which are contained in the previously defined LOD200 building objects. In this way, a building object in a "1.1Building level" could be a "1.1.1Floor slab", "1.1.2Beam", "1.1.3Column", "1.1.4Structural wall", "1.1.5Footing" or "1.1.6Pile"; and inside the "1.2Façade", there is the "1.2.1Wind braces". Regarding the building object "1.3Staircase", requirements from LOD300 apply also to it, although it cannot be subdivided into a simpler building object.

The requirements that are applicable to the building objects in the LOD300 are loadbearing (Functional requirement), stability (Functional requirement), material quality (Object requirement), prevention (Aspect requirement), usability (Aspect requirement), and external and internal system interactions (System Interaction requirements). Concerning load bearing (F1), although it was also considered for LOD200 too, in LOD300 the load bearing requirements that are applied to building objects are represented by normal forces, shear forces and moments.

Stability (F2) is represented by wind load, earthquake load and second order deflection. Material quality (O1) is represented by strength and stiffness. Aspect requirements prevention (A2) and usability (A2) are represented by fire resistance, crash load; and deflection and crack width, respectively.

Finally, external system interactions (S1) are represented by ground water, soil and nuisance while the internal system interactions (S2) are represented by clashes and element connections.

With respect to the applicability of all these requirements to the different building objects in the LOD300, the requirements that apply to all objects contained in this level (LOD300) are normal force (F1); shear force (F1), strength (O1), stiffness (O1), nuisance (S1), clashes (S2) and element connections (S2).

On the other hand, moment (F1) is applicable to all building objects as well except for the "1.3Staircase". For the functional requirement stability (F2), wind load is applicable to "1.1.2Beam", "1.1.3Column", "1.1.4Structural wall", "1.2.1Wind braces" and "1.3Staircase"; earthquake load is applicable to all building objects except for the "1.3Staircase"; and second order deflection is applicable to all building objects except for the "1.1.5Footing", "1.1.6Pile" and "1.3Staircase". In the same way as second order deflection, prevention aspect requirements (A1) fire resistance and crash load are applicable to all building objects except for the "1.1.5Footing", "1.1.6Pile" and "1.3Staircase".

Concerning usability aspect requirement (A2), deflection requirement is applicable to all building objects except for the "1.1.5Footing", "1.1.6Pile"; while crack width is applicable to all building objects except for the "1.2.1Wind braces".

Lastly, external system interactions (S1) ground water and soil apply only to "1.1.5Footing" and "1.1.6Pile".

3.5.2 Process map: communication workflow

The communication workflow process map for the MAVO project (School) case study can be found in section 7.3 Appendix III: Process map MAVO project (Communication workflow). The process map is divided into two stages: preliminary design (VO) and final design (DO), being the last one where the project is located on time at the moment that this process map was created.

Regarding the VO, the tasks contained in the process, as well as the parties' responsibilities and their dependencies are explained next:

First of all, it is the client who defines the project requirements as a first task (Task 1). These requirements are gathered and formalised as different PDF files by the project manager (ARCADIS), who creates the “Programma van Eisen (PvE)” with all general, technical and process requirements of the project (Task 2). Once the PvE is ready, it is sent to the architect (Frencken Scholl) and the structural engineer (Pieters Bouwtechniek) so they can start creating the first drafts of the architectural model (LOD 200) and the starting points (LOD 200) respectively. During the elaboration of these tasks (Tasks 3 and 4), both architect and structural engineer are in contact with the external advisors, who are the ones elaborating advice reports like geotechnical, acoustic or ground studies (Task 5).

Next step after the architectural model and the starting points from the structural engineer are ready is that the project manager gathers the results coming from these two parties and carries out a verification of the requirements (Task 7). In case any of the resulting documents do not meet the requirements from the PvE, then the project manager will inform the liable party (architect or structural engineer) in order to edit and update their work according to their indications. Specifically, in this case study, the architects have also an internal checking before the check by the project manager is done where they compare the architectural model with the PvE originally received by the project manager (Task 6).

Finally, if the project manager confirms that all requirements are met, then the preliminary design is over and the final design stage starts. This confirmation is carried out by elaborating different reports that will contain the functional and technical verifications, and a deviation report to indicate how much the preliminary design differs from the original requirements.

Concerning the DO, for this case study, the parties involved in it were different from the ones that carried out the VO, so that means that there was a new project manager (SMT), architect (Vendev) and a **structural engineer (V&L)** in charge of continuing with the designs and making the calculations. For that reason, all documents generated during the VO were shared with the new parties responsible for executing the final design.

The first task of the DO was for the new project manager (SMT) to verify all documents coming from the previous phase (Task 1), and as a result, create a verification Excel sheet that was shared with the new architect (Vendev) and a structural engineer (V&L). This Excel sheet will represent the only document that is used for the implementation of systems engineering during the DO of the MAVO project.

Once the architect and the structural engineer receive this verification document, which they will use as a task list for their own work, they will start creating the new architectural model (LOD 300) and making the calculations (LOD 300) for the final design, respectively (Task 2 and 3). Both architect and a structural engineer will collaborate with external parties who will create the MEP installations and building physics design in the same way previous parties did during the VO (Task 4).

During the DO, the structural engineer will be in charge of creating a structural model as well. This structural model will be based on the architectural model (LOD 300) that the architect has elaborated in task 2 and the calculations they made (Task 3). The creation of this structural model (Task 5) involves an internal verification process (Task 7) of the model (clash detection) and the calculated requirements, which sets up the scenario for the thesis' scope. Basically, there is a verification/communication loop (Task 7) between the BIM modeller (the one creating the structural model) and the structural engineer (the one making the calculations) where information is exchanged through BCF zip files and face-to-face feedback, which represents a poor internal validation framework. For that reason, as it will be explained in the discussion of results, with the developed prototype it is aimed to improve the internal validation framework making it capable of keeping track of all modifications, amongst other functionalities.

At the same time, architects will also carry out an internal verification process in order to make sure that they meet the requirements (Task 6).

After structural and architectural models are finished, the project manager will be verifying the requirements (Task 8) in the same way it was done during the VO, and it will only after the project manager confirms that both architectural and structural engineer meet the requirements that the final design will be closed.

3.5.3 Process map: input/output workflow

The input/output workflow process map for the MAVO project (School) case study can be found in section 7.4 Appendix IV: Process map MAVO project (Input/output workflow). Basically, it contains the description of the same design process that was described in the communication workflow process map. The difference is that the importance this time lies on which documents are needed as an input for every task (activity column in the process map) and which documents are obtained as an output, while in the previous representation the importance was on tasks and the parties.

<u>Preliminary design (VO):</u>

For the task/activity 1 and 2, the input document will be the client's demands and the output the "Programma van Eisen (PvE)". It should be pointed out that client's demands are not necessarily a document, but also could be a meeting between the client and the project manager where the client informs what he wants to build.

For activities 3 and 4, which are the definition of the 2D drawings by the architect and the definition of the starting points by the structural engineer, the PvE will be the input document. The output will be the 2D drawings (LOD 200) and the starting points (LOD 200).

The previous output (2D drawings and starting points) is the input for activity 5, where the external parties elaborate the advice reports (output activity 5) that will be used again by the architect and structural engineer for activities 3 and 4 as input.

Concerning the architectural internal verification (activity 6), the input needed will be the PvE and the 2D drawings, and the output will be the comparison document where a deviation of the design regarding the original requirements is made. On the other hand, the verification of requirements carried out by the project manager (activity 7) will require the

2D drawings and the comparison document made by the architects; and the starting points made by the structural engineer. The output of this verification will contain PDF files with the technical and functional verification and a deviation report in relation to the original client requirements.

Final Design (DO):

As it was mentioned for the communication workflow, the parties involved in the final design are different from the ones that elaborated the preliminary design. Therefore, for the verification of the VO, all documents created in this stage will be the input for activity 1, obtaining as output a verification table (Excel sheet) and comments (PDF file) that will be used as task list by the new parties in charge of the DO.

The activities 2 and 3 will be using the verification table, the comments made by the project manager, the original 2D drawings and the starting points created in the VO. The output will be the architectural model (LOD 300) made by the architect and the calculations (LOD 300) made by the structural engineer in V&L.

Similarly to the VO workflow, the architectural model and calculations will be the input for the external parties in charge of creating the MEP installations model and the building physics (activity 4), whose output (MEP installations design and building physics) will be sent back to the architect and structural engineer as an input for the loop where the 3D and structural model are defined (activities 2 and 5).

Although both architect and structural engineer have an internal verification of the model in relation to the client requirements (activities 6 and 7), it is the structural verification (activity 7) where the scope of this thesis will focus its attention. This internal verification of the structural model is happening between the BIM modeler and the structural engineer, and it creates a communication loop where BCF zip files, which contain comments about issues that were found in the model by the structural engineer, are exchanged as output for the verification (activity 7) and input for the definition of the structural model (activity 5).

Once the architectural, structural and integrated model are finished, then, the project manager, also by making use of the initial verification table, will check whether all models comply with the requirements (activity 8), creating as final output the final design model and closing the final design stage.

3.6 Discussion

Firstly, creating the SBS/RBS matrix resulted in gaining insight about which building elements and requirements are usually contained in building structures and civil projects. By inspecting project documentation (mainly calculations and advisory reports) it was possible to compare which similarities and differences existed between the same building elements and how they were calculated and checked against requirements. Moreover, the feedback coming from the personnel in V&L was really valuable in order to understand which requirements were relevant for the matrix as well as to which elements they were applicable to.

It should be pointed out that, despite inspecting thoroughly three different construction projects and interviewing engineers in V&L, the obtained matrix is far from being considered as a final or a universal template that will be applicable to new upcoming projects in the building structures division of V&L. The stated applicability relationships between building objects and requirements should be considered dynamic and never static. Moreover, the list of building objects and requirements included are based on 3 different projects, so these lists can be improved with further research on more projects.

On the other hand, the elaborated matrix sets a helpful template of identified common building objects and requirements in Building Structures projects that, when implemented in an actual project, could be used to create an initial list/overview of building objects and requirements. In addition, this matrix could be the starting point for facing important ideas mentioned during the literature review such as the fact that current design tools do not support recording of client's requirements (Gu & London, 2010), the fact that there is no connection between requirements and design documents (Arto Kiviniemi & Martin Fischer, 2004), to reinterpret fuzzy requirements provided in natural language into something formally understood (Krijnen & van Berlo, 2016) and to capture requirements patterns, used relationships and their evolution during the design process (Ozkaya & Akin, 2007).

The truth is that by elaborating this matrix in the beginning of every project, engineers will count on a document that connects requirements and building objects during the design phase and therefore supports the recording of client's requirements. Moreover, as soon as this task is integrated into the design process of the company, requirements patterns and relationships will become easily recognisable with every different project, as well as the reinterpretation of fuzzy requirements coming from the client.

Secondly, after carrying out the analysis of the Preliminary Design (VO) and the Final Design (DO) design processes for the MAVO project (School), including the representation of both communication and input/output workflow as process maps; a better understanding of how parties involved in the design process collaborate and share information has been obtained. Moreover, concerning the scope of this thesis, after elaborating these process maps it was easier to identify the actors and information involved in the internal verification process (tasks 5 and 7) that the structural engineer makes for the structural model (Jeff Wix & Jan Karlshøj, 2010).

As it is explained in the description of the results, the verification loop in V&L occurs between the BIM modeler (in charge of defining the structural model), and the structural engineer, who makes the calculations of the structure and the one in charge of verifying that the structural model complies with the calculations and, therefore, client's requirements.

It was observed during the research stage at the company that for this verification, only face-to-face feedback, 2D sketches or BCF zip files containing simple comments were exchanged between the BIM modeller and the structural engineer. This behaviour, which was mentioned in the literature review by several authors (L. van Berlo et al., 2015; Gu & London, 2010; W. Shen et al., 2010), is justified by the authors due to the lack of trust on completeness and accuracy of 3D models. Nevertheless, it is believed that for engineers in V&L, the fact that their desks in the office are next to each other makes it easier for them to discuss face-to-face if something in the model needs to be changed, rather than creating a BCF or make an official request about it.

Nevertheless, although this way of interaction could be simple and effective in the office environment, the truth is that all these changes are not being registered in any log that can be consulted for future validation. What is more, these changes are justified only by the knowledge of the structural engineer concerning the client requirements. It is only because the structural engineer has previously consulted the verification table (Excel sheet), provided by the project manager, that he knows how the structural model should be and why changes need to be done.

The analysis of the design process has helped not only to identify which are the flaws of the current performance but also to know which parts of the process we want to maintain and which ones should be removed and/or improved.

To begin with, the possibility for all decisions to be registered during the design process with respect to the structural model is something to take into account. If every element has an edition history, it would be possible to track the design changes and evolution of requirements in the model.

Secondly, BCF is a well-known and used tool for informal communication in the design process; however, it is mainly used for exchanging simple comments because other features, for instance referencing documents, are not provided by any of the software tools that support this technology (Treldal et al., 2016). Moreover, although file exchange has been overcome by web-based BCF management tools that synchronize BCF issues without having to export BCF ZIP files (“BIMcollab,” 2017; Linhard, K. & Steinmann, R., 2015; L. van Berlo & Krijnen, 2014), the stored BCF data still requires human interpretation when filtering or retrieving the BCF information. In such way, when projects that could generate large amounts of issues are considered, the management of this information could become an overwhelming task.

Last but not least, the way IFC models are validated nowadays using BCF is based on the BCF issues that have been previously created for the IFC model. In other words, elements are checked because BCF issues were created about them. So it could be that objects without any BCF information have been checked but no remark was necessary to be done, or on the contrary, they have not been properly checked. Consequently, an opportunity to mark objects “as closed” would provide the structural engineer with a useful functionality for validation of objects and the possibility to enrich the information that the model can offer.

Finally, addressing the discussed points will be the objective of this thesis through the development of a tool (described in section 4. Tool development) that aims to provide a solution to the identified functionalities by combining BCF with Semantic Web technologies.

4. Tool development

4.1 Introduction

This chapter deals with the description of the developed tool combines BCF and Semantic Web technologies for the purpose of internal validation attestation.

The application, which has been programmed using Python, connects the informal communication provided by the BCF with the Semantic Web technologies. In this way, the user is able to create BCF issues for a given IFC model within an RDF environment. Specifically, the application allows the user to create BCF issues in the same way it is possible with other software tools (like SMC or Tekla), but with the advantage that information does not require its serialization as BCF zip files. Instead, while keeping the same structure that the original BCF schema has, the data is stored as an RDF context that is uploaded to an RDF repository. The repository works as a SPARQL endpoint, and all the created BCF information is stored and retrieved directly from this web repository through the desktop application by using SPARQL queries.

Moreover, the application also provides backwards and external compatibility with the BCF Manager from KUBUS so that the information created through it is compatible with current BCF management tools. In such manner, it is possible to serialize the RDF contexts previously created for a given model as the "old" BCF zip files, read them in the BCF Manager, edit them, upload them back in the repository in order to update the information in the SPARQL endpoint, and visualize those changes/updates in the desktop viewer as a result.

This chapter also contains a validation of the generated data and data visualization, limitations analysis and a final discussion of the results achieved with recommendations for further developments or upgrades.

4.2 Case study

The project that has been used as case study once the tool is developed is the **Nieuwbouw MAVO Schravenlant XL te Schiedam (School),** which is described in section 3.3.1 Case studies. The school is located in Schiedam, near the Burgemeester of Haarenlaan and one side of the new building borders the water (Figure 19).

The elements of the IFC model of the project (Figure 20), which was provided by V&L, are used for the creation of BCF information within an RDF environment, the visualization of different BCF status and, on the whole, to run tests for the functionalities that have been developed.

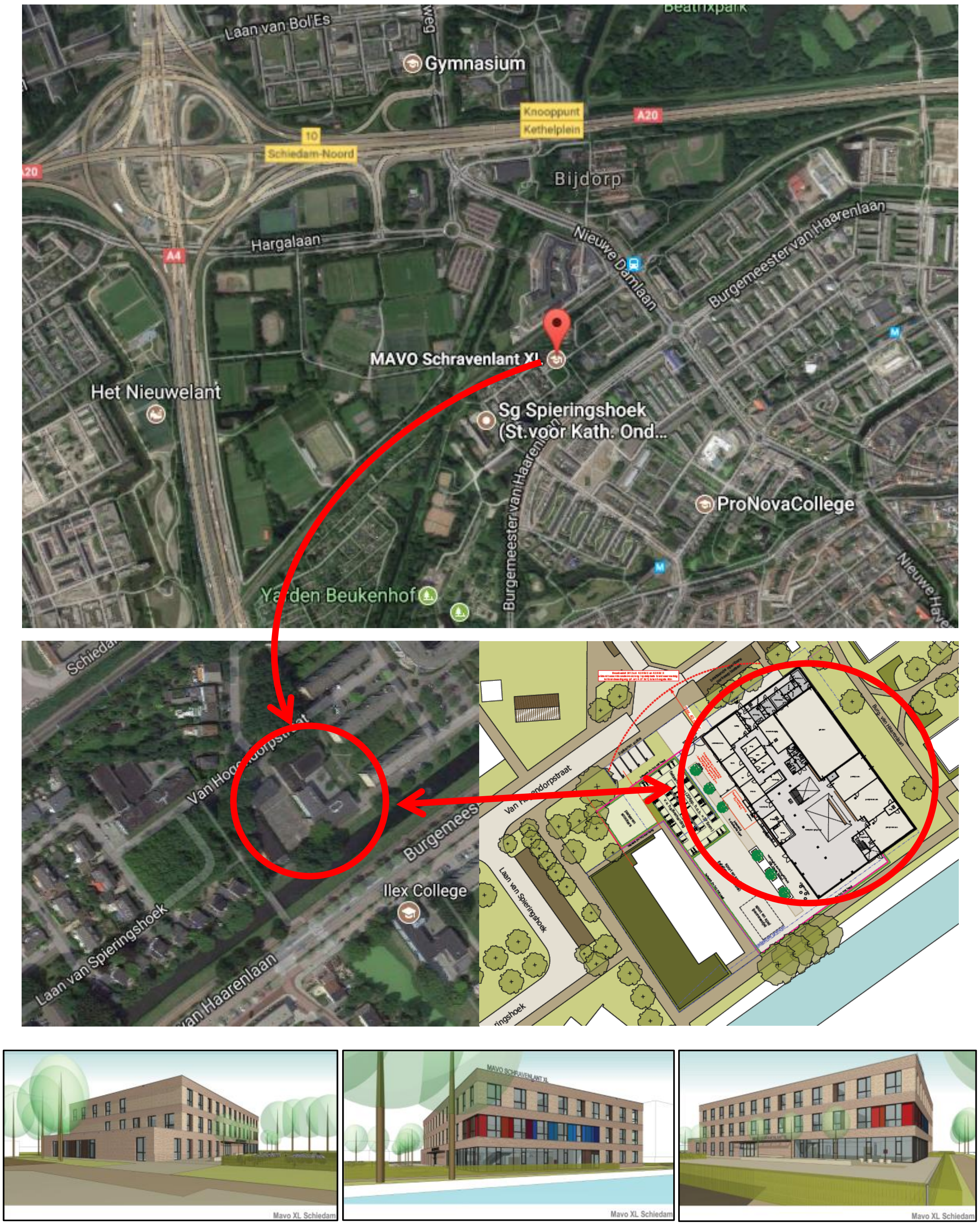


Figure 19. Location of the MAVO XL Schiedam (Source: Google Maps and V&L documentation)

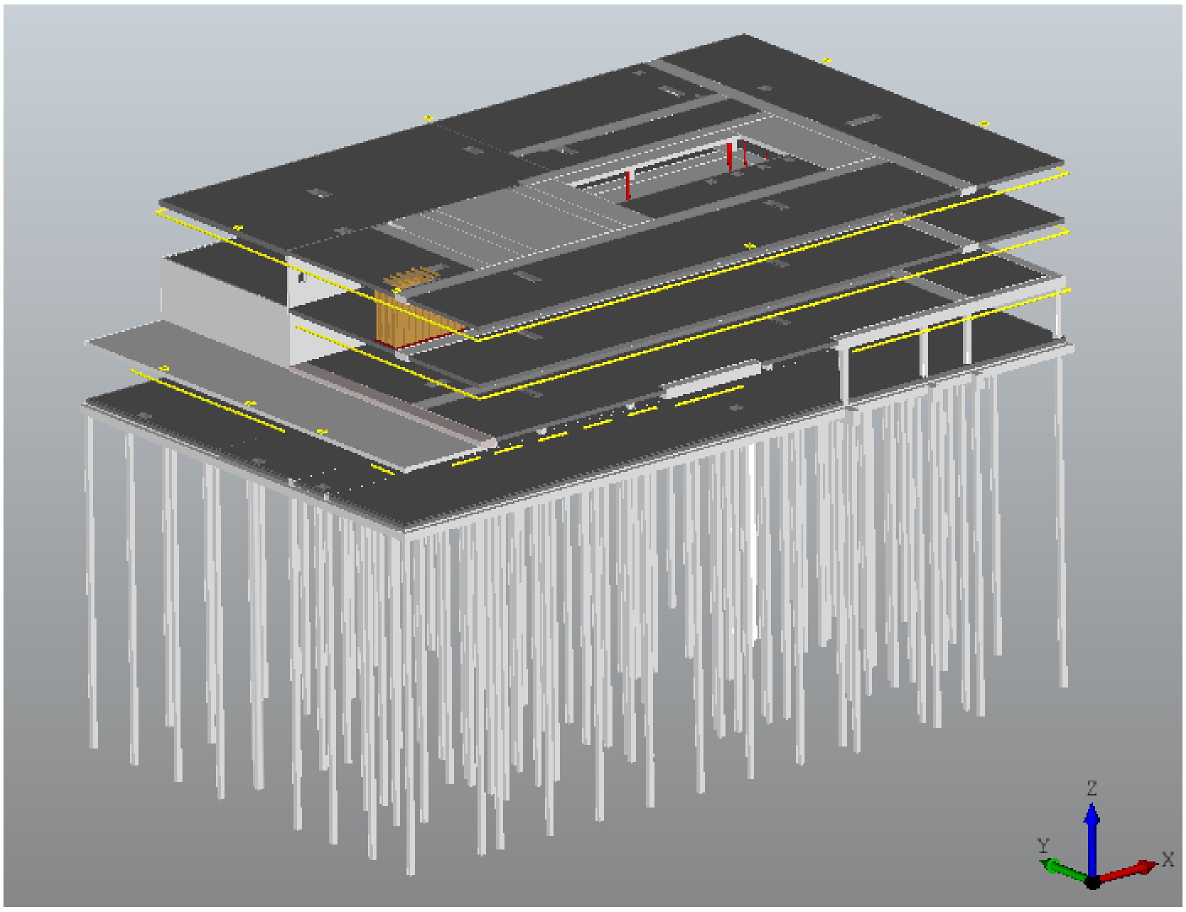


Figure 20. Structural model of the MAVO XL Schiedam project (IFC, Source: V&L)

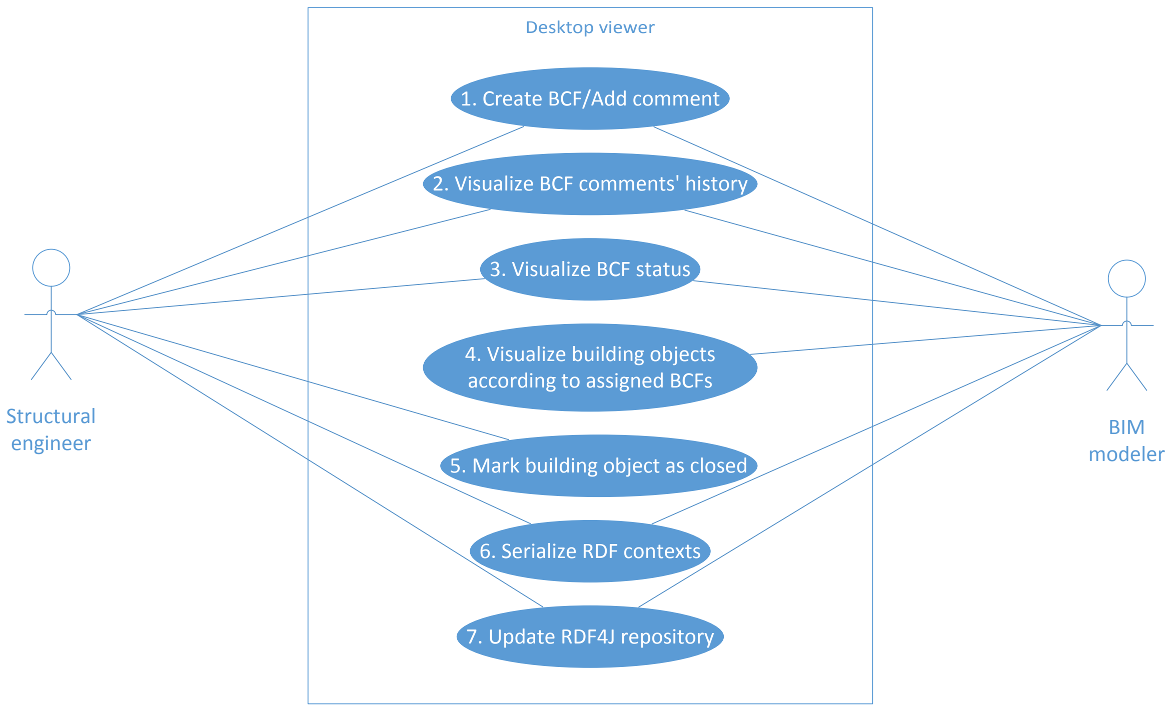
4.3 Use case diagram

The discussion in the methodology (section 3.6 Discussion) pointed out some important ideas with respect to the functionalities that the developed tool should be able to provide to the actors involved in the internal validation of the structural model. For that reason, based on those points and feedback coming from the supervisors, the list in Table 2 enumerates the functionalities that the developed tool should be able to address.

Table 2. Functionalities of the developed tool

Nº	Functionalities
1	Create BCF/Add comment (Create BCF information as RDF contexts)
2	Visualize BCF comments' history (Retrieve BCF information)
3	Visualize BCF status (Retrieve BCF information)
4	Visualize building objects according to assigned BCFs (Retrieve BCF information)
5	Mark building objects "as closed" using BCF (Create BCF information as RDF contexts)
6	Serialize RDF contexts as "old" BCF zip files (Backwards compatibility)
7	Upload BCF zip files and update RDF contexts in the repository (External compatibility)

In order to capture the interaction of the engineers with the developed tool during the internal design validation process, a use case diagram (Figure 21) and three workflow examples in section 7.8 Appendix VIII: Flowcharts of the developed tool (Figure 70) are also elaborated. Apart from the initial process map (Figure 3) presented in 1.1 Problem definition/objective of the thesis.



Desktop viewer
1. Create BCF/Add comment
2. Visualize BCF comments' history
3. Visualize BCF status
4. Visualize building objects according to assigned BCFs
5. Mark building object as closed
6. Serialize RDF contexts
7. Update RDF4J repository
Structural engineer
BIM modeler

Figure 21. Use case diagram of the developed tool

The use case diagram presents two actors involved: the structural engineer and the BIM modeller; and the functionalities (Table 2) that each of them will be able to carry out through the desktop viewer. The three workflow examples that both actors will be able to perform when using the desktop viewer for internal validation are explained next.

The first workflow example presents a case where the structural engineer is revising the structural model created by the BIM modeller for the first time. The first task, once the IFC model is loaded into the viewer, is to revise the model elements (or building objects) that the structural model contains in order to check whether the building objects comply with the project requirements regarding every building aspect (dimensions, properties, clashes…). In case the structural engineer notices an error in a building object, he will use the desktop viewer to create a BCF issue with the necessary information, so that the BIM modeller understands what is meant to be changed, that will be automatically uploaded and serialized as an RDF context in the RDF repository. On the other hand, if the inspected building object has no remarks to be made, then the structural engineer will mark the building object as closed, creating a “closure” BCF. The “closure” BCF will be created as an RDF context in the RDF repository in the same way the normal BCF issue was created before, but the information of the BCF will indicate that the building object has been checked and, therefore, the BCF status is closed (Author, date and comment will be also specified).

The second workflow example represents the step after the structural engineer has finished revising the structural model. In this workflow example, the BIM modeller will load the same IFC file containing the structural model, and the first task will be to visualize the BCF issues that have been assigned to him by the structural engineer. By introducing his name after clicking the “Show assigned objects” button, the desktop viewer will show only the building objects of the structural model that have BCF information that is meant to be solved specifically by him. The second task will be to visualize the BCF comments history and status by clicking on the “Check object status” button. Once the BIM modeller understands what needs to be changed, the BIM modeller will update the structural model (Task 3) in an appropriate the design software (Revit or Tekla) and will use the desktop viewer to add a comment (Task 4) indicating that the issue is solved. The comment will be added to the existing RDF context created previously by the structural engineer by selecting the building object again in the desktop viewer and clicking the “Create new BCF / RDF” button.

Finally, third workflow example brings back the structural engineer as the main actor, who will be revising the structural model again in order to check that all BCF issues he created on his first revision have been solved by the assigned engineers. To that end, the structural engineer will be visualizing the BCF comments’ history of the updated building objects and he will check them against the project requirements. In case the building object does still not comply with the requirements after the changes, then he will be adding a new comment to the existing RDF context so that the BIM modeller corrects it. On the other hand, if the building object is approved, then the structural engineer will mark it “as closed” in order to close the BCF status and validate the building object.

The three workflow examples could be used as a loop that the users should follow until all building objects in the BIM model are marked as closed, which means that the model is validated.

4.4 Ontology: BCF schema

The ontology for the BCF schema was created as the template that all BCF issues generated through the developed tool will follow when modelling the BCF information in an RDF format. The original XSD Schema for the BCF specification on which the ontology is based can be found in the appendix, specifically in section 7.7 Appendix VII: BCF Schema (XSD), as well as the developed ontology for the BCF schema, which can be found in section 7.6 Appendix VI: Ontology BCF schema (TTL).

The BCF ontology was created using TopBraid Composer software tool. In order to translate the XSD schema’s elements and attributes; classes, sub-classes and properties (data type and object type) were created and serialized in a turtle (TTL) file. Although most of the papers discussed in the literature review provide an overview about the status in the AEC field concerning representation of IFC models as ontologies and the use of Semantic Web technologies to connect data and improve interoperability, other authors (Natalya F. Noy & Deborah L. McGuinness, 2001) and specifically the chapters 3, 5 and 7 of the book “*Semantic Web for the Working Ontologist”* (Allemang & Hendler, 2011b, 2011a, 2011c) were useful for the practical development of the ontology.

4.4.1 Describing the ontology

<u>Classes</u>

- fbf:Markup
- fbf:Header
- fbf:File
- fbf:Topic
- fbf:Comment
- fbf:Viewpoints
- fbf:BimSnippet
- fbf:Document Reference
- fbf:Related Topic
- fbf:Viewpoint

<u>Properties</u>

Owl:ObjectProperty

- fbf:containsHeader
- fbf:containsFile
- fbf:containsTopic
- fbf:containsComment
- fbf:containsViewpoints
- fbf:containsViewpoint
- fbf:containsRelatedTopic
- fbf:containsDocumentReference
- fbf:containsBimSnippet

Owl:Datatype Property

- fbf:hasAssignedTo
- fbf:hasAuthor
- fbf:hasBimSnippetIsExternal
- fbf:hasBimSnippetReference
- fbf:hasComment
- fbf:hasCommentDate
- fbf:hasCommentGuid
- fbf:hasCommentModifiedAuthor
- fbf:hasCommentModifiedDate
- fbf:hasCreationAuthor
- fbf:hasCreationDate
- fbf:hasDocumentReferencedDescription
- fbf:hasDocumentReferenceGuid
- fbf:hasDocumentReferenceIsExternal
- fbf:hasDueDate
- fbf:hasFileDate
- fbf:hasFileIsExternal

- fbf:hasFileName
- fbf:hasFileReference
- fbf:hasIfcProject
- fbf:hasIfcSpatialStructureElement
- fbf:hasLabels
- fbf:hasModifiedAuthor
- fbf:hasModifiedDate
- fbf:hasPriority
- fbf:hasReferencedDocument
- fbf:hasReferenceLink
- fbf:hasReferenceSchema
- fbf:hasRelatedTopicGuid
- fbf:hasSnapshot
- fbf:hasSnippetType
- fbf:hasStage
- fbf:hasTitle
- fbf:hasTopicDescription
- fbf:hasTopicGuid
- fbf:hasTopicIndex
- fbf:hasTopicStatus
- fbf:hasTopicType
- fbf:hasViewpoint
- fbf:hasViewpointGuid
- fbf:hasViewpointsGuid
- fbf:hasViewpointsIndex

The domain and ranges of all properties can be visualized at the next figures (Figure 22, Figure 23, Figure 24, Figure 25, Figure 26, Figure 27, Figure 28 and Figure 29), where the relation between the different classes is described.

Markup is the main element of the BCF issue; it contains a **Header**, a **Topic**, a **Comment** and a **Viewpoints** (Figure 22). One markup is related to one Header and Topic, but it can have more than one comment or viewpoint attached in the same issue (Markup).

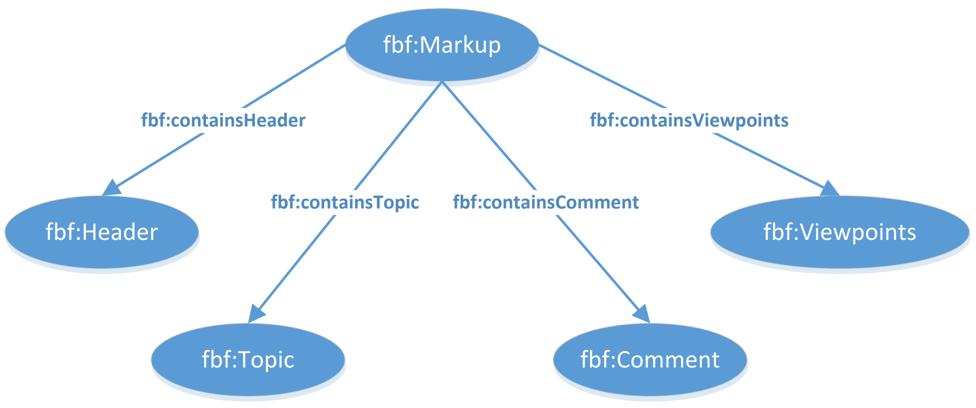


Figure 22. RDF schema – Part 1

The **Header** contains a **File**, which has as "Datatype Properties" the next attributes: **FileName, FileDate, FileReference, Ifcproject, IfcSpatialStructureElement** and **FileIsExternal** (Figure 23).

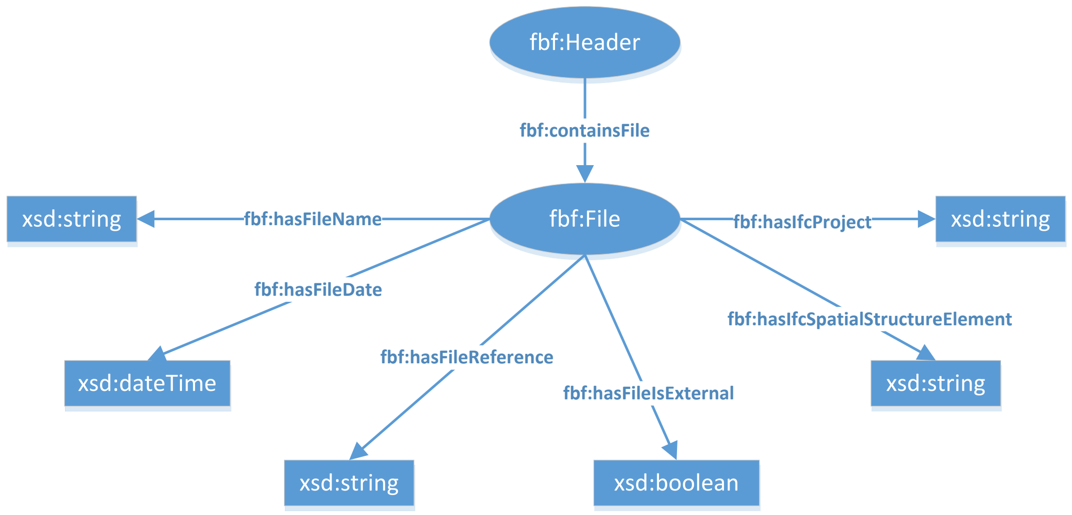


Figure 23. RDF schema – Part 2

The **Viewpoints** contain the following "Datatype Properties": **Snapshot**, **Viewpoint**, **ViewpointIndex** and **ViewpointGuid** (Figure 24).

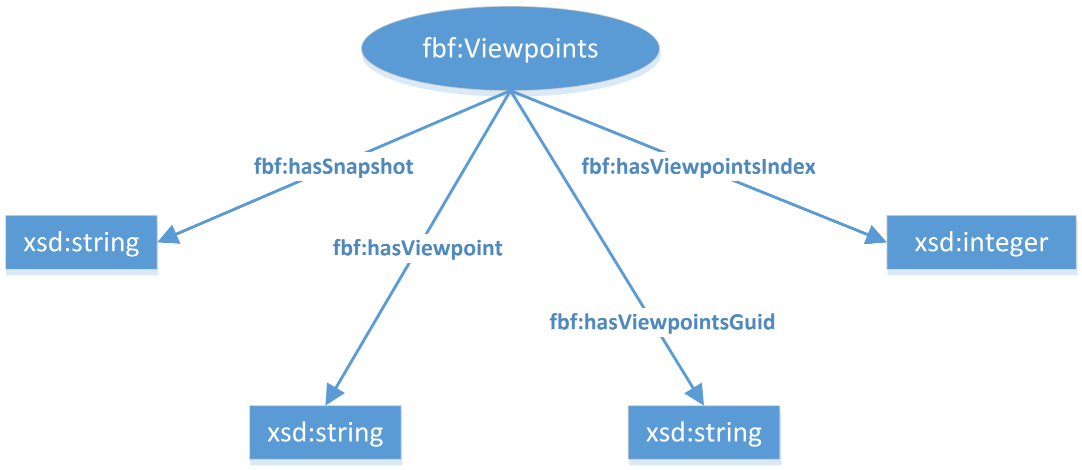


Figure 24. RDF Schema – Part 3

The **Comment** contains as "DataType Properties" a **CommentDate**, a **CommentModifiedAuthor**, a **CommentGuid**, an **Author**, a **CommentModifiedDate** and a **Comment**. In addition, it also contains the **class Viewpoint,** which contains a **ViewpointGuid** as DataType Property too (Figure 25).

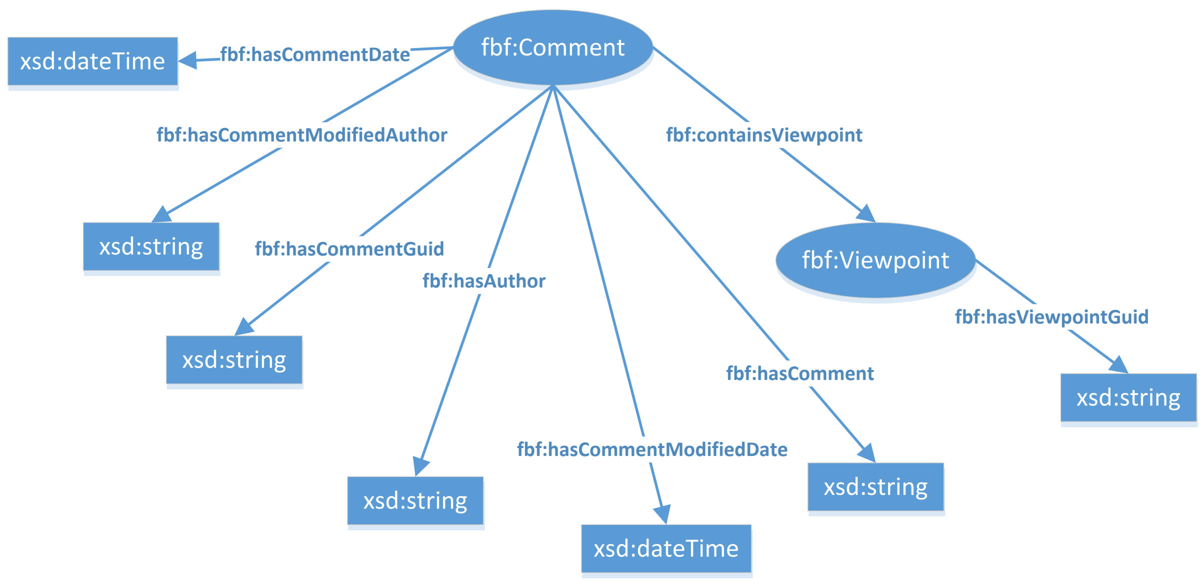


Figure 25. RDF Schema – Part 4

The **Topic** contains several classes apart from DataType Properties. These classes also contain other DataType Properties. All classes, properties and their hierarchical relations (Figure 26, Figure 27, Figure 28 and Figure 29) are shown next:

- AssignedTo
- CreationAuthor
- CreationDate
- DueDate
- Labels
- ModifiedAuthor
- ModifiedDate
- Priority
- ReferenceLink
- Stage
- Title
- TopicDescription
- TopicGuid
- TopicIndex
- TopicStatus
- TopicType
- **BimSnippet**
    - BimSnippetIsExternal
    - BimSnippetReference
    - ReferenceSchema
    - SnippetType

- **Document Reference**
    - DocumentReferenceDescription
    - DocumentReferenceGuid
    - DocumentReferenceIsExternal
    - ReferencedDocument
- **RelatedTopic**
    - RelatedTopicGuid

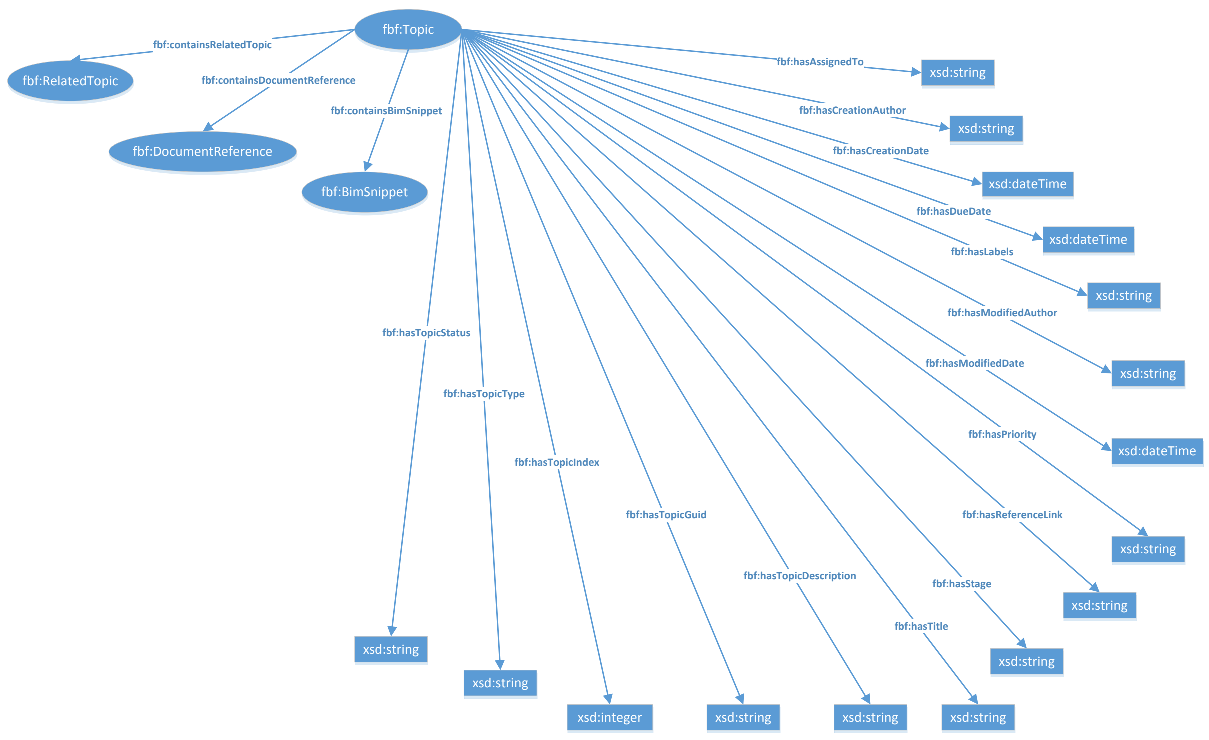


Figure 26. RDF Schema – Part 5.1

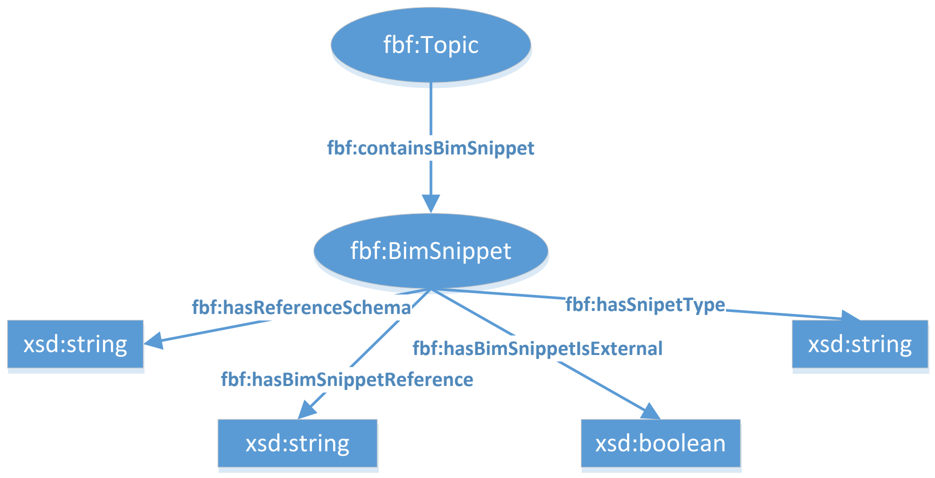


Figure 27. RDF Schema – Part 5.2

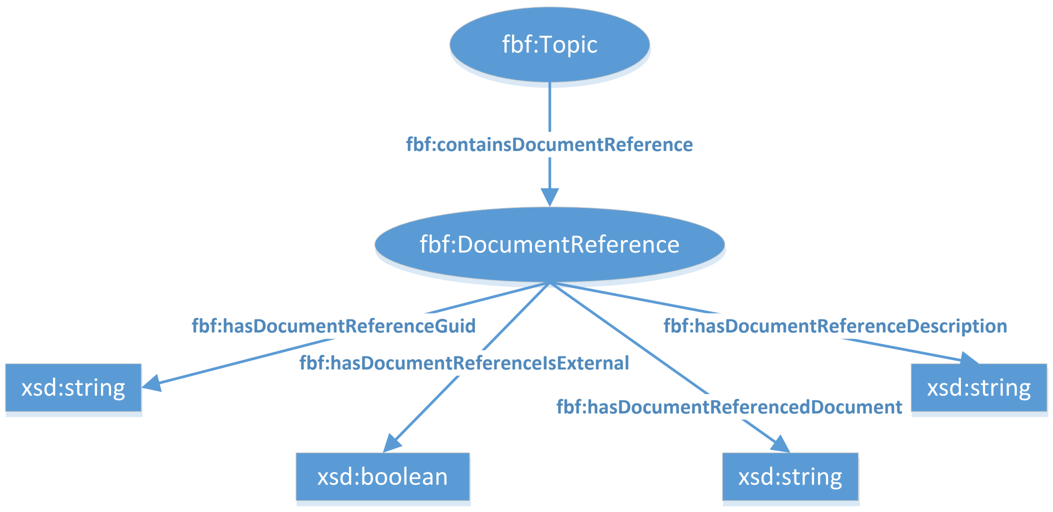

fbf:Topic
fbf:containsDocumentReference
fbf:DocumentReference
fbf:hasDocumentReferenceGuid
fbf:hasDocumentReferenceIsExternal
fbf:hasDocumentReferenceDescription
fbf:hasDocumentReferencedDocument
xsd:string
xsd:boolean
xsd:string
xsd:string

Figure 28. RDF Schema – Part 5.3

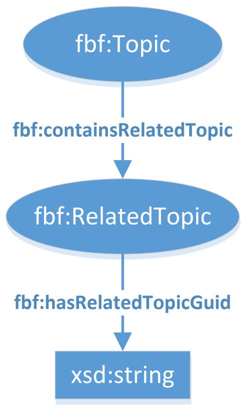


Figure 29. RDF Schema – Part 5.4

4.5 Prototype development

The functionalities and workflow examples that the developed tool provides have been previously explained. However, how these functionalities are actually implemented and how the information is managed within the RDF environment is explained in depth in this section while showing the interface's layout and how the modelled information looks like. Moreover, the complete Python code for the developed tool can be found in section 7.5 Appendix V: Application code (Python).

The layout of the interface is divided into two tabs. The first tab is called "3D viewer" (Figure 30) and it contains the necessary buttons to carry out the functionalities 1 to 5 from Table 2. On the other hand, tab 2 is called "Backwards and external compatibility" (Figure 31) and it contains the necessary buttons to carry out functionalities 6 and 7 from Table 2.

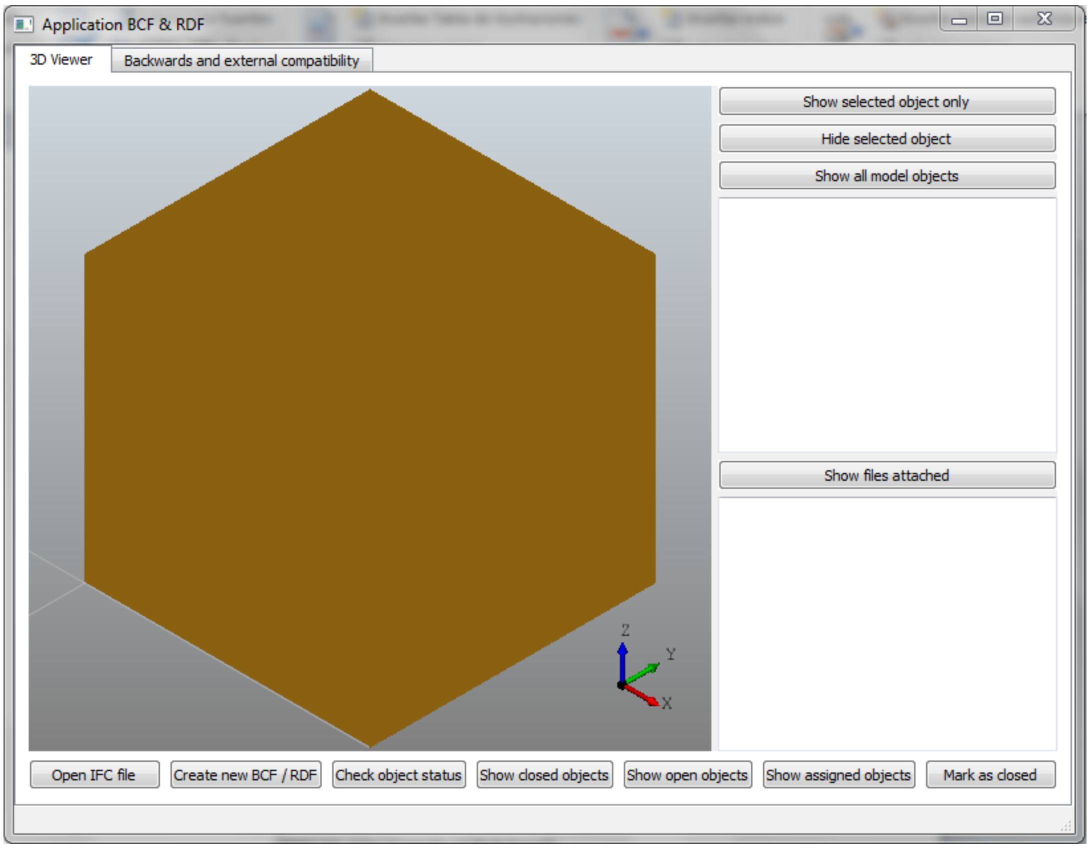
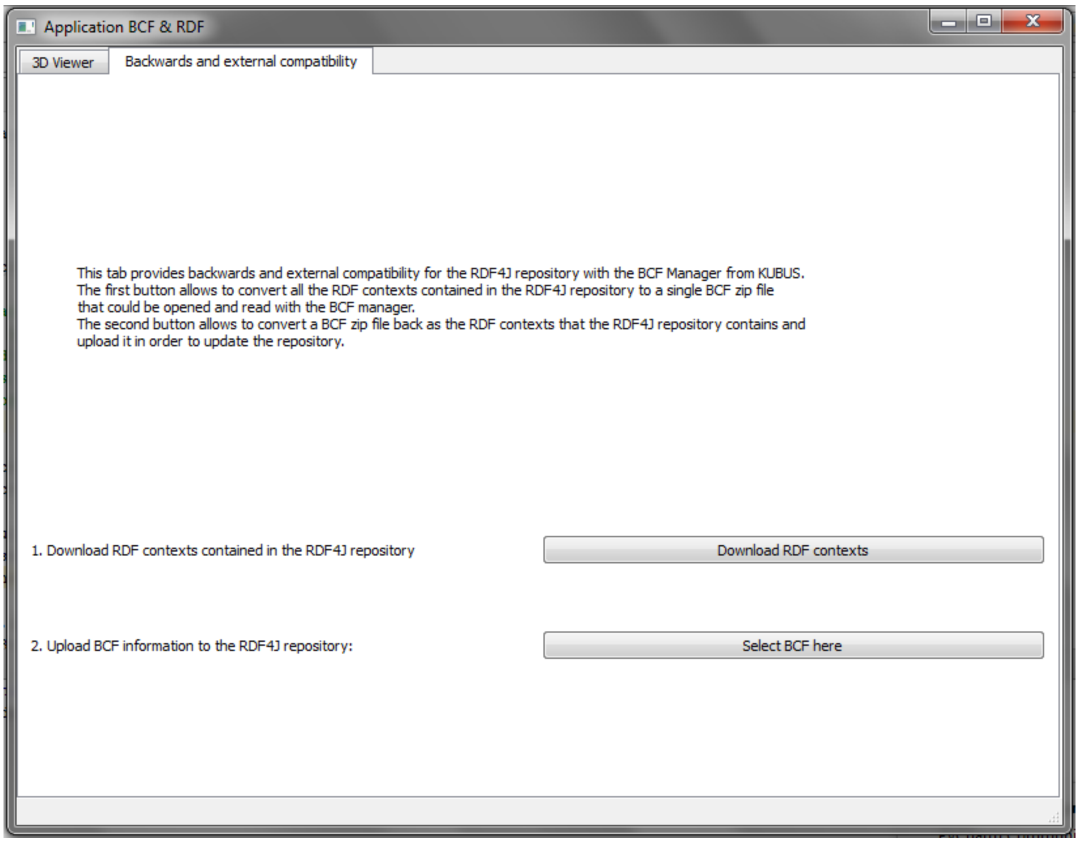


Figure 30. 3D viewer tab



This tab provides backwards and external compatibility for the RDF4J repository with the BCF Manager from KUBUS.
The first button allows to convert all the RDF contexts contained in the RDF4J repository to a single BCF zip file that could be opened and read with the BCF manager.
The second button allows to convert a BCF zip file back as the RDF contexts that the RDF4J repository contains and upload it in order to update the repository.

1. Download RDF contexts contained in the RDF4J repository

Download RDF contexts

2. Upload BCF information to the RDF4J repository:

Select BCF here

Figure 31. Backwards and external compatibility tab

4.5.1 Tab 1: 3D viewer

This tab provides the user with all functionalities related to the creation and visualization of BCF information with respect to the structural model. The buttons that can be found in it are listed next:

1. Open IFC file
2. Create new BCF / RDF
3. Check object status
4. Show closed objects
5. Show open objects
6. Show assigned objects
7. Mark as closed
8. Show selected object only
9. Hide selected object
10. Show all model objects
11. Show files attached

After loading the IFC model by clicking on the open IFC button (1), the viewer shows the selected IFC model (Figure 32).

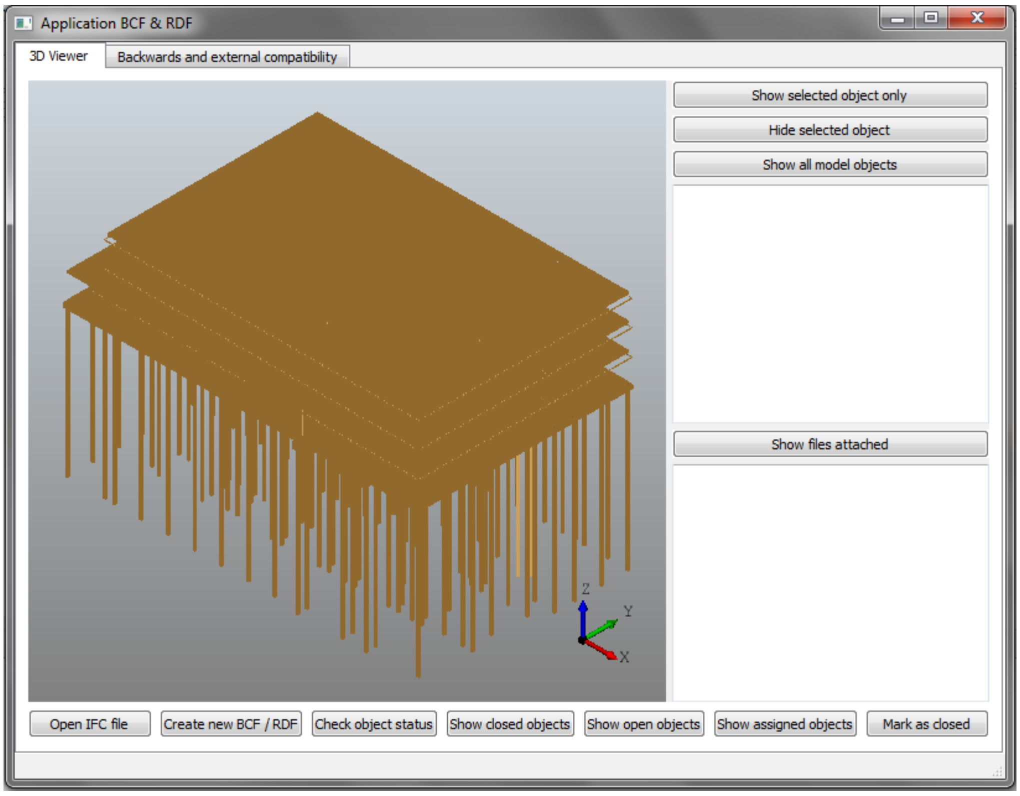
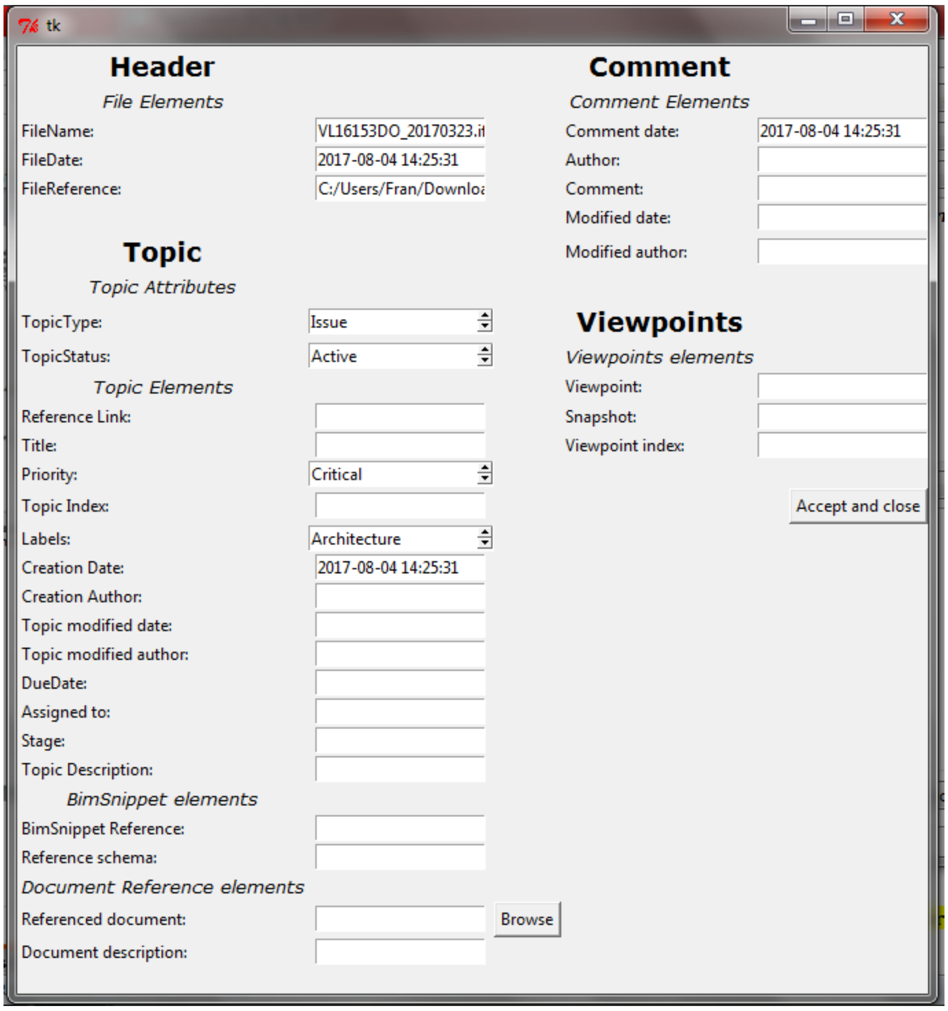


Figure 32. 3D viewer tab – Loaded model

In case the user would like to create a BCF issue about a building object, by clicking the “Create new BCF / RDF” button (2) the BCF interface appears (Figure 33). Once the necessary information has been filled in with the entry boxes from the BCF interface, the RDF context containing the filled BCF information is created in the RDF repository (Figure 34) after clicking on the “Accept and close” button.



Header

File Elements

FileName:	VL16153DO_20170323.i
FileDate:	2017-08-04 14:25:31
FileReference:	C:/Users/Fran/Downlo

Topic

Topic Attributes

TopicType:	Issue
TopicStatus:	Active

Topic Elements

Reference Link:	
Title:	
Priority:	Critical
Topic Index:	
Labels:	Architecture
Creation Date:	2017-08-04 14:25:31
Creation Author:	
Topic modified date:	
Topic modified author:	
DueDate:	
Assigned to:	
Stage:	
Topic Description:	

BimSnippet elements

BimSnippet Reference:	
Reference schema:	

Document Reference elements

Referenced document:	
Document description:	

Comment

Comment Elements

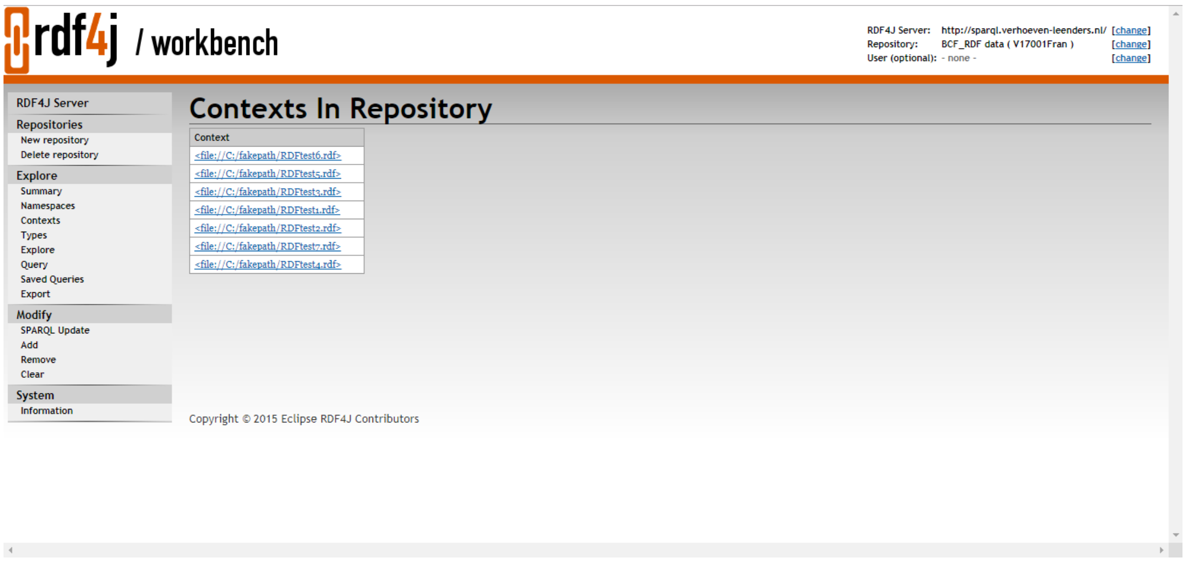
Comment date:	2017-08-04 14:25:31
Author:	
Comment:	
Modified date:	
Modified author:	

Viewpoints

Viewpoints elements

Viewpoint:	
Snapshot:	
Viewpoint index:	

Figure 33. BCF creator interface



RDF4J Server:	http://sparql.verhoeven-leenders.nl/	[change]
Repository:	BCF_RDF data (V17001Fran)	[change]
User (optional):	- none -	[change]

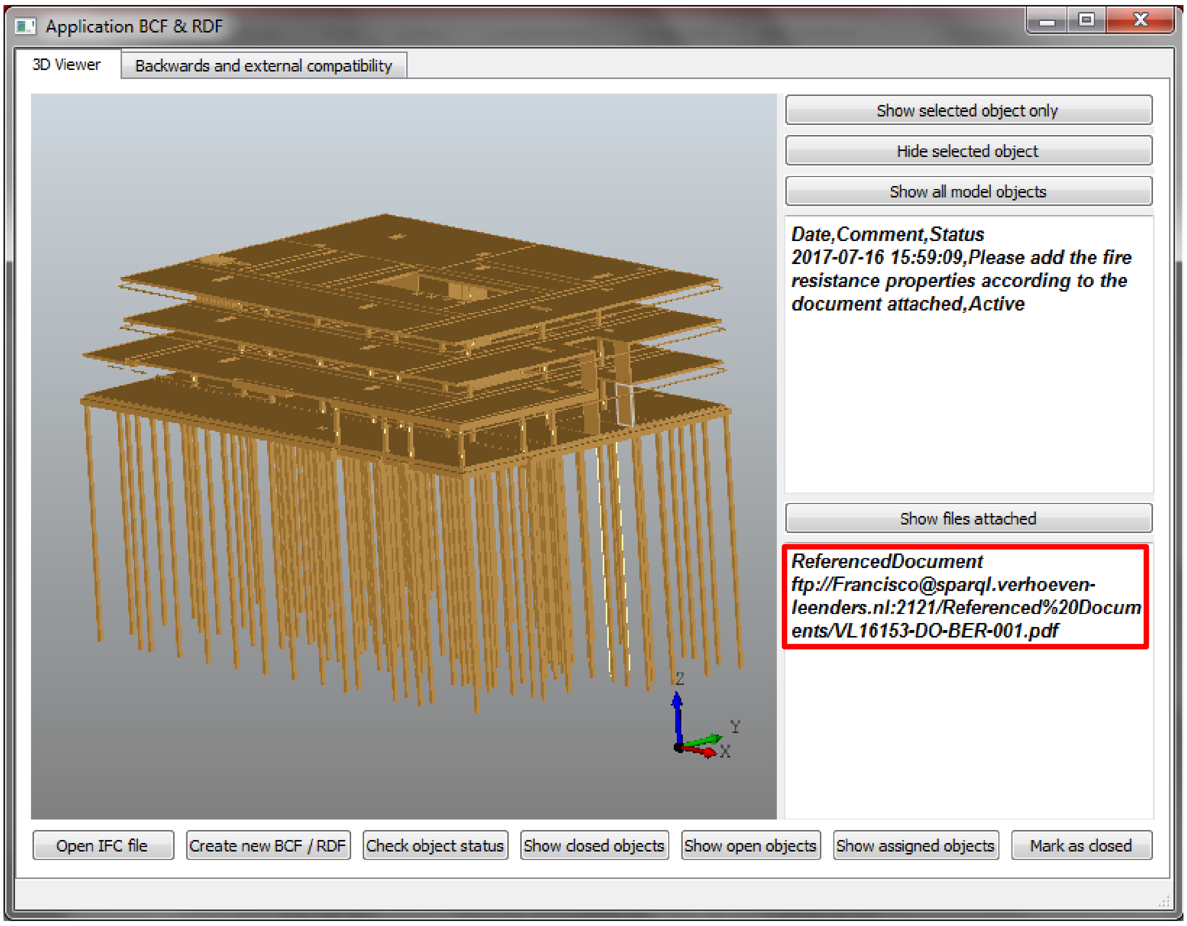
Contexts In Repository

Context
<file://C:/fakepath/RDFtest6.rdf>
<file://C:/fakepath/RDFtest5.rdf>
<file://C:/fakepath/RDFtest3.rdf>
<file://C:/fakepath/RDFtest1.rdf>
<file://C:/fakepath/RDFtest2.rdf>
<file://C:/fakepath/RDFtest7.rdf>
<file://C:/fakepath/RDFtest4.rdf>

Copyright © 2015 Eclipse RDF4J Contributors

Figure 34. List of RDF contexts in the RDF4J repository

It should be pointed out that the BCF interface (Figure 33) includes parts of the BCF specification that are not supported by other model checking or BCF management softwares (such as Solibri or BCF Manager). These parts are, for instance, the “Referenced Document” which allows the user to attach a scanned sketch or a PDF calculation that will be stored in the FTP-server of the company. In this way, links to explanatory documents (Figure 35) can be obtained by clicking the “Show files attached” button (11), so that these kinds of documents are being stored and retrieved in/from the company server automatically and users don’t have to spend time looking for them.

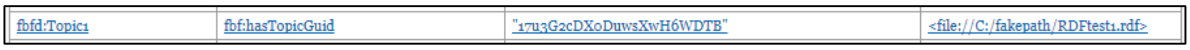


Date,Comment,Status
2017-07-16 15:59:09,Please add the fire resistance properties according to the document attached,Active

ReferencedDocument
ftp://Francisco@sparql.verhoeven-leenders.nl:2121/Referenced%20Documents/VL16153-DO-BER-001.pdf

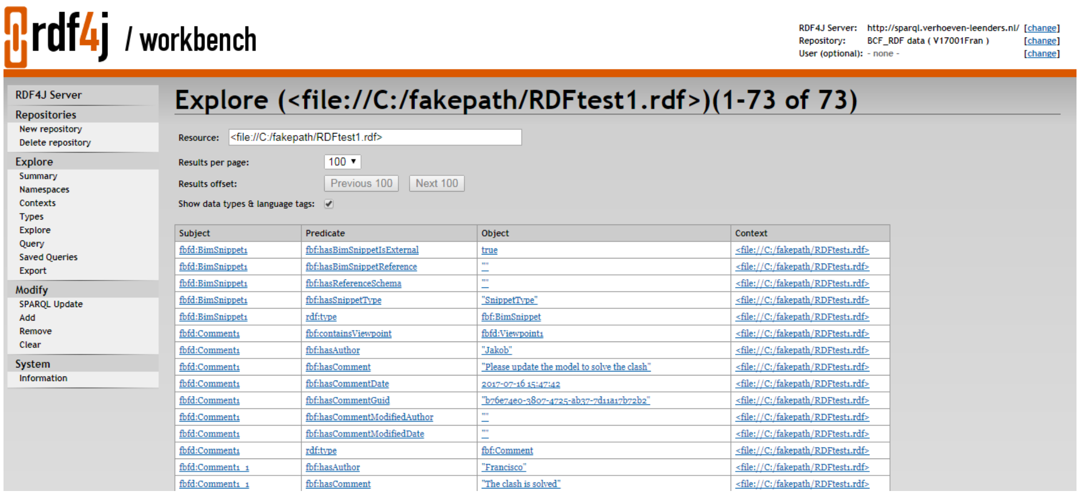
Figure 35. Show files attached button example

A closer look into the RDF contexts contained in the RDF repository shows how all the information that has been introduced in the BCF creator interface has been stored as RDF triples. For instance, the RDF context 1, which can be seen in Figure 37, lists a total of 73 triples that contain the BCF information of a building element. The structure follows the RDF schema introduced in section 4.4 Ontology: BCF schema, which also originates and is based on the knowledge discussed by the authors mentioned in the literature review (section 2.6 Semantic Web). Moreover, the key to link all these triples to a single building element is the GUID (J. Beetz et al., 2014), which is specified in the Topic of the BCF (Figure 36).



fbfd:Topic1	fbf:hasTopicGuid	"17u3G2cDXoDuwsXwH6WDTB"	<file://C:/fakepath/RDFtest1.rdf>

Figure 36. Triple containing the Topic GUID of the RDF context 1



RDF4J Server: http://sparql.verhoeven-leenders.nl/ [change]
Repository: BCF_RDF data (V17001Fran) [change]
User (optional): - none - [change]

Explore (<file://C:/fakepath/RDFtest1.rdf>)(1-73 of 73)

Resource: <file://C:/fakepath/RDFtest1.rdf>

Results per page: 100

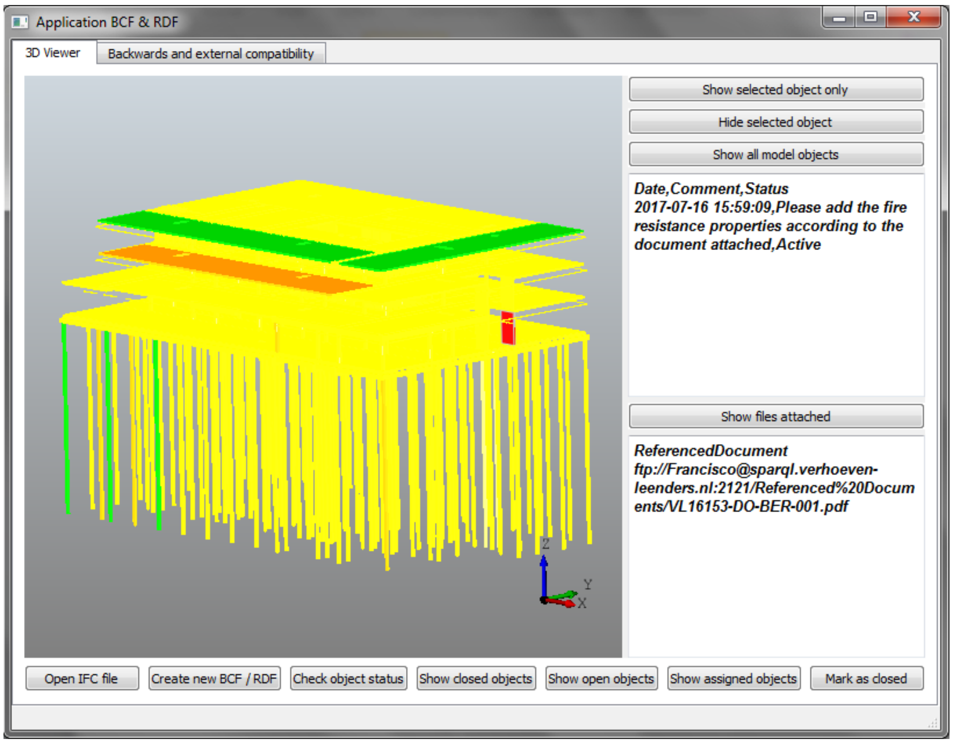
Results offset:

Show data types & language tags: ✓

Subject	Predicate	Object	Context
fbfd:BimSnippet1	fbf:hasBimSnippetIsExternal	true	<file://C:/fakepath/RDFtest1.rdf>
fbfd:BimSnippet1	fbf:hasBimSnippetReference	""	<file://C:/fakepath/RDFtest1.rdf>
fbfd:BimSnippet1	fbf:hasReferenceSchema	""	<file://C:/fakepath/RDFtest1.rdf>
fbfd:BimSnippet1	fbf:hasSnippetType	"SnippetType"	<file://C:/fakepath/RDFtest1.rdf>
fbfd:BimSnippet1	rdf:type	fbf:BimSnippet	<file://C:/fakepath/RDFtest1.rdf>
fbfd:Comment1	fbf:containsViewpoint	fbfd:Viewpoint1	<file://C:/fakepath/RDFtest1.rdf>
fbfd:Comment1	fbf:hasAuthor	"Jakob"	<file://C:/fakepath/RDFtest1.rdf>
fbfd:Comment1	fbf:hasComment	"Please update the model to solve the clash"	<file://C:/fakepath/RDFtest1.rdf>
fbfd:Comment1	fbf:hasCommentDate	2017-07-16 15:47:42	<file://C:/fakepath/RDFtest1.rdf>
fbfd:Comment1	fbf:hasCommentGuid	"b76e74e0-3807-4725-ab37-7d11a17b72b2"	<file://C:/fakepath/RDFtest1.rdf>
fbfd:Comment1	fbf:hasCommentModifiedAuthor	""	<file://C:/fakepath/RDFtest1.rdf>
fbfd:Comment1	fbf:hasCommentModifiedDate	""	<file://C:/fakepath/RDFtest1.rdf>
fbfd:Comment1	rdf:type	fbf:Comment	<file://C:/fakepath/RDFtest1.rdf>
fbfd:Comment1_1	fbf:hasAuthor	"Francisco"	<file://C:/fakepath/RDFtest1.rdf>
fbfd:Comment1_1	fbf:hasComment	"The clash is solved"	<file://C:/fakepath/RDFtest1.rdf>

Figure 37. Part of the RDF triples contained in an RDF context

After BCF information has been already created, the user can check the BCF status of the building objects as well as their comments' history by clicking on the "Check object status" button (3). Moreover, in order to visualize the BCF status of the building objects, there are two buttons that can be clicked: the "Show closed objects" button (4) and the "Show open objects" button (5). The first button (4) colours all the building objects with a "Closed" BCF status green, and makes the rest of the objects disappear. On the other hand, the second button (5) shows only the building objects that whether have no BCF information or have an "Active" or "Resolved" BCF status, and makes the "closed" building objects disappear. The building objects that have no BCF information are coloured yellow, the "Active" are coloured red and the "Resolved" building objects are coloured orange. An example of the structural model with all building objects coloured according to their BCF status is shown in Figure 38.



Date,Comment,Status
2017-07-16 15:59:09,Please add the fire resistance properties according to the document attached,Active

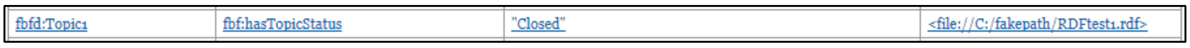
ReferencedDocument
ftp://Francisco@sparql.verhoeven-leenders.nl:2121/Referenced%20Documents/VL16153-DO-BER-001.pdf

Figure 38. Building objects coloured according to their BCF status

Table 3. Visualization of BCF status – Colour code

BCF status	Colour
Active	Red
Resolved	Orange
Closed	Green
No BCF information	Yellow

The functionalities that create and retrieve BCF information (Table 2) are based on SPARQL queries that access the RDF repository and check the available BCF information based on the GUID of the selected building object (or all the building objects depending on the functionality). For instance, the query (Figure 40) used for the visualization of the BCF status (buttons 4 and 5) checks whether BCF information exists, specifically the triple concerning the Topic Status (Figure 39), in order to colour all the building objects in the loaded IFC model based on the colour codes from Table 3.



fbfd:Topic1	fbf:hasTopicStatus	"Closed"	<file://C:/fakepath/RDFtest1.rdf>

Figure 39. Triple containing the Topic Status from RDF context 1

```
"PREFIX fbf:<http://example.org/bcfschema#>
SELECT ?Status
WHERE {
?Markup fbf:containsTopic ?b.
?b fbf:hasTopicGuid "%s".
?b fbf:hasTopicStatus ?Status.
?Markup fbf:containsComment ?c.
?c fbf:hasCommentDate ?Date.
}
ORDER BY ?Date" % guid_selection
```

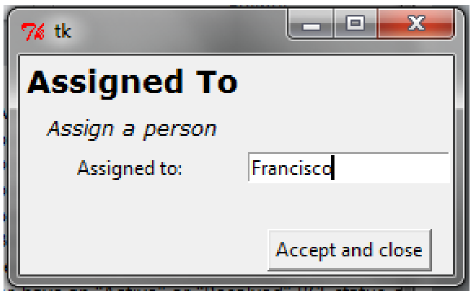
Figure 40. Query example

In case the query applies to all building objects in the IFC model, then a loop is used to access all GUIDs in the structural model (Figure 41).

```
for room in rooms:
        q = """
        PREFIX fbf:<http://example.org/bcfschema#>
        SELECT ?Status
        WHERE {
        ?Markup fbf:containsTopic ?b.
        ?b fbf:hasTopicGuid "%s".
        ?b fbf:hasTopicStatus ?Status.
        ?Markup fbf:containsComment ?c.
        ?c fbf:hasCommentDate ?Date.
        }
        ORDER BY ?Date""" % room.GlobalId
```

Figure 41. Query example 2

The “Show assigned objects” button (6) allows the user to visualize only the building objects that have a BCF issue that has been assigned to the user. When this button is clicked, a small interface appears (Figure 42) where the user can introduce his name. Then a query retrieves the information from the RDF repository and shows only the building objects which have the name of the user as the object of the triple “fbf:hasAssignedTo” (Figure 43).

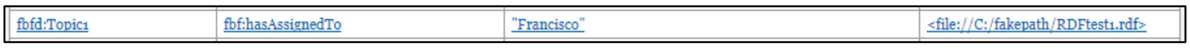


Assigned To

Assign a person

Assigned to: Francisco

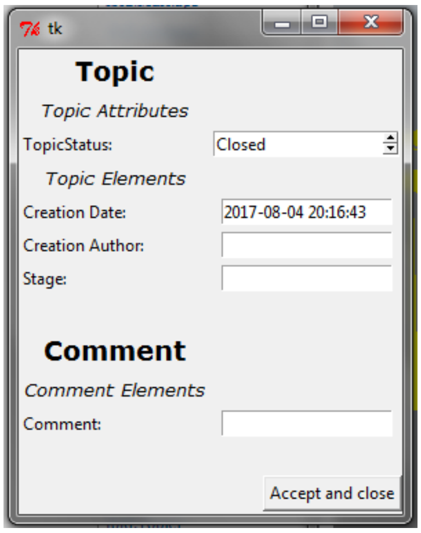
Figure 42. Assigned To interface



fbfd:Topic1	fbf:hasAssignedTo	"Francisco"	<file://C:/fakepath/RDFtest1.rdf>

Figure 43. Assigned To triple in the RDF context 1

Another important functionality is the one in the “Mark as closed” button (7). This button models BCF information as a new RDF context in the same way it was done for the “Create new BCF / RDF” button (2). However, in the interface that appears once the button is clicked (Figure 44), the topic status for the selected building object is predefined as closed and the user is able to introduce only his name, the stage of development and a comment to justify why the building object is being closed (if necessary).



Topic

Topic Attributes

TopicStatus: Closed

Topic Elements

Creation Date: 2017-08-04 20:16:43

Creation Author:

Stage:

Comment

Comment Elements

Comment:

Figure 44. Interface for the “Mark as closed” button

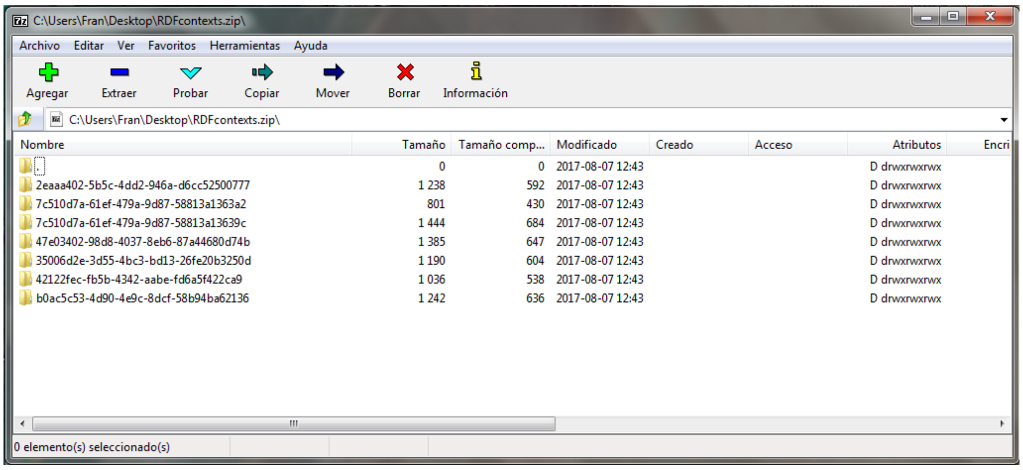
Finally, the “show selected object only” button (8), the “hide selected object” button (9) and the “show all model objects” button (10) do not have an influence in the way the BCF information is modelled. They are located on the right corner of the interface and their purpose is only to ease the visualization of the structural model by showing/hiding building objects in order to visualize only the building objects that are relevant for the user.

4.5.2 Tab 2: Backwards and external compatibility

This tab deals with the compatibility of the BCF information stored in the RDF repository with the BCF Manager tool developed by KUBUS. The buttons that are displayed on this tab are:

1. Download RDF contexts
2. Select BCF here

The first button called “Download RDF contexts” allows the user to download the existing BCF information in the RDF repository as a BCF zip file. Basically, it serializes every RDF context from the repository as a folder in the created BCF zip file using the expanded version of Topic GUID of the context (also the GUID of the building object associated with the BCF) as the name of the folder (Figure 45).


C:\Users\Fran\Desktop\RDFcontexts.zip\

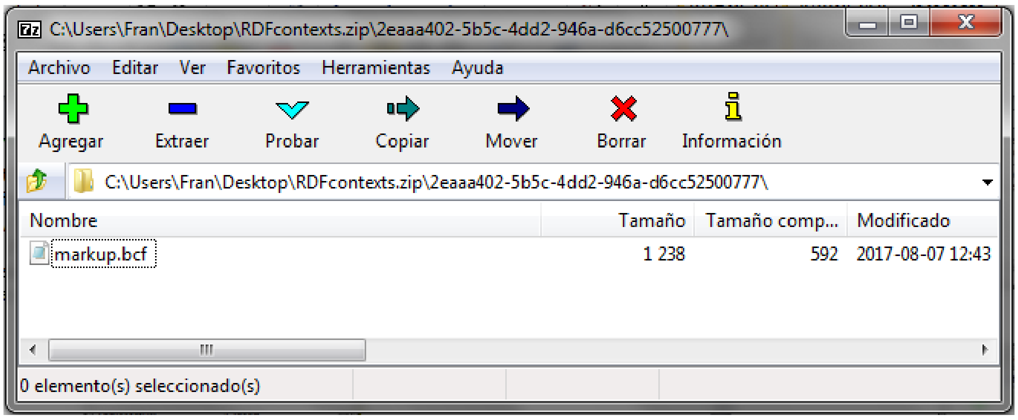
C:\Users\Fran\Desktop\RDFcontexts.zip\

Nombre	Tamaño	Tamaño comp...	Modificado	Creado	Acceso	Atributos	Encri
.	0	0	2017-08-07 12:43			D drwxrwxrwx	
2eaaa402-5b5c-4dd2-946a-d6cc52500777	1 238	592	2017-08-07 12:43			D drwxrwxrwx	
7c510d7a-61ef-479a-9d87-58813a1363a2	801	430	2017-08-07 12:43			D drwxrwxrwx	
7c510d7a-61ef-479a-9d87-58813a13639c	1 444	684	2017-08-07 12:43			D drwxrwxrwx	
47e03402-98d8-4037-8eb6-87a44680d74b	1 385	647	2017-08-07 12:43			D drwxrwxrwx	
35006d2e-3d55-4bc3-bd13-26fe20b3250d	1 190	604	2017-08-07 12:43			D drwxrwxrwx	
42122fec-fb5b-4342-aabe-fd6a5f422ca9	1 036	538	2017-08-07 12:43			D drwxrwxrwx	
b0ac5c53-4d90-4e9c-8dcf-58b94ba62136	1 242	636	2017-08-07 12:43			D drwxrwxrwx	

0 elemento(s) seleccionado(s)

Figure 45. Contents of BCF zip files

Inside every folder, there is a BCF file called “Markup” (Figure 46) which contains the BCF information that was created through the desktop viewer as an RDF context, but in an XML format (Figure 47). Specifically, this XML file is created by generating first an XML template containing the elements and attributes fields of the original BCF schema (section 7.7 Appendix VII: BCF Schema (XSD)), and filling the fields with the values stored in the RDF context; all of it programmed using Python.

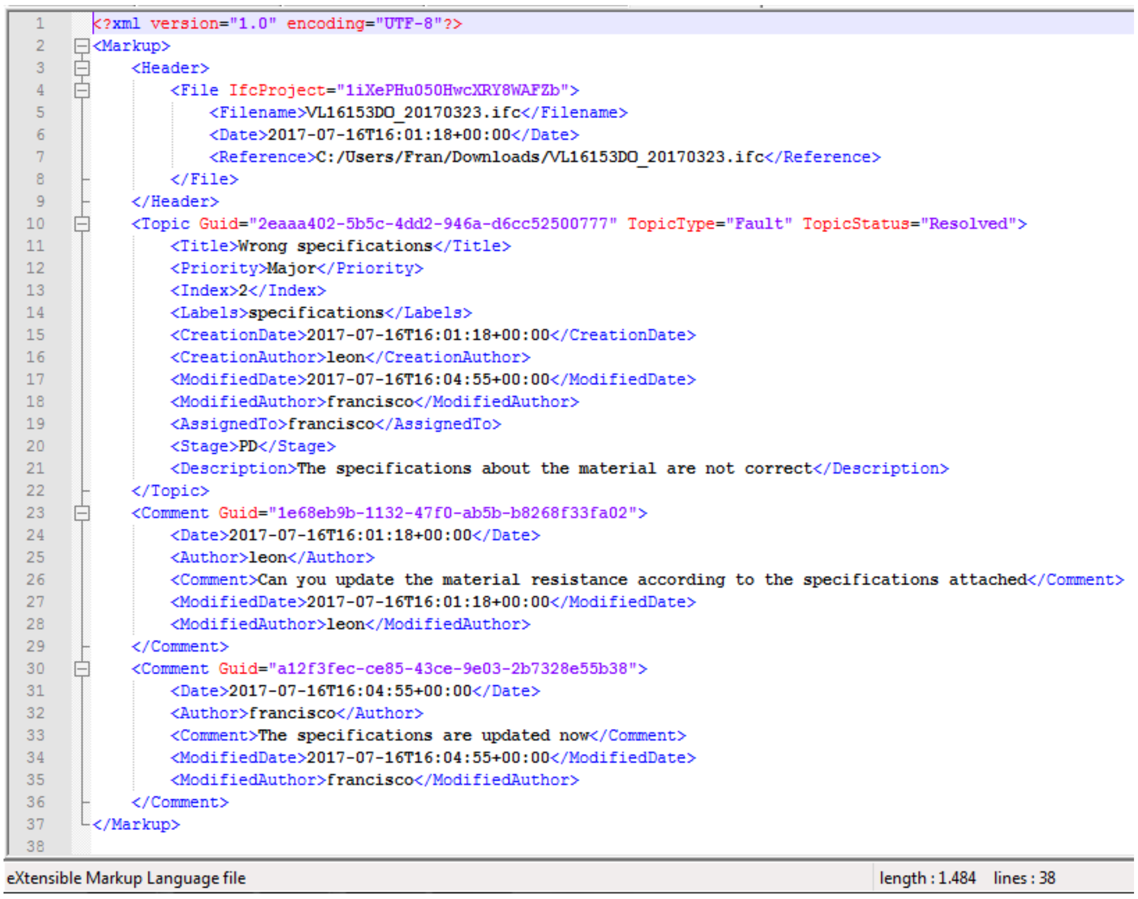

C:\Users\Fran\Desktop\RDFcontexts.zip\2eaaa402-5b5c-4dd2-946a-d6cc52500777\

C:\Users\Fran\Desktop\RDFcontexts.zip\2eaaa402-5b5c-4dd2-946a-d6cc52500777\

Nombre	Tamaño	Tamaño comp...	Modificado
markup.bcf	1 238	592	2017-08-07 12:43

0 elemento(s) seleccionado(s)

Figure 46. Content of folders inside the BCF zip file

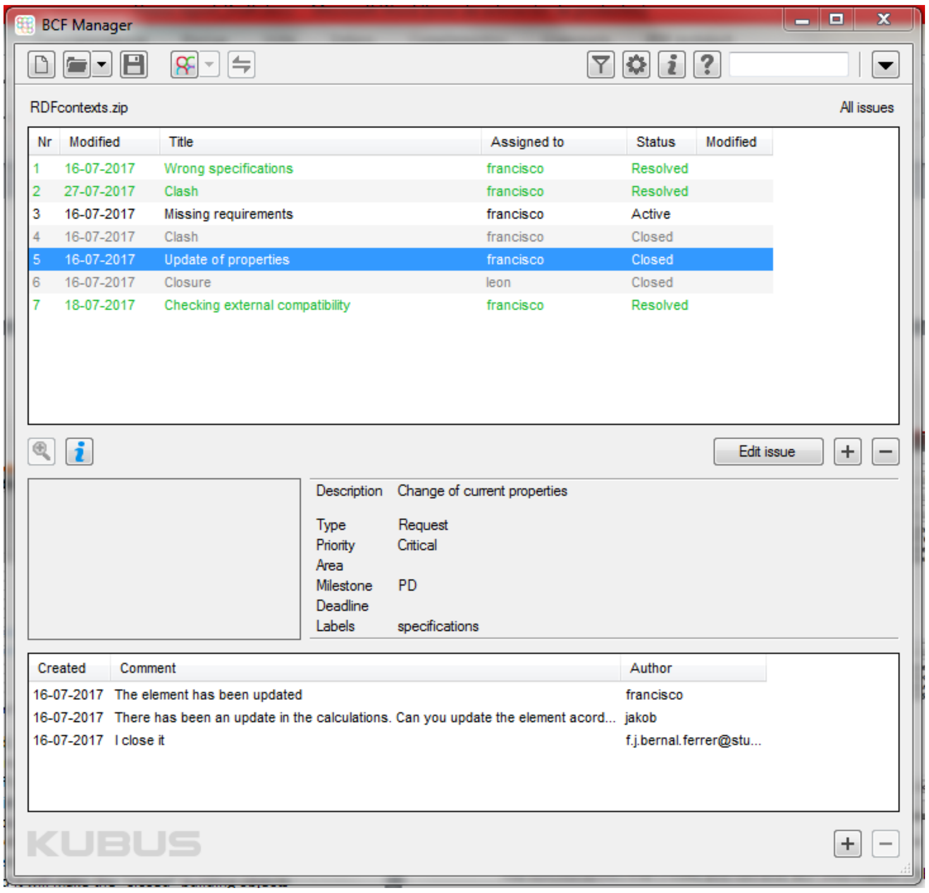


```xml
<?xml version="1.0" encoding="UTF-8"?>
<Markup>
    <Header>
        <File IfcProject="1iXePHu050HwcXRY8WAFZb">
            <Filename>VL16153DO_20170323.ifc</Filename>
            <Date>2017-07-16T16:01:18+00:00</Date>
            <Reference>C:/Users/Fran/Downloads/VL16153DO_20170323.ifc</Reference>
        </File>
    </Header>
    <Topic Guid="2eaaa402-5b5c-4dd2-946a-d6cc52500777" TopicType="Fault" TopicStatus="Resolved">
        <Title>Wrong specifications</Title>
        <Priority>Major</Priority>
        <Index>2</Index>
        <Labels>specifications</Labels>
        <CreationDate>2017-07-16T16:01:18+00:00</CreationDate>
        <CreationAuthor>leon</CreationAuthor>
        <ModifiedDate>2017-07-16T16:04:55+00:00</ModifiedDate>
        <ModifiedAuthor>francisco</ModifiedAuthor>
        <AssignedTo>francisco</AssignedTo>
        <Stage>PD</Stage>
        <Description>The specifications about the material are not correct</Description>
    </Topic>
    <Comment Guid="1e68eb9b-1132-47f0-ab5b-b8268f33fa02">
        <Date>2017-07-16T16:01:18+00:00</Date>
        <Author>leon</Author>
        <Comment>Can you update the material resistance according to the specifications attached</Comment>
        <ModifiedDate>2017-07-16T16:01:18+00:00</ModifiedDate>
        <ModifiedAuthor>leon</ModifiedAuthor>
    </Comment>
    <Comment Guid="a12f3fec-ce85-43ce-9e03-2b7328e55b38">
        <Date>2017-07-16T16:04:55+00:00</Date>
        <Author>francisco</Author>
        <Comment>The specifications are updated now</Comment>
        <ModifiedDate>2017-07-16T16:04:55+00:00</ModifiedDate>
        <ModifiedAuthor>francisco</ModifiedAuthor>
    </Comment>
</Markup>
```

eXtensible Markup Language file   length : 1.484   lines : 38

Figure 47. BCF information serialized as XML file

Once the RDF contexts have been serialized inside the BCF zip file, their contents can be visualized and edited using the BCF Manager from KUBUS ("BIMcollab," 2017). As it can be seen in Figure 48, the seven RDF contexts (and their BCF information) that were created and stored in the RDF repository using the desktop viewer (Figure 34) appear.



RDFcontexts.zip

Nr	Modified	Title	Assigned to	Status	Modified
1	16-07-2017	Wrong specifications	francisco	Resolved	
2	27-07-2017	Clash	francisco	Resolved	
3	16-07-2017	Missing requirements	francisco	Active	
4	16-07-2017	Clash	francisco	Closed	
5	16-07-2017	Update of properties	francisco	Closed	
6	16-07-2017	Closure	leon	Closed	
7	18-07-2017	Checking external compatibility	francisco	Resolved	

Description	Change of current properties
Type	Request
Priority	Critical
Area	
Milestone	PD
Deadline	
Labels	specifications

Created	Comment	Author
16-07-2017	The element has been updated	francisco
16-07-2017	There has been an update in the calculations. Can you update the element acord...	jakob
16-07-2017	I close it	f.j.bernal.ferrer@stu...

Figure 48. Visualization of downloaded RDF contexts in the BCF Manager from KUBUS

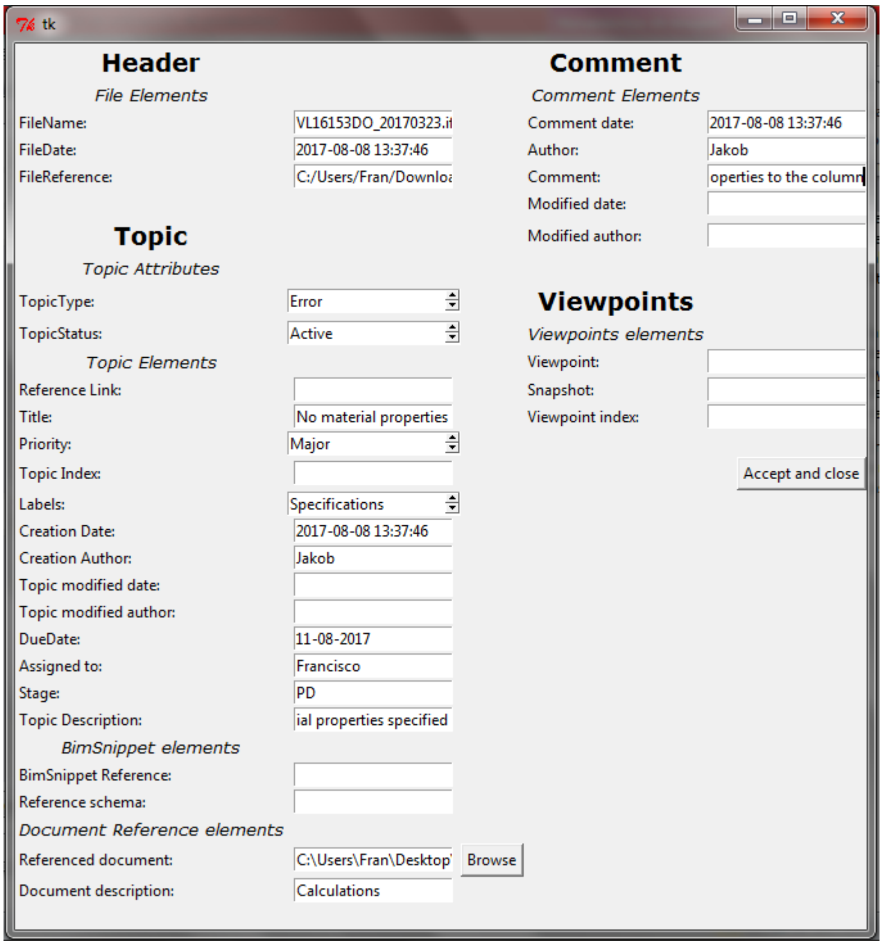
After the appropriate changes have been done and the new BCF zip file has been saved in the BCF Manager, the new BCF information can be uploaded back to the RDF repository by using the second button included in this tab, called “Select BCF here”. By just selecting the BCF zip file again, the new comments and BCF status are added to their respective RDF contexts, updating like this the information in the RDF repository.

This is also programmed in Python by using the same template that was implemented for creating the RDF contexts and triples on the first tab (button 2 called “Create new BCF / RDF”). However, the information to be filled in the triples’ objects (Literals) will not come from the entry boxes of the BCF interface this time, but from the filled fields in the XML “markup” files (.bcf) inside every folder of the updated BCF zip file.

4.6 Prototype validation

This chapter deals with the validation of the developed tool in terms of correctness and completeness of the generated data and data visualization. To that end, the snapshots of the RDF4J workbench together with the software BCF Manager from KUBUS is used as means to check whether the created data using the developed tool is visualized completely and correctly.

For the purpose of the validation, a new RDF context containing the BCF information specified in Figure 49 has been created for the selected column in Figure 50.



tk

Header

File Elements

FileName:	VL16153DO_20170323.i
FileDate:	2017-08-08 13:37:46
FileReference:	C:/Users/Fran/Downloa

Topic

Topic Attributes

TopicType:	Error
TopicStatus:	Active

Topic Elements

Reference Link:	
Title:	No material properties
Priority:	Major
Topic Index:	
Labels:	Specifications
Creation Date:	2017-08-08 13:37:46
Creation Author:	Jakob
Topic modified date:	
Topic modified author:	
DueDate:	11-08-2017
Assigned to:	Francisco
Stage:	PD
Topic Description:	ial properties specified

BimSnippet elements

BimSnippet Reference:	
Reference schema:	

Document Reference elements

Referenced document:	C:\Users\Fran\Desktop\
Document description:	Calculations

Browse

Comment

Comment Elements

Comment date:	2017-08-08 13:37:46
Author:	Jakob
Comment:	operties to the column
Modified date:	
Modified author:	

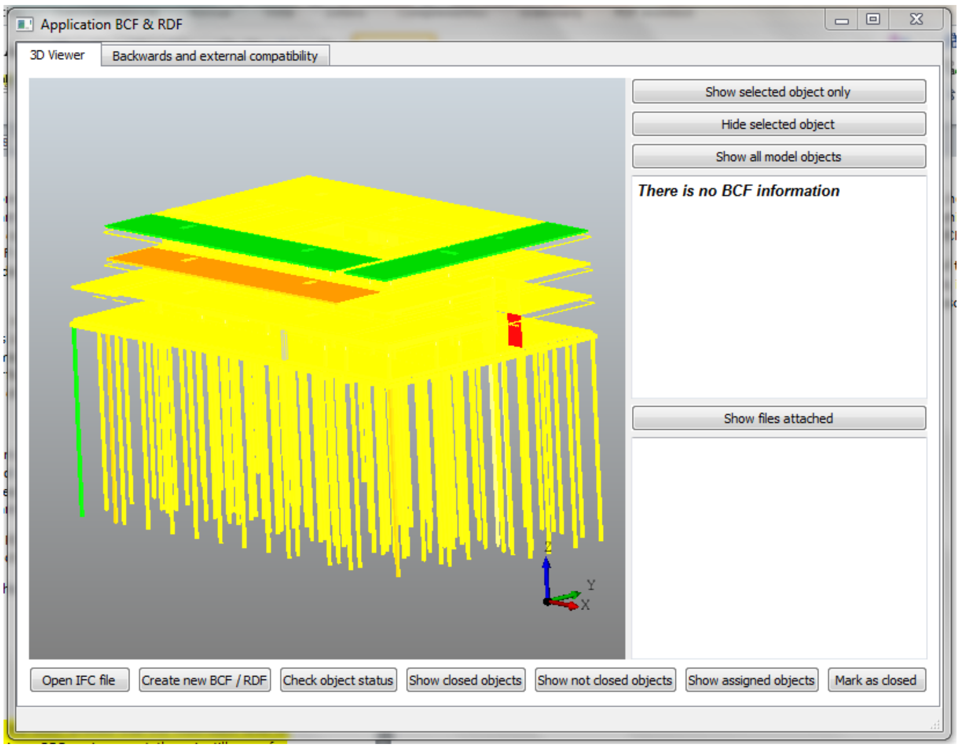
Viewpoints

Viewpoints elements

Viewpoint:	
Snapshot:	
Viewpoint index:	

Accept and close

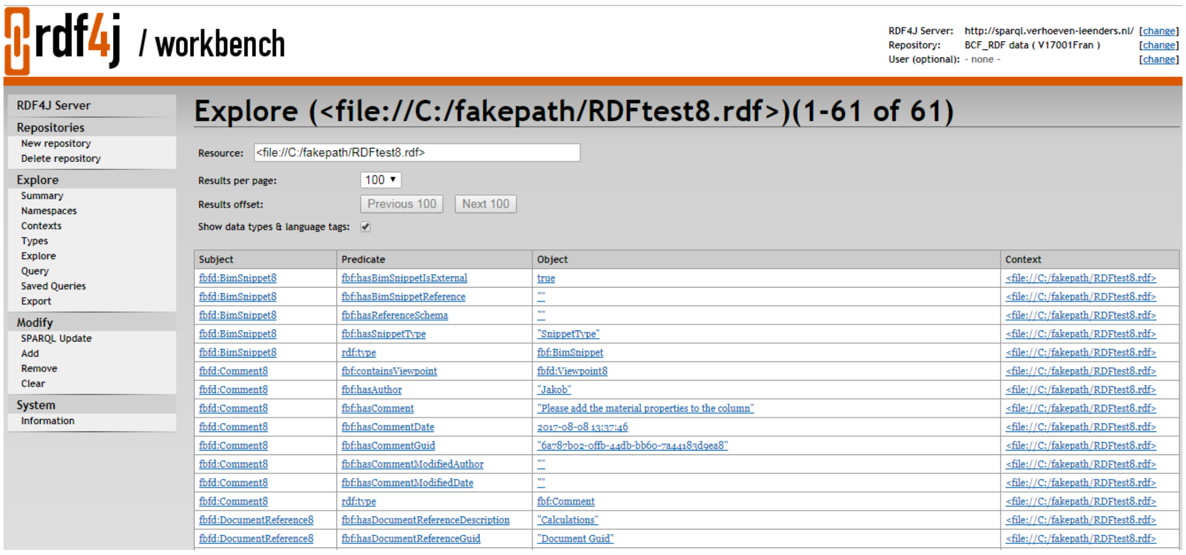
Figure 49. BCF information for prototype validation



There is no BCF information

Figure 50. Selected column for prototype validation

The new RDF context (number 8) appears in the RDF repository as well as the 61 triples that the RDF context consists of (Figure 51). Moreover, Table 4 shows specifically the triples that have been created based on the introduced information.



RDF4J Server: http://sparql.verhoeven-leenders.nl/ [change]
Repository: BCF_RDF data (V17001Fran) [change]
User (optional): - none - [change]

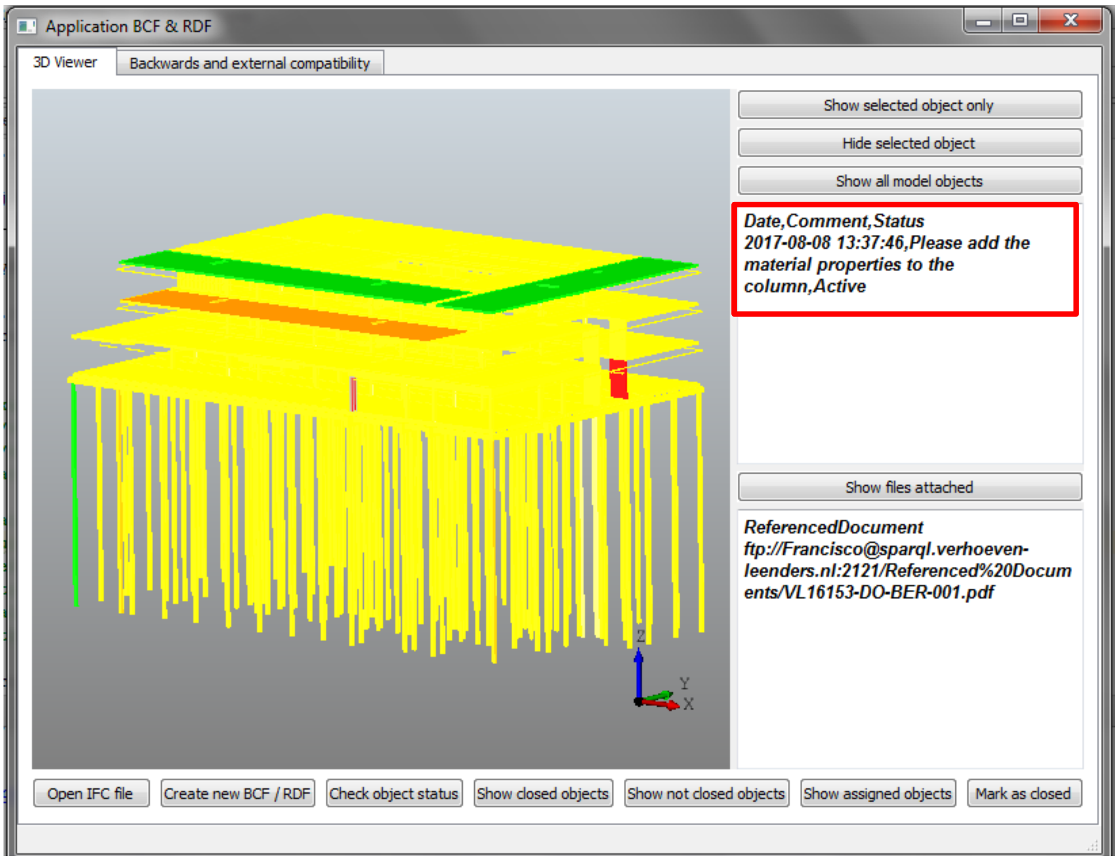
Explore (<file://C:/fakepath/RDFtest8.rdf>)(1-61 of 61)

Resource: <file://C:/fakepath/RDFtest8.rdf>
Results per page: 100
Results offset:
Show data types & language tags: ✓

Subject	Predicate	Object	Context
fbfd:BimSnippet8	fbf:hasBimSnippetIsExternal	true	<file://C:/fakepath/RDFtest8.rdf>
fbfd:BimSnippet8	fbf:hasBimSnippetReference	""	<file://C:/fakepath/RDFtest8.rdf>
fbfd:BimSnippet8	fbf:hasReferenceSchema	""	<file://C:/fakepath/RDFtest8.rdf>
fbfd:BimSnippet8	fbf:hasSnippetType	"SnippetType"	<file://C:/fakepath/RDFtest8.rdf>
fbfd:BimSnippet8	rdf:type	fbf:BimSnippet	<file://C:/fakepath/RDFtest8.rdf>
fbfd:Comment8	fbf:containsViewpoint	fbfd:Viewpoint8	<file://C:/fakepath/RDFtest8.rdf>
fbfd:Comment8	fbf:hasAuthor	"Jakob"	<file://C:/fakepath/RDFtest8.rdf>
fbfd:Comment8	fbf:hasComment	"Please add the material properties to the column"	<file://C:/fakepath/RDFtest8.rdf>
fbfd:Comment8	fbf:hasCommentDate	2017-08-08 13:37:46	<file://C:/fakepath/RDFtest8.rdf>
fbfd:Comment8	fbf:hasCommentGuid	"6a787b02-offb-44db-bb60-7a44183d9ea8"	<file://C:/fakepath/RDFtest8.rdf>
fbfd:Comment8	fbf:hasCommentModifiedAuthor	""	<file://C:/fakepath/RDFtest8.rdf>
fbfd:Comment8	fbf:hasCommentModifiedDate	""	<file://C:/fakepath/RDFtest8.rdf>
fbfd:Comment8	rdf:type	fbf:Comment	<file://C:/fakepath/RDFtest8.rdf>
fbfd:DocumentReference8	fbf:hasDocumentReferenceDescription	"Calculations"	<file://C:/fakepath/RDFtest8.rdf>
fbfd:DocumentReference8	fbf:hasDocumentReferenceGuid	"Document Guid"	<file://C:/fakepath/RDFtest8.rdf>

Figure 51. RDF context 8 for prototype validation

The details of the comment, as well as the BCF status, can be visualized in the desktop viewer too (Figure 52).



Date,Comment,Status
2017-08-08 13:37:46,Please add the material properties to the column,Active

ReferencedDocument
ftp://Francisco@sparql.verhoeven-leenders.nl:2121/Referenced%20Documents/VL16153-DO-BER-001.pdf

Figure 52. Visualization of the new BCF information in the developed desktop viewer

Table 4. Comparison between the introduced (BCF interface) and generated (RDF triples) information

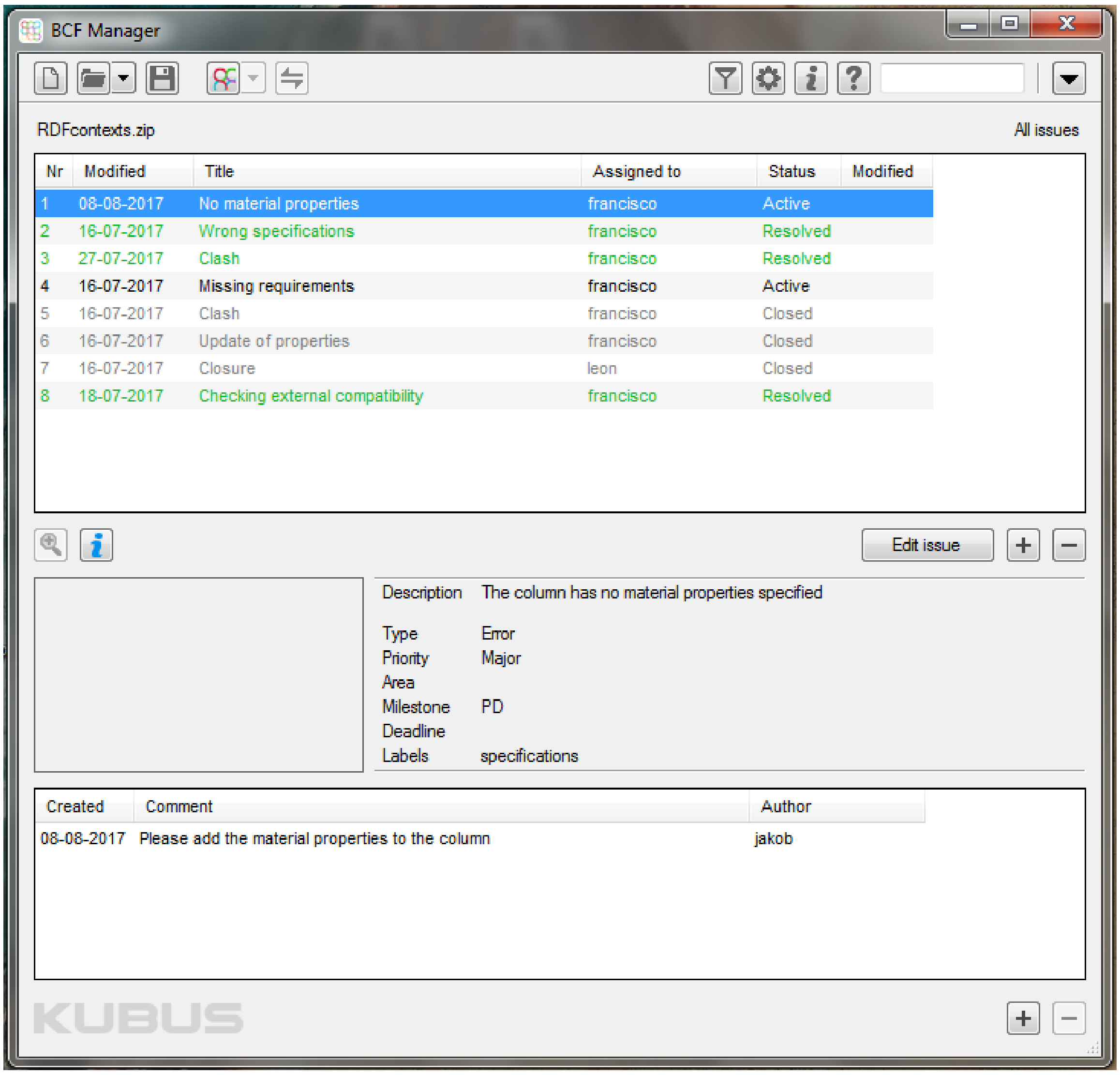
Element	**Value**			
Filename				
Introduced value (BCF interface):	VL16153DO_20170323.ifc			
Triple (RDF repository):	***Subject***	***Predicate***	***Object***	***RDF context***
	fbfd:File8	fbf:hasFileName	"VL16153DO_20170323.ifc"	<file://C:/fakepath/RDFtest8.rdf>
File date				
Introduced value (BCF interface):	2017-08-08 13:37:46			
Triple (RDF repository):	***Subject***	***Predicate***	***Object***	***RDF context***
	fbfd:File8	fbf:hasFileDate	2017-08-08 13:37:46	<file://C:/fakepath/RDFtest8.rdf>
File reference				
Introduced value (BCF interface):	C:/Users/Fran/Downloads/VL16153DO_20170323.ifc			
Triple (RDF repository):	***Subject***	***Predicate***	***Object***	***RDF context***
	fbfd:File8	fbf:hasFileReference	"C:/Users/Fran/Downloads/VL16153DO_20170323.ifc"	<file://C:/fakepath/RDFtest8.rdf>
Topic Type				
Introduced value (BCF interface):	Error			
Triple (RDF repository):	***Subject***	***Predicate***	***Object***	***RDF context***
	fbfd:Topic8	fbf:hasTopicType	"Error"	<file://C:/fakepath/RDFtest8.rdf>
Topic Status				
Introduced value (BCF interface):	Active			
Triple (RDF repository):	***Subject***	***Predicate***	***Object***	***RDF context***
	fbfd:Topic8	fbf:hasTopicStatus	"Active"	<file://C:/fakepath/RDFtest8.rdf>
Title				
Introduced value (BCF interface):	No material properties			
Triple (RDF repository):	***Subject***	***Predicate***	***Object***	***RDF context***
	fbfd:Topic8	fbf:hasTitle	"No material properties"	<file://C:/fakepath/RDFtest8.rdf>
Priority				
Introduced value (BCF interface):	Major			
Triple (RDF repository):	***Subject***	***Predicate***	***Object***	***RDF context***
	fbfd:Topic8	fbf:hasPriority	"Major"	<file://C:/fakepath/RDFtest8.rdf>
Labels				
Introduced value (BCF interface):	Specifications			
Triple (RDF repository):	***Subject***	***Predicate***	***Object***	***RDF context***
	fbfd:Topic8	fbf:hasLabels	"Specifications"	<file://C:/fakepath/RDFtest8.rdf>

Creation Date				
Introduced value (BCF interface):	2017-08-08 13:37:46			
Triple (RDF repository):	***Subject***	***Predicate***	***Object***	***RDF context***
	fbfd:Topic8	fbf:hasCreationDate	“2017-08-08 13:37:46”	<file://C:/fakepath/RDFtest8.rdf>
Creation Author				
Introduced value (BCF interface):	Jakob			
Triple (RDF repository):	***Subject***	***Predicate***	***Object***	***RDF context***
	fbfd:Topic8	fbf:hasCreationAuthor	“Jakob”	<file://C:/fakepath/RDFtest8.rdf>
Due date				
Introduced value (BCF interface):	11-08-2017			
Triple (RDF repository):	***Subject***	***Predicate***	***Object***	***RDF context***
	fbfd:Topic8	fbf:hasDueDate	“11-08-2017”	<file://C:/fakepath/RDFtest8.rdf>
Assigned To				
Introduced value (BCF interface):	Francisco			
Triple (RDF repository):	***Subject***	***Predicate***	***Object***	***RDF context***
	fbfd:Topic8	fbf:hasAssignedTo	“Francisco”	<file://C:/fakepath/RDFtest8.rdf>
Stage				
Introduced value (BCF interface):	PD			
Triple (RDF repository):	***Subject***	***Predicate***	***Object***	***RDF context***
	fbfd:Topic8	fbf:hasStage	“PD”	<file://C:/fakepath/RDFtest8.rdf>
Topic description				
Introduced value (BCF interface):	The column has no material properties specified			
Triple (RDF repository):	***Subject***	***Predicate***	***Object***	***RDF context***
	fbfd:Topic8	fbf:hasTopicDescription	“The column has no material properties specified”	<file://C:/fakepath/RDFtest8.rdf>
Referenced Document				
Introduced value (BCF interface):	ftp://Francisco@sparql.verhoeven-leenders.nl:2121/Referenced%20Documents/VL16153-DO-BER-001.pdf			
Triple (RDF repository):	***Subject***	***Predicate***	***Object***	***RDF context***
	fbfd:DocumentReference8	fbf:hasReferencedDocument	“ftp://Francisco@sparql.verhoeven-leenders.nl:2121/Referenced%20Documents/VL16153-DO-BER-001.pdf”	<file://C:/fakepath/RDFtest8.rdf>
Document description				
Introduced value (BCF interface):	Calculations			
Triple (RDF repository):	***Subject***	***Predicate***	***Object***	***RDF context***
	fbfd:DocumentReference8	fbf:hasDocumentReferenceDescription	“Calculations”	<file://C:/fakepath/RDFtest8.rdf>

Comment Date				
Introduced value (BCF interface):	2017-08-08 13:37:46			
Triple (RDF repository):	***Subject***	***Predicate***	***Object***	***RDF context***
	fbfd:Comment8	fbf:hasCommentDate	"2017-08-08 13:37:46"	<file://C:/fakepath/RDFtest8.rdf>
Author				
Introduced value (BCF interface):	Jakob			
Triple (RDF repository):	***Subject***	***Predicate***	***Object***	***RDF context***
	fbfd:Comment8	fbf:hasAuthor	"Jakob"	<file://C:/fakepath/RDFtest8.rdf>
Comment				
Introduced value (BCF interface):	Please add the material properties to the column			
Triple (RDF repository):	***Subject***	***Predicate***	***Object***	***RDF context***
	fbfd:Comment8	fbf:hasComment	"Please add the material properties to the column"	<file://C:/fakepath/RDFtest8.rdf>

On the other hand, Figure 53 and Figure 54 illustrate how the same BCF information can be visualized in the BCF Manager from KUBUS. The new RDF context 8 appears listed as a new topic in the BCF manager together with the date, title, status, description, priority, labels, stage (called milestone in the BCF manager interface), and who the topic is assigned to. In addition, the comment that is related to this topic is shown as well, including the comment's date and author.

Information that cannot be visualized is the referenced document; however, this is due to the fact that the BCF Manager from KUBUS does not support such functionality. Therefore, although the appropriate lines of code for translating this information are included in the final version of the code, they are commented.

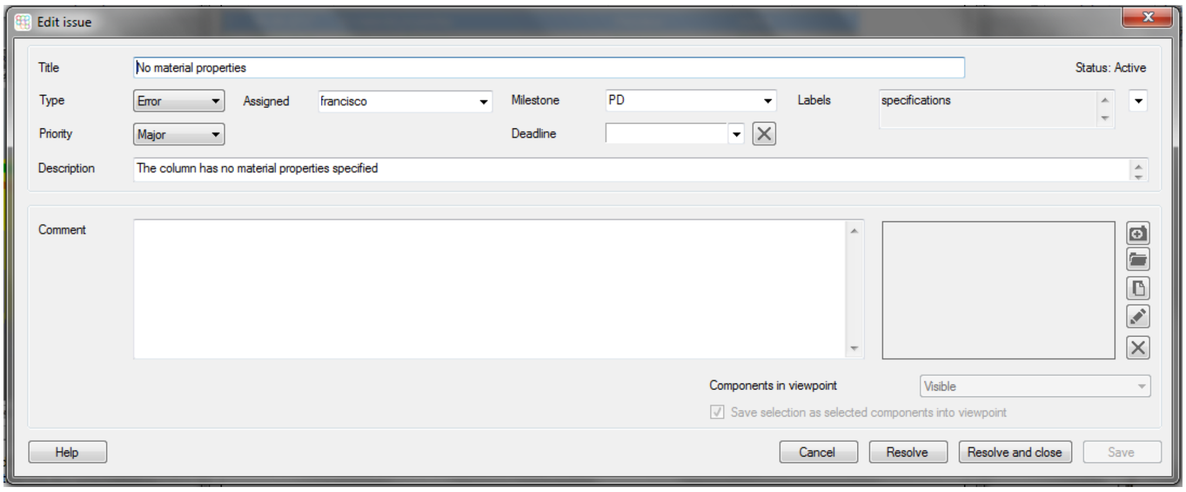


Nr	Modified	Title	Assigned to	Status	Modified
1	08-08-2017	No material properties	francisco	Active	
2	16-07-2017	Wrong specifications	francisco	Resolved	
3	27-07-2017	Clash	francisco	Resolved	
4	16-07-2017	Missing requirements	francisco	Active	
5	16-07-2017	Clash	francisco	Closed	
6	16-07-2017	Update of properties	francisco	Closed	
7	16-07-2017	Closure	leon	Closed	
8	18-07-2017	Checking external compatibility	francisco	Resolved	

Description	The column has no material properties specified
Type	Error
Priority	Major
Area	
Milestone	PD
Deadline	
Labels	specifications

Created	Comment	Author
08-08-2017	Please add the material properties to the column	jakob

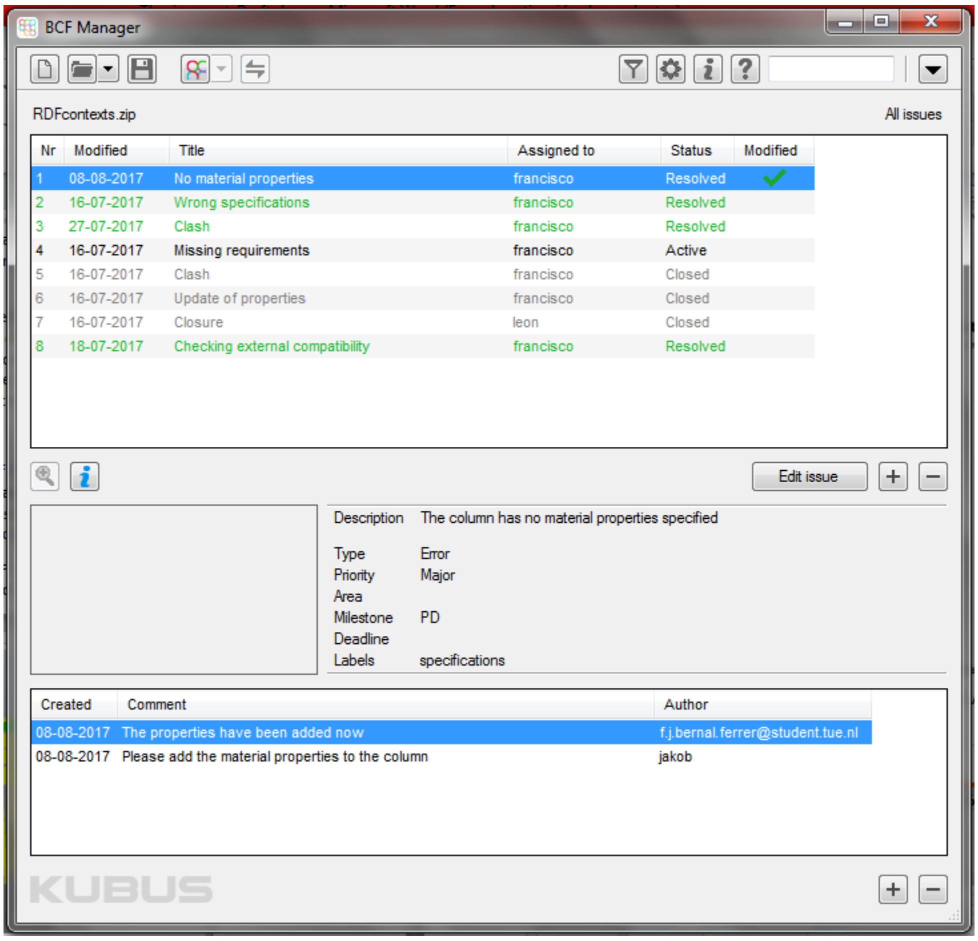
Figure 53. Visualization of RDF context 8 in the BCF Manager



Title: No material properties
Status: Active
Type: Error
Assigned: francisco
Milestone: PD
Labels: specifications
Priority: Major
Deadline:
Description: The column has no material properties specified
Comment:
Components in viewpoint: Visible
☑ Save selection as selected components into viewpoint

Figure 54. Visualization of RDF context 8 in the BCF Manager 2

Finally, in case the BCF information is edited in the BCF Manager, specifically by adding more comments and changing the BCF status to "Resolved" (Figure 55), Figure 56 shows how the new comment's triples are added to the RDF context 8 once BCF information has been uploaded back to the RDF repository. The RDF context 8 now lists 67 triples due to the new comment. In addition, the topic status has been updated as well to "Resolved".



RDFcontexts.zip

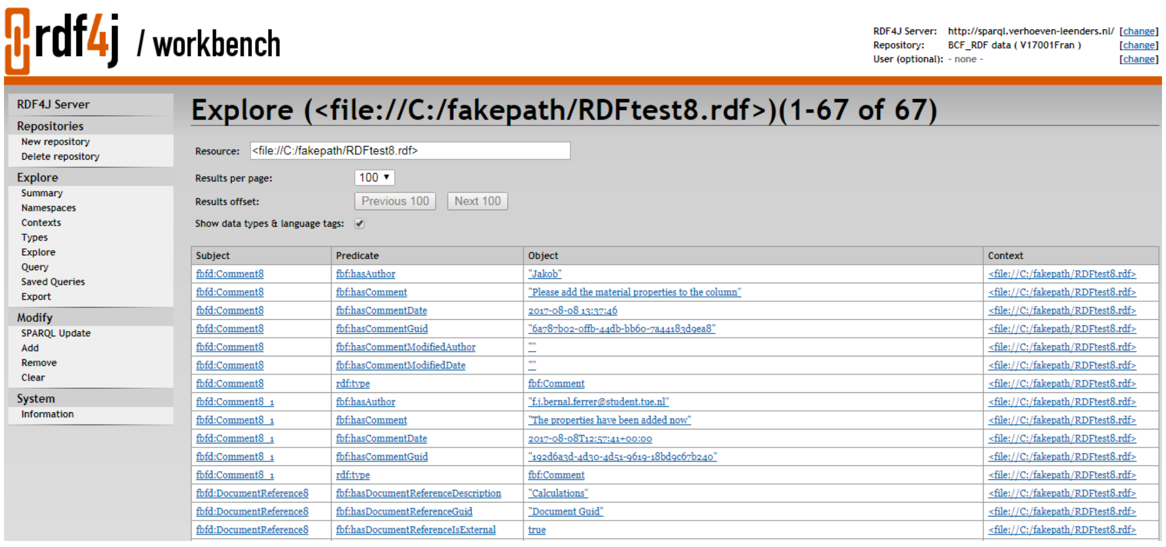
All issues

Nr	Modified	Title	Assigned to	Status	Modified
1	08-08-2017	No material properties	francisco	Resolved	✓
2	16-07-2017	Wrong specifications	francisco	Resolved	
3	27-07-2017	Clash	francisco	Resolved	
4	16-07-2017	Missing requirements	francisco	Active	
5	16-07-2017	Clash	francisco	Closed	
6	16-07-2017	Update of properties	francisco	Closed	
7	16-07-2017	Closure	leon	Closed	
8	18-07-2017	Checking external compatibility	francisco	Resolved	

Description	The column has no material properties specified
Type	Error
Priority	Major
Area	
Milestone	PD
Deadline	
Labels	specifications

Created	Comment	Author
08-08-2017	The properties have been added now	f.j.bernal.ferrer@student.tue.nl
08-08-2017	Please add the material properties to the column	jakob

Figure 55. Edition of the RDF context 8 in the BCF Manager



RDF4J Server: http://sparql.verhoeven-leenders.nl/ [change]
Repository: BCF_RDF data (V17001Fran) [change]
User (optional): - none - [change]

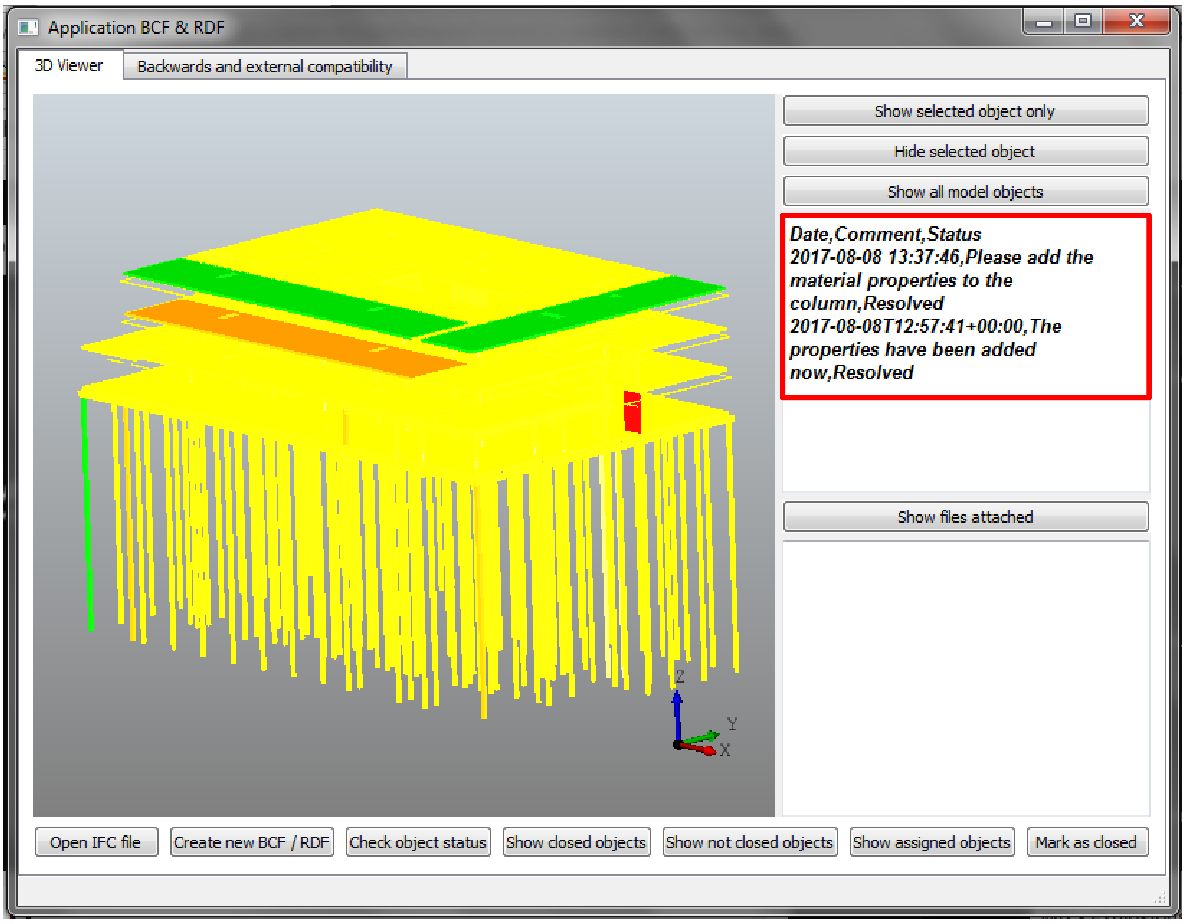
Explore (<file://C:/fakepath/RDFtest8.rdf>)(1-67 of 67)

Resource: <file://C:/fakepath/RDFtest8.rdf>
Results per page: 100
Results offset:
Show data types & language tags: ✓

Subject	Predicate	Object	Context
fbfd:Comment8	fbf:hasAuthor	"Jakob"	<file://C:/fakepath/RDFtest8.rdf>
fbfd:Comment8	fbf:hasComment	"Please add the material properties to the column"	<file://C:/fakepath/RDFtest8.rdf>
fbfd:Comment8	fbf:hasCommentDate	2017-08-08 13:37:46	<file://C:/fakepath/RDFtest8.rdf>
fbfd:Comment8	fbf:hasCommentGuid	"6a787b02-offb-44db-bb60-7a44183d9ea8"	<file://C:/fakepath/RDFtest8.rdf>
fbfd:Comment8	fbf:hasCommentModifiedAuthor	""	<file://C:/fakepath/RDFtest8.rdf>
fbfd:Comment8	fbf:hasCommentModifiedDate	""	<file://C:/fakepath/RDFtest8.rdf>
fbfd:Comment8	rdf:type	fbf:Comment	<file://C:/fakepath/RDFtest8.rdf>
fbfd:Comment8_1	fbf:hasAuthor	"f.j.bernal.ferrer@student.tue.nl"	<file://C:/fakepath/RDFtest8.rdf>
fbfd:Comment8_1	fbf:hasComment	"The properties have been added now"	<file://C:/fakepath/RDFtest8.rdf>
fbfd:Comment8_1	fbf:hasCommentDate	2017-08-08T12:57:41+00:00	<file://C:/fakepath/RDFtest8.rdf>
fbfd:Comment8_1	fbf:hasCommentGuid	"192d6a3d-4d30-4d51-9619-18bd9c67b240"	<file://C:/fakepath/RDFtest8.rdf>
fbfd:Comment8_1	rdf:type	fbf:Comment	<file://C:/fakepath/RDFtest8.rdf>
fbfd:DocumentReference8	fbf:hasDocumentReferenceDescription	"Calculations"	<file://C:/fakepath/RDFtest8.rdf>
fbfd:DocumentReference8	fbf:hasDocumentReferenceGuid	"Document Guid"	<file://C:/fakepath/RDFtest8.rdf>
fbfd:DocumentReference8	fbf:hasDocumentReferenceIsExternal	true	<file://C:/fakepath/RDFtest8.rdf>

Figure 56. Visualization of the edited RDF context 8 in the RDF4J repository

The edited and new BCF information can be visualized in the desktop viewer as well (Figure 57), where the column has changed its colour from red to orange and the new comment appears on the property box.



Date,Comment,Status
2017-08-08 13:37:46,Please add the material properties to the column,Resolved
2017-08-08T12:57:41+00:00,The properties have been added now,Resolved

Figure 57. Visualization of the edited RDF context 8 in the developed tool

4.7 Prototype limitations

Previously in this chapter, it was validated that the prototype is able to perform the identified and necessary use cases for the purpose of internal design validation attestation; however, due to the academic nature of this prototype, some limitations exist.

First of all, the current level of development of the tool allows the user to create only one BCF per building object. In such way, if an RDF context has already been created for a building object, the next time that a BCF needs to be created for the same building object the desktop viewer only allows the addition of comments to the existing RDF context. In other words, it is not possible to have more than one BCF topic per building object.

The main reason it was decided to keep a 1:1 relation between the BCF topics and the building objects is because when the BCF information is retrieved using queries, the BCF information that is printed in the property box is based on the GUID of the building object. That means that if more than one BCF topic exists for the same building object when the status or comments' history of that building object is retrieved, the desktop viewer prints all the information mixed in the property box. And the way the prototype is designed (and/or programmed) currently, it is not possible to filter or distinguish between different topics.

Secondly, in order to create BCF information, whether it is creating a new BCF issue or marking a building object as closed, it can only be done selecting building objects one by one. For instance, given the situation that several columns have been already checked by the structural engineer, it will not be possible to mark all of them as closed at once, which considering the practical use, decreases the performance speed with which the user can operate or manage building objects and BCF information with the developed tool.

Thirdly, the desktop viewer does not support viewpoints and snapshots of the IFC model when creating BCF issues. Although for practical purposes, they play an important role in the description of BCF data, implementing them was considered on top of the scope of this thesis due to time and programming complexity reasons. In fact, it can be noticed that both, snapshots and viewpoints files, were missing in the serialized BCF zip files during the explanation of the prototype development and its validation. Nevertheless, it did not affect the development of the tool or its academic value.

Solutions to the explained limitations are suggested and discussed in both section 4.8 Prototype discussion and section 5.1 Limitations and recommendations (for possible follow-up research).

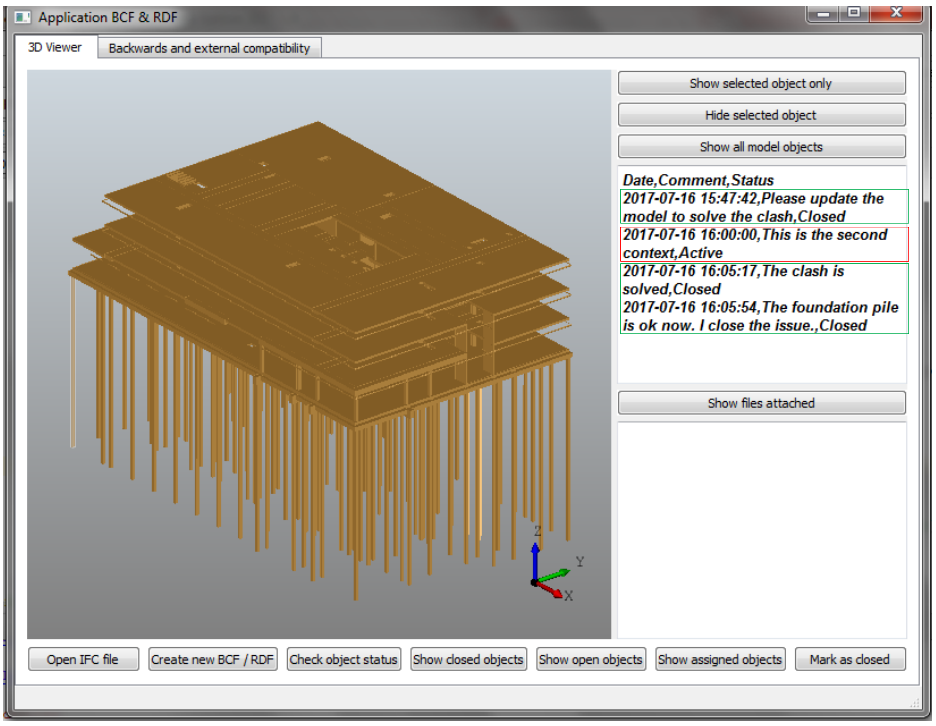
4.8 Prototype discussion

After developing, validating and analysing the limitations of the prototype, there are some points that could be discussed as a conclusion of what the benefits, downsides and, on the whole, the findings of this development are.

First of all, modelling the BCF information as triples in RDF contexts has improved the interoperability of the BCF data due to the fact that the data itself is connected through the GUID of the building object that the BCF information is related to. In such manner, BCF information is linked to the IFC model being the building object's GUID the key of that link.

This puts into practice the statements of Jakob Beetz (J. Beetz et al., 2014), who indicates that one of the benefits of using RDF for modelling the information was that a single GUID could identify a whole cluster of concepts, their values and relationships.

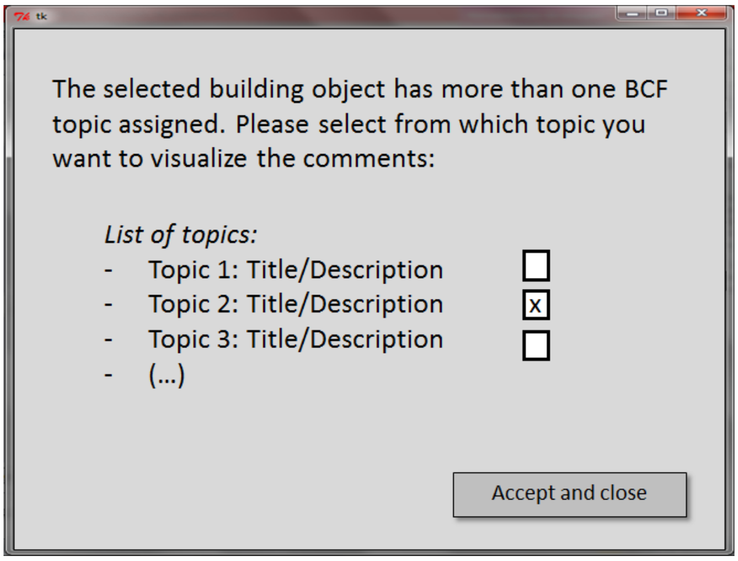
Nevertheless, this way of modelling information has shown to have downsides as well if the recovery of information is not filtered properly. It has been explained in section 4.7 Prototype limitations that without a way of distinguishing between several BCF topics for the same building object's GUID, the relation between the BCF topics and the building object's GUID had to be 1:1, otherwise information will be mixed (Figure 58) when retrieving it using SPARQL queries even if comments are filtered by multiple factors (ORDER BY).



Date,Comment,Status
2017-07-16 15:47:42,Please update the model to solve the clash,Closed
2017-07-16 16:00:00,This is the second context,Active
2017-07-16 16:05:17,The clash is solved,Closed
2017-07-16 16:05:54,The foundation pile is ok now. I close the issue.,Closed

Figure 58. Example showing the mixed comments coming from different topics related to the same object

A solution could be possibly found in the way information is queried by adapting the design of the SPARQL query and adding a second identifier on top of the building object's GUID. This second identifier would filter all comments related to the same building object depending on which markup (topic) they belong to. Nevertheless, this solution will require the user to modify the queries in the back-end of the prototype everytime he wants to specify the markup's ID that acts as the second identifier. For that reason, a graphic user interface (GUI) that allows adding this second identifier is suggested to prevent the user from edition in a back-end level (Figure 59). The proposed GUI would appear when retrieving comments (Check object status button) from a building object with more than one topic/issue assigned. In this way, the back-end of the GUI will query the existing topics' title and description in order to display them to the user. Then the user is able to select the topic/issue that he/she wants to consult, and after confirming the selection, only the comments of the selected topic/issue will be displayed in the property box.

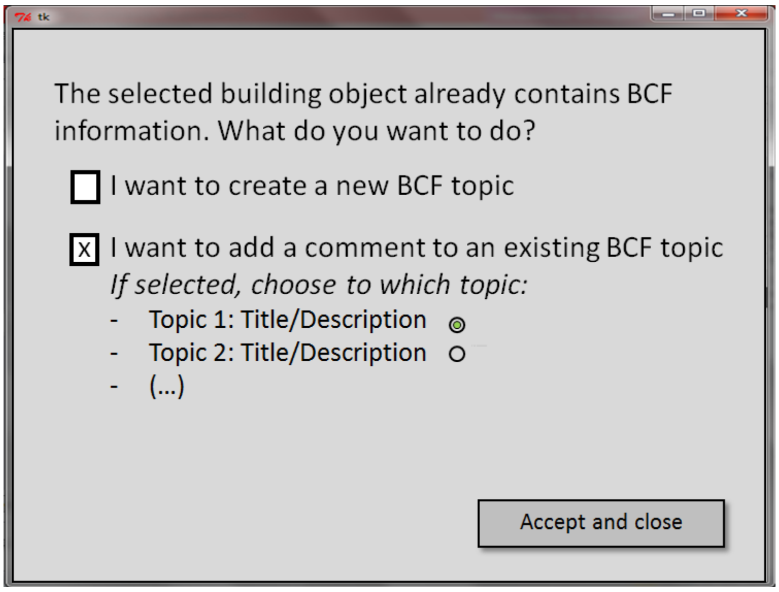

The selected building object has more than one BCF topic assigned. Please select from which topic you want to visualize the comments:

List of topics:

- Topic 1: Title/Description ☐
- Topic 2: Title/Description ☒
- Topic 3: Title/Description ☐
- (...)

Figure 59. Sketch of the proposed Graphical User Interface 1

On the other hand, as a consequence of relating a single building object to multiple topics/issues, an extra GUI when creating BCF data is also needed in order to allow the user to decide whether to create a new BCF topic or add comments to an existing BCF topic. Currently, when creating new BCF data (Create new BCF / RDF button), the programmed back-end displays an interface to add comments when BCF information is already related to the selected building object. For that reason, the suggested GUI (Figure 60) for this case would appear when creating new BCF information in order to ask the user if he/she wants to create a new topic or, on the contrary, he/she wants to add information to an existing BCF topic (displaying a list of the existing topics so that the user can select the BCF topic he wants to add comments to in the same way as when retrieving comments).


The selected building object already contains BCF information. What do you want to do?

☐ I want to create a new BCF topic

☒ I want to add a comment to an existing BCF topic
If selected, choose to which topic:

- Topic 1: Title/Description ◉
- Topic 2: Title/Description ○
- (...)

Figure 60. Sketch of the proposed Graphical User Interface 2

Secondly, storing all BCF information as RDF contexts in an SPARQL endpoint prevents users from exchanging BCF zip files that are exported as the output of their revision of the structural model. The approach to this particular point has been also addressed by other authors by developing an API capable of automatically synchronising BCF tasks and avoid file exchange ("BIMcollab," 2017; Linhard, K. & Steinmann, R., 2015; L. van Berlo & Krijnen, 2014). The point is that, without files, the limitation of managing a large amount of files (Thomas Froese, 2003) is overcome and automated (Krijnen & van Berlo, 2016; Y.-C. Lee et al., 2015). Therefore, time and effort are no longer spent organising the information, but in retrieving and filtering it.

In addition, this semantic approach allows users to access only the required information in the SPARQL endpoint. For instance, the "Referenced Document" or "Check object status" functionalities present clear examples of how information could be accessed by using SPARQL queries based on the building object's GUID. Comments that have been made about a building object or the connection to an explanatory document can be obtained instantly, saving users the time of having to look for it. However, the predefined SPARQL queries that have been designed for the identified use cases and functionalities limit the implementations' range to the ones that have been actually designed for this thesis. For that reason, knowledge in composing SPARQL queries as well as the RDFS schema is required by the user in case new functionalities (or at least different from the ones programmed for this thesis development) need to be developed. In this situation, the web application SPARKLIS (described in section 2.6 Semantic Web) represents a useful tool for users that do not know the RDFS schema and/or do not have much experience with the query language. This is because it allows them to explore the data of any SPARQL endpoint while building queries in natural language ("Sparklis - Semantic Web Standards," 2017).

On the other hand, the fact that the BCF information is linked to the IFC model by the GUID of the building objects in the IFC model makes the BCF information directly dependable on the existence of the building object in the IFC model. In other words, given the situation that a building object in the IFC model is erased, the connection to any related BCF information previously created will be broken and, therefore, it will not be retrievable anymore through the desktop viewer. However, it would be always possible to query GUIDs in BCFs that do not have an instance in the IFC model. Moreover, using only a single GUID as the identifier of a BCF issue could also limit the description of the issue in case more than one building object's GUID is involved in the description of the same issue.

Thirdly, the backwards and external compatibility with current BCF management tools represents an important part of the development carried out in this thesis as well. According to the findings of Leon van Berlo (L. A. H. M. van Berlo et al., 2012), it is difficult to make the different parties collaborating in a construction project work with the same tools. In fact, it is mentioned that *"engineers should be free to choose their own software tools in order to achieve a higher performance in the execution of their engineering tasks"* (L. A. H. M. van Berlo et al., 2012). For that reason, these two functionalities provide a bridge between the semantic approach carried out in this thesis and other strategies.

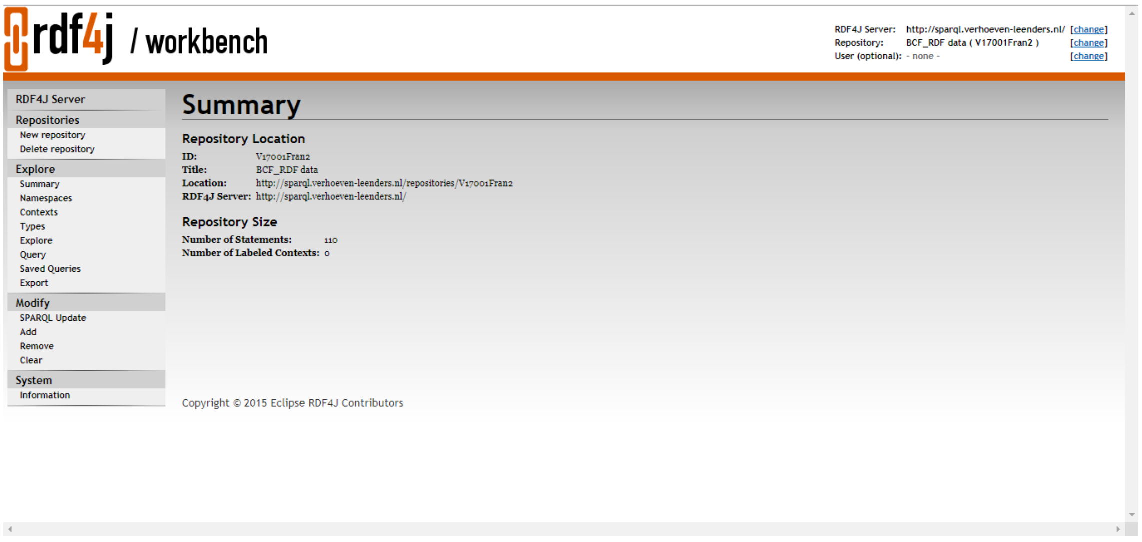
Fourthly, concerning management of requirements, the functionalities of the developed tool are more focused on providing communication between the structural engineer and the BIM modeller. However, the truth is that in combination with the linking of requirements from the project that this thesis is part of (Stancheva, 2017), the visualization of project requirements that are linked to the building objects in the IFC model is also possible. In this way, requirements management is closer to be in correlation with form exploration and not a front end task that is addressed marginally (Ozkaya & Akin, 2007).

Last but not least, the current design of the RDF structure is basically the equivalent of the XML markup file structure. The BCF information that is being modelled as RDF contexts by the developed prototype keeps the hierarchical relationships of the XSD schema's elements and attributes (section 7.7 Appendix VII: BCF Schema (XSD)). Moreover, each RDF context will contain only RDF triples that are related to the same building object's GUID, creating one RDF context per building object's GUID.

Although this way of structuring the information eases its visualization in the RDF repository, it is partially not necessary. The point of the semantic approach usage is that the user does not have to visualize an endless list of triples when looking for the information, but just retrieve the information that is useful for him by knowing how the information is structured and using the appropriate SPARQL queries.

For that reason, the design of differentiated RDF contexts could have been omitted (Figure 61 and Figure 62) as well as part of the hierarchy of the XML file (Figure 63). Specifically, the hierarchical relationship of the XML elements that can be modelled only once per "Markup" (File and Topic classes) could have been left out while it should be maintained for the XML elements that can have multiple instances (Comment and Viewpoints classes) in order to associate the "Markup" to those several instances. In such manner, the building object's GUID would be directly associated with the Markup class as a Data Type Property as well as the Data Type properties that were modelled inside the File and Topic's classes, making the last ones disappear. This will simplify the structure of the SPARQL queries by avoiding the hierarchical relationships between the Markup and the File and Topic classes. Concerning Comment and Viewpoints' classes, they would be modelled in the same way they are currently designed.

In case more than one Topic is created per building object given the hypothesis that the explained limitation is solved, then the Topic class would have to be modelled in the same way it was done originally because it will have multiple instances. However, even if Comment classes are related to their own Topic instead of the markup, comments will be still connected through the same building object's GUID, which will mix the information when retrieving it.



RDF4J Server: http://sparql.verhoeven-leenders.nl/ [change]
Repository: BCF_RDF data (V17001Fran2) [change]
User (optional): - none - [change]

Summary

Repository Location

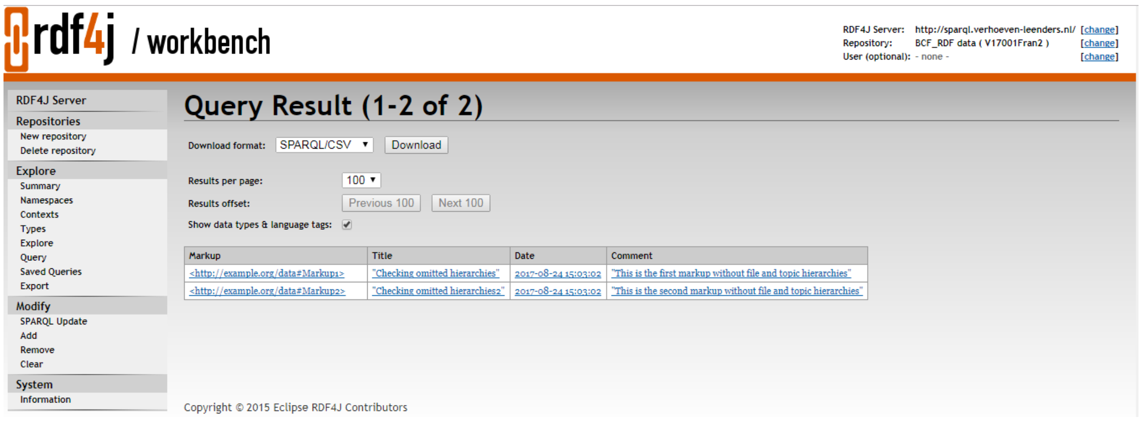
ID:	V17001Fran2
Title:	BCF_RDF data
Location:	http://sparql.verhoeven-leenders.nl/repositories/V17001Fran2
RDF4J Server:	http://sparql.verhoeven-leenders.nl/

Repository Size

Number of Statements:	110
Number of Labeled Contexts:	0

Copyright © 2015 Eclipse RDF4J Contributors

Figure 61. RDF repository with no contexts



RDF4J Server: http://sparql.verhoeven-leenders.nl/ [change]
Repository: BCF_RDF data (V17001Fran2) [change]
User (optional): - none - [change]

Query Result (1-2 of 2)

Download format: SPARQL/CSV

Results per page: 100

Results offset:

Show data types & language tags: ✓

Markup	Title	Date	Comment
<http://example.org/data#Markup1>	"Checking omitted hierarchies"	2017-08-24 15:03:02	"This is the first markup without file and topic hierarchies"
<http://example.org/data#Markup2>	"Checking omitted hierarchies2"	2017-08-24 15:03:02	"This is the second markup without file and topic hierarchies"

Copyright © 2015 Eclipse RDF4J Contributors

Figure 62. Query of triples in the RDF repository with no contexts

```
PREFIX fbf:<http://example.org/bcfschema#>
SELECT ?Markup ?Title ?Date ?Comment
WHERE {
        ?Markup fbf:hasTitle ?Title.
        ?Markup fbf:containsComment ?c.
        ?c fbf:hasComment ?Comment.
        ?c fbf:hasCommentDate ?Date.
        }
ORDER BY ?Markup ?Date
```

Figure 63. Query example without hierarchy

5. Conclusion

As a conclusion of the thesis, an answer to each of the initially considered research questions will be given based on the knowledge and insight gained during the research, development and validation stages.

1. Can Semantic Web technologies be integrated into a building structures design process for internal validation and attestation using BCF?

First of all, any internal design validation process requires that the actors involved in it are able to communicate in order to share feedback regarding the validation of the model. The prototype provides this functionality by first creating BCF information, then linking it to any building element (based on its GUID) in the IFC model and storing it in a SPARQL endpoint. Moreover, this information can be later retrieved through the desktop viewer as well by using SPARQL queries. It can be consulted as text, when retrieving the comments' history of the building element, or visualized in the viewer by colouring building elements, which will inform the users what elements are active, resolved, closed or have no information.

Secondly, in order to validate the design, the desktop viewer allows the user to create BCF information that will mark the building element as closed. This BCF closed status should be considered by the actors involved in the validation process as the final status that all elements in the model should be able to reach in order to consider an IFC model as validated.

Thirdly, the attestation part is covered automatically by storing any BCF information created as triples in the SPARQL endpoint, so that any building element will have an edition history. This building element's history can be consulted in the future in case some past decision about a property needs to be checked.

Last but not least, three different workflow examples (presented in section 7.8 Appendix VIII: Flowcharts of the developed tool) were elaborated and suggested as well to show how the prototype can be integrated into the design process of any project. These examples demonstrate how the structural engineer and BIM modeller can collaborate and make use of the prototype to revise and validate any structural model.

To conclude, the developed prototype is the result of researching the application linked data for the purpose of internal design validation and attestation. It can be confirmed that the studied approach successfully combines the semantic web technologies (RDF and SPARQL) with the use of BCF in the IfcOpenShell desktop viewer and improves the interoperability of the BCF data by connecting it with the IFC model. What is more, it provides any design process with the necessary functionalities to perform the validation and attestation of the BIM model.

2. Is BCF a capable tool for the management of requirements?

First of all, concerning management of requirements, it should be put into context that the requirements within the scope of this thesis refer to internal requirements that relate to the creation of a structural model. In other words, internal requirements have to do with changes and/ or requests in the structural model based on calculation or design properties that the model and its elements should have.

In this way, the research and development carried out in this thesis have managed to combine BCF with the Semantic Web technologies for the purpose of internal design validation attestation. Its combination has been proved not only to bring benefits in terms of data interoperability by connecting BCF data and the IFC model, but also provide the user with a framework (prototype) that keeps track of design changes and/or requests and internal communication.

A clear example is the possibility to visualize the stage of validation of every building element by connecting (and later retrieving using SPARQL queries) a BCF status to a building element in the IFC model, or to visualize a comments' history related to a building element.

On the other hand, external requirements concern the validation of the model but with regards to the information/requirements demanded by external parties that will base their work in future building project phases on the provided model. These external requirements can be also linked to the IFC model by using Semantic Web and Linked Data principles as it is described in Miryana Stancheva's thesis (Stancheva, 2017). Therefore, requirements can be retrieved as well if they have been previously linked to the building element.

To sum up, the BCF specification has been proven to manage and deal with internal requirements by connecting BCF data to the IFC model; while on the other hand, Miryana Stancheva's thesis (Stancheva, 2017) has managed to link external requirements to the IFC model as well. The truth is that, although the approach has divided external and internal requirements into two different theses, they both work with Linked Open Data; and therefore, there is an opportunity to combine the both internal and external requirements' ontologies.

3. What differences exist between the implementation of systems engineering management tools (Relatics) in building structure projects and civil projects?

First of all, the conditions demanded by clients in building structure's projects are not as strict as the one in infrastructure projects. Management of requirements is only carried out by using a systems engineering management tool such as Relatics in the civil division due to the strict framework demanded by some clients when it comes to civil projects. Basically, client's request this process so that they can make use of it for the maintenance and operations phases of the building lifecycle. It is convenient for them that requirements management has been implemented from the beginning of the design, otherwise, they will have to start from scratch in case they want to use it after the infrastructure is completed and delivered.

Moreover, for the same reason that clients demand a well-defined management system of requirements as a process requirement, the framework for Systems Engineering (SE) management within civil projects is more developed. On the other hand, SE is less developed within building structures projects because it is not demanded yet.

Secondly, using a system engineering management tool such as Relatics in building structures projects will suppose the structural engineer to repeat the work he is already doing regarding management of requirements by using the verification table (Excel sheet). It was observed during the research at V&L that the influence on the requirements' management for engineers in V&L is greater in civil projects due to the fact that the client requirements are delivered directly to them; while in a building structures project, requirements are already managed and defined by the architect in the architectural model. For that reason, it is more feasible to implement a systems engineering management tool from the beginning of the project design in civil projects than in building structure projects because requirements have already been handled and documented by other party when structural engineers start their work in the building structures project's design. In fact, requirements that concern a structural engineer are also different from the ones that are relevant for an architect.

In addition, regardless a system engineering management tool is implemented in a civil or a building structures' project, the truth is that usually these tools handle requirements separately with respect to the BIM models' design. Specifically with Relatics, management of requirements and modelling the building structural model are two different tasks. On the other hand, the semantic approach that is addressed within the scope of the project that this thesis is part of gets closer to the idea of connecting requirements management with form exploration supported by Ozkaya and Akin (Ozkaya & Akin, 2007).

4. What requirements apply specifically to projects in the building structures sector and which requirements have a general scope? And how will these requirements look like in a structural engineering context?

Regarding the structural engineering context, requirements that apply to it are gathered and classified in the elaborated System Breakdown Structure (SBS)/Requirements Breakdown Structure (RBS) matrix (section 7.2 Appendix II: Requirements' System Breakdown Structure). On the other hand, the matrix has also helped to understand that concepts such as project specific or general requirements are far from being precise to identify requirements' types. This is because it was observed by researching different construction projects (both civil and building structures) that client's requirements can take many forms and be accurate or fuzzy depending on what the client demands. In fact, the key to improve the perception of requirements is to understand that the same requirement can be general and accurate at the same time; and that it is the level of detail of the requirement what determines to which building element or system it can be applied.

The requirements' types are first classified according to what the requirement's objective is related to. This could be a function that needs to be realised by a building element or system (**Functional requirement**), to supporting functions or aspects of the system (**Aspect requirements**), to objects that have an impact on an element or system's shape, colour or strength (**Object requirements**), to interactions between the building elements or their environment (**System Interaction requirements**) and to activities that are necessary to be performed (**Process requirements**).

Moreover, each of these five types of requirements can be divided into six more different types depending on the level of detail of the construction project, except for the process requirements. These could be requirements regarding capacity or social security amongst others (**Policy requirements**), related to the functioning of the building structure (**Use requirements**), the performance of the structure (**Performance requirements**), the behaviour of the structure, sustainability, strength and stiffness (**Construction**

requirements), the choice of materials (**Building material requirements**) and the raw materials comprising the various building materials (**Raw material requirements**).

Based on these classifications and three construction projects, the obtained matrix for a structural engineering context presents: **load bearing (normal forces, shear force and moment)** and **stability (wind load, earthquake load and second order deflection)** as functional requirements, **material quality (strength and stiffness)** as object requirement, **prevention** (fire resistance and crash load) and **usability (deflection and crack width)** as aspect requirements; and **external system interactions (groundwater impact, soil impact, nuisance impact)** and **internal system interactions (element clashes, element connections)** as system interactions requirements.

5. How can BCF information be structured and stored using Semantic Web technologies (RDF)? Can this information be accessed and used for internal validation attestation of requirements (SPARQL endpoint)?

The first step to represent any type of information as RDF triples is to structure the information as an ontology. In such manner, the ontology that has been created for the purpose of this thesis is based completely on the structure of the BCF specification. This means that all the elements and attributes that the BCF specification contains in the XSD schema have been modelled as RDF triples, including hierarchal relationships (4.4 Ontology: BCF schema). Moreover, this equality is maintained for the storage as well, being each of the RDF contexts created in the RDF repository the equivalent of the traditional XML file. However, although this exact representation is how the current prototype models the information, is not completely necessary.

It was observed after the elaboration of the ontology that the classification of values like Filename, File date or File reference under the class File, and in turn the classification of the last one under the Header class, were not necessary when modelling the information. It is a fact that these elements contained under the File class refer to the same Markup, and that the distinction between File and Topic's sub-elements is done only for the purpose of structuring the information when visualizing it. For that reason, when represented as RDF triples, the Filename, File date or File reference and the equivalent sub-elements inside the class Topic can be directly related to the Markup class. In addition, triples for different issues can be added to the same repository without creating a new RDF context each time.

Concerning the recovery and usage of the information stored, several functionalities have been programmed to provide users with the identified use cases. Each of the functionalities has successfully retrieved specific BCF data from the repository using SPARQL queries and the queried information has been used for showing/hiding building elements in the model or directly printed in a property box so the user can see it.

On the whole, it is concluded that hierarchical relationships could be omitted for classes which do not have multiple instances within the same Markup (File and Topic). In fact, without those hierarchical relationships, the SPARQL queries that are currently used could be simpler. On the other hand, due to the fact that the version bcfXML v2 allows having more than one Comment and Viewpoint in the same Markup, a hierarchical relationship between the class and the values contained is needed so the values of each instance are gathered at once.

5.1 Limitations and recommendations (for possible follow-up research)

This section brings together several recommendations for further research and possible improvements of both the research and the developed prototype based on the insight gained through the elaboration on this thesis. Due to the scope of thesis and its limited time frame, some of these recommendations have not been developed or thoroughly covered in this thesis, and for that reason, they could be considered the starting point of upcoming investigations.

Firstly, the elaborated matrix which gathers requirements and building objects identified in building structure projects sets a template which has room for improvement. Because the research on management of requirements and its organization was carried out based on three different projects, the list of identified requirements could be increased/reduced as soon as more projects are inspected. What is more, the three studied projects were provided by the same company, so it is also possible that the obtained results are company oriented. For that reason, more projects coming from several companies would bring a more representative identification of requirements and building objects.

In addition, the same applies to the identified use cases and functionalities for the developed prototype. The exchange of information processes were analysed only for the three case studies (projects) provided by the company, so the workflow captured by the prototype could be considered company oriented as well. Therefore, studying design processes and information exchange processes in other companies could bring in more useful functionalities and/or use cases.

Secondly, the approach presented in this thesis has managed to represent BCF information in an RDF format. Amongst the most immediate advantages we find the fact that BCF zip files are no longer exported when creating BCF data, and that the BCF information is linked to the IFC model creating a small network of interconnected knowledge concerning BCF data and the IFC model. However, the approach presents more possibilities in the long run.

BCF information is now machine-processable and retrievable. What is more, there is an opportunity to combine it with data sets from other domains that use RDF as main data format as well in order to create a bigger network of interconnected knowledge. In this way, further research should be focused on the study of possible combinations of the BCF data with other resources or data sets (such as costs). In fact, a possible starting point would be to study the combination of the ontologies from this thesis and the one from Miryana Stancheva (Stancheva, 2017). As it is described in Figure 64, while this thesis deals with internal requirements, hers does with external ones. However, now that both types of requirements are represented as ontologies, their combination could be studied using the same Semantic Web and Linked Data principles.

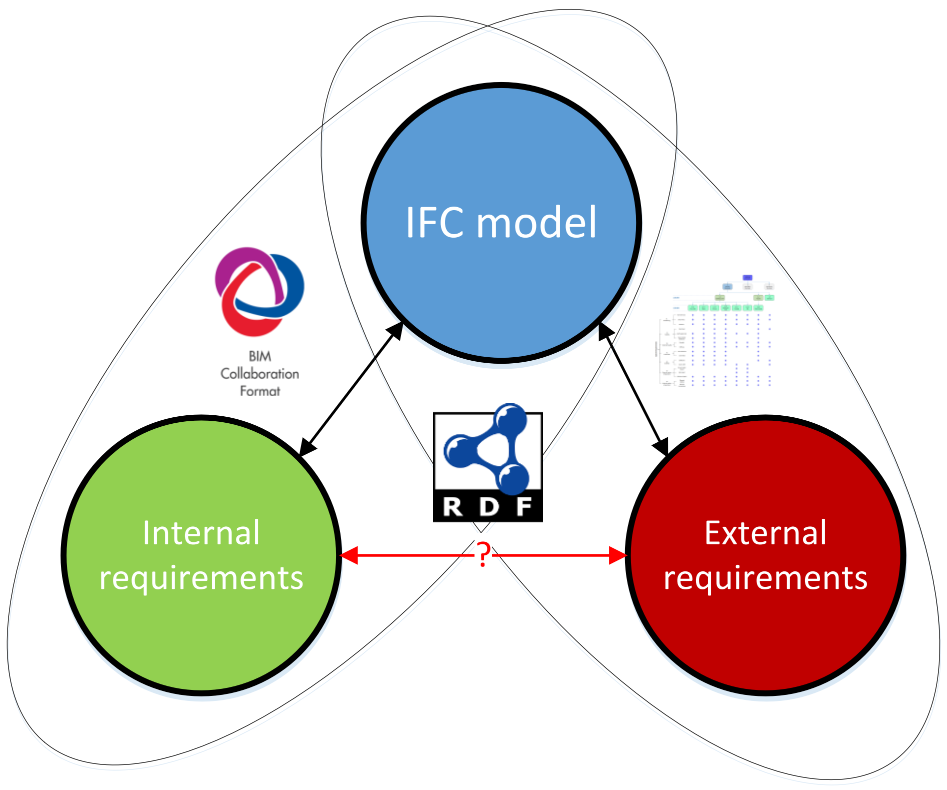


Figure 64. Combination of thesis for further research

Nevertheless, although the combinations of the BCF data can be done with as many data sets as domains are involved in the AEC industry, the data sets should be semantically represented and exchanged (e.g. RDF format) before the combination between domains is considered. To that end, governmental strategies that compel industry stakeholders to adapt their information exchange workflows in order to include RDF as a minimum requirement will be necessary in the long run (such as the UK's government strategy for Level 2 BIM status adoption by 2016). For instance, governments could give extra points to a company that counts on RDF for knowledge management when awarding a project in a tendering process.

Industry stakeholders should be introduced to the benefits that the semantic approach could bring as well, which are currently considered complementary to BIM softwares and/or undisclosed to some of them. Promoting pilot projects between universities and companies that show the benefits to other companies within the AEC field in terms of time saved when consulting information or interoperability could be an option too. In any case, the means would include collaboration between private and public sectors, industry bodies, software developers and researchers.

Thirdly, concerning the ontology development, there are some suggestions about the modelling of classes, properties and the relationships between them. These suggestions are considered relevant because they had an influence in the way the ontology was modelled.

- First of all, in case a property domain and/or range apply to several classes, it is important to keep a relation 1:1 regarding the domain and range relation and never include several classes as a domain or range of the same property. Instead, it is advisable to create a superclass that contains all classes, and include that super class as domain and/or range. Otherwise, different classes considered as the domain of the same property will intersect each other instead of being united.
- Secondly, the hierarchical relationship between classes in represented as an “is-a” relation between the class and the sub-class, and not a “belongs to” relation. In other words, you want to have a class A and a subclass B if the instances of classes A and B are the same and you want B to inherit the properties of A. On the other hand, when you have a hierarchical relation relationship between classes but still they are not the same, then each class should be modelled independently and be related to each other through another property. The same applies for properties and sub-properties relationships.
- Thirdly, classes should be named with a capital first letter and they always should be singular. Concerning properties, they are named with a lower case first letter.
- Last but not least, not all elements coming from the original XSD schema are translated as classes in the ontology. Only the “Complex Type” elements from the XSD schema were modelled as a class, using “Data Type” properties to describe the rest of the elements. Moreover, “Object Type” properties were used to define the hierarchy of the XSD schema, creating one property for each hierarchical relationship between two classes in order to maintain the 1:1 relation regarding domain and ranges.

Fourthly, the back-end of the prototype was written and developed while learning from scratch how the Python programming language works. For that reason, the final version of the prototype’s code (section 7.5 Appendix V: Application code (Python)) can be improved. For instance, the text boxes where users can introduce information do not provide error handling. This means that if “integers” are written were “strings” are supposed to be written, or vice versa, the prototype will not be able to inform the user that the information has not been correctly introduced. Moreover, there are other programming issues that could be easily overcome by an ICT programmer, such as the fact that you need to manually introduce the name of users or BCF responsibilities. In a professional system, this would be provided by a user administrator.

On the other hand, the highlighted and discussed limitations of the developed prototype (section 4.7 Prototype limitations and section 4.8 Prototype discussion) could be the beginning for further upgrades of the prototype as well.

An interesting point is related to the creation and association of more than one BCF issue per building object. It was explained in section 4.7 Prototype limitations that if more than one BCF issue per building element exists (connected through the same GUID); the retrieved information would be mixed. In fact, even if comments are associated with the Topic instead of the Markup, still BCF data will be mixed because both comments are linked to the building object with the same GUID. So the proposed and sketched solution in section 4.8 Prototype discussion suggests a Graphical User Interface (GUI) that allows the user to add a second subidentifier that will filter the comments appearing in the property box depending on to which topic they belong to. The GUI's back-end will be able to query (SPARQL) and display the current list topics (title and description) that are related to a building element, and then use the ID of the selected topic to query the comments associated with it.

It was also explained in section 4.8 Prototype discussion that as a consequence of relating one building object to multiple topics/issues, a second GUI will be necessary when creating BCF data. This is because the current back-end only allows adding comments to the existing BCF issue in case the selected building object contains already information. So the proposed and sketched solution would ask the user first if he/she wants to create a new BCF topic or add information to an existing one, and in the last case, the back-end of the GUI will query and display a list of the existing topics so the user can select to which the information should be added (similarly to the first GUI when filtering comments).

Another discussed limitation is the fact that it was only possible to create BCF information by selecting building elements one by one. This could be overcome by adding loops in each function that has been designed in the prototype. Currently, when a function is run, the variable in the code is substituted by the GUID of the building element that is selected (guid_selection) in the viewer. In case the GUID selection contains more than one GUID, then a loop in the function will run the same functionality for every GUID included in the GUID selection.

The last limitation in section 4.7 Prototype limitations concerns the attachment of snapshots and viewpoints of the building object that is selected in the viewer. As it was explained, creating a visual description of the element that the BCF issue is related to is not possible with the developed prototype. However, including this functionality should be considered for practical purposes in the design process. Model checking software applications (SMC) already allow the attachment of this visual information when creating and exporting BCF data. The snapshot is stored as a PNG file in the same markup folder that contains the XML file with the BCF information, and the viewpoint's coordinates are stored in the same folder as well, but as a different XML file.

Research on the Open Cascade documentation ("Documentation | OPEN CASCADE," 2017) will help to design the back-end that will capture and generate the snapshot as a PNG file, and the 3d viewer coordinates as an XML file the moment that the BCF is created. However, once this information is generated, it is necessary to study how it would be stored as RDF triples and later retrieved or serialized. On the one hand, the PNG file could be stored in the FTP-server similarly to the way PDF files were stored when referencing a documents, while the coordinates of the viewpoint can be stored in the same way other elements (such as filename or file date) were represented as RDF triples. On the other hand, in case RDF triples are serialized as a XML markup file inside the BCF ZIP file, the PNG file has to be serialized and included within the same folder that the XML markup is, so that any BCF management tool can access it. Concerning the coordinates, it would be necessary to program (Python) the serialization of another XML file containing them in the same way the XML markup contains the BCF data.

Finally, although it is not mentioned as a limitation in section 4.7 Prototype limitations, the organization of RDF triples was done by storing triples at the same repository (called “VL17001Fran”). As the IFC model used for the validation of the tool was also the same, then all triples inside the repository were related to the same IFC model. However, for further development of the prototype and practical purposes, it is advisable to create a preliminary step where the user can select the repository that he/she would like to store the future created information. In this way, the triples could be stored in the repositories by projects.

6. References

Abanda, F. H., Tah, J. H. M., & Keivani, R. (2013). Trends in built environment semantic Web applications: Where are we today? *Expert Systems with Applications*, *40*(14), 5563–5577. https://doi.org/10.1016/j.eswa.2013.04.027

Alexander Kossiakoff, William N. Sweet, Samuel J. Seymour, & Steven M. Biemer. (2011). *Systems Engineering Principles and Practice*. A JOHN WILEY & SONS, INC. PUBLICATION.

Allemang, D., & Hendler, J. (2011a). Chapter 3 - RDF—The basis of the Semantic Web. In *Semantic Web for the Working Ontologist (Second Edition)* (pp. 27–50). Boston: Morgan Kaufmann. https://doi.org/10.1016/B978-0-12-385965-5.10003-2

Allemang, D., & Hendler, J. (2011b). Chapter 5 - Querying the Semantic Web—SPARQL. In *Semantic Web for the Working Ontologist (Second Edition)* (pp. 61–112). Boston: Morgan Kaufmann. https://doi.org/10.1016/B978-0-12-385965-5.10005-6

Allemang, D., & Hendler, J. (2011c). Chapter 7 - RDF schema. In *Semantic Web for the Working Ontologist (Second Edition)* (pp. 125–152). Boston: Morgan Kaufmann. https://doi.org/10.1016/B978-0-12-385965-5.10007-X

Arto Kiviniemi, & Martin Fischer. (2004). Requirements Management Interface to Building Product Models.

Asier Mediavilla, José Luis Izkara, & Iñaki Prieto. (2015). HOLISTEEC – Plataforma colaborativa en la nube basada en BIM para el diseño de edificios energéticamente eficientes. Retrieved May 11, 2017, from https://www.researchgate.net/publication/308304305_HOLISTEEC_-_Plataforma_colaborativa_en_la_nube_basada_en_BIM_para_el_diseno_de_edificios_energeticamente_eficientes

Autodesk. (2017). Autodesk. Retrieved April 3, 2017, from http://www.autodesk.com/solutions/bim/overview

Azhar, S., Hein, M., & Sketo, B. (2014). Building Information Modeling (BIM): Benefits, Risks and Challenges. *ResearchGate*. Retrieved from https://www.researchgate.net/publication/237569739_Building_Information_Modeling_BIM_Benefits_Risks_and_Challenges

Beetz, J., Coebergh, W., Botter, R., Zlatanova, S., & De Laat, R. (2014). Interoperable data models for infrastructural artefacts - a novel IFC extension method using RDF vocabularies exemplified with quay wall structures for harbors. *eWork and eBusiness in Architecture, Engineering and Construction: ECPPM 2014, Vienna, (Austria), 17-19th Sept., 2014; Authors Version*. Retrieved from https://repository.tudelft.nl/islandora/object/uuid%3Abb9a7dff-52c7-4aaf-a6b8-898432270620

Beetz, Jakob, Berlo, L. van, Laat, R. de, & Bonsma, P. (2011). Advances in the development and application of an open source model server for building information. Retrieved

April 3, 2017, from http://repository.tue.nl/a7689318-b00c-4b2d-9ede-4fd7d9cb5895

Beetz, Jakob, van Leeuwen, J., & de Vries, B. (2009). IfcOWL: A case of transforming EXPRESS schemas into ontologies. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, *23*(01), 89. https://doi.org/10.1017/S0890060409000122

Berlo, L. A. H. M. van, Beetz, J., Bos, P., Hendriks, H., & Tongeren, R. C. J. van. (2012). Collaborative engineering with IFC : new insights and technology. Retrieved April 3, 2017, from http://repository.tue.nl/fc4f57d4-6dfb-45b0-9170-41faeb19664c

Berlo, L. van, Derks, G., Pennavaire, C., & Bos, P. (2015). Collaborative Engineering with IFC: common practice in the Netherlands.

BIM protocol generator. (2014, May 24). Retrieved April 4, 2017, from http://bimprotocolgenerator.com/about/

BIMcollab. (2017). Retrieved August 23, 2017, from http://www.bimcollab.com/en/BIMcollab/BIMcollab

Bouzidi, K. R., Fies, B., Faron-Zucker, C., Zarli, A., & Thanh, N. L. (2012). Semantic Web Approach to Ease Regulation Compliance Checking in Construction Industry. *Future Internet*, *4*(3), 830–851. https://doi.org/10.3390/fi4030830

BuildingSMART. (2011). Information Delivery Manuals — buildingSMART. Retrieved April 3, 2017, from http://iug.buildingsmart.org/idms/

BuildingSMART. (2017). BuildingSMART. Retrieved April 3, 2017, from http://buildingsmart.org/

Curry, E., O’Donnell, J., Corry, E., Hasan, S., Keane, M., & O’Riain, S. (2013). Linking building data in the cloud: Integrating cross-domain building data using linked data. *Advanced Engineering Informatics*, *27*(2), 206–219. https://doi.org/10.1016/j.aei.2012.10.003

Documentation | OPEN CASCADE. (2017). Retrieved September 12, 2017, from https://www.opencascade.com/content/documentation

Edward Corry, James O’Donnell, Edward Curry, Daniel Coakley, Pieter Pauwels, & Marcus Keane. (2014, June). Using semantic web technologies to access soft AEC data - ScienceDirect. Retrieved July 19, 2017, from http://www.sciencedirect.com/science/article/pii/S1474034614000366

Elghamrawy, T., & Boukamp, F. (2008). A vision for a framework to support management and learning from construction problems (pp. 1–10). Presented at the Improving the management of Construction Projects through IT adoption, University of Talca. Retrieved from https://researchbank.rmit.edu.au/view/rmit:13820

Elghamrawy, Tarek, & Boukamp, F. (2010). Managing construction information using RFID-based semantic contexts. *Automation in Construction*, *19*(8), 1056–1066. https://doi.org/10.1016/j.autcon.2010.07.015

García, A. C. B., Kiviniemi, A., & Ekstrom, M. (2003, November 21). Building a project ontology with extreme collaboration and virtual design and construction. Retrieved April 5, 2017, from https://www.researchgate.net/publication/222524380_Building_a_project_ontology_with_extreme_collaboration_and_virtual_design_and_construction

Gonçal Costa, & Leandro Madrazo. (2015, June). Connecting building component catalogues with BIM models using semantic technologies: An application for precast concrete components. Retrieved July 20, 2017, from https://www.researchgate.net/publication/279520321_Connecting_building_component_catalogues_with_BIM_models_using_semantic_technologies_An_application_for_precast_concrete_components

Grilo, A., & Jardim-Goncalves, R. (2010). Value proposition on interoperability of BIM and collaborative working environments. *ResearchGate*. https://doi.org/http://dx.doi.org/10.1016/j.autcon.2009.11.003

Gu, N., & London, K. (2010). Understanding and facilitating BIM adoption in the AEC industry. *Automation in Construction*, *19*(8), 988–999. https://doi.org/10.1016/j.autcon.2010.09.002

Hans A. Schevers, John Mitchell, Paul Akhurst, & Chris Linning. (2007, July). Towards digital facility modelling for Sydney Opera House using IFC and semantic web technology. Retrieved July 19, 2017, from https://www.researchgate.net/publication/27483940_Towards_digital_facility_modelling_for_Sydney_Opera_House_using_IFC_and_semantic_web_technology

Hans Hoeber, & Daan Alsem. (2016). Life-cycle information management using open-standard BIM. Retrieved April 15, 2017, from https://www.researchgate.net/publication/310667357_Life-cycle_information_management_using_open-standard_BIM

Hitchcock, R. (1995). Improving Building Life-Cycle Information Management Through Documentation and Communication of Project Objectives.

INCOSE. (2017). INCOSE. Retrieved April 3, 2017, from http://www.incose.org/AboutSE/WhatIsSE

J. Beetz. (2014). A Scalable Network of Concept Libraries Using Distributed Graph Databases. *Computing in Civil and Building Engineering (2014)*. https://doi.org/10.1061/9780784413616.071

Jakob Beetz, Bauke de Vries, & Jos van Leeuwen. (2007). RDF-based distributed functional part specifications for the facilitation of service-based architectures.

Jakob Beetz, Jos P. van Leeuwen, & Bauke de Vries. (2006, January). Towards a topological reasoning service for IFC-based building information models in a Semantic Web context. Retrieved July 17, 2017, from https://www.researchgate.net/publication/228613888_Towards_a_topological_reasoning_service_for_IFC-based_building_information_models_in_a_Semantic_Web_context

Jeff Wix, & Jan Karlshøj. (2010, May 12). Information Delivery Manual Guide to Components and Development Methods. Retrieved from http://iug.buildingsmart.org/idms/development/IDMC_004_1_2.pdf

Krijnen, T., & van Berlo, L. (2016). Methodologies for requirement checking on building models.

Kvan, T. (2000). Collaborative design: what is it? *Automation in Construction*, *9*(4), 409–415. https://doi.org/10.1016/S0926-5805(99)00025-4

L.A.H.M. van Berlo, F. Bomhof, & G. Korpershoek. (2014). Creating the Dutch National BIM Levels of Development.

Lee, S.-K., Kim, K.-R., & Yu, J.-H. (2014). BIM and ontology-based approach for building cost estimation. *Automation in Construction*, *41*, 96–105. https://doi.org/10.1016/j.autcon.2013.10.020

Lee, Y.-C., Eastman, C. M., & Lee, J.-K. (2015). Validations for ensuring the interoperability of data exchange of a building information model. *ResearchGate*. https://doi.org/http://dx.doi.org/10.1016/j.autcon.2015.07.010

Linhard, K., & Steinmann, R. (2015). BIM-collaboration processes – from fuzziness to practical implementation. Retrieved from https://books.google.nl/books?hl=es&lr=&id=uemsBAAAQBAJ&oi=fnd&pg=PA27&dq=BIM-collaboration+processes+%E2%80%93+from+fuzziness+to+practical+implementation.&ots=aetkG6vFgG&sig=9ae1_TIGwZlajsjynnZpIbiy02w#v=onepage&q=BIM-collaboration%20processes%20%E2%80%93%20from%20fuzziness%20to%20practical%20implementation.&f=false

Matthias Weise, & Pieter Pauwels. (2015). Best practices for publishing and linking BIM data: Scoping of IFC models.

Miettinen, R., & Paavola, S. (2014). Beyond the BIM utopia: Approaches to the development and implementation of building information modeling. *Automation in Construction*, *43*, 84–91. https://doi.org/10.1016/j.autcon.2014.03.009

Natalya F. Noy, & Deborah L. McGuinness. (2001, March). Ontology Development 101: A Guide to Creating Your First Ontology. Retrieved August 1, 2017, from http://www-ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness-abstract.html

Neff, G., Fiore-Silfvast, B., & Dossick, C. S. (2010). A Case Study of the Failure of Digital Communication to Cross Knowledge Boundaries in Virtual Construction. *Information, Communication & Society*, *13*(4), 556–573. https://doi.org/10.1080/13691181003645970

Niknam, M., & Karshenas, S. (2015). Sustainable Design of Buildings using Semantic BIM and Semantic Web Services. *Procedia Engineering*, *118*, 909–917. https://doi.org/10.1016/j.proeng.2015.08.530

Niknam, M., & Karshenas, S. (2017). A shared ontology approach to semantic representation of BIM data. *Automation in Construction*, *80*, 22–36. https://doi.org/10.1016/j.autcon.2017.03.013

Ozkaya, I., & Akin, Ö. (2007). Tool support for computer-aided requirement traceability in architectural design: The case of DesignTrack. *Automation in Construction*, *16*(5), 674–684. https://doi.org/10.1016/j.autcon.2006.11.006

Pan, J., Anumba, C., & Ren, Z. (2004). Potential application of the semantic web in construction. *Proceedings of the 20th Annual Conference of the Association of Researchers in Construction Management (ARCOM)*, 923–929.

Pauwels, P., Van Deursen, D., Verstraeten, R., De Roo, J., De Meyer, R., Van de Walle, R., & Van Campenhout, J. (2011). A semantic rule checking environment for building performance checking. *Automation in Construction*, *20*(5), 506–518. https://doi.org/10.1016/j.autcon.2010.11.017

Pauwels, Pieter, & Terkaj, W. (2016). EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology. *Automation in Construction*, *63*(Supplement C), 100–133. https://doi.org/10.1016/j.autcon.2015.12.003

Pauwels, Pieter, Zhang, S., & Lee, Y.-C. (2017). Semantic web technologies in AEC industry: A literature overview. *Automation in Construction*, *73*, 145–165. https://doi.org/10.1016/j.autcon.2016.10.003

Pieter Pauwels, & Davy Van Deursen. (2012, March). IFC-to-RDF: Adaptation, Aggregation and Enrichment. Retrieved July 17, 2017, from https://www.researchgate.net/publication/266478243_IFC-to-RDF_Adaptation_Aggregation_and_Enrichment

Pieter Pauwels, Seppo Törmä, Jakob Beetz, & T. Liebich. (2015, September). Linked Data in Architecture and Construction. Retrieved July 18, 2017, from https://www.researchgate.net/publication/281612334_Linked_Data_in_Architecture_and_Construction

Relatics. (2017). Relatics. Retrieved January 27, 2017, from https://www.relatics.com/

Rezgui, Y., Boddy, S., Wetherill, M., & Cooper, G. (2011). Past, present and future of information and knowledge sharing in the construction industry: Towards semantic service-based e-construction? *Computer-Aided Design*, *43*(5), 502–515. https://doi.org/10.1016/j.cad.2009.06.005

Riet, M. van de. (2016, February). *Semantic model enrichment for BIM-enabled risk-based operation and maintenance. A case study approach with Industry Foundation Classes*.

Shadbolt, N., Berners-Lee, T., & Hall, W. (2006). The Semantic Web Revisited. *IEEE Intelligent Systems*, *21*(3), 96–101. https://doi.org/10.1109/MIS.2006.62

Shen, L., & Chua, D. (2011). Application of Building Information Modeling (BIM) and Information Technology (IT) for Project Collaboration. *International Conference on Engineering, Project, and Production Management (EPPM)*, 67–76.

Shen, W., Hao, Q., Mak, H., Neelamkavil, J., Xie, H., Dickinson, J., … Xue, H. (2010). Systems integration and collaboration in architecture, engineering, construction, and facilities management: A review. *Advanced Engineering Informatics*, *24*(2), 196–207. https://doi.org/10.1016/j.aei.2009.09.001

Simeone, D., & Cursi, S. (2016). The role of semantic enrichment in Building Information Modelling. *Tema:Technology,Engineering,Materials and Architecture*, *2*(2), 22–30. https://doi.org/10.17410/tema.v2i2.105

Sparklis | DBpedia. (2017). Retrieved September 6, 2017, from http://wiki.dbpedia.org/projects/sparklis

Sparklis - Semantic Web Standards. (2017). Retrieved September 5, 2017, from https://www.w3.org/2001/sw/wiki/Sparklis

Stancheva, M. (2017, September). *Improving the management of structural engineering requirements in the design phase. Linking project requirements to BIM, based on the Semantic Web and Linked Data principles*.

Svetel, I., & Pejanović, M. (2010). The Role of the Semantic Web for Knowledge Management in the Construction Industry. *Informatica*, *34*(3). Retrieved from http://www.informatica.si/index.php/informatica/article/view/307

Tamer E. El-Diraby. (2013). Domain Ontology for Construction Knowledge. *Journal of Construction Engineering and Management*, *139*(7), 768–784. https://doi.org/10.1061/(ASCE)CO.1943-7862.0000646

Thomas Froese. (2003). Future directions for IFC-based interoperability.

Treldal, N., Parsianfar, H., & Karlshøj, J. (2016, August). Using BCF as a mediator for task management in building design. Retrieved April 20, 2017, from https://www.researchgate.net/publication/308113942_USING_BCF_AS_A_MEDIATOR_FOR_TASK_MANAGEMENT_IN_BUILDING_DESIGN

van Berlo, L., & Krijnen, T. (2014). Using the BIM Collaboration Format in a Server Based Workflow. *Procedia Environmental Sciences*, *22*, 325–332. https://doi.org/10.1016/j.proenv.2014.11.031

Vanlande, R., Nicolle, C., & Cruz, C. (2008). IFC and building lifecycle management. *Automation in Construction*, *18*(1), 70–78. https://doi.org/10.1016/j.autcon.2008.05.001

Ven, N. van de. (2017, February). *Mutation management in BIM models during Operations & Maintenance. Managing Operations & Maintenance building models with the use of Industry Foundation Classes and Linked Data*.

Wicaksono, H., Rogalski, S., & Kusnady, E. (2010). Knowledge-based intelligent energy management using building automation system. In *2010 Conference Proceedings IPEC* (pp. 1140–1145). https://doi.org/10.1109/IPECON.2010.5696994

Wicaksono, Hendro, Dobreva, P., Häfner, P., & Rogalski, S. (2013). Ontology Development towards Expressive and Reasoning-enabled Building Information Model for an Intelligent Energy Management System. *5th International Conference on Knowledge*

Engineering and Ontology Development, KEOD 2013; Vilamoura, Algarve; Portugal; 19 September 2013 through 22 September 2013. Retrieved from https://publikationen.bibliothek.kit.edu/1000040882

Xu, X., Ma, L., & Ding, L. (2014). A Framework for BIM-Enabled Life-Cycle Information Management of Construction Project. *International Journal of Advanced Robotic Systems*, *11*(8), 126. https://doi.org/10.5772/58445

Zarli, A., A.Yurchyshyna, Le Thanh, N., & Faron Zucker, C. (2008). Towards an ontology-based approach for formalizing expert knowledge in the conformity-checking model in construction. In *eWork and eBusiness in Architecture, Engineering and Construction* (Vols. 1–0, pp. 447–456). Taylor & Francis. https://doi.org/10.1201/9780203883327.ch50

Zhang, C., & Beetz, J. (2014, September). Model View Checking: Automated Validation for IFC Building Models. Retrieved August 17, 2017, from https://www.researchgate.net/publication/266326324_Model_View_Checking_Automated_Validation_for_IFC_Building_Models

7. Appendices

7.1 Appendix I: Concept matrix

The concepts contained in the concept matrix are enumerated next:

1. Building Information Modeling (BIM)
2. IFC and interoperability
3. BIM implementation and adoption
4. Collaboration technologies and Information Delivery Manual (IDM)
5. BIM Collaboration Format (BCF)
6. Requirements, issues, tasks and life-cycle information management
7. Semantic web, RDF and SPARQL

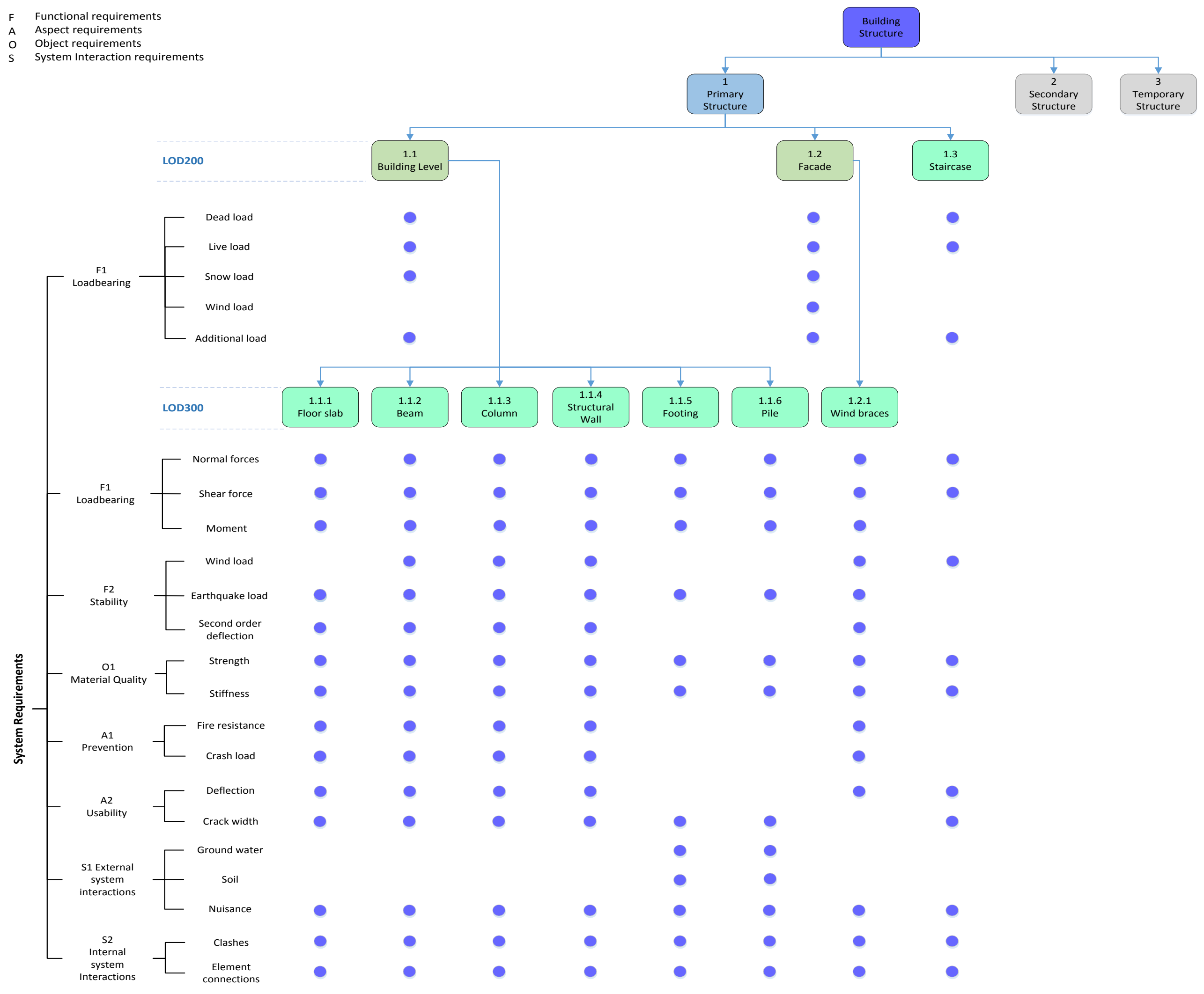
Table 5. References' concept matrix

Articles	**Concepts**						
	1	**2**	**3**	**4**	**5**	**6**	**7**
Beyond the BIM utopia: Approaches to the development and implementation of building information modeling (2014)	X		X				
Collaborative engineering with IFC : new insights and technology (2012)	X	X	X	X			
Future directions for IFC-based interoperability (2003)		X		X			
Life-cycle information management using open-standard BIM (2016)	X	X		X		X	
Value proposition on interoperability of BIM and collaborative working environments (2010)	X	X					
Collaborative Engineering with IFC: common practice in the Netherlands (2015)	X	X	X		X		
A Case Study of the Failure of Digital Communication to Cross Knowledge Boundaries in Virtual Construction (2010)	X		X				
Systems integration and collaboration in architecture, engineering, construction, and facilities management: A review (2010)	X	X	X	X			
Understanding and facilitating BIM adoption in the AEC industry (2010)	X		X				
Methodologies for requirement checking on building models (2016)	X	X					
Validations for ensuring the interoperability of data exchange of a building information model (2015)	X	X		X			
Collaborative design: what is it? (2000)				X			
HOLISTEEC – Plataforma colaborativa en la nube basada en BIM para el diseño de edificios energéticamente eficientes (2015)	X	X		X	X		
Information Delivery Manual Guide to Components and Development Methods (2010)				X			
Creating the Dutch National BIM Levels of Development (2014)	X	X					
Improving Building Life-Cycle Information Management Through Documentation and Communication of Project Objectives (1995)						X	

A Framework for BIM-Enabled Life-Cycle Information Management of Construction Project (2014)	X		X	X		X	
Advances in the development and application of an open source model server for building information (2011)	X	X		X	X		
BIM-collaboration processes – from fuzziness to practical implementation (2015)	X	X	X		X		
Using the BIM Collaboration Format in a Server Based Workflow (2014)	X				X		
Using BCF as a mediator for task management in building design (2016)	X			X	X	X	
Building a project ontology with extreme collaboration and virtual design and construction (2003)				X		X	
Requirements Management Interface to Building Product Models (2004)				X		X	
Tool support for computer-aided requirement traceability in architectural design: The case of DesignTrack (2007)				X		X	
Trends in built environment semantic Web applications: Where are we today? (2013)							X
Systems Engineering Principles and Practice (2011)						X	
Chapter 3 - RDF—The basis of the Semantic Web. In Semantic Web for the Working Ontologist (2011)							X
Chapter 5 - Querying the Semantic Web—SPARQL. In Semantic Web for the Working Ontologist (2011)							X
Building Information Modeling (BIM): Benefits, Risks and Challenges (2014)	X		X				
Interoperable data models for infrastructural artefacts - a novel IFC extension method using RDF vocabularies exemplified with quay wall structures for harbors (2014)							X
IfcOWL: A case of transforming EXPRESS schemas into ontologies (2009)							X
Semantic Web Approach to Ease Regulation Compliance Checking in Construction Industry (2012)							X
Information Delivery Manuals — buildingSMART (2011)				X			
Using semantic web technologies to access soft AEC data (2014)							X
A vision for a framework to support management and learning from construction problems (2008)							X
Managing construction information using RFID-based semantic contexts (2010)							X
Connecting building component catalogues with BIM models using semantic technologies: An application for precast concrete components (2015)							X
Towards digital facility modelling for Sydney Opera House using IFC and semantic web technology (2007)							X
A Scalable Network of Concept Libraries Using Distributed Graph Databases (2014)							X
RDF-based distributed functional part specifications for the facilitation of service-based architectures (2007)							X
Towards a topological reasoning service for IFC-based building information models in a Semantic Web context (2006)							X
BIM and ontology-based approach for building cost estimation (2014)	X						X
Best practices for publishing and linking BIM data: Scoping of IFC models (2015)	X						X
Sustainable Design of Buildings using Semantic BIM and Semantic Web Services (2015)							X
A shared ontology approach to semantic representation of BIM data (2017)							X

Potential application of the semantic web in construction (2004)							X
A semantic rule checking environment for building performance checking (2011)							X
Semantic web technologies in AEC industry: A literature overview (2017)							X
IFC-to-RDF: Adaptation, Aggregation and Enrichment (2012)							X
Linked Data in Architecture and Construction (2015)							X
Past, present and future of information and knowledge sharing in the construction industry: Towards semantic service-based e-construction? (2011)							X
The Semantic Web Revisited (2006)							X
Application of Building Information Modeling (BIM) and Information Technology (IT) for Project Collaboration (2011)	X						X
The role of semantic enrichment in Building Information Modelling (2016)							X
The Role of the Semantic Web for Knowledge Management in the Construction Industry (2010)							X
Domain Ontology for Construction Knowledge (2013)							X
Knowledge-based intelligent energy management using building automation system (2010)							X
Ontology Development towards Expressive and Reasoning-enabled Building Information Model for an Intelligent Energy Management System (2013)							X
Towards an ontology-based approach for formalizing expert knowledge in the conformity-checking model in construction (2008)							X

7.2 Appendix II: Requirements’ System Breakdown Structure



		1.1 Building Level	1.2 Facade	1.3 Staircase
F1 Loadbearing	Dead load	●	●	●
	Live load	●	●	●
	Snow load	●	●	
	Wind load		●	
	Additional load	●	●	●

		1.1.1 Floor slab	1.1.2 Beam	1.1.3 Column	1.1.4 Structural Wall	1.1.5 Footing	1.1.6 Pile	1.2.1 Wind braces	
F1 Loadbearing	Normal forces	●	●	●	●	●	●	●	●
	Shear force	●	●	●	●	●	●	●	●
	Moment	●	●	●	●	●	●	●	
F2 Stability	Wind load		●	●	●			●	●
	Earthquake load	●	●	●	●	●	●	●	
	Second order deflection	●	●	●	●			●	
O1 Material Quality	Strength	●	●	●	●	●	●	●	●
	Stiffness	●	●	●	●	●	●	●	●
A1 Prevention	Fire resistance	●	●	●	●			●	
	Crash load	●	●	●	●			●	
A2 Usability	Deflection	●	●	●	●			●	●
	Crack width	●	●	●	●	●	●		●
S1 External system interactions	Ground water					●	●		
	Soil					●	●		
	Nuisance	●	●	●	●	●	●	●	●
S2 Internal system Interactions	Clashes	●	●	●	●	●	●	●	●
	Element connections	●	●	●	●	●	●	●	●

Figure 65. Requirements’ System Breakdown Structure

7.3 Appendix III: Process map MAVO project (Communication workflow)

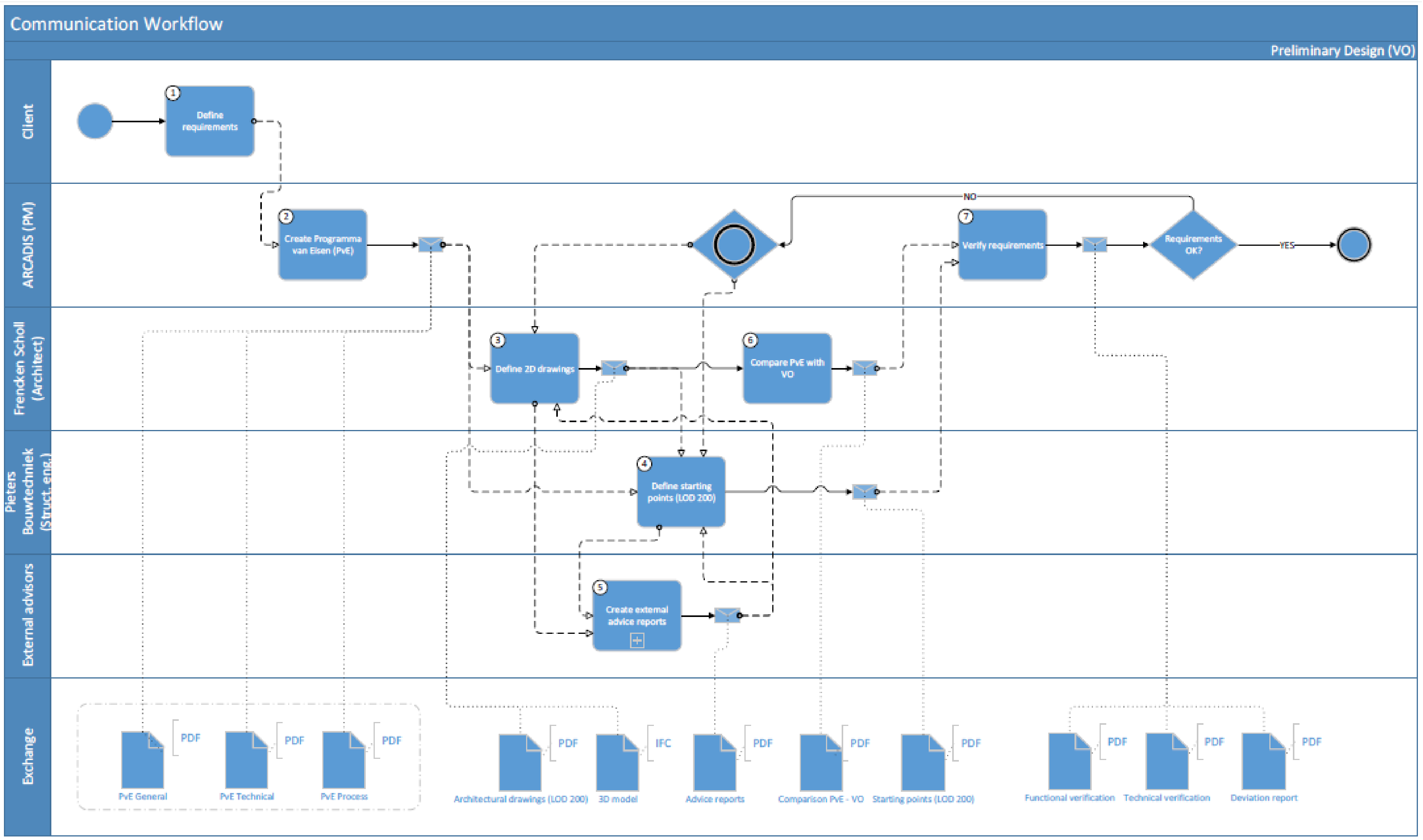


Figure 66. Communication workflow – MAVO project (Preliminary Design)

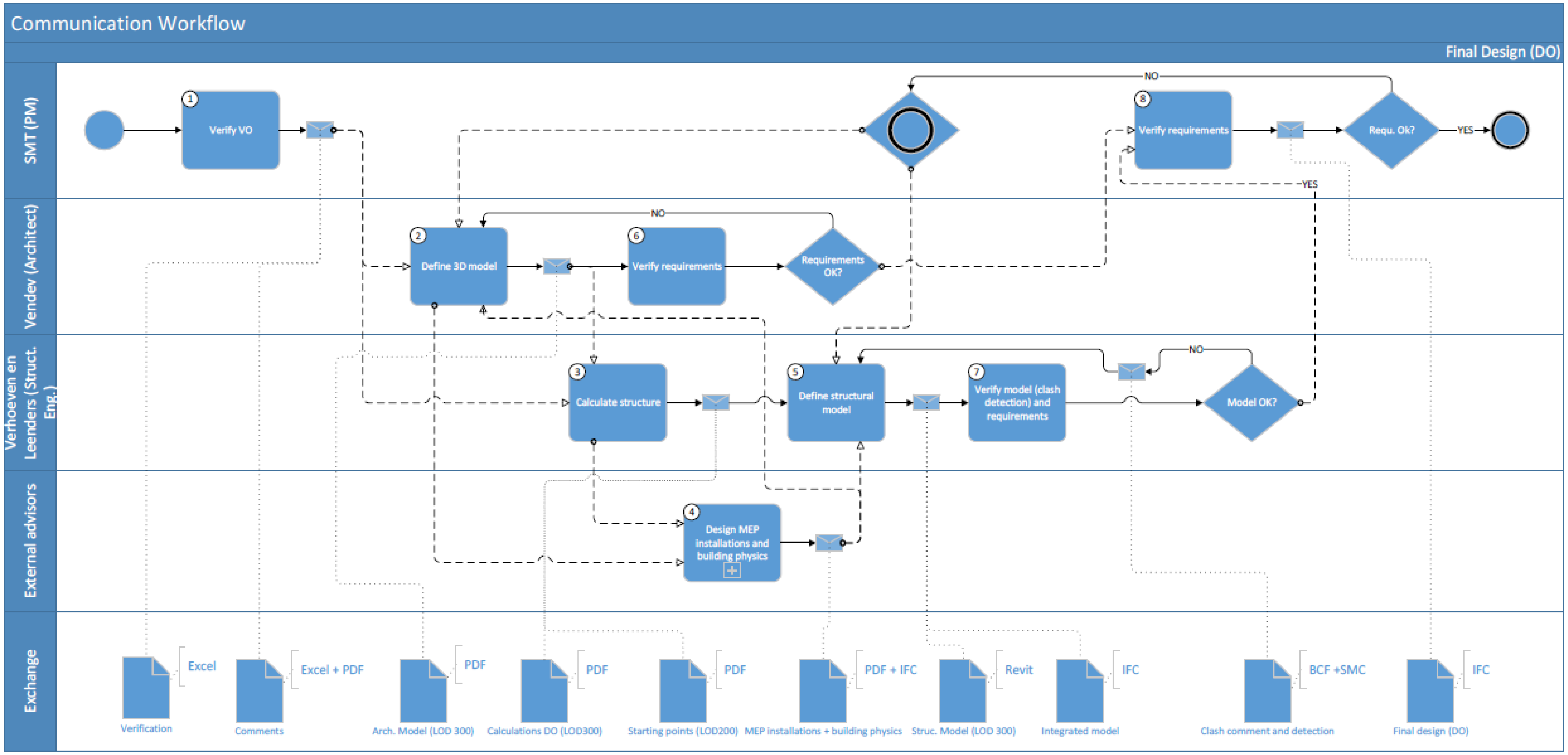


Figure 67. Communication workflow – MAVO project (Final Design)

7.4 Appendix IV: Process map MAVO project (Input/output workflow)

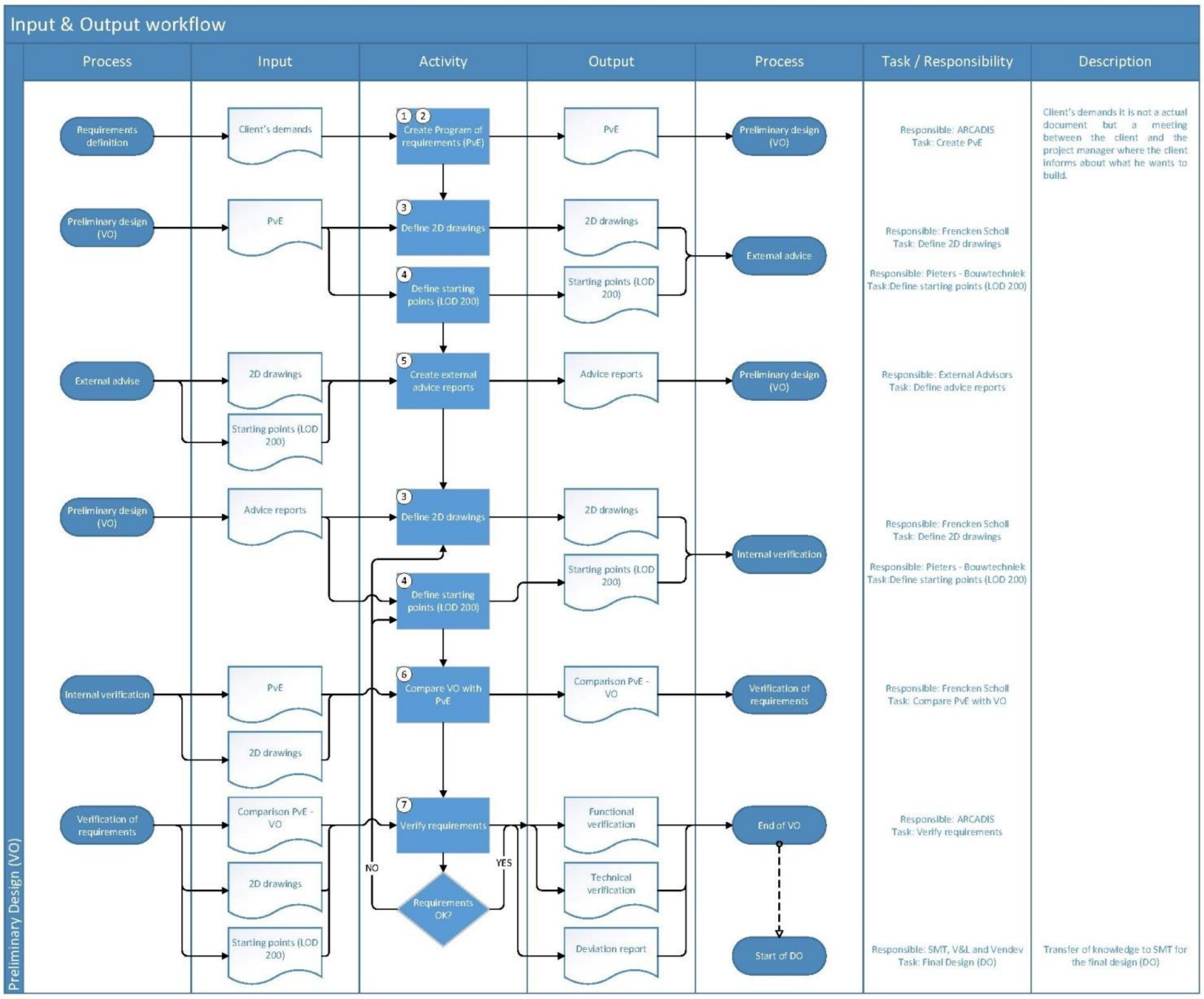

Input & Output workflow
Process
Input
Activity
Output
Process
Task / Responsibility
Description
Preliminary Design (VO)
Requirements definition
Client's demands
1 2 Create Program of requirements (PvE)
PvE
Preliminary design (VO)
Responsible: ARCADIS
Task: Create PvE
Client's demands it is not a actual document but a meeting between the client and the project manager where the client informs about what he wants to build.
Preliminary design (VO)
PvE
3 Define 2D drawings
2D drawings
4 Define starting points (LOD 200)
Starting points (LOD 200)
External advice
Responsible: Frencken Scholl
Task: Define 2D drawings
Responsible: Pieters - Bouwtechniek
Task:Define starting points (LOD 200)
External advise
2D drawings
Starting points (LOD 200)
5 Create external advice reports
Advice reports
Preliminary design (VO)
Responsible: External Advisors
Task: Define advice reports
Preliminary design (VO)
Advice reports
3 Define 2D drawings
2D drawings
4 Define starting points (LOD 200)
Starting points (LOD 200)
Internal verification
Responsible: Frencken Scholl
Task: Define 2D drawings
Responsible: Pieters - Bouwtechniek
Task:Define starting points (LOD 200)
Internal verification
PvE
2D drawings
6 Compare VO with PvE
Comparison PvE - VO
Verification of requirements
Responsible: Frencken Scholl
Task: Compare PvE with VO
Verification of requirements
Comparison PvE - VO
2D drawings
Starting points (LOD 200)
7 Verify requirements
NO
YES
Requirements OK?
Functional verification
Technical verification
Deviation report
End of VO
Start of DO
Responsible: ARCADIS
Task: Verify requirements
Responsible: SMT, V&L and Vendev
Task: Final Design (DO)
Transfer of knowledge to SMT for the final design (DO)

Figure 68. Input/output workflow – MAVO project (Preliminary Design)

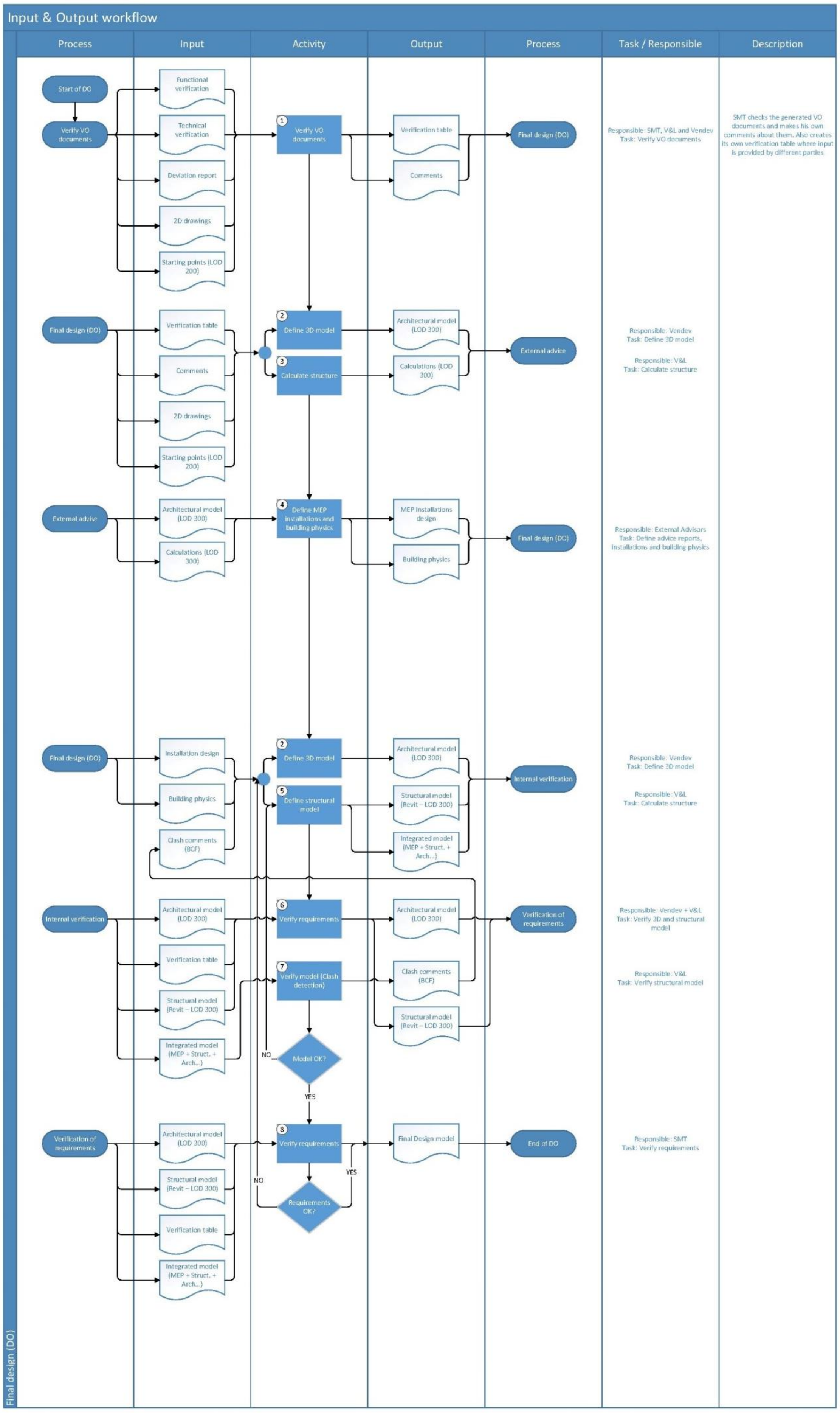

Input & Output workflow
Process
Input
Activity
Output
Process
Task / Responsible
Description
Final design (DO)
Start of DO
Verify VO documents
Functional verification
Technical verification
Deviation report
2D drawings
Starting points (LOD 200)
1 Verify VO documents
Verification table
Comments
Final design (DO)
Responsible: SMT, V&L and Vendev
Task: Verify VO documents
SMT checks the generated VO documents and makes his own comments about them. Also creates its own verification table where input is provided by different parties
Final design (DO)
Verification table
Comments
2D drawings
Starting points (LOD 200)
2 Define 3D model
3 Calculate structure
Architectural model (LOD 300)
Calculations (LOD 300)
External advice
Responsible: Vendev
Task: Define 3D model
Responsible: V&L
Task: Calculate structure
External advise
Architectural model (LOD 300)
Calculations (LOD 300)
4 Define MEP installations and building physics
MEP installations design
Building physics
Final design (DO)
Responsible: External Advisors
Task: Define advice reports, installations and building physics
Final design (DO)
Installation design
Building physics
Clash comments (BCF)
2 Define 3D model
5 Define structural model
Architectural model (LOD 300)
Structural model (Revit – LOD 300)
Integrated model (MEP + Struct. + Arch...)
Internal verification
Responsible: Vendev
Task: Define 3D model
Responsible: V&L
Task: Calculate structure
Internal verification
Architectural model (LOD 300)
Verification table
Structural model (Revit – LOD 300)
Integrated model (MEP + Struct. + Arch...)
6 Verify requirements
7 Verify model (Clash detection)
Model OK?
NO
YES
Architectural model (LOD 300)
Clash comments (BCF)
Structural model (Revit – LOD 300)
Verification of requirements
Responsible: Vendev + V&L
Task: Verify 3D and structural model
Responsible: V&L
Task: Verify structural model
Verification of requirements
Architectural model (LOD 300)
Structural model (Revit – LOD 300)
Verification table
Integrated model (MEP + Struct. + Arch...)
8 Verify requirements
Requirements OK?
NO
YES
Final Design model
End of DO
Responsible: SMT
Task: Verify requirements

Figure 69. Input/output workflow – MAVO project (Final design)

7.5 Appendix V: Application code (Python)

```
import os
import datetime
import uuid
import shutil
import glob
import zipfile
import xml.etree.cElementTree as ET
import ifcopenshell.guid
import ifcopenshell, ifcopenshell.geom
settings = ifcopenshell.geom.settings()
settings.set(settings.USE_PYTHON_OPENCASCADE, True)

from PyQt4 import QtCore, QtGui
from OCC.Display.backend import get_backend
get_backend("qt-pyqt4")
import OCC.Display.qtDisplay
from OCC.Display.qtDisplay import qtViewer3d

from OCC.gp import *
import OCC.Bnd, OCC.BRepBndLib
from OCC.Aspect import Aspect_GT_Rectangular, Aspect_GDM_Lines
from OCC.BRepPrimAPI import BRepPrimAPI_MakeBox

from rdflib import ConjunctiveGraph, Graph, Literal, URIRef,
Namespace, XSD, RDF
import urllib
import httplib2

from Tkinter import *
from Tkinter import W, E
import tkMessageBox
from tkFileDialog import askopenfilename

import Tkinter as tk

from ftplib import FTP

from SPARQLWrapper import SPARQLWrapper, RDFXML

guid_selection = None

class ProductViewer(qtViewer3d):
    def __init__(self, *args):
        qtViewer3d.__init__(self)
        self.objects = {}

    @staticmethod
    def Hash(shape):
        return shape.HashCode(1 << 30)

    displayed_shapes = {}

    def Show(self, key, shape, color=None):
        self.objects[ProductViewer.Hash(shape)] = key
        qclr = OCC.Quantity.Quantity_Color(.35, .25, .1,
OCC.Quantity.Quantity_TOC_RGB)
        ais = self._display.DisplayColoredShape(shape, qclr)
        self.displayed_shapes[key] = ais
        self._display.FitAll()

    def ColorasActive(self, key):
        ais = self.displayed_shapes[key]
        qclr = OCC.Quantity.Quantity_Color(1, 0, 0,
OCC.Quantity.Quantity_TOC_RGB)
        ais.GetObject().SetColor(qclr)

    def ColorasNoCheck(self,key):
        ais = self.displayed_shapes[key]
        qclr = OCC.Quantity.Quantity_Color(1, 1, 0,
OCC.Quantity.Quantity_TOC_RGB)
        ais.GetObject().SetColor(qclr)

    def ColorasResolved(self, key):
        ais = self.displayed_shapes[key]
        qclr = OCC.Quantity.Quantity_Color(1, 0.5, 0,
OCC.Quantity.Quantity_TOC_RGB)
        ais.GetObject().SetColor(qclr)

    def ColorasClosed(self, key):
        ais = self.displayed_shapes[key]
        qclr = OCC.Quantity.Quantity_Color(0, 0.7, 0,
OCC.Quantity.Quantity_TOC_RGB)
        ais.GetObject().SetColor(qclr)

    def Show2(self,key):
        ais = self.displayed_shapes[key]
        self._display.Context.Display(ais)

    def Hide(self, key):
        ais = self.displayed_shapes[key]
        self._display.Context.Erase(ais)

    def mouseReleaseEvent(self, *args):
        # Process selection by parent class
        qtViewer3d.mouseReleaseEvent(self, *args)
        if self._display.selected_shape:
            global guid_selection
            global selected_shape
            selected_shape = self._display.selected_shapes
            guid_selection =
(self.objects[ProductViewer.Hash(self._display.selected_shape)])
            # guid_selection = [self.objects[ProductViewer.Hash(x)]
for x in self._display.selected_shapes]

            print guid_selection

# Main class of the application
class initUI(object):
    def __init__(self, *args):
        # Constructing an application
        app = QtGui.QApplication(sys.argv)

        # Viewer initialization
        self.main = Main(self)
        self.main.show()
        self.main.canvas.InitDriver()
        self.main.statusBar()
        self.display = self.main.canvas._display
        #  Methods to feed the viewer with content
        self.geometry_box()
        self.geometry_grid()

        # Raise a system exit
        sys.exit(app.exec_())

    def geometry_box(self):
        box = BRepPrimAPI_MakeBox(10., 10., 10.).Shape()
        self.display.DisplayShape(box)
        self.display.FitAll()

    def geometry_grid(self):
        ax3 = gp_Ax3(gp_Pnt(0, 0, 0), gp_Dir(0, 0, 1))
        self.display.GetViewer().GetObject().SetPrivilegedPlane(ax3)

self.display.GetViewer().GetObject().SetRectangularGridValues(0, 0,
10, 10, 0)

self.display.GetViewer().GetObject().SetRectangularGridGraphicValues(1
0, 10, 0)

self.display.GetViewer().GetObject().ActivateGrid(Aspect_GT_Rectangula
r, Aspect_GDM_Lines)
        self.display.FitAll()

class MyApp(tk.Tk):
    def open_file(self):
        global file_path
        filename = askopenfilename()
        file_path = os.path._getfullpathname(filename)
        return file_path

    def __init__(self, ifcfilename, ifcfilelocation, dateandtime):
        tk.Tk.__init__(self)

        # Header
        self.label1 = tk.Label(self, text="Header", font="Verdana 14
bold")
        self.label1.grid(row=0, column=0, sticky=E + W)

        # File Elements
        self.label2 = tk.Label(self, text="File Elements",
font="Verdana 10 italic")
        self.label2.grid(row=1, column=0, sticky=E + W)

        # FileName
        self.label3 = tk.Label(self, text="FileName: ")
        self.label3.grid(row=2, column=0, sticky=W)

        self.filename = tk.StringVar()
        self.filename.set(ifcfilename)

        self.entry1 = tk.Entry(self, textvariable=self.filename)
        self.entry1.grid(row=2, column=1)

        # FileDate
        self.label4 = tk.Label(self, text="FileDate: ")
        self.label4.grid(row=3, column=0, sticky=W)
        self.filedate = tk.StringVar()
        self.filedate.set(dateandtime)

        self.entry2 = tk.Entry(self, textvariable=self.filedate)
        self.entry2.grid(row=3, column=1)

        # FileReference
        self.label5 = tk.Label(self, text="FileReference: ")
        self.label5.grid(row=4, column=0, sticky=W)

        self.filereference = tk.StringVar()
        self.filereference.set(ifcfilelocation)

        self.entry3 = tk.Entry(self, textvariable=self.filereference)
        self.entry3.grid(row=4, column=1)

        # Blank line
        self.label6 = tk.Label(self, text="")
        self.label6.grid(row=5, column=0, sticky=E + W)

        # Topic
        self.label7 = tk.Label(self, text="Topic", font="Verdana 14
bold")
        self.label7.grid(row=6, column=0, sticky=E + W)

        # Topic Attributes
        self.label8 = tk.Label(self, text="Topic Attributes",
font="Verdana 10 italic")
        self.label8.grid(row=7, column=0, sticky=E + W)

        # TopicType
        self.label9 = tk.Label(self, text="TopicType: ")
        self.label9.grid(row=8, column=0, sticky=W)

        self.entry4 = tk.Spinbox(self, value=("Issue", "Fault",
"Clash", "Request", "Inquiry", "Remark", "Undefined", "Error"))
        self.entry4.grid(row=8, column=1)

        # TopicStatus
        self.label10 = tk.Label(self, text="TopicStatus: ")
        self.label10.grid(row=9, column=0, sticky=W)

        self.entry5 = tk.Spinbox(self, value=("Active", "Resolved",
"Closed"))
        self.entry5.grid(row=9, column=1)

        # Topic Elements
        self.label11 = tk.Label(self, text="Topic Elements",
font="Verdana 10 italic")
        self.label11.grid(row=10, column=0, sticky=E + W)

        # Reference Link
        self.label12 = tk.Label(self, text="Reference Link: ")
        self.label12.grid(row=11, column=0, sticky=W)

        self.entry6 = tk.Entry(self)
        self.entry6.grid(row=11, column=1)

        # Title
        self.label13 = tk.Label(self, text="Title: ")
        self.label13.grid(row=12, column=0, sticky=W)

        self.entry7 = tk.Entry(self)
        self.entry7.grid(row=12, column=1)

        # Priority
        self.label14 = tk.Label(self, text="Priority: ")
        self.label14.grid(row=13, column=0, sticky=W)

        self.entry8 = tk.Spinbox(self, value=("Critical", "Major",
"Normal", "Minor", "On hold", "Undefined"))
        self.entry8.grid(row=13, column=1)

        # Topic Index
        self.label15 = tk.Label(self, text="Topic Index: ")
        self.label15.grid(row=14, column=0, sticky=W)

        self.entry9 = tk.Entry(self)
        self.entry9.grid(row=14, column=1)

        # Labels
        self.label16 = tk.Label(self, text="Labels: ")
        self.label16.grid(row=15, column=0, sticky=W)

        self.entry10 = tk.Spinbox(self, value=("Architecture",
"Structure", "Mechanical", "Electrical", "Specifications",
"Technology", "Undefined"))
        self.entry10.grid(row=15, column=1)

        # Creation Date
        self.label17 = tk.Label(self, text="Creation Date: ")
        self.label17.grid(row=16, column=0, sticky=W)

        self.entry11 = tk.Entry(self, textvariable=self.filedate)
        self.entry11.grid(row=16, column=1)

        # Creation Author
        self.label18 = tk.Label(self, text="Creation Author: ")
        self.label18.grid(row=17, column=0, sticky=W)

        self.entry12 = tk.Entry(self)
        self.entry12.grid(row=17, column=1)

        # Topic Modified Date
        self.label19 = tk.Label(self, text="Topic modified date: ")
        self.label19.grid(row=18, column=0, sticky=W)

        self.entry13 = tk.Entry(self)
        self.entry13.grid(row=18, column=1)

        # Topic Modified Author
        self.label20 = tk.Label(self, text="Topic modified author: ")
        self.label20.grid(row=19, column=0, sticky=W)

        self.entry14 = tk.Entry(self)
        self.entry14.grid(row=19, column=1)

        # DueDate
        self.label21 = tk.Label(self, text="DueDate: ")
        self.label21.grid(row=20, column=0, sticky=W)

        self.entry15 = tk.Entry(self)
        self.entry15.grid(row=20, column=1)

        # Assigned to
        self.label22 = tk.Label(self, text="Assigned to: ")
        self.label22.grid(row=21, column=0, sticky=W)

        self.entry16 = tk.Entry(self)
        self.entry16.grid(row=21, column=1)

        # Stage
        self.label23 = tk.Label(self, text="Stage: ")
        self.label23.grid(row=22, column=0, sticky=W)

        self.entry17 = tk.Entry(self)
        self.entry17.grid(row=22, column=1)

        # Topic Description
        self.label24 = tk.Label(self, text="Topic Description: ")
        self.label24.grid(row=23, column=0, sticky=W)

        self.entry18 = tk.Entry(self)
        self.entry18.grid(row=23, column=1)

        # Bim Snippet
        self.label25 = tk.Label(self, text="BimSnippet elements",
font="Verdana 10 italic")
        self.label25.grid(row=24, column=0, sticky=E + W)

        # Bim Snippet Reference
        self.label26 = tk.Label(self, text="BimSnippet Reference: ")
        self.label26.grid(row=25, column=0, sticky=W)

        self.entry19 = tk.Entry(self)
        self.entry19.grid(row=25, column=1)

        # Reference Schema
        self.label27 = tk.Label(self, text="Reference schema: ")
        self.label27.grid(row=26, column=0, sticky=W)

        self.entry20 = tk.Entry(self)
        self.entry20.grid(row=26, column=1)

        # Document reference
        self.label28 = tk.Label(self, text="Document Reference
elements", font="Verdana 10 italic")
        self.label28.grid(row=27, column=0, sticky=E + W)

        # Referenced document
        self.label29 = tk.Label(self, text="Referenced document: ")
        self.label29.grid(row=28, column=0, sticky=W)

        self.entry21 = tk.Entry(self)
        self.entry21.grid(row=28, column=1)

        # Document description
        self.label30 = tk.Label(self, text="Document description: ")
        self.label30.grid(row=29, column=0, sticky=W)
        self.entry22 = tk.Entry(self)
        self.entry22.grid(row=29, column=1)

        # Comment
        self.label31 = tk.Label(self, text="Comment", font="Verdana 14
bold")
        self.label31.grid(row=0, column=5, sticky=E + W)

        # Comment elements
        self.label32 = tk.Label(self, text="Comment Elements",
font="Verdana 10 italic")
        self.label32.grid(row=1, column=5, sticky=E + W)

        # Comment date
        self.label33 = tk.Label(self, text="Comment date: ")
        self.label33.grid(row=2, column=5, sticky=W)

        self.entry23 = tk.Entry(self, textvariable=self.filedate)
        self.entry23.grid(row=2, column=6)

        # Author
        self.label34 = tk.Label(self, text="Author: ")
        self.label34.grid(row=3, column=5, sticky=W)

        self.entry24 = tk.Entry(self)
        self.entry24.grid(row=3, column=6)

        # Comment of comment
        self.label35 = tk.Label(self, text="Comment: ")
        self.label35.grid(row=4, column=5, sticky=W)

        self.entry25 = tk.Entry(self)
        self.entry25.grid(row=4, column=6)

        # Comment modified date
        self.label36 = tk.Label(self, text="Modified date: ")
        self.label36.grid(row=5, column=5, sticky=W)

        self.entry26 = tk.Entry(self)
        self.entry26.grid(row=5, column=6)

        # Comment modified author
        self.label37 = tk.Label(self, text="Modified author: ")
        self.label37.grid(row=6, column=5, sticky=W)

        self.entry27 = tk.Entry(self)
        self.entry27.grid(row=6, column=6)

        # blank line
        self.label38 = tk.Label(self, text="")
        self.label38.grid(row=7, column=5, sticky=E + W)

        # Viewpoints
        self.label40 = tk.Label(self, text="Viewpoints", font="Verdana
14 bold")
        self.label40.grid(row=8, column=5, sticky=E + W)

        # Viewpoints elements
        self.label41 = tk.Label(self, text="Viewpoints elements",
font="Verdana 10 italic")
        self.label41.grid(row=9, column=5, sticky=E + W)

        # Viewpoint
        self.label42 = tk.Label(self, text="Viewpoint: ")
        self.label42.grid(row=10, column=5, sticky=W)

        self.entry28 = tk.Entry(self)
        self.entry28.grid(row=10, column=6)

        # Snapshot
        self.label43 = tk.Label(self, text="Snapshot: ")
        self.label43.grid(row=11, column=5, sticky=W)

        self.entry29 = tk.Entry(self)
        self.entry29.grid(row=11, column=6)

        # Viewpoint index
        self.label44 = tk.Label(self, text="Viewpoint index: ")
        self.label44.grid(row=12, column=5, sticky=W)

        self.entry30 = tk.Entry(self)
        self.entry30.grid(row=12, column=6)

        # blank line
        self.label39 = tk.Label(self, text="")
        self.label39.grid(row=13, column=5, sticky=E + W)

        # Close button
        close_button = tk.Button(self, text="Accept and close",
command=self.close)
        close_button.grid(row=14, column=6, sticky=E)

        # Browse button
        browse_button = tk.Button(self, text="Browse",
command=self.select_file)
        browse_button.grid(row=28, column=2, sticky=E)

    def select_file(self):
        fp = self.open_file()
        self.entry21.delete(0, "end")
        self.entry21.insert(0, fp)

    def close(self):
        # First all entry values are added to a list
        global result
        result = []
        result.append(self.entry1.get())
        result.append(self.entry2.get())
        result.append(self.entry3.get())
        result.append(self.entry4.get())
        result.append(self.entry5.get())
        result.append(self.entry6.get())
        result.append(self.entry7.get())
        result.append(self.entry8.get())
        result.append(self.entry9.get())
        result.append(self.entry10.get())
        result.append(self.entry11.get())
        result.append(self.entry12.get())
        result.append(self.entry13.get())
        result.append(self.entry14.get())
        result.append(self.entry15.get())
        result.append(self.entry16.get())
        result.append(self.entry17.get())
        result.append(self.entry18.get())
        result.append(self.entry19.get())
        result.append(self.entry20.get())
        result.append(self.entry21.get())
        result.append(self.entry22.get())
        result.append(self.entry23.get())
        result.append(self.entry24.get())
        result.append(self.entry25.get())
        result.append(self.entry26.get())
        result.append(self.entry27.get())
        result.append(self.entry28.get())
        result.append(self.entry29.get())
        result.append(self.entry30.get())

        # show confirmation message
        tkMessageBox.showinfo("Message", "RDF file uploaded
successfully")

        # close UI
        self.destroy()

    def mainloop(self):
        tk.Tk.mainloop(self)
        return result

class MyApp2(tk.Tk):
    def __init__(self, dateandtime):
        tk.Tk.__init__(self)

        # configure grid columns
        self.columnconfigure(0, pad=3)
        self.columnconfigure(1, pad=3)
        # configure grid rows
        self.rowconfigure(0, pad=3)
        self.rowconfigure(1, pad=3)
        self.rowconfigure(2, pad=3)
        self.rowconfigure(3, pad=3)
        self.rowconfigure(4, pad=3)
        self.rowconfigure(5, pad=3)
        self.rowconfigure(6, pad=3)
        self.rowconfigure(7, pad=3)
        self.rowconfigure(8, pad=3)
        self.rowconfigure(9, pad=3)
        self.rowconfigure(10, pad=3)

        # Topic
        self.label7 = tk.Label(self, text="Topic", font="Verdana 14
bold")
        self.label7.grid(row=0, column=0, sticky=E + W)

        # Topic Attributes
        self.label8 = tk.Label(self, text="Topic Attributes",
font="Verdana 10 italic")
        self.label8.grid(row=1, column=0, sticky=E + W)

        # TopicStatus
        self.label10 = tk.Label(self, text="TopicStatus: ")
        self.label10.grid(row=2, column=0, sticky=W)

        self.entry5 = tk.Spinbox(self, value=("Active", "Resolved",
"Closed"))
        self.entry5.grid(row=2, column=1)

        # Blank line
        self.label6 = tk.Label(self, text="")
        self.label6.grid(row=3, column=0, sticky=E + W)

        # Comment
        self.label31 = tk.Label(self, text="Comment", font="Verdana 14
bold")
        self.label31.grid(row=4, column=0, sticky=E + W)

        # Comment elements
        self.label32 = tk.Label(self, text="Comment Elements",
font="Verdana 10 italic")
        self.label32.grid(row=5, column=0, sticky=E + W)

        # Comment date
        self.label33 = tk.Label(self, text="Comment date: ")
        self.label33.grid(row=6, column=0, sticky=W)

        self.filedate = tk.StringVar()
        self.filedate.set(dateandtime)

        self.entry23 = tk.Entry(self, textvariable=self.filedate)
        self.entry23.grid(row=6, column=1)

        # Author
        self.label34 = tk.Label(self, text="Author: ")
        self.label34.grid(row=7, column=0, sticky=W)

        self.entry24 = tk.Entry(self)
        self.entry24.grid(row=7, column=1)

        # Comment of comment
        self.label35 = tk.Label(self, text="Comment: ")
        self.label35.grid(row=8, column=0, sticky=W)

        self.entry25 = tk.Entry(self)
        self.entry25.grid(row=8, column=1)

        # blank line
        self.label38 = tk.Label(self, text="")
        self.label38.grid(row=9, column=0, sticky=E + W)

        # Close button
        close_button = tk.Button(self, text="Accept and close",
command=self.close)
        close_button.grid(row=10, column=1, sticky=E)

    def close(self):
        # First all entry values are added to a list
        global result
        result = []
        result.append(self.entry5.get())
        result.append(self.entry23.get())
        result.append(self.entry24.get())
        result.append(self.entry25.get())

        # show confirmation message
        tkMessageBox.showinfo("Message", "RDF file uploaded
successfully")

        # close UI
        self.destroy()

    def mainloop(self):
        tk.Tk.mainloop(self)
        return result

class MyApp3(tk.Tk):
    def __init__(self, dateandtime):
        tk.Tk.__init__(self)

        # configure grid columns
        self.columnconfigure(0, pad=3)
        self.columnconfigure(1, pad=3)
        # configure grid rows
        self.rowconfigure(0, pad=3)
        self.rowconfigure(1, pad=3)
        self.rowconfigure(2, pad=3)
        self.rowconfigure(3, pad=3)
        self.rowconfigure(4, pad=3)
        self.rowconfigure(5, pad=3)
        self.rowconfigure(6, pad=3)
        self.rowconfigure(7, pad=3)
        self.rowconfigure(8, pad=3)
        self.rowconfigure(9, pad=3)
        self.rowconfigure(10, pad=3)
        self.rowconfigure(11, pad=3)
        self.rowconfigure(12, pad=3)

        # Topic
        self.label7 = tk.Label(self, text="Topic", font="Verdana 14
bold")
        self.label7.grid(row=0, column=0, sticky=E + W)

        # Topic Attributes
        self.label8 = tk.Label(self, text="Topic Attributes",
font="Verdana 10 italic")
        self.label8.grid(row=1, column=0, sticky=E + W)

        # TopicStatus
        self.label10 = tk.Label(self, text="TopicStatus: ")
        self.label10.grid(row=2, column=0, sticky=W)

        self.entry5 = tk.Spinbox(self, value=("Closed"))
        self.entry5.grid(row=2, column=1)

        # Topic Elements
        self.label11 = tk.Label(self, text="Topic Elements",
font="Verdana 10 italic")
        self.label11.grid(row=3, column=0, sticky=E + W)

        # Creation Date
        self.label17 = tk.Label(self, text="Creation Date: ")
        self.label17.grid(row=4, column=0, sticky=W)

        self.filedate = tk.StringVar()
        self.filedate.set(dateandtime)

        self.entry11 = tk.Entry(self, textvariable=self.filedate)
        self.entry11.grid(row=4, column=1)

        # Creation Author
        self.label18 = tk.Label(self, text="Creation Author: ")
        self.label18.grid(row=5, column=0, sticky=W)

        self.entry12 = tk.Entry(self)
        self.entry12.grid(row=5, column=1)

        # Stage
        self.label23 = tk.Label(self, text="Stage: ")
        self.label23.grid(row=6, column=0, sticky=W)

        self.entry17 = tk.Entry(self)
        self.entry17.grid(row=6, column=1)

        # blank line
        self.label38 = tk.Label(self, text="")
        self.label38.grid(row=7, column=0, sticky=E + W)

        # Comment
        self.label31 = tk.Label(self, text="Comment", font="Verdana 14
bold")
        self.label31.grid(row=8, column=0, sticky=E + W)

        # Comment elements
        self.label32 = tk.Label(self, text="Comment Elements",
font="Verdana 10 italic")
        self.label32.grid(row=9, column=0, sticky=E + W)

        # Comment of comment
        self.label35 = tk.Label(self, text="Comment: ")
        self.label35.grid(row=10, column=0, sticky=W)

        self.entry25 = tk.Entry(self)
        self.entry25.grid(row=10, column=1)

        # blank line
        self.label39 = tk.Label(self, text="")
        self.label39.grid(row=11, column=0, sticky=E + W)

        # Close button
        close_button = tk.Button(self, text="Accept and close",
command=self.close)
        close_button.grid(row=12, column=1, sticky=E)

    def close(self):
        # First all entry values are added to a list
        global result
        result = []
        result.append(self.entry5.get())
        result.append(self.entry11.get())
        result.append(self.entry12.get())
        result.append(self.entry17.get())
        result.append(self.entry25.get())

        # show confirmation message
        tkMessageBox.showinfo("Message", "RDF file uploaded
successfully")

        # close UI
        self.destroy()

    def mainloop(self):
        tk.Tk.mainloop(self)
        return result

class MyApp4(tk.Tk):
    def __init__(self):
        tk.Tk.__init__(self)

        #configure grid columns
        self.columnconfigure(0, pad=3)
        self.columnconfigure(1, pad=3)
        #configure grid rows
        self.rowconfigure(0, pad=3)
        self.rowconfigure(1, pad=3)
        self.rowconfigure(2, pad=3)
        self.rowconfigure(3, pad=3)
        self.rowconfigure(4, pad=3)

        self.label1 = tk.Label(self, text="Assigned To", font="Verdana
14 bold")
        self.label1.grid(row=0, column=0)

        self.label2 = tk.Label(self, text="Assign a person",
font="Verdana 10 italic")
        self.label2.grid(row=1, column=0)

        self.label3 = tk.Label(self, text="Assigned to: ")
        self.label3.grid(row=2, column=0)

        self.entry = tk.Entry(self)
        self.entry.grid(row=2,column=1)

        self.label4 = tk.Label(self, text="")
        self.label4.grid(row=3, column=0)

        close_button = tk.Button(self, text="Accept and close",
command=self.close)
        close_button.grid(row=4,column=1)

        self.string = ""

    def close(self):
        global result
        self.string = self.entry.get()
        self.destroy()

    def mainloop(self):
        tk.Tk.mainloop(self)
        return self.string

# Main class of the Graphical User Interface
```

```
class Main(QtGui.QMainWindow):
    def __init__(self, parent=None):
        self.parent = parent
        QtGui.QMainWindow.__init__(self)

        # Instantiating the tabs
        global filename
        global bcffilename
        self.filename = None
        self.bcffilename = None

        self.tabs = QtGui.QTabWidget()
        self.setCentralWidget(self.tabs)

        self.viewer_tab = QtGui.QWidget()
        self.tabs.addTab(self.viewer_tab, "3D Viewer")

        self.compatibility_tab = QtGui.QWidget()
        self.tabs.addTab(self.compatibility_tab, "Backwards and
external compatibility")

        # Implementing the OCC viewer
        self.canvas = ProductViewer(self)
        self.setGeometry(375, 25, 800, 550)
        self.setWindowTitle("Application BCF & RDF")

        # Calling both the tabs
        self.tab_3dview()
        self.tab_compatibility()

    def tab_3dview(self):
        # Initializing a split-view layout
        self.propertybox = QtGui.QTextBrowser()
        font = QtGui.QFont("Arial", 10, QtGui.QFont.Bold, True)
        sizePolicy = QtGui.QSizePolicy(QtGui.QSizePolicy.Fixed,
QtGui.QSizePolicy.MinimumExpanding)
        self.propertybox.setFont(font)
        self.propertybox.setSizePolicy(sizePolicy)

        # self.componentbox = QtGui.QListWidget()
        # self.componentbox.setFont(font)
        # self.componentbox.setSizePolicy(sizePolicy)

        self.propertybox2 = QtGui.QTextBrowser()
        self.propertybox2.setFont(font)
        self.propertybox2.setSizePolicy(sizePolicy)

        # item = QtGui.QListWidgetItem()
        # self.componentbox.addItem(item)

        # Define a widget for the 3D viewer
        center = QtGui.QWidget()

        # Define and set layout
        mainLayout = QtGui.QHBoxLayout(center)
        viewer_hbox = QtGui.QHBoxLayout()
        viewer_vbox = QtGui.QVBoxLayout()

        #Buttons
        createbcf_btn = QtGui.QPushButton("Create new BCF / RDF",
self)
        createbcf_btn.clicked.connect(self.create_BCF_RDF)

        viewer_open_ifc_btn = QtGui.QPushButton("Open IFC file", self)
        viewer_open_ifc_btn.clicked.connect(self.open_ifc_file)

        query_show_comments_btn = QtGui.QPushButton("Check object
status", self)

query_show_comments_btn.clicked.connect(self.query_getComments)

        view_selected_object_only_btn = QtGui.QPushButton("Show
selected object only", self)

view_selected_object_only_btn.clicked.connect(self.view_selected_objec
t_only)

        hide_selected_object_btn = QtGui.QPushButton("Hide selected
object", self)

hide_selected_object_btn.clicked.connect(self.hide_selected_object)

        view_model_objects_btn = QtGui.QPushButton("Show all model
objects", self)
        view_model_objects_btn.clicked.connect(self.view_model)

        show_closed_objects_btn = QtGui.QPushButton("Show closed
objects", self)

show_closed_objects_btn.clicked.connect(self.show_closed_objects)

        show_notclosed_objects_btn = QtGui.QPushButton("Show open
objects", self)

show_notclosed_objects_btn.clicked.connect(self.show_notclosed_objects
)

        show_assigned_objects_btn = QtGui.QPushButton("Show assigned
objects", self)

show_assigned_objects_btn.clicked.connect(self.show_assigned_objects)

        mark_as_closed_btn = QtGui.QPushButton("Mark as closed", self)
        mark_as_closed_btn.clicked.connect(self.mark_as_closed)

        query_get_attached_files_btn = QtGui.QPushButton("Show files
attached", self)

query_get_attached_files_btn.clicked.connect(self.query_getAttachedFil
es)

        splitter = QtGui.QSplitter(QtCore.Qt.Horizontal)
        splitterH = QtGui.QSplitter(QtCore.Qt.Vertical)

        splitter.addWidget(self.canvas)
        splitter.addWidget(splitterH)
        splitterH.addWidget(view_selected_object_only_btn)
        splitterH.addWidget(hide_selected_object_btn)
        splitterH.addWidget(view_model_objects_btn)
        splitterH.addWidget(self.propertybox)
        # splitterH.addWidget(self.componentbox)
        splitterH.addWidget(query_get_attached_files_btn)
        splitterH.addWidget(self.propertybox2)

        viewer_vbox.addWidget(splitter)
        viewer_vbox.addLayout(viewer_hbox)
        self.viewer_tab.setLayout(viewer_vbox)

        viewer_hbox.addWidget(viewer_open_ifc_btn)
        viewer_hbox.addWidget(createbcf_btn)
        viewer_hbox.addWidget(query_show_comments_btn)
        viewer_hbox.addWidget(show_closed_objects_btn)
        viewer_hbox.addWidget(show_notclosed_objects_btn)
        viewer_hbox.addWidget(show_assigned_objects_btn)
        viewer_hbox.addWidget(mark_as_closed_btn)

    def open_ifc_file(self, filename=None):
        self.filename = QtGui.QFileDialog.getOpenFileName(self, 'Open
file', ".", "Industry Foundation Classes(*.ifc)")
        if self.filename:
            self.parent.display.EraseAll()
            self.propertybox.clear()
        self.parse_ifc(self.filename)

    def parse_ifc(self, filename):
        self.created_shapes = {}
        self.ifc_file = ifcopenshell.open(filename)
        rooms = self.ifc_file.by_type("IfcBuildingElement")
        for room in rooms:
            if room.Representation:
                ifcgeom = ifcopenshell.geom.create_shape(settings,
room).geometry
                shp = self.canvas.Show(room.GlobalId, ifcgeom, None)
            print "IFC file successfully loaded!"
        return self.ifc_file

    def view_selected_object_only(self):
        rooms = self.ifc_file.by_type("IfcBuildingElement")
        for room in rooms:
            if room.Representation:
                self.canvas.Hide(room.GlobalId)

        self.canvas.Show2(guid_selection)

    def hide_selected_object(self):
        self.canvas.Hide(guid_selection)

    def view_model(self):
        rooms = self.ifc_file.by_type("IfcBuildingElement")
        for room in rooms:
            if room.Representation:
                self.canvas.Show2(room.GlobalId)

    def create_BCF_RDF(self):
        dateandtime = datetime.datetime.now().strftime("%Y-%m-%d
%H:%M:%S")
        if not self.filename:
            QtGui.QMessageBox.warning(self,
                                          "No IFC loaded!",
                                    "Please load a model first!")
            return

        if guid_selection == None:
            QtGui.QMessageBox.warning(self, "Select element first!",
"Please select the element that you want to create the BCF for")

            return

        product = self.ifc_file.by_type("IfcProject")
        pguid = product[0].GlobalId

        ifcfilename = os.path.basename(str(self.filename))

        ifcfilelocation = self.filename

        # Guids
        y = uuid.uuid4()
        z = uuid.uuid4()

        # Uploading referenced document to FTP
        def ftp_filelocation():
            localfile = result[20]
            if localfile == "":
                print "No file attached"
            else:
                host = 'sparql.verhoeven-leenders.nl'
                # host = '192.168.5.2'
                username = 'Francisco'
                password = 'Fran123'
                remotefile = '/Referenced Documents'

                ftp = FTP()
                ftp.set_debuglevel(2)
                ftp.connect(host, 2121)
                ftp.login(username, password)
                ftp.cwd(remotefile)

                fp = open(localfile, 'rb')
                ftp.storbinary('STOR %s' %
os.path.basename(localfile), fp, 2121)
                fp.close()
                print "after upload " + localfile + " to " +
remotefile

                reffile = os.path.basename(str(file_path))
                ftpfilelocation = 'ftp://Francisco@sparql.verhoeven-
leenders.nl:2121/Referenced%20Documents/' + str(reffile)
                # ftpfilelocation =
'ftp://Francisco@192.168.5.2:2121/Referenced%20Documents/' +
str(reffile)
                ftp.quit()
                return ftpfilelocation

        def next_context_name():
            repository2 = 'V17001Fran'
            graph2 = 'file://C:/fakepath/RDFtest%d.rdf'
            params2 = {'context': '<' + graph2 + '>'}
            # endpoint2 = "http://192.168.5.2/repositories/%s/rdf-
graphs?%s" % (repository2, urllib.urlencode(params2))
```

```
            endpoint2 = "http://sparql.verhoeven-
leenders.nl/repositories/%s/rdf-graphs?%s" % (repository2,
urllib.urlencode(params2))
            (response, content) = httplib2.Http().request(endpoint2)
            return graph2 % content.count('\n')

        def next_file_name():
            result2 = []
            markup = 'http://example.org/data#Markup%d'
            header = 'http://example.org/data#Header%d'
            file = 'http://example.org/data#File%d'
            topic = 'http://example.org/data#Topic%d'
            bimsnippet = 'http://example.org/data#BimSnippet%d'
            documentreference =
'http://example.org/data#DocumentReference%d'
            relatedtopic = 'http://example.org/data#RelatedTopic%d'
            comment = 'http://example.org/data#Comment%d'
            viewpoint = 'http://example.org/data#Viewpoint%d'
            viewpoints = 'http://example.org/data#Viewpoints%d'

            repository2 = 'V17001Fran'
            graph2 = 'file://C:/fakepath/RDFtest%d.rdf'
            params2 = {'context': '<' + graph2 + '>'}
            # endpoint2 = "http://192.168.5.2/repositories/%s/rdf-
graphs?%s" % (repository2, urllib.urlencode(params2))
            endpoint2 = "http://sparql.verhoeven-
leenders.nl/repositories/%s/rdf-graphs?%s" % (repository2,
urllib.urlencode(params2))
            (response, content) = httplib2.Http().request(endpoint2)

            r1 = markup % content.count('\n')
            r2 = header % content.count('\n')
            r3 = file % content.count('\n')
            r4 = topic % content.count('\n')
            r5 = bimsnippet % content.count('\n')
            r6 = documentreference % content.count('\n')
            r7 = relatedtopic % content.count('\n')
            r8 = comment % content.count('\n')
            r9 = viewpoint % content.count('\n')
            r10 = viewpoints % content.count('\n')

            result2.append(r1)
            result2.append(r2)
            result2.append(r3)
            result2.append(r4)
            result2.append(r5)
            result2.append(r6)
            result2.append(r7)
            result2.append(r8)
            result2.append(r9)
            result2.append(r10)

            return result2

        def create_RDF_graph():
            g = Graph()

            # Namespaces
            fbf = Namespace("http://example.org/bcfschema#")
            # fbfd = Namespace("http://example.org/data#")
            g.namespace_manager.bind('fbf', fbf)

            # Classes
            markup = URIRef(next_file_name()[0])
            header = URIRef(next_file_name()[1])
            file = URIRef(next_file_name()[2])
            topic = URIRef(next_file_name()[3])
            bimsnippet = URIRef(next_file_name()[4])
            documentreference = URIRef(next_file_name()[5])
            relatedtopic = URIRef(next_file_name()[6])
            comment = URIRef(next_file_name()[7])
            viewpoint = URIRef(next_file_name()[8])
            viewpoints = URIRef(next_file_name()[9])

            # Properties
            # Hierarchy properties
            containsheader = URIRef(fbf.containsHeader)
            containsfile = URIRef(fbf.containsFile)
            containstopic = URIRef(fbf.containsTopic)
            containscomment = URIRef(fbf.containsComment)
            containsviewpoints = URIRef(fbf.containsViewpoints)
            containsbimsnippet = URIRef(fbf.containsBimSnippet)
            containsdocumentreference =
URIRef(fbf.containsDocumentReference)
            containsrelatedtopic = URIRef(fbf.containsRelatedTopic)
            containsviewpoint = URIRef(fbf.containsViewpoint)

            # File properties
            hasfilename = URIRef(fbf.hasFileName)
            hasfiledate = URIRef(fbf.hasFileDate)
            hasfilereference = URIRef(fbf.hasFileReference)
            hasifcproject = URIRef(fbf.hasIfcProject)
            hasifcspatialstructureelement =
URIRef(fbf.hasIfcSpatialStructureElement)
            hasfileisexternal = URIRef(fbf.hasFileIsExternal)

            # TopicProperties
            hastopicguid = URIRef(fbf.hasTopicGuid)
            hastopictype = URIRef(fbf.hasTopicType)
            hastopicstatus = URIRef(fbf.hasTopicStatus)
            hasreferencelink = URIRef(fbf.hasReferenceLink)
            hastitle = URIRef(fbf.hasTitle)
            haspriority = URIRef(fbf.hasPriority)
            hastopicindex = URIRef(fbf.hasTopicIndex)
            haslabels = URIRef(fbf.hasLabels)
            hascreationdate = URIRef(fbf.hasCreationDate)
            hascreationauthor = URIRef(fbf.hasCreationAuthor)
            hasmodifieddate = URIRef(fbf.hasModifiedDate)
            hasmodifiedauthor = URIRef(fbf.hasModifiedAuthor)
            hasduedate = URIRef(fbf.hasDueDate)
            hasassignedto = URIRef(fbf.hasAssignedTo)
            hasstage = URIRef(fbf.hasStage)
            hastopicdescription = URIRef(fbf.hasTopicDescription)

            # BimSnippet properties
            hasbimsnippetreference =
URIRef(fbf.hasBimSnippetReference)
            hasreferenceschema = URIRef(fbf.hasReferenceSchema)
            hasbimsnippetisexternal =
URIRef(fbf.hasBimSnippetIsExternal)
```

```
            hassnippettype = URIRef(fbf.hasSnippetType)

            # DocumentReference properties
            hasreferenceddocument = URIRef(fbf.hasReferencedDocument)
            hasdocumentdescription =
URIRef(fbf.hasDocumentReferenceDescription)
            hasdocumentguid = URIRef(fbf.hasDocumentReferenceGuid)
            hasdocumentisexternal =
URIRef(fbf.hasDocumentReferenceIsExternal)

            # RelatedTopic properties
            hasrelatedtopicguid = URIRef(fbf.hasRelatedTopicGuid)

            # Comment properties
            hascommentguid = URIRef(fbf.hasCommentGuid)
            hascommentdate = URIRef(fbf.hasCommentDate)
            hasauthor = URIRef(fbf.hasAuthor)
            hascomment = URIRef(fbf.hasComment)
            hascommentmoddate = URIRef(fbf.hasCommentModifiedDate)
            hascommentmodauthor = URIRef(fbf.hasCommentModifiedAuthor)

            # Viewpoint properties
            hascommentviewpoinguid = URIRef(fbf.hasViewpointGuid)

            # Viewpoints properties
            hasviewpointsguid = URIRef(fbf.hasViewpointsGuid)
            hasviewpoint = URIRef(fbf.hasViewpoint)
            hassnapshot = URIRef(fbf.hasSnapshot)
            hasviewpointsindex = URIRef(fbf.hasViewpointsIndex)

            # Triples
            # Markuptriples
            g.add((markup, RDF.type, fbf.Markup))
            g.add((markup, containsheader, header))
            g.add((markup, containstopic, topic))
            g.add((markup, containscomment, comment))
            g.add((markup, containsviewpoints, viewpoints))

            # Headertriples
            g.add((header, RDF.type, fbf.Header))
            g.add((header, containsfile, file))

            # Filetriples
            g.add((file, RDF.type, fbf.File))
            g.add((file, hasfilename, Literal(result[0],
datatype=XSD.string)))
            g.add((file, hasfiledate, Literal(result[1],
datatype=XSD.dateTime)))
            g.add((file, hasfilereference, Literal(result[2],
datatype=XSD.string)))
            g.add((file, hasifcproject, Literal(pguid,
datatype=XSD.string)))
            g.add((file, hasifcspatialstructureelement,
Literal('IfcGuid2', datatype=XSD.string)))
            g.add((file, hasfileisexternal, Literal('True',
datatype=XSD.boolean)))

            # Topictriples
            g.add((topic, RDF.type, fbf.Topic))
            g.add((topic, containsbimsnippet, bimsnippet))
            g.add((topic, containsdocumentreference, 
documentreference))
            g.add((topic, containsrelatedtopic, relatedtopic))
            g.add((topic, hastopicguid, Literal(guid_selection, 
datatype=XSD.string)))
            g.add((topic, hastopictype, Literal(result[3], 
datatype=XSD.string)))
            g.add((topic, hastopicstatus, Literal(result[4], 
datatype=XSD.string)))
            g.add((topic, hasreferencelink, Literal(result[5], 
datatype=XSD.string)))
            g.add((topic, hastitle, Literal(result[6], 
datatype=XSD.string)))
            g.add((topic, haspriority, Literal(result[7], 
datatype=XSD.string)))
            g.add((topic, hastopicindex, Literal(result[8])))
            g.add((topic, haslabels, Literal(result[9], 
datatype=XSD.string)))
            g.add((topic, hascreationdate, Literal(result[10], 
datatype=XSD.dateTime)))
            g.add((topic, hascreationauthor, Literal(result[11], 
datatype=XSD.string)))
            g.add((topic, hasmodifieddate, Literal(result[12])))
            g.add((topic, hasmodifiedauthor, Literal(result[13], 
datatype=XSD.string)))
            g.add((topic, hasduedate, Literal(result[14])))
            g.add((topic, hasassignedto, Literal(result[15], 
datatype=XSD.string)))
            g.add((topic, hasstage, Literal(result[16], 
datatype=XSD.string)))
            g.add((topic, hastopicdescription, Literal(result[17], 
datatype=XSD.string)))

            # BimSnippet triples
            g.add((bimsnippet, RDF.type, fbf.BimSnippet))
            g.add((bimsnippet, hasbimsnippetreference, 
Literal(result[18], datatype=XSD.string)))
            g.add((bimsnippet, hasreferenceschema, Literal(result[19], 
datatype=XSD.string)))
            g.add((bimsnippet, hasbimsnippetisexternal, 
Literal('True', datatype=XSD.boolean)))
            g.add((bimsnippet, hassnippettype, Literal('SnippetType', 
datatype=XSD.string)))

            # DocumentReference triples
            g.add((documentreference, RDF.type, 
fbf.DocumentReference))
            g.add((documentreference, hasreferenceddocument, 
Literal(ftp_filelocation())))
            g.add((documentreference, hasdocumentdescription, 
Literal(result[21], datatype=XSD.string)))
            g.add((documentreference, hasdocumentguid, 
Literal('Document Guid', datatype=XSD.string)))
            g.add((documentreference, hasdocumentisexternal, 
Literal('True', datatype=XSD.boolean)))

            # RelatedTopic triples
            g.add((relatedtopic, RDF.type, fbf.RelatedTopic))
            g.add((relatedtopic, hasrelatedtopicguid, 
Literal('RelatedTopicGuid', datatype=XSD.string)))
```

```
            # Comment triples
            g.add((comment, RDF.type, fbf.Comment))
            g.add((comment, containsviewpoint, viewpoint))
            g.add((comment, hascommentguid, Literal(y,
datatype=XSD.string)))
            g.add((comment, hascommentdate, Literal(result[22],
datatype=XSD.dateTime)))
            g.add((comment, hasauthor, Literal(result[23],
datatype=XSD.string)))
            g.add((comment, hascomment, Literal(result[24],
datatype=XSD.string)))
            g.add((comment, hascommentmoddate, Literal(result[25])))
            g.add((comment, hascommentmodauthor, Literal(result[26],
datatype=XSD.string)))

            # Viewpoint triples
            g.add((viewpoint, RDF.type, fbf.Viewpoint))
            g.add((viewpoint, hascommentviewpoinguid, Literal(z,
datatype=XSD.string)))

            # Viewpoints triples
            g.add((viewpoints, RDF.type, fbf.Viewpoints))
            g.add((viewpoints, hasviewpointsguid, Literal(z,
datatype=XSD.string)))
            g.add((viewpoints, hasviewpoint, Literal(result[27],
datatype=XSD.string)))
            g.add((viewpoints, hassnapshot, Literal(result[28],
datatype=XSD.string)))
            g.add((viewpoints, hasviewpointsindex,
Literal(result[29])))

            # Creating the file
            rdfdata = g.serialize(format='pretty-xml')

            # Uploading data to the repository
            repository = 'V17001Fran'
            # graph = 'file://C:/fakepath/RDFtest1.rdf'
            graph = next_context_name()
            params = {'context': '<' + graph + '>'}
            print params
            # endpoint =
"http://192.168.5.2/repositories/%s/statements?%s" % (repository,
urllib.urlencode(params))
            endpoint = "http://sparql.verhoeven-
leenders.nl/repositories/%s/statements?%s" % (repository,
urllib.urlencode(params))
            data = rdfdata
            (response, content) = httplib2.Http().request(endpoint,
'PUT', body=data, headers={'content-type': 'application/rdf+xml'})
            print "Response %s" % response.status

        def context_number():
            q = """
PREFIX fbf:<http://example.org/bcfschema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX fbfd: <http://example.org/data#>
SELECT ?Markup ?c ?Date ?Comment
WHERE {
    ?Markup fbf:containsTopic ?b.
    ?b fbf:hasTopicGuid "%s".
    ?Markup fbf:containsComment ?c.
    ?c fbf:hasComment ?Comment.
    ?c fbf:hasCommentDate ?Date.
}
ORDER BY ?Date""" % guid_selection

            sparql = SPARQLWrapper("http://sparql.verhoeven-
leenders.nl/repositories/V17001Fran")
            # sparql =
SPARQLWrapper("http://192.168.5.2/repositories/V17001Fran")
            sparql.setQuery(q)

            sparql.setReturnFormat(RDFXML)
            results = sparql.query().convert()

            mylist = re.split(",|\r\n", results)
            for i in range(4):
                mylist.remove(mylist[0])

            textandnumber = mylist[0]
            contextnumber =
textandnumber.replace("http://example.org/data#Markup","")
            return contextnumber

        def sub_comment():
            q = """
PREFIX fbf:<http://example.org/bcfschema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX fbfd: <http://example.org/data#>
SELECT ?c
WHERE {
    ?Markup fbf:containsTopic ?b.
    ?b fbf:hasTopicGuid "%s".
    ?Markup fbf:containsComment ?c.
    ?c fbf:hasComment ?Comment.
    ?c fbf:hasCommentDate ?Date.
}
ORDER BY ?Date""" % guid_selection

            sparql = SPARQLWrapper("http://sparql.verhoeven-
leenders.nl/repositories/V17001Fran")
            # sparql =
SPARQLWrapper("http://192.168.5.2/repositories/V17001Fran")
            sparql.setQuery(q)

            sparql.setReturnFormat(RDFXML)
            results = sparql.query().convert()

            mylist = re.split(",|\r\n", results)

            numberofcomments = len(mylist)-2
            return numberofcomments

        def create_RDF_graph2():
            graph1 = ConjunctiveGraph()
            graph2 = ConjunctiveGraph()

            # Namespaces
            fbf = Namespace("http://example.org/bcfschema#")
            fbfd = Namespace("http://example.org/data#")
            graph2.namespace_manager.bind('fbf', fbf)
            graph2.namespace_manager.bind('fbfd', fbfd)

            # Classes
            markup = URIRef("http://example.org/data#Markup%s") %
context_number()
            topic = URIRef("http://example.org/data#Topic%s") %
context_number()
            comment = URIRef("http://example.org/data#Comment%s_%d") %
(context_number(), sub_comment())

            # Properties
            # Hierarchy
            containscomment = URIRef(fbf.containsComment)

            # TopicProperties
            hastopicstatus = URIRef(fbf.hasTopicStatus)

            # Comment properties
            hascommentguid = URIRef(fbf.hasCommentGuid)
            hascommentdate = URIRef(fbf.hasCommentDate)
            hasauthor = URIRef(fbf.hasAuthor)
            hascomment = URIRef(fbf.hasComment)

            # Triples
            # Markuptriples
            graph2.add((markup, containscomment, comment))

            # Topictriples
            graph2.add((topic, hastopicstatus, Literal(result[0],
datatype=XSD.string)))

            # Comment triples
            graph2.add((comment, RDF.type, fbf.Comment))
            graph2.add((comment, hascommentguid, Literal(y,
datatype=XSD.string)))
            graph2.add((comment, hascommentdate, Literal(result[1],
datatype=XSD.dateTime)))
            graph2.add((comment, hasauthor, Literal(result[2],
datatype=XSD.string)))
            graph2.add((comment, hascomment, Literal(result[3],
datatype=XSD.string)))

            repository = 'V17001Fran'
            graph = 'file://C:/fakepath/RDFtest%s.rdf' %
context_number()
            params = {'context': '<' + graph + '>'}
            endpoint = "http://sparql.verhoeven-
leenders.nl/repositories/%s/statements?%s" % (repository,
urllib.urlencode(params))
            # endpoint =
"http://192.168.5.2/repositories/%s/statements?%s" % (repository,
urllib.urlencode(params))
            graph1.parse(endpoint)
            graph1.remove((None, fbf.hasTopicStatus, None))
            graph3 = graph1 + graph2
            rdfdata = graph3.serialize(format='pretty-xml')

            data = rdfdata
            (response, content) = httplib2.Http().request(endpoint,
'PUT', body=data, headers={'content-type': 'application/rdf+xml'})
            print "Response %s" % response.status

        q = """
PREFIX fbf:<http://example.org/bcfschema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX fbfd: <http://example.org/data#>
SELECT ?Markup ?Date ?Comment
WHERE {
    ?Markup fbf:containsTopic ?b.
    ?b fbf:hasTopicGuid "%s".
    ?Markup fbf:containsComment ?c.
    ?c fbf:hasComment ?Comment.
    ?c fbf:hasCommentDate ?Date.
}
ORDER BY ?Date""" % guid_selection

        sparql = SPARQLWrapper("http://sparql.verhoeven-
leenders.nl/repositories/V17001Fran")
        # sparql =
SPARQLWrapper("http://192.168.5.2/repositories/V17001Fran")
        sparql.setQuery(q)

        sparql.setReturnFormat(RDFXML)
        results = sparql.query().convert()

        mylist = re.split(",|\r\n", results)
        a = len(mylist)

        if a == 4:
            app = MyApp(ifcfilename, ifcfilelocation, dateandtime)
            result = app.mainloop()
            create_RDF_graph()
        else:
            app = MyApp2(dateandtime)
            result = app.mainloop()
            create_RDF_graph2()

    def query_getComments(self):
        if not self.filename:
            QtGui.QMessageBox.warning(self,
                                      "No IFC loaded!",
                                      "Please load a model first!")
            return

        q = """
PREFIX fbf:<http://example.org/bcfschema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX fbfd: <http://example.org/data#>
SELECT ?Date ?Comment ?Status
WHERE {
  ?Markup fbf:containsTopic ?b.
  ?b fbf:hasTopicGuid "%s".
  ?b fbf:hasTopicStatus ?Status.
  ?Markup fbf:containsComment ?c.
  ?c fbf:hasComment ?Comment.
  ?c fbf:hasCommentDate ?Date.
}
ORDER BY ?Date""" % guid_selection
```

```
        sparql = SPARQLWrapper("http://sparql.verhoeven-
leenders.nl/repositories/V17001Fran")
        # sparql =
SPARQLWrapper("http://192.168.5.2/repositories/V17001Fran")
        sparql.setQuery(q)

        sparql.setReturnFormat(RDFXML)
        results = sparql.query().convert()
        numberofresults = results.count("\r\n")

        if numberofresults == 1:
            self.propertybox.clear()
            self.propertybox2.clear()
            self.propertybox.append("There is no BCF information")
        else:
            self.propertybox.clear()
            self.propertybox2.clear()
            self.propertybox.append(results)

    def query_getAttachedFiles(self):
        if not self.filename:
            QtGui.QMessageBox.warning(self,
                                      "No IFC loaded!",
                                      "Please load a model first!")
            return

        q = """
PREFIX fbf:<http://example.org/bcfschema#>
SELECT ?ReferencedDocument
WHERE {
  ?Markup fbf:containsTopic ?b.
  ?b fbf:hasTopicGuid "%s".
  ?b fbf:containsDocumentReference ?d.
  ?d fbf:hasReferencedDocument ?ReferencedDocument.
}""" % guid_selection

        sparql = SPARQLWrapper("http://sparql.verhoeven-
leenders.nl/repositories/V17001Fran")
        # sparql =
SPARQLWrapper("http://192.168.5.2/repositories/V17001Fran")
        sparql.setQuery(q)

        sparql.setReturnFormat(RDFXML)
        results = sparql.query().convert()
        print results
        numberofresults = results.count("\r\n")
        print numberofresults

        mylist = re.split(",|\r\n", results)
        print mylist

        if numberofresults == 1:
            self.propertybox2.clear()
            self.propertybox2.append("There is no file attached")
        else:
            if mylist[-2] == ("None" or ''):
                self.propertybox2.clear()
                self.propertybox2.append("There is no file attached")
            else:
                self.propertybox2.clear()
                self.propertybox2.append(results)

    def show_notclosed_objects(self):
        if not self.filename:
            QtGui.QMessageBox.warning(self,
                                        "No IFC loaded!",
                                        "Please load a model first!")
            return

        rooms = self.ifc_file.by_type("IfcBuildingElement")
        for room in rooms:

            q = """
    PREFIX fbf:<http://example.org/bcfschema#>
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    PREFIX fbfd: <http://example.org/data#>
    SELECT ?Status
    WHERE {
        ?Markup fbf:containsTopic ?b.
        ?b fbf:hasTopicGuid "%s".
        ?b fbf:hasTopicStatus ?Status.
        ?Markup fbf:containsComment ?c.
        ?c fbf:hasCommentDate ?Date.
    }
    ORDER BY ?Date""" % room.GlobalId

            sparql = SPARQLWrapper("http://sparql.verhoeven-
leenders.nl/repositories/V17001Fran")
            # sparql =
SPARQLWrapper("http://192.168.5.2/repositories/V17001Fran")
            sparql.setQuery(q)

            sparql.setReturnFormat(RDFXML)
            results = sparql.query().convert()
            mylist = results.split("\r\n")
            a = len(mylist)-2
            b = len(mylist)

            if mylist[a] == "Active":
                if room.Representation:
                    self.canvas.ColorasActive(room.GlobalId)
            else:
                if b == 2:
                    if room.Representation:
                        self.canvas.ColorasNoCheck(room.GlobalId)
                else:
                    if mylist[a] == "Resolved":
                        if room.Representation:
                            self.canvas.ColorasResolved(room.GlobalId)
                    else:
                        if room.Representation:
                            self.canvas.Hide(room.GlobalId)

    def show_closed_objects(self):
        if not self.filename:
            QtGui.QMessageBox.warning(self,
                                        "No IFC loaded!",
                                        "Please load a model first!")
            return
        rooms = self.ifc_file.by_type("IfcBuildingElement")
        for room in rooms:

            q = """
    PREFIX fbf:<http://example.org/bcfschema#>
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    PREFIX fbfd: <http://example.org/data#>
    SELECT ?Status
    WHERE {
        ?Markup fbf:containsTopic ?b.
        ?b fbf:hasTopicGuid "%s".
        ?b fbf:hasTopicStatus ?Status.
        ?Markup fbf:containsComment ?c.
        ?c fbf:hasCommentDate ?Date.
    }
    ORDER BY ?Date""" % room.GlobalId

            sparql = SPARQLWrapper("http://sparql.verhoeven-
leenders.nl/repositories/V17001Fran")
            # sparql =
SPARQLWrapper("http://192.168.5.2/repositories/V17001Fran")
            sparql.setQuery(q)

            sparql.setReturnFormat(RDFXML)
            results = sparql.query().convert()
            mylist = results.split("\r\n")
            a = len(mylist)-2

            if mylist[a] == "Closed":
                if room.Representation:
                    self.canvas.ColorasClosed(room.GlobalId)
            else:
                if room.Representation:
                    self.canvas.Hide(room.GlobalId)

    def show_assigned_objects(self):
        name = self.show_assigned_objects2()
        if name == "":
            self.propertybox.append("No name has been introduced")
        else:
            rooms = self.ifc_file.by_type("IfcBuildingElement")
            for room in rooms:

                q = """
    PREFIX fbf:<http://example.org/bcfschema#>
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    PREFIX fbfd: <http://example.org/data#>
    SELECT ?Assigned ?Status
    WHERE {
        ?Markup fbf:containsTopic ?b.
        ?b fbf:hasTopicGuid "%s".
        ?b fbf:hasAssignedTo ?Assigned.
        ?b fbf:hasTopicStatus ?Status.
        ?Markup fbf:containsComment ?c.
        ?c fbf:hasCommentDate ?Date.
    }
    ORDER BY ?Date""" % room.GlobalId

                sparql = SPARQLWrapper("http://sparql.verhoeven-
leenders.nl/repositories/V17001Fran")
                # sparql =
SPARQLWrapper("http://192.168.5.2/repositories/V17001Fran")
                sparql.setQuery(q)

                sparql.setReturnFormat(RDFXML)
                results = sparql.query().convert()
                mylist = re.split(",|\r\n", results)
                c = len(mylist)-3

                if mylist[c] == name:
                    print "It is assigned to %s" % name
                else:
                    if room.Representation:
                        self.canvas.Hide(room.GlobalId)

    def show_assigned_objects2(self):
        if not self.filename:
            QtGui.QMessageBox.warning(self,
                                            "No IFC loaded!",
                                            "Please load a model first!")
            return

        app = MyApp4()
        result = app.mainloop()
        return result

    def mark_as_closed(self):
        dateandtime = datetime.datetime.now().strftime("%Y-%m-%d
%H:%M:%S")
        if not self.filename:
            QtGui.QMessageBox.warning(self,
                                          "No IFC loaded!",
                                          "Please load a model first!")
            return

        if guid_selection == None:
            QtGui.QMessageBox.warning(self, "Select element first!",
                                          "Please select the element that
you want to create the BCF for")

            return

        product = self.ifc_file.by_type("IfcProject")
        pguid = product[0].GlobalId

        ifcfilename = os.path.basename(str(self.filename))

        ifcfilelocation = self.filename

        # Guids
        y = uuid.uuid4()

        def next_context_name():
            repository2 = 'V17001Fran'
            graph2 = 'file://C:/fakepath/RDFtest%d.rdf'
            params2 = {'context': '<' + graph2 + '>'}
            # endpoint2 = "http://192.168.5.2/repositories/%s/rdf-
graphs?%s" % (repository2, urllib.urlencode(params2))
            endpoint2 = "http://sparql.verhoeven-
```

```
leenders.nl/repositories/%s/rdf-graphs?%s" % (repository2,
urllib.urlencode(params2))
            (response, content) = httplib2.Http().request(endpoint2)
            return graph2 % content.count('\n')

        def next_file_name():
            result2 = []
            markup = 'http://example.org/data#Markup%d'
            header = 'http://example.org/data#Header%d'
            file = 'http://example.org/data#File%d'
            topic = 'http://example.org/data#Topic%d'
            bimsnippet = 'http://example.org/data#BimSnippet%d'
            documentreference =
'http://example.org/data#DocumentReference%d'
            relatedtopic = 'http://example.org/data#RelatedTopic%d'
            comment = 'http://example.org/data#Comment%d'
            viewpoint = 'http://example.org/data#Viewpoint%d'
            viewpoints = 'http://example.org/data#Viewpoints%d'

            repository2 = 'V17001Fran'
            graph2 = 'file://C:/fakepath/RDFtest%d.rdf'
            params2 = {'context': '<' + graph2 + '>'}
            # endpoint2 = "http://192.168.5.2/repositories/%s/rdf-
graphs?%s" % (repository2, urllib.urlencode(params2))
            endpoint2 = "http://sparql.verhoeven-
leenders.nl/repositories/%s/rdf-graphs?%s" % (repository2,
urllib.urlencode(params2))
            (response, content) = httplib2.Http().request(endpoint2)

            r1 = markup % content.count('\n')
            r2 = header % content.count('\n')
            r3 = file % content.count('\n')
            r4 = topic % content.count('\n')
            r5 = bimsnippet % content.count('\n')
            r6 = documentreference % content.count('\n')
            r7 = relatedtopic % content.count('\n')
            r8 = comment % content.count('\n')
            r9 = viewpoint % content.count('\n')
            r10 = viewpoints % content.count('\n')

            result2.append(r1)
            result2.append(r2)
            result2.append(r3)
            result2.append(r4)
            result2.append(r5)
            result2.append(r6)
            result2.append(r7)
            result2.append(r8)
            result2.append(r9)
            result2.append(r10)

            return result2

        def create_RDF_graph():
            g = Graph()

            # Namespaces
            fbf = Namespace("http://example.org/bcfschema#")
            # fbfd = Namespace("http://example.org/data#")
            g.namespace_manager.bind('fbf', fbf)
            # Classes
            markup = URIRef(next_file_name()[0])
            header = URIRef(next_file_name()[1])
            file = URIRef(next_file_name()[2])
            topic = URIRef(next_file_name()[3])
            bimsnippet = URIRef(next_file_name()[4])
            documentreference = URIRef(next_file_name()[5])
            relatedtopic = URIRef(next_file_name()[6])
            comment = URIRef(next_file_name()[7])
            viewpoint = URIRef(next_file_name()[8])
            viewpoints = URIRef(next_file_name()[9])

            # Properties
            # Hierarchy properties
            containsheader = URIRef(fbf.containsHeader)
            containsfile = URIRef(fbf.containsFile)
            containstopic = URIRef(fbf.containsTopic)
            containscomment = URIRef(fbf.containsComment)
            containsviewpoints = URIRef(fbf.containsViewpoints)
            containsbimsnippet = URIRef(fbf.containsBimSnippet)
            containsdocumentreference =
URIRef(fbf.containsDocumentReference)
            containsrelatedtopic = URIRef(fbf.containsRelatedTopic)
            containsviewpoint = URIRef(fbf.containsViewpoint)

            # File properties
            hasfilename = URIRef(fbf.hasFileName)
            hasfiledate = URIRef(fbf.hasFileDate)
            hasfilereference = URIRef(fbf.hasFileReference)
            hasifcproject = URIRef(fbf.hasIfcProject)
            hasifcspatialstructureelement =
URIRef(fbf.hasIfcSpatialStructureElement)
            hasfileisexternal = URIRef(fbf.hasFileIsExternal)

            # TopicProperties
            hastopicguid = URIRef(fbf.hasTopicGuid)
            hastopictype = URIRef(fbf.hasTopicType)
            hastopicstatus = URIRef(fbf.hasTopicStatus)
            hasreferencelink = URIRef(fbf.hasReferenceLink)
            hastitle = URIRef(fbf.hasTitle)
            haspriority = URIRef(fbf.hasPriority)
            hastopicindex = URIRef(fbf.hasTopicIndex)
            haslabels = URIRef(fbf.hasLabels)
            hascreationdate = URIRef(fbf.hasCreationDate)
            hascreationauthor = URIRef(fbf.hasCreationAuthor)
            hasmodifieddate = URIRef(fbf.hasModifiedDate)
            hasmodifiedauthor = URIRef(fbf.hasModifiedAuthor)
            hasduedate = URIRef(fbf.hasDueDate)
            hasassignedto = URIRef(fbf.hasAssignedTo)
            hasstage = URIRef(fbf.hasStage)
            hastopicdescription = URIRef(fbf.hasTopicDescription)

            # BimSnippet properties
            hasbimsnippetreference =
URIRef(fbf.hasBimSnippetReference)
            hasreferenceschema = URIRef(fbf.hasReferenceSchema)
            hasbimsnippetisexternal =
URIRef(fbf.hasBimSnippetIsExternal)
            hassnippettype = URIRef(fbf.hasSnippetType)
            # DocumentReference properties
            hasreferenceddocument = URIRef(fbf.hasReferencedDocument)
            hasdocumentdescription =
URIRef(fbf.hasDocumentReferenceDescription)
            hasdocumentguid = URIRef(fbf.hasDocumentReferenceGuid)
            hasdocumentisexternal =
URIRef(fbf.hasDocumentReferenceIsExternal)

            # RelatedTopic properties
            hasrelatedtopicguid = URIRef(fbf.hasRelatedTopicGuid)

            # Comment properties
            hascommentguid = URIRef(fbf.hasCommentGuid)
            hascommentdate = URIRef(fbf.hasCommentDate)
            hasauthor = URIRef(fbf.hasAuthor)
            hascomment = URIRef(fbf.hasComment)
            hascommentmoddate = URIRef(fbf.hasCommentModifiedDate)
            hascommentmodauthor = URIRef(fbf.hasCommentModifiedAuthor)

            # Viewpoint properties
            hascommentviewpoinguid = URIRef(fbf.hasViewpointGuid)

            # Viewpoints properties
            hasviewpointsguid = URIRef(fbf.hasViewpointsGuid)
            hasviewpoint = URIRef(fbf.hasViewpoint)
            hassnapshot = URIRef(fbf.hasSnapshot)
            hasviewpointsindex = URIRef(fbf.hasViewpointsIndex)

            # Triples
            # Markuptriples
            g.add((markup, RDF.type, fbf.Markup))
            g.add((markup, containsheader, header))
            g.add((markup, containstopic, topic))
            g.add((markup, containscomment, comment))
            g.add((markup, containsviewpoints, viewpoints))

            # Headertriples
            g.add((header, RDF.type, fbf.Header))
            g.add((header, containsfile, file))

            # Filetriples
            g.add((file, RDF.type, fbf.File))
            g.add((file, hasfilename, Literal(ifcfilename,
datatype=XSD.string)))
            g.add((file, hasfiledate, Literal(dateandtime,
datatype=XSD.dateTime)))
            g.add((file, hasfilereference, Literal(ifcfilelocation,
datatype=XSD.string)))
            g.add((file, hasifcproject, Literal(pguid,
datatype=XSD.string)))
            g.add((file, hasifcspatialstructureelement,
Literal('IfcGuid2', datatype=XSD.string)))
            g.add((file, hasfileisexternal, Literal('True',
datatype=XSD.boolean)))

            # Topictriples
            g.add((topic, RDF.type, fbf.Topic))
            g.add((topic, containsbimsnippet, bimsnippet))
            g.add((topic, containsdocumentreference,
documentreference))
            g.add((topic, containsrelatedtopic, relatedtopic))
            g.add((topic, hastopicguid, Literal(guid_selection,
datatype=XSD.string)))
            g.add((topic, hastopictype, Literal("Closure",
datatype=XSD.string)))
            g.add((topic, hastopicstatus, Literal(result[0],
datatype=XSD.string)))
            g.add((topic, hasreferencelink, Literal("",
datatype=XSD.string)))
            g.add((topic, hastitle, Literal("Closure",
datatype=XSD.string)))
            g.add((topic, haspriority, Literal("",
datatype=XSD.string)))
            g.add((topic, hastopicindex, Literal("")))
            g.add((topic, haslabels, Literal("",
datatype=XSD.string)))
            g.add((topic, hascreationdate, Literal(result[1],
datatype=XSD.dateTime)))
            g.add((topic, hascreationauthor, Literal(result[2],
datatype=XSD.string)))
            g.add((topic, hasmodifieddate, Literal("")))
            g.add((topic, hasmodifiedauthor, Literal("",
datatype=XSD.string)))
            g.add((topic, hasduedate, Literal("")))
            g.add((topic, hasassignedto, Literal("",
datatype=XSD.string)))
            g.add((topic, hasstage, Literal(result[3],
datatype=XSD.string)))
            g.add((topic, hastopicdescription, Literal("",
datatype=XSD.string)))

            # BimSnippet triples
            g.add((bimsnippet, RDF.type, fbf.BimSnippet))
            g.add((bimsnippet, hasbimsnippetreference, Literal("",
datatype=XSD.string)))
            g.add((bimsnippet, hasreferenceschema, Literal("",
datatype=XSD.string)))
            g.add((bimsnippet, hasbimsnippetisexternal, Literal("",
datatype=XSD.boolean)))
            g.add((bimsnippet, hassnippettype, Literal("",
datatype=XSD.string)))

            # DocumentReference triples
            g.add((documentreference, RDF.type,
fbf.DocumentReference))
            g.add((documentreference, hasreferenceddocument,
Literal("")))
            g.add((documentreference, hasdocumentdescription,
Literal("", datatype=XSD.string)))
            g.add((documentreference, hasdocumentguid, Literal("",
datatype=XSD.string)))
            g.add((documentreference, hasdocumentisexternal,
Literal("", datatype=XSD.boolean)))

            # RelatedTopic triples
            g.add((relatedtopic, RDF.type, fbf.RelatedTopic))
            g.add((relatedtopic, hasrelatedtopicguid, Literal("",
datatype=XSD.string)))
```

```
            # Comment triples
            g.add((comment, RDF.type, fbf.Comment))
            g.add((comment, containsviewpoint, viewpoint))
            g.add((comment, hascommentguid, Literal(y,
datatype=XSD.string)))
            g.add((comment, hascommentdate, Literal(result[1],
datatype=XSD.dateTime)))
            g.add((comment, hasauthor, Literal(result[2],
datatype=XSD.string)))
            g.add((comment, hascomment, Literal(result[4],
datatype=XSD.string)))
            g.add((comment, hascommentmoddate, Literal("")))
            g.add((comment, hascommentmodauthor, Literal("",
datatype=XSD.string)))

            # Viewpoint triples
            g.add((viewpoint, RDF.type, fbf.Viewpoint))
            g.add((viewpoint, hascommentviewpoinguid, Literal("",
datatype=XSD.string)))

            # Viewpoints triples
            g.add((viewpoints, RDF.type, fbf.Viewpoints))
            g.add((viewpoints, hasviewpointsguid, Literal("",
datatype=XSD.string)))
            g.add((viewpoints, hasviewpoint, Literal("",
datatype=XSD.string)))
            g.add((viewpoints, hassnapshot, Literal("",
datatype=XSD.string)))
            g.add((viewpoints, hasviewpointsindex, Literal("")))

            # Creating the file
            rdfdata = g.serialize(format='pretty-xml')

            # Uploading data to the repository
            repository = 'V17001Fran'
            # graph = 'file://C:/fakepath/RDFtest1.rdf'
            graph = next_context_name()
            params = {'context': '<' + graph + '>'}
            print params
            # endpoint =
"http://192.168.5.2/repositories/%s/statements?%s" % (repository,
urllib.urlencode(params))
            endpoint = "http://sparql.verhoeven-
leenders.nl/repositories/%s/statements?%s" % (repository,
urllib.urlencode(params))
            data = rdfdata
            (response, content) = httplib2.Http().request(endpoint,
'PUT', body=data, headers={'content-type': 'application/rdf+xml'})
            print "Response %s" % response.status

        def context_number():
            q = """
PREFIX fbf:<http://example.org/bcfschema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX fbfd: <http://example.org/data#>
SELECT ?Markup ?c ?Date ?Comment
WHERE {
    ?Markup fbf:containsTopic ?b.
    ?b fbf:hasTopicGuid "%s".
    ?Markup fbf:containsComment ?c.
    ?c fbf:hasComment ?Comment.
    ?c fbf:hasCommentDate ?Date.
}
ORDER BY ?Date""" % guid_selection

            sparql = SPARQLWrapper("http://sparql.verhoeven-
leenders.nl/repositories/V17001Fran")
            # sparql =
SPARQLWrapper("http://192.168.5.2/repositories/V17001Fran")
            sparql.setQuery(q)

            sparql.setReturnFormat(RDFXML)
            results = sparql.query().convert()

            mylist = re.split(",|\r\n", results)
            for i in range(4):
                mylist.remove(mylist[0])

            textandnumber = mylist[0]
            contextnumber =
textandnumber.replace("http://example.org/data#Markup", "")
            return contextnumber

        def sub_comment():
            q = """
PREFIX fbf:<http://example.org/bcfschema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX fbfd: <http://example.org/data#>
SELECT ?Markup ?c ?Date ?Comment
WHERE {
    ?Markup fbf:containsTopic ?b.
    ?b fbf:hasTopicGuid "%s".
    ?Markup fbf:containsComment ?c.
    ?c fbf:hasComment ?Comment.
    ?c fbf:hasCommentDate ?Date.
}
ORDER BY ?Date""" % guid_selection

            sparql = SPARQLWrapper("http://sparql.verhoeven-
leenders.nl/repositories/V17001Fran")
            # sparql =
SPARQLWrapper("http://192.168.5.2/repositories/V17001Fran")
            sparql.setQuery(q)

            sparql.setReturnFormat(RDFXML)
            results = sparql.query().convert()

            mylist = re.split(",|\r\n", results)

            number = mylist[-4]
            commentnumber =
number.replace("http://example.org/data#Comment", "")
            t = "%s_" % context_number()
            commentnumber2 = commentnumber.replace(t, "")
            commentnumber3 = int(commentnumber2)

            return commentnumber3 + 1

        def create_RDF_graph2():
            graph1 = ConjunctiveGraph()
            graph2 = ConjunctiveGraph()

            # Namespaces
            fbf = Namespace("http://example.org/bcfschema#")
            fbfd = Namespace("http://example.org/data#")
            graph2.namespace_manager.bind('fbf', fbf)
            graph2.namespace_manager.bind('fbfd', fbfd)

            # Classes
            markup = URIRef("http://example.org/data#Markup%s") %
context_number()
            topic = URIRef("http://example.org/data#Topic%s") %
context_number()
            comment = URIRef("http://example.org/data#Comment%s_%d") %
(context_number(), sub_comment())

            # Properties
            # Hierarchy
            containscomment = URIRef(fbf.containsComment)

            # TopicProperties
            hastopicstatus = URIRef(fbf.hasTopicStatus)

            # Comment properties
            hascommentguid = URIRef(fbf.hasCommentGuid)
            hascommentdate = URIRef(fbf.hasCommentDate)
            hasauthor = URIRef(fbf.hasAuthor)
            hascomment = URIRef(fbf.hasComment)

            # Triples
            # Markuptriples
            graph2.add((markup, containscomment, comment))

            # Topictriples
            graph2.add((topic, hastopicstatus, Literal(result[0],
datatype=XSD.string)))

            # Comment triples
            graph2.add((comment, RDF.type, fbf.Comment))
            graph2.add((comment, hascommentguid, Literal(y,
datatype=XSD.string)))
            graph2.add((comment, hascommentdate, Literal(result[1],
datatype=XSD.dateTime)))
            graph2.add((comment, hasauthor, Literal(result[2],
datatype=XSD.string)))
            graph2.add((comment, hascomment, Literal(result[3],
datatype=XSD.string)))

            repository = 'V17001Fran'
            graph = 'file://C:/fakepath/RDFtest%s.rdf' %
context_number()
            params = {'context': '<' + graph + '>'}
            endpoint = "http://sparql.verhoeven-
leenders.nl/repositories/%s/statements?%s" % (repository,
urllib.urlencode(params))
            # endpoint =
"http://192.168.5.2/repositories/%s/statements?%s" % (repository,
urllib.urlencode(params))
            graph1.parse(endpoint)
            graph1.remove((None, fbf.hasTopicStatus, None))
            graph3 = graph1 + graph2
            rdfdata = graph3.serialize(format='pretty-xml')

            data = rdfdata
            (response, content) = httplib2.Http().request(endpoint,
'PUT', body=data, headers={'content-type': 'application/rdf+xml'})
            print "Response %s" % response.status

        q = """
PREFIX fbf:<http://example.org/bcfschema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX fbfd: <http://example.org/data#>
SELECT ?Markup ?Date ?Comment
WHERE {
    ?Markup fbf:containsTopic ?b.
    ?b fbf:hasTopicGuid "%s".
    ?Markup fbf:containsComment ?c.
    ?c fbf:hasComment ?Comment.
    ?c fbf:hasCommentDate ?Date.
}
ORDER BY ?Date""" % guid_selection

        sparql = SPARQLWrapper("http://sparql.verhoeven-
leenders.nl/repositories/V17001Fran")
        # sparql =
SPARQLWrapper("http://192.168.5.2/repositories/V17001Fran")
        sparql.setQuery(q)

        sparql.setReturnFormat(RDFXML)
        results = sparql.query().convert()

        mylist = re.split(",|\r\n", results)
        a = len(mylist)

        if a == 4:
            app = MyApp3(dateandtime)
            result = app.mainloop()
            create_RDF_graph()
        else:
            app = MyApp2(dateandtime)
            result = app.mainloop()
            create_RDF_graph2()

    def tab_compatibility(self):
        vbox = QtGui.QVBoxLayout()
        hbox = QtGui.QHBoxLayout()
        hbox2 = QtGui.QHBoxLayout()
        hbox3 = QtGui.QHBoxLayout()
        vbox2 = QtGui.QVBoxLayout()
        vbox3 = QtGui.QVBoxLayout()

        vbox.addLayout(hbox)

        vbox.addLayout(hbox2)
        hbox2.addLayout(vbox2)
        hbox2.addLayout(vbox3)
        vbox3.addLayout(hbox3)
        self.compatibility_tab.setLayout(vbox)

        vbox2.setSpacing(50)
        vbox3.setSpacing(50)
        hbox3.setSpacing(10)

        upload_download_BCF_introduction = QtGui.QLabel(
            """
            This tab provides backwards and external compatibility for
the RDF repository with the BCF Manager from KUBUS.
            The first button allows to convert all the RDF contexts
contained in the RDF repository to a single BCF zip file
            that could be opened and read with the BCF manager.
            The second button allows to convert a BCF zip file back as
the RDF contexts that the RDF repository contains and
            upload it in order to update the repository.
            """
        )
        upload_download_BCF_introduction.setMaximumHeight(150)
        upload_download_BCF_introduction.setWordWrap(True)

        downloadBCF_btn_title = QtGui.QLabel("1. Download RDF contexts
contained in the RDF repository")
        downloadBCF_btn = QtGui.QPushButton("Download RDF contexts",
self)
        downloadBCF_btn.clicked.connect(self.download_bcf_file)

        uploadBCF_btn_title = QtGui.QLabel("2. Upload BCF information
to the RDF repository:")
        uploadBCF_btn = QtGui.QPushButton("Select BCF here", self)
        uploadBCF_btn.clicked.connect(self.open_bcf_file)

        hbox.addWidget(upload_download_BCF_introduction)
        vbox2.addWidget(downloadBCF_btn_title)
        vbox2.addWidget(uploadBCF_btn_title)

        hbox3.addWidget(downloadBCF_btn)
        vbox3.addWidget(uploadBCF_btn)

    def download_bcf_file(self):
        def create_BCF_xml():
            # Expand IFC Topic Guid
            expandedtopicguid =
ifcopenshell.guid.split(ifcopenshell.guid.expand(mylist[18]))[1:-1]

            os.chdir("C:/Users/Fran/Desktop")
            location = "C:/Users/Fran/Desktop/test/" +
expandedtopicguid
            os.makedirs(location)

            root = ET.Element("Markup")

            header = ET.SubElement(root, "Header")

            file = ET.SubElement(header, "File", IfcProject=mylist[3])

            ET.SubElement(file, "Filename").text = mylist[0]
            ET.SubElement(file, "Date").text = mylist[1]
            ET.SubElement(file, "Reference").text = mylist[2]

            topic = ET.SubElement(root, "Topic",
Guid=expandedtopicguid, TopicType=mylist[21], TopicStatus=mylist[20])
```

```
            # ET.SubElement(topic, "ReferenceLink").text = mylist[14]
            ET.SubElement(topic, "Title").text = mylist[16]
            ET.SubElement(topic, "Priority").text = mylist[13]
            ET.SubElement(topic, "Index").text = mylist[19]
            ET.SubElement(topic, "Labels").text = mylist[10]
            ET.SubElement(topic, "CreationDate").text = mylist[7]
            ET.SubElement(topic, "CreationAuthor").text = mylist[8]
            ET.SubElement(topic, "ModifiedDate").text = mylist[12]
            ET.SubElement(topic, "ModifiedAuthor").text = mylist[11]
            # Conversion datetime
            DD = mylist[9]
            if DD == "":
                print "Closure"
            else:
                DDmod = '%s-%s-%sT00:00:00+00:00' % (DD[-4:], DD[3:5],
DD[:2])
                ET.SubElement(topic, "DueDate").text = DDmod
            ET.SubElement(topic, "AssignedTo").text = mylist[6]
            ET.SubElement(topic, "Stage").text = mylist[15]
            ET.SubElement(topic, "Description").text = mylist[17]

            # bimsnippet = ET.SubElement(topic, "BimSnippet",
SnippetType=mylist[25], isExternal=mylist[22])
            #
            # ET.SubElement(bimsnippet, "Reference").text = mylist[23]
            # ET.SubElement(bimsnippet, "ReferenceSchema").text =
mylist[24]
            #
            # documentreference = ET.SubElement(topic,
"DocumentReference", Guid=mylist[27], isExternal=mylist[28])
            #
            # ET.SubElement(documentreference,
"ReferencedDocument").text = mylist[29]
            # ET.SubElement(documentreference, "Description").text =
mylist[26]
            #
            # relatedtopic = ET.SubElement(topic, "RelatedTopic",
Guid=mylist[30])

            for s in range(numberofchunks - 1):
                comment = ET.SubElement(root, "Comment",
Guid=chunks[s][2])

                # ET.SubElement(comment, "VerbalStatus").text =
mylist[20]
                # ET.SubElement(comment, "Status").text = mylist[21]
                ET.SubElement(comment, "Date").text = chunks[s][1]
                ET.SubElement(comment, "Author").text = chunks[s][0]
                ET.SubElement(comment, "Comment").text = chunks[s][5]
                ET.SubElement(comment, "Viewpoint", Guid=chunks[s][6])
                # ET.SubElement(comment, "ModifiedDate").text =
mylist[35]
                # ET.SubElement(comment, "ModifiedAuthor").text =
mylist[34]

            viewpoints = ET.SubElement(root, "Viewpoints",
Guid=mylist[40])

            ET.SubElement(viewpoints, "Viewpoint").text = mylist[39]
            ET.SubElement(viewpoints, "Snapshot").text = mylist[38]
            # ET.SubElement(viewpoints, "Index").text = mylist[41]

            tree = ET.ElementTree(root)
            tree.write(location + "/markup.bcf")

        repository2 = 'V17001Fran'
        graph2 = 'file://C:/fakepath/RDFtest%d.rdf'
        params2 = {'context': '<' + graph2 + '>'}
        # endpoint2 = "http://192.168.5.2/repositories/%s/rdf-
graphs?%s" % (repository2, urllib.urlencode(params2))
        endpoint2 = "http://sparql.verhoeven-
leenders.nl/repositories/%s/rdf-graphs?%s" % (repository2,
urllib.urlencode(params2))
        (response, content) = httplib2.Http().request(endpoint2)

        numberofcontexts = content.count("\n")

        for n in range(numberofcontexts - 1):

            q = """
        PREFIX fbf: <http://example.org/bcfschema#>
        SELECT ?Fname ?Fdate ?Fref ?IfcProject ?IfcSpaStrEle ?FisExt
?AssignedTo ?CreationDate ?CreationAuthor ?DueDate ?Labels ?ModAuthor
?ModDate ?priority ?RefLink ?Stage ?Title ?TopDescrip ?TopGuid
?TopIndex ?TopStatus ?TopType ?bsIsExt ?bsRef ?bsRefSche ?bsType ?drd
?drg ?drIsExt ?refdoc ?rtg ?author ?CommentDate ?CommentGuid
?CModAuthor ?CModDate ?Comment ?VGuid ?snapshot ?viewpoint
?viewpointsGuid ?viewpointsIndex
        WHERE {
        GRAPH <file://C:/fakepath/RDFtest%s.rdf> {
            ?m a fbf:Markup .
              OPTIONAL{
            ?m fbf:containsHeader ?h .
            ?h fbf:containsFile ?f .
            ?f fbf:hasFileName ?Fname .
            ?f fbf:hasFileDate ?Fdate .
            ?f fbf:hasFileReference ?Fref .
            ?f fbf:hasIfcProject ?IfcProject .
            ?f fbf:hasIfcSpatialStructureElement ?IfcSpaStrEle .
            ?f fbf:hasFileIsExternal ?FisExt .
            ?m fbf:containsTopic ?t .
            ?t fbf:hasAssignedTo ?AssignedTo .
            ?t fbf:hasCreationAuthor ?CreationAuthor .
            ?t fbf:hasCreationDate ?CreationDate .
            ?t fbf:hasDueDate ?DueDate .
            ?t fbf:hasLabels ?Labels .
            ?t fbf:hasModifiedAuthor ?ModAuthor .
            ?t fbf:hasModifiedDate ?ModDate .
            ?t fbf:hasPriority ?priority .
            ?t fbf:hasReferenceLink ?RefLink .
            ?t fbf:hasStage ?Stage .
            ?t fbf:hasTitle ?Title .
            ?t fbf:hasTopicDescription ?TopDescrip .
            ?t fbf:hasTopicGuid ?TopGuid .
            ?t fbf:hasTopicIndex ?TopIndex .
            ?t fbf:hasTopicStatus ?TopStatus .
            ?t fbf:hasTopicType ?TopType .
            ?t fbf:containsBimSnippet ?b .
            ?b fbf:hasBimSnippetIsExternal ?bsIsExt .
            ?b fbf:hasBimSnippetReference ?bsRef .
            ?b fbf:hasReferenceSchema ?bsRefSche .
            ?b fbf:hasSnippetType ?bsType .
            ?t fbf:containsDocumentReference ?dr .
            ?dr fbf:hasDocumentReferenceDescription ?drd .
            ?dr fbf:hasDocumentReferenceGuid ?drg .
            ?dr fbf:hasDocumentReferenceIsExternal ?drIsExt .
            ?dr fbf:hasReferencedDocument ?refdoc .
            ?t fbf:containsRelatedTopic ?rt .
            ?rt fbf:hasRelatedTopicGuid ?rtg .}
            ?m fbf:containsComment ?c .
            ?c fbf:hasAuthor ?author .
            ?c fbf:hasCommentDate ?CommentDate .
            ?c fbf:hasCommentGuid ?CommentGuid .
            ?c fbf:hasComment ?Comment .
            OPTIONAL{
            ?c fbf:hasCommentModifiedAuthor ?CModAuthor .
            ?c fbf:hasCommentModifiedDate ?CModDate .}
            OPTIONAL {
            ?c fbf:containsViewpoint ?v .
            ?v fbf:hasViewpointGuid ?VGuid .
            ?m fbf:containsViewpoints ?vs .
            ?vs fbf:hasSnapshot ?snapshot .
            ?vs fbf:hasViewpoint ?viewpoint .
            ?vs fbf:hasViewpointsGuid ?viewpointsGuid .
            ?vs fbf:hasViewpointsIndex ?viewpointsIndex . }
        }
        }
        ORDER BY ?CommentDate""" % int(n + 1)

            sparql = SPARQLWrapper("http://sparql.verhoeven-
leenders.nl/repositories/V17001Fran")
            # sparql =
SPARQLWrapper("http://192.168.5.2/repositories/V17001Fran")
            sparql.setQuery(q)
            sparql.setMethod("POST")
            sparql.setReturnFormat(RDFXML)
            results = sparql.query().convert()

            mylist = re.split(",|\r\n", results)

            for i in range(42):
                mylist.remove(mylist[0])

            chunks = [mylist[x:x + 42] for x in xrange(0, len(mylist),
42)]

            numberofchunks = results.count("\r\n")

            for s in range(numberofchunks - 1):
                for i in range(31):
                    chunks[s].remove(chunks[s][0])

            create_BCF_xml()

        # Create Zip file and erase folder
        shutil.make_archive('RDFcontexts', 'zip', 'test')
        shutil.rmtree("C:/Users/Fran/Desktop/test")

    def open_bcf_file(self, bcffilename=None):
        self.bcffilename = QtGui.QFileDialog.getOpenFileName(self,
'Open file', ".", "Bim Collaboration Format (*.zip)")

        markuppath = self.bcffilename

        def next_context_name():
            repository2 = 'V17001Fran'
            graph2 = 'file://C:/fakepath/RDFtest%d.rdf'
            params2 = {'context': '<' + graph2 + '>'}
            # endpoint2 = "http://192.168.5.2/repositories/%s/rdf-
graphs?%s" % (repository2, urllib.urlencode(params2))
            endpoint2 = "http://sparql.verhoeven-
leenders.nl/repositories/%s/rdf-graphs?%s" % (repository2,
urllib.urlencode(params2))
            (response, content) = httplib2.Http().request(endpoint2)
            return graph2 % content.count('\n')

        def next_file_name():
            result2 = []
            markup = 'http://example.org/data#Markup%d'
            header = 'http://example.org/data#Header%d'
            file = 'http://example.org/data#File%d'
            topic = 'http://example.org/data#Topic%d'
            bimsnippet = 'http://example.org/data#BimSnippet%d'
            documentreference =
'http://example.org/data#DocumentReference%d'
            relatedtopic = 'http://example.org/data#RelatedTopic%d'
            comment = 'http://example.org/data#Comment%d'
            viewpoint = 'http://example.org/data#Viewpoint%d'
            viewpoints = 'http://example.org/data#Viewpoints%d'

            repository2 = 'V17001Fran'
            graph2 = 'file://C:/fakepath/RDFtest%d.rdf'
            params2 = {'context': '<' + graph2 + '>'}
            # endpoint2 = "http://192.168.5.2/repositories/%s/rdf-
graphs?%s" % (repository2, urllib.urlencode(params2))
            endpoint2 = "http://sparql.verhoeven-
leenders.nl/repositories/%s/rdf-graphs?%s" % (repository2,
urllib.urlencode(params2))
            (response, content) = httplib2.Http().request(endpoint2)

            r1 = markup % content.count('\n')
            r2 = header % content.count('\n')
            r3 = file % content.count('\n')
            r4 = topic % content.count('\n')
            r5 = bimsnippet % content.count('\n')
            r6 = documentreference % content.count('\n')
            r7 = relatedtopic % content.count('\n')
            r8 = comment % content.count('\n')
            r9 = viewpoint % content.count('\n')
            r10 = viewpoints % content.count('\n')

            result2.append(r1)
            result2.append(r2)
            result2.append(r3)
            result2.append(r4)
            result2.append(r5)
            result2.append(r6)
            result2.append(r7)
            result2.append(r8)
            result2.append(r9)
            result2.append(r10)

            return result2

        def create_RDF_graph():
            g = ConjunctiveGraph()

            # Namespaces
            fbf = Namespace("http://example.org/bcfschema#")
            # fbfd = Namespace("http://example.org/data#")
            g.namespace_manager.bind('fbf', fbf)

            # Classes
            markup = URIRef(next_file_name()[0])
            header = URIRef(next_file_name()[1])
            file = URIRef(next_file_name()[2])
            topic = URIRef(next_file_name()[3])
            bimsnippet = URIRef(next_file_name()[4])
            documentreference = URIRef(next_file_name()[5])
            relatedtopic = URIRef(next_file_name()[6])
            comment = URIRef(next_file_name()[7])
            viewpoint = URIRef(next_file_name()[8])
            viewpoints = URIRef(next_file_name()[9])

            # Properties
            # Hierarchy properties
            containsheader = URIRef(fbf.containsHeader)
            containsfile = URIRef(fbf.containsFile)
            containstopic = URIRef(fbf.containsTopic)
            containscomment = URIRef(fbf.containsComment)
            containsviewpoints = URIRef(fbf.containsViewpoints)
            containsbimsnippet = URIRef(fbf.containsBimSnippet)
            containsdocumentreference =
URIRef(fbf.containsDocumentReference)
            containsrelatedtopic = URIRef(fbf.containsRelatedTopic)
            containsviewpoint = URIRef(fbf.containsViewpoint)

            # File properties
            hasfilename = URIRef(fbf.hasFileName)
            hasfiledate = URIRef(fbf.hasFileDate)
            hasfilereference = URIRef(fbf.hasFileReference)
            hasifcproject = URIRef(fbf.hasIfcProject)
            hasifcspatialstructureelement =
URIRef(fbf.hasIfcSpatialStructureElement)
            hasfileisexternal = URIRef(fbf.hasFileIsExternal)

            # TopicProperties
            hastopicguid = URIRef(fbf.hasTopicGuid)
            hastopictype = URIRef(fbf.hasTopicType)
            hastopicstatus = URIRef(fbf.hasTopicStatus)
            hasreferencelink = URIRef(fbf.hasReferenceLink)
            hastitle = URIRef(fbf.hasTitle)
            haspriority = URIRef(fbf.hasPriority)
            hastopicindex = URIRef(fbf.hasTopicIndex)
            haslabels = URIRef(fbf.hasLabels)
            hascreationdate = URIRef(fbf.hasCreationDate)
            hascreationauthor = URIRef(fbf.hasCreationAuthor)
            hasmodifieddate = URIRef(fbf.hasModifiedDate)
            hasmodifiedauthor = URIRef(fbf.hasModifiedAuthor)
            hasduedate = URIRef(fbf.hasDueDate)
            hasassignedto = URIRef(fbf.hasAssignedTo)
            hasstage = URIRef(fbf.hasStage)
            hastopicdescription = URIRef(fbf.hasTopicDescription)

            # BimSnippet properties
            hasbimsnippetreference =
URIRef(fbf.hasBimSnippetReference)
            hasreferenceschema = URIRef(fbf.hasReferenceSchema)
            hasbimsnippetisexternal =
URIRef(fbf.hasBimSnippetIsExternal)
            hassnippettype = URIRef(fbf.hasSnippetType)

            # DocumentReference properties
            hasreferenceddocument = URIRef(fbf.hasReferencedDocument)
            hasdocumentdescription =
URIRef(fbf.hasDocumentReferenceDescription)
            hasdocumentguid = URIRef(fbf.hasDocumentReferenceGuid)
            hasdocumentisexternal =
URIRef(fbf.hasDocumentReferenceIsExternal)

            # RelatedTopic properties
            hasrelatedtopicguid = URIRef(fbf.hasRelatedTopicGuid)

            # Comment properties
            hascommentguid = URIRef(fbf.hasCommentGuid)
            hascommentdate = URIRef(fbf.hasCommentDate)
            hasauthor = URIRef(fbf.hasAuthor)
            hascomment = URIRef(fbf.hasComment)
            hascommentmoddate = URIRef(fbf.hasCommentModifiedDate)
            hascommentmodauthor = URIRef(fbf.hasCommentModifiedAuthor)

            # Viewpoint properties
            hascommentviewpoinguid = URIRef(fbf.hasViewpointGuid)

            # Viewpoints properties
            hasviewpointsguid = URIRef(fbf.hasViewpointsGuid)
            hasviewpoint = URIRef(fbf.hasViewpoint)
            hassnapshot = URIRef(fbf.hasSnapshot)
            hasviewpointsindex = URIRef(fbf.hasViewpointsIndex)

            # Triples
            # Markuptriples
            g.add((markup, RDF.type, fbf.Markup))
            g.add((markup, containsheader, header))
            g.add((markup, containstopic, topic))
            g.add((markup, containscomment, comment))
            g.add((markup, containsviewpoints, viewpoints))

            # Headertriples
            g.add((header, RDF.type, fbf.Header))
            g.add((header, containsfile, file))

            # Filetriples
            g.add((file, RDF.type, fbf.File))
            g.add((file, hasfilename, Literal("Filename",
datatype=XSD.string)))
            g.add((file, hasfiledate, Literal("Filedate",
datatype=XSD.dateTime)))
            g.add((file, hasfilereference, Literal("Filereference",
datatype=XSD.string)))
```

```
            g.add((file, hasifcproject, Literal("IfcProject",
datatype=XSD.string)))
            g.add((file, hasifcspatialstructureelement,
Literal('IfcGuid2', datatype=XSD.string)))
            g.add((file, hasfileisexternal, Literal('True',
datatype=XSD.boolean)))

            # Topictriples
            g.add((topic, RDF.type, fbf.Topic))
            g.add((topic, containsbimsnippet, bimsnippet))
            g.add((topic, containsdocumentreference,
documentreference))
            g.add((topic, containsrelatedtopic, relatedtopic))
            g.add((topic, hastopicguid, Literal(compressedTopicGuid,
datatype=XSD.string)))
            g.add((topic, hastopictype, Literal(topicattributevalue3,
datatype=XSD.string)))
            g.add((topic, hastopicstatus,
Literal(topicattributevalue2, datatype=XSD.string)))
            g.add((topic, hasreferencelink, Literal("",
datatype=XSD.string)))
            g.add((topic, hastitle, Literal(titlevalue,
datatype=XSD.string)))
            g.add((topic, haspriority, Literal(priorityvalue,
datatype=XSD.string)))
            g.add((topic, hastopicindex, Literal(indexvalue)))
            g.add((topic, haslabels, Literal(labelsvalue,
datatype=XSD.string)))
            g.add((topic, hascreationdate, Literal(creationdatevalue,
datatype=XSD.dateTime)))
            g.add((topic, hascreationauthor,
Literal(creationauthorvalue, datatype=XSD.string)))
            g.add((topic, hasmodifieddate,
Literal(modifieddatevalue)))
            g.add((topic, hasmodifiedauthor,
Literal(modifiedauthorvalue, datatype=XSD.string)))
            g.add((topic, hasduedate, Literal("")))
            g.add((topic, hasassignedto, Literal(assignedtovalue,
datatype=XSD.string)))
            g.add((topic, hasstage, Literal(stagevalue,
datatype=XSD.string)))
            g.add((topic, hastopicdescription,
Literal(descriptionvalue, datatype=XSD.string)))

            # BimSnippet triples
            g.add((bimsnippet, RDF.type, fbf.BimSnippet))
            g.add((bimsnippet, hasbimsnippetreference, Literal("",
datatype=XSD.string)))
            g.add((bimsnippet, hasreferenceschema, Literal("",
datatype=XSD.string)))
            g.add((bimsnippet, hasbimsnippetisexternal, Literal("",
datatype=XSD.boolean)))
            g.add((bimsnippet, hassnippettype, Literal("",
datatype=XSD.string)))

            # DocumentReference triples
            g.add((documentreference, RDF.type,
fbf.DocumentReference))
            g.add((documentreference, hasreferenceddocument,
Literal("")))
```

```
            g.add((documentreference, hasdocumentdescription,
Literal("", datatype=XSD.string)))
            g.add((documentreference, hasdocumentguid, Literal("",
datatype=XSD.string)))
            g.add((documentreference, hasdocumentisexternal,
Literal("", datatype=XSD.boolean)))

            # RelatedTopic triples
            g.add((relatedtopic, RDF.type, fbf.RelatedTopic))
            g.add((relatedtopic, hasrelatedtopicguid, Literal("",
datatype=XSD.string)))

            # Comment triples
            for i in range(len(listofcommentguids)):
                if i == 0:

                    g.add((comment, RDF.type, fbf.Comment))
                    g.add((comment, containsviewpoint, viewpoint))
                    g.add((comment, hascommentguid,
Literal(listofcommentguids[i], datatype=XSD.string)))
                    g.add((comment, hascommentdate,
Literal(listofcommentdates[i], datatype=XSD.dateTime)))
                    g.add((comment, hasauthor,
Literal(listofcommentauthors[i], datatype=XSD.string)))
                    g.add((comment, hascomment,
Literal(listofcomments[i], datatype=XSD.string)))
                    g.add((comment, hascommentmoddate, Literal("")))
                    g.add((comment, hascommentmodauthor, Literal("",
datatype=XSD.string)))

                else:
                    g.add((markup, containscomment, comment + "_%s" %
i))
                    g.add((comment + "_%s" % i, RDF.type,
fbf.Comment))
                    g.add((comment + "_%s" % i, hascommentguid,
Literal(listofcommentguids[i], datatype=XSD.string)))
                    g.add((comment + "_%s" % i, hascommentdate,
Literal(listofcommentdates[i], datatype=XSD.dateTime)))
                    g.add((comment + "_%s" % i, hasauthor,
Literal(listofcommentauthors[i], datatype=XSD.string)))
                    g.add((comment + "_%s" % i, hascomment,
Literal(listofcomments[i], datatype=XSD.string)))

            # Viewpoint triples
            g.add((viewpoint, RDF.type, fbf.Viewpoint))
            g.add((viewpoint, hascommentviewpoinguid, Literal("",
datatype=XSD.string)))

            # Viewpoints triples
            g.add((viewpoints, RDF.type, fbf.Viewpoints))
            g.add((viewpoints, hasviewpointsguid, Literal("",
datatype=XSD.string)))
            g.add((viewpoints, hasviewpoint, Literal("",
datatype=XSD.string)))
            g.add((viewpoints, hassnapshot, Literal("",
datatype=XSD.string)))
            g.add((viewpoints, hasviewpointsindex, Literal("")))

            # Creating the file
            rdfdata = g.serialize(format='pretty-xml')

            # Uploading data to the repository
            repository = 'V17001Fran'
            # graph = 'file://C:/fakepath/RDFtest1.rdf'
            graph = next_context_name()
            params = {'context': '<' + graph + '>'}
            print params
            # endpoint =
"http://192.168.5.2/repositories/%s/statements?%s" % (repository,
urllib.urlencode(params))
            endpoint = "http://sparql.verhoeven-
leenders.nl/repositories/%s/statements?%s" % (repository,
urllib.urlencode(params))
            data = rdfdata
            (response, content) = httplib2.Http().request(endpoint,
'PUT', body=data, headers={'content-type': 'application/rdf+xml'})
            print "Response %s" % response.status

        def context_number():
            q = """
        PREFIX fbf:<http://example.org/bcfschema#>
        PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
        PREFIX fbfd: <http://example.org/data#>
        SELECT ?Markup ?c ?Date ?Comment
        WHERE {
            ?Markup fbf:containsTopic ?b.
            ?b fbf:hasTopicGuid "%s".
            ?Markup fbf:containsComment ?c.
            ?c fbf:hasComment ?Comment.
            ?c fbf:hasCommentDate ?Date.
        }
        ORDER BY ?Date""" % compressedTopicGuid

            sparql = SPARQLWrapper("http://sparql.verhoeven-
leenders.nl/repositories/V17001Fran")
            # sparql =
SPARQLWrapper("http://192.168.5.2/repositories/V17001Fran")
            sparql.setQuery(q)

            sparql.setReturnFormat(RDFXML)
            results = sparql.query().convert()

            mylist = re.split(",|\r\n", results)
            for i in range(4):
                mylist.remove(mylist[0])

            textandnumber = mylist[0]
            contextnumber =
textandnumber.replace("http://example.org/data#Markup", "")
            return contextnumber

        def sub_comment():
            q = """
        PREFIX fbf:<http://example.org/bcfschema#>
        PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
        PREFIX fbfd: <http://example.org/data#>
        SELECT ?Markup ?c ?Date ?Comment
        WHERE {
            ?Markup fbf:containsTopic ?b.
            ?b fbf:hasTopicGuid "%s".
            ?Markup fbf:containsComment ?c.
            ?c fbf:hasComment ?Comment.
            ?c fbf:hasCommentDate ?Date.
        }
        ORDER BY ?Date""" % compressedTopicGuid

            sparql = SPARQLWrapper("http://sparql.verhoeven-
leenders.nl/repositories/V17001Fran")
            # sparql =
SPARQLWrapper("http://192.168.5.2/repositories/V17001Fran")
            sparql.setQuery(q)

            sparql.setReturnFormat(RDFXML)
            results = sparql.query().convert()

            mylist = re.split(",|\r\n", results)

            number = mylist[-4]
            commentnumber =
number.replace("http://example.org/data#Comment", "")
            t = "%s_" % context_number()
            commentnumber2 = commentnumber.replace(t, "")
            commentnumber3 = int(commentnumber2)

            return commentnumber3 + 1

        def create_RDF_graph2():
            graph1 = ConjunctiveGraph()
            graph2 = ConjunctiveGraph()

            # Namespaces
            fbf = Namespace("http://example.org/bcfschema#")
            fbfd = Namespace("http://example.org/data#")
            graph2.namespace_manager.bind('fbf', fbf)
            graph2.namespace_manager.bind('fbfd', fbfd)

            # Classes
            markup = URIRef("http://example.org/data#Markup%s") %
context_number()
            topic = URIRef("http://example.org/data#Topic%s") %
context_number()
            comment = URIRef("http://example.org/data#Comment%s") %
context_number()

            # Properties
            # Hierarchy
            containscomment = URIRef(fbf.containsComment)

            # TopicProperties
            hastopicstatus = URIRef(fbf.hasTopicStatus)

            # Comment properties
            hascommentguid = URIRef(fbf.hasCommentGuid)
            hascommentdate = URIRef(fbf.hasCommentDate)
            hasauthor = URIRef(fbf.hasAuthor)
            hascomment = URIRef(fbf.hasComment)

            # Triples
            # Topictriples
            graph2.add((topic, hastopicstatus,
Literal(topicattributevalue2, datatype=XSD.string)))

            # Comment triples
            for i in range((int(results.count("\r\n"))-1),
len(listofcommentguids)):
                graph2.add((markup, containscomment, comment + "_%s" %
i))
                graph2.add((comment + "_%s" % i, RDF.type,
fbf.Comment))
                graph2.add((comment + "_%s" % i, hascommentguid,
Literal(listofcommentguids[i], datatype=XSD.string)))
                graph2.add((comment + "_%s" % i, hascommentdate,
Literal(listofcommentdates[i], datatype=XSD.dateTime)))
                graph2.add((comment + "_%s" % i, hasauthor,
Literal(listofcommentauthors[i], datatype=XSD.string)))
                graph2.add((comment + "_%s" % i, hascomment,
Literal(listofcomments[i], datatype=XSD.string)))

            repository = 'V17001Fran'
            graph = 'file://C:/fakepath/RDFtest%s.rdf' %
context_number()
            params = {'context': '<' + graph + '>'}
            endpoint = "http://sparql.verhoeven-
leenders.nl/repositories/%s/statements?%s" % (repository,
urllib.urlencode(params))
            # endpoint =
"http://192.168.5.2/repositories/%s/statements?%s" % (repository,
urllib.urlencode(params))
            graph1.parse(endpoint)
            graph1.remove((None, fbf.hasTopicStatus, None))
            graph3 = graph1 + graph2
            rdfdata = graph3.serialize(format='pretty-xml')

            data = rdfdata
            (response, content) = httplib2.Http().request(endpoint,
'PUT', body=data, headers={'content-type': 'application/rdf+xml'})
            print "Response %s" % response.status

        zip_file = str(markuppath)
        zip_file2 = zip_file.replace(".zip", "")

        os.chdir(os.path.dirname(zip_file))
        zip_ref = zipfile.ZipFile(zip_file, 'r')
        zip_ref.extractall(os.path.basename(str(zip_file2)))
        zip_ref.close()

        dir = zip_file2
        for file in glob.iglob(os.path.join(dir, '*/*.bcf')):
            with open(file) as f:
                tree = ET.ElementTree(file=f)
                root = tree.getroot()

                # fileattribute = tree.find('Header/File')
                # file = tree.find('Header/File/Filename')
                # filedate = tree.find('Header/File/Date')
                # filereference = tree.find('Header/File/Reference')
                #
                # fileattributevalue = fileattribute.get('IfcProject')
                # filevalue = file.text
                # filedatevalue = filedate.text
                # filereferencevalue = filereference.text

                topicattribute = tree.find('Topic')
                title = tree.find('Topic/Title')
                priority = tree.find('Topic/Priority')
                index = tree.find('Topic/Index')
                labels = tree.find('Topic/Labels')
                creationdate = tree.find('Topic/CreationDate')
                creationauthor = tree.find('Topic/CreationAuthor')
                modifieddate = tree.find('Topic/ModifiedDate')
                modifiedauthor = tree.find('Topic/ModifiedAuthor')
                assignedto = tree.find('Topic/AssignedTo')
                stage = tree.find('Topic/Stage')
                description = tree.find('Topic/Description')

                topicattributevalue1 = topicattribute.get('Guid')
                # Compress Topic Guid
                compressedTopicGuid =
ifcopenshell.guid.compress(topicattributevalue1.replace('-', ''))
                topicattributevalue2 =
topicattribute.get('TopicStatus')
                topicattributevalue3 = topicattribute.get('TopicType')
                titlevalue = title.text

                if titlevalue == "Closure":
                    print "Closure context is not uploaded"
                else:
                    priorityvalue = priority.text
                    indexvalue = index.text
                    labelsvalue = labels.text
                    creationdatevalue = creationdate.text
                    creationauthorvalue = creationauthor.text
                    modifieddatevalue = modifieddate.text
                    modifiedauthorvalue = modifiedauthor.text
                    assignedtovalue = assignedto.text
                    stagevalue = stage.text
                    descriptionvalue = description.text

                    listofcommentguids = []
                    listofcommentdates = []
                    listofcommentauthors = []
                    listofcomments = []

                    for comments in root.findall(".//Comment[@Guid]"):
                        a = comments.attrib.get('Guid')
                        listofcommentguids.append(a)

                    for dates in root.findall(".//Comment/Date"):
                        b = dates.text
                        listofcommentdates.append(b)

                    for authors in root.findall(".//Comment/Author"):
                        c = authors.text
                        listofcommentauthors.append(c)

                    for comment in root.findall(".//Comment/Comment"):
                        d = comment.text
                        listofcomments.append(d)
                    # viewpointsattribute = tree.find('Viewpoints')
                    # viewpoint = tree.find('Viewpoints/Viewpoint')
                    # snapshot = tree.find('Viewpoints/Snapshot')
                    #
                    # viewpointsattributevalue =
viewpointsattribute.get('Guid')
                    # viewpointvalue = viewpoint.text
                    # snapshotvalue = snapshot.text

                    q = """
        PREFIX fbf:<http://example.org/bcfschema#>
        PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
        PREFIX fbfd: <http://example.org/data#>
        SELECT ?Markup ?Date ?Comment
        WHERE {
            ?Markup fbf:containsTopic ?b.
            ?b fbf:hasTopicGuid "%s".
            ?Markup fbf:containsComment ?c.
            ?c fbf:hasComment ?Comment.
            ?c fbf:hasCommentDate ?Date.
        }
        ORDER BY ?Date""" % compressedTopicGuid

                    sparql = SPARQLWrapper("http://sparql.verhoeven-
leenders.nl/repositories/V17001Fran")
                    # sparql =
SPARQLWrapper("http://192.168.5.2/repositories/V17001Fran")
                    sparql.setQuery(q)
                    sparql.setReturnFormat(RDFXML)
                    results = sparql.query().convert()

                    mylist = re.split(",|\r\n", results)
                    a = len(mylist)

                    if a == 4:
                        print "There is NO information"
                        create_RDF_graph()
                    else:
                        print "There is information"
                        create_RDF_graph2()

        shutil.rmtree(zip_file2)


    def closeEvent(self, event):
        result = QtGui.QMessageBox.question(self,
                                            "Confirm Exit",
                                            "Are you sure you want to
exit ?",
                                            QtGui.QMessageBox.Yes |
QtGui.QMessageBox.No)
        event.ignore()

        if result == QtGui.QMessageBox.Yes:
            event.accept()

init = initUI()
```

7.6 Appendix VI: Ontology BCF schema (TTL)

```
# baseURI: http://example.org/bcfschema

@prefix : <http://example.org/bcfschema#> .
@prefix fbf: <http://example.org/bcfschema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://example.org/bcfschema>
  rdf:type owl:Ontology ;
  owl:versionInfo "Created with TopBraid Composer" ;
.
fbf:BimSnippet
  rdf:type owl:Class ;
  rdfs:subClassOf owl:Thing ;
.
fbf:Comment
  rdf:type owl:Class ;
  rdfs:subClassOf owl:Thing ;
.
fbf:DocumentReference
  rdf:type owl:Class ;
  rdfs:subClassOf owl:Thing ;
.
fbf:File
  rdf:type owl:Class ;
  rdfs:subClassOf owl:Thing ;
.
fbf:Header
  rdf:type owl:Class ;
  rdfs:subClassOf owl:Thing ;
.
fbf:Markup
  rdf:type owl:Class ;
  rdfs:subClassOf owl:Thing ;
.
fbf:RelatedTopic
  rdf:type owl:Class ;
  rdfs:subClassOf owl:Thing ;
.
fbf:Topic
  rdf:type owl:Class ;
  rdfs:subClassOf owl:Thing ;
.
fbf:Viewpoint
  rdf:type owl:Class ;
  rdfs:subClassOf owl:Thing ;
.
fbf:Viewpoints
  rdf:type owl:Class ;
  rdfs:subClassOf owl:Thing ;
.
fbf:containsBimSnippet
  rdf:type owl:ObjectProperty ;
  rdfs:domain fbf:Topic ;
  rdfs:range fbf:BimSnippet ;
.
```

```
fbf:containsComment
  rdf:type owl:ObjectProperty ;
  rdfs:domain fbf:Markup ;
  rdfs:range fbf:Comment ;
.
fbf:containsDocumentReference
  rdf:type owl:ObjectProperty ;
  rdfs:domain fbf:Topic ;
  rdfs:range fbf:DocumentReference ;
.
fbf:containsFile
  rdf:type owl:ObjectProperty ;
  rdfs:domain fbf:Header ;
  rdfs:range fbf:File ;
.
fbf:containsHeader
  rdf:type owl:ObjectProperty ;
  rdfs:domain fbf:Markup ;
  rdfs:range fbf:Header ;
.
fbf:containsRelatedTopic
  rdf:type owl:ObjectProperty ;
  rdfs:domain fbf:Topic ;
  rdfs:range fbf:RelatedTopic ;
.
fbf:containsTopic
  rdf:type owl:ObjectProperty ;
  rdfs:domain fbf:Markup ;
  rdfs:range fbf:Topic ;
.
fbf:containsViewpoint
  rdf:type owl:ObjectProperty ;
  rdfs:domain fbf:Comment ;
  rdfs:range fbf:Viewpoint ;
.
fbf:containsViewpoints
  rdf:type owl:ObjectProperty ;
  rdfs:domain fbf:Markup ;
  rdfs:range fbf:Viewpoints ;
.
fbf:hasAssignedTo
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Topic ;
  rdfs:range xsd:string ;
.
fbf:hasAuthor
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Comment ;
  rdfs:range xsd:string ;
.
fbf:hasBimSnippetIsExternal
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:BimSnippet ;
  rdfs:range xsd:boolean ;
.
fbf:hasBimSnippetReference
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:BimSnippet ;
  rdfs:range xsd:string ;
.
```

```
fbf:hasComment
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Comment ;
  rdfs:range xsd:string ;
.
fbf:hasCommentDate
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Comment ;
  rdfs:range xsd:dateTime ;
.
fbf:hasCommentGuid
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Comment ;
  rdfs:range xsd:string ;
.
fbf:hasCommentModifiedAuthor
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Comment ;
  rdfs:range xsd:string ;
.
fbf:hasCommentModifiedDate
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Comment ;
  rdfs:range xsd:dateTime ;
.
fbf:hasCreationAuthor
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Topic ;
  rdfs:range xsd:string ;
.
fbf:hasCreationDate
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Topic ;
  rdfs:range xsd:dateTime ;
.
fbf:hasDocumentReferenceDescription
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:DocumentReference ;
  rdfs:range xsd:string ;
.
fbf:hasDocumentReferenceGuid
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:DocumentReference ;
  rdfs:range xsd:string ;
.
fbf:hasDocumentReferenceIsExternal
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:DocumentReference ;
  rdfs:range xsd:boolean ;
.
fbf:hasDueDate
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Topic ;
  rdfs:range xsd:dateTime ;
.
fbf:hasFileDate
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:File ;
  rdfs:range xsd:dateTime ;
.
```

```
fbf:hasFileIsExternal
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:File ;
  rdfs:range xsd:boolean ;
.
fbf:hasFileName
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:File ;
  rdfs:range xsd:string ;
.
fbf:hasFileReference
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:File ;
  rdfs:range xsd:string ;
.
fbf:hasIfcProject
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:File ;
  rdfs:range xsd:string ;
.
fbf:hasIfcSpatialStructureElement
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:File ;
  rdfs:range xsd:string ;
.
fbf:hasLabels
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Topic ;
  rdfs:range xsd:string ;
.
fbf:hasModifiedAuthor
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Topic ;
  rdfs:range xsd:string ;
.
fbf:hasModifiedDate
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Topic ;
  rdfs:range xsd:dateTime ;
.
fbf:hasPriority
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Topic ;
  rdfs:range xsd:string ;
.
fbf:hasReferenceLink
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Topic ;
  rdfs:range xsd:string ;
.
fbf:hasReferenceSchema
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:BimSnippet ;
  rdfs:range xsd:string ;
.
fbf:hasReferencedDocument
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:DocumentReference ;
  rdfs:range xsd:string ;
.
```

```
fbf:hasRelatedTopicGuid
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:RelatedTopic ;
  rdfs:range xsd:string ;
.
fbf:hasSnapshot
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Viewpoints ;
  rdfs:range xsd:string ;
.
fbf:hasSnippetType
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:BimSnippet ;
  rdfs:range xsd:string ;
.
fbf:hasStage
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Topic ;
  rdfs:range xsd:string ;
.
fbf:hasTitle
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Topic ;
  rdfs:range xsd:string ;
.
fbf:hasTopicDescription
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Topic ;
  rdfs:range xsd:string ;
.
fbf:hasTopicGuid
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Topic ;
  rdfs:range xsd:string ;
.
fbf:hasTopicIndex
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Topic ;
  rdfs:range xsd:integer ;
.
fbf:hasTopicStatus
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Topic ;
  rdfs:range xsd:string ;
.
fbf:hasTopicType
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Topic ;
  rdfs:range xsd:string ;
.
fbf:hasViewpoint
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Viewpoints ;
  rdfs:range xsd:string ;
.
fbf:hasViewpointGuid
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Viewpoint ;
  rdfs:range xsd:string ;
.
```

```
fbf:hasViewpointsGuid
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Viewpoints ;
  rdfs:range xsd:string ;
.
fbf:hasViewpointsIndex
  rdf:type owl:DatatypeProperty ;
  rdfs:domain fbf:Viewpoints ;
  rdfs:range xsd:integer ;
.
```

7.7 Appendix VII: BCF Schema (XSD)

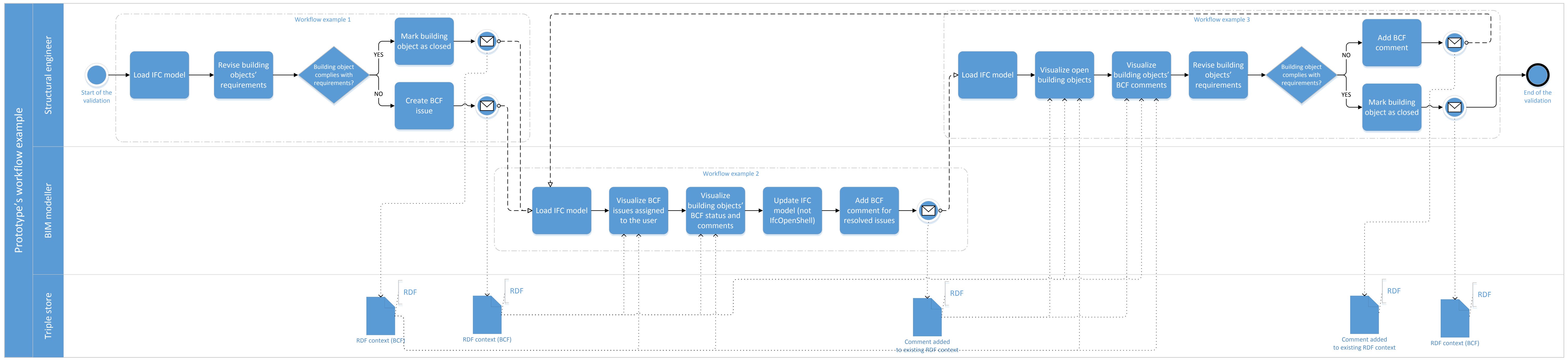
```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Mit XMLSpy v2011 rel. 2 sp1 (http://www.altova.com) von Klaus
Linhard (IABI e.V.) bearbeitet -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:element name="Markup">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="Header" type="Header"
minOccurs="0"/>
				<xs:element name="Topic" type="Topic"/>
				<xs:element name="Comment" type="Comment"
minOccurs="0" maxOccurs="unbounded"/>
				<!-- ISG Jira issue BCF-9. Add support for
several viewpoints and snapshots per issue -->
				<xs:element name="Viewpoints" type="ViewPoint"
minOccurs="0" maxOccurs="unbounded"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:complexType name="Header">
		<xs:sequence>
			<xs:element name="File" maxOccurs="unbounded">
				<xs:complexType>
					<xs:sequence>
						<xs:element name="Filename"
type="xs:string" minOccurs="0"/>
						<xs:element name="Date"
type="xs:dateTime" minOccurs="0"/>
						<!-- Reference (URL) of the file -
->
						<xs:element name="Reference"
type="xs:string" minOccurs="0"/>
					</xs:sequence>
					<xs:attributeGroup
ref="FileAttributes"/>
				</xs:complexType>
			</xs:element>
		</xs:sequence>
	</xs:complexType>
	<!-- ISG Jira issue BCF-9. Add support for several viewpoints
and snapshots per issue -->
	<xs:complexType name="ViewPoint">
		<xs:sequence>
			<!-- viewpoint file (xml) -->
			<xs:element name="Viewpoint" type="xs:string"
minOccurs="0"/>
			<!-- the snapshot png -->
			<xs:element name="Snapshot" type="xs:string"
minOccurs="0"/>
			<!-- the viewpoint index (sort order) -->
			<xs:element name="Index" type="xs:int"
minOccurs="0"/>
		</xs:sequence>
		<xs:attribute name="Guid" type="Guid" use="required"/>
		<!-- Guid of the viewpoint -->
	</xs:complexType>
	<!-- BimSnippet -->
	<xs:complexType name="BimSnippet">
```

```
            <xs:sequence>
                  <!--
        Name of the file in the topic folder containing the snippet or
a URL.
         E.G.- Expresscode containing p.e Issue, Request
        // Maybe some header infos ?? // IfcEntites // Geometry
         -->
                  <!-- Reference (name) to the snippet file -->
                  <xs:element name="Reference" type="xs:string"/>
                  <xs:element name="ReferenceSchema"
type="xs:string"/>
            </xs:sequence>
            <xs:attribute name="SnippetType" type="xs:string"
use="required"/>
            <xs:attribute name="isExternal" type="xs:boolean"
default="false"/>
            <!-- This flag is true when the reference is a URL
pointing outside of the BCF file-->
      </xs:complexType>
      <xs:complexType name="Topic">
            <xs:sequence>
                  <xs:element name="ReferenceLink" type="xs:string"
minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element name="Title" type="xs:string"/>
                  <xs:element name="Priority" type="Priority"
minOccurs="0"/>
                  <!-- ISG Jira issue BCF-8 Add a way save order the
topics -->
                  <xs:element name="Index" type="xs:int"
minOccurs="0"/>
                  <xs:element name="Labels" type="TopicLabel"
minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element name="CreationDate" type="xs:dateTime"
minOccurs="1"/>
                  <xs:element name="CreationAuthor" type="UserIdType"
minOccurs="1"/>
                  <xs:element name="ModifiedDate" type="xs:dateTime"
minOccurs="0"/>
                  <xs:element name="ModifiedAuthor" type="UserIdType"
minOccurs="0"/>
                  <xs:element name="DueDate" type="xs:dateTime"
minOccurs="0"/>
                  <xs:element name="AssignedTo" type="UserIdType"
minOccurs="0"/>
                  <xs:element name="Stage" type="Stage"
minOccurs="0"/>
                  <xs:element name="Description" type="xs:string"
minOccurs="0"/>
                  <xs:element name="BimSnippet" type="BimSnippet"
minOccurs="0"/>
                  <!-- Name of the file in the topic folder or url -->
                  <xs:element name="DocumentReference" minOccurs="0"
maxOccurs="unbounded">
                        <xs:complexType>
                              <xs:sequence>
                                    <!-- Name of the file in the topic
folder or url -->
                                    <xs:element
name="ReferencedDocument" type="xs:string" minOccurs="0"/>
                                    <!-- Human readable name of the
```

```
document -->
                                    <xs:element name="Description"
type="xs:string" minOccurs="0"/>
                              </xs:sequence>
                              <xs:attributeGroup
ref="DocumentReference"/>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="RelatedTopic" minOccurs="0"
maxOccurs="unbounded">
                        <xs:complexType>
                              <xs:attribute name="Guid" type="Guid"
use="required"/>
                        </xs:complexType>
                  </xs:element>
            </xs:sequence>
            <xs:attribute name="Guid" type="Guid" use="required"/>
            <xs:attribute name="TopicType" type="TopicType"/>
            <xs:attribute name="TopicStatus" type="TopicStatus"/>
      </xs:complexType>
      <!-- Reference to a document inside of the topic folder or a url
pointing to the web -->
      <xs:attributeGroup name="DocumentReference">
            <!-- Guid of the DocumentReference -->
            <xs:attribute name="Guid" type="Guid"/>
            <!-- A flag that is true when the ReferencedDocument
points outside of the BCF file (a URL) -->
            <xs:attribute name="isExternal" type="xs:boolean"
default="false"/>
      </xs:attributeGroup>
      <xs:complexType name="Comment">
            <xs:sequence>
                  <xs:element name="Date" type="xs:dateTime"/>
                  <xs:element name="Author" type="UserIdType"/>
                  <xs:element name="Comment" type="xs:string"/>
                  <xs:element name="Viewpoint" minOccurs="0">
                        <xs:complexType>
                              <xs:attribute name="Guid" type="Guid"
use="required"/>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="ModifiedDate" type="xs:dateTime"
minOccurs="0"/>
                  <xs:element name="ModifiedAuthor" type="UserIdType"
minOccurs="0"/>
            </xs:sequence>
            <xs:attribute name="Guid" type="Guid" use="required"/>
      </xs:complexType>
      <xs:simpleType name="TopicStatus">
            <xs:restriction base="xs:string"/>
      </xs:simpleType>
      <xs:simpleType name="TopicType">
            <xs:restriction base="xs:string"/>
      </xs:simpleType>
      <xs:simpleType name="TopicLabel">
            <xs:restriction base="xs:string"/>
      </xs:simpleType>
      <xs:simpleType name="Priority">
            <xs:restriction base="xs:string"/>
      </xs:simpleType>
```

```
	<xs:simpleType name="UserIdType">
		<xs:restriction base="xs:string"/>
	</xs:simpleType>
	<xs:simpleType name="Stage">
		<xs:restriction base="xs:string"/>
	</xs:simpleType>
	<xs:simpleType name="Guid">
		<xs:restriction base="xs:string">
			<xs:pattern value="[a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-
fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{12}"/>
		</xs:restriction>
	</xs:simpleType>
	<xs:simpleType name="IfcGuid">
		<xs:restriction base="xs:string">
			<xs:length value="22"/>
			<xs:pattern value="[0-9,A-Z,a-z,_$]*"/>
		</xs:restriction>
	</xs:simpleType>
	<xs:attributeGroup name="FileAttributes">
		<xs:attribute name="IfcProject" type="IfcGuid"/>
		<xs:attribute name="IfcSpatialStructureElement"
type="IfcGuid"/>
		<xs:attribute name="isExternal" type="xs:boolean"
default="true"/>
	</xs:attributeGroup>
</xs:schema>
```

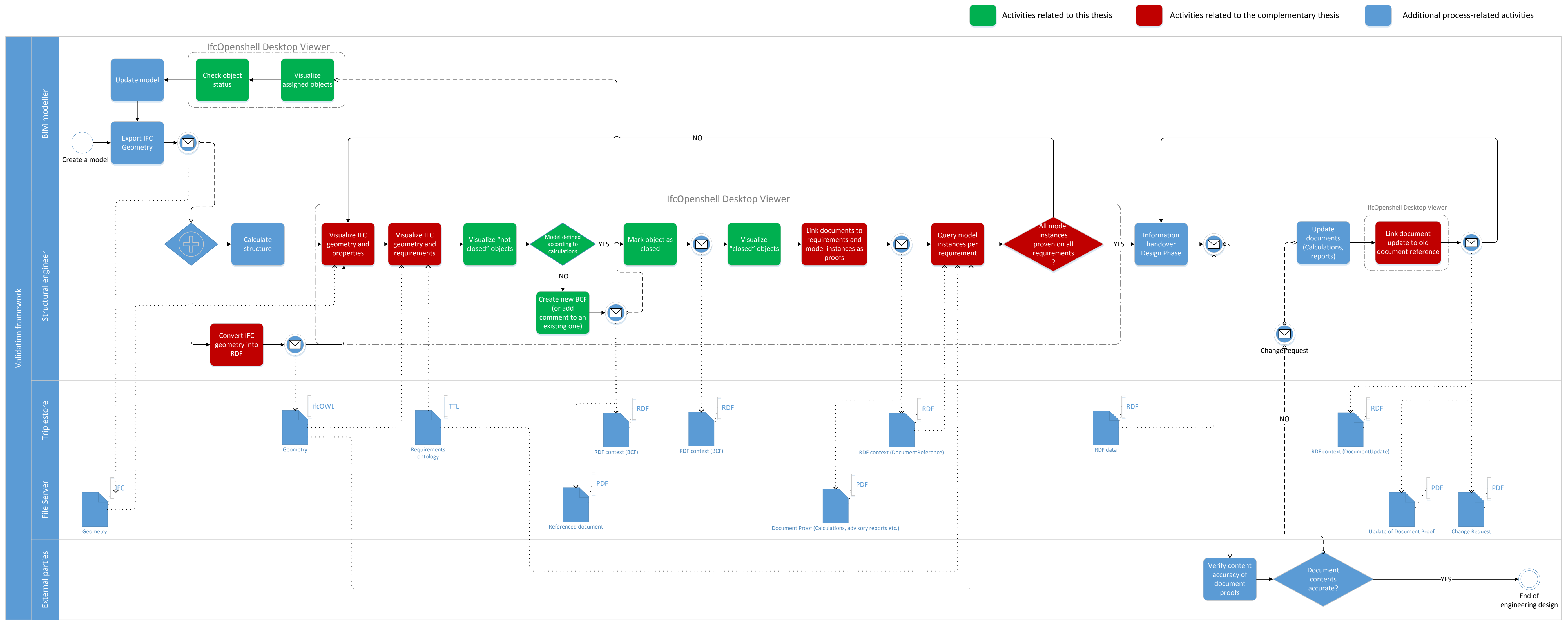
7.8 Appendix VIII: Flowcharts of the developed tool



Prototype's workflow example
Structural engineer
BIM modeller
Triple store
Workflow example 1
Start of the validation
Load IFC model
Revise building objects' requirements
Building object complies with requirements?
YES
NO
Mark building object as closed
Create BCF issue
Workflow example 2
Load IFC model
Visualize BCF issues assigned to the user
Visualize building objects' BCF status and comments
Update IFC model (not IfcOpenShell)
Add BCF comment for resolved issues
Workflow example 3
Load IFC model
Visualize open building objects
Visualize building objects' BCF comments
Revise building objects' requirements
Building object complies with requirements?
NO
YES
Add BCF comment
Mark building object as closed
End of the validation
RDF
RDF context (BCF)
RDF
RDF context (BCF)
RDF
Comment added to existing RDF context
RDF
Comment added to existing RDF context
RDF
RDF context (BCF)

Figure 70. Workflow examples

7.9 Appendix IX: Thesis’ scope



🟩 Activities related to this thesis
🟥 Activities related to the complementary thesis
🟦 Additional process-related activities
Validation framework
BIM modeller
Structural engineer
Triplestore
File Server
External parties
IfcOpenshell Desktop Viewer
Update model
Check object status
Visualize assigned objects
Create a model
Export IFC Geometry
NO
Calculate structure
Visualize IFC geometry and properties
Visualize IFC geometry and requirements
Visualize “not closed” objects
Model defined according to calculations
YES
NO
Mark object as closed
Visualize “closed” objects
Link documents to requirements and model instances as proofs
Query model instances per requirement
All model instances proven on all requirements ?
YES
Information handover Design Phase
Create new BCF (or add comment to an existing one)
Convert IFC geometry into RDF
Update documents (Calculations, reports)
IfcOpenshell Desktop Viewer
Link document update to old document reference
Change request
NO
ifcOWL
Geometry
TTL
Requirements ontology
RDF
RDF context (BCF)
RDF
RDF context (BCF)
RDF
RDF context (DocumentReference)
RDF
RDF data
RDF
RDF context (DocumentUpdate)
IFC
Geometry
PDF
Referenced document
PDF
Document Proof (Calculations, advisory reports etc.)
PDF
Update of Document Proof
PDF
Change Request
Verify content accuracy of document proofs
Document contents accurate?
YES
End of engineering design

Figure 71. Thesis’ scope