Lehrstuhl für Computergestützte Modellierung und Simulation
Ingenieurfakultät Bau Geo Umwelt
Technische Universität München

Validierung von BIM-Modellen auf Basis graphenbasierter Repräsentation

Wissenschaftliche Arbeit zur Erlangung des Grades

Bachelor of Science (B.Sc.)

an der Ingenieurfakultät Bau Geo Umwelt der
Technischen Universität München.

Betreut von Prof. Dr.-Ing. André Borrmann
Sebastian Esser
Lehrstuhl für Computergestützte Modellierung und Simulation

Eingereicht von Ayu Justina Giehl ([redacted])
[redacted]

Eingereicht am 01. September 2022

Vorwort

Mein persönliches Interesse für die Modellierung mit Building Information Modeling führte zu der Gelegenheit meine Bachelorarbeit am Lehrstuhl für Computergestützte Modellierung und Simulation zu verfassen

Diese Arbeit wäre ohne der besonderen Unterstützung meines Betreuers Sebastain Essers nicht möglich gewesen. Er hat mir einen ersten Einblick in die 3D-Modellierung gegeben und hat darüber hinaus mit seiner Fähigkeit, komplexe Zusammenhänge verständlich und logisch darzustellen, die Arbeit maßgeblich beeinflusst.

Abschließend möchte ich mich an dieser Stelle bei meiner Familie und bei meinem Freund für die Unterstützung bedanken.

Abstract

Computational building modeling via BIM is gaining increasing popularity and importance in the AEC industry. These data intensive models, with their properties and relations, can digitally be modeled into a data structure, which allows for simple data transfer. For an effective workflow making use of such data structures tests validating the authenticity of these data structures are necessary.
The goal of this thesis is to explore the potential of graph-based representations of the BIM data structures and to enable user-friendly alternatives to validate such data structures. For this we make use of the validator BIMTester, which approaches are applied to graphs using pattern search.Here, the state of the art is explained by presenting the existing alternatives for describing BIM models. An overview of existing scientific and technical techniques for the validation of building data structures will be given. A comparison of validation tools in the next subsection considers the suitability of the BIMTester as a basis for the audit on graphs. Additionally, the tests of BIMTester will be translated into patters using the Cypher language, which will subsequently be applied to the graph data structures. The proposed will be explored in a case study where the validation tests of BIMTester and Solibri Model Checkers will be compared in their technical abilities, using various criteria.
The results of the case study indicate that the implementation of the validation tests of BimTester is applicable to property graphs and complex higher-order validation procedures. Despite this however, for sufficiently complex tasks, several complications arise, which require a more detailed search pattern.

Zusammenfassung

Die digitale Gebäudemodellierung unter Verwendung von BIM gewinnt in der AEC-Industrie zunehmend an Bedeutung. Die datenintensiven Modelle mit ihren Informationen und Relationen können dabei digital als Datenmodelle abgebildet werden, die gleichzeitig als Medium von Datenaustauschprozessen verwendet werden können. Für einen effektiven Arbeitsablauf unter Verwendung solcher Datenmodelle, ist dabei die Gewährleitung der Modellqualität durch passende Validierungswerkzeuge unerlässlich.
Ziel dieser Arbeit ist das Potenzial der Validierung auf graphenbasierte Repräsentationen zu untersuchen und somit eine benutzerfreundliche Alternative zu den bereits vorhandenen Software-Appliaktionen zur Überprüfung von Datenstrukturen bereitzustellen. Dafür werden die auf unit-testing basierten Ansätze des Validierungswerkzeuges BIMTester auf die Graphen mittels Musterabgleich (pattern search) angewendet.
Dazu wird ein Einblick über vorhandene wissenschaftliche und technische Ansätze zur Überprüfung von Gebäudedaten gegeben. Hier wird der Stand der Wissenschaft erläutert, indem die vorhandenen Alternativen zur Beschreibung von BIM-Modellen und anschließend die Methodik der Modellvalidierung unterschiedlicher Software-Anwendungen vorgestellt werden. In einem Vergleich der Validierungstools wird erwägt, inwiefern der BIMTester als Grundlage für die Prüfungen auf Graphen geeignet ist. Anschließend werden die Regelprüfungen des BIMTesters durch die Abfragesprache Cypher zu Mustern übersetzt, die auf den Graphen gesucht werden sollen. Der vorgestellte Ansatz wird in einer Case Study mit den Validierungsansätzen des BIMTesters und des Solibri Model Checkers nach veschiedenen Kriterien in ihrer technischen Fähigkeit verglichen und bewertet.
Die Ergebnisse der Case Study zeigen, dass die Regel-Implementierung des BIMTesters auf Property Graphen möglich ist und darauf aufbauend eigenständige Prüfungen ausgeführt werden können. Dennoch ergeben sich einige Schwierigkeiten und Komplikationen hinsichtlich komplexerer Abfragen, die eine hohe Detailliertheit des Musters erfordern.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Abkürzungen

BDD	Behavior-Driven Development
BIM	Building Information Modeling
bSI	buildingSMART International
GML	Geography Markup Language
IFC	Industry Foundation Classes
OGC	Open Geospatial Consortium
RDF	Resource Description Framework
SHACL	Shapes Constraint Language
SMC	Solibri Model Checker
XML	Extensible Markup Language

Kapitel 1

Einführung

1.1 Motivation und Ziel der Arbeit

Die Gebäudemodellierung mit BIM als datenintensive Methode zur Beschreibung von Bauwerken gewinnt in der AEC-Industrie zunehmend an Bedeutung (»NBS«, 2020). Die umfangreichen Modelle mit ihren Informationen und Relationen können digital als Datenmodell abgespeichert werden. Basierend auf solche Datenmodelle werden derzeit zahlreiche automatisierte Qualitätsprüfungen durchgeführt (SIRTL, 2020)

Angesichts der BIM-Implementierung in Projekten mit steigender Größe und Komplexität ist die Sicherstellung der Qualität des BIM-Modells unerlässlich. Die Modellqualität kann großen Einfluss auf die Interoperabilität und die Effizienz des Arbeitsverlaufs durch Inkonsistenz oder fehlerhafte Implementierungen bei der Modellverwendung nehmen (BAUMGÄRTEL & PIRNBAUM, 2016). Neben der technischen Umsetzung und der Fähigkeit der Validierung, spielt die Benutzerfreundlichkeit der Validierungswerkzeuge zunehmend eine große Rolle. BIM selbst ist in seiner Bedienung vielfältig und benötigt somit viel Engagement und Zeit (»NBS«, 2020). Nach Angaben von »NBS« (2020) ist der Mangel an in-house Expertise und Training einer der Haupthürde der Einführung von BIM im Arbeitsplatz. Insbesondere im Bausektor, in dem die Zusammenarbeit von technischen und Nicht-technischen Projektbeteiligten regelmäßig vorkommt, ist es umso wichtiger, dass Projektbeteiligte ohne tiefes Wissen über Datenmodelle in der Lage sind, die Validierungstools zu bedienen. Eine benutzerfreundliche und vielversperchende Alternative zur Überprüfung von komplexeren Sachverhalten ist dabei der BIMTester. BIMTester basiert sich auf das Unit-testing und überprüft somit Datensätze in der frühen Planungsphase des Modells. Dabei wird ein großer Wert auf eine Regelformulierung durch eine einfache und verständliche Sprache gesetzt.

Die für die Überprüfung benötigten Informationen sind in Datenmodelle wie in IFC abgespeichert, die gewöhnlicherweise strikte Definition der Datenmodelle durch z.B. Datenmodellschemata aufweisen. Dadurch können Einschränkungen im Zuge der Informationsextraktion von komplexeren Datennetzstrukturen ergeben, die zu Validierungslücken führen oder für die Bedienung der Validierungssoftware ein tiefes Wissen über der Software-Anwendung oder über das Datenmodell voraussetzen (ISMAIL, NAHAR & SCHERER, 2017).

Aus diesem Grund wird in der Arbeit versucht, die Validierung auf Graphen zu ermöglichen. Graph-Modelle stellen eine Methodik dar, die Informationen und Relationen effizient durch Knoten und Kanten abbilden können. Graphen bzw- Graph-Datenbanken eigenen sich als

eine effiziente und einfache Alternative zu der Darstellung, Speicherung und Verwaltung von komplexen Informationszusammenhänge.

Ziel dieser Arbeit ist es, die Prüfregeln des BIMTesters auf Property Graphen anzuwenden, sodass das Potenzial des Property Graphens in Hinblick auf die Validierung komplexerer Datennetzstrukturen getestet wird.

1.2 Aufbau der Arbeit

Zur Erläuterung des theoretischen Hintergrundes der Arbeit wird in Kapitel 2 zunächst grundlegende Möglichkeiten zur Datenspeicherung von Modellinhalten vorgestellt. Hier werden objektorientierte Modellinhalte in ihrer Struktur und Modellierungsart vorgestellt. Anschließend wird ein Überblick über bereits vorhandene Software-Applikationen zur automatisierten Überprüfung vorgestellt und daraus die technischen Vor-, und Nachteile des Validierungswerkzeugs BIMTester dargestellt. Im weiteren Abschnitt wird das Konzept der Modellierung mit graphenbasierter Repräsentationen erläutert.

Kapitel 3 erklärt die Methodik, mit der die objektorientierten Daten in ein Graph-Modell transformiert werden. Die technische Umsetzung der BIMTester-Ansätze auf den Property Graphen wird genauer erläutert.

Im folgenden Kapitel 4 wird der in dieser Arbeit vorgstellte Ansatz über seine technische Praktikabilität und Funktionalität innerhalb einer Case Study dargestellt. Dabei wird der neue Ansatz mit den Validierungswerkzeugen Solibri Model Chekcer und BIMTester über ausgewählte Anwendungsfälle miteinander verglichen und bewertet.

Im letzten Kapitel 5 wird eine Diskussion von den vorherigen Kapitel abgeleitet und anschließend ein Ausblick gegeben.

Kapitel 2

Theoretischer Hintergrund

2.1 Darstellungsmöglichkeiten von Modellinhalten

Die Gebäudemodellierung mit Building Information Modeling (BIM) setzt sich zunehmend als eine effiziente Methode zur Förderung der Interoperabilität zwischen Planern und den Projektbeteiligten durch. Um die datenintensiven Modelle effizient nach Anwendungsszenarien zu erfassen und einen reibungslosen Datenaustausch der verschiedenen Fachdisziplinen zu erzielen, spielt die Verwendung eines offenen Standards eine wichtige Rolle (NAWARI, 2012).

Durch die universelle Spezifizierung kann somit sehr viel Aufwand und Zeit erspart werden, indem keinerlei direkte Übersetzer für die Softwareanwendung benötigt wird, die den von den anderen Beteiligten erfassten Informationsgehalt des BIM-Modells in ein für sich passendes Anwendungsformat transformiert (LAAKSO, M & KIVINIEMI, 2012).

Derzeit integrieren gebräuchliche Datenmodelle das Prinzip der objektorientierten Programmierung, indem Klassen als Objekttyp die Art und Weise der Instanziierung von Objekten beschreiben, die wiederum durch Attribute und Methoden definiert werden. Durch Assoziationen bzw. Aggregationen werden Klassen in Beziehung gebracht. In objektorientierten Modellinhalten werden zudem das Prinzip der Vererbung umgesetzt, die durch Assoziationen beschrieben werden können (BORRMANN, BEETZ, KOCH & LIEBICH, 2015; ESSER, 2021).

Neben dem Industry Foundation Classes (IFC)-Standard, das sich in den vergangenen Jahren als beliebtes Standard etabliert hat, wurde vom Open Geospatial Consortium (OGC) weitere BIM bzw. GIS-Standards, wie CityGML (zur Beschreibung drei-dimensionaler Stadtmodelle), etc. entwickelt (DAVILA DELGADO & OYEDELE, 2020). Einen weiteren Standard wurde von LandXML.org entwickelt, welches sich hauptsächlich mit der Modellierung von Tiefbau- und Vermessungsobjekte beschäftigt. Nach einem fehlgeschlagenen Versuch LandXML als ein OGC-Standard in 2013 aufzunehmen, wurde im Rahmen von OGC entschlossen stattdessen einen eigenständigen Standard zu entwickeln, der die Kriterien in dem Themengebiet des OGC erfüllt (BEETZ, AMANN & BORRMANN, o.D.).

Im Folgenden werden als repräsentative Standard-Datenmodelle IFC, CityGML und LandXML betrachtet.

2.1.1 IFC Standard

Der IFC-Standard ist eine herstellerneutrale Spezifzierung zur Beschreibung von Bauwerken in der - Industrie, die international ihre Verwendung findet (*ISO 16739*, 2021). Mit der Einführung des IFC-Standards im Jahr 1994 wurde von buildingSMART International (bSI), ehemalige International Alliance for Interoperability (IAI), eine einheitliche Grundlage für die elektrische Kommunikation zwischen verschiedener Softwareprodukten zum Datenaustausch insbesondere hochbauspezifischer Informationen (Geometrie, Semantik, etc.) erschaffen (LAAKSO et al., 2012).

Basierend auf dem EXPRESS-Schema wird das IFC-Datenmodell durch hierarchisch aufgebauten Entitäten mit Informationen über Vererbung zusammengesetzt. Die Entitäts-Attribute eines Objekts im IFC-Modell sind als Teil des IFC-Standardcodes von buildingSMART International festgelegt (ISMAIL et al., 2017). Das IFC-Modell kann als XML-Datei (nach ISO 10303-21) oder als STEP Physical File (SPF) (nach ISO 10303-28) gespeichert werden (BUILDINGSMART, o.D. a).

EXPRESS-Sprache

Die deklarative Datenmodellierungssprache EXPRESS basiert sich auf den STEP-Standard Teil 11 und definiert im Allgemeinen das Produktdatenschema des Modells, d.h. das EXPRESS Schema legt die Art unf Weise der Beschreibung eines Datenmodells fest (LEE, 2009) . Es werden daher keine Instanzen der jeweiligen Klassenobjekte beschrieben. EXPRESS beschreibt das Datenmodell in objektorientierten Strukturen, sodass jegliche Abstraktionen von Objekten innerhalb einer Welt (real oder imaginär) als Klassen (bzw. Entitäten) dargestellt werden, die Attribute, sowie Beziehungen zu anderen Klassen aufweisen können. Die Assoziation zwischen zwei Objekten wird mit dem Erhalten eines Attributes vom referenzierten Objekt definiert. D.h. erst durch die Vergabe eines Attributs von einem Objekt an einem anderen Objekt, kann eine Beziehung entstehen. Analog zu den objektorientierten Prinzipien unterliegt das Schema den Vererbungsmerkmalen, sodass Objektattribute und Beziehungen an Subklassen/-typen weitervererbt werden (BORRMANN et al., 2015).

BORRMANN et al. (2015) beschreibt zwei Besonderheiten des EXPRESS-Standards wie folgend:
1. Deklaration der inversen Beziehung. Durch die Einführung der inversen Beziehung zum Konzept der objektorientierten Theorie ist es möglich zusätzlich zu einer bereits bestehenden Beziehung eine zweite Relation zur Beziehung entgegengesetzter Richtung zu modellieren, ohne neue Informationen hinzuzufügen.
2. Einbau von Aggregations-Datentypen.

Im Algorithmus 2.1 wird als Beispiel ein EXPRESS-Schema dargestellt, das die Entitäten und die Relationen einer Familie beschreibt. Im Schema wird eine Beziehung zwischen der abstrakten Superklasse `Person` und dessen Subklasse `Female` definiert. Die Entität

Person besitzt das Attribut `mother`, wobei die Entität `Female` das inverse Attribut zu `mother`, `motherof` erhält.

```
SCHEMA schema_UT_family_1;
TYPE Name = STRING;
END_TYPE;

ENTITY Person
   ABSTRACT SUPERTYPE OF (ONEOF (Female));
     name : Name;
     mother : OPTIONAL Female;
END_ENTITY;

ENTITY Female
   SUBTYPE OF (Person);
   INVERSE
     motherof : SET [0:?] OF Person FOR mother;
END_ENTITY;
```

Algorithmus 2.1: Definition der Entitytypen mit EXPRESS

2.1.2 Grundstruktur des IFC-Modells

Das IFC-Datenmodell wird in vier konzeptionellen Layers (Schichten) eingeteilt (Abbildung 5). Diese werden durch den Domain Layer, Interop Layer, Core Layer und Resource Layer gebildet. Elemente der oben liegenden Layers können auf die der unteren Layers referenzieren, umgekehrt ist dies jedoch nicht möglich. Der Core Layer stellt den Kern des IFC-Datenmodells dar und gibt das Grundkonzept an, das in den unterliegenden Layers erweitert werden kann. Zum Core Layer gehören die superabtrakten Klassen (Basisklassen) `IfcRoot`, `IfcObject`, `IfcActor`, `IfcProcess`, `IfcProduct`. Aufbauend vom Core Layer werden die Erweiterungsmodule Control Extension, Process Extension und Product Extension definiert. Als Teil des Datenmodells werden in Product Extension physische und räumlichen Objekte (wie Wände, Türen, Räume, Stockwerke), sowie ihre Beziehungen eines Gebäudes durch die Ifc-Entitäten `IfcElement`, `IfcBuildingStorey`, `IfcSpatialElement` und `IfcBuilding` beschrieben. Im Shared Layer werden Oberklassen, wie `IfcWall`, `IfcColumn`, etc. von einem bzw. mehreren Objekten spezifiziert (BORRMANN et al., 2015; *ISO 16739*, 2021).

Im Domain Layer werden acht domainspezifische Schemata konzipiert. Jedes einzelne Schema in dieser Schicht spezifiziert Klassen, die ausschließlich der jeweiligen Domäne eingeführt werden. Die für den Hochbau wichtigen Architecture Domain definiert hauptsächlich die Gebäude-spezifischen Merkmale (BORRMANN et al., 2015).

Alle Klassen im Resource Layer können unabhängig der Vererbungshierarchien über das gesamte Datenmodell verwendet werden (*ISO 16739*, 2021). Die Klassen in diesem Layer unterliegen keiner Vererbungsprinzipien des IfcRoots, sodass diese erst von Subklassen des IfcRoots (bspw. `IfcRelationship`) in Assoziation gebracht werden müssen, um als Objekt im Datenmodell zu existieren. Beispielsweise sind für die Geometriebeschreibung

folgender Teil des IFC von großer Bedeutung: Geometry Resource, Topolgy Resource, Geometric Model Resource, etc. (BORRMANN et al., 2015; *ISO 16739*, 2021).

Objektbeziehungen

Innerhalb des IFC-Modells gilt zur Beschreibung von Beziehungen das Konzept der sogenannten objektifizierten Beziehung. Beziehungen zwischen zwei Objektelementen werden mit einer Aggregationsklasse - einem Objekt, welches die Beziehung an sich darstellt – beschrieben (BORRMANN et al., 2015).

Bei der Beschreibung von Beziehungen ist die Richtung der Beziehung wichtig. Im IFC-Datenmodell wird dafür das Prinzip der Vorwärtsbeziehungen definiert (BORRMANN et al., 2015). So werden die zu referenzierenden Objekte ausgehend vom Beziehungsobjekt mit einem Beziehungsattribut in Beziehung gebracht. Beziehungsobjekte können als Synonym der Aggregationsklassen in der objektorientierten Theorie verwendet werden und können durch Instanzen der Subklassen von `IfcRelationship` dargestellt werden. D.h. alle Beziehungsklassen erben von der abstrakten Superklasse `IfcRelationship`, welche die objektifizierten Beziehungen abbildet. Bspw. wird die Materialzugehörigkeit eines Bauteilelements mit einem Material `IfcMaterial` durch das Beziehungsobjekt `IfcRelAssociatesMaterial` aufgestellt. `IfcRelAssociatesMaterial` ist dabei eine Subklasse der `IfcRelAssociates`, welche externen Informationen, wie in diesem Fall die Materialzugehörigkeit, mit einem Objekt assoziiert. Mit der Aggregationsklasse werden Objekte mit ihren Eigenschaften (Properties) unter Einbindung des Property Sets (`IfcPropertySet`) referenziert (BORRMANN et al., 2015; *ISO 16739*, 2021)

Geometrie

Im IFC-Modell können Objekte eine oder mehrere geometrische Repräsentationen aufzeigen. Solche Objekte können in Einzelteile dekomponiert werden, die sogenannte Items bilden (BOTH, 2009). In IFC werden Items durch die abstrakte Klasse `IfcRepresentationItem` definiert, die eine abstrakte Superklasse von allen Geometrieklassen darstellt (BORRMANN et al., 2015).

Bei der geometrischen Modellierung ist dabei die semantische Bedeutung mit der geometrischen strikt zu trennen. Dies dient zum Vorteil, dass semantische Objekte mehrere geometrische Repräsentaionen zugewiesen werden können, sodass die Flexibilität in verschiedenen Anwendungsfällen gewährleistet ist (BORRMANN et al., 2015). Für die semantische Unterscheidung verrschiedener Darstellungstypen, unter Miteinbeziehung der Darstellungsansicht und dem Zielmaßstab, kann `IfcGeometricRepresentationSubContext` verwendet werden. Abhängig vom geometrischen Darstellungskontext kann durch die Modifizierung des `IfcGeometricRepresentationSubContext` die Detaillierung der Formdarstellung eines Objektes kontrolliert werden (BUILDINGSMART, o.D. b).

Unter Verwendung der Subklassen von `IfcGeometricRepresentation` können für 2D-Darstellungen Curve2D oder Annotation2D, für 3D-Flächenmodelle das SurfaceModel und für 3D-Volumenkörper mit SweptSolid, Brep, CSG, Clipping, BoundungBox modelliert werden (BOTH, 2009).

Das Begrenzungsflächenmodell (Boundary Representation, kurz: Brep) ist die mächtigste Methode zur Beschreibung von 3D-Modellen. Die Modellierung mit Brep wird durch die Beschreibung seiner begrenzenden Oberflächen definiert und stellt ausschließlich geschlossene Körper dar. Die Geometriebeschreibung mit Brep kann für die Modellierung von Solid Model, insbesondere wenn die explizite Geometriedarstellung, wie Extrusion-, Rotations-CSG-Modellierungen, nicht angewendet werden kann (BORRMANN et al., 2015).

Die Grundstruktur des Begrenzungsflächenmodells wird durch die `IfcManifoldSoliBrep` definiert, welches eine abstrakte Superklasse von `IfcFacetedBrep` und `IfcAdvancedBrep` ist. `IfcFacetedBrep` referenziert auf die Beschreibung von 3D-Körpern mit ebenen Flächen und kann anschließend Information über die Geometriehülle enthalten. Durch die Klassen `IfcClosedShell` und `IfcOpenShell` werden geschlossene und offene Hüllen (eine Zusammensetzung von mehreren Flächen) beschrieben. Bei einer geometrischen Beschreibung eines Objektes mit Brep wird in diesem Fall auf die `IfcClosedShell` referenziert. Bei einem Flächenmodell ist dies abhängig von dem zu beschreibenden Objekt. Die Oberflächen einer Hülle werden von `IfcFace` definiert (BOTH, 2009).

Neben der Brep-Beschreibung von begrenzenden Flächen, gibt es auch die Möglichkeit Flächen durch das Flächenmodell oder durch die Tesselationen zu beschreiben. Das Flächenmodell beschreibt analog zu Brep die Oberflächen eines Körpers. Im Gegensatz zur Brep müssen die Flächen dabei nicht geschlossen sein, das Flächenmodell definiert Oberflächen aus mehreren ebenen un- bzw. geschlossene Teilfläche (BORRMANN et al., 2015). Zur Beschreibung von Oberflächen wird bei der Tesselation Dreiecksnetze verwendet. Die Tesselation stellt eine einfache Darstellungsform von Geometrien und wird anhand der Superklasse `IfcTesselatedItem` definiert, welche die Klasse `IfcTriangulatedFaceSet` als Subklasse besitzt. Als Instanz der `IfcTesselatedFaceSet` gehört zudem das `IfcPolygonalFaceSet` mit `IfcIndexedFace`. Hier wird anders als bei der Beschreibung mit `IfcTriangulatedFaceSet` die Flächen als Polygone mit impliziten Polylinien dargestellt (*ISO 16739*, 2021).

Klasseninstanzen in STEP Physical File

Während im IFC Schema die Struktur der Entitäten und Relationen beschrieben werden, speichert das Instanzdokument die Modelldaten nach Vorgaben des EXPRESS-Schemas ab (BORRMANN et al., 2015).

Von BUILDINGSMART (o.D. a) empfehlt bei einer Prioritisierung einer hohen Kompatibilität und einer kleinen Dateigröße die SPF. Daneben kann das schema-konforme Instanzmodell in XML, Terse RDF Triple Language (Turtle) und in anderen Formaten umgewandelt werden.

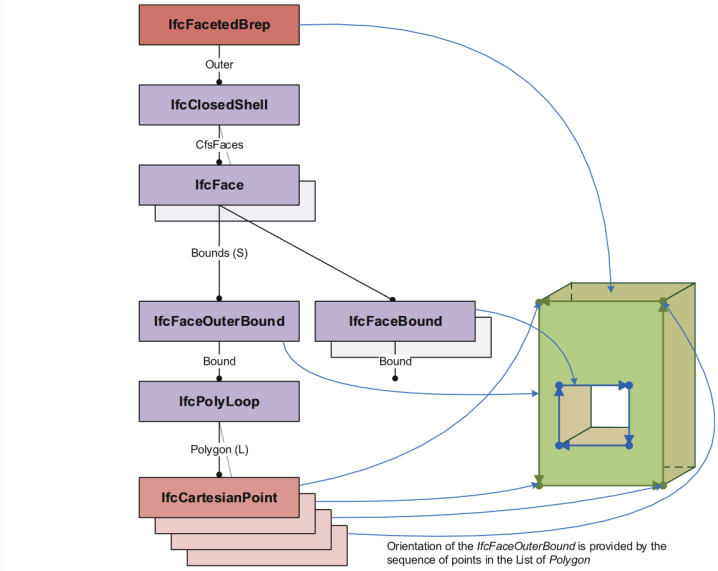

Abbildung 2.1: Darstellung der Hierarchiestruktur von `IfcFacetedBrep` (BORRMANN et al., 2015)

Innerhalb der IFC-Datei in SPF werden ausschließlich Klasseninstanzen in einer Reiehnfolge gespeichert. Die Objekte in SPF weisen keine inversen Referenzen auf und sind lediglich Klasseninstanzen, sodass bspw. keine abstrakte Supeklassen (wie `IfcElement` oder inverse Beziehungen im Dokument vorhanden sind (SIRTL, 2020).

In Algorithmus 2.1 wird der Aufbau eines solchen Instanzdokumentes abgebildet. Am Anfang jeder Datei in SPF wird im Header neben der verwendeten ISO Norm, allgemeine Informationen über die IFC-Datei beschrieben. Danach folgt die Aufstellung der Klasseninstanzen mit den zugehörigen Referenzen und Attribute.

```
1  ISO-10303-21;
2  HEADER;
3  FILE_DESCRIPTION((’ViewDefinition [ReferenceView_V1.2]’,’ExchangeRequirement [Architecture
       ]’),’2;1’);
4  FILE_NAME(’2018_01’,’2022-07-26T06:40:0201:00’,(’’),(’’),’ODA SDAI 22.12’,’23.0.1.318 -
       Exporter 23.0.1.318 - Alternativ-UI 23.0.1.318’,’’);
5  FILE_SCHEMA((’IFC4’));
6  ENDSEC;
7
8  DATA;
9  #1=IFCORGANIZATION(,′ AutodeskRevit2023(DEU)′,,);
10 #2=IFCAPPLICATION(#1,’2023’,’Autodesk Revit 2023 (DEU)’,’Revit’);
11 #3=IFCCARTESIANPOINT((0.,0.,0.));
12 #4=IFCCARTESIANPOINT((0.,0.));
13 #5=IFCDIRECTION((1.,0.,0.));
14 #6=IFCDIRECTION((-1.,0.,0.));
15 #7=IFCDIRECTION((0.,1.,0.));
16
17 ENDSEC;
18
19 END-ISO-10303-21;
```

Algorithmus 2.2: Darstellung eines IFC-SPF

2.1.3 LandXML

Zur Beschreibung von Geoinformationssysteme (GIS) eigent sich das LandXML-Standard, welches zur Spezifizierung von Vermessungs- und Tiefbaudaten verwendet wird. Der non-proprietäre Datenstandard LandXML wurde 2000 durch die Organisation LandXML.org gegründet und verfolgt seitdem das Ziel der Übermittlung von Geoinformationen, sodass die Modellierung von Bauingenieur- bzw. Vermessungsobjekte, wie Leistungsnetze, Fahrbahnen, Messpunkte, etc. ermöglicht wird. (LANDXML.ORG, 2006). Als weit verbreiteter Standard findet LandXML bei zahlreichen Regierungs- und Landbehörden, wie das ICSM in Australien oder in der Abteilung für Landvermessung der Landbehörde von Singapur (SLA) Verwendung (SOON, THOMPSON & KHOO, 2014).

LandXML ist ein XML-basiertes Datenformat, d.h. es baut sich auf die XML-Spezifikation auf und übernimmt damit auch die XML-Syntax mit der hierarchischen Elementstruktur und die Modellierungsregeln (KARKI, THOMPSON, MCDOUGALL, CUMERFORD & OOSTEROM, 2011).

Extensible Markup Language XML

Die Extensible Markup Language (XML), auf dt.: erweiterbare Auszeichnungssprache, wird zur Darstellung strukturierter Daten verwendet. Als text-basiertes Datenformat gilt es als grundlegender Standard (W3C konform), der von Menschen, sowohl als von Maschinen lesbar ist, und so einen reibungslosen Informationsfluss zwischen unterschiedlichen Ebenen ermöglicht (SCHILD & MOCHOL, 2006).

Ein XML Schema spezifiziert die Struktur des Instanzdokumentes (XML Dokument). Ein XML-Dokument weist eine Dateierweiterung *.xml auf, wobei das XML Schema die Endung *.xsd hat (SCHILD & MOCHOL, 2006). Im XML Schema wird zunächst der Namensraum (namespace) angegeben (Algorithmus 2.3). Der Namensraum spezifiziert das Schema der Elemente und Attribute (XML-Vokabular) und fungiert somit als eine Vorlage des Schemas(SCHILD & MOCHOL, 2006).

```
1 <?xml version="1.0" encoding="UTF-8"?>
2    <university>
3       <student>
4          <name>Victor</name>
5          <age>25</age>
6       </student>
7    </university>
```

Algorithmus 2.3: Darstellung eines XML-Dokuments

Ähnlich zur EXPRESS-Modellierungssprache, weist auch das XML-Schema objektorientierte Strukturen auf. XSD deklariert Element-Klassen (parent-element) und seine Subklassen (child-element). Durch die Deklaration werden die Elemente und Attribute spezifiziert, die im Instanzdokument beschrieben werden. Die Element-Deklaration wird dabei durch xsd:element eingeleitet (SCHILD & MOCHOL, 2006).

In Zeile 3 wird das parent-element university deklariert. Dem child-element student werden die Attribute name und age mit den jeweiligen Objekt-Datentypen spezifiziert. Der Datentyp `<xsd:Complex Type>` beschreibt eine wiederverwendbare Struktur unter Verwendung der zuvor deklarierten Elementen/Attributen. Ein Datentyp spezifiziert den gültigen Inhalt eines Elements oder Attributes. So bildet `<xsd:ComplexType>` größtenteils das Grundfundament zur Beschreibung eines XML Schemas. Dadurch können Objektattribute, aber auch Subklassen oder Referenzierungen zwischen Elementen erzeugt werden. `<xsd:sequence>` legt eine feste Reihenfolge der Abläufe von Elementen und Attributen fest (SCHILD & MOCHOL, 2006).

```
1  <?xml version="1.0"?>
2  <xsd:schema xmlns:xs=https://www.w3.org/org/2001/XMLSxhema>
3  <xsd:element name="university">
4    <xsd:complexType>
5      <xsd:sequence>
6        <xsd:element name="student">
7          <xsd:complexType>
8            <xsd:sequence>
9              <xsd:element name=\"name\" type=\"xsd:string\"/>
10             <xsd:element name=\"age\" type=\"xsd:integer\"/>
11           </xsd:sequence>
12         </xsd:complexType>
13 </xsd:element>
14 </xsd:sequence>
15   </xsd:complexType>
16 </xsd:element>
```

Algorithmus 2.4: Darstellung eines XML Schemas

Modellstruktur

Ein wichtiger Bestandteil der Modellierung in LandXML sind die Geometrie-, Vermessungsdaten. Durch Geometriedaten werden topologische und geometrische Daten, wie Koordinaten, Flurtücke oder Flächen vermittelt (RAJABIFARD, 2012). Anders als bei anderen Datenmodellen, ist kein konzeptionelles Modell integriert (KANG, 2018).

In LandXML 1.2 wird die abstrakte Superklasse LandXML in 16 Subklassen (Units, Coordinate System, Project, Application, CgPoints, Amendment, GradeModel, Monuments, Parcels, PlanFeature, PipeNetworks, Roadway, Surfaces, Survey, FeatureDictionary and Alignments) unterteilt. Diese Subklassen können anschließend mit anderen Elementen referenziert werden. (LANDXML.ORG, o.D.).

Jede direkte Instanz einer LandXML-Klasse stellt eine Sammlung von Objektklassen dar. Bspw. wird die Klasse `Alignments` als eine Sammlung von Alignment-Objekten definiert, sodass das Objekt `Alignemnt` ein child-element des parent-elements `Alignments` beschreibt. Daneben können Klassen das Element `Feature` zugeteilt werden. Das `Feature`-Element beschreibt zusätzliche Informationen, die nicht explizit im LandXML Schema spezifiziert sind. Diese können in die Subklassen Property, DocFileRef und Feature unterteilt werden. Die Properties eines Elements, hängt jeweils individuell von dem Objekt ab und besitzt die Attribute label und value. (LANDXML.ORG, o.D.; SOON et al., 2014).

Zum besseren Verständnis der Objektstruktur wird als Beispiel die Klasse `Alignment`, die eine geometrischen Ausrichtung, PGL oder Kette beschreibt, betrachtet (Abb. 2.3). Das Objekt `Alignment` definiert gewöhnlicherweise eine Straßenentwurfmittellinie (LANDXML.ORG, o.D.). Dem Objekt `Alignment` wird ein obligatorisches Attribut wie z.B. `name`, `length` und `staStart` zugeordnet, wobei die restlichen Attribute optional sind (Abb. 2.2).

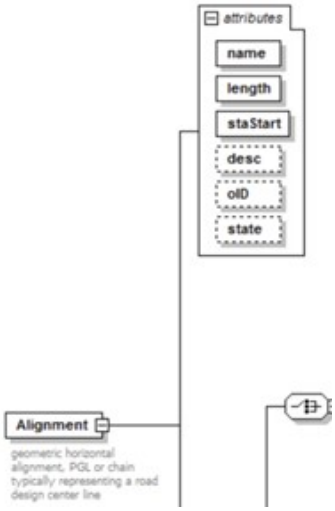

Abbildung 2.2: Darstellung der Attribute des Elements `Alignment` (LANDXML.ORG, o.D.)

In Relation zu `Alignment` steht das Element `CoordGeom`, das eine Liste von Linien- und Kurvenelemente enthält. Weitere Elementrefrenzen können mit `StaEquation`, `Profile`, `CrossSects`, `Superelevation` und `Feature` gestellt werden.

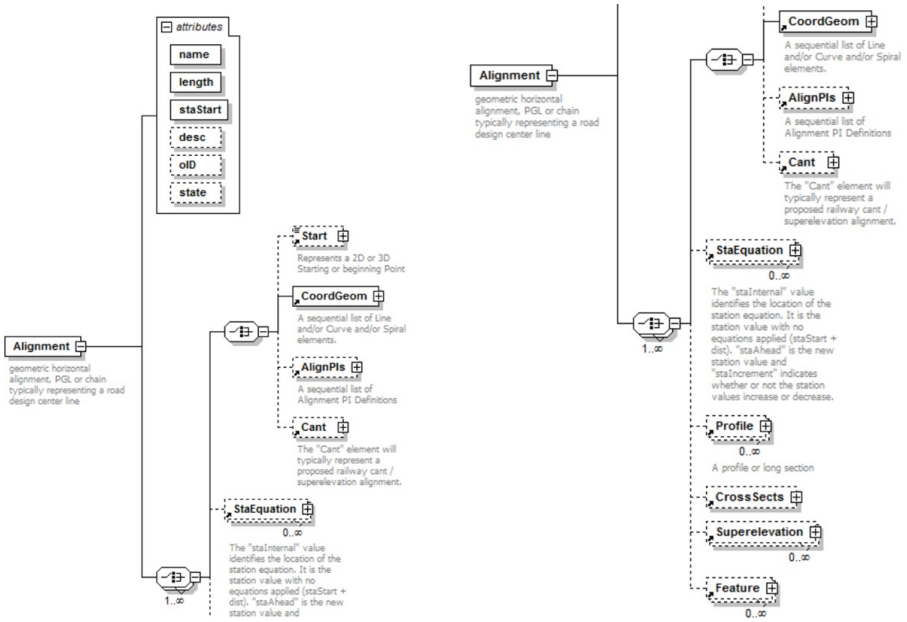

Abbildung 2.3: Darstellung der Relationen der Klasse `Allignment` (LANDXML.ORG, o.D.)

2.1.4 CityGML

Im CityGML-Modell wird der Fokus, im Gegensatz zu LandXML oder IFC, auf die 3D-Städte-, sowie Landschaftsmodellierung gerichtet. Es dient als offenes, neutrales Datenmodell zur differenzierten Abbildung von geografischen und topografischen Stadt- bzw. Landschaftsstrukturen. CityGML wurde 2002 durch Mitglieder des SIG 3D entwickelt und wurde mit CityGML 1.0 im Jahr 2008 offiziell als internationaler OpenGIS-Standard in das OGC eingeführt (BRÜGGEMANN & von BOTH, 2015)

Als eine Implementierung des ISO-konformen GML3.0-Anwendungsschemas erweitert der CityGML-Standard auf Grundlage des GML-Schemas/Modellierungsregeln die urbane Objekmodellierung (BRÜGGEMANN & von BOTH, 2015; RAJABIFARD, 2012).

CityGML ermöglicht die Darstellung von dreidimensionale Geometrie, Topologie und das Erscheinungsbild. Dabei steht der semantische Kontext in der Modellierung in Fokus (BRÜGGEMANN & von BOTH, 2015; GRÖGER & PLÜMER, 2012).

Geography Marup language GML

Geography Markup Language (GML) ist eine herstellerunabhängige, neutrale Auszeichnungssprache zur Beschreibung von dreidimensionalen Geometrien und deren topologischen Eigenschaften. GML ist eine XML-Kodierung (bzw. XML-Erweiterung) und kann als Spezifikation von Austauschformaten für Geodaten verstanden werden (*ISO 19136-1*, 2020). Als Grundlage werden Konzeptmodelle der Normreihen ISO 191xx des Internationalen Standards zur Spezifikation von Geoinformationen angewendet (BRÜGGEMANN & von BOTH, 2015). Im folgenden werden wichtige Konzeptmodelle aufgelistet.

1. ISO19109 bestimmt die Regeln zu Anwendungsschemata.
   Nach *ISO 19109* (2016) werden sogenannte Features – welches ähnlich zu verwenden ist wie ein Objekt – innerhalb des GML-Schemas definiert.
2. *ISO 19107* (2020) definiert die räumliche Geometrie- und Topologiedaten
   GML-Geometrieobjekte werden u.a. durch gml:AbstractGeometry abgeleitet

Basierend auf diese Normen, leitet das Anwendungsschema Klassen, Attribute und Relationen vom Feature-Modell ab und spezifiziert diese (KUTZNER, CHATURVEDI & KOLBE, 2020). D.h. erst durch die Konkretisierung des Anwendungsschemas können Objekte instanziiert werden. Als solches Anwendungsschema wird in dieser Arbeit der CityGML 3.0 Standard vorgestellt.

CityGML-Modellstruktur

Im Konzeptmodell werden insgesamt 17 Module spezifziert. Im CityGML Core werden alle abstrakte Subklassen der Klasse CityModel, sowie auch die Geometrie basierend auf

ISO19107 definiert (KUTZNER et al., 2020). Dazu sind weitere 12 thematische Module im Datenmodell spezifiziert, die auf die thematischen Module Building, Bridge, Tunnel, CityFurniture, CityObjectGroup, LandUse, Relief, Transportation, vegetattion, WaterBrody und Construction bezieht. Dazu ergeben sich noch fünf Erweiterungsmodule (siehe Abbildung Abb. 2.4) (KUTZNER et al., 2020).

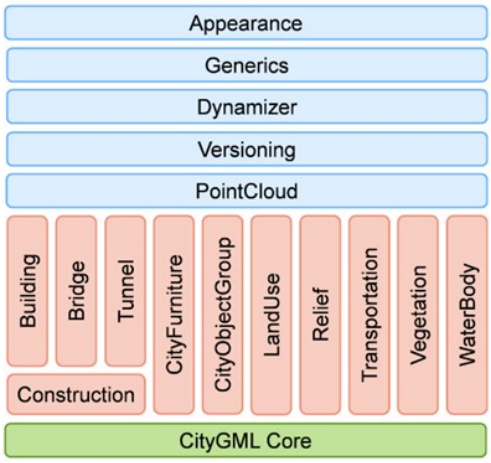

Abbildung 2.4: Konzeptmodell des CityGML 3.0 (KUTZNER et al., 2020)

Basierend auf dem GML-Schema werden Klassen, Attribute und Eigenschaften durch objektorientierte Strukturen definiert. Innerhalb des CityGML-Datenmodells werden die einzelnen Objekte vom Feature-Modell nach ISO 19109 abgeleitet (BRÜGGEMANN & von BOTH, 2015).

Seit der Aktualisierung von CityGML 2.0 auf 3.0 wurden zusätzlich die Interoperabilität zwischen veschiedenen Datenmodelle wie IndoorGML oder RDF durch die Revisionierung der Verlinkung externen Objektreferenzen verbessert. Durch die Bestimmung von Relationstypen können externe Relationen mit einem URI versehen werden, sodass eine Transformation in ein RDF-Tripel möglich ist. Die abstrakte Klasse CityModel bildet dabei das Root-Element aller CityGML-Objekten (KUTZNER et al., 2020). In Abb. 2.5 wird die abstrakte Klasse AbstractBuilding in zwei individuelle FeatureTypes Building und BuildingPart unterteilt, die die Attribute von der Superklasse AbstractBuilding erben.

Wie bereits im vorherigen Abschnitt erwähnt können CityGML-Fetauretypen durch die in der ISO 19107 definierten Geometrieelementen beschrieben werden. Volumetrische Objektformen in CityGML werden in Brep-Strukturen beschrieben und integrieren die primitiven Geometrien von GML wie Punkte, Linien, etc. (KANG, 2018).

2.2 Validierungsansätze auf objektorientierte Modellinhalten

Für einen erfolgreichen Arbeitsprozess mit BIM ist die Validierung bzw. Konformitätsprüfung der Modell-Datensätze für die Qualitätssicherung essenziell. Mit der Einführung von BIM und der zunehmend fortschreitender Entwicklung neuer Technologien in dem Themengebiet wurden zahlreiche Validierungssoftware und -ansätze entwickelt, um eine

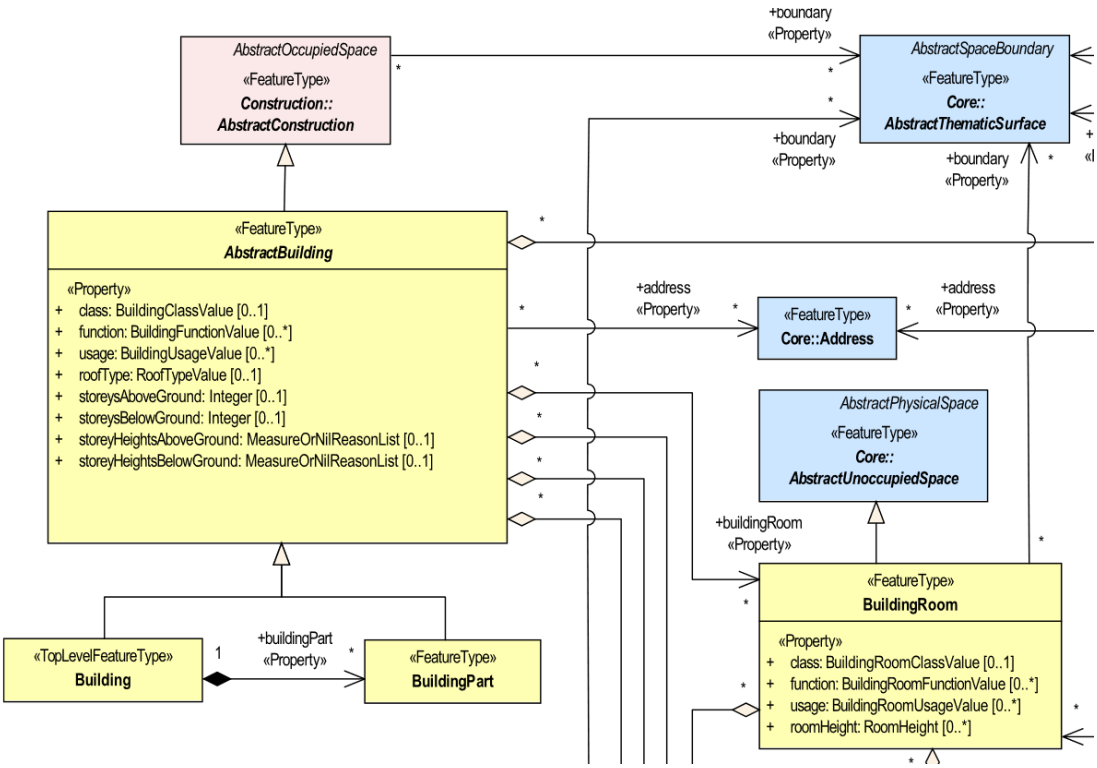

Abbildung 2.5: Subklassen `AbstractBuilding` (KUTZNER et al., 2020)

effiziente automatisierte bzw. teilautomatisierte Code Compliance Checking auszuführen (PREIDEL & BORRMANN, 2015).

Im Folgenden werden die vier-schrittige Grrundstruktur des Validierungsprozesses nach EASTMAN, LEE, JEONG und LEE (2009) und PREIDEL und BORRMANN (2015) vereinfacht vorgestellt:

1. Regelwerke, die in verschiedener Art und Weise dargestellt sind (bspw. als Fließtext, Graphik) werden im Schritt der Rule Interpretation in eine maschinen-lesbare Sprache transformiert.

2. Der Schritt Rule Execution führt die zuvor in maschinen-interpretiebarer Sprache umgewandelte Regeln auf das zu validierenden Modell an. Hier wird bevor die Überprüfung der eigentlichen Regeln ausgeführt werden, eine sogenannte Model View syntactic pre-checking umgesetzt. Dieser Zwischenschritt gilt als Vorbedingung der Rule Execution und überprüft, ob die benötigten Daten im Modell vorhanden sind und keine Unstimmigkeiten herrschen.

3. Im Building Model Preparation wird ausschließlich die zu benötigten Information eines Gebäudemodells extrahiert und abgeleitet.

4. Abschließend, wird im letzten Schritt die Ergebnisse der Prüfung verarbeitet und angezeigt.

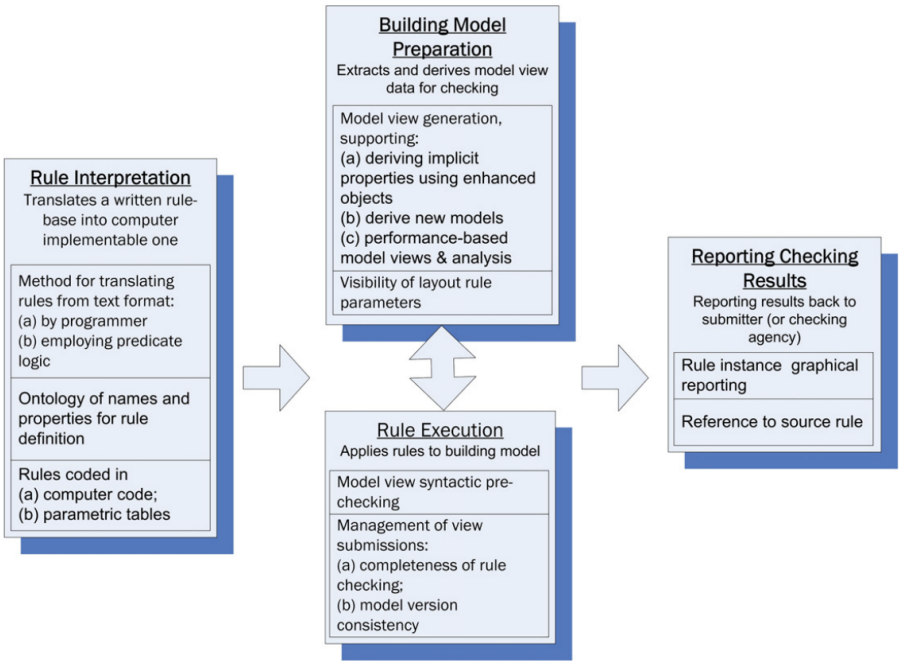

Abbildung 2.6: Darstellung des vier-schrittigen Validierungsprozesses nach EASTMAN et al. (2009)

Mit zunehmender Digitalisierung des Bauprozesses wird in Hinblick auf die Validierung von Datensätzen neben der Formulierung von einfacher Sprache verfassten Prüfregeln, die Benutzerfreundlichkeit der Validierungswerkzeuge in den Vordergrund gestellt.

2.2.1 BIM Tester

BIMTester ist ein plattformübergreifendes Softwaretool zur Überprüfung von IFC-Datenmodelle. Es handelt sich hierbei um eine AEC Free Software, welche die Grundsätze des OpenBIMs nachgeht (IFCOPENSHELL, 2022).

BIMTester implementiert das Konzept des behave bei der Beschreibung und Ausführung von Prüfregeln (MOULT & KRIJNEN, 2020). Behave gehört zum Rahmenwerk des Behavior-Driven Development (BDD), welches einen Ansatz in der agilen Softwareentwicklung zur Förderung der Interoperabilität zwischen Propgrammierern und Nicht-Programmierern durch eine allgegenwärtige Sprache beschreibt. Zusammengefasst arbeitet behave hauptsächlich mit folgenden Kernelementen (BEHAVE, 2017b).:

- Feature-Dateien (Anforderungen werden in einfacher Sprache geschrieben)
- Step-Directory mit Python step Implementierungen (Technische Umsetzung der steps mit Python Codes)
- Environment.py (Definition der Umgebungseinstellung für die Validierung. Die Codes in environmnet.py können vor oder nach ein Ausführungsereignis erfolgen.)

Zur Ausführung der Modellprüfung kann der BIMTester im Standalone Modus verwendet werden. Daneben stehen derzeit Open Source Softwares, die den BIMTester in ihrer Anwendung implementiert haben (wie bspw. BlenderBIM Add-On oder FreeCAD), zur Verfügung (IFCOPENSHELL, 2022). Die durch die Prüfung resultierenden Validierungsergebnisse können als Bericht in verschiedenen Formaten (HTML, JSON oder XUnit) generiert werden (MOULT & KRIJNEN, 2020).

Im folgenden Kapitel wird die Beschreibung und Ausführung der Prüfregeln im BIMTester auf Grundlage des BDD-Prinzips genauer erläutert.

Feature Files

Basierend auf dem Konzept von behave werden die Prüfanforderungen der Feature-Dateien in der Gherkin Sprache geschrieben. Der Gherkin Syntax zeichnet sich durch die einfach formulierte Sprache aus. So können Anforderungen an ein BIM-Modell mit dem Gherkin Syntax in einfacher Sprache für Technische, sowie Nicht-Technische verständlich verfasst werden (CAUCHI, COLOMBO, FRANCALANZA, MICALLEF & PACE, 2016).

Die Struktur einer Feature-Datei in der Gherkin Sprache des BIMTesters ist in Algorithmus 2.5 dargestellt.

```
1 Feature: # Enter feature name here
2   # Enter feature description here
3
4   Scenario: # Enter scenario name here
5     # Enter steps here
```

Algorithmus 2.5: Beispiel einer Feature file

Eine `Feature` beinhaltet eine oder mehrere thematisch gleichgestellte `Szenarien`. Durch die Bestimmung von `Features` können, ähnlich wie im unit test, einzelne Einheiten geprüft werden. Optional kann im nächsten Absatz mit dem Präfix # eine Beschreibung des Features einkommentiert werden (BEHAVE, 2017b).

Anschließend wird ein bzw. mehrere `Szenarien` spezifiziert. Ein `Szenario` enthält eine Sammlung von spezifischen Anforderungen in Form von einen oder mehreren `steps`. Steps beginnen gewöhnlicherweise mit einzelnen Schlüsselwörtern wie `GIVEN`, `WHEN` und `THEN`. Für die Validierung mit dem BIMTester wird eine Annotation * genutzt. Semantisch ist dies gleichzustellen mit einem `THEN`-step. Erst durch die Verwendung des `THEN`-steps ist es für den Benutzer möglich, spezifische Anforderungen an das Modell zu stellen (CAUCHI et al., 2016; TIDMARSH, 2021). Zum besseren Verständnis wird in Algorithmus Algorithmus 2.6 die Feature-Datei Grundlagen.feature dargestellt.

BIMTester hat für die Validierung von IFC-Datenmodelle die sogenannten MicroMVDs integriert. MicroMVDs sind in Feature files formuliert und sind zum Teil veröffentlicht. Es ist jedoch festzuhalten, dass die MicroMVD nach aktuellem Stand nicht vollständig im BIMTester implementiert sind (BUILDINGSMART, 2022a).

```
# language: de

Feature: Basisdaten
 Um BIM-Daten anzusehen
 Für alle beteiligten Akteure
 Wir brauchen eine IFC-Datei

 Scenario: Bereitstellen von IFC-Daten
 * Die IFC-Daten müssen das "IFC2X3" Schema benutzen
```

Algorithmus 2.6: Darstellung einer Grundlagen.feature file

Python-Step-Implementierung

Die step-Anforderungen einer Feature file wird mittels Step-Definitionen im Verzeichnis `./feature/steps` implementiert (siehe Abb. 2.7). Step-Definitionen sind Implementierungsfunktionen, die in Python Code beschrieben werden und befinden sich in Python Dateien mit der Dateierweiterung `*.py` (BEHAVE, o.D.).

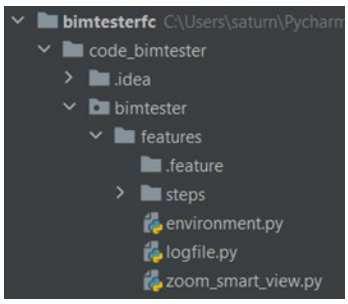

Abbildung 2.7: Verzeichnis des BIMTesters in PyCharm

Die Python-step Funktionen werden durch sogenannte Dekoratoren identifiziert, die einen string Argument besitzen. Die Zeichenkette ist dabei identisch mit der step-Anforderung der Feature Datei. Der step-Dekorator verwendet eine Funktion step.impl(context), dessen Variable context bei jeder Step-Definition an weitere Funktionen übergeben werden kann. Innerhalb der Funktion können weitere step-Parameter bestimmt und eingebracht werden (BEHAVE, 2017b). Im Algorithmus Algorithmus 2.7 wird bspw. neben dem Argument context, die value-Variable in Zeile 2 definiert .

```
1 @step('The project name, code, or short identifier must be "{value}"')
2 def step_impl(context, value):
3     util.assert_attribute(IfcStore.file.by_type("IfcProject")[0], "Name", value)
```

Algorithmus 2.7: Beispiel einer Step-Definition

Im Header des Python-Dokuments wird mit:

```
from behave import step
```

der step-Dekorator von behave importiert. Dieser verweist auf ein step-Register (step.registry.py), welches das Matching der steps (feature file) und der Python-step-Implementierung erst ermöglicht (BEHAVE, 2017a).

2.2.2 MVD-Konzept

Grundkonzept

Das IFC-Modell enthält Informationen aus verschiedenen Domänen (beschrieben im Domain Layer) der AEC-Industrie und ist daher ein sehr umfangreiches Datenmodell zur Darstellung von BIM-Modellinhalten (siehe Abschnitt 2.1.1).

Im Zuge eines Datenaustausches zwischen einzelnen Fachdisziplinen und Fachmodellen innerhalb eines Projektes kann es durch Verwendung des kompletten IFC-Modells sehr schnell zur Unübersichtlichkeit und zu Datenfehler im Arbeitsprozess führen, welche erheblichen Einfluss auf die Zeitplanung des Projekts durch einen zusätzlichen Korrekturprozess herbeiführen kann. Für einen effektiven und vorteilhaften Datenaustausch von BIM-Daten ist es von großer Wichtigkeit ausschließlich der benötigten Informationen bzw. eine Teilmenge des Informationsmodells herauszufiltern und zu verwenden (HÄRINGER, 2017).

Model View Definition (MVD) stellt als eine prozessgestützte Definition eines Modellinhalts eine Teilmenge des IFC-Modells dar und konkretisiert den Umfang des Datenaustausches durch die Beschreibung spezifischer Austauschanforderungen (eng.: Exchange Requirement) (HÄRINGER, 2017). In BUILDINGSMART (2022b) sind bspw. folgende offizielle MVDs veröffentlicht:

1. Model View Reference: kann für den Datenaustausch zwischen den Domänen Architektur, Tragwerk und Haustechnik verwendet werden.
2. Das Quantity Takeoff View MVD schätzt und verfolgt Baumaterialien und Kosten.

Konfiguration des mvdXML

Für die Validierung der Modellinhalte wird zunächst ein mvdXML generiert und dieses gegen das zu prüfende IFC-Modell validiert. MvdXMl können manuell, aber auch derzeit durch folgende Softwares, wie ifcDoc Tool oder BIMQ generiert werden (HÄRINGER, 2017). Das Grundprinzip der Validierung basiert sich auf Constraints. Durch Constraints werden Bedingungen an ausgewählte Instanzen gestellt, die erfüllt werden müssen. Ein mvdXML kann in zwei Teilen, dem Concept template und dem Model View unterteilt werden. Das Concept template definiert den Subgraphen, d.h. spezifiziert als eine Art Vorlage die instanzierbaren Attribute des Concept Templates. Das Model View spezifiziert dabei die Formulierung der Austauschanforderungen (LIU et al., 2022; WEISE, LIEBICH, NISBET & BENGHI, 2016).

Das Concept template definiert den Umfang der anwendbaren Entitäten (applicableEntity), auf welche die Vorlage gilt und legt Regelsätze <Rules> für den Subgraphen fest. Ein solcher Subgraph wird durch <AttributeRules> definiert, die dann mit <EntityRules> erweitert werden können. D.h. innerhalb eines Subgraphens werden Attributregeln der jeweiligen

instanzierbaren Attribute definiert. Ein Subgraph des `IfcRoot` ist im Beispiel abgebildet. Hier werden die Regeln/Form der Attribute `Name` und `Description` des Entitäts `IfcRoot` festegelegt. Ein Name soll ein IfcLabel sein, die Beschreibung hingegen ein `IfcText` (WEISE et al., 2016).

```
<Concept
 uuid="00000003-0000-0000-0000-000000349910"
 name="Accessibility Labels">
  <Template
   ref="00000000-0000-0000-0001-000000000001"/>
  <Requirements>
  <Requirement applicability="import"
   exchangeRequirement="00000003-0000-0000-
   0000-000000000105" requirement="mandatory"/>
  </Requirements>
  <TemplateRules operator="and">
    <TemplateRules operator="or">
      <TemplateRule Parameters=
      "Set[Value]='STREAMER_Labels_PoR' AND
      Property[Value]='AccessSecurity' AND
      Value[Value]='A1'"/>
      <TemplateRule Parameters=
      "Set[Value]='STREAMER_Labels_PoR' AND
      Property[Value]='AccessSecurity' AND
      Value[Value]='A2'"/>
    </TemplateRules>
  </TemplateRules>
</Concept>
```

Abbildung 2.8: Beispiel eines Concept Templates (WEISE et al., 2016)

Erst mit dem ModelView können spezifische Austauschanforderungen festgelegt und überprüft werden. Die Constraints bzw. die Austauschanforderungen werden durch ConceptRoot und Concepts festgelegt. Das Prinzip des Model Views basiert sich auf die IF-THEN-Struktur (WEISE et al., 2016). ConceptRoot definiert die Grundlagen des MVDs und stellt eine Vorbedingung (IF-Anweisung) für den im Concepts-Teil spezifizierten Anforderungen dar. Concepts enthält <TemplateRules>, die Austauschanforderungen der im ConceptTemplate definierten Attribute formuliert. Attribute werden durch das RuleID referenziert (WEISE et al., 2016).

```
<ConceptRoot
 uuid="00000035-0000-0000-2000-000000067001"
 name=" Beam-206"
 applicableRootEntity="IfcBeam">
  <Applicability><Template
  ref="c19ec186-9cfd-47fc-a4d4-9fb35008d04a"/>
  <TemplateRules operator="and">
    <TemplateRule
      Parameters="Name[Value]='Beam-206'"/>
  </TemplateRules>
  </Applicability>
```

```
<Concept
 uuid="00000003-0000-0000-0000-000000349910"
 name="Accessibility Labels">
  <Template
   ref="00000000-0000-0000-0001-000000000001"/>
  <Requirements>
  <Requirement applicability="import"
   exchangeRequirement="00000003-0000-0000-
   0000-000000000105" requirement="mandatory"/>
  </Requirements>
  <TemplateRules operator="and">
    <TemplateRules operator="or">
      <TemplateRule Parameters=
      "Set[Value]='STREAMER_Labels_PoR' AND
      Property[Value]='AccessSecurity' AND
      Value[Value]='A1'"/>
      <TemplateRule Parameters=
      "Set[Value]='STREAMER_Labels_PoR' AND
      Property[Value]='AccessSecurity' AND
      Value[Value]='A2'"/>
    </TemplateRules>
  </TemplateRules>
</Concept>
```

Abbildung 2.9: Beispiel eines Concept Roots bzw. Concepts (WEISE et al., 2016)

Zur Validierung des Modells gegen das generierte mvdXML werden derzeit bspw. die Softwares xBIM Xplorer oder SimpleBIM zur Verfügung gestellt. Das Validierungsresultat

gibt schließlich ein booleansches Ergebnis oder wird im Interface der jeweiligen Software angezeigt (HÄRINGER, 2017).

2.2.3 Validierung mit SHACL

Shapes Constraint Language (SHACL) spezifiziert Daten des World Wide Web Consortiums (W3C) und kann als Validierungssprache zur Beschreibung und Überprüfung von Resource Description Framework (RDF)-Graphen bzw. RDF-Terme verwendet werden, die als Linked Data (hier: durch turtle Sytax) serialisiert sind (KNUBLAUCH & KONTOKOSTAS, 2017; ORASKARI, BEETZ & SENTHILVEL, 2021). Die Prinzipien eines RDF-Graphens werden in Abschnitt 2.3.1 näher erläutert.

SHACL definiert sogenannte shapes mit dem Präfix sh:, die eine oder mehrere Constraints deklarieren und bestimmt die targets, für die die Constraints. In der Validierung wird geprüft, ob die Knoten des RDF-Graphens dem shape entsprechen (STARDOG, 2019).

Die Validierung mit SHACL erfolgt durch den Data Graph und den Shapes Graphen. Im Shape Graphen werden die shapes spezifiziert. Der Shapes Graph, welches die Bedingungen und die Anforderungen an das RDF-Graph in der SHACL-Sprache beschreibt, wird gegen den Data Graph validiert. Das Ergebnis der Validierung wird automatisch in einem Validierungsbericht dargestellt, der in SHACL-Vokabular beschrieben ist. Unter Verwendung eines SHACL-Prozessors (das System, mit der die Validierung durchgeführt wird) kann der Validierungsprozess mit SHACL ausgeführt werden (KNUBLAUCH & KONTOKOSTAS, 2017). Zu solchen Softwares gehören bspw. Stardogs (STARDOG, 2019).

Shape Graph

Ein Shape Graph besteht aus folgenden Komponenten:

1. Target: Ein target definiert ein Zielobjekt, welches gegen eine shape validiert werden. D.h. Die Target-Deklaration wählt die Subjekte aus, für welche die Constraints gelten (KNUBLAUCH & KONTOKOSTAS, 2017).
2. Focus Node: Ein focusNode beschreibt ein Knoten des RDF-Graphens, der gegen die shapes validiert wird. Gewöhnlicherweise ist ein FocusNode gleich der TargetNode (KNUBLAUCH & KONTOKOSTAS, 2017).
3. Property Shapes: Ein PropertyShape enthält constraints über ein Objekt eines focus nodes. Solche Constraints werden durch Constraint Components beschrieben, die Parameter zur Spezifizierung der individuellen Einschränkungen enthalten (STARDOG, 2019). In Abb. 2.10 werden bspw. die Constraints sh:maxCount, sh:datatype und sh:pattern dargestellt.
4. Value node: Ein value node beschreibt den zu validierenden Wert. Vor der Validierung ist der value node gleich der focus node. Mit der Definition des Property Shapes wird der value node auf das zu validierende Objekt verschoben (STARDOG, 2019).

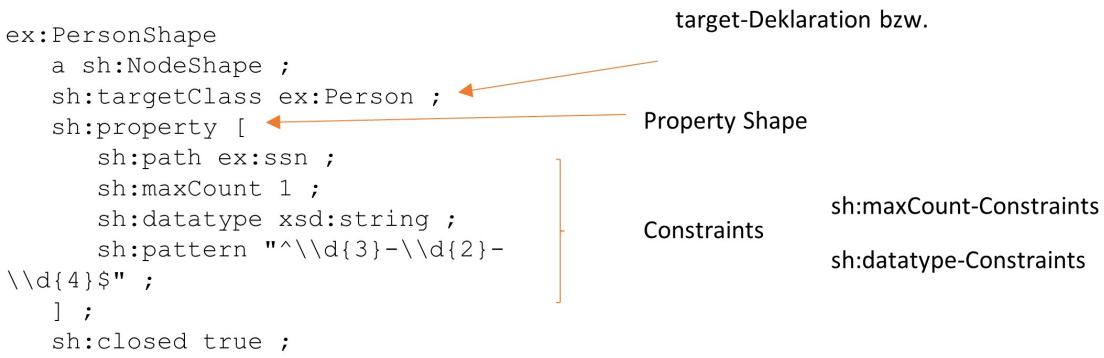

Abbildung 2.10: Darstellung einer SHACL Deklaration

Data Graph

Ein Data Graph ist ein RDF-Graph und beschreibt die zu prüfenden RDF-Triple in bspw. turtle-Syntax dar. D.h. ein Data Graph enthält Instanzen des jeweiligen targets des Shape Graphs bzw. ein target im Shape Graph beschreibt Constraints für die zu prüfenden Nodes im Data Graph. D.h. es werden die jeweiligen Subklassen einer SHACL Klasse (hier ex:Person) als Subjekt mit den zugehörigen Prädikat und Objekt beschrieben, die vom Shape Graph validiert werden soll (KNUBLAUCH & KONTOKOSTAS, 2017).

2.2.4 Solibri Model Checker (SMC)

Solibri Model Checker (SMC) ist eine java-basierte Anwendung zur Qaulitätskontrolle von Gebäudemodellen verschiedenener Domänen, die im Jahr 2000 von der finnischen Technologieforma Solibri Inc. entwickelt wurde (PREIDEL, BORRMANN & BEETZ, 2015).

Als eine weltweit renommierte Validierungssoftware, enthält SMC zahlreiche Regelfunktionen zur Prüfung von Modellen. Dabei können Modelle in verschiedenen Formaten (IFC-, DWG-Format) importiert werden, die in ein Plattform-internes Datenmodell gespeichert und anschließend im Zuge des Überprüfungsprozesses verarbeitet werden (TEMPLETON, 2015).

Der Solibri Model Checker ist eine JAVA-basierte Plattform zur Validierung und Optimierung von Gebäudemodellen. SMC wurde in erster Linie für die digitale Massenermittlung im Jahr 2000 eingeführt und gehört seitdem zu einer der verbreitesten Validierungssoftware. SMC wird vor allem in Qualitätskontrollen, Konformitätsprüfungen, aber auch als Werkzeug für die Designanalyse verwendet (TEMPLETON, 2015).

Ruleset Manager

Die Auswahl der Regeln wird im Ruleset Manager unter Verwendung einer Regelformelsammlung in einer SMC-internen Bibliothek bestimmt. Innerhalb eines Ruleset Managers können vordefinierte Prüfregeln ausgewählt und abhängig von der Anwendungs-

situation über die Tabellensteuerungsparamter auf der grafischen Benutzeroberfläche angepasst werden. In SMC stehen grundlegende Prüfregeln für Architektur, Mechanik, Elektrik, etc. , sowie Regeln, die die Modellstrukutr überprüfen zur Verfügung (TEMPLETON, 2015).

Innerhalb der standardgemäßen Regeln werden hauptsächlcih Modellkomponente und deren Eigenschaften bzw. Eigenschaftssets geprüft. Daneben kann man mit der Regelsammlung des Solibri Acessibility Rule-Ordners auch Regeln in Bezug auf Raumgeometrie und der Zugänglichkeit prüfen (PREIDEL & BORRMANN, 2015). Die Validierung an sich wird im Layout ÜBERPRÜFEN durchgeführt (SOLIBRI, 2022).

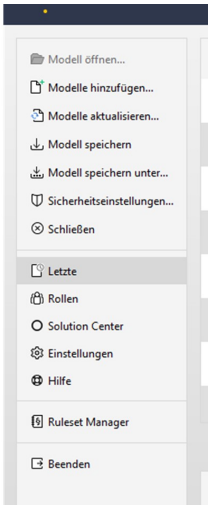

Abbildung 2.11: Ruleset Manager im Datei-Layout

Vor der tatsächlichen Ausführung der Validierung werden ggf. vom SMC zusätzliche Informationen benötigt, die separat vom Benutzer eingegebn werden müssen. Dies dient als Vorbedingung damit die Validierung vollständig stattfinden kann. Diese wird in Form einer To-Do- Liste angezeigt (TEMPLETON, 2015).

Die Erstellung von neuen Regelstrukturen ist derzeit nur von der Firma Solibri Inc. möglich, da alle Regeln durch eine SMC-API (Application Programming Interface) verwaltet werden (EASTMAN et al., 2009). Die Datenextraktion vom Modell wird im Zuge der Validierung somit über eine fest-kodierte Programmierschnittstelle durchgeführt. Diese Schnittstelle ist für den Benutzer nicht zugänglich, sodass die Art und Weise, welche Informationen mit welcher Methodik wie abgefragt werden nicht ersichtlich ist (PREIDEL et al., 2015).

Überprüfen-Layout

Die Ergebnisse der Überprüfung werden abschließend verarbeitet und in der Ansicht ÜBERPRÜFEN hierarchisch dargestellt. Nicht erfüllte Anforderungen einer Prüfregel werden in der Result View nach Kategorie, Subkategorie und Issues zur besseren Analyse des Ergebnisses unterteilt (SOLIBRI, 2022).

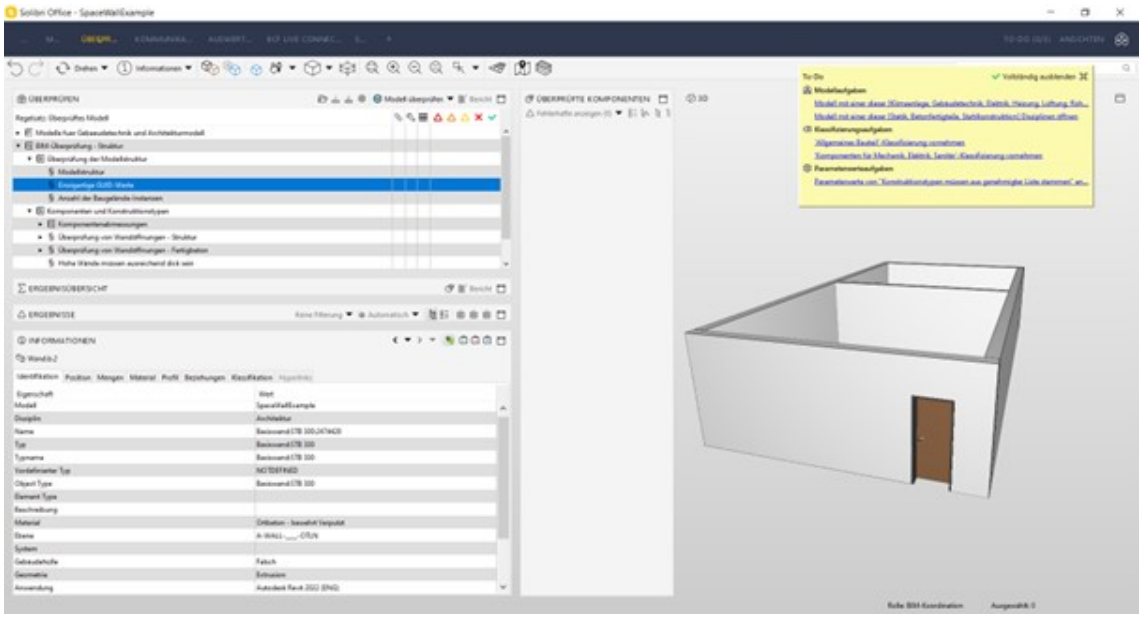

Abbildung 2.12: SMC Benutzeroberfläche

2.2.5 Vergleich der Validierungsansätze

Ein wesentlicher Unterschied zwischen den Validierungsansätzen besteht in der jeweiligen Regel-Interpretation. Abhängig von der Methodik der Regelimplementierung in den Lösungsansätzen, wird größtenteils auch die Fähigkeit der Validierung im Allgemeinen bestimmt. Im BIMTester wird die Übersetzung der Regeln durch die Implementierung der Python-Funktionen ermöglicht. Die Regelformulierung mit der Programmiersprache Python kann für zahlreiche Anwendungsgebiete verwendet werden und ermöglicht eine übersichtliche Struktur. Die Python-step-Implementierung hat den Vorteil, dass bspw. zahlreiche Library importiert werden können, sodass es eine umfangreiche Datenanalyse ermöglicht. Das MVD-Konzept stellt hingegen eine kompliziertere Struktur zur Übersetzung der Regeln in eine maschinen-lesbare Sprache. Durch den XML-Syntax und der Aufteilung des gesamten Validierungsprozess in Concept Template und Model View ergeben sich mehrere Beschränkungen. Bei mvdXML kann die Validierung von Objekten ausschließlich in der Schema-Ebene ausgeführt werden, da mvdXML durch die Constraints nur die Form der Daten bestimmen. So können keine Regeln wie bspw. „Alle Wände sollen eine Höhe von 5m betragen “ mit mvdXML nicht ausgeführt werden. Ähnlich zu mvdXML kann die Validierungssprache SHACL nur vorhandene Daten des RDF-Graphens auf ihre (Austausch-) Anforderungen gegen Constraints eines shape graphs validiert werden. Komplexe Validierungen von Objektzusammenhänge sind somit nur im geringen Maße möglich. Damit Validierungsansätze in anderen Lösungen implementiert werden können, besteht die Notwendigkeit, dass die Implementierung der Regeln transparent und für jeden zugänglich sind (White Box-Methode). In der properitären Validierungssoftware Solibri Model Checker ist dies nicht der Fall, da die Regelstruktur in der Software fest-kodiert sind und für den Benutzer nicht offen zur Verfügung stehen. Man spricht hier von einer Black Box, d.h. Kodierungsstrukturen der Solibri-Regeln sind nicht öffentlich zugänglich.

Im BIMTester wird die Ausführung der feature files mit der Python-step-Implementierung strikt getrennt. Dies ermöglicht die Beschreibung der feature files im Gherkin Syntax, während die eigentliche Regelimplementierung durch eine vergleichsweise kompliziertere Programmiersprache Python durchgeführt wird. Die Prüfregeln (feature files), die vom Benutzer bedient werden, sind in Gherkin Sprache verfasst, die der menschlichen Sprachgebrauchs sehr nahe liegt. Dies dient dem Vorteil, dass alle Projektbeteiligte (Technisch- oder Nicht-Technisch) ohne tieferes Wissen über die Programmierung bzw. technischen Implementierung die Validierung ausführen können. MvdXML und SHACL stellen verglichen zum BIMTester eine komplizierte Sprachstruktur dar. In mvdXMl wird durch den XML-Syntax die Lesabrkeit und somit auch das Versändnis der Prüfregeln stark erschwert. Das gilt auch für die Prüfregelformulierung in SHACL-Sprache. Bei der Überprüfung von komplexeren Datensätze führt dies zu einer großen Unübersichtlichkeit durch die Länge (verursacht durch die Verschachtelung). Mit SMC werden benötigte parametrische Regeln ausgewählt, sodass es keine technische Sprache wie in den restlichen Validierungsansätzen der Fall ist, gibt. Durch eine einfache Bedienung der Software durch die Benutzeroberfläche stellt es eine benutzerfreundliche Alternative dar. Die Benutzer benötigen keinerlei Vorwissen über die Struktur und Modellierungssprache der Datenmodelle, da SMC diese intern in einfachen Wörtern konvertiert. Bspw. werden vom IFC-Modell die Klasse IfcWall in SMC zu Wand übersetzt. MvdXML und SHACL legen vergleichsweise deutlich mehr Wert auf die technische Deklaration und die Umsetzung.

2.3 Graphenbasierte Darstellung von BIM-Modellen

2.3.1 Grundsatz und Eigenschaft

Grundsatz und Eigenschaft

Graphen ermöglichen eine effiziente, vereinfachte Darstellung von Beziehungen zwischen einzelnen Objekten und ermöglichen als eine Repräsentationsform eine Beschreibung von relationalen Strukturen. Objekte werden durch Knoten (nodes) dargestellt, wobei die Beziehung zwischen jedem Objektpaar durch eine Kante (edge) veranschaulicht wird (ANGLES et al., 2017).

Graphen finden im Alltag überall ihre Anwendung. Verschiedene Netzwerke wie bspw. soziale-, technologische- oder biologische Netzwerke können als Graphen dargetsellt werden (ANGLES et al., 2017). Graphen, die ein solziales Netzwerk beschreiben, können einzelne Personen bzw. Menschengruppen als Knoten und deren jeweilige Beziehungen als Kanten abbilden. Beispiele dafür wären Freundschaftsnetzwerke oder Familienbäume. In technologsichen Netzwerken, wie Geographische Informationssysteme (GIS), Flugrouten oder Computernetzwerke wird sich mehr auf geografische und raumbezogene Merkmale fokussiert. Graphen in der Bioinformatik werden bspw. in ProteinInteraktionsnetzwerke oder Metabolom-Netzwerke angewendet (ANGLES & GUTIERREZ, 2008; SCHUBERT & BORGWARDT, 2007).

In Abb. 2.13 wird der Graph G durch die Knotenmenge $V(G)$ mit den Knoten v, u, z, w und der Kantenmenge $E(G)$ mit den Kanten $\{vu, vz, vw, zu\}$ beschrieben (ANGLES et al., 2017).

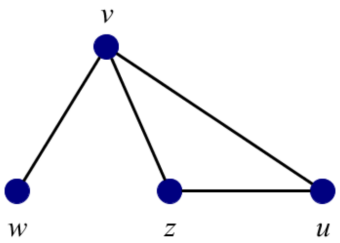

Abbildung 2.13: Darstellung eines ungerichteten Graphens (ISMAIL et al., 2017)

Anders als bei ungerichteten Graphen, in denen die Richtung der Kanten keine wichtige Rolle spielt, wird bei einem gerichteten Graph die Kanten von einem Objekt zum anderen zugewiesen. D.h. die Knoten $u \mapsto v$ können durch die Beziehung (u, v) beschrieben werden, umgekehrt ist dies jedoch nicht möglich (v, u) (ISMAIL et al., 2017).

Graphen können sich in ihrer Struktur jeder Anwendungssituation anpassen und erlauben den Anwender kontinuerlich dem Graphen zusätzliche Informationen hinzuzufügen. Dadurch können komplexe Datennetzstrukturen unter Verwendung eines Graph-Datenmodells flexibel und effizient gespeichert und organisisert werden (ANGLES & GUTIERREZ, 2008).

Im Folgenden werden zwei weltweit verbreitete Graph-Arten, die aus dem einfachen Graphen Abb. 2.13 herausgehen vorgestellt. Diese sind der RDF-Graph und der Property Graph.

Resource Description Framework (RDF)

Der RDF-Graph ist eine W3C Empfehlung zum Austausch semantischer Daten im Web, welches sich nach dem Konzept des edge-labelled Graphen (Abb. 2.15) orientiert (ANGLES et al., 2017; FRANCIS et al., 2018). Bei einem edge-labelled Graphen werden erweiternd zum einfachen Graphen, Kanten mit Etiketten (Labels) beschriftet. Die formale Definition des Konzepts des edge-labelled Graphens ist nach ANGLES et al. (2017) wie folgend definiert:

Definition 2.1 (Edge-labelled graph). An edge-labelled graph G is a pair (V, E), where:
(1) V is a finite set of vertices (or nodes).
(2) E is a finite set of edges; formally, $E \subseteq VxLabxV$ where Lab is a set of labels.

RDF definiert durch die Prinzipien des edge-labelled Graphen sogenannte statements. Rdf-statements beschreiben die Beziehungen der einzelnen Knoten anhand rdf-triples. Ein triple besteht jeweils aus der Reihenfolge von: Subjekt – Prädikat – Objekt (CYGANIAK, WOOD & LANTHALER, 2014).

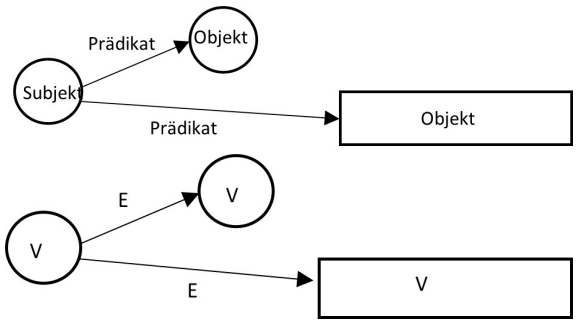

Abbildung 2.14: Zusammenhang des Konzepts eines edge-labelled Graphen (unten) und eines RDF-Triples (oben)

Das Prädikat wird im RDF als eine Kante, die Labels zur Beschreibung des Beziehungstyps aufweist, dargestellt (ANGLES et al., 2017). Damit die Informationen im Web kodiert dargestellt werden können, sind Knoten und Kanten eines RDF-Graphens als Rdf-terms (IRI, Literal und Blank nodes) definiert (CYGANIAK et al., 2014).

Als Beispiel wird in Abb. 2.15 ein RDF-Graph gezeigt. Es beschreibt ein statement einer Wand `wll://wall`, der hier ein Subjekt darstellt. Durch die Kanten werden der Wand wll://wall ein guid-Attributwert und ein Material zugewiesen.

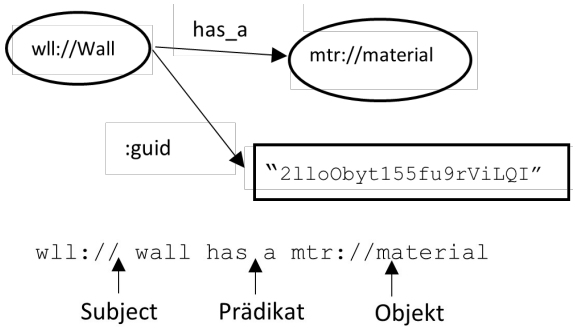

Abbildung 2.15: Beispiel eines RDF-Graphen (oben) und einem RDF-Statement (unten)

V = {wll://Wall, mtr://material}
E = { (wll://wall, has.a, mtr://material), (wll://wall, :guid, "2lloObyt155fu9rViLQIdU"}

Property Graph

Ein weiterer Graph-Modell zur Speicherung von Informationen ist der Property Graph. In einem Property Graph werden ausschließlich Objekte bzw. Klassen und die Beziehung zwischen einzelnen Objekten dargestellt. Bei einem Property Graph wird – anders als bei einem RDF-Graph – nicht nur die Kanten, sondern auch die Knoten mit einem oder mehreren Label beschriftet (ANGLES et al., 2017).

Das Grundprinzip des Property Graphen basiert sich auf das Eigenschafts- und Klassensystem objekorientierte Paradigmen. Knoten stellen im Property Graphen Objekte bzw. Klassen mit den jeweiligen – falls vorhanden – Attribute dar, die als Key-Value-Pair

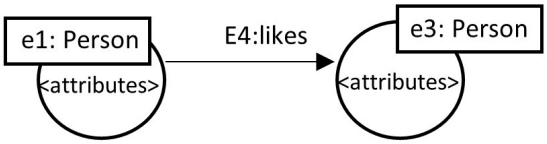

Abbildung 2.16: Darstellung eines Property Graphens

beschrieben werden. Im Gegensatz zu einem RDF-Graphen werden nicht nur die Kanten, sondern auch die Knoten beschriftet (ANGLES et al., 2017).

Durch die abstrakte Modellierungsstruktur des Property Graphens können Vererbungen zwischen einzelnen Klassen ermöglicht werden. Durch dieses Grundkonzept des Property Graphens ergibt sich damit eine gewisse Schnittstelle bzw. Ähnlichkeit mit den objektorientierten Datenmodelle, wie in Kapitel 2.1 beschrieben (ANGLES et al., 2017).

Definition 2.3 (Property graph). A property graph G is a tuple (V, E, ρ, λ, Σ), where:

1. V is a finite set of $vertices$ (or $nodes$).
2. E is a finite set of edges such that V and E have no elements in common.
3. $\rho : E \mapsto (V$ x $V)$ is a total function. Intuitively, $(e) = (v1, v2)$ indicates that e is a directed edge from node $v1$ to node $v2$ in G.
4. $\lambda : (V \cup E) \mapsto Lab$ is a total function with Lab a set of labels. Intuitively, if $v \in V(respectively, e \in E)$ and $(v) = l$ (respectively, $\rho(e) = l$) , then l is the label of node v (respectively, edge e) in G.
5. $\sigma : (V \cup$ E) Prop $\mapsto Val$ is a partial function with Prop a finite set of properties and Val a set of values. Intuitively, if $v \in V$ (respectively, $e \in E$), $p \in$ Prop and $\sigma(v, p) = s$ (respectively, $\sigma(e, p) = s$), then s is the value of property p for node v (respectively, edge e) in the property graph G.

2.3.2 Graphen-Datenmanagement

Grundsatz und Konzept

Im Allgemeinen werden in Datenbanksystemen mit Hilfe eines Datenbankmanagementsystem der vorhandene Datenbestand verwaltet (SAAKE, SATTLER & HEUER, 2018). Innerhalb einer Datenbank bzw. eines Informationssystems können so Datensätze implizit abgespeichert und manipuliert werden (BASTIAN, 2002). Um die impliziten Informationen explizit darzustellen, können Daten durch Abfragen extrahiert werden, die durch Operatoren und Befehlen formuliert werden(BASTIAN, 2002; ISMAIL et al., 2017). Bei Graph-Datenbanken können die Ergebnisse der Abfrage graphisch abgebildet werden.

Basierend auf den oben erwähnten Grundsätzen können in einer Graph-Datenbank Informationen des Graphens gespeichert, manipuliert und abgerufen werden. Als eine weit

Tabelle 2.1: Cypher Syntax (inspiriert von FRANCIS et al. (2018), ISMAIL et al. (2017)

node_pattern ::=	(a:b{c:d})	mit:
rel_pattern ::=	-[a:b{c:d})]- or - [len{c:d}]-	`b` node label
	-[a:b{c:d})]- or - [len{c:d}]-	`a` node identifier
		`b` node label
		`c` property key
		`d` property key
		`len` *, *x, *..x, *x1..x2

verbreitete Open-Source Graph-Datenbank hat sich vor allem Neo4j mit der Abfragesprache Cypher durchgestezt (ANGLES et al., 2017).

Abfragesprache Cypher

Abfragen anhand einer Abfragesprache werden durch bedingte Schlüsselwörter (eng.: key words), anliegend an die Wörter des menschlichen Sprachgebrauchs formuliert, um so die Redundanz zu vermeiden. (BASTIAN, 2002).

Zur Informationsabfrage einer Graph-Datenbank hat sich zunehmend die Abfragesprache Cypher neben der Sprache SPARQL zu einem der Standards in Graph-Datenbanksystemen etabliert. Cypher ist eine deklarative Sprache, d.h. im Fokus steht die Frage, was abgefragt bzw. gesucht werden soll (ANGLES et al., 2017). Der Cypher Syntax basiert sich auf eine für den Menschen einfach lesbarer Grammatik, die ASCII-art verwendet. Dabei werden die Knoten der Graph-Datenbanken mit ihren Beziehungen zu Mustern (patterns) dargestellt (FRANCIS et al., 2018).

In Cypher wird ein Knoten `(a:b)` durch runde Klammern mit `a` node identifier und `b` node label beschrieben werden. Die Kante wird mit eckigen Klammern `[a:b]` dargestellt. Ähnlich zum Knotenelement wird eine Beziehung mit `a` relationship identifier und `b` relationship label bezeichnet. Die Beziehung zwischen Knoten- und Kantenelementen werden durch die Bindestriche zwischen den Elementen `()-[]-()` gekennzeichnet. Eigenschaften zu den jeweiligen Knoten- bzw. Kantenelementen werden durch geschweifte Klammern `c:d` mit `c` property key und `d` property value beschrieben (FRANCIS et al., 2018; ISMAIL et al., 2017).

Innerhalb der Relationsbeschreibung unter Verwendung der eckigen Klammern, können durch * die Anzahl der Relationen eines Pfades bestimmt werden. Mit bspw. *5 wird die Anzahl der Kanten auf 5 beschränkt (FRANCIS et al., 2018).

In Tabelle 2.1 werden die beschriebenen Cypher Syntax dargestellt.

Zusätzlich sind für eine Datenabfrage folgende Klauseln nach NEO4J (2022) von großer Relevanz:
`MATCH`: Anhand des `MATCH`-Schlüsselwortes werden zunächst alle innerhalb der Klausel beschriebenen Elemente wie Knoten, Beziehungen, Label, Eigenschaft oder patterns innerhalb der Datenbank gesucht.

`WHERE(Optional)`: Legt zusätzliche Abfragekriterien, die in der `MATCH`- Klausel nicht definiert wurden, fest. Die `WHERE`-Klausel ist meist durch einen logischen Ausdruck dargestellt.
`RETURN`: Gibt an, welche Elemente zurückgegeben werden sollen. Diese können beispielsweise Knoten, Beziehungen, Eigenschaften, etc. sein. Die Werte, die das Schlüsselwort `RETURN` zurückgeben kann ist abhängig von der Definition der `MATCH` Klasuel.

2.4 Erkannte Forschungslücke

Die vorgestellten Validierungsansätze weisen bereits je nach Anwendungssituation vielfältige Funktionalitäten auf. Einige Beschränkungen ergeben sich jedoch bei der Überprüfung von objektorientierten Zusammenhängen. Mit der Validierung des BIMTesters können solche Strukturen der Modelle sehr gut überprüft werden, jedoch stoßt der BIMTetser bei der Abbildung von komplexeren Datenstrukturen auf seine Grenzen. Eine vielversprechende Validierungsmethode stellt daher die Qualitätskontrolle auf graphenbasierter Repräsentationen dar. Bisher wurde noch keine dergleichen Methode zur automatisierten Prüfung entwickelt, obwohl es ein gewisses Interesse der Abbildung von Datensätze vorhanden ist (bspw. in EXPRESS-G).

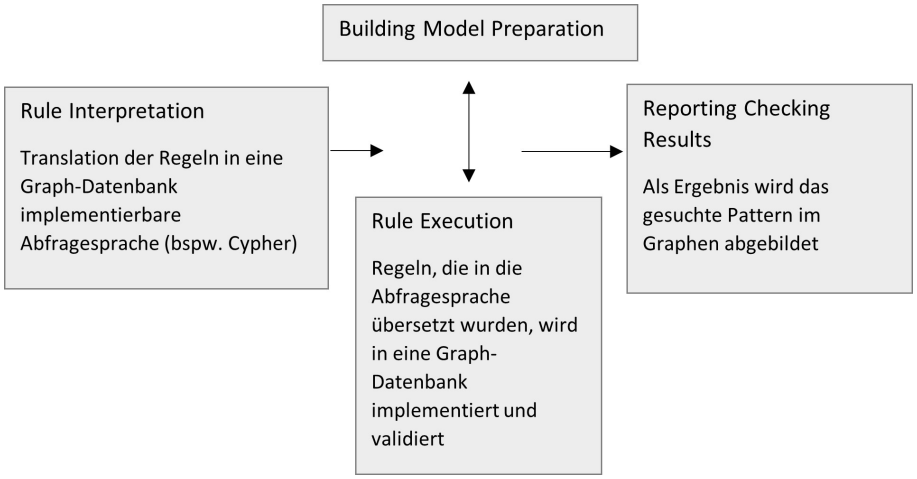

Abbildung 2.17: Die vier Schritte eines Validierungsprozesses (inspiriert durch EASTMAN et al. (2009)

Kapitel 3

Methodik

3.1 Transformation objektorientierter Datenstruktur zu graphenbasierter Repräsentationen

Wie bereits in vorherigen Kapiteln erläutert, folgen gebräuchliche, herstellerneutrale Datenmodelle den objektorientierten Prinzipien zur Darstellung von BIM-Modellen, dessen Darstellung sehr gut als Graphen beschrieben werden können.

In diesem Abschnitt wird ein generischer Ansatz zur Erstellung eines Metagraphenmodells nach ESSER (2021) vorgestellt, indem Datenmodelle - nicht nur für Referenzzwecken (wie in EXPRESS G-Schemata), sondern auch für die Implementierung verschiedener Simulation und Prpzeduren übersetzt werden. Graphenbasierte Modellrepräsentationen, wie sie in dieser Arbeit vorgestellt werden, bestehen zunächst aus Instanz-Klassen und die ihnen jeweils zugehörigen Attribute, welche als Knoten dargestellt werden, als auch aus Beziehungen zwischen zwei Instanzen, die als Kanten abgebildet werden. Der Ansatz wandelt die Datenmodelle zu Graphenrepräsentationen in Form von gerichteten Property Graphs mit beschriftete Knoten und Kanten.

3.1.1 Klassen und Aggregationsklassen

Die einem BIM-Modell zugrundeliegenden Klassen können als drei verschiedene Knotentypen dargestellt werden. Diese sind: Primär-, Sekundär- und Relationsknoten.

1. Primärknoten: Primärknoten im Graphen gehören zu den Grundelemente der Datenstruktur; Diese entsprechen den Klasseninstanzen, die von der Basisklasse erben und durch das Vererbungsprinzip GUID-Identifier besitzen
2. Sekundärknoten: Sekundärknoten sind Instanzen einer Klasse, die nicht von der Basisklasse erben und besitzen so keinen eindeutigen Identifizierer, können aber dennoch von Superklassen geerbte Attribute besitzen.
3. Relationsknoten: Modellieren 1:n oder m:n Beziehungen zwischen Objekten und besitzen eine eindueitge Id.

Zusätzlich werden zu all den drei Knoten ein Attribut `EntityType` am Knoten angegeben. Das Attribut EntityType repräsentiert den Namen der zugehörigen Klasse, die im jeweiligen Datenmodell definiert ist.

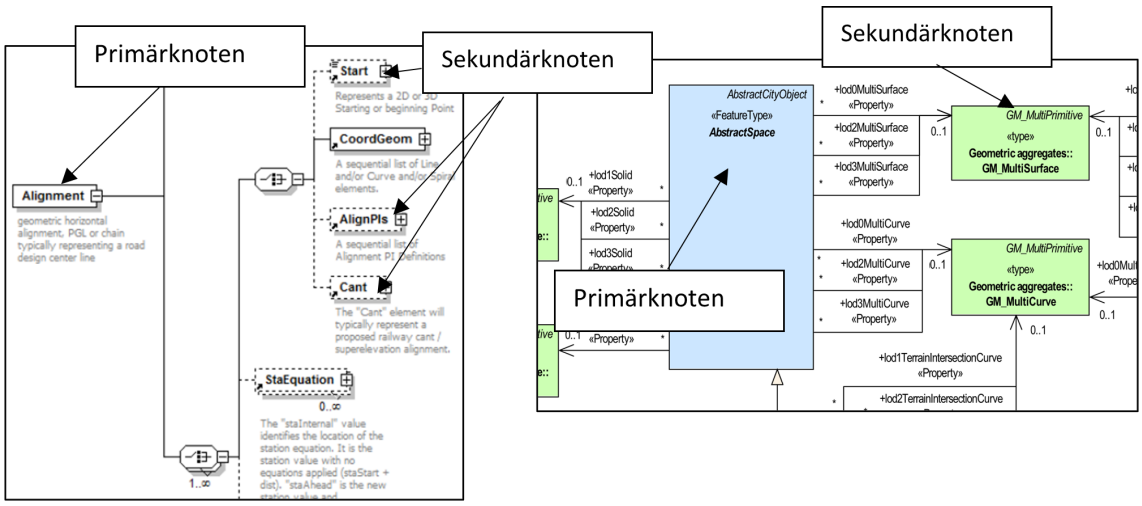

Abbildung 3.1: Abbildung von Primär-, Sekundärknoten in CityGML, Bild: (KUTZNER et al., 2020) und LandXML, Bild:(LANDXML.ORG, o.D.)

3.1.2 Beziehungen und Assoziation

Assoziationen und Referenzen werden durch Kanten zwischen zwei Instanzen dargestellt. Neben den Attributen der jeweiligen Kanten wird im Metamodell des Graphs ein zusätzliches Attribut relType für alle Kanten implementiert. Analog zum Attribut EntityType besitzt relType den Namen der Assoziation nach der Schemaspezifikation.

3.2 IFC-basiertes Metamodell

Für die Generierung von IFC-basierten Graphen wird in dem Ansatz von ESSER (2021) das Instanzmodell in SPF verwendet. Dadurch können im Graphen keine inversen Beziehungen, sowie abstrakte Superklassen angeziegt werden. Jede Entität im IFC-Datenmodell wird in einem Knoten umgewandelt. Knoten beinhalten jegliche IFC-Klasseninstanzen mit den zugehörigen Attribute, die im IFC-Datenmodell innerhalb des EXPRESS Schemas spezifiziert wurde. Dabei werden jedem Knoten neben dem Attribut `EntityType` das Attribut `p21.id` angehängt, welches die Objektreihenfolge des IFC-Instanzmodells wiedergibt. Im Gegensatz zu Attributen mit einfachem Datentyp wie int, double, string oder boolean, werden Assoziationen zu anderen Klassen durch Kanten implementiert.

Sekundärknoten entsprechen im IFC-Datenmodell überwiegend alle Klasseninstanzen des Resource Layers, wie `IfcMaterial, IfcGeometricShapeContext`, etc.

Verbindungsknoten können im IFC mit Aggregationsklassen gleichgestellt werden. Diese sind bspw. `IfcRelAggregatesMaterial`, etc.

3.3 Musterabgleich auf Property Graph

Zum Grundkonzept einer Graphenabfrage gehört die Abbildung von Mustern, die auf einem gegebenen Graphen gesucht werden sollen (FRANCIS et al., 2018). Im Folgenden werden verschiedene Muster vorgestellt, die bei einem Musterabgleich auf Property Graphen von großer Relevanz sind.

Für die Mustersuche eines Knotens mit einem bestimmten Attribut und einem Label im Property Graph, wird mit dem Token `MATCH` das zu suchende Muster folgend eingeleitet:

Tabelle 3.1: Cypher-Abfrage zur Rückgabe eines Knotens mit bestimmter Eigenschaft

Suche einen Knoten a mit einem Attribut b. Gebe diesen Knoten zurück.	MATCH(a{EntityType:xx}) WHERE a.b= c RETURN a

Für die Suche einer bestimmten Relation von Entitäten (bzw. Knoten) kann das Muster `(a)-(b)` verwendet werden. Optional wird für eine Konkretisierung des Relationstyps das Muster `(a)-[r]-(g)-[r]-(d)` in der MATCH-Klausel angewendet. Die Formel sieht folgend aus:

Tabelle 3.2: Cypher-Abfrage zur Darstellung eines Zusammenhanges von Knoten und Beziehungen

Zeige eine Relation, in welcher die Knoten a mit einem Knoten d verbunden sind.	MATCH(a{EntityType:xx})-(g)-(d) RETURN a,g,d

Nimmt man bspw. den step der Feature-Datei Projectsetup `* The project must have coordinate transformations to convert from local to global coordinates` in Kontext mit der zugehörigen step-Definition, ergibt sich folgende Anforderung:

IfcProject must contain IfcGeometricRepresentationContext with the attribute ContextType: Model, ContextIdentifier: Body und TargetView: MODEL VIEW.

Darauf aufbauend wird folgende Cypher-Formel vom BIMTester abgeleitet.

```
1 MATCH (a{EntityType:'IfcProject'})-[*]-(b{EntityType:'IfcMapConversion'})
2 WHERE b.ContextType = Model
3 AND b.ContextIdentifier = 'Body'
4 AND b.TargetView = 'MODEL VIEW'
5 RETURN a
```

Algorithmus 3.1: Cypher-Algorithmus zur Transformation der lokalen Koordinaten zu globalen

Soll eine Anzahl des gesuchten Musters als Ergebnis zurückgegeben werden, wird folgende Cypher Formel im Tabelle 3.3 verwendet. Um zu prüfen, ob eine Musteranzahl kleiner oder größer als eine bestimmten Zahl ist, kann die Formel in Tabelle 3.4 angewendet werden. Diese Formel gibt eine boolesche Antwort zurück. Boolesche Rückgaben können neben der Einführung von logischen Operatoren durch die Zeichenkette IS NOT

NULL erzielt werden. Hier kann überprüft werden, ob ein bestimmtes Muster im Graphen vorhanden ist (Tabelle 3.5).

Tabelle 3.3: Abfrage zur Überprüfung der Knotenanzahl

Zähle alle Knoten b	MATCH(b) RETURN count(b)

Tabelle 3.4: Cypher-Abfrage zur Überprüfung der minimalen Knotenanzahl

Anzahl der Knoten b soll größer als 1 betragen	MATCH(b) RETURN count(b)>1

Tabelle 3.5: Cypher-Befehl für die Überprüfung der Existenz eines Knotens mit bestimmter Eigenschaft

Anzahl der Knoten b soll größer als 1 betragen	MATCH (a)-(g)-(d) RETURN d.e = f IS NOT NULL

So wird bspw. für den Gherkin Syntax:

```
* There must be at least one "IfcSite" element
```

des Szenarios `Geometry is georeferenced to a coordinate reference system` von der Featuredatei Geolocation mit der Cypher-Formel in Algorithmus 3.2 folgend ausgedrückt:

```
1 MATCH (n{EntityType:'IfcSite' )
2 RETURN count(n) >=1
```

Algorithmus 3.2: Cypher-Abfrage zur Überprüfung der Anzahl der Knoten mit dem EntityType `IfcSite`

In dieser step-Anforderung wird geprüft, ob das Modell mindestens eine IfcSite-Klasse besitzt. Hier wird ein boolesches Ergebnis zurrückgegeben.

Kapitel 4

Case Study

In der Case Study wird der Validierungsprozess des Property Graphens eines Beispielmodells auf seine Funktionalität und Praktikabilität untersucht. Dazu wird die technische Umsetzung der Graph-Validierung anhand verschiedener Kriterien mit den vorgestellten Software-applikationen BIMTester und Solibri Model Checker verglichen. Im Folgenden soll anhand vier Fallbeispiele die jeweilige Validierungsfähigkeit der einzelnen Validierungsansätze hervorgehoben werden.

4.1 Aufbau des Experiments

Für die Case Study wurde ein BIM-Modell mit der CAD-Software Revit 2023 erstellt (Abb. 4.1) und in eine IFC-Datei der Version IFC 4 konvertiert. Die Translation des IFC-Datenmodells in den Graphen erfolgte durch einen Python Code, der vom Lehrstuhl für Computergestützte Modellierung und Simulation zur Verfügung gestellt wurde.

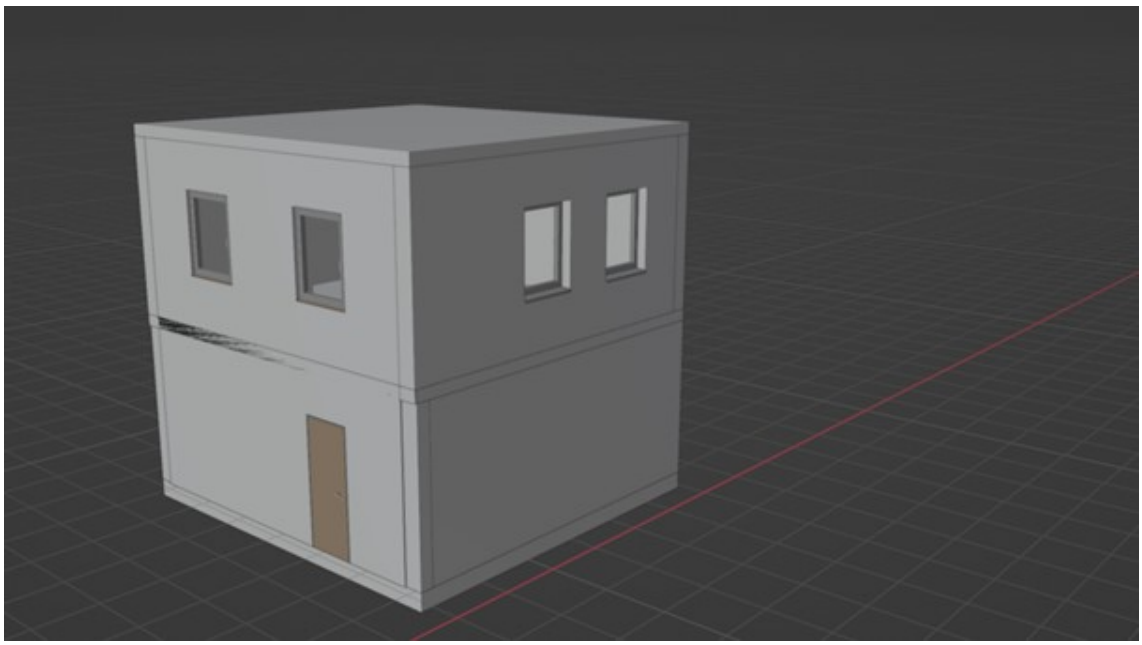

Abbildung 4.1: Beispielmodell VillaKunterbunt

Programminformationen:

1. Für die Graph-Datenbank wurde die Datenbank Neo4j in der Version 4.4.3 verwendet.

2. Der BIMTester wurde als CLI in der Standalone Version verwendet und wird gegen die Feature-Dateien basierend auf den MicroMVDs überprüft. Die Ausführung des BIMTetsers erfolgte durch PyCharm und eine Open-Source Paketdistribution Anaconda 3.

3. Die Validierung mit der Software Solibri Model Checker wurde unter Verwendung von Solibri Office mit der Build-Nummer 9.12.9.15 ausgeführt.

4.2 Vergleich der Validierungsansätze

4.2.1 Fall 1: Attribute und Eigenschaften

Die Sicherstellung der Gültigkeit von Attributen und Eigenschaften eines Objektelements setzt ein grundlegendes Fundament in der BIM-Modellierung dar. Zusätzlich wird die Existenz der Objektattribute und Eigenschaften geprüft.

BIMTester

Für die Validierung von Objektattributen und Eigenschaften wird in diesem Abschnitt zunächst der step:

`*The project must have an identifier of "guid"`. aus der MicroMVD-Datei Project-Setup betrachtet. Die zugehörige step-Definition wird in Algorithmus 4.1 abgebildet. Die Regel-Implementierung zur Validierung von Objektattributen kann in Python hauptsächlich mit der Funktion util.assert_attribute(x) ausgeführt werden. Diese Funktion gleicht das Attribut einer Ifc-Klasse mit dem vom Benutzer eingegebenen Attributwert ab und überprüft die Übereinstimmung beider Werte. In Algorithmus 4.1 wird das Attribut `GlobalId` von der Klasse `IfcProject` überprüft.

```
1 @step('The project must have an identifier of "{guid}"')
2 def step_impl(context, guid):
3     util.assert_attribute(IfcStore.file.by_type("IfcProject")[0], "GlobalId", guid)
```

Algorithmus 4.1: Step-Definition zur Überprüfung des Attributs GlobalId der Klasse IfcProject

Property Graph

Für die Überprüfung des Attributs `GlobalId` der Klasse `IfcProject` wird im Property Graph zunächst folgende Cypher-Abfrage verwendet (Algorithmus 4.2).

Für die Anwendung der Cypher-Formel auf den Graphen des Beispielmodells wird der Knoten der Klasse `IfcProject` mit dem Attribut `GlobalId` und dem Attributwert `3p4kxXAWH3PhSKz7mMWxUo` gesucht (Algorithmus 4.3). Optional kann durch Verwendung

```
MATCH (n{EntityType:'IfcProject'})
WHERE n.GlobalId = "guid"
RETURN n
```

Algorithmus 4.2: Cypher-Abfrage zur Überprüfung der `GlobalId` von `IfcProject`

von IS NOT NULL in der RETURN-Klasusel ein boolesches Ergebnis (true oder false) erzielt werden.

```
MATCH (n{EntityType:'IfcProject'})
WHERE n.GlobalId = '3p4kxXAWH3PhSKz7mMWxUo'
RETURN n
```

Algorithmus 4.3: Cypher-Befehl zur Überprüfung der `GloablId`

Für die Validierung eines Knoten, dessen Ifc-Klasse bzw. EntityType nicht definiert ist, kann direkt nach dem EntityTyp des Knotens mit dem bestimmten Attributwert gesucht werden.

```
MATCH (n{ GlobalId: '3p4kxXAWH3PhSKz7mMWxUo'})
RETURN n.EntityType
```

Algorithmus 4.4: Abfrage zur Bestimmung des EntityTyps eines Knotens mit dem gegebenen Attribut und Attributwert

Solibri Model Checker

Anforderungen bzgl. Eigenschaften und Attributen können mit der Solibri Regel SOL/230 Eigenschaftsregel mit Komponentenfiltern über manuelle Eingabe von Parametern im Ruleset Manager definiert werden. Innerhalb der Regel wird zwischen folgenden Tabellen differenziert.

1. Zu überprüfenden Komponenten: Die zur Validierung benötigte Komponente werden gefiltert
2. Anforderungen: Überprüfende Anforderungen der Eigenschaft und Eigenschaftswerte wird festgelegt.

In Abb. 4.2 wird die Komponente Projekt über die Anforderungen der Eigenschaftswerte GUID, Name und Beschreibung validiert.

In SMC ist eine vollständige Überprüfung der Eigenschaften begrenzt. Wird bspw. das Attribut `Land_title_number` von der Klasse `IfcSite` überprüft, so kann festgestellt werden, dass diese Eigenschaft in SMC nicht auswählbar ist.

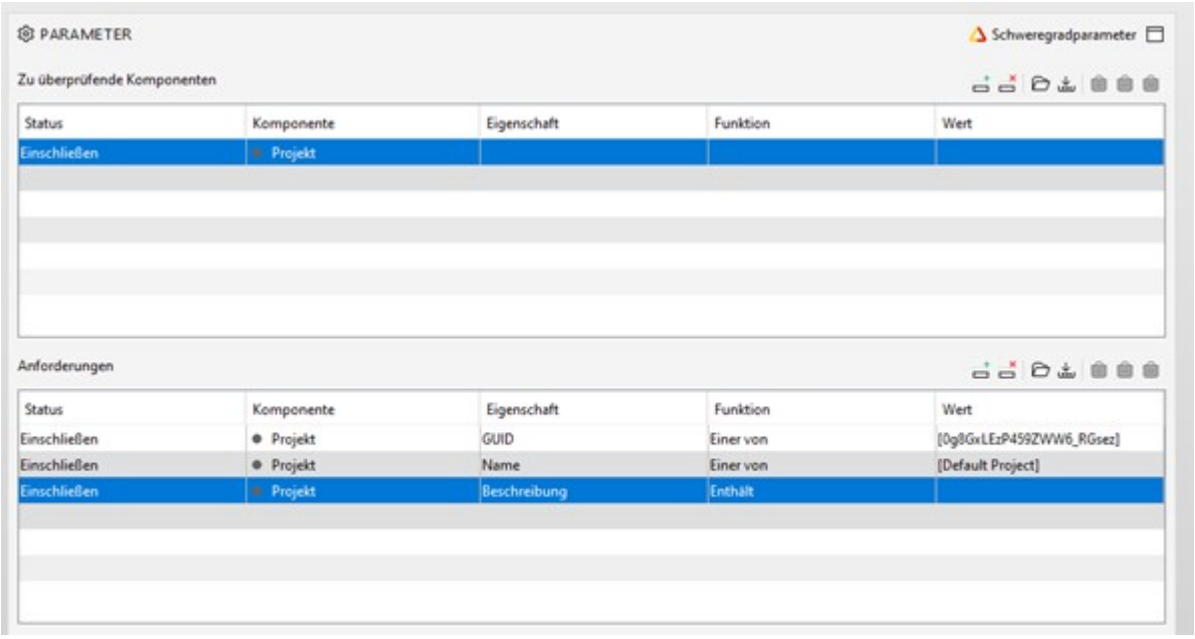

Abbildung 4.2: Parameterbestimmung der Regel SOL 230 im Parameter View

4.2.2 Fall 2: Property Sets und Quantity Sets

Analog zur Überprüfung von Attributen ist die Gewährleistung anwendungsspezifischer Eigenschaften durch Darstellung eines Property Sets bzw. Quantity Sets essenziell für eine valide Qualitätskontrolle. In diesem Abschnitt wird das Vorhandensein der Eigenschaften der Property Sets `Pset_WallCommon` und `Pset_ConcreteGeneralElement` validiert.

BIMTester

Für das Fallbeispiel wird ein Teil der Feature-Datei Quantity take-off verwendet, die folgende Szenarien und steps beinhaltet:

```
1 Scenario: All walls must have cost-significant metadata
2 * All walls must have a LoadBearing property
3 * All walls must have an AcousticRating property
4 * All walls must have a FireRating property
5
6 Scenario: All concrete elements are identifiable and contain data
7 * All concrete elements must be assigned to a concrete material
8 * All concrete elements must have their strength class assigned with one of the following
      values: |Value|, |N|, |S|, |{value}|
```

Algorithmus 4.5: Szenarien und steps einer Quantity takeoff MicroMVD

Im ersten Szenario werden steps ausgeführt, die überprüfen, ob alle Wände eine bestimmte Eigenschaft (hier: `LoadBearing, AcousticRating und FireRating`) vom Property Set `Pset_WallCommon` besitzen. Im zweiten Szanrio `All concrete elements are identifiable and contain data` werden alle Beton-Elemente geprüft, ob sie aus dem Material Beton bestehen. Im darauffolgenden Schritt wird die Richtigkeit des Eigenschaftswertes `Strength class` des Property Sets `Pset_ConcreteElementGeneral` überprüft.

```
MATCH (i{EntityType:'IfcWall'})
MATCH (m{EntityType:'IfcWall'})--(n{EntityType:'IfcRelDefinesByProperties'})--(b{EntityType:'IfcPropertySet'})--
(u{EntityType:'IfcPropertySingleValue'})
WHERE b.Name='Pset_WallCommon' AND u.Name='LoadBearing' AND u.NominalValue='IfcLabel(False)'
RETURN  m.GlobalId, count(i) AS All_walls
```

	m.GlobalId	All_walls
1	"2kn5jwy31Fsxqrv$tc_Ibo"	9
2	"1G9otslDDFofM_erGnGkF$"	9
3	"1G9otslDDFofM_erGnGjnd"	9
4	"1G9otslDDFofM_erGnGjna"	9

Abbildung 4.3: Output der Überprüfung von der Existenz der Eigenschaft LoadBearing

Es ist anzumerken, dass die Python-Step-Implementierung der MicroMVD Quantity takeoff nach aktuellem Stand noch nicht vollständig implementiert ist.

Property Graph

Die Abfrage für die Anforderung:

`* All walls must have a LoadBearing property`

kann durch die Cypher-Abfrage in Algorithmus 4.6 auf indirekter Weise überprüft werden. Hier werden zum einen alle `GlobalIds` der Wände innerhalb des Graphens aufgelistet und die Zahl aller Wände in der nächsten Spalte abgebildet. Unter Modifizierung dieses Befehls von `u.NominalValue='IfcLabel(True)'` zu `u.NominalValue='IfcLabel(False)'` kann aus Abb. 4.3 entnommen werden, dass im Beispielmodell keine Wand die Eigenschaft `LoadBearing` aufweist.

```
1 MATCH (i{EntityType:'IfcWall'})
2 MATCH (m{EntityType:'IfcWall'})--(n{EntityType:'IfcRelDefinesByProperties'})--(b{
      EntityType:'IfcPropertySet'})--(u{EntityType:'IfcPropertySingleValue'})
3 WHERE b.Name='Pset_WallCommon' AND u.Name='LoadBearing' AND u.NominalValue='IfcLabel(True)
      '
4 RETURN m.GlobalId, count(i) AS All_walls
5
```

Algorithmus 4.6: Cypher-Abfrage für den steps `All walls must have a LoadBearing property`

Optional kann der Validierungsprozess in zwei separaten Schritten erfolgen. In einem ersten Schritt wird dafür die Anzahl aller Wände, im zweiten die Anzahl der Wände mit den erforderlichen Eigenschaften gezählt. Hier muss der Benutzer das Ergebnis vergleichen und bewerten.

Bei dieser Anforderung muss zusätzlich beachtet werden, dass Wandelemente im IFC-Modell in verschiedene Klassen, wie bspw. `IfcWall` oder `IfcWallStandardCase` beschrieben werden können. Im Beispielmodell ist der Projektumfang gering, sodass erkannt

werden kann, dass die Wände ausschließlich der Klasse `IfcWall` angehören. Dies kann bei größeren und komplexeren Projekten ein Problem darstellen.

```
MATCH (m{EntityType:'IfcWall'})
RETURN count(m)

MATCH (m{EntityType:'IfcWall'})--(n{EntityType:'IfcRelDefinesByProperties'})--(b{
    EntityType:'IfcPropertySet'})--(u{EntityType:'IfcPropertySingleValue'})
WHERE b.Name='Pset_WallCommon' AND u.Name='LoadBearing' AND u.NominalValue='IfcLabel(True)
    '
RETURN count(m)

```

Algorithmus 4.7: Zwei-schrittiger Validierungsprozess zur Überprüfung des steps `* All walls must have a LoadBearing property`

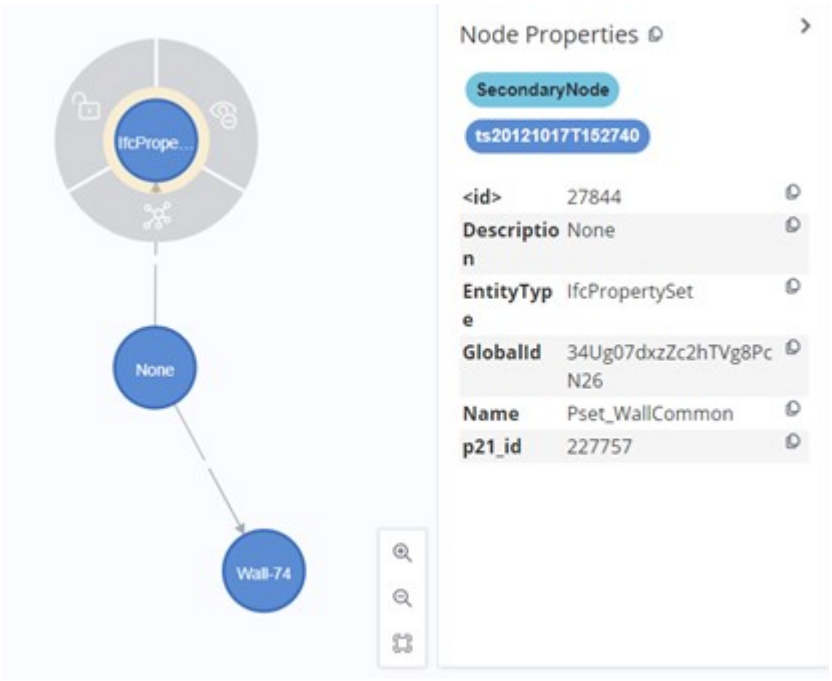

Abbildung 4.4: Darstellung des Proeprty Sets `Pset_WallCommon` im Property Graph

Für die Überprüfung des folgenden steps `* All concrete elements must be assigned to a concrete material` kann die Abfrage wie folgt ausgeführt werden (Algorithmus 4.9). In der `RETURN`-Klausel werden alle Elemente des Modells, die diese Materialeigenschaft aufwesien aufgelistet.

```
MATCH (n) - [g:rel]-(k{EntityType:'IfcRelAssociatesMaterial'})-[z:rel]-(q{EntityType:'
    IfcMaterial'})
WHERE q.Category='Concrete'
RETURN n.EntityType, n.GlobalId, count(n)

```

Das zu überprüfende Objekt `All concrete elements` wird im BIMTester nicht weiter spezifziert, sodass das Material eines unbestimmten Knoten n unter Verwendung des Beziehungsobjekts `IfcRelAssociatesMaterial` bestimmt wird.

Die Anforderung:

`* All concrete elements must have their strength class assigned with one of the following values:|Value|, |N|, |S|, |value|.`

kann ähnlich zu Algorithmus 4.7, durch eine zwei-schrittige Validierung durchgeführt werden, indem im ersten die Anzahl aller Elemente mit einem Material `Concrete` gezählt werden, und im nächsten Schritt die bestimmten Eigenschaften des `IfcPropertySingleValue` des Property Sets `Pset_ConcreteGeneralElement` aufweisen.

```
1
2 MATCH (n) - [g:rel]-(k{EntityType:'IfcRelAssociatesMaterial'})-[z:rel]-(q{EntityType:'
      IfcMaterial'})
3 WHERE q.Name='Ortbeton - bewehrt Verputzt'
4 RETURN count(n)
5
6 MATCH (o{EntityType:'IfcMaterial'})--(u{EntityType:'IfcRelAssociatesMaterial'})--(a)--(b{
      EntityType:'IfcRelDefinesByProperties'})--(n{EntityType:'IfcPropertySet'})--(v{
      EntityType:'IfcPropertySingleValue',Name:'ConstructionMethod'})
7 WHERE v. NominalValue = IfcLabel(In-situ) AND o.Name='Ortbeton - bewehrt Verputzt'
8 RETURN count(a)
9
```

Algorithmus 4.9: Cypher-Funktion zur Überprüfung der richtigen Materialzugehörigkeit und der richtigen Materialeigenschaft

Solibri Model Checker

Die Eigenschaftssätze des Pset_WallCommon im Solibri Model Checker können, analog im Fall 1 mit der Solibri Regel SOL 230.1.6 validiert werden.

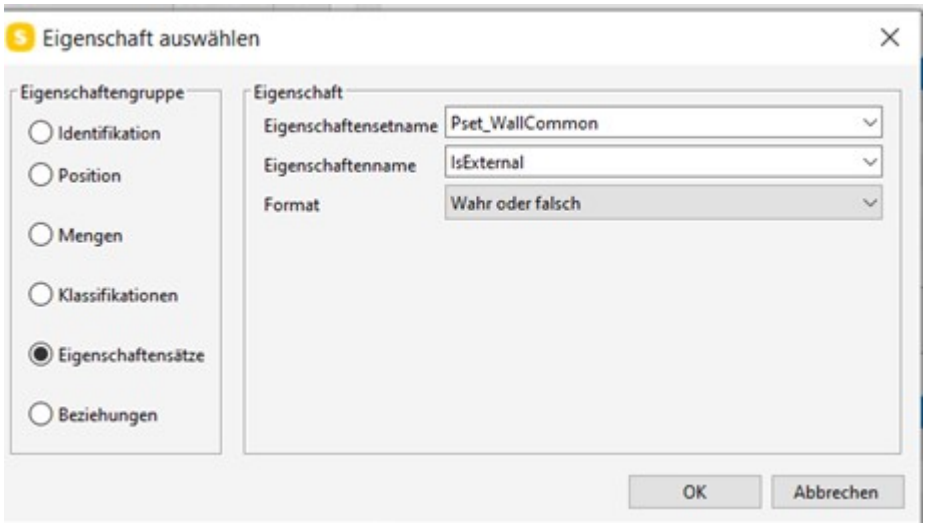

Abbildung 4.5: Eigenschaftsauswahl in Solibri

Für die Validierung von mehreren Eigenschaften eines Objektes ist ein weiterer Filter anzulegen. Abb. 4.6. Hier wird das Objekt `All concrete elements` als eine beliebige Komponente dargestellt.

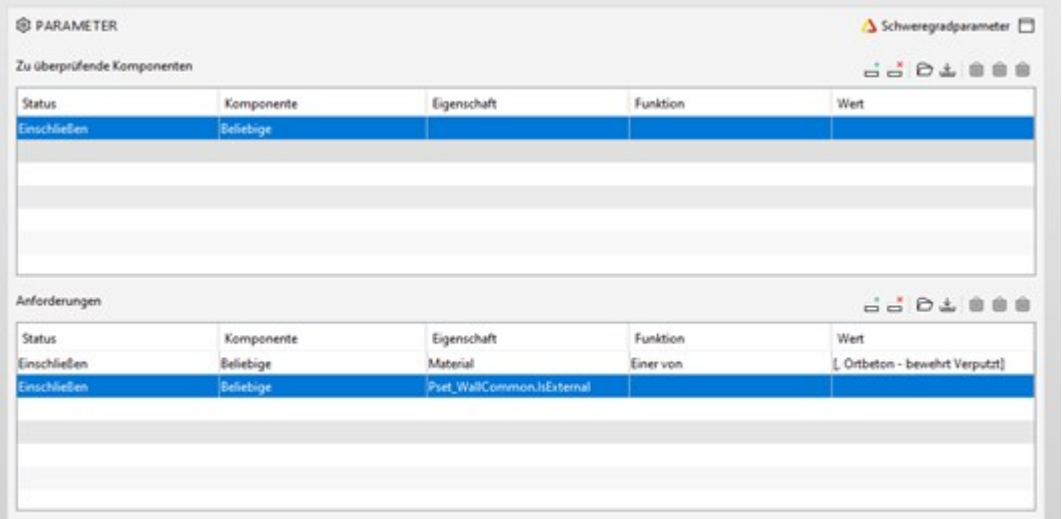

Abbildung 4.6: Bestimmung mehrerer Eigenschaften in SOL 230

4.2.3 Fall 3: Bestimmung von Geometrie

Jedem physischen Objekt eines BIM-Modells kann eine geometrische Repräsentation hinzugefügt werden. Es ist von großer Bedeutung das Vorhandensein von geometrischen Objektformen zu überprüfen und zu kontrollieren, ob diese Formen effizient gestaltet sind.

BIMTester

Im BIMTester kann durch die Anforderung `*All elements must be under {number} polygon` in der Feature-Datei Geometric Detail auf eine effiziente Geometrie-Modellierung geprüft werden. Dazu wird die Anzahl der Polygone der gegebenen Formen gezählt und geprüft. Die step-Definition ist in Algorithmus 4.10 dargestellt.

```
@step('All elements must be under "{number}" polygons')
def step_impl(context, number):
    number = int(number)
    errors = []
    for element in IfcStore.file.by_type("IfcElement"):
        if not element.Representation:
            continue
        total_polygons = 0
        tree = IfcStore.file.traverse(element.Representation)
        for e in tree:
            if e.is_a("IfcFace"):
                total_polygons += 1
            elif e.is_a("IfcPolygonalFaceSet"):
                total_polygons += len(e.Faces)
            elif e.is_a("IfcTriangulatedFaceSet"):
                total_polygons += len(e.CoordIndex)
        if total_polygons > number:
            errors.append((total_polygons, element))
    if errors:
        message = "The following {} elements are over 500 polygons:\n".format(len(errors))
        for error in errors:
            message += "Polygons: {} - {}\n".format(error[0], error[1])
        assert False, message
```

In der Python-Step Implementierung wird die Validierung auf Instanzen der Ifc-Klasse `IfcElement` beschränkt. Der Zeile 6 prüft zunächst alle Instanzen des `IfcElement` auf das Vorhandensein des Attributs `Representation`. Falls dieser nicht existiert, sind keine Polygone zur Beschriebung der Flächen vorhanden.

Ab Zeile 9 wird mit dem Python tree traversal die Anzahl von `IfcFace` der jeweiligen Instanz der Klasse `IfcElement` gezählt. Bei `IfcPolygonalFaceSet` und `IfcTriangulatedFaceSet` werden dabei nicht die Klasse selbst aufgezählt, sondern die von der Klasse ausgehenden Relationen `Faces` und `CoordIndex`. Anschließend werden die innerhalb der Python-Funktion berechneten Ergebnisse mit der Nummer der maximal erlaubten Anzahl der Polygone in Zeile 17 abgeglichen .

Für die Überprüfung der Existenz von 3D-Geometri-Formen werden im BIMTester folgende Szenarien und steps verwendet.

```
Scenario: Project geometry is stored

*The project must contain 3D geometry representing the shape of objects
```

Die zugehörige Python-Implementierung ist in Algorithmus 4.11 abgebildet. In der Python Funktion get_subcontext (x) werden Bedingungen für Instanzen der Klasse `IfcProject` gestellt. `IfcProject` soll eine Subklasse `IfcGeometricSubcontext` besitzen, die die jeweiligen Attribute `ContextIdentifier`, `ContextType` und `TargetView` enthält. Das Vorhandensein der Objektformen wird im BIMTester erst dann bewiesen, wenn `IfcGeometricSubcontext` anschließend den `ContextIdentifier Body`, den `ContextType Model` und das `TargetView` mit dem Wert `MODEL VIEW` aufweist.

```
1 @step("The project must contain 3D geometry representing the shape of objects")
2 def step_impl(context):
3     assert get_subcontext("Body", "Model", "MODEL_VIEW")
4
```

Algorithmus 4.11: Step-Defintition der `The project must contain 3D geometry representing the shape of objects`

```
1  def get_subcontext(identifier, type, target_view):
2      project = IfcStore.file.by_type("IfcProject")[0]
3      for rep_context in project.RepresentationContexts:
4          for subcontext in rep_context.HasSubContexts:
5              if (
6                  subcontext.ContextIdentifier == identifier
7                  and subcontext.ContextType == type
8                  and subcontext.TargetView == target_view
9              ):
10                 return True
11     assert False, "The subcontext with identifier {}, type {}, and target view {} could
           not be found".format(
12         identifier, type, target_view
13     )
14
15
```

Algorithmus 4.12: Definition einer Python-Funktion get_sunbcontext(x,y)

Property Graph

Im Property Graph kann die Geometric detail-Datei einige Schwierigkeiten aufweisen. Die Aufzählung der Klassen `IfcFace`, `IfcPolygonalFaceSet` und `IfctriangulatedFaceSet` kann theoretisch durch die drei separaten Cypher-Formel in Algorithmus 4.13 dargestellt werden.

```
MATCH (n:PrimaryNode)-[*]->( z{EntityType:'IfcFace'})
WHERE n.ObjectType IS NOT NULL
RETURN n.EntityType, n.GlobalId, count(z)

MATCH (n:PrimaryNode)-[*]->( z{EntityType:'IfcPolygonalFaceSet'})-[r{rel_type:'Faces'}]-()
WHERE n.ObjectType IS NOT NULL
RETURN n.EntityType, n.GlobalId, count(r)

MATCH (n:PrimaryNode)-[*]->( z{EntityType:'IfcTriangulatedFaceSet'})-[r{rel_type:'
    CoordIndex'}]-()
WHERE n.ObjectType IS NOT NULL
RETURN n.EntityType, n.GlobalId, count(r)


```

Algorithmus 4.13: Abfrage zur Abzählung der Polygone

Im Beispielmodell sind alle Polygone durch die Klasse `IfcPolygonalFaceSet` beschrieben, sodass folgende Cypher-Abfrage in Algorithmus 4.14 angewendet wird. In diesem Befehl soll die maximale Polygonzahl nicht mehr als 50 betragen. In der Abfrage ist es zu beachten, dass dem Knoten das Attribut `ObjectType` zugeordnet wird, da ansonsten die Instanzen der Superklasse `IfcElementType` mitgefiltert werden. Das somit erzielte Ergebnis wird in Abb. 4.7 abgebildet.

```
MATCH (n)-[*]->
(r{EntityType:'IfcPolygonalFaceSet'})-[ü{rel_type:'Faces'}]-()
WHERE n.ObjectType IS NOT NULL
RETURN n.GlobalId, n.EntityType, count(ü)<50

```

Algorithmus 4.14: Überprüfung der Anzahl vom Relationstyp `Faces`

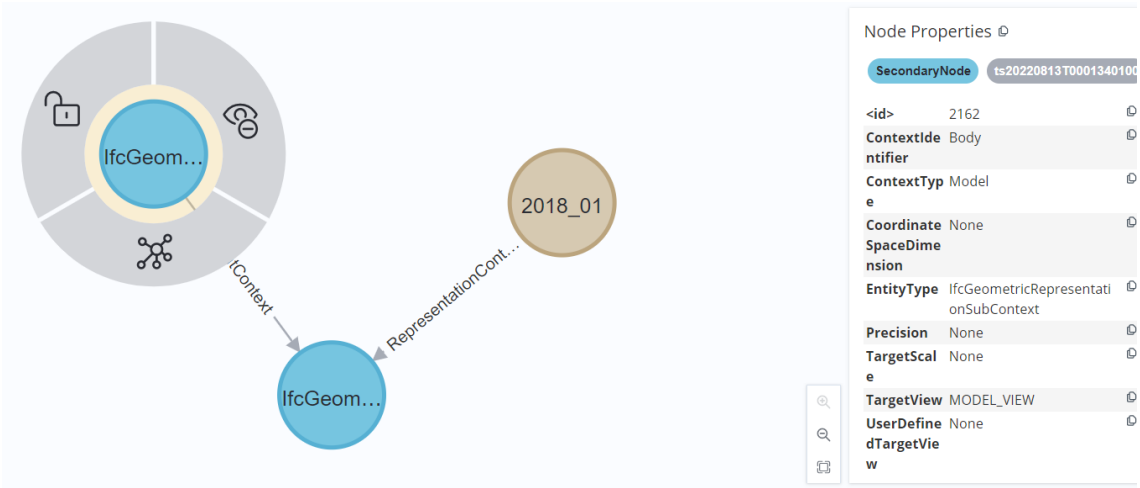

Abbildung 4.8: Abbildung des Knoten `IfcGeometricRepresentationSubContext`

```
MATCH (n)-[*]→
(r{EntityType:'IfcPolygonalFaceSet'})-[ü{rel_type:'Faces'}]-()
WHERE n.ObjectType IS NOT NULL
RETURN n.GlobalId, n.EntityType, count(ü)<50
```

	n.GlobalId	n.EntityType	count(ü)<50
1	"2kn5jwy31Fsxqrv_xc_H7K"	"IfcOpeningElement"	true
2	"2kn5jwy31Fsxqrv_xc_H4A"	"IfcOpeningElement"	true
3	"2kn5jwy31Fsxqrv_xc_It6"	"IfcOpeningElement"	true
4	"2kn5jwy31Fsxqrv_xc_ImN"	"IfcOpeningElement"	true
5	"2kn5jwy31Fsxqrv$tc_ImN"	"IfcWindow"	false
6	"2kn5jwy31Fsxqrv$tc_It6"	"IfcWindow"	false
7			

Abbildung 4.7: Cypher-Abfrage zur Validierung der Polygonzahl

Die zweite Anforderung, die in dieser Arbeit überprüft wird, ist die Sicherstellung von 3D-Objektformen im Modell. Dafür wird in Neo4j folgende Abfrage getätigt (Algorithmus 4.15). Das Vorhandensein der Subklasse `IfcGeometricSubContext` mit den Attributen `Contexttype`, `TargetView` und `ContextIdentifier` wird hier überprüft. Wie in Abb. 4.8 dargestellt, bezieht sich `IfcProject` mit dem Beziehungsattribut `RepresentationContext` auf die Klasse `IfcGeometricContext`, die sich durch `ParentContext` aud `IfcGeometricSubContext` assoziiert.

```
1 MATCH (d{EntityType:'IfcGeometricRepresentationSubContext',
2 ContextType:'Model',ContextIdentifier:'Body', TargetView:'MODEL_VIEW'})--(n{EntityType:'
      IfcGeometricRepresentationContext'}) - [b:rel]-
3 (c{EntityType:'IfcProject'})
4 RETURN n,b,c,d
```

Algorithmus 4.15: Cypher-Abfrage zur Bestimmung von 3D-Objektformen

Solibri Model Checker

Für eine Prüfung der Geometrie von Modellelementen können bspw. folgende Solibri Regeln SOL/167/2.2 und SOL/216 verwendet werden. Mit der Solibri Regel SOL/167/2.2 wird die Modellstruktur überprüft. Mit der Regel kann die Geometrie im Modell unter `"Maximale Polygonanzahl"` die maximale Polygonzahl von den jeweiligen Objekt-Geometrien überprüft werden.

Im SMC kann zudem die Geometrie der Modellelemente mit fünf vordefinierten Geometrietypen (Extrusion, Festkörper, Darstellung der Begrenzungen, Begrenzungsrahmen und Beliebige) beschrieben werden.

Durch die Modifizierung der Regel SOL/167/2.2 Modellstruktur im Ruleset Manager kann überprüft werden, ob das Gelände eine Geometrie aufweist.

Mit der Solibri-Regel 216 kann man zusätzlich zusammen mit der Wanddimensionierung, auch die Wandgeometrie überprüfen. Zusätzlich ist es möglich eine bestimmte Extrusionsrichtung zu bestimmen.

4.2.4 Fall 4: Lokalisierung

Die Ortsbestimmung (bzw. Lokalisation) der Modelle kann eine wichtige Grundlage im Kontext der Integration von BIM- und GIS-Datensätzen darstellen. Um eine reibungslose Interoperabilität sicherzustellen, ist es von großer Wichtigkeit die Geolokalisierungsdaten des Modells zu prüfen. Hier wird anhand der MicroMVD Geolocation die technischen Gegebenheiten der Kodierung mit mathematischer Funktionen in den Fokus gestellt.

BIMTester

Folgender step `* The model must be rotated clockwise by "number"to derive its global coordinates` gibt die Anforderung an, dass das Modell um einen bestimmten Faktor gedreht werden soll, damit das lokale Koordinatensystem relativ zum globalen System angegeben wird. Die zugehörige Python-Implementierung wird folgend dargestellt (Algorithmus 4.16).

Als Vorbedingung der Evaluation des Ergebnisses wird zunächst mit der Funktion check_ifc4_geolocation(x) (Algorithmus 4.17) die Richtigkeit und das Vorhandensein der Attribute `XAxisAbscissa` und `XAxisOrdinate` überpüft. Die Funktion für check_ifc4_geolocation(`"IfcMapConversion "`, `"XAxisOrdinate"`, should_assert=False) gibt an, dass `ifcMapConversion` eine Referenz zu `IfcGeometricRepresentationContext` besitzt, dessen Attribute `HasCoordinateOperation` und `ContextType` mit dem Attributswert `Model` ist. Anschließend wird das Attribut des `XAxisAbscissa` und `XAxisOrdinate` der Klasse `IfcMapConversion` auf ihre Existenz geprüft.

```
@step(u'The model must be rotated clockwise by "{number}" to derive its global coordinates')
def step_impl(context, number):
    number = util.assert_number(number)
    if IfcStore.file.schema == "IFC2X3":
        return check_ifc2x3_geolocation("EPset_MapConversion", "Height", number)
    abscissa = check_ifc4_geolocation("IfcMapConversion", "XAxisAbscissa", should_assert=False)
    ordinate = check_ifc4_geolocation("IfcMapConversion", "XAxisOrdinate", should_assert=False)
    actual_value = round(ifcopenshell.util.geolocation.xaxis2angle(abscissa, ordinate), 3)
    value = round(number, 3)
    assert actual_value == value, _('We expected a value of "{}" but instead got "{}"').format(value, actual_value)
```

Algorithmus 4.16: Step-Definition von `The model must be rotated clockwise by "number"to derive its global coordinates`

```
def check_ifc4_geolocation(entity_name, prop_name=None, value=None, should_assert=True):
    if entity_name not in IfcStore.bookmarks:
        has_entity = False
        project = IfcStore.file.by_type("IfcProject")[0]
        for context in project.RepresentationContexts:
            if entity_name == "IfcMapConversion":
                if (
                    context.is_a("IfcGeometricRepresentationContext")
                    and context.ContextType == "Model"
                    and context.HasCoordinateOperation
                ):
                    IfcStore.bookmarks[entity_name] = context.HasCoordinateOperation[0]
                    has_entity = True
            elif entity_name == "IfcProjectedCRS":
                if (
                    context.is_a("IfcGeometricRepresentationContext")
                    and context.ContextType == "Model"
                    and context.HasCoordinateOperation
                    and context.HasCoordinateOperation[0].TargetCRS
                ):
                    IfcStore.bookmarks[entity_name] = context.HasCoordinateOperation[0].TargetCRS
                    has_entity = True
        if not has_entity:
            assert False, _("No model geometric representation contexts refer to an {}").format(entity_name)
    if not prop_name:
        return
    actual_value = getattr(IfcStore.bookmarks[entity_name], prop_name)
    if should_assert:
        assert actual_value == value, _('We expected a value of "{}" but instead got "{}"')
            .format(value, actual_value)
    else:
        return actual_value
```

In einem weiteren Schritt wird die Funktion:
round(ifcopenshell.util.geolocation.xaxis2angle(abscissa, ordinate), 3) (Zeile 8) ausgeführt. Hier werden die beiden Attribute `XAxisAbscissa` und `XAxisOrdinate` der Klasse `IfcMapConversion` in eine trigonometrische Funktion xy2angle(x,y) des Python-Skripts

geolocation.py (Algorithmus 4.18) eingesetzt. Hiermit wird der mathematische Grad zurückgegeben.

```
# Used for converting the X and Y vectors of the X Axis in IFC geolocation
def xy2angle(x, y):
    return math.degrees(math.atan2(y, x))
```

Algorithmus 4.18: Python-Funktion zur Berechnung des Drehwinkels im step * The model must be rotated clockwise by "number"to derive its global coordinates

Property Graph

Im Zuge der Validierung auf das Graph-Modell lässt sich die step Funktion des BIMTesters `* The model must be rotated clockwise by "number"to derive its global coordinates` durch folgende Cypher Formel implementieren:

```
MATCH (n{EntityType:'IfcMapConversion'})
RETURN round(degrees(atan2(n.XAxisAbscissa,n.XAxisOrdinate)),3)
```

Algorithmus 4.19: Cypher-Befehl zur Berechnung des steps `* The model must be rotated clockwise by "number"to derive its global coordinates`

Für das folgende BIMTester-Szenario `A true north rotation of the project origin is provided for convenient reference` mit dem step `* The model must be rotated clockwise by "number"for true north to point up`, soll das Modell prüfen, ob das Modell um einen bestimmten Grad gedreht wurde, sodass TrueNorth in den Norden zeigt. Wie in Algorithmus 4.20 dargestellt, wird ein Pattern, welches den Knoten der Klasse IfcGeometricRepresentationContext und `IfcDirection` mit der Relationstypen TrueNorth assoziiert, abgefragt. Mit der `RETURN`- Klausel wird die Wertergebnisse der mathematischen Funktion zurückgegeben.

Hier kann die Validierung nur ausgeführt werden, wenn der Benutzer jeweils zusätzlich den ersten und zweiten Attributwert im Tupel von `DirectionRatios` der Klasse `IFcDirection` manuell in die Funktion $atan2(x)$ eingibt. Als Ergebnis erhält man folgende zwei Werte: `6.123031769111886e-17` und `1.0`.

```
MATCH (n{EntityType:'IfcGeometricRepresentationContext'})
-[r{rel_type:'TrueNorth'}]-(m{EntityType:'IfcDirection'})
RETURN round(degrees(atan2(6.123031769111886e-17,1.0)),3) =0.0
```

Algorithmus 4.20: Cypher-Abfrage für die Überprüfung `The model must be rotated clockwise by "number"for true north to point up`

Die Steps, die für die Prüfung notwendig sind, um die globalen Koordinaten des Standordursprungs zu überprüfen werden wie folgt beschrieben:

`* The element "guid"has a global easting, northing, and elevation of "number", "number", and "number"respectively.`

`* The element "guid"has a local X, Y, and Z coordinate of "number", "number", and "number"respectively.`

Auch hier kann nicht direkt die step-Anforderung in die Cypher-Sprache umgewandelt werden. Die Implementierung scheitert hauptsächlich daran, dass Cypher die Tupelzahlen oder Matrizen nicht berechnen kann.

Solibri Model Checker

Für diesen Fall stellt SMC keine passenden Regeln zur Verfügung. Eine Umsetzung unter Zuhilfenahme der Entwciklerschnittstelle könnten aber herangezogen werden, um ebendies zu erstellen. Dies war allerdings nicht Teil der vorliegenden Arbeit und wird nicht näher behandelt.

4.3 Ergebnis und Bewertung der Case Study

Die Überprüfung von Attributen und Eigenschaften konnten größtenteils in allen drei Validierungswerkzeuge auf ihre Richtigkeit und Vorhandensein geprüft werden, da im Prozess einfache Datenstrukturen verwendet werden. In der Software Solibri Model Cheker zeigen sich dennoch einige Beschränkungen auf, da nicht alle Attribute von SMC verarbeitet werden. So ist für das Objekt `IfcSite` die Eigenschaft `Adresse`, jedoch nicht die `Grundbuchnummer` (land_title_number) vorhanden.

BIMTester beschreibt steps, die nicht nur für bestimmte Instanzklassen (z.B. `IfcSite`, `IfcProject`), sondern auch für alle Objekte abstrakter Superklassen gelten können. In der Fallstudie wurden dafür bspw. abstrakte Begrifflichkeiten wie `All concrete elements` oder `All elements` verwendet, um auf alle Instanzen der Superklasse `IfcElement` zu referenzieren. Im Property Graph können solche abstrakte Begriffe einige Schwierigkeiten bereiten, da die zugrundeliegende Vererbungshierarchie bisher nicht im Metamodell des Graphen berücksichtigt wird. Das Meta-Modell wurde auf Basis der IFC-SPF generiert, sodass dementsprechend nur Instanzklassen abgebildet werden. Im Zuge dessen spielt der Detailgrad der Abfragestruktur eines Musters eine wichtige Rolle.

Durch die Formulierung von bestimmten Muster können bereits ausgewählte Knoten herausgefiltert werden.Für die Überprüfung der Materialzugehörigkeit von Elementen genügt somit bspw. das Muster, indem im Property Graphen der Knoten n gesucht wird, der eine *-lange Relation zu einer IfcMaterial-Klasse hat. Dennoch wird in manchen Fällen eine zusätzliche Spezifzierung des Musters benötigt. Für die step-Anforderung `All elements must be under 'number' polygon` muss zusätzlich im Muster das Attribut `ObjectType` definiert werden, sodass nur die Klassen des `IfcElements` und nicht z.B. die Klassen des `IfcElementType` im Graphen gesucht werden. In SMC werden alle Objekte in ein SMC-spezifischer Komponententyp transformiert. Für die Spezifizierung des zu validierenden Bezugobjekts werden als Oberbegriff Superklassen wie `IfcProject`, `IfcSite`, `IfcBuilding`, die als Container (Platzhalter) dienen definiert, sodass im Gegensatz zum

Proeprty Graph keine Schwierigkeiten bzgl. der Validierung aufkamen. Anschließend können im Architektur-Domain auch die Objekte der `IfcBuildingElements` bestimmt werden.

Bei Anforderungen die eine Implementierung mathematischer Funktionen benötigen ist der BIMTester mit der Python-step-Implementierung von großem Vorteil. In Python sind verschiedene Verlinkungen von Funktionen möglich. In der Überprüfung vom step `The model must be rotated clockwise by "number"to derive its global coordinates` wurde bspw. in einer step-Funktion die Funktion check_ifc4_geolocatiion(x,y,z) aufgerufen. Im Zuge der Validierung solcher Anforderungen im Property Graph ergeben sich jedoch einige Beschränkungen. Zwar ist die Implementierung einfacher mathematischer Funktionen, wie in Bereichen des Logarithmus oder Trigonometrie mit Cypher möglich, komplexere Verschachtelungen von Matrizen oder Trupel technisch jedoch nicht umsetzbar. Für die spezifischen Fälle der Case Study konnten im SMC keine passenden Regeln gefunden werden.

Kapitel 5

Diskussion

5.1 Zusammenfassung

Für die Gestaltung eines produktiven Arbeitsverlaufs mit BIM ist die Qualitätskontrolle der Modelle von großer Bedeutung. Die Modellinhalte werden in schema-basierte Datenmodelle gespeichert, auf deren Grundlage die Validierung stattfindet. Vorhandene Validierungssoftware - und werkzeuge bereiten bereits gute Lösungsansätze, sind jedoch in ihrer Fähigkeit komplexere Datennetzstrukturen zu überprüfen sehr beschränkt und/oder kompliziert zu bedienen.

Der BIMTester stellt eine benutzerfreundliche unit-test-basierte Lösung mit großer Fähigkeit für die Überprüfung von grundlegenden Datenstrukturen dar und können direkte Überpürfungen auf schema-basierte Modelle ausführen. Dennoch können komplexere Datennetze der Datenmodelle nur sehr erschwert überprüft werden.

Der in dieser Arbeit vorgestellte Ansatz erzielt die Validierung von komplexen intern vernetzten Informationen auf graphenbasierter Repräsentationen auf Grundlage des Validierungsansatzes des BIMTesters.

Graphen besitzen die einzigartige Fähigkeit komplexere Datennetzstrukturen zu erkennen und diese durch Musterformulierungen abzugfragen. Durch die Abbildung der intern verbundenen Daten als Knoten und Kanten auf einem Property Graph sind die Ergebnisse der Überprüfung verständlich. Cypher bietet eine vielfältige und einfach-erlernbare Abfragesprache dar und ermöglicht die bereits geformten Abfragen wiederzuverwenden und zu modifizieren. Dabei ist die Visualisierung des Musters durch Cypher ein wesentlicher Vorteil der Graph-Validierung gegenüber anderen Validierungsansätze, da dem Benutzer direkt die Datenstruktur verständlich gemacht wird.

Im Zuge der Regel-Implementierung des BIMTesters auf den Property Graph haben sich einige Einschränkungen ergeben, die sich auf den Informationsverlust in der Transformation von BIM-Modellen in den Graphen zurückzuführen ist. So werden Zusammenhänge der objektorientierten Strukturen, wie bspw. die Abbildung von abstrakten Klassen in den Graph-Modellen nicht berücksichtigt und können nur durch genauere Mustern, spezifiziert werden. Das jedoch fordert ein tieferes Verständnis der Strukturen und Darstellungen des BIM-Modells bzw. Datenmodells.

Bei Modellprüfungen, die eine mathematische Lösung als Ergebnis bereitstellen, ist die Validierung mit der Abfragesprache Cypher nur zum Teil ausführbar, da sich Cypher in erster Linie mit der Suche von Objekten und Knoten beschäftigt.

5.2 Ausblick

Diese Arbeit zeigt, dass der Großteil der Regel-Implementierungen des BIMTesters auf einen Graphen ausgeführt werden können. Daraus ergeben sich vielfältige Möglichkeiten und Potenzial für zukünftige Forschung in der Thematik der Validierung auf graphenbasierte Repräsentationen.

Es wäre zudem sinnvoll zu untersuchen, ob eine Methode zur Erfassung der abstrakten Superklassen oder der inversen Beziehungen entwickelt werden sollte.

Der Einsatz von graphenbasierten Repräsentationen im Zuge der Validierung zeigt großes Potenzial, das weiterhin ausgearbeitet werden kann, um auch z.B. komplette objektorientierte Strukturen oder ggf. inverse Beziehungen der Daten vollständig zu erkennen und validieren zu können.

Anhang A

Anhang A

A.1 Transformation der BIMTester-steps in Cypher

ProjectSetup	
* IFC data must use the "{schema}" schema	-
Scenario: Exempt files	
* The IFC file "{file}" is exempt from being provided	-
* No further requirements are specified because "{reason}"	-
Scenario: Project metadata is organised and correct	
* The project must have an identifier of "{guid}"	MATCH (n{EntityType:'IfcProject'}) WHERE n.GlobalId = 'guid' RETURN n IS NOT NULL
* The project name, code, or short identifier must be "{name}"	MATCH (n{EntityType:'IfcProject'}) WHERE n.Name='name' RETURN (n)
* The project must have a longer form name of "{long_name}"	MATCH (n{EntityType:'IfcProject'}) WHERE n.LongName =’Long_name’ RETURN (n)
* The project must be described as "{description}"	MATCH (n{EntityType:'IfcProject'}) WHERE n.Description='description' RETURN (n)
* The project must be categorised under "{object_type}"	MATCH (n{EntityType:'IfcProject'}) WHERE n.ObjectType='object_type' RETURN (n)
* The project must contain information about the "{phase}" phase	MATCH (n{EntityType:'IfcProject'}) WHERE n.Phase='phase' RETURN (n)
* The project must contain 3D geometry representing the shape of objects	MATCH (d{EntityType: 'IfcGeometricRepresentationSubContext', ContextType:'Model',ContextIdentifier:'Body', TargetView:'MODEL_VIEW'})-- (n{EntityType:'IfcGeometricRepresentationContext'}) - [b:rel]- (c{EntityType:'IfcProject'}) RETURN c IS NOT NULL

Geolocation	
Scenario: Geometry is georeferenced to a	

coordinate reference system	
* There must be at least one "IfcSite" element	MATCH (n{EntityType:'IfcSite') RETURN count(n) >= 1
* The project must have coordinate reference system data	MATCH (a{EntityType:'IfcProject'})-- (b{EntityType:'IfcGeometricRepresentationContext'})-- (n{EntityType:'IfcMapConversion'})-- (m{EntityType:'IfcProjectedCRS'}) RETURN a IS NOT NULL
* The name of the CRS must be "{coordinate_reference_name}"	MATCH (n{EntityType:'IfcProjectedCRS'}) WHERE n.Name = ' coordinate_reference_name ' RETURN (n)
* The description of the CRS must be "{value}"	MATCH (n{EntityType:'IfcProjectedCRS'}) WHERE n.Description='value' RETURN n IS NOT NULL
* The geodetic datum must be "{coordinate_reference_name}"	MATCH (n{EntityType:'IfcProjectedCRS'}) WHERE n.GeodeticDatum = ' coordinate_reference_name ' RETURN (n)
* The vertical datum must be "{coordinate_reference_name}"	MATCH (n{EntityType:'IfcProjectedCRS'}) WHERE n.VerticalDatum = ' coordinate_reference_name ' RETURN (n)
* The map projection must be "{coordinate_reference_name}"	MATCH (n{EntityType:'IfcProjectedCRS'}) WHERE n.MapProjection = ' coordinate_reference_name ' RETURN (n)
* The map zone must be "{coordinate_reference_name}"	MATCH (n{EntityType:'IfcProjectedCRS'}) WHERE n.MapZone = ' coordinate_reference_name ' RETURN (n)
* The map unit must be "{unit}"	MATCH (n{EntityType:'IfcProjectedCRS'})-- (m{EntityType:'IfcSIUnit'}) RETURN m.Name='unit' or MATCH (n{EntityType:'IfcProjectedCRS'})-- (m{EntityType:'IfcConversionBasedUnit'}) RETURN m.Name = 'unit' or MATCH (n{EntityType:'IfcProjectedCRS'}) RETURN m.Mapunit = 'unit'

Scenario: Local coordinate systems are specified relative to a global system	
* The project must have coordinate transformations to convert from local to global coordinates	MATCH (n{EntityType:'IfcMapConversion'}) – (u{EntityType:'IfcProject'}) RETURN u IS NOT NULL
* The eastings of the model must be offset by "{number}" to derive its global coordinates	MATCH (n{EntityType:'IfcMapConversion'}) RETURN n.Eastings=number
* The northings of the model must be offset by "{number}" to derive its global coordinates	MATCH (n{EntityType:'IfcMapConversion'}) RETURN n.Northings=number
* The height of the model must be offset by "{number}" to derive its global coordinates	MATCH (n{EntityType:'IfcMapConversion'}) RETURN n.OrthogonalHeight=number
* The model must be rotated clockwise by "{number}" to derive its global coordinates	MATCH (n{EntityType:'IfcMapConversion'}) RETURN round(degrees(atan2(n.XAxisAbscissa,n.XAxisOrdinate)),3)
* The model must be scaled along the horizontal axis by "{number}" to derive its global coordinates	MATCH (n{EntityType:'IfcMapConversion'}) WHERE n.Scale = number RETURN (n)
Scenario: A true north rotation of the project origin is provided for convenient reference	
* The model must be rotated clockwise by "{number}" for true north to point up	MATCH (n{EntityType:'IfcGeometricRepresentationContext'}) -[r{rel_type:'TrueNorth'}]-(m{EntityType:'IfcDirection'}) RETURN round(degrees(atan2(x,y)),3) mit: x = IfcMapConversion.XAxisAbscissa y = IfcMapConversion.XAxisOrdinate
Scenario: Global coordinates of the site origins are provided for convenient reference	
* The site "{guid}" has a longitude of "{longlat}"	-
* The site "{guid}" has a latitude of "{longlat}"	-
* The site "{guid}" has an elevation of "{number}"	MATCH (n{EntityType:'IfcSite'}) WHERE n.GlobalId ='guid'

	AND n.RefElevation = number RETURN (n)
* The site "{guid}" must be coincident with the project origin	MATCH (n{EntityType:'IfcSite',GlobalId:'guid'})--(m{EntityType:'IfcLocalPlacement'})--(u{EntityType:'IfcAxis2Placement3D'})--(z{EntityType:'IfcCartesianPoint'}) RETURN z.Coordinates= "(0.0, 0.0, 0.0)")

Model Federation	
Scenario: Ensure that an agreed datum for the project is in the right location	
* There is a datum element "{guid}" as an "{ifc_class}"	MATCH (n{GlobalId:guid') WHERE n.EntityType = 'ifc_class' RETURN (n)
* The element "{guid}" has a global easting, northing, and elevation of "{number}", "{number}", and "{number}" respectively	-
* The element "{guid}" has a local X, Y, and Z coordinate of "{number}", "{number}", and "{number}" respectively	-

Element classes	
Scenario: Ensure all IFC type elements use the correct IFC class	
* The element "{guid}" is an "{ifc_class}"	MATCH (n{GlobalId:'guid'}) WHERE n.EntityType= 'ifc_class' RETURN (n)
* The element "{guid}" is an "{ifc_class}" only	MATCH (n{GlobalId:'2lloObyt155fu9rViLQIbT'}) WHERE n.EntityType= 'IfcWall' RETURN count(n)=1
* The element "{guid}" is further defined as a "{predefined_type}"	MATCH (n{GlobalId:'2lloObyt155fu9rViLQIbT'}) WHERE n.PredefinedType='USERDEFINED' AND n.ObjectType RETURN n IS NOT NULL Or MATCH (n{GlobalId:'2lloObyt155fu9rViLQIbT'}) WHERE n.PredefinedType='predefined_type' RETURN n IS NOT NULL
* The element "{guid}" should not exist because "{reason}"	MATCH (n{GlobalId:'2lloObyt155fu9rViLQIbT'}) WHERE n.GlobalId IS NULL RETURN (n)

Geocoding (Hier ist nur ein Teil der steps aufgelistet, da sich das Prinzip der Abfrage nicht ändert. Es wird statt IfcSite die Klassen IfcBuilding und IfcFacility für die anderen steps geprüft.)	
Scenario: The site is tied to a physical building address	
* The site "{guid}" has a name of "{name}"	MATCH (n{EntityType: 'IfcSite'}) WHERE a.Name = 'name' RETURN (n)
* The site "{guid}" has a description of "{description}"	MATCH (n{EntityType: 'IfcSite'}) WHERE a.Description = 'description' RETURN (n)
* The site "{guid}" has a land title number of "{land_title_number}"	MATCH (n{EntityType: 'IfcSite'}) WHERE a.LandTitleNumber = 'None' RETURN (n)
* The site "{guid}" has the address "{address_lines}"	MATCH (n{EntityType: 'IfcSite'}) -- (n{EntityType: 'IfcPostalAdress'}) WHERE a.AdressLines = 'adress_lines' RETURN (n)
* The site "{guid}" has a postal box of "{postal_box}"	MATCH (n{EntityType: 'IfcSite'}) -- (n{EntityType: 'IfcPostalAdress'}) WHERE a.PostalBox = 'postal_box' RETURN (n)
* The site "{guid}" is in the town "{town}"	MATCH (n{EntityType: 'IfcSite'}) -- (n{EntityType: 'IfcPostalAdress'}) WHERE a.Town = 'town' RETURN (n)
* The site "{guid}" is in the region "{region}"	MATCH (n{EntityType: 'IfcSite'}) -- (n{EntityType: 'IfcPostalAdress'}) WHERE a.Region = 'region' RETURN (n)
* The site "{guid}" has a post code of "{post_code}"	MATCH (n{EntityType: 'IfcSite'}) -- (n{EntityType: 'IfcPostalAdress'}) WHERE a.PostalCode = 'postal_code' RETURN (n)
* The site "{guid}" is in the country "{country}"	MATCH (n{EntityType: 'IfcSite'}) -- (n{EntityType: 'IfcPostalAdress'}) WHERE a.Country ='country' RETURN (n)
* The site "{guid}" has an address description of "{description}"	MATCH (n{EntityType: 'IfcSite'}) -- (n{EntityType: 'IfcPostalAdress'}) WHERE a.Name = 'description'

	RETURN (n)
Classification	
Scenario: The appropriate classification systems are referenced in the model	
* The classification "{name}" must be used	MATCH (n{EntityType:'IfcClassification'}) WHERE n.Name = 'name' RETURN (n)
* The classification "{name}" is published by "{source}"	MATCH (n{EntityType:'IfcClassification'}) WHERE n.Name = 'name' AND n.Source = 'source' RETURN (n)
* The classification "{name}" is the edition "{edition}" on "{edition_date}"	MATCH (n{EntityType:'IfcClassification'}) WHERE n.Name = 'name' AND n.EditionDate = 'edition_date' RETURN (n)
* The classification "{name}" has the description "{description}"	MATCH (n{EntityType:'IfcClassification'}) WHERE n.Name = 'name' AND n.Description = 'description' RETURN (n)
* The classification "{name}" is referenced by the website "{location}"	MATCH (n{EntityType:'IfcClassification'}) WHERE n.Name = 'name' AND n.Location = 'location' RETURN (n)
* The classification "{name}" has a hierarchy denoted by the tokens "{tokens}"	MATCH (n{EntityType:'IfcClassification', Name = 'Uniformat' }) WHERE n.ReferenceTokens = 'tokens' RETURN (n)
Scenario: The relevant elements are assigned to the classification system	
* The element "{guid}" is classified as a "{identification}" with name "{reference_name}"	MATCH (n{GlobalId:'guid'})-- (u{EntityType:'IfcRelAssociatesClassification'})- (i{EntityType:'IfcClassificationReferences'}) WHERE i.Identification = 'identification' AND i.Name = 'refernce_name' RETURN (n)
Geometric detail	
Scenario: Geometry must be efficiently modeled	
* All elements must be under "{number}" polygons	MATCH (n:PrimaryNode)-[*]- >(z{EntityType:'IfcFace'}) WHERE n.ObjectType IS NOT NULL RETURN n.EntityType, n.GlobalId, count(z) or

	MATCH (n:PrimaryNode)-[*]- >(z{EntityType:'IfcPolygonalFaceSet'})- [r{rel_type:'Faces'}]-() WHERE n.ObjectType IS NOT NULL RETURN n.EntityType, n.GlobalId, count(r) or MATCH (n:PrimaryNode)-[*]- >(z{EntityType:'IfcTriangulatedFaceSet'})- [r{rel_type:'CoordIndex'}]-() WHERE n.ObjectType IS NOT NULL RETURN n.EntityType, n.GlobalId, count(r)

Literaturverzeichnis

ANGLES, R., ARENAS, M., BARCELÓ, P., HOGAN, A., REUTTER, J. & VRGOČ, D. (2017). Foundations of Modern Query Languages for Graph Databases. *ACM Comput. Surv. 50*(5), 68:1–68:40. doi:10.1145/3104031

ANGLES, R. & GUTIERREZ, C. (2008). Survey of graph database models. *ACM Computing Surveys*, *40*. doi:10.1145/1322432.1322433

BASTIAN, G. (2002). *Abfragesprache für geometrische und semantische Information aus rasterbasierten topografischen Karten* (Diss., ETH Zurich). Artwork Size: 153 S. Medium: application/pdf Pages: 153 S. doi:10.3929/ETHZ-A-004445706

BAUMGÄRTEL, K. & PIRNBAUM, S. (2016). *Automatische Prüfung und Filterung in BIM mit Model View Definitions*.

BEETZ, J., AMANN, J. & BORRMANN, A. (o.D.). Analyse von Einsatzmöglichkeiten von verbundenen Informationen (Linked Data) und Ontologien und damit befassten Technologien (Semantic Web) im Bereich des Straßenwesens, 82.

BEHAVE. (o.D.). Behave - Step Implementations. Zugriff 4. Juli 2022 unter https://www.tutorialspoint.com/behave/behave_step_implementations.htm

BEHAVE. (2017a). Behave API Reference — behave 1.2.6 documentation. Zugriff 14. August 2022 unter https://behave.readthedocs.io/en/stable/api.html#environment-file-functions

BEHAVE. (2017b). Tutorial — behave 1.2.6 documentation. Zugriff 4. Juli 2022 unter https://behave.readthedocs.io/en/stable/tutorial.html#python-step-implementations

BORRMANN, A., BEETZ, J., KOCH, C. & LIEBICH, T. (2015). Industry Foundation Classes – Ein herstellerunabhängiges Datenmodell für den gesamten Lebenszyklus eines Bauwerks. In A. BORRMANN, M. KÖNIG, C. KOCH & J. BEETZ (Hrsg.), *Building Information Modeling: Technologische Grundlagen und industrielle Praxis* (S. 83–127). VDI-Buch. Wiesbaden: Springer Fachmedien. doi:10.1007/978-3-658-05606-3_6

BOTH, P. v. (2009). *Forum Bauinformatik 2009*. Google-Books-ID: CpUWAeWAN5EC. KIT Scientific Publishing.

BRÜGGEMANN, T. & von BOTH, P. (2015). 3D-Stadtmodellierung: CityGML. In A. BORRMANN, M. KÖNIG, C. KOCH & J. BEETZ (Hrsg.), *Building Information Modeling: Technologische Grundlagen und industrielle Praxis* (S. 177–192). VDI-Buch. Wiesbaden: Springer Fachmedien. doi:10.1007/978-3-658-05606-3_10

BUILDINGSMART. (2022a). MicroMVDs for exchange requirements - Wiki.OSArch. Zugriff 14. August 2022 unter https://wiki.osarch.org/index.php?title=MicroMVDs_for_exchange_requirements

BUILDINGSMART. (2022b). MVD Database. Zugriff 16. August 2022 unter https://technical.buildingsmart.org/standards/ifc/mvd/mvd-database/

BUILDINGSMART. (o.D. a). IFC Formats. Zugriff 23. August 2022 unter https://technical.buildingsmart.org/standards/ifc/ifc-formats/

buildingSmart. (o.D. b). IfcGeometricRepresentationSubContext. Zugriff 15. August 2022 unter https://standards.buildingsmart.org/IFC/RELEASE/IFC4_1/FINAL/HTML/schema/ifcrepresentationresource/lexical/ifcgeometricrepresentationsubcontext.htm

Cauchi, A., Colombo, C., Francalanza, A., Micallef, M. & Pace, G. (2016). Using Gherkin to Extract Tests and Monitors for Safer Medical Device Interaction Design. In *Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (S. 275–280). EICS '16. Brussels, Belgium: Association for Computing Machinery. doi:10.1145/2933242.2935868

Cyganiak, R., Wood, D. & Lanthaler, M. (2014). RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation (2014). *URL: http://www. w3. org/TR/rdf-concepts*.

Davila Delgado, J. M. & Oyedele, L. O. (2020). BIM data model requirements for asset monitoring and the circular economy. *Journal of Engineering, Design and Technology*, *18*(5), 1269–1285. Publisher: Emerald Publishing Limited. doi:10.1108/JEDT-10-2019-0284

DIN EN ISO 16739-1:2021-11, Industry Foundation Classes_(IFC) für den Datenaustausch in der Bauwirtschaft und im Anlagenmanagement_- Teil_1: Datenschema (ISO_16739-1:2018); Englische Fassung EN_ISO_16739-1:2020, nur auf CD-ROM. (2021). Beuth Verlag GmbH. doi:10.31030/3144077

DIN EN ISO 19107:2020-09, Geoinformation_- Raumbezugsschema (ISO_19107:2019); Englische Fassung EN_ISO_19107:2019. (2020). Beuth Verlag GmbH. doi:10.31030/3136631

DIN EN ISO 19109:2016-05, Geoinformation_- Regeln zur Erstellung von Anwendungsschemata (ISO_19109:2015); Englische Fassung EN_ISO_19109:2015. (2016). Beuth Verlag GmbH. doi:10.31030/2407428

Eastman, C., Lee, J.-m., Jeong, Y.-s. & Lee, J.-k. (2009). Automatic rule-based checking of building designs. *Automation in Construction*, *18*(8), 1011–1033. doi:10.1016/j.autcon.2009.07.002

Esser, S. (2021). Graphenbasiertes Änderungsmanagement von BIM-Modellen.

Francis, N., Green, A., Guagliardo, P., Libkin, L., Lindaaker, T., Marsault, V., … Taylor, A. (2018). Cypher: An Evolving Query Language for Property Graphs. In *Proceedings of the 2018 International Conference on Management of Data* (S. 1433–1445). SIGMOD '18. New York, NY, USA: Association for Computing Machinery. doi:10.1145/3183713.3190657

Gröger, G. & Plümer, L. (2012). CityGML – Interoperable semantic 3D city models. *ISPRS Journal of Photogrammetry and Remote Sensing*, *71*, 12–33. doi:10.1016/j.isprsjprs.2012.04.004

Häringer, P. (2017). Anwendung der MVD-Methode in einem Datenaus-tauschszenario im BIMsite Forschungsprojekt. In *Proc. of the 29th Forum Bauinformatik*.

IfcOpenShell. (2022). BIMTester - Wiki.OSArch. Zugriff 8. Juli 2022 unter https://wiki.osarch.org/index.php?title=BIMTester

Ismail, A., Nahar, A. & Scherer, R. (2017). Application of graph databases and graph theory concepts for advanced analysing of BIM models based on IFC standard.

ISO 19136-1:2020-12, Geoinformation_- Geography Markup Language_(GML)_- Teil_1: Grundsätze (ISO_19136-1:2020); Englische Fassung EN_ISO_19136-1:2020, nur auf CD-ROM. (2020). Beuth Verlag GmbH. doi:10.31030/3166603

KANG, T. (2018). Development of a Conceptual Mapping Standard to Link Building and Geospatial Information. *ISPRS International Journal of Geo-Information*, *7*(5), 162. Number: 5 Publisher: Multidisciplinary Digital Publishing Institute. doi:10.3390/ijgi7050162

KARKI, S., THOMPSON, R., MCDOUGALL, K., CUMERFORD, N. & OOSTEROM, P. (2011). ISO land administration domain model and LandXML, in the development of digital survey plan lodgement for 3D cadastre in Australia.

KNUBLAUCH, H. & KONTOKOSTAS, D. (2017). Shapes constraint language (shacl), w3c recommendation 20 july 2017. *URL: https://www. w3. org/TR/shacl*.

KUTZNER, T., CHATURVEDI, K. & KOLBE, T. H. (2020). CityGML 3.0: New Functions Open Up New Applications. *PFG*, *88*(1), 43–61. doi:10.1007/s41064-020-00095-z

LAAKSO, M & KIVINIEMI. (2012). The IFC standard: A review of History, development, and standardization, Information Technology. Open Publishing Services. Zugriff unter http://usir.salford.ac.uk/id/eprint/28373/

LANDXML.ORG. (o.D.). LandXML-1.2Doc. Zugriff 31. August 2022 unter http://www.landxml.org/schema/LandXML-1.2/documentation/LandXML-1.2Doc.html

LANDXML.ORG. (2006). Zugriff 24. August 2022 unter https://view.officeapps.live.com/op/view.aspx?src=http%3A%2F%2Fwww.landxml.org%2Fschema%2FDocumentation%2FLandXML.org_2006.ppt&wdOrigin=BROWSELINK

LEE, G. (2009). Concept-Based Method for Extracting Valid Subsets from an EXPRESS Schema. *Journal of Computing in Civil Engineering*, *23*(2), 128–135. Publisher: American Society of Civil Engineers. doi:10.1061/(ASCE)0887-3801(2009)23:2(128)

LIU, H., GAO, G., ZHANG, H., LIU, Y.-S., SONG, Y. & GU, M. (2022). MVDLite: a Fast Validation Algorithm for Model View Definition Rules. Text.Chapter. Zugriff 24. August 2022 unter https://ebooks.au.dk/aul/catalog/view/455/312/1840-2

MOULT, D. & KRIJNEN, T. (2020). Compliance checking on building models with the Gherkin language and Continuous Integration. In *EG-ICE 2020 Workshop on Intelligent Computing in Engineering, Proceedings* (S. 294–303).

NAWARI, N. O. (2012). BIM Standard in Off-Site Construction. *Journal of Architectural Engineering*, *18*(2), 107–113. _eprint: https://ascelibrary.org/doi/pdf/10.1061/%28ASCE%29AE.1943-5568.0000056. doi:10.1061/(ASCE)AE.1943-5568.0000056

NBS. (2020). Zugriff 30. August 2022 unter https://www.thenbs.com/knowledge/national-bim-report-2020

NEO4J. (2022). Querying with Cypher - Developer Guides. Zugriff 31. August 2022 unter https://neo4j.com/developer/cypher/querying/

ORASKARI, J., BEETZ, J. & SENTHILVEL, M. (2021). SHACL is for LBD what mvdXML is for IFC.

PREIDEL, C. & BORRMANN, A. (2015). Automated Code Compliance Checking Based on a Visual Language and Building Information Modeling. Oulu, Finland. doi:10.22260/ISARC2015/0033

PREIDEL, C., BORRMANN, A. & BEETZ, J. (2015). BIM-gestützte Prüfung von Normen und Richtlinien. In A. BORRMANN, M. KÖNIG, C. KOCH & J. BEETZ (Hrsg.), *Building Information Modeling: Technologische Grundlagen und industrielle Praxis* (S. 321–331). VDI-Buch. Wiesbaden: Springer Fachmedien. doi:10.1007/978-3-658-05606-3_20

RAJABIFARD, A. (2012). Development of a 3D ePlan/LandXML visualisation system in Australia. Zugriff 14. August 2022 unter https://www.academia.edu/es/66611790/Development_of_a_3D_ePlan_LandXML_visualisation_system_in_Australia

SAAKE, G., SATTLER, K.-U. & HEUER, A. (2018). *Datenbanken: Konzepte und Sprachen*. Google-Books-ID: nfpgDwAAQBAJ. MITP-Verlags GmbH & Co. KG.

SCHILD, K. & MOCHOL, M. (2006). Definition von XML-Sprachen DTDs und XML-Schema anhand eines einheitlichen Beispiels.

SCHUBERT, M. & BORGWARDT, K. (2007). Kapitel 7: Graph-Strukturierte Daten.

SIRTL, F. (2020). *Extended Imperative Model Checking – A visual programming approach for a user-friendly MVD generation and validation* (Bachelor's thesis, Technische Universität München).

SOLIBRI. (2022). Understanding Rules. Zugriff 16. August 2022 unter https://help.solibri.com/hc/en-us/articles/1500004751182-Understanding-Rules

SOON, K. H., THOMPSON, R. & KHOO, V. (2014). Semantics-based Fusion for CityGML and 3D LandXML, 16.

STARDOG. (2019). [Training] Data Validation and SHACL. Zugriff 27. August 2022 unter https://www.youtube.com/watch?v=vNdbNeLfSmY

TEMPLETON, K. (2015). *Analyse von Überprüfüngswerkzeügen der inhaltlichen Korrektheit von BIM-Modellen*.

TIDMARSH, I. (2021). BDD – An introduction to feature files. Zugriff 23. Mai 2022 unter https://modernanalyst.com/Resources/Articles/tabid/115/ID/3871/BDD-An-introduction-to-feature-files.aspx

WEISE, M., LIEBICH, T., NISBET, N. & BENGHI, C. (2016). IFC model checking based on mvdXML 1.1. *undefined*. Zugriff 7. Mai 2022 unter https://www.semanticscholar.org/paper/IFC-model-checking-based-on-mvdXML-1.1-Weise-Liebich/7b458d6fead644e514cd615facce198498135109