

SOLCONPRO – GANZHEITLICHE INTEGRATION ENER- GETISCH AKTIVER FASSADENKOMPO- NENTEN IN BAUPROZESSE

Schlussbericht

Gefördert durch:



Bundesministerium
für Wirtschaft
und Energie

aufgrund eines Beschlusses
des Deutschen Bundestages

Das diesem Bericht zugrundeliegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 03ET1290 gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt bei den Autoren.

Schlussbericht

Vorhabenbezeichnung:

Ganzheitliche Integration energetisch aktiver Fassadenkomponenten in Bauprozesse (SolConPro)

Projektbeteiligte:

Ed. Züblin AG, TU Darmstadt, Fraunhofer ISE

Laufzeit des Vorhabens:

4/2015 – 3/2018

Bearbeiter:

Johannes Eisenlohr, Gesa Benndorf, Wendelin Sprenger, Julian Wengzinek, Dawit Ghebrehiwet, Anna Wagner, Christoph Maurer, Laura Kristina Möller, Christian Eller, Christian Leifgen, Uwe Rüp-
pel, Tilmann Kuhn

Datum: 28.09.2018

Dr. Tilmann E. Kuhn

Fraunhofer ISE

Inhalt

1	Kurzdarstellung	5
1.1	Aufgabenstellung	5
1.2	Voraussetzungen	5
1.3	Kurzzusammenfassung der Ergebnisse	5
1.4	Planung und Ablauf des Vorhabens	6
1.5	Stand der Wissenschaft und Technik	7
1.6	Zusammenarbeit mit anderen Stellen	9
1.6.1	buildingSMART	9
1.6.2	W3C	9
2	Eingehende Darstellung der Ergebnisse	10
2.1	Arbeitspaket 1 – Bauprozessintegration „zu Fuß“ für konkrete Beispielfälle – Schwerpunkt Gebäudehülle	10
2.2	Arbeitspaket 2 – Evaluierung und Identifikation von Produktdatenmodellen (BIM) für Systeme der Technischen Gebäudeausrüstung und energierelevante Planungsprozesse	11
2.3	Arbeitspaket 3 – Erweiterung/Neudefinition von „Rollen von Akteuren“ im Bauprozess, die für die Funktionalität der aktiven Komponenten verantwortlich sind	12
2.4	Arbeitspaket 4 – Theoretische Modellbildung für aktive Gebäudehüllkomponenten im Gebäudekontext und theoretische Rollendarstellung der verantwortlichen Akteure	14
2.4.1	Betrachtete Methoden, Systeme und Technologien	14
2.4.1.1	Semantic-Web-Technologien im Bauwesen	14
2.4.1.2	Demilitarisierte Zonen	16
2.4.2	IT-Architektur	17
2.4.2.1	Projektplattformen	17
2.4.2.1	Anforderungen an Projektplattformen	17
2.4.2.2	Datenzugänglichkeitskategorien	18
2.4.2.3	Auswirkungen des Datenaustauschs zwischen Plattformen auf die Datenzugänglichkeit	20
2.4.2.4	Angestrebte Datenhaltung und Herausforderungen	22
2.4.2.5	Allgemeine Anforderungen an externe APIs der Plattformen	23
2.4.3	Konzeptionierung virtueller Komponenten für energetisch aktive Fassadenelemente	25
2.4.3.1	Anforderungen an die Datenstruktur	25
2.4.3.2	Vorstellung Grundkonzept der Produktbeschreibungs-Ontologie	27
2.4.3.3	Abbildung parametrischer Zusammenhänge	28
2.4.3.4	Produktkatalog zur Integration energetisch aktiver Fassadenkomponenten (ViKoDB und ViKoLink)	29
2.4.4	Integration von Verarbeitungsprozessen	29
2.4.4.1	Datenfilterung (RBV)	30
2.4.4.2	Anreicherung von Daten (Worker)	32
2.4.4.3	Einbindung in Modellierungs- und Simulationsumgebungen (OBW)	32
2.4.4.4	Zusammenführung in Plattform/Cloud (SolConPro-Cloud)	33
2.4.5	Modellbasierte EnEV-Berechnung	35
2.4.5.1	Aufgabenstellung	35
2.4.5.2	Einfluss solaraktiver Fassaden auf die energetische Bewertung	35
2.4.5.3	Proprietäre Schnittstellen für die modellbasierte EnEV-Berechnung	36

2.4.5.4	Datenaustausch mit offenen Schnittstellen	36
2.4.5.5	Fazit	37
2.5	<i>Arbeitspaket 5 – Prototypische Umsetzung der Ergebnisse für einzelne Produkte</i>	38
2.5.1	Produkt-Ontologie	38
2.5.1.1	Semantische Bezeichnung	38
2.5.1.2	Produktkonstruktion	38
2.5.1.3	Geometrie	42
2.5.1.4	Mehrfachklassifizierung	43
2.5.1.5	Parametrik	44
2.5.1.6	Ontologie-Kerndaten	45
2.5.2	Produktdatenbank (ViKoDB)	45
2.5.3	Produktdatenkatalog (ViKoLink)	46
2.5.3.1	Parametrik-Evaluation (param)	47
2.5.4	SolConPro-Cloud	51
2.5.4.1	Backend	52
2.5.4.2	Frontend	57
2.5.5	Firmen- und softwareübergreifendes Kollisionsmanagement	60
2.6	<i>Arbeitspaket 6 – Validierung und Verifikation</i>	65
2.6.1	Anwendungsfälle zur Einbindung geometrischer und semantischer Produktinformationen in bestehende BIM-Abläufe	66
2.6.1.1	Distributive Verteilung virtueller Komponenten und deren Visualisierung	66
2.6.1.2	Einlesen geometrischer und semantischer Informationen in Revit	73
2.6.1.3	RBV & OBW zur Bestrahlungsstärkeberechnung auf Gebäudehüllen	77
2.6.1.4	RBV & OBW für BIPV-Simulationen (Kombination von Produkt- und Projektebene)	78
2.6.1.5	Neue digitale Methoden für die technische Gebäudeausrüstung	79
2.6.2	Anwendungsfälle zur anwendungsunabhängigen und firmenübergreifenden Datenhaltung	85
2.6.2.1	Software- und firmenübergreifende Kollisionsbearbeitung und Arbeitsweise zur Nutzung semantischer Produktdatenbeschreibungen in Autodesk Revit	85
2.6.2.2	Datensammlung anhand von Webformularen und Dashboard-Visualisierung	87
2.6.2.3	Arbeitsweise mit software-seitig bereitgestellten REST-APIs und Umsetzung anhand des Beispiels Dalux	88
2.6.2.1	Betreiben einer ViKoDB in einer DMZ aus der Sicht eines Produktherstellers	90
3	Notwendigkeit der geleisteten Arbeit	91
4	Voraussichtlicher Nutzen und Verwertbarkeit der Ergebnisse	91
5	Fortschritt bei anderen Stellen	92
6	Veröffentlichungen	93
7	Literaturverzeichnis	94
	Anhang A: Technologischer Glossar	96
	Anhang B: Übersicht der implementierten Datenverarbeitungsmodule	103
	Anhang C: Beispiel für API-Dokumentation	106

1 Kurzdarstellung

1.1 Aufgabenstellung

Die Hülle von Bestands- und Neubauten wird immer stärker zur Nutzung von regenerativen Energiequellen – vor allem Solarenergie – herangezogen. Sowohl der Umbau des Gesamtenergiesystems mit großem Flächenbedarf für solare Energiegewinnung also auch schrittweise strenger werdende Gebäudeenergie Richtlinien werden diesen Trend weiter verstärken. Die Einführung von komplexeren, solaraktiven Fassadenkomponenten führt zu großen Herausforderungen, u.a. in Bezug auf Planungsprozesse, Planungsmethoden, Produktdaten, Datenhaltung, Bauausführung und Betrieb von Gebäuden. Das übergeordnete Ziel des Forschungsprojekts SolConPro besteht in der ganzheitlichen Integration solaraktiver Fassadenkomponenten in Bauprozesse.

1.2 Voraussetzungen

Solaraktive Fassadenkomponenten führen trotz der rapiden Kostendegression z. B. der Photovoltaik nach wie vor ein Nischendasein, obwohl rechnerisch in den meisten Fällen die erzeugte Nutzenergie die initialen Zusatzkosten langfristig mehr als ausgleichen sollte. Der Grund ist darin zu sehen, dass für den Einsatz allgemein komplexerer Systeme im Bauwesen eine Abstimmung zwischen vielen verschiedenen Baubeteiligten erfolgen muss, die jeweils unterschiedliche Verantwortungsbereiche abdecken. Für solaraktive Fassadenkomponenten sind z. B. Gebäude- und Fassadenstatik, Architektur, technische Systemauslegung, Technische Gebäudeausrüstung, Logistik, Kostenberechnung etc. beteiligt. Ohne digitale Unterstützung gestaltet sich die Kommunikation technischer Aspekte oft herausfordernd, vor allem dann, wenn verschiedene Planungsprozesse zu sich widersprechenden Ergebnissen führen.

Im Bauwesen wird versucht, die Problemstellung einerseits mittels Aufstellung und Standardisierung sequentieller Bauabläufe, andererseits mittels Verwendung eines digitalen Gebäudemodells zu lösen. Im Forschungsprojekt SolConPro wurde versucht, die genannten Lösungswege konkret für solaraktive Fassadenkomponenten umzusetzen.

1.3 Kurzzusammenfassung der Ergebnisse


Für viele Themen der Digitalisierung im Bauwesen, zu dem auch die vorliegenden Projekthalte gehören, werden in der Fachwelt die Konzepte der Standardisierungsorganisation buildingSMART vorgeschlagen. Das Projektkonsortium hat sich aus diesem Grund zu Beginn des Projekts ausführlich mit deren wichtigsten Formaten *Industry Foundation Classes* (□ *IFC*), *Information Delivery Manuals* (□ *IDM*) und *Model View Definitions* (□ *MVD*) beschäftigt. Bereits nach dem ersten Arbeitspaket jedoch erwies sich die fachliche Ausrichtung der genannten Formate als unzureichend für die Umsetzung der Projektziele, überraschenderweise stehen deren Entwicklungen der Digitalisierung des Bauwesens und damit den Projektzielen selbst sogar diametral entgegen.

Im Projektverlauf erwies sich die vollständige und zeitgleich software-unabhängige Beschreibung von Bauprodukten als Dreh- und Angelpunkt für die Bewältigung der Komplexität solaraktiver Fassadensysteme in Gebäudemodellen. Da das Format IFC nur der Standardisierung von Datenübertragungsprozessen zwischen proprietären Softwareprogrammen dient, kam dessen Verwendung für den obengenannten Zweck nicht infrage. Zudem stellte sich im Projektverlauf heraus, dass eine kollaborative statt sequentielle Arbeitsweise zur Bewältigung komplexer Bauprojekte un-

umgänglich ist, weshalb aufgrund der darin definierten Rollenfestlegungen auch das IDM-Format verworfen werden musste. Der hohe Informationsgehalt software-übergreifender Gebäudemodelle erfordert schließlich nicht nur Filter-, sondern auch Vorverarbeitungsmethoden. Auch für das Format MVD musste deshalb Ersatz gefunden werden.

Für die vollständige Wiedergabe von Bauproduktbeschreibungen wurde das Projektkonsortium schließlich im W3C-Umfeld fündig. Auf Basis der Semantic-Web-Technologie wurden für einige solaraktive Fassadenkomponenten (semantisch und geometrisch, letzteres auch parametrisch) vollständige Bauproduktbeschreibungen erstellt und per HTTP-Zugriff verfügbar gemacht. Da diese Beschreibungen für unterschiedliche Bauprozesse und Berechnungen angewandt werden, müssen Vorverarbeitungsroutinen für die Bereitstellung der jeweiligen benötigten Informationsmenge sorgen. Hierfür wurden die beiden Konzepte *Rule Based Views* (RBV) und *Ontology Based Wrappers* (OBW) entwickelt. Für die Umsetzung einer nicht-sequentiellen Arbeitsweise schließlich waren Untersuchungen zur IT-Architektur erforderlich, die in eine Implementierung von Demilitarisierten Zonen mündeten. Die angestrebte Form des Informationsaustauschs zwischen unterschiedlichen Softwareprogrammen und auch Firmen wurde prototypisch für das Kollisionsmanagement umgesetzt. Auf diese Weise wurde sowohl für die Serverbereitstellung als auch für die Dateninhalte die Basis für die Umsetzung komplexer Gebäudemodelle geschaffen, von denen Gebäude mit solaraktiven Fassadensystemen eine Unter-
menge darstellen.

1.4 Planung und Ablauf des Vorhabens

Zur Erreichung der Projektziele wurden sechs Arbeitspakete veranschlagt. In AP1 wurden konkrete Bauprojekte mit durchgeführter oder geplanter Installation solaraktiver Fassadenkomponenten prozesstechnisch untersucht. In AP2 wurde untersucht, auf welche Weise solaraktive Fassadenkomponenten digitalisiert werden müssen, um für die verschiedenen technischen Untersuchungen herangezogen werden zu können, die für eine Installation erforderlich sind. Hierbei wurde das  IFC-Format einer ausführlichen Analyse unterzogen. In AP3 sollten im Bauprozess standardisierte Akteure definiert werden, die zu unterschiedlichen Prozessphasen Entscheidungen treffen können, die für die Installation solaraktiver Fassadenkomponenten relevant sind. In AP4 wurden Produktmodelle für aktive Gebäudehüllkomponenten geschaffen und mit den Ergebnissen von AP3 kombiniert. AP5 schließlich befasste sich mit der prototypischen Umsetzung, AP6 mit der Validierung der erreichten Ergebnisse. Der zeitliche Ablauf ist in Abbildung 1 dargestellt.

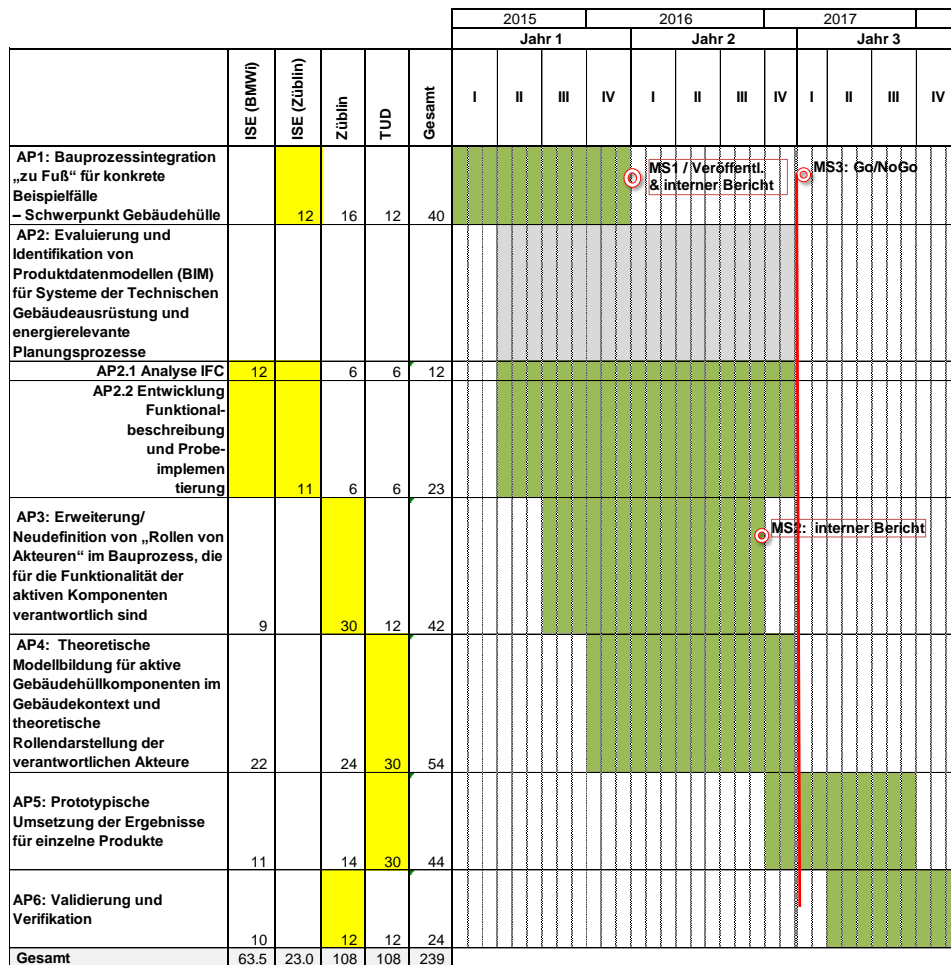


Abbildung 1: GANTT-Chart des Projekts

Während AP1 und AP2 zunächst entsprechend der ursprünglichen Planung durchgeführt werden konnten, sorgten deren Ergebnisse und die Erkenntnisse in AP3 für die Notwendigkeit, die Projekthinhalte speziell zu AP3 und AP4 an den neuen Kenntnisstand anzupassen. Auch AP5 und AP6 wurden thematisch erweitert. Auf die genannten Änderungen im Projektablauf wird ausführlich in Kapitel 2 eingegangen.

1.5 Stand der Wissenschaft und Technik

Die Projekthinhalte vereinen die Themen solaraktive Fassadenkomponenten und *Building Information Modeling* (BIM).

Fassadenkomponenten (bzw. allgemeiner Gebäudehüllkomponenten) können auf verschiedene Weisen energetisch aktiv sein. Gebäudeintegrierte Photovoltaik (BIPV) sowie gebäudeintegrierte Solarthermie (BIST) sind zwei Optionen, regenerative Energie direkt im Gebäude zur Verfügung zu stellen.

Gebäude besitzen eine Schlüsselposition auf dem Weg zu einer Energieversorgung in Deutschland, die zu 100% auf lokalen erneuerbaren Energiequellen basiert. Gebäude sind einerseits zu ca. 35% für den derzeitigen Energiebedarf verantwortlich, stellen aber andererseits auch enorme Flächen auf Dächern und Fassaden für die solare Energiewandlung zur Verfügung [1]. Besonders die Dächer und Fassaden von Neubauten bieten sich für die Installation von PV-Systemen an, da hier die solare Energiegewinnung in der Planung bereits integral berücksichtigt werden kann. Die Gebäudehülle definiert in starkem Maße die Effizienz eines Gebäudes. Energie-

tisch aktive Fassadenkomponenten müssen jedoch nicht nur im Hinblick auf maximalen Energieertrag optimiert werden, sondern sind immer multifunktionale Bauprodukte, die zunächst klassische Aufgaben der Gebäudehülle erfüllen müssen, u.a. auch energetisch relevante Funktionalitäten wie Sonnenschutz, Tageslichtnutzung oder thermische Isolation. Zahlreiche Pilotprojekte und Demonstratoren haben gezeigt, dass mit solaraktiven Fassadenkomponenten eine drastische Reduktion des Primärenergiebedarfs von Gebäuden technisch möglich ist. Produkte aus dem Bereich BIPV und BIST für verschiedene Anwendungsbereiche sind am Markt verfügbar, aber trotz zahlreicher Leuchtturmprojekte sind Planung, Bau und Betrieb von energetisch aktiven Gebäudehüllen nach wie vor eine komplexe Aufgabe, die großen Zusatzaufwand gegenüber der Verwendung herkömmlicher Baukomponenten erzeugt.

Die Digitalisierung des Bauwesens ist leider nach wie vor wenig fortgeschritten. Baufirmen beschäftigen sich mehrheitlich mit der Anwendung und Verbesserung von bestehenden BIM-Software-Lösungen, die aufgrund der fachlichen Distanz zwischen Software-Entwicklern und bauseitigen Anforderungen nur begrenzt ihren Zweck erfüllen und somit dafür sorgen, dass BIM-Projekte noch immer erheblich teurer in der Umsetzung sind als konventionell durchgeführte Bauprojekte. Da im Moment noch keine bauseitige Digitalisierung stattfindet, führen digitale Methoden bisher nur selten zu Effizienzsteigerungen von Baustellenabläufen.

Zum Zeitpunkt des Projektstarts wird in der Fachwelt verstärkt die Strategie verfolgt, mittels Standardisierung von Bauprozessen die Durchschlagskraft der Digitalisierung zu steigern. Hierfür werden von der im Bauwesen einflussreichen Organisation buildingSMART (ehemals *International Alliance for Interoperability*, IAI) sog. *Information Delivery Manuals* (□ *IDMs*) vorgeschlagen, die 2010 zur ISO-Norm erhoben wurden. Auch das von buildingSMART entwickelte Austauschformat für geometrische Informationen, die *Industry Foundation Classes* (□ *IFC*), wurden einer detaillierten Untersuchung unterzogen. Das Ziel dieses Formats besteht darin, den Informationsaustausch zwischen zwei BIM-Softwareprogrammen zu standardisieren, um die Anzahl der notwendigen bilateralen Absprachen zu reduzieren.

Der Stand der Wissenschaft und Technik hinsichtlich der digitalen Unterstützung von baurelevanten Abläufen besteht somit darin, eine Abfolge von Aktivitäten zu definieren, deren anfallende Informationen mithilfe eines standardisierten Austauschformats von einem Softwareprogramm zum nächsten übertragen werden. Zur technischen Umsetzung der Herangehensweise wurden von buildingSMART weitere Formate definiert: *Model View Definitions* (□ *MVDs*) sorgen für eine Vorfilterung der aus einem Softwareprogramm exportierten Daten, während das *buildingSMART data dictionary* (□ *bsDD*) gewerkeübergreifende Begriffsdefinitionen festlegt. Deshalb die vorgeschlagene Herangehensweise zur digitalen Unterstützung von Bauabläufen bisher so wenig Umsetzung in der Praxis erfahren hat, konnte durch die Ergebnisse des vorliegenden Forschungsprojekts aufgedeckt werden. Für deren Beschreibung muss auf Kapitel 2 verwiesen werden.

Zur digitalen Wiedergabe von Bauteilen wurden zwei Formate einer genaueren Analyse unterzogen. Das von buildingSMART vorgeschlagene SimpleIFC beinhaltet keine Möglichkeit zur gleichungsbasierten Geometriedefinition („Parametrik“) und eignet sich deshalb nicht als Format zur software-unabhängigen Bauteilbeschreibung. Im Bereich der Technischen Gebäudeausrüstung (TGA) kommt das VDI-Format zum Einsatz. Dieses Format wäre für die Projekthinhalte prinzipiell geeignet, wurde aber aufgrund der veralteten Technologie und fehlenden offenen Parsingmöglichkeiten nicht weiterverwendet.

Stattdessen wurde das Projektkonsortium hinsichtlich geometrischer Beschreibungen im W3C-Umfeld fündig. Browserbasierte 3D-Visualisierungen (WebGL) existie-

ren bereits seit 2011, und im Semantic-Web-Kontext wurden Geometriekernels entwickelt.¹ Mit ifcOWL beginnt auch buildingSMART seit 2015, ihre BIM-Aktivitäten koordiniert mit der Anwendung von Webtechnologien zu verbinden, wenn auch mit den oben genannten Einschränkungen, die sich aus der Verwendung des IFC-Formats ergeben.

1.6 Zusammenarbeit mit anderen Stellen

1.6.1 buildingSMART

Vor allem zu Beginn des Forschungsprojekts war der fachliche Austausch mit Externen zum Thema BIM von erheblicher Bedeutung. Die Fachdiskussionen konzentrieren sich in Deutschland vor allem auf das buildingSMART-Umfeld, weshalb wir an einigen Konferenzen teilgenommen haben. Dort kam es zu etlichen, für die technische Ausrichtung des Projekts wesentlichen Gesprächen mit führenden Fachexperten. Die erste Projektphase war von dem Versuch geprägt, die in buildingSMART entwickelten technischen Konzepte für die Erfüllung der Projektziele heranzuziehen. Bald aber musste festgestellt werden, dass diese an mehreren Stellen den allgemeinen technischen Anforderungen nicht genügen. Produktbeschreibungen auf Basis von □ *IFC* sind technisch ungenügend, da die Parametrik ausgespart wird; die vorgeschlagenen Filtermethoden mittels □ *MVD* entsprechen bei weitem nicht den Anforderungen, die in AP3 festgestellt wurden, und auch das □ *IDM*-Konzept war wegen der in diesem fest vorgegebenen Rollendefinitionen mit den Ergebnissen aus AP1 nicht vereinbar. Der schließlich gefasste Entschluss, die von buildingSMART eingeschlagene technische Richtung vollständig aufzugeben, wäre hingegen ohne ausführliche Diskussionen mit führenden buildingSMART-Vertretern nicht umsetzbar gewesen, da uns ohne den ausführlichen Austausch das Verständnis für die technischen Anforderungen gefehlt hätte. Spätestens mit der buildingSMART-International-Konferenz in Barcelona 2017 haben wir auch die Hoffnung aufgeben müssen, deren Aktivitäten in Richtung der Bedürfnisse der Bauindustrie zu korrigieren. Für den Austausch mit Fachkollegen ist eine Teilnahme an deren Veranstaltungen dennoch weiterhin zu empfehlen.

Im Umfeld von buildingSMART entstand zudem eine *Linked Data Working Group* (LDWG), die sich mit dem Einsatz von Semantic-Web-Technologien im Bauwesen befasst. Nach der Entscheidung des Forschungsprojektes, sich auf diese Technologien zu stützen, sind wir in Kontakt zur LDWG getreten. Entwicklungen der Gruppe waren, die IFC in Ontologien zu überführen. Allerdings wurde die Gruppe während der Projektlaufzeit aufgrund mangelndem Supports seitens buildingSMART inaktiv, sodass wir uns an eine entsprechende Gruppe des W3C gewandt haben.

1.6.2 W3C

Der Projektverlauf zeigte die Notwendigkeit auf, für die software-unabhängige Informationsverwaltung Webtechnologien anzuwenden. Aus diesem Grund wurde der Kontakt zur *Linked Building Data Community Group* (LBDCG) des *World Wide Web Consortiums* (W3C) aufgenommen. Die LBDCG befasst sich mit dem Verfassen von Empfehlungen, wie Semantic-Web-Technologien wie bspw. Ontologien und Linked Data im Bauwesen Anwendung finden können. Aus diesen Empfehlungen resultieren Ontologien, die von Anwendern und Forschern für ihre eigenen Zwecke wieder verwenden oder adaptieren können. Die Gruppe arbeitet für einen intensiven Austausch der Anforderungen der Bauindustrie auch mit buildingSMART zusammen,

¹ Siehe z. B.: <http://rdf.bg/>

Eingehende Darstellung der Ergebnisse	wobei sie sich nicht von diesem bestimmen lässt. Für den Austausch mit der LBDCG wurden Workshops besucht und an zweiwöchentlich stattfindenden Telefonkonferenzen teilgenommen. In Zukunft plant die LBDCG sich zu einer <i>Working Group</i> umzudefinieren, was die Teilnahme an der Gruppe verbindlicher gestaltet. Inwiefern sich die Mitarbeit mit einer <i>Working Group</i> aus Forschungsprojekten heraus realisieren lässt, ist für zukünftige Projekte abzuwarten.
---	---

2 Eingehende Darstellung der Ergebnisse

2.1 Arbeitspaket 1 – Bauprozessintegration „zu Fuß“ für konkrete Beispielfälle – Schwerpunkt Gebäudehülle¹

Für eine Bauprozessintegration der Installation solaraktiver Fassadenkomponenten ist es zunächst von großer Wichtigkeit, die derzeitigen Bauprozessabläufe zu untersuchen und, wenn möglich, zu standardisieren. Zu diesem Zweck wurden folgende Installationen solaraktiver Fassadenkomponenten von realen Bauprojekten für eine genauere Prozessanalyse herangezogen:

1. Der solaraktive Sonnenschutz im Z3-Gebäude am Stuttgarter Züblin-Campus, der im Rahmen des Forschungsprojekts EnOB REG II [2] hinzugefügt wurde,
2. die opaken Photovoltaikmodule von *ertex solartechnik GmbH*, zu installieren in der Fassade von Gebäude N am Fraunhofer-Institut für Solare Energiesysteme ISE in Freiburg i. B.,
3. die transparenten, verglasungsintegrierten thermischen Fassadenkollektoren und Vakuum-Luftkollektoren in der Fassade eines Gebäudes in Ljubljana / Slowenien,
4. die winkelselektiven transparenten BIPV-Verglasungen *PVShade*, die im Brüstungsbereich einer Ganzglasfassade angebracht wurden,
5. die opaken PV-Module mit MWT-Zellen und innovativem Modulkonzept (*TPEdge*), die in einer vorgehängten Fassade am Gebäude R, ebenfalls Fraunhofer ISE, angebracht wurden,
6. das semitransparente BIPV-System mit Glasprismen der Firma *SolarOr*, Israel (generalisierte Prozessbetrachtung),
7. die PCM-Verglasung mit statischen Prismen der Firma *GlassX* (ebenfalls mit generalisierter Prozessbetrachtung).

Die Bauprozesse zu den Installationen in 1, 3, 4 und 5 konnten besonders detailliert untersucht werden, da die zugehörigen Projekte bereits abgeschlossen wurden und die beteiligten Personen teilweise direkt befragt werden konnten. Projekt 2 wurde während der Prozessaufnahme durchgeführt, dessen Abläufe konnten also direkt beobachtet werden.

Zu den aufgelisteten Bauprojekten wurde jeweils ein BPMN-Prozessdiagramm (ein Standard zur Beschreibung von Prozessen, *Business Process Modeling Notation*), erstellt, die im ersten Meilensteinbericht abgebildet sind. Es war deutlich zu beobachten, dass die Komplexität der Prozessabläufe selbst bei grob vereinfachter Wiedergabe für eine Prozessstandardisierung eine Herausforderung darstellen wür-

¹ Die Ergebnisse aus AP1 sind ausführlich im ersten Meilensteinbericht beschrieben, weshalb sich das vorliegende Kapitel auf eine kurze Zusammenfassung beschränkt.

de. Dieser Eindruck wurde durch die Ergebnisse der Erstellung von BIPV- und BIST-Overlays noch verstärkt.

In der Untersuchung der von buildingSMART erarbeiteten □ *IDM*-Konzeption in AP3 wird aus den in diesem Arbeitspaket gesammelten Daten die Schlussfolgerung gezogen werden, dass die Herangehensweise der Prozessstandardisierung selbst zu hinterfragen ist. Für den Projektverlauf stellen die Ergebnisse dieses Arbeitspakets deshalb eine wesentliche Informationsgrundlage dar.

2.2 Arbeitspaket 2 – Evaluierung und Identifikation von Produktdatenmodellen (BIM) für Systeme der Technischen Gebäudeausrüstung und energierelevante Planungsprozesse¹


Das Ziel des zweiten Arbeitspakets bestand in der Evaluierung bestehender Formate für die Beschreibung von Bauprodukten im Rahmen des Building Information Modeling zur Ermöglichung der Durchführung von energierelevanten Planungsprozessen. Ein naheliegender erster Schritt bestand in der Untersuchung des im Bauwesen mit Abstand verbreitetsten applikationsunabhängigen Formats, den *Industry Foundation Classes* (□ *IFC*). Dabei stellte sich heraus, dass die hierarchische Gliederung des Dateiformats bei der Beschreibung komplexer Bauprodukte für erhebliche Schwierigkeiten sorgen kann. Zunächst wurde untersucht, inwiefern BIPV-Module mittels IFC beschrieben werden können. Da diese aber sowohl Bauelemente als auch Solarmodule darstellen und deren Zuordnung zu beiden Kategorien relevant sein kann, entstand die Idee der Etablierung einer neuen IFC-Klasse namens *IFCActiveBuildingComponent*, unterhalb von *IFCElement* und parallel zu *IFCBuildingElement* angeordnet [3]. Ein weiterer interessanter Befund bestand darin, dass die IFC-Beschreibungsmöglichkeiten weit hinter den für die technische Systemauslegung der Elemente notwendigen Detaillierungsgrad zurückfallen. Der Austausch mit führenden buildingSMART-Vertretern ergab das Fazit, dass dem Detaillierungsgrad der Produktbeschreibungen Grenzen auferlegt wären, da diese sonst in proprietären BIM-Softwareprogrammen nicht verwendet werden könnten. Es erwies sich also, dass IFC-Beschreibungen für solaraktive Fassadenkomponenten entweder in den gängigen BIM-Programmen und/oder für grundlegende technische Auslegeberechnungen nicht verwendbar sind.

Aus den genannten Punkten ergibt sich die Schlussfolgerung, dass es eine applikationsunabhängig verwaltete Informationsmenge geben muss, die sowohl für die Verwendung in gängigen BIM-Softwareprogrammen als auch für die technischen Auslegungsprogramme aufgearbeitet werden muss. Es besteht also die Notwendigkeit einer vollständigen, software-unabhängigen Datenhaltung mit applikationsbezogenen Filter- und Vorverarbeitungsmethoden vor Software-Import. Beide Anforderungen wurden im Projektverlauf ausführlich untersucht, wobei sich herausstellte, dass die Vollständigkeitsanforderung durch IFC strukturell nicht erfüllt ist. Es wird von buildingSMART vorgegeben, dass es sich bei IFC um ein statisches Austauschformat handelt. Für dynamische Beschreibungen, v. a. für Bauprodukte, fehlt die Möglichkeit der gleichungsbasierten Geometriedefinition („Parametrik“). Eine solche aber würde einen Standard für die Beschreibung von Geometriekernelfunktionen erfordern, wofür buildingSMART keine Umsetzung anstrebt.

¹ Auch die Ergebnisse des zweiten Arbeitspakets sind ausführlich in einem Meilensteinbericht (Nr. 3) zusammengefasst, weshalb die Inhaltsbeschreibung hier kurz gehalten wird.

Für die Umsetzung der Projekthinhalte war es also erforderlich, ein Format für die applikationsunabhängige und vollständige Wiedergabe geometrischer Informationen zu finden. Schließlich wurde im Projekt der notwendige Schritt unternommen, von IFC auf Webtechnologien umzustellen, woraus sich die Zusammenarbeit mit dem *World Wide Web Consortium* (W3C) anschloss. Daraus ergab sich gleichzeitig eine Möglichkeit, auf hierarchische Strukturen in der Datenhaltung zu verzichten und auf semantische Beschreibungen überzugehen. Diese Arbeiten sind dem vierten Arbeitspaket zuzuordnen.

2.3 Arbeitspaket 3 – Erweiterung/Neudefinition von „Rollen von Akteuren“ im Bauprozess, die für die Funktionalität der aktiven Komponenten verantwortlich sind¹

Im dritten Arbeitspaket wurde detailliert untersucht, inwiefern eine Standardisierung von Bauprozessen die Integration solaraktiver Fassadenkomponenten technisch unterstützen kann. Es ist nämlich häufig zu beobachten, dass sich aufgrund von früh im Bauprozess getroffenen Entscheidungen (v. a. architektonischer Art) entsprechende Installationen deutlich schwieriger gestalten können. Zur Erfüllung der Ziele dieses Arbeitspakets war es naheliegend, zunächst die buildingSMART-Konzeption der *Information Delivery Manuals* ( *IDM*) heranzuziehen. Es zeigte sich jedoch, dass die Varianz der im ersten Arbeitspaket dargestellten Bauprozesse eine Standardisierung auf BPMN-Basis, wie in den IDMs vorgeschlagen, nicht zulässt. Für das Projektkonsortium noch überraschender aber war die Beobachtung, dass die in der Untersuchung der realen Prozessabläufe festgestellten Herausforderungen in der Konzeptionierung der IDMs gar nicht bedacht wurden. Dazu gehören die Schwierigkeit einer Standardisierung der Rollenzuordnung zu Aktivitäten, deren Verantwortlichkeiten in der Realität je nach beteiligten Firmen, technischem Komplexitätsgrad oder auch Beauftragungskette stark schwanken, oder die zeitliche Einordnung eines Optimierungsprozesses, die stark von technischen Details des Bauprojekts abhängig ist (die den im IDM ausgeführten Detaillierungsgrad aber überschreiten). Die in anderen Industriezweigen vorhandene Automatisierbarkeit ist im Bauwesen nur begrenzt gegeben, weshalb die inhaltliche Form der IDMs nicht die Anforderungen des Bauwesens erfüllt.

Nach längeren Diskussionen ergab sich die Schlussfolgerung, dass aus den Prozessdarstellungen sowohl der Zeit- als auch der Firmenbezug gestrichen werden muss, wenn eine Standardisierung erfolgen soll. Die beobachteten Bauabläufe müssen manchmal in umgekehrter Reihenfolge verlaufen, während die Durchführung an den vorigen Prozessschritt gebunden sein kann, aber nicht immer gebunden sein muss. Für eine Standardisierung muss auch der Firmenbezug offengelassen werden, da manche Prozessschritte von unterschiedlichen Beteiligten in variabler Form übernommen werden müssen.

Hinsichtlich auch der Ergebnisse des zweiten Arbeitspakets bedeutet dies die Notwendigkeit der Realisierung einer firmen- und software-unabhängigen, vollständigen Datenhaltung. Von dieser ausgehend werden Bauprozessschritte standardisiert durch Filterung der für die jeweilige Anwendung notwendigen Informationen, der Durchführung des Arbeitsschrittes und der Rückführung dabei entstandener Informationen in die software-unabhängig gehaltene Datenmenge (letztere wurde im vorliegenden Forschungsprojekt aus Komplexitätsgründen ausgespart). Die dabei

¹ Eine detaillierte Inhaltsbeschreibung des dritten Arbeitspakets ist im zweiten Meilensteinbericht zu finden.

notwendige Filterung und Vorverarbeitung des Datenpools für die Applikation erfordert dabei eine von der buildingSMART-Konzeption *Model View Definition* (□ MVD) grundlegend verschiedene Struktur. Die MVDs sind bereits Bestandteil einer Applikation, was eine sequentielle Arbeitsweise notwendig macht. Die im Forschungsprojekt gesammelten Erkenntnisse machen aber eine Filterung und Vorverarbeitung vor der Applikation erforderlich.

Ein Beispiel soll den Unterschied der im Projekt identifizierten technischen Anforderungen zur Filter- und Vorverarbeitung zum buildingSMART-Konzept MVD verdeutlichen. Für die Planung solaraktiver Fassadensysteme sind meist umfangreiche Berechnungen erforderlich, die eine gute Abgrenzung der benötigten Datenmenge erfordern. Ausgehend von einer beliebig großen Datenmenge zur Beschreibung eines Gebäudemodells werden unter anderem folgende Abfragen benötigt:

- Entfernung von Objekten, die für eine Verschattung oder Reflexion von Lichtstrahlen auf das betreffende Gebäude nicht infrage kommen
- Verringerung der geometrischen Detaillierung von Objekten, die nicht zum betreffenden Gebäude gehören
- Vereinfachung der Geometrie der externen Objekte des betreffenden Gebäudes, indem jeweils der gebäudeintern liegende Objektteil weggeschnitten wird
- Entfernung aller gebäudeintern liegenden Objekte
- Auffinden der meteorologischen Daten und der genauen geographischen Lage
- Auffinden der Reflexionseigenschaften aller verbleibenden Objekte
- Entfernung aller Objekteigenschaften mit Ausnahme der optischen Eigenschaften

Das MVD-Format berücksichtigt entsprechende Abfragemöglichkeiten nicht, da es ausschließlich zur klassenbasierten Filterung gedacht ist (d. h. einen anderen Zweck erfüllt, nämlich denjenigen der Verbesserung der Informationsübertragung zwischen Softwareprogrammen, die in der Beschreibung des Gebäudemodells mit einem ähnlichen Detaillierungsgrad arbeiten). Erst mit diesen Abfragemöglichkeiten können Berechnungsprogramme angedockt werden, die zur Umsetzung solaraktiver Fassadensysteme unumgänglich sind.

Mit der Feststellung der Notwendigkeit umfassender Filter- und Vorverarbeitungsroutinen ist gleichzeitig eine Grundlage geschaffen für die Behebung der im ersten Arbeitspaket identifizierten Schwierigkeiten der Prozessstandardisierung. Die „Rollen von Akteuren“ bestehen also in der Anwendung von Programmcode für Datenvollständigkeitsprüfung und Datenaufarbeitung für jeden Prozessschritt, während generalunternehmerseitig Datenverwaltungssysteme etabliert werden müssen. Anders formuliert: buildingSMART strebt eine sequentielle Prozessstandardisierung an, d. h. eine Veränderung derzeitiger Bauabläufe zu einer generalisierten Reihenfolge der Abläufe. Im Forschungsprojekt hingegen werden die digitalen Methoden adaptiert an die bestehenden Bauabläufe. Die Festlegung einer Reihenfolge sowie einer Zuordnung von Aktivitäten zu Firmenkategorien wird vermieden. Dadurch entsteht die Notwendigkeit einer software-unabhängigen Datenhaltung, die die Entstehung eines Korsetts an sequentiellen Bauabläufen unterbindet und eine firmenübergreifend parallele Datenverarbeitung sowie die Adaptierung der Bauabläufe an den jeweiligen Informationsstand zulässt und anstrebt.

Für das Forschungsprojekt ergeben sich dadurch folgende Erfordernisse. Erstens müssen, im Sinne der „Rollen von Akteuren“, die Anforderungen einer software- und firmenübergreifenden Datenhaltung dargestellt werden. Dafür werden IT-Konzepte diskutiert, die deren technische Umsetzung sowohl hinsichtlich Datenbereitstellung als auch auf Verwaltungsebene (Nutzer-, Änderungsmanagement) erlauben. Zweitens müssen sowohl für geometrische als auch für nicht-geometrische Informationen Formate gefunden werden, die der Anforderung einer vollständigen Datenwiederga-

2.4 Arbeitspaket 4 – Theoretische Modellbildung für aktive Gebäudehüllkomponenten im Gebäudekontext und theoretische Rollendarstellung der verantwortlichen Akteure

Die Ergebnisse der vorangegangenen Arbeitspakete – insbesondere von Arbeitspaket 3 – haben starke Einflüsse auf die Entwicklung eines Konzepts zur Unterstützung bei der Integration energetisch aktiver Fassadenkomponenten in Bauprozesse. Um auf die Erkenntnisse korrekt reagieren zu können, wurde die Aufgabenstellung dieses und der folgenden Arbeitspakete angepasst.

Die Aufgabenstellung besteht hierdurch weniger in der bisher in der Thematik BIM hochgehaltenen „gemeinschaftlichen“ Arbeitsweise und der Standardisierung von Prozessen als vielmehr in der formal ausgeprägten Abkopplung der einzelnen Arbeitsschritte voneinander. Die Grundvoraussetzung für die Umsetzung dieses Prinzips ist eine applikationsunabhängige Datenhaltung, die es in dieser Form im Baukontext zurzeit nicht gibt. Die genannte Herangehensweise unterscheidet sich zudem grundlegend von den in der Organisation buildingSMART entwickelten Konzepten, da der Vorverarbeitungsprozess der Datenmengen nicht während des Exports, sondern während des Imports in Software-Applikationen stattfinden muss.

Entsprechend wurde von ursprünglich thematisierten Technologien wie BPMN Prozessmodellierungen und Multi-Agenten-Systemen zur automatisierten Durchführung der Prozesse abgesehen und neue Technologien, die im folgenden Unterkapitel vorgestellt werden, eingebracht. Als Folge der Anpassung des Projekt-Fokus, sind die Arbeitspakete 5 und 6 ebenfalls abgeändert worden. So betrifft das Arbeitspaket 5 nun die Implementierung der im Arbeitspaket 4 vorgestellten Grundkonzepte während Arbeitspaket 6 die Validierung der Methodik durch exemplarische Implementierungen verschiedener Anwendungsfälle beinhaltet.

2.4.1 Betrachtete Methoden, Systeme und Technologien

2.4.1.1 Semantic-Web-Technologien im Bauwesen

Semantic-Web-Technologien setzen das Ziel Informationen nicht nur maschinenlesbar, sondern maschinenverständlich aufzubereiten. Um dies zu erreichen, müssen die Begrifflichkeiten nicht nur eindeutig gewählt werden, sondern auch in ihrem Kontext definiert werden. Als Kontext verstehen sich hier die Beziehungen zwischen verschiedenen Kontexten und deren Eigenschaften. Bspw. besitzt eine Wand den Kontext eines Gebäudeelements. Objekte können allerdings in mehreren Kontexten bekannt sein; auf das Beispiel der Wand bezogen, kann diese zudem in Kontexten der thermischen Hülle oder der Raumgrenzen bekannt sein. Eine Abbildung dieses Beispiels ist Abbildung 2 zu entnehmen. Im Falle des Kontexts der thermischen Hülle gilt zudem die Bedingung, dass die Wand ein Außenbauteil ist. Solche Bedingungen können in einer Ontologie definiert und anschließend auf Dateninhalte automatisiert angewandt werden. Die entsprechende Definition eines Ontologieschemas findet in der sog. *T-Box* (T: *Terminology*) statt.

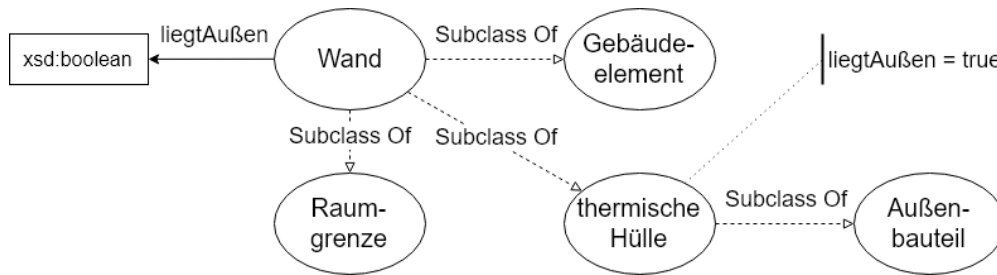


Abbildung 2: Beispiel einer T-Box

Anhand eines vorgegebenen Schemas können Informationen beschrieben werden. Es werden jedoch nicht wie in objektorientierten Programmiersprachen die Klassen mit ihren Eigenschaften definiert, sondern lediglich Konzepte bzw. Klassen (Knoten) und Relationen zwischen den Knoten. Da hierfür die *Open World Assumption* (OWA) gilt, können Knoten auch nicht im Schema definierte Relationen zu anderen, ggf. ebenfalls im Schema unbekannten Knoten oder Literalen (Werten) zugewiesen werden. Somit wird eine flexible Modellierung von Informationen möglich, obwohl zuvor ein Schema definiert wurde, das einen einheitlichen Umgang mit verschiedenen Informationen erlaubt. Informationen werden zu diesem Zweck in der sog. *A-Box* (A: Annotation) abgelegt. Dabei wird für jedes Objekt ein *Individuum* einer Klasse erzeugt, das anschließend mit Relationen zu Literalen oder anderen Individuen ergänzt wird. Die Relation *typeOf* ist für die Relation zwischen Individuum und Klasse reserviert. Anhand des zuvor genannten Beispiels ergibt sich bspw. eine in Abbildung 3 dargestellte A-Box.

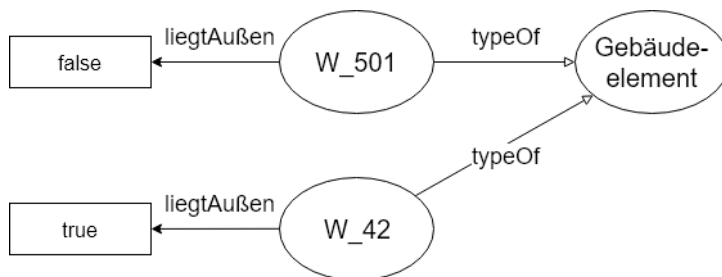


Abbildung 3: Beispiel einer A-Box

Auf Basis des zuvor definierten Schemas (T-Box) können neben den tatsächlich beschriebenen Informationen weitere abgeleitet werden. Die Individuen sind lediglich mit der *Gebäudeelement*-Klasse verbunden, können jedoch durch die Verwendung der *liegtAußen*-Relation, die per Definition zu der *Wand*-Klasse gehört, ebendieser Klasse zugewiesen werden. Zudem können beide Individuen durch *Reasoning* (Auswerten der Zusammenhänge) als Individuen der Klasse *Raumgrenze* klassifiziert und zudem kann das Individuum *W_42* als Individuum der Klassen *thermische Hülle* und *Außenbauteil* erkannt werden. Eine solche Erkennung von Zugehörigkeiten aufgrund von *Reasoning* nennt sich auch *inferieren*.

Semantic-Web-Technologien werden bereits im Bauwesen untersucht und sind Bestandteil laufender Forschungen. Nennenswerte Entwicklungen zu diesem Thema sind z.B. die *ifcOWL* [4], eine Überführung des STEP-basierten IFC-Schemas in eine Ontologie, die *ifcWOD* [5], die ebenfalls auf dem IFC-Schema basieren, die *BOT* [6], die als grundlegende Topologie für digitale Gebäudemodelle entwickelt wird. Zudem gibt es von der W3C LBDG Bemühungen, Ontologien zur Abbildungen von Produkten (*PROD* [7]), Eigenschaften (*PROPS* [8], *OPM* [9]), und Geometrien (*GEOM* [10]) zu definieren und publizieren. Weitere Forschungen, die Semantic-Web-Technologien einbeziehen können [11] entnommen werden.

Im Forschungsprojekt wurden speziell Ontologien zur Abbildung von Geometrien betrachtet. Hierzu zählen die *GEOM*-Ontologie, die als der Teil *CMO with extensions* Ontologie [12] in dem EU-Forschungsprojekt PROFICIENT entwickelt wurde, und die *OntoBREP*-Ontologie [13], die für Anwendungen in der Robotik von der TU München vorgestellt wurde. Mit der *GEOM*-Ontologie können Geometrien in verschiedenen Kontexten beschrieben werden, u. a. in *Constructive Solid Geometry* (CSG) oder auch *Boundary Representation* (BRep). Die Interpretation und Auswertung der Beschreibung kann durch einen von der Firma RDF Ltd. zur Verfügung gestellten, aber nicht quellcode-offenen Geometrie-Kernel erfolgen. Für die Visualisierung der Geometrie bietet RDF Ltd. zudem eine Desktop Anwendung sowie einen Webservice zur eigenen Anwendung an. Da diese jedoch auf dem geschlossenen Geometrie-Kernel basieren, können sie nur bedingt auf die eigenen Anwendungsfälle angepasst werden. Ein weiterer Teil der *CMO-with-extensions*-Ontologie, den wir betrachtet haben, beschäftigt sich mit der Abbildung mathematischer Zusammenhänge für bspw. parametrische Beschreibungen. Die Auswertung der Zusammenhänge findet jedoch ebenfalls in einem frei zugänglichen aber nicht quellcodeoffenen Mathematik-Kernel statt. Als Vergleich zu der *GEOM*-Ontologie wurde die *OntoBREP*-Ontologie betrachtet. Diese beschränkt sich derzeit auf Oberflächenbeschreibungen von Geometrien und bietet entsprechend eine niedrigere Vielfalt an Möglichkeiten zur Geometriedefinition.

2.4.1.2 Demilitarisierte Zonen

In einem fortgeschritten digitalisierten Bauablauf wird jeder Baubeteiligte Daten erstellen, die für andere relevant sein können, und wird auch seinerseits Zugriff auf bestehende Informationen benötigen. Es ist aus diesem Grund naheliegend, im Forschungsprojekt SolConPro die im World Wide Web bereits etablierten technischen Kommunikationsstrukturen zum Austausch von dezentralisierten, aber applikationsunabhängig gespeicherten Informationen heranzuziehen. Jeder Baubeteiligte stößt im Zuge der Datenbereitstellung und -verarbeitung auf folgende Anforderungen:

- Die bereitgestellten Daten müssen Sicherheitsanforderungen genügen, d. h. sie dürfen bspw. nicht von Unbefugten verändert oder gelöscht werden können. Ein entwickeltes Zugriffsrechtssystem wird in jedem Fall notwendig sein, weshalb ein Rückgriff auf bestehende, z. B. firmeninterne Nutzerverwaltungssysteme sinnvoll erscheint.
- Langfristig werden auch automatisierte Abgleiche mit besonders zugriffsgesicherten Daten unvermeidlich sein, so etwa mit personenbezogenen Daten, Projektfinanzzahlen oder Firmengeheimnissen. Bestehende Firewall-Einstellungen stehen zurzeit aber solchen Abgleichen im Weg.

Eine Cloud-Umsetzung der Datenbereitstellung berücksichtigt obengenannte Anforderungen bisher nur begrenzt. Nutzerverwaltungssysteme können unter Umständen vom Firmenintranet auf Cloudsysteme übertragen werden, was aber den Nachteil hat, dass sensitive Informationen außerhalb der Firewall gelagert werden. Zudem müssen die Firewall-Einstellungen des Intranets einen Datentransfer zur externen Cloud ermöglichen, was die Sicherheit des Intranets gegenüber externem Zugriff beeinträchtigen kann.

Ein vielversprechendes Konzept zur Bewältigung der genannten Herausforderungen nennt sich *Demilitarisierte Zone* (DMZ). Es handelt sich hierbei um Webserver, die sowohl von innerhalb als auch von außerhalb einer Firewall bidirektional und automatisiert zugänglich sind und dennoch einen Bestandteil der firmeninternen IT-Infrastruktur bilden. In Abbildung 4 ist skizzenhaft das Verhältnis zwischen einer DMZ zur Firewall dargestellt.

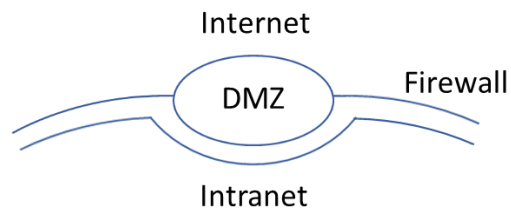


Abbildung 4: Verhältnis von Demilitarisierten Zonen (DMZ) zu Intranet und Firewall

Der DMZ-Webservice befindet sich außerhalb der Firewall, dennoch ist sein Aufbau an bestehende Intranet-Applikationen angelehnt und wird mit diesen gemeinsam verwaltet. Dies hat zur Folge, dass bereits bestehende technische Herangehensweisen, etwa das Load Balancing, die Datenarchivierung, die Sicherung der Serverbereitstellung durch multiple Instanzen etc. parallel auf Webapplikationen von Intranet und DMZ angewandt werden können. Dies vereinfacht zudem die Adaptierung der Firewall für Fälle, die einen direkten automatisierten Informationsaustausch erfordern.

Für das Deployment von Webservices in der Ed. Züblin AG, sei es für Intranetanwendungen oder in der DMZ, kommen zwei Technologien zum Einsatz, die sich beide im IT-Umfeld in den letzten Jahren etablieren konnten: [□ Docker](#) und [□ Gitlab](#). Beide sind für die Errichtung von DMZ-Anwendungen der Ed. Züblin AG unverzichtbar, weshalb deren Beschreibungen dem Glossar (Anhang A) beigelegt sind.

2.4.2 IT-Architektur

2.4.2.1 Projektplattformen

Im Thema Digitalisierung ist die zugrundeliegende IT-Architektur von großer Bedeutung. Während in anderen Branchen, vor allem bei Fließbandautomatisierung, ein bedeutender firmeninterner IT-Bezug existiert, der teilweise eigene Software-Entwicklung, teilweise bessere Adaptionen bestehender Softwarelösungen an die branchentypischen Anforderungen zufolge hat, besteht im Bauwesen eine große Lücke zwischen (meist externer) Software-Entwicklung und baustellenseitigen Erfordernissen. Dennoch können IT-Werkzeuge für die Verbesserung der Effizienz von bauseitigen Abläufen von großer Bedeutung sein.

Derartige Werkzeuge werden seit der in den letzten Jahren zu beobachtenden deutlichen Verbesserung der Internetanbindung immer häufiger als Cloud-Lösungen angeboten. Unter einer „Plattform“ wird dabei eine Cloud-Anwendung verstanden, die zur firmenübergreifenden und automatisierten Verwaltung und Bearbeitung von Projektinformationen herangezogen werden kann. Im Rahmen von BIM-Projekten spielt hierbei die modell- und bauteilbasierte Arbeitsweise eine zentrale und führende Rolle. Gängige Plattformen sind derzeit beispielsweise *BIM360* [14], [□ Dalux](#) oder *Aconex* [15], die die Themen Dokumentenmanagement und 3D-Modellvisualisierung für mobile Anwendungen anbieten und in der Umsetzung individuelle Schwerpunkte aufweisen (*Dalux* erlaubt z.B. ein modellbasiertes Änderungsmanagement). Weitere wichtige Plattformen sind *think project!* [16] und *FusionLive* [17], deren Fokus im Dokumentenmanagement in der Verwaltung von 2D-Plänen besteht.

2.4.2.1 Anforderungen an Projektplattformen

Aufgrund der deutlich verbesserten Kommunikationsmöglichkeiten bei Zusammenarbeit auf Plattformbasis stehen in der Formulierung von Anforderungen für Projekt-

Eingehende Darstellung der Ergebnisse	plattformen häufig die gebotenen Funktionalitäten im Vordergrund, die derzeit auch in erster Linie über deren Anwendung entscheiden. Die erforderlichen oder gewünschten Anforderungen an Projektplattformen konnten in Zusammenarbeit mit verschiedenen Züblin-Unternehmenseinheiten ermittelt und festgehalten werden. Im Rahmen von BIM-Projekten spielen die modell- und bauteilbasierte Arbeitsweise, aber auch Themen rund um Daten- und Informationsmanagement, Workflows, Visualisierung sowie spezifische Anforderungen aus den Phasen Planung, Ausführung und Betrieb eine Rolle. Auch nicht-funktionale Anforderungen wie beispielsweise ein zeitgemäßes <i>Look and Feel</i> sind im Sinne einer angemessenen Akzeptanzquote zu berücksichtigen. Eine zunehmende Relevanz nehmen beispielsweise auch die Kriterien mobiles sowie standortübergreifendes Arbeiten ein. Alle Betrachtungen in diesem Themenfeld unterliegen hierbei technischen, rechtlichen und kaufmännischen Randbedingungen. Zudem ist zu beachten, dass neue Technologien und veränderte Arbeitsweisen eine fortlaufende Anpassung der Anforderungen erfordern. Es konnten in verschiedenen Bauprojekten Anforderungs- und Informationsanalysen durchgeführt und somit projektspezifische Anforderungsprofile definiert werden. Hierbei ist auch zu berücksichtigen, welche Informationen zu welchem Zeitpunkt und von welchen Beteiligten benötigt werden. Das Anforderungsprofil diene – unter ergänzender Berücksichtigung von übergeordneten Firmeninteressen – als Grundlage zur nachfolgenden Wahl einer oder mehrerer Projektplattformen. Für schlüssige, projektspezifische Lösungsarchitekturen konnten nach Bedarf Schnittstellen analysiert und ganzheitliche Projektkonzepte entwickelt werden. Um der Summe der Anforderungen zu entsprechen, müssen Produkte mitunter ein großes Funktionsspektrum aufweisen. Bereits in naher Zukunft aber werden die digitalen Anforderungen nicht mehr von einzelnen Plattformanbietern realisierbar sein, wodurch die Notwendigkeit eines plattform- bzw. software-unabhängigen Datenmanagements entsteht. Dies erfordert eine externe Zugänglichkeit der innerhalb dieser Plattformen generierten und verwalteten Informationen, um eine plattformübergreifende Synchronisierbarkeit der Daten zu gewährleisten. Von dieser in Kapitel 2.4.2.5 genauer beschriebenen Forderungen sind zwei Themenfelder abzugrenzen:
---	---

- Die Forderung nach firmeninternem Betrieb einer Cloud-Software. Hierbei erlaubt die Herstellerfirma der Software, für die Verwendung der Software eigene Rechnerarchitekturen heranzuziehen (sog. *On-Premise*-Lösung), wodurch für den Nutzer die Verwendung der Software nicht an die Notwendigkeit gekoppelt ist, seine Daten auf den Servern des Software-Anbieters zu speichern (sog. *Software As A Service*, SAAS). Die Unterscheidung hat dann Relevanz, wenn die auf der Plattform verwalteten Daten vertraulichen Charakter haben und es keine strikte Vereinbarung zum Datenschutz zwischen Unternehmen und Software-Anbieter gibt.
- Die Forderung einer *Common Data Environment* (CDE), wie in ISO19650 oder PAS1192 beschrieben. Hierbei handelt es sich um eine Maßnahme zur Verbesserung der Datenübersicht v. a. bei der Anwendung einer Kombination von Plattformen, die eine plattformsspezifische Datenverwaltung unterbinden soll. Auch mit dieser Maßnahme bleiben weitere Plattformen unter Umständen von einer Datensynchronisierung ausgeschlossen.

Eine Herangehensweise für die Erläuterung der wesentlichen thematischen Unterschiede zwischen software-unabhängiger Datenhaltung, *On-Premise*-Betrieb und CDE besteht in der Nennung der Datenzugänglichkeitskategorien, wie im Folgenden geschildert.

2.4.2.2 Datenzugänglichkeitskategorien

Die zentrale Bedeutung des applikationsunabhängig automatisierten Datenzugriffs und dessen Nichtbereitstellung durch proprietäre Software-Anbieter hat zu einer Einordnung der Datenzugänglichkeit in die folgenden vier Kategorien geführt:

1. Mittels quelloffener API abrufbare Daten

Die Grundvoraussetzung für Daten der Kategorie 1 ist das Vorliegen und die verlässliche Einhaltung eines Datenschemas, unabhängig davon, ob die Daten selbst dem *ASCII*-Code folgen (d. h. mit einem Editor lesbar gemacht werden können) oder nicht. Mit Hilfe dieses Datenschemas ist die applikationsunabhängige Erstellung von Parsern möglich. Für verbreitete Datenschemata sind diese im Internet verfügbar. Beispiele für die Datenkategorie 1 sind *HTML*, *CSS*, die Dateninhalte von [GitLab](#), [MongoDB](#), [Docker](#), *Hadoop* [18] u. v. a.

2. Mittels proprietärer API abrufbare Daten mit unbekanntem Schema

Stellen Software-Entwickler die API gleich mit zur Verfügung, so ist der automatisierte Datenzugriff nicht auf die Offenlegung des Schemas angewiesen. Für den Anwender entsteht eine Abhängigkeit in der Datenzugänglichkeit vom jeweiligen Softwarehersteller, die in Kategorie 1 nicht vorhanden ist. Der Parser kann vom Anwender nicht an seine Anforderungen angepasst werden. Datenzugriffskategorie 2 ist selten geworden, lebt aber z. B. in proprietären Datenbanken wie *MSSQL* fort. Teilweise werden auch proprietäre, kommandozeilenbasierte Programme mit Datenzugänglichkeit der vorliegenden Kategorie übergeben (z. B. *Jotne EDM* [19]).

3. Nur über eine GUI abrufbare Daten (mit oder ohne API)

Die effektivere Arbeitsweise, die durch die Ersetzung von Kommandozeilenbefehlen durch eine Arbeitsoberfläche (z. B. Applikationen mit GUI, auch Web-Frontend) entsteht, kann von Softwareherstellern dazu genutzt werden, Zugriffe auf die Kommandozeile generell zu unterbinden und den Datenzugriff für den Nutzer auf die Arbeitsoberfläche zu beschränken. Beispiele sind [Autodesk Revit](#), *RIB iTWO* [20], *Microsoft PowerPoint* sowie proprietäre *On-Premise*-Lösungen wie *KanBo* [21]. Die im Softwareprogramm erstellten Daten sind so gar nicht oder nur mehr über Umwege automatisiert abgreifbar (manche Hersteller, z. B. *Autodesk*, stellen programminterne APIs zur Verfügung, die jedoch keinen externen Datenzugriff erlauben).

4. Nur über eine per fremdem Webserver erreichbare GUI abrufbare Daten

Die zunehmende Rechnernetzwerk erlaubt es, GUIs nicht mehr lokal, sondern auf Fremdrechnern zu betreiben. Auf diese Weise können die Datenzugriffsmöglichkeiten zusätzlich vom Webservice-Betreiber abhängig gemacht werden. Beispiele sind SaaS-Produkte wie *Autodesk Forge* [22] oder *3D Experience* [23], aber auch *Word Online*, *Doodle* oder *Gmail*. Die Speicherung der mittels Webapplikation erstellten Informationen erfolgt zudem nicht mehr verschlüsselt lokal, sondern zentralisiert auf dem Rechner des Software-Betreibers, was diesem erhebliche Datenauswertungsmöglichkeiten verschafft.

Bei Webanwendungen sind für die Absicherung einer Datenhaltung nach Zugriffskategorie 1 rechtliche Vereinbarungen notwendig. Im Rahmen dieses Forschungsprojekts wurde ein erster Entwurf erstellt (siehe Kap. 2.4.2.5).

2.4.2.3 Auswirkungen des Datenaustauschs zwischen Plattformen auf die Datenzugänglichkeit

Der Notwendigkeit eines automatisierten Datenaustauschs zwischen Plattformen wird von deren Anbietern unterschiedlich begegnet. Da die anbieterseitig angestrebte IT-Architektur für die nutzerseitige Auswahl der Plattformen von großer Bedeutung ist, sollen im Folgenden die möglichen Szenarien vorgestellt werden.

- Geteilter gemeinsamer Arbeitsspeicher

Laufen die Plattformen als Prozesse auf einem gemeinsamen Server oder einer geteilten IT-Infrastruktur, dann können die Plattformen ihre Daten über eine Interprozesskommunikation austauschen. Hierzu können Teile des Arbeitsspeichers als gemeinsamer Speicherbereich benutzt werden (siehe Abbildung 5).

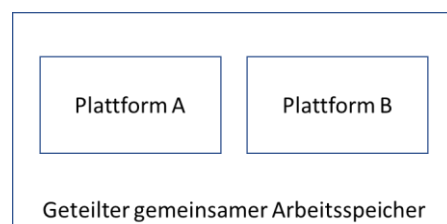


Abbildung 5: Vereinigung zweier Plattformen in denselben Server

Der Zugriff auf den gemeinsamen Speicher wird entweder über das Betriebssystem oder von der geteilten IT-Infrastruktur realisiert. Diese Form des Austausches von Daten ist besonders performant, weil die Anzahl der Indirektionen zwischen den Anwendungen geringgehalten wird und keine HTTP-Kommunikation zwischen den Plattformen nötig ist. Sie erfordert allerdings, dass die Plattformen diesen Datenaustausch direkt in ihrer Applikation implementieren. Veränderungen an der Kommunikation erfordern Veränderungen am Quellcode der Plattform. Deshalb ist der Ansatz dann geeignet, wenn die Plattformen sehr eng verzahnt sind und eventuell vom selben Hersteller zur Verfügung gestellt werden.

Im Fall fehlender Implementierung weiterer Datenaustauschoptionen stellt diese Form des Datenaustauschs die per se zugriffstechnisch restriktivste dar (Datenzugänglichkeitskategorie 4). Aufgrund der Anbindungsbeschränkung hinsichtlich weiterer Plattformen wird diese Form aber selten firmenübergreifend realisiert.

- Datei-Import und -Export

Datei-Import und -Export (siehe Abbildung 6) ist eine Form der Zusammenarbeit, in der die Projektinformationen in den jeweiligen Autorensystemen als Dateien bearbeitet und gespeichert und auch als solche von einer Plattform zur nächsten übergeben werden. Innerhalb der Plattformen werden die Informationen häufig als *Binary Large Objects* (BLOBS) verwaltet, d.h. die Plattform kümmert sich nur um die Dateiverwaltung (mit Rechtemanagement, Versionshistorie, Ersteller etc.), nicht um die Datei-Inhalte. Ein Beispiel stellen die Dateimanagementplattformen *think project!* und *FusionLive* dar.

Die Informationen, die in den Dateien vorhanden sind, erfordern so den Einsatz des Autorensystems, deren automatisierte Zugänglichkeit bleibt also von diesem abhängig. Für die Dateizuordnungen werden häufig umfangreiche Plankodierungen verwendet (d.h. der Dateiname selbst, als einzige unverschlüsselte Informationsquelle, beschreibt die Inhaltszuordnung). Plankodierungen ermöglichen das Auslesen von

Metainformationen aus den Dateinamen. So können Informationen wie Projektphasen und Freigabestatus hinterlegt und von der Plattform ausgewertet werden, um nachgelagerte Prozesse anzustoßen oder die Verteilung der Informationen an einen ausgesuchten Kreis der Projektteilnehmer zu ermöglichen. Manche Plattformen werten auch die Informationen aus der Datei selbst aus, wodurch diese Informationen der Indizierung einer Such- und Filterfunktion zur Verfügung gestellt werden können.

Diese Form der Informationsverwaltung ist wenig zukunftssträftig, aber dennoch nach wie vor weit verbreitet. Die Informationsübertragung zwischen Plattformen muss so manuell geschehen, während selbst nach erfolgtem Export die Datenzugänglichkeitskategorie von den zur Dateierstellung verwendeten Autorensystemen abhängig bleibt.

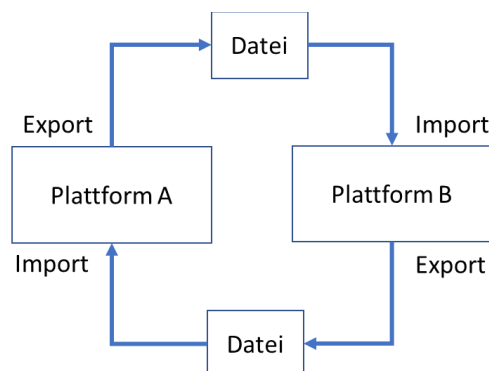


Abbildung 6: Kommunikation zwischen Plattformen mittels Import und Export

- Kommunikation via extern zugänglicher API

Programmierschnittstellen (APIs) können in Softwareprogrammen, speziell Plattformen, sowohl intern als auch extern umgesetzt sein und dienen der automatisierten Abrufbarkeit der gespeicherten Informationen mittels Kommandozeilenbefehlen. In einer internen Umsetzung ist die Verwendung der Software zur Ansteuerung des Backends erforderlich, d.h. die Software selbst stellt eine automatisierte Zugriffsmöglichkeit zur Verfügung. Eine externe API ermöglicht die Ansteuerung des Backends ohne technologische Einschränkung, solange Datenzugriff und -bearbeitung via HTTP-Schnittstelle erfolgen. Die Datenbearbeitung kann somit von der Verwendung des Softwareprogramms abgekoppelt werden. Für einen automatisierten Informationsaustausch zwischen Plattformen muss mindestens eine der beiden Plattformen eine externe API aufweisen, da Informationen sonst nicht direkt übertragbar sind. Dies erlaubt unter bestimmten Umständen, die noch näher erläutert werden, eine software-unabhängige Datenzugänglichkeit.

Für externe APIs hat sich im Webumfeld der Einsatz von REST etabliert. Für die Datenzugänglichkeit ist die Existenz einer externen API aber aus folgenden Gründen keine hinreichende Anforderung:

- Die Abrufbarkeit von Informationen besagt noch nichts darüber, ob auch deren Bedeutung verstanden wird. Bereits eine fehlende Schemabeschreibung bzw. eine Datenverschlüsselung bewirkt bereits eine Datenzugänglichkeitskategorie 3. Dies kann dazu führen, dass nur die Plattformhersteller die Bedeutung der Daten kennen.

- Die Funktionalitäten der externen API können beliebig begrenzt sein, je nachdem, welche Daten der Plattform der Anbieter mit anderen Plattformen austauschen möchte.
- Die für den Informationsaustausch zwischen Plattformen genutzten API-Funktionalitäten müssen nicht zwangsläufig nutzerseitig bekannt sein. Hierfür ist eine verständliche API-Dokumentation erforderlich.

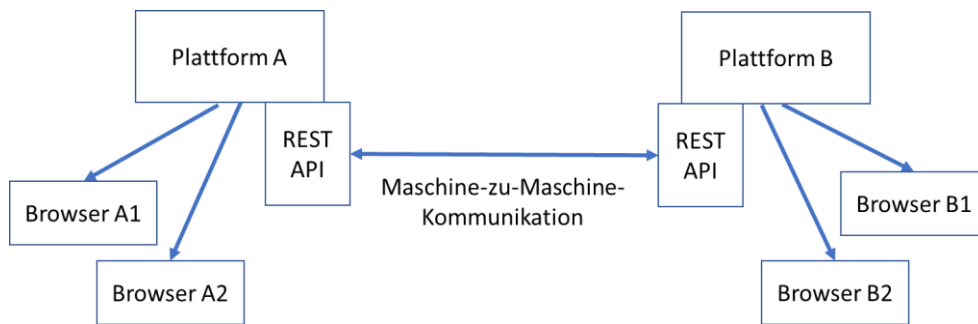


Abbildung 7: Kommunikation zwischen Plattformen mit extern zugänglichen APIs

Beispiele für Plattformen mit externer API sind *BIM 360*, *Dalux* und *think project*!. Meist wird die Kommunikation mittels Maschine-zu-Maschine umgesetzt, d.h. die Plattformanbieter nutzen die REST-Schnittstelle der anderen Plattform(en), um sie direkt über das eigene Backend anzusprechen (siehe Abbildung 7). Alternativ findet die Kommunikation via Frontend statt, d.h. die externe API des Zielsystems wird mittels interner API des Quellsystems adressiert. Für diese Zugriffsart kann vom Plattformanbieter wiederum der Quellcode proprietär gehalten werden (sog. *hooks*), wodurch auch der Datenzugriff wiederum eingeschränkt wird, solange keine generelle API-Dokumentation vorliegt.

Eine weitere Dimension der Automatisierung kann dadurch erreicht werden, dass die via externer API abrufbaren Daten in Datenzugänglichkeitskategorie 1 vorliegen. Die innerhalb einer Plattform generierten Informationen sind so einem automatisierten Zugriff anderer Plattformen zugänglich. Erst diese Entwicklungsstufe kann sinnvollerweise als *Web 3.0* bezeichnet werden.

2.4.2.4 Angestrebte Datenhaltung und Herausforderungen

Das breite Themenfeld des Bauwesens hat zufolge, dass derzeitige Plattformen jeweils hohe datentechnische Überschneidungen aufweisen. Das 3D-Gebäudemodell kommt in vielfältigen Applikationen zum Einsatz und wird an unterschiedlichen Stellen bearbeitet. Um eine Datendurchgängigkeit zu gewährleisten, kommen zwei Herangehensweisen infrage:

1. Einige wenige Plattformen erweitern derart ihren Funktionsumfang, dass keine Datenanbindung von außen mehr erforderlich ist. Dies führt meist zu einer sequentiellen Kette im Bauablauf.
2. Die Plattformanbieter erlauben die automatisierte Anbindung eines software-unabhängigen Datenmanagementsystems.

Zurzeit wird von den Softwarefirmen, im speziellen auch von der Organisation *buildingSMART*, ersterer Weg vorgeschlagen. Wie im ersten Arbeitspaket festgestellt werden konnte, scheitert diese Herangehensweise aber an der viel zu hohen Kom-

plexität der Bauabläufe. Ein weiterer Hindernisgrund besteht in der großen fachlichen Distanz zwischen Softwareherstellern und Anwendern, weshalb die Software-Entwicklung nutzerseitig flexibel gehalten werden muss. Die erste Möglichkeit lässt eine Adaption der Informationsauswertung nur von Seiten des Softwareherstellers zu.

Der zweite Weg aber erfordert ein zentrales Datenmanagement, das vor allem bei gleichzeitiger Änderung derselben Teildatenmenge durch unterschiedliche Akteure relevant wird (siehe Abbildung 8).

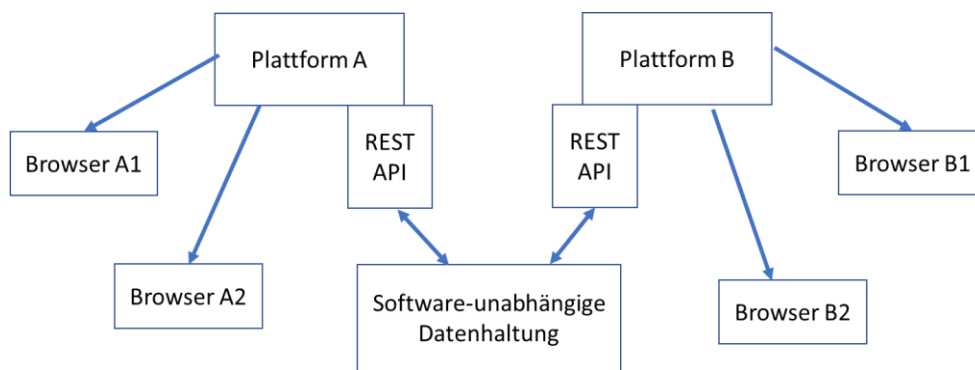


Abbildung 8: Software-unabhängige Datenhaltung

Eine solche software-unabhängige Datenhaltung kann auf unterschiedliche Weise umgesetzt werden. Der erste Gegensatz besteht in der Schaffung einer zentralen, aber hier software-unabhängigen Common Data Environment (CDE), in der alle Informationen gespeichert und verwaltet werden, oder in der Verteilung dieser Informationen auf eine Vielzahl von Webservern (dezentrale Umsetzung). Der zweite Gegensatz besteht im Umgang der Daten auf den Plattformen selbst, die entweder Informationen zu sich synchronisieren (redundantes Verfahren) oder ohne eine Verbindung zur externen Datenhaltung gar keine Bearbeitung der Informationen zulassen (eine existierende Internetverbindung ist zurzeit ohnehin Grundbedingung für die Plattformanwendung).

Im Gegensatz zu einem weitverbreiteten Irrtum ist zur Schaffung einer software-unabhängigen Datenhaltung keine Begriffs-, Struktur- oder Schemaeinigung zwischen Plattformanbietern erforderlich, da deren Informationen separat adressiert werden. Sie hat darüber hinaus den Vorteil, von weiteren Entwicklungen der beteiligten Software-Programme unabhängig zu sein, wodurch die Daten aller Voraussicht nach über den gesamten Gebäudelebenszyklus lesbar bleiben können. Zudem werden die Informationen plattform- und daher anwendungsübergreifend adressierbar, was die Auswertungsmöglichkeiten hinsichtlich *Big Data* und maschinellem Lernen erheblich verbessert.

Neben dem Änderungsmanagement erfordert die software-unabhängige Datenhaltung ein separates Zugriffsmanagement, eine API-Dokumentation und die Möglichkeit serverseitig durchführbarer Abfrageroutinen. Diese Anforderungen werden im folgenden Kapitel beschrieben.

2.4.2.5 Allgemeine Anforderungen an externe APIs der Plattformen

In der Verhandlung mit Plattformanbietern müssen also folgende Forderungen gestellt werden, um eine automatisierte Adressierbarkeit der auf diesen Plattformen generierten Informationen zu gewährleisten:

- | | |
|---------------------------------------|---|
| Eingehende Darstellung der Ergebnisse | <ul style="list-style-type: none"> – Funktionsvollständigkeit. Jede via Webapplikation verfügbare Funktionalität muss auch via □ REST-API abrufbar sein. Dies gilt in besonderem Maße für Funktionalitäten, die Automatisierungen erlauben. Folgende Punkte sollen besonders hervorgehoben werden: <ul style="list-style-type: none"> ○ Externe Editierbarkeit der Daten. Manche Plattformen beschränken die Funktionalität der REST-API auf den Lesezugriff, wodurch eine Synchronisierbarkeit nicht gegeben ist. ○ Nutzerseitig erstellbare und serverseitig durchführbare Abfragemöglichkeiten. Vor allem bei großen Datenmengen ist diese Anforderung essenziell, da die Übertragungszeiten via HTTP-Datenabrufe ineffektiv machen können. – Änderungsmanagementsystem. Ein externer Schreibzugriff wirft die technische Frage nach dem Umgang mit gleichzeitiger Editierung auf. Diese lässt sich auf folgende Arten lösen: <ul style="list-style-type: none"> ○ Nutzerseitige Unterbindung der gleichzeitigen Bearbeitung. Die Anwendbarkeit dieser Option ist von der Zugriffshäufigkeit und damit vom Datentyp abhängig. ○ Bereitstellung von API-Funktionen zur vollständigen Datensynchronisation, um Änderungsmanagementsystem und Plattform zu trennen. (In dieser Option müssen die synchronisierbaren Daten der Plattform dringend vollständig vorliegen, da ansonsten ein plattformübergreifendes Änderungsmanagementsystem nicht sinnvoll umsetzbar ist; siehe □ IFC). ○ Plattformseitige Umsetzung eines Änderungsmanagementsystems, das eine gleichzeitige Dateneditierung erlaubt und mögliche Konflikte extern bearbeitbar macht. – Dokumentation der Dateninhalte. Die abgreifbaren Informationen müssen für den Anwender inhaltlich verständlich sein. Dies gilt sowohl für Codierungen, deren Parserfunktionen geteilt werden müssen, als auch für die Inhalte selbst, deren Anordnung (welche Informationen repräsentieren die jeweiligen Einträge, sog. Datenschema) als auch Bedeutung (an welcher Stelle steht bspw. ein Temperatursensor, dessen Messwerte gelistet sind, auf welche Weise ist er am Objekt befestigt, etc.) erläutert sein müssen. |
|---------------------------------------|---|

Die genannten Anforderungen gelten natürlich auch für die software-unabhängige Datenhaltung selbst, da deren Inhalte von den jeweiligen Nutzern in analoger Weise adressiert werden müssen. Prototypische (Teil-)Umsetzungen sind in Kap. 2.5.5 beschrieben.

Rechtlich gesehen gibt es zwei Möglichkeiten der nutzerseitigen Regelung oben genannter Anforderungen. Mit Softwareherstellern kann einerseits ein Kauf- oder Mietvertrag abgeschlossen werden, der nur unter der Bedingung abgeschlossen wird, dass die genannten Funktionalitäten der Datenzugänglichkeit im Vertrag genannt sind (in der „Beschaffensvereinbarung“). Entsprechend müssen diese auch vor Vertragsabschluss implementiert worden sein, da diese Vertragsform nur für bereits bestehende Produkte zulässig ist.

Es gibt andererseits auch die Möglichkeit, mit dem Softwarehersteller einen sogenannten Werkvertrag abzuschließen. Mit diesem ist direkt die angebotene Funktionalität der Vertragsgegenstand. Besonders günstig ist diese Vertragsform für den Kauf der aufgearbeiteten Daten selbst; so ist nicht mehr die Software, sondern die Datenerstellung und -überlassung der Vertragsgegenstand.

In beiden Fällen (entweder vor Abschluss eines Kauf- oder Mietvertrags oder als Bestandteil eines Werkvertrags) könnte eine entsprechende Vertragsklausel folgendermaßen aussehen:

- Alle Daten, die den Käufer betreffen, sind via Webprotokolle automatisierten Abrufen zugänglich zu machen. Dies bedeutet insbesondere:

- a) die Mitteilung der jeweiligen Webadressen, mit denen die Daten abgerufen werden können;
 - b) die vollständige Beschreibung des Datenschemas (beinhaltend die Unterlassung proprietärer Verschlüsselungen),
 - c) die genaue Definition der Inhalte (Ort der Datenaufnahme, etc.).
- Die Nutzung der vom Käufer erzeugten Informationen zur Verbesserung von IT-Routinen ist dem Verkäufer ausdrücklich erlaubt, solange obengenannte Konditionen erfüllt werden.

Zurzeit wird innerhalb Ed. Züblin AG an der Standardisierung entsprechender Vertragsvorlagen gearbeitet. Für bestehende Softwareprogramme bleibt hingegen nur übrig, die Datenzugänglichkeitskategorie zu untersuchen und den Funktionsumfang der API zu bestimmen. Im besten Fall kann die Softwareanwendung auf den API-Umfang begrenzt werden, um eine vollständige externe Datenanbindung zu gewährleisten.

2.4.3 Konzeptionierung virtueller Komponenten für energetisch aktive Fassadenelemente

Aufgrund der bestehenden Komplexität und Individualität energetisch aktiver Fassadenelementen, ist eine Beschreibung dieser auf Basis von Templates oder vorgeschriebenen Hierarchiestrukturen nicht realistisch. Daher wurde ein Konzept entwickelt, mit dem derartige Produkte frei modelliert werden können. Als Grundlage hierfür dient die Anwendung von Semantic-Web-Technologien.

Der Einsatz dieser Technologien ermöglicht eine dynamische Erweiterung des Datenschemas, ohne die Unterstützung bereits vorhandener Daten zu erschweren, die nach diesem Schema beschrieben wurden. Zudem erlauben Methoden der Technologie, dass Hersteller ihre eigenen proprietären Produktdatenschemata entwickeln und mit der vorgeschlagenen Produktbeschreibung verbinden können. Neben der Flexibilität der Produktbeschreibung selbst, kann zudem die Verteilung und Bereitstellung von Produkten speziell durch Linked-Data-Methoden profitieren. So können die Produkte direkt mit den Bauprojekten verlinkt werden, wodurch redundante Datenhaltung vermieden wird, stets aktuelle und vollständige Produktbeschreibungen verfügbar sind und Hersteller ihre Kunden über neue Produkte oder Probleme vorhandener Produkte einfach informieren können. Da Linked Data ein Teil der Semantic-Web-Technologien ist, können ihre Vorteile besonders dann ausgeschöpft werden, wenn die Daten, die offen verlinkt werden, ebenfalls auf dieser Technologie basieren.

Ausgehend von gesammelten Anforderungen an die Datenstruktur wurde eine Ontologie zur Abbildung der Produkte auf semantischer und geometrischer Ebene entwickelt. Für eine intelligente Anbindung der Produktdaten an die Semantic-Web-Plattform und zur Unterstützung einer individuellen Zugriffs- und Abfrageregulierung, werden die in der Ontologie hinterlegten *virtuellen Komponenten* zur Speicherung in einer Ontologie-basierten verteilten Produktdatenbank hinterlegt.

2.4.3.1 Anforderungen an die Datenstruktur

Zusätzlich zu den zuvor genannten Vorteilen durch den Einsatz von Semantic-Web-Technologien, die teilweise auch als Anforderungen an die Datenstruktur gestellt wurden (freie Modellierung, einfache Erweiterbarkeit), wurden Kompetenzfragen gesammelt, anhand welcher die Funktionalitäten der Ontologie entwickelt und geprüft werden konnte (siehe Liste 1). Die Ontologie wurde zudem in späteren Iterationsschritten nach der UPON Vorgehensweise zur Modellierung von Ontologien an neu aufgekommene Fragen angepasst [24]. Die Kompetenzfragen wurden z. T. gemeinsam mit der *Linked Building Data Community Group* des *World Wide Web Consortiums* (W3C LBDCCG) erarbeitet und im Workshop-Bericht des Linked Data in

Architecture and Construction Workshop 2017 in Dijon (LDAC 2017) veröffentlicht [25]. Die Fragen lassen sich in zwei Kategorien aufteilen: allgemeingültige Produkt-bezogene Fragen und spezifisch für BIST und BIPV Produkte formulierte Fragen. Während des Entwurfs der Ontologie wurden die gelisteten Kompetenzfragen berücksichtigt, sodass sichergestellt werden kann, dass diese auch beantwortet werden können. Aufgrund der abstrakten und allgemeingültigen Natur der entwickelten Ontologie ist aber nicht auszuschließen, dass durch diese auch nicht aufgelistete Fragen beantwortet werden können.

Allgemein

- Welche Eigenschaften hat Produkt A?
- In welchen Produkten ist Element E verbaut?
- Aus welchen Elementen setzt sich Produkt A zusammen?
- Welche Produkte bietet Hersteller H an?
- Welche Eigenschaften hätte Produkt A, wenn es an eine bestimmte Größe angepasst wird?
- Wie teuer ist Produkt A?
- Welches ist das geeignetste Produkt für meine Anforderungen?
- Welche Eigenschaften von Produkt A können individuell angepasst werden?
- Wie unterscheiden sich die Produkte A und B?
- Wie sieht Produkt A aus?

Spezifisch

- Wie ist die elektrische Verschaltung bzw. hydraulische Verbindung innerhalb Modul A realisiert?
- Welche Zellen können in Modul A eingebaut werden?
- Wie kann ich Modul A an mein System anschließen?
- Welche Eigenschaften nimmt das Modul unter gegebenen Vorbedingungen an? (IV-Kurve, machine code functions)

Liste 1: Kompetenzfragen der Produkt-Ontologie

2.4.3.2 Vorstellung Grundkonzept der Produktbeschreibungs-Ontologie

Aus den in 2.4.3.1 vorgestellten Kompetenzfragen und der Anforderung, dass die Ontologie eine freie Modellierung ohne Verwendung von Templates ermöglichen soll, wurde ein Konzept für eine mehrschichtige Ontologie entworfen. Diese besteht jeweils aus einer Ebene für die semantische Bezeichnung der verwendeten Komponenten und ihren Attributen (blau), den inneren Aufbau des Produkts mit seinen Bestandteilen und konkreten Attributwerten (grün) und der zugrundeliegenden Produktgeometrie (schwarz). Die Schichten werden durch Mehrfachklassifizierung einzelner Knoten (gelb) miteinander verbunden. Durch diese Mehrfachklassifizierung wird ermöglicht, dass ein Knoten in verschiedenen Kontexten unterschiedlichen Klassen zugewiesen werden kann ohne eine Kopie des Knotens anfertigen zu müssen; die hier auftretenden Kontexte sind Geometrie, Konstruktion und semantische Bezeichnung. Um weiterhin die Abbildung parametrischer Zusammenhänge zu ermöglichen, soll eine eigens hierfür entworfene Parametrik-Ontologie (pink) Knoten der Schichten der Produkt-Ontologie (grün) und der Geometrie-Ontologie (schwarz) verbinden können.

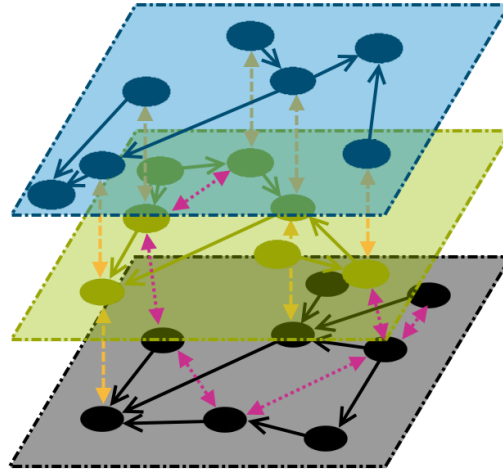


Abbildung 9: Mehrschichtige Produkt-Ontologie

Vor der Implementierung wurden zunächst existente Ontologien betrachtet und auf ihre Eignung zur Wiederverwendung innerhalb des Projektes untersucht. Für die semantische Schicht wurde das *buildingSMART data dictionary* ([□ bSDD](#)) als umfassendste multilinguale existente Domain-Ontologie ausgewählt. Da die zum Zeitpunkt der Konzeption bekannten Ontologien, die zur Abbildung der Produktkonstruktion herangezogen werden könnten (bspw. *ifcOWL*, *PROD*), den formulierten Anforderungen nicht genügen, wurde beschlossen, diese Ontologie selbst zu entwerfen. Als Basis für die Geometrie-Schicht wurden die *GEOM*- und *OntoBREP*-Ontologien betrachtet und aufgrund der Entscheidung der W3C LBDCG, die *GEOM*-Ontologie in ihre Empfehlungen aufzunehmen, und der besseren abbildbaren Geometrie-Vielfalt sowie der vorhandenen nutzbaren Tools zur Visualisierung die *GEOM*-Ontologie ausgewählt. Diese Entscheidung hat zudem die Überlegung bezüglich der Parametrik-Geometrie beeinflusst, da die selbe Ontologie, aus der die *GEOM*-Ontologie stammt, auch Ontologien zur Beschreibung mathematischer Zusammenhänge bereitstellt (*CMO with extensions*). Entsprechend wurde diese Ontologie als Grundlage für die Parametrik-Ontologie bestimmt.

2.4.3.3 Abbildung parametrischer Zusammenhänge

Für die Beschreibung multifunktionaler Fassadenkomponenten ist es relevant, dass diese parametrisch zur Verfügung gestellt werden kann. Innerhalb des Projektes wurde eine parametrische Produktbeschreibung so definiert, dass verschiedene geometrische und nicht-geometrische Eigenschaften voneinander abhängig sind und diese Abhängigkeit mithilfe von Formeln abgebildet werden kann. Eine parametrische Produktbeschreibung muss über die mathematische Definition der Abhängigkeiten verschiedener Eigenschaften hinaus auch die Geometrie des Produktes in Abhängigkeit von solchen Eigenschaften beschreiben können. Dies ist insofern anspruchsvoll, da die parametrischen Zusammenhänge nicht nur innerhalb eines Kontextes, z. B. Geometrie, sondern auch darüber hinaus zwischen verschiedenen Kontexten, z. B. Konstruktion und Geometrie, abzubilden sind.

Diese Voraussetzung bestimmt, dass sowohl Geometrie als auch die semantische Produktkonstruktion im Idealfall im gleichen Datenformat vorliegen, und die Schemata der beiden Kontexte aneinander angelehnt werden sollten. Sollte dies nicht der Fall sein, würden zusätzliche Fehlerquellen durch das Übersetzen der Datenformate bzw. -schemata entstehen. Entsprechend werden auch die parametrischen Zusammenhänge mithilfe einer Ontologie abgebildet, sodass sich möglichst viele Kontexte im gleichen Datenformat befinden.

Die Evaluation der parametrischen Zusammenhänge kann jedoch nicht durch die Ontologie und ihren zugehörigen Reasoner erfolgen, da Reasoner logische und nicht mathematisch-analytische Funktionen beinhalten. Entsprechend muss ein Modul zur Auswertung der Parametrik entwickelt werden, das die Evaluation parametrischer Produkte, ggf. auch innerhalb von Produktsuchen, ermöglicht.

2.4.3.4 Produktkatalog zur Integration energetisch aktiver Fassadenkomponenten (ViKoDB und ViKoLink)

Um einen stets aktuellen Produktdatenkatalog zu entwerfen, der die Datenhoheit der Hersteller respektiert, wurde ein verteilter Produktdatenkatalog (ViKoLink) konzipiert (siehe Abbildung 10). Dieser greift zudem Semantic-Web-Technologien auf, um die Vorteile dieser Technologien, die aus der Produktbeschreibung resultieren, möglichst effizient wiederverwenden zu können.

Herstellern soll es möglich sein, ihre Produkte in eigenen *Virtuellen Komponenten-Datenbanken* (ViKoDB) zu veröffentlichen. Die Implementierung und das Layout der ViKoDBs können hierbei größtenteils von den Anbietern selbst bestimmt werden. Einzige Rahmenbedingungen für die Architektur einer ViKoDB werden durch die Verknüpfung mit dem Produktdatenkatalog gestellt.

Die Verknüpfung der ViKoDBs mit dem Produktdatenkatalog soll über *den Virtuelle-Komponenten-Link-Knoten* (ViKoLink) geschehen. Durch die Verwendung von Linked-Data-Methoden werden die ViKoDBs mit ihren bereitgestellten Produkten in dem ViKoLink gesammelt und können dort durch einen zur Verfügung gestellten Endpunkt von Anwendern – manuell oder durch Software automatisiert – durchsucht werden. Dies bedeutet den Vorteil, dass obwohl die Hersteller eigene Datenbanken nach ihren Sicherheitsanforderungen aufsetzen, die Anwender einen breiten Anbieterkreis einsehen und damit effektive Produktsuchen durchführen können.

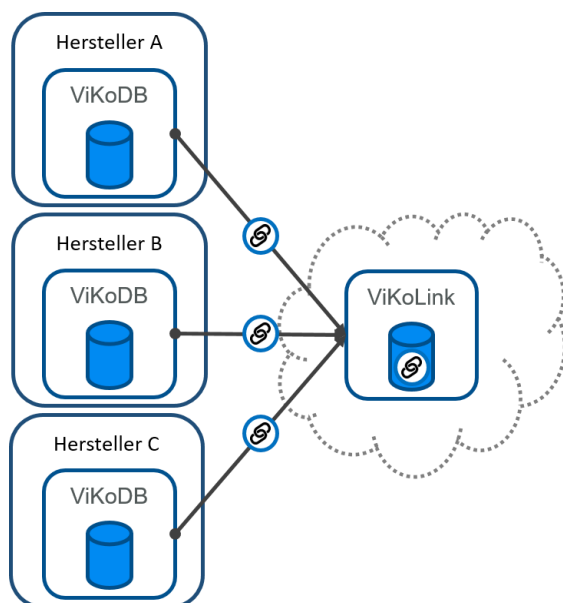


Abbildung 10: Konzept des verteilten Produktdatenkatalogs

2.4.4 Integration von Verarbeitungsprozessen

Für die Zusammenführung und Bereitstellung von Daten im Hinblick auf die speziellen Bedürfnisse verschiedener Fachplaner, muss eine Datenverarbeitungsroutine definiert werden. Diese Routine sollte sowohl Projekt- als auch Gebäude- und Produktdaten umfassen. Im Konzept der Datenverarbeitung wurde zunächst eine Unterscheidung zwischen drei Elementen getroffen: Informationsreduktion, Informati-

onsmanipulation und Datenformattransformation. Weiterhin müssen Datenquellen – z. B. Produktdatenkataloge – in die Datenverarbeitungsprozesse einbezogen werden.

Als Datenverarbeitungsprozess sollen diese Elemente in möglichst beliebiger Reihenfolge ausgeführt werden können, um eine freie Modellierung der Datenverarbeitung zu ermöglichen. Zudem sollen einzelne Elemente nach Möglichkeit allgemeingültig formuliert werden können, sodass diese in unterschiedlichen Anwendungsfällen wiederverwendet werden können. Aus diesem Grund wurde eine strikte Trennung zwischen den drei Elementen gezogen. Für die Reduktion des Informationsgehaltes wurden sog. Rule Based Views (RBV) vorgeschlagen, die Daten unter Berücksichtigung verschiedener Anforderungen zur weiteren Nutzung filtern können. Neben der Reduktion müssen Informationen ggf. auch aus mehreren Quellen zusammengeführt oder auf Basis anderer Informationsquellen durch Algorithmen manipuliert werden. Für diese Funktionalität wurden die sog. Worker konzeptioniert. Abschließend ist eine Transformation der Datenformate notwendig, um eine einfache Verwendung der Informationen in beliebigen Software-Umgebungen zu erlauben. Die Elemente, die diese Funktionalität erfüllen sollen, wurden ursprünglich basierend auf Semantic-Web-Technologien konzeptioniert, woher auch der Name Ontology Based Wrapper (OBW) stammt. Aufgrund der noch geringen Verbreitung von Daten im Format von Ontologien, wurde der ursprüngliche Ansatz jedoch verworfen. Der Name hingegen wurde als Bekenntnis zu dem ursprünglichen Ansatz beibehalten.

2.4.4.1 Datenfilterung (RBV)

Für die Filterung und Aufbereitung kann die vorgeschlagene □ *IDM/MVD-Methode* von buildingSMART als Grundlage verwendet werden, um das Vorgehen der regelbasierten Filterung von Daten daraus abzuleiten. Dies ist in Abbildung 11 veranschaulicht.

Das Ablegen aller Informationen, sowohl projektspezifischer, als auch dauerhaft bereitgestellter Produktdaten, in zentralen Datenhaltungsstrukturen führt dazu, dass aus den Blickwinkeln derjenigen, die auf diese Daten zugreifen, stets deutlich mehr Informationen vorhanden sind, als in den jeweiligen Anwendungsfällen benötigt werden. Um die zu übermittelnden und insbesondere die zu verarbeitenden Datenmengen in einem für die verarbeitenden Softwareanwendungen handhabbaren Umfang zu halten, müssen daher stets aus der Gesamtmenge die für den jeweiligen Betrachter relevanten Informationen extrahiert werden.

Die abgelegten Informationen beschreiben semantische sowie geometrische Aspekte der digitalen Gebäude- und Produktmodelle. Diese Unterscheidung ist im Rahmen der Datenverarbeitung notwendig, da geometrische Informationen – im Gegensatz zu semantischen – mathematisch-analytisch verarbeitet werden müssen. Das Gesamtkonzept der Datenfilterung ist in Abbildung 12 zu sehen. Diese erfolgt nicht durch die Anwendung eines einzigen umfassenden Filters, sondern durch die Verknüpfung mehrerer kontextbezogener, spezieller Filter und Regeln, welche dadurch in verschiedenen Konstellationen wiederverwendet werden können. Zur softwareseitigen Durchführung des Filterungsprozesses wird ein Framework benötigt, das eine Ausführung des Programmcodes erlaubt.

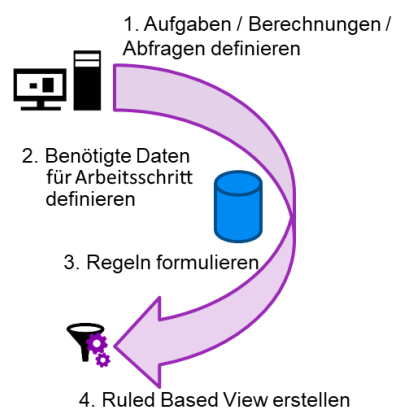


Abbildung 11: Einmalige Definition eines RBVs

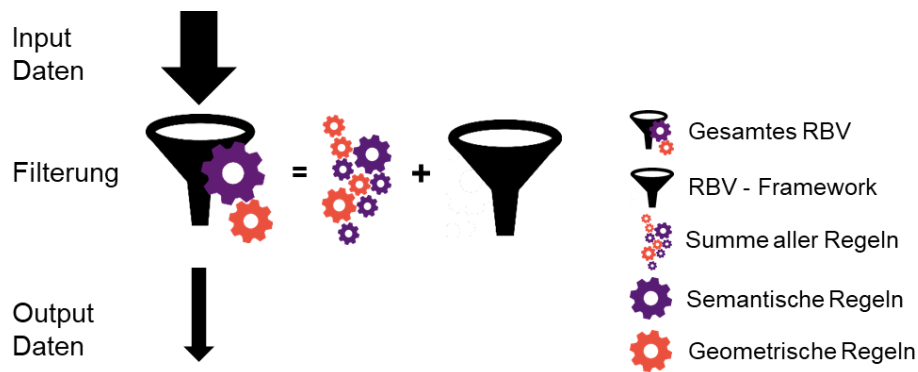


Abbildung 12: Übersicht Datenfilterung

Informationsextraktion durch Regeln

Um aus einer Menge von Informationen gezielt Inhalte zu extrahieren, müssen Vorgehensweisen definiert werden, die der verarbeitenden Maschine vorgeben, was sie zu tun hat. Dies kann auf verschiedene Weisen geschehen. Im einfachsten Fall werden eindeutige Regeln der Form

Wenn [Prämisse] Dann [Konklusion]

definiert. Die allgemeine Regel zur Reduktion von Daten kann somit bspw. wie folgt lauten:

Wenn [Daten entsprechen nicht den Suchkriterien] Dann [lösche diese Daten]

Um komplexere Zusammenhänge abzubilden, können analog zu einfachen Regeln Entscheidungstabellen angelegt werden, in denen für Zusammenstellungen beliebig vieler Bedingungen eine oder mehrere auszuführende Aktionen definiert werden können (s. Tabelle 1). Dies ist dann notwendig, wenn die Regel nicht eine einzige, sondern mehrere Prämissen umfasst.

	Regel 1	Regel 2	Regel 3
Bedingungen			
1: Modulbreite < 1m	J	J	N
2: Modulhöhe > 0,5m	J	J	N
3: Farbe = schwarz	J	N	N
Aktionen			
Modul kaufen	X		
Modul vor-merken		X	
Modul ignorieren			X

Tabelle 1: Entscheidungstabelle (Beispiel); Legende: J: Bedingung erfüllt, N: Bedingung nicht erfüllt, X: Aktion durchführen

In komplexeren Anwendungsfällen mit großen unstrukturiert vorliegenden Datenmengen können anstelle fest vorgeschriebener Regeln auch andere Verfahren, bspw. aus dem Bereich der künstlichen Intelligenz angewandt werden. Da solche

2.4.4.2 Anreicherung von Daten (Worker)

Neben dem Filtern von Daten ist es in einigen Fällen auch erforderlich, aus mehreren Datensätzen eine oder mehrere Ausgabe/n zu erzeugen. Dies kann bspw. dann notwendig werden, wenn zur Durchführung eines Skriptes neben den eigentlich zu verarbeitenden Daten noch weitere Randbedingungen angegeben werden sollen. Ein Beispiel dafür ist das Übersetzen einer Datei. Es wird neben dem zu übersetzenden Inhalt auch eine gewünschte Sprache benötigt. Um solche Anwendungsfälle mit unterschiedlichen Anzahlen von Inputs und Outputs abbilden zu können, werden *Worker* eingesetzt.

Ähnlich den RBV sollen die Worker durch ausführbaren Programmcode anwendungsfallspezifisch implementiert werden. Entsprechend kann dasselbe Framework wie das, welches für die RBV entwickelt werden muss, genutzt werden.

2.4.4.3 Einbindung in Modellierungs- und Simulationsumgebungen (OBW)

Um gefilterte Daten in verschiedenen Anwendungen verwenden zu können, müssen diese ggf. vom bestehenden in ein anderes Format überführt werden. Dies kann mittels sog. *Ontology Based Wrapper* (OBW) für die jeweilige Anforderung erfolgen. Ein solcher Wrapper erhält die Daten in einem bestimmten Inputdateiformat und wandelt sie in das gewünschte Output-Dateiformat um. So können die aufbereiteten Daten in einer bestimmten Software oder anderen angebundenen Schnittstelle verwendet werden. Bspw. erlaubt dies, Gebäudedaten, die in einer IFC-Datei beschrieben wurden, für eine Bestrahlungssimulation in eine Radiance-Datei umzuwandeln. Da konzeptionell ein Großteil der im Rahmen von SolConPro erstellten Wrapper auf in Ontologien vorliegenden Daten basiert, werden diese als *Ontology Based Wrapper* (OBW) bezeichnet, wenngleich darunter auch nicht-ontologiebasierte Wrapper fallen.

Für die Überführung der Daten in ein anderes Datenformat werden in Abhängigkeit des Zielformats zwei Arten von OBWs unterschieden und in Abbildung 13 dargestellt:

1. Zielformate in offenen Standards und nichtbinären Formaten:
Dies ist bei dem OBW für Nutzer B zu sehen, der anhand regelgefilterter Daten eine Datei erzeugt, die in die Software importiert werden kann. Der OBW kann in einem Schritt ausgeführt werden, der die Informationen im Zielformat zum Download für Anwender bereitstellt.
2. Proprietäre, binäre Zielformate von Software-Applikationen mit API:
Dieser OBW ist in zwei Schritte unterteilt. Der erste Schritt erfolgt durch den Server und wandelt die Informationen in ein allgemeingültiges, webbasiertes Format um, welches von der Ziel-Software abgefragt und empfangen werden kann. Diese Software kann über ein Add-In mit der Semantic-Web-Umgebung kommunizieren und Daten empfangen, die direkt in dem Add-In verarbeitet und integriert werden können. Ein Informationsverlust durch zusätzliches Übersetzen in andere Datenformate und Zwischenspeichern wird somit minimiert. Die Software geht nur mit den Daten um, die sie tatsächlich braucht, und fragt diese nur zum jeweiligen Zeitpunkt ab.
Es ist erstrebenswert, dass langfristig die Softwarehersteller ihre Software für die Technik des Semantic Web in Form dieser Methode anpassen, um gänzlich ihre Datenhaltung auszulagern.

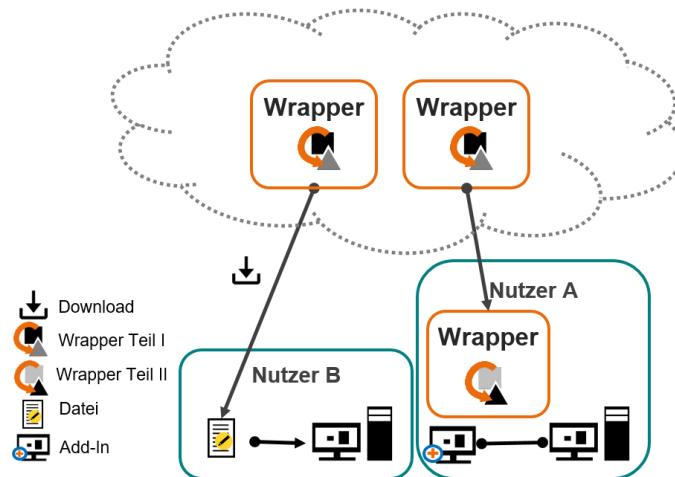


Abbildung 13: Konzeptionierung der OBW

2.4.4.4 Zusammenführung in Plattform/Cloud (SolConPro-Cloud)

Die vorgestellten Komponenten sollen für eine einfache Bedienbarkeit durch Anwender auf einer Plattform, weiterhin als *SolConPro-Cloud* bezeichnet, gebündelt werden. Die SolConPro-Cloud soll es Anwendern ermöglichen, produkt- und projektspezifische Daten für ihre Anwendungsfälle verarbeiten zu lassen, sodass diese anschließend mit möglichst geringem Aufwand weiterverwendet werden können.

Grundlage der Plattform bilden diverse Datenquellen, sowohl für Produktdaten als auch für digitale Gebäudemodelle. Produktdaten können über den verteilten Produktdatenkatalog (ViKoLink) eingebunden werden, während digitale Gebäudemodelle über einen BIM-Modell-Server verknüpft werden sollen. Weitere Datenquellen, wie bspw. meteorologische Daten, sollen ebenfalls über Web-Services angesprochen werden können.

Für die Datenverarbeitungen können sog. *Sets* erstellt werden, die sich aus beliebig vielen RBV, einem OBW und einem Worker zusammensetzen können. Voraussetzung für die Bildung solcher Sets ist, dass die Dateiformate des Outputs einer Komponente identisch mit denen des Inputs der darauffolgenden Komponente sind. Aufgrund des modularen Systems ist es zudem möglich, eine Komponente in verschiedenen Sets wiederzuverwenden.

Das Zusammenspiel der Komponenten auf der Plattform ist in Abbildung 14 aus Anwendersicht dargestellt. Die Produktdaten der ViKoDBs können über ihre Verlinkung mit der ViKoLink für die Verarbeitungsprozesse genutzt werden. Nach der Aufbereitung durch RBV, Worker und OBW können diese anschließend von Planern direkt heruntergeladen und in Applikationen (direkt oder über ein Add-In) verwendet werden. Zudem soll es die Möglichkeit geben, Nutzern die Daten auf einer Website zu visualisieren, bevor die Daten heruntergeladen werden.

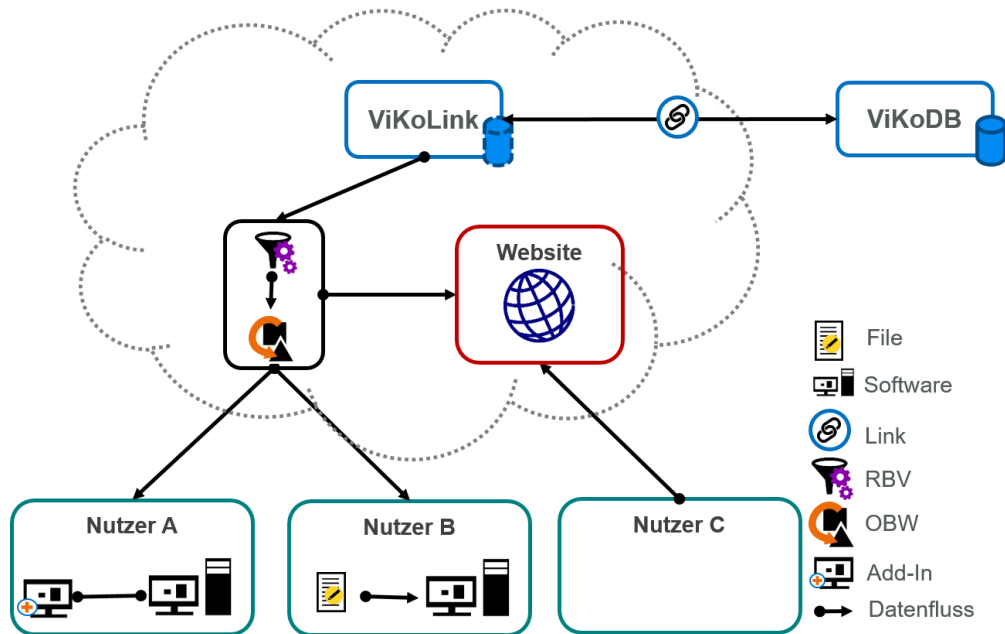


Abbildung 14: Aufbau der SolConPro-Cloud aus Anwendersicht

Die SolConPro-Cloud soll nach Präsentations- und Verarbeitungsschicht getrennt werden. Alle Funktionalitäten bzgl. Datenabfragen und Datenverarbeitungen sollen sowohl über die Präsentationsschicht als auch im Backend über Webservices ansprechbar sein.

Um Datenquellen und Datenverarbeitungsmodule projektspezifisch einbinden zu können, sollen zudem Projekte angelegt werden können. Den Projekten sollen entsprechend externe Datenquellen (Gebäudemodell-Server, meteorologische Daten) und spezifische Datenverarbeitungsmodule und deren Sets zugewiesen werden können. Die projektspezifischen Verarbeitungsmodule müssen hierfür zunächst eingegeben und hochgeladen werden können. Für die Bildung der projektspezifischen Sets können auf projektspezifische und allgemeingültige Datenverarbeitungsmodule zurückgegriffen werden. Den Projekten sollen weiterhin Accounts der Nutzer zugewiesen werden können, wobei ein Account in mehrere Projekte involviert sein kann.

Die hieraus resultierende Gesamtübersicht ist Abbildung 15 zu entnehmen. Die SolConPro-Cloud ist als Online-Service für Nutzer und Hersteller zugänglich. Nutzer können über ihnen zugewiesen Projekte auf die Dienste, externe Datenquellen sowie den Produktdatenkatalog (ViKoLink) zugreifen. Hersteller hingegen können die Produkte in ihren ViKoDBs – ggf. geschützt in einer DMZ – mit der ViKoLink verknüpfen, sodass sie von Nutzern gefunden werden können. Die Datenverarbeitungsmodule können ebenfalls auf externe Datenquellen, z.B. auf das *buildingSMART data dictionary* ([bSDD](#)), zugreifen, um bspw. die Daten in beliebige Sprachen zu übersetzen.

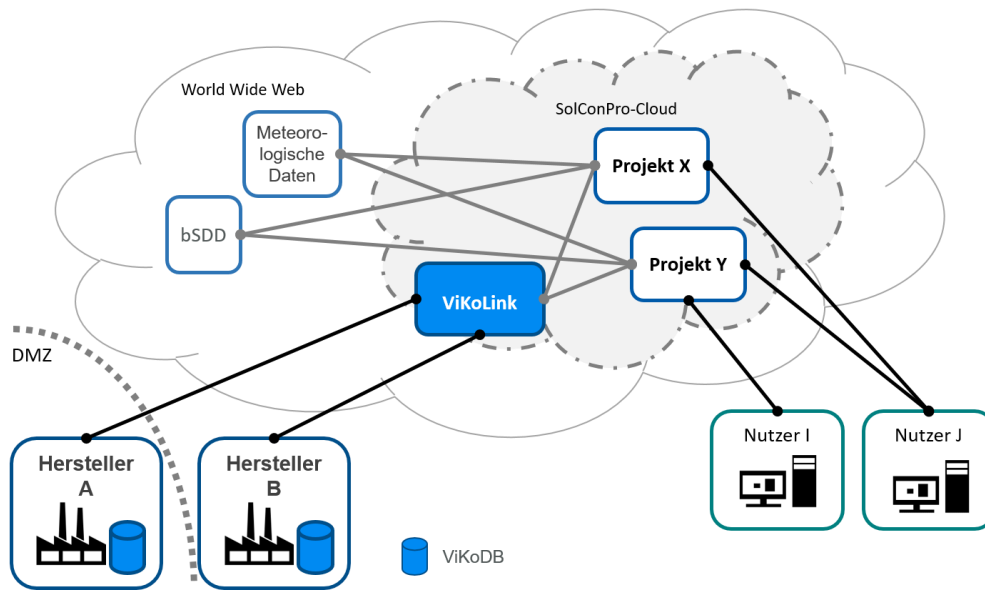


Abbildung 15: Gesamtübersicht SolConPro-Cloud

2.4.5 Modellbasierte EnEV-Berechnung

Für die modellbasierte EnEV-Berechnung, ist weniger die vollständige geometrische Bauproduktbeschreibung nötig, sondern mehr eine Beschreibung der technischen Eigenschaften. Daher wurde das derzeit gängige Format *IFC* herangezogen, um den für die Durchführung von EnEV-Berechnungen notwendigen Detaillierungsgrad in der technischen Beschreibung zu dokumentieren und die in den vorigen Abschnitten theoretisch aufgezeigten Problemstellungen zu veranschaulichen.

2.4.5.1 Aufgabenstellung

Solaraktive Fassaden können einen Beitrag dazu liefern, die steigenden gesetzlichen Anforderungen an die Energieeffizienz zu erfüllen. Dies ist ein wichtiges Argument bei der weiteren Markteinführung dieser Technik. Ziel ist es, die Berechnung des Primärenergiebedarfs nach Energieeinsparverordnung (EnEV) in eine modellbasierte Arbeitsweise (BIM) zu implementieren. Mit der Rechenmethode nach EnEV kann der energetische Nutzen solaraktiver Fassaden am besten dargestellt, argumentiert und in den Bauprozess implementiert werden, da dieser Nachweis im Rahmen der Baugenehmigung erbracht werden muss.

2.4.5.2 Einfluss solaraktiver Fassaden auf die energetische Bewertung

Mit solaraktiven Fassaden kann die Sonnenenergie entweder in elektrische (BIPV – Building-Integrated Photovoltaics) oder thermische (BIST – Building-Integrated Solar Thermal) Energie umgewandelt werden. Die Integration in die Fassade bietet den Vorteil, dass zusätzlich zum Dach weitere Flächen für die solare Energienutzung erschlossen werden können und im Jahresgang eine relativ gleichmäßige solare Einstrahlung erfolgt. Allerdings ist die Jahressumme der solaren Einstrahlung niedriger als bei einem flacheren Montagewinkel und die Umgebungsbebauung kann sich ebenfalls negativ auf die jährliche Einstrahlung auswirken. Das folgende Bild zeigt die mittlere monatliche solare Einstrahlungsintensität in W/m^2 für den Standort Potsdam (Region 4). Dies ist der Standardwetterdatensatz, mit welchem der Energiebedarf für den sog. öffentlich-rechtlichen Nachweis gemäß EnEV zu berechnen

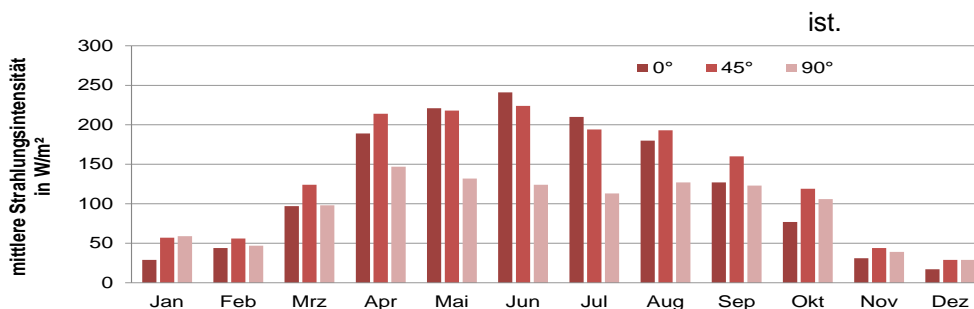


Abbildung 16: Mittlere monatliche Strahlungsintensität für den Wetterdatensatz von Potsdam für Anstellwinkel von 0°, 45° und 90°

Dargestellt sind die mittleren Strahlungsintensitäten bei 0°, 45° und 90°. Das Diagramm zeigt, dass bei 90° Anstellwinkel (entspricht dem Fassadeneinbau) die jährliche Einstrahlsumme zwar niedriger, die monatliche Verteilung jedoch gleichmäßiger ist. Bei der Berechnung des Primärenergiebedarfs gemäß EnEV wird die monatliche Energiebilanz des Gebäudes berücksichtigt (das sog. Monatsbilanzverfahren). Ist in einem Monat das Angebot an solarer Strahlungsenergie größer als der anzurechnende Energiebedarf „verfällt“ das Überangebot der Solarenergie. Durch die gleichmäßigere monatliche Verteilung der Solarenergie beim Einbau in der Fassade kann **prozentual** ein größerer Anteil der Solarenergie genutzt werden.

2.4.5.3 Proprietäre Schnittstellen für die modellbasierte EnEV-Berechnung

Betrachtet werden Programme zur Berechnung des Energiebedarfs gemäß Energieeinsparverordnung (EnEV) mit einer Schnittstelle zum CAD-Programm [AutoCAD Revit](#). Es wurde ein Programm ausgewählt, dass sowohl alle benötigten Berechnungsverfahren nach EnEV unterstützt, als auch eine geeignete proprietäre Schnittstelle zur Anbindung an *Autodesk Revit* hat. Andere Softwareprodukte hatten zum Zeitpunkt der Recherche entweder keine geeignete Schnittstelle oder nicht alle benötigten Berechnungsverfahren unterstützt.

Bislang werden lediglich Gebäude- bzw. Raumeigenschaften durch die Schnittstelle übertragen (Geometrie, Ausrichtung, U-Werte und g-Werte der Umfassungsflächen). Parameter, die für die Abbildung der Anlagentechnik in der DIN V 18599 benötigt werden, können bislang nicht zwischen *Autodesk Revit* und dem EnEV-Berechnungsprogramm ausgetauscht werden.

Bei weiteren Tests der Schnittstelle hat sich herausgestellt, dass insbesondere die thermische Anbindung übereinanderliegender Räume Probleme bereitet. Z. B. passiert es bei abgehängten Decken, dass der Zwischenraum als Außenluft interpretiert wird. Dadurch werden die Transmissionswärmeverluste falsch berechnet. Der vertikale Nachbarraum wird häufig nicht korrekt erkannt. Der Austausch dieser Informationen funktioniert nur mit einigen Einschränkungen bzgl. der Modellierung des Architekturmodells (richtige Definition der Ebenen, korrekte Klassifizierung der Bauteile als raumbegrenzend usw.) welche teilweise im Widerspruch zu den Modellierungsvorgaben der anderen Akteure stehen. Da die korrekte Anbindung benachbarter Räume im Wesentlichen nur für die Energieberechnung relevant zu sein scheint, wird diesem Thema von anderen Gewerken oder in anderen Phasen keine Beachtung geschenkt.

2.4.5.4 Datenaustausch mit offenen Schnittstellen

Die im vorherigen Kapitel beschriebene Variante mit einer proprietären Schnittstelle zum Datenaustausch hat mehrere Nachteile. Zum einen funktioniert sie nur, wenn alle am Projekt beteiligten kompatible Software benutzen, und zum anderen sind die Parameter zum Datenaustausch bislang auf die Übermittlung der Gebäudegeometrie beschränkt. Daher ist ein Datenaustausch mit einer offenen Schnittstelle anzustreben. Das Ablaufdiagramm für die modellbasierte EnEV-Berechnung mit einer offenen Datenschnittstelle, wie z. B. [IFC](#), zeigt das folgende Ablaufdiagramm.

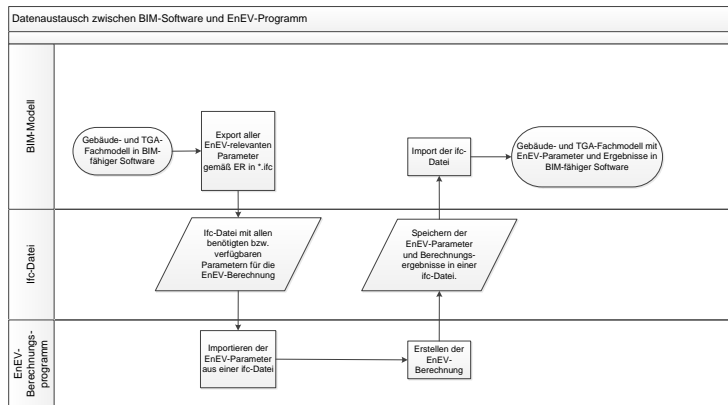


Abbildung 17: Datenaustausch zwischen einem BIM-fähigen CAD-Programm und einem EnEV-Berechnungsprogramm mit einer offenen Datenschnittstelle

Untersucht wurde der mögliche Datenaustausch mit dem IFC-Schema. Die Auswertung der verfügbaren Eigenschaftssammlungen (Property-Sets) hat gezeigt, dass bislang nur wenige TGA-Komponenten als Property-Set vorhanden sind. Die in den Property-Sets gesammelten Parameter passen zudem meist nicht zu den in der EnEV-Berechnung geforderten Parameter. Zum Beispiel ist bei Solarkollektoren (thermisch und elektrisch) im IFC-Schema die Brutto-Fläche als Parameter vorhanden, was für die Abbildung der Geometrie die relevante Fläche ist. Für die EnEV-Berechnung ist jedoch die „energetisch-aktive“ Fläche, die sog. Aperturfläche relevant. Prinzipiell ist es mit den im IFC-Schema vorhandenen Typen von Eigenschaften (Properties) möglich alle notwendigen Größen abzubilden. Es müssten neue Eigenschaftssammlungen und Eigenschaften angelegt werden.

2.4.5.5 Fazit

Ein großes Problem beim Datenaustausch zwischen dem Revit-Modell und den Software-Anwendungen zur Energieberechnung stellt die korrekte Übermittlung der thermischen Umfassungsflächen und der „thermischen“ Verbindungen übereinanderliegender Räume dar. Dies funktioniert nur, wenn das Architekturmodell konkrete Anforderungen an die Modellierung und Modellierungsqualität erfüllt. Da dieses Problem auch für andere Berechnungen der TGA-Planung (z. B. Heizlastberechnung) relevant ist, besteht hier Verbesserungsbedarf. Es werden robustere Schnittstellen benötigt, welche flexibler bzgl. der Qualität des Architekturmodells sind. Parameter welche die Anlagentechnik betreffen, werden bislang nicht ausgetauscht. Auch das [IFC](#)-Schema beinhaltet bislang nicht die nötigen Eigenschaftssammlungen, um die Informationen für einen EnEV-Nachweis abzubilden, da diese Eigenschaftenlisten wiederum die Arbeit der Modellerstellung erschweren würden. Mit der in Kap. 2.4.2.4 beschriebenen vollständigen und software-unabhängigen Form der Datenhaltung wäre dieses Problem automatisch gelöst.

2.5 Arbeitspaket 5 – Prototypische Umsetzung der Ergebnisse für einzelne Produkte

Für die prototypische Umsetzung der vorgestellten Ergebnisse wurden die in AP5 konzeptionierten Komponenten zur Produktbeschreibung (Produkt-Ontologie) und –suche sowie solche zur Datenbereitstellung, -filterung (RBV) und -vorverarbeitung (Worker, OBW) entwickelt. Die Komponenten werden auf einem Server (SolConPro-Cloud) vereint, der mit Projektplattformen (z.B. BIMserver) und über eine Schnittstelle (ViKoLink) mit den Produktdatenbanken (ViKoDBs) verbunden ist. Der Zugriff auf die von der SolConPro-Cloud bereitgestellten Methoden kann manuell über eine Webseite (Frontend der SolConPro-Cloud) oder automatisiert über Webservice-Schnittstellen (Zugriff direkt am Backend) ausgeführt werden.

2.5.1 Produkt-Ontologie

Entsprechend den in 2.4.3.2 vorgestellten Überlegungen für die Produkt-Ontologie wurde eine mehrschichtige Ontologie entwickelt. Diese beinhaltet drei Schichten und eine schichtenübergreifende Ontologie zur Abbildung parametrischer Zusammenhänge. Die Schichten bestehen aus einer Schicht zur Beschreibung semantischer Bezeichnungen (Taxonomie), Darstellung abstrakter Produktkonstruktionen (Produktkonstruktion) und Wiedergabe der optischen Eigenschaften des Produktes (Geometrie). Die Ontologie zur Darstellung abstrakter Produktkonstruktionen stellt innerhalb der mehrschichtigen Ontologie die zentrale Schicht dar. Für die semantische und geometrische Schicht werden bereits existierende Ontologien für die projektspezifischen Anforderungen angepasst und wiederverwendet. Auch die Ontologie zur Abbildung der parametrischen Zusammenhänge besteht aus einer bereits entwickelten und auf unsere Anforderungen angepassten Ontologie.

2.5.1.1 Semantische Bezeichnung

Für die semantische Bezeichnung der verwendeten Komponenten und ihrer Attribute soll gemäß dem entwickelten Konzept eine Domain-Ontologie verwendet werden. Zu dem Zeitpunkt der Implementierung war jedoch keine offene vollumfängliche Domain-Ontologie bekannt, die alle für SolConPro relevanten Anwendungsfälle abdeckt. Aus diesem Grund wird zunächst auf das *buildingSMART data dictionary* ([□ bSDD](#)) zurückgegriffen. Während der Implementierung ist jedoch kenntlich geworden, dass das bSDD zwar prinzipiell auch weitere Funktionalitäten neben der Begriffsdefinition und -übersetzung bereitstellen kann (z.B. Zusammenhänge zwischen Begriffen und/oder Normen), diese aber nicht korrekt gepflegt wurden. Durch die mangelnde Aufsicht über die im bSDD hinterlegten Begriffe und deren Kontexte sind viele Begriffe mehrfach definiert. Besonders problematisch ist hierbei, dass die mehrfach definierten Begriffe in den Definitionen abweichend beschrieben wurden; bspw. sind unterschiedliche Übersetzungen, Einordnungen in Kontexte oder Normen hinterlegt worden. Dieses Problem entstammt der Begriffsdefinition durch die Community und nicht durch ein zentrales Gremium. Zudem erschwert der restriktive Zugriff auf die Daten (nicht über einen SPARQL-Endpunkt, sondern eine [□ REST-API](#), die nur Ausschnitte der dem bSDD zugrundeliegenden Ontologie zurückgibt) eine Anbindung des bSDD über Linked-Data-Methoden. Aus diesem Grund wurden lediglich die GUIDs der Begriffe aus dem bSDD genutzt und als Attribut in die Ontologie der Produktkonstruktion eingefügt.

2.5.1.2 Produktkonstruktion

Zur Abbildung der Produktkonstruktion wurde eine Ontologie entworfen, die in Abbildung 18 dargestellt ist. Basisklasse von der Hälfte der verwendeten Klassen ist die *ClassifiedObject*-Klasse, die eine Relation *bSDD_GUID* besitzt, um die [□ bSDD](#)

ID des Objektes zur Benennung abzulegen. Die weiteren Klassen der Ontologie leiten sich von den folgenden Konzepten ab: Komponenten, Entitäten, Attribute, Attributwerte und Komponentenverbindungen. Diese werden im Folgenden kurz getrennt dargestellt.

Eingehende
Darstellung der
Ergebnisse

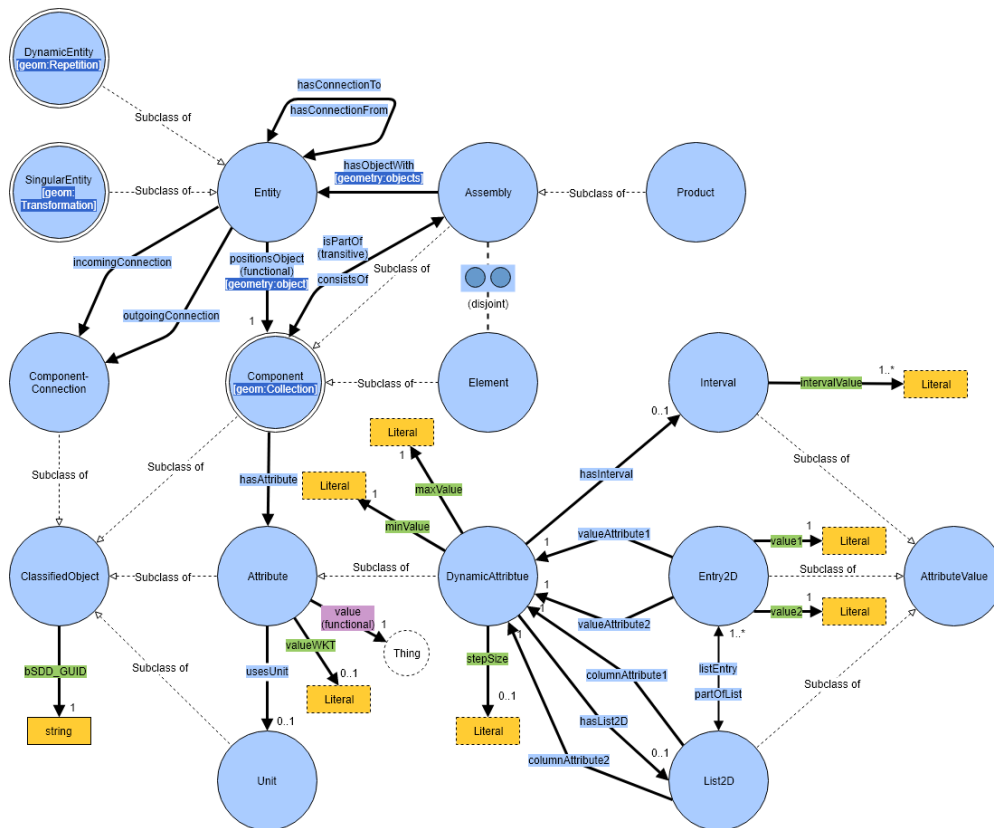


Abbildung 18: Architektur der entwickelten Produkt-Ontologie

Komponenten: Die Basisklasse dieser Kategorie ist die *Component*-Klasse. Diese ist als abstrakte Klasse zu verstehen und soll entsprechend keine Instanzen besitzen. Stattdessen sollen Objekte einer ihrer Subklassen zugeordnet werden: *Element*, *Assembly* oder *Product*.

Hierbei repräsentiert die *Element*-Klasse die kleinstmöglichen Komponenten-Objekte, die sich nicht aus weiteren Objekten zusammensetzen können (bspw. eine PV-Zelle, die nicht detailliert definiert ist). Sollte ein Objekt aus einem oder mehreren Objekten bestehen, kann dieses der *Assembly*-Klasse zugeordnet werden (bspw. ein Zellstrang, der mehrere PV-Zellen beinhaltet). Da ein Objekt nicht zeitgleich eine kleinstmögliche Komponente sein und aus anderen Komponenten bestehen kann, wurden diese beiden Klassen so definiert, dass keine Objekte beiden Klassen zugeordnet sein können. Sie sind somit disjunkt bzw. *disjoint* ((1) und (2)).

$$\forall x (Assembly(x) \rightarrow \neg Element(x)) \quad (1)$$

$$\forall x (Element(x) \rightarrow \neg Assembly(x)) \quad (2)$$

Die Relationen *isPartOf* und *consistsOf* bilden eine solche Zusammensetzung ab, wobei sie als Kettenattribute automatisch eingefügt werden können (3), invers zueinander (4) sowie transitiv ineinander ((5), (6)) sind.

$$\forall x, y, z (hasObjectWith(x, y) \wedge positionsObject(y, z) \rightarrow consistsOf(x, z)) \quad (3)$$

$$\forall x, y (isPartOf(x, y) \leftrightarrow consistsOf(y, x)) \quad (4)$$

$$\forall x, y, z (isPartOf(x, y) \wedge isPartOf(y, z) \rightarrow isPartOf(x, z)) \quad (5)$$

$$\forall x, y, z (consistsOf(x, y) \wedge consistsOf(y, z) \rightarrow consistsOf(x, z)) \quad (6)$$

Als Spezialfall der *Assembly*-Klasse gilt die *Product*-Klasse, die solche Objekte darstellt, die als fertige Produkte angeboten werden können (bswp. ein BIPV-Modul).

Entitäten: Um die Komponenten, die als abstrakte Konzepte beschrieben sind, zu platzieren bzw. zu konkretisieren, werden die *Entity*-Klassen benötigt. Objekte dieser Klassen beziehen sich mit der Relation *positionsObject* auf das Komponenten-Objekt, das konkretisiert werden soll und können über die Relation *hasObjectWith* Komponenten, die sich aus anderen Komponenten zusammensetzen (*Assembly*), zugewiesen werden. Die Relation *positionsObject* ist hierbei als funktional definiert, sodass das Komponenten-Objekt, das platziert werden soll, eindeutig definiert ist (7).

$$\forall x, y, z (positionsObject(x, y) \wedge positionsObject(x, z) \rightarrow y \equiv z) \quad (7)$$

Um eine Unterscheidung zwischen einfachen und wiederholten Platzierungen von Objekten treffen zu können, wurden die beiden Subklassen *DynamicEntity* und *SingularEntity* erstellt. Erstere repräsentiert Objekte, die mehrfach platziert werden sollen und bei denen jede Instanz – abgesehen von ihrer Platzierung – identisch ist. Ein Beispiel hierfür stellt die Platzierung eines Zellstranges in einem BIPV-Modul dar, unter der Annahme, dass es nur diesen einen Zellstrang gibt oder alle Zellstränge individuell modelliert oder platziert werden. Die zweite Klasse wurde für den Fall eingeführt, dass das Objekt nur einmal platziert wird oder, im Falle einer wiederholten Platzierung, die zu positionierenden Objekte sich voneinander unterscheiden (z. B. in der Komponentenverbindung). Eine wiederholte Platzierung kann bspw. bei der Beschreibung eines Zellstranges angewandt werden, da dieser aus mehreren identischen Zellen bestehen kann, die nach einem einheitlichen Schema platziert werden (gleicher Versatz).

Attribute: Alle Komponenten können beliebig viele Attribute über die Relation *hasAttribute* anbinden. Diese wurden generisch modelliert, um eine Modellierungsfreiheit zu gewährleisten. Die Definition des Attributs geschieht über seine □ *bSDD*-ID. Ein Attribut hat immer einen Wert, der über die Relation *value* dem Knoten angehängt werden kann. Diese Relation ist funktional, d.h., wenn sie mehrfach an einem Knoten vorkommt, sind alle über sie verbundenen Knoten und Literale (z. B. String, Double) identisch (ähnlich (7)). Somit wird verhindert, dass ein Attribut mehrere, sich widersprechende Werte besitzen kann. Weiterhin können Attribute mit der Relation *usesUnit* mit Einheiten verbunden sein. Einheiten werden über die *Unit*-Klasse abgebildet und beinhalten derzeit keine Informationen neben ihrer *bSDD*-ID zur Definition der Einheit. Als Beispiel für die Anwendung der *Attribute*-Objekte kann die Modulbreite betrachtet werden. Dieser Knoten beinhaltet die *bSDD*-ID der Bezeichnung „Breite“, den konkreten Wert der Eigenschaft sowie eine Verbindung zu einem Individuum des Typs *Unit*, der entsprechend die *bSDD*-ID der verwendeten Einheit, z. B. Meter, besitzt. Neben Attributen, die einen konkreten Wert besitzen, können auch solche abgebildet werden, die Wertebereiche haben. Hierfür wurde die Klasse *DynamicAttribute* erstellt. Objekte dieser Klasse können neben einem konkreten Wert über die Relationen *maxValue*, *minValue*, *stepSize* und *hasInterval* (in Kombination mit der Klasse *Interval*, die mehrere Werte über die Relation *intervalValue* halten kann) Informationen zu Wertebereichen oder Intervallen beinhalten. Für die Veranschaulichung dieser Objekte kann ebenfalls auf das Beispiel Modulbreite zurückgegriffen werden. Angenommen, dass das beschriebene Modul parametrisch ist und in jeder Breite zwischen 0,5 und 2 m produziert werden kann, würde dem *DynamicAttribute*-Objekt neben den bereits vorhandenen Informationen (Einheit, *bSDD*-ID und Standardwert) folgendes hinzugefügt werden: „0,5“ über die Relation *minValue* und „2“ über die Relation *maxValue*. Im Falle, dass eben nicht jede Breite angeboten werden kann, sondern lediglich bestimmte Werte, können diese über die anderen Relationen definiert werden.

Komplexe Attribute (2D-Listen): Aufgrund der Komplexität der betrachteten Produkte müssen auch komplexe Attribute wie z. B. Machine Code Functions oder 2D-Listen im Ontologie-Entwurf berücksichtigt werden. Diesbezüglich wurde die Abbildung von 2D-Listen als Attribute untersucht. Solche Listen sind als Listen von Wertepaaren zu verstehen, Zusammenhänge sind lediglich innerhalb eines Listeneintrages mit zwei Werten vorhanden und nicht innerhalb der gesamten Liste. Da eine Darstellung solcher ungeordneten Listen mithilfe von Semantic-Web-Technologien zu enormen Datenmengen führt und die Vorteile dieser Technologie nicht oder nur bedingt nutzen kann, wurden zwei unterschiedliche Ansätze betrachtet: die Modellierung der Liste im Semantic Web und die Verwendung von Well Known Text (□ *WKT*).

Zunächst wurden die Klassen *List2D* und *Entry2D* hinzugefügt, um die einzelnen Objekte abzubilden. Dabei entspricht ein Objekt der *List2D*-Klasse einer 2D-Liste, die über die Relation *hasList2D* als Wert einem Objekt der *DynamicAttribute*-Klasse zugeordnet werden kann. Aufgrund dessen, dass eine 2D-Liste zwei Spalten besitzt, die unterschiedliche Attribute repräsentieren, muss zudem eine genaue Zuordnung der Attribute zu den Spalten vorhanden sein. Dies kann über die Relationen *columnAttribute1* und *columnAttribute2* realisiert werden. Die *Entry2D*-Klasse ist ähnlich aufgebaut und kann einzelne Einträge der 2D-Liste darstellen, die über die zueinander inversen Relationen *listEntry* und *partOfList* (ähnlich (4)) einer Liste hinzugefügt werden können. Sie besitzt ähnlich wie die *List2D*-Klasse zwei Relationen, um die eine eindeutige Zuweisung von Attributen zu den Spalten bzw. Werten zulassen (*valueAttribute1*, *valueAttribute2*). Diese sind als Kettenrelationen umgesetzt, so dass sie sich automatisch aus anderen Relationen ergeben ((8), (9)). Neben diesen Relationen gibt es zudem zwei Relationen, um die Werte zu definieren (*value1* und *value2*).

$$\forall x, y, z (partOfList(x, y) \wedge columnAttribute1(y, z) \rightarrow valueAttribute1(x, z)) \quad (8)$$

$$\forall x, y, z (partOfList(x, y) \wedge columnAttribute2(y, z) \rightarrow valueAttribute2(x, z)) \quad (9)$$

Für die Verwendung von WKT wurde zunächst ein Textschema entworfen, mit dem die Liste abgebildet werden kann. Hierfür wurde definiert, dass die Spalten mit Kommas und Zeilen mit Semikolons getrennt werden und die erste Zeile statt Werten die □ *bsDD*-IDs der zugehörigen Attribute beinhaltet. Zur Anbindung des WKT an Attribute wurde die Relation *valueWKT* der *Attribute*-Klasse hinzugefügt.

Abschließend wurde die Struktur so angepasst, dass eine einfache Durchsuchung der Daten nach Werten ermöglicht wird. Hierfür wurde einerseits die Klasse *AttributeValue* erzeugt und als abstrakte Basisklasse der Klassen *Interval*, *List2D* und *Entry2D* definiert. Dies ermöglicht eine Suche nach Werten ohne im Vorhinein wissen zu müssen, von welchem Typ der Wert ist. Weiterhin wurden alle Relationen, die zu Werten weisen (*valueWKT*, *hasInterval*, *hasList2D*), zu Subrelationen der *value* Relation gemacht. Somit sind sie als solche klassifiziert und erhalten auch ihre Eigenschaften. Weiterhin wurden die Relationen der Spaltenattribute in einer abstrakten Basisrelation *columnAttributes* zusammengefasst.

Komplexe Attribute benötigt man beispielsweise für die Beschreibung von IV-Kurven, sog. Strom-Spannungs-Kurven von BIPV-Modulen, die typischerweise als 2D-Listen mit einer Spalte für die Stromstärke (I) und eine für die Spannung (V/U) beschrieben werden.

Komponentenverbindungen: Um weiterhin die Verbindungen zwischen verbauten Komponenten (*Entity*) abbilden zu können, wurde die Klasse *ComponentConnection* eingeführt, die über die Relationen *incomingConnection* und *outgoingConnection* mit Objekten der Klasse *Entity* verbunden werden können. Objekte dieser *ComponentConnection*-Klasse können Verbindungen jeglicher Art sein (z. B. elektrische Verschaltung oder hydraulische Verbindungen) und werden über eine □ *bsDD*-ID definiert. Zusätzlich wurden die zwei Relationen *hasConnectionTo* und *hasConnectionFrom*, die von einem *Entity*-Objekt auf sich selbst oder ein anderes *Entity*-Objekt weisen, vorgesehen. Diese sind invers zueinander definiert (ähnlich (4)) und als Kettenrelationen realisiert (10), sodass beim Durchsuchen der Daten nicht zwangsläufig der Umweg über die *ComponentConnection*-Objekte gemacht werden muss.

$$\forall x, y, z (incomingConnection(x, y) \wedge outgoingConnection(z, y) \rightarrow hasConnectionTo(x, z)) \quad (10)$$

Diese Objekte können u. a. dafür verwendet werden, die elektrischen Verschaltungen oder hydraulischen Verbindungen innerhalb BIPV- bzw. BIST-Modulen zu beschreiben.

2.5.1.3 Geometrie

Für die geometrische Repräsentation der Produkte innerhalb der Produktbeschreibung konnte die ausgewählte *GEOM*-Ontologie unverändert genutzt werden (siehe Kap. 2.4.1.1). Unterschied zur von RDF Ltd. vorgeschlagenen Anwendungsweise ist die Nutzung der A- und T-Box. Von RDF Ltd. wird empfohlen, neue Konzepte nicht als Individuen des Schemas in der A-Box, sondern als Subklassen innerhalb des Schemas in der T-Box abzulegen. Die A-Box soll laut diesem Ansatz lediglich die

fertigen Produkte bzw. Individuen beinhalten. Da diese Herangehensweise Anforderungen unsererseits speziell für das Einbeziehen statischer Produkte (laut RDF Ltd. in A-Box) in die parametrische Modellierung von generischen Produkten (laut RDF Ltd. in T-Box) nicht erfüllen kann, wurde hiervon abgesehen. So werden das Schema der *GEOM*-Ontologie in der T-Box und die individuelle Beschreibung in der A-Box abgelegt. Dies hat zudem Einflüsse auf die adaptierte Parametrik-Ontologie, die auf demselben Grundsatz wie die *GEOM*-Ontologie basiert.

2.5.1.4 Mehrfachklassifizierung

Die Verbindung zwischen Geometrie und Produktkonstruktion erfolgt über eine Mehrfachklassifizierung von Objekten. Diese beinhaltet das Mapping zwischen *Component*-Klasse und der Geometry-Klasse *Collection*. Instanzen der *Collection*-Klasse fassen geometrische Objekte zusammen (z. B. Primitive Beschreibungen, Boolesche Operatoren, Transformationen bzw. Repetitionen, oder andere Kollektionen). Hiermit ist sichergestellt, dass Elemente aus beliebigen Geometrien bestehen können und zeitgleich *Assembly*-Objekte andere Komponenten zusammenfassen können.

Um weiterhin die Platzierung der Elemente relativ zueinander zu erlauben, wurde die Klasse *SingularEntity* mit der *Transformation*-Klasse der Geometry-Ontologie gemappt. Die *Transformation*-Klasse erlaubt sowohl die Rotation, Translation und Verzerrung von beinhalteten Geometrien und entspricht somit der Idee hinter der *Entity*-Klasse, konkrete Objekte einer Komponente innerhalb des Produktes bzw. einer Kollektion (*Assembly*) zu platzieren. Für die wiederholte Platzierung mehrerer Objekte derselben Komponente wird die *DynamicEntity*-Klasse mit der Geometry *Repetition*-Klasse gleichgesetzt. Ein *Repetition*-Objekt erlaubt das Platzieren mehrerer Objekte mit in der Transformationsmatrix festgelegten Rotationen, Translationen oder Verzerrungen.

Neben der Äquivalenz der Klassen wurden auch Relationen gleichgesetzt. So gleicht die *hasObjectWith*-Relation der *objects*-Relation, die in der Geometry-Ontologie erlaubt mehrere Objekte mit einer Collection zu verbinden. Zudem wurde die Relation *positionsObject* mit der Relation *object* gleichgesetzt, die ein einzelnes Objekt mit Verschiebungsoperationen (*Transformation*, *Repetition*) zusammenbringt. Mithilfe dieser Mehrfachklassifizierung ist es möglich, ein Produkt gleichzeitig semantisch und geometrisch zu beschreiben, wie beispielhaft in Abbildung 19 zu sehen ist. Eine direkte Verbindung zwischen den geometrischen und semantischen Eigenschaften (z. B. der Gesamthöhe des Moduls als Summe der Länge einzelner Abschnitte) kann hiermit noch nicht realisiert werden. Daher wurde dieser Punkt bei der Abbildung der Parametrik berücksichtigt.

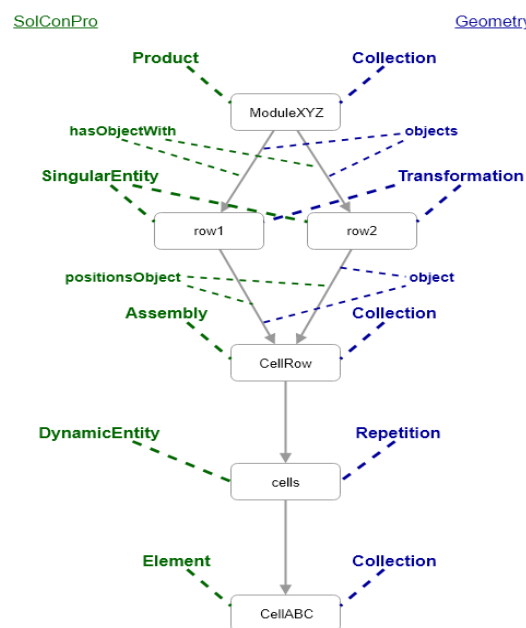


Abbildung 19: Mehrfachklassifizierung für Geometriepäsentationen

2.5.1.5 Parametrik

Als Grundlage für die Ontologie zur Abbildung parametrischer Zusammenhänge wurde ein Teil der „CMO with extensions“, aus der auch die Geometry-Ontologie stammt, verwendet [4]. Diese beinhaltet bereits eine Vielzahl an Klassen zur Repräsentation von Gleichungen und Operatoren (inkl. Variablen und Konstanten). Für eine Applikation dieser Ontologie auf sowohl die Geometry-Ontologie als auch die Produktkonstruktions-Ontologie wurden die Ontologie um die in Abbildung 20 in Türkis dargestellten Klassen bzw. Relationen erweitert.

In der Klassenstruktur wurde die *Constraint*-Klasse als Subklasse der *Equation*-Klasse hinzugefügt. Diese soll Gleichungen repräsentieren, die nicht für die Auswertung der Parametrik, sondern die Zusammenhänge im Produkt relevant sind. Dies wird bspw. benötigt, um die Dimensionierung der einzelnen Schichten eines BIPV-Moduls in Abhängigkeit der Dimensionierung des gesamten Moduls abzubilden. Zudem wurde der *UnaryOperation*-Klasse, die als Basisklasse für alle Operationen dient, die nur einen Operator besitzen (z.B. trigonometrische Funktionen), die neue Unterklasse *floor* definiert, um das Abrunden eines Wertes beschreiben zu können. Anwendungsfall hierfür ist z.B. die Berechnung der PV-Zellen-Anzahl.

Die stärksten Anpassungen wurden bei den Relationen vorgenommen, um die Evaluationslogik stärker in die Ontologie zu integrieren. Die hinzugefügten Relationen werden von *Equation*-, *Operation*- bzw. *Variable*-Objekten verwendet.

Relationen, die der *Equation*-Klasse hinzugefügt wurden ermöglichen es, einfachere Abfragen zu allen der Gleichung zugehörigen Operationen durchführen zu können. So wurde die Relation *equationProperty* als abstrakte Basisrelation für alle enthaltenen Operationen sowie die Rechte-Hand- und Linke-Hand-Seite der Gleichung eingeführt. Für eine weitere Vereinfachung der Abfrage der Gleichungs-Handseiten wurde die Basisrelation *equationHS* definiert, von der die gegebenen Relationen *LHS* und *RHS* erben. In der Gleichung enthaltene Operationen werden über die Ketten-Relation *equationOperation* abgebildet (11).

$$\forall x, y, z (equationHS(x, y) \wedge operationsOperation(y, z) \rightarrow equationOperations(x, z)) \quad (11)$$

Der *Operation*-Klasse wurde zunächst die Basisrelation *operationsOperation*, die alle mit dem Objekt verbundenen Operatoren durch Vererbung zusammenfasst (*firstOperator*, *secondOperator*) und transitiv ist (ähnlich (5)).

Die meisten der neuen Relationen werden von *Variable*-Objekten verwendet. So können Variable mit *variableHigherDOFThan* in Relation zueinander gesetzt werden, um zu beschreiben, dass eine Variable zur Lösung einer Gleichung flexibler als gesucht gesetzt werden kann als eine andere. Um die Anzahl der Verbindungen (z.B. eine Verbindung zwischen X und Z, wenn zuvor definiert ist, dass X flexibler als Y und Y flexibler als Z ist) gering zu halten, wurde diese Relation als transitiv charakterisiert. Weitere Objekt-Relationen sind *VariableParameter* und *variab-*

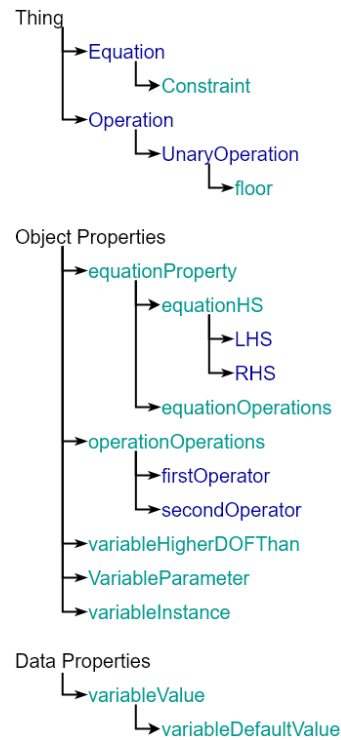


Abbildung 20: Für Parametrik hinzugefügte Objekte

leInstance, die, aufbauend auf die ursprüngliche Verbindung der Variablen an Werte der „CMO with extensions“, ein *Variable*-Objekt mit einem anderen Objekt (*variableInstance*) und seiner Relation, dessen Wert von der Variable abgedeckt werden soll (*VariableParameter*) verbindet. Der Wert der Variablen kann über die Relationen *variableValue* und *variableDefaultValue* hinzugefügt werden. Unterschied zwischen den beiden durch Vererbung zusammenhängenden Relationen ist, dass *variableDefaultValue* einen Standardwert wiedergibt (z. B. der konkrete Wert eines *DynamicAttributes*), während *variableValue* einen fest gegebenen Wert darstellt (z. B. durch Benutzereingaben oder nicht dynamische Attribute definierte Werte). Bei Iterationen der Berechnung, bspw. um ein Optimierungsproblem zu lösen, können nur solche Werte verändert werden, die über die Relation *variableDefaultValue* mit der Variablen verbunden sind. Wie sich der Variablenwert ergibt ist in (12) zu erkennen. Da sowohl *variableValue* als auch *variableDefaultValue* Datentyp Relationen sind, ist es nicht möglich, die abgebildete Formel als Ketten-Axiom der Relation zu hinterlegen. Stattdessen muss die Zuordnung über Abfragen oder Algorithmen im Quellcode der Applikation geschehen.

$$\begin{aligned} & \forall \text{ variable, instance, parameter, value} \\ & (\text{variableInstance}(\text{variable, instance}) \wedge \text{parameter}(\text{instance, value}) \\ & \wedge \text{VariableParameter}(\text{variable, parameter}) \\ & \rightarrow \text{variableValue}(\text{variable, value})) \end{aligned} \quad (12)$$

2.5.1.6 Ontologie-Kerndaten

Die vorgestellte Ontologie besitzt die DL Expressivität SRIQ(D) und entspricht somit einer OWL-2-Ontologie. Die Überprüfung mit dem Reasoner ELK ergab, dass die Ontologie konsistent und kohärent ist. Weitere Kerndaten der Ontologie sind *Tabelle 2* zu entnehmen. Die inferierten Triple beziehen sich auf die Domains und Ranges von Attributen, die von anderen Attributen erben und somit dieselben Anforderungen an Domains und Ranges erhalten.

Typ	Lokal	Importiert	Inferiert	Gesamt
Anzahl Triples	290	2129	9	2428
rdf:Property	1	0	0	1
owl:Class	19	116	0	135
owl:DatatypeProperty	12	92	0	104
owl:FunctionalProperty	7	0	0	7
owl:ObjectProperty	27	47	0	74
owl:Ontology	1	3	0	4
owl:TransitiveProperty	3	0	0	3

Tabelle 2: Kerndaten der entwickelten Produkt-Ontologie

2.5.2 Produktdatenbank (ViKoDB)

Die aus den in 2.4.3.4 beschriebenen Anforderungen resultierende Architektur ist in Abbildung 21 zu erkennen. Es ist jedoch wichtig anzumerken, dass es sich hierbei vielmehr um eine Empfehlung als eine feste Definition zur Architektur einer Produktdatenbank handelt. Einzige für die Anbindung an den Produktdatenkatalog notwendige Anforderung ist das Vorhandensein eines SPARQL-Endpunktes, der von außerhalb des eigenen Netzwerkes angesprochen werden kann. In dem Vorschlag wurde ein Webserver mit zugrundeliegenden Tripeldatenbanken (TDB) in Form eines Fuseki-Servers aufgesetzt. Diesem wurden zudem drei Services – eines für jedes der für die Anwendungsfälle umgesetzten Produkte – zugeteilt. Die Services sind identisch aufgebaut: Es gibt einen SPARQL-Endpunkt, der den *default*-Graphen des TDB abfragt. Dieser *default*-Graph setzt sich wiederum aus beliebig

Eingehende
Darstellung der
Ergebnisse

vielen eigenen Graphen, die durch einen Namen in der TDB definiert sind (*Named Graphs*). Diese Architektur erlaubt jedoch nicht, einen *Reasoner*, der auf Basis der entwickelten Produkt-Ontologie neues Wissen generieren kann (z.B. durch Auswertung von Ketten-Relationen oder transitiven Relationen), in eine im *default*-Graph ver-einte TDB zu integrieren. Daher muss die Wissensgenerierung über die Abfragen geschehen und jedem Datensatz die entwickelte Produkt-Ontologie hinterlegt werden. Für weitere Entwicklungen wäre es denkbar, die Wissensgenerierung auf den Produktdatenkatalog zu verlegen, um die redundante Datenhaltung des Schemas (T-Box) zu vermeiden.

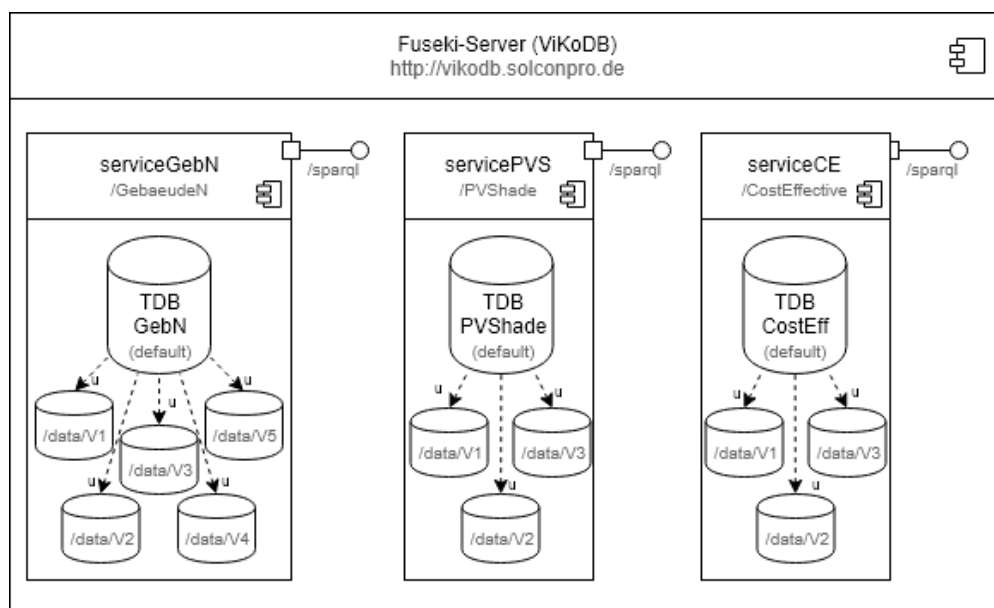


Abbildung 21: Kompositionsstrukturdiagramm der ViKoDB

2.5.3 Produktdatenkatalog (ViKoLink)

Die Virtuelle-Komponenten-Linkdatenbank (ViKoLink) dient als Produktdatenkatalog und verwendet Linked-Data-Methoden, um redundante Datenhaltung der Produkte zu vermeiden. So kann jede ViKoDB, die über einen SPARQL-Endpunkt verfügt, mit der ViKoLink verbunden werden. Als Grundlage werden von der ViKoLink Produkte im Schema der vorgestellten Produkt-Ontologie, die zudem gemeinsam mit der Produkt-Ontologie selbst abgelegt sind, erwartet.

Abbildung 22 zeigt die Architektur der ViKo-Link. Über einen Server werden Services für die folgenden Anwendungsfälle zur Verfügung gestellt: (1) Registrierung neuer und vorhandener ViKoDBs, (2) Produkt-abfragen und (3) Produktsuche. Die ein-kommenden Service-Anfragen, die jeweils eine **JSON**-Datei nach vorgegebenem

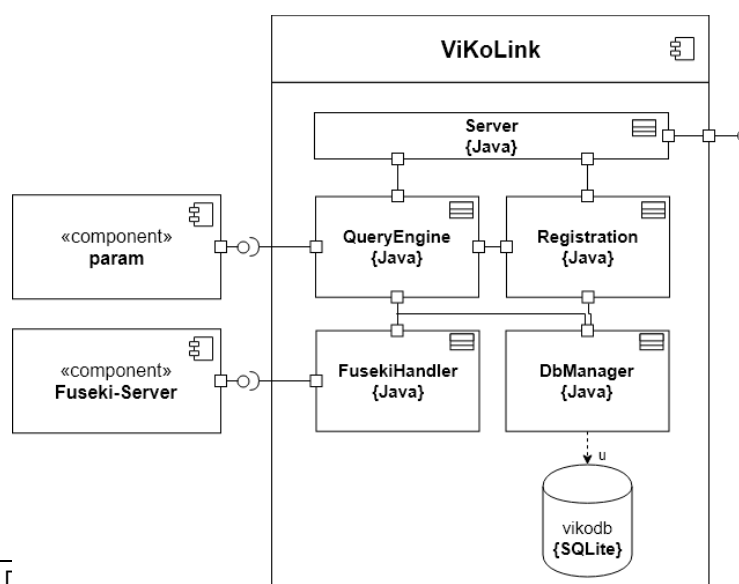


Abbildung 22: Architektur der ViKoLink

Schema beinhalten, werden anschließend vom Server an die entsprechenden Komponenten (*Registration* für Registrierung, *QueryEngine* für Produktsuchen und Produktabfragen) weitergeleitet.

Die *Registration* (1) kann über den *DbManager* neue ViKoDBs der zugrundeliegenden *vikodb*-Datenbank mit allen registrierten SPARQL-Endpunkten hinzufügen sowie vorhandene Einträge aktualisieren oder löschen. Die Einträge dieser Datenbank beinhalten neben Informationen zu den Anbietern der ViKoDBs und den zugehörigen SPARQL-Endpunkten auch solche über die auf den Endpunkten angebotenen Produktkategorien, anhand derer bei Produktsuchen eine Vorfilterung der anzusprechenden Endpunkte vorgenommen werden kann.

Im Falle einer Produktsuche (3) werden in der *QueryEngine* zunächst die in der JSON-Datei enthaltenen Suchkriterien ausgelesen und über den *DbManager* alle SPARQL-Endpunkte herausgesucht, in denen die angefragte Produktkategorie vorgehalten wird. Anschließend wird eine verteilte SPARQL-Abfrage erzeugt, die über den *FusekiHandler* an eine lokale Instanz eines Fuseki-Servers gesendet wird. Die Wahl, dass an dieser Stelle ebenfalls ein Fuseki-Server verwendet wird, ist aus dem Grund gefallen, dass er verteilte Abfragen mit dem Schlagwort `SERVICE<URL>` unterstützt. Die Abfrage wird so formuliert, dass sowohl konkrete Attributwerte als auch Wertebereiche von Attributen auf Eignung bezüglich der Suchkriterien geprüft werden. Nach Erhalten der Abfrage-Ergebnisse werden die zurückgegebenen Produkte daraufhin untersucht, ob sie parametrische Zusammenhänge beinhalten. Sollte dies der Fall sein, werden zunächst alle Variablen, die mit abgefragten Attributen zusammenhängen, mit allen möglichen Wertekombinationen aus der Suchanfrage ermittelt. Diese Informationen werden dann der *para*-Komponente über ihren Webservice zur Evaluation weitergeleitet. Wenn die Evaluation abgeschlossen ist, können die resultierenden Produkte über ein temporäres Daten-Set des Fuseki-Servers ausgelesen und den Gesamtergebnissen hinzugefügt werden. Diese werden anschließend in eine in der JSON-Datei definierten TTL-Datei serialisiert.

Bei Produktanfragen werden gezielt die Daten eines in der übertragenen JSON-Datei über den Graph-Namen und SPARQL-Endpunkt definierten Produktes abgefragt. Auf Basis dieser Daten wird in der *QueryEngine* eine simple verteilte SPARQL-Abfrage generiert und über den *FusekiHandler* an den Fuseki-Server weitergeleitet. Sollte es sich bei dem angefragten Produkt um ein parametrisches Produkt mit gesetzten Parametern handeln, wird das Produkt und seine gesetzten Parameter zur Evaluation der *param*-Komponente weitergeleitet und das berechnete Produkt zurückgegeben. Ansonsten wird das aus der ViKoDB erhaltene Produkt in die in der JSON-Datei definierten TTL-Datei geschrieben.

2.5.3.1 Parametrik-Evaluation (param)

Um die Parametrik eines Produktes evaluieren zu können, wurde ein Modul zur Parametrik-Evaluation entwickelt, das über die Cloud als Webservice zur Verfügung steht. Die parametrischen Produktdaten eines übergebenen Produktes werden hierfür von der entsprechenden ViKoDB abgerufen. Die Abfrage findet jedoch nicht direkt auf die ViKoDB statt, sondern erfolgt über die ViKoLink, da diese die Zugriffe auf die Datenbanken einheitlich regelt. Eine Übersicht über die *param*-Komponente, die für die Auswertung der parametrischen Zusammenhänge entwickelt wurde, ist Abbildung 23 zu entnehmen.

Eingehende
Darstellung der
Ergebnisse

Das Modul ist als Webservice auf der SolConPro-Cloud angesiedelt und kann dort in drei verschiedenen Kontexten angesprochen werden:

(1) Für die Kalkulation eines parametrischen Produktes ohne Benutzervorgaben unter Berücksichtigung der Standard-Parameterwerte; (2) zur Evaluation von durch eine Benutzervorgabe gegebenen Parametern für ein spezielles Produkt und (3) als Teil der Produktsuche, um parametrische Produkte in diese mit einbeziehen zu können. Um den Webservice über einen HTTP-Request aufzurufen, müssen die gesetzten Parameter, die Identifikation des Zielproduktes (über Graph-Name und SPARQL-Endpoint) sowie die Zieldatei in einer JSON-Datei definiert und entsprechend übergeben werden. Neben eben dieser Schnittstelle zur Bereitstellung des Webservices sind zwei weitere Schnittstellen innerhalb des Moduls vorhanden: eine Schnittstelle zur Ausführung verteilter SPARQL-Abfragen von der ViKoLink verwendet wird und eine Schnittstelle zu einem externen [SageMath](#)-Webservice (z.B. den *SageMathCell-Server*), an welchen Gleichungen zu deren Lösung übergeben werden.

Das Modul selbst ist in fünf Klassen aufgeteilt (vgl. Abbildung 23). Die Klasse *ParametricWebService* dient der Bereitstellung des Webservice, seiner Kontexte und der Weiterleitung der erhaltenen Informationen an die *Calculation*-Klasse. Diese beinhaltet die Logik zur Verarbeitung der Daten, um aus der im Graph hinterlegten Parametrik lösbare Gleichungen zu erhalten, Benutzervorgaben zu berücksichtigen und Optimierungsprobleme (z.B. Runden) in den Ablauf zu integrieren. Hierfür werden weitere Hilfsklassen zur Anbindung der notwendigen Schnittstellen verwendet: Die *FusekiHandler*-Klasse formuliert SPARQL-Abfragen, führt diese auf dem Fuseki-Server aus und übersetzt die Ergebnisse in vordefinierte Objektstrukturen. Für die Übermittlung der aus der parametrischen Produktbeschreibung ausgelesenen Gleichungen an einen *SageMath*-Webservice werden zwei Klassen umgesetzt: Zunächst werden die Objektstrukturen in der *SageGen*-Klasse in ein String-Format mit einem derartigen Schema geparsed, das es von dem *SageMath*-Webservice interpretiert werden kann. Der Aufruf des Service selbst und die anschließende Übersetzung der Ergebnisse zurück in die definierten Objektstrukturen finden in der *SageHandler*-Klasse statt.

Das in Abbildung 24 dargestellte Sequenzdiagramm zeigt einen detaillierten Einblick in den Ablauf der Evaluation. Im Folgenden wird zunächst auf diesen und im Anschluss auf den zugrundeliegenden Algorithmus eingegangen.

Ablauf

Zu Beginn wird der Webservice in einem der möglichen Kontexte aufgerufen und die bei der Anfrage im HTTP-Request übermittelte [JSON](#)-Datei empfangen. Diese JSON-Datei wird in der *ParametricWebService*-Klasse zunächst in Objekte übersetzt, die anschließend mit weiteren Informationen zu dem aufgerufenen Kontext an eine neue Instanz der *Calculation*-Klasse übergeben werden. Dort werden die vom Benutzer gesetzten Werte registriert und über die *FusekiHandler*-Klasse die gesamte Produkt-Parametrik des angefragten Produkt-Graphen ausgelesen.

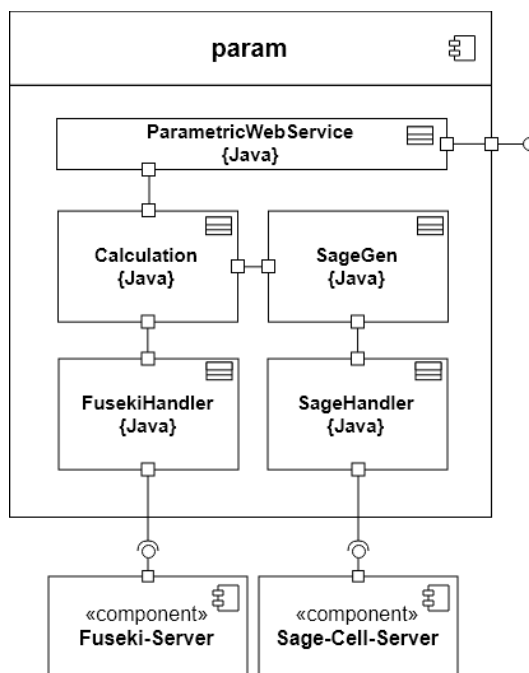



Abbildung 23: Architektur der param-Komponente

Zunächst wird die ausgelesene Parametrik des Graphen in der *Calculation*-Instanz auf hinterlegte Integer-Constraints, wie beispielsweise das Runden bestimmter Werte (Optimierungsproblem), untersucht und diese falls vorhanden für ein späteres Durchführen weiterer Iterationsschritte vorgemerkt.

Anschließend werden die vom Benutzer vergebenen Werte innerhalb der *FusekiHandler*-Klasse mit den für die jeweiligen Attribute im Graphen hinterlegten Wertebereiche abgeglichen und auf Gültigkeit geprüft. Liegt ein Wert über bzw. unter dem für diesen Wert definierten Grenzwert, wird der nächste gültige Wert angenommen und die codeseitige Änderung des vom Benutzer vergebenen Wertes im Log dokumentiert. Handelt es sich bei der Anfrage um eine allgemeine Produktsuche unter Einbeziehung parametrischer Produkte (Kontext 3), entfällt dieser Schritt. Liegen die bei der Suche vom Benutzer vorgegebenen Werte außerhalb des möglichen Wertebereichs eines Attributes des Produkts, so wird das Produkt nicht weiter in der Suche berücksichtigt, da es in gewünschter Konfiguration nicht verfügbar ist. Im nächsten Schritt werden die Gleichungen und verfügbaren Attribut-Werte in der *Calculation*-Instanz so verarbeitet, dass sie lösbar Gleichungen mit nur maximal einer Lösung bilden. Hierzu werden die Gleichungsvariablen mit den Werten der Benutzervorgabe oder in Abhängigkeit des Kontextes mit ihren Standardwerten gesetzt, sodass gültige Gleichungen entstehen (siehe Algorithmus).

Nach der Aufbereitung der Benutzervorgaben und Produkt-Parametrik werden diese als zusammengefasste Objekte einer neuen Instanz der *SageGen*-Klasse übergeben. In dieser Instanz wird das Objektformat in ein vordefiniertes String-Schema übersetzt, das von dem eingesetzten  *SageMath*-Mathematiksystem interpretiert werden kann. Die übersetzten Objekte werden daraufhin der *SageHandler*-Klasse übergeben, die einen *SageMath*-Webservice aufruft und die zur Abfrage erhaltenen Ergebnisse aufbereitet. Die Ergebnisse werden anschließend von der *Calculation*-Instanz über die *SageGen*-Instanz ausgelesen.

Wurden zu Beginn der Evaluation in der Parametrik hinterlegte Bedingungen (z. B. Integer-Constraints) vermerkt, so werden die ermittelten Ergebnisse noch einmal auf die Einhaltung derselben geprüft. Variablen, die einer Bedingung widersprechen, werden dann so korrigiert, dass sie diese einhalten. Es findet ein weiterer Iterationsschritt statt, in dem die Ermittlung der unbekannten Variablenwerte erneut durchgeführt wird. Der aufgrund der Bedingung korrigierte Wert wird für diesen Iterationsschritt als Benutzervorgabe definiert, sodass dieser bei der Iteration als gesetzt angenommen wird und nicht verändert werden kann. Die Produkt-Parametrik wird somit erneut ausgewertet.

Sobald die iterative Prüfung der Evaluationsergebnisse keine Widersprüche zu vorhandenen Bedingungen mehr ergibt, werden die final ermittelten Variablenwerte über die *FusekiHandler*-Klasse den Attributen des ursprünglichen Produktgraph in einer lokalen Kopie übergeben und alle geometrischen Zusammenhänge, welche über *Constraints* im Graphen hinterlegt wurden (vgl. Kap. 2.5.1), ausgewertet. Somit sind alle geometrischen und semantischen Eigenschaften des Produktes für die gesuchte Konfiguration des Produktes ermittelt worden. Neben den Attribut- und Geometriewerten wird in der lokalen Kopie des Graphen auch der Namespace so erweitert, dass aus diesem die Konfiguration der Parametrik-Evaluation eindeutig erkennbar ist.

Der aktualisierte Graph wird abschließend gemäß den Anforderungen in der Abfrage serialisiert (als TTL-Datei in Kontext 1 und 2 oder als Eintrag eines temporären Datensatzes des Fuseki-Servers in Kontext 3). Für die Antwort an die Abfrage wird zuzüglich zum HTTP-Statuscode das Log der Parametrik-Evaluation an die *ParametricWebService*-Klasse übergeben, die diese Informationen dann als HTTP-Response an den Sender zurückgibt.

Evaluation parametrischer Produkte

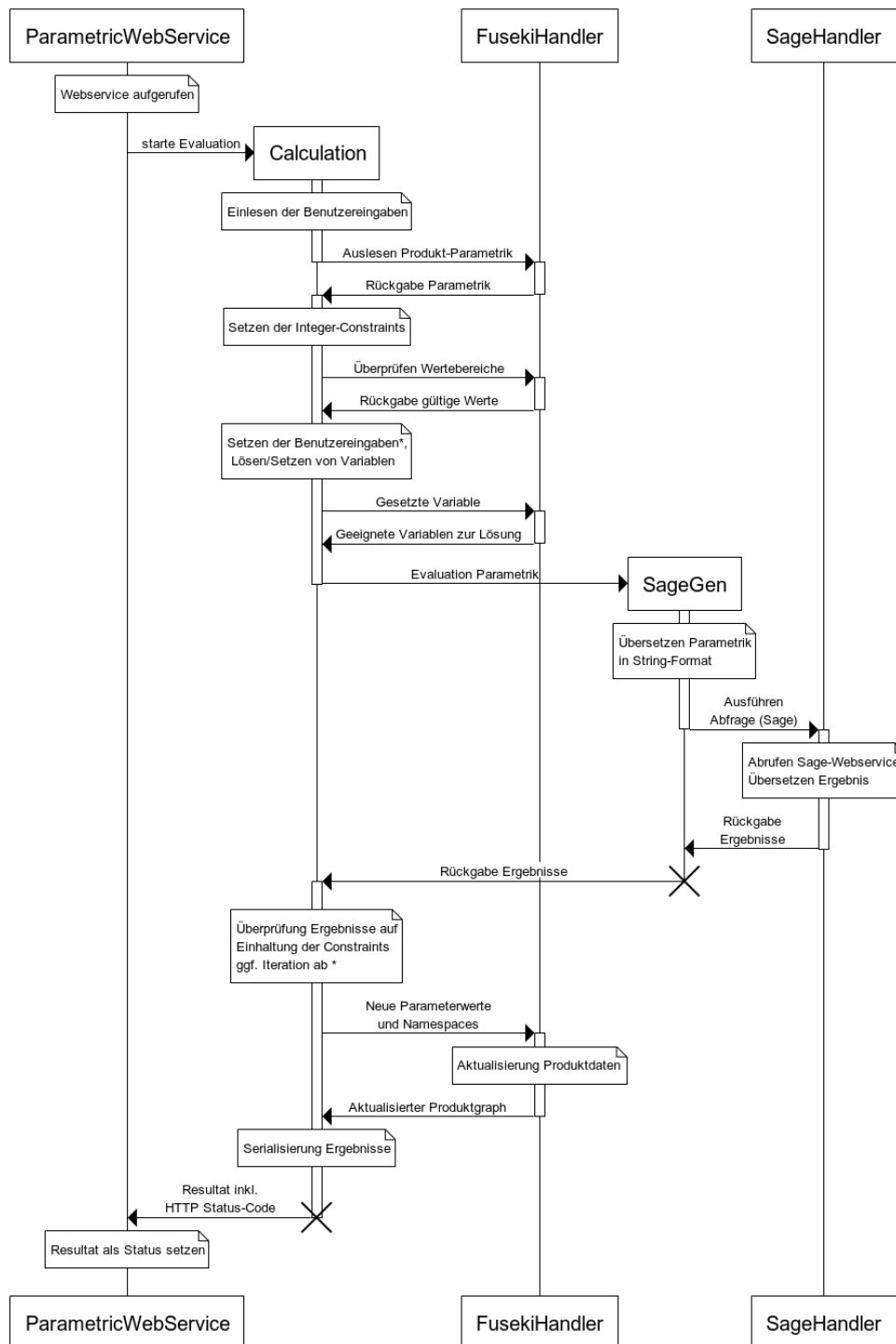


Abbildung 24: Ablauf der Parametrik-Evaluation

Algorithmus zur Erstellung lösbarer Gleichungen

Die im Graphen hinterlegten Gleichungen (*Equation*) zur mathematischen Beschreibung der Produkt-Parametrik setzen sich aus einer Linke- und Rechtheandseite zusammen und können Variablen (*Variable*), Konstanten (*Constant*) und Operationen (*Operation*) beinhalten (vgl. Kap. 2.5.1.5). Die Linkehandseite (LHS) besteht hierbei per Axiom aus nur einer Variablen, welche durch die mathematische Be-

schreibung auf der Rechtheandseite (RHS) ihre parametrische Zuweisung erhält. Diese kann aus Konstanten und Variablen verbunden über mathematische Operationen bestehen. Für die Evaluation der Parametrik beziehen sich die Variablen entweder auf Individuen der Typen *Attribute* oder *DynamicAttribute*. Variablen mit Bezug zu einem *DynamicAttribute* können innerhalb eines vorgegebenen Wertebereich ihren Wert ändern, wohingegen solchen mit Bezug zu einem *Attribute* bereits durch die Produktbeschreibung ein bestimmter Wert zugewiesen wird.

Zur Lösung der aus der parametrischen Produktbeschreibung abgeleiteten Gleichungen werden die in einer Gleichung verwendeten Variablen nach bekannt geltenden und gesuchten Variablen unterschieden. Bekannt geltende Variablen sind solche, denen Werte zugeordnet sind oder die in einer anderen Gleichung als gesucht gelten. Gesuchte Variablen besitzen keine Wertzuordnung und sollen durch das Lösen der Gleichung bestimmt werden. Die Variablen einer Gleichung werden iterativ als bekannt und gesucht gesetzt, bis (1) jede Gleichung nur eine gesuchte Variable besitzt und (2) diese gesuchte Variable in der Gleichung nur in der 1. Potenz vorkommt. Hierdurch stellt jede Gleichung für sich betrachtet eine eindeutig lösbare lineare Gleichung dar.

Der Vorgang zur Bestimmung der gesuchten Variable wird wie folgt durchlaufen: Zunächst werden alle im Graphen hinterlegten Gleichungen eingelesen. Neben den im Graphen enthaltenen Gleichungen werden weitere Gleichungen für die direkte Wertzuweisung der Variablen mit Bezug auf ein *Attribute*-Objekt hinzugefügt. Solche Variablen sind durch die Produktbeschreibung festgeschrieben und nicht veränderbar, wodurch sie als bekannt gelten. Im darauffolgenden Schritt werden alle Variablen auf den LHS der Gleichungen als gesuchte Variablen angenommen. Variablen, die ausschließlich auf den RHS der Gleichungen vorkommen, wird – sofern diese noch keinen durch den Nutzer vorgegebenen Wert zugewiesen bekommen haben – der in der Produktbeschreibung hinterlegten Standardwert des bezogenen *DynamicAttribute*-Objektes zugewiesen.

Gilt jedoch die Variable der LHS einer Gleichung als bekannt (z. B. durch Benutzerangaben oder Definition als gesuchte Variable in einer anderen Gleichung), muss die Bestimmtheit der Gleichung aufrechterhalten werden. Hierfür wird eine der Variablen mit Bezug auf ein *DynamicAttribute*-Objekt, die Standardwerte zugewiesen haben und in keiner anderen Gleichung als gesucht definiert sind, in der betrachteten Gleichung als gesucht definiert. Dies geschieht unter Berücksichtigung der Eigenschaft *variableHigherDOFThan*.

2.5.4 SolConPro-Cloud

Die SolConPro-Cloud wird als Server-Anwendung umgesetzt. Eine solche beinhaltet eine klare Trennung von Backend (Datenverarbeitung und -speicherung) und Frontend (Darstellung, Nutzerschnittstelle). Die SolConPro-Cloud bietet darüber hinaus die Möglichkeit zur Anbindung externer Server (Bspw. Fuseki, BIMserver o. ä.). Eine Übersicht der Komponenten der Cloud ist Abbildung 25 zu entnehmen. Das Frontend dient vorrangig zur webbasierten Definition der Verarbeitungslogik durch den Nutzer. Dies bedeutet, dass darin die auszuführenden Aktionen und deren Verschaltung untereinander festgelegt werden können. Zwar besteht im Frontend auch die Möglichkeit, zu Testzwecken Verarbeitungsvorgänge auszuführen, damit soll jedoch lediglich die Administration ermöglicht und nicht Routineabläufe durchgeführt werden. Die eigentliche Verarbeitung der Daten geschieht auf Basis der hinterlegten Definitionen im Backend. Dieses kann aus Software (bspw. ProductViewer oder [Autodesk Revit](#), siehe Anwendungsfälle in 2.6) heraus direkt angesprochen werden, sodass der Aufruf bestimmter Abläufe automatisiert in die jeweiligen Endanwendungen integriert werden kann. Es findet somit eine starke Vereinfachung der Arbeitsvorgänge für die Endanwender statt, da diese lediglich Eingangsdaten und auszuführende Aktionen definieren können und mit den von der SolConPro-Cloud zurückgegebenen Ergebnissen weiterarbeiten können. Weiterhin besteht die Möglichkeit, zur Kontrolle der durchgeführten Vorgänge die erzeugten Ergebnisse aller Zwischenschritte abzufragen.

Eingehende
Darstellung der
Ergebnisse

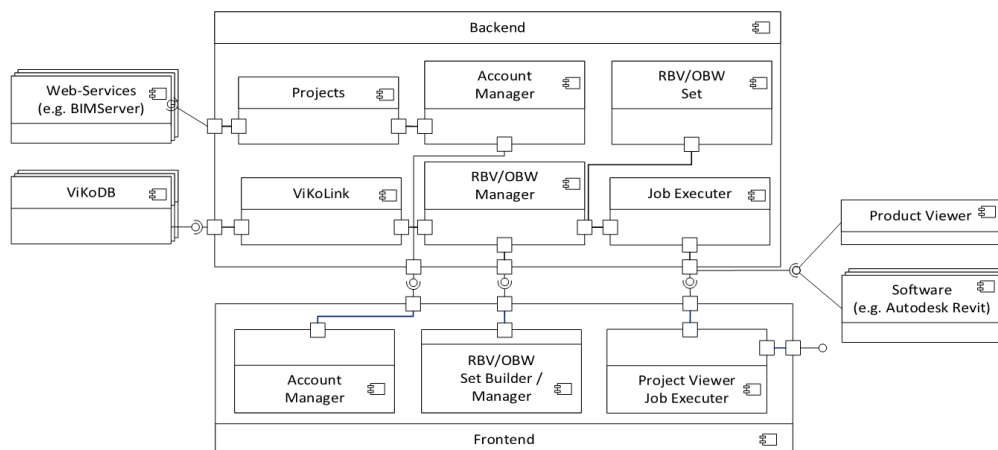


Abbildung 25: Aufbau und Bestandteile der SolConPro-Cloud

2.5.4.1 Backend

Das Backend dient zur Bereitstellung der Verarbeitungslogik der SolConPro-Cloud. Die einzelnen Komponenten sowie die Schnittstellen (engl. *Interfaces*, in Abbildung 25 dargestellt durch Quadrate) werden in den folgenden Abschnitten beschrieben.

Account Management: Um die SolConPro-Cloud nutzen zu können, ist ein Benutzerkonto (Account) nötig. Dieser Account besteht aus einem Namen, einem Benutzernamen und einem Passwort. Er wird durch das Account Management verwaltet. Mittels des Account Managements können Nutzer hinzugefügt, entfernt und bearbeitet werden. Darauf aufbauend können in Zukunft Funktionen implementiert werden, die das Bilden von Organisationen erlauben und Einstellungen für Projekte nutzerübergreifend verfügbar machen. Damit ist gewährleistet, dass eine Anbindung an einen Webservice zentral für ein Projekt innerhalb der Organisation verwendet werden kann. Im Rahmen von SolConPro werden Rollen mit zugehörigem Rechtemanagement nicht betrachtet. Durch diese Komponente wird jedoch die Schnittstelle geboten, ein solches nachträglich einzupflegen.

Umsetzung: □ *JavaScript* im □ *MEAN-Stack*

Projects: Die Backend Komponente Projects stellt die Logik zur Abbildung von Projekten, ihren dazugehörigen Anbindungen an Webservices und zusammenhängenden Methoden bereit. Wichtig für die Integration in den allgemeinen BIM-Prozess ist insbesondere die Möglichkeit, andere Projektplattformen zu verknüpfen. Beispielhaft wurde die Möglichkeit bereitgestellt, eigene Instanzen des BIMserver so zu hinterlegen, dass Gebäudeinformationen über die integrierte □ *IFC*-Schnittstelle zur Weiterverarbeitung von der SolConPro-Cloud abgerufen werden können. Auch können in diesem Bereich alle weiteren projektbezogenen Daten verwaltet werden. Beispielsweise können favorisierte Produkte oder vorverarbeitete Daten gespeichert werden. Im Project Management sind auch die projektbezogenen RBV/OBW-Sets hinterlegt.

Umsetzung: *JavaScript* im *MEAN-Stack*

Interfaces: Um den Zugriff resp. die Anbindung hin zu und von externen Anwendungen auf die SolConPro-Cloud zu ermöglichen, werden Schnittstellen (Interfaces) bereitgestellt. Solche Schnittstellen werden bspw. zur Anbindung des BIMservers oder von □ *Autodesk Revit* benötigt. In beiden Fällen wird □ *JSON* als Standard Datenformat zum webbasierten Austausch über die □ *REST-API* von Inhalten ge-

nutzt. Das Interface spiegelt die API der Backend-Anwendung auf dem SolConPro-Server wider. Über diese Schnittstelle können Daten auf den Server hochgeladen, RBV/OBW-Sets über die Job Execution ausgeführt und die Ergebnisse heruntergeladen bzw. weiterverwendet werden, ohne dass eine Visualisierung für den Nutzer notwendig ist. So kann die entwickelte Schnittstelle genutzt werden, um sich z. B. über *Autodesk Revit* im Addin auf der Cloud einzuloggen und Produktdaten zu finden (siehe 2.6.1.2).

Umsetzung: JavaScript im MEAN-Stack

ViKoLink: Die Virtuelle-Komponenten-Linkdatenbank (VikoLink) dient als eine Datenquelle für die SolConPro-Cloud. Sie besitzt Verbindungen zu beliebig vielen ViKoDBs, aus denen sie Informationen extrahieren kann. Die Produktdaten können einerseits über gezieltes Abfragen eines bestimmten Produktes als auch über Produktsuchen von der ViKoLink angefordert werden. Sollte eine solche Anforderung ein parametrisches Produkt betreffen, wird dies anhand der übergebenen Parameter evaluiert und berechnet. Eine genaue Beschreibung der Funktionalitäten und Implementierung der ViKoLink ist in Kapitel 2.5.3 gegeben.

Umsetzung: □ Java-Anwendungen mit integrierten □ Jena-Bibliotheken und Anbindung an einen Sage Cell Server, Fuseki-Server

RBV/OBW-Set: RBV/OBW-Sets können jeweils aus verschiedenen Bausteinen (Komponenten) zusammengesetzt werden, die in Abbildung 26 dargestellt sind und im Folgenden erläutert werden. Sie können von Benutzern über das Frontend zusammengestellt werden. Backend-seitig werden die so erstellten Kombinationen aus RBVs und OBWs in einer Datenbank abgelegt. Dies erfolgt projektbezogen, sodass einmal definierte Sets beliebig oft und – falls gewünscht – mit unterschiedlichen Datensätzen ausgeführt werden können. Die auszuführenden Schritte werden durch den Node.JS-Server in Listen, sog. Jobs, abgelegt, sodass sie getrennt von dem MEAN-Stack durch eine Java-Anwendung (s. *Job Execution*) abgearbeitet werden können.

Umsetzung: Eintrag in □ MongoDB eines in JSON serialisierten JavaScript-Objekts

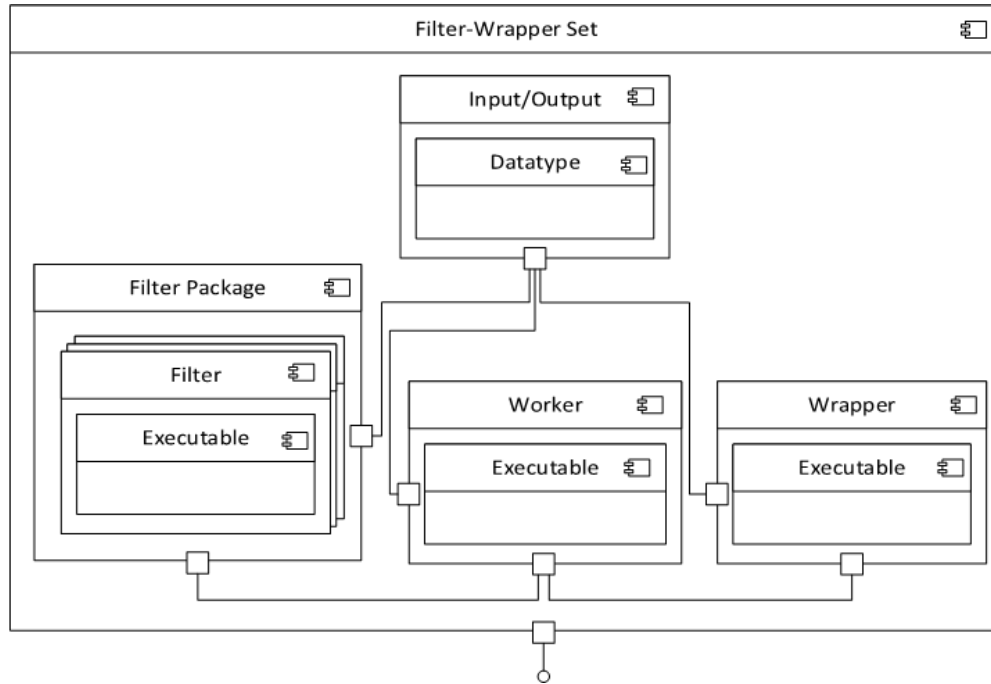


Abbildung 26: Aufbau und Bestandteile der RBV/OBW-Sets

Datatype: Zur eindeutigen Beschreibung der über die SolConPro-Cloud verarbeitbaren Daten können beliebige Datentypen und Dateiformate (zusammengefasst als *Datatype*) hinterlegt werden. Dies können bestehende (Datei-)Formate, wie bspw. TTL oder die Industry Foundation Classes ([IFC](#)) sein, aber auch neu entwickelte, wie speziell auf das Forschungsprojekt angepasste JSON-basierte Strukturen. Datatypes werden bei der Zusammenstellung von RBV/OBW-Sets benötigt, um bei der Verknüpfung der verschiedenen Bausteine prüfen zu können, ob diese kompatibel sind.

Umsetzung: Eintrag in MongoDB eines in JSON serialisierten JavaScript-Objekts

Input / Output: Bei der Definition eines RBVs, Filters, Workers oder OBWs müssen eindeutige Datenquellen und Datentypen für alle Inputs und Outputs festgelegt werden. Dies ist nötig, da für alle dahinterliegenden, auszuführenden *Executables* eindeutige Vorschriften zur Verarbeitung der Daten hinterlegt sind, die in Abhängigkeit der jeweiligen Inputdaten Output-Daten erzeugen. Durch die eindeutige Beschreibung eines RBV/OBW-Sets durch die Zusammenstellung einzelner RBV, Filter, Worker und OBW sind damit direkt sowohl Input (Input(s)) des/der jeweils in dem erstellten Graphen zuerst auszuführenden Komponente(n)) als auch Output (Output der zuletzt ausgeführten Komponente(n)) des gesamten Sets definiert.

Umsetzung: Eintrag in MongoDB eines in JSON serialisierten JavaScript-Objekts

Executable: Filter, Wrapper und Worker sind in der Lage, unterschiedliche Daten auf verschiedene Weisen zu verarbeiten. Die dazu benötigten Anwendungen und Skripte können entsprechend vielfältig sein. Um dies zu ermöglichen, wird beim Anlagen der Filter, Wrapper und Worker eine auszuführende Anwendung mit beliebigen Parametern derart angegeben, dass der eingegebene Befehl lediglich in einer Linux-Konsole ausführbar sein muss. Somit können verschiedene Programmiersprachen und Frameworks genutzt werden. Bspw. werden Java-Archive (.jar) oder [Python](#)-Anwendungen (.py) wie folgt aufgerufen:


```
Last login: Mon Apr 16 09:23:09 2018 from 172.20.1.80
solconpro@solconproSERVER:~$ java -jar filter.jar
```

Abbildung 27: Aufrufen einer Java-JAR-Anwendung

```
Last login: Mon Apr 16 09:23:09 2018 from 172.20.1.80
solconpro@solconproSERVER:~$ python skript.py -i input.txt -o output.json
```

Abbildung 28: Aufrufen eines Python-Skripts

Umsetzung: Eintrag in MongoDB eines in JSON serialisierten JavaScript-Objekts

Filter: Die Basis von Filtern bilden Skripte oder Anwendungen (s. *Executable*), die die eigentlichen Filterprozesse durchführen und auf Regeln oder vorgegebenen Ablaufstrukturen basieren. Diese Filterprozesse haben die Aufgabe, Datenmengen zu reduzieren, nicht aber Daten anzureichern oder zu verändern. Daher muss das Datenformat (*Datatype*) des Outputs immer dem des Inputs entsprechen. Filter müssen somit genau einen definierten Input sowie Output besitzen. Je Filter kann eine Datenquelle verarbeitet werden. Filter haben weiterhin eine Bezeichnung sowie eine Beschreibung.

Umsetzung: Eintrag in MongoDB eines in JSON serialisierten JavaScript-Objekts

Rule Based View (RBV) / Filterpackage: In 2.4.2 wurde das Konzept zur Datenfilterung mittels RBV beschrieben. Die RBV können als Filterpakete (Filterpackages) aufgefasst werden, die aus einer Zusammenstellung einzelner Filter bestehen.

Analog zu bspw. Views in SQL¹ sind Views der RBV als Sichten auf bestimmte, von einem RBV verarbeitete Datenmengen definiert. So kann eine View den Umfang der Daten vor und eine weitere den Umfang der Daten nach dem Filterprozess darstellen. Auch können Views für Kontrollen während des Filtervorgangs ausgegeben und verwendet werden (z. B. in Form von Dateien, die nach jeder Ausführung eines *Executable* erzeugt und nicht gelöscht werden).

RBVs und die enthaltenen Filter werden stets in den beschriebenen Strukturen verwendet. Sie können nicht direkt an externe Anwendungen angebunden werden, wodurch nie Inhalte direkt aus Filtern verwendet werden. Dazu ist immer ein OBW notwendig.

Die einfachste Form eines RBV besteht aus einem Filterpaket mit einem leeren Filter, bei dem die Datenmenge des Outputs der des Inputs entspricht. Ein solches ist bspw. notwendig, wenn Datensätze vollständig abgefragt werden sollen. Weitere, komplexere Beispiele sind in den in 2.6.1 behandelten Use Cases zu finden.

Umsetzung: Eintrag in MongoDB eines in JSON serialisierten JavaScript-Objekts

Ontology Based Wrapper (OBW): Die in 2.4.4.3 konzeptionierten OBW wurden als Komponente der Cloud umgesetzt und beinhalten neben dem Wissen über die zu überführenden Formate einen Namen, eine Beschreibung und ein *Executable*. Dieses Skript kann auf der SolConPro-Cloud ausgeführt werden und übersetzt die Inputdatei in das Output-Dateiformat und ist somit der wesentliche Bestandteil des OBW. Das Ergebnis des OBW steht über ein Interface auch außerhalb des Backends zur Verfügung und kann somit in externen Softwareprogrammen genutzt und die Daten verarbeitet werden.

¹ https://www.w3schools.com/sql/sql_view.asp

Die einfachste Form des OBW ist ein Wrapper, bei dem das Inputformat dem Output-Dateiformat entspricht und somit keine Umwandlung stattfindet. Weitere, komplexere Beispiele sind in den in 2.6.1 behandelten Anwendungsfällen zu finden.

Umsetzung: Eintrag in MongoDB eines in JSON serialisierten JavaScript-Objekts

Worker: Die in 2.4.4.2 vorgestellten Worker werden in der Cloud analog zu den RBV und OBW als Komponenten umgesetzt. Sie beinhalten einen Namen, eine Beschreibung, ein *Executable* und die Information, welche Aufgaben dieser erfüllt. Im Gegensatz zu den RBV und OBW könne Worker jedoch mehrere Inputs besitzen. Die *Executables* der Worker beinhalten Verarbeitungsprozesse, die nicht ein Reduzieren oder Umwandeln der Daten enthalten, wie zum Beispiel ein Anreichern der Produktinformationen um Übersetzungen oder im Allgemeinen das Zugreifen auf die ViKoLink (siehe auch 2.6.1.1).

Umsetzung: Eintrag in MongoDB eines in JSON serialisierten JavaScript-Objekts

RBV/OBW-Management: Die zuvor beschriebenen Komponenten können sehr flexibel in verschiedenen Kombinationen zusammengesetzt werden. Abbildung 29 und Abbildung 30 veranschaulichen zwei Beispiele für Zusammenstellungen; jeweils zunächst schematisch als Datenflussdiagramm. RBV, Wrapper und Worker können – solange die Verarbeitung in einer zulässigen Reihenfolge erfolgt – beliebig zusammengestellt werden. Die zusammengestellten Sets werden auf der SolConPro-Cloud hinterlegt.

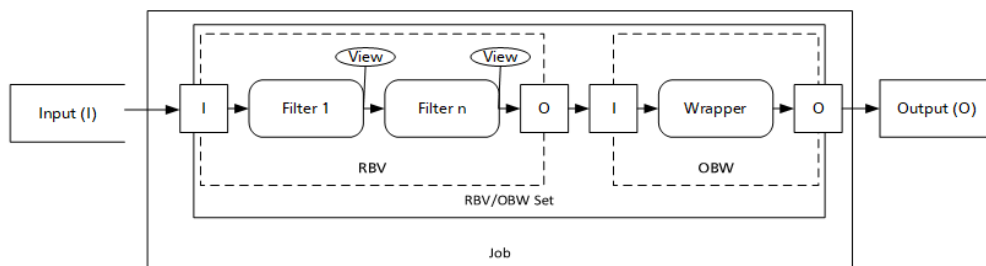


Abbildung 29: Datenfluss in einem RBV/OBW-Set mit Wrapper

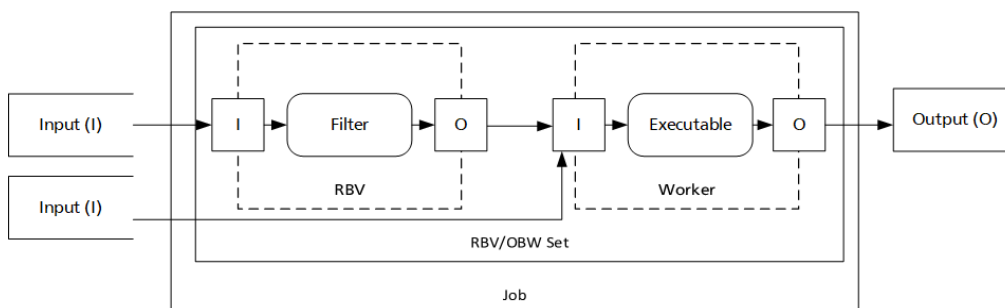


Abbildung 30: Datenfluss in einem RBV/OBW-Set mit Worker

Umsetzung: JavaScript, [GoJS](#) im MEAN-Stack

Jobs: Zur Durchführung konkreter RBV/OBW-Sets werden diese inkl. der auszuführenden Skripte, Datenquellen, Datenspeicherorte in einer separaten Datenbank hinterlegt. Diese sog. Jobs, die Berechnungen der teils aufwändigen Filter, Wrapper und Worker beinhalten, können so getrennt von der SolConPro-Cloud durchgeführt werden. Anwender können die Ergebnisse der RBV/OBW-Sets zu beliebigen Zeit-

punkten abrufen, ohne auf dem Frontend der Cloud (Webseite) verbleiben zu müssen.

Umsetzung: Eintrag in MongoDB eines in JSON serialisierten JavaScript-Objekts

Job Execution: Zur Ausführung der Jobs wird eine in `Java` geschriebene Anwendung eingesetzt. Diese läuft permanent im Hintergrund und umfasst im Wesentlichen einen Prozess, der alle in der Datenbank wartenden Jobs nacheinander ausführt. Den Jobs werden dabei verschiedene Zustände (wartend, in Ausführung, erfolgreich, fehlgeschlagen) zugewiesen.

Umsetzung: Java-Anwendung

2.5.4.2 Frontend

Um die vom Backend bereitgestellten Funktionalitäten für Anwender visuell aufzubereiten und zugänglich zu machen, wird ein Frontend benötigt. Dieses wird im Rahmen des `MEAN`-Frameworks in Form einer Webseite bereitgestellt (<http://cloud.solconpro.de:8080/>). Es beinhaltet in der hier vorgestellten Implementierung einen Bereich, in dem Komponenten (Filter, OBW, Worker und RBV/OBW-Sets) erstellt und verwaltet werden, einen, in dem projektbezogene Informationen eingesehen werden können, sowie einem Bereich zur (Vor-)Verarbeitung von Daten (Ausführen von RBV/OBW-Sets).

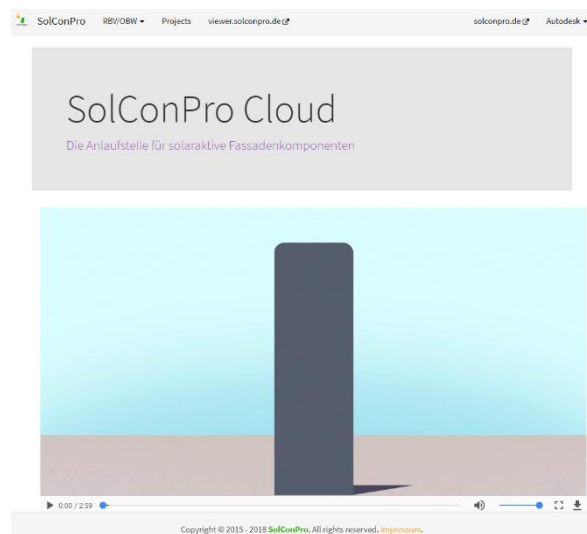


Abbildung 31: Startseite der SolConPro-Cloud

RBV/OBW Set Manager: Der RBV/OBW Set Manager stellt eine grafische Oberfläche zur Erstellung und Bearbeitung von Datentypen, Filtern, Filterpaketen, OBWs und Workern bereit. Die einzelnen Seiten sind jeweils ähnlich zueinander aufgebaut. Abbildung 32 zeigt exemplarisch den Bereich zum Verwalten der Worker. In Abbildung 33 wird die Eingabemaske zum Erstellen eines neuen Workers dargestellt.

Eingehende
Darstellung der
Ergebnisse

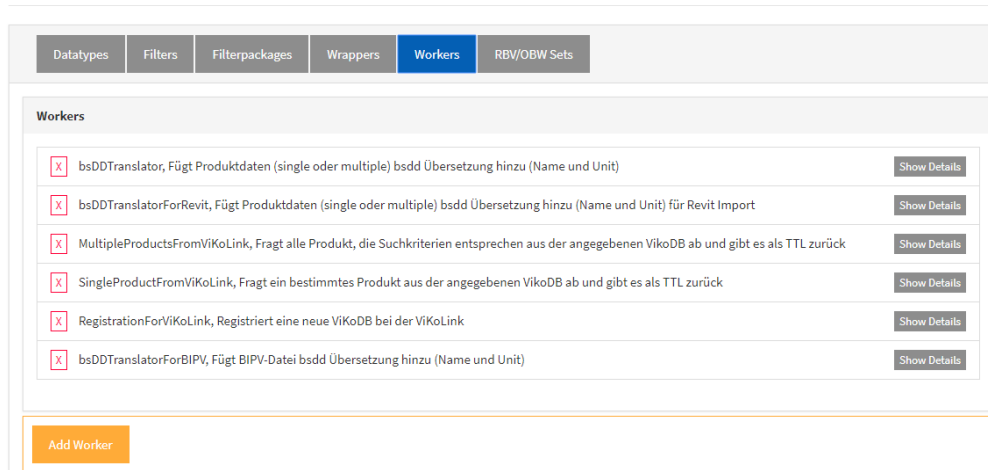


Abbildung 32: Übersicht der Worker

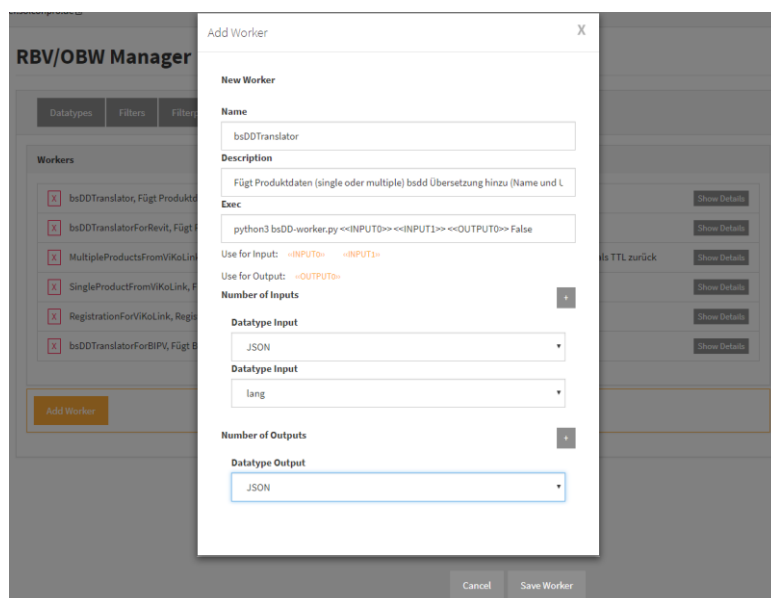


Abbildung 33: Erstellen eines neuen Workers

RBV/OBW Set Builder: Der RBV/OBW Set Builder erlaubt es den Benutzern über das Frontend Filter, RBVs, OBWs, Worker zu RBV/OBW-Sets zusammenzustellen und zu verwalten. Dazu werden mittels der grafischen Oberfläche und der [GoJS](#)-Bibliothek die einzelnen Bausteine per Drag&Drop miteinander verbunden (siehe Abbildung 34 und Abbildung 35). Die Bausteine werden als Knoten und deren Verbindungen als Kanten in einem gerichteten Graphen hinterlegt. Dieser Graph wird bei der Ausführung eines Sets als Basis genutzt. Er wird vom ersten bis zum letzten Knoten durchlaufen und die hinter den Bausteinen liegenden Executables werden nacheinander ausgeführt.

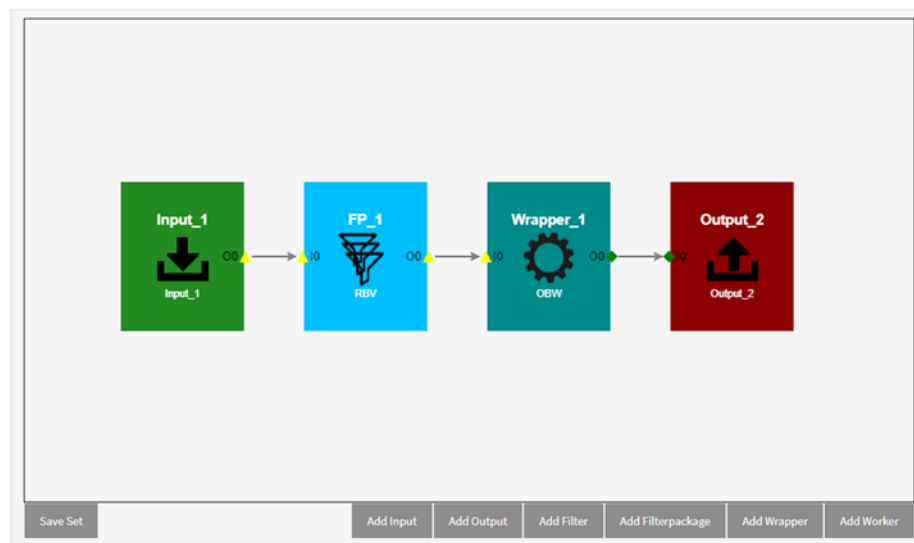


Abbildung 34: Visualisierung der Zusammenstellung des in Abbildung 29 beschriebenen RBV/OBW-Sets auf der SolConPro-Cloud im Set Builder

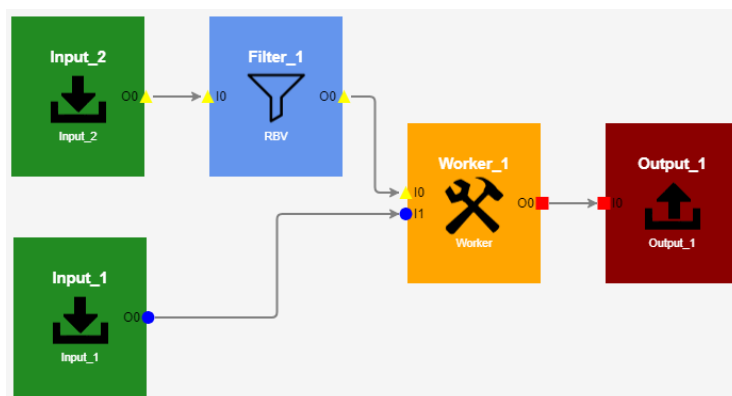


Abbildung 35: Visualisierung der Zusammenstellung des in Abbildung 30 beschriebenen RBV/OBW-Sets auf der SolConPro-Cloud (Auszug aus Set Builder)

Umsetzung: GoJS, [JavaScript](#) im MEAN-Stack

Account Manager: Im Frontend wurde kein komplexes Nutzermanagement umgesetzt. Der Account Manager umfasst derzeit eine einfache Registrierung, eine Möglichkeit zum Login auf der SolConPro-Cloud sowie eine Seite zum Anzeigen der Benutzerdaten.

Umsetzung: [JavaScript](#) im MEAN-Stack

Project Viewer / Job Executer Im Frontend können Projekte auf einer Übersichtsseite verwaltet werden. Neben Informationen zu dem Projekt selbst sind hier hochgeladene Dateien sowie eine Anbindung an einen BIMserver vorhanden. In diesem Bereich ist auch das Ausführen der in Projekten hinterlegten RBV/OBW-Sets möglich. Dazu wird das auszuführende Set mit (einer) Datenquelle(n) ausgewählt und gestartet. Nachdem der Job Executer die Ausführung des jeweiligen Sets beendet

2.5.5 Firmen- und softwareübergreifendes Kollisionsmanagement

Mit dem Thema firmen- und softwareübergreifendes Kollisionsmanagement werden vier im Rahmen von SolConPro konzipierte Themenbereiche gleichzeitig adressiert: die Nutzung von Webtechnologien, das Thema [Gitlab](#) und *Continuous Delivery* (CD), die applikations- und firmenübergreifende Datenbearbeitung mittels DMZ und die Datenanbindung bestehender proprietärer Softwareprogramme. Aus diesem Grund soll die Beschreibung des Kollisionsmanagements ausführlicher gestaltet werden.

Die Behandlung von Kollisionen (engl. *clashes*) spielt in BIM-Projekten eine herausragende Rolle. Sie stellt gleichzeitig auch eine wesentliche Kommunikationsebene zwischen den Baubeteiligten dar. Viele Änderungen, die das Gebäudemodell betreffen, sind gleichzeitig auch für einen oder mehrere andere Baubeteiligte relevant, weshalb es wichtig ist, zu Kollisionen firmenübergreifende Kommunikationswege anzubieten.

Derzeit finden Kollisionsprüfungen in Softwareprogrammen wie *Autodesk Navisworks* oder [Autodesk Revit](#) statt, d. h. in lokal auf dem Rechner installierten Applikationen. Aus diesem Grund müssen Themen mit Abstimmungsbedarf zwischen unterschiedlichen Firmen zurzeit mittels Screenshots (und z. B. telefonischer Einigung) gelöst werden, da die lokale Installationsform keinen synchronen Zugriff mehrerer Firmen auf dasselbe Gebäudemodell zulässt. Dies führt vor allem bei komplexeren Themen wie der Installation solaraktiver Fassadenkomponenten zu erheblichem Mehraufwand.

Die im Rahmen dieses Forschungsprojekts gewonnenen Erkenntnisse zur applikationsunabhängigen Datenhaltung lassen sich anhand des vorliegenden Themenfelds hervorragend demonstrieren.

Kollisionen und deren Kommentierungen können durch einfache Textdateien beschrieben werden und sind auch über die APIs der jeweiligen Softwareprogramme adressierbar. Auf diese Weise können deren Inhalte in externe Datenbanken synchronisiert werden. Diese wiederum werden innerhalb einer DMZ zugänglich gemacht, wodurch ein applikations- und firmenübergreifender Informationsaustausch gewährleistet wird.

Die APIs der *Autodesk*-Produkte *Navisworks* und *Revit* müssen in der Programmierumgebung .NET bedient werden. Für beide Programme wurde im Rahmen des Forschungsprojekts ein Frontend geschaffen, das ein Nutzerverwaltungssystem (inkl. Registrierung und Login-Funktion, siehe Abbildung 36) bereitstellt und angemeldeten Nutzern Zugriff zu projektspezifischen Kollisionen und dessen Kommentierungen bietet.

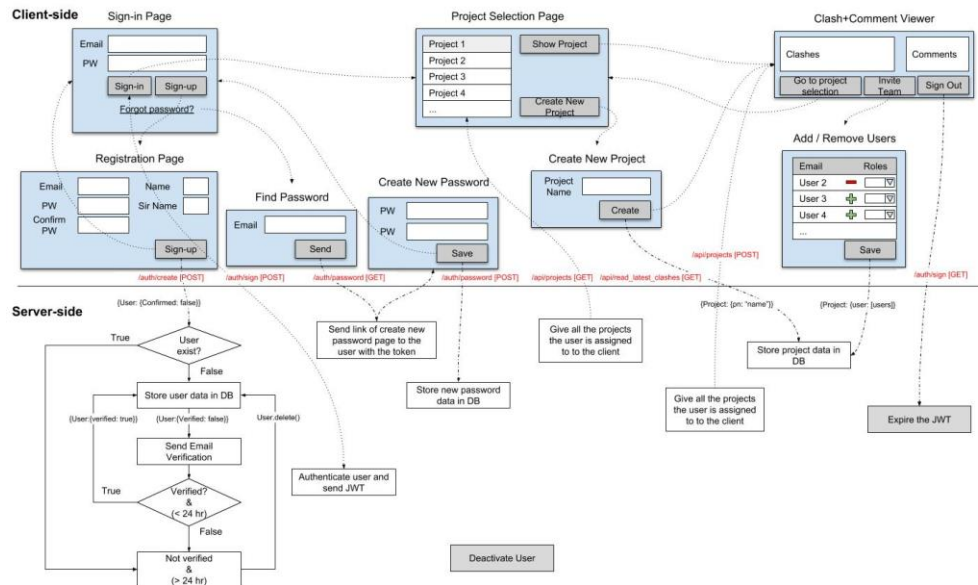


Abbildung 36: Technische Übersicht des auf C#-Basis implementierten Nutzer- und Projektmanagement-Systems

Die eigentliche Bedeutung des Kollisionsmanagements für das Forschungsprojekt SolConPro aber liegt in der Bereitstellung (*Deployment*) des Backends. Die Plug-Ins für **Autodesk Revit** und **Navisworks** sollen auch für beliebige externe Firmen verfügbar gemacht werden, weshalb eine zentrale Datenbank hinterlegt werden muss. Diese ist via HTTP-Abfragen über clashmgmt-bim5d.ds.strabag.com erreichbar. Die externe Abrufbarkeit der vorliegenden Datenbank erlaubt es anderen Firmen, eigene BIM-Softwareprogramme an dieselbe Datenmenge anzubinden, sofern das verwendete Programm mit Kollisionen umzugehen weiß und eine API dessen Adressierung erlaubt. Sobald die Anbindung erfolgt ist, können Kollisionen auch in deren Softwareprogrammen bearbeitet und mit Kommentaren versehen werden. Die Kollisionsdaten selbst sind also sowohl firmen- als auch applikationsunabhängig, auch wenn zu ihrer Erstellung proprietäre Softwareprogramme erforderlich sind. (Für den Fall, dass nur Kommentierungen bestehender Kollisionen erfolgen sollen, ist das Kollisionsmanagementprogramm auch als eigenständige Webapplikation verfügbar, für das interaktive Selektieren von kollidierenden Objekten oder das Hochladen neuer Kollisionen in die Datenbank sind die proprietären Plug-Ins erforderlich.)

Da die vorliegende Datenbank innerhalb einer DMZ erstellt wurde, befindet sich diese innerhalb des direkten Kontrollbereiches der Züblin-internen IT-Abteilung. Die Kollisionsdaten wären auch direkt mit anderen, auch vertraulichen Projektinformationen verknüpfbar (was im vorliegenden Fall keine Anwendung findet). Zudem erlaubt die vorliegende Konfiguration eine direkte Verarbeitung der Kollisionsinformationen, z. B. zu firmenübergreifenden graphischen Übersichten (*Dashboard*, siehe Kap. 2.6.2.2).

Die vier Entwicklungsschritte der IT-Architektur werden in Abbildung 37 bis Abbildung 40 illustriert. Abbildung 37 zeigt die ursprüngliche Methode, die zur Synchronisation von Kollisionsdaten zwischen Softwareprogrammen zur Anwendung gekommen wäre. Mit der Schaffung einer software-unabhängigen Datenverwaltung, dargestellt in Abbildung 38, werden die Daten durch beliebige Routinen darstellbar; deren Auswertung ist nicht mehr auf die software-internen APIs beschränkt. Dadurch können auch beliebige weitere Softwareprogramme leichter angebunden werden. Ein weiterer Fortschritt ist mit der Implementierung von Webservices zur Datenanbindung erreicht, da die Datenbank im gesamten Intranet adressiert werden

kann und Auswertungen so auch mittels Smartphone zugänglich sind (Abbildung 39; die Auswertung erfordert keine lokale Installation mehr und ist somit unabhängig vom Betriebssystem). Abbildung 40 schließlich zeigt den Vorteil der Verwendung von Demilitarisierten Zonen auf. Diese erlauben eine Datensynchronisierung via Internet. Dadurch wird die Erstellung eines bidirektionalen Datenzugriffs durch andere Firmen ermöglicht, bei vollständigem und verständlichen Vorliegen einer API-Dokumentation sogar ohne technische Rücksprache. Auf diese Weise können Informationen zukünftig software- und firmenübergreifend bearbeitet werden. Dies stellt für die Realisierung einer ganzheitlichen Planung einen erheblichen Fortschritt dar, der auf alle Arten von Daten übertragbar ist.

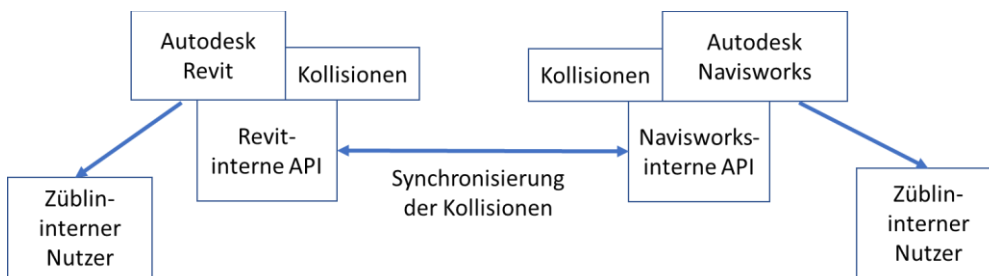


Abbildung 37: Implementierungsmethode zur softwareübergreifenden Synchronisierung von Kollisionsdaten vor den Projektergebnissen

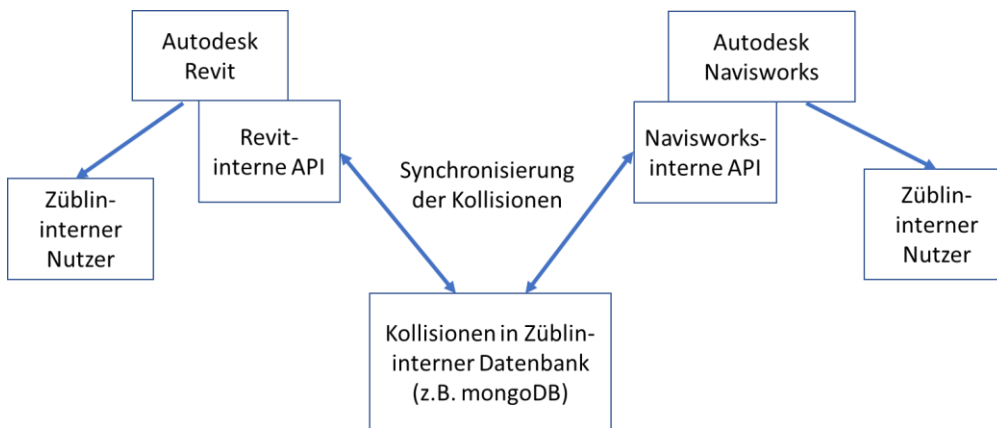


Abbildung 38: Softwareübergreifende Synchronisierung durch Verwendung einer Züblin-internen Datenbank und flexiblerer Anbindung weiterer Softwareprogramme

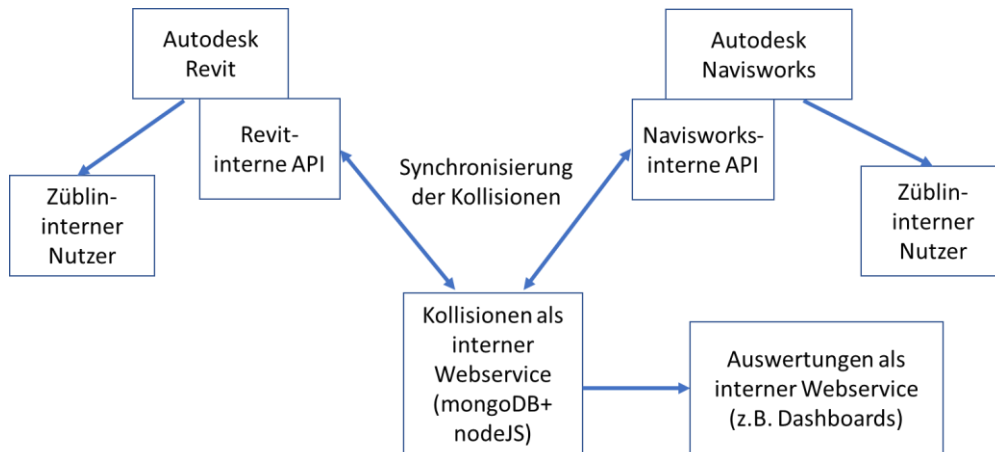


Abbildung 39: Softwareübergreifende Synchronisierung mittels Webservice und internen browserbasierten Datenauswertungen

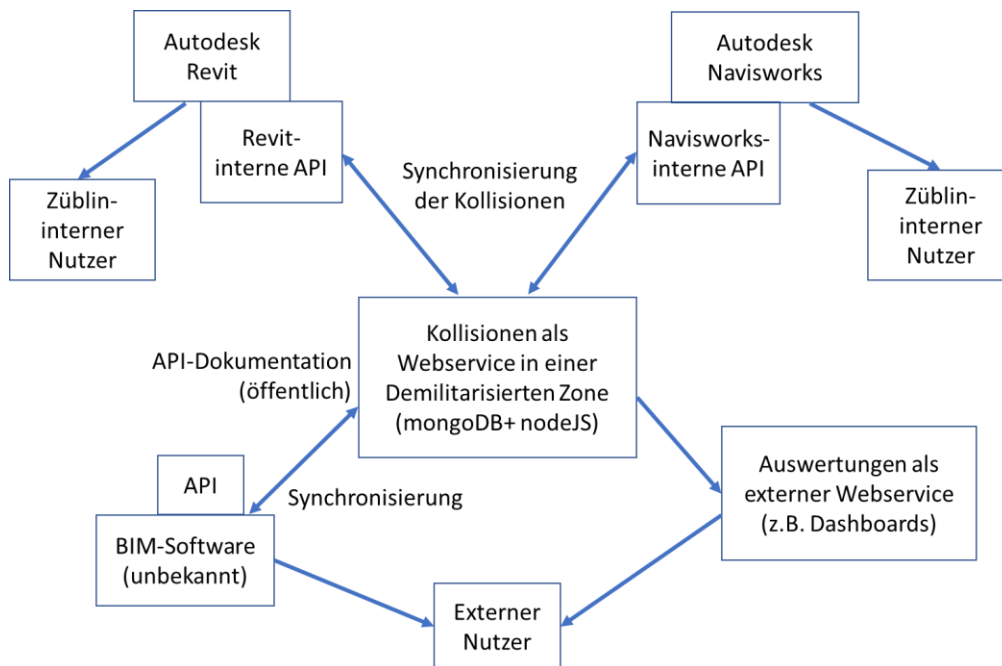


Abbildung 40: Software-unabhängige Kollisionsdatenverwaltung in einer Demilitarisierten Zone mit beliebigen Datenanbindungsmöglichkeiten durch externe Firmen („software- und firmenübergreifendes Kollisionsmanagement“)

Das Kollisionsmanagement wurde mit einer Kommentarfunktion versehen, um den Beteiligten eine digitale Kommunikationsmöglichkeit zu spezifischen Kollisionen zu verschaffen. Sowohl die Kommentare als auch die Kollisionen selbst können als einfache `JSON`-Objekte übertragen werden.

Um die vorliegenden Daten für externe Firmen verwendbar zu machen, ist für einen externen Nutzer einerseits erforderlich, die Daten abrufen zu können, und andererseits, sie auch zu verstehen. Beide Fragestellungen müssen durch eine öffentlich zugängliche API-Dokumentation¹, die am besten via Webseite zur Verfügung gestellt wird, beantwortet

werden.

Die Endpunkte des Kollisionsmanagements sind in Tabelle 1 Tabelle 3 gelistet.

Route	HTTP-Methode	Aufgabe
<i>/auth/users</i>	<i>GET / POST</i>	Ausgabe aller Benutzer / Registrierung eines neuen Benutzers
<i>/auth/confirm</i>	<i>POST</i>	Senden der Bestätigungs-E-Mail
<i>/auth/sign</i>	<i>GET / POST</i>	Einloggen / Ausloggen des Benutzers
<i>/auth/me</i>	<i>GET</i>	Rückgabe der Benutzerdaten
<i>/auth/password</i>	<i>POST</i>	Versenden eines neuen Passworts und einer Bestätigungs-E-Mail
<i>/auth/verified/:token</i>	<i>GET</i>	Verifizieren der Registrierung eines neuen Benutzers
<i>/api/projects</i>	<i>GET / POST / PUT</i>	Ausgabe aller Projekte, die ein Benutzer einsehen darf / Erstellen eines neuen Projekts / Aktualisieren eines Projekts
<i>/api/read_all_projects</i>	<i>GET</i>	Ausgabe aller Projekte in der Datenbank
<i>/api/delete_project</i>	<i>PUT</i>	Entfernen eines Projekts
<i>/api/create</i>	<i>POST</i>	Erstellen eines neuen Kommentars / einer neu-

¹ Die API-Funktionen selbst sollen langfristig ebenfalls einem Schema folgen, damit aus diesen automatisiert Quellcodes erzeugt werden können. Dieser Schritt wurde im vorliegenden Forschungsprojekt noch nicht getätigt.

<i>/api/create_list</i>	<i>GET / POST</i>	en Kollision Hinzufügen ei- ner Kollisionslis- te zu einem Projekt
<i>/api/read_all_comments/?p_id=project_id</i>	<i>GET</i>	Ausgabe aller Kommentare eines Projektes
<i>/api/read_comments_from_clash/?p_id=project_id&c_id=clash_id</i>	<i>GET</i>	Ausgabe aller Kommentare zu einer Kollision in einem Projekt
<i>/api/read_latest_clashes/?p_id=project_id</i>	<i>GET</i>	Ausgabe kürzli- cher Kollisionen eines Projekts
<i>/api/table</i>	<i>POST</i>	Hinzufügen ei- nes Eintrags zur Nutzerbeschrei- bung

Tabelle 3: Endpunkte des Kollisionsmanagementsystems

Jedem Endpunkt muss für jede verfügbare HTTP-Methode eine ausführlichere Beschreibung zugrunde liegen, die aus dem Funktionsinhalt, einer beispielhaften HTTP-Abfrage, der zu sendenden Daten, der vorgesehenen Serverantwort (sowohl bei erfolgreicher als auch bei fehlerhafter Abfrage) und deren Schemabeschreibung besteht. Aufgrund der Wichtigkeit der vollständigen Beschreibung der API-Funktionalitäten ist – repräsentativ für alle Endpunkte dieses Berichts – im Anhang C das Beispiel */api/create* POST geschildert.

Für die Entwicklung des Kollisionsmanagements wurde die DevOps-Methode verwendet. Das Continuous Integration (CI) erlaubt die gleichzeitige Bearbeitung von Backlogs durch unterschiedliche Programmierer, während das *Continuous Delivery* (CD) ein direktes Deployment des Webservices möglich macht. Das Deployment wird in einer Demilitarisierten Zone durchgeführt, was Fehlerkorrekturen ohne Beeinträchtigung bestehender Kollisionsinformationen für einen späteren Betrieb besonders bedeutend macht. Zudem gliedert sich das Webhosting so in die Züblin-interne IT-Abteilung ein, wodurch Themen wie Lastverteilung, Betriebssicherung und Datenarchivierung automatisch abgehandelt werden.

Das Kollisionsmanagement verdeutlicht die Möglichkeiten, die sich bei einer applikationsunabhängigen Datenhaltung ergeben, stellt aber natürlich nur einen allerersten Schritt dar. Zur Vermeidung von Datenredundanz und für eine zufriedenstellende Datensynchronisierbarkeit langfristig alle Daten applikationsunabhängig verwaltet werden können. Die im Zuge des beschriebenen Beispielfalls erschlossene IT-Architektur (DMZ, CI / CD, Webserver, Datenbanken, Nutzermanagement, [REST-API](#)) kann jedoch auf alle anderen Datenmengen übertragen werden. Die in SolConPro gewonnenen Erkenntnisse bewirken innerhalb des Bereichs BIM.5D den Beginn eines strategischen Umdenkens, das durch Prototypen wie das beschriebene Kollisionsmanagement gefördert wird.

2.6 Arbeitspaket 6 – Validierung und Verifikation

Die Durchführung einer Validierung und Verifikation der in SolConPro entwickelten Herangehensweisen muss zwei Ebenen berücksichtigen. Auf der einen Seite wurde in SolConPro eine IT-Konzeption erarbeitet, die die bisher gängigen Modelle (v. a. aufbauend auf buildingSMART) in Frage stellt. In AP6 werden daher die strukturellen Forderungen (an Server-Architekturen und Software-Hersteller) beschrieben, die aus der Notwendigkeit der software-unabhängigen Datenhaltung entstehen. Diese

betreffen alle im Bauwesen digitalisierten Informationen, wodurch erst eine ganzheitliche Bearbeitung der Themenstellung möglich wird. Das Kapitel 2.6.2 befasst sich deshalb sowohl mit Fragestellungen zur Serverarchitektur als auch zur Integration des in SolConPro entwickelten IT-Konzepts in die bestehende BIM-Softwarelandschaft. Auf der anderen Seite wurde im Projekt besonderer Fokus auf die Speicherung und Verwaltung produktbezogener Inhalte (inklusive der semantischen und geometrischen Beschreibung) gesetzt, deren serverseitige Umsetzung die größte Hürde in der Umsetzung der Datenstrukturen darstellt. In den unten beschriebenen Anwendungsfällen wird gezeigt, dass auch die software-unabhängige Umsetzung von Produktbeschreibungen in bestehende BIM-Softwareabläufe integriert werden kann.

2.6.1 Anwendungsfälle zur Einbindung geometrischer und semantischer Produktinformationen in bestehende BIM-Abläufe

Mittels Ontologien wurden Produktinformationen generisch hinterlegt und über Web-Applikationen miteinander zugänglich verknüpft. Eine zentrale Anlaufstelle für Anfragen durch die ViKoLink wurde geschaffen, welches es ermöglicht über das Netzwerk Zugriff auf alle angebundenen Produktdatenbanken zu erhalten. Allerdings muss durch die folgenden Anwendungen gezeigt werden, dass die generische Ablage der Informationen innerhalb von Ontologien auch für den Planer, Hersteller und Anwender nutzbar zur Verfügung steht. Wie die semantischen und geometrischen Produktinformationen einfach nutzbar für den Endanwender vorliegen, zeigen folgend implementierte Anwendungsfälle. Zudem wird die Funktionalität der vorgeschlagenen mehrschichtigen Produkt-Ontologie demonstriert. Unter diesem Gesichtspunkt beinhalten die einzelnen produktbezogenen Anwendungsfälle, Abfragen und Filterroutinen, die die in 2.4.3.1 formulierten Kompetenzfragen beantworten müssen.

2.6.1.1 Distributive Verteilung virtueller Komponenten und deren Visualisierung

Zur Validierung der entwickelten mehrschichtigen Ontologie (siehe 2.5.1) und insb. der Produktontologie (siehe 2.5.1.2) zur Repräsentation virtueller Komponenten werden ausgewählte Produkte im Schema der Ontologie modelliert: Zwei BIPV-Module, ein BIST-Modul, ein Wechselrichter sowie zwei verschiedene Zelltypen (vgl. Tabelle 4).

Tabelle 4: Übersicht modellierter Produkte

Produkttyp	Modellierungsarten	Bezeichnung
BIPV-Modul 1	3 statische Modellierungen	GebN_V3-5
BIPV-Modul 2	3 statische Modellierungen	PVShade_V1-3
BIST-Modul	3 statische Modellierungen	CE_V1-3
Wechselrichter	1 statische Modellierung	Inverter
PV-Zelle 1	1 statische Modellierung	PVCell_1
PV-Zelle 2	1 statische Modellierung	PVCell_2
BIPV-Modul 1 mit PV-Zelle 1	1 parametrische Modellierung	GebN_V1
BIPV-Modul 1 mit PV-Zelle 2	1 parametrische Modellierung	GebN_V2

Um die Modellierungsfreiheit der Ontologie und einhergehende Abfragen auf die Produkte unabhängig von der Strukturierung der Modellierung validieren zu können, werden die BIPV-Module jeweils auf drei verschiedene Arten modelliert. Hierbei

variieren die Modellierung bei den Zusammenfassungen der *Element*-Objekte in *Assembly*-Objekte, der Verwendung von *Repetition*- sowie *DynamicEntity*-Objekten und Berücksichtigung komplexer Attribute wie 2D-Listen oder □ *WKT*. Überdies finden für das BIPV-Modul 1 zwei parametrische Modellierungen statt, die u.a. in Anwendungsfall 2.6.1.2 Anwendung finden. Diese parametrischen Modellierungen berücksichtigen auch Konzepte des Linked Data, da sie andere Produktbeschreibungen (PV-Zellen) verwenden, ohne die Daten dieser Produkte zu kopieren. Die modellierten Produkte / virtuellen Komponenten werden verteilt auf verschiedene Produktdatenbanken (ViKoDBs) abgelegt und, angebunden über die ViKoLink, von der SolConPro-Cloud zur Verfügung gestellt (siehe 2.5.2 und 2.5.3). Zur Validierung werden verschiedene Abfragen über den auf der SolConPro-Cloud zur Verfügung gestellten Webservice auf die Daten ausgeführt. Die zurückgegebenen Produkte entsprechen hierbei den in der Abfrage vorgehenden Suchkriterien (z. B. Modulbreite) und werden unabhängig von der Modellierung des Produkts korrekt zurückgegeben. Zur Darstellung der Suchergebnisse sind in Tabelle 5 Auszüge der modellierten Produkteigenschaften und in Tabelle 6 die Ergebnisse folgender Produktsuchen zu entnehmen:

1. Länge = XX & Breite = YY
2. Länge > XX || Breite < YY
3. (Länge <XX & Breite < YY) || (Länge >ZZ & Breite >AA)
4. ((Länge <XX & Breite < YY) || (Länge >ZZ & Breite >AA)) & Preis < BB

Die einzelnen Abfragen repräsentieren die Suche nach passenden Produkten innerhalb des gesamten SolConPro-Produktkataloges (siehe auch Kap. 2.6.1.2).

Tabelle 5: Übersicht ausgewählter Eigenschaften der modellierten Produkte

Eigenschaft	GebN_V1-2	GebN_V3-5	PVS_V1-3	CE_V1-3
Breite [mm]	900 – 2.000	1.080	974	1.500
Länge [mm]	900 – 2.000	1.090	1.524	3.065
Preis [€]	–	400 / 350 / 450	950 / 1.100 / 1.000	1.500 / 1.300 / 1.600

Tabelle 6: Übersicht der Ergebnisse ausgewählter Produktsuchen

Abfrage	GebN					PVS			CE		
	V1	V2	V3	V4	V5	V1	V2	V3	V1	V2	V3
1											
2											
3											
4											
5											

ProductViewer

Für eine nutzerfreundliche Darstellung verfügbarer Produkte und ihrer Eigenschaften steht der *ProductViewer* als eine eigenständige Webanwendung zur Verfügung. Diese ist unabhängig vom SolConPro-Cloud Frontend einsetzbar und greift auf die im Backend der SolConPro-Cloud zur Verfügung stehenden Dienste durch Aufrufen von RBV/OBW-Sets zu.

Abrufen einzelner Produkte

Zur Abfrage einzelner Produkte steht dem Benutzer über den *ProductViewer* eine Eingabemaske zur direkten Produktabfrage zur Verfügung (siehe Abbildung 41). Anzugeben sind hier die erforderlichen Inputs zur Abfrage des Produkts über die

Eingehende
Darstellung der
Ergebnisse

Dienste der SolConPro-Cloud. Diese beinhalten Graphname, Endpoint und die Sprache in der die Eigenschaften des Produkts zurückgegeben werden sollen. Im umgesetzten Anwendungsfall kann der Nutzer zwischen den Sprachen Englisch, Deutsch, Französisch, Spanisch und Polnisch wählen.

The screenshot shows the SolConPro web interface. At the top, there is a header with the SolConPro logo and links for 'Login' and 'Register'. Below the header is a form titled 'Request Product'. The form contains three input fields: 'Graphname:', 'Endpoint:', and 'Language:'. The 'Language:' field is a dropdown menu. Below the input fields is a purple button labeled 'Get Product'. At the bottom of the page, there is a footer with the text 'Copyright © 2015 - 2018 SolConPro. All rights reserved. Impressum.'

Abbildung 41: Benutzeroberfläche zur Abfrage eines einzelnen Produkts

Nach Eingabe der erforderlichen Angaben Sprache wird der Dienst auf der Cloud ausgeführt. Dabei werden zwei RBV/OBW-Sets ausgeführt: Das erste Set (*RequestSingleProduct*) bestehend aus dem Worker *SingleProductFromViKoLink* (siehe Abbildung 42) erhält eine □ JSON-Datei mit Graphname und Endpoint als Input und bezieht den Graphen zum gewünschten Produkt über die ViKoLink von der entsprechenden ViKoDB. Dieser wird im ttl-Format temporär abgelegt und eine Referenz als Übergabewert für das nächste Set ausgegeben.

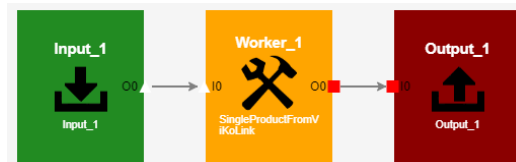


Abbildung 42: Set RequestSingleProduct

Im darauffolgenden RBV/OBW-Set *ProcessProduct* werden die abgerufenen Daten zur Ausgabe für den Benutzer aufgearbeitet. Dieses setzt sich aus dem Filterpaket *FP_SingleProduct*, dem Wrapper *ttl2json_singleProduct* und dem Worker *bsDDTranslator* zusammen (siehe Abbildung 43). Es erhält als ersten Input eine ttl-Datei mit dem betrachteten Produkt als Inhalt und als zweiten Input die vom Nutzer gewählte Sprache.

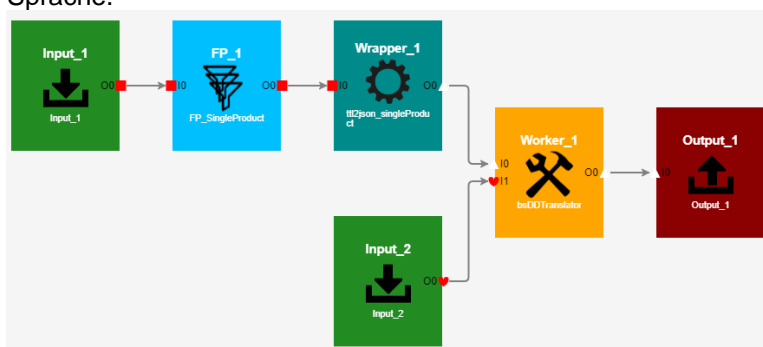


Abbildung 43: RBV/OBW-Set 'ProcessProduct'

Zunächst filtert das Filterpaket *FP_SingleProduct* die Informationen zum ausgewählten Produkt aus der Produktmenge heraus, die sich nicht auf Attribute mit konkreten Werten beziehen. Die *ttl*-Datei wird anschließend mithilfe des Wrappers *ttl2json_singleProduct* in ein JSON-Format überführt. Der Worker *bsDDTranslator* liest abschließend alle in der JSON-Datei hinterlegten Produktattribute und deren Einheiten aus und fügt jeweils basierend auf der für das Attribut bzw. die Einheit hinterlegten GUID des *buildingSMART data dictionary* ([□ bsDD](#)) eine Übersetzung in der vom Nutzer gewählten Sprache hinzu. Hierzu fragt der Worker die zur GUID verfügbaren Einträge aus dem bsDD ab. Er liest, falls vorhanden, die Bezeichnung in der entsprechenden Sprache aus dem bsDD aus und fügt diese den Produktinformationen hinzu. Da teils zu GUIDs keine Übersetzung in den dem Nutzer zur Auswahl gestellten Sprachen (Englisch, Deutsch, Französisch, Spanisch und Polnisch) verfügbar ist und zu einigen im Beispielprodukt vorhandenen Attributen keine Einträge im bsDD bestehen, wird in diesen Fällen auf ein temporäres Wörterbuch zurückgegriffen. Die in diesem Wörterbuch hinterlegten Begrifflichkeiten und Übersetzungen wurden im Rahmen des Projekts gemeinsam für die umgesetzten Anwendungsfälle erarbeitet. Die mit der Übersetzung angereicherte JSON-Datei der Produktbeschreibung wird als Output zurück an den *ProductViewer* gegeben. Zur graphischen Darstellung und nutzerfreundlichen Ausgabe werden die übermittelten Produktinformationen in der Benutzeroberfläche tabellarisch dargestellt (siehe Abbildung 44). Falls ein aus der GEOM-Ontologie generiertes 3D-Modell zu dem entsprechenden Produkt auf der Cloud hinterlegt wurde wird dies über den in der Benutzeroberfläche eingebetteten CMO-Viewer (2.4.1.1) im oberen Teil der Seite dargestellt.

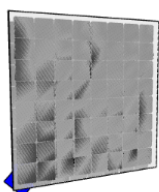
SolConPro
Request Product
Product Search
cloud.solconpro.de
solconpro.de

Product Details

Requesting product: **Completed** ✓
Processing data: **Completed** ✓

Building-integrated photovoltaics module
http://vikodb.solconpro.de/GebaeudeN/data/V3

http://rdf.bg/generated.ttl:E_Layer_5



Incidence angle modifier (IAM)	0.2	
short circuit voltage	5.86	ampere
Temperature coefficient Isc	0.06	percent per kelvin
product name	Gebäude N, statisch, V3	
length	1.09	metre
Temperature coefficient Pmpp	-0.32	percent per kelvin
efficiency	16.56472987	percent
weight	29.7	volt
tolerance of the nominal output power	5.0	plus/minus percent
pricing	400	euro
nominal current	5.43	ampere
Nominal voltage	36.1	volt
fill factor	77.325529	percent
Nominal power	195.0	watt
open circuit voltage	43.26	volt
Temperature coefficient Uoc	-0.26	percent per kelvin
thickness	0.01128	metre
manufacturer	Manufacturer B	
width	1.08	metre
maximum system voltage	1000.0	volt
rear embedding material	[http://vikodb.solconpro.de/GebaeudeN/data/V3#Co_RearEmbeddingMaterial]	
front embedding material	[http://vikodb.solconpro.de/GebaeudeN/data/V3#Co_FrontEmbeddingMaterial]	
PV cell layer	[http://vikodb.solconpro.de/GebaeudeN/data/V3#Co_PVCellLayer]	
photovoltaic cell	[http://vikodb.solconpro.de/GebaeudeN/data/V3#Co_PVCells]	

Copyright © 2015 - 2018 SolConPro. All rights reserved. [Impressum](#).

Abbildung 44: Darstellung eines Produkts im ProductViewer (englische Ausgabe)

Durchsuchen verfügbarer Produkte

Neben der Ausgabe einzelner Produkte ermöglicht die Webanwendung *Product-Viewer* es dem Benutzer die über die ViKoLink verfügbaren Produkte zu durchsuchen. Die Suche kann hierbei durch verschachtelte Kriterien festgesetzt werden. Dies umfasst bspw. die Einschränkung oder Vorgabe der Abmessungen des Produkts oder die Wahl bestimmter Hersteller (siehe Abbildung 45).

Abbildung 45: Benutzeroberfläche zur Produktsuche

Die Abfragen werden in der Webanwendung unter Verwendung des [Angular-QueryBuilder](#) so aufbereitet, dass sie dem Set *ProductProposal* zur Produktsuche auf der SolConPro-Cloud übergeben werden können. Dieses führt den Worker *MultipleProductsFromViKoLink* aus (siehe Abbildung 46). Dieser formt die erhaltenen Suchkriterien so um, dass sie als verteilte SPARQL-Abfrage an die ViKoLink zur Abfrage aller passender Produkte gesendet werden können. Bei der Suche werden sowohl statische als auch parametrische Produkte berücksichtigt (vgl. 2.5.3).



Abbildung 46: RBV/OBW-Set 'ProductProposal'

Die zurückerhaltenen Ergebnisse werden temporär abgelegt und die Referenz gemeinsam mit der vom Benutzer in der Oberfläche gewählten Sprache an das zweite auszuführende RBV/OBW-Set *ProcessSearchResults*, bestehend aus einem Filterpaket, einem Filter, einem Wrapper und einem Worker (siehe Abbildung 47) übergeben.

Die Filter werden in einer vordefinierten Reihenfolge ausgeführt, dass zunächst alle Informationen, die einen der folgenden Punkte betreffen entfernt werden: die Produkt-Geometrie, Komponenten-Verschaltung, Parametrik, nicht auf Produkt bezogene Attribute und Attribute, die für eine Standard Produktsuche nicht relevant sind. Anschließend werden die Produktkonstruktion und solche Attribute, die keine konkreten Werte besitzen herausgefiltert.

Analog zum Wrapper *ttl2json_singleProduct* überführt der hier eingesetzte Wrapper *ttl2json_multipleProducts* die im Graphen hinterlegten Produkte in ein [JSON](#)-Format, berücksichtigt hierbei jedoch, dass im Graphen mehrere Produkte abgebildet werden.

Der Worker *bsDDTranslator* fügt wie oben genannt die Übersetzungen in der vom Nutzergewählten Sprache als Eigenschaft der Produkte hinzu.

Eingehende
Darstellung der
Ergebnisse

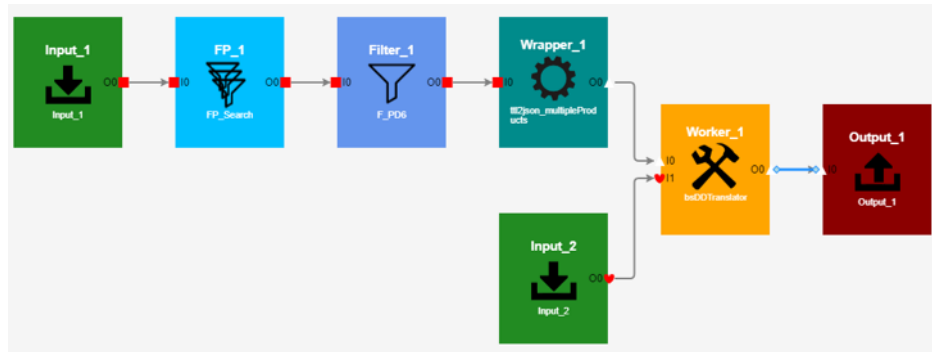


Abbildung 47: RBV/OBW-Set 'ProcessSearchResults'

Der *ProductViewer* erhält die entsprechenden Ergebnisse in aufbereiteter Form zurück und stellt die Produkte mit einem ausgewählten Umfang an Attributen aufgelistet für den Nutzer bereit. Dieser kann aus der Liste ein bestimmtes Produkt zur detaillierten Ansicht auswählen (siehe Abbildung 48). Um dieses darzustellen wird der zuvor beschriebene Dienst zum „Abrufen einzelner Produkte“ genutzt. Übergeben werden hier Graphname, Endpoint und Sprache des vom Nutzer aus der Liste gewählten Produkts.

Product Search

Choose product category:

Choose language:

Lengths in [mm]

AND OR

+ Condition + Constraint - Constraint

Länge > 1000

Breite < 1000

Search Products

Searching for products: **Completed** ✓
Processing data: **Completed** ✓

5 products found:

Produktname	Länge [Meter]	Breite [Meter]	Nennleistung [Watt]	Hersteller	Preisangabe [Euro]	
Gebäude N, parametrisch, Zelle 156	1.01	0.99	125.0	Manufacturer C	600	Choose Product
PV Shade V2	1.524	0.974	101.2	Manufacturer B	1100	Choose Product
PV Shade V3	1.524	0.974	101.2	Manufacturer A	1000	Choose Product
Gebäude N, parametrisch, Zelle 125	1.01	0.99	133.98	Manufacturer A	500	Choose Product
PV Shade V1	1.524	0.974	101.2	Manufacturer C	950	Choose Product

Copyright © 2015 - 2018 SolConPro. All rights reserved. [Impressum](#).

Abbildung 48: Product Viewer - Suchmaske mit Ergebnissen

Umsetzung:

Die Umsetzung des ProductViewer erfolgt analog zur SolConPro-Cloud in [□ Java-Script](#) in Form eines [□ MEAN-Stack](#). Die für den Anwendungsfall erforderlichen Executables zu den auf der Plattform erstellten und hinterlegten RBV/OBW-Sets werden in verschiedenen Sprachen implementiert. Tabelle 7 gibt eine Übersicht derselben.

Tabelle 7: Übersicht Umsetzung Executables

Executable	Komponente	Umsetzung
SingleProductFromViKoLink	Worker	Python
FP_SingleProduct	Filterpaket	Java, SPARQL
ttl2json_singleProduct	Wrapper	Java, SPARQL
bsDDTranslator	Worker	Python
MultipleProductsFromViKoLink	Worker	Python
FP_Search	Filterpaket	Java, SPARQL
F_PD6	Filter	Java, SPARQL
ttl2json_multipleProducts	Wrapper	Java, SPARQL

Mithilfe dieses Anwendungsfalls konnten mehrere Aspekte der Produktmodellierung und des verteilten Produktdatenkatalogs demonstriert werden. Durch die unterschiedlichen Modellierungsarten und einheitlichen und verschachtelten Produktsuchen konnte gezeigt werden, dass unter Verwendung der vorgeschlagenen mehrschichtigen Produkt-Ontologie mit wenigen Modellierungsrestriktionen gearbeitet werden kann. Auch die Evaluation parametrisch beschriebener Produkte konnte validiert werden. Durch die Anbindung des [□ bSDD](#) konnte zudem die mehrsprachliche Produktbeschreibung aufgezeigt werden. Die ViKoLink kann auf verteilte ViKoDBs zugreifen und diese durchsuchen und kann durch die offene [□ JSON](#)-basierte Schnittstelle einfach integriert werden, wie am Beispiel des ProductViewers zu erkennen ist. Durch die Evaluation parametrischer Produkte auf der SolConPro-Cloud und die Handhabung kompletter – und somit großer – Produktbeschreibungen hat sich jedoch ein Flaschenhals an der ViKoLink bzw. dem Job Executer Framework gebildet, der die Performanz der Anwendung reduziert. Auch die Visualisierung der Produktgeometrie ist nicht optimal gelöst, da der CMOViewer alle Modellierungsschritte und nicht nur das erwünschte Endergebnis abbildet. Dies basiert auf der veränderten Modellierungs-Herangehensweise bei Verwendung derselben Ontologie (GEOM) und Werkzeuge.

2.6.1.2 Einlesen geometrischer und semantischer Informationen in Revit

Bei der Nutzung von BIM in heutigen Planungsprozessen werden zahlreiche Softwarelösungen zur Erstellung, Verwaltung und Visualisierung der Modelle genutzt. Eine solche Softwareanwendung ist [□ Autodesk Revit](#), welches von Fachplanern, Architekten und Ingenieuren genutzt wird. In diesem Anwendungsfall soll der Prozess gezeigt werden, wie ein Anwender der Software mittels einer Erweiterung in Form eines Revit-Addins Produkte passend zu dem geplanten Fassadenelement über die SolConPro-Cloud suchen und in das Projekt laden kann. Hierfür soll der Nutzer eine Auswahl an Ergebnissen erhalten, welche betrachtet und durch den Fachplaner verifiziert werden können. Durch die Auswahl eines Produktes soll dies automatisch von der Produktdatenbank in das Revit-Projekt geladen werden, so dass der Anwender dies für die Planung nutzen, visualisieren und die für die Revit-Anwendung relevanten Produkteigenschaften betrachten kann (siehe Abbildung 49). Fachkenntnisse bezüglich der SolConPro-Cloud und den ablaufenden Prozessen müssen nicht vorliegen, der Anspruch ist eine einfache und selbsterklärende Bedienung.

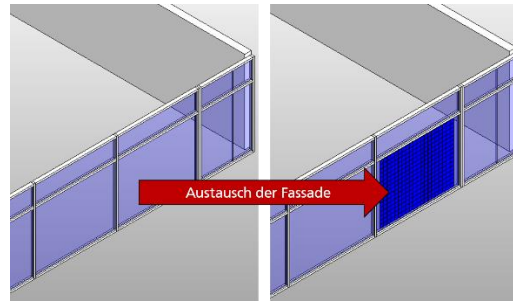


Abbildung 49: Austausch der Fassade durch die heruntergeladene Revit-Familie

Durch diesen Anwendungsfall wird gezeigt, dass eine einfache Anbindung an die Ontologie-basierte Produktdatenbanken über die SolConPro-Cloud ohne technische Kenntnisse für den Anwender möglich ist. Mit wenigen Schritten und geringen Nutzereingaben wird der Ablauf wie dargestellt praxisnah umgesetzt. Außerdem stehen die Produkte dem Anwender im *Revit*-Projekt als *Revit*-Familie zur Verfügung. Des Weiteren wird die Anbindung des Backend der SolConPro-Cloud (siehe Kap. 2.5.4.1) an eine externe Software implementiert und validiert.

Für die Umsetzung wurde eine C#-Anwendung implementiert, welche mittels der *Revit*-API ein *Revit*-Addin darstellt. Zusätzlich zur *Revit*-API [26] wurden die Bibliotheken *RestSharp* [27] und *Newtonsoft.Json* [28] in der Anwendung verwendet. Mithilfe der *Revit*-API können Anwendungen und Transaktionen innerhalb von *Revit* erzeugt und getätigt werden. *RestSharp* ist eine Softwarelösung, um über eine [REST-API](#) eine Verbindung zu einem Server – hier die SolConPro-Cloud – aufzubauen und Daten senden und empfangen zu können. Diese Daten werden in Form von [JSON](#)-Dateien über die SolConPro-Cloud gesendet. Deshalb wurde ebenfalls *Newtonsoft.Json* angebunden, welches JSON-Dateien in eine informationsverlustfreie, programmierfreundliche und objektorientierte Struktur überführt. Die Funktionalitäten und der Ablauf des *Revit*-Addins wird im Folgenden beschrieben.

Das *Revit*-Addin kann direkt in *Revit* über einen Tab in der Benutzeroberfläche gefunden und aktiviert werden (siehe Abbildung 50 und Abbildung 51). Es erscheinen zwei Schaltflächen, eine mit dem Titel „SolConPro-Website“, welche den Anwender direkt auf die SolConPro-Webseite (<http://www.solconpro.de/>) weiterleitet. Die zweite Schaltfläche mit dem Titel „ProductSearch“ dient dem Start der eigentlichen Anwendung. Um dem Anwender eine übersichtliche und einfache Nutzung zu ermöglichen, wurde eine eigene Nutzeroberfläche erstellt (siehe Abbildung 51), welche sich beim Betätigen der Schaltfläche öffnet. Die Oberfläche beinhaltet alle nötigen Abfragen des Nutzers, um eine Produkthanfrage an die SolConPro-Cloud zu stellen und anhand der Ergebnisse ein ausgewähltes Produkt in *Revit* zu importieren. Zu Beginn steht die Sprachauswahl, welche dafür sorgt, dass die Nutzeroberfläche in die jeweilige Sprache übersetzt wird, wie in Abbildung 51 im Vergleich der beiden Zustände der Oberfläche zu sehen ist. Hierfür wurden einzelne Label wie die Breite oder Länge mittels der Anbindung an das [bSDD](#) übersetzt und eingefügt. Dadurch wurde gezeigt, dass das bSDD zur Übersetzung der Nutzeroberfläche verwendet werden kann, solange die Begriffe im bSDD hinterlegt sind. Zusätzlich dient die Sprachauswahl als Suchparameter des Produktes in der jeweilig gewünschten Sprache. Die weiteren Angaben betreffen das konkret zu suchende Produkt. Der Anwender kann wählen welche Art von aktiven Fassadenelement gesucht werden und welchen geometrischen Ausmaßen dieses Produkt entsprechen soll. Hierfür wurde zusätzlich eine Funktion implementiert, die über die Schaltfläche mit der Aufschrift „Werte übernehmen“ ausgelöst wird. Die Funktion ermittelt anhand eines ausgewählten Fassadenelements wie in Abbildung 49 dargestellt im *Revit*-Projekt die geometrischen Maße aus und fügt diese den jeweiligen Textfeldern hinzu. Wenn kein Fassadenelement ausgewählt ist, wird der Anwender über diesen Fehler informiert. Diese automatischen Eingaben kann der Nutzer durch die Bearbeitung der

Textfelder individuell anpassen. Um den Suchradius der Parameter zu erweitern, können ebenfalls minimale und maximale Abweichungen übergeben werden.

Eingehende
Darstellung der
Ergebnisse

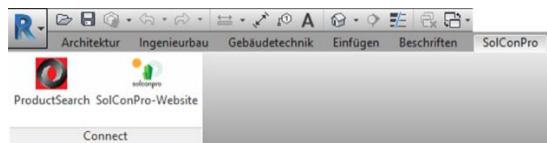


Abbildung 50: Revit-Addin in Tab-Ansicht von Autodesk Revit – Auszug aus der Revit-Oberfläche

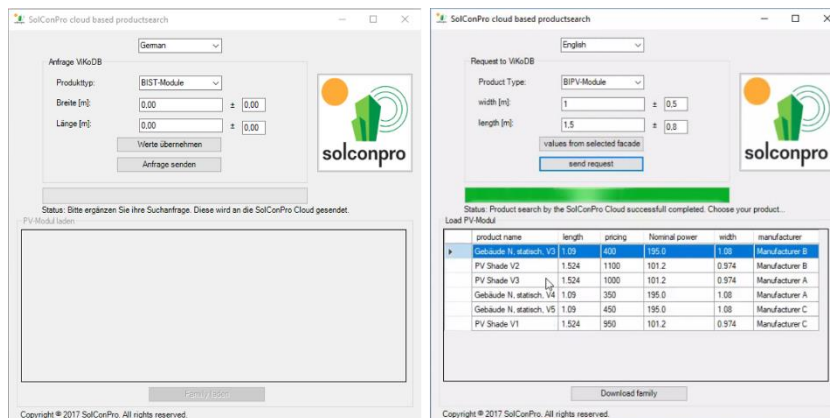


Abbildung 51: Nutzeroberfläche des SolConPro-Revit-Addins. Links zum Start, Rechts mit dem Ergebnis der Suchanfrage

Durch die Eingabe der Suchparameter, welche in einer Anwendung außerhalb der Demonstrator-Anwendung ergänzt und erweitert werden können, ist der erste Schritt des Addin-Ablaufs erledigt (siehe Abbildung 52). Nun kann der Anwender mit dem Betätigen der Schaltfläche „Anfrage senden“ die Suche starten. Hier geschieht im Hintergrund die Anbindung an die SolConPro-Cloud durch einen zentralen Login. Nach dem erfolgreichen Login werden die Suchparameter in Form einer JSON-Datei hochgeladen und mittels der [REST-API](#) der *Job Executer* ausgeführt, welcher das *RBV/OBW-Set* startet, das anhand der Suchparameter in der ViKoLink alle ViKoDBs auf passende Produkte durchsucht. Sobald die Ergebnisse auf der SolConPro-Cloud zur Verfügung stehen, werden diese über ein weiteres Set, namens *ProcessSearchResultsForRevit*, mittels dem bSDD in die gewünschte Sprache übersetzt und in eine JSON-Datei geparkt. Diese Datei wird automatisch heruntergeladen und verwendet, um auf der Nutzeroberfläche eine übersichtliche tabellarische Darstellung aller Produkte zu erzeugen. Hierbei werden alle Produkte und die vom Hersteller zur Verfügung gestellten Produktdetails angezeigt (siehe rechts in Abbildung 51). Es spielt dabei keine Rolle, ob die Hersteller die gleichen Angaben gemacht haben oder diese vollständig sind, das Addin führt alle vorhandenen Informationen auf. Nun liegt die Verantwortung erneut am Anwender sich für ein Produkt zu entscheiden und mit dem Betätigen der Schaltfläche („Familie laden“) den Download-Prozess zu starten. Für diesen wird im Hintergrund die Identifikationsnummer des Produkts auf die SolConPro-Cloud hochgeladen und das *RBV/OBW-Set RequestingSingleProduct* gestartet, welches über die ViKoLink das entsprechende Produkt aus der ViKoDB abfragt. Dieses wird über das *RBV/OBW-Set ProcessProductForRevit* gefiltert, in die Wunschsprache übersetzt, in eine JSON-Datei geschrieben und mit einer vorgefertigten *Revit-Familie* automatisch in das *Revit-Addin* geladen. Die *Revit-Familie* ist eine rechteckige Familie vom Typ Fassadenelement, hat aber außer den geometrischen Informationen nur wenige, semantische Informationen wie den Herstellernamen enthalten. Der Grund für eine Vorfertigung sind *Autodesk Revit* und seiner API geschuldet und wird im Fazit genauer erläutert. An-

Eingehende
Darstellung der
Ergebnisse

schließlich wird die *Revit*-Familie innerhalb des *Revit*-Addins mit den Informationen der Suchergebnisse überschrieben und verändert. Hierbei werden die Informationen aus der JSON-Datei mit den Eingaben der *Revit*-Familie verglichen und die Werte gegebenenfalls angepasst, sowie die Namen in die übersetzte Version verändert oder als neue Angaben hinzugefügt. Ebenfalls werden Formeln eingefügt und können über *Revit* ausgeführt werden. Allerdings liegen auch hier durch *Autodesk Revit* Einschränkungen vor, wenn die Formeln mit der Geometrie zusammenhängen sollen. Abschließend wird die erstellte *Revit*-Familie als Fassadenelement dem aktuellen Projekt hinzugefügt und kann vom Anwender, wie in Abbildung 49 gezeigt, in die Fassade integriert werden. Die Produktinformationen des Elements können über die Familienparameter direkt in *Revit* angezeigt und verwendet werden (siehe Abbildung 53).

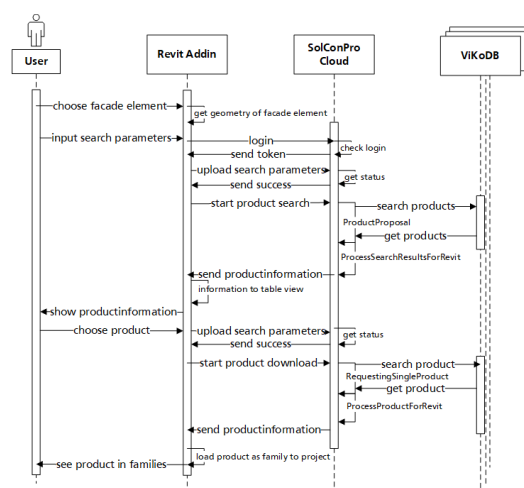


Abbildung 52: Sequenzdiagramm Produktsuche des Revit-Addins über die SolCon-Pro-Cloud

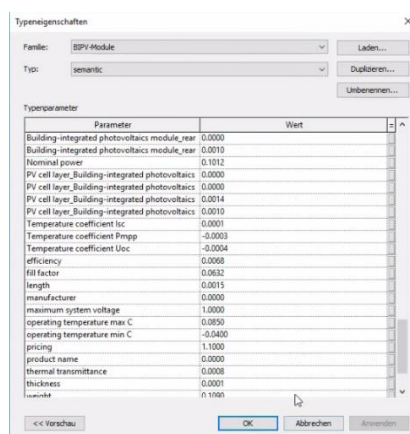


Abbildung 53: Produktinformationen angezeigt in Autodesk Revit für beispielhaft das BIPV-Modul


Für die in AP 5 entwickelten Produktbeschreibungen wurde exemplarisch anhand *Autodesk Revit* untersucht, inwiefern sich eine Anbindung der extern über die ViKoDB verwalteten Produktdatenmengen realisieren lässt. Innerhalb der *Revit*-API wurde eine Applikation umgesetzt, die es erlaubt, mittels *RBV/OBW*-Sets gefilterte semantische und teilweise geometrische Objektinformationen in das Softwareprogramm zu laden. Die Hauptschwierigkeit besteht hierbei in den inhärenten Be-

schränkungen der API, die deshalb im Detail untersucht wurden. Trotz gemeinschaftlicher Anstrengungen von TU Darmstadt und Ed. Züblin AG konnten folgende Schwierigkeiten nicht behoben werden:

- 1) Die API von *Autodesk Revit* lässt keine programmiergestützte Erstellung von parametrisierten und verschachtelten *Revit*-Familien zu, weshalb jedes Mal vor dem ersten Einlesen einer externen Beschreibung eine Vorfertigung derselben stattfinden muss.
- 2) Weitere Einschränkungen beziehen sich auf die Parametrik (gleichungsbaasierte Geometriedefinition). Externe Änderungen zu parametrischen Zusammenhängen können nicht nach *Autodesk Revit* importiert werden. Dies gilt nur für Änderungen der Parameter bei bereits bestehenden und intern und extern synchron vorhandenen parametrischen Zusammenhängen.
- 3) Bei verschachtelten *Revit*-Familien – *Revit*-Familien, die aus anderen *Revit*-Familien bestehen (z.B. Photovoltaik-Anlage aus vielen Zellelementen) – gibt es weitere Einschränkungen der *Revit*-API, die einen Import verkomplizieren, weshalb vorgefertigte Familien verwendet und individuell modifiziert wurden.

Dennoch konnte in diesem Anwendungsfall die Funktionalität und Stärke der SolConPro-Cloud gezeigt werden. Als Ergebnis liegt nun eine Demonstrator-Anwendung vor, die über die automatische Anbindung an die SolConPro-Cloud Produktabfragen direkt in *Revit* integriert, die Ergebnisse für *Revit* aufbereitet und als *Revit*-Familie mit allen Produktdaten einfügt und für den weiteren Planungsprozess genutzt werden können.

2.6.1.3 RBV & OBW zur Bestrahlungsstärkeberechnung auf Gebäudehüllen

Die am Fraunhofer ISE entwickelte Methodik zur Ertrags- und Auslegungsberechnung von BIPV-Systemen¹ kann als zentrales Planungstool für BIPV-Systeme genutzt werden. Dazu müssen Produkt- und Projektdaten durch geeignete Filter (RBV) und Parser (OBW) so verarbeitet werden, dass automatisch der benötigte Input für die Simulationsrechnung generiert wird und diese mit möglichst geringem zusätzlichem Aufwand automatisiert durchgeführt werden können. Ein Schritt zur Erreichung dieses Ziels ist die Bereitstellung der für die Bestrahlungsstärkeberechnung auf der Gebäudehülle erforderlichen Informationen. Dies soll in diesem Anwendungsfall anhand von zwei realen Bauprojekten demonstriert werden: Z3-Gebäude der Ed. Züblin AG (bereits fertiggestellt inkl. BIPV) und ZHS des Fraunhofer ISE (Zentrum für höchsteffiziente Solarzellen, in Planung inkl. BIPV). Für beide reale Bauprojekte wird als Input eine  IFC-Datei herangezogen. Trotz der im Rahmen von SolConPro festgestellten Unzulänglichkeiten dieses Formats und der entsprechenden Exportfunktionen aus proprietären Softwareprogrammen kommt es hier zum Einsatz, da es zumindest in begrenztem Umfang eine Übergabe von geometrischen Daten aus 3D-Programmen ermöglicht. In Zukunft können an dieser Stelle auch Semantic-Web-basierte Datenformate zum Einsatz kommen.

Die Funktionalitäten RBV+OBW sind in diesem Anwendungsfall skriptbasiert im IFC-Parsing-Tool (IFC2rad²) enthalten. Dieses erlaubt 1.) die Filterung nicht benötig-

¹ Eine ausführliche Beschreibung der Methodik findet sich im fünften Zwischenbericht des vorliegenden Forschungsprojekts.

² Das IFC-Parsing-Tool wurde ausführlich im dritten Zwischenbericht beschrieben.

2.6.1.4 RBV & OBW für BIPV-Simulationen (Kombination von Produkt- und Projektebene)

Dieser Anwendungsfall baut auf den vorigen Anwendungsfall auf. Nach der Bestrahlungsstärkeberechnung kann ein Planer weitere Berechnungen und Simulationen durchführen, um das gesamte BIPV-System zu evaluieren. Hierfür wurde eine Übersicht der notwendigen Berechnungs- und Simulationsschritte erstellt. Anhand dieser Übersicht wurden Stellen identifiziert, in denen Input-Dateien benötigt werden, die komplett oder zum Teil auf Produktdaten basieren. Da die SolConPro-Cloud ausschließlich der Datenaufbereitung und -bereitstellung dient, wird in diesem Anwendungsfall davon ausgegangen, dass Planer die notwendigen Produktdaten über die SolConPro-Cloud beziehen und auf Basis derselben die Berechnungen und Simulationen eigenständig durchführen. Allerdings wurde das benötigte Format der Simulationsschritte berücksichtigt, sodass die für die Schritte aufbereiteten Daten ohne großen Aufwand in diesen verwendet werden können.

Die Aufbereitung findet für insgesamt vier Schritte statt: Die *ds2dm*-Berechnung (*ds2dm* = *data sheet to diode model*, ein Berechnungsverfahren für die Ermittlung von elektrischen Kennlinien aus Datenblattangaben), Stromkreisberechnung (inkl. der Modul-internen Verschaltung und Wechselrichter-Daten), sowie die *Schmidt-sauer*- und *ACSIM*-Berechnung. Für diese Schritte werden Produktdaten von sowohl BIPV-Modulen als auch Wechselrichtern aufbereitet. Zur Aufbereitung der Daten werden fünf *RBV/OBW*-Sets zur Verfügung gestellt, deren Zusammensetzung in Anhang B beschrieben ist.

Der Aufbau der *RBV/OBW*-Sets ähnelt sich mit Ausnahme des *Verschaltung RBV/OBW*-Sets (siehe Abbildung 54 am Beispiel *ACSIMCalculation*): Sie erfordern eine Input-Datei in TTL-Format (ungefilterte Produktdaten), anschließend werden die Produktdaten über das *Filter-Package* *FP_ProductAttributeProcessing* die Attribute des Produktes herausgefiltert. D. h., dass Geometrie, Verschaltung, Parametrik und nicht Produkt-bezogene Attribute (z. B. solche, die an Komponenten des Produktes hängen) entfernt werden. Nach dieser Reduktion wird die Konstruktion des Produktes entfernt, sodass lediglich Produkt-Attribute und ihre Einheiten übrigbleiben. Diese Attribute werden daraufhin für jeden Simulationsschritt individuell gefiltert, sodass nur für den Schritt relevante Attribute weitergegeben werden. Aus diesen werden zudem nicht-konkrete Attribute – also solche, die einen Wertebereich, 2D-Listen oder □ *WKT* statt einem konkreten Attributwert besitzen – herausgefiltert. Die hieraus resultierenden Attribute werden mit einem *OBW* in eine TXT-Datei übersetzt, deren Schema an das der geforderten Input-Dateien angelehnt ist. Für eine universelle und einfache Anwendung muss eine Sprache angegeben werden, in die die Attributbezeichnungen und Einheiten mittels eines *Workers* übersetzt werden. Die übersetzte TXT-Datei wird abschließend zum Download zur Verfügung gestellt.

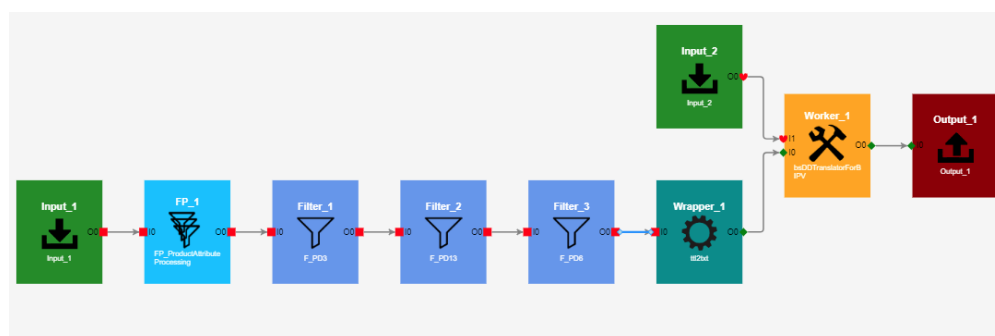


Abbildung 54: Exemplarisches Layout des *RBV/OBW*-Sets *ACSIMCalculation*

Sonderfälle in diesem Ablauf sind einerseits die Ermittlung der BIPV-Modulkennwerte für die *ds2dm*-Berechnung, da diese sowohl über konkrete Attribute als auch über die Abbildung der IV-Kurve (als Objekte oder WKT) beschrieben werden, und andererseits die Abbildung der Modul-internen Verschaltung. Diese Fälle werden folgend detaillierter erläutert.

Für die Filterung von Attributen, die für die *ds2dm*-Berechnung relevant sind, wurde ein komplexer Filter erzeugt, der im ersten Schritt evaluiert, in welcher Form die Kennwerte beschrieben sind. Sollten sie als konkrete Attribute vorliegen, filtert er diese heraus. Anderenfalls wird zwischen Objekten der *List2D*-Klasse und WKT unterschieden. In beiden Fällen werden die Daten in Objekte der Programmsprache übersetzt und anschließend analysiert. Resultate der Analyse sind die gesuchten Kennwerte, die an charakteristischen Stellen der IV-Kurve abzulesen sind.




Um die Verschaltung des Moduls zu extrahieren, werden zunächst Attribute, Geometrie und Parametrik entfernt. Anschließend liest ein Algorithmus die Verschaltung der verbauten Komponenten mittels der *ComponentConnection*-Objekte aus. Der Algorithmus kann sowohl die Verschaltung von *SingleEntity*- als auch *DynamicEntity*-Objekten interpretieren. Hierfür wurde die Annahme getroffen, dass die Verschaltung der in *DynamicEntity*-Objekten enthaltenen Komponenten für jede Wiederholung der Komponenten identisch ist. Weiterhin wurde angenommen, dass die Wiederholungen in Reihe zueinander platziert werden. Als Resultat dieses RBV/OBW-Sets wird eine TXT-Datei in einem an die geforderte Input-Datei angelehnten Schema ausgegeben.

Dieser Anwendungsfall zeigt, dass die Kompetenz und Verantwortung beim zuständigen Planer liegt, aber diesem die Aufbereitung, Verarbeitung und Bereitstellung der nötigen Daten durch die Cloud ermöglicht wird.

2.6.1.5 Neue digitale Methoden für die technische Gebäudeausrüstung

Eine durchgängige und vollständige digitale Dokumentation der technischen Gebäudeausrüstung (TGA) erleichtert die Zusammenarbeit verschiedener Gewerke in der Planungsphase. Darüber hinaus können die über die TGA abgelegten Informationen bei der Inbetriebnahme und während des Gebäudebetriebs verwendet werden, um die in der Planung vorgesehenen Steuer- und Regelungsmechanismen zu überprüfen und ggf. Abweichungen zu erkennen. Dazu werden Sensoren an relevanten Stellen in der Anlage platziert und ausgelesen.

Im hier beschriebenen Anwendungsfall wird eine Methodik vorgestellt, die es ermöglicht, die oben genannten Arbeitsschritte zu automatisieren.

Zunächst werden Informationen über die TGA, insbesondere die Regelung betreffend, in einem standardisierten Austauschformat (IFC4) hinterlegt. Die Anreicherung der aus  *Autodesk Revit* stammenden  IFC-Dateien wird mittels der  *Python* Bibliothek IFC-openshell vorgenommen.

Im nächsten Schritt wird die erweiterte IFC-Datei in RDF-Triple umgewandelt (IFC-to-RDF Converter von Pieter Pauwels und Jyrki Oraskari), um semantische Webtechnologie verwenden zu können und insbesondere sog. SPARQL-Queries durchführen zu können. Für die Abfragen (Queries) wird wiederum eine Python Bibliothek (rdflib) genutzt. Die Abfragen dienen zur Plausibilitäts- und Qualitätskontrolle der hinterlegten Informationen über die TGA.

Der gesamte Ablauf ist in Abbildung 55 schematisch dargestellt.

Eingehende
Darstellung der
Ergebnisse

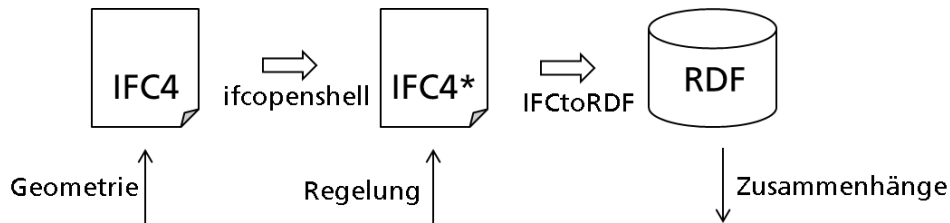


Abbildung 55: Schematische Darstellung der Anreicherung und Extraktion von TGA-Informationen.

Im Folgenden wird der obengenannte Ablauf an Hand eines einfachen Use Cases illustriert. Als zugrundeliegende IFC-Datei wird ein in *Autodesk Revit* modelliertes und als IFC exportiertes TGA-Modell genutzt, welches einen Kessel, einen Speicher, eine Lüftungsanlage und eine Fußbodenheizung, sowie Rohrleitungen, Luftkanäle, Ventile, Klappen und Pumpen enthält (siehe Abbildung 56).

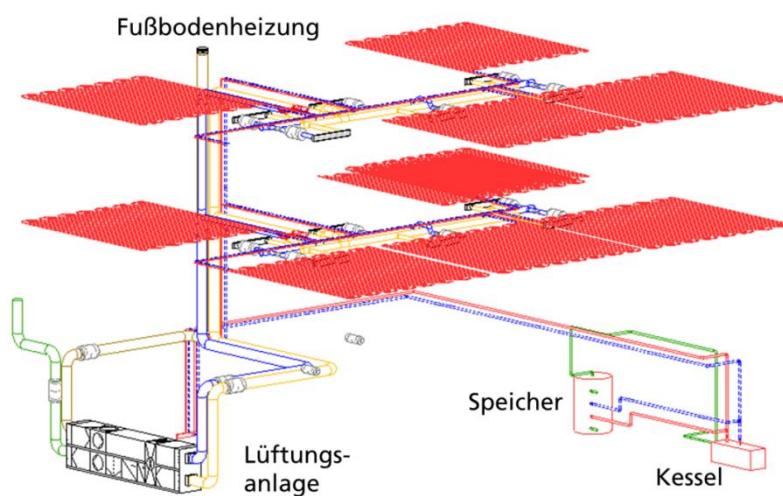


Abbildung 56: Verwendetes TGA-Modell in Revit.

Im nächsten Schritt wird die IFC-Datei mit Informationen über die Regelung angereichert. Hierfür werden folgende einfache Standard-Regelungsmechanismen betrachtet:

1. Heizkurve: Außentemperaturabhängige Vorlauftemperatur als Tabelle
2. Zeitplan: Soll-Volumenstrom abhängig von Tageszeit
3. PID-Regelung: Ventil-Stellsignal abhängig von aktueller Temperatur

Um eine außentemperaturabhängige Vorlauftemperatur in der IFC-Datei zu hinterlegen wird ein neues PropertySet definiert, welches allen Systemen mit dem Namen „H VL“ angeheftet wird. Innerhalb des PropertySets gibt es eine Tabelle (PropertyTableValue) angelegt, welche die Abhängigkeit zwischen Vorlauf- und Außenlufttemperatur beschreibt. Eine schematische Darstellung dieser Vorgehensweise findet sich in Abbildung 57.

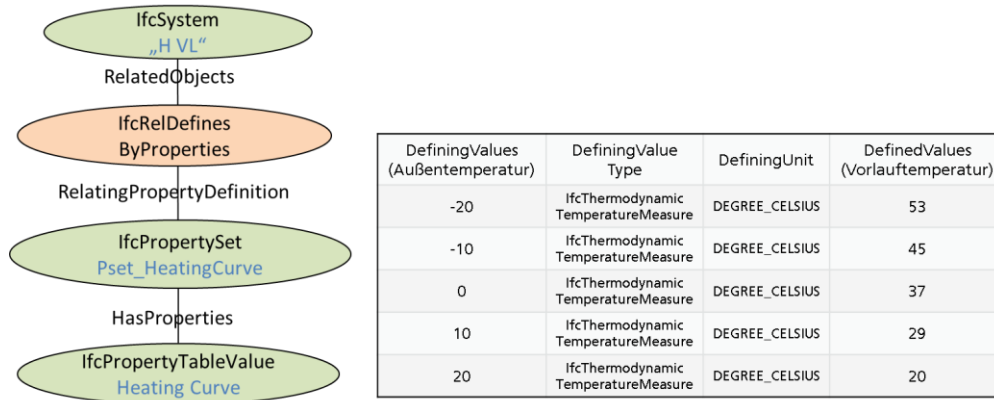


Abbildung 57: Schematische Darstellung der Beschreibung einer Heizkurve in IFC.

Um eine Zeitplan-abhängige Absenkung des Soll-Volumenstrom in der IFC-Datei zu hinterlegen, wird zunächst in ein bestehendes PropertySet des Volumenstromreglers eine zusätzliche Eigenschaft integriert, welche den ursprünglichen Sollwert als Maximalwert und den Wert Null, für die Absenkung, als Minimalwert enthält. Dann wird mit dieser neuen Eigenschaft eine Bedingung verknüpft (PropertyDependency-Relationship). Die Bedingung bezieht sich auf einen Zeitplan (RecurrencePattern) und ist in Textform als „IF-THEN“-Ausdruck hinterlegt. Bei der Formulierung wurde hierbei auf Konformität mit VDI 3805 geachtet. Der Zeitplan wiederum enthält die Art der Wiederholung (wöchentlich), die zutreffenden Wochentage (Montag bis Freitag) und die zutreffenden Zeiten (6 Uhr bis 20 Uhr). Die beschriebene Herangehensweise ist schematisch in Abbildung 58 dargestellt.

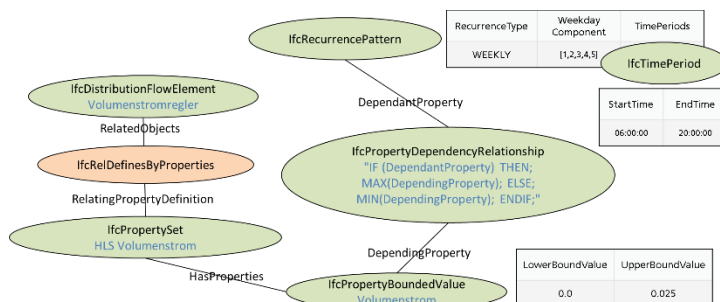


Abbildung 58: Schematische Darstellung der Beschreibung eines Zeitplans in IFC.

Um eine Solltemperatur-Regelung in der IFC-Datei zu hinterlegen wird zunächst ein Regler hinzugefügt (IFCController). Der Regler wird mit einem vordefinierten PropertySet für PID-Regler verknüpft. Die im PropertySet definierten Parameter können geeignet gewählt werden (siehe Abbildung 59).

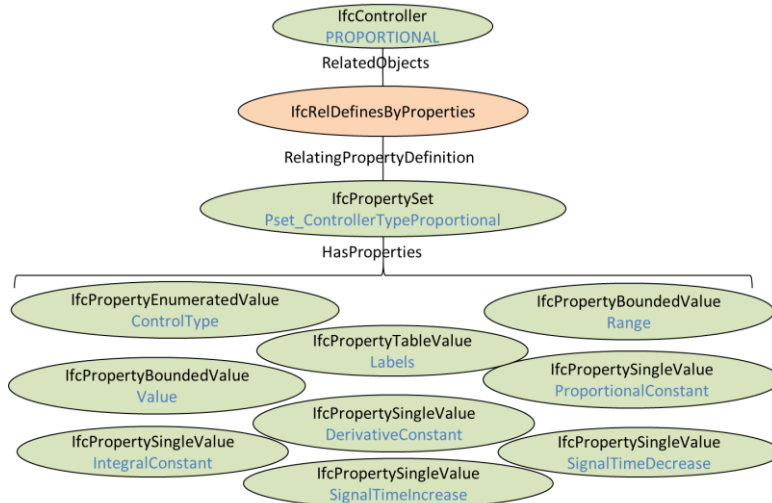


Abbildung 59: Schematische Darstellung der Beschreibung der Reglerparameter in IFC.

Im folgenden Schritt werden ein Sensor (IFCSensor) und ein Aktor (IFCActuator) hinzugefügt. Die logische Verknüpfung von Sensor, Aktor und Controller erfolgt mit Hilfe von sog. „Ports“ (IFCDistributionPort). Insgesamt werden vier Ports angelegt, zwei vom Typ „source“ und zwei vom Typ „sink“. Dem Controller werden sowohl ein Port vom Typ source als auch ein Port vom Typ sink zugeordnet. Dem Sensor wird ein Port vom Typ source und dem Aktor ein Port vom Typ sink zugeordnet. Dann werden die Ports entsprechend verbunden. Schließlich wird dem Sensor ein Rohrsegment zugeordnet, in welchem die aktuelle Wassertemperatur gemessen werden soll und der Aktor wird mit dem zugehörigen Stellventil verknüpft. Die gesamte Beschreibung des Regelungsmechanismus ist Abbildung 60 zu entnehmen.

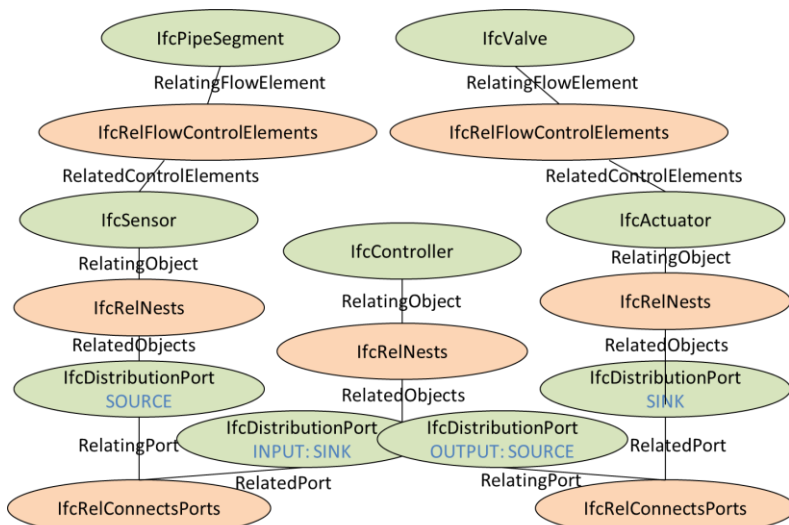


Abbildung 60: Schematische Darstellung der Beschreibung des Regelungsmechanismus in IFC.

Nachdem die Erweiterung um Regelungsmechanismen erfolgt ist, wird die IFC-Datei in ein RDF-Format umgewandelt. Mittels Queries können nun relevante Informationen über die Anlage extrahiert werden.

Beispielsweise kann eine Ausgabe aller oder bestimmter TGA-Komponenten, Sensoren, Regler, etc. mit Eigenschaften erfolgen. In Abbildung 61 wird beispielhaft die Abfrage derjenigen Systeme dargestellt, welche dem Vorlauf eines Heizkreises zugeordnet sind.

Die folgenden Screenshots geben Queries und Ausgaben unter Nutzung des Programms Ontotext Graph DB wieder.

```

1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX ifcowl: <http://www.buildingsmart-tech.org/ifcowl/IFC4_ADD1#>
3 PREFIX express: <https://w3id.org/express#>
4 PREFIX list: <https://w3id.org/list#>
5
6 SELECT ?sys ?sys_name ?prop ?prop_name
7 WHERE {
8   ?sys rdf:type ifcowl:IfcSystem.
9   ?sys ifcowl:name_ifcowl/express:hasString ?sys_name .
10  FILTER(REGEX(str(?sys_name), 'H VL')).
11  ?x ?y ?sys .
12  ?x ?z ?prop .
13  FILTER(REGEX(str(?prop), 'PropertySet')).
14  ?prop ifcowl:name_ifcowl/express:hasString ?prop_name .
15 }
16
17

```

	sys	sys_name	prop	prop_name
1	inst:ifcSystem_773573	H VL 2	inst:ifcPropertySet_774971	HLS-Volumenstrom
2	inst:ifcSystem_773573	H VL 2	inst:ifcPropertySet_774963	HLS
3	inst:ifcSystem_773573	H VL 2	inst:ifcPropertySet_774977	Sonstige
4	inst:ifcSystem_773573	H VL 2	inst:ifcPropertySet_776296	Pset_HeatingCurve
5	inst:ifcSystem_773613	H VL 4	inst:ifcPropertySet_775082	HLS-Volumenstrom
6	inst:ifcSystem_773613	H VL 4	inst:ifcPropertySet_775073	HLS
7	inst:ifcSystem_773613	H VL 4	inst:ifcPropertySet_775088	Sonstige
8	inst:ifcSystem_773613	H VL 4	inst:ifcPropertySet_776296	Pset_HeatingCurve
9	inst:ifcSystem_773635	H VL 5	inst:ifcPropertySet_775096	HLS
10	inst:ifcSystem_773635	H VL 5	inst:ifcPropertySet_775110	Sonstige
11	inst:ifcSystem_773635	H VL 5	inst:ifcPropertySet_775104	HLS-Volumenstrom
12	inst:ifcSystem_773635	H VL 5	inst:ifcPropertySet_776296	Pset_HeatingCurve
13	inst:ifcSystem_773711	H VL 1	inst:ifcPropertySet_775140	HLS
14	inst:ifcSystem_773711	H VL 1	inst:ifcPropertySet_775148	HLS-Volumenstrom
15	inst:ifcSystem_773711	H VL 1	inst:ifcPropertySet_775154	Sonstige

Abbildung 61: SPARQL-Query zur Abfrage der zum Heizkreis (VL) gehörigen Systeme.

Weiterhin kann die hinterlegte Heizkurve mittels Queries extrahiert werden. Die entsprechende Abfrage mit Ausgabe ist in Abbildung 62 dargestellt. Außerdem ist eine graphische Darstellung der Heizkurve gezeigt, welche aus den Query-Ergebnissen in `Python` automatisiert erzeugt wird. Diese kann unmittelbar zur Plausibilisierung oder zum Vergleich mit Messdaten verwendet werden.

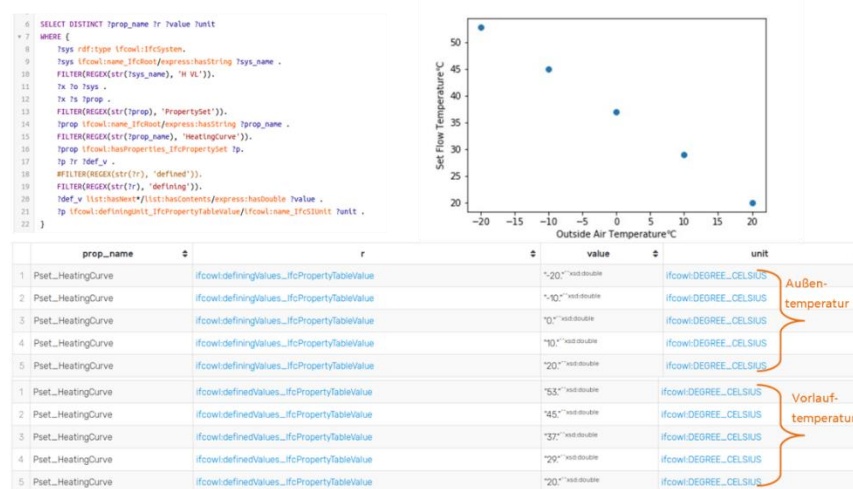


Abbildung 62: SPARQL-Query zur Extraktion der Heizkurve und (nachprozessierte) Ausgabe.

Für die Extraktion des Zeitplan-abhängigen Volumenstroms werden mehrere SPARQL-Queries hintereinander durchgeführt, um die Übersichtlichkeit zu erhöhen. Die entsprechenden Abfragen und die zugehörigen Ausgaben sind in Abbildung 63 dargestellt. Zusätzlich ist der extrahierte Zeitplan visualisiert. Dies geschieht in einer Python-Routine, welche die semantischen Informationen über den Zeitplan in einer automatisiert in eine graphische Darstellung umwandelt.

Eingehende Darstellung der Ergebnisse

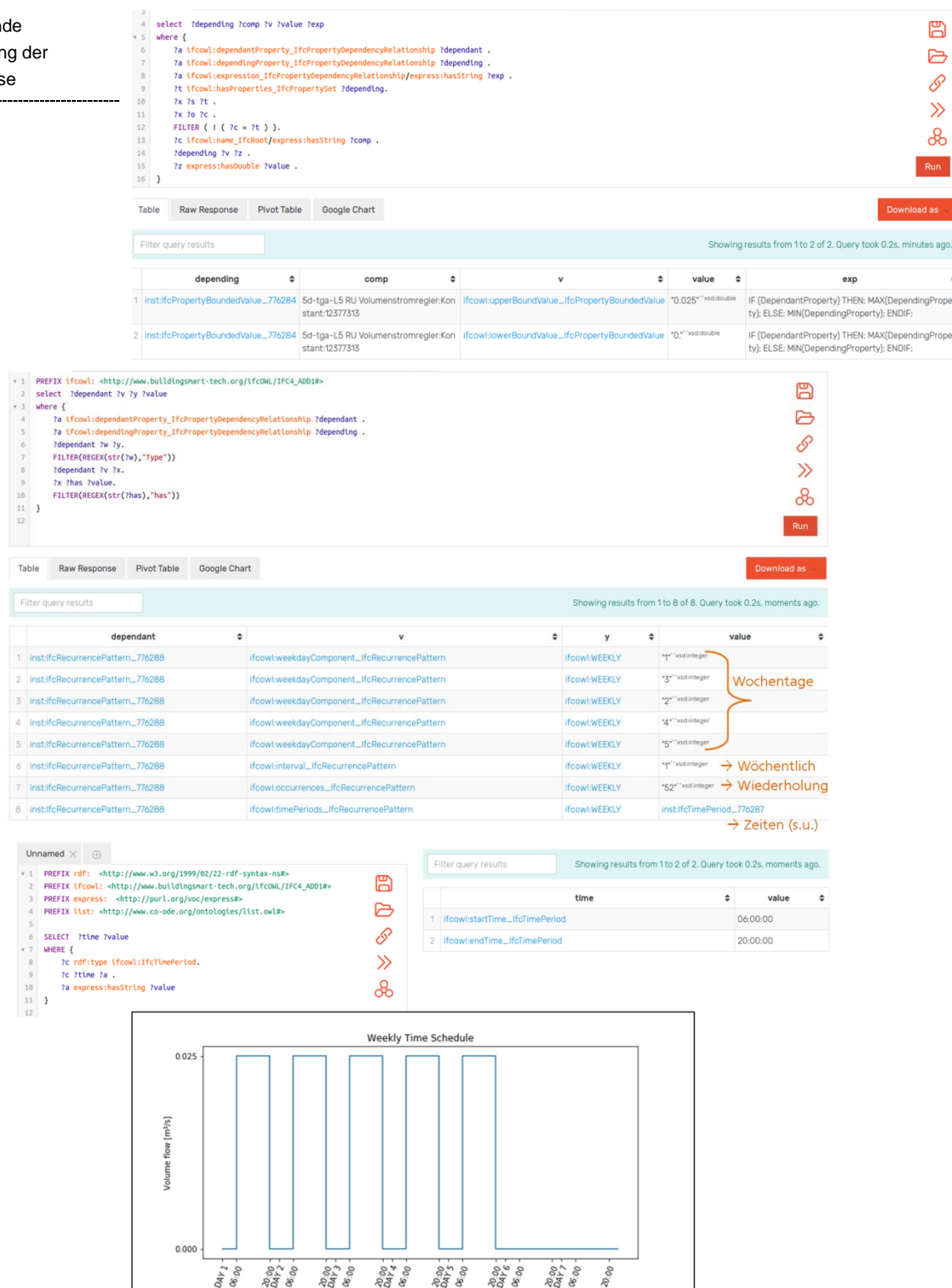


Abbildung 63: SPARQL-Queries zur Extraktion der Abhängigkeit des Volumenstroms und des entsprechenden Zeitplans und (nachprozessierte) Ausgabe.

Schließlich wird die Reglerparametrierung mittels SPARQL-Query extrahiert (siehe Abbildung 64) und der Regelungsmechanismus geprüft (siehe Abbildung 65).



Eingehende
Darstellung der
Ergebnisse

Die Kombination aus software-übergreifendem Kollisionsmanagement und der Einlesbarkeit semantischer und geometrischer Bauproduktinformationen in **Autodesk Revit** erlaubt eine präzise und effektive Arbeitsweise in der Fassadenplanung.

Unterschiedliche Anforderungen an die geometrische Fassadenstruktur können immer wieder Kollisionen verursachen, deren Lösung unter Umständen den Einsatz ursprünglich vorgesehener solaraktiver Fassadenelemente unmöglich macht. Bisher aber gab es keine Möglichkeit eines unmittelbaren Informationsaustauschs zwischen den beteiligten Fassadenplanern und den Herstellern solaraktiver Komponenten. Mit den im Forschungsprojekt entwickelten Konzepten lässt sich der Planungsprozess erheblich vereinfachen.

Für das Kollisionsmanagement wurden für *Autodesk Revit* und *Navisworks* zwei Frontend-Applikationen umgesetzt, die über programmspezifische Funktionen verfügen. Es können beispielsweise direkt die an einer Kollision beteiligten Objekte im Gebäudemodell markiert werden (siehe Abbildung 66).

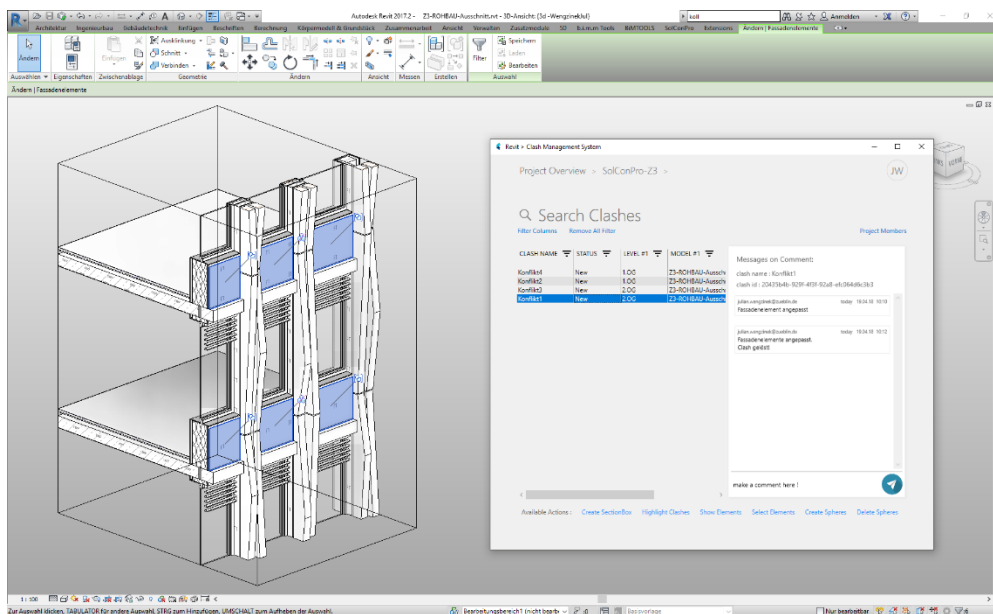


Abbildung 66: Das Clash-Management-System als Plug-In für Autodesk Revit. Für das angelegte Projekt SolConPro-Z3 werden vier Konflikte angezeigt. Das Plug-In erlaubt die direkte Verbindung der Kollisionsdaten mit dem Gebäudemodell über die Funktionen „Create SectionBox“, „Highlight Clashes“, „Show Elements“, „Select Elements“, „Create Spheres“ und „Delete Spheres“

Auch Kollisionen, die durch externe Modellbearbeitung in anderen Softwareprogrammen entstanden sind, lassen sich so transparent in *Autodesk Revit* beheben. Sollten solaraktive Fassadenkomponenten von den Kollisionen betroffen sein, so lassen sich diese über die in Kap. 2.6.1.2 beschriebene Herangehensweise aktualisieren bzw. austauschen. Der beschriebene Ablauf erlaubt eine zeitnahe Bearbeitung von Folgewirkungen, die sich aus Gestaltungsänderungen beliebiger Baubeteiligter an der Fassade ergeben (siehe Abbildung 67).

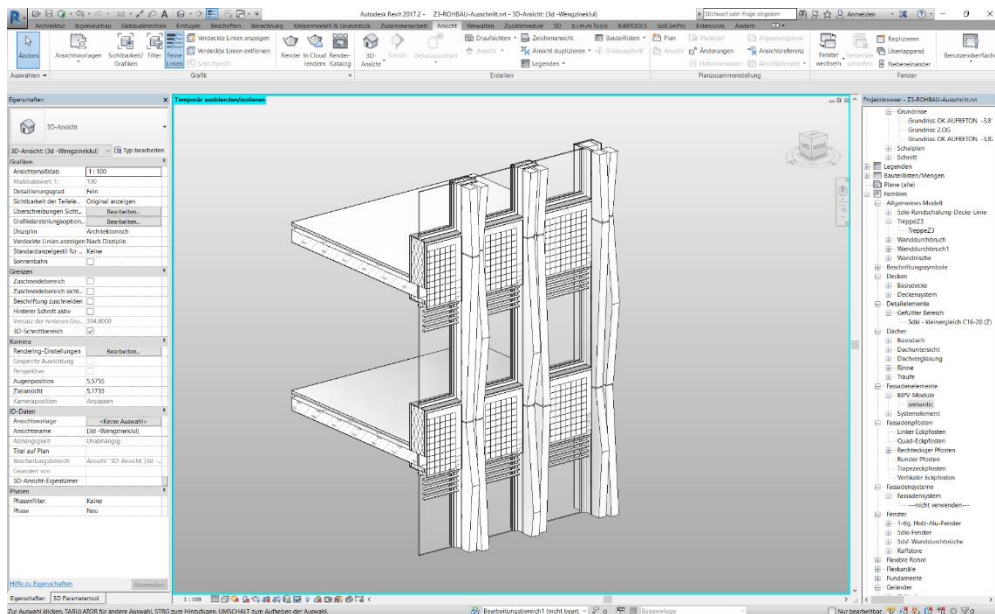


Abbildung 67: Einfügen solaraktiver Fassadenkomponenten nach Beseitigung der auftretenden Kollisionen mittels Clash-Management-System

2.6.2.2 Datensammlung anhand von Webformularen und Dashboard-Visualisierung

Mit den in Kap. 2.4.2.4 beschriebenen Möglichkeiten der Bereitstellung von Webservern und Datenbanken muss eine Baufirma mit der Digitalisierung nicht auf Software-Unternehmen warten. Einige, nicht unerhebliche Datenmengen können bereits mit den derzeit verfügbaren Werkzeugen gesammelt und ausgewertet werden, wodurch die Kommunikation zwischen den Baubeteiligten enorm gefördert werden kann, ohne dass dafür Gebäudemodelle erforderlich wären.

In Kooperation mit dem Bereich GPM (Geschäftsprozessmanagement) der Ed. Züblin AG wurden die in Kap. 2.4.1.2 beschriebenen Werkzeuge herangezogen, um Datenmengen, die auf Baustellen mittels Webformularen gewonnen wurden, auswertbar zu gestalten und zu visualisieren. Zur schnellen Umsetzung der Webformulare wurde die kommerzielle Software *MoreApp* [29] verwendet. Diese erlaubt die schnelle Erstellung von Apps für Smartphones, die sich zur Datenaufnahme beliebiger Art eignen. Die gesammelten Daten werden proprietär in der Cloud gespeichert, sind jedoch von außen via [REST-API](#) abgreifbar.

Analog zur Herangehensweise in Kap. 2.4.2.4 wurden mittels [Gitlab](#) Datenbanken zur Verfügung gestellt, zudem ein Webserver implementiert, der über die angebotene REST-API eine Synchronisierung der bestehenden Datenmengen durchführt. Die Inhalte der Datenbanken wurden anschließend mittels der Open-Source-Applikation *Grafana* [30] visualisiert und deren Ergebnisse via GET-Request der Adresse more-app-gpm.ds.strabag.com zur Verfügung gestellt. Da es sich teilweise um vertrauliche Informationen handelt, wurde ein externer Zugriff zunächst unterbunden. Dennoch lässt sich die entwickelte und getestete Architektur für beliebige formularbasierte Datenerhebungen nutzen.

Für den genannten Anwendungsfall wird in der Datei `docker-swarm.yml` mit den beiden Datenbanken *postgreSQL* [31] und *InfluxDB* [32] bzw. der Applikation *Grafana* ein *Docker-Network* erstellt, damit eine *tcp*-Verbindung zwischen den *Docker-Containern* durchgeführt werden kann.

In allen drei Applikationen werden bestimmte Laufwerke lokal gemountet, damit bei einem *Container*-Neustart diese Daten nicht verlorengehen. Es muss dabei bekannt sein, in welchen Ordnern die jeweilige Applikation Informationen ablegt. Die *Dash-*

Eingehende
Darstellung der
Ergebnisse

board-Software *Grafana* wird mit oben aufgeführter Einstellung bereits während des Installationsprozesses (= Start des *Containers*) mit zwei Plug-Ins versehen.

Mit diesem Anwendungsbeispiel konnte gezeigt werden, wie für beliebige an der Baustelle mittels Webformularen aufgenommene Daten innerhalb kurzer Zeit Dashboard-Visualisierungen umsetzbar sind (siehe Abbildung 68).

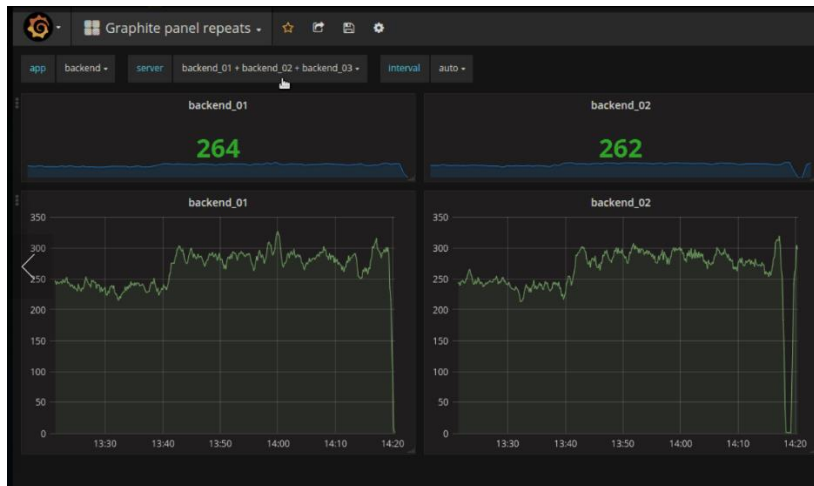


Abbildung 68: Dashboard-basierte Datenvisualisierung mittels Grafana.

Für die Schaffung einer allgemeinen Datenumgebung (CDE) für Züblin-interne Daten fehlt für das vorliegende Beispiel die Schaffung einer DMZ-internen [REST-API](#). Dazu muss der bereitgestellte *NodeJS*-Webserver mit Datenanbindungsmöglichkeiten versehen werden und die zugehörige Dokumentation offen zugänglich sein. Konzepte zur Strukturierung der *REST-API* werden in Kap. 2.6.2.1 erläutert.

2.6.2.3 Arbeitsweise mit software-seitig bereitgestellten REST-APIs und Umsetzung anhand des Beispiels Dalux

Im Rahmen unterschiedlicher Bauprojekte werden bei Züblin unter anderem folgende Projektplattformen eingesetzt: *BIM 360 / Forge* (Autodesk) [22], *think project!* (*think project!*) [16], *FusionLive* (McLaren Software) [17], *aconex* (Aconex) [15], *Dalux Field* ([Dalux](#)), *IRIS* (ITC Engineering) [33] und *iTWO 4.0* (RIB Software) [20]. Im Wesentlichen sind diese Plattformen als Webapplikationen implementiert und bieten teilweise den automatisierten Zugriff auf Daten und Funktionen über [REST-APIs](#) an. So ist es unter anderem möglich, auf die Projektinformationen oder abgelegte Plan- und Modelldaten zuzugreifen und sie anderen Systemen software-unabhängig zur Verfügung zu stellen. Mit *luxnet* wurde eine exemplarische prototypische zentrale software-unabhängige Datenhaltung unter Nutzung der REST-APIs von *Dalux* umgesetzt. *Dalux* ist eine webbasierte Projektplattform, die eine standortübergreifende Zusammenarbeit ermöglicht und verschiedene Module zur Unterstützung der modellbasierten Arbeitsweise beinhaltet. *Luxnet* synchronisiert die Projektinformationen aus *Dalux* über die REST-API in eine software-unabhängige Datenbank (siehe Abbildung 69).

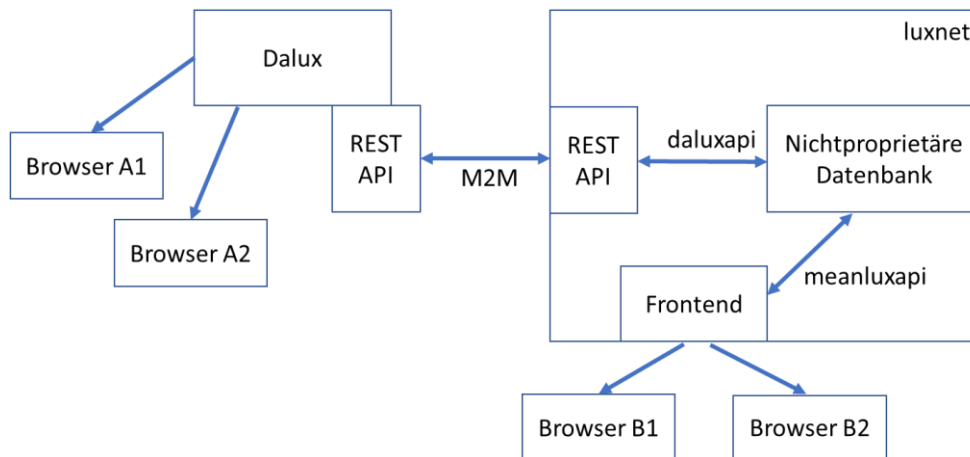


Abbildung 69: Prototypische Umsetzung einer software-unabhängigen Datenhaltung. M2M: Ma-schine-zu-Maschine-Kommunikation

Luxnet implementiert eine interne REST-API Namens *meanluxapi*, die den Zugriff auf eine software-unabhängige Datenbank abstrahiert und den Zugriff auf die Daten über einer HTTP-basierte Kommunikation gewährleistet. Die von *meanluxapi* zur Verfügung gestellten Routen und Funktionen sind in Tabelle 8 gelistet.

Route	HTTP-Methode	Aufgabe
/meanluxapi/projects	GET	Rückgabe der Projektliste aus der externen Datenbank
/meanluxapi/projects/:id	GET	Rückgabe eines Projekts mit einer bestimmten ID
/meanluxapi/projects	POST	Projekte in externe Datenbank speichern
/meanluxapi/projects/:id	PUT	Aktualisierung eines Projekts mit einer bestimmten ID
/meanluxapi/projects/:id	DELETE	Löschung eines Projektes mit einer bestimmten ID aus der externen Datenbank
/meanluxapi/projects	DELETE	Löschung aller Projekte aus der externen Datenbank

Tabelle 8: Auflistung der *meanluxapi*-Endpunkte

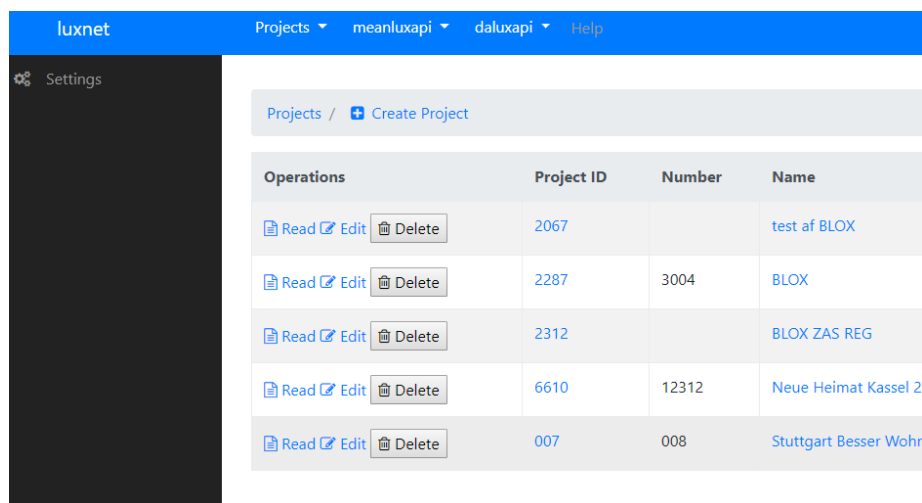
Die Schnittstelle zur API von *Dalux* wird durch die interne *daluxapi* abgebildet. Die zur Verfügung stehenden Routen und Funktionen der *daluxapi* sind in Tabelle 9 aufgelistet.

Route	HTTP-Methode	Aufgabe
/daluxapi/projects/	GET	Rückgabe der Projektliste aus <i>Dalux</i>
/daluxapi/projects/:id	GET	Rückgabe eines Projekts mit einer bestimmten ID
/daluxapi/projects/:id/issues	POST	Rückgabe aller Issues zu einem bestimmten Projekt
/daluxapi/importprojects/	PUT	Import aller Projekte

Tabelle 9: Auflistung der *daluxapi*-Endpunkte

Das Frontend von *luxnet* nutzt diese interne REST-API, um den Zugriff auf die Projektdaten zu realisieren. *Luxnet* ist eine Webapplikation, die Funktionen unabhängig

von *Dalux* zur Verfügung stellt. So können die Projektdaten aus der Datenbank über eine eigene Oberfläche verwaltet werden (siehe Abbildung 70).



The screenshot shows the 'luxnet' web application interface. The top navigation bar includes 'luxnet', 'Projects', 'meanluxapi', 'daluxapi', and 'Help'. A left sidebar contains a 'Settings' link. The main content area displays a 'Projects / Create Project' header and a table with the following data:

Operations	Project ID	Number	Name
Read Edit Delete	2067		test af BLOX
Read Edit Delete	2287	3004	BLOX
Read Edit Delete	2312		BLOX ZAS REG
Read Edit Delete	6610	12312	Neue Heimat Kassel 2
Read Edit Delete	007	008	Stuttgart Besser Wohn

Abbildung 70: Prototyp luxnet

Luxnet kann unabhängig von *Dalux* weiterentwickelt werden und Funktionen zur Verfügung stellen, die von der ursprünglichen Anwendung *Dalux* nicht vorgesehen sind. *Dalux* kann weiterhin unabhängig von *luxnet* genutzt werden.

Dieser Prototyp verdeutlicht den Wert einer software-unabhängigen zentralen Datenhaltung. Er ermöglicht die Entwicklung von weiteren Anwendungen unabhängig von der ursprünglichen Software unter Verwendung der gleichen synchronisierten Daten. Darüber hinaus wird eine Analyse dieser Daten mit Methoden des maschinellen Lernens und Big Data möglich. Die Anforderungen aus Kap. 2.4.2.5 werden von *Dalux* dennoch nicht erfüllt. Es fehlt bspw. die Editierbarkeit der plattforminternen gespeicherten Informationen. Dadurch müssen die Funktionalitäten von *luxnet* auf Datenauswertungen beschränkt bleiben, wenn eine doppelte Datenhaltung vermieden werden soll.

2.6.2.1 Betreiben einer ViKoDB in einer DMZ aus der Sicht eines Produktherstellers

Wie in Kap. 2.5.3 beschrieben, werden im Forschungsprojekt die originären Bauproduktbeschreibungen mittels Fuseki-Servern angeboten und verteilt. Die Ursprungsdaten werden auf den ViKoLink-Server synchronisiert und dort vorverarbeitet. Sollen Bauprodukt Daten von den jeweiligen Herstellern angeboten werden, wäre das Betreiben des jeweiligen Webservers in einer Demilitarisierten Zone des Herstellers wünschenswert.

Es wurde deshalb im Projekt untersucht, inwiefern sich *Fuseki* auch innerhalb einer DMZ betreiben lässt. Die Software ließ sich aufwandsarm in eine [Docker](#)-Umgebung überführen und auf Testservern betreiben, für einen von außerhalb der Strabag-Firewall zugänglichen Betrieb stießen wir jedoch auf erhebliche technische Schwierigkeiten (die Konfiguration des firmeninternen Load Balancings verursachte offenbar einen Konflikt mit den SSL-Einstellungen von *Fuseki*), die innerhalb der Projektlaufzeit nicht mehr behoben werden konnten. Außerhalb einer Firewall ist der Betrieb einer ViKoDB jedoch problemlos durchführbar. Da es zur Software *Fuseki* vielversprechende Alternativen gibt, wird die Behebung des genannten technischen Problems erst aufgenommen, sobald eine Festlegung auf die Software *Fuseki* in zukünftigen Forschungsaktivitäten erfolgt ist.

3 Notwendigkeit der geleisteten Arbeit

Das Thema des vorliegenden Forschungsprojekts bestand in der Integration solaraktiver Fassadenkomponenten in Bauprozesse. Ursprünglich war das Projektkonsortium von der Umsetzbarkeit sequentiell standardisierter Bauabläufe ausgegangen. Diese Annahme musste bereits nach Ablauf des ersten Arbeitspakets zugunsten einer synchronen Datenbearbeitung verworfen werden. Das Projektkonsortium sah sich vor die herausfordernde Aufgabe gestellt, den standardisierten Datenaustausch zwischen Softwareprogrammen, wie er von der Organisation buildingSMART vertreten wird, durch eine plattformunabhängige Datenhaltung zu ersetzen. □ IFC wurde durch die Anwendung von Webtechnologien zur vollständigen geometrischen und semantischen Beschreibung von Bauprodukten ersetzt, wie auch die zu Beginn angestrebten, auf sequentiellen Abläufen beruhenden Multiagentensysteme und die IDM/MVD-Methode von buildingSMART durch die im Forschungsprojekt neu entwickelte RBV/OBW-Methodik ersetzt wurden. Auch die IT-Herangehensweise wurde im Projektverlauf umgestellt. Anstelle der nutzerseitigen Anwendung von Softwareprogrammen im BIM-Kontext wurde die für eine nichtsequentielle Arbeitsweise erforderliche IT-Architektur ermittelt und prototypisch umgesetzt.

Das Forschungsprojekt hat mit der Umsetzung dreier vollständiger, software-unabhängiger Bauproduktbeschreibungen und deren prototypischer Anbindung an bestehende Planungsabläufe sowie durch die prototypische Umsetzung eines software- und firmenübergreifenden Informationsverwaltungssystems (am Beispiel der Kollisionsdaten) einen runden Abschluss gefunden. Einige im Projektverlauf entstandene Fragestellungen (wie z. B. die plattformunabhängige Wiedergabe von BIM-Projektinformationen, der Datenrückfluss der an die dezentrale Datenverwaltung angebotenen Software-Programme oder die Standardisierung software-externer APIs) konnten bisher nur peripher behandelt werden und werden im Rahmen der Verwertung vertieft.

4 Voraussichtlicher Nutzen und Verwertbarkeit der Ergebnisse

Das Forschungsprojekt SolConPro hat in der herkömmlichen BIM-Arbeitsweise erhebliche Schwächen aufdecken können, die sich vor allem bei komplexen Bauvorhaben stark bemerkbar machen. Zur Umsetzung von Gebäuden mit verbesserter Energiebilanz, speziell zur ganzheitlichen Planung und Installation solaraktiver Fassadensysteme, ist nicht eine verbesserte Umsetzung, sondern eine Korrektur der bisher angestrebten Arbeitsweise erforderlich. Den zentralen Dreh- und Angelpunkt stellt hierbei das Thema Datenhaltung dar. Bisher werden Daten im BIM-Kontext verschlüsselt hinterlegt, wodurch sowohl Datenauswertungen als auch Vernetzungen zwischen Unternehmen und Softwareprogrammen nur von den jeweiligen Softwareherstellern vollumfänglich realisiert werden können. Wie der Projektverlauf gezeigt hat, stellt dies für die Digitalisierung des Bauwesens eine erhebliche Einschränkung und Bremse dar.

Die Bedeutung dieser Erkenntnis ist für den weiteren Verlauf der Digitalisierung der Bauabläufe kaum zu überschätzen. Die Notwendigkeit einer applikationsunabhängigen Datenhaltung schließt aber gleichzeitig auch die Fähigkeit mit ein, Datenmengen eigenständig zu speichern und zu verwalten. Vor allem für geometrische Informationen, die im Bauwesen überall benötigt werden, stellt dies eine erhebliche Herausforderung dar. Im Rahmen des Forschungsprojekts SolConPro wurden erste Bauproduktbeschreibungen komplexer Fassadenelemente umgesetzt und mittels Filter- und Vorverarbeitungsmethoden einerseits, mittels Plug-Ins zur Anbindung von Softwareprogrammen andererseits in die bestehende BIM-Arbeitsweise integriert. Für die Umsetzung einer generellen applikationsunabhängigen Datenhaltung stellt dies nur den ersten wichtigen Schritt dar.

Fortschritt bei anderen Stellen	Der voraussichtliche Nutzen der genannten Projektergebnisse besteht in einem deutlich verbesserten Informationsfluss über alle Bauprojektphasen, wodurch auch komplexere Bauprojektoptimierungen wie z. B. energetische Verbesserungen bzw. solar-aktive Fassadensysteme effektiv umgesetzt werden können. Dies gilt für Optimierungen aller Art, was die umfassende Verwertbarkeit der Projektergebnisse deutlich macht. Der zeitliche Rahmen lässt sich nur schwer quantifizieren, jede Verbesserung der Kommunikation steigert jedoch bereits per se die Effektivität von Bauabläufen (und damit auch das Vertrauen in die Umsetzbarkeit auch komplexerer Bauprojekte). Das Forschungsprojekt SolConPro trägt erheblich dazu bei, dass auch das Bauwesen von der Umsetzung anderweitig bereits etablierter Methoden der Digitalisierung zu profitieren beginnt.
---------------------------------	--

5 Fortschritt bei anderen Stellen

Im Bauwesen selbst sind dem Projektkonsortium nur wenige Aktivitäten bekannt geworden, die auf eine vollständige und applikationsunabhängige Wiedergabe geometrischer Informationen abzielen. Vereinzelt Aktivitäten bei buildingSMART (v.a. der Projektvorschlag *PA-1 Parametric*) werden dokumentiert, verlaufen aber aufgrund der thematischen Implikationen (erschwerter Verhandlungsgrundlage mit den BIM-Softwareherstellern) meist im Sand. Die getroffenen Schemavereinbarungen sind im Rahmen von [IFC](#) nur nützlich, sofern sie auch fehlerfrei in Software-Schnittstellen übernommen werden; diese Forderung jedoch widerspricht dem Interesse von buildingSMART, die ihre Standardisierungsaktivitäten auf die Informationsübertragung zwischen verschiedenen proprietären BIM-Softwareprogrammen begrenzen möchten. Eine erforderliche Standardisierung des Geometrie-kerns liegt somit in weiter Ferne. Analoge Aktivitäten in Vereinigungen von Baukonzernen (z. B. ENCORD) sind dem Projektkonsortium nicht bekannt geworden.

Außerhalb des Bauwesens sind vor allem die Standardisierungsaktivitäten von W3C und ECMA hervorzuheben. Die Semantic-Web-Technologie als W3C-Empfehlung und die entsprechenden Aktivitäten der *Linked Building Data Community Group* (LBDCG) wurden im Rahmen des Forschungsprojekts genauer untersucht und sind in Kap. 2.4.1.1 beschrieben. Zur 3D-Visualisierung geometrischer Inhalte in Webbrowsern gab es im Rahmen von ECMAScript 3 / WebGL einige spannende Fortschritte zu verzeichnen, die wegen der Notwendigkeit der Standardisierung geometrischer Beschreibungen zur browserübergreifenden Visualisierung auch für die vorliegenden Projekthinhalte relevant sind. Mittlerweile existieren einige Dutzend WebGL-Frameworks, von denen v. a. die Bibliothek *three.js* [34] hervorzuheben ist (sie wird u. a. von Autodesk Forge[22] verwendet). Derzeit sind deren Geometrie-klassen auf Basis-klassen beschränkt, wenn auch diverse komplexe geometrische Objekte (z. B. 3D-Bezierkurven, Dodekaeder, Texturen) bereits enthalten sind. Allgemein sind Aktivitäten zur plattformunabhängigen Datenhaltung geometrischer Informationen verblüffend wenig verbreitet, wodurch die im Forschungsprojekt als notwendig erkannte Entkopplung von Bauproduktherstellern und Gebäudemodellplanern derzeit leider nur prototypisch umgesetzt werden kann. Das Bauwesen scheint hier hinsichtlich der Anforderung einer hochgradig firmen- und software-übergreifenden Kommunikationsstruktur unter den Industriesparten eine Sonderrolle einzunehmen. Automobil- und Flugzeugindustrie arbeiten häufig mit proprietären oder eben statischen Formaten (wie z. B. JT [35]), da offenbar zu geometrischen Inhalten (noch) kein besonderer automatisierter Austauschbedarf mit anderen Firmen oder Softwareprogrammen existiert.

Eisenlohr, Johannes; Eller, Christian; Leifgen, Christian; Boudhaim, Marouane; Maurer, Christoph; Sprenger, Wendelin; Kuhn, T. E.:

Flexible Filtering of Heterogeneous Data Using the Example of the Design and Simulation of Building Integrated Photovoltaics

In: BauSIM 2018, 26.-28. September, Karlsruhe, Deutschland

Wagner, Anna; Möller, Laura Kristina; Leifgen, Christian; Rüppel, Uwe:

SolConPro: Describing multi-functional building products using Semantic Web Technologies

Akzeptiert zum oralen Vortrag in: 12th European Conference on Product and Process Modelling 2018, Kopenhagen, Denmark.

Wagner, Anna; Möller, Laura Kristina; Leifgen, Christian:

SolConPro: Verteilte Produktdatenkataloge mit automatisierter Datenverarbeitung

Zum oralen Vortrag eingereicht in: 30. Forum Bauinformatik 2018, Weimar

Wagner, Anna; Möller, Laura Kristina; Eller, Christian; Leifgen, Christian; Rüppel, Uwe

Mela, Kristo; Pajunen, Sami; Raasakka, Ville (eds.):

SolConPro: An Approach for the Holistic Integration of Multi-Functional Façade Components into Buildings' Lifecycles. In: 17th International Conference on Computing in Civil and Building Engineering Conference Proceedings, Tampere, Finland.

In: International Conference on Computing in Civil and Building Engineering, 17. [Konferenz-Beitrag], (2018)

Benndorf Gesa; Réhault, Nicolas; Clairembault, M; Rist, Tim

Describing HVAC controls in IFC – Method and application

In: Energy Procedia, 122 pp. 319-324. DOI: 10.1016/j.egypro.2017.07.330

Eisenlohr, Johannes; Wilson, Helen Rose, Kuhn, Tilmann E.

IFC-Based Electricity Simulation of a Complex BIPV Façade

In: Advanced Building Skins, 3. Oktober 2017, Bern. [Konferenz-Beitrag], (2017)

Maurer, Christoph; Sprenger, Wendelin; Franz, Steffen; Boudhaim, Marouane; Lodewijks, Jan; Rüppel, Uwe; Kuhn, Tilmann E.:

Machine-code functions in BIM for cost-effective high-quality buildings.

In: Energy and Buildings, 155 pp. 467-474. ISSN 03787788 [Artikel], (2017)

Maurer, Christoph; Sprenger, Wendelin; Franz, Steffen; Lodewijks, Jan; Rüppel, Uwe; Kuhn, Tilmann E.

Auer, Thomas; Knaack, Ulrich; Schneider, Jens (eds.):

Machine code functions in BIM for cost-effective high-quality buildings.

In: PowerSkin Conference, 19. January 2017, Munich. PowerSkin Conference Proceedings Munich [Konferenz-Beitrag], (2017)

Franz, Steffen; Rüppel, Uwe; Kuhn, Tilmann E.; Teizer, Jochen:

A Multi-Agent-Based Platform for the Early Integration of Photovoltaic Systems in Building Façades.

In: International Conference on Computing in Civil and Building Engineering (ICCCBE), 06.-08. Juli 2016, Osaka, Japan. Proceedings of the 16th International Conference on Computing in Civil and Building Engineering [Konferenz-Beitrag], (2016)

Leifgen, Christian; Rüppel, Uwe; Kuhn, Tilmann E.; Teizer, Jochen:

BIM-based Collaboration Platform for the Holistic Integration of Energy Active Façade Components.

In: International Conference on Computing in Civil and Building Engineering (ICCCBE), 06.-08. Juli 2016, Osaka, Japan. Proceedings of the 16th International Conference on Computing in Civil and Building Engineering [Konferenz-Beitrag], (2016)

Wagner, Anna; Franz, Steffen; Leifgen, Christian

Berthold, Tim; Brandt, Sebastian; Schiermeyer, Chris (eds.):

BIM-basierte Kollaborationsplattform mit objektorientierter Versionierungshistorie und Produktdatenkatalog zur Integration energetisch aktiver Fassadenelemente in Bauprozesse. [Online-Edition: <http://dx.doi.org/10.15488/686>]

In: 28. Forum Bauinformatik, 19.-21. September 2016, Hannover, Deutschland. In: Berichte aus der Bauinformatik, 28. Institut für Risiko und Zuverlässigkeit, Hannover [Workshop-Beitrag], (2016)

Wagner, Anna; Rüppel, Uwe

Slusarczyk, Grazyna; Strug, Barbara; de Wilde, Pieter (eds.):

Cooperative Design Methods for Energy Active Facade Elements Based on BIM-server and Multi-Agent-Services using JADE.

In: 23rd EG-ICE International Workshop, 29. June - 01. July 2016, Kraków, Poland. Proceedings of the 23rd International Workshop of the European Group for Intelligent Computing in Engineering [Konferenz-Beitrag], (2016)

7 Literaturverzeichnis

- [1] Bundesministerium für Wirtschaft und Energie (BMWi), Hrsg., „Energieeffizienzstrategie Gebäude - Wege zu einem nahezu klimaneutralen Gebäudebestand“. Schöne Drucksachen GmbH, Berlin, 18-Nov-2015.
- [2] EnOB REG II, „Energetische Optimierung der zentralen Kälteversorgung im Verwaltungsgebäude Z3 der Ed. Züblin AG in Stuttgart“. Mittelgeber: Bundesministerium für Wirtschaft und Technologie (BMWi).
- [3] C. Maurer u. a., „Machine-code functions in BIM for cost-effective high-quality buildings“, *Energy Build.*, Bd. 155, S. 467–474, Nov. 2017.
- [4] J. Beetz, J. van Leeuwen, und B. de Vries, „IfcOWL: A case of transforming EXPRESS schemas into ontologies“, *Artif. Intell. Eng. Des. Anal. Manuf.*, Bd. 23, Nr. 01, S. 89, Feb. 2009.
- [5] T. M. de Farias, A. Roxin, und C. Nicolle, „IfcWoD, Semantically Adapting IFC Model Relations into OWL Properties“, *ArXiv151103897 Cs*, Nov. 2015.
- [6] M. H. Rasmussen, P. Pauwels, C. A. Hviid, und J. Karlshøj, „Proposing a Central AEC Ontology That Allows for Domain Specific Extensions“, 2017, S. 237–244.
- [7] P. Pauwels, M. H. Rasmussen, und G. F. Schneider, „PRODUCT“. [Online]. Verfügbar unter: <https://github.com/w3c-lbd-cg/product>. [Zugegriffen: 01-Aug-2018].
- [8] M. Lefrançois, „PROPS“. [Online]. Verfügbar unter: <https://github.com/w3c-lbd-cg/props>. [Zugegriffen: 01-Aug-2018].
- [9] M. H. Rasmussen, „OPM“. [Online]. Verfügbar unter: <https://github.com/w3c-lbd-cg/opm>. [Zugegriffen: 01-Aug-2018].
- [10] P. Bonsma, I. Bonsma, T. Zayakova, A. van Delft, R. Sebastian, und M. Böhms, „Open standard CMO for parametric modelling based on semantic web“, in *eWork and eBusiness in Architecture, Engineering and Construction*, A. Mahdavi, B. Martens, und R. Scherer, Hrsg. CRC Press, 2014, S. 923–928.
- [11] P. Pauwels, S. Zhang, und Y.-C. Lee, „Semantic web technologies in AEC industry: A literature overview“, *Autom. Constr.*, Bd. 73, S. 145–165, Jan. 2017.
- [12] R. Sebastian, H. M. Böhms, P. Bonsma, und P. W. van den Helm, „SEMANTIC BIM AND GIS MODELLING FOR ENERGY-EFFICIENT BUILDINGS INTE-

- GRATED IN A HEALTHCARE DISTRICT“, *ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci.*, Bd. II-2/W1, S. 255–260, Sep. 2013.
- [13] A. Perzylo, N. Somani, M. Rickert, und A. Knoll, „An ontology for CAD data and geometric constraints as a link between product models and semantic robot task descriptions“, 2015, S. 4197–4203.
- [14] „BIM 360“. [Online]. Verfügbar unter: <https://bim360.autodesk.com/>. [Zugegriffen: 01-Aug-2018].
- [15] „Aconex“. [Online]. Verfügbar unter: <https://www.aconex.com/>. [Zugegriffen: 01-Aug-2018].
- [16] „thinkproject!“. [Online]. Verfügbar unter: <https://www.thinkproject.com/>. [Zugegriffen: 01-Aug-2018].
- [17] „FusionLive“. [Online]. Verfügbar unter: <https://eu.mclarenonair.com/>. [Zugegriffen: 01-Aug-2018].
- [18] Apache Software Foundation, „hadoop“. [Online]. Verfügbar unter: hadoop.apache.org. [Zugegriffen: 01-Aug-2018].
- [19] „Jotne IT“. [Online]. Verfügbar unter: <http://www.jotneit.no/express-data-manager-edm>. [Zugegriffen: 01-Aug-2018].
- [20] „iTWO“. [Online]. Verfügbar unter: <https://www.rib-software.com/de/landingpage/rib-itwo.html>. [Zugegriffen: 01-Aug-2018].
- [21] „KanBo“. [Online]. Verfügbar unter: www.kan.bo/. [Zugegriffen: 01-Aug-2018].
- [22] „Autodesk Forge“. [Online]. Verfügbar unter: <https://forge.autodesk.com/>. [Zugegriffen: 01-Aug-2018].
- [23] *3DEXPERIENCE Plattform*. Dassault Systèmes.
- [24] A. De Nicola, M. Missikoff, und R. Navigli, „A Proposal for a Unified Process for Ontology Building: UPON“, in *Database and Expert Systems Applications*, Bd. 3588, K. V. Andersen, J. Debenham, und R. Wagner, Hrsg. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, S. 655–664.
- [25] P. Pauwels u. a., „5th Workshop on Linked Data in Architecture and Construction“, Dijon, Frankreich, Workshop Report 5, 2017.
- [26] G. Talarico, „Revit API Docs“. [Online]. Verfügbar unter: <http://www.revitapidocs.com/>. [Zugegriffen: 01-Aug-2018].
- [27] „RestSharp“. [Online]. Verfügbar unter: <http://restsharp.org/>. [Zugegriffen: 01-Aug-2018].
- [28] „json.NET“. [Online]. Verfügbar unter: <https://www.newtonsoft.com/json>. [Zugegriffen: 01-Aug-2018].
- [29] „MoreApp“. [Online]. Verfügbar unter: <https://moreapp.com/de/>. [Zugegriffen: 01-Aug-2018].
- [30] „Grafana“. [Online]. Verfügbar unter: <https://grafana.com/>. [Zugegriffen: 01-Aug-2018].
- [31] „PostgreSQL“. [Online]. Verfügbar unter: <https://www.postgresql.org/>. [Zugegriffen: 01-Aug-2018].
- [32] „InfluxData“. [Online]. Verfügbar unter: <https://www.influxdata.com/>. [Zugegriffen: 01-Aug-2018].
- [33] „IRIS“. [Online]. Verfügbar unter: www.itc-engineering.de/en/products/iris/. [Zugegriffen: 01-Aug-2018].
- [34] „three.js“. [Online]. Verfügbar unter: <https://threejs.org/>. [Zugegriffen: 01-Aug-2018].
- [35] „ISO 14306:2017: Industrial automation systems and integration -- JT file format specification for 3D visualization“. Nov-2017.

Anhang A: Technologischer Glossar

Hardware und Betriebssystem:

Als Hardware für das Webhosting der von der TU Darmstadt entwickelten Anwendungen wurde ein Dell PowerEdge R730 Server (CPU: 2 * Intel Xeon E5-2650, Arbeitsspeicher: 64GB) eingerichtet. Darauf laufen alle Server-Anwendungen, mit Ausnahme einer Instanz der ViKoDB, die physisch getrennt auf einem weiteren Server (CPU: Core2Quad Q9300, Arbeitsspeicher: 4GB) ausgeführt wird.

Die von der Ed. Züblin AG entwickelten Webanwendungen sind mittels *Docker*-Containern (siehe unten) an das Serverlastverteilungssystem der internen IT-Abteilung angebunden. Alle im Forschungsprojekt entwickelten Anwendungen verwenden in Entwicklung und Betrieb das Linux-Betriebssystem Ubuntu 16.04 LTS.

Software:

Autodesk Revit

Autodesk Revit ist eine proprietäre CAD-Planungssoftware zur Erstellung und Verarbeitung von BIM-Modellen. Die von Autodesk entwickelte Software stellt Planern verschiedener Disziplinen (Architektur, Gebäudetechnik, Ingenieurbau) Werkzeuge im Sinne des Building Information Modelling zur Verfügung und ermöglicht es Entwicklern diese durch eine bereitgestellte .NET-API zu erweitern. (<https://www.autodesk.de/products/revit/overview>, 10.08.2018)

Dalux

Das Unternehmen *Dalux* bietet mit *Dalux Build*, *Dalux Field* und dem *Dalux BIM Viewer* Software-Werkzeuge an, die eine standortübergreifende Zusammenarbeit erlauben. An Funktionalitäten hervorzuheben sind das modellbasierte Änderungsmanagement und die synchrone Darstellung von 2D-Plan- und 3D-Modellinformationen. (<https://www.dalux.com>, 10.08.2018)

Docker

Docker ist eine erstmals im Jahr 2013 veröffentlichte Open-Source-Software, die es allgemein erlaubt, Software-Anwendungen zu isolieren. Die Installation einer Software findet somit unabhängig davon statt, welche sonstigen Anwendungen bisher auf dem Rechner installiert wurden.

Die aus der *Docker*-Software resultierenden sog. Container ähneln ihrer Struktur nach den virtuellen Maschinen (VM), die innerhalb eines Rechners weitere Rechner simulieren. Für die Verwaltung von VMs kommt ein Programm zum Einsatz, das sich Hypervisor nennt und auf dem Basis-Betriebssystem des Rechners aufgesetzt wird (siehe Abbildung 71).

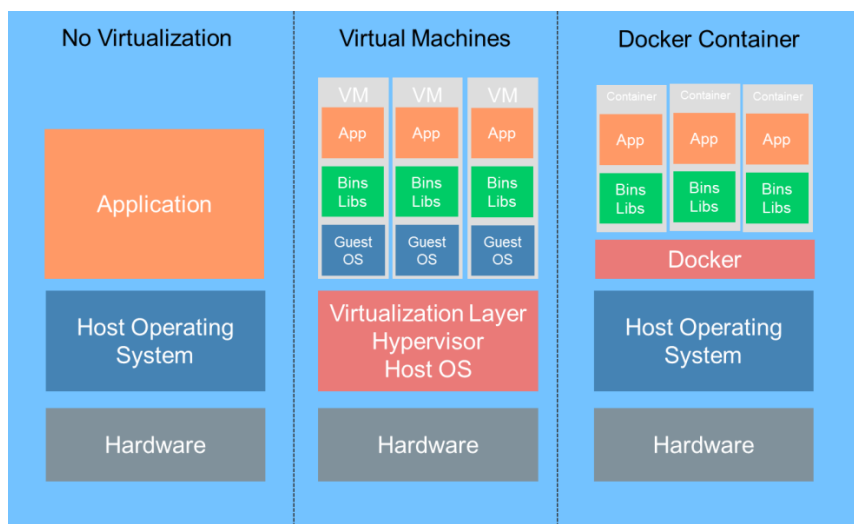


Abbildung 71: Darstellung der Software-Architekturen ohne Virtualisierung (links), mit Hypervisor (Mitte) und auf Docker-Basis (rechts).

Die Docker-Container hingegen basieren nicht auf einem Hypervisor, sondern auf der Docker-Engine, die auf einem bestehenden Betriebssystem installiert wird. Mit diesem lassen sich Anwendungen in sog. Containern virtualisieren und sind somit von anderen Anwendungen isoliert. Abhängigkeitskonflikte (z. B. von unterschiedlichen *Python*- oder *Java*-Versionen) gehören so der Vergangenheit an, zudem kann der Installationsprozess vollständig automatisiert werden. Letzteres ist vor allem für die Bereitstellung von Webservern von Belang, die im Rahmen der SolConPro-Datenbereitstellung auf die Server von unterschiedlichen Baubeteiligten übertragen werden müssen. Sollten im Installationsprozess für Applikationen bestimmte Einstellungen vorgenommen oder Pakete installiert werden müssen, kann dies direkt in der Docker-Datei geschehen. Der gesamte Installationsprozess ist auf wenige Textdateien beschränkt und kann so auf Server beliebiger anderer Firmen übertragen werden.

Der Prozess zur Erstellung und Verwaltung von Docker-Anwendungen verläuft dreiteilig. In einer Textdatei, der Docker-Datei, werden die einzelnen Bestandteile der Applikation aufgeführt, die zur Installation notwendig sind. Diese werden dann zu einem binären Docker-Image verarbeitet. So können Nutzer von Standard-Anwendungen bereits auf bestehende Docker-Images zurückgreifen, wodurch die Erstellung der Textdatei nur einmal durchgeführt werden muss. Zur Verwaltung der entstehenden Docker-Images bietet das Programm einen sog. Docker-Hub an, eine Art Bibliothek für Docker-Images, die bereits über 100 000 Applikationen beinhaltet. Aus diesen Docker-Images nun können bei Vorliegen der Docker-Engine Container erstellt werden, ohne dass weitere Voraussetzungen des lokalen Aufbaus erforderlich wären.

Die Automatisierung der Installation von Applikationen unabhängig von bereits bestehender Software hat aber noch weitere Vorteile, auf die an anderer Stelle näher eingegangen wird (siehe Kap. 2.5.2 ff.). So kann eine Anwendung mehreren Rechnern zugeordnet werden, auf die bei Ausfall einer Instanz zurückgegriffen werden kann („docker swarm“). Dies stellt vor allem im Bereich Webhosting einen erheblichen Vorteil dar. Unterschiedliche Applikationen können auch innerhalb einer Datei zusammengefasst werden, was die Übersichtlichkeit eines Anwendungspakets erheblich verbessert („docker compose“). Zur Vereinfachung der Kommunikation zwischen Container-Anwendungen können diese innerhalb von Netzwerken („docker network“) betrieben werden. Auch diese Funktionalität kam innerhalb SolConPro

zum Einsatz (siehe Kap. 2.6.2.2). Für die genannten Erweiterungen wurde von Docker das mit **JSON** verwandte YAML-Format herangezogen.

Bei der Ed. Züblin AG ist die Container-Virtualisierung mittels Docker für Webapplikationen vorgeschrieben. Spezifisch kommt eine Docker-Swarm-Herangehensweise zum Einsatz, die die zentralisierte und automatisierte Verwaltung aller Webapplikationen (inkl. der innerhalb der Container gemounteten Laufwerke) ermöglicht. Zudem kommt für die entwickelten Docker Images ein interner Docker Hub zum Einsatz. (<https://www.docker.com>, 10.08.2018)

GitLab

GitLab ist eine Plattform zur Erstellung von Webapplikationen. Bereitgestellt wird eine umfassende Nutzeroberfläche zur Verwaltung von git-Ablagen, zur Organisation von Aufgaben und zum automatisierten Web-Deployment. Die Anwendung erlaubt sowohl eine git-Umgebung zur Zusammenfügung unterschiedlicher Quellcode-Inhalte (Continuous Integration, CI) als auch eine entwicklerseitige Verwaltung und Aktualisierung von extern zugänglichen Webinhalten (sog. Continuous Delivery, CD) und stellt damit einen wesentlichen Baustein zur DevOps-Umsetzung dar.

Abbildung 72: Graph der Quellcode-Entwicklung zum Clash-Management-System mit zeitlicher Darstellung der Änderungen, der zugehörigen Personen und der git-Verzweigungen („Branches“) in *gitlab* Abbildung 72 zeigt die graphische Visualisierung der git-basierten Verwaltung von *git branches* in *GitLab*.

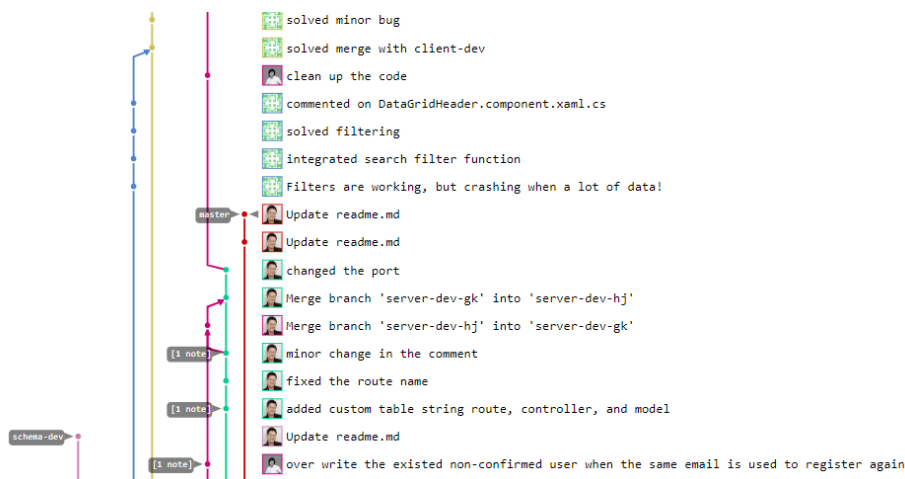


Abbildung 72: Graph der Quellcode-Entwicklung zum Clash-Management-System mit zeitlicher Darstellung der Änderungen, der zugehörigen Personen und der git-Verzweigungen („Branches“) in *gitlab*

In Abbildung 73 wird die graphische Benutzeroberfläche zum Continuous Delivery gezeigt, die die erneute Bereitstellung eines veränderten Quellcodes auf der verbundenen Seite auf einen Mausklick reduziert.

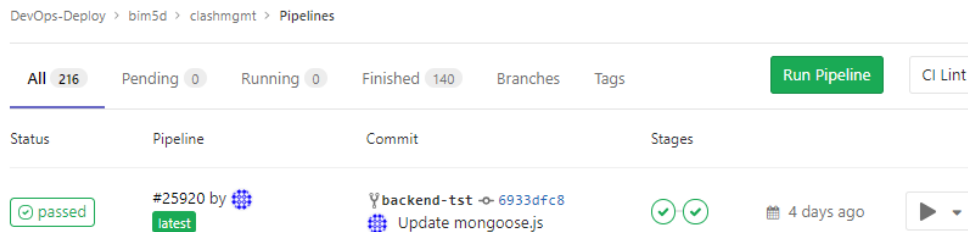


Abbildung 73: Herangehensweise in gitlab für Continuous Delivery (CD) von Webseiten (vereinfachte Darstellung). Jede Aktualisierung einer Webseite wird automatisch protokolliert und archiviert.

Für die Verbindung von Continuous Integration und Continuous Delivery sorgt eine Datei namens `gitlab-ci.yml`, die für jeden Applikationsbestandteil die technischen Schritte beschreibt, die für das automatisierte Webdeployment stattfinden müssen. Auf diese Weise werden automatisiert *Docker-Images* erstellt, in einem frei wählbaren *Docker-Hub* abgelegt und aus diesen *Docker-Container* erzeugt. (<https://gitlab.com/>, 10.08.2018)

SageMath

SageMath ist ein unter der GPL-Lizenz entwickeltes freies Open-Source-Mathematiksystem. Aufbauend auf existierenden Open-Source-Paketen und unter Verwendung einer auf Python basierten Sprache ermöglicht das System verschiedenste mathematische Berechnungen. (<http://www.sagemath.org/>, 10.08.2018)

Angular-QueryBuilder

Angular-QueryBuilder ist ein unter der MIT-Lizenz stehendes JavaScript-Paket zur Nutzung in der Laufzeitumgebung Node.js. Basierend auf dem *jQuery QueryBuilder* (<https://querybuilder.js.org/>, 01.08.2018) stellt er einen konfigurierbaren Abfrage-Generator zur Integration in das Front-End-Webapplikationsframework Angular zur Verfügung. (<https://www.npmjs.com/package/angular2-query-builder>, 10.08.2018)

MongoDB

MongoDB ist eine sogenannte Not-Only-SQL-Datenbank („NoSQL“). Sie ist die weltweit verbreitetste NoSQL-Datenbank und steht quelloffen zur Verfügung. Der wesentliche Unterschied zu SQL-Datenbanken besteht in der Möglichkeit der Speicherung und Verarbeitung schemafreier Informationsmengen, zudem haben NoSQL-Datenbanken Vorteile in der Verteilung von Datenbestand und Rechenleistung auf unterschiedliche Server (sog. Clustering). (<https://www.mongodb.com>, 10.08.2018)

Programmiersprachen:

Java

Java ist eine einfache, objektorientierte, plattformunabhängige und robuste Programmiersprache. Sie wurde 1995 entwickelt und wurde durch den Einsatz in Haushaltsgeräten und in Webbrowsern Anfang des 21. Jahrhundert durch die Firma Oracle zu einer mächtigen Programmiersprache mit vielseitiger Technologieunterstützung. Durch Java entwickelte Applikationen können als sogenannte JAR-Dateien als alleinstehende Applikation übersetzt und somit direkt auf Computern und Servern ausgeführt werden. Diese JAR-Dateien können einzelne Aufgaben erfüllen oder ganze Softwareapplikationen darstellen. (<https://www.java.com/de/>, 10.08.2018)

Python

Python ist eine universelle, objektorientierte, plattformunabhängige höhere Programmiersprache und wurde Anfang der 90er Jahre von Guido van Rossum entwickelt. Sie legt besonderen Wert auf eine knappe und lesbare Struktur. Auf Klammern wird zugunsten von Einrückungen verzichtet, wodurch viel Platz gespart wird. Python verfügt über eine umfangreiche Standardbibliothek, die z. B. auch das Lösen impliziter Gleichungssysteme erlaubt. Die Geschwindigkeit der Programmiersprache ist niedriger als bei anderen kompilierbaren Sprachen, da die Entwickler der Codeverständlichkeit gegenüber der Geschwindigkeit den Vorrang einräumen. (<https://www.python.org>, 10.08.2018)

JavaScript

JavaScript ist eine 1995 von Netscape entwickelte und 1997 veröffentlichte Skriptsprache zur Darstellung von dynamischen Browserinhalten. Seit 1997 wurde die Skriptsprache auch zum ECMA-Standard und wird seither von den gängigen Webbrowsern unterstützt. Laut aktueller Umfrage von Stackoverflow.com (2018)¹ ist JavaScript unter den Programmier-, Skript- und Markup-Sprachen mit 69.8% am weitesten verbreitet (Java: 45.3%, Python: 38.8%). Eine wesentliche Rolle spielte hierbei die Entwicklung des JavaScript-basierten Webservers Node.js, dessen Paketmanager npm mittlerweile mit 350 000 Open-Source-Paketen den größten Umfang aller Paketmanagementsysteme hat. Mit der Entwicklung von Node.js konnte sowohl die Frontend- als auch die Backendentwicklung in einer einzigen Skriptsprache durchgeführt werden (für das Backend wurde vorher meist PHP verwendet). (<https://www.javascript.com>, 10.08.2018)

Web-Technologien:

MEAN

Als technische Umsetzung wurde ein MEAN (*MongoDB, Express Framework, AngularJS (Frontend) und Node.JS (Server)*) Stack gewählt. MEAN ermöglicht die Implementierung verschiedenster Schnittstellen. Beispielsweise bietet eine *REST-API* die Möglichkeit der Anbindung an beliebige andere Systeme. MEAN-Stacks sind als asynchron aufgebaute Systeme schnell und skalierbar. Außerdem ist die Entwicklung durch ein einfaches Aufsetzen und „builden“ für Entwickler förderlich. MongoDB als NoSQL-Datenbank bietet eine größere Flexibilität gegenüber herkömmlichen relationalen Datenbanksystemen. Dies liegt u.a. daran, dass alle Inhalte im JSON-Format gespeichert werden. Dieses wird auch nativ in Node.JS verwendet, wodurch die Daten direkt genutzt werden können und kein Parsen und keine Schemamigration notwendig sind. (<http://mean.io>, 10.08.2018)

REST-API

REST ist die Abkürzung für *Representational State Transfer* und ist eine Schnittstelle, um mittels der Paradigmen des World Wide Webs miteinander zu kommunizieren. Über REST werden verteilte Systeme in Form von Clients und Server miteinander verbunden, Abfragen durchgeführt und Daten ausgetauscht. Zum Datenaustausch wird JSON oder XML genutzt und über die Netzwerkadresse (*Uniform Resource Identifier: URI*) können mittels HTTP/S-Befehle gesendet und verarbeitet werden. REST ermöglicht somit die Anbindung von Software Applikation mit der Applikation eines Servers.

¹ <https://insights.stackoverflow.com/survey/2018/>, 10.08.2018

Eine API (Application Programming Interface) dient der automatisierten und nutzerseitig flexiblen Verarbeitung der Daten eines Softwareprogramms. Die API kann entweder intern (d.h. die Erstellung bzw. Ausführung der automatisierten Verarbeitung erfordert die Verwendung des Softwareprogramms) oder extern (für die Erstellung bzw. Ausführung der automatisierten Datenverarbeitung ist das Softwareprogramm selbst nicht erforderlich) umgesetzt sein. REST-APIs sind externe API-Umsetzungen, wodurch die Daten eines Softwareprogramms automatisiert von außen bzw. mittels eines anderen Softwareprogramms angesteuert werden können. (<https://www.cloudcomputing-insider.de/was-ist-eine-rest-api-a-611116/>, 10.08.2018)

Jena

Apache Jena ist ein offenes Java-Framework, das die Anwendung von Semantic-Web-Technologien und Linked Data unterstützt. Es besteht aus mehreren Teilen, die für die Verarbeitung verschiedener Ebenen des Semantic Webs eingebunden werden können. Im Rahmen von SolConPro wurden die Komponenten *RDF API*, *Fuseki*, *TDB* und *ARQ* verwendet. Die RDF-API dient der Unterstützung beim Einlesen, Erstellen und Serialisieren von Graphen im *Resource Description Framework* (RDF) und ARQ zum Abfragen und Manipulieren solcher Daten mittels der Abfragesprache SPARQL (*SPARQL Protocol And RDF Query Language*). Für die Datenhaltung wurden Fuseki und TDB verwendet. Fuseki stellt einen SPARQL-Endpunkt zur Verfügung, der über HTTP angesprochen werden kann und eine REST-Schnittstelle besitzt. Die zugrundeliegenden Daten des Fuseki-Servers wurden über eine TDB (*Triple Database*) persistent abgelegt. (<https://jena.apache.org/>, 10.08.2018)

JSON

JavaScript Object Notation (kurz: JSON) ist ein menschenlesbarer Datenaustausch Standard. Basierend auf einer Untermenge des JavaScript Standards ist das JSON Format jedoch Programmiersprachen unabhängig und kann durch verfügbare Parser in den verschiedenen Sprachen einfach generiert und eingelesen werden. (<https://www.json.org/>, 10.08.2018)

Turtle

Turtle (TTL) ist eine Serialisierung für *Resource Description Framework* (RDF)-Graphen, die von der Abfragesprache SPARQL interpretiert werden kann und von Jena unterstützt wird. (<https://www.w3.org/TR/2014/REC-turtle-20140225/>, 10.08.2018)

WKT

Well Known Text (WKT) ist eine Markup-Sprache, die derzeit hauptsächlich für die Abbildung von Geodaten oder Vektorgeometrien verwendet wird. Das Prinzip hinter der Sprache ist, dass komplexe Objekte nach einem vorher definierten Schema, das sich aus Zeichenketten und Trennzeichen zusammensetzen kann, als Zeichenkette (String) serialisiert werden (z.B. POINT (30 10)). (ISO 19162:2015, "Geographic information – Well-known text representation of coordinate reference systems")

GoJS

GoJS ist eine Bibliothek, die es ermöglicht, auf einfache und anwenderfreundliche Weise (per „drag & drop“) interaktive und auf Graphen basierende Diagramme zu erstellen. Programmiert ist GoJS in JavaScript, sodass es direkt über HTML in Webseiten eingebunden werden und damit von aktuellen Internetbrowsern interpretiert werden kann. Bei der Anwendung, also dem Erstellen von Diagrammen, werden automatisch im Hintergrund alle Verknüpfungen aller Elemente gespeichert. Der so entstehende Graph kann im JSON-Format zurückgegeben und weiterverarbeitet werden. (<https://gojs.net>, 10.08.2018)

Formate der Organisation buildingSMART:

IFC

Die Industry Foundation Classes sind ein Dateiformat zur Standardisierung der Informationsübertragung vorrangig geometrischer 3D-Inhalte zwischen BIM-Softwareprogrammen. Das Ziel besteht in der Vermeidung aufwendiger bilateraler Absprachen zu Objekthierarchien und Begriffsdefinitionen zugunsten eines standardisierten Austauschformats. Die Entwicklung von IFC begann im Jahr 2000. Seit 2008 ist IFC unter der ISO-Norm 16739 registriert. (<http://www.buildingsmart-tech.org/specifications/ifc-overview>, 10.08.2018)

MVD

Model View Definitions stellen Festlegungen von Teildatenmengen einer IFC-Beschreibung dar, die eine Filterung von deren Inhalten für bestimmte Anwendungsfälle erlauben soll. buildingSMART reagiert mit der MVD-Konzeption auf den Einwand der Informationsflut, die aufgrund der Komplexität von BIM-Gebäudemodellen entsteht und Übertragungsprozesse von einer BIM-Software zur anderen schwierig bis unmöglich macht. Um aufwendige Software-Entwicklungen zu vermeiden, wird konzeptionell eine Filterung, d.h. MVD-Anwendung, während des IFC-Exports angestrebt. Dabei werden die Filtermöglichkeiten auf die Auswahl von IFC-Klassen beschränkt. (<http://www.buildingsmart-tech.org/specifications/ifc-view-definition>, 10.08.2018)

IDM

Mit den *Information Delivery Manuals* hat buildingSMART einen ISO-Standard geschaffen, der eine standardisierte Beschreibung von Bauprozessen anstrebt. IDMs bestehen aus drei Teilen. Die Functional Parts (FPs) definieren die durchzuführende Aktivität, die Process Maps (PMs) standardisieren mittels BPMN-Diagramm die zur Zielsetzung führende sequentielle Prozessabfolge, und die Exchange Requirements (ERs) definieren via MVDs die zur Durchführung der Aktivität notwendige Informationsslage zu Beginn und diejenige nach Abschluß der Prozesskette. (<http://iug.buildingsmart.org/idms/>, 10.08.2018)

IDM/MVD-Methode

Die Anwendung der Formate MVD und IDM führt zu einer sequentiellen Abfolge von Softwareprogrammen ohne Trennung der Datenhaltung von den Applikationen. Diese Methode steht im Gegensatz zu Technologien, die eine parallele Bearbeitung einer softwareübergreifenden Datenmenge erlauben.

bSDD

Das *buildingSMART data dictionary* (kurz: bSDD) ist ein von der Organisation buildingSMART bereitgestelltes Ontologie basiertes Wörterbuch. Dieses umfasst in verschiedenen Sprachen Definitionen zu Objekten aus dem Bereich der Bauindustrie. Diese werden hierbei so hinterlegt, dass eine eindeutige jedoch sprachenunabhängige Identifikation des Objektes und seiner Attribute möglich ist. Die Identifikation der Begrifflichkeiten erfolgt durch eigene Globally Unique Identifier (GUID). Die Übersetzung eines Begriffs in eine bestimmte Sprache kann auf Grundlage dieser Datenbasis und mithilfe der GUID aus dem bSDD über eine definierte REST Schnittstelle abgerufen werden. (<http://bsdd.buildingsmart.org/>, 10.08.2018, <https://www.buildingsmart.org/standards/standards-tools-services/data-dictionary/>, 10.08.2018)

Anhang B: Übersicht der implementierten Datenverarbeitungsmodule

Anhang B: Übersicht
der implementierten
Datenverarbeitungs-
module

Datatypes

Name	Description	Color	Shape
IFC2x3 tc1	IFC2x Edition 3 Technical Corrigendum 1	blau	Circle
IFC4	Industry Foundation Classes Release 4	rot	Circle
TXT	Textdatei	grün	Diamond
RAD	Radiance Dateiformat	gelb	Diamond
RFA	Revit Family	black	Spade
JSON	Datenaustauschformat	schwarz	Triangle
TTL	Turtle - allgemeines Ontologiedatenformat	blau	Rectangle
TTL	Turtle - SolConPro-Schema	rot	Rectangle
lang	Sprachdateien	rot	Herz
Meteo	Wetterdaten	grau	Triangle

Filters

Name	Description	Datatype	FilterType
F_PD1	PD: Geometrie entfernen	TTL_SCP	Semantic
F_PD2	PD: Verschaltung entfernen	TTL_SCP	Semantic
F_PD3	PD: Konstruktion entfernen	TTL_SCP	Semantic
F_PD4	PD: Attributs-Hierarchie plätten	TTL_SCP	Semantic
F_PD5	PD: Parametrik entfernen	TTL_SCP	Semantic
F_PD6	PD: Nicht-konkrete Attributswerte entfernen	TTL_SCP	Semantic
F_PD7	PD: Units als bSDD-GUID Attributen hinzufügen	TTL_SCP	Semantic
F_PD8	PD: Attribute entfernen	TTL_SCP	Semantic
F_PD9	PD: Attribute, die für IV Berechnung irrelevant sind, entfernen	TTL_SCP	Semantic
F_PD10	PD: Aus IV Kurve konkrete Attribute ermitteln	TTL_SCP	Semantic
F_PD11	PD: Attribute, die für Stromkreisberechnung irrelevant sind, entfernen	TTL_SCP	Semantic
F_PD12	PD: Attribute, die für Schmidtsauer Berechnung irrelevant sind, entfernen	TTL_SCP	Semantic
F_PD13	PD: Attribute, die für ACSIM Berechnung irrelevant sind, entfernen	TTL_SCP	Semantic
F_PD14	PD: Attribute, die nicht direkt am Produkt hängen, entfernen	TTL_SCP	Semantic
F_PD15	PD: Attribute, die nicht für die Standard-Produktsuche relevant sind, entfernen	TTL_SCP	Semantic
Meteodaten reduzieren	Entfernt 1. und letzte Spalte ("year" und "diffuse-horizontal irradiance [W/m^2]")	Meteo	Semantic

Filterpackages

Name	Description	Filterpackage Datatype	Filters
FP_Revit1	Entfernt Geometrie, Verschaltung und plättet die Attributs-Hierarchie	TTL	F_PD1, F_PD2, F_PD4
FP_Revit2	Entfernt Konstruktion, Parametrik und nicht-konkrete Attributswerte	TTL	F_PD3, F_PD5, F_PD6
FP_CMO-Viewer	Entfernt Verschaltung, Konstruktion, Parametrik und Attribute	TTL	F_PD2, F_PD3, F_PD5, F_PD8
FP_ds2dmCalculation	Entfernt Attribute, die für IV Berechnung irrelevant sind und ermittelt konkrete Werte aus IV Kurve	TTL	F_PD9, F_PD10
FP_ComponentConnection	Entfernt Parametrik und Attribute	TTL	F_PD5, F_PD8
FP_Search	Rückgabe für Suchanfragen zur Listendarstellung	TTL	F_PD1, F_PD2, F_PD5, F_PD14, F_PD15
FP_ProductAttributeProcessing	Entfernt Geometrie, Verschaltung, Parametrik und Attribute, die nicht direkt am Produkt hängen.	TTL	F_PD1, F_PD2, F_PD5, F_PD14
FP_SingleProduct	Entfernt Geometrie, Verschaltung, Parametrik und nicht-konkrete Attributswerte	TTL	F_PD1, F_PD2, F_PD5, F_PD6

Wrappers

Name	Description	Datatype Input	Datatype Output
ttl2json_singleProduct	Umwandlung von TTL mit einem Product zu json	TTL	JSON
ttl2json_multipleProducts	Umwandlung von TTL mit mehreren Product zu json	TTL	JSON
ttl2txt	Übersetzt Attribute aus TTL Format in TXT Format	TTL	TXT
ifc2rad	Umwandlung IFC-Datei zu RAD Datei	IFC	RAD
ttl2txt_ec	Umwandlung der Verschaltung von TTL in TXT	TTL	TXT

Anhang B: Übersicht
der implementierten
Datenverarbeitungsmodu-
le

Workers

Name	Description	Inputs	Outputs
bsDDTranslator	Fügt Produktdaten (single oder multiple) bsdd Übersetzung hinzu (Name und Unit)	JSON, language	JSON
bsDDTranslatorForRevit	Fügt Produktdaten (single oder multiple) bsdd Übersetzung hinzu (Name und Unit) für Revit Import	JSON, language	JSON
bsDDTranslatorForBIPV	Fügt BIPV-Datei bsdd Übersetzung hinzu (Name und Unit)	TXT, language	TXT
MultipleProductsFromViKoLink	Fragt alle Produkt, die Suchkriterien entsprechen aus der angegebenen ViKoDB ab und gibt es als TTL zurück	JSON (ViKoLink-Anfrage)	TTL
SingleProductFromViKoLink	Fragt ein bestimmtes Produkt aus der angegebenen ViKoDB ab und gibt es als TTL zurück	JSON (ViKoLink-Anfrage)	TTL
RegistrationForViKoLink	Registriert ViKoDB bei ViKoLink	JSON (ViKoLink-Anfrage)	TXT

RBVOBW-Sets

Name	Description	Inputs	Filterpackages	Workers	Wrappers	Outputs
ds2dmCalculation	Stellt notwendige Informationen für die ds2dm-Berechnung zur Verfügung	TTL - SolConPro-Schema (Modul), language	FP_ProductAttributeProcessing, F_PD3, FP_ds2dmCalculation	bsDDTranslatorForBIPV	ttl2txt	TXT
ElectricalConnections	Bereitet die Verschaltung der Module für die Weiterverarbeitung vor	TTL - SolConPro-Schema (Modul)	FP_ComponentConnection	-	ttl2txt_ec	TXT
ProductProposal	Frägt Liste aller den Suchkriterien entsprechenden Produkte ab	JSON (Suchanfrage)	-	MultipleProductsFromViKoLink	-	TTL - SolConPro-Schema
ProcessSearchResultsForRevit	Gibt ausgewählte Liste der Suchergebnisse in gewählter Sprache zurück (für Revit)	TTL - SolConPro-Schema, language	FP_Search, F_PD3, F_PD6	bsDDTranslatorForRevit	ttl2json_multipleProducts	JSON
ProcessSearchResults	Gibt ausgewählte Liste der Suchergebnisse in gewählter Sprache zurück	TTL - SolConPro-Schema, language	FP_Search, F_PD3, F_PD6	bsDDTranslator	ttl2json_multipleProducts	JSON
RequestSingleProduct	Gibt das durch URI&Endpoint definierte Produkt zurück	JSON (Suchanfrage)	-	SingleProductFromViKoLink	-	TTL - SolConPro-Schema
RegisterViKoDB	Registriert eine neue ViKoDB in der ViKoLink	JSON (Registrierung)	-	RegistrationForViKoLink	-	TXT mit Statusmeldung
ProcessProductForRevit	Gibt ausgewähltes Produkt der Anfrage in gewählter Sprache zurück	TTL - SolConPro-Schema, language	FP_Revit1, FP_Revit2	bsDDTranslatorForRevit	ttl2json_singleProduct	JSON
CMOViewer	Bereitet die Daten zur Abbildung im CMO Viewer vor	TTL - SolConPro-Schema	FP_CMO-Viewer	-	-	TTL - CMO-Schema
ECCalculation	Stellt notwendige Informationen für die Berechnung des Stromkreislaufes in gewählter Sprache zur Verfügung	TTL - SolConPro-Schema (WR), language	FP_ProductAttributeProcessing, F_PD3, F_PD11, F_PD6	bsDDTranslatorForBIPV	ttl2txt	TXT
SchmidtSauerCalculation	Stellt notwendige Informationen für die SchmidtSauer-Berechnung in gewählter Sprache zur Verfügung	TTL - SolConPro-Schema (WR), language	FP_ProductAttributeProcessing, F_PD3, F_PD12, F_PD6	bsDDTranslatorForBIPV	ttl2txt	TXT
ACSIMCalculation	Stellt notwendige Informationen für die ACSIM-Berechnung in gewählter Sprache zur Verfügung	TTL - SolConPro-Schema (WR), language	FP_ProductAttributeProcessing, F_PD3, F_PD13, F_PD6	bsDDTranslatorForBIPV	ttl2txt	TXT
ProcessProduct	Gibt ausgewähltes Produkt der Anfrage in gewählter Sprache zurück	TTL - SolConPro-Schema (WR), language	FP_SingleProduct	bsDDTranslator	ttl2json_singleProduct	JSON

**Anhang B: Übersicht
der implementierten
Datenverarbeitungsmodu-
le**

Anhang C: Beispiel für API-Dokumentation

Da die Vollständigkeit der API-Dokumentation für die firmenübergreifende Zusammenarbeit von erheblicher Bedeutung ist, ist hier das Beispiel /api/create POST aus dem Kollisionsmanagementsystem beschrieben:

/api/create POST

Usage: To add a new clash or comment

Requirements: Body with clash or comment data, and JSON web Token (JWT) in the Authorization header

Body Example

1. Clash:

```
{
  "a": "string; assignment of a collision to a responsible team as
    agreed at project start",
  "au": "string; login name on operation system level",
  "cd1": "string; class display name 1",
  "cd2": "string; class display name 2",
  "cn": "string; clash name (from Navisworks Manage)",
  "d1": "string; name of the first colliding object (from Revit)",
  "d2": "string; name of the second colliding object (from Revit)",
  "f1": "string; name of the first procedural model",
  "f2": "string; name of the second procedural model",
  "g": "string; group name (from Navisworks Manage)",
  "i1": "string; first object ID from Revit",
  "i2": "string; second object ID from Revit",
  "l1": "string; level of first object (from Revit)",
  "l2": "string; level of second object (from Revit)",
  "s": "string; status of collision (new/active/solved/etc.)",
  "t": "string; name of the test run (e.g. support structure plan-
    ning)",
  "x": "float; x coordinate of collision (Revit coordinate system)",
  "y": "float; y coordinate of collision (Revit coordinate sys-
    tem)",
  "z": "float; z coordinate of collision (Revit coordinate sys-
    tem)",
  "c_id": "string; ID of collision (Navisworks)",
  "ot": "string; object type",
  "p_id": "string; project ID (from Revit)",
  "d": "date; time when collision was detected (from Navisworks Man-
    age)"
}
```

2. Comment:

```
{
  "au": "string; author of comment",
  "c_id": "string; ID of collision (Navisworks)",
  "p_id": "string; project ID (from Revit)",
  "ot": "string; comment itself"
}
```

Response on success: {}

CURL Example:

```
curl -X POST -H "Content-Type: application/json" -H "Authorization:
Bearer JWT" -d '{"a": "you", "au": "name", "cd1": "display name 1",
"cd2": "display name 2", "cn": "clash name yo", "d1": "display name
yo", "d2": "display name2 yo", "f1": "file name yo", "f2": "file name2
yo", "g": "group name yo", "i1": "revit id yo", "i2": "revit id2 yo",
"l1": "level 1 yo", "l2": "level 2 yo", "s": "this is the status",
"t": "test name", "x": "100", "y": "10", "z": "120", "c_id": "some random
clash id", "ot": "clash", "p_id": "8a3e1695cb2c80745eb2",
"d": "1191890"}' localhost:3006/api/create
```

Es gilt zu beachten, dass eine API-Dokumentation nur dann als vollständig zu erachten ist, wenn keine Rückfragen an den Softwarehersteller erforderlich sind. Hierfür muss sowohl die Beschreibung der Endpunkte als auch die Schemadefinition

(inkl. Begriffsbeschreibungen) eindeutig sein. Für die Koordinaten x , y , z aus dem oben genannten Beispiel ist deshalb die Angabe des Referenzsystems erforderlich.