

Department of Civil, Geo and Environmental Engineering
Chair of Computational Modeling and Simulation
Prof. Dr.-Ing. André Borrmann

An approach to enrich and validate IFC models by translating given data into standardised information requirements

Kilian Speiser

Master's thesis
of the Master of Science program Civil Engineering

Author:	Kilian Speiser
Student umber:	[redacted]
Supervisor:	Prof. Dr.-Ing. André Borrmann
	Alex Braun
Date of issue:	October 1, 2019
Date of submission:	April 1, 2020

Abstract

BIM models comprise extensive data that the project team can use to automatise BIM Uses based on algorithms. However, such an algorithm queries and analyses information regarding their labelling. As various stakeholders may label the same information differently, project participants need to update their algorithms for every data exchange manually. These adjustments especially result in a high workload when applying multiple BIM Uses on one BIM model. The thesis presents a novel approach to enrich BIM models, in the form of IFC, by translating provided information into internally standardised information requirements that base on Exchange Information Requirements (EIR), but extend them by another dimension: the labelling requirements. The author proposes the model of Internal Model Information Requirements (IMIR) to store and maintain both dimensions. Additionally, it introduces hierarchically structured information requirements. The hierarchy is essential as different data triggers different requirements. For example, the material *Concrete* requires different information than the material *Timber*. The IMIR-model eventually allows for mvdXML export. Afterwards, the IMIR-model functions to assign data from an IFC model to data from the IMIR itself. The Enrich-IFC-model stores these assigned pairs and derives an enriched IFC model containing both, the original information and the internally standardised information. Moreover, the Enrich-IFC-model allows for model validation and export of model-related errors for collaboration in the form of BCF files. The author converts the approach into a prototype that provides core functionalities of the Enrich-IFC-model, including a graphical user interface. The thesis compares the innovative approach with a conventional approach applying them to the BIM Use *Cost Estimation* to prove the value. The Enrich-IFC approach does not yet decrease the workload. However, developing the usability of the prototype and applying it to multiple BIM Uses is expected to minimise manual work intensively. Compared to the conventional approach, the main benefits of the Enrich-IFC-approach are: (1) coherent and flexible structure of information requirements, (2) comprehensible model validation supporting BCF communication and (3) the expert only needs to work with one application instead of multiple.

Zusammenfassung

BIM-Modelle beinhalten umfangreiche Daten, mit denen das Projektteam BIM Anwendungsfälle basierend auf Algorithmen automatisieren kann. Ein solcher Algorithmus fragt jedoch Informationen bezüglich ihrer Bezeichnung ab und analysiert diese. Da aber verschiedene Stakeholder dieselbe Information unterschiedlich bezeichnen könnten, müssen Projektteilnehmer ihre Algorithmen für jeden Datanaustausch manuell aktualisieren. Diese Anpassungen resultieren in einem hohen Arbeitsaufwand, vor allem, wenn ein BIM Model mehrere BIM Anwendungsfälle abdecken soll. Die vorliegende Masterarbeit präsentiert einen neuartigen Ansatz zur Anreicherung von BIM Modellen (in Form von IFC), indem vorhandene Informationen in intern standardisierte Informationsanforderungen übersetzt werden. Diese Anforderungen erweitern Exchange Information Requirements (EIR) um eine zusätzliche Dimension: die Beizeichnungsanforderungen. Dazu führt der Autor das Modell der Internal Model Information Requirements (IMIR) ein, welches beide Dimensionen speichert und pflegt und zusätzlich Informationsanforderungen hierarchisch strukturiert. Diese Hierarchie ist essentiell, da verschiedene Daten unterschiedliche Anforderungen implementieren. So erfordert beispielsweise das Material *Beton* andere Eigenschaften als das Material *Holz*. Abschließend ermöglicht das IMIR-Modell den Export von mvdXML. Danach fungiert das IMIR-Modell als Grundlage, Informationen aus einem IFC-Modell mit Daten aus dem IMIR-Modell zu verknüpfen. Das Enrich-IFC-Modell speichert die zugewiesenen Paare und erstellt daraus ein angereichertes IFC-Modell, das beide Anteile beinhaltet: Die Informationen aus dem ursprünglichen IFC-Modell sowie die Informationen aus den internen, standardisierten Anforderungen. Zusätzlich ermöglicht das Enrich-IFC-Modell das Validieren von BIM Modellen und den Export von modellbezogenen Fehlern als BCF-Dateien. Der Autor erstellt aus dem Enrich-IFC-Ansatz einen Prototypen, der die Kernfunktionen des Enrich-IFC-Modells einschließlich einer grafischen Benutzeroberfläche bereitstellt. Diese Arbeit vergleicht den innovativen Enrich-IFC-Ansatz mit einem konventionellen Ansatz anschließend am Beispiel des BIM Anwendungsfalls *Kostenabschätzung*, um den Nutzen nachzuweisen. Dabei verringert der Enrich-IFC-Ansatz noch nicht den Arbeitsaufwand. Eine deutliche Minimierung der manuellen Arbeit erwartet der Autor durch die Weiterentwicklung der Benutzerfreundlichkeit des Prototypen sowie durch den Einsatz des selben Modells für mehrerer BIM Anwendungsfälle. Verglichen mit dem konventionellen Ansatz, punktet der vorgeschlagenen Enrich-IFC-Ansatz mit folgenden Vorteilen: (1) Schlüssige und flexible Struktur der Informationsanforderungen, (2) verständliche Modell Validierung, die Kommunikation über BCF ermöglicht und (3) manuelle Arbeit ist nur in einer Anwendung, anstelle von mehreren, erfordert.

Preface

The vision of implementing such a license-free software tool grew during my work in the department of innovation and digitalisation at *Geiger*. The topic enhanced within the past year. What for me personally firstly appeared like a problem of information translation, become more and more a problem of model validation.
This thesis forms the last part of my studies. Both my studies and this work has involved multiple persons and institutions. For that reason, I want to specially thank

- *Christopher Klimesch* for supporting me with ideas, hints and critics. It has always been a pleasure to work with him.
- my supervisors at the *Technical University of Munich*, *Prof. André Borrmann* and *Alex Braun*. Both have significantly formed my profile with their knowledge and support throughout my studies, especially while working on this work.
- the *Technical University of Munich* for providing such an exclusive and comprehensive range of studies.
- the *Technical University of Denmark*, where I spent one year of exchange and started working on this thesis.
- and lastly, my parents and sisters for supporting me throughout my studies.

Kilian Speiser
Sulzberg, 31^{st} of March 2020

Contents

1 Introduction **1**
1.1 Data Exchange in AEC Industry . . . 1
1.2 Motivation . . . 2
1.3 Objectives and Structure . . . 5

2 Background Information **6**
2.1 Industry Foundation Class . . . 6
2.1.1 The History of IFC . . . 6
2.1.2 Data Structure . . . 7
2.2 Level of Development . . . 9
2.3 Information Management in BIM Projects . . . 10
2.3.1 Project Perspectives . . . 10
2.3.2 Information Requirements According to ISO 19650-1 . . . 12
2.3.3 The Information Delivery According to ISO 19650 . . . 16
2.4 Technical Implementation of EIR . . . 17
2.4.1 Model View Definitions and Exchange Requirements . . . 17
2.4.2 Purpose and Origin of mvdXML . . . 18
2.4.3 MvdXML Schema . . . 19
2.4.4 MvdXML to Validate IFC Documents . . . 22
2.5 Issue Communication in BIM projects . . . 25
2.5.1 Motivation . . . 25
2.5.2 Distinction in Markup and Visualisation . . . 25
2.6 Classify and Structure Deliverables . . . 27
2.6.1 Work Breakdown Structure . . . 27
2.6.2 Usage in AEC industry . . . 28
2.7 Summary . . . 29

3 Current Situation **30**
3.1 Ifc Validation Based on MvdXML . . . 30
3.1.1 Identify EIR . . . 31
3.1.2 Derive the MvdXml Document . . . 31

3.1.3 Validation Using XBim Xplorer . . . 34
3.1.4 Conclusion . . . 34
3.2 Information Requirements with BimQ . . . 36
3.2.1 Functionalities . . . 36
3.2.2 Example . . . 37
3.2.3 Conclusion . . . 38
3.3 Summary . . . 38

4 Enrich-IFC-Approach 40
4.1 Initial State . . . 40
4.2 Exchange Requirements and Their Structure . . . 42
4.2.1 Overview . . . 43
4.2.2 Requirements on the Structure of the EIR . . . 44
4.2.3 Convert into MVD . . . 47
4.2.4 Summary . . . 47
4.3 Enrich-IFC-Model . . . 48
4.3.1 Prepare . . . 49
4.3.2 Entity-Relationships . . . 50
4.3.3 Add Information to the Model . . . 52
4.3.4 Validate the Information . . . 54
4.3.5 Review Errors . . . 57
4.3.6 Finalise . . . 58
4.4 Summary . . . 59

5 Data Model Design 60
5.1 Overview . . . 61
5.2 Internal Model Information Requirements . . . 61
5.2.1 Requirements on the Data Model . . . 61
5.2.2 IMIR Design . . . 62
5.2.3 Store and Export Data Model . . . 62
5.2.4 Instance of the IMIR-Data-Model . . . 65
5.3 Enrich-IFC-Model . . . 66
5.3.1 Enrich-IFC-Model Design . . . 67
5.3.2 Instance of a Data Model . . . 69
5.3.3 Agile Principles . . . 71

6 Prototype 72
6.1 Ideal System . . . 72
6.2 Implemented Features . . . 73
6.2.1 Menu . . . 74
6.2.2 Information Trees . . . 75

6.2.3 Assigned Pairs . . . 75
6.2.4 Validate . . . 76
6.3 Prototype in Use . . . 76
6.4 Summary . . . 80

7 Case-study: Quantity-Takeoff 82
7.1 Using iTWO for QTO . . . 82
7.2 Enrich-IFC-Approach . . . 84
7.2.1 Overview . . . 84
7.2.2 Determine Information Requirements . . . 86
7.2.3 Create Deliverables Library in iTWO . . . 86
7.2.4 Adapt External Model to IMIR . . . 88
7.2.5 Validate the Model . . . 89
7.2.6 Execute Quantity-takeoff (QTO) Using iTWO . . . 89
7.3 Conclusion . . . 90

8 Conclusion and Future Development 92
8.1 Conclusion . . . 92
8.2 Vision . . . 93

A MvdXML for Validation 96

B Data Model Design 100
B.1 IMIR UML Diagram . . . 100
B.2 How to Create an IMIR Model . . . 101
B.3 Enrich-IFC: UML Diagram . . . 104

C Case-Study: Floor Plans 105

D Digital Content 107

List of Figures

1.1 Information exchange during the whole life time of a built asset. 2
1.2 Conventional approach: For every BIM Use, the acting party must adapt downstream processes. 4
1.3 Enrich -IFC-approach. 4
1.4 The Enrich-IFC-approach. 5

2.1 The four layers of the IFC schema. 8
2.2 This excerpt from the IFC data schema shows the highest levels of the schema. 9
2.3 Levels of Development: the example shows a steel beam connecting to a column. 10
2.4 Information requirements in BIM projects. 11
2.5 Information requirements throughout the asset's lifecycle. 12
2.6 BEP, TIDP, and MIDP in BIM Projects. 14
2.7 The asset delivery process divides into eight subprocesses. 16
2.8 mvdXML describes a subset of the IFC schema technically. 18
2.9 Schema of the mvdXML. 20
2.10 WBS subdivides the tasks in the smallest possible deliverable. 28

3.1 How to use mvdXML for model validation in a BIM project. 31
3.2 Model validation 34
3.3 Two walls defining different properties. 35
3.4 Requirements view in BimQ 37
3.5 Model validation using xBim Xplorer 38

4.1 Principle of the federated model approach 41
4.2 The concept of IMIR. 43
4.3 The figure (left) shows the structure of the data model based on references. Deriving a tree from that creates the WBS codes (right). 46
4.4 Overall Enrich-IFC-approach 48
4.5 The first step of the approach: Preparation. 49
4.6 Process to assign values from the information requirements model to the IFC model. 50
4.7 Input and output for model validation. 54

4.8 Process to handle errors in the Enrich-IFC-approach. 57
4.9 Complete the Enrich-IFC-approach. 58

5.1 The roles of the two data models within the Enrich-IFC-approach. 60
5.2 Simplified ML diagram of the IMIR model 63
5.3 Instance of the IMIR data model. 65
5.4 IMIR-model as tree, including WBS codes. 66
5.5 Reduced UML diagram of the Enrich-IFC-model design 68
5.6 Instance of the Enrich-IFC- model. 69

6.1 Overview of the ideal system design. 73
6.2 Main view of the GUI. 74
6.3 Component 1 from figure 6.2. 74
6.4 Assigned Value Tuples View of the prototype. 75
6.5 3D view of the model. 77
6.6 Tree view of the GUI after reading the data from the IFC model and the MVD. 77
6.7 *Assigned Value Tuples* view after assigning eight pairs. 78
6.8 Model quality of the enriched IFC model. 78
6.9 *WA:06* before (left) and after (right) the enrichment. 79

7.1 3D View of the sample projects. Left: Project A. Right: Project B. 85
7.2 Information requirements for the model. 87
7.3 Define deliverables in iTWO. 87
7.4 Properties of the slab before (left) and after (right) applying the Enrich-IFC-approach. 88
7.5 Validation: the xBim Xplorer highlights the passed (green) and failed (red) concepts. 89
7.6 Comparison of the workload for the different tasks of both approaches. 91

C.1 Ground floor. 105
C.2 First floor. 106

List of Listings

2.1 Excerpt from a simple file introducing the syntax and structure of XML. . . . 19
2.2 mvdXML: ConceptRoot 23
2.3 *Concept* checking properties of objects of the IFC schema 24
2.4 *ConceptTemplate* tag in a mvdXML document to retrieve property values. . . 24
2.5 BCF: Markup file. 26
2.6 BCF: Visualization Information. 26
3.1 Strict applicability in mvdXML. 32
3.2 Strict heirarchy in mvdXML. 33
5.1 XML file comprising an IMIR model. 64
5.2 BCF export from the Enrich-IFC-model. 70
5.3 Assigned pair in XML generated from the Enrich-IFC-model. 70
6.1 Markuf file of a BCF export. 80
6.2 XML file comprising assigned value tuples. 80
A.1 Strict applicability: the *Applicability* specifies the subset of the IFC schema, and the *TemplateRules* test define tests that apply on the subset. 96
A.2 Strict hierarchy: one concept comprises all rules. The *TemplateRules* apply to all elements specified in the *applicableRootEntity* attribute. 98
B.1 The following listing creates a IMIR data model and saves it as an XML file. Moreover, it derives a mvdXML file from it. 103

List of Tables

3.1 All external walls must define the properties in the table to the left. If the component's 'Material' equals 'Concrete' it must also implement the properties in the table to the right. . . . 31
3.2 Comparison of both options regarding structure, complexity, readability, and usage. . . . 35

4.1 Exchange information requirements for a beam. . . . 42
4.2 The left table lists the properties of the IFC element and the right table represents the required information from the MVD. . . . 51
4.3 The model comprises four relationships: Three value to value relationships and one property to property relationship. Each relationship implies that the IFC Information matches the Required Information . . . 52
4.4 Final properties of the beam after adding the required information according to existing information. . . . 53
4.5 Requirements for the data models to convert the process technically. . . . 59

6.1 Summary of the functionalities of the prototype. The numbers refer to the requirements mentioned in section 6.1. . . . 81

7.1 Excerpt of the deliverables library for the components 'Room' (left) and 'Ground Plate' (right). . . . 83
7.2 Time consumption of executed tasks within using the two approaches. . . . 90

Glossary

AEC	Architecture, Engineering and Construction
AIM	Asset Information Model
AIR	Asset Information Requirements
BCF	BIM Collaboration Format
BEP	BIM Project Execution Plan
BIM	Building Information Modelling
bsDD	buildingSMART Data Dictionary
CDE	Common Data Environment
ER	Exchange Requirements
EIR	Exchange Information Requirements
GUI	Graphical User Interface
GUID	Globally Unique Identifier
HVAC	Heating, ventilation, and air conditioning
IFC	Industry Foundation Class
IMIR	Internal Model Information Requirements
ISO	Internation Organization for Standardization
IAI	International Alliance for Interoperability
LOD	Level of Development
LOG	Level of Geometry
LOI	Level of Information
LOR	Level of Reliability
MIDP	Master Information Delivery Plan
MVD	Model View Definition
OIR	Organisational Information Requirements
PIM	Project Information Model
PIR	Project Information Requirements
QTO	Quantity-takeoff
TIDP	Task Information Delivery Plan
UML	Unified Modeling Language
WBS	Work Breakdown Structure
XML	Extensible Markup Language

Chapter 1

Introduction

What does the above image show? One might see reinforcement, one might see armoring, and a third person might see armouring, rebar or rebars. Hence, humans use different words to express the same information. Human intelligence can interpret these words and conclude that they refer to the same object. Algorithms, however, compare characters and digits and cannot detect that all words mean the same. Thus, the computer system cannot search, filter, and analyse the information. But, this is essential to implement Building Information Modelling (BIM) in construction projects.

1.1 Data Exchange in AEC Industry

Digitalisation has changed industries over the last two decades. This has increased productivity and product quality tremendously. Architecture, Engineering and Construction (AEC) industry, however, has dragged behind other sectors. To catch up, AEC adopts digital tools more frequently to design, construct, operate, and modify built facilities. Those tools process data which is saved in a Building Information Model. (Borrmann *et al.*, 2018)

Figure 1.1: Information exchange during the whole life time of a built asset. (Speiser (2019), based on Borrmann *et al.* (2018))

Borrmann *et al.* (2018) define BIM as a process of creating, maintaining, using, and exchanging in such a model during the lifetime of an asset. Its lifetime divides into five phases. Figure 1.1 illustrates the stages of BIM and introduces BIM Uses for each stage. A BIM Use represents the *"method of applying BIM during a facility's lifecycle to achieve one or more specific objectives"* (CIC, 2011). Especially for projects with multiple BIM Uses, the primary benefit of BIM becomes operative: using data and information throughout project phases reduces the error-prone and time-consuming re-entering of data to the lowest (Borrmann *et al.*, 2015b).

One of these BIM Uses is Cost Estimation. It aims at estimating exact quantities and costs (CIC, 2019). Here, an algorithm selects elements from the BIM model, determines quantities, and derives costs. The algorithm requires information to select the correct elements. However, if different stakeholders use different words for the same information, how will the algorithm derive the costs? Hence, the stakeholder must edit the information. But, how does this decrease the error-prone re-entering of data?

1.2 Motivation

At the beginning of a BIM project, the project delivery team has agreed on a BIM Project Execution Plan (BEP). This plan defines all information requirements for every data exchange

in Exchange Information Requirements (EIR). Hence, every project participant knows when to provide what information (ISO 19650-1, 2017).

For meaningful automation of processes, current EIR lack two concepts. For example, Die Deutsche Bauindustrie (2018) publishes EIR that require a wall to implement the property 'Reinforcement Ratio'. However, a timber wall cannot define this requirement as it does not comprise reinforcement. Thus, the current EIR lack a hierarchical structure that allows defining dynamic information requirements depending on other information. The previous example illustrates that. While the material concrete may require to specify information about the reinforcement ratio, the material timber may need to define the tensile strength.

The second issue of current EIR is that they indeed rule what information the receiver of information (appointing party) can expect, but not rule how the provider of information (appointed party) labels the data. For example, an automated Cost Estimation only works sensibly if the filters search for keywords that they know. The following example explains this problem.

A client receives a BIM model from an architect. This model implements the information requirements from the BEP. To execute the Cost Estimation, the client needs to determine whether the BIM model complies with the requirements from the BEP or not. This process is called 'Validation'. For automatised model validation, the algorithm needs to know where to search for what kind of information. For example, the BEP defines that a component of type 'Wall' must define the attribute 'LoadBearing'. However, the architect names this information 'Structural Usage'. Hence, the client has to adapt the algorithm according to the labelling standard of the architect. While this may look easy on this scale, it becomes intense in a real project defining thousands of building components. Nevertheless, after adapting the algorithm, the client can validate the model. If the model complies with the information requirements, the client can continue with the Cost Estimation.

The following filter determines the volume of all exterior concrete walls:

Select(ComponentType == 'WALL'(Where(
IsExternal == 'TRUE' AND
Material == 'CONCRETE' AND
Level == 'EG')))

This filter assumes that all concrete elements in the model define the material property with the value 'CONCRETE'. In a different project, the client receives a model defining the same material as 'reinforced concrete'. Hence, the client has to adapt the filters and algorithms that execute the Cost Estimation.

Using the same model for several BIM Uses in different software environments even intensifies this problem. The receiver of information has to adapt several algorithms in different software (figure 1.2). However, if the appointing party knew what syntax and labelling the model

comprises, they could adjust the filters and algorithms of downstream processes to it. Figure 1.3 illustrates how this thesis proposes to achieve this. The appointing party creates their internal labelling standard and enriches the external BIM model to that. Enrichment in this context implies that depending on the given information, the appointed party adds the same information with their internal labelling preference. For example, if the model comprises a wall with the material 'RC', the appointing party adds another property defining that the material is 'Reinforced Concrete' (labelling from the internal standard). Now, all downstream processes know what information they can expect. For that reason, the client develops the algorithms for the downstream processes only once. Adapting them to the data from the external BIM model is not required anymore.

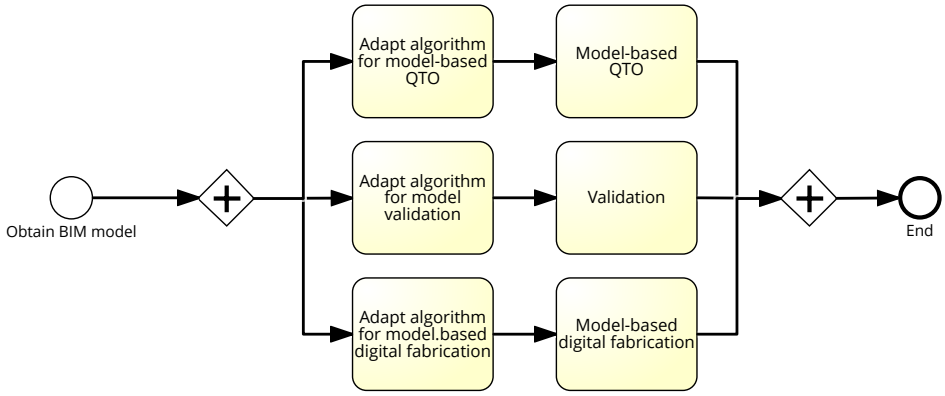


Figure 1.2: Conventional approach: For every BIM Use, the acting party must adapt downstream processes.

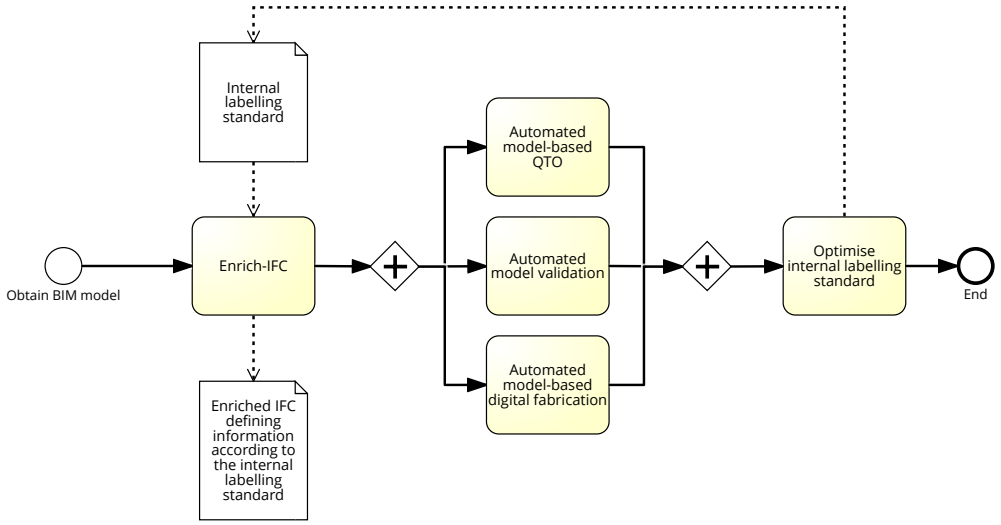
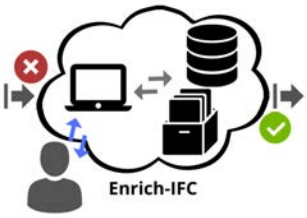


Figure 1.3: Enrich-IFC-approach: The BIM model is enriched with the information once. After that, all downstream processes are executed automatised.

Received	
Type	Girder
Material	Concrete C30/37
Width	0.20m



Desired	
Component Type	Beam
Material	Concrete
Strength	C30/37
Thickness	0.20m

Figure 1.4: Left: What the project participant obtains. Right: What they want. The Enrich-IFC-approach closes this gap.

1.3 Objectives and Structure

To sum up, there are two issues: Firstly, the information requirements shall structure hierarchically. The hierarchy allows defining dynamic requirements depending on different information. For example, a masonry wall requires different attributes than a concrete wall. However, a wall always expects to have a material property. Secondly, the receiver of information shall establish an internal labelling standard and adapt algorithms and automatised downstream processes to it. Then, they can enrich received BIM models with this standard according to given information and execute downstream processes automatised.

Figure 1.4 illustrates the primary objective of this work. A project participant receives a BIM model comprising a beam which implements information about its material, compressive strength, width, and component type. This is what the project participant obtains, while the right part of figure 1.4 shows what they would like to have. They require a BIM model to follow their internal labelling standard.

This thesis proposes the Enrich-IFC-approach to close this gap. The approach means to enrich a BIM model by translating given information into standardised data to perform automatised downstream processes. The author aims at developing an approach that fits in current regulations and bases on open-based and license-free standards. Moreover, the author converts this approach into a software tool. The application provides functionalities to create an enriched BIM model according to their preferences defined in the internal standard.

Therefore, the work firstly introduces background information and significant technical coherences about information management in BIM projects according to specific standards. After that, chapter 3 overviews current solutions on defining information requirements and validation of BIM models using mvdXML. Chapter 4 summarises the problem and explains the proposed solution. Chapter 5 sketches the data model design that realises the Enrich-IFC-approach technically. In chapter 6, the author outlines the workflow of a prototype that implements the Enrich-IFC-model. This prototype illustrates the usage of the tool by applying it to a project in chapter 7. Finally, section 8 concludes the outcome and proposes future work to improve the concepts.

Chapter 2

Background Information

This chapter describes topics that significantly relate to the content of the thesis. Firstly, it overviews the only license-free data schema for information exchange in the AEC industry, the Industry Foundation Class (IFC). After that, section 2.2 compares different concepts for the Level of Development (LOD). Thirdly, the chapter investigates information management in BIM projects and distinguishes different types of information requirements. Section 2.4 unites the information requirements and the IFC by describing mvdXML. Lastly, the chapter introduces the Work Breakdown Structure (WBS) after explaining a file format for collaboration in BIM projects.

2.1 Industry Foundation Class

For collaboration in AEC projects, the participants have two options, working in a closed or an open BIM environment (Borrmann *et al.*, 2018). While in closed BIM, the project team agrees to work with the same software, open BIM allows them to choose individual software applications. However, this only works if all stakeholders agree on a common data exchange file format to ensure loss-free information flow. An open BIM workflow requires this to guarantee interoperability (Borrmann *et al.*, 2018). This is the starting point of the first version of IFC published by buildingSMART in 1997. (Laasko & Kiviniemi, 2012).

2.1.1 The History of IFC

Software vendors developed the first approaches for loss-free information exchange between different Computer-aided Design systems in the 1970s. These methods, however, only shared geometry. Thus, various stakeholders wanted to develop a solution for exchanging semantics. After bureaucratic issues with Internation Organization for Standardization (ISO), a group of

software vendors, engineering offices, and construction companies founded the International Alliance for Interoperability (IAI) in 1995 to expedite the standardisation. IAI established the IFC standard and published IFC 1.0 in 1997. In 1998, the first commercial software used the standard. IAI released further IFC versions within the next years, although the construction industry questioned the benefit. Thus, IAI changed its vision. Instead of enabling interoperability in the AEC industry, they aimed at improving communication, quality, time and cost, throughout the lifecycle of an asset. Since there is no license fee for using IFC, the standard has popularised over the last decades. (Borrmann *et al.*, 2018; Laasko & Kiviniemi, 2012; Speiser, 2019)

Today, over 160 software products support IFC. Additionally, incorporating IFC in the ISO Standard contributes to using IFC in the public sector. IFC has become the ultimate standard for working in an open BIM environment. The open data structure and the neutrality of the format have convinced most public sector initiatives to use IFC. Currently, IFC primary covers buildings. However, buildingSMART focuses on developing it for other disciplines, such as infrastructure. (Borrmann *et al.*, 2018; Laasko & Kiviniemi, 2012; Speiser, 2019)

2.1.2 Data Structure

The IFC schema establishes an object-oriented data schema based on inheritance. It comprises objects in four layers that can reference objects of lower layers but not vice versa. The following paragraphs describe the object hierarchy after overviewing the relations of the four segments (buildingSMART, 2019c).

2.1.2.1 The Four Layers of the IFC

The IFC data schema implements four layers: Domain Layer, Interoperability Layer, Core Layer and Resource Layer (figure 2.1). The Core Layer builds the root and contains the kernel representing general, abstract classes. It provides three extensions: Process Extension, Control Extension, and Product Extension. The latter defines classes to describe the physique and space of buildings, e.g., *IfcOpeningElement*, *IfcSpace*, *IfcBuildingElement*. The Interoperability Layer stands between the abstract core of the schema and the domain-specific core. It implements classes that derive from classes of the Core Layer. The Domain Layer describes domain-specific types (such as HVAC, Architecturea and Structural Analysis). Elements of other layers cannot reference the classes defined in the Domain Layer. Lastly, the Resource Layer provides basic elements for the entire IFC model. These elements do not derive from *IfcRoot*. For that reason, they cannot exist independently and require other elements to reference them. Classes of the Resource Layer may define geometries, materials, or topology. (International Organization for Standardization, 2018; Borrmann *et al.*, 2018; Speiser, 2019)

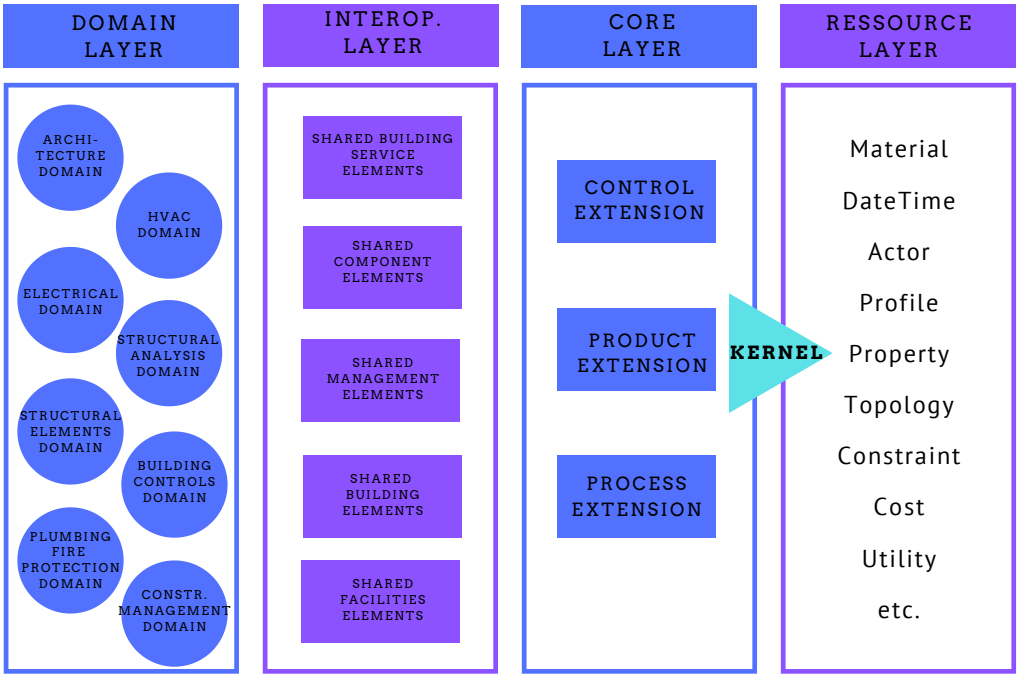


Figure 2.1: The four layers of the IFC schema. (Speiser (2019), based on buildingSMART (2019c))

2.1.2.2 IfcRoot and Subclasses

Figure 2.2 depicts the hierarchy of the classes. *IfcRoot* builds the root of the schema. All classes from the Domain, Interoperability, and Core Layer derive from *IfcRoot*. Three abstract classes form the next level: *IfcPropertyDefinition*, *IfcRelationship*, and *IfcObjectDefinition*. *IfcPropertyDefinition* generalises characteristics of objects through property sets. Classes deriving from *IfcRelationship* realise objectified relationships in the IFC. This concept separates semantic information from the objects themselves. Finally, *IfcObjectDefinition* generalises all semantically treated objects. (buildingSMART, 2005)

The two sub-types of *IfcObjectDefinition* are *IfcTypeObject* and *IfcObject*. While *IfcObjectType* describes a type, an *IfcObject* defines physical objects of a model. *IfcProduct* inherits from *IfcObject* and describes any object that defines geometry or space. Thus, two subclasses are *IfcSpatialStructureElement* and *IfcElement*. *IfcSpatialStructureElement* provides classes to organise a building such as *IfcBuilding* or *IfcSpace*. *IfcElement* is the subtype of all components related to the AEC industry (such as columns, pipes and chairs). (buildingSMART, 2005)

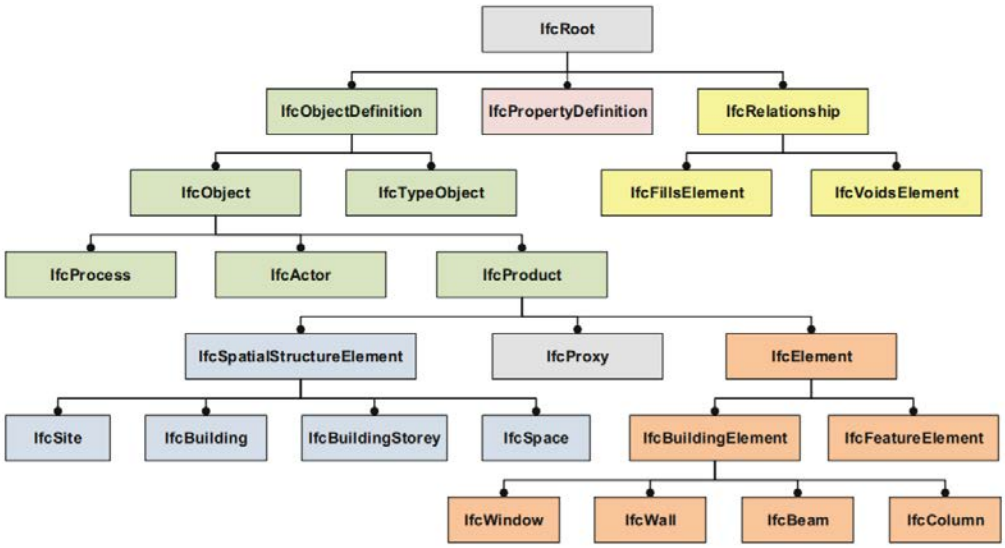


Figure 2.2: This excerpt from the IFC data schema shows the highest levels of the schema. (Borrmann *et al.*, 2015a)

2.2 Level of Development

Project delivery teams define information requirements by using a Level of Development (LOD). The concept of LOD distinguishes different grades of information. It allows the recipient to evaluate the reliability of a model. Several institutes have defined their standards for the LOD. For example, the BIMForum (2019) has established six levels:

LOD 100:	No requirements
LOD 200:	A generic object with approximated orientation, shape, dimensions and location
LOD 300:	A specific object with accurate information about orientation, shape, dimensions and location
LOD 350:	LOD 300 + relationships to other objects
LOD 400:	LOD 350 + information about detailing, assembly and fabrication
LOD 500:	Field verified element

These definitions only cover the graphical representation of a component. Beyond that, Die Deutsche Bauindustrie (2018) introduces the Level of Geometry (LOG) and Level of Information (LOI). The LOG concentrates purely on the graphics, and the LOI treats semantics. Digital Konvergens *et al.* (2019) extends this concept by another dimension, the Level of Reliability (LOR). (Speiser, 2019)

Figure 2.3 shows the different LODs for a steel beam. It illustrates that there is no distinct line between a LOD, LOI, and LOR. For example, a LOD 300 may require material information to represent the geometry accurately. However, the LOI may also define semantics that requires

Figure 2.3: Levels of Development: the example shows a steel beam connecting to a column. (Speiser (2019), based on BIMForum (2019))

information about the material. Hölzlwimmer (2019) investigates different approaches in detail. In this work, LOD mainly refers to the semantic information that other institutes may name LOI.

2.3 Information Management in BIM Projects

This section describes information management in BIM projects by firstly identifying project perspectives. After that, the chapter explains the different types of information requirements and information delivery plans, when they occur, and what objectives they follow. Thirdly, the section describes how the project delivery team generates information during different phases.

2.3.1 Project Perspectives

Several project participants appear during a construction project. They all define their requirements on information exchanges. For this reason, the project team defines a common project structure, including the flow of information. ISO 19650-1 (2017) specifies various perspectives that occur during the project.

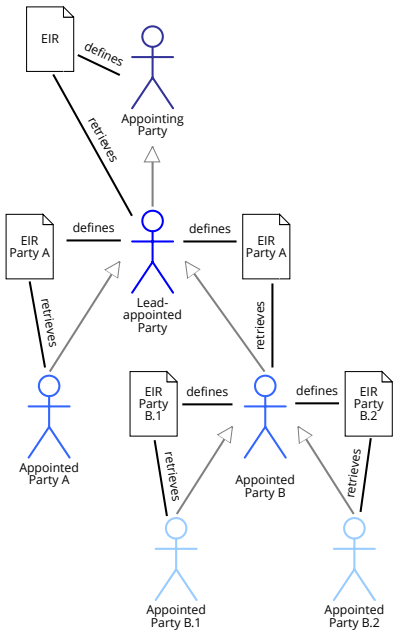


Figure 2.4: Parties define their Information Requirements (EIR). Sub-parties retrieve these and deliver the information according to the requirements (grey arrows) to their appointing party.

The asset owner takes a strategic, global, and profit-oriented perspective. Regardless of specific knowledge about construction methods or project execution, this perspective determines the basis for the expected value. The asset user should meet the requirements of the asset owner's business plan by defining Asset Information Requirements (AIR) (compare section 2.3.2). They form the basis for the strategic information requirements. The user perspective aims at complying with the owner's schedule and budget during the project delivery phase. Finally, there is the project perspective defining precise information requirements derived from the owner's perspective. Here, several project delivery teams contribute subsets of information. However, information loss often occurs at crucial decision points. Hence, it is necessary to clearly define the relevant information to be supplied in terms of quality and quantity and implement it in the project process beforehand. Thus, project participants need to define key decision points throughout the asset life cycle to depict the information flow between project participants and project stages. (ISO 19650-1, 2017; Scheffer *et al.*, 2018)

The project delivery team divides into three levels. The appointing party (usually the client or asset operator) represents the highest level. They receive approved information from the lead-appointed party (level 2). The lead-appointed party is responsible for merging data from all appointed parties (level 3) into the federated model. Level 3 may divide into several sub-levels as one appointed party may subdivide their tasks. (ISO 19650-2, 2017)

Figure 2.4 visualises the concept of the roles in the project delivery team. Further, we simplify this scenario and differentiate between two parties: An appointing party (the receiver of information), and the appointed party (the provider of information).

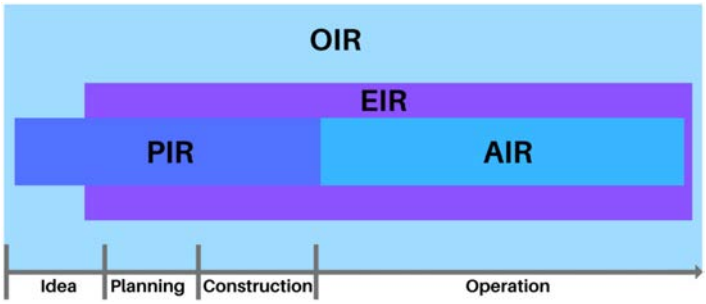


Figure 2.5: Information requirements throughout the asset's lifecycle. (based on Scheffer *et al.* (2018))

2.3.2 Information Requirements According to ISO 19650-1

BIM project execution requires certain information at certain stages. ISO 19650-1 (2017) distinguishes in three stages: the definition of information requirements, the planning for information delivery, and information delivery. The latter generates information in the form of models that occur in different types, such as 3D models, 2D plans or equipment schedules. All models must meet the requirements defined by the client. To obtain all the required information, the client assigns deliverables to the project participants. (Scheffer *et al.*, 2018)

The ISO standards distinguish between information requirements and information delivery plans. An information requirement specifies how information is produced and what, when and who it is produced for (ISO 19650-1, 2017). An information delivery plan, however, regulates how the project delivery carries out specific aspects of the information management (ISO 19650-2, 2017). The following sections explain both concepts and how they correlate.

2.3.2.1 Information Requirements over the Asset Life Cycle

Figure 2.5 shows the information requirements described in ISO 19650-1 (2017) and depicts their meaning during the project phases. While Organisational Information Requirements (OIR) cover the whole lifecycle of an asset, Project Information Requirements (PIR) and AIR only occur partially. The PIR impact the development of an asset, and the AIR influence the operation. Lastly, the EIR arise throughout all project phases apart from the first.

The owner or operator of an asset defines the OIR. These aim at realising organisational goals. OIR base on strategic asset management, regulatory obligations or policymaking. Concerning the life cycle of the building, the asset management phase influences the life and resulting costs decisively. Thus, OIR mainly affect the AIR, but also serve as the basis for defining the requirements for the Project Information Model (PIM). (Scheffer *et al.*, 2018)

Project Information Requirements (PIR) support strategic objectives on a project level. Thus, the appointing party identifies these information requirements for each critical decision. Additionally, repeat *"clients may develop a generic set of PIR that can be adopted, with or without amendment, on all of their projects"* (ISO 19650-1, 2017).

The Project Information Model (PIM) functions as the basis for the Asset Information Model (AIM) and affects the project delivery phase as it provides the required information for use cases such as clash detection, cost estimation or scheduling. (Scheffer *et al.*, 2018)

Asset Information Requirements (AIR) establish the information required to answer the OIR. They organise in a way that asset management appointments can incorporate them to support decision-making. The derived Asset Information Model (AIM) supports asset management processes and represents the interest of the asset owner or operator. The AIM contains both graphical and non-graphical information. Additionally, an existing AIM may serve as a source for the PIM. For instance, the AIM delivers information for the project brief. (ISO 19650-1, 2017; Scheffer *et al.*, 2018)

The appointing party specifies EIR. They define the data and information the appointed party needs to provide during information exchange. EIR mostly cover PIR but may also include parts of the AIR. As project delivery teams in construction projects typically consist of sub-appointments with higher hierarchy, the appointed party may sub-divide and pass on the EIR they have received to sub-appointments. (Scheffer *et al.*, 2018)

2.3.2.2 Information Delivery Plans

PAS 1992-2 (2013) differentiates in three information delivery plans: Task Information Delivery Plan (TIDP), Master Information Delivery Plan (MIDP) and BEP. Figure 2.6 describes how they interact with each other. PAS 1992-2 (2013) defines the TIDP as an internal document for planning teams. Each team indicates responsible team members for each deliverable. Furthermore, the TIDP includes milestones aligned with the programs in the MIDP.

According to PAS 1992-2 (2013), the MIDP summarises information deliverables throughout the project. It includes, amongst others, models, drawings and specifications. The planning teams develop the MIDP according to their TIDP.

Finally, the project team creates the BEP based on the PIR. The BEP describes how the team manages and exchanges information during the project. For this reason, the BEP defines processes to create models in accordance with the EIR. In general, two different kinds of BEP exist, the pre-appointment and post-appointment BEP.

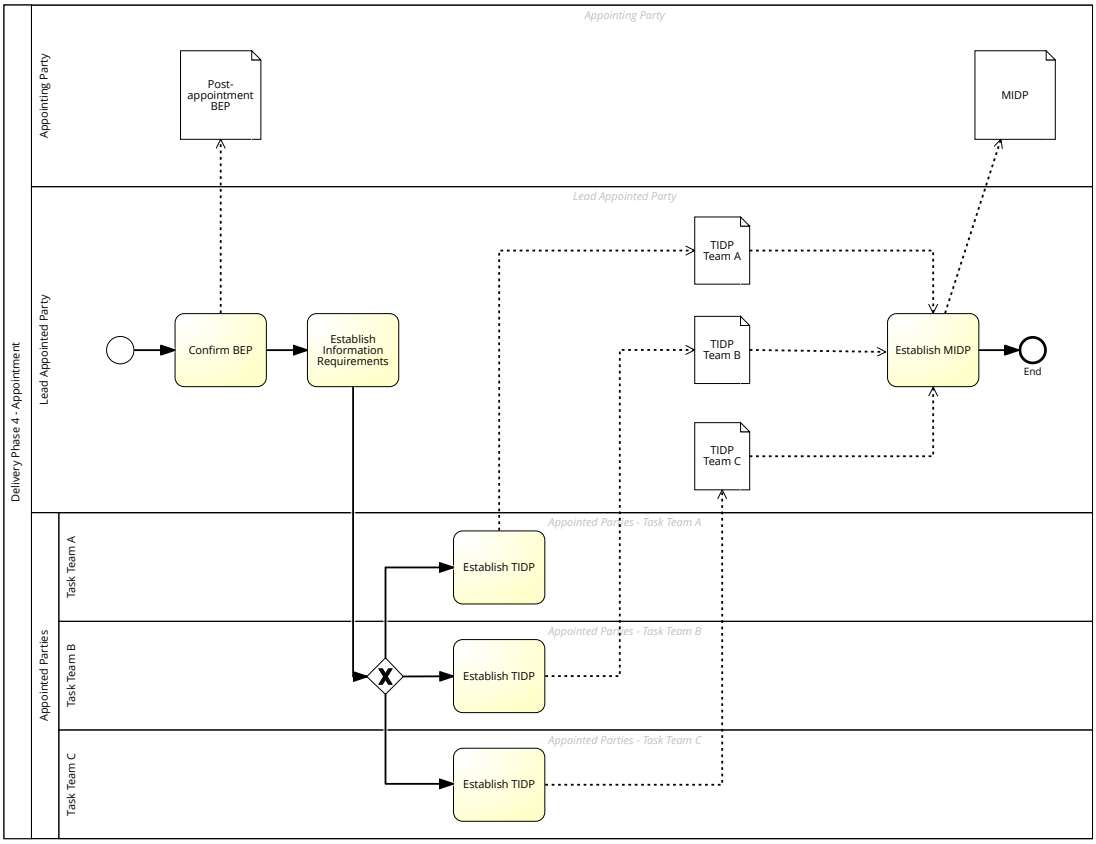


Figure 2.6: The lead-appointed party confirms the BEP. After establishing the Exchange Requirements, each appointed party aligns TIDPs with the BEP. Finally, the lead-appointed party develops the MIDP.

The pre-appointment BEP extends the EIR at least by the following aspects:

- Project Information Plan
- Project goals regarding collaboration and information modelling
- Major milestones
- Project information deliverables in the PIM

The post-appointment BEP focuses on information management within the project delivery team. Moreover, it identifies the abilities and limits of each project participant concerning information management. As projects grow over time and fuzziness decreases (Abualdenien & Borrmann, 2019), the content becomes more complex, and so do the information requirements. For this reason, the post-appointment BEP implements the following aspects additionally:

- Exchange Information Requirements
- Concerning project management:
  - Roles and responsibilities
  - Milestones according to the project program
  - Authorization and approval of data
- Concerning planning and documentation:
  - Revised Project Information Plan to confirm the capability of the supply chain
  - Processes for collaboration and information modelling
  - Responsibilities throughout the supply chain
  - Task Information Delivery Plan
  - Master Information Delivery Plan
- Concerning standards and methods:
  - Project Information Model orientation and origin
  - Naming conventions for files
  - Project tolerances
  - Abbreviations, annotations, dimensions, and symbols
- Software solutions:
  - Software versions
  - Data exchange formats
  - Data management systems

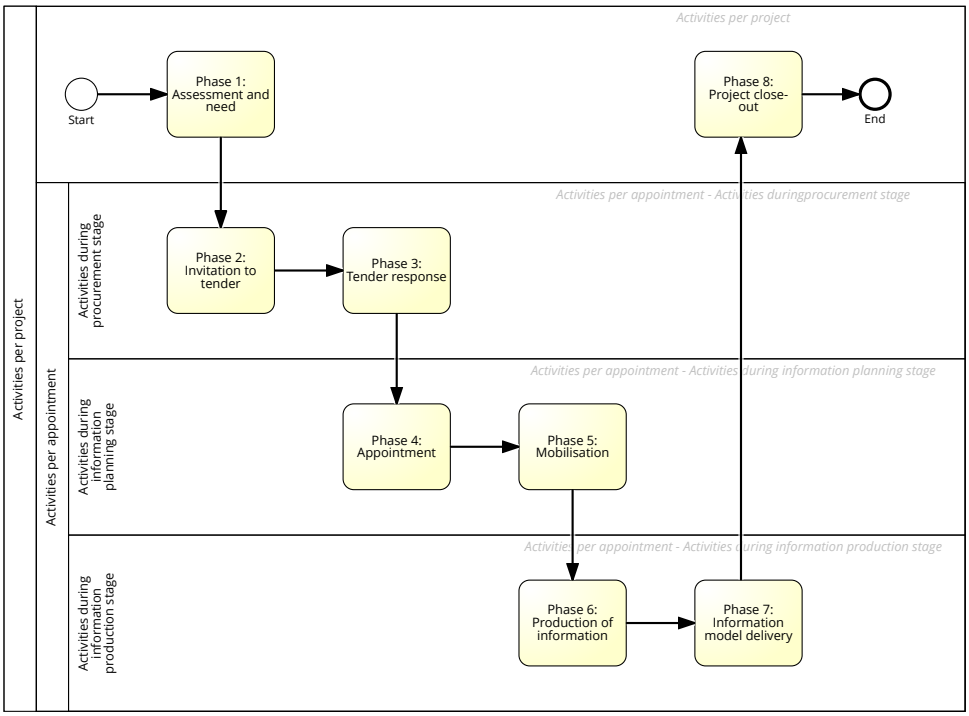


Figure 2.7: The asset delivery process divides into eight subprocesses. (based on ISO 19650-2 (2017))

2.3.3 The Information Delivery According to ISO 19650

This thesis focuses on information management during project delivery. Figure 2.7 shows how to manage information according to ISO 19650-2 (2017). Especially the first phase is significant as the project team develops the PIR. The appointing party defines milestones, project standards, information delivery methods, as well as a Common Data Environment (CDE). In phase 2, the appointing party determines EIR. Here, they also define the LOG and LOI (compare section 2.2). In Phase 3, each appointed party responds to the EIR in their pre-appointment BEP. At this stage, the delivery team must express its methods and strategy for information management. During the appointment (phase 4), the project team confirms the BEP, and appointed parties must establish their EIR for expected information exchanges. During mobilisation (phase 5), the appointing party and the appointed party collaborate to agree on critical roles, responsibilities and an information delivery plan (UK BIM Alliance, 2019). Throughout both phases, 6 and 7, the project teams generate the actual information and contribute it to the PIM. If this model complies with the EIR, the appointing party accepts the model and closes the project in phase 8.

2.4 Technical Implementation of EIR

The section above explained the concepts of ISO standards regulating information management in BIM projects. The EIR chiefly receive importance in context with this work. Hence, to understand the concept better, the following section describes significant relationships of EIR and puts them into context with the regulations published by buildingSMART. Eventually, the chapter describes the technical implementation of exchange information requirements using mvdXML.

Note: The abbreviation EIR may interfere with the abbreviation Exchange Requirements (ER). Both are widely used in current regulations. While ER rather refer to the technical implementation of information requirements, EIR instead relate to project management. The following chapter always uses the abbreviation of the cited work.

2.4.1 Model View Definitions and Exchange Requirements

An IFC model saves multiple topic schemas. Each comprises their persistent idea, e.g., scheduling, fabrication or structural analysis, and the IFC schema unites them in one single model. When it comes to information exchange, however, project participants transfer information related to one specific topic. Thus, they need to query the required information of a BIM model exclusively, as information exchange is smoother whilst sharing fewer data. (Wix & Karlshøj, 2010; Aram *et al.*, 2010)

The National BIM Standard proposes MVDs to close this gap. A Model View Definition (MVD) is a subset of the IFC schema containing all the information that a particular exchange scenario of an AEC project requests (Eastman *et al.*, 2011; Venugopal *et al.*, 2012). Hence, an MVD brings the information requirements in context with the IFC schema by defining a subset of the IFC schema. They can be broad and define almost the entire schema but also be very specific and only define a couple of elements (buildingSMART, 2019d). In general, every IFC file follows an MVD. Exporting a file from a software follows the MVD that determines how to export and derive the information.

Within an exchange scenario, one or multiple Exchange Requirements (ER) describe what information the model must define. ER depend on the topic ideas mentioned above and determine the required information to exchange data between two business processes (Venugopal *et al.*, 2015). They structure needed properties by model elements. Such a property can either be optional or mandatory. Moreover, the ER regulate the unit and data type of properties. (Borrmann *et al.*, 2018; Wix & Karlshøj, 2010)

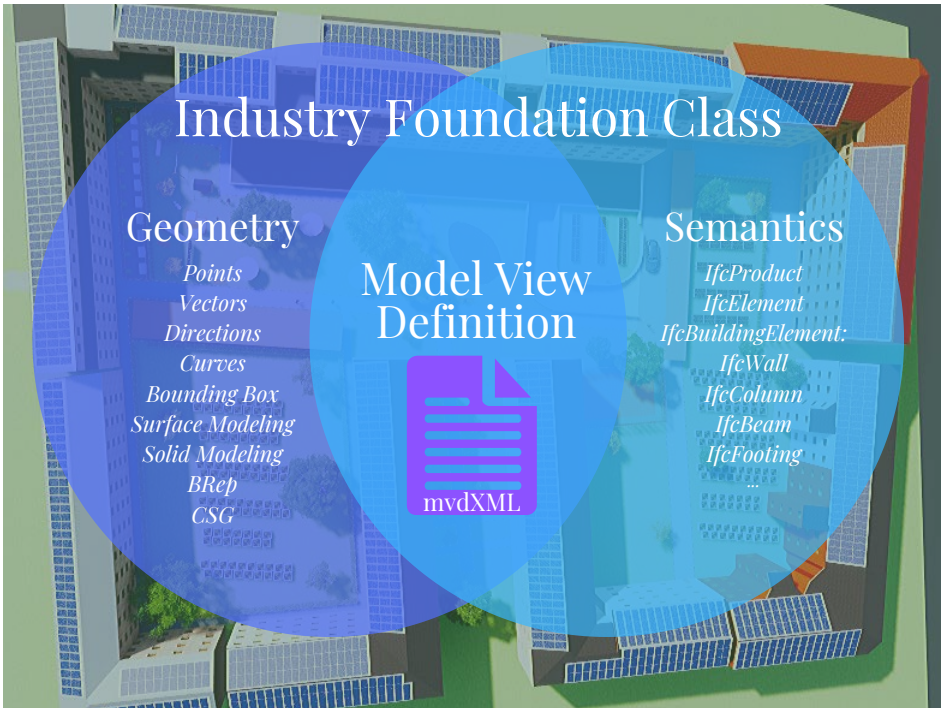


Figure 2.8: mvdXML describes a subset of the IFC schema technically. (Speiser, 2019)

The ER structure into three parts: a header, an overview, and an information section. The header lists administrative information. The overview section describes the objectives of the ER textually. Lastly, the information section represents the technical part of the ER. (Wix & Karlshøj, 2010; Speiser, 2019)

Exchange requirements integrate MVDs into the ISO standard described in section 2.3. Thus, ER convert EIR in a technical concept (ISO 29481-1, 2016).

2.4.2 Purpose and Origin of mvdXML

mvdXML formalises MVDs and ER in an electronic format (Chipman *et al.*, 2016). Figure 2.8 visualises the purpose of mvdXML within IFC. mvdXML implements two functionalities: firstly, it defines the partial schema for the MVD based on IFC, and secondly, it validates IFC data. The features limit the scope of IFC depending on their use case and ease the querying of data from the model, respectively. In this context, mvdXML for validation becomes significant. (Chipman *et al.*, 2016)

The mvdXML schema relies on the Extensible Markup Language (XML). XML is a platform-independent language to store data and structure hierarchical data with ease. Both characteristics have made XML a commonly accepted standard for information exchange. Before looking into the structure of mvdXML, the following paragraph introduces the basic concepts of XML. Kahate (2009) provides in-depth information about this topic.

```
1 <Property DataType="String" Description="The Discpline corresponding to the
    IMIR" Code="1" Name="e8e877a5-c3e8-4da3-963f-b1e1beec01eb" Guid="
    e8e877a5-c3e8-4da3-963f-b1e1beec01eb">
2     <PropertySets>
3         <PropertySet ref="078b8c8c-3721-4038-a5bc-8527b8a62dc0" />
4     </PropertySets>
5     <PropertyValues>
6         <PropertyValue ref="163fb8e9-c91e-44fa-92d4-b34ef41f0340" />
7         <PropertyValue ref="406f4b3c-12ad-46ad-9313-3fb3e512dc9a" />
8         <PropertyValue ref="4015dabf-69c3-4837-9b39-9a9efc0ea018" />
9     </PropertyValues>
10 </Property>
```

Listing 2.1: Excerpt from a simple file introducing the syntax and structure of XML.

Listing 2.1 shows a short excerpt from an XML file to introduce the standard. In general, there are three types of information: tags (or elements), attributes, and values. In Listing 2.1, *Property* represents a tag: they always start with an opening (<) sign and closing (>) sign. Within both symbols, a tag can define attributes. In Listing 2.1, *Property* implements the attribute *DataType* with the value *"String"*. Attributes use the equals symbol (=) to implement their value that starts and ends with double quotation marks ("). After the > symbol of an element, this tag can implement further tags of a lower hierarchy. For instance, *PropertySets* is tag of the next level. XML does not limit the levels of the hierarchy. To define an element of the same level, an element must conclude by using the slash symbol (/): line 4 completes the *PropertySets* tag (*</PropertySets>*). (Kahate, 2009)

The mvdXML data structure extends this concept by providing standardised elements and hierarchies. However, as the structure of both is the same, mvdXML is also platform-independent and usable for most software applications. (Chipman *et al.*, 2016)

2.4.3 MvdXML Schema

mvdXL represents the root of the schema that implements two main tags: *Templates* and *Views*. These elements sub-divide further. Not all elements are significant in this context. Thus, figure 2.9 shows a reduced version of the schema that the following section describes in detail. The overview refers to Chipman *et al.* (2016) and updates findings from a previous report from the author (Speiser, 2019).

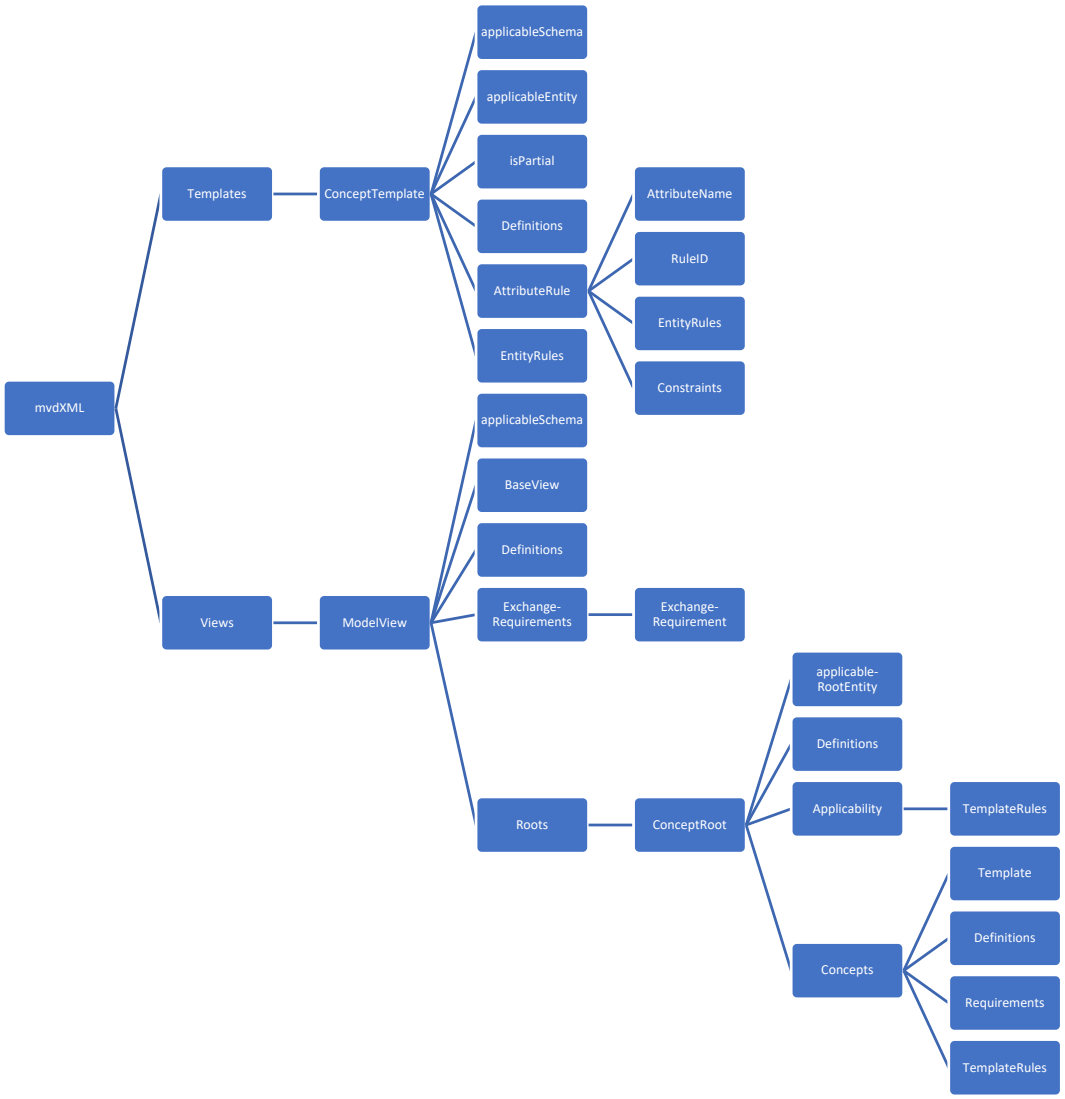


Figure 2.9: Simplified schema of the mvdXML.

2.4.3.1 Templates

The *Templates* tag comprises a list of *ConceptTemplate*s. Each ConceptTemplate defines a graph within a referenced IFC schema starting from an applicable entity. From there, the graph follows attributes and entities until it reaches individual attributes that contain the requested information. In summary, the graph comprises the part of the IFC schema needed for this functionality. Thus, a *ConceptTemplate* includes the following attributes:

- *applicableSchema* sets the default IFC schema the template refers to (IFC2x3, IFC4).
- *applicableEntity* represents the root of the graph. MVD tools use it to filter for particular entities.
- *isPartial* indicates if the *ConceptTemplate* is part of another *ConceptTemplate*.
- *SubTemplates* is set of *ConceptTemplate*s that extends a *ConceptTemplate* by specifying the graph.
- *Rules*: a collection of attribute definitions that relate to the root entity or their subtypes. The *Rules* element structures in *AttributeRules*, implementing *EntityRules*, referring to *AttributeRules*, and so on.

An *AttributeRule* defines *"the specification of an attribute on an entity, with related constraints, and/or entity rules"* (Chipman *et al.*, 2016). It implements an *AttributeName*, *RuleID*, *Constraints*, and *EntityRules*.

- The *AttributeName* represents the name of an entity in the IFC schema.
- The *RuleID* optionally defines an identifier that the MVD uses to refer to this concept.
- The *Constraints* is a set containing *Constraint* elements. Each implements a restriction of an attribute.

AttributeRules may reference an *EntityRules* defining a set of *EntityRule* elements. Each *EntityRule* represents the specification of the referenced entity, enumeration or type. An *EntityRule* consists of:

- an *EntityName* representing the name of the referenced type
- a *RuleID*
- *Constraints*
- *AttributeRules*

2.4.3.2 Views

The *Views* tag defines a set of *ModelView* elements. Each*ModelView* explains how to use the *ConceptTemplates* and comprises:

- an *applicableSchema*
- a *BaseView* that references a base MVD if this *ModelView* extends another *ModelView*
- *ExchangeRequirements*: A collection of *ExchangeRequirement* elements. Each element covers the requirements for a specific exchange scenario.
- *Roots* defining a list of *ConceptRoot* elements.

A *ConceptRoot* implements *Concept*s that define rules on applicable instances of the IFC schema. A *ConceptRoot* references an IFC entity that shall derive from *IfcRoot*. Additionally, it comprises:

- the *applicableRootEntity* attribute ruling to which IFC entity the *ConceptRoot* applies.
- a collection of *Concept* elements, each defining:
  - a *Template* that references a *ConceptTemplate*.
  - a collection of *Requirement* elements linking to *ExchangeRequirement*s. This tag determines whether an *ExchangeRequirement* applies to this *Concept*.
  - a hierarchically structured tree of *TemplateRules*. The tree uses logical operators to link its *TemplateRule* elements.
- an *Applicability* specifying the applicable instances of the IFC schema by applying *TemplateRules* as conditions.

A *TemplateRule* specifies rules using standardised grammar. It consists of a *Description* and a *Parameter*. The *Parameter* uses the *RuleID* to reference an *AttributeRule* or *EntityRule*. This concept enables the *TemplateRule* to retrieve data from the IFC model. Section 3.1 describes how to use mvdXML to validate an IFC model by defining *TemplateRules* and the *Applicability* for them.

2.4.4 MvdXML to Validate IFC Documents

Three entities of the schema are of particular interest to validate an IFC model. For that reason, the following section describes the structure and purpose of them starting from the bottom.

```
1 <ConceptRoot uuid="0e93f597-f5e1-475b-87a7-eb007993a50d" name="All External
      Walls" applicableRootEntity="IfcWall">
2     <Definitions/>
3     <Applicability>
4         <Template ref="5c252c86-5bff-4372-9a27-b794069f9fbb"/>
5         <!-- Applicability: apply concepts only to external walls-->
6         <TemplateRules operator="and">
7             <TemplateRule Parameters="O_PsetName[Value]='Pset_WallCommon'
                  AND O_PName[Value]='IsExternal' AND O_PSingleValue[Value]=
                  TRUE"/>
8         </TemplateRules>
9     </Applicability>
10    <Concepts>
11        <Concept>
12            <!-- Test #1: check existence of property1 -->
13        </Concept>
14        <Concept>
15            <!-- Test #2: check value of property2 -->
16        </Concept>
17    </Concepts>
18 </ConceptRoot>
```

Listing 2.2: This excerpt of an mvdXML file demonstrates the structure of the *ConceptRoot* entity. The *Applicability* selects all objects that the *Concept*s (listing 2.3) validate.

The *ConceptRoot* tag implements one ore more *Concept*s, each defining rules. These rules check whether an entity conforms to the requirements or not. Listing 2.3, for instance, rules that the objects of the IFC document must define the value 'Concrete' or 'Masonry' in the 'Material' property of the 'Pset_ks' property set (line 10-13). The mvdXML schema introduces two possibilities to determine applicable objects for these *Concept*s within the *ConceptRoot*.

Listing 2.4 illustrates an example. Line 1, 'applicableRootEntity=IfcWall', determines that all concepts of this root exclusively apply to IFC objects of the *IfcExpressType* 'IfcWall' (first possibility). This element represents the root of the graph that defines a subset of the IFC schema. The *Applicability* tag specifies this selection. Here, the *TemplateRules* tag implements conditions (second possibility). For instance, the rule in line 7 introduces that the concept only applies to external objects.

To sum up, the *Concept* in Listing 2.3 validates if all external 'IfcWall' instances in the IFC model have either the material 'Concrete' or 'Masonry'. This is only possible, however, as the *ConceptTemplates* tag defines how to retrieve data from the IFC schema. Therefore, it implements *AttributeRule*s. These rules describe how to access information in the model and what type they have. Listing 2.4 enforces three rules that apply to the applicable Entity 'IfcPropertyValue' (line 1). The first rule retrieves the data of the Attribute 'Name' (line 3). Their type is 'IfcIdentifier' (line 5), and within the mvdXML document, it can be retrieved by using the ID 'PName' (*RuleID*, line 3). To access the 'NominalValue' of the 'IfcPropertySingleValue', line 13 implements 'PSingleValue'.

```
<!-- Test #1: check value of Material property -->
<Concept uuid="983ddc5d-c0c8-47c9-8491-97add7677139" name="load bearing walls required to have property 'Material'">
    <Definitions/>
    <!-- References aConceptTemplate defined in the Templates tag -->
    <Template ref="5c252c86-5bff-4372-9a27-b794069f9fbb"/>
    <Requirements>
    <!-- References a ExchangeRequirement of the ModelView tag -->
        <Requirement applicability="export" exchangeRequirement="ae70f764-938b-4cf7-9814-c29a47f56b0e" requirement="mandatory"/>
    </Requirements>
    <TemplateRules operator="or">
        <TemplateRule Description="1.1.1" Parameters="O_PsetName[Value]='Pset_ks' AND O_PName[Value]='Material' AND O_PSingleValue[Value]='Concrete'"/>
        <TemplateRule Description="1.1.2" Parameters="O_PsetName[Value]='Pset_ks' AND O_PName[Value]='Material' AND O_PSingleValue[Value]='Masonry'"/>
    </TemplateRules>
</Concept>
```

Listing 2.3: *Concept* checking properties of objects of the IFC schema

```
<ConceptTemplate uuid="6655f6d0-29a8-47b8-8f3d-c9fce9c9a620" name="Single Value" applicableSchema="IFC2x3" applicableEntity="IfcPropertySingleValue" isPartial="true">
    <Rules>
        <AttributeRule RuleID="PName" AttributeName="Name">
            <EntityRules>
                <EntityRule EntityName="IfcIdentifier"/>
            </EntityRules>
        </AttributeRule>
        <AttributeRule AttributeName="Description">
            <EntityRules>
                <EntityRule EntityName="IfcText"/>
            </EntityRules>
        </AttributeRule>
        <AttributeRule RuleID="PSingleValue" AttributeName="NominalValue">
            <EntityRules>
                <EntityRule EntityName="IfcValue"/>
            </EntityRules>
        </AttributeRule>
    </Rules>
</ConceptTemplate>
```

Listing 2.4: *ConceptTemplate* tag in a mvdXML document to retrieve property values.

2.5 Issue Communication in BIM projects

Shafiq *et al.* (2012) state that the inconsistent nature of BIM models impedes collaboration in BIM projects. Providers of information often do not fulfil requirements. To communicate missing information with others, buildingSMART has developed the BIM Collaboration Format (BCF) since 2009 (buildingSMART, 2019b). This section describes the motivation of BCF and the structure of the the generated files.

2.5.1 Motivation

BIM projects require high technical knowledge throughout the contributing disciplines but also a well-implemented interdisciplinary collaboration. The different disciplines communicate information such as instructions, comments or issues. However, the data is often obsolete, ambiguous or inconsistent as different disciplines interact using various platforms and file formats. (Dossick & Neff, 2011)

For that reason, buildingSMART has introduced BCF for sharing model-based issues by referring to previously published IFC models. It transfers data about an issue in XML format from one application to another. This issue references a view, an image, IFC coordinates and the concerned elements from the IFC model. As the elements are referenced via their Globally Unique Identifier (GUID), all applications can uniquely interpret BCF. Hence, BCF enables open-based communication in BIM projects and enhances interoperability. (buildingSMART, 2019b)

2.5.2 Distinction in Markup and Visualisation

An issue using BCF addresses one topic comprising three files: The markup (*.bcf), the visualisation (*.bcfv), and a snapshot (*.png). The markup and visualisation provide XML-formatted information. Both are mandatory. An optional image visualising the topic represents the snapshot. (buildingSMART, 2019a)

Listing 2.5 shows the structure of the markup. Lines 3-7, the header, provides information about the project the BCF file refers to and the date of generation of the file. Lines 9-11, the topic, define information about the BCF itself by saving the GUID . Finally, lines 12-17 store the actual communication. Here, the file records discussions between different parties by using comments. Each comment also obtains its unique identifier. Thus, parties can refer to previous comments using the GUID of that comment. (buildingSMART, 2019a)

```
<?xml version="1.0" encoding="utf-8"?>
<Markup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="
    http://www.w3.org/2001/XMLSchema">
  <Header>
    <File IfcProject="3ZGD7y6S5209$mGLi_sPlj" isExternal="false">
      <Filename>D:\Dokumente\Uni\Masterarbeit\GUITestDocuments\
          rac_basic_sample_project.ifc</Filename>
      <Date>2020-03-11T17:59:17.4535225+01:00</Date>
    </File>
  </Header>
  <Topic Guid="617b091c-47c4-4447-b7b8-8bd4dc202b2b">
    <Title>Issue regarding 2pfAHb2EL46hq_sMVbImE4</Title>
  </Topic>
  <Comment Guid="f695d686-467c-43a5-bd0a-fc5b3fa5c65c">
    <Status>Error</Status>
    <Date>2020-03-11T17:59:17.4535225+01:00</Date>
    <Author>unknown user</Author>
    <Comment>102  Does not implement the Property: floor_finish</Comment>
    <Topic Guid="617b091c-47c4-4447-b7b8-8bd4dc202b2b" />
    <ModifiedDate>2020-03-11T17:59:17.4535225+01:00</ModifiedDate>
  </Comment>
</Markup>
```

Listing 2.5: The markup file provides general information about the issue and records the communication.

```
<?xml version="1.0" encoding="utf-8"?>
<VisualizationInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Components>
    <Component IfcGuid="2pfAHb2EL46hq_sMVbImE4" Selected="false" Visible="
        false" />
  </Components>
  <Lines />
  <ClippingPlanes />
</VisualizationInfo>
```

Listing 2.6: The Visualization Information stores components related to the BCF file.

The second part of a BCF issue generates the viewpoint visualising the issue (listing 2.6). It contains information about the related topics, camera settings, and concerned instances from the IFC model. Besides several optional elements, the 'Components' tag comprises the only obligatory element. This tag (lines 3-5) can refer to several elements of the IFC model in the 'Component' tag by storing the GUID of the IFC element. Both files meet in one folder. This folder's name corresponds to the GUID of the topic.

2.6 Classify and Structure Deliverables

As already mentioned, different people may refer to the same object, using different words. Several institutions have tackled this problem by classifying terms. For instance, the buildingSMART Data Dictionary (bsDD) introduces a library to classify objects and properties related to the AEC industry. But, instead of referring to words, they use a GUID. Hence, the labelling of information can change without losing meaning. However, a GUID has a complex syntax: *0GSQP$JDv5PALB8zok7r9q*. (buildingSMART, 2020b)

The NBS (2015) also approaches this issue. They published their system 'Uniclass' to classify products and objects related to the AEC industry. Besides using terms, they also introduce unique identifiers using a more straightforward format. For example, 'Stairs' belong to the code 'EF_35_10'. This code is unique and follows the structure of a WBS code. The following sections describe the origin of WBS and explain how to apply them to classifications such as Uniclass. (The NBS, 2015)

2.6.1 Work Breakdown Structure

Managing projects implies estimating the time and cost of tasks. The uncertainties will reduce if managers subdivide the tasks hierarchically into small deliverables. This process eases estimating the cost and time of each deliverable. The concept of subdividing work into deliverables is called WBS. (Lewis, 2007; Project Management Institut, 2012)

The following example explains the concept. A student wants to clean their apartment. As the condo comprises several rooms, cleaning each room represents one deliverable. Cleaning a room can subdivide into further subtasks: clean the curtains, hoover the floor, and dust furniture. The deliverables describing those tasks are: cleaned curtains, cleaned floor, and cleaned furniture. The student can subdivide these deliverables again. For instance, a cleaned floor requires to tidy it, hoover it, and wipe it. Figure 2.10 depicts the example graphically. (Lewis, 2007)

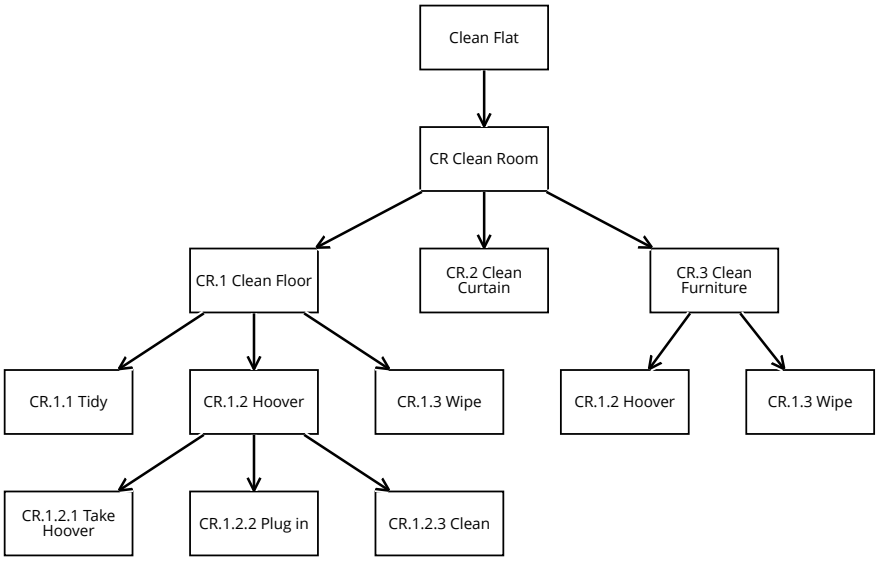


Figure 2.10: WBS subdivides the tasks in the smallest possible deliverable. (based on Lewis (2007))

This figure already assigns codes to the entities. For example, 'Clean Room' belongs to the code 'CR'. All sub-elements use this code and add their own code corresponding to the level. This concept introduces two benefits. Firstly, the codes are unique and, secondly, they can identify parent entities in a Work Breakdown Structure as the code comprises the codes of the parent entities. For example, code 'CR.1.2' implies that its parent is 'CR.1', and 'CR' represents its root.

2.6.2 Usage in AEC industry

Uniclass bases on WBS codes. Uniclass subdivides into twelve categories, such as 'Products', 'Systems', or 'Tools and Equipment'. Each of them defines several root elements and sub-elements. For example, Uniclass defines the term 'Stairs' and assigns it to the code 'EF_35_10'. This code concludes that

- 'EF_35_10' is a sub-element of 'EF_35: Stairs and ramps'.
- 'EF_35' is a root of the category 'EF: Elements/functions'.
- 'EF_35_10' is unique. (The NBS, 2015)

2.7 Summary

ISO 19650-1 (2017) and ISO 19650-2 (2017) establish a concept of managing information in a BIM project during project delivery and asset management. They introduce three perspectives in such a project: the view of the owner, the user and the project. Talking about the project's perspective, the project delivery team generates information that the asset owner requires for their Project Information Model (PIM). These are defined in the Project Information Requirements (PIR). ISO 19650-1 (2017) proposes to use a generic set of PIR for repeat clients. Similar to this, chapter 5.2 describes a data model to specify a generic set of Exchange Information Requirements (EIR). EIR contain requirements the appointed party has to provide for an exchange scenario.

ISO 19650-2 (2017) regulates how the project team shall carry out information management. Therefore, it introduces three delivery plans defining responsibilities and tasks for different stakeholders. Here, they differentiate between the appointing party, the lead-appointed party, and the appointed parties. The appointed parties summarise their tasks and deliverables in a Task Information Delivery Plan (TIDP). The lead-appointed party creates a Master Information Delivery Plan (MIDP) from it and informs appointed parties to adapt their MIDP to potential changes. The MIDP is part of the post-appointment BIM Project Execution Plan (BEP). This plan finally rules information management and exchange to achieve defined BIM goals. Hence, it implements EIR for each data exchange scenario.

The IFC standard is the only open-based and license-free standard. After initially missing acceptance, plenty of software vendors implement this standard for information exchange nowadays. The IFC schema, however, is manifold and complex. Thus, buildingSMART introduced MVDs. They define a subset of the overall schema, depending on exchange requirements. The mvdXML converts MVDs into a technical format. Moreover, mvdXML allows validating models whether they correspond to the MVD or not. mvdXML bases on the XML, which enables exchanging data platform-independently.

This work aims at defining information requirements on data models. Hence, the work relies on the concept of MVDs. Moreover, section 3.1 tests how to use mvdXML to validate IFC documents, and section 5.2 introduces how to structure information requirements. For this chapter, WBS come into play. They provide an opportunity to structure deliverables and assign unique identifiers to them. Besides this, appointed parties may not provide the data they have agreed on in the BEP. In this case, the appointing party can communicate the missing information using the BIM Collaboration Format (BCF).

Chapter 3

Current Situation

Information management in BIM projects divides into two aspects: define information requirements and validate if a BIM model implements the required information. This chapter investigates existing approaches for both. Firstly, it describes the process of creating information requirements and checking models if they apply to them. Secondly, the chapter illustrates how to use BimQ for defining information requirements.

3.1 Ifc Validation Based on MvdXML

Figure 3.1 indicates how to validate the information contained in an IFC model using mvdXML. Firstly, the appointed party identifies relevant information requirements (EIR) and derives an mvdXML document. Once the project starts, they send EIR to the appointed party who incorporate the required information to their BIM model. After that, they send the model to the appointing party. The recipient validates the model by using the mvdXML file. If some information is missing, the appointed party must revise their model. To sum up, this process divides into three tasks:

- Identify information requirements
- Derive mvdXML document from the EIR
- Validate 3D model using mvdXML

The next sections explain the process shown in figure 3.1 by applying it to a simplified scenario.

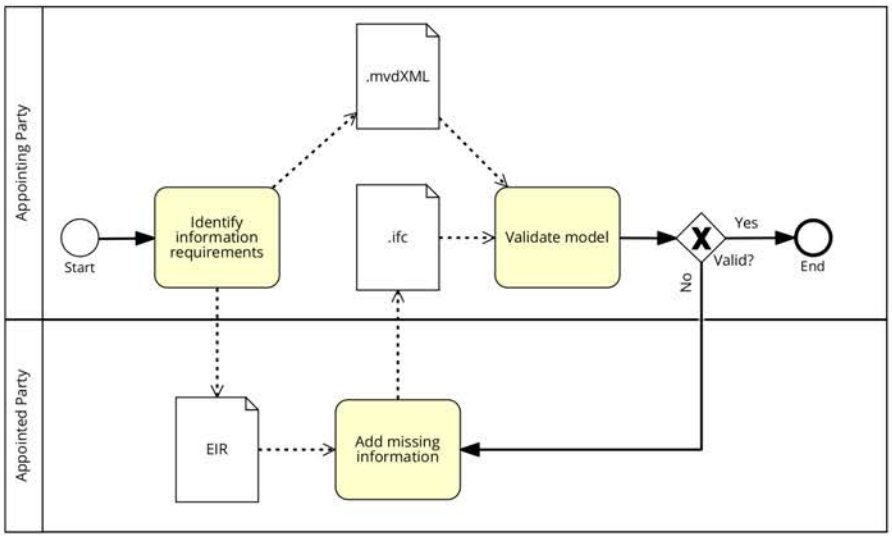


Figure 3.1: How to use mvdXML for model validation in a BIM project. (based on Speiser (2019))

Component Type: Wall

Property	Example Value	Property	Example Value
Thickness	0.14m	Strength	C30/37
Material	Concrete	Production Method	prefabricated
Fire Resistance Class	REI45	Exposure Class	XC4
Level	0		
LoadBearing	true		

Table 3.1: All external walls must define the properties in the table to the left. If the component's 'Material' equals 'Concrete' it must also implement the properties in the table to the right.

3.1.1 Identify EIR

In this shortened example, all elements of type 'Wall' must define the properties shown in table 3.1. If their material is concrete, they have to implement information about their strength, production method, and exposure class.

3.1.2 Derive the MvdXml Document

Section 2.4.3 describes the mvdXLM schema providing several options for model validation. This section compares two options structuring the XML tags introduced in section 2.4.4 in a way that allows mvdXML to validate IFC models. Both options must implement the *ConceptTemplates* to retrieve data from the IFC model. Listing 2.4 shows one of three Concept templates. The three templates enable accessing property sets and properties, as well as retrieving property values from individual elements. Besides this, both options define *ConceptRoot*s, including *Applicability* and *Concept*s. These elements differ in both approaches. While the first one implements strict applicability the second option bases on strict hierarchy.

```
<ConceptRoot uuid="0e93f597-f5e1-475b-87a7-eb007993a50d" name="All External Walls" applicableRootEntity="IfcElement">
    <Applicability>
        <Template ref="5c252c86-5bff-4372-9a27-b794069f9fbb" />
        <TemplateRules operator="and">
            <TemplateRule Parameters="O_PsetName[Value]='Pset_ks' AND O_PName[Value]='Material' AND O_PSingleValue[Value]='Concrete'"/>
        </TemplateRules>
    </Applicability>
    <Concepts>
        <Concept uuid="983ddc5d-c0c8-47c9-8491-97add7677140" name="Material:Concrete">
            <Template ref="5c252c86-5bff-4372-9a27-b794069f9fbb"/>
            <Requirements>
                <Requirement applicability="export" exchangeRequirement="ae70f764-938b-4cf7-9814-c29a47f56b0e" requirement="mandatory"/>
            </Requirements>
            <TemplateRules operator ="and">
                <!-- Check if exposure class, strength and production method exist-->
            </TemplateRules>
        </Concept>
    </Concepts>
</ConceptRoot>
```

Listing 3.1: Strict Applicability: The *Applicability* ensures that the concept only applies to elements defining the material 'Concrete'. The *Concept* checks if these components implement the required information (replaced by a comment). Annex A shows the full *ConceptRoot*.

3.1.2.1 Option 1: Strict Applicability

Strict applicability means that a *Concept* only applies to instances of the IFC model that satisfy previous requirements. It also implies that the hierarchy is flat and flexible. In other words, the *Applicability* tag of the *ConceptRoot* lists all prior information requirements, and the *Concept*s list the actual information to be tested. Hence, every property that requires further attributes implements one *ConceptRoot* entity.

In context with the previously developed information requirements (table 3.1), the derived mvdXML comprises two *ConceptRoot*s. ConceptRoot A applies to all elements of type 'Wall' and checks if they implement the properties shown in the left list of table 3.1. ConceptRoot B only applies to all elements whose material property equals 'Concrete' and checks the existence of the right part of table 3.1.

The *Applicability* tag of *ConceptRoot* A defines that exclusively elements that implement the property 'Component Type' with the value 'Wall' are applicable for the *Concept*s of this *ConceptRoot*. Besides this, *ConceptRoot* A defines one *Concept* that comprises five *TemplateRule*s. These rules make sure that all five required properties exist in every applicable element.

```
1 <Concept uuid="983ddc5d-c0c8-47c9-8491-97add7677139" name="All Elements">
2     <Template ref="5c252c86-5bff-4372-9a27-b794069f9fbb" />
3     <Requirements>
4         <Requirement applicability="export" exchangeRequirement="ae70f764
              -938b-4cf7-9814-c29a47f56b0e" requirement="mandatory" />
5     </Requirements>
6     <TemplateRules operator="and">
7         <!--Validate that ComponentType is set to wall-->
8         <TemplateRules operator="and">
9             <!--Check if thickness, fire rating, loadbearing and level
                  properties exist-->
10            <TemplateRules operator="or">
11                <TemplateRules operator="and">
12                    <!--Check if material exists-->
13                </TemplateRules>
14                <TemplateRules operator ="and">
15                    <TemplateRule Parameters="O_PsetName[Value]='Pset_ks'
                          AND_O_PName[Value]='Material' AND_O_PSingleValue[
                          Value]='Concrete'" />
16                    <!--Check if Strength, Production Method and Exposure
                          Class Exist-->
17                </TemplateRules>
18            </TemplateRules>
19        </TemplateRules>
20    </TemplateRules>
21 </Concept>
```

Listing 3.2: Strict hierarchy: One *Concept* tests all requirements. Hence, the *TemplateRules* comprise all conditions and the *Applicability* becomes empty.

Listing 3.1 shows *ConceptRoot* B in a simplified version. It structures similar to *ConceptRoot* A. The *Applicability* rules that the *Concept* only applies to element defining the material 'Concrete' (line 2-7) and the *Concept* tests (line 14-16) if these elements implement the properties 'Strength', 'Production Method' and 'Exposure Class'. The *Concept* only returns true if an element passes all *TemplateRule*s. The *TemplateRules* themselves were replaced by a comment to show the structure more lucid. Please look at appendix A to find the whole *ConceptRoot* A (listing A.1) and and ConceptRoot B (listing A.2).

3.1.2.2 Option 2: Strict Hierarchy

While strict applicability entails flexible hierarchy, strict hierarchy entails flexible applicability. The approach of strict hierarchy implies that one *Concept* applies to all elements, and the concept itself determines if the component is applicable or not. In other words, the *Applicability* tag of the *ConceptRoot* does not restrict to which elements the Concepts apply, and the *Concept* entity itself regulates that by defining *TemplateRules* hierarchically.

In context with the previously developed information requirements (table 3.1), the derived mvdXML comprises one *ConceptRoot* defining one *Applicability* and one *Concept*. This *Con-*

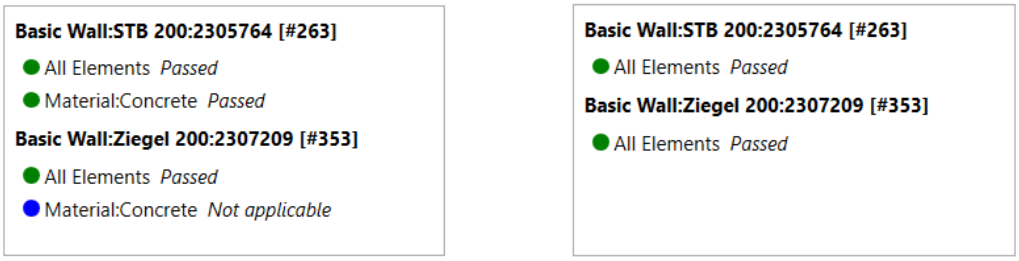


Figure 3.2: Validation of the model. Left: Option 1. Right: Option 2. Green implies that the concept has passed, blue that the concept is not applicable for this element.

ceptRoot applies to all elements. Hence, the *Applicability* does not restrict the selection. The *Concept* lists all further requirements hierarchically.

Listing 3.2 shows an excerpt from the mvdXML of option 2. The first *TemplateRule* of the *Concept* tests if the current element implements 'Wall' for the property 'Component Type' (line 7). If not, the concept has failed and the validation stops. For the elements that passed the first rule, line 9 tests the next level properties (table 3.1 left). As there are further requirements on all element defining concrete as their material, the Concept introduces a third level (lines 15 and 16).

3.1.3 Validation Using XBim Xplorer

The open-source software xBim Xplorer implements functionalities to validate an IFC model using an mvdXML file. The tested model (figure 3.3) consists of two walls with different properties; a concrete wall, and a masonry wall. The XbimXplorer tool applies the rules defined in the mvdXML to the model and graphically indicates if elements comply with them. Figure 3.2 compares the results of both options. The significant difference is that option 1 defines multiple *Concept*s. Therefore, it is easier to identify error sources. Option 2, however, only returns one passed/failed for every component. For MVDs with deep hierarchies, it is more complex to determine what exact information is missing.

3.1.4 Conclusion

Table 3.2 overviews the advantages and disadvantages of the two options. Strict applicability primarily stands out with the simple structure of the *Concept* elements. However, the rules mainly move to the *Applicability* element. The *Applicability* tag is empty for strict hierarchy. Thus, the Concept defines all rules. While the *Applicability* tag is simple, the Concept becomes complex.

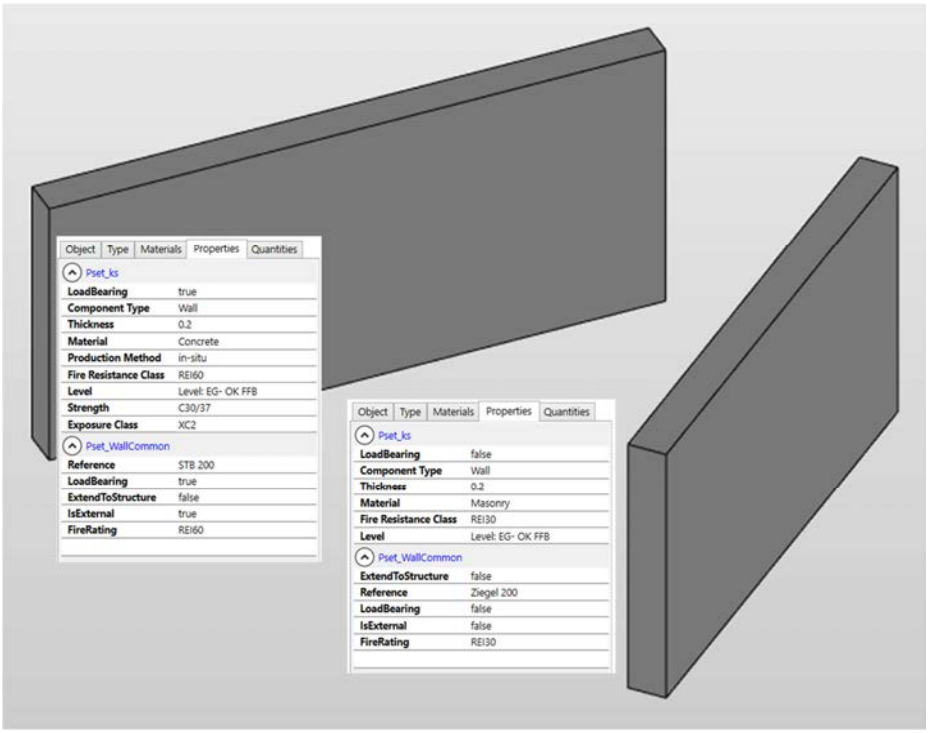


Figure 3.3: Two walls defining different properties.

The complexity of the *Applicability* and *Concept* tags primarily depends on two indicators: the depth of the hierarchy and the number of properties. While the strict applicability is simple for one or two levels, the strict hierarchy approach starts being simple for higher levels.

Finally, strict hierarchy files follow straight-forward rules. Thus, an algorithm can read and create these files with ease. For strict applicability, however, the algorithm requires referenced objects and may lose consistency. The last issue represents the reason why the author has decided to use the approach of strict hierarchy in context with this work.

	Strict applicability	**Strict hierarchy**
ConceptRoot	One for every property	One in total
Concept	Simple Concepts	One complex Concept
Applicability	Complex Applicability for deep hierarchy Simple Applicability for low hierarchy	No Applicability
Algorithm	Complex to read and create	Simple to read and write
Validation	Uniquely possible	Only one error for every component

Table 3.2: Comparison of both options regarding structure, complexity, readability, and usage.

3.2 Information Requirements with BimQ

BimQ is a web-based software application designed by AEC3 supporting clients to define their Exchange Information Requirements (EIR). It covers the process from the previous section. Here, the user creates information requirements for components depending on the project stage and the BIM Use. Moreover, project roles determine what party must deliver what information. Lastly, BimQ exports mvdXML document defining all the requirements on an IFC model. These requirements depend on the project role, project stage, and the BIM Use. This chapter summarises the workflow of BimQ and visualises how to use this for defining information requirements. The goal is to create an mvdXML document that can validate IFC models. (AEC3, 2020a)

3.2.1 Functionalities

A project in BimQ comprises three components: actors, project phases and use cases. All three define a name, code and description. The user is flexible in defining them. Thus, language or country-specific regulations do not limit the software. After defining the components, the user can assign use cases to project phases by using one-to-many relationships. Thus, one project phase references multiples use cases. This is helpful because one use case may occur during various project phases. Each relationship represents one exchange scenario that can implement information requirements. However, before defining requirements, BimQ asks to create the model components. (AEC3, 2020d)

BimQ enables the user to create their model components. The software tool does not restrict how to structure them. In their project template, they distinguish in domain-specific models, model elements, and properties. Multiple model elements are not strictly necessary. It is also possible to save all entities in one category. However, structuring them by categories eases filtering. Each model component can reference an IFC entity. For example, the user can define a property 'm12p1' and assign that to the property 'Material'. (AEC3, 2020c)

After creating all model elements, the user can define relationships between entities. For example, they can assign properties to components and components to models. From these relationships, BimQ derives hierarchically structured requirements. Hence, the final result is a tree. In this tree, the user can assign the elements to exchange scenarios. From this selection, the user can derive mvdXML files for one or multiple exchange scenarios. These files support model validation. (AEC3, 2020b)

Requirements for Architectural Model (A	Code	Type	IFC 2X3 TC1	P02-U01	P02-U02	P02-U03
ModelElement	-	Object	IfcProduct			
Common	1	Group	Pset_ks	x		
Component Type	1.1	Property	Pset_Shell.ComponentType	X	-	-
Wall	1.1.1	Value	*Wall*			
Thickness	1.1.1.1	Property (!)	Pset.Thickness	X	-	-
Slab	1.1.2	Value	*Slab*			
Column	1.1.3	Value	*Column*			

Figure 3.4: Requirements view: BimQ can structure elements hierarchically. The user can define codes and assign IFC entities. In the three columns to the right, the user allocates the elements to exchange scenarios.

3.2.2 Example

The following example pictures the usage of BimQ by applying it to a case. For the BIM Use Cost Estimation in the Conceptual Phase, the asset owner defines information requirements. Therefore, they create one element 'ModelElement', structuring all information requirements hierarchically. The owner also assigns the entities to IFC classes. Figure 3.4 shows the outcome.

All elements of type 'IfcProduct' must implement a 'Component Type' property in the 'Pset_Shell' property set. The component type can either have the value 'Wall', 'Slab' or 'Column'. Moreover, every wall requires to define the attribute 'Thickness'.

After creating the requirements, the owner can derive an mvdXML file for model validation. This document saves the requirements in the *ConceptRoot* entities. What the figure already indicates with the red warning, confirms looking at the mvdXML file. The generated file does not structure the rules hierarchically. Even though the user interface shows the tree view, the mvdXML export creates one *Concept* for each property. And this *Concept* does not define further restrictions regarding the applicability. Hence, the *Concept*s apply to more elements than they should. Figure 3.5 visualises this problem.

The model comprises four elements: A wall, a slab, stairs and a column. The column implements the 'ComponentType' property with the value 'Column'. Defining this property represents the only requirement for the element. However, as BimQ does not export the hierarchical structure, the mvdXML also requires the column to define the 'Thickness' property. Hence, the mvdXML returns a failed *Concept*, although the element does comply with all requirements. This may evoke multiple issues.

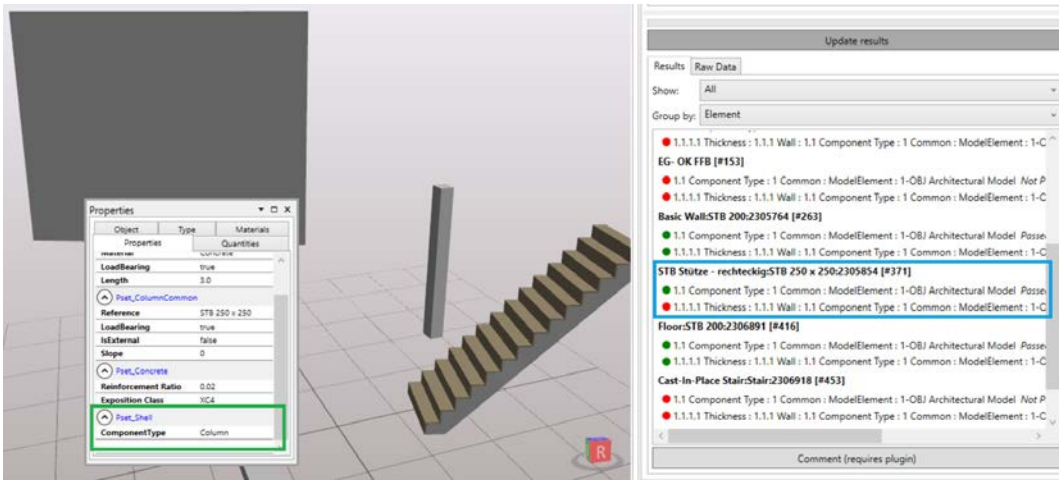


Figure 3.5: Checking the model with the xBim Xplorer visualises the problem: the column only requires to implement its component type (green). However, the mvdXML also requires it to implement a thickness (blue).

3.2.3 Conclusion

BimQ is a powerful tool to define information requirements. Using the software, in the way the vendor recommends, does not raise problems. Particularly, the flexibility to create model elements seems convincing. Its functionalities, however, do not cover the requirements for this work. The fact that BimQ does not export defined hierarchies represents the primary issue. Besides this, the author struggled with intense server problems while creating or deleting elements. Especially the latter seems to cause exceptions and errors as created properties dis- and reappeared continuously.

BimQ is a commercial software application. However, the author intends to develop a fully open-based approach using license-free standards such as mvdXML. Thus, BimQ does not comply with the author's vision. For that reason establishes chapter 5.2 a data model defining information requirement hierarchically. The introduced model supports exporting mvdXML files that include the hierarchy and fully validate IFC models.

3.3 Summary

mvdXML is a convincing standard for model validation. It firstly helps a user to define information requirements and, secondly, validates BIM models. In between, the project participants create their models and provide the information. The previous chapter has shown two approaches to define information requirements using mvdXML. Strict applicability entails a flexible hierarchy, while a strict hierarchy entails flexible applicability. Both concepts have their benefits. The main advantage of the strict hierarchy represents the straight-forward structure of the rules. Thus, an algorithm can read and create these files easily. For that

reason, the author has chosen to follow this approach in the upcoming chapters. Specifically, section 5.2 provides a data model that derives mvdXML files supporting the 'strict hierarchy' concepts.

BimQ supports asset owners to define their information requirements. With this, the user can create project actors, phases, and use cases and derive exchange scenarios. For each scenario, the owner can define information requirements and extract them as mvdXML files. These documents validate BIM models. Hence, the software provides a cohesive approach from defining requirements to model validation. However, using the tool for specific needs limits its usage. The author could not apply the functionalities to the given problem. In particular, the model does not export mvdXML files hierarchically.

Chapter 4

Enrich-IFC-Approach

The previous chapters provide the necessary background information to understand the problem. Besides this, they introduced current solutions and identified lacks. After that, the following chapter proposes the Enrich-IFC-approach to close the described gaps. This approach separates into two major aspects: the definitions of information requirements, and the enrichment of IFC models by translating given data. However, before introducing the approach, this chapter overviews the initial state to complete the insight regarding the problem.

4.1 Initial State

The National BIM Standard differentiates BIM projects in three levels. Depending on this BIM maturity level, the collaboration processes differently (McPartland, 2014). The AEC industry aims at reaching BIM Level 3. This level implies that project participants work on the same model at the same time and share data continuously. However, Borrmann *et al.* (2018) state, practical experience has shown that level 3 is not yet suitable for several reasons, e.g., it does not support accountability. Hence, BIM Level 2 is state of the art. Here, multiple parties work on their models and share them periodically to combine them in one coordination model. Several guidelines implement this collaborative approach by using the principle of domain-specific federated models. (Borrmann *et al.*, 2018; Preidel *et al.*, 2018)

The approach based on domain-specific federated models (compare figure 4.1) guarantees that model authors can only access domain-specific sub-models according to their responsibilities. Those sub-models, called domain-specific models, treat individual topics of the overall federated model. Hence, in the federated-model-approach, project participants create and maintain their domain-specific models exclusively and share them periodically. This evolves in a tremendous number of data-transition points that the project participants need to coordinate for assuring validity and consistency. (Preidel *et al.*, 2018)

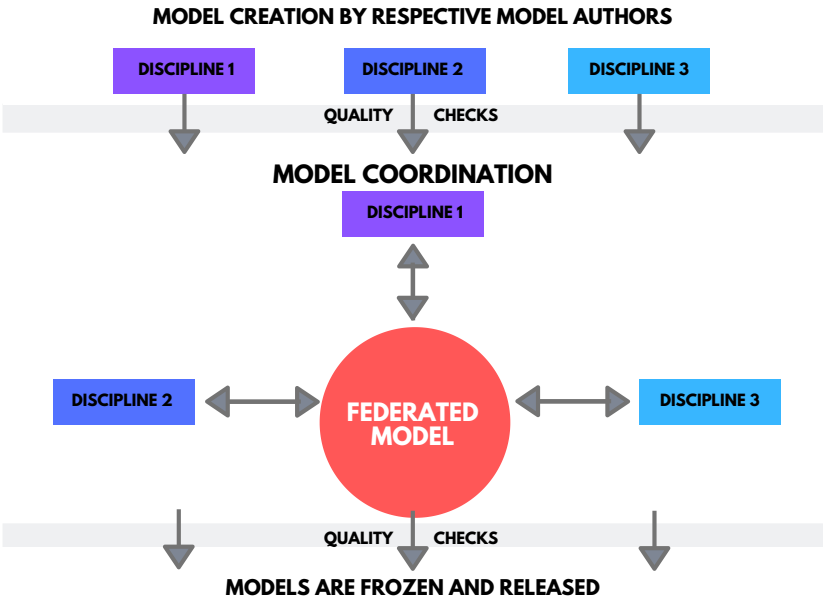


Figure 4.1: Principle of the federated model approach: Authors create domain-specific sub-models independently while coordinating them continuously. (based on (BCA Singapore, 2013))

Transferring the data from one party to another is called exchange scenario. The BEP regulates these scenarios. Each exchange scenario implements EIR, where the receiver of information (appointing party) defines what the provider of information (appointed party) must deliver (Borrmann *et al.*, 2018; Zhang *et al.*, 2013). Appointing parties and appointed parties require specific information and unite this in the EIR. The EIR, however, does not determine what form and syntax an appointed party must provide the data in. It is not yet common to agree on labelling and structuring standards.

For instance, Die Deutsche Bauindustrie (2018) published EIR defining requirements on several components. Table 4.1 shows an excerpt from the document. The document does not define the labelling conventions of information. However, automated processes require this to interpret the data. Besides this, a wooden beam does, for example, not have properties like 'Exposition Class' or 'Reinforcement Ratio'. Rather than that, the wooden beam may require information about the tensile strength. This receives special attention when it comes to model validation: How can an algorithm check whether a component must implement specific properties or not?

To sum up, there are two issues. (1) How to structure information requirements in a way that enables proper for model validation. (2) How to adapt a model's information to internal preferences in a way that allows for automatised downstream processes, and validate whether a model meets the requirements or not. The following sections propose a structure for semantic information requirements and show how to incorporate them into the regulations developed by buildingSMART.

Property	Example Value
Component Type	Beam
Classification	Uniclass EF_15_12_2
Height	1.20m
Width	0.30m
Material	Concrete
Load Bearing	true
Reinforcement Ratio	0.025
Exposure Class	XC3
Compressive Strength	C25/30
Production Method	prefabricated
Fire Resistance Class	F90
Visual Quality	visual
Level	0
IsExternal	false

Table 4.1: Exchange information requirements for a beam. (based on Die Deutsche Bauindustrie (2018))

4.2 Exchange Requirements and Their Structure

There are two dimensions of requirements for semantic information. The first dimension demands information to be existent. However, different disciplines may use different terms, referring to the same information. What an architect may call 'Insulation Factor', a building engineer may name 'U-value', and a product manufacturer may refer to 'Thermal Transmittance'. While all three of them understand that they refer to the same information, a computer system cannot search, filter, and analyse the data. Thus, the second dimension introduces how to label information.

To close this gap, buildingSMART (2020b) has developed the bsDD. The bsDD shares a library of objects and their properties for the AEC industry. It helps to identify objects and their attributes regardless of language by referring to GUIDs (buildingSMART, 2020a). Its complexity requires educated users with a high technical understanding (Jönsson, 2015). This is probably one of the reasons why most companies in the AEC industry do not use the bsDD. Still, companies define their internal standardisation.

The following sections describe how to unite an internal labelling standard with EIR by proposing the Internal Model Information Requirements (IMIR) data-model. The primary motivation of this model is to create an MVD allowing for two activities: Storing information requirements and labelling requirements, and validating IFC models. Chapter 5.2 explains the data model itself technically.

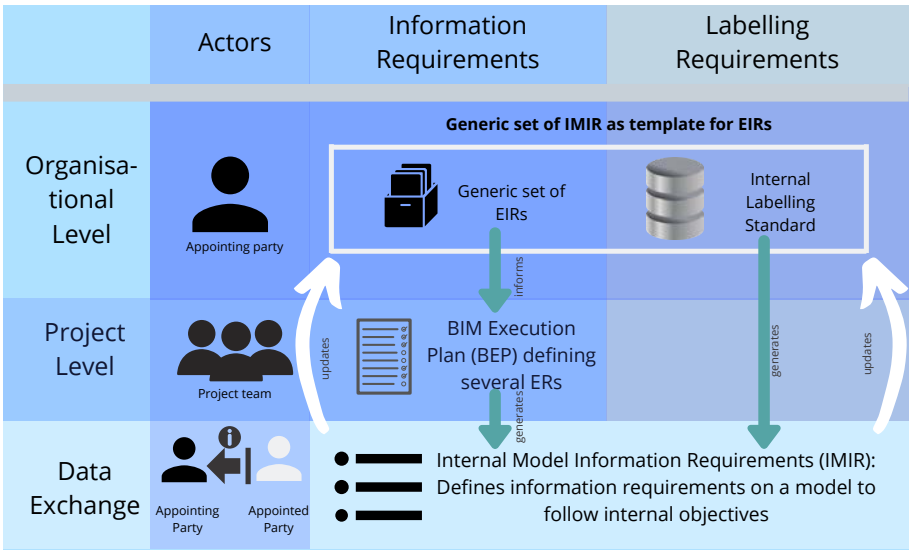


Figure 4.2: The appointing party defines information requirements on the project level and labelling requirements on an organisational level. Moreover, the IMIR functions as a source to create a generic set of IMIRs on the organisational level.

4.2.1 Overview

The BEP defines information deliverables for each party in a project. Therefore, the BEP covers the first dimension: Information Requirements. Besides this, it is highly recommended for every stakeholder to implement their internal labelling standard at least for, but not limited to, documents, component types, properties and property values (dimension 2: Labelling Requirements). Both dimensions occur on three levels of information management: the organisational level, the project level, and the level of data exchange. Figure 4.2 illustrates the role of IMIR throughout these levels.

The receiver of information (appointing party) maintains an internal labelling standard on the organisational level. This standard regulates how to name and structure properties and property values in BIM models. As the appointing party cannot demand other parties to label information explicitly, this standard only applies to internal processes. Additionally, the appointing party defines a generic set of information requirements for specific BIM Uses and exchange scenarios on the organisational level. This set evolves from earlier projects and functions as a source for the BEP on the project level.

The project level conflates task teams into the project team. Project participants agree on a BEP (compare section 2.3.2.2), and the appointing party assures that the BEP implements their Exchange Information Requirements (EIR).

Finally, both, the EIR from the project level and the internal labelling convention from the organisational level, generate the Internal Model Information Requirements (IMIR) for every

exchange scenario on the data exchange level. The IMIR extends the required information by the second dimension: how to label information. The appointing party can update their database on the organisational level from the latest experiences after each data exchange.

The internal labelling standard is, in context with section 2.3.2.1, part of the Organisational Information Requirements (OIR), and the IMIR specifies EIR. Figure 4.2 illustrates this concept highlighting that the appointing party should develop a collection of IMIRs for all their repeat exchange scenarios (ISO 19650-1, 2017).

The following example illustrates the situation: The appointed party sends an architectural BIM model to the appointing party for QTO. The appointing party has defined a generic set of IMIRs. These depend on the exchange scenario. One of them regulates the information requirements for QTO. Hence, the appointing party incorporates the information requirements (not the labelling) from their IMIR into the BEP. After the appointed party created the model, they issue it to the appointing party, who can now add the internal labelling standard to the information and verify whether the model meets the requirements or not.

For instance, the appointing party requires all components of the model to define their type. They incorporate this into the BEP. Additionally, the appointing party has their internal labelling standard, ruling that every component of the model must contain a property 'Component Type' specifying values such as 'Wall', 'Slab', or 'Column'. The appointed party does not need to use these terms. After sharing the model, the appointing party can use the IMIR to check the model.

To sum up, IMIR rules the naming of information while EIR only specifies the required information.

4.2.2 Requirements on the Structure of the EIR

The previous paragraphs described the motivation of developing a new data model to store EIR. This data model must meet certain criteria for reasonable usage. These criteria arise from the use cases the model must cover: Firstly, it shall store requirements in a way that the user can edit and maintain it easily. Secondly, EIR validate BIM models. The following sections highlight these criteria.

4.2.2.1 Hierarchical Structure for Model Validation

Current EIRs list required properties. This does not suffice for achieving BIM goals. Different attributes request different information depending on other properties. As already mentioned, a wooden beam requires different information compared to a steel beam or a concrete beam.

Thus, a hierarchical structure can represent requirements better than a list, which becomes significant when it comes to model validation.

For model validation, an algorithm needs to know what information which elements of the BIM model require. Current ERs, however, list them by component type. These component types mainly depend on other properties. Hence, the algorithm does not have a clear starting point.

For example, the EIR define the component types, wall and drywall. Each of them requires specific information. Model validation checks if components of theses types meet the requirements. However, an algorithm cannot evaluate which element has the type of wall or drywall without assigning it beforehand. Hence, finding an algorithm without using machine learning or artificial intelligence technologies is difficult.

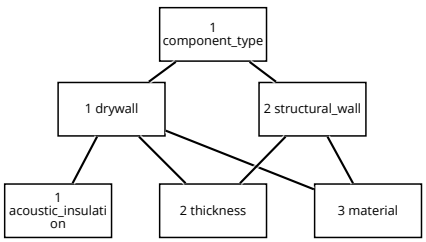
IFC tries to solve this problem by using *IfcExpressTypes*. The algorithm may use these to start validating. Applying it to the previous example, however, underlines that this does not suffice: both, wall and drywall, are types of *IfcWall* requiring different information. Hence, the first level of the EIR must also represent a property. This property determines what further information is relevant, depending on the value of the initial property.

For instance, the first level is a component type. This property allows different values such as slab, external wall, or internal wall. Each of them requires further information. The slab, for example, needs to define the thickness, material, and fire rating. Each property defines valid values. And again, each value may require more information,e.g., the material 'concrete' has a certain compressive strength.

The example shows that it is essential to structure information requirements hierarchically. The hierarchy starts with a property on the first level and defines several property values. Values can require further properties defining, again, property values requiring properties. This may continue infinitely.

4.2.2.2 Requirements on Flexible Maintenance

Same properties may characterise different objects within the same model. For instance, all materials have a density, no matter whether it is 'steel' or 'concrete'. It requires more effort to maintain a labelling standard storing the same information repeatedly. Hence, information requirements should base on references: The property itself exists once and multiple properties of higher hierarchy reference to it (compare section 4.2.2.3), which complicates the creating of unique identifiers using WBS codes.



1. component_type
    1.1. drywall
        1.1.1. acoustic_insulation
        1.1.2. thickness
        1.1.3. material
    1.2. structural_wall
        1.2.2. thickness
        1.2.3. material

Figure 4.3: The figure (left) shows the structure of the data model based on references. Deriving a tree from that creates the WBS codes (right).

Section 2.6 described the concept of WBS. Information requirements represent information deliverables. Thus, the WBS concept also applies to the requirements structure. Exchange requirements shall use this concept to create unique identifiers using WBS codes. These codes ease maintaining the information as every entity refers to the code, not to a value or a word.

The IMIR concept must enable both a reference-based data model and unique WBS codes. For this reason, every entity receives its code as an integer on a level basis. From that, the data model derives the unique codes by following entities' parents up to the root. The example in the following section explains this in detail.

4.2.2.3 Apply Concepts to an Example

To sum up, exchange requirements need to meet three criteria:

1. Structure hierarchically
2. Base on references
3. Introduce codes as unique identifiers

Figure 4.3 applies these concepts to two types of components. While drywalls require three properties - the acoustic insulation, thickness, and material - structural walls only need information about the material and thickness. Thus, thickness and material reference to both values: 'drywall' and 'structural_wall'. Each entity receives a number as its code. After generating a tree from these requirements (figure 4.3 r.), each object of the tree has a unique identifier by combining the code of each parent-node from the data model. In this example, the WBS code of thickness is '1.1.2' or '1.2.2'. Still, both of them are unique and always refer to the same entity. This only applies if the model does not comprise the same codes on the same level. Besides this, knowing the code '1.1.2' implies that the property 'thickness' belongs to the property value 'drywall' of the property 'component'.

4.2.3 Convert into MVD

In context with this work, EIR aim at (1) storing information requirements and (2) checking whether an IFC model meets the requirements or not. Both objectives were the purpose of developing the mvdXML specification for documentation and validation, respectively (Chipman *et al.*, 2016). This section describes how to unite both. Hence, the author introduces how to incorporate the structure described above into the mvdXML standard.

IMIR store information requirements. Therefore, an IMIR shall derive mvdXML files for better interoperability. These files cover the following criteria:

- Store information requirements in a hierarchy
- Store the WBS code of the entities
- Validate IFC models

Section 3.1.2 describes two options for storing information requirements in an mvdXML file for validation. The first option proposes to create one *ConceptRoot* for every required property. The *ConceptRoot* determines in the *Applicability* tag what element of the BIM model this concept applies to. In the *Concept* tags, it then adds the rules: either that a value has to be existent or that property needs to specify a particular value. This option allows for saving the WBS code of the property in the *ConceptRoot*'s 'code' attribute and the property values' WBS codes in the *Description* attribute of the *TemplateRule* tag. The second option defines one *ConceptRoot* element. The root comprises all information requirements in the *Concept* tags, and the *Description* attribute of the *TemplateRule*s stores the WBS codes.

Both variants do not support storing information requirements based on references. The first option is more complex and produces longer mvdXML files. For that reason, the IMIR model exports mvdXML files using the second option.

4.2.4 Summary

The previous sections described the core functionalities of the IMIR model. This model must enable the following activities:

- Store information requirements hierarchically
- Store information based on references
- Create WBS codes for all entities of the model
- Derive MVDs in the form of mvdXML files

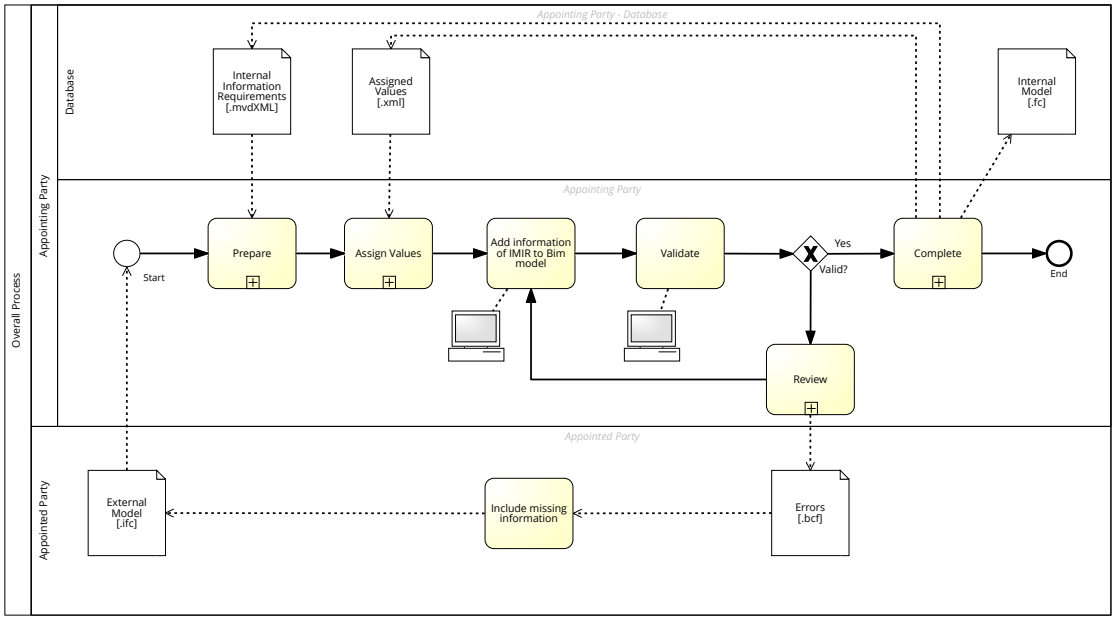


Figure 4.4: Overall process to enrich an IFC model.

4.3 Enrich-IFC-Model

After defining the internal information requirements in an MVD, the next section explains how to enrich an externally created IFC model with this MVD. This section relates to the conceptual process of enriching the model, while chapter 5.3 describes the technical implementation by proposing the Enrich-IFC-model.

Figure 4.4 shows the overall process. Two parties act: the appointed party (provides information) and the appointing party (receives the data). Any party of the project team, such as engineers, contractors, architects, or clients, can take both roles. Hence, it is not significant which role they represent. However, the process demands the following conditions:

- The project delivery team agreed on a BEP and an open-based collaboration.
- The appointing party developed their internal information requirements and derived an MVD. The MVD must define a WBS code for each property/property value.
- The appointing party does not aim at adjusting or evaluating the geometry of the model.
- The appointing party received the BIM model from the appointed party.

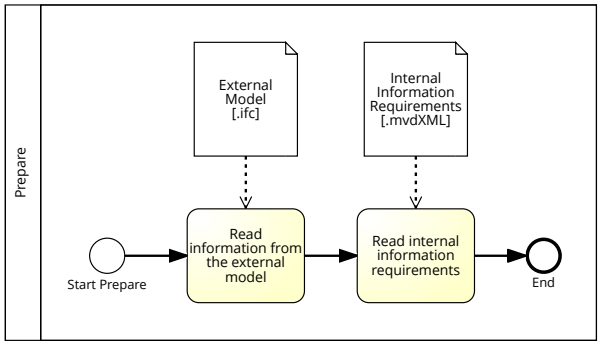


Figure 4.5: Preparing the process requires two steps: reading the information from the external BIM model and reading the information requirements from the MVD.

In the beginning, the appointing party receives a BIM model, which triggers the process. Then, the user executes the steps to prepare the data model. After that, an algorithm adds the information requirements to the BIM model and validates it. If the model is valid, the process completes by exporting the enriched IFC model. However, if the model is not valid the user must review the errors. The following sections describe these activities more precisely.

4.3.1 Prepare

Receiving a BIM model triggers the process for the appointing party. In the first task, they prepare the information. Figure 4.5 illustrates the two steps: reading the information from the external model and reading the information and labelling requirements from the MVD. Applying the process in an open-based environment suggests using the IFC standard for the BIM model and the mvdXML standards for the MVD.

Reading the information requirements creates a database containing all entities of the MVD. These can either be properties or property values. For instance, the MVD requires a slab to implement the property 'thickness'. Besides this, a slab requires defining a specific value for the material, 'concrete'. Hence, 'concrete' represents a property value. For adding these requirements to the IFC model, both, property and property value, need to define, an *IfcExpressType*, a WBS code and the property set they belong to. As this code specifies a unique identifier, there is no need to store the entries from the MVD hierarchically. A database stores all requirements in an unordered list where data can be queried from, using the WBS code. Chapter 5.3 explains this concept.

Reading the information from the BIM model, stores all property values of that model in an unstructured database. It assures that no value exists multiple times. Each entry comprises the *IfcExpressType*, the name of the property set, the name of the property, and the property value.

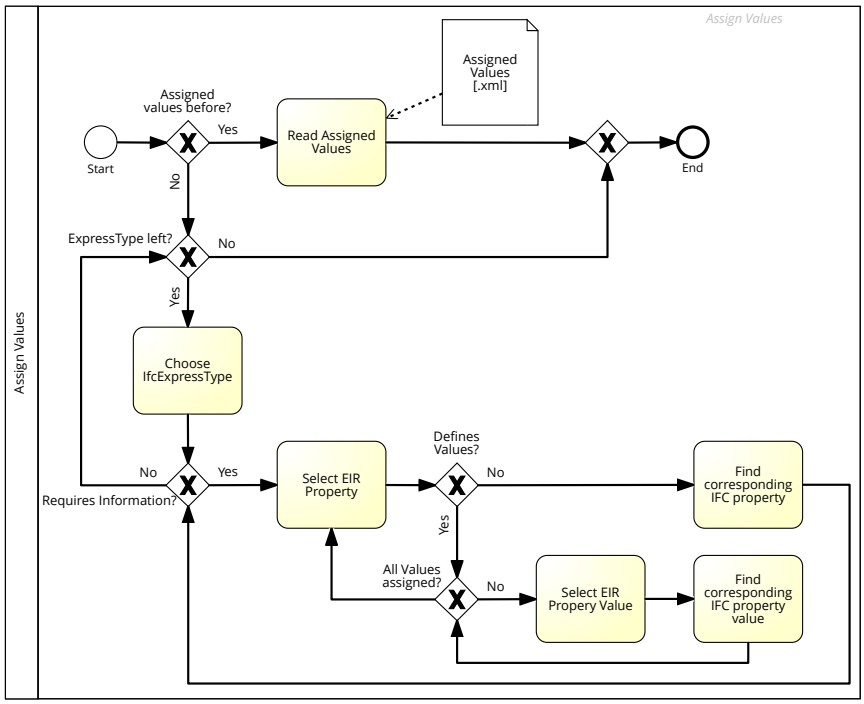


Figure 4.6: Process to assign values from the information requirements model to the IFC model.

4.3.2 Entity-Relationships

The second sub-process explains how to create relationships between entities of the Enrich-IFC-model. It can assign information requirements defined in the EIR to properties and property values from the IFC document. The users can filter and search for values of both models and assign them to each other, either properties to properties or values to values. The model then stores the assigned values as value tuples. These tuples consist of a property or property value from the BIM model and a property or property value from the MVD. Each tuple supplements information to the external IFC document according to the MVD. Each value tuple expresses that if the IFC model defines a particular property or property value, algorithm 1 shall add the assigned property or property value of the MVD to the IFC model. For example, the value tuple 'IfcValue, MvdValue' requests that every element of the BIM model defining the value 'IfcValue' shall also define 'MvdValue'.

The model can save and export the assigned value tuples as the user does not want to start from scratch for every data drop during the project. As the provider of the IFC model follows their labelling standards, they will not change the labelling of the information. This may also be useful if they work together in future projects. Exporting assigned tuples does not store the entity of the MVD itself, but only the WBS code for more flexibility. For instance, in the first data exchange, the IMIR requires to label a property 'thickness' and assigns it to

Property	Value
Component Type	Girder
Structural Material	Concrete C30/37
IfcExpressType	IfcBeam
Width	0.20m

Required Property	Required Values
Component Type	'Beam' 'Slab' 'Window'
Material	'Concrete' 'Timber'
Strength	'C20/25' 'C30/37'
Thickness	EXISTS

Table 4.2: The left table lists the properties of the IFC element and the right table represents the required information from the MVD.

the IFC property 'width'. In the next data exchange, the appointing party decided to name it 'width'. As the code has not changed, algorithm 1 still adds the information correctly.

Figure 4.6 illustrates the approach for assigning the values. If the appointing party has never worked with the appointed party, and this is the first data exchange in the project, the appointing party has to start from scratch. If one of the previous conditions is wrong, the user can import the assigned values of earlier projects or data exchanges and skip the rest of this step. In the other case, they have to follow the iterative tasks to ensure that the user does not forget about any required information. The author suggests selecting an *IfcExpressType* from the BIM model. For this type, the user now tries to assign all defined information. Therefore, they choose a property from the MVD and check whether this property requires a particular value or only requires to be existent. For the latter, they look for the property in the BIM model and assign both. Then, the algorithm adds the MVD property of the IFC model and assigns the value from the IFC model to it. If the property requires specific values, however, the user must identify the values of the BIM model and assign these.

Table 4.2 illustrates an example. An IFC element implements four properties: 'Component Type', 'Material', 'IfcExpressType' and 'Thickness'. The process starts by selecting the *IfcExpressType* 'IfcBeam'. After that, the user looks at the information requirements. They define that the element must implement the properties 'Component Type', 'Material', Strength' and 'Thickness'. Hence, the user assigns 'Beam' to 'Girder'. By that, the element obtains another property. It specifies 'Component Type' with the value 'Beam'. Besides this, the 'Structural Material' attribute of the IFC object defines two required properties: 'Material' and 'Strength'. Thus, both values can be assigned to this property. Finally, the 'Width' property of the elements corresponds to the required property 'Thickness'. Table 4.3 lists the final value tuples.

While the first three entities assign property values to property values, the last one assigns a property to a property. This is possible as the MVD does not require certain values for this attribute. Algorithm 1 can now select the required property 'Thickness' from the MVD and add the property value '0.20m' from the IFC model.

	IFC information	Required information
Value	Component Type: Girder	Component Type: Beam
Value	Structural Material: Concrete C30/37	Material: Concrete
Value	Structural Material: Concrete C30/37	Strength: C30/37
Property	Width	Thickness

Table 4.3: The model comprises four relationships: Three value to value relationships and one property to property relationship. Each relationship implies that the IFC Information matches the Required Information

4.3.3 Add Information to the Model

After assigning the values, an algorithm adds the information from the MVD to the IFC model. Algorithm 1 shows the pseudo-code. Calling the function requires the IFC model (*BimModel*) and the assigned value tuples (*AssignedTuples*) as input. Each value tuple comprises two entities: a property or property value from the IFC model (*IfcEntity*) and a property or property value from the MVD (*EirEntity*). The algorithm iterates for every tuple through every object of the IFC model (*ModelElements*) and checks if the objects define a value for the IFC property (line 7). If this is false, the algorithm continues with the next object. If it is true, the object also requires the entity from the MVD, the *EirEntity*. To add this information, the function firstly determines the type of the *EirEntity*. There are two options: property (line 8) or property value (line 10). For both options, the algorithm adds the property from the *EirEntity* to the object. However, the property value is different. If the *EirProperty* is a property value, the value of the new property is taken from the MVD. This means that the element now defines a new property with all information from the *EirEntity* (property and value). If the *EirEntity* is a property however, the element now implements a new property with the property name from the *EirEntity*, and the property value from the object itself.

Going back to the previous example in table 4.3 illustrates how algorithm 1 works. The model contains one beam with the characteristics shown in table 4.2. The *AssignedTuples* consist of four pairs: the first pair comprises the *EirEntity* 'ComponentType Beam', and the *IfcEntity* 'ComponentType: Girder'. As the current element, the beam, defines the *IfcEntity* (line 7), the algorithm checks whether the *EirEntity* is a property or property value. As it is a value, the object obtains a new property 'Component Type' and adds the value 'Beam' from the *EirEntity* (line 11). The same procedure applies to the second and third tuple. The fourth, however, is a property. For that reason, the value is not retrieved from the *EirEntity* but from the object itself (line 9). Thus, the element obtains a new property 'Thickness' with the value '0.20m'. Table 4.4 shows the properties of the element after adding the required information using algorithm 1.

Algorithm 1 Adds the assigned values from the exchange requirements to the IFC model for each value tuple.

```
1: function AddValue(BimModel,AssignedTuples)
2:     for i = 0 : AssignedTuples.Count − 1 do
3:         ModelElements ← elements from BimModel
4:         IfcEntity ← IfcEntity from AssignedTuple[i]
5:         EirEntity ← EirEntity from AssignedTuple[i]
6:         for k = 0 : ModelElements.Count − 1 do
7:             if ModelElements[k] defines IfcEntity then
8:                 if EirEntity is type of Property then
9:                     value ← PropertyValue of IfcEntity from ModelElements[k]
10:                else if EirEntity is type of PropertyValue then
11:                    value ← Value from EirEntity
12:                end if
13:                Add EirEntity to ModelElements[k] and set Value to value
14:            end if
15:            k++
16:        end for
17:        i++
18:    end for
19: end function
```

Property	Value
Component Type	Girder
Structural Material	Concrete C30/37
IfcExpressType	IfcBeam
Width	0.20m
Component Type	Beam
Material	Concrete
Strength	C30/37
Thickness	0.20m

Table 4.4: Final properties of the beam after adding the required information according to existing information.

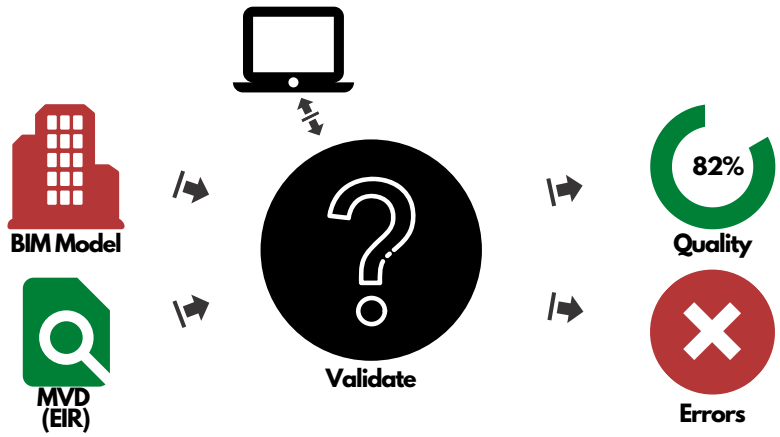


Figure 4.7: Model validation requires the BIM model and MVD as an input. Validating the model using an algorithm returns the quality of the model and the failed information requirements.

4.3.4 Validate the Information

Starting the next sub-process, the user has already assigned values from the MVD to the IFC model. However, the user may have missed some properties, or the IFC model does not contain all the required information. To identify missing information, this step validates the model and returns errors for each failed requirement. Hence, it aims at:

- Providing an overview of the IFC's quality: how many conditions are passed/ failed?
- Listing what components fail what requirements.
- Communicating failed requirements with the responsible appointed party.

Figure 4.7 visualises the input and output of this process step. Using the BIM model and the MVD, the validation shall return the model's quality and the failed requirements once the activity has finished. A computer system in the form of an algorithm executes the validation itself.

Algorithm 2 shows the pseudo-code validating the model. This code comprises two methods. The primary function requires the BIM model and the EIR as input and returns the quality of the model. The quality expresses a ratio of failed conditions in comparison to the total requirements and lists all failures. From this list, the data model can derive BCF files for issue communication. The main function iterates through all elements of the BIM model. For each component, it calls the second method (line 5). This method requires the following four input sources:

Algorithm 2 The method's input are information requirements in a tree form. From the root, the function recursively checks for each element of the IFC model, whether conditions pass or fail. For fulfilled concepts, the algorithm calls itself. It returns a Quality comprising a passed/failed ratio and an error entity for each failure.

```
1: function VALIDATE(EirTree, BimModel)
2:     AllElements ← elements from BimModel
3:     Quality                                  ▷ represents the quality of the BimModel
4:     for i = 0 : AllElements.Count − 1 do
5:         RECURSIVEVALIDATE(AllElements[i], root, Quality)
6:                                  ▷ callls the validate recursively until a leaf not is reached
7:         i + +
8:     end forreturn Quality
9: end function
10: function RECURSIVEVALIDATE(Element, Root, Quality)
11:     Errors ← Errors from Quality
12:                                  ▷ Set of all errors. Each representing a failed requirement
13:     Properties ← all properties from Root
14:     for k = 0 : Properties.Count − 1 do
15:         Quality.RequiredValues + +
16:         if Element defines Properties[k] then
17:                 ▷ Element defines the property. Does the property require a specific value?
18:             if Properties[k] defines PropertyValues then
19:                                  ▷ Specific value required          ▷ test if value os correct
20:                 for n = 0 : properties[k].PropertyValues.Count do
21:                     if Element defines properties[k].PropertyValues[n] then
22:                                                                        ▷ Value correct
23:                         Quality.CorrectValues + +
24:                         value ← properties[k] ← PropertyValue[n]
25:                         RECURSIVEVALIDATE(element, value, Quality)
26:                                  ▷ Check if property value requires further properties
27:                     else Errors.Add()
28:                     end if
29:                     n + +
30:                 end for
31:             else
32:                 Quality.CorrectValues. + +
33:             end if
34:         else Errors.Add()
35:         end if
36:         k + +
37:     end for
38: end function
```

Element: A single element of the BIM model.

Root: An exchange requirement from the MVD. The MVD structures hierarchically. The *Root* also contains all child requirements, including their child requirements.

Quality: The quality of the model: the *Quality* changes every time the recursive validate method is called. To comply with the before mentioned objectives of this process, the *Quality* comprises (1) a number of passed requirements (*CorrectValues*), (2) the number of total requirements (*RequiredValues*) and (3) a list of failed requirements with information about the related building element of the BIM model (*Errors*)

Calling this method retrieves all properties the *Root* requires (line13). The current *Element* must define all these properties. Hence, the algorithm iterates through those and increments the *RequiredValues* (line 15). Then, the code validates whether the *Element* defines the property or not. If not, the concept has failed, and the *Errors* extend by one (line 34). If the *Element* defines the property, the code must check whether the MVD requires the property to be existent or to defines a specific value (line 18). If the value only needs to exist, the *CorrectValues* increment (line 32), and the algorithm continues with the next property (line 14). If the MVD requires a specific value, however, the algorithm queries the value options from the MVD (line 20), and checks if the value of the BIM model matches one of them (line 21). If not, an error is added. If the test passes, the *CorrectValues* increment. Finally, the code has to check whether this specific property value requires further information in the MVD (if it comprises child nodes). If this is the case, the method calls itself recursively; this time, with the same *Element* and *Quality* but with the current property value as *Root*.

At the end of the validation, the *Quality* can express the ratio of correct values to the required information. However, this is not always very precise. For instance, an EIR tree with many levels cannot estimate the failed conditions in lower levels if a requirement in a higher level has failed. For example, a wall requires to have a material. If the material has the value concrete, it needs to implement a strength. The IFC model comprises one wall with no information about the material. Calling the method in algorithm 1, returns that zero values are correct, and one value is required. However, the model actually requires two values. This example shows that these methods approximate the quality. It is not possible to make it more precise, as the computer system cannot evaluate how the lower levels affect the overall quality.

The last objective of this step is to communicate issues with the appointed party. The next section describes how to achieve this.

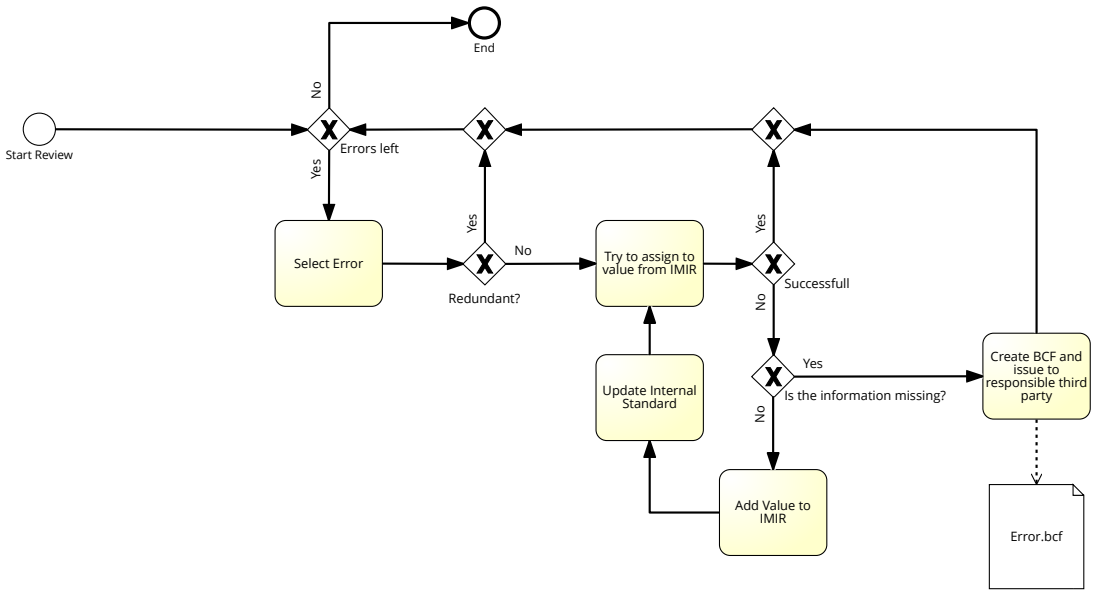


Figure 4.8: Process to handle errors in the Enrich-IFC-approach. For external failures, the user can communicate via BCF.

4.3.5 Review Errors

Validating the model returns a list of failed requirements. These failures can arise from three reasons:

1. The user has missed out provided information on the IFC model.
2. The internal information requirements do not cover the provided information.
3. The appointed party has not provided the information they agreed on.

The users should now go through every error message and solve them in this order. In the beginning, the users shall check if they have missed out information in the IFC model. It is crucial to start with this, as the other errors require more attention to solve. Figuring out that the users themselves have failed, may frustrate them even more if they have spent time on other sources. Hence, they should first go through the properties again and identify if the model stores the information. For example, the MVD requires a component to define the property 'Component Type'. The external model names this information 'ct'. As it varies, the user may have missed it.

Secondly, missing information in the MVD can trigger the error. The IFC model implements the material 'timber'. As the appointing party has only worked in concrete projects, the internal information requirements may not cover this possibility. Hence, the second option to solve an error represents editing the MVD. In this case, it means adding 'timber' to the

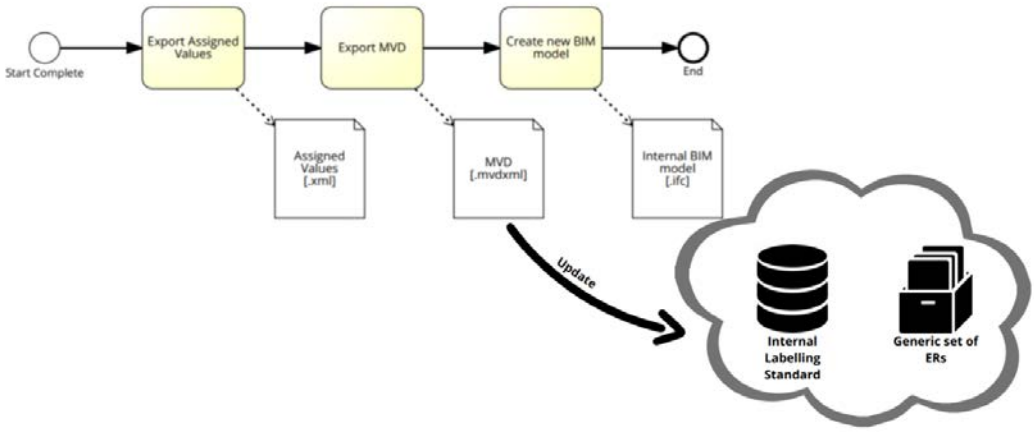


Figure 4.9: Finalising the Enrich-IFC-approach stores progress and updates related data.

MVD. Updating the MVD may represent the most sophisticated issue as certain values may comprise further requirements - 'timber' requires information such as the tensile strength or visual quality. This error may mainly occur during the early stages after implementing this approach.

Lastly, after the appointing party has eliminated error sources (1) and (2), they can communicate issues with the appointed party. To do this, the Enrich-IFC data model allows for BCF export. These files contain the information that the recipient needs to understand what information the model misses: the component that lacks information, the missing attribute, and the responsible party. The Enrich-IFC-model shall communicate with a BIM collaboration platform, automatically upload the BCF files and notify the responsible party.

If there are no errors left, the user can progress to the next step. Having errors triggered by source (3) refuses that. Then, the appointing party has to wait until the appointed party has incorporated the missing information to restart with the whole process.

4.3.6 Finalise

After reviewing the errors, the model ideally exhibits a quality of 100%. High quality implies that the BIM model implements the required information for executing the desired downstream processes, e.g., Cost Estimation or Structural Analysis. This part of the approach exports the adapted IFC model that is the source of truth for internal uses.

Additionally, for later data exchanges, the user wants to save all operations. Hence, the user exports the assigned value tuples from section 4.3.2 in a database, and the adjusted internal requirements in an MVD. The MVD automatically uploads changes to a server storing the IMIRs. The server updates the IMIRs according to potential changes. Figure 4.9 illustrates this concept.

IMIR-model	Enrich-IFC-model
1. Define property sets, properties and values in a hierarchical structure 2. Assign properties to multiple property sets 3. Derive WBS codes for properties and property values 4. Create MVD in the form of mvdXML for model validation	1. Read information requirements from IMIR or mvdXML files and derive properties and values including their WBS codes 2. Read IFC documents and store property sets, properties, and property values 3. Assign instances of the IFC model to instances of the information requirements 4. Save value tuples by referring to the WBS codes 5. Create enriched IFC model 6. Validate model and highlight missing information 7. Communicate missing information with the project team via BCF

Table 4.5: Requirements for the data models to convert the process technically.

4.4 Summary

Currently, information requirements in the form of EIR define in the BEP what information a party must provide. They neither cover hierarchical requirements nor labelling requirements. The previous chapter described how to close this gap. The author proposes the IMIR data model for defining dynamic information requirements in a hierarchical structure and the Enrich-IFC data model to supplement IFC models with labelling requirements. To finally validate whether the enriched IFC document complies with the IMIR, the IMIR data model derives MVD as mvdXML files.

Table 4.5 summarises the findings from the previous chapter and defines requirements on both data models. Chapter 5.2 explains the technical implementation of the IMIR data model and chapter 5.3 of the Enrich-IFC data model, respectively.

Chapter 5

Data Model Design

The previous chapter has introduced the novel Enrich-IFC-approach. It concluded with a list of requirements that a data model shall convert. For that reason, the author has established two data models. Firstly, the IMIR-model for storing and maintaining information requirements hierarchically and, secondly, the Enrich-IFC-model to translate given data of an IFC model into the required information of an MVD (or IMIR). The following chapter introduces both models after overviewing their role within the Enrich-IFC-approach.

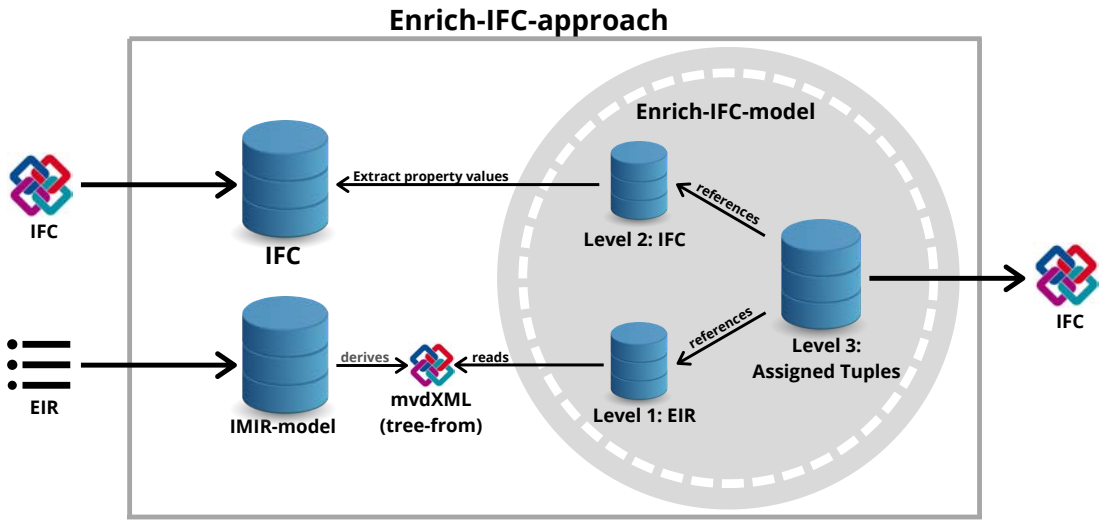


Figure 5.1: The roles of the two data models within the Enrich-IFC-approach.

5.1 Overview

The Enrich-IFC-approach comprises two data models (figure 5.1) that both are implemented using C#. The first one, the IMIR-model, provides structures and methods to create and maintain information requirements. These requirements represent an MVD that the model can convert into an mvdXML file. The derived file contains a hierarchically structured tree and provides unique identifiers in the form of WBS codes. The second part, the Enrich-IFC-model imports the hierarchically assembled information requirements into an unstructured database. Moreover, it allows for the import of IFC models and creates an unordered database from all its property values. The Enrich-IFC-model generates value tuples. Each tuple references two objects; one from the IFC data and one from the requirements data. Using these tuples, the Enrich-IFC-model translates the information from the IFC into the data from the MVD and derives an enriched IFC file.

5.2 Internal Model Information Requirements

The Internal Model Information Requirements (IMIR) unite information requirements from the EIR with the labelling requirements from internal standard. Hence, it specifies the exchange requirements of an MVD. IMIR aim at creating and maintaining model requirements according to the BIM goals of a project. Thus, it describes a subset of the IFC schema that a BIM model must define. As this represents the exact purpose of MVDs, the IMIR-model proposes to define exchange requirements and export them into mvdXML afterwards. This approach extends the concept of EIR by also ruling the hierarchy and labelling of information.

5.2.1 Requirements on the Data Model

The previous chapter identified the requirements for the IMIR data model and summarises them in table 4.5. These requirements introduce the following features, that the proposed model shall implement.

- Create, edit and maintain requirements triggered by certain property values.
- Create, edit and assign property sets.
- Create properties, add values to properties, and assign properties to property sets.
- Save and export, read and import the data model.
- Derive mvdXML files.

The IMIR model design means to store information requirements. Therefore, it defines the required properties and potential property values hierarchically. Each property value may,

in turn, trigger another property. However, properties may not request specific values. The IMIR model also allows requiring certain properties. Properties and property values are labelling sensitive. To not mix up two instances, each entity obtains a GUID.

5.2.2 IMIR Design

The design adheres to the object-oriented principles to assure high extendibility and flexibility. It allows structuring property requirements in any hierarchy. This offers to deal flexibly with different domains, construction types, and software tools. Moreover, the model provides a way to query information. Figure 5.2 illustrates the four classes that the data model introduces: *PropertyValue, Property, PropertySet, and Requirements*. All four classes have a GUID comprising 36 digits. Hence, the possibility of creating two of the same is low.

PropertyValue is an abstract class providing basic properties and methods for the generic *PropertyValue*$< T >$ class. This class allows creating instances with four different data types for their *Value* (bool, int, double, string). The data type is defined in the *Property* class. A *Property* may define several *PropertyValues*, and references one or multiple *PropertySets*. Through that, the data-model may assign properties to several property sets. Thus, it reduces the workload of potential changes in single properties. For example, multiple component types implement a 'Material' *Property*. Now, the user wants to change the name to 'Structural Material'. By referencing multiple *PropertySets*, the user only needs to change the name once. The *PropertySet* class defines multiple instances of type *Property*, and references one *Requirements*. This is a one to one relationship. Every instance of *Requirements* may only refer to one instance of *PropertySet*. An instance of type *PropertyValue* may trigger *Requirements*, which imply that a component implementing this property value must also define the requirements. These requirements always reference one *PropertySet*.

The data model unites all instances in dictionaries. Between several types of collections, the author decided to implement the data-model using dictionaries as they provide quick querying by a key (Microsoft, 2015). The key in the dictionaries is either the GUID or the *Requirements* entity. Besides querying by the key, the data model also provides methods to query and filter instances by names and values.

5.2.3 Store and Export Data Model

The data-model implements two interfaces to create and read file formats. The static classes, *XmlReader* and *XmlCreator*, allow for reading and creating XML files respectively. Such an XML file stores the model, including the referenced structure. Therefore, it saves all model instances and their GUIDs. The model needs these identifiers to determine dependencies.

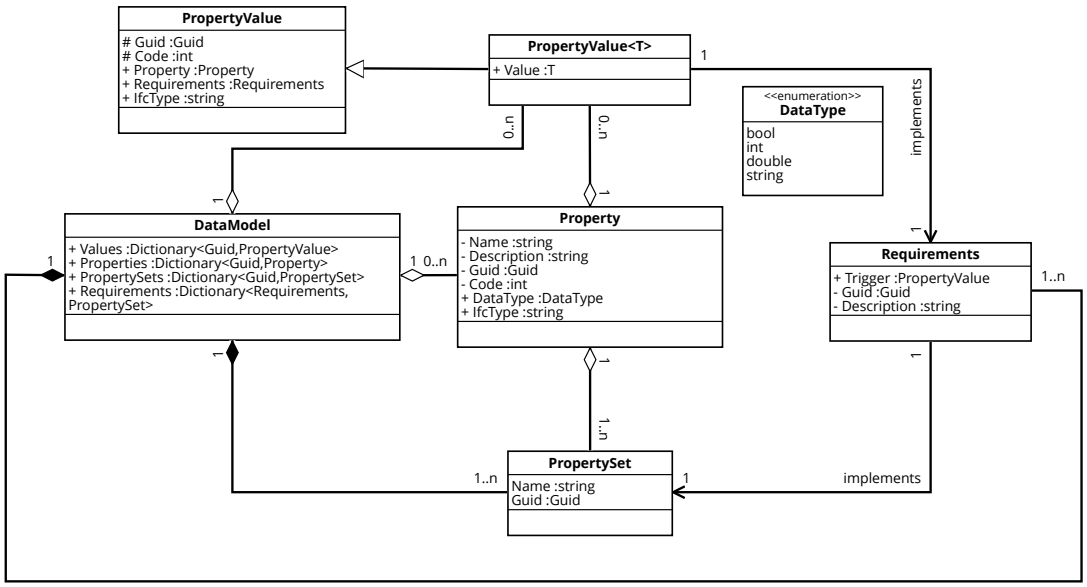


Figure 5.2: Simplified UML diagram of the IMIR-model (appendix B.1 shows the complete diagram).

For example, a *Property* may reference multiple *PropertySet*s. Now, the *XmlReader* can determine where the XML file stores this *PropertySet* by searching for the GUID.

Listing 5.1 illustrates how the references work. The *PropertySet* starting at line 11, defines its attribute *Guid*. The *Property* starting at line 19, references this *PropertySet* by defining the *PropertySet*'s GUID in its *ref* attribute (line21). Without the reference-based structure, the *XmlReader* could not identify the dependency between both entities.

Secondly, the static class *MvdXmlCreator* supports exporting files in accordance with the mvdXML standard. mvdXML files exported by the model follow the structure of 'strict hierarchy' (compare section 3.1). Therefore, an algorithm converts the IMIR model into a tree that structures strictly hierarchically. For that reason, it does not base on references. Hence, it is not possible to recreate the initial IMIR model from this tree. Deriving this tree, however, is crucial for generating WBS codes. The *MvdXmlCreator* generates an mvdXML file that defines all requirements according to this tree. The derived mvdXML document defines unique identifiers as WBS codes arising from the *Code* property of the *Property* and *PropertyValue* classes. An algorithm follows the tree from an entity up to the root and strings the *Code* integers together. The resulting string represents the WBS code. The following example illustrates that concept.

```
<IMIR>
    <AllRequirements>
        <Requirements Trigger="" Description="All building components must
            implement a componentType Property" Guid="6e2a88bc-31d4-4a52-
            ba46-c90efdb48d49" /><!--further Requirement(s)-->
    </AllRequirements>
    <AllValues>
        <PropertyValue IfcType="IfcProduct" Value="Wall" Guid="bcb31bd6-
            c0e7-4e3a-8c71-92b8731e5478" Code="1" Property="0814332b-a6b1-49
            e4-a86d-e9b8cf3f1bbd" Requirements="ad62b7dd-9fd4-4b1c-b438-80
            dc75b862a1" /><!--further PropertyValues-->
    </AllValues>
    <AllPropertySets>
        <PropertySet Guid="3cdba1e0-9c48-44c0-b4f1-89a1a6be17eb" Name="
            Pset1" Requirements="6e2a88bc-31d4-4a52-ba46-c90efdb48d49">
            <Properties>
                <Property ref="0814332b-a6b1-49e4-a86d-e9b8cf3f1bbd" />
            </Properties>
        </PropertySet><!--further PropertySets-->
    </AllPropertySets>
    <AllProperties>
        <Property IfcType="IfcProduct" Guid="0814332b-a6b1-49e4-a86d-
            e9b8cf3f1bbd" Name="Component Type" Code="1" Description="The
            Component Type of the Property" DataType="String">
            <PropertySets>
                <PropertySet ref="3cdba1e0-9c48-44c0-b4f1-89a1a6be17eb" />
            </PropertySets>
            <PropertyValues>
                <PropertyValue ref="bcb31bd6-c0e7-4e3a-8c71-92b8731e5478"/>
            </PropertyValues>
        </Property><!--further Properties-->
    </AllProperties>
</IMIR>
```

Listing 5.1: The IMIR export lists all instances of the four major classes of the model: *Requirements, PropertySet, Property, PropertyValue*. Green comments replace parts of the file (full version: annex D).

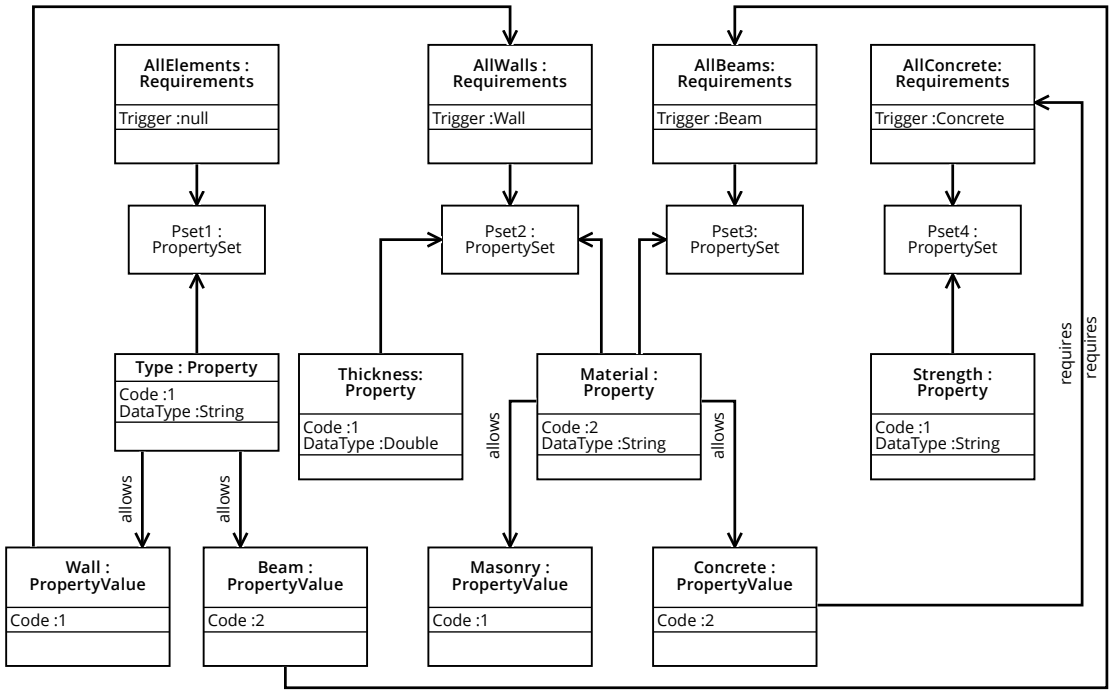


Figure 5.3: Instance of the data model comprising four Requirements, four PropertySets, four Properties, and four PropertyValues.

5.2.4 Instance of the IMIR-Data-Model

Figure 5.3 explains the IMIR-design by applying it to a scenario. All elements must define the *Properties* referencing the *PropertySet* 'Pset1'. This set only references one attribute, 'Type'. The model allows the *PropertyValue* to either be 'Wall' or 'Beam'. Both values trigger the new *Requirements* 'AllWalls' and 'AllBeams', which implement the *PropertySet*s 'Pset2' and 'Pset3' respectively. Both reference the property 'Material'. Assigning the same *Property* to different *PropertySet*s characterises the IMIR model and improves flexibility and maintainability. 'Material' comprises two *PropertyValues*, 'Concrete' and 'Masonry'. The latter requires further data and introduces the third level of the requirements tree.

Listing 5.1 shows how the model exports the requirement into an XML file. This file divides into four XML elements, one for each main class of the IMIR data model. The first tag stores all *Requirements*, the second all *PropertyValue*s, the third tag represents all *PropertySet*s, and the last one all entities of type *Property*. The *XmlCreator* also exports all GUIDs. What may look unstructured is indispensable as the *XmlReader* needs this information to assemble the correct structure of the model.

Lastly, the model creates the mvdXML file. Therefore, the model generates a tree structuring all elements hierarchically, including their WBS codes. From that tree, the algorithm can derive an mvdXML file that incorporates requirements in the form of *TemplateRule*s. These

```
<TemplateRules operator="and">
    <TemplateRule Description="1.2.2.1."
        Parameters="O_PsetName[Value]='Pset3' AND
        O_PName[Value]='Material' AND
        O_PSingleValue[Value]='Concrete'" />
    <TemplateRules operator="and">
        <TemplateRules operator="and">
            <TemplateRule Description="1.2.2.1.1."
                Parameters="O_PsetName[Value]='
                Pset4' AND O_PName[Value]='
                Strength' AND O_PSingleValue[
                Exists]=TRUE" />
        </TemplateRules>
    </TemplateRules>
</TemplateRules>
```

1. Component Type
   1.1. Wall
      1.1.1. Thickness
      1.1.2. Material
         1.1.2.1. Concrete
            1.1.2.1.1 Strength
         1.1.2.2. Masonry
   1.2. Beam
      1.2.2 Material
         1.1.2.1. Concrete
            1.1.2.1.1 Strength
         1.1.2.2. Masonry

Figure 5.4: The IMIR-model derives a tree (left) to creates an mvdXML file. The excerpt shows two *TemplateRules* exemplarily. The *Description* attribute defines the WBS code.

rules comprise two attributes. Firstly, the *Description* to store the WBS code, and secondly the *Parameter* that stores the information about the *Property*, *PropertySet*, and optionally the *PropertyValue*. Figure 5.4 shows the tree and an excerpt from the mvdXML file. Moreover, it illustrates that creating an IMIR model from this tree is not possible as the same entities have different WBS codes.

5.3 Enrich-IFC-Model

The second model of the Enrich-IFC-approach means to enrich the IFC document by translating given information of the external IFC model into the Exchange Information Requirements (EIR). Therefore, it assigns properties or property values of the external model to entities of the internal requirements. This enables the model to derive a new IFC document, including the internal information requirements.

The model introduces three levels: (1) the EIR-level defines the semantic information requirements on an internal model to use it for automated processes. The model saves these requirements in an unordered database and provides methods to query the data. (2) The IFC-level represents a list of the property values that exist in the IFC document. (3) The assigned-data-level defines value tuples combining level (1) and (2). Each tuple references both, one instance of the EIR-level, and one entity of the IFC-level. The tuples function as the basis for the model to export the enriched BIM model. Moreover, this level validates the quality of a BIM model regarding the internally defined EIR from level (1). In case of insufficient model quality, the data model derives BCF comments for issue communication.

5.3.1 Enrich-IFC-Model Design

Figure 5.5 shows the UML diagram of the design. The Data model comprises the three levels described above. For level (1) and level (2), the abstract *Property* represents the origin. It implements the properties *IfcType*, *PSetName* and *PropertyName*, as well as the interfaces *IXml* and *IIfcFinder*. These interfaces provide methods to read and create XML elements and find IFC instances respectively. *IfcType* defines the *IfcExpressType* of the IFC schema, e.g. *IfcBeam*, *IfcWall*, *IfcProject*, the object relates to. Both classes, *EirProperty* and *IfcProperty*, inherit from *Property*.

The data model stores the information requirements in the form of *EirProperty* and *EirPropertyValue* objects in a collection. *EirProperty* extends *Property* by a *Code* representing a unique combination of numbers separated by dots. It follows the concept of WBS codes. To guarantee that the model does not contain multiple codes, the collection in the data model is represented by a dictionary with the *Code* as the key. A Dictionary must not contain multiple entities of the same key and provides querying the data (Microsoft, 2015). Thus, this collection guarantees that there are not various objects defining the same *Code*. Moreover, *EirProperty* realises the *IError* interface to derive an error message. The *EirPropertyValue* class extends *EirProperty* by the *Value*. As it inherits from *EirProperty*, the previously mentioned dictionary can contain both, *EirProperty* and *EirPropertyValue* objects.

Secondly, the data model implements a collection of *IfcPropertyValue* objects. These objects inherit from *IfcProperty* that also derives from the abstract class *Property*. To assign a value to elements defining this *IfcProperty*, the class realises the *IAssign* interface. Objects of *IfcPropertyValue* extend *IfcProperty* by a *Value*.

An instance of *AssignedValues* represents the third level of the Enrich-IFC-model. This class stores value tuples comprising either an *IfcProperty* and an *EirProperty* or an *IfcPropertyValue* and an *EirPropertyValue*. Thus, this class assigns entities of the IFC model to entities of the EIR. The method *Run(IfcStore model)*, adds the values from the EIR to the IFC model by calling the *Add*-method from the *IAssign* interface. Algorithm 1 shows the pseudo-code of this method. Besides this, the *AssignedValues* class also implements a method to store itself in an XML file. Listing 5.3 shows such a file. Section 5.3.2 describes the structure.

Additionally, the data model implements an object of *Quality* representing the required and correct conditions of a model depending on information requirements. It consists of a dictionary comprising *LevelQuality*s with the level as their key. Each *LevelQuality* specifies the passed and failed conditions on a specific level. In this context, the level represents the difference between an entity in the EIR-tree and the root. To estimate the quality of the IFC model, the *Validate(XmlElement tree)* method recursively checks the information requirements (compare algorithm 2). For each failure, the *Quality* obtains a *BcfError* representing the failed condition. This class allows for deriving BCF files for issue communication. For

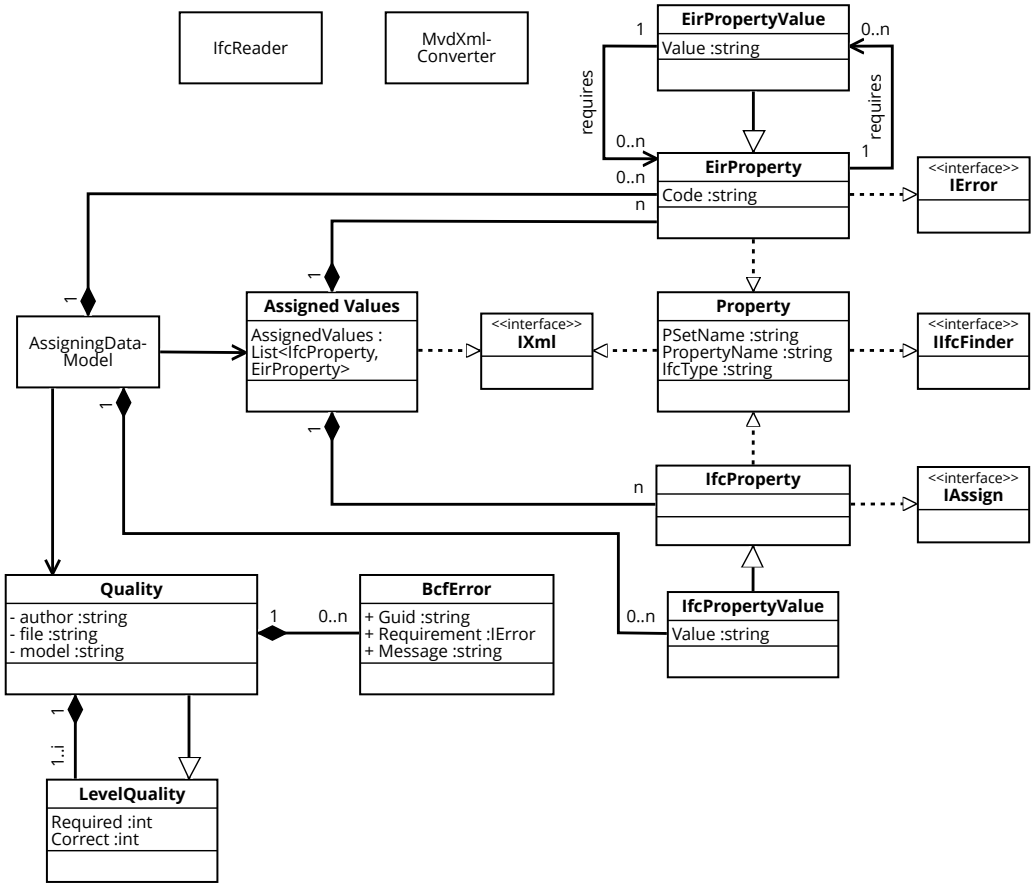


Figure 5.5: Reduced UML diagram of the Enrich-IFC-model design (full version: appendix B.3).

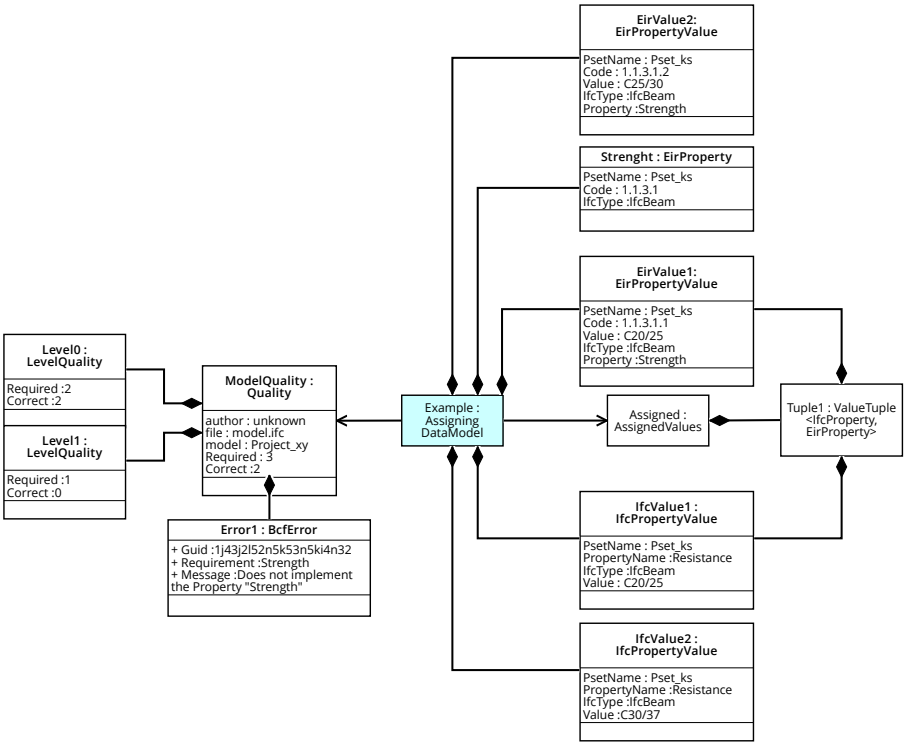


Figure 5.6: Instance of the Enrich-IFC- model.

this reason, the class stores the GUID of the IFC entity that does not define the required information. Moreover, it defines an object of *IError* to derive the information on what condition has failed.

Lastly, the model comprises three static classes: *MvdXmlConverter*, *EirReader*, and *IfcReader*. All three classes provide methods to read data from different file formats. The *MvdXmlConverter* can interpret mvdXML files, the *EirReader* reads XML files that follow the structure of an IMIR and the *IfcReader* extracts the information from an IFC model.

5.3.2 Instance of a Data Model

Figure 5.6 shows an example of such a data model. It contains two instances of *IfcPropertyValue*, 'IfcValue1' and 'IfcValue2'. Moreover, it defines two instances of *EirPropertyValue*, 'EirValue2' and 'EirValue2', as well as the *EirProperty* 'Strength'. In 'Assigned', 'EirValue1' is assigned to 'IfcValue1'. This tuple means that the algorithm adds the 'EirValue1' to every instance of the IFC model that defines 'IfcValue1'. The 'ModelQuality' validates the IFC model and implements the *BcfError* 'Error1'. This instance can derive files of the BCF format for issue collaboration in BIM projects. Besides this, the 'ModelQuality' comprises two objects of type *LevelQuality* representing the quality of the model in corresponding levels of the EIR.

```
1 <!-- Excerpt from the Markup file-->
2 <Topic Guid="617b091c-4788-4447-b7b8-8bd4dc202b2b">
3     <Title>Issue regarding 1j43j2l52n5k53n5ki4n32</Title>
4 </Topic>
5 <Comment Guid="f695d686-467c-43a5-bccc-fc5b3fa5c65c">
6     <Status>Error</Status>
7     <Date>2020-03-11T17:59:17.4535225+01:00</Date>
8     <Author>ArchitectureS</Author>
9     <Comment>102  Does not implement the Property: Strength</Comment>
10    <Topic Guid="617b091c-47c4-aaa7-b7b8-8me4dc202b2b" />
11    <ModifiedDate>2020-03-11T17:59:17.4535225+01:00</ModifiedDate>
12 </Comment>
13 <!-- Excerpt from the VisualizationInfo-->
14 <Components>
15    <Component IfcGuid="1j43j2l52n5k53n5ki4n32" Selected="false" Visible="
          false" />
16 </Components>
```

Listing 5.2: BCF export from the Enrich-IFC-model.

```
1 <AssignedPair>
2     <IfcItem IfcType="IfcBeam" PsetName="Pset_ks" PropertyName="Resistance"
          PropertyValue="C20/25" />
3     <EirItem ExpressType="IfcBeam" PsetName="Pset_ks" PropertyName="
          Strength" PropertyValue="C20/25" Code="1.1.3.1.1." />
4 </AssignedPair>
```

Listing 5.3: Assigned pair in XML generated from the Enrich-IFC-model.

Listing 5.2 illustrates how the model exports BCF issues. The Listing shows excerpts of both files comprising BCF, the markup and the visualisation info. The *BcfError* 'Error1' generates this file and adds the required information. The title of the topic (line 3) refers to the GUID of the related component that is also part of the visualisation info (line 15). Besides this, the model adds a comment that indicates what information is missing (line 9).

The model saves the assigned tuples in an XML document shown in listing 5.3. Each pair comprises an *IfcItem* and an *EirItem*. Those can either represent a property or a value. Listing 5.3 implements values. Both items convert all their data in XML attributes. The author decided to export all properties as different purposes may require more information in the future. For this application, however, exporting the *Code* would suffice as for reading the information, the model uses the *Code* to find their values. Hence, if the attribute has changed, the Enrich-IFC-model updates the assigned pairs. For instance, after exporting the file, the user decides to name the property 'Compressive Strength'. Whilst importing it again, the assigned value uses the updated information as it only refers to the *Code*. This fact also implies that the *Code* must not change to ensure consistency.

5.3.3 Agile Principles

Figure 5.5 illustrates that the design adheres to several agile, object-oriented principles to assure high extendibility and flexibility. For instance, it implements several interfaces (*IXml, IError, IAssign, IIfcFinder*) with single responsibilities to comply with the interface segregation principle defined by Martin (1996). For decoupling modules, the *BcfError* class references the *IError* interface instead of the *EirProperty* class. This inverts the dependency (Martin, 1995). This concept enables extending the design by other information requirements in the future. For instance, it may implement geometric requirements. To conform to the substitution principle by Liskov & Wing (1994), the *EirPropertyValue* and *IfcPropertyValue* classes inherit from their subtypes in a way that objects of *EirPropertyValue* and *IfcPropertyValue* can replace objects of their subtypes without altering the correctness of the design.

Additionally, the model follows the principle of single responsibilities and relies on the functionalities of the *XBIM* package developed by Lockley *et al.* (2017) to read and edit IFC documents (Martin, 2002). The author followed agile approaches for software development. For example, the author identified user stories for each feature and developed the models test-driven using unit and integration tests (Martin, 2008). The overall test coverage amounts to 87%.

Chapter 6

Prototype

To evaluate the proposed approach for practical use, the author implements a software tool, including a Graphical User Interface (GUI). This chapter describes the prototype by firstly listing all ideal features and, secondly, explaining the implemented features and core functionalities. Lastly, it leads the user through the prototype by applying it to an example.

6.1 Ideal System

The Ideal system covers the following activities:

1. Define and edit information requirements in the IMIR data model and derive mvdXML or XML documents
2. Read and export MVDs in the form of mvdXML or IMIR files
3. Read IFC documents and store the semantic information
4. Assign values of the IFC model to entities of the required information
5. Validate the model
6. Handle Errors
    - Export BCF files and automatically upload them to a BIM collaboration platform
    - Assign values depending on the error message
    - Update the information requirements
7. Read and export the assigned values as XML files
8. Create a new IFC file containing the added information

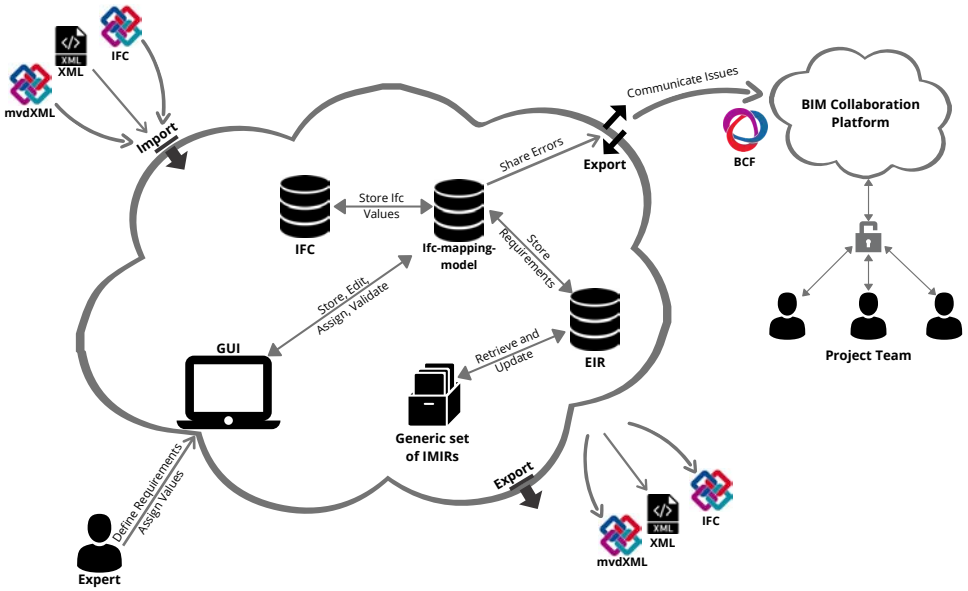


Figure 6.1: Overview of the ideal system design.

Figure 6.1 illustrates the system of the ideal solution. The expert accesses the data using the GUI and executes all activities described above. The system can read and export XML, mvdXML, and IFC files. Moreover, it automatically uploads issues to a BIM Collaboration platform using BCF files. From this platform, the project team can retrieve the data.

The central concept is that the users can assign properties and property values from the required information to specific properties or property values of the external IFC document. After that, the user can export an IFC file implementing all the required information. The following section leads through the features of the prototype.

6.2 Implemented Features

The implemented prototype does not cover all the features of the ideal system. Currently, it is not possible to create information requirements via the GUI. However, the IMIR data model provides a library to create and maintain these and derive mvdXML documents. Appendix B.2 describes the core functionalities by creating the data model described in section 5.2.4. In this context, the prototype starts after the user has created the information requirements. Figure 6.2 shows the main window of the GUI. The view comprises nine components, which the following sections overview.

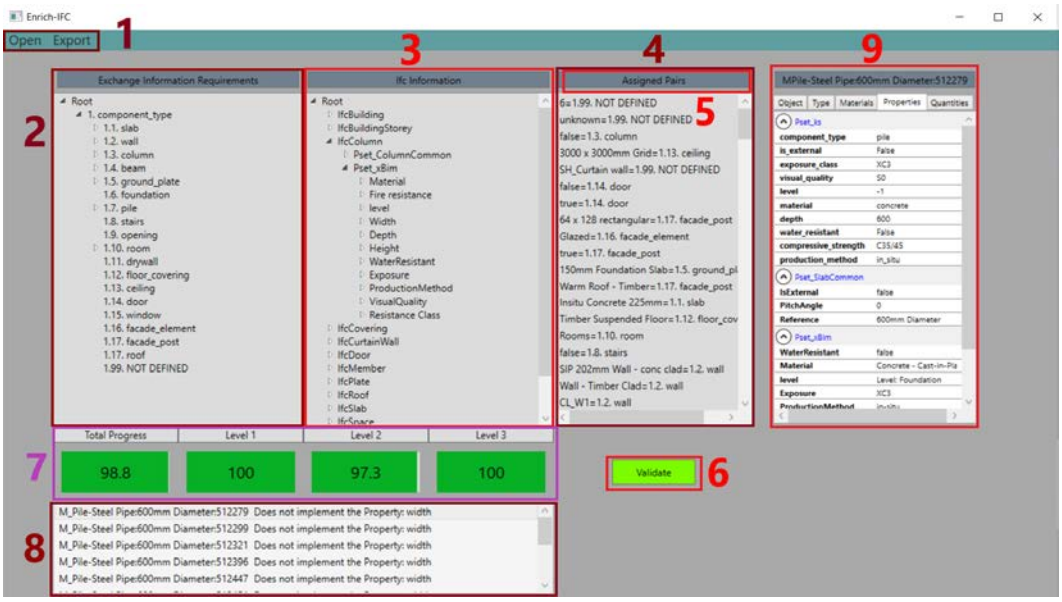


Figure 6.2: Main view of the GUI.

6.2.1 Menu

The menu shown in figure 6.3 allows the user to open and export different files. The prototype reads information requirements in the form of MVDs (*.mvdxml) or IMIR (*.xml), IFC documents (*.ifc), and assigned value tuples (*.xml). To store progress, the user can export IFC documents (*.ifc) and assigned value tuples (*.xml). Moreover, they can derive BCF files from error messages. Creating updated information requirements in the form of MVD or IMIR, however, is not yet supported.

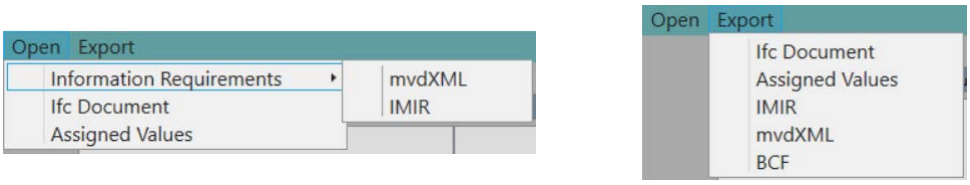
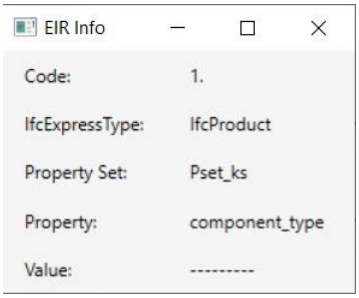


Figure 6.3: Component 1 from figure 6.2.

6.2.2 Information Trees



Once imported information requirements (MVD, IMIR), component 2 of figure 6.2 displays a tree that structures by properties and property values. Clicking on the items extends or collapses a node. Double-clicking opens the *EIR Info*, and the user obtains more information about the item. This view also lists the property set and *IfcExpressType* of an entity (figure to the left).

Opening an IFC model reads information from all elements of type *IfcProduct*. For each *IfcExpressType*, the tree in component 3 of figure 6.2 lists one element on the first level. The second level builds all property sets that the corresponding *IfcExpressType* implements. The third level repeats this concept with the properties of every property set, and lastly, all property values of the property create the fourth level.

6.2.3 Assigned Pairs

Component 4 in figure 6.2 lists all assigned value tuples. If the user has not imported an *Assigned Values* file (*.XML), the list is empty. For assigning the items, the user must import an IFC document as well as an MVD. Using both trees, the user can assign elements from the exchange requirements to items of the IFC tree. The user can either assign values to values or properties to properties. To assign an entity, the user drags the element from component 2 to component 3 and drops it on the required entity. Afterwards, component 4 lists the value tuple. This list only shows limited information. Thus, the user can click on a button (figure 6.2, component 5) to see further details.

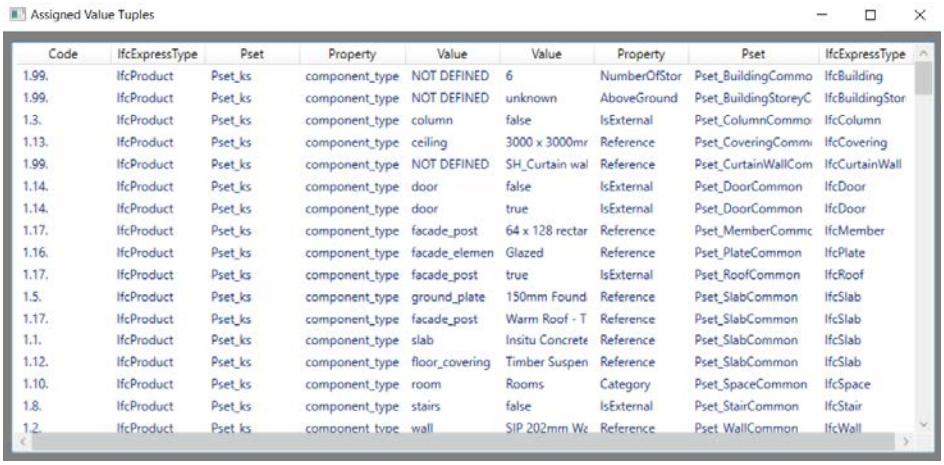


Figure 6.4: Assigned Value Tuples View: provide more information about the assigned pairs.

Clicking on the *Assigned Pairs* button opens a new window, the *Assigned Value Tuples* view (figure 6.4). The window lists the assigned pair and provides more information about the tuples. On the left side, it shows the data of the EIR value and, on the right side, the information of the linked IFC value.

6.2.4 Validate

If the user wants to receive feedback about the progress, they can click on the *Validate* button (figure 6.2, component 6). This button runs the validate method from the Enrich-IFC model. After that, the progress bars of the main view (component 7) display the overall model quality.

The four bars represent different levels. While the left one relates to the whole model, the other progress bars only represent one level of the required information. For each failed requirement, component 8 shows an error. Every error entry comprises a missing attribute and a related element. Once clicking on an error message, component 9 displays information about the associated element. This window is part of the xBim Xplorer repository (Lockley *et al.*, 2017). This data helps the user to identify whether they can solve this issue or not. If not, the Export menu enables the user to create BCF files and issue them manually, as the prototype does not connect to a BIM communication platform yet.

6.3 Prototype in Use

The following example describes the workflow of the prototype by applying it to a recent project. The project was executed by Geiger, who has provided the model for this thesis (figure 6.5). As this was a German project, the author translates the German properties and values where required. In this section, an expert wants to enrich the IFC model in a way that it complies with the information requirements from figure 5.6.

Once the application has started, the user imports the external IFC and the mvdXML using the Menu. Figure 6.6 displays the trees. Now, the user wants to assign the component types to the elements. Hence, they look through the information of the IFC model and identify overlaps. In this case, the 'Bauteiltyp' (component type) property of 'Pset_AC' clearly states that 'Unterzug' (joist) elements are beams. Therefore, the user drags 'Beam' and drops it on 'Unterzug'.

Figure 6.5: 3D view of the model.

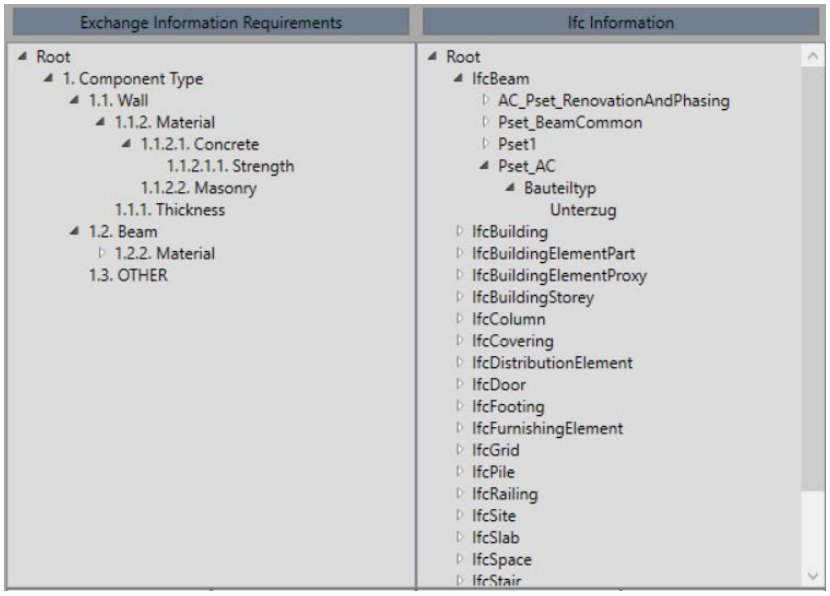


Figure 6.6: Tree view of the GUI after reading the data from the IFC model and the MVD.

Code	IfcExpressType	Pset	Property	Value	Value	Property	Pset	IfcExpressType
1.2.	IfcProduct	Pset1	Component Ty	Beam	Unterzug	Bauteiltyp	Pset_AC	IfcBeam
1.2.2.1.	IfcProduct	Pset3	Material	Concrete	Beton, Stahlbe	Reference	Pset_BeamCon	IfcBeam
1.2.2.1.	IfcProduct	Pset3	Material	Concrete	Beton, Stahlbe	Reference	Pset_BeamCon	IfcBeam
1.2.2.1.	IfcProduct	Pset3	Material	Concrete	Beton, Stahlbe	Reference	Pset_BeamCon	IfcBeam
1.2.2.1.	IfcProduct	Pset3	Material	Concrete	Beton, Stahlbe	Reference	Pset_BeamCon	IfcBeam
1.2.2.1.	IfcProduct	Pset3	Material	Concrete	Beton, Stahlbe	Reference	Pset_BeamCon	IfcBeam
1.2.2.1.	IfcProduct	Pset3	Material	Concrete	Beton, Stahlbe	Reference	Pset_BeamCon	IfcBeam
1.2.2.1.	IfcProduct	Pset3	Material	Concrete	Beton, Stahlbe	Reference	Pset_BeamCon	IfcBeam

Figure 6.7: *Assigned Value Tuples* view after assigning eight pairs.

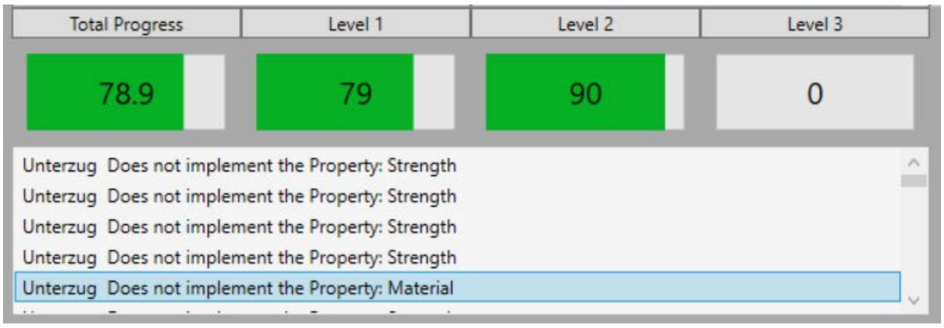


Figure 6.8: Model quality of the enriched IFC model.

On the second level, the components of type 'Beam' need information about their material. The user can find this data in the 'Reference' attribute of the 'Pset_BeamCommon'. All values imply that the material is concrete ('Beton'). Hence, the user drags the value 'Concrete' and drop it on these values. After that, the *Assigned Value Tuples* view displays the pairs shown in figure 6.7.

Now, the user clicks the validate button to calculate the overall model quality. The progress bars displayed in figure 6.8 express that the model quality on level 1 amounts to 79%. The quality is already high, as we only consider walls and beams in this example. All other components set the component type to 'OTHER'. For that reason, these components pass all their conditions. The progress bar for the second level displays a quality of 90%.

Moreover, the GUI lists the errors. The next step is to identify their sources. There are three potential error triggers: (1) missing information in the IFC model, (2) a missing option in the information requirements and (3) a missed out value tuple. Firstly, the user shall always check if they have missed out information. By double-clicking on such an error message, component 9 (compare figure 6.2) shows the information of the related IFC object. In this example, all failed conditions related to beams are attributable to (1). Thus, the user derives BCF files for issue communication.

The mvdXML also requires defining the component type 'Wall'. Looking through the IFC information shows that the 'Reference' property of 'Pset_WallCommon' implements this information. The three values, 'Wand Stahlbeton' (concrete wall), 'Wand Mauerwerk' (masonry

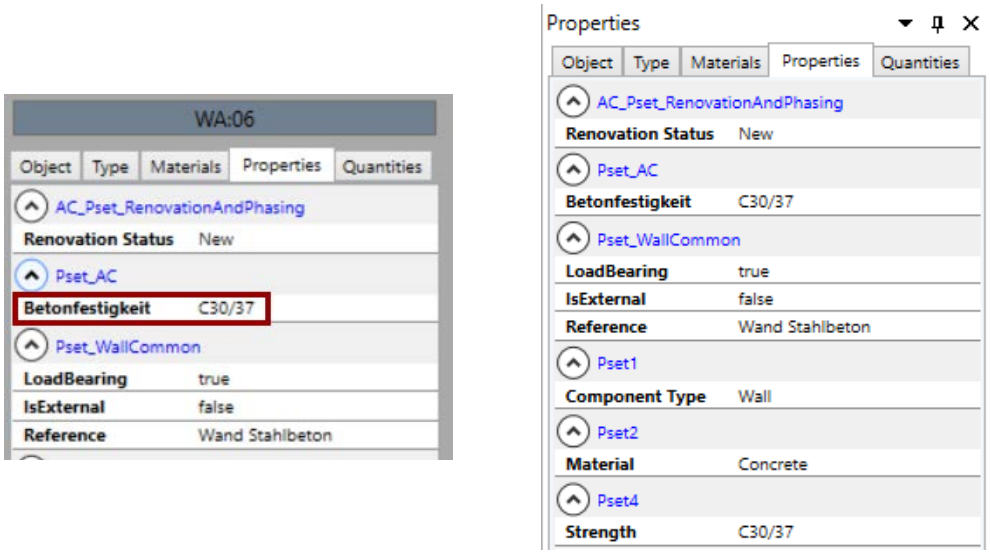


Figure 6.9: *WA:06* before (left) and after (right) the enrichment.

wall), and 'Holzwand' (timber wall) imply that these elements are walls. Besides this, they also express their material. However, the mvdXML does not define a value for the timber wall. Thus, the user must edit the EIR. The Prototype does not cover this feature. For that reason, the user must modify the mvdXML file (or the IMIR model) and import it again.

The concrete also needs to implement the compressive strength. Clicking on the error message shows the properties. The left part of figure 6.9 states that the 'Betonfestigkeit' property implements this data. After signing these values, there are two types of errors left: beams do not define the strength of the concrete and walls do not implement their thickness. As the model does not contain this information, the user exports the BCF file and sends it to the project team. The menu (compare figure 6.3) provides this feature.

Listing 6.1 shows the comment of such a BCF file. It highlights the GUID of the related component and the property that is missing. After sharing the data, the user can save the files using the menu. Therefore, they shall export the assigned values into an XML file. Listing 6.2 shows one assigned pair in the format the model exports it. The pair contains all information. However, once the model rereads the information, it will refer to the code. Thus, the user can edit the other attributes but not the code to assure consistency. If the model quality is high enough, the user can also export the enriched IFC document. This file implements all the information that the user has mapped before. Figure 6.9 shows exemplarily the properties of a wall before and after the enrichment.

Appendix D provides more details about the previous example, including a screen-cast showing the author interacting with the prototype's GUI

```
<Topic Guid="1be1c406-40a7-4cc9-a736-0b169034e815">
    <Title>Issue regarding alilWaCeRyxagIrqUrESqz</Title>
</Topic>
<Comment Guid="7198bb70-e776-4195-a995-2b7aa554c85f">
    <Status>Error</Status>
    <Date>2020-03-27T00:47:12.2859945+01:00</Date>
    <Author>unknown user</Author>
    <Comment>WA:80  Does not implement the Property: Thickness</Comment>
    <Topic Guid="1be1c406-40a7-4cc9-a736-0b169034e815" />
    <ModifiedDate>2020-03-27T00:47:12.2859945+01:00</ModifiedDate>
</Comment>
```

Listing 6.1: Markuf file of a BCF export.

```
<AssignedPair>
    <IfcItem IfcType="IfcBeam" PsetName="Pset_BeamCommon" PropertyName="
        Reference" PropertyValue="Beton, Stahlbeton 600 x 500" />
    <EirItem ExpressType="IfcProduct" PsetName="Pset3" PropertyName="
        Material" PropertyValue="Concrete" Code="1.2.2.1." />
</AssignedPair>
```

Listing 6.2: Assigned pair: the *IfcItem* defines the information from the IFC model and the *EirItem* from the MVD. When reading the file, the algorithm refers to the code attribute of the *EirItem*.

6.4 Summary

This chapter has described the functionalities of the prototype. The prototype is still under development. Thus, it does not cover all the requirements that the ideal system should have. Table 6.1 lists the required features from section 6.1 and concludes whether the prototype covers them or not. The prototype lacks the possibility of maintaining information requirements using a GUI. However, the IMIR-model provides methods to do this.

To conclude, the author has developed a prototype to enrich a model according to information requirements. This prototype integrates the most common specification published by buildingSMART: IFC, mvdXML, and BCF.

Feature	Comment	Covered?
(1) Maintain information requirements	IMIR model provides methods; No GUI implemented	partially
(1) Derive IMIR and mvdXML files		yes
(2) Read and export IMIR and mvdXML files	mvdXML import does not cover the whole schema	partially
(3) Read IFC		yes
(4) Assign values		yes
(5) Validate the model	Approximation of the quality	yes
(6) Export BCF files	No connection to a BIM collaboration platform	yes
(6) Assign values depending on given information		yes
(6) Update EIR		no
(7) Read and export assigned values		yes
(8)Create enriched IFC file		yes

Table 6.1: Summary of the functionalities of the prototype. The numbers refer to the requirements mentioned in section 6.1.

Chapter 7

Case-study: Quantity-Takeoff

The previous chapters have introduced the Enrich-IFC-approach and converted it into a prototype. After showing the general functionalities, the following chapter applies the approach to a fictive project.

The primary purpose of the Enrich-IFC-approach is to prepare BIM models for automatised BIM Uses. This case study applies the proposed approach to the BIM Use Cost Estimation. From the perspective of the main contractor, it compares the conventional approach with the innovative Enrich-IFC-approach and highlights the benefits.

7.1 Using iTWO for QTO

RIB iTWO is a software application providing solutions for QTO, tendering and more. In this context, we only focus on QTO. Using iTWO for QTO separates into two levels: tasks on the organisational level and tasks on the project level. On the organisational level, the appointing party must maintain a library defining all deliverables from further projects. This library defines deliverables listed by components. Table 7.1 exemplifies what such a library can look like. On the project level, applying iTWO to QTO structures in four steps:

1. Check the model quality
2. Define deliverables
3. Assign deliverables to components of the BIM model
4. Calculate quantities

After the main contractor received the external BIM model, they check if it complies with the information requirements from the BEP. This section does not describe conventional approaches to validate a model. However, the author highlights that the Enrich-IFC-approach

Component Type : Room		**Ground Plate**	
Floor finish: parquet	$[m^2]$	Framework surface outer parts	$[m^2]$
Floor finish: tiles	$[m^2]$	Concrete volume C20/25	$[m^3]$
Ceiling finish: suspended	$[m^2]$	Concrete volume C25/30	$[m^3]$
Ceiling finish: painted	$[m^2]$	Concrete volume C30/37	$[m^3]$
Wall finish: painted	$[m^2]$	Concrete volume C35/45	$[m^3]$
Wall finish: plaster	$[m^2]$	Reinforcement B500B	$[t]$
Wall finish: tiles	$[m^2]$		

Table 7.1: Excerpt of the deliverables library for the components 'Room' (left) and 'Ground Plate' (right).

covers both, validation of the model and automating the BIM Use (for this application: Cost Estimation). Model Validation is also a BIM Use. Hence, this chapter covers two BIM Uses.

For defining the deliverables, the user can upload the library mentioned before from the organisational level. The two examples in table 7.1 illustrate the idea: the library contains all deliverables that occurred in previous projects. For a room, these may be different finishes, and for a slab, volumes of different concrete strengths classes.

After defining the deliverables, the contractor starts assigning them to components of the BIM model. For this, iTWO provides three options. Firstly, the user can manually drag deliverables from a table and drop them on selected components. Although this is the most intuitive way, it is not very suitable due to the high manual workload. A second option represents creating dynamic filters that generate a selection based on attributes. This is faster than the first option. Moreover, it is more comprehensible to identify assigned components. Such a filter queries the elements with the following syntax: $Object(@Pset\_ks/component\_type ==' ground\_plate')$. This filter selects all objects that implement the value 'ground_plate' in the 'component_type' property. Lastly, the user can assign all components to each deliverable and define the subset in the calculation of the quantities. The main benefit of this option is the centralised filtering as all selection procedures happen in one equation. However, the filters may become very long and incomprehensible. Such a filter may look as follows:

$$QTO(Type := "Volume"; Unit := "m^3"; Component := "\${Pset\_ks/component\_type} ==' ground\_plate' and \${Pset\_ks/compressive\_strength} ==' C25/C30'")$$

This filter firstly defines the expected result. In this case, the calculation returns a volume with m^3 as their unit. After that, the 'Component' argument specifies the instances the calculation applies to. Here, the equation selects all components of type 'ground_plate' that also define a compressive strength of 'C25/30'.

The syntax of the filters perfectly illustrates the problem of this approach and the motivation of the Enrich-IFC-approach. The filter refers to specific attributes of the IFC model. As different parties use different naming, the main contractor has to update all filters for every project. For complex projects, these can easily amount to several hundreds of filters. For that reason, it is not possible to execute the Cost Estimation automatised. However, if all models were using the same labelling, the main contractor could adapt the deliverables-library to the labelling standard. Hence, they could import the IFC model and automatically execute the QTO. And this is what the Enrich-IFC-approach realises.

7.2 Enrich-IFC-Approach

The previous chapter concluded two issues that foil successful model-based and automatised QTO: (1) model validation is not automatised and (2) the library of deliverables must be updated for every project. The Enrich-IFC-approach validates a BIM model by using an MVD. Besides this, instead of updating the deliverables-library, the approach enables the contractor to enrich the BIM model and execute the QTO automatised.

7.2.1 Overview

The project is about a building comprising two levels. The ground floor consists of a living room, a kitchen and four minor rooms. The first floor has three bedrooms, three bathrooms and a balcony. It has piles as a foundation and connects to a garage. Figure 7.1 shows a 3D view of the asset, and appendix C illustrates the floor plans. The model is one of Autodesk's sample projects (Autodesk, 2020).

The fictive project team consists of two parties: The contractor (appointing party) and the architect (appointed party). At the beginning of the project, both participants agree on a BEP. This plan contains the information requirements defined by the appointing party. Here, both parties agreed that they use the BIM method for Cost Estimation. This guarantees that the appointed party has to provide all information required to execute a model-based QTO. Hence, the contractor must define the required data before the architect starts creating the model and before finalising the BEP.

Applying this BIM Use to projects requires two tasks to be executed on an organisational level before the project starts:

1. Define IMIR including information and labelling requirements, and derive MVD
2. Create and maintain a library in iTWO defining all deliverables and adapt filters and equations to the labelling standard

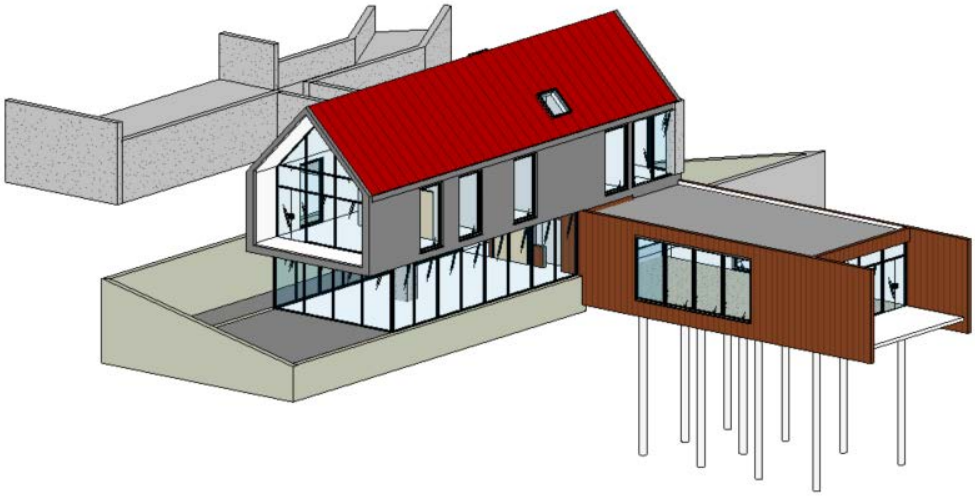


Figure 7.1: 3D View of the sample projects. Left: Project A. Right: Project B. (Autodesk (2020))

The project delivery starts after that. The architect creates the model and issues it to the contractor at significant moments. These data drops are also part of the BEP. At every information issue, the contractor estimates the costs. For that reason, they have to create rules for quantity take-off. These rules filter objects by their attributes to retrieve quantities. In this context, the contractor has already implemented a library of filters from other projects using the internal labelling standard. Thus, they adapt the external model to the IMIR using the prototype (section 6).

In the last step, before starting with the actual Cost Estimation in iTWO, the contractor must validate whether the model applies to the IMIR or not. The prototype from section 6 also covers this feature. Additionally, they use the xBIM Xplorer to validate the model twice.

To sum up, there are six tasks:

1. Define IMIR and derive MVD	
2. Create library defining deliverables and adapt filters and equations to (1)	*Organisational Level*
3. Receive external BIM model	*Project Level*
4. Adapt the external model to IMIR and derive new internal BIM model	
5. Validate the internal BIM model	
6. Estimate costs	

7.2.2 Determine Information Requirements

The contractor firstly determines the required information by creating an MVD representing a set of information and labelling requirements. Hence, it specifies an IMIR. The project described in section 7.2.1 contains architectural and structural elements. It does not include HVAC equipment. Thus, it must comprise

- all structural and non-structural components.
- shape, size, and location of the components.
- materials, visual quality, production method and fire rating of the components.

The contractor has decided to use the IMIR data model to define the information requirements. The component type represents the first level of the model. This property must define significant component types, such as walls, beams, foundations, and rooms. The second level represents the component type-specific properties. For instance, the component type 'ground_slab' requires the properties: 'thickness, 'material', and 'is_external'. The material, for example, requires further information. Figure 7.2 shows an excerpt of the information requirements and appendix D provides more information about the IMIR model. This data model allows deriving an MVD that the contractor can now incorporate to the BEP.

7.2.3 Create Deliverables Library in iTWO

Section 7.1 introduced how to use iTWO for QTO. On the organisational level, the contractor has developed a library defining deliverables. Figure 7.3 lists all deliverables by component and shows that the library already selects the elements from the model that are used for calculating the quantities. The author has decided to use a mixture of the second and third option (compare 7.1). This generates more flexibility and always makes sure that the correct elements are selected. Therefore, dynamic filters select all elements of one component type (figure 7.3, column 4). The calculation of the quantities bases on these selections. However, the equation to obtain quantities also implement filters for further distinction (figure 7.3, column 6). The following example illustrates that.

Using the example from table 7.1, the author has created a dynamic filter selecting all elements of type 'Room': $Object(@Pset\_ks/component\_type ==' room')$.

For these elements, the author wants to calculate the area of specific floor finishes, wall finishes and ceiling finishes. Hence, every deliverable defines a filter in their equation. This filter selects all elements with the corresponding property. For instance, the following line shows the equation to calculate the floor area for all rooms with a parquet finish:
$QTO(Type := "FloorArea"; Unit := "m^2"; Component :=$
$"\${Pset\_ks/floor\_finish} ==' parquet'")$

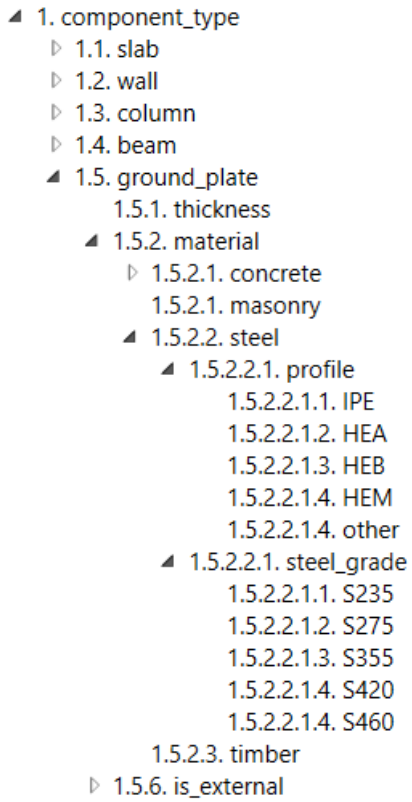


Figure 7.2: Information requirements for the model.

	1			Ceiling			
	2			Column			
	3			Door			
	4			Drywall			
	5			Facade_Elements			
	6			Facade_Post			
	7			Floor_Covering			
	8			Foundation			
	9		Ground_plate	Ground_Plate			
§	9.10			Formwork	QTO(Typ:="Mantelfläche";ME:="m²")	m2	9,612
§	9.20			Volume Concrete C20/25	QTO(Typ:="Volumen";ME:="m³";Bauteil:="${Pset_ks\compressive_strength} =='C20/25' ")	m3	0,000
§	9.30			Volume Concrete C25/30	QTO(Typ:="Volumen";ME:="m³";Bauteil:="${Pset_ks\compressive_strength} =='C25/30' ")	m3	0,000
§	9.40			Volume Concrete C30/37	QTO(Typ:="Volumen";ME:="m³";Bauteil:="${Pset_ks\compressive_strength} =='C30/37' ")	m3	18,509
§	9.50			Volume Concrete C35745	QTO(Typ:="Volumen";ME:="m³";Bauteil:="${Pset_ks\compressive_strength} =='C35/45' ")	m3	0,000
	10			Pile			
	11		rooms	Room			
§	11.10			Floor: Tiles	QTO(Typ:="Bodenfläche";ME:="m²";Bauteil:="${Pset_ks\floor_finish} =='tiles' ")	m2	7,917
§	11.20			Floor: Parquet	QTO(Typ:="Bodenfläche";ME:="m²";Bauteil:="${Pset_ks\floor_finish} =='parquet' ")	m2	187,005
§	11.30			Walls: Paint	QTO(Typ:="Mantelfläche";Bauteil:="${Pset_ks\wall_finish} =='paint' ")	m2	530,284
§	11.40			Walls: plaster	QTO(Typ:="Mantelfläche";Bauteil:="${Pset_ks\wall_finish} =='plaster' ")	m2	72,339
§	11.50			walls: tiles	QTO(Typ:="Mantelfläche";Bauteil:="${Pset_ks\wall_finish} =='tiles' ")	m2	37,501
§	11.60			ceiling suspended	QTO(Typ:="Deckenfläche";Bauteil:="${Pset_ks\ceiling_finish} =='suspended' ")	m2	0,000
§	11.70			ceiling paint	QTO(Typ:="Deckenfläche";Bauteil:="${Pset_ks\ceiling_finish} =='paint' ")	m2	95,603
	12			Slab			
	13			Wall			
	14			Window			

Figure 7.3: Screenshot from iTWO listing the deliverables and calculating the quantities. Some of them amount to zero as not all deliverables occur in the project (language is set to German).

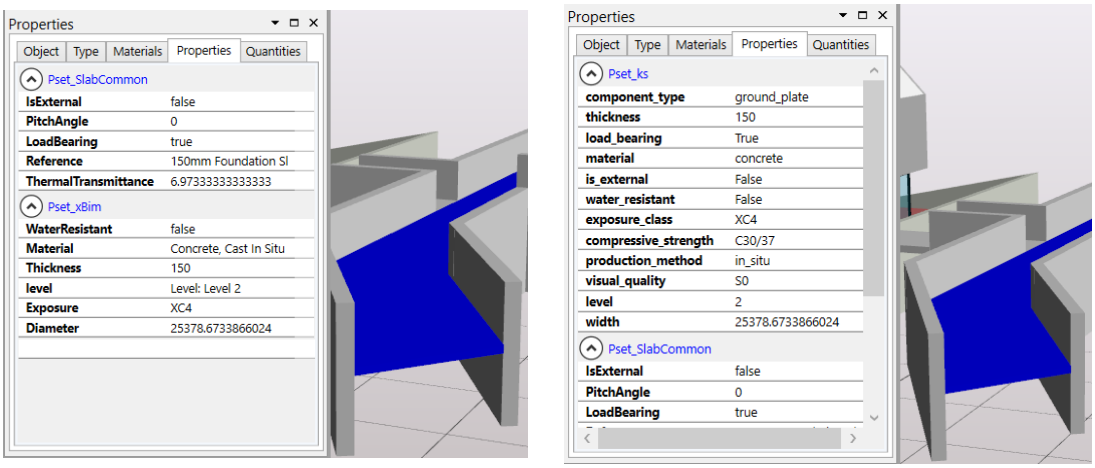


Figure 7.4: Properties of the slab before (left) and after (right) applying the Enrich-IFC-approach.

The author defines those filters and calculation for every deliverable of the library. Still, this happens on an organisational level and does not belong to the project. Figure 7.3 shows the deliverables of the project. In the fourth column, the dynamic filter is assigned, and the sixth column lists the calculation. Column eight already shows the result of the calculation for the model from figure 7.1. Some results amount to zero as the model does not define elements of all types. Section 7.2.6 describes the process to obtain these results.

7.2.4 Adapt External Model to IMIR

The main contractor executes the previous steps on an organisational level. Thus, the step is not project-specific. Now, the project starts and the project team has agreed on a BEP. After that, the project delivery begins, and the architect issues the model to the main contractor. However, the architect used their own labelling standard and did not follow the terms in the MVD. Thus, the contractor must adapt the model to the IMIR.

The author assigns values from the requirements to the external IFC model. This happens for the whole model and its components. Here, an example illustrates the usage. The model contains foundation piles that were modelled as 'IfcSlab' entities. The main contractor can easily assign this element to the component type 'pile'. Now, poor model quality has not too much impact. Additionally, the mvdXML created above will check the required information for piles and not for slabs.

Figure 7.4 illustrates another benefit. The foundation slab requires information about the material. The external model defines this property as 'Concrete, Cast in Situ'. Hence, one property stores two characteristics. The author assigns both values to the IMIR, 'concrete' of 'material' and 'in_situ' of 'production_method', to 'Concrete, Cast in Situ'.

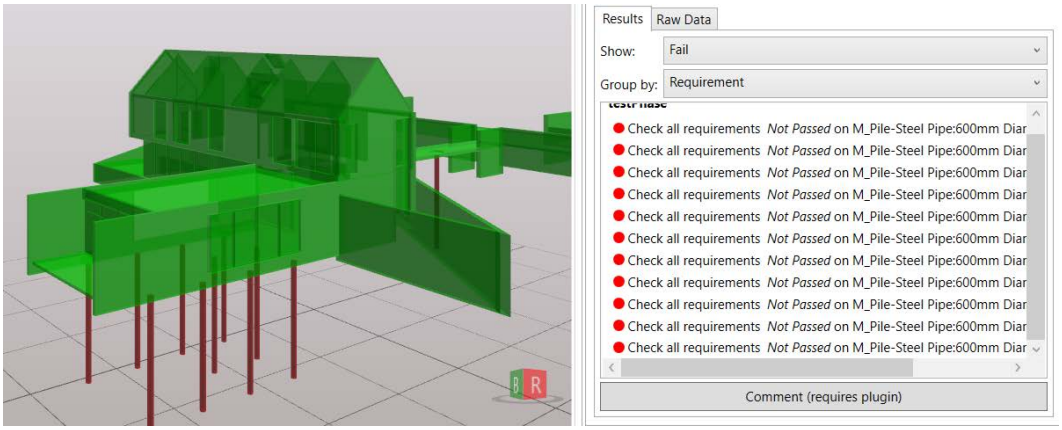


Figure 7.5: Validation: the xBim Xplorer highlights the passed (green) and failed (red) concepts.

7.2.5 Validate the Model

After assigning all information, the prototype validates the model. This concludes that the external model does not define the thickness of the foundation piles. Hence, the user derives BCF issues and communicates them with the architect.

Even though the prototype implements a validation feature, this section describes the validation of the model using xBIM Xplorer to double-check the model's quality. The tool applies all concepts defined in the MVD and shows graphically if they are fulfilled. Figure 7.5 concludes the same as the prototype: All components of the model comply with the information requirements of the MVD. The only exceptions are the foundation piles. As they do not implement the property 'width', the tool returns that the concept has failed. Due to the hierarchy of the mvdXML file, the tool only shows that the concept fails, however, it does not illustrate why. In this example specifically, it expresses that a concepts has failed, but the user cannot determine what information is missing.

7.2.6 Execute QTO Using iTWO

Section 7.1 introduced how to use iTWO for QTO, and in section 7.2.3, the contractor has developed a library defining deliverables. All work already happened on the organisational level. The deliverables refer to the labelling from the internal labelling convention. Hence, there is no need to adapt the filters or the equations. The user can simply import the enriched internal IFC model into iTWO. From there they can calculate all quantities. Thus, the approach is automatised and fully independent. However, the library of deliverables may not contain all components. Hence, the contractor has to update and extend the library. That library is growing continuously over several projects.

Task	Enrich-IFC	Conventional
Create IMIR	5.0	-
Create EIR	-	3.0
Enrichment	3,0	-
Validate	0.50	2.50
Deliverables	4.0	4.0
iTWO	1.0	3.0
Total	**13.5**	**12.5**

Table 7.2: Time consumption of executed tasks within using the two approaches.

7.3 Conclusion

The author has applied both approaches to the fictive project and compared the time consumption of the different tasks. Table 7.2 and figure 7.6 show the results. Using the conventional approach to the project amounts to a total of 12.5 hours, which represents the reference value and corresponds to 100%. The Enrich-IFC method exceeds that by 8%. Here, the Enrichment needed 3 hours, the model validation 0.5 hours, and the adjustments in iTWO only 1 hour. In contrast to that, the conventional approach required 2.5 hours for model validation and 3 hours for calculating the quantities using iTWO.

The required workload illustrates that the primary value of the Enrich-IFC-approach takes effect when using one model for multiple BIM Uses. While the conventional approach requires to adjust the algorithms for each BIM Use, the Enrich-IFC-approach enables a company to create automatised downstream processes. Assuming that the workload for the second approach also amounts to 3 hours, the Enrich-IFC-approach only increases by 1 hour. Thus, the Enrich-IFC-approach is already faster in total (14.5 hours) than the conventional approach (15.5 hours).

These assumptions are highly uncertain. However, there are other potentials to minimise the workload of the Enrich-IFC-approach. For example, when the IMIR misses a value, the user must update and reload the IMIR into the prototype. Moreover, the author has detected improvements regarding the usability of the prototype. For instance, copying assigned values to other entities, declaring errors as redundant, or improving the performance of the algorithms would decrease time consumption.

The author expects the novel approach to become more efficient than the conventional after improving some of the previously mentioned problems. However, there is no conclusion possible, as this requires further testing. Still, there are other benefits of the proposed approach.

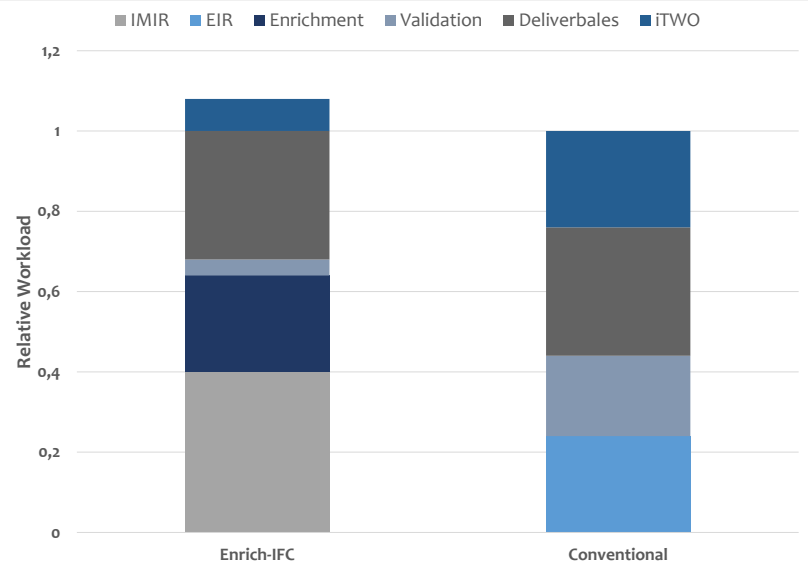


Figure 7.6: Comparison of the workload for the different tasks of both approaches.

One main difference is the cohesive and flexible structure of the IMIR model compared to common EIR structures. Changes in the labelling are more comfortable to adjust, and the hierarchy simplifies model validation by a lot. Thus, the approach secondly establishes a comprehensible way to validate models and identify information leaks. The missing data can then be communicated with project participants using BCF. Another benefit is that the user does not need in-depth knowledge about a software application such as iTWO. There is no need to write complicated filters or algorithms. The user simply translates information via drag and drop.

Chapter 8

Conclusion and Future Development

This work has contributed a new approach for structuring both information requirements and labelling requirements and enrich an IFC document by assigning desired information from the labelling standard to given information from the IFC model. In the following chapter, the author discusses the findings of the work and proposes future work to improve the concept.

8.1 Conclusion

One main objective of this work was to provide a cohesive model that covers both storing information requirements and validating BIM models. Therefore, the author has proposed the IMIR data model that structures information hierarchically and stores properties based on references, which eases maintaining the model. The hierarchy extends information requirements by a second dimension: the labelling requirements. These enable an algorithm to analyse data and decide what element of a model must implement what attributes. To finally include this model into current regulations, the IMIR model derives mvdXML files.

The second part of the approach begins here. It uses information requirements and enriches IFC models according to them. The work has illustrated a process to supplement data to an IFC model according to the given data. The Enrich-IFC-model converts this approach into a technical concept. It provides methods to execute the process and, eventually, a prototype that enables a user to interact with the model graphically. This part of the thesis has closed the gap of how to 'translate' given information of an IFC model into individual preferences.

Applying the proposed solution to a fictive project and comparing it with a conventional approach has detected benefits and potential improvements. The prototype lacks an opportunity

to adjust information requirements using a GUI. Besides this, it experiences performance issues while interpreting large IFC files. But, the approach stands out with a comprehensive structure for information requirements that minimise maintenance efforts. Additionally, it eases model validation by including it into the prototype, as well as deriving mvdXML files. Lastly, realising the automatisation of BIM Uses is impossible without an approach like this.

In summary, the thesis has closed the identified gaps. The Enrich-IFC-approach introduces conceptual solutions for the mentioned problems and adapts to the current regulations by buildingSMART as it reads and creates IFC, BCF, and mvdXML files. Finally, it only relies on open-based standards and software tools, which was the main objective of the author. Still, some aspects require further investigations:

- The IMIR shall include several LODs. For example, instead of defining several property values, a property implements different LODs which, again, reference the property values. This would decrease the workload for maintaining multiple IMIR models.
- The Enrich-IFC-approach does not support updating IMIR models using a GUI. Thus, the features implemented in the IMIR model shall integrate into the GUI of chapter 6.
- The Enrich-IFC-model requires the full support of mvdXML import. The author has compared several approaches and proposed using the 'Description' attribute of *TemplateRule* to store the unique code of an element. It may be more promising to use GUIDs instead of WBS codes.
- The prototype does not connect to a BIM collaboration platform what increases the workload to communicate issues via BCF.
- Machine Learning algorithms may replace the user and assign the values automatically. In this case, a neural network identifies connections between the two information sources (IFC and IMIR) and derives the Enrich-IFC data model.

8.2 Vision

The proposed approach has introduced a solution that fits in current regulations. However, the BIM environment progresses steadily and changes rapidly. The frame for the proposal may seem tight, and the change of some boundary conditions may decrease its usage. However, there are more situations where to apply the Enrich-IFC-approach. The following paragraphs describe three of them.

1. Model validation before publishing information:
The Enrich-IFC-approach cannot only enrich documents received by other parties, but it may also enrich IFC models before sending them to the appointing party. The appointed party can derive IMIR models from the information requirements defined in the BEP and store them in an mvdXML file. This file then functions as the base to validate IFC models before sending it to the appointed party. The primary outcome here is that an appointed party exclusively publishes models that comply with the EIR.

2. Clients oblige planners to follow the client's internal labelling convention:
Similar to (1), the provider of the model uses the Enrich-IFC-approach. In this case, the appointed party can still follow their internal labelling standard. After creating the model, they can supplement it using the proposed approach and issue it to the client. Similar to the previous scenario, the appointed party makes sure to comply with the information requirements defined by the client.

3. BIM maturity level 3 becomes a reality:
As mentioned before, the AEC industry currently aims at BIM level 2. This surely is the next step and is partly implemented yet. Although significant problems simulate impossibility, the future can always change. So how can the Enrich-IFC-approach contribute to projects where all participants work on the same model continuously?

This change has an intense impact on the Enrich-IFC-approach and requires adjustments. Firstly, however, one part still fulfils its purpose: the IMIR model stores information requirements and the derived mvdXML check the server-based model continuously.

The second part, the supplement of the IFC model, experiences more impact. Still, on this BIM level, it is not regulated how to label information. Developing a labelling standard like the bsDD may solve the issue. However, the Enrich-IFC-approach could also close this gap. Whenever a model author adds or retrieves information, the Enrich-IFC-model translates the values in their individual preference. The translation bases on the *AssignedValues* form the Enrich-IFC-model. Thus, the user will always see the information in a way they want to see it, although other project participants see the data differently. As all the collaboration happens server-based, the Enrich-IFC-model works in real-time. For this situation, it is indispensable to incorporate the Enrich-IFC-model into CDEs.

The primary difference to the bsDD represents the flexibility in naming the data. While the bsDD uses a similar concept, it only proposes a few languages. Incorporating the Enrich-IFC-model into a CDE, however, allows a project participant to still label information according to their preferences. Besides this, the project participant is more flexible with data that the bsDD does not cover, as they can simply add internal entities.

—Appendix—

Appendix A

MvdXML for Validation

Listing A.1: Strict applicability: the *Applicability* specifies the subset of the IFC schema, and the *TemplateRules* test define tests that apply on the subset.

```
<ConceptRoot uuid="0e93f597-f5e1-475b-87a7-eb007993a50d" name="All External Walls" applicableRootEntity="IfcElement">
    <Applicability>
        <Template ref="5c252c86-5bff-4372-9a27-b794069f9fbb" />
        <TemplateRules operator="and">
            <TemplateRule Parameters="O_PsetName[Value]='Pset_ks' AND O_PName[Value]='Component_Type' AND O_PSingleValue[Value]='Wall'" />
        </TemplateRules>
    </Applicability>
    <Concepts>
        <Concept uuid="983ddc5d-c0c8-47c9-8491-97add7677139" name="All Elements">
            <Template ref="5c252c86-5bff-4372-9a27-b794069f9fbb" />
            <Requirements>
                <Requirement applicability="export" exchangeRequirement="ae70f764-938b-4cf7-9814-c29a47f56b0e" requirement="mandatory" />
            </Requirements>
            <TemplateRules operator="and">
                <TemplateRule Parameters="O_PsetName[Value]='Pset_ks' AND O_PName[Value]='Material' AND O_PSingleValue[Exists]=True" />
                <TemplateRule Parameters="O_PsetName[Value]='Pset_ks' AND O_PName[Value]='Thickness' AND O_PSingleValue[Exists]=True" />
                <TemplateRule Parameters="O_PsetName[Value]='Pset_ks' AND O_PName[Value]='Fire Resistance Class' AND O_PSingleValue[Exists]=True" />
```

```
18                  <TemplateRule Parameters="O_PsetName[Value]='Pset_ks' AND
                        O_PName[Value]='Level' AND O_PSingleValue[Exists]=True" />
19                  <TemplateRule Parameters="O_PsetName[Value]='Pset_ks' AND
                        O_PName[Value]='LoadBearing' AND O_PSingleValue[Exists]=
                        True" />
20              </TemplateRules>
21          </Concept>
22      </Concepts>
23  </ConceptRoot>
24  <ConceptRoot uuid="0e93f597-f5e1-475b-87a7-eb007993a50d" name="All External
        Walls" applicableRootEntity="IfcElement">
25      <Applicability>
26          <Template ref="5c252c86-5bff-4372-9a27-b794069f9fbb" />
27          <TemplateRules operator="and">
28              <TemplateRule Parameters="O_PsetName[Value]='Pset_ks' AND
                    O_PName[Value]='Material' AND O_PSingleValue[Value]='
                    Concrete'" />
29          </TemplateRules>
30      </Applicability>
31      <Concepts>
32          <Concept uuid="983ddc5d-c0c8-47c9-8491-97add7677140" name="
                Material:Concrete">
33              <Template ref="5c252c86-5bff-4372-9a27-b794069f9fbb" />
34              <Requirements>
35                  <Requirement applicability="export" exchangeRequirement="
                        ae70f764-938b-4cf7-9814-c29a47f56b0e" requirement="
                        mandatory" />
36              </Requirements>
37              <TemplateRules operator ="and">
38                  <TemplateRule Parameters="O_PsetName[Value]='Pset_ks' AND
                        O_PName[Value]='Strength' AND O_PSingleValue[Exists]=
                        True" />
39                  <TemplateRule Parameters="O_PsetName[Value]='Pset_ks' AND
                        O_PName[Value]='Production Method' AND O_PSingleValue[
                        Exists]=True" />
40                  <TemplateRule Parameters="O_PsetName[Value]='Pset_ks' AND
                        O_PName[Value]='Exposure Class' AND O_PSingleValue[
                        Exists]=True" />
41              </TemplateRules>
42          </Concept>
43      </Concepts>
44  </ConceptRoot>
```

Listing A.2: Strict hierarchy: one concept comprises all rules. The *TemplateRules* apply to all elements specified in the *applicableRootEntity* attribute.

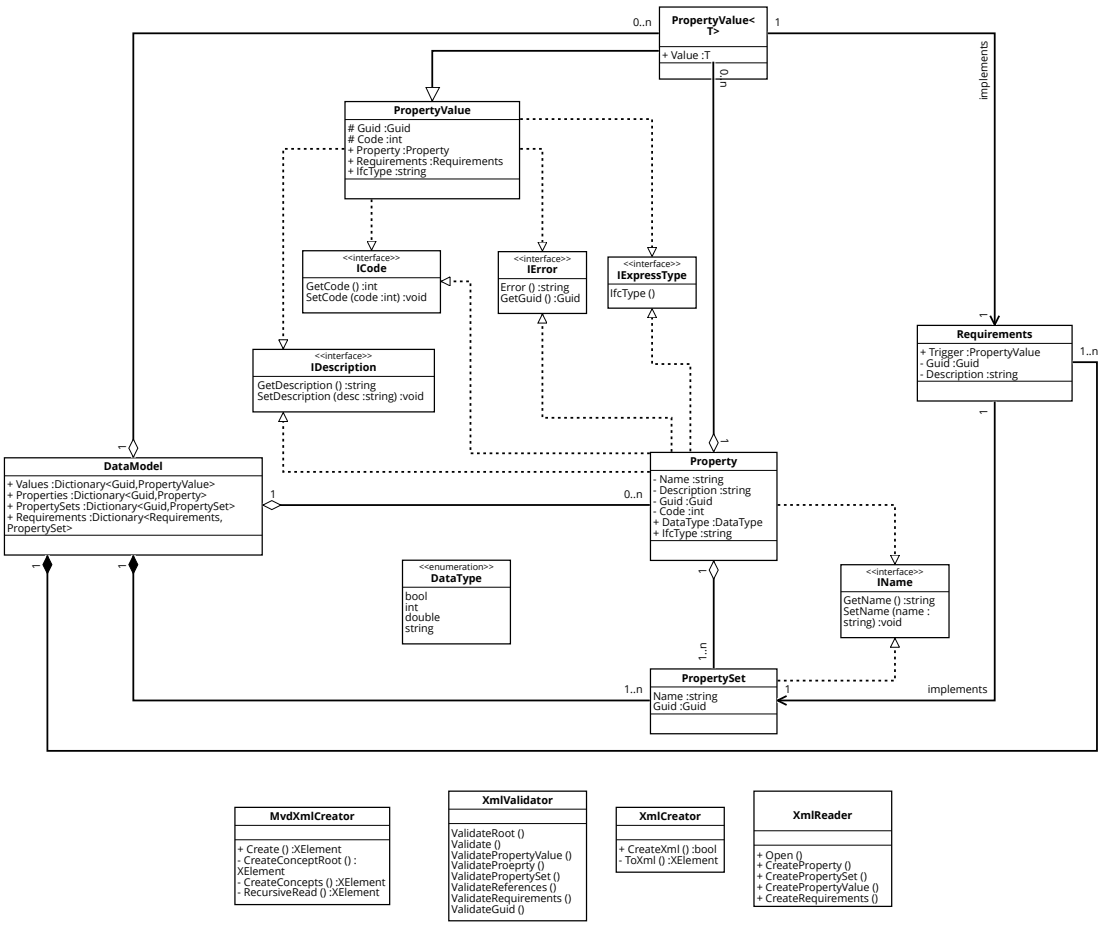
```
<ConceptRoot uuid="0e93f597-f5e1-475b-87a7-eb007993a50d" name="All External Walls" applicableRootEntity="IfcElement">
    <Applicability>
        <Template ref="5c252c86-5bff-4372-9a27-b794069f9fbb" />
        <TemplateRules operator="and">
            <TemplateRule Parameters="O_PsetName[Exists]='True'" />
        </TemplateRules>
    </Applicability>
    <Concepts>
        <Concept uuid="983ddc5d-c0c8-47c9-8491-97add7677139" name="All Elements">
            <Template ref="5c252c86-5bff-4372-9a27-b794069f9fbb"/>
            <Requirements>
                <Requirement applicability="export" exchangeRequirement="ae70f764-938b-4cf7-9814-c29a47f56b0e" requirement="mandatory"/>
            </Requirements>
            <TemplateRules operator="and">
                <TemplateRule Parameters="O_PsetName[Value]='Pset_ks' AND O_PName[Value]='Component_Type' AND O_PSingleValue[Value]='Wall'"/>
                <TemplateRules operator="and">
                    <TemplateRule Parameters="O_PsetName[Value]='Pset_ks' AND O_PName[Value]='Thickness' AND O_PSingleValue[Exists]=True"/>
                    <TemplateRules operator="or">
                        <TemplateRules operator="and">
                            <TemplateRule Parameters="O_PsetName[Value]='Pset_ks' AND O_PName[Value]='Material' AND O_PSingleValue[Value]='Masonry'"/>
                        </TemplateRules>
                        <TemplateRules operator ="and">
                            <TemplateRule Parameters="O_PsetName[Value]='Pset_ks' AND O_PName[Value]='Material' AND O_PSingleValue[Value]='Concrete'"/>
                            <TemplateRule Parameters="O_PsetName[Value]='Pset_ks' AND O_PName[Value]='Strength' AND O_PSingleValue[Exists]=True"/>
                            <TemplateRule Parameters="O_PsetName[Value]='Pset_ks' AND O_PName[Value]='Production Method' AND O_PSingleValue[Exists]=True"/>
                            <TemplateRule Parameters="O_PsetName[Value]='Pset_ks' AND O_PName[Value]='Exposure Class' AND O_PSingleValue[Exists]=True"/>
                        </TemplateRules>
                    </TemplateRules>
```

```
29                          <TemplateRule Parameters="O_PsetName[Value]='Pset_ks' AND O_PName[Value]='Fire Resistance Class' AND O_PSingleValue[Exists]=True" />
30                          <TemplateRule Parameters="O_PsetName[Value]='Pset_ks' AND O_PName[Value]='Level' AND O_PSingleValue[Exists]=True" />
31                          <TemplateRule Parameters="O_PsetName[Value]='Pset_ks' AND O_PName[Value]='LoadBearing' AND O_PSingleValue[Exists]=True" />
32                      </TemplateRules>
33                  </TemplateRules>
34              </Concept>
35          </Concepts>
36  </ConceptRoot>
```

Appendix B

Data Model Design

B.1 IMIR UML Diagram



The following chapter describes the core features of the IMIR *DataModel* class that functions to create and maintain information requirements hierarchically. The data model is under development. Hence, not all features are fully incorporated nor tested. The following table shows the properties of class and explains their purpose:

B.2 How to Create an IMIR Model

Property	Purpose
Values	Type: $Dictionary < Guid, PropertyValue >$. Contains all *PropertyValue*s of the data model.
Properties	$Dictionary < Guid, Property >$. Contains all *Property*s of the data model.
Sets	$Dictionary < Guid, PropertySet >$. Contains all*PropertySet*s of the data model.
Requirements	$Dictionary < Requirements, PropertySet$. Contains all *Requirements* of the data model. Moreover, it contains relationships between *Requirements* and *PropertySet*s

The following table describes the methods in the *DataModel* class. To readability reasons, the methods do not show their input values. For further information, please check appendix D.

Method	Purpose	Returns
GetType()	Determines the data type of an object and returns the corresponding enum: 0=Bool, 1=Int, 2=Double,3=String	*DataType*
ValidateDataType()	Checks whether the input value equals to a *DataType* or not.	**bool**
AddRequirements()	Adds *Requirements* to a property value of the data model. If the value equals **null** it means that all elements of the model must implement these *Requirements*. Returns **false** if an error occured.	**bool**
CopyRequirements()	Copies existing *Requirements* from the model and adds them to another *PropertyValue*. Returns **false** if an error occurred.	**bool**

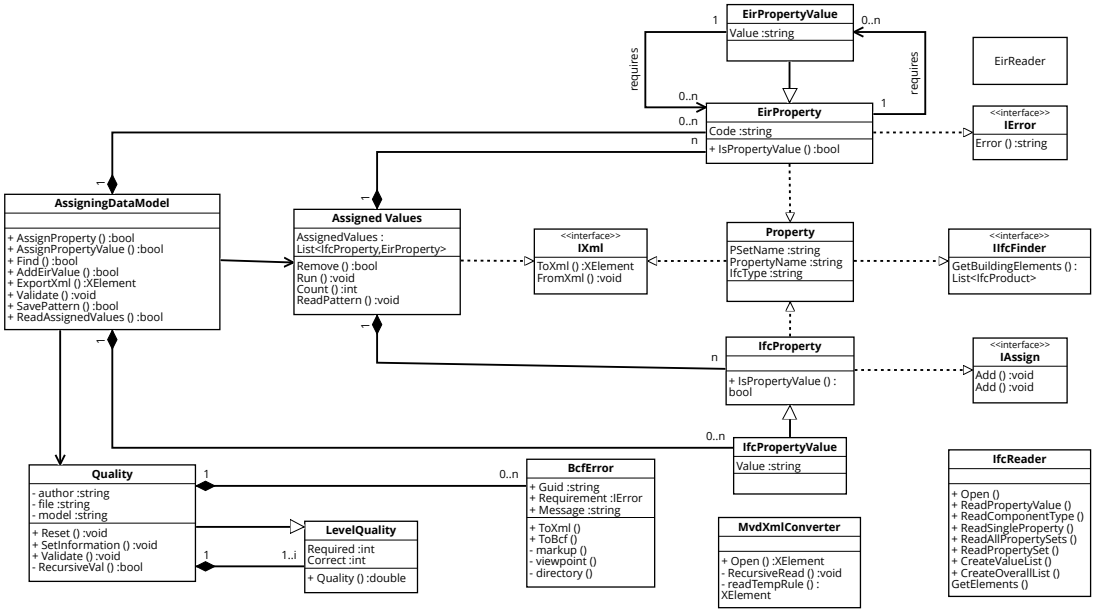
Remove-Requirements()	Removes *Requirements* from a *PropertyValue*. It does not delete any proeprties. However, the connected *PropertySet* also gets erased. Returns false if an error occurred.	**bool**
CreateProperty()	Creates a new *Property* and adds it to the data model. The property is not assigned with any proeprty set yet. Returns **false** if an error occurred.	**bool**
AddProperty()	Assigns a *Property* to a *PropertySet*.	**bool**
RemoveProperty()	Removes a *Property* from a *PropertySet*. It does not delete the property from the model.	**bool**
DeleteProperty()	Deletes a *Property* from the model. It also updates all *PropertySet*s that referenced this *Property*.	**bool**
AddPropertyValue()	Creates a *PropertyValue* and adds it to an existing *Property*	**bool**
Remove-PropertyValue()	Removes a *PropertyValue* from a *Property*.	**bool**
ToTreeStructure()	Recursive method that derives a tree structure from the model by using *XElement*s. Return the root of the tree.	*XElement*
Generate-TreeCode()	Creates the WBS code of an entity in the tree structure. Return the *XAttribute* defining the *Code*	*XAttribute*

The following listing illustrates how to use the methods to generate an IMIR data model that contains the information requirements from section 5.2.4.

```
IMIR.DataModel model = new DataModel();
//Create Requirements valid for all components
model.AddRequirement(null, "Pset1", "All building components must implement a componentType Property", out Requirements R1);
model.CreateProperty("Component Type", "The Component Type of the Property", 1, DataType.String, out Property P1);
model.AddProperty(P1, R1);
//Create ComponentTypes
model.AddPropertyValue<string>(P1, "Wall", 1, out PropertyValue<string> V1);
model.AddPropertyValue<string>(P1, "Beam", 2, out PropertyValue<string> V21);
model.AddPropertyValue<string>(P1, "OTHER", 3, out PropertyValue<string> V23);
//Add requirememts
model.AddRequirement(V1, "Pset2", "All walls require these properties", out Requirements R2);
model.AddRequirement(V21, "Pset3", "All beams require these properties", out Requirements R3);
//Create Properties
model.CreateProperty("Material", "Material of an Element", 2, DataType.String, out Property P1_2);
model.CreateProperty("Thickness", "Thickness of an Element", 1, DataType.String, out Property P1_3);
//Assign Properties
model.AddProperty(P1_2, R2);
model.AddProperty(P1_2, R3);
model.AddProperty(P1_3, R2);
//Add Material Values
model.AddPropertyValue<string>(P1_2, "Concrete", 1, out PropertyValue<string> V2);
model.AddPropertyValue<string>(P1_2, "Masonry", 2, out PropertyValue<string> V3);
//Create Requirements for material:Concrete
model.AddRequirement(V2, "Pset4", "Concrete requires these elements", out Requirements R4);
model.CreateProperty("Strength", "Strength of the material", 1, DataType.String, out Property P1_4);
model.AddProperty(P1_4, R4);
//Create XML file of IMIR and save it in a directory
System.Xml.Linq.XElement p = XmlCreator.ToXml(model);
XmlCreator.CreateXml(p, @"C:\InstanceTest.xml");
//Derive tree structure of the model
System.Xml.Linq.XElement tree = model.ToTreeStructure();
//Create mvdXML file from the tree and save it.
XElement neu = MvdXmlCreator.Create(tree);
using (XmlWriter xw = XmlWriter.Create(@"C:\InstanceTest.mvdxml",
            new XmlWriterSettings { Indent = true }))
{
    XDocument print = new XDocument();
    print.Add(neu);
    print.Save(xw);
    xw.Close();
}
```

Listing B.1: The following listing creates a IMIR data model and saves it as an XML file. Moreover, it derives a mvdXML file from it.

B.3 Enrich-IFC: UML Diagram



Appendix C

Case-Study: Floor Plans

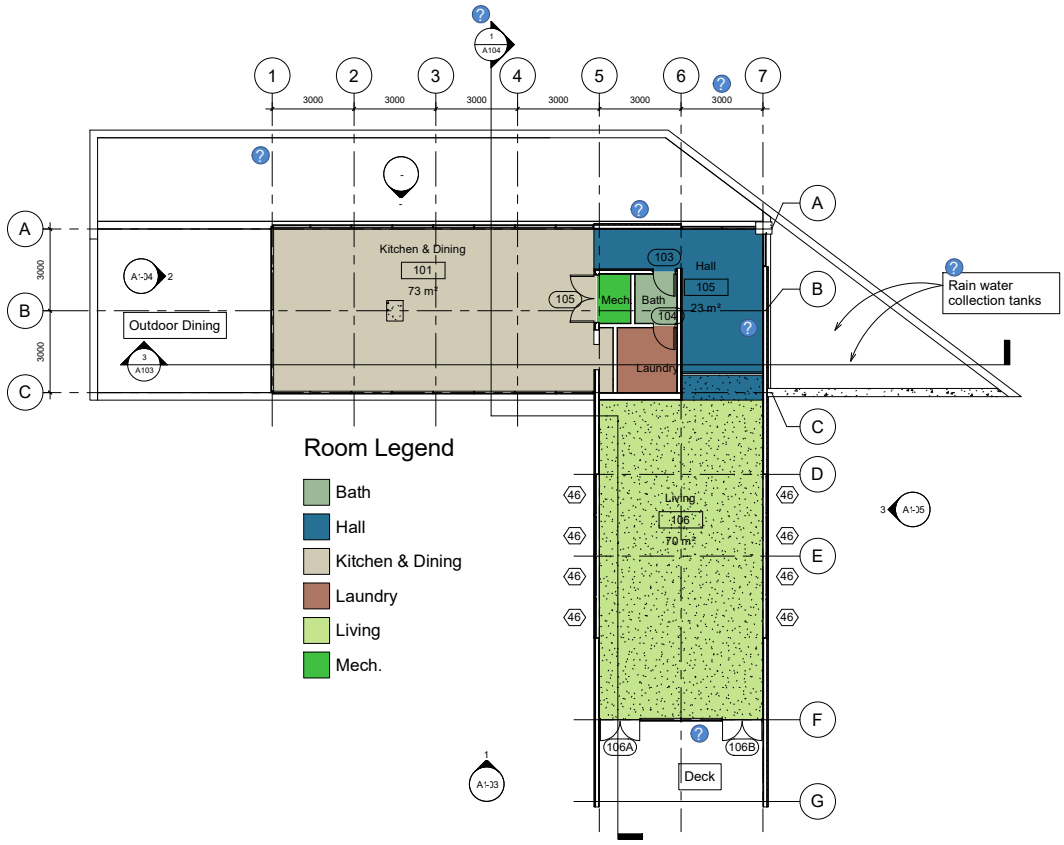


Figure C.1: Ground floor. (based on Autodesk (2020))

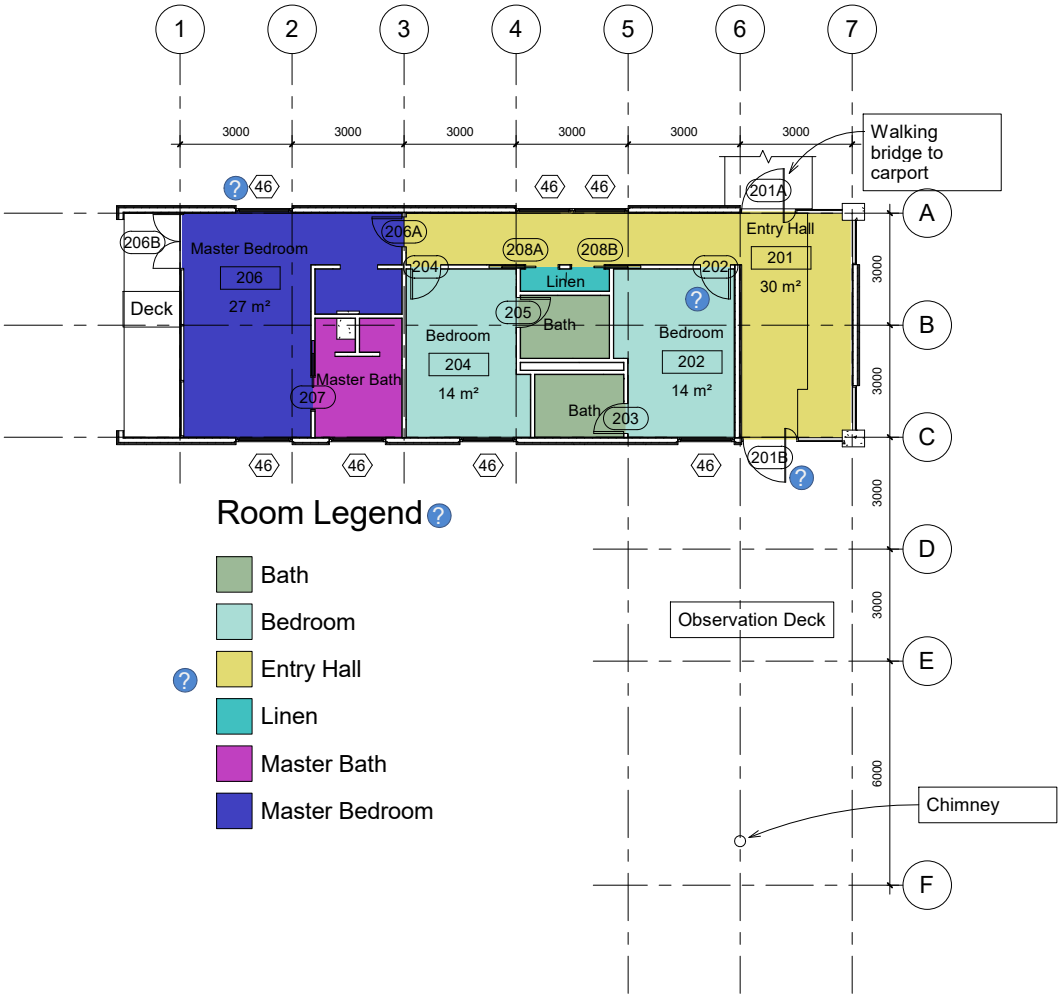


Figure C.2: First floor. (based on Autodesk (2020))

Appendix D

Digital Content

The following data is available on .

1. **ToolV2:** The source code of both data-models: IMIR and Enrich-IFC
2. *Prototype_GUI*: .exe file to run the prototype (shortcut to the Release)
3. **03_mvdXML**: Information related to Chapter 3:
   (a) **01:** Information related to chapter 3.1:
      - *Option1.mvdxml:* introducing strict hierarchy
      - *Option2.mvdxml:* introducing strict applicability
      - *PropertyMapping.txt:* adds information to the ifc derived from revit
      - *Test_Wall.ifc:* File that was used to compare option 1 and 2
      - *Test_wall.rvt:* Revit project comprising the two walls

   (b) **02:** Information related to chapter 3.2:
      - *4Components.ifc:* Ifc model comprising the components to test BimQ
      - *bimQ.mvdxml:* information requirements derived from BimQ
4. **05_ModelDesign**: Information related to figure 5.3
   - *InstanceText.xml:* Listing 5.1 shows a reduced version of this file
5. **06_GeigerExample** Information related to Chapter 6.3:
   - **BCF**: Derived BCF files
   - *GeigerTestModel.ifc*: The Ifc Document provided by Geiger before the supplement.
   - *GeigerModelExport.ifc*: The exported model after the enrichment.
   - *InstanceTest.mvdxml*: The mvdXML file comprising the information requirements from section 5.3.2
   - *assignedValues.xml*: XML file comprising all assigned value tuples
   - *SetUp3*.cs: Test generating the IMIR model

- *Screencast.mp4*: Video showing the prototype in Use; also available here: https://bit.ly/39wfW4N

6. **07_CaseStudy**: Information related to chapter 7:

- **BCF**: folder containing bcf files for the missing information
- *01_SetUp2.cs*: testClass to create the imir model
- *02_IMIR.mvdXML*: mvdXML derived from the IMIR model
- *03_IMIR.xml*: IMIR model exported to an XML file
- *04_externalModel.ifc*: external IFC file
- *05_assignedVaues.xml*: file storing all assigned value paris
- *06_enrichedModel.ifc*: model ernrichment
- *07_InUse.mp4*: video showing the prototype
- *08_RevitProject.rvt*:

Bibliography

Abualdenien, J. & Borrmann, A. (2019). A meta-model approach for formal specification and consistent management of multi-LOD building models. *Advanced Engineering Informatics*. Retrieved: https://www.sciencedirect.com/science/article/abs/pii/S1474034618304324. Accessed: 30/03/2020.

Aram, S., Eastman, C. M., Sacks, R., Panushev, I. & Venugopal, M. (2010). Introducing a new methodology to develop the Information Delivery Manual for AEC projects. *Proceedings of the CIB-W078 2010: 27th International Conference - Cairo, Egypt, 16-18 November*. Retrieved: https://www.irbnet.de/daten/iconda/CIB21885.pdf. Accessed: 30/03/2020.

BIMForum (2019). Level of Development (LOD) Specification Part I & Commentary: For Building Information Model and Data. Retrieved: https://bimforum.org/lod/. Accessed: 13/02/2020.

Borrmann, A., Beetz, J., Koch, C., Liebich, T. & Muhic, S. (2018). Industry Foundation Classes: A Standardized Data Model for the Vendor-Neutral Exchange of Digital Building Models. In: A. Borrmann, M. König, C. Koch, & J. Beetz (Hrsg.), *Building Information Modeling*, Volume 17, S. 81–126. Cham: Springer International Publishing.

Borrmann, A., König, M., Koch, C. & Beetz, J. (Hrsg.) (2015a). *Building Information Modeling*. Wiesbaden: Springer Fachmedien Wiesbaden.

Borrmann, A., König, M., Koch, C. & Beetz, J. (2015b). Einführung. In: A. Borrmann, M. König, C. Koch, & J. Beetz (Hrsg.), *Building Information Modeling*, Volume 1, S. 1–21. Wiesbaden: Springer Fachmedien Wiesbaden.

Borrmann, A., König, M., Koch, C. & Beetz, J. (2018). Building Information Modeling: Why? What? How? In: A. Borrmann, M. König, C. Koch, & J. Beetz (Hrsg.), *Building Information Modeling*, Volume 145, S. 1–24. Cham: Springer International Publishing.

buildingSMART (2005). IFC2x Edition3. Retrieved: https://standards.buildingsmart.org/IFC/RELEASE/IFC2x3/FINAL/HTML/. Accessed: 30/03/2020.

buildingSMART (2019a). BCF-XML: File-Based Implementation of BIM Collaboration Format. https://github.com/buildingSMART/BCF-XML.

buildingSMART (2019b). BIM Collaboration Format (BCF) - An Introduction. Retrieved: https://technical.buildingsmart.org/standards/bcf/. Accessed: 30/03/2020.

buildingSMART (2019c). Industry Foundation Classes: Version 4.2 bSI Darst Standard: IFC Bridge Proposed Extension. Retrieved: https://standards.buildingsmart.org/IFC/DEV/IFC4_2/FINAL/HTML/. Accessed: 30/03/2020.

buildingSMART (2019d). Model View Definition (MVD). Retrieved: https://technical.buildingsmart.org/standards/mvd/. Accessed: 30/03/2020.

buildingSMART (2020a). bSDD Content Guidelines. Retrieved: https://docs.google.com/document/d/1YUiR07A27lK0UB8ImYoaoLKCUvh1QFG1FfcvvLOYdP0/edit. Accessed: 30/03/2020.

buildingSMART (2020b). buildingSMART Data Dictionary. Retrieved: http://bsdd.buildingsmart.org/. Accessed: 30/03/2020.

Chipman, M., Liebich, T. & Weise, M. (2016). mvdXML: Specification of a standardized format to define and exchange Model View Definitions with Exchange Requirements and Validation Rules. Retrieved: https://standards.buildingsmart.org/MVD/RELEASE/mvdXML/v1-1/mvdXML_V1-1-Final.pdf. Accessed: 30/03/2020.

CIC (2011). BIM Project Execution Planning Guide Version 2.1.

CIC (2019). Cost Estimation.

Die Deutsche Bauindustrie (2018). BIM im Hochbau: Technisches Positionspapier der Arbeitsgruppe Hochbau im Arbeitskreis Digitalisiertes Bauen im Hauptverband der Deutschen Bauindustrie e.V. Retrieved: https://www.bauindustrie.de/presse/presseinformationen/positionspapier-bim-im-hochbau/. Accessed: 30/03/2020.

Dossick, C. S. & Neff, G. (2011). Messy talk and clean technology: communication, problem-solving and collaboration using Building Information Modelling. *Engineering Project Organization Journal* 1(2), S. 83–93. Retrieved: https://www.tandfonline.com/doi/citedby/10.1080/21573727.2011.569929?scroll=top&needAccess=true. Accessed: 30/03/2020.

Eastman, C., Panushev, I., Sacks, R., Venugopal, M., Aram, S., See, R. & Yagmur, E. (2011). A Guide for Development and Preparation of a National BIM Exchange Standard. Retrieved: http://dcom.arch.gatech.edu/pcibim/documents/IDM-MVD_Development_Guide_v4.pdf. Accessed: 30/03/2020.

Hölzlwimmer, V. (2019). Prüfung von Fertigstellungsgraden in digitalen Gebäudemodellen. Diplomarbeit, Technische Universität München. Retrieved: https://publications.cms.bgu.tum.de/theses/2019_Hoelzlwimmer_LOD.pdf. Accessed: 30/03/2020.

International Organization for Standardization (2018). Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries – Part 1: Data schema.

Jönsson, E. (2015). Consequences of Implementing the buildingSMART Data Dictionary. Retrieved: http://www.diva-portal.org/smash/get/diva2:839483/FULLTEXT01.pdf. Accessed: 30/03/2020.

Kahate, A. (2009). *XML & related technologies*. Delhi: Dorling Kindersley (India).

Laasko, M. & Kiviniemi, A. (2012). The IFC Standard - A Review Of History, Development, And Standardization. *Journal of Information Technology in Construction* 17, S. 134–161. Retrieved: https://www.researchgate.net/publication/252069448_The_IFC_Standard_-_A_Review_of_History_Development_and_Standardization. Accessed: 30/03/2020.

Lewis, J. P. (2007). *Fundamentals of project management* (3rd ed. Aufl.). WorkSmart. New York: American Management Association.

Liskov, B. H. & Wing, J. M. (1994, 11). A Behavioral Notion of Subtyping. *ACM Transactions on Programming Languages and Systems* 16. Retrieved: https://www.cs.cmu.edu/~wing/publications/LiskovWing94.pdf. Accessed:30/03/2020.

Lockley, S., Benghi, C. & Černý, M. (2017). Xbim.Essentials: A Library for Interoperable Building Information Applications.

Martin, R. C. (1995). The Dependency Inversion Principle. *The C++ Report: Engineering Notebook* 3. Retrieved: https://web.archive.org/web/20110714224327/http://www.objectmentor.com/resources/articles/dip.pdf. Accessed: 30/03/2020.

Martin, R. C. (1996). The Interface Segregation Principle. *The C++ Report: Engineering Notebook* 4. Retrieved: https://drive.google.com/file/d/0BwhCYaYDn8EgOTViYjJhYzMtMzYxMC00MzFjLWJjMzYtOGJiMDc5N2JkYmJi/view. Accessed: 30/03/2020.

Martin, R. C. (2002). *Agile Software Development, Principles, Patterns, and Practices*. London: Pearson.

Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. New Jersey: Prentice Hall.

McPartland, R. (2014). BIM Levels explained. *The National BIM Standard*. Retrieved: https://www.thenbs.com/knowledge/bim-levels-explained. Accessed: 30/03/2020.

Microsoft (2015). Collections (C#). Retrieved: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/collections#BKMK_KeyValuePairs. Accessed: 30/03/2020.

Preidel, C., Borrmann, A., Mattern, H., König, M. & Schapke, S.-E. (2018). Common Data Environment. In: A. Borrmann, M. König, C. Koch, & J. Beetz (Hrsg.), *Building Information Modeling*, S. 279–291. Cham: Springer International Publishing.

Project Management Institut (2012). *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)-Fifth Edition (ENGLISH)* (5th ed. Aufl.). Newtown Square, PA: Project Management Institute.

Scheffer, M., Mattern, H. & König, M. (2018). BIM Project Management. In: A. Borrmann, M. König, C. Koch, & J. Beetz (Hrsg.), *Building Information Modeling*, S. 235–249. Cham: Springer International Publishing.

Shafiq, M., Matthews, J. & Lockley, S. (2012, 10). Requirements for Model Server Enabled Collaborating on Building Information Models. *International Journal of 3-D Information Modeling* 1, S. 8–17. Retrieved: https://www.researchgate.net/publication/262215057_Requirements_for_Model_Server_Enabled_Collaborating_on_Building_Information_Models. Accessed: 30/03/2020.

Speiser, K. (2019). The Information Delivery Manual and Model View Definitions.

AEC3 (2020a). FUNCTIONALITY. Retrieved: https://bim-plattform.com/en/bimq/functionality/. Accessed: 30/03/2020.

AEC3 (2020b). Mapping of elements (IFC). Retrieved: https://bimq.centraldesk.com/en/articles/JI1y-mapping-of-elements-ifc-. Accessed: 30/03/2020.

AEC3 (2020c). Model Components. Retrieved: https://bimq.centraldesk.com/en/categories/uT5X-model-components. Accessed: 30/03/2020.

AEC3 (2020d). Project Components. Retrieved: https://bimq.centraldesk.com/en/categories/bxGC-project-components. Accessed: 30/03/2020.

Autodesk (2020). Revit Sample Project Files. Retrieved: https://knowledge.autodesk.com/support/revit-products/getting-started/caas/CloudHelp/cloudhelp/2019/ENU/Revit-GetStarted/files/GUID-61EF2F22-3A1F-4317-B925-1E85F138BE88-htm.html. Accessed: 30/03/2020.

BCA Singapore (2013). Singapore BIM guide - Version 2. Retrieved: https://www.corenet.gov.sg/media/586132/Singapore-BIM-Guide_V2.pdf. Accessed: 30/03/2020.

Digital Konvergens, BIM7AA & BIM i Landskabet (2019). Specification of Building Parts - for selected building parts in building models. Retrieved: http://www.bim7aa.dk/DIKON_BIM7AA_Bygningsdelsspecifikationer_UK.html. Accessed: 30/03/2020.

ISO 19650-1 (2017). Organization and digitalization of information about buildings and civil engeneering works, including building information modelling (BIM) - Information management using building information modelling - Part 1: Concepts and principles.

ISO 19650-2 (2017). Organization and digitalization of information about buildings and civil engeneering works, including building information modelling (BIM) - Information management using building information modelling - Part 2: Delivery phase of assets.

ISO 29481-1 (2016). Buidling information models - information delivery manual - Part 1: Methodology and format.

PAS 1992-2 (2013). Specification for information management for the capital/delivery phase of construction projects using building information modeling.

The NBS (2015). Uniclass 2015. Retrieved: https://www.thenbs.com/our-tools/uniclass-2015. Accessed: 30/03/2020.

UK BIM Alliance (2019). Information Management according to BS EN ISO 19650 - Guidance Part 1: Concepts. Retrieved: https://www.ukbimalliance.org/wp-content/uploads/2019/04/Information-Management-according-to-BS-EN-ISO-19650_-Guidance-Part-1_Concepts_2ndEdition.pdf. Accessed: 30/03/2020.

Venugopal, M., Eastman, C. M., Sacks, R. & Teizer, J. (2012). Semantics of model views for information exchanges using the industry foundation class schema. *Advanced Engineering Informatics* 26(2), S. 411–428. Retrieved: https://www.sciencedirect.com/science/article/abs/pii/S1474034612000067. Accessed: 30/03/2020.

Venugopal, M., Eastman, C. M. & Teizer, J. (2015). An ontology-based analysis of the industry foundation class schema for building information model exchanges. *Advanced Engineering Informatics* 29(4), S. 940–957. Retrieved: https://www.sciencedirect.com/science/article/abs/pii/S1474034615001019. Accessed: 30/03/2020.

Wix, J. & Karlshøj, J. (2010). Information Delivery Manual: Guide to Components and Development Methods. Retrieved: https://standards.buildingsmart.org/documents/IDM/IDM_guide-CompsAndDevMethods-IDMC_004-v1_2.pdf. Accessed: 30/03/2020.

Zhang, C., Beetz, J. & Vries, d. (2013, 07). Towards model view definition on semantic level : a state of the art review. *Proceedings of European Group for Intelligent Computing in Engineering (EG-ICE)*. Retrieved:https://www.researchgate.net/publication/260763029_Towards_model_view_definition_on_semantic_level_a_state_of_the_art_review. Accessed: 30/03/2020.

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Master-Thesis selbstständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Ich versichere außerdem, dass die vorliegende Arbeit noch nicht einem anderen Prüfungsverfahren zugrunde gelegen hat.

Sulzberg, April 1, 2020

Kilian Speiser

Kilian Speiser
[redacted]