

Master Thesis

# Implementation of Information Container for Data Drop - Exchange of Linked Building Models

A thesis submitted in fulfillment of the requirements for the degree  
Master of Science in Structural Engineering

Author: Philipp Hagedorn

First Supervisor: Prof. Dr.-Ing. Volker Berkhahn

Second Supervisor: Prof. Dr.-Ing. Markus König

Submission date: 31<sup>st</sup> August 2018

# Declaration of Authorship

I, Philipp Hagedorn, declare that this thesis with the title “Implementation of Information Container for Data Drop - Exchange of Linked Building Models” and the work presented in it are my own and have been generated by me as the result of my own original research. I confirm that this work was wholly done by myself and where I have consulted the work of others, this is always clearly attributed. This thesis has not been submitted for a degree or any other qualification at this University or any other institution previously.

Hannover, 31<sup>st</sup> August 2018

# Abstract

During the last years, the Architecture, Engineering and Construction (AEC) industry intensified the utilization of Building Information Modeling (BIM) for the complete building lifecycle. Different planning participants equip multi-dimensional building models with information to achieve exact delivery dates, detailed cost estimation, resilient quality management, and a sustainable facility management. This heterogeneous information can be linked to geometric building models to support the planning process and the asset management over the building lifecycle. However, these connections are mainly established using software tools that use proprietary data formats, leading to a lack of interoperability in information exchange. To regain interoperability, an international standard has been drafted under the ISO 21597 “Information Container for Data Drop”. The question behind this thesis is how linked building models can be exchanged using the standard and how this standard can be implemented into a validation framework.

Therefore, a literature review on the exchange of linked building models and state-of-the-art Linked Open Data (LOD) approaches is performed as a basis for the analysis of the ISO standard. The results of the analysis are discussed and compared to the Multi-Model-Container. This thesis introduces an implementation for the import and validation of the standardized file format, the visualization of the contained documents, the manipulation of links inside the container, and a loss-free export into the data format. The application was evaluated with an exemplary BIM-LV-Container. The outcome of the thesis is a suitable framework for users that need to validate their files. It can be seen as a guideline for developers that have to implement import and export methods of information containers into their individual software solution. The practical application of both the standard and the framework has to be assessed in future work.

# Acknowledgements

I would first like to thank my thesis advisors Prof. Volker Berkhahn of the Institute of Risk and Reliability at Leibniz Universität Hannover and Prof. Markus König of the Chair for Computing in Engineering at Ruhr-Universität Bochum for their helpful advice during the writing process. Especially Prof. Berkhahn had always an open door for me during my whole time at university.

Furthermore, I would like to thank all those whose freely available applications and open source libraries I have used in the context of this work.

Finally, I wish to express my profound gratitude to my family and my girlfriend for their support and encouragement throughout my years of study and during the process of writing this thesis.

# Contents

|   |           |
|---|-----------|
| <b>List of Figures</b>  | <b>II</b> |
| <b>List of Tables</b>   | <b>IV</b> |
| <b>List of Listings</b>   | <b>IV</b> |
| <b>List of Abbreviations</b>  | <b>V</b>  |
| <b>1. Introduction</b>  | <b>1</b>  |
| 1.1. Motivation . . . . .   | 1         |
| 1.2. Structure . . . . .  | 3         |
| <b>2. Related Work</b>  | <b>4</b>  |
| 2.1. Linked Open Data . . . . .   | 4         |
| 2.1.1. Resource Description Framework . . . . .                             | 4         |
| 2.1.2. Web Ontology Language . . . . .                                      | 5         |
| 2.1.3. Ontology Modeling Example . . . . .                                  | 6         |
| 2.1.4. Querying information with SPARQL . . . . .                           | 7         |
| 2.2. Linked Building Models . . . . .                                       | 8         |
| 2.3. Linked Open Data Approaches for AEC . . . . .                          | 9         |
| 2.4. Conclusions . . . . .  | 10        |
| <b>3. Exchange of Linked Building Models</b>                                | <b>11</b> |
| 3.1. Multi-Model Container and BIM-LV-Container . . . . .                   | 11        |
| 3.2. Information Container for Data Drop . . . . .                          | 14        |
| 3.2.1. Part 1: Container . . . . .  | 14        |
| 3.2.2. Part 2: Dynamic Semantics . . . . .                                  | 16        |
| 3.2.3. Integration of DynamicSemantics with Container and Linkset . . . . . | 18        |
| 3.3. Comparison and Discussion . . . . .                                    | 19        |
| <b>4. Requirement Engineering</b>   | <b>21</b> |
| 4.1. Specification of Functional Requirements . . . . .                     | 22        |
| 4.2. Specification of Non-Functional Requirements . . . . .                 | 24        |
| 4.3. Requirement Management . . . . .                                       | 25        |
| <b>5. Software Implementation</b>   | <b>26</b> |
| 5.1. Basic Framework Design . . . . .                                       | 26        |
| 5.1.1. Web application and interface design . . . . .                       | 26        |
| 5.1.2. RDF Library . . . . .  | 27        |
| 5.1.3. IFC Library . . . . .  | 28        |
| 5.2. Import and Export Components . . . . .                                 | 29        |
| 5.2.1. Data Handling . . . . .  | 29        |
| 5.2.2. Session Handling . . . . .   | 31        |
| 5.2.3. Code Generation . . . . .  | 31        |
| 5.2.4. Web API . . . . .  | 32        |

|           |  |           |
|-----------|--|-----------|
| 5.2.5.    | Evaluation . . . . .                                 | 33        |
| 5.3.      | Validation Components . . . . .                      | 34        |
| 5.3.1.    | Conformance Validation . . . . .                     | 34        |
| 5.3.2.    | Consistency Validation . . . . .                     | 34        |
| 5.3.3.    | Evaluation . . . . .                                 | 35        |
| 5.4.      | Visualization Components . . . . .                   | 36        |
| 5.4.1.    | Concept and Implementation . . . . .                 | 36        |
| 5.4.2.    | Link Maintenance . . . . .                           | 37        |
| 5.4.3.    | Evaluation . . . . .                                 | 38        |
| <b>6.</b> | <b>Evaluation</b>                                    | <b>39</b> |
| 6.1.      | Setup . . . . .                                      | 39        |
| 6.2.      | Results . . . . .                                    | 39        |
| 6.2.1.    | Input and Validation . . . . .                       | 40        |
| 6.2.2.    | Visualization . . . . .                              | 41        |
| 6.2.3.    | Links . . . . .                                      | 43        |
| 6.2.4.    | Metadata . . . . .                                   | 44        |
| 6.2.5.    | Web API . . . . .                                    | 44        |
| 6.2.6.    | Quality . . . . .                                    | 44        |
| 6.2.7.    | Performance . . . . .                                | 45        |
| 6.3.      | Discussion . . . . .                                 | 45        |
| <b>7.</b> | <b>Conclusion and Outlook</b>                        | <b>47</b> |
|           | <b>References</b>                                    | <b>49</b> |
| <b>A.</b> | <b>Appendices</b>                                    | <b>53</b> |
| A.1.      | Codemap for ContainerModel Namespace . . . . .       | 54        |
| A.2.      | Method Documentation: SetInstanceAttribute . . . . . | 55        |
| A.3.      | Method Documentation: StartSession . . . . .         | 56        |

# List of Figures

|  |    |
|--|----|
| 1.1. Multi-dimensional building model integration . . . . .                                  | 2  |
| 2.1. RDF triple definition . . . . .   | 5  |
| 2.2. Example ontology: "BuildingOntology" . . . . .  | 5  |
| 2.3. Integration of Linked Building Models . . . . .   | 8  |
| 3.1. UML diagram of the Generic Multi-Model (see Fuchs et al. (2011)) . . . . .              | 11 |
| 3.2. Generic Multi-Model Container (see Demharter et al. (2014)) . . . . .                   | 12 |
| 3.3. File content of a BIM-LV-Container . . . . .  | 13 |
| 3.4. Structure of the ICDD (see ISO 21597-1) . . . . .                                       | 14 |
| 3.5. UML diagram of the Link inheritance (see ISO 21597-1) . . . . .                         | 15 |
| 3.6. UML diagram of the AbstractProperty inheritance (see ISO 21597-2) . . . . .             | 16 |
| 3.7. UML diagram of the ContainsRelation inheritance (see ISO 21597-2) . . . . .             | 17 |
| 3.8. Integration of DynamicSemantics with Container and Linkset (see ISO 21597-1) . . . . .  | 18 |
| 4.1. Process model for software engineering according to Royce (1987) . . . . .              | 21 |
| 4.2. Product backlog inside Team Foundation Server . . . . .                                 | 25 |
| 4.3. Requirement management inside Team Foundation Server . . . . .                          | 25 |
| 5.1. Design of the Web API . . . . .   | 26 |
| 5.2. Efficiency of RDF libraries . . . . .   | 27 |
| 5.3. Outsourcing the geometry conversion into separate worker instance . . . . .             | 29 |
| 5.4. Integrating RDF and object-oriented programming as stated by Völkel (2005) . . . . .    | 29 |
| 5.5. Desktop application for Web API evaluation . . . . .                                    | 33 |
| 5.6. Mixed origins of identifiers . . . . .  | 36 |
| 5.7. Design of the Graphical User Interface . . . . .  | 37 |
| 5.8. Link Visualization and Manipulation . . . . .   | 38 |
| 6.1. Upload of a container file . . . . .  | 40 |
| 6.2. Validation of the exemplary container file . . . . .                                    | 40 |
| 6.3. Exploring the validated container file example . . . . .                                | 41 |
| 6.4. Exploring the IFC model from the container example . . . . .                            | 42 |
| 6.5. Adding a new link to the 4D-Links . . . . .   | 43 |
| 6.6. Efficiency of the developed web application according to performed operations . . . . . | 45 |
| 6.7. Optimization of the relation between Identifier, Link and Document . . . . .            | 46 |

## List of Tables

|   |    |
|---|----|
| 2.1. Extracted RDF triples . . . . .                            | 7  |
| 3.1. Comparison of ISO 21597 and DIN SPEC 91350 / MMC . . . . . | 19 |
| 4.1. Definition of user stories . . . . .                       | 22 |
| 5.1. Excerpt from the Web API definition . . . . .              | 32 |

## List of Listings

|   |    |
|---|----|
| 2.1. RDF file header . . . . .                | 6  |
| 2.2. OWL class declaration . . . . .          | 6  |
| 2.3. OWL properties and individuals . . . . . | 6  |
| 2.4. SPARQL query example . . . . .           | 7  |
| 3.1. MMC file header . . . . .                | 12 |



# List of Abbreviations

|                 |  |
|-----------------|--|
| <b>3D model</b> | Geometric representation of a building model             |
| <b>4D model</b> | Geometric representation + scheduling information        |
| <b>5D model</b> | Geometric representation + scheduling + cost information |
| <b>AEC</b>      | Architecture, Engineering and Construction               |
| <b>API</b>      | Application Programming Interface                        |
| <b>BIM</b>      | Building Information Modeling                            |
| <b>BLC</b>      | BIM-LV-Container   |
| <b>BOQ</b>      | Bill of Quantities                                       |
| <b>BOT</b>      | Building Topology Ontology                               |
| <b>CSV</b>      | Comma Separated Values                                   |
| <b>DIN</b>      | German Institute for Standardization                     |
| <b>GAEB</b>     | German Joint Committee for Electronics in Construction   |
| <b>GAEB DA</b>  | GAEB Data exchange and structure of Bill of Quantities   |
| <b>GUI</b>      | Graphical User Interface                                 |
| <b>GUID</b>     | Globally Unique Identifier                               |
| <b>HTTP</b>     | Hypertext Transfer Protocol                              |
| <b>HVAC</b>     | Heating, Ventilation, Air Condition                      |
| <b>ICDD</b>     | Information Container for Data Drop                      |
| <b>JSON</b>     | JavaScript Object Notation                               |
| <b>ID</b>       | Identifier   |
| <b>IFC</b>      | Industry Foundation Classes                              |
| <b>ISO</b>      | International Organization for Standardization           |

*List of Abbreviations*

|               |  |
|---------------|--|
| <b>LBD</b>    | Linked Building Data                   |
| <b>LOD</b>    | Linked Open Data                       |
| <b>LV</b>     | German: Leistungsverzeichnis, see BOQ  |
| <b>MMC</b>    | Multi-Model-Container                  |
| <b>MVC</b>    | Model-View-Controller                  |
| <b>OOP</b>    | Object-Oriented Programming            |
| <b>OWL</b>    | Web Ontology Language                  |
| <b>REST</b>   | Representational State Transfer        |
| <b>RDF</b>    | Resource Description Framework         |
| <b>RDFS</b>   | Resource Description Framework Schema  |
| <b>SPARQL</b> | SPARQL Protocol And RDF Query Language |
| <b>UDO</b>    | User Defined Ontology                  |
| <b>URI</b>    | Uniform Resource Identifier            |
| <b>W3C</b>    | World Wide Web Consortium              |
| <b>XML</b>    | Extensible Markup Language             |
| <b>XSD</b>    | XML Schema Definition                  |

# 1. Introduction

## 1.1. Motivation

The emergence of Building Information Modeling (BIM) during the last years leads to an enormous range of capabilities for Architecture, Engineering and Construction (AEC) industry utilizing building models for the complete planning and construction process (Eastman et al., 2011). As large recent projects show, the construction lifecycle is an enormous challenge due to their complexity for both technical and project management tasks. Besides cost, time and quality aims, there are a lot of factors that contribute to the success of a construction project. The complexity of planning and realizing buildings can be countered using digital methods. With the increasing demand for delivery dates, detailed cost estimation and sustainable quality management during the planning process, building models are increasingly enriched with information from different domains and heterogeneous sources (Borrmann et al., 2015).

Due to the unique characteristics of each construction project, varying project partners create information with different functional and technical characteristics and inject it into the overall planning process. The amount of different software applications for the specific domains also provides a large number of file formats. Additionally, not all data is available digitally. The entirety of information needs to be structured and semantically stored to be accessible for every project participant. However, this will lead to large-scaled, complex data schemes that cannot serve the information of every participant during the planning process. For instance, cost parameters can be attached to geometric building objects, but the Bill of Materials (BOM) provides much more information that cannot be stored within an Industry Foundation Classes (IFC) model appropriately. Nonetheless, BIM as a method is ideally suited for the complete lifecycle of a building using digital geometric and information models. Although, the pervasive use of models and the digital cooperation between involved persons necessitate a process-oriented optimization of information exchange (Borrmann et al., 2015).

Storing all relevant information in a common building model does not increase productivity and therefore has little relevance for practical application (Fuchs, 2015). The reasons for this are diverse and rely in both the technical and organizational areas. From the technical standpoint, the current software landscape does not support a cross-domain, fully interoperable BIM collaboration process, because no application can handle the aspects of all domains (Fuchs, 2015). Equally important are the organizational aspects, which say that BIM is more than just the geometric modeling of a building, but the adaptation of the working methods of everyone involved in the planning (Eastman et al., 2011). A common building model needs to be maintained by a single organization. In contrast, the reality in Germany is that responsibilities are generally considered finished when a result is handed over (Fuchs, 2015). Furthermore, project partners deny to integrate their complete information into a model as it may be required. These factors make it difficult to manage and exchange a common building model.

There are a lot of standardized formats and processes that do not fit into a joint building model since they have established themselves in the construction industry. One example is the

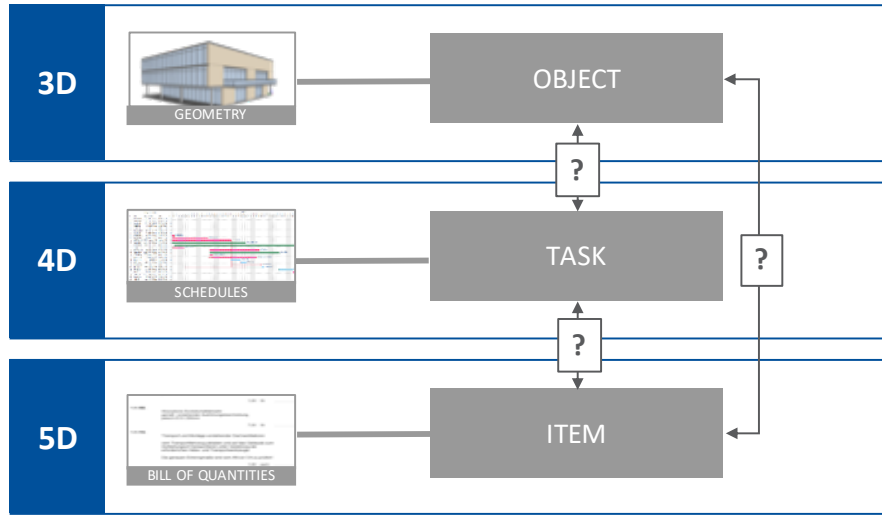


Figure 1.1.: Multi-dimensional building model integration

cost estimation and the creation of a BOQ, which in Germany follows a standardized schema according to DIN 267 and VOB. The raised data can be digitally exchanged between project participants via the German Joint Committee for Electronics in Construction (GAEB) interface definition (Schiller and Faschingbauer, 2016). Therefore, it is clear that the information cannot be included in a single building model, but need to be divided into domain models and related independent information models. This procedure is often referred to in science and practice as nD-modeling (Ding et al., 2014). To perform a BIM-based cost estimation, three-dimensional geometric models are connected to cost information models, e.g. a BOQ (Hanff and Wörter, 2015). In the case of a so called 5D model, every building object is linked to an item from the BOQ (see fig. 1.1). To maintain the functionality of the BOQ and not inflate the building model, it makes sense to leave both models in their original file format. The question is what the link between the object and the item must look like in order to be universally formulated and transferable. The connection between geometric objects and information can be established with several software tools. But in the most cases, the exchange of the linked model is done with proprietary data formats.

Since the proprietary data formats are rarely interoperable, they interfere with the openBIM approaches that are mostly gained through the IFC data format (Du Juan and Zheng, 2014). Proprietary data formats pose a lot of challenges to collaborative working methods because changes of original models could cause inconsistencies due to unmanageable links between models. The same issue concerns domain specific software that is not able to read or manage links because the formats are not legible. When saving a proprietary format, the changes to a certain object could be lost due to missing interoperability. To retrieve the openBIM principle and restore interoperability, the term of information container has been formed within German and International Standards. The aim is to provide interoperable data inside a container while keeping the original data available for domain specific application. This container requires a system with a high generality to achieve a cross-domain data exchange format.

In the context of an information container, especially the link between the BOQ and the building model became relevant with the implementation of BIM-LV-Container (BLC) in a national standard in Germany (DIN SPEC 91350, 2016), which has been originally researched as the Multi-Model-Container. To achieve more than a single case for cost estimation and provide a cross-AEC approach, an international standard has been drafted under the ISO 21597 “Information Container for Data Drop”. This information container provides heterogeneous data from any domain in any format and allows a complex metadata structure to organize

models and manage links in a standardized format. This leads to the fact that the users can access the semantic data inside the container without using proprietary data formats. The standard has been the motivation for this thesis in order to improve the exchange of linked building models. It delivers a file format for the transfer of building lifecycle information in a single data drop.

The motivation for this thesis is the systematic presentation of a relevant topic for research and practice. With a view to the future publication of the standard, there is little published information and no known implementation of the Information Container for Data Drop (ICDD). This thesis with its implementation can serve as a reference for future implementations and as a guide for other developers that have to deal with the implementation of the standard. The developed toolkit can be used in further projects as it offers a Web API for operations on the ICDD file format. In procedural terms, the toolkit can be embedded into a global BIM workflow so that all participants can exchange their file stock on a project in a single data drop. The use of ICDD in the planning workflow allows a semantic archival of the file stock which results in a higher quality assurance of the planning at any data drop. The container allows to integrate different planning stages and alternatives of participants from several domains which brings advantages in transparency compared to the conventional planning processes.

## 1.2. Structure

This thesis is structured as follows: After this introduction, the thesis begins with an overview of the state-of-the-art in the exchange of Linked Building Data (LBD) and the related work in science. The principles of Linked Open Data are defined and explained and AEC specific use cases are examined. The scientific background is enriched by the introduction of Linked Building Models. This background knowledge is transferred to the analysis in the third chapter. The concepts of the Multi-Model-Container, the BIM-LV-Container, and the Information Container for Data Drop are analyzed, compared and discussed. This analysis forms the basis for the specification of requirements in the fourth chapter and enables understanding of the implementation.

The fourth chapter gives an introduction to software development workflows and requirement engineering. In addition, it summarizes the requirements of the software application that is developed within the scope of this thesis. Therefore, a distinction has been made between functional and non-functional requirements. Possible use cases for a web-based allocation of the application are presented. After this, the design and implementation stages are documented and explained in chapter five. The implementation stage is accompanied by a parallel evaluation as well as an overall evaluation in chapter six at the end. Finally, the outcomes of the thesis are summarized and an outlook on future fields of research and implementation is given.

## 2. Related Work

The following sections serve as background for the analysis of the ISO standard and offer a background on linked building models as well as general and AEC-specific linked data approaches.

### 2.1. Linked Open Data

Linked Open Data (LOD) is a concept for information management along the World Wide Web. It has been introduced by Berners-Lee (2006) as a part of the Semantic Web. The original idea of Berners-Lee was to use Uniform Resource Identifier (URI) as the unique identifier for the data to provide a consistent accessor. The URIs can directly be addressed via the Hypertext Transfer Protocol (HTTP) protocol and referenced in other data resources. Four basic principles for sharing and connecting data were defined by Berners-Lee (2006):

1. "Use URI as names for things"
2. "Use HTTP URIs so that people can look up those names"
3. "When someone looks up a URI, provide useful information using the standards"
4. "Including links to other relevant URIs so that people can discover more things"

Concludingly, the concept of LOD delivers data from different sources linked in a data resource network on the web covering more than 50 billion data entries by 2012 (Bauer and Kaltenböck, 2012). Being state-of-the-art in information management, popular examples for Linked Open Data applications can be found in libraries such as the LOD service of the German National Library (Hannemann and Kett, 2010). The benefits for interoperability in information exchange were stated by Curry et al. (2013) as the independent design of systems, the incremental interoperability and the connection of heterogeneous data. With these benefits, the availability and usability of data can be increased significantly.

#### 2.1.1. Resource Description Framework

Technically, the ontology for linked data can be developed using the Resource Description Framework (RDF). The RDF is a framework for information representation in the web recommended by Cyganiak et al. (2014) and the World Wide Web Consortium (W3C). The RDF is a non-building specific modeling format for ontologies which creates interoperable data for both human perception and machine processing (Abanda et al., 2013). An RDF file consists of a set of semantic statements based on the directed graph theory. Each statement is represented by a triple of elementary atomic statements using a subject, a predicate and an object (see

fig. 2.1). Generally, objects are resources that are related to each other through the predicate. The first subject and the predicate usually are resources defined by a URI. The second object can either be a URI resource or an atomic literal. The summary of triples can be represented in an RDF graph. A complete documentation of the RDF concept can be found at Cyganiak et al. (2014).

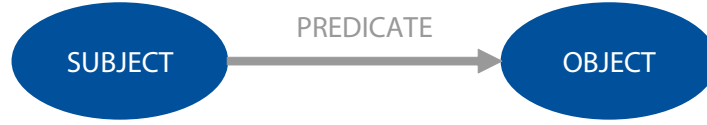


Figure 2.1.: RDF triple definition

Moreover, the RDF data type is based on the Extensible Markup Language (XML) format. While the RDF file is the common syntax for exchange, the interpretation of the RDF data needs a separate ontology definition called vocabulary. Generally speaking, an ontology in the meaning of building models is the collection of relations between objects or resources in a single model (Beetz, 2015). Ontologies offer consistent classifications, relations and properties of objects. The Resource Description Framework Schema (RDFS) can be used as an ontology language for defining and documenting RDF vocabularies (Cyganiak et al., 2014).

### 2.1.2. Web Ontology Language

Within the semantic web, the Web Ontology Language (OWL) has attracted significant interest due to the limited expressivity of the RDFS (Antoniou and van Harmelen, 2009). OWL is a descriptive language to publish and exchange ontologies within the syntax of RDF and mainly extends the expressivity of RDF with vocabularies for describing classes. It is available in the second standardized version OWL2 introduced by Motik et al. (2012). The definition of ontologies occurs in the RDF file using `<owl>`, `<rdf>` and `<rdfs>` tags. These tags can be used to create entities such as classes, properties and individuals. Individuals are instances of a class, which can hold properties in different data types. Therefore, common data types from the XML schema are available and can be extended. A complete documentation of the OWL concept can be found at Motik et al. (2012). Within this thesis, the Protégé ontology editor is used to view and edit OWL ontologies manually which has been developed according to W3C recommendations by a research group at Stanford University (Musen, 2015). Furthermore, the basic principles of the ontology modeling using OWL will be described in the next section with an example.

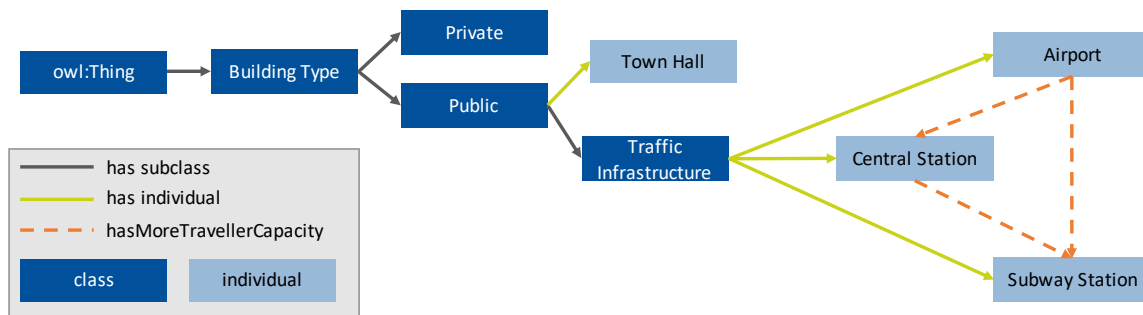


Figure 2.2.: Example ontology: "BuildingOntology"

### 2.1.3. Ontology Modeling Example

The graph in fig. 2.2 shows a building ontology with classes, individuals and three types of edges which can be transferred into an OWL/RDF ontology. In a first step, the namespaces within the file header need to be defined as seen in Listing 2.1 with an `<rdf>` tag. The RDF, OWL and RDFS namespaces are defined with the URI for the respective syntax resource to offer the provided functionalities. The URI of the “buildings” ontology in line 4 is set to a fictive resource in this example. The `#` symbol is fixed as a delimiter within the URI.

Listing 2.1: RDF file header

```

1 <?xml version="1.0"?>
2 <rdf:RDF
3     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4     xmlns:owl="http://www.w3.org/2002/07/owl#"
5     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
6     xmlns:buildings="http://www.example.org/buildings#">
7     <!-- Ontology Description inside -->
8 </rdf>
```

Secondly, the classes structure needs to be created. While the `owl:Thing` class is always the root of the ontology, an user-defined entity of the type class can be declared with the `<owl:class>` tag as seen in Listing 2.2. The class declaration can contain the resource as an URI attached to the `rdf:about` attribute or an `rdf:ID` attribute depending on the current use-case. Within the class declaration, annotations like a label or comment can be attached. This example has a given class hierarchy which can be realized through the `<rdfs:subClassOf/>` tag using either the resource or the ID to create the relation.

Listing 2.2: OWL class declaration

```

1 <owl:Class rdf:about="http://www.example.org/buildings#publicBuilding#
   trafficInfrastructure">
2     <rdfs:subClassOf rdf:resource="http://www.example.org/buildings#
   publicBuilding"/>
3     <rdfs:label>Traffic Infrastructure</rdfs:label>
4 </owl:Class>
```

Furthermore, classes, as well as properties, can be derived from each other with the `rdfs:subClassOf` or `rdfs:subPropertyOf` assignment to obtain a hierarchy of classes or properties. The definition of a property can be found in Listing 2.3, where the object property called `hasMoreTravellerCapacity` is introduced in line 1. Generally, properties are distinguished by object properties and data type properties such as literals or integer. Several additional characteristics like relations and constraints can be attached to properties.

Listing 2.3: OWL properties and individuals

```

1 <owl:ObjectProperty rdf:about="http://www.example.org/buildings#
   hasMoreTravellerCapacity"/>
2 <rdf:Description rdf:about="http://www.example.org/buildings#Airport">
3     <rdf:type rdf:resource="http://www.example.org/buildings#
   publicBuilding#trafficInfrastructure"/>
4     <rdfs:label>Airport</rdfs:label>
5     <buildings:hasMoreTravellerCapacity rdf:resource="http://www.example.
   org/buildings#CentralStation"/>
6 </rdf:Description>
```



To instantiate a class, the `<rdf:Description>` tag needs to be called. Within this tag the individual is characterized by the class type and properties. The defined object property can be assigned to the individual as seen in lines 6 and 7. Therefore, the user-defined buildings namespace is called with the respective property and the corresponding object. This expression again shows the “Subject-Predicate-Object” structure of the RDF file. For instance, the triples from the code extracts can be summarized as follows:

Table 2.1.: Extracted RDF triples

|   |   |  |
|---|---|--|
| <code>...#trafficInfrastructure</code>    | <code>22-rdf-syntax-ns#type</code>        | <code>owl#Class</code>                 |
| <code>...#trafficInfrastructure</code>    | <code>rdf-schema#label</code>             | <code>“Traffic Infrastructure”</code>  |
| <code>...#trafficInfrastructure</code>    | <code>rdf-schema#subClassOf</code>        | <code>...#buildingType</code>          |
| <code>...#hasMoreTravellerCapacity</code> | <code>22-rdf-syntax-ns#type</code>        | <code>owl#ObjectProperty</code>        |
| <code>...#Airport</code>                  | <code>rdf-schema#label</code>             | <code>“Airport”</code>                 |
| <code>...#Airport</code>                  | <code>22-rdf-syntax-ns#type</code>        | <code>...#trafficInfrastructure</code> |
| <code>...#Airport</code>                  | <code>...#hasMoreTravellerCapacity</code> | <code>...#CentralStation</code>        |

Most of the predicates that concern the general structure come with the RDFS or OWL. Hence, there are some triples that are user-defined and describe individual parts of the ontology. While this example features a small insight into OWL modeling, the concept of modeling ontologies with OWL is valuable and extendable to certain situations.

#### 2.1.4. Querying information with SPARQL

SPARQL Protocol And RDF Query Language (SPARQL) is a graph based query language for RDF defined in a recommendation of W3C (Harris and Seaborne, 2013). It can be used to query triples from RDF files using a specified query syntax. The results can be delivered in XML, JSON or CSV format. Main component of SPARQL is a query pattern in the Turtle-Syntax<sup>1</sup> using the `WHERE` statement. The query pattern may contain variables declared with `?` or `$`. A query pattern can be executed using the `SELECT` keyword. URIs can be abbreviated with the definition of a `PREFIX`.

Listing 2.4: SPARQL query example

```

1 PREFIX buildings: <http://example.org/buildings/>
2 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
4
5 SELECT ?label
6 WHERE
7 { ?x rdfs:label "Subway Station" .
8   ?y buildings:hasMoreTravellerCapacity ?x
9 }
```

The exemplary query in Listing 2.4 returns a table of labels from buildings that fulfill the query pattern for selecting only buildings with a higher traveler capacity than the Subway

<sup>1</sup>see Turtle definition: <https://www.w3.org/TR/turtle/>; accessed: May 8, 2018

Station. In this case, a formatted table with the literals "Airport" and "Central Station" will be delivered according to the example in section 2.1.3.

Besides the output as a sequential table using **SELECT**, there are three more output formulations that can be executed utilizing **CONSTRUCT** (RDF graph), **ASK** (boolean query) or **DESCRIBE** (RDF description). Furthermore, complex query patterns may contain the keywords **UNION** and **OPTIONAL** to do a more specific query. Results can be modified by several filters, a defined order, specified limits, an offset or the removal of duplicate results. The documentation of SPARQL is available online at W3C by Harris and Seaborne (2013).

## 2.2. Linked Building Models

A building model or building information model is an object-based geometric representation attributed with valuable information for the design and construction phase and throughout the lifecycle of a building (Borrmann et al., 2015). Moreover, a more collaborative design and construction process is established with the use of BIM for the complete project team (Eastman et al., 2011). As a result of the work with different project participants, the centralized model could not be handled properly. For reasons of collaborations, the term part model has been introduced to include specific design domains as well as spatial parts of a building.

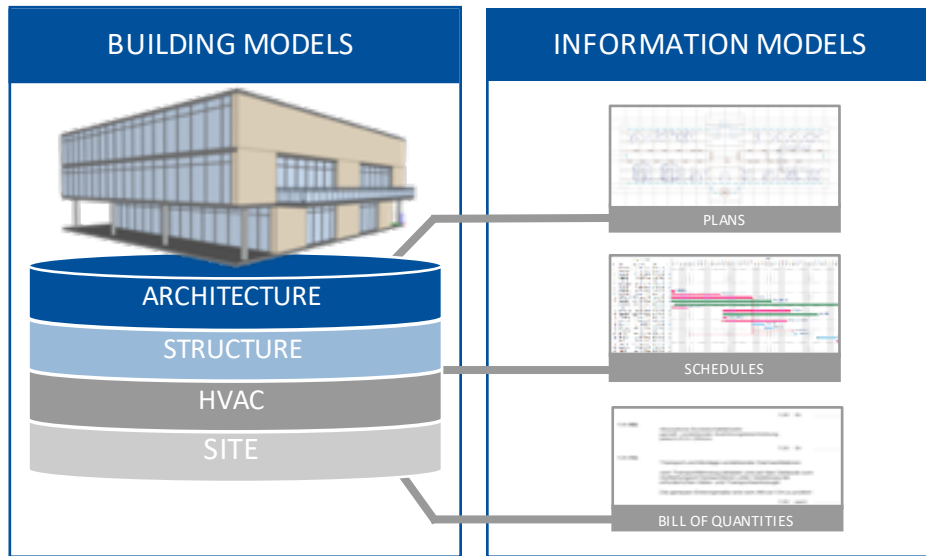


Figure 2.3.: Integration of Linked Building Models

The integration of separate part models into a coordination model by using the IFC data format is supported by most authoring tools and collaboration platforms. In addition to the part model and its inherent information, further data can be linked to the model such as documents, plans, schedules or BOQ (see fig. 2.3). The data structures of the schedule and BOQ are separate information models which facilitate the 4D construction sequencing simulation or the 5D cost estimation. In general, software applications for 4D simulation create a link between the Globally Unique Identifier (GUID) of an object from the IFC part model and the Identifier (ID) of a process from the schedule (Hanff and Wörter, 2015). This link can be generated manually or (semi-)automated. The manual link creation requires the user to connect a single object or a set of objects to a process, which can be inappropriate for large projects. Basically, the semi-automated link generation relies on a set of mapping rules (Opitz et al., 2014). These rules identify objects with a certain attribute or property (e.g. process number) and create the link to the specific process object. As stated by Törmä et al. (2012),

the combination of the different information models leads to a lack of interoperability. The technical issue in transferring linked models between software applications is also identified by Forgues et al. (2012).

The exchange of linked building models has been focused by the Mefisto project<sup>2</sup> in Germany funded by the Federal Ministry of Education and Research. Outcome of the research project is the application of multi models for linking building models with other information models (Scherer and Schapke 2014). The definition of multi model is associated with the definition of a Multi-Model-Container (MMC). Inside of the container, the part models are independent instances in their own format which can be connected using a link model. This link model provides the information about the dependencies between single elements from different part models.

A practical orientated case of application of the multi model approach is the BIM-LV-Container (see section 3.1). The BIM-LV-Container has been standardized by the DIN SPEC 91350 (2016) for the purpose of exchanging building models in the IFC format with attached BOQ in the GAEB DA XML format. The BIM-LV-Container has been developed to be a central part of the tender phase and allows data exchange especially between planner, client, contractors and manufacturers.

These approaches represent the recent stages in exchange of linked building models in Germany. Both, the MMC and the BIM-LV-Container are powerful structures for exchanging linked building models. However, a different consideration for exchanging linked building models is the use of LOD (see section 2.1). The following section deals AEC-specific LOD approaches especially for linking and exchanging building models.

### 2.3. Linked Open Data Approaches for AEC

The relationship between BIM and LOD has been examined by Abanda et al. (2013) looking at IFC and other common formats in AEC. As a result, the weakness of IFC due to constraints in the file format's expressivity has been identified. This is especially true for the use of unique building elements. Consequently, Abanda et al. (2013) consider the application of the IFC-to-RDF approach introduced by Pauwels and Van Deursen (2012). This toolbox transfers any IFC file into the ifcOWL ontology, which is standardized by buildingSMART. It instantiates ontologies from IFC files, so that the generated RDF-graph can be used to query information from the model or to link individual elements with product or material links. With these insights, the authors called LOD "a valid approach for addressing existing interoperability issues in the AEC domain" (Pauwels and Van Deursen, 2012). This ontology can directly represent the structure of an IFC file for elements such as IfcBuilding, IfcBuildingStorey, IfcSpace or IfcElement. Because of the direct translation from IFC to RDF, the ontology inherits a complexity that makes it less extendable (Rasmussen et al., 2017b). Meanwhile, Pauwels and Terkaj (2016) critically questioned the development of ifcOWL and claimed that ifcOWL will not develop into a standard according to the current state of ontology.

Nevertheless, several generic and domain-specific ontologies have been investigated by Rasmussen et al. (2017a) concluding that almost all of them violate W3C ontology policies due to redundant data sets. To prevent this, Rasmussen et al. (2017a) introduced the Building Topology Ontology (BOT) that focuses on simplicity and usability in order to replace older ontologies. Particularly the domain specific ontologies are very limited and fast outdated. To

---

<sup>2</sup>The Mefisto project: [http://www.mefisto-bau.de/overview\\_en.html](http://www.mefisto-bau.de/overview_en.html); accessed: Apr. 11, 2018

keep maintenance effort low and provide an extendable structure, the authors developed the BOT as a general and simple ontology. This straightforwardness allows specific ontologies to expand the general building ontology and use its key concepts. The structure was chosen by the authors to facilitate a fast spread of the ontology and to make BOT become a standard application in the building industry.

For the purpose of linked building models, Madrazo and Costa (2012) named building ontologies as an alternative to centralized models for flexible and dynamic data modeling. Furthermore, Törmä et al. (2012) have presented the outcome of the Distributed Transactional Building Information Management project, which contains studies on cross-model interactions and interrelations of part models. The approach relies on the before mentioned IFC-to-RDF approach and provides converted RDF data sets in a store which enables users to query information. Moreover, linksets can directly be created using RDF. The research focuses on the common problems of modeling and generating links as well as tracking and managing changes due to IFC revisions. A more advanced approach is the extension using BimSPARQL as introduced by Zhang et al. (2018). BimSPARQL is based on the general SPARQL query language and can be used to retrieve information or find building objects using simplified queries on the IFC data. There are possible use cases of BimSPARQL especially for automated logic checks.

## 2.4. Conclusions

As stated in the previous sections, IFC is the central element during the BIM planning process. Hence, it has limitations in the expandability and has not been designed to transfer additional information models. For this purpose, the MMC offers an appropriate solution to exchange multiple building models linked with information models in a container file. Furthermore, there are many approaches to adapt LOD into the construction industry. Building ontologies like BOT or the DRUM project definitely cannot yet productively replace the IFC interface, but they can extend it and improve its expressability. The concept of LBD will definitely have a future in construction as it has been researched a lot in the last 10 years. Nevertheless, there are a lot of open questions how to use the technology correctly within the AEC domain. The combination of both, a container and acLOD, results in a powerful foundation for semantic data exchange, which was the intention for the development of the ICDD.

## 3. Exchange of Linked Building Models

### 3.1. Multi-Model Container and BIM-LV-Container

#### Multi-Model-Container (MMC)

The exchange of linked building models was focused by the Mefisto<sup>1</sup> project in Germany funded by the Federal Ministry of Education and Research. The aim of the project has been the development of methods for collaborative and process-orientated planning and realization of construction projects. Using a non model-centric approach, the outcome of the research project is the formal definition of a data schema for Generic Multi-Models (Scherer and Schapke, 2011). The development of the Multi-Model approach is based on the following requirements by Fuchs (2015): cross-domain applicability, ability to store domain models in their original format, standardized implementation, serialized format, persistent and restorable links and variable link types.

To meet these requirements, the object-orientated data schema of a Multi-Model is structured as shown in the UML diagram in fig. 3.1. The root element is the Multi-Model instance. A Multi-Model is associated to one or more Elementary Models, which can be any instances of self-contained data models, e.g. IFC models or schedules. Furthermore, the Multi-Model can be associated with any number of Linked Models, which each represent a non-empty set of Links. These Links consist at least of two or more Linked Elements from different Elementary Models. Each Linked Element only belongs to a single Elementary Model and contains the information to identify the corresponding Element from the Elementary Mode. With these information, the link between building models and information models can be established. To identify and connect the entities inside of the Elementary Models, the links

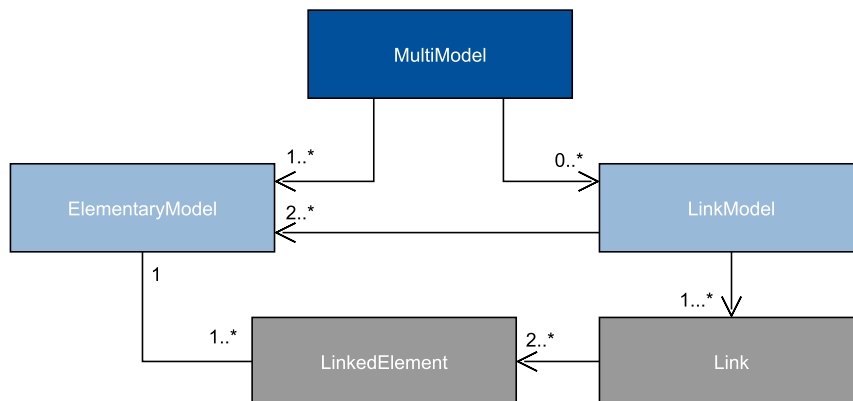


Figure 3.1.: UML diagram of the Generic Multi-Model (see Fuchs et al. (2011))

inside the Multi-Model schema are based on unique IDs. This method allows to leave all

<sup>1</sup>further information at <http://www.mefisto-bau.de/>; accessed: July 2, 2018

### 3. Exchange of Linked Building Models

elements in their original state as long as there is no inconsistency concerning the allocation of IDs within the Elementary Models (Fuchs, 2015). Therefore, the original data requires a uniquely identifying attribute. For instance, that could be the GUID of an IFC object, the item ID of a BOQ position or the task ID from a schedule. Moreover, Demharter et al. (2014) presented manipulative and non-manipulative methods to determine the identity of an object without an explicit ID attribute.

The definition of the Multi-Model schema is associated with the implementation of a MMC to exchange serialized instances of Multi-Models. Inside of the container, the Elementary Models are independent instances in their original format which can be connected using a Link Model (see fig. 3.2). This Link Model provides the information about the dependencies between single elements, for example, from different part models (Fuchs et al., 2011). In addition to the document-based data inside the container, annotations and descriptions can be attached using metadata. The metadata entries inside of the container are structured as key-value pairs inside the header file of the container (see Listing 3.1).

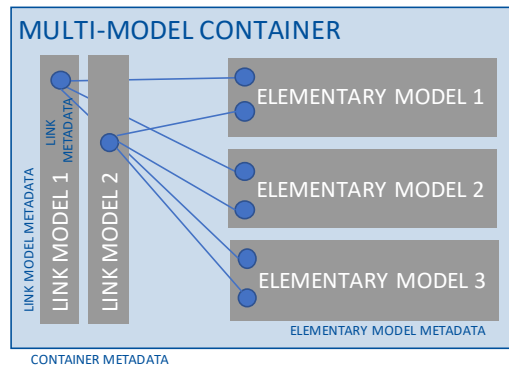


Figure 3.2.: Generic Multi-Model Container (see Demharter et al. (2014))

Every MMC file consists of an archive with conformity to the `application/zip` format and is characterized by the `MultiModel.xml` header file formalized by a specific XML Schema Definition (XSD) (Demharter et al., 2014). The file has the sections `meta`, `models`, `linkModels` and `context` and with this defines the container type and locates the models inside of the container (see Listing 3.1). Moreover, it provides the location of the links and identifies the models to be linked. In analogy to the UML diagram, a Link Model can be described with the `Links.xml` file (linkXML). The XSDs for the mmcXML and linkXML are provided by buildingSMART Germany<sup>2</sup>. Besides the header meta section, each element in the models and linkModels section is equipped with a meta section to provide the respective information.

Listing 3.1: MMC file header

```

1 <?xml version="1.0"?>
2 <container guid="..." formatVersion="1.0">
3   <meta>
4     <origin>
5     <created>2018-04-06T13:52:31</created>
6     </origin>
7     <info>
8     <i k="ContainerDescription" t="xs:string" v="MMC-
      Datenaustausch"/>
9     </info>
10  </meta>

```

<sup>2</sup>available at <https://github.com/BuildingSMART/MMC>; accessed: Apr. 22, 2018

```

11     <models>[...]</models>
12     <linkModels>[...]</linkModels>
13     <context>[...]</context>
14 </container>

```

### BIM-LV-Container (BLC)

The BLC is a specialization of the Multi-Model-Container and is standardized in the DIN SPEC 91400. It contains building models and BOQs according to ISO 16739 and PAS 1067 GAEB Data exchange and structure of Bill of Quantities (GAEB DA) XML. The BLC bases on the presented structure and also has the .mmc file type extension. The standard delivers a meta data definition in key-value pairs due to the specific use case of BOQ exchange as it is defined in Germany. The metadata definitions for the building model and BOQ can be found in the standard.






|   |                      |                  |                      |          |
|---|----------------------|------------------|----------------------|----------|
|  | [Content_Types].xml  | 22.04.2018 16:07 | XML-Datei            | 1 KB     |
|  | BillOfQuantities.x82 | 22.04.2018 16:07 | X82-Datei            | 1.397 KB |
|  | BuildingModel.ifc    | 22.04.2018 16:07 | Industry Foundati... | 393 KB   |
|  | Links.xml            | 22.04.2018 16:07 | XML-Datei            | 58 KB    |
|  | MultiModel.xml       | 22.04.2018 16:07 | XML-Datei            | 2 KB     |

Figure 3.3.: File content of a BIM-LV-Container

Inside the MMC archive, a minimum of four files describes the BIM-LV-Container. It contains a building model in the IFC format, a BOQ in a GAEB format and three more XML files. The specification of the DIN standard provides one-to-one links between building elements and information elements only. Moreover, both need a unique identifier (e.g. the GUID and GAEB Item ID) which are then stored within the link model. In addition to work items, quantity splits can also be referenced to a building element. With this, a direct link between building object and the position from the BOQ using attributes of the IFC model is no longer required, because both are dynamically linked through the link models (Schiller and Faschingbauer, 2016). Further application cases of the BLC are model-based spatial descriptions and model-based specifications for tenders using the classification catalog from the related DIN SPEC 91400 (2017).

## 3.2. Information Container for Data Drop

Besides the approaches in Germany, there have been efforts in the Netherlands to develop an interdisciplinary container for the exchange of information called the COINS project (Hoeber et al., 2015). The project targets the standardization of a flexible information container for connecting the complete amount of building data using linked data approaches. An initial version of the container has been published in 2010, an update was provided in 2014. These versions are the predecessors of the ISO 21591: Information Container for Data Drop.

The resulting ISO 21597 standard consists of two parts and introduces a specification for a container for exchange of multiple information models in a single data drop. The first part (ISO 21597-1, 2018) comprises the container definition whereas the second part (ISO 21597-2, 2018) focuses on the dynamic semantics. While the container enables the storage of included or remote documents and the connection to other separate data in a single container, the dynamic semantics part focuses on the semantic integration of custom data models according to linked data principles. The analysis will focus on both parts in the following sections and outline the main structure of the container to provide a clearly defined basis for the implementation.

### 3.2.1. Part 1: Container

A container specified according to this standard has the filetype extension `.icdd` and represents an archive conforming to the `application/zip` format. The default configuration of the container provides that at least the three folders and the header file as seen in fig. 3.4 are required. The header file of the container is always the `index.rdf` file which is serialized as an RDF (see ch. 2.1.1) and related to the container ontology and the linkset ontology.

The ontology files define the container and link concept and can be found in the ontology resources folder. They do not necessarily need to be provided inside the container as they are available online. Hence, the local files always have a higher priority than the remote files. The container ontology delivers syntax for the creation of the header file, which then can define the version of the container, a set of external documents, a set of inherent documents and the references to link data sets. This details are provided inside the header file using an individual of the `ct:ContainerDescription` class, which is derived from the `owl:Ontology` class (see ch. 2.1.2).

The documents, for instance the models or files, belong to the payload documents folder. Within this standard every type of self-contained data is designated as 'payload'. Documents can be expressed with an individual of the abstract class `Document` from the container ontology. It supplies metadata information and the link to the container individual as well as a prior version of a document. An individual instantiated from the document class must either be the type `InternalDocument` or an `ExternalDocument` and include the respective location of the data. Moreover, the `Document` class is specialized with certain sub-classes that define an attributed document

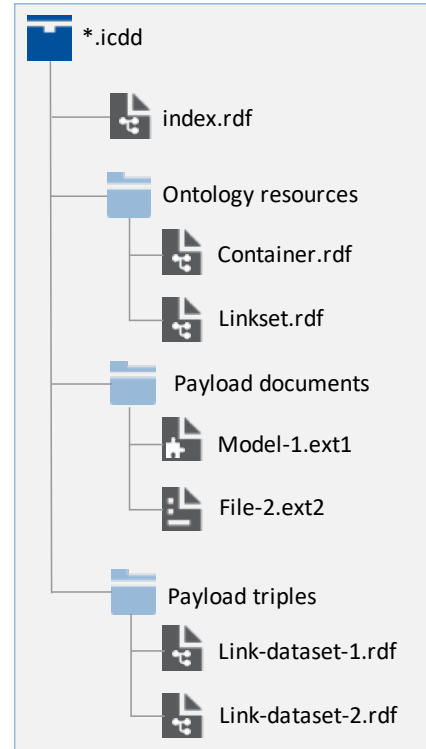


Figure 3.4.: Structure of the ICDD (see ISO 21597-1)



like an **EncryptedDocument** or a **SecuredDocument**.

The corresponding data sets are located in the payload triples folder. These link data sets are characterized by the linkset ontology which specifies linkages between documents or entities inside documents. **Links** are defined as a group of two or more **LinkElements**, and can be specialized as shown in fig. 3.5. On the one hand, the ontology provides **DirectedLinks** that contain any number of **LinkedElements** differentiated as 'From' and 'To' elements. A subclass of the **DirectedLink** is the **Directed1toNLink** which restricts the number of incoming **LinkElements** to one. On the other hand, there are **BinaryLinks** that link exactly two **LinkElements**. A specialization of both, the **DirectedLink** and the **BinaryLink**, is the **DirectedBinaryLink** combining the two characteristic properties of the super classes. The linkset ontology can be extended with user-defined subtypes of the class **Link** using the `rdfs:subClassOf` expression for the class definition.

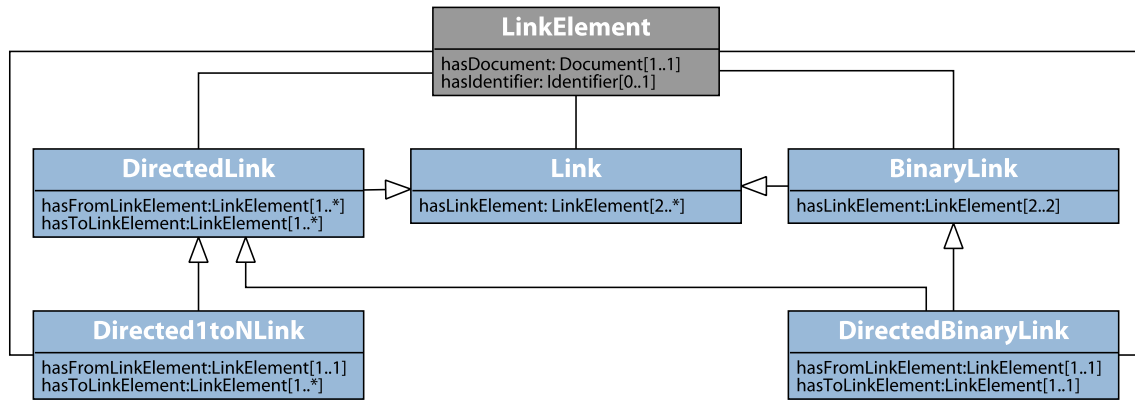


Figure 3.5.: UML diagram of the Link inheritance (see ISO 21597-1)

The **LinkElement** can be related to exactly one **Document** from the container ontology using the `hasDocument` object property. Using the `hasIdentifier` object property enables the **LinkElement** to refer directly to a document's entity using the abstract class **Identifier**. Three subclasses extend the **Identifier** according to the identification method of entities, in particular query based, string based or URI-based identifiers. In case a document is an IFC model, a **StringBasedIdentifier** could contain the GUID to provide an entity link.

In addition to the structure of the container and the link between its payload, the ontologies enable users to perform various versioning tasks, for example for design case studies, tracing versions or referencing different planning states. Therefore, the classes **ContainerDescription**, **Document** and **Linkset** are equipped with `versionInfo` properties. Moreover, instances of these classes offer the possibility to link to a predecessor from the same class type to track the version history.

Both ontologies can be extended with additional metadata using functional or data type properties. Therefore, the tags `rdfs:label`, `rdfs:domain` and `rdfs:range` are mandatory and no other tags may be used. A complete overview of the ontologies with their inherited objects and properties can be found in the standard. A summary of the conformity criteria can be found in chapter five of the first part of the standard.

### 3.2.2. Part 2: Dynamic Semantics

After the basic functionality of the information container has been outlined in the section before, this section focuses on the second part of the standard which are the Dynamic Semantics. This part extends the container with semantic information by harnessing LOD. Therefore, the `DynamicSemantics.rdf` file is deposited into the `Ontology Resources` folder. A summary of the conformity criteria can be found in chapter five of the second part of the standard. The ontology extends the container with six major concepts:

#### Metadata

In order to extend the metadata of an information container and its content, the root class `InformationModel` is introduced in the `DynamicSemantics` ontology. The `InformationModel` delivers an abstract subclass `Concept` which is associated with the basic metadata for authors and versions from the `ContainerDescription` and can be instantiated using the subclasses `Entity`, `Relation`, `AbstractProperty`, `ComplexPropertyValue` or the `Container` ontology class `ct:Party`. These classes and their correct use in relation to `DynamicSemantics` are described below.

#### Information Model Entities

An instance of the class `Entity` can be used to provide information about any entity included in an information model, e.g. physical components like building elements. Besides the inherited properties from the `Concept` class, it has properties for versioning according to ch. 3.2.1 and enables connections between instances of the `Entity` class.

#### Properties

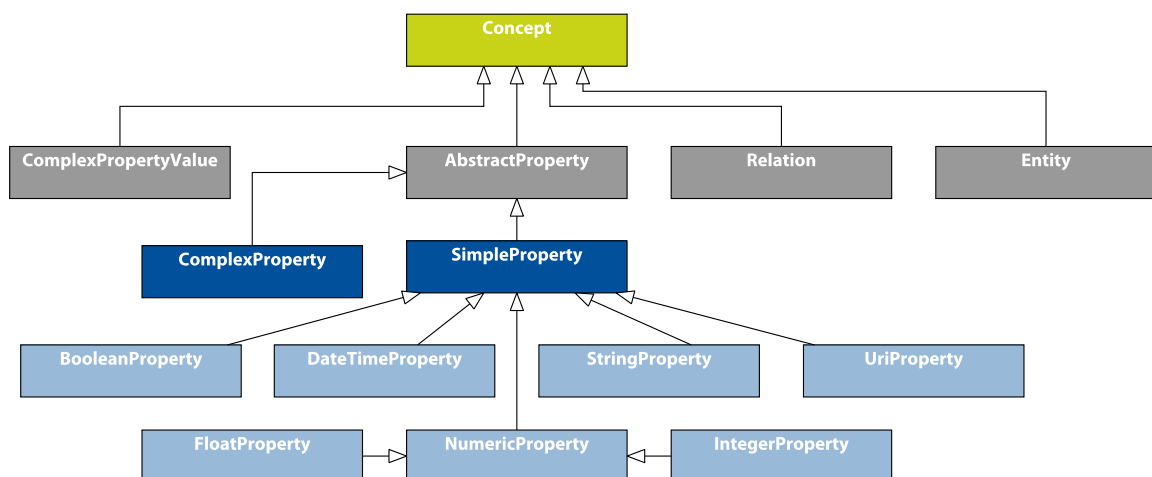


Figure 3.6.: UML diagram of the `AbstractProperty` inheritance (see ISO 21597-2)

The standard provides a class structure for modeling either primitive types of properties or complex properties with mutable content. Therefore, the two classes `ComplexProperty` and `SimpleProperty` are inherited from the abstract class `AbstractProperty`, which itself

is inherited from the **Concept** (see fig. 3.6). Because of this heredity, properties can also be provided with version and author information that are defined in the **Concept** class.

The types that form the **SimpleProperty** are referring to the XML built-in data types which are defined in the XML Schema Definition by W3C. Numeric properties can be equipped with units that refer to the ICDD-QUDT-Units extension (see ch. 3.2.2). Complex properties consist of an **objectValue** attribute that is from the type of a **ComplexPropertyValue**. For instance, a complex object value can be a **ct:Party** to introduce a new role like an inspector, a supplier or an engineering company.

## Relations

For modeling relationships between instances of the class **Concept**, the **Relation** class is introduced as an abstract class. It can be specialized by the disjointed classes **Connection** or **ContainsRelation**.

On the one hand, the **Connection** class has to be used to connect exactly two instances of the **Entity** class directly. Furthermore, the specialized class **DirectedConnection** also allows directional connections between a **fromEntity** and a **toEntity**. Therefore, the **Entity** class defines the properties **hasConnections**, **hasIncomingConnections** and **hasOutgoingConnections**.

On the other hand, the **ContainsRelation** can be used to define hierarchies and aggregations within a structure. The relation connects an instance of the **Part** class to an instance of the **Assembly** class. The **Assembly** is aggregated from a group of **ContainsRelations** as seen in fig. 3.7. Using this structure, for example, building topologies like the relation of a wall and its including windows can be modeled within the metadata of the container.

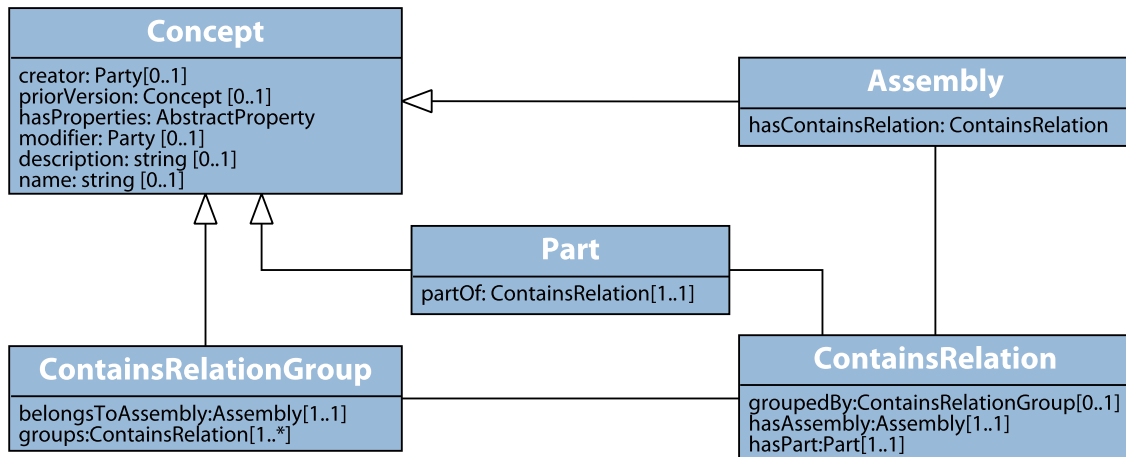


Figure 3.7.: UML diagram of the ContainsRelation inheritance (see ISO 21597-2)

## Expiring Concept

To distinguish prior and current versions of an **Entity**, an object property **priorVersion** can be used to refer to an outdated version which has the same class as the object itself. Expired entities have to be connected to the **ExpiredConcept** class to be tagged as expired. The

expiring concept is an extending development to the versioning information of the container in part 1.

### User Defined Ontologies (UDO)

The `DynamicSemantics.rdf` can be extended using RDF files. UDOs belong to the Ontology Resources folder within the container archive or may be available through an URI. Furthermore, they can extend the classes introduced within part 2 of the standard. In most cases, a specialization of the `Entity` class is relevant. The above presented concepts can be applied in an UDO to extend the metadata of the container. Use cases of UDOs are presented in the standard itself with the ICDD-QUDT-Units extension, which can be found in Annex C of the standard. QUDT is an ontology-based specification and stands for quantity kinds, units of measurement, dimensions and types. With a manipulation of the `unit` object property from the numeric simple properties, the range of the property values can be extended to accept types of `qudt:Unit`. The adapted version of the general QUDT ontology is provided with the standard.

### 3.2.3. Integration of DynamicSemantics with Container and Linkset

The integration of both parts of the standard is handled with the definition of `subClassOf` triples between part 1 and part 2. With the definition of subclasses (see fig. 3.8), the definition of `Links` can be seen as a specialization of the `Entity` structure with its defined properties in the DynamicSemantics. Altogether, an analogy between the two parts can be created as well as an inheritance structure. Besides the classes, also the properties are defined as sub-properties. With this adaption the container is supplemented by a wide range of expressions and UDOs for linking documents as well as entities.

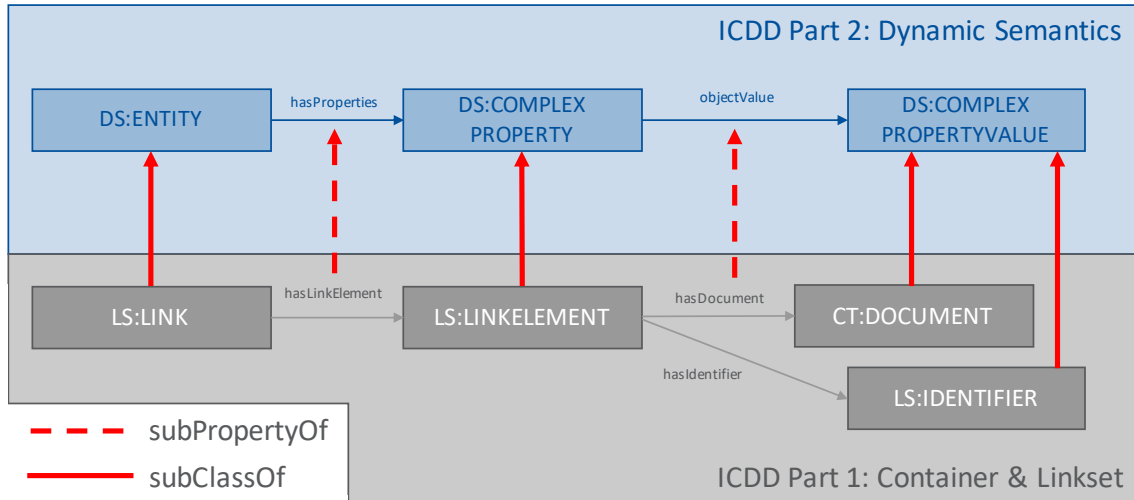


Figure 3.8.: Integration of DynamicSemantics with Container and Linkset (see ISO 21597-1)

### 3.3. Comparison and Discussion

The following chapter compares the presented containers and discusses the characteristics of both. In table 3.1, criteria for both the container and the relying data model are listed. The data model quality criteria are defined according to the criteria of Fettke (2001) for classification systems.

Table 3.1.: Comparison of ISO 21597 and DIN SPEC 91350 / MMC

|                     |  |   |
|---------------------|--|---|
| Structure           | well structured with files organized in folders  | simple structure inside container with all files on top level                                   |
| Linksets            | provides various extendable linksets and relations   | provides simple one-to-one links only   |
| Versioning          | versioning is fully implemented for documents, entities, and other objects   | supplies metadata information about the creation date and the software with which it is created |
| Metadata            | almost unlimited possibilities for metadata annotation   | allows key-value-pairs with primitive data types for metadata annotation                        |
| Applicability       | high relevance and wide range of applications  | high application relevance but less use cases   |
| Extendability       | highly extendable through UDO and external LOD extensions  | very specialized and less extendable  |
| Consistency         | inconsistent data due to discrepancies between RDF and object-orientation and error-prone implementation for UDO parsing | no automatic control of inconsistencies, but generally less prone to error                      |
| Economic efficiency | high effort in implementation due to RDF parsing and dynamic semantics   | medium effort in implementation due to simple structure and less complexity                     |

The general structure of both containers is quite similar and has a clear arrangement. The use of zip archives with a defined header file is common practice for the simple creation of a container file type. With this file type, the container can easily be associated with specific applications, which is important for the future user experience. An example for the use of archives for a container in the building industry is the `.bcfzip` format. It can be used for the exchange of issues and markups of building models and includes a header file which localizes screenshot files and geometric position files inside the container. Because of the amount of ontology and linkset files the ICDD needs to be organized in a defined folder hierarchy.

The definition of the linksets in ICDD is different to the BLC: While the BIM-LV-Container uses the simple structure one-to-one links due to the application case, the ICDD provides a more complex linkset structure with directed or undirected links which can be extended by sub-typing links with an additional RDF ontology (see annex C of ISO 21597-2 (2018)). It should be noted that the Multi-Model-Container as a generalization of the BLC also supports the use of specialized links.

### 3. Exchange of Linked Building Models

Moreover, the ICDD delivers functionalities for the versioning of documents, entities, links, and the whole container. The interrelationships between versioned elements can be handled with the relation classes from the DynamicSemantics ontology. Through these structures, not only outdated versions can be managed, but also alternative versions can be provided in a container. The MMC can handle specific versions of an elementary model, but cannot annotate the interrelationship between them. This can lead to an inconsistent repository and orphaned files inside the container.

Generally speaking, the ICDD structure has been developed to manage metadata first. With the Linked Data approach and the UDOs, there are nearly unlimited possibilities to store and structure additional metadata for different objects. The extension of properties in part 2 of the standard delivers several data types and the opportunity to annotate with complex user defined properties. While the ICDD offers these features for metadata annotation, the BLC and MMC offer a key-value structured metadata dictionary, which can be applied to most of the included data-types, e.g. documents or links. There is no possibility to express more complex properties.

The applicability of both approaches is given. While the MMC has been applied in commercial software applications with the usage of the BLC, the ICDD is still under development and has not been validated in its current version. Nonetheless, the predecessor of ICDD was developed close to the practice by a consortium of construction companies in the Netherlands. Furthermore, it can cover a lot of application cases due to its extendability.

On the one hand, the ICDD is very extendable through the development of UDO and external semantic web extensions. This offers possibilities to extend containers according to special needs and individually for every project. On the other hand, the BLC is very specialized, and also the MMC has a static data structure, so that there is less extendability. Nevertheless, the MMC can serve as a container in a lot of use cases in its actual configuration.

The ICDD is based on semantic data in the RDF format with the OWL modeling language. Because of the difference between OWL modeled ontologies and object-oriented software applications, the implementation has to focus on several aspects which also have been examined by Völkel (2005). If one or more of these aspects have been neglected, especially when parsing UDOs, this can lead to inconsistencies during the parsing process. Data consistency is therefore strongly dependent on actual implementation. By contrast, the MMC has a solely static data structure, which prevents inconsistencies. Both do not feature an automatic inconsistency control, which then has to be implemented into the client's software application.

As the aforementioned paragraph already states, there are differences in the economic efficiency due to the implementation effort. The BLC can be implemented with less effort which is already evident from the standard. More complexity in implementation is caused by a general MMC implementation. Due to the static implementation, there is only medium effort. However, because of the dynamic semantics and the semantic web implementation of the ICDD, it is obviously the alternative with the most implementation effort.

In conclusion, the high flexibility and extensibility have negative effect on the error-proneness and complexity of the implementation. Nevertheless, the ICDD provides far more functionality and, due to the current technology, has large potential for the future.

## 4. Requirement Engineering

The analysis and specification of requirements for an application are the first and thus most challenging steps in software engineering (Balzert, 2009). The requirement analysis determines user-oriented needs and conditions that a software system needs to fulfill. The specification of requirements is the complete, unambiguous and consistent composition of all requirements in a standardized format (Balzert, 2009). Together these steps describe the term requirement engineering.

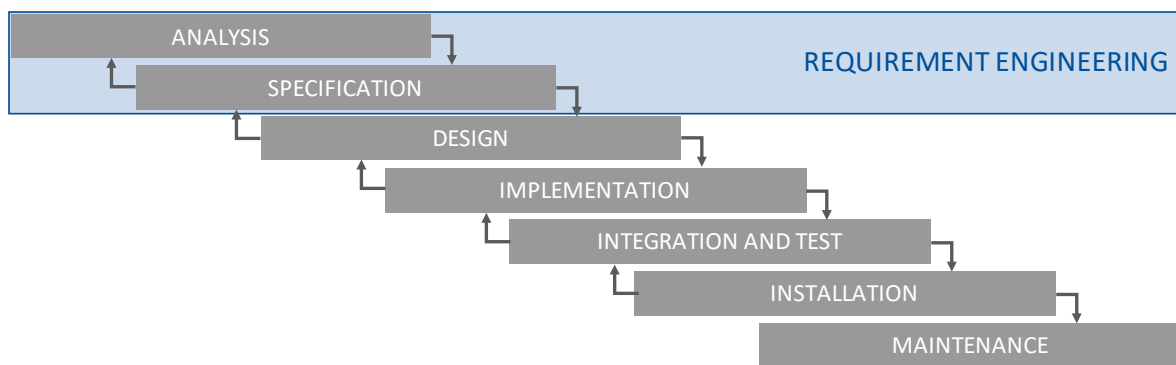


Figure 4.1.: Process model for software engineering according to Royce (1987)

As shown in fig. 4.1, requirement engineering, as well as overall software engineering, is an iterative process between the single stages. The described requirements must be validated, re-analyzed and newly specified if they are not appropriate to forward them to the design stage. Requirements of a software application are classified into functional and non-functional requirements. On the one hand, functional requirements describe a function or service provided by a software application (Balzert, 2009). These include the structure, usage and transfer of data, the functions for processing data and the dynamic system behavior, e.g. between functions or data. On the other hand, non-functional requirements concern the quality, performance and boundary conditions (Glinz, 2007). Quality requirements include statements about the availability, usability, expandability, maintainability and data security. Performance requirements focus on the time behavior, the resource utilization and efficiency of a software system. Furthermore, boundary conditions can be technical or physical constraints, but also legal, cultural or environmental statements (Glinz, 2007).

Most requirements for software applications use natural language for specification (Balzert, 2009). However, the pros and cons of this are on the one hand the flexibility and intelligibility but on the other hand the ambiguity of interpretation. An alternative is the avoidance of natural language requirements using formal model-based specifications. In this thesis, the requirements are specified in natural language. The functional requirements are represented as so-called user stories that were introduced by Beck (1999) within his Extreme Programming approach. A user story is a incisive written description of a single function that the software system offers to a user. The summary of user stories forms the product backlog. As the user stories use natural language, a precise pattern needs to be defined. For Object-Oriented Programming (OOP) the following pattern has turned out to be successful according to Zeaaraoui

et al. (2013):

As a **<role>**, I want to **<action>** **<object>**, so that **<business value>**.

In the context of programming, role means a user that has a specific role within a software system, e.g. a logged in user. Furthermore, action and option are a combination which achieve a specific goal for an object using the defined action. The resulting business value can be additionally described as a reason or motivation. User stories whose complexity exceeds this pattern must be divided and defined in smaller user stories. In the following sections, the beforementioned requirements are specified.

## 4.1. Specification of Functional Requirements

The table 4.1 contains the specifications of functional requirements which arise from the analysis of the ISO standard in ch. 3 and the related use cases that can be found in the standard's appendix. The user stories are numbered and sorted by the category. The category IO stands for all operations in the field of import and export, VAL for the validation, VIS for the visualization, LINK for all requirements concerning links and linksets, DATA for all requirements according metadata and API for all requirements related to the Web API.

Table 4.1.: Definition of user stories

|     |     |   |
|-----|-----|---|
| 001 | IO  | As a website user, I want to upload an ICDD file, so that it is available on the server for further operations.   |
| 002 | IO  | As a website user, I want to access an uploaded ICDD file during the active session, so that I can perform operations on it.  |
| 003 | IO  | As a website user, I want to delete an uploaded ICDD file, so that no data remains on the server.   |
| 004 | IO  | As a website user, I want all data to be deleted after the session has expired, so that no data remains on the server.  |
| 005 | IO  | As a website user, I want to download my manipulated ICDD file, so that I can use it outside the application.   |
| 006 | IO  | As a website user, I want to upload additional documents to the website, so that I can use them inside the container as a payload document.                           |
| 007 | VAL | As a website user, I want to start a validation for an uploaded ICDD file, so that I can be sure that my file is conform to the ISO standard.                         |
| 008 | VAL | As a website user, I want to view and download validation results for a validated file, so that I can see which aspects have been validated.                          |
| 009 | VAL | As a website user, I want to correct failed validation aspects for an ICDD file, so that I can get a validated file.  |
| 010 | VIS | As a website user, I want to get a structured representation of an uploaded ICDD file, so that I can see the contained payload documents, the metadata and the links. |



#### 4. Requirement Engineering

|     |      |   |
|-----|------|---|
| 011 | VIS  | As a website user, I want to get a visualization of payload documents, so that I can see what information is contained by the documents and what the appearance of a geometric document looks like. |
| 012 | VIS  | As a website user, I want to get a graphical visualization of links, so that I can see which elements of the payload documents are concerned by a link.   |
| 013 | LINK | As a website user, I want to get a representation of the links from a specific linkset, so that I can see which link types and link elements are included.  |
| 014 | LINK | As a website user, I want to create new links into a specific linkset, so that I can extend the existing data.  |
| 015 | LINK | As a website user, I want to update and delete existing links, so that I can manipulate the existing data.  |
| 016 | LINK | As a website user, I want to filter existing links, so that I can see specific link types.  |
| 017 | LINK | As a website user, I want to validate linksets for consistency according to user-defined rules, so that I can see whether a linkset is consistent or not.   |
| 018 | DATA | As a website user, I want to create metadata for any object within the container, so that I can extend the existing data.   |
| 019 | DATA | As a website user, I want to update and delete metadata for any object within the container, so that I can manipulate the existing data.  |
| 020 | DATA | As an website user, I want to read metadata for any object within the container, so that I get information about any object.  |
| 021 | API  | As an interface user, I want to perform a <b>POST</b> request containing an ICDD file, so that I can upload the file and start the validation process.  |
| 022 | API  | As an interface user, I want to perform a <b>GET</b> request for the validation results, so that I can be sure that my file is conform to the ISO standard.   |
| 023 | API  | As an interface user, I want to perform a <b>GET</b> request for the payload documents, so that I access the original data of a payload document.   |
| 024 | API  | As an interface user, I want to perform a <b>GET</b> request for metadata of the container, so that I can access single metadata entries.   |
| 025 | API  | As an interface user, I want to perform a <b>POST</b> request containing metadata, so that I can update or delete single metadata entries.  |
| 026 | API  | As an interface user, I want to perform a <b>GET</b> for the ICDD container, so that I can access a file based version of the manipulated container.  |
| 027 | API  | As an interface user, I want all data from a closed session being deleted, so that I can be sure that no information from my container is stored on the server.                                     |

## 4.2. Specification of Non-Functional Requirements

### Technical Constraints

The application must be made available as a web application as well as a Web Application Programming Interface (API). Both do not need an authentication to perform the implemented operations, but therefore the user session (=30 min) need to be linked to an uploaded ICDD file. As long as the session is alive, the user can access the associated file. If the session is expired (>30 min) or the user explicitly deletes the files, the connection must be disposed and the uploaded data as well as all further created data needs to be deleted. As Web APIs are generally stateless, the session ID provided at the file upload must be carried along all requests. The server must recognize, whether a file has been accessed within the standard session time and dispose the generated data if the session is expired.

### Quality Requirements

The application must satisfy the following qualities:

**Availability:** The web application as well as the Web API must be available for public access via HTTP protocol. The definition of the API must be documented and be available online.

**Usability:** The web application should be easy to understand. Therefore, the Graphical User Interface (GUI) layout and its elements should be consistent in functionality and design. Error messages, especially when processing an ICDD file should be available for the user and explain how to recover the file.

**Maintainability:** Maintenance activities concerning the web application and the API are carried out exclusively on the server. There is no need for any client-based updates. The API definition should be formulated as generally as possible so that subsequent changes are not noticeable to the client.

**Data Security:** As already stated in ch. 4.2, the complete user data must be deleted automatically after the session timeout or a manually delete. This also applies to all log files of the validation. A global logfile for logging application errors is not affected by this.

### Performance Requirements

**Time behaviour:** Generally, the application has to react within an appropriate time span (<5 sec) for enterprise software application to every request either from the website or the API. An exception to this is the upload of ICDD and other files.

**Resource utilization and efficiency:** The web application should utilize not more than 8 GB of physical memory of the server capacity.

### 4.3. Requirement Management

In order to manage the requirements along the design and implementation processes, the product backlog with the use cases has been moved onto a MS Team Foundation Server (see fig. 4.2). This development tool supplies a software project management system including the product backlog. As a distinctive feature it provides a source code repository. This repository can be utilized with MS Visual Studio, so that developers can directly access their work items and link source code change sets to them.

| ID  | Arbeitsauf... | Titel  | Zustand   | Rücks... | Tags |
|-----|---------------|--|-----------|----------|------|
| 693 | Product Ba... | As a website user, I want to get a representation of the links from a specific linkset, so that I can see which link types and ... | Bestätigt | 2        | LINK |
| 688 | Product Ba... | As a website user, I want to view and download validation results for a validated file, so that I can see which aspects have...    | Fertig    | 2        | VAL  |
| 683 | Product Ba... | As a website user, I want to delete an uploaded ICDD file, so that no data remains on the server.                                  | Fertig    | 3        | IO   |
| 699 | Product Ba... | As an website user, I want to read metadata for any object within the container, so that I get information about any object        | Bestätigt | 3        | DATA |
| 685 | Product Ba... | As a website user, I want to download my manipulated ICDD file, so that I can use it outside the application.                      | Fertig    | 4        | IO   |
| 684 | Product Ba... | As a website user, I want all data to be deleted after the session has expired, so that no data remains on the server              | Bestätigt | 5        | IO   |
| 696 | Product Ba... | As a website user, I want to filter existing links, so that I can see specific link types.   | Neu       | 5        | LINK |

Figure 4.2.: Product backlog inside Team Foundation Server

Using agile methods in software development, the user stories were rated with story points from one to ten in order to obtain a ranking according to priority (Beck, 1999). Priority one stories are more likely to be developed in early stages than priority ten stories. As seen in fig. 4.3, the user stories can be linked to other items, such as other user stories, functions or tasks. With this, the development can be structured and the process can be monitored.

|     |               |   |               |                |      |
|-----|---------------|---|---------------|----------------|------|
| 709 | Aufgabe       | ICDD: Implement metadata parsing with RDF   | Philipp Ha... | Aufgaben...    | DATA |
| 698 | Product Ba... | As a website user, I want to create metadata for any object within the container, so... | Philipp Ha... | Neu            | DATA |
| 699 | Product Ba... | As an website user, I want to read metadata for any object within the container, so...  | Philipp Ha... | Bestätigt      | DATA |
| 662 | Funktion      | ICDD: Parse CtContainerDescription from container header                                | Philipp Ha... | In Bearbeit... | DATA |
| 672 | Fehler        | ICDD: parsing subfolders for payload documents  | Philipp Ha... | Neu            | IO   |
| 675 | Fehler        | ICDD: add modifier object to container  | Philipp Ha... | Neu            | DATA |
| 673 | Funktion      | ICDD: parsing the creationDate and modificationDate of objects                          | Philipp Ha... | Fertig         | DATA |

Figure 4.3.: Requirement management inside Team Foundation Server

With this preliminary work, the design and implementation stages can be focused. Furthermore, during the implementation process, the work items in the development tool are used for tracking development progress and managing issues and errors during evaluation phases.

## 5. Software Implementation

This chapter deals with the conceptual design and implementation of a software application according to the requirements defined in Chapter 4. Before the functional requirements can be implemented, the architecture of the software must first be defined. Therefore, the first section sets up a basic framework for a development environment. It describes the use of third-party libraries for RDF serialization and the viewing of IFC files. The following sections then explain the respective functional implementations.

### 5.1. Basic Framework Design

#### 5.1.1. Web application and interface design

The development of a web-based application is being done using ASP.NET Core. This framework is a platform independent technology for creating web applications and is provided by Microsoft. When using ASP.NET Core, Razor Pages is a simplified page-centric view engine for dynamically created websites, which is based on the Model-View-Controller (MVC) pattern. This pattern generally splits an application into a data model, a representation (view) of data and a controller, which maintains the view and the model. The Razor Pages framework separates the client-based HTML site from the server-based **C#** code in the so-called page model. Data from the model can be rendered into the HTML site at runtime to generate content for browsers dynamically.

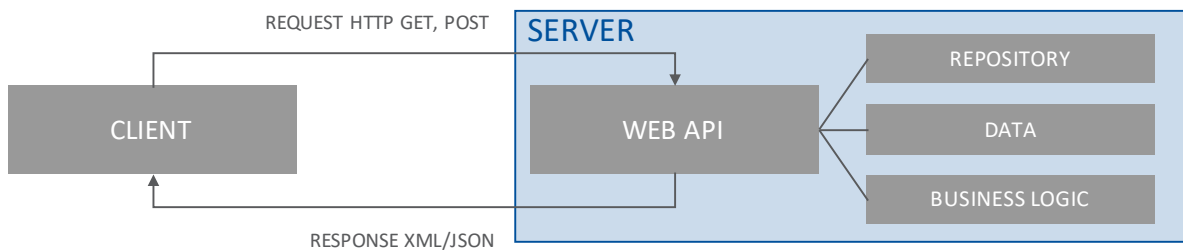


Figure 5.1.: Design of the Web API

In order to establish a uniform, machine-readable client-server architecture, the web application is equipped with an accessible web interface. With the additional provided ASP.NET Web API, web services like Representational State Transfer (REST) can be implemented. REST uses the stateless HTTP protocol, in this case with the most common methods **GET**, **POST**, **PUT** and **DELETE**. Resources on the server can be addressed using URIs and are usually transferred as XML or JavaScript Object Notation (JSON) (Rodriguez, 2008). Web APIs build on the ASP.NET Web API framework are based on the REST principle and serve the common requirements that the interface is resource-oriented, stateless, cacheable and structured in a layered system. The layered system means that the access to data from different resources

is granted at exactly on location, the Web API, as it is described in fig. 5.1. The client, e.g. a desktop application or a web application, sends an HTTP request to the Web API. The server backend processes the request using the business logic, e.g. the validation tool, the data and the files from the repository. A response from the Web API returns in either XML or JSON format. For implementation, the server backend is used for both Web API and the web application.

### 5.1.2. RDF Library

To develop a powerful and efficient application, a third-party library to deserialize and parse the RDF files from the container is needed. The library is required to be open-source and free licensed. Furthermore, three major functions need to be supported: parsing from RDF/XML serialized files, querying data from the parsed file and writing RDF/XML files. When selecting the libraries, care was also taken to ensure that they are maintained at regular intervals and are compatible with the above environment. Overall, the following libraries were considered:

- Intellidimension Semantics.Framework 2.0, latest commit: 2010-09-29
- RDFSharp, <https://github.com/mdesalvo/RDFSharp>, latest commit: 2018-08-25
- dotNetRDF, <https://github.com/dotnetrdf/dotnetrdf>, latest commit: 2018-06-07

Due to the activity of the last two libraries, these are subject to closer examination. Both frameworks work with .NET, while dotNetRDF has been proved to work with the latest .NET Core version. They are able to create and manage common RDF models (graphs, triples, namespaces). Moreover, the dotNetRDF is distinguished by the fact that it also has an ontology graph model. Both frameworks support the representation of model elements in memory, but also deliver methods for storing data in a backend store or a database. This is especially important regarding the scalability of the backend. As an advantage, dotNetRDF supports more than six different storage providers. Finally, both libraries deliver a SPARQL request engine and with this fulfill the functional requirements.

For further examination of the performance, an efficiency test has been conducted. An implementation with both libraries has been evaluated with RDF/XML files with the file sizes 1 MB, 5 MB, 15 MB and 150 MB (see fig. 5.2).

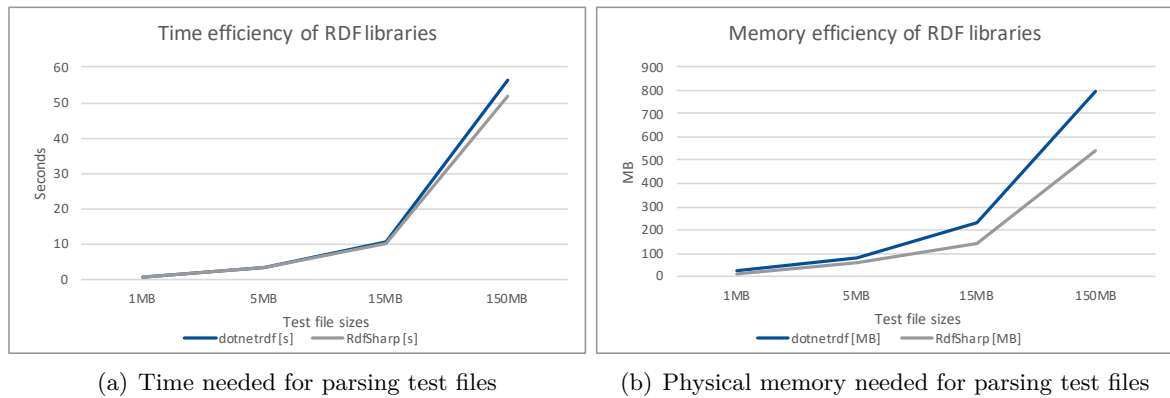


Figure 5.2.: Efficiency of RDF libraries

In fig. 5.2(a), the time needed by the libraries to parse the given files into triple graphs has been visualized. In the range of 1 MB to 15 MB, no significant differences can be seen between both graphs and the required time is nearly the same. At 150 MB dotNetRDF needs approximately 4.4 seconds longer to parse the file than RDFSharp, which effectively is about 8 % difference.

In fig. 5.2(b), the memory needed by the libraries to parse the given files into triple graphs has been visualized. The difference between both libraries can already be seen in the small range of file sizes. In the arithmetic mean, the dotNetRDF library needs around 38 % more physical memory. Reason for this could be the method of how the triplets are inserted into the graph. Nonetheless, there are several other reasons causing memory needs.

Concludingly, RDFSharp is more efficient than dotNetRDF, especially with large files. However, the file sizes within the ICDD will probably not exceed 15 MB per file (225.000 triples). Despite the efficiency, dotNetRDF offers more functionalities, an easy implementation, a large documentation, and an evaluated compatibility. Because of this, dotNetRDF is used for the implementation of this software application.

### 5.1.3. IFC Library

As it is stated in the functional requirements, the software application needs to provide an IFC visualization to fulfill the requirements 011 and 012. Therefore, the following aspects apply, when a library for IFC visualization is examined. The library must import files in the IFC data format. It must read and visualize the 3D geometry as well as the attributes of the related objects and at least the GUID of the objects. For interactivity with the models, the library must support navigation within the model viewer using the mouse and/or keyboard. For the visualization of link elements, the library must support highlighting of elements according to a specific GUID and the selection of elements with a mouse click to get the related links for a specific element. Moreover, the adaptability to all other components must be ensured.

The xBIM Toolkit is a popular tool for visualizing and analyzing IFC models within the .NET Framework developed by Lockley et al. (2017) at Northumbria University. It has been developed since 2007 and implements both standards IFC2x3 and IFC4. The framework mainly consists of two core libraries the xBIM Essentials and xBIM Geometry which are written in C# and C++. On the one hand, the toolkit provides support for retrieving alphanumeric information from any object and represents the object model of the IFC Schema. On the other hand, it delivers a geometric representation of the complete model using the WebGL based xViewer.

By means of the JavaScript library WebGL, the geometric data from the IFC file can be rendered as interactive 3D graphics within a browser. To utilize the geometry for web browsers, it has to be extracted and transformed into a binary format that has been introduced in the Geometry library as **wexBIM**. Since the conversion of the geometry requires memory resources, this is outsourced to a separate worker to maintain the response time of the web application. The worker runs in a separate instance on the web server and can be migrated on any other web server. The requests to the worker can be performed as HTTP POST methods including the original IFC file as form data. It responds with a binary file stream containing the converted geometry file which then can be further processed.

The geometry stream is transferred to the web application and stored in the file system. The web application creates the xViewer instance and loads the geometry file into an xViewer

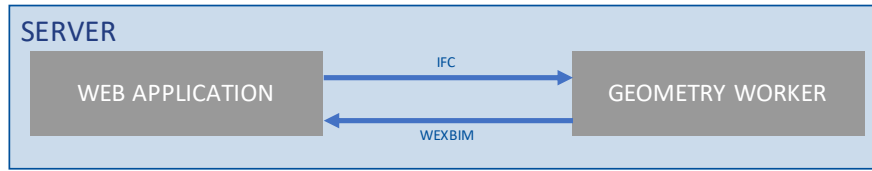


Figure 5.3.: Outsourcing the geometry conversion into separate worker instance

canvas. The user can handle and navigate within the loaded model using the mouse. Methods for camera interactions, clipping, click events, object picking, hiding objects, and highlighting objects are implemented and documented by Lockley et al. (2016). The connection with the model information can be made using the xBrowser library. The information can be queried with the picked objects presented in the xViewer and their ID. In summary, the xBIM Toolkit meets the defined requirements for an IFC library and is used for visualization.

## 5.2. Import and Export Components

### 5.2.1. Data Handling

The first step after an ICDD file is processed onto the server is the decompression and storage into a file repository. The data model holds the information of the file localization. After the file-based operations are completed, the inherent metadata files need to be imported and exported. The RDF import and export workflows follow the basic principles according to fig. 5.4.

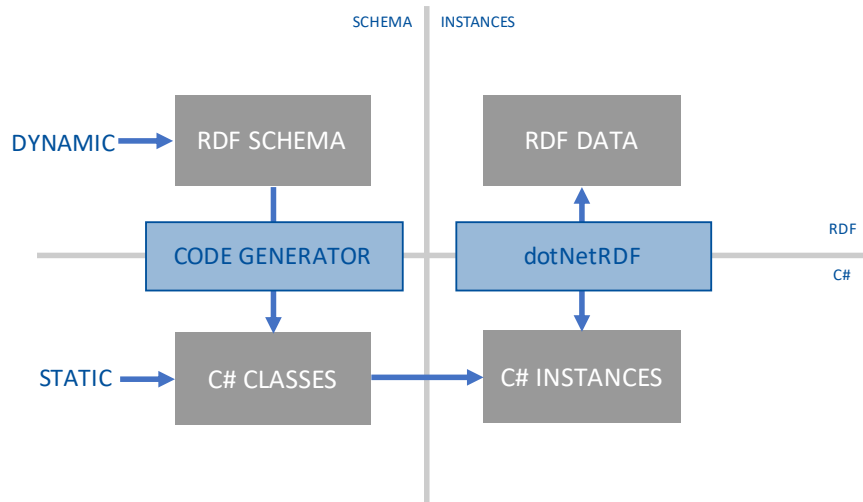


Figure 5.4.: Integrating RDF and object-oriented programming as stated by Völkel (2005)

### RDF-to-C#

The transformation of instances between the RDF graph and the corresponding C# instances can be done at runtime using the introduced dotNetRDF library (fig. 5.4). Since the ontologies for the static content are standardized, the data model has been implemented into the software application directly. This has the advantage, that the C# classes from the standardized RDF schema do not have to be generated at runtime what leads to time and memory savings.

However, a disadvantage is the server-side maintenance effort due to changes in the standard. The imported RDF data can directly be instantiated from the respective classes. Every class in the data model can be instantiated either from the triple graph constructor or from a constructor, which creates an instance that is not yet existing in the triple graph. At runtime of the process, the RDF data is kept available as a triple graph and stored in memory within the container data structure.

An overview of the data model can be found in appendix A.1. Generally, the data model is developed according to the classes that are defined by the standardized ontologies from the ISO 21597. There are three namespaces according to the three ontologies for container, linkset, and dynamic semantics. The `DynamicSemantics` namespace has two sub-namespaces for properties and relations. In addition to this, a general namespace `ContainerModel` provides the `IcddContainer` class, that is the basis for every imported container and provides the repository and the metadata. The `IcddManagerIO` class supports the application with functionalities for the transformation between RDF and C#. For instance, it includes methods for getting namespace URIs, the GUIDs of a single instance or a set of instances depending on a certain type, or updating an attribute of an instance. Moreover, the `ContainerModel` namespace contains the class `IcddObject` which is base class for every object read from an RDF file. It includes the `ID` attribute which is important for uniquely identifying an RDF instance. The `ID` attribute must be set when creating a new instance and cannot be changed afterward. The `IcddObject` class provides information about the author and modifier as well as the creation and modification dates.

The remaining class within this namespace is the `IcddUserDefinedOntologies` class which serves as the entry point for UDOs and provides the respective RDF triple graph from which the Code Generator (see fig. 5.4) generates the classes at runtime. In the case of a present UDO, there are no instances created until all classes are generated at runtime. The ontology files serve as a blueprint for the C# classes. The code generation is focused in ch. 5.2.3. In addition to these classes, an interface `IVersioning` provides the required methods for versioning of objects and indemnifies that the respective methods return valuable information.

### C#-to-RDF

Conversely, the C# classes offer methods to transfer the actual instance into RDF triples and use the RDF library to write changes into the file. Literal properties of instances can be directly manipulated using the implemented set methods of the respective properties. These methods refer to the `IcddManagerIO` class and apply the properties using the `SetInstanceAttribute` method (see app. A.2) into the RDF graph. Object properties of instances first can either be edited or replaced. To edit an object property, the respective object has to be addressed with its GUID and can be changed according to the procedure above. An object property can be replaced by a new object which has first be instantiated from the constructor. The constructor creates the triples that are necessary for the description of this instance. The object has to be added to the property as an URI node which can again be done using the `SetInstanceAttribute` method.

Every operation on the RDF graph has to be done using the `Retract` and the `Assert` methods on the RDF graph. First, the triple which defines the property has to be retracted from the graph. Then a new triple with the instance as a subject node, the property as a predicate node and the new object's URI as an object node has to be created and asserted to the graph. The changes are only applied to the RDF graph. As long as the container is not saved manually, the modifications are not persistent in the container. When the container is going to be saved,



the modified RDF graphs are written into the RDF file using the `RdfXmlWriter` from the `dotNetRDF` framework.

During the implementation, the characteristics of the OWL modeling have to be mentioned. There are structures that cannot be transferred one-to-one into the object-oriented data model. For example, OWL supports the inheritance of properties with the `subPropertyOf` attribute (see fig. 3.8) which cannot be directly implemented within the object-oriented programming language `C#`. More examples of differences between both can be found at Völkel (2005).

### 5.2.2. Session Handling

A static `SessionManager` class handles all sessions for the web application and the Web API. The manager provides the methods `StartSession`, `CloseSession`, and `GetSession`. Each uploaded file or POST upload request calls the `StartSession` method which creates a new instance of the class `Session`. This instance contains a GUID, a timestamp for the last activity, and the connected container data with the validation results. The timestamp is updated on every access of the session data. The `IsAlive` method proves whether the last activity is newer than 30 minutes. In case the session is not alive, the `Session` object is disposed and unregistered from the `SessionManager`.

The `Session` class implements the `IDisposable` interface and allows to delete the session and all related container data in memory and on the disk. If the session is called when it is expired, the dispose method is called and all container data is deleted. Additionally, a background task proves all user folders on the server and deletes created user downloads and log files from the server that were not accessed within the last 30 minutes. The sessions can only be identified by the GUID. This is why every API call has to include the GUID within its request to get access to the correct session. Each page refresh of the web application including updates of the data inside the user interface cause a session request so that the activity timestamp is updated frequently.

### 5.2.3. Code Generation

The code generation is programmatically realized for the example of user-defined links. As already stated in Chapter 3.2, the link types of the standard can be extended using the `subClass` property of OWL. According to fig. 5.4, the Container ontology is transferred into a `C# OntologyGraph` from the `dotNetRDF OntologyAPI`. This graph differs from the triple graph because it provides an ontology-centric view of the RDF file. It is an extension of the triple graph and provides support for resources, classes, properties, and individuals of an ontology. To create the respective subclasses of the link types, the `OntologyApi` offers the function to get the subclasses of a certain class from the ontology. With this, the inheritance between classes in `C#` can be transferred and positioned into the static data model at runtime. Each link class can be created calling the implemented function `createLinkTypeFromOntologyGraph()` which proceeds the data from the ontology to the `CodeDom Generator` in order to create the dynamic data model. The method is supplemented with the parameters `OntologyClass`, `CSharpCodeProvider`, and the `CodeNamespace`, which provide the class information, the code generation methods, and the belonging namespace.

The inherited methods need to be implemented correctly during the code generation to prevent runtime errors. Constructors have to fill inherited fields so that a possible `NullReference-`

**Exception** can be avoided. It is important that the inherited classes are generated according to their inheritance hierarchy until the bottom class of the hereditary line is reached. The created classes are provided in a `Dynamic.ContainerModel.dll` file. When instantiating the container, the user-defined schemes have to be taken into account. Thus, a dictionary where all additional types are registered with the respective inheritance is iterated and proves whether there are instances for an additional type.

#### 5.2.4. Web API

The toolkit can be integrated into third-party desktop or web applications using the Web API. The user stories in ch. 4.1 describe the required API functions. A separate controller is integrated into the web application to handle the API requests. The URL routes are mapped within the controller file. For a first evaluation of the functionality, the requests in table 5.1 are implemented.

Table 5.1.: Excerpt from the Web API definition

|                                     |  |   |   |  |
|-------------------------------------|--|---|---|--|
| /container                          | BadRequest   | Upload container file and start web session         | BadRequest                                  | BadRequest                                 |
| /container/{GUID}                   | returns a compressed container file                  | BadRequest  | BadRequest                                  | Delete container file and stop web session |
| /container/{GUID}/Results           | returns a list of JSON serialized validation results | BadRequest  | BadRequest                                  | BadRequest                                 |
| /container/{GUID}/Documents         | returns a list of JSON serialized document metadata  | uploads a new file with metadata into the container | BadRequest                                  | Deletes all files from the container       |
| /container/{GUID}/Documents/{docID} | returns document file and metadata                   | BadRequest  | Updates the file and metadata of a document | Deletes the document from the container    |

The requests are separated into the HTTP methods `GET`, `POST`, `PUT`, and `DELETE` so that a logical context is set for each request and the routes can be kept short. These do not require authentication but have to provide the session GUID to connect the user to the correct session and provide the container data. The session is started when posting a container file to the `/container` resource and the request return the GUID for further operations.

Generally, the body of the request as well as the return statements can contain form data which means literal variables and file binaries. In the most cases, literal variables are JSON-serialized

objects from the `ContainerModel` namespace. These objects are serialized on the server-side and can be deserialized at client-side when using the `IcddToolkitLibrary`. The client can work with the objects as long as there is no transfer of RDF. The changes have to be requested using the API. The consumption of this API requires an error management. For simplicity, the requests only return the HTTP responses 200 OK and 400 Bad Request. The `Bad Request` error is provoked by default when a resource could not be found. Additionally, `Bad Request` is also returned when a web session is no longer available or a request does not match the required pattern, e.g. an invalid file type is uploaded.

### 5.2.5. Evaluation

The evaluation of the implementation is done in three steps. Firstly, only the data model for part 1 of the standard has been implemented. The standard delivers exemplary ICDD files for parts 1 and 2. The exemplary files are provided stepwise what also allows a step by step evaluation. After the first part has been implemented successfully, the second part is developed and evaluated. Both implementations are concurrent with the development of the web application which visualizes the resulting implementation of the data model. Besides, most of the evaluation can be done using the built-in debugger. For debug reasons, the application exports the complete triple graphs into text files to allow a perpetual view on the triple graphs.

As a result of the evaluation, the implementation has been improved in some important aspects. The `IcddObject` class has been optimized due to the integration of both parts. For interactions with the different physical RDF files, each instance that is inherited from the `IcddObject` has a read-only attribute linking to the graph instance from which it was initially instantiated. In addition, the class `CtContainerDescription` has been extended with a dictionary containing all `CtParty` instances that are defined in the RDF Graph regardless of whether they were referenced in the container metadata or not. This allows the future user to create new participants and assign them to an object at a later step.

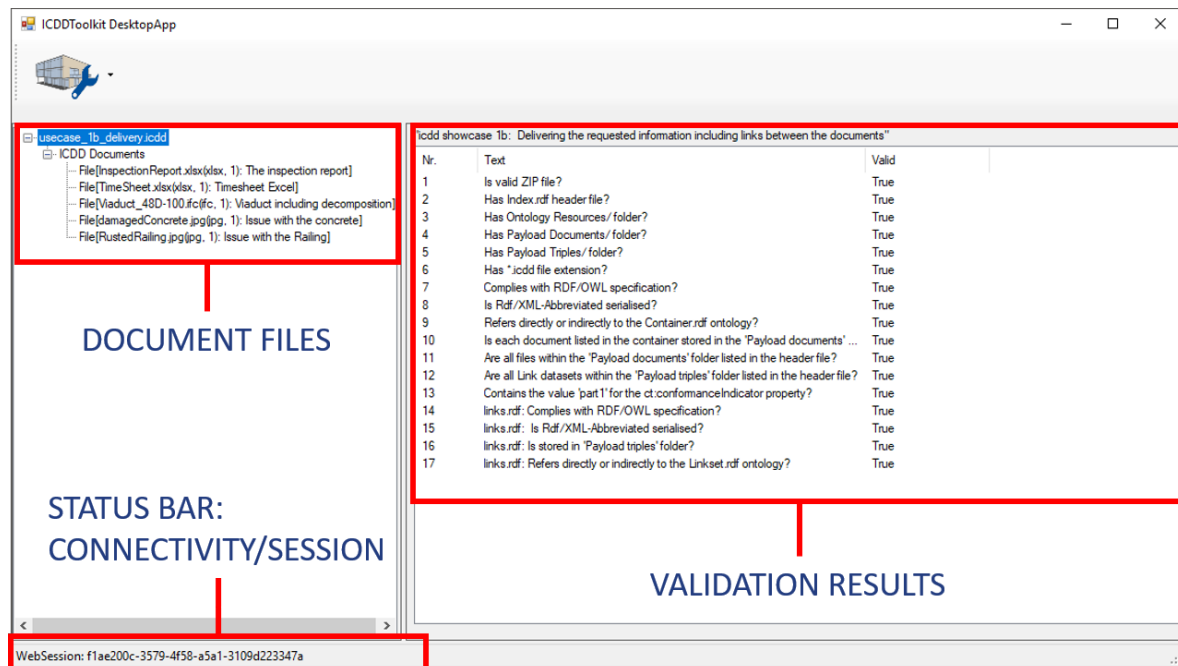


Figure 5.5.: Desktop application for Web API evaluation

In a third step, the implementation of the Web API is evaluated using a desktop application (see fig. 5.5). The application uses an HTTP client and the URI of the API to upload a file from the local file system to the server. As a result, the application gets the web session ID from the server. Asynchronous API calls using the session information and the HTTP client provide validation results and document information.

### 5.3. Validation Components

The validation consists of two parts, the validation according to the conformance criteria given in the standard hereafter referred to as conformance validation and the user-defined logical consistency validation.

#### 5.3.1. Conformance Validation

The application is designed so that each uploaded container must pass the conformance validator first because a not conform container might not be readable. This validator is instantiated from an ICDD file which is then extracted and copied to a local folder. According to the number of requests on the RDF files, it evolved during the implementation that it is not appropriate to handle files or the container within the memory. The implementation of the validator relies on the data structure of the developed classes `IcddValidator` and the `IcddValidationResult`. The validation results class is serializable and has attributes for the label of the criterion, the respective expected value, and an examined value.

The validation of a container using the `validate()` method returns a collection of validation results where each result can be determined comparing the expected and examined value. During the validation process, each validation result is written into a log file, that can be downloaded after the process has finished. Moreover, while the ICDD file is in inspection, the data model is created from the physical file so that not only file-based but also metadata-based criteria can be validated. The validator proves, whether there are versions of the container, linkset, or dynamic semantics ontologies included in the file or need to be referenced from the standard's web resource. If one of them is not included, the validator gets the files from the server and includes it in the container for further operations.

After the validation has performed, the `isValid()` method returns whether all results in the result collection are valid. Only then, an object of the `IcddContainer` class is returned by the `getValidContainer()` method. For purpose of the Web API, the validation results have been designed serializable using JSON serialization to provide a standardized and readable schema.

#### 5.3.2. Consistency Validation

As the application allows visualization and manipulation of links, these need to be validated according to user-selected logical consistency rules. Therefore, in this implementation mathematical definitions of binary relations are used to create consistency rules. These rules then can be applied to any Linkset that contains instances of the class `BinaryLink` or links from a subclass of it.

**Example:** Let us assume that document  $A$  is an IFC-model with a set of objects  $a$  and document  $B$  is a BOQ-model with a set of work items  $b$ . The relation  $R$  is bitotal when the following equation is satisfied:

$$(\forall a \in A \exists b \in B : (a, b) \in R) \wedge (\forall b \in B \exists a \in A : (a, b) \in R) \quad (5.1)$$

This can be applied to the given example. For every building object, a link exists to at least one work item  $b$ . In the same way, every work item  $b$  is linked at least to one building object  $a$ . In words, this means that every built object has to be calculated within the bill of quantities and every work item has to be accounted with a specific dimension of a geometric object. This issue can be validated by means of a validation test.

The validation is done using the imported data from a container. The user can specify the linkset and the relational property and starts the process. Entities that do not fulfill the property rule are returned by the process and can be inspected and repaired afterward. Besides the bitotal relation property, biunique and, respectively right and left unique as well as right and left total properties are implemented for linkset validation. To make results available for the user interface, the abstract class `IcddLinkValidation` is introduced. Every derived class needs to implement the function `GetKeyValueResults()` which provides a dictionary with results after an instance of the class has been created from the constructor.

Furthermore, the program implementation provides a check of link types within a linkset and highlights the Links that logically do not belong in the set of links. Therefore, either the user can select a single link type or the program determines a link type on basis of the most used type. For the determination, the linkset is partitioned by the specific link types into a dictionary. The decisive link type is determined from the dictionary selecting the link type with the most instances. After that, the respective method returns a set of links that logically do not belong to the set.

### 5.3.3. Evaluation

While evaluating the implementation, it quickly became evident that the functions for consistency validation in chapter 5.3.2 are not able to record the total set of elements within a document. This is because not all of the entities from the documents are mapped into the metadata storage of the container using the respective identifiers. The document identifiers are connected to the link elements of a link instead of being connected solely with the document itself. As a consequence, the linksets only contain the document identifiers that are addressed within a link in the linkset (see fig. 5.6). To sum it up, the standard has not been developed to maintain all document identifiers within the metadata but the ones that are addressed by a linkset. On the one hand, this concept helps to avoid redundant data. On the other hand, developers need to put more effort into the implementation as the data for validation has to be parsed at runtime.

As a quick optimization of the underlying implementation, a data dictionary structure containing all identifiers from the container's linksets is developed to enable a meaningful validation (see fig. 5.6 bottom box). With this dictionary of links per document, identifiers that are not contained in the linkset but have been instantiated in the container can also be used for validation. This is not the qualitative best solution for the issue but enables the possibility to perform a linkset validation.

This issue can be finally resolved by reading all entity instances from a document. The effort to implement a document-specific reader for identifiers is very high, as every relevant document

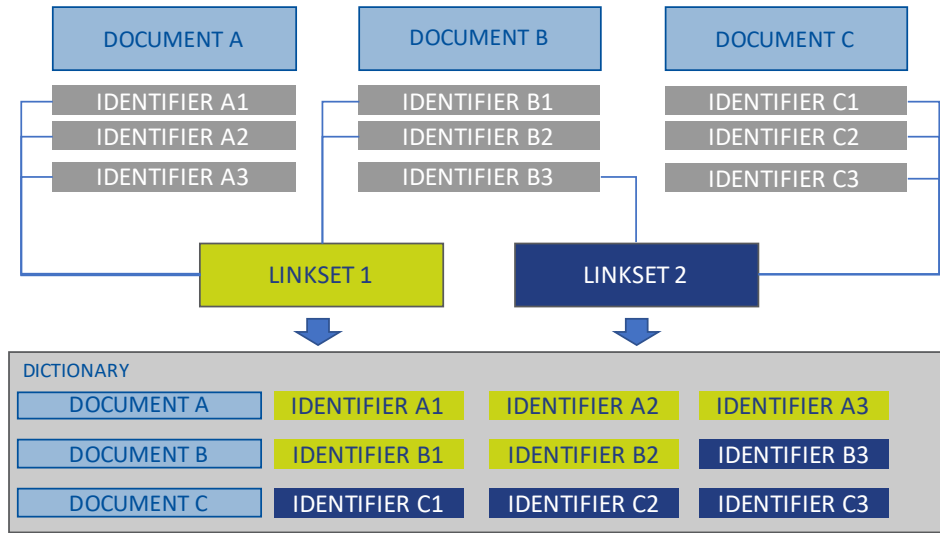


Figure 5.6.: Mixed origins of identifiers

type such as IFC models needs to be read and interpret into document identifiers. Furthermore, identifiers can also be available in different types for instance as query-based identifiers and the respective document parsers must handle these queries.

## 5.4. Visualization Components

### 5.4.1. Concept and Implementation

The GUI and methods of visualizing container contents account for a large part of the implementation work. Basically, the design of the ICDD Explorer consists of four major parts according to fig. 5.7. The first one is the Explorer which is a tree view showing a combination of files and metadata. Both can be discerned with the tree icons. On the one hand, the Windows-like file system icons signalize a file that is physical existent in the container. On the other, the black colored icons denote metadata information. All nodes of the tree view serve either as informative content or to navigate through the container. For instance, the index.rdf metadata can be read directly from the tree view. Into the bargain, the files and some metadata can be selected to retrieve more information on the respective object. The tree view is based on the open source JavaScript library jstree.

As the figure shows, there are three more parts whose content depends on the selection of the explorer. The second component is the Document Viewer. The form of representation of the document depends on the document type. In the example, an IFC file is selected and the viewer shows a geometric representation of the model using the xViewer from the library introduced in ch. 5.1.3. The model can be inspected using the mouse and the navigation cube. Besides IFC models, the viewer can visualize image files using simple HTML code but other file types are not supported yet for visualization.

The third component is the Extended Document Viewer which is filled with additional information on the selected document. In case of an IFC file, it provides a table listing all elements from the model. The elements are read from the file using the xBim.Essentials library and stored in a list of objects. These are characterized using the simplified `IfcObject` class which only provides the name, the type, the GUID, and a material for the single objects. For other

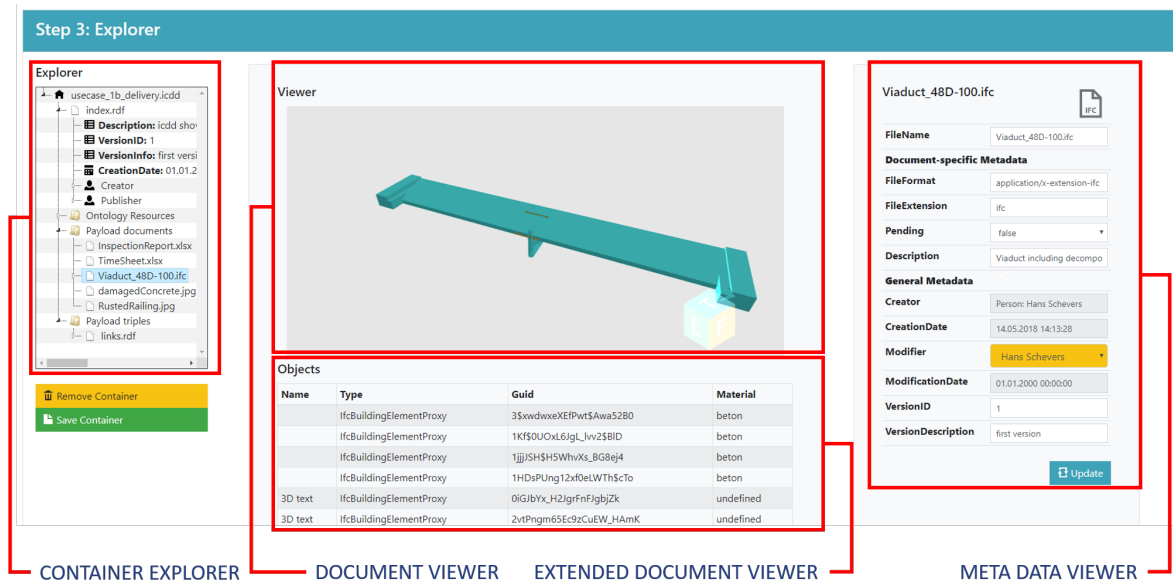


Figure 5.7.: Design of the Graphical User Interface

document types, it gives a cross-linkset overview of the links from other documents. In this stage of development, the explorer is not able to select elements from the 3D model and use the selection for the creation of links. This is where further implementations can be realized to achieve an interaction with the geometric model.

Finally, the fourth component is the metadata viewer. This section provides information on general metadata extracted from the container as well as object-specific respectively document-specific metadata. Herein, the literal properties of the object are presented directly while the object properties are issued as representative strings using the `ToString()` method. The viewer allows making changes on both literal and object properties. In the latter case, these can be selected from the stock of appropriate objects inside the container. For instance, the yellow dropdown box inside the viewer enables the user to select a certain modifier. The update button triggers the corresponding methods for the `C#` data model which pass on the respective changes to the RDF graph. To write the changes into the physical files, the save container button executes the parser for the RDF files.

#### 5.4.2. Link Maintenance

Particular importance during the implementation was also assigned to the representation of links and their manipulation. Basically, the explorer offers an ability to view the details of a link, delete it, or add a new link element to a linkset. The GUI in fig. 5.8 provides a viewer for the linkset but also for each link inside of them. This enables the user to easily switch between a linkset and the linked documents from the link details. The extended document viewer also displays the links as well as the entities that have been defined across all linksets for the respective document. Altogether it is possible to get a quick impression about the composition of the links.

In the next step, the functions for deleting and creating links have been implemented in the user interface. Due to the implementation effort, in the first version of the tool, the creation of links only provides the standard `LsLink` type. Links can be created by selecting two documents from the container and adding an optional identifier to it. The link can then be added to a

selected linkset and will be integrated into the data model.





| Viewer  |  |   |   |
|---|--|---|---|
| Type  | LinkElements   |   |   |
| Link  |  InspectionReport.xlsx (Identifier: [4])                    |   |   |
|   |  Viaduct_48D-100.ifc (Identifier: [20dnvrgv16XurcDISZ1qI3]) |  |  |
| Link  |  RustedRailing.jpg  |   |   |
|   |  InspectionReport.xlsx (Identifier: [2])                    |  |  |
|   |  Viaduct_48D-100.ifc (Identifier: [1D_dzaxZf4_QcddQY3uvjg]) |   |   |
| Link  |  damagedConcrete.jpg (Identifier: [3])                      |   |   |
|   |  InspectionReport.xlsx (Identifier: [3])                    |  |  |
|   |  Viaduct_48D-100.ifc (Identifier: [2ptPWWWHjCCfWE1B3yaT2A]) |   |   |
|  |  |   |   |

Figure 5.8.: Link Visualization and Manipulation

### 5.4.3. Evaluation

During the stepwise evaluation, errors in the implementation were discovered. Firstly, the model viewer had problems to open the geometry file. This error is due to the Cross-Origin Resource Sharing (CORS) policy of latest web browser that is used to prevent a website from loading resources from another server. In this case, the CORS error was caused by the geometry file that has been requested from the IfcGeometryWorker instance. To avoid this, during the HTTP request the file has first been migrated to the correct server and then processed to the client-side viewer.

A different error has occurred when adding links to a linkset. When creating a link between two documents where one has an identifier that is already existing in the data model, the link element is instantiated with the same parameters. This behavior could lead to certain errors because the identifier is listed more than once and, for instance, the link validation cannot get appropriate results. Furthermore, when viewing a document, it has several instances for the identical element. That is the reason why it is important to prevent redundant data storage. As a solution, the link creation routine has been modified so that it is tested whether the element is already existent in combination with this document.

After adding links, deleting links is also a challenge. Delete elements from an RDF file can always result in inconsistencies so that a larger change in the code has been conducted. On the one hand, the `IcddObject` class gets a private `Delete()` method which performs the deletion process on the RDF level. On the other hand, the class `IcddObject` defines an abstract method `SafeDelete()` which allows checking additional relations between objects. For example, the safe delete method of the class `LsLink` indemnifies that every link element inside it is also safe deleted. These again can be deleted safely with a check whether the element is used in other links inside of the container. In total, this structure contributes significantly to the consistency of the data sets.

These two issues lead to the more general question how the standard handles the factual situation whether link elements are unique inside the container or can be referenced if the document and identifier are identical.



## 6. Evaluation

This chapter deals with the accomplishment of an evaluation regarding the overall project and the structure of the implementation. Basically, the aim of the evaluation is to determine to what extent the presented application fulfills the purpose of validating files in the ICDD file type. As seen in fig. 4.1, evaluation and testing is part of the general software engineering process model. Together with development, it forms a fundamental step towards the practical application of a software product. The chapter is partitioned into three sections, in which the first section defines the setup and test data, the second records the conduction of the evaluation and the last formulates the results.

### 6.1. Setup

An example project in the format of the BIM-LV-Container is used so that the evaluation has practical relevance. This project represents the relationship between a building model and a BOQ. Additionally, a simple project schedule is appended to the project data to get a complete 4D and 5D planning. The data setup must be converted to the standardized ICDD format which can be done using software tools like RDF editors to manually edit the respective files. This workaround is highly prone to error when interrelationships between elements have to be established. The first creation of the ICDD container failed due to duplicate element IDs. Having this in mind, two container setups have been developed. The first one is a minimal example of the container which has the three files and two linksets, each containing three binary links. The second one is conceptualized for a stress test of the implementation and contains the same files but has one small and one larger linkset with 400 links included. The links were extracted from the `Links.xml` file of the BIM-LV-Container and programmatically added to the linkset using the XML formulation of the RDF triples.

The evaluation is done in two steps in which the first step is the gradual examination of the functional software requirements defined in ch. 4.1. The assessment of non-functional requirements is the second step and focuses on the quality of the implementation and its performance. The results are summarized and discussed afterward.

### 6.2. Results

Inside the web application, the overall procedure is divided into three steps: Upload, Validate and Explore. The web application does not let the process step Validate be skipped to ensure data consistency for the Explore step. Having the test data converted into an ICDD file, the data can be uploaded as `BLC.icdd` within the form according to the requirement 001 (see fig. 6.1).

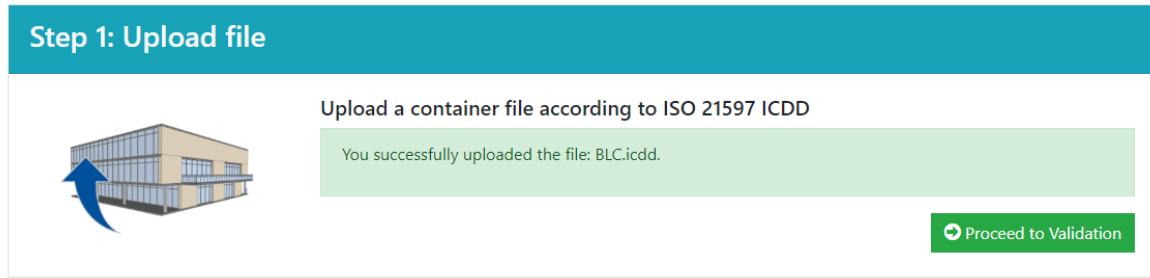


Figure 6.1.: Upload of a container file

### 6.2.1. Input and Validation

After the file has been uploaded successfully, the user can proceed to the validation. The conformance validation automatically starts when a user opens the validation page and the validation view is generated from the file. This view (see fig. 6.2) shows a validation overview section on the left side, which provides quick information about the status of the validation e.g. failure or success. Furthermore, it shows the number of documents, linksets, and UDO contained within this file. After the file has been uploaded, it is available for the validator and explorer as long as the user is inactive for longer than 30 minutes or removes the file from the server using the function from the menu on the left side. Thus, the requirements 002, 003, and 004 are met.

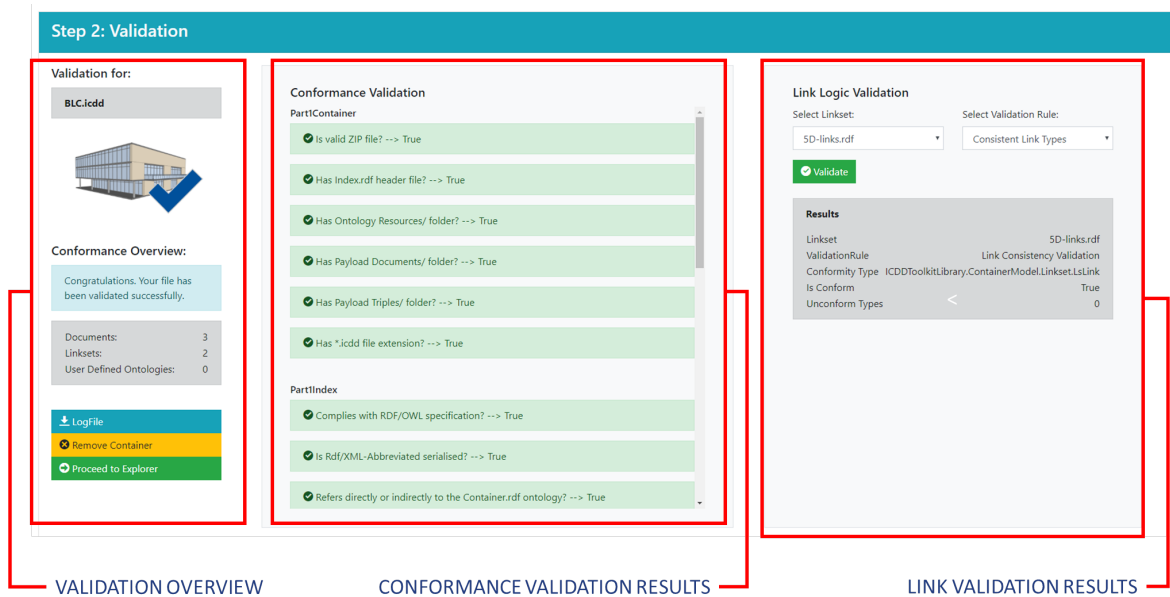


Figure 6.2.: Validation of the exemplary container file

The results of the conformance validation are presented inside the central view element and are categorized according to the standard. Beyond that, a more detailed validation log including the results can be downloaded using the respective menu entry on the left. The visual processing and the download of the results accomplish the requirements 007 and 008. With requirement 009, a method to directly correct the failing aspects has not been implemented because there have been fatal errors when parsing misformulated files RDF/XML files. In order to offer an assistance for the validation, the log file was extended so that the respective XML exceptions were also issued within the logs and exactly identify the positions in the files that caused an error. This has been especially helpful when manually preparing the ICDD file so that syntax errors can easily be removed.

The test data has been validated according to the conformance guidelines defined by the standard and is now prepared for further usage. In this evaluation, a link consistency validation has been conducted to identify links that deviate from the prevailing link type according to the requirement 017. This requirement demands that users of the toolkit can validate linksets with user-defined rules. To facilitate implementation, a set of validation rules has been developed and made available to the user for selection. These validations can be found in the right box which is called Link Logic Validation. As already stated in section 5.3, most of the rules apply to binary relations. Therefore, a positive validation result can only be achieved when every link is instantiated from a type within the **BinaryLink** inheritance. The links in the example have been defined in the general link type and thus cause a falsification although they were defined as binary one-to-one links. In a further development, the area of link validation should be considered more closely, in particular also towards the mixed entity origin issues that have been stated in section 5.3 within the evaluation.

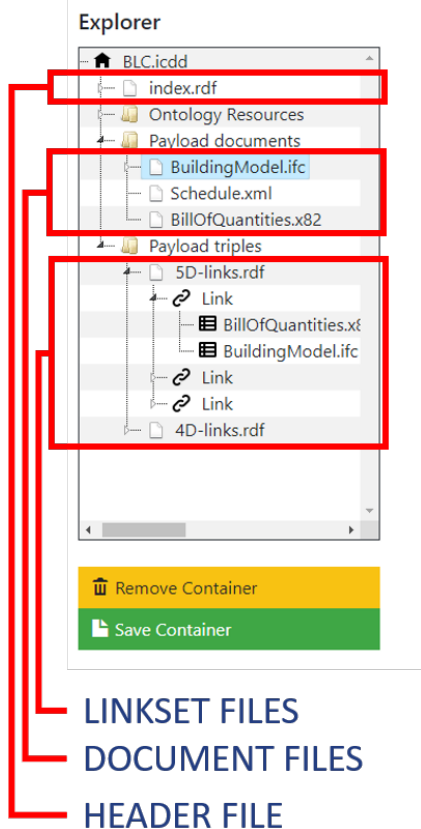


Figure 6.3.: Exploring the validated container file example

After the exemplary file has been validated successfully, the content of it can be viewed in the explorer. In addition to those in the validator, the overview of the container file shows the number of contained metadata entries, i.e. the link, link elements and all contained data on metadata entries created from the **Party** class. The tree view in fig. 6.3 shows the three files in the payload documents folder as expected. The two different linksets with the overall amount of six links were read successfully. The structural representation can be achieved using the tree view with additional selected metadata information (see ch. 5.4) and with this fulfills the functional requirement 010. The tree view is the central navigation element within the container and allows to control the viewer for linksets and documents. If-objects are extracted from the IFC documents to enrich the user interface with further information and to allow a comparison between metadata and physical data. These objects are added without hierarchy into the tree view for the respective documents. During the evaluation, it became obvious, however, that this can only be reasonably represented as long as the models contain only a few elements.

### 6.2.2. Visualization

The visualization of payload documents is defined in requirement 011. Basically, the focus in this thesis regarding visualization lies on the 3D geometry of building models. Nevertheless, the textual visualization is available for all documents through the contained entities evaluating the respective identifiers. Together with the original documents and the identifier definition the user can understand what information within the documents is addressed through the links.

Fig. 6.4 shows the visualization of the example building model. On the top of the viewer, the geometric model is displayed. It can be rotated using the mouse drag event on the canvas, scaled scrolling the mouse wheel, or translated dragging the mouse wheel. Additionally, the navigation cube can be used to inspect the file from different predefined positions. Directly below the geometric model, the list of extracted IfcObjects has been generated. It is

## 6. Evaluation

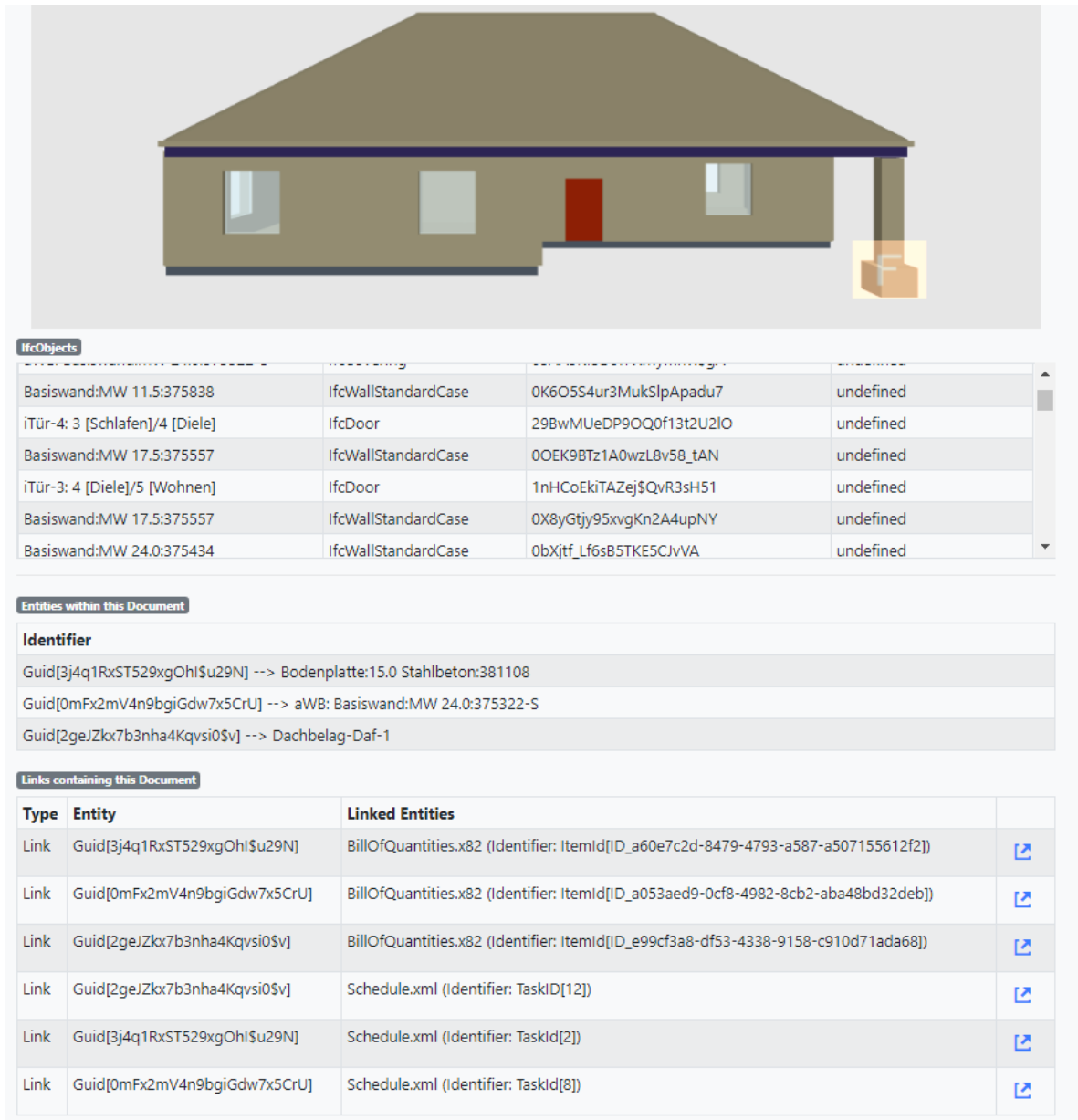


Figure 6.4.: Exploring the IFC model from the container example

noteworthy that the material is displayed as undefined for all components. Compared to the original document, indeed the material of some elements is not defined while the material for the `IfcWallStandardCase` with the GUID `0X8yGtjy95xvgKn2A4upNY` is defined as lime sandstone. This fact is not displayed in the table. In comparison to fig. 5.7, the objects in the test data are not defined as `IfcElementProxy` with a direct definition of the material but as `IfcWallStandardCase` in which the material is set in a material layer set. When implementing the IFC reader, this was not taken into consideration. Nonetheless, the entities evaluated from the identifiers inside of the links are mapped together with the document information in the below table via the GUID. Chapter 5.4 already pointed out the missing reference between the graphical and textual representation of the entities, which needs to be improved in the further development work. The last requirement in the visualization category (no. 012) deals with the representation of the elements inside a document that are concerned by a link. In order to develop a general medium that can be used for all document types, the tabular presentation method was selected for displaying the sum of links referencing this document.

### 6.2.3. Links

After the visualization category was completed with a link-related requirement, the evaluation is continued with the category link requirements. The first link requirement (no. 013) demands a representation of the links inside a linkset. The presentation of linksets was already introduced in chapter 5.4. However, the evaluation revealed an error that had not previously occurred, because only a single linkset was considered before. Whenever a link is added to any linkset, the routine checks whether the affected link element is already existent for the document. If this is true, the element was only referenced inside the RDF file. This was unproblematic as long as there was only one linkset. The references along two or more linksets are not supported by either the implemented data model or the standardized file format and finally lead to a reference to a non-existing element. It was decided for this implementation that a link element can be created once in every linkset to be consistent inside the single linksets but to constrain the redundancy of the whole structure, which will be discussed in the results.

Figure 6.5.: Adding a new link to the 4D-Links

Another requirement implemented in the application is the creation of new links in any linksets (no. 015). The program fulfills this with a further input form in which firstly a linkset and a type can be selected. Furthermore, two of the existing files in the container can be chosen for linking. The facultative identifier input fields can be provided with a free text or left empty. The corresponding link element is created from this information and the link is attached to the linkset. For requirement 015, the updating and deleting of objects, only the delete function was implemented into the application. A reason for this is the already mentioned function `SafeDelete()`, which ensures that no fragments and orphaned links are contained in an RDF file. A function to provide updates to link elements would have to implement the same checks, which is not appropriate due to redundancy and maintenance of the source code. This means, an update has to be performed in two steps: deleting a link and creating a new link with the desired properties.

The filter for existing links which has been defined for development in requirement 016 was realized in the internal code for the link validation. An implementation for the user interface was postponed to a new version due to the low priority.

### 6.2.4. Metadata

The metadata for the objects inside of the container can be read from the metadata viewer according to chapter 5.4. The viewer allows to update literal and also object properties. It is not possible to completely delete a property but to overwrite it either with an empty string or with an empty object. Some metadata fields are locked for editing because they must not be edited, such as the `Creator` or the `CreationDate`. Altogether, the metadata requirements 018, 019, and 020 are implemented as defined.

In this evaluation, several processes were carried out and the saved files were reloaded into the web application without any information being lost. In addition, the individual process steps, especially in metadata processing, were saved in the container file, which is unpacked after saving and the RDF files are viewed with the Protégé ontology editor. This was done to ensure a compliant and consistent data format even after the writing process into the RDF files. This also satisfies issue 005 of the file operation requirements.

### 6.2.5. Web API

Another seven requirements were defined for the implementation of the Web API. Since the priority of developing the Web API was lower than the priority of the web application, only the functions mentioned in ch. 5.2.4 Web API were developed. This means that not all requirements can be met, but a demonstration of the interface is possible.

### 6.2.6. Quality

Basically, the quality requirements are satisfied by the current implementation. To evaluate the availability or rather the accessibility, the application was migrated onto a web server instead of running on a local machine. This required some adjustments, especially to file paths and the connectivity between the web application and the IfcWorker. On the server environment, the execution of C++ class libraries within the IfcWorker application caused an unexpected error due to an incompatible server configuration. However, the application is running on the server and a connectivity test between the local desktop application and the remote web application was successful.

The usability of the web application was focused in the chapter 5.4 Visualization Components. All important functions are highlighted in color and arranged in a user-friendly way. However, an objective result on usability can only be determined in a user survey. As maintainability is always an important topic for application distribution, the web application is a very maintainable way to provide updated versions of a software to the end user. The Web API, on the other hand, should not be updated as users are implementing the specific version of the API. The API is not finally defined and therefore has no final documentation what should be implemented in the next version of the application. Finally, all data security requirements have been satisfied by the implementation of the session manager. The complete user directory is deleted after the session is closed.

### 6.2.7. Performance

In addition to the functional and quality requirements, also the performance requirements are evaluated. Therefore, the small and the large exemplary files were uploaded to the server. After that, the actions of validation, opening the explorer, modifying metadata of a document and saving the container file were contemplated. The measurements refer to the respective methods that are executed on the server-side and were averaged over two sequences each. The client-side time and memory efficiency are not considered in this evaluation. The graph 6.6

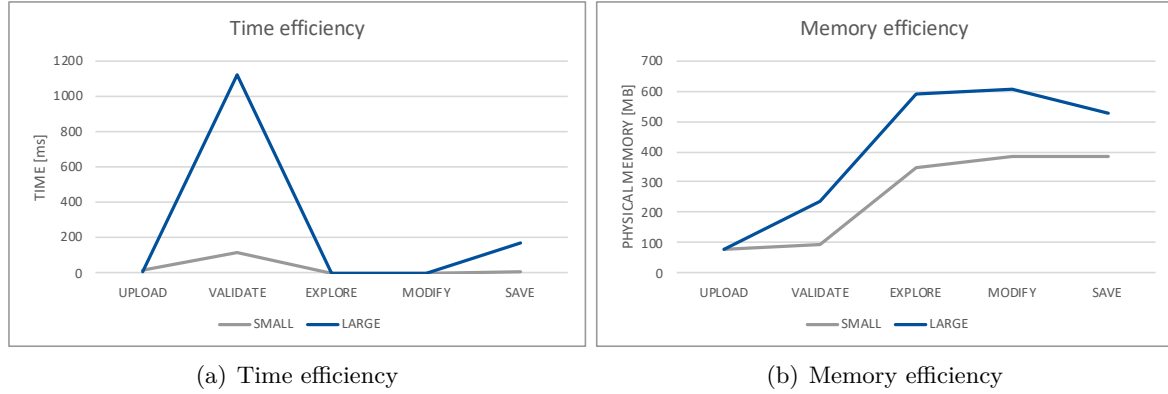


Figure 6.6.: Efficiency of the developed web application according to performed operations

a) shows that the time usage started climbing after upload, peaking at the validation, and flattened out at a level of nearly zero milliseconds for opening the explorer and modifying a document. For the large file, about 200 milliseconds are used for the saving process. These peaks indicate the operations where the RDF graphs are read from the files or written into the files. An increase of factor 100 in entries results in an increase of factor 10 in time use. Within the validation method, the complete container file is instantiated and moved into the memory. A positive aspect is that the explorer does not need the time and can be opened directly. The graph does not show that the explorer still needs to be rendered for all entries on the client side, which in turn leads to a slowdown and loading times.

According to the graph in fig. 6.6 b), the demand for memory rose sharply between the upload of the file and the opening of the explorer before it stabilizes at an approximately constant level. All in all, both measurements meet the expectations and requirements. Nevertheless, for example, the overall time efficiency can be improved by asynchronous method calls from the client-side.

## 6.3. Discussion

In the synopsis of requirements and the evaluation performed, the state of implementation reflects an advanced application. Nonetheless, this application is not yet ready for practical use. During implementation and evaluation improvements were recorded which are summarized for further implementation. Firstly, the validation process quickly raises the question of what happens after unsuccessful validation. The results are recorded into a log file. However, a more comfortable solution is the implementation of an auto-correction function. In addition, an extended consistency or completeness check can be carried out during validation, which compares the entities from the documents with those from the link sets. To do this, additional components for reading different file types would have to be implemented, which

## 6. Evaluation

can also display additional file types in the viewer. Finally, a converter can be implemented that transfers container files from the MMC format to the ICDD format. To achieve a high reusability of the framework, the Web API needs to be refined and documented to foster the easy access of container files. It was also noticeable when inspecting RDF files before import and after export that the structure of entries inside the file has changed. The structure of the original data from the standard's annex had a nested pattern whereas the exported data from the application had a sequential pattern. However, this did not affect the quality of imports but simply had an impact on readability.

To address the discussion about the definition of link elements and their use within linksets, the question arises whether the combination of document and identifier describes a unique object. Does every identical combination of document and identifier have to be instantiated again? The standard defines a link element consisting of a link to a document and an identifier as a proxy class which is actually not a unique element and just allows the assembled view of an element. Unique physical document entities must be defined using the **Entity** structure of part 2 of the standard which allows to define e.g. building objects or functional entities.

Nonetheless, it must be discussed whether an identifier is unique. A string based identifier that specifies a combination of the field and its value can be unique if referring to a unique value as it was done in this evaluation with the GUID or TaskId. However, an identifier can also address several elements. For instance, if the field is **Room** and the value is **Kitchen**, every element tagged with the specific room attribute are concerned by the identifier. This can also be transferred to the other identifier classes. Should each identifier be instantiated regardless of whether it is identical to an existing identifier? At this point, the object-oriented programming view contrasts the systematic logic just mentioned for the identification of objects. A problem while reusing already instantiated identifiers is the origins of these inside of the linksets. This could be avoided by placing references to the respective files which leads to a higher complexity.

As it was already stated in chapter 5.3.3, the identifiers are bound to the linksets. It is debatable whether it has advantages to attach identifiers to the documents. On the one hand, the identifiers can only be applied to the corresponding documents so it is logical to locate them in the same file and namespace as the definition of the documents (see fig. 6.7).

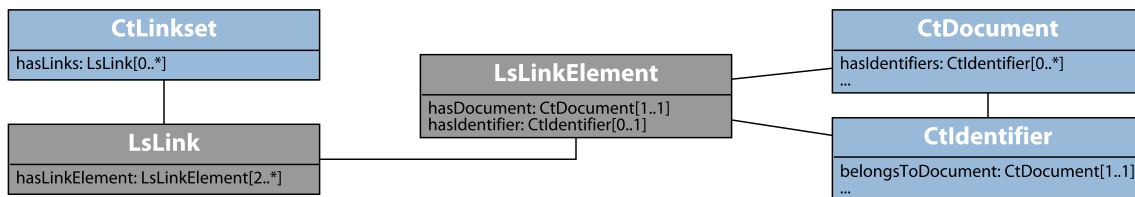


Figure 6.7.: Optimization of the relation between Identifier, Link and Document

On the other hand, the implementation as it was done in the standard has the advantage that the individual files are not inflated too much and only individuals are kept in each file that are actually referenced. If the identifier is attached to the document, the **LinkElement** proxy class is negligible. This, however, means that no more links can be created without identifiers which in turn speaks for the use of the proxy class. All these findings show the complex structure of the standard and reflects the high challenges that the standardization committee has to face.



## 7. Conclusion and Outlook

In the introduction, the thesis has opened with the question of how individual models and documents can be linked and exchanged within a container. To approach this question, the thesis delivers background on the current exchange methods of linked building models and technologies in the field of Linked Open Data. It critically examines the drafted ISO 21591 “Information Container for Data Drop” standard and discusses it in detail on the basis of application cases. Furthermore, a toolkit for validation, import, and export of the ICDD file type is introduced providing web-based access to the dynamic data model inside an ICDD file.

The research and the development of this thesis lead to the conclusion that the ICDD is an excellent medium for the exchange of linked building models as the respective standard is based on the emerging technology of Linked Data and the proven concept of an information container.

Basically, the container can be applied in many use cases and fulfills the purpose of data transfer in a single data drop. It is based on the paradigm of Linked Data which is a promising technology. This can be corroborated with the number of approaches for integration of Linked Data into AEC as well as in other knowledge-based fields of application. The structure of the container is well organized in folders and can provide documents and link sets in a large number. It is highly extendable and flexible due to User Defined Ontologies and dynamic metadata. Although, this flexibility leads to a higher complexity regarding the implementation of import and export functionalities. This applies in particular to the dynamic semantics and the parsing from RDF into a programmatic data model.

The development of an import, export and validation toolkit has demonstrated that background knowledge is absolutely necessary for an implementation. This work has the claim to give other developers an overview of the file type and the technical boundary conditions on the one hand, and provide a possibility for end-users to validate and view the content of their files on the other hand.

The tool for validation of files according to the standard was equipped with an explorer to view and manipulate data inside a container file. It is suitable for users that need to validate their files. Developers who have to implement import and export methods of information containers into their individual software solution can extract information for implementation approaches from this thesis. The developed and presented class library can be used in further projects and the Web API allows to easily integrate the complete framework into a new application. Furthermore, the thesis demonstrates the technologies and structures that can be used to develop an RDF import and export in accordance with the ICDD standard. A stepwise evaluation demonstrates that the defined requirements for the toolkit were mostly fulfilled and the performance of the application was successfully assessed.

During the implementation and the evaluation of the toolkit, it became obvious that the ICDD is an advanced approach for handling complex relationships between documents, links and additional information and making it available for all participants of the planning process.

## 7. Conclusion and Outlook

The implementation of string-based, query-based or URI-based identifiers, as well as the different link types, open up new possibilities for the definition of links. The discussion about the position of identifiers inside a container in the previous chapter has attempted to point out the intensive thoughts of the committee. These must be made during standardization to achieve a qualitative interoperable data type for the exchange of linked building models. Nevertheless, the standard is still under development and there are a few points that could be focused for further optimization of the standard. While future work will assess the practicality of the standard and the application cases, the process-related integration of the container into construction planning requires further research. It is important to point out that this work is based on the draft of the standard. The final version may vary and can result in changes for this framework.

In the end, Linked Data and the Information Container for Data Drop can significantly change the way how Linked Building Data can be exchanged and made accessible during the complete lifecycle.

# References

- Abanda, F. H., Zhou, W., Tah, J. H. M., and Cheung, F. (2013). Exploring the Relationships between Linked Open Data and Building Information Modelling. *Proceedings of the Sustainable Building Conference 2013*, pages 176–185.
- Antoniou, G. and van Harmelen, F. (2009). Web Ontology Language: OWL. In Staab, S. and Studer, R., editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 91–110. Springer Verlag, Berlin Heidelberg.
- Balzert, H. (2009). *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*. Lehrbücher der Informatik. Spektrum Akademischer Verlag, Heidelberg, third edition edition.
- Bauer, F. and Kaltenböck, M. (2012). *Linked open data: the essentials: A quick start guide for decision makers*. Ed. Mono / Monochrom, Vienna.
- Beck, K. (1999). *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, Reading, Massachusetts, US., 1. edition edition.
- Beetz, J. (2015). Ordnungssysteme im Bauwesen: Terminologien, Klassifikationen, Taxonomien und Ontologien. In Borrmann, A., König, M., Koch, C., and Beetz, J., editors, *Building Information Modeling*, VDI-Buch, pages 163–175. Springer Vieweg, Wiesbaden.
- Berners-Lee, T. (2006). Linked Data - Design Issues.
- Borrmann, A., König, M., Koch, C., and Beetz, J., editors (2015). *Building Information Modeling: Technologische Grundlagen und industrielle Praxis*. VDI-Buch. Springer Vieweg, Wiesbaden.
- Curry, E., O'Donnell, J., Corry, E., Hasan, S., Keane, M., and 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.
- Cyganiak, R., Wood, D., and Lanthaler, M. (2014). RDF 1.1 Concepts and Abstract Syntax.
- Demharter, J., Fuchs, S., Schapke, S.-E., and Scherer, R. J. (2014). Multimodell und Multimodellcontainer. In Scherer, R. J. and Schapke, S.-E., editors, *Informationssysteme im Bauwesen 1*, pages 39–63. Springer Berlin Heidelberg, Berlin, Heidelberg.
- DIN SPEC 91350 (2016). Linked BIM data exchange comprising building information model and specified bill of quantities.
- DIN SPEC 91400 (2017). Building Information Modeling (BIM) – Classification according to STLB-Bau.

- Ding, L., Zhou, Y., and Akinci, B. (2014). Building Information Modeling (BIM) application framework: The process of expanding from 3D to computable nD. *Automation in Construction*, 46:82–93.
- Du Juan and Zheng, Q. (2014). Cloud and Open BIM-Based Building Information Interoperability Research. *Journal of Service Science and Management*, 07(02):47–56.
- Eastman, C. M., Teicholz, P., Sacks, R., and Liston, K. (2011). *BIM Handbook: A guide to building information modeling for owners, managers, designers, engineers, and contractors*. Wiley, Hoboken N.J., 2nd edition edition.
- Fettke, P. (2001). Eine Ordnungslehre für Informationsmodelle. *Doctoral Consortium im Vorfeld der WI-IF 2001 - Kolloquium für Doktoranden der Wirtschaftsinformatik - 18. September 2001, Schloß Reisenburg bei Günzburg*.
- Forgues, D., Iordanova, I., Valdivesio, F., and Staub-French, S. (2012). Rethinking the Cost Estimating Process through 5D BIM: A Case Study. In Cai, H., Kandil, A., Hastak, M., and Dunston, P. S., editors, *Construction Research Congress 2012*, pages 778–786, Reston, VA. American Society of Civil Engineers.
- Fuchs, S. (2015). *Erschließung domänenübergreifender Informationsräume mit Multimodellen: Dresden, Technische Universität Dresden, Diss., 2015*, volume 11 of *Schriftenreihe des Instituts für Bauinformatik*. Sächsische Landesbibliothek- Staats- und Universitätsbibliothek Dresden, Dresden.
- Fuchs, S., Kadolsky, M., and Scherer, R. J. (2011). Formal Description of a Generic Multi-Model. In Reddy, S., editor, *20th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Piscataway, NJ. IEEE.
- Glinz, M. (2007). On Non-Functional Requirements. In Sutcliffe, A., editor, *15th IEEE International Requirements Engineering Conference, 2007*, Los Alamitos, Calif. IEEE Computer Soc.
- Hanff, J. and Wörter, J. (2015). BIM für die Mengenermittlung. In Borrmann, A., König, M., Koch, C., and Beetz, J., editors, *Building Information Modeling*, VDI-Buch, pages 333–341. Springer Vieweg, Wiesbaden.
- Hannemann, J. and Kett, J. (2010). Linked Data for Libraries. *World Library and Information Congress: 76th IFLA General Conference and Assembly*.
- Harris, S. and Seaborne, A. (2013). SPARQL 1.1 Query Language: <https://www.w3.org/TR/2013/REC-sparql11-query-20130321/>; accessed: 08.05.2018.
- Hoeber, J. G., Alsem, D. M., and Willems, P. H. (2015). The management of information over the life-cycle of a construction project using open-standard BIM. *Proceedings of the 32nd CIB W78 Conference 2015, 27th-29th 2015, Eindhoven, The Netherlands*.
- ISO 21597-1 (2018). Information Container for Data Drop - Exchange specification - Part 1: Container.
- ISO 21597-2 (2018). Information Container for Data Drop - Exchange specification - Part 2: Dynamic semantics.

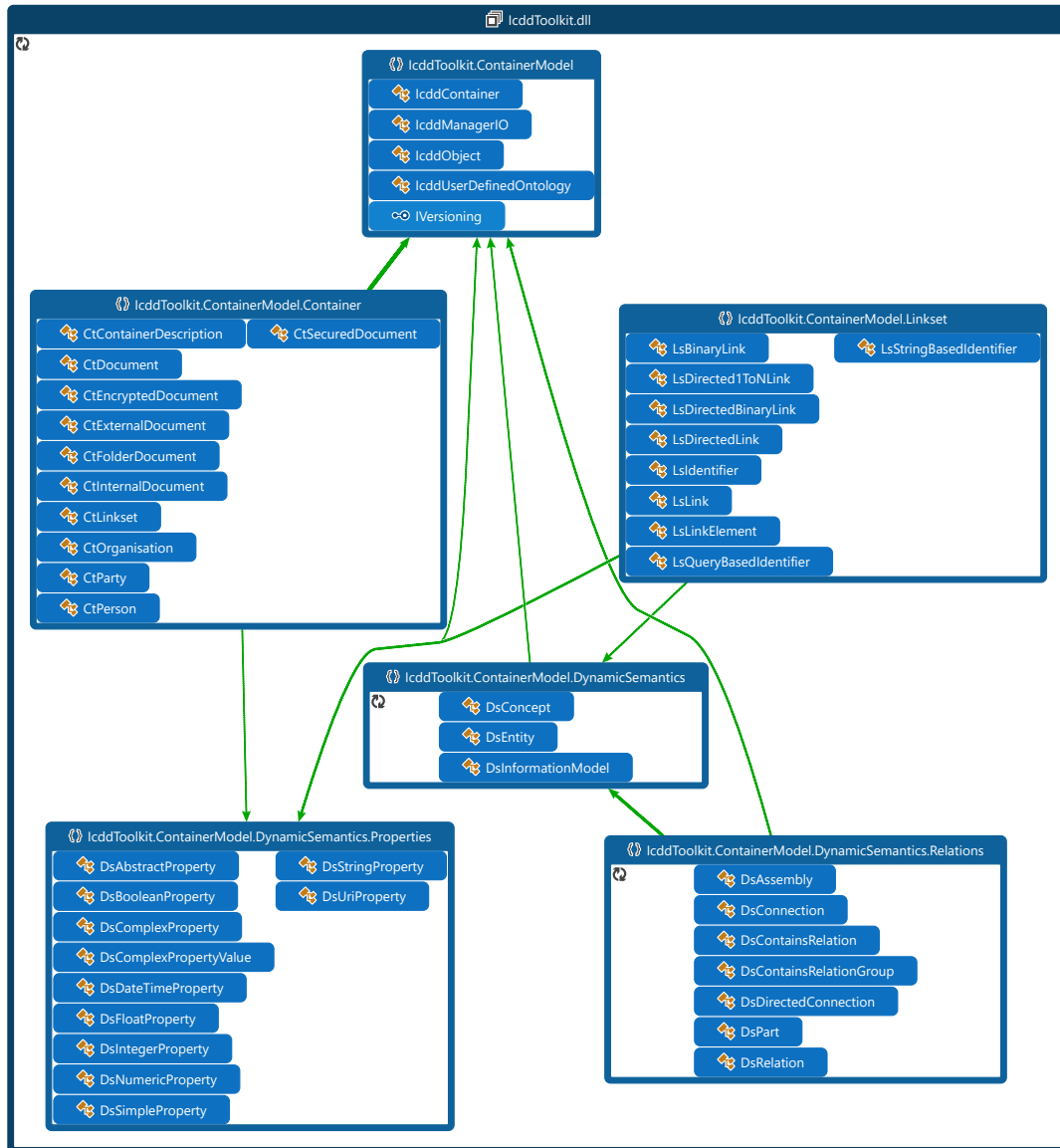
- Lockley, S., Benghi, C., and Černý, M. (2017). Xbim.Essentials: a library for interoperable building information applications. *The Journal of Open Source Software*, 2(20):473.
- Lockley, S., Cerny, M., Benghi, C., and Ward, A. (2016). xBIM xViewer Documentation.
- Madrazo, L. and Costa, G. (2012). Open Product Modelling and Interoperability in the AEC sector. *First International Workshop on Linked Data in Architecture and Construction (LDAC)*.
- Motik, B., Patel-Schneider, P., and Parsia, B. (2012). OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition).
- Musen, M. A. (2015). The Protégé Project: A Look Back and a Look Forward. *AI matters*, 1(4):4–12.
- Opitz, F., Windisch, R., and Scherer, R. J. (2014). Integration of Document- and Model-based Building Information for Project Management Support. *Procedia Engineering*, 85:403–411.
- Pauwels, P. and Terkaj, W. (2016). EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology. *Automation in Construction*, 63.
- Pauwels, P. and Van Deursen, D. (2012). IFC-to-RDF: Adaptation, Aggregation and Enrichment. *Workshop on Linked Data in Architecture and Construction (LDAC)*, pages 4–6.
- Rasmussen, M. H., Pauwels, P., Hviid, C. A., and Karlshøj, J. (2017a). Proposing a Central AEC Ontology That Allows for Domain Specific Extensions. *Lean and Computing in Construction Congress - Joint Conference on Computing in Construction*, pages 237–244.
- Rasmussen, M. H., Pauwels, P., Lefrancois, M., Schneider, G. F., Hviid, C. A., and Karlshøj, J. (2017b). Recent changes in the Building Topology Ontology. *5th Workshop on Linked Data in Architecture and Construction - Workshop Report 2017*.
- Rodriguez, A. (2008). RESTful Web services: The basics. *IBM developerWorks*.
- Royce, W. W. (1987). Managing the development of large software systems: concepts and techniques. In *Proceedings of the 9th International Conference on Software Engineering, March 30 - April 2, 1987, Monterey, California, USA*. IEEE Computer Soc. Press, Washington, DC.
- Scherer, R. J. and Schapke, S.-E. (2011). A distributed multi-model-based Management Information System for simulation and decision-making on construction projects. *Advanced Engineering Informatics*, 25(4):582–599.
- Schiller, K. and Faschingbauer, G. (2016). *Die BIM-Anwendung der DIN SPEC 91400*. Beuth Verlag, Berlin, Wien, Zürich, 1. auflage edition.
- Törmä, S., Jyrki, O., and Hoang, N. V. (2012). Distributed Transactional Building Information Management. *First International Workshop on Linked Data in Architecture and Construction (LDAC)*, pages 12–21.
- Völkel, M. (2005). RDFReactor – From Ontologies to Programmatic Data Access. *Poster and Demo at International Semantic Web Conference (ISWC) 2005, Galway, Ireland*.

## References

- Zeaaraoui, A., Bougroun, Z., Belkasmi, M. G., and Bouchentouf, T. (2013). User stories template for object-oriented applications. In *Third International Conference on Innovative Computing Technology (INTECH 2013)*, pages 407–410. IEEE.
- Zhang, C., Beetz, J., and de Vries, B. (2018). BimSPARQL: Domain-specific functional SPARQL extensions for querying RDF building data. *Semantic Web*, 3(4):1–27.

## **A. Appendices**

## A.1. Codemap for ContainerModel Namespace





## A.2. Method Documentation: SetInstanceAttribute

### IcddManagerIO.SetInstanceAttribute Method

Sets the attribute of a specific instance with a value in a specific type.

**Namespace:** ICDDToolkitLibrary.ContainerModel

**Assembly:** ICDDToolkitLibrary (in ICDDToolkitLibrary.dll)

#### Syntax

C# VB C++

```
public static bool SetInstanceAttribute(  
    Graph graph,  
    string id,  
    string attr,  
    Object val,  
    string type,  
    bool createIfNotExist  
)
```

#### Parameters

*graph*

Type: [Graph](#)  
The respective RDF graph.

*id*

Type: [String](#)  
The unique identifier of the instance.

*attr*

Type: [String](#)  
The respective attribute from the graph.

*val*

Type: [Object](#)  
The value that is to be set.

*type*

Type: [String](#)  
The type of the attribute.

*createIfNotExist*

Type: [Boolean](#)  
if set to `true` creates a new attribute for this instance.

## A.3. Method Documentation: StartSession

### SessionManager.StartSession Method

Starts a new *Session* with a specified *IcddContainer* file.

**Namespace:** IcddToolkit

**Assembly:** IcddToolkit (in IcddToolkit.exe)

#### Syntax

C# VB C++

```
public static string StartSession(  
    IcddContainer container  
)
```

#### Parameters

*container*

Type: *IcddContainer*

The container file for which the session is started.

#### Return Value

a *string* containing the GUID of the *Session*