Erschließung domänenübergreifender Informationsräume mit Multimodellen

Sebastian Fuchs

Schriftenreihe des Instituts für Bauinformatik
Herausgeber: Prof. Dr.-Ing. Raimar J. Scherer

ISBN: 978-3-86780-451-6

Institut für Bauinformatik, TU Dresden

Postanschrift:	Besucheranschrift:
Technische Universität Dresden	Nürnberger Str. 31a
01062 Dresden	2. OG, Raum Nr. 204
	01187 Dresden

Tel.: +49 351 / 463-32966
Fax: +49 351 / 463-33975
E-Mail: Raimar.Scherer@tu-dresden.de
WWW: http://cib.bau.tu-dresden.de

Erschließung domänenübergreifender Informationsräume mit Multimodellen

Access of cross-domain information spaces using multi-models

An der Fakultät Bauingenieurwesen der Technischen Universität Dresden
zur Erlangung des akademischen Grades eines Doktor-Ingenieurs (Dr.-Ing.)
genehmigte

Dissertation

vorgelegt von
Dipl.-Ing. Sebastian Fuchs
geboren am 26. Februar 1977 in Berlin

Erster Gutachter: Prof. Dr.-Ing. Raimar J. Scherer (TU Dresden)
Zweiter Gutachter: Prof. Dr.-Ing. André Borrmann (TU München)

Tag der Einreichung: 24. Februar 2014
Tag der Verteidigung: 02. Juli 2015

Vorwort

Wege entstehen dadurch, dass man sie geht.

(Franz Kafka)

Als ich im Juli 2008 das erste mal bei Herrn Professor Dr. Raimar J. Scherer vorsprach, lag mein Weimarer Ingenieurstudium bereits einige Jahre hinter mir. Ich hatte mich inzwischen der Softwareentwicklung verschrieben und bei der Firma TragWerk eine interessante Aufgabe. Es mussten Industrie-Forschungsprojekte durchgeführt und die Ergebnisse in Softwareprodukte umgesetzt werden. Die konzeptionellen und methodischen Anforderungen bewegten mich dazu, mir dahingehend noch einmal grundlegende Fähigkeiten anzueignen.

Meinem Wunsch nach einer externen Promotion an der TU Dresden begegnete Professor Scherer mit einem weisen Beschluss. Ich durfte ab April 2009 abwechselnd eine Woche bei TragWerk und eine Woche als wissenschaftlicher Mitarbeiter am Institut für Bauinformatik arbeiten. Diese Integration in den universitären Alltag stellte sich als unentbehrlich heraus. Neben einem Überblick über die Bauinformatik bekam ich so die Möglichkeit – und Pflicht – zu kreativer Tätigkeit, zum Publizieren sowie zu Austausch und Kritik mit anderen Forschern.

Für die inhaltliche Arbeit erhielt ich, auch durch meine Kollegen Dr. Peter Katranuschkov und Sven-Eric Schapke, viele Freiheiten. Eine nicht immer einfache Voraussetzung. Die nach einem Grundsatz von Professor Scherer eigenständige Suche nach einem Dissertationsthema empfinde ich nach wie vor als größte Herausforderung – aber auch als größte Erfüllung. So entstand die ursprüngliche Idee zum Multimodell eher naiv als randständige, technische Hilfestellung eines eigentlich anderen Problems. Dass sich daraus eine eigenständige Methode entwickeln ließ, ist dieser gewährten Freiheit zuzuschreiben.

Ein ebensolches Vertrauen wurde mir auch von TragWerk entgegengebracht. So durfte ich meine Arbeit nicht nur eigenverantwortlich ausgestalten, auch deren Umsetzung konnte ich über viele Jahre meinen zeitlichen Erfordernissen anpassen. Viel mehr noch standen mir meine Chefs Dr. Frank Purtak und Wolfgang Döking immer als kompetente Diskussionspartner zur Seite.

Die parallele Tätigkeit in Forschung und Praxis empfand ich von Anfang an gewinnbringend. Der permanente Perspektivwechsel half dabei, mein Wissen im jeweils anderen Gebiet anzuwenden. Jedoch musste dieser Vorteil mit einem hohen Aufwand bezahlt werden. Zwei halbe Stellen beanspruchen mehr als eine ganze – so waren Wochen mit 50 bis 60 Arbeitsstunden plus Lesen und Schreiben in der Freizeit die Regel. Dies spürte allen voran meine Familie, deren Verzicht oftmals größer als meiner war. Dennoch unterstützte sie mich bei meinem Vorhaben mit allen Kräften. Dafür möchte ich mich aus tiefstem Herzen bedanken. Mein Dank gilt gleichermaßen all den bereits genannten Menschen sowie meinen Freunden und Kollegen, die durch ihr Vertrauen, ihre Motivation, Förderung und Unterstützung diese Arbeit mit ermöglicht haben.

Dresden, im August 2015
Sebastian Fuchs

Kurzfassung

Mit dem Übergang von bauwerksorientierter zu prozessorientierter Arbeitsweise erlangt die domänenübergreifende Bereitstellung von Informationen wachsende Bedeutung. Das betrifft bspw. die Erstellung von Controlling-Kennwerten, die Vorbereitung von Simulationen oder die Betrachtung neuer Aspekte wie Energieeffizienz. Aktuelle Datenformate und Erschließungsmethoden können diese Herausforderung jedoch nicht befriedigend bewältigen. Daher bedarf es einer Methode, welche interdisziplinäre Bauinformationsprozesse uneingeschränkt ermöglicht. Vorhandene Kommunikationsprozesse und Fachanwendungen sollen dabei beibehalten und weitergenutzt werden können.

Mit der Multimodell-Methode wird ein Lösungsansatz für die strukturellen Probleme interdisziplinärer Bauinformationsprozesse vorgestellt. Multimodelle bündeln heterogene Fachmodelle unterschiedlicher Domänen und erlauben die Verbindung ihrer Elemente in externen, ID-basierten Linkmodellen. Da die Fachmodelle unberührt bleiben, wird auf diesem Weg eine lose und temporäre Kopplung ermöglicht. Durch den Verzicht auf ein führendes oder integrierendes Datenschema werden keine Transformationsprozesse benötigt, können etablierte und heute übliche Datenformate weitergenutzt und die verlinkten Fachmodelle neutral ausgetauscht werden.

Die in Multimodellen verknüpften Daten bieten einen informationellen Mehrwert gegenüber alleinstehenden Fachmodellen. Zusammengehörende Informationen können über die persistenten Links automatisch ausgewertet werden, anstelle manuell vom Menschen immer wieder flüchtig neu zugeordnet werden zu müssen. Somit erscheint ein Multimodell gegenüber einem Benutzer wie ein einziger abgeschlossener Informationsraum.

Um solche datenmodell-, datenformat- und domänenübergreifenden Informationsräume komfortabel erstellen und filtern zu können, wird die deklarative Multimodell-Abfragesprache MMQL eingeführt. Diese erlaubt einen generischen Zugriff auf die Originaldaten und bildet die Kernkonzepte der Multimodell-Erschließung – mehrwertige Linkerzeugung und strukturelle Linksemantik – ab. Ein zugehöriger Interpreter ermittelt den Lösungsweg für konkrete Anweisungen und führt diesen auf realen Daten aus.

Die Umsetzung und Bereitstellung der Konzepte als IT-Komponenten auf verschiedenen Ebenen – von der Datenstruktur über Bibliotheken und Services bis hin zur alleinstehenden, universellen Multimodell-Software M2A2 – erlaubt die sofortige und direkte Anwendung der Multimodell-Methode in der Praxis.

Abstract

With the transition of building-oriented to process-oriented work, the provision of cross-domain information gained growing importance—for example in the creation of controlling parameters, the preparation of simulations or when considering new aspects such as energy efficiency. However, current data formats and access methods cannot cope with this challenge satisfactory. Therefore, a method is required, that enables interdisciplinary construction information processes fully. Thereby existing communication processes and domain applications have to be retained and continued to be used as possible.

With the multi-model method, an approach to structural problems of such interdisciplinary construction information processes is presented. Multi-models combine heterogeneous models of different domains and allow the connection of their elements in external ID-based link models. As the domain models remain unaffected, a loose and temporary coupling is possible in this way. By not using a leading or integrating data schema, no transformation processes are required, common established data formats can be retained and the linked domain models can be exchanged neutrally.

The linked data in multi-models offer an additional value of information over single domain models. Information belonging together can be automatically evaluated by the persistent links—instead of being repeatedly reassigned by people in a volatile way. Thus, a multi-model appears to a user as a single self-contained information space.

In order to create and filter such cross-format and cross-domain information spaces comfortably, the declarative multi-model query language MMQL is introduced. It allows for generic access to the original data and integrates the core concepts of the multi-model development—n-ary link generation and structural link semantics. An associated interpreter determines the approach for specific instructions and executes it on real data.

The implementation and deployment of the concepts as IT components at various levels—from the data structure via libraries and services, to the universal multi-model software M2A2—allows an immediate and direct application of the multi-model method in practice.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

Verzeichnis der MMQL-Syntaxdiagramme

Abkürzungsverzeichnis

AEC . . . . . Architecture, Engineering, Construction
AG . . . . . Auftraggeber
AN . . . . . Auftragnehmer
API . . . . . Application Programming Interface
ASCII . . . . . American Standard Code for Information Interchange
BGL . . . . . Baugeräteliste
BI . . . . . Bauinformatik
BIM . . . . . Building Information Model / Building Information Modelling
BPMN . . . . . Business Process Model and Notation
CAD . . . . . Computer Aided Design
CIS/2 . . . . . CIMSteel Integration Standards
CityGML . . . . . City Geography Markup Language
CPU . . . . . Central Processing Unit
CRUD . . . . . Create Read Update Delete
CSV . . . . . Comma Separated Values
DMS . . . . . Document Management System
DOM . . . . . Document Object Model
DSL . . . . . Domain Specific Language
DV . . . . . Datenverarbeitung
DXF . . . . . Drawing Interchange File Format
EBNF . . . . . Erweiterte Backus-Naur-Form
EFH . . . . . Einfamilienhaus
EM . . . . . Elementarmodell
EMF . . . . . Eclipse Modeling Framework
EQL . . . . . EXPRESS Query Language
ER . . . . . Entity-Relationship
ERM . . . . . Entity-Relationship-Modell
FEM . . . . . Finite-Elemente-Methode
GAEB . . . . . Gemeinsamer Ausschuss Elektronik im Bauwesen
gbXML . . . . . Green Building XML
GMSD . . . . . Generalized Model Subset Definition Schema
GUI . . . . . Graphical User Interface
HOAI . . . . . Verordnung über die Honorare für Architekten- und Ingenieurleistungen
HTTP . . . . . Hypertext Transfer Protocol
HUTN . . . . . Human-Usable Textual Notation
HVAC . . . . . Heating, Ventilation, Air Conditioning
ID . . . . . Identifikator

IFC . . . . . Industry Foundation Classes
IFD . . . . . International Framework for Dictionaries
IKT . . . . . Informations- und Kommunikationstechnologie
INI . . . . . Initialisierungsdatei (Format)
ISYBAU . . . . . Integriertes DV-System-Bauwesen
IT . . . . . Informationstechnologie
JSON . . . . . Javascript Object Notation
JVM . . . . . Java Virtual Machine
KML . . . . . Keyhole Markup Language
LDAP . . . . . Lightweight Directory Access Protocol
LDIF . . . . . LDAP Data Interchange Format
LINQ . . . . . Language Integrated Query
LM . . . . . Linkmodell
LoD . . . . . Level of Detail
LV . . . . . Leistungsverzeichnis
M2A2 . . . . . Multi-Model Assembly and Analyzing Platform
MDA . . . . . Model Driven Architecture
MDSD . . . . . Model Driven Software Development
MIME . . . . . Multipurpose Internet Mail Extensions
MM . . . . . Multimodell
MMC . . . . . Multimodell-Container
MMQL . . . . . Multi-Model Query Language
MOF . . . . . Meta Object Facility
MVC . . . . . Model View Controller
OCL . . . . . Object Constraint Language
OG . . . . . Obergeschoss
OMG . . . . . Object Management Group
OSGi . . . . . Open Services Gateway initiative
OWL . . . . . Web Ontology Language
PMQL . . . . . Partial Model Query Language
QM . . . . . Qualitätsmanagement
QVT . . . . . Query View Transformation
RCP . . . . . Rich Client Platform
RDB . . . . . Relationale Datenbank
RDF . . . . . Resource Description Framework
REST . . . . . Representational State Transfer
SDAI . . . . . Standard Data Access Interface
SPARQL . . . . . SPARQL Protocol And RDF Query Language
SPF . . . . . STEP Physical File
SQL . . . . . Structured Query Language
STEP . . . . . Standard for the Exchange of Product model data
UI . . . . . User Interface
UML . . . . . Unified Modeling Language
URI . . . . . Uniform Resource Identifier

URL	Uniform Resource Locator
VDI	Verein Deutscher Ingenieure
VOB	Vergabe- und Vertragsordnung für Bauleistungen
WWW	World Wide Web
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XSD	XML Schema Definition

Verwendete Symbole und Notationen

Mathematische und logische Symbole

Die in dieser Arbeit verwendeten mathematischen und logischen Symbole richten sich nach DIN 1302 (1999) sowie DIN 5473 (1992). Die wesentlichen Zeichen und Begriffe sind:

$:=$	Definition
$:$	es gilt
$\mid$	mit, unter der Bedingung dass ...
$()$	Vorrang
$\Rightarrow$	Implikation
$\Leftrightarrow$	Äquivalenz
$\neg$	Negation
$\wedge$	Konjunktion
$\vee$	Disjunktion
$\oplus$	Kontravalenz
$\forall$	Allquantor
$\exists$	Existenzquantor
$\{e_1, \ldots, e_n\}$	Menge mit den Elementen $e_1, \ldots, e_n$
$(r_1, \ldots, r_n)$	Relation mit den Relata $r_1, \ldots, r_n$
$\emptyset$, $\{\}$	Leere Menge
$\subseteq$	Teilmenge
$\subset$	Echte Teilmenge
$\supseteq$	Obermenge
$\supset$	Echte Obermenge
$\cup$	Vereinigungsmenge
$\cap$	Schnittmenge
$\in$	Element
$\notin$	Kein Element
$\lvert A \rvert$	Kardinalität der Menge A
$f\,(x)$	Funktion f mit dem Argument x
$a \mapsto b$	Abbildung von a auf b
$\times$	Kreuzprodukt
$=$	ist gleich
$\neq$	ungleich
$<$	kleiner als
$\leq$	kleiner gleich
$>$	größer als
$\geq$	größer gleich

Bezeichner der Mengen und Elemente

Bezeichner mit Großbuchstaben stellen Mengen dar, Kleinbuchstaben beziehen sich auf Elemente dieser Mengen. Zur besseren Unterscheidung werden Mengen und Elemente von MMQL-Anweisungen in `Schreibmaschinenschrift` dargestellt.

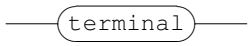
`addl`, `ADDL` . . . . . AddLinks-Klausel
`assrt`, `ASSRT` . . . . . Assertion-Klausel (Kriterium)
e, E . . . . . Multimodell-Element
em, EM . . . . . Elementarmodell
`ems`, `EMS` . . . . . Elementarmodell-Spezifikationsklausel
emt, EMT . . . . . Elementarmodell-Typ
et, ET . . . . . Element-Typ
`from`, `FROM` . . . . . From-Klausel
id, ID . . . . . Identifikator (ID)
kv, KV . . . . . Kontrolliertes Vokabular
l, L . . . . . Link
lm, LM . . . . . Linkmodell
le, LE . . . . . Verlinktes Element
`lws`, `LWS` . . . . . Linkedwith-Spezifikationsklausel
md, MD . . . . . Metadatum / Metadaten
mm, MM . . . . . Multimodell
`mmbe`, `MMBE` . . . . . MultiModelByEditor-Klausel
`mmbf`, `MMBF` . . . . . MultiModelByFile-Klausel
`mmql`, `MMQL` . . . . . MMQL-Anweisung
`mms`, `MMS` . . . . . Multimodell-Spezifikationsklausel
$mod_{\mathrm{ARI}}, MOD_{\mathrm{ARI}}$ . . . . . Modus der Arität
$mod_{\mathrm{DUP}}, MOD_{\mathrm{DUP}}$ . . . . . Modus der Duplizität
$mod_{\mathrm{EK}}, MOD_{\mathrm{EK}}$ . . . . . Modus der Elementkombination
$mod_{\mathrm{KARD}}, MOD_{\mathrm{KARD}}$ . . . . . Modus der Kardinalität
$mod_{\mathrm{LI}}, MOD_{\mathrm{LI}}$ . . . . . Modus der Linkinterpretation
p, P . . . . . Property
pm, PM . . . . . Modelleigenschaft
`prop`, `PROP` . . . . . Property-Spezifikationsklausel
s, S . . . . . Propertyschlüssel
`slct`, `SLCT` . . . . . Select-Klausel
`stmt`, `STMT` . . . . . Statement-Klausel
t, T . . . . . Terminus
v, V . . . . . Value

Bedeutung der Indizes

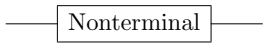
i . . . . . Spalte des ResultSets
j . . . . . Zeile des ResultSets
k . . . . . Kombination von Multimodell-Elementen
l . . . . . Multimodell-Element der rechten Seite im Iterationsschritt q
m . . . . . Anzahl von Elementarmodellen

n . . . . . Anzahl allgemein
o . . . . . Spalte des Link-Sets im Iterationsschritt q
p . . . . . Nach Element-Typ einwertiger Link, Bestandteil von r
q . . . . . Iterationsschritt der sukzessiven Linkauswertung
r . . Potentiell mehrwertiger Link aus den relevanten Linkmodellen im Iterationsschritt q
s . . . . . Zu verlinkender Element-Typ
t . . . . . Multimodell-Element eines Element-Typs s
u . . . . . Nach Element-Typ einwertig erzeugter Link
v . . . . . Nach Element-Typ mehrwertig gruppierter Link
w . . . . . Iterationsschritt der sukzessiven Elementgruppierung
x, y, z . . . . . Laufvariable, jedes beliebige Element

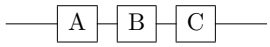
Syntaxdiagramme



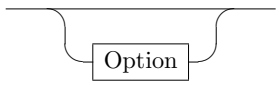
. . . . . Terminalsymbol



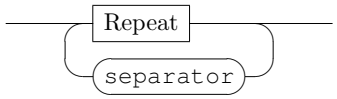
. . . . . Nonterminalsymbol



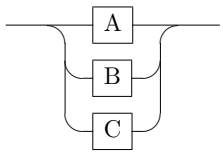
. . . . . Komplexprodukt



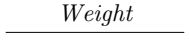
. . . . . Option



. . . . . Wiederholung

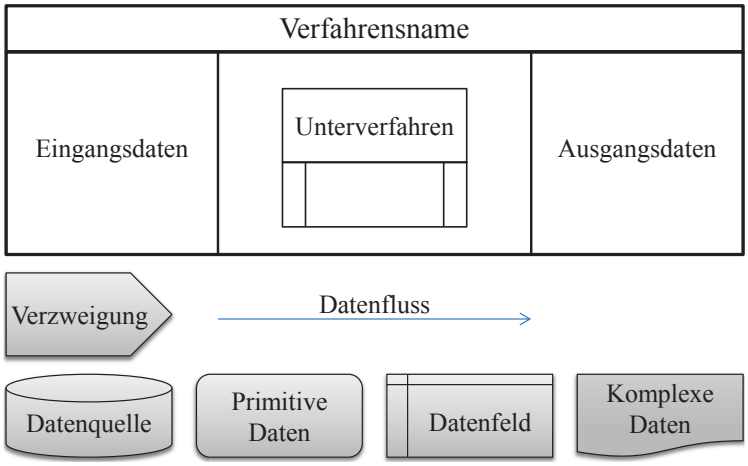


. . . . . Alternative



. . . . . Priorität

Grafische Notation der Verfahren



Kapitel 1.

Einleitung

Alles Wissen geht aus einem Zweifel hervor

(Marie Freifrau von Ebner-Eschenbach)

1.1. Motivation

Bauprojekte sind interdisziplinäre Aufgabenstellungen, bei denen eine Vielzahl von Spezialisten ein gemeinschaftliches, einzigartiges Werk erschaffen. Hierbei bilden kommunizierte, fachübergreifende Informationen die Entscheidungsgrundlage der meisten Bauplanungs-, Ausführungs- und Betriebsprozesse. Der Erzeugung, Übertragung und Erschließung solcher Informationen kommt somit ein hoher Stellenwert zu.

Die fachübergreifende Informationsunterstützung bewegt sich dabei in einem schwierigen Spannungsfeld. Auf der einen Seite werden durch den interdisziplinären Projektcharakter mit immer neuen Anwendungsgebieten – wie z. B. Energieeffizienz, Nachhaltigkeit oder Gebäudeautomation – auch *domänenübergreifende* Daten benötigt. Auf der anderen Seite basiert der Datenaustausch in der Praxis auf etablierten, heterogenen, größtenteils *domänenspezifischen* Datenformaten.

Bisherige Lösungsansätze beruhen entweder auf integrierten Systemen – bei welchen diverse Quelldaten in ein internes Format überführt werden – oder auf nativen oder erweiterten Produktdatenmodellen. Zwar eignen sich integrierte Systeme sehr gut zur Bearbeitung fokussierter Aufgabenstellungen; sie setzen jedoch in der Regel eine zentrale Datenhaltung voraus, die keinen nutzbringenden dezentralen Austausch domänenübergreifender Informationen mehr erlaubt.

Produktdatenmodelle wie IFC ermöglichen die Abbildung und den Austausch einer Vielzahl von bauprojektrelevanten Informationen in *einem* Format. Allerdings existieren noch nicht in allen unterstützten Fachbereichen genügend Softwareprodukte, welche dieses Format verarbeiten können. Außerdem sind die unterstützten Disziplinen prinzipiell beschränkt, weswegen oftmals proprietäre Erweiterungen der Produktdatenmodelle um neue Fachbereiche vorgeschlagen werden. Für eine praxisrelevante domänenübergreifende Informationsunterstützung müssen jedoch für alle beteiligten Domänen Fachanwendungen sowie etablierte Datenformate vorhanden sein.

In dieser Arbeit wird daher eine Methode vorgestellt, welche eine domänenübergreifende Informationsunterstützung auf Basis austauschbarer, verknüpfter Fachmodelle mit heterogenen Datenformaten ermöglicht.

1.2. Ausgangspunkt

Die Baubranche ist ein bedeutender Wirtschaftszweig in der Bundesrepublik Deutschland. Das Bauvolumen betrug im Jahr 2011 306,7 Mrd. € (Bundesinstitut für Bau-, Stadt- und Raumforschung BBSR, 2012a); 2012 entfielen 8,8 % des Bruttoinlandsproduktes auf Bauinvestitionen[1]. Diese Gesamtleistung wird von einer Industrie erbracht, deren Struktur von kleinen und mittleren Unternehmen geprägt ist. So gab es im Jahr 2012 insgesamt 350.000 Betriebe im Baugewerbe mit durchschnittlich 6 Beschäftigten. Davon betrug die Anzahl von Firmen mit mehr als 100 Beschäftigten 750; in der Region Ost hatten nur 8 % der Betriebe mehr als 20 Mitarbeiter (Bundesinstitut für Bau-, Stadt- und Raumforschung BBSR, 2012b).

Darüber hinaus besitzen die Bauablauf- und -managementprozesse eines Bauvorhabens eine hohe Vielfalt und Komplexität. Nicht nur das Bauwerk als Endprodukt ist ein Unikat – auch die notwendigen Produktionsprozesse, die Produktionsstätte *Baustelle* sowie die Zusammensetzung des Konsortiums, inklusive Bauherr, Planer, Ausführenden, Nutzer, Kapitalgeber und Baubehörden, sind einzigartig. Es können weder Prototypen erstellt, noch Produkt-Optimierungsmethoden wie in der stationären Großserienfertigung angewendet werden.

Die Fragmentierung der Bauindustrie, die immer neue Zusammensetzung der Konsortien sowie die Einzigartigkeit von Bauwerk und Randbedingungen wirken sich unmittelbar auf die Zusammenarbeit im Bauvorhaben aus. Das *Leitbild Bau* stellt daher auch ein Entwicklungspotential in diesem Bereich heraus:

> „Die Zukunft des Bauens liegt auch in der Optimierung der Zusammenarbeit entlang der gesamten Wertschöpfungskette. Diese Potenziale der Zusammenarbeit, die Kosten senken und Qualität verbessern können, sind bei weitem nicht ausgeschöpft. Die Schnittstellenprobleme sollen durch innovative Planungsmethoden und neue technische Lösungen verringert werden. Dies gilt auch für die Mehrzahl der Bauaufgaben, die weiterhin unter einer Trennung von Planung und Bauausführung realisiert werden. [...]
> Wegen der unterschiedlichen Größe und Komplexität der Bauprojekte und besonderer Kundenwünsche entwickeln sich unterschiedliche Kooperationsmodelle (z. B. Arbeitsgemeinschaften, Bauteam, Partnering) nebeneinander. Dabei muss sichergestellt werden, dass auch mittelständische Unternehmen gleiche Wettbewerbschancen haben." (Leitbild Bau, 2009, S. 10)

Für die Entwicklung einer neuen Methode zur domänenübergreifenden Informationsunterstützung ergibt sich daher die Notwendigkeit zur Integration in vorhandene Informations- und Kommunikationsprozesse. Der Ansatz soll getätigte Investitionen in Software und Datenstandards schützen, anstelle jene grundlegend neu zu definieren. Ausgehend von diesen Rahmenbedingungen gelten in der vorliegenden Arbeit folgende Prämissen:

1. Es gibt einen wachsenden Bedarf an modell-, format- und domänenübergreifenden, interdisziplinären Bauinformationsprozessen.
2. Domänenspezifische Informationen des Bauwesens können überwiegend mit etablierten Datenformaten abgebildet werden.
3. Mit den aktuellen Methoden und Werkzeugen der Datenmodellierung können zwar unkompliziert neue Datenformate erstellt werden. Es ist jedoch sehr aufwändig, neue,

[1] Hauptverband der Deutschen Bauindustrie e. V.; http://www.bauindustrie.de/zahlen-fakten/statistik/bedeutung-der-bauwirtschaft/anteil-am-bruttoinlandsprodukt/

geänderte oder erweiterte Datenformate innerhalb der AEC-Anwendergruppe zu etablieren. Eventuell sind dafür Standardisierungsprozesse zu absolvieren.

4. Ein globaler Konsens zur Definition und Nutzung eines s. g. Superdatenmodells, welches die Informationen aller potentiellen Planungs- und Bauprozesse abbilden kann, ist nicht möglich.
5. Eine vollständige Integration aller Bauinformationsprozesse in eine einzige logische, zentrale Ressource erfordert eine geschlossene Organisationsstruktur, wie sie in Bauprojekten nicht üblich ist. Auch in Zukunft wird daher – zumindest in der Peripherie – ein dezentraler Datenaustausch notwendig sein.
6. Datenformate haben prinzipiell eine längere Lebensdauer als die Softwareprodukte, welche diese verarbeiten können.
7. Baufachmodelle werden auch in Zukunft mit spezialisierten Fachanwendungen erzeugt und bearbeitet.
8. Aufgrund der Fragmentierung der Bauindustrie sowie der Diversität der Bauprojekte ist keine branchenweite Softwareproduktkette zur durchgehenden Begleitung der Planungs- und Ausführungsprozesse realisierbar.
9. Softwareentwicklung ist kostenintensiv. Die Neuentwicklung, Erweiterung oder Änderung von Fachanwendungen muss daher durch wiederverwendbare Softwarekomponenten unterstützt werden.

1.3. Zielsetzung

Ziel der Arbeit ist die Entwicklung einer Methode zur Erzeugung, Übertragung und Erschließung datenformat-, datenmodell- und domänenübergreifender Informationsräume.

Der in der Baupraxis vorherrschende Datenaustausch mit unabhängigen, domänenspezifischen Fachmodellen führt im Allgemeinen lediglich zu getrennten, domänenspezifischen Informationsräumen. Das automatisiert erschließbare Informationspotential beschränkt sich dabei auf die jeweiligen Grenzen der genutzten Datenformate. Um beispielsweise einen Zahlungsplan ermitteln zu können, muss ein Anwender das Leistungsverzeichnis, das Bauwerksmodell und den Vorgangsplan parallel in den entsprechenden Anwendungen öffnen (vgl. Abbildung 1.1 auf der nächsten Seite). Zwar ist es ihm dann möglich, computergestützt Fragen an das *jeweilige* Modell zu beantworten – wie der Vergleich von Einheitspreisen der Leistungspositionen, die Lokalisierung von Bauteilen oder die Bestimmung von Vorgangsdauern.

Domänenübergreifende Aufgabenstellungen – wie die Ermittlung der bei einem bestimmten Vorgang anteilig auszuführenden, bauteilbezogenen Leistungspositionen – können so jedoch nur manuell bearbeitet werden. Der Anwender muss zu diesem Zweck die impliziten Beziehungen zwischen den Elementen der getrennten Informationsräume selbst erkennen und herstellen. Im genannten Beispiel wären jeder Leistungsposition die korrespondierenden Bauteile zuzuordnen und entsprechend der definierten Vorgänge zu gruppieren[2].

Die Lösung umfangreicher Problemstellungen mit drei oder mehr Fachmodellen ist auf eine manuelle Weise nicht mehr präzise und effizient zugleich möglich. Wären die Informationsräume

[2] Das beschriebene Beispiel wird in Abschnitt 6.3 auf Seite 167 aufgegriffen und mit der Multimodell-Methode gelöst (vgl. auch Abbildung 6.6).

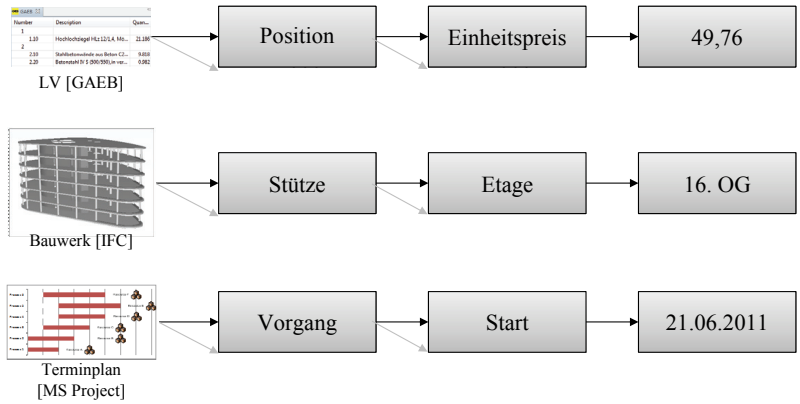


Abbildung 1.1.: Unabhängige Fachmodelle mit exemplarischen Daten, wie sie beispielsweise zur Ermittlung eines Zahlungsplans benötigt werden

nicht getrennt, sondern domänenübergreifend und singulär, ergäbe sich der folgende Mehrwert für den Anwender (vgl. Abbildung 1.2 auf der nächsten Seite):

Hypothese 1 Domänenübergreifende Informationsräume besitzen ein gesteigertes Informationspotential und die Möglichkeit zur automatisierten Bearbeitung domänenübergreifender Aufgabenstellungen gegenüber getrennten, domänenspezifischen Informationsräumen.

Wenn dem Benutzer ein Informationsraum mit verknüpften Domänen zur Verfügung steht, können interdisziplinäre Fragestellungen inhärent beantwortet werden. Im o. g. Beispiel würden sich anteilig zu bearbeitende Leistungspositionen quasi wie eine Eigenschaft eines konkreten Vorgangs darstellen. Diese Fähigkeit basiert auf domänenübergreifenden Beziehungen der Informationsobjekte untereinander. Würden diese Beziehungen bereits in der zugrunde liegenden Datenbasis existieren, müssten sie nicht innerhalb des verwendeten Informationssystems flüchtig erstellt werden und wären damit im Sinne der Datenübertragung austauschbar (vgl. Abbildung 1.2 rechts unten). Blieben dabei die ursprünglichen, getrennten Fachmodelle unverändert erhalten, wäre auch die traditionelle, domänenspezifische Arbeitsweise mit existierender Bausoftware weiterhin gewährleistet.

Hypothese 2 Die Verknüpfung von diversen, potentiell aus verschiedenen Datenformaten bestehenden, unveränderten Fachmodellen, ermöglicht sowohl getrennte, domänenspezifische als auch übertragbare domänenübergreifende Informationsräume.

Um Anwendern das volle domänenübergreifende Informationspotential verknüpfter Fachmodelle verfügbar zu machen, müssen existierende und zukünftige Baufachanwendungen eine solche Datenbasis erzeugen und erschließen können. Dies erfordert Fähigkeiten zur Verarbeitung der heterogenen Datenformate sowie zum Erstellen und Auswerten von Verknüpfungen zwischen den Datenelementen der Fachmodelle. Demzufolge muss die zu entwickelnde Methode solche Implementierungen fördern. Aus Sicht der Softwareentwicklung kann eine praktische Akzeptanz zudem nur erreicht werden, wenn die zu entwickelnde Methode in einer eigenständigen Softwarekomponente umsetzbar ist und diese in Baufachanwendungen wiederverwendbar eingebunden werden kann. Die Methode sollte daher eine softwaretechnische Unterstützung analog zu der von relationalen Datenbanken ermöglichen:

- Genügende Generalität, um möglichst viele Bauinformationsprozesse zu unterstützen

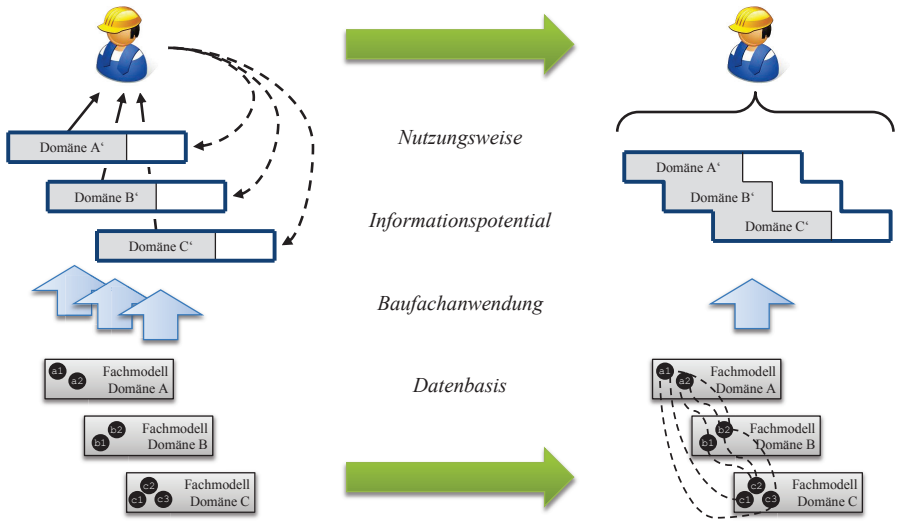


Abbildung 1.2.: Übergang von getrennten, domänenspezifischen Informationsräumen zu einem übertragbaren domänenübergreifenden Informationsraum auf Basis der originalen Fachmodelle

- Einbindung in Baufachsoftware als Service oder Bibliothek
- Eigenständige Lösung typischer Aufgabenstellungen von verknüpften Fachmodellen
- Ausdrucksstarke Kommunikation zwischen Nutzer und Komponente

Hypothese 3 Eine generische Methode zur Erzeugung und Erschließung verknüpfter, heterogener Fachmodelle ermöglicht die Implementierung einer wiederverwendbaren Softwarekomponente zur Realisierung domänenübergreifender Informationsräume in Baufachanwendungen.

1.4. Lösungsansatz

Die Grundidee zur Realisierung austauschbarer, verknüpfter Fachmodelle ist die Nutzung von externen Linkmodellen zu Speicherung von expliziten, objektifizierten, potentiell mehrwertigen Links zwischen den Datenelementen der heterogenen Fachmodelle. Dabei werden die IDs der Datenelemente als Repräsentant des Originals verwendet. Auf diese Weise bleiben die originalen Fachmodelle unverändert. Gemeinsam mit den Linkmodellen und möglichen Metadaten bilden sie **Multimodelle**. Abbildung 1.3 auf der nachfolgenden Seite zeigt das Prinzip der Erzeugung, Übertragung und Erschließung von Multimodellen.

Die Erzeugung von Links geschieht über die Festlegung der fachlich sinnvollen Elementkombinationen aus allen denkbaren modellübergreifenden Kombinationsmöglichkeiten (Abbildung 1.3 oben). Zum Zweck des Datenaustauschs können auf verschiedenen Ebenen Metadaten angefügt werden. Diese sollen die Intention des Absenders verdeutlichen und damit dem Empfänger die inhaltliche Analyse vereinfachen. Die Übertragung des Multimodells geschieht in einem geeigneten Container-Format und ist mit üblichen Mitteln möglich (Abbildung 1.3 Mitte). Die

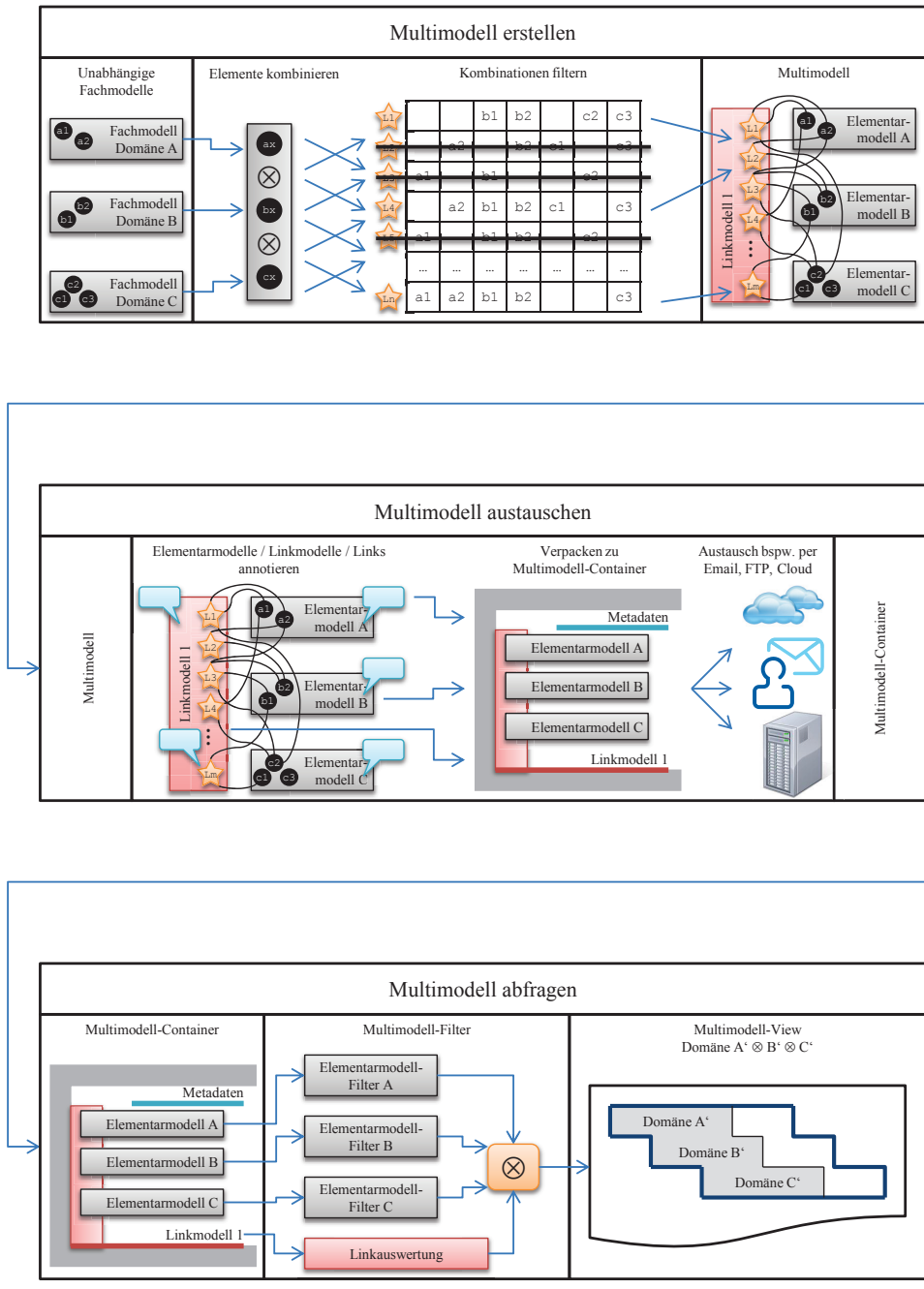


Abbildung 1.3.: Das Multimodellprinzip

Abfrage domänenübergreifender Informationen basiert neben fachmodellspezifischen Filtern maßgeblich auf der Linkauswertung. Erst damit können Elemente aus anderen Fachmodellen als zusätzliches Filterkriterium herangezogen werden. Im Resultat entstehen Multimodell-Views, welche die ursprünglich getrennt vorliegenden Daten als gemeinsamen domänenübergreifenden Informationsraum abbilden (Abbildung 1.3 unten).

1.5. Aufbau der Arbeit

Kapitel 2 gibt einen Überblick zum Stand von Forschung und Technik. Es werden notwendige Fachbegriffe erklärt und definiert. Ausgehend von den Grundlagen der Datenmodelle werden die Eigenschaften von Baufachmodellen erörtert und die bisherigen Ansätze für modell- und domänenübergreifende Informationsräume untersucht.

Kapitel 3 beschreibt den strukturellen Aufbau von Multimodellen und die sich daraus ergebenden Einsatzmöglichkeiten. Es wird gezeigt, wie sich Multimodelle in den Datenaustausch integrieren, welche Fachmodelle anwendbar und welche grundlegenden Operationen mit Multimodellen möglich sind.

Kapitel 4 erklärt die Prinzipien des Multimodell-Filterns und der Linkerzeugung anhand der Abfragesprache MMQL. Die Sprache ermöglicht die prägnante Formulierung von Angaben zur Erstellung von Multimodell-Views und Zustandsänderungen von Linkmodellen. Dazu werden Syntax und Semantik der MMQL beschrieben und an Beispielen verdeutlicht.

Kapitel 5 erläutert die Arbeitsweise eines Interpreters für MMQL-Anweisungen. Es wird erklärt, wie die Abfragesprache für konkrete Daten zur Ausführung gebracht wird, indem die Anweisungen in Ketten von elementaren Multimodell-Operationen überführt werden.

Kapitel 6 zeigt die beispielhafte Anwendung des Multimodellkonzepts und der Abfragesprache MMQL. Dazu wird die universelle, erweiterbare Multimodell-Software M2A2 vorgestellt, in welcher auch der MMQL-Interpreter implementiert wurde. Anhand der automatisierten Ermittlung eines Zahlungsplans werden die erarbeiteten Konzepte überprüft.

Kapitel 7 fasst die Arbeit zusammen, diskutiert die Ergebnisse und gibt einen Ausblick auf weitere Arbeiten zu Multimodellen und domänenübergreifenden Informationsräumen.

Kapitel 2.

Informationsräume im Bauwesen

Zeige mir, wie du baust, und ich sage dir, wer du bist.

(Christian Morgenstern)

Der Datenaustausch im Bauwesen geschieht auf der Basis etablierter Datenmodelle. Gemeinsam mit Methoden zum Datenzugriff bilden sie Informationsräume, welche die Versorgung von Bauprozessen mit Informationen sicherstellen. Dieses Kapitel gibt einen Überblick zum Stand von Forschung und Technik und zeigt die verschiedenen Ansätze zur Unterstützung domänenübergreifender Bauinformationsprozesse.

Abschnitt 2.1 beschreibt Daten und Informationen als Grundlagen der Datenmodelle. Abschnitt 2.2 charakterisiert die Datenmodelle des Bauwesens mit ihrer hohen Komplexität und Vielfalt. Einführend wird die abweichende Verwendung des Modellbegriffs in Bauinformatik und Informationstechnologie analysiert. Das Filtern von Baufachmodellen ermöglicht die Erschließung großer Datenmengen durch Anwender. Die Filtermethoden singulärer Datenmodelle werden aus der Sichtweise von Filtersprachen betrachtet. Abschnitt 2.3 beschreibt einführend Informationsräume als Informationspotential von Daten und zugehörigen Operationen. Anschließend wird analysiert, wie domänen- und fachmodellübergreifende Informationsräume bisher aufgebaut sind. Dazu werden die existierenden Methoden und Ansätze der Systemintegration im Bauwesen mit einem Fokus auf Datenstrukturen erörtert.

2.1. Grundlagen der Datenmodelle

2.1.1. Daten und Informationen

Planung und Ausführung von Bauleistungen sind ohne Informationen undenkbar. Die Einzigartigkeit und Komplexität von Bauvorhaben erfordern seit Anbeginn der Baukultur in allen Bereichen eine Arbeitsteilung – und damit auch Informationsaustausch (Mislin, 1988; Brooks, 2003; Lochmann, 2008)[1]. Um innerhalb der Bauprozesse Entscheidungen treffen zu können, sind Fachleute auf kommunizierte Informationen angewiesen.

Informationen tragen demnach wesentlich zum Gelingen von Bauprozessen bei. Dieser hohe Stellenwert sowie der große Umfang von Informationen in allgemeinen betrieblichen Prozessen machen es notwendig, die Informationshandhabung selbst in sogenannten *Informationsprozessen* zu organisieren. Diese sind wie folgt definiert:

> „Informationsprozess: aus den selbstständigen, gleichzeitigen und sich gegenseitig bedingenden Teilprozessen der Informationsgewinnung, Informationsübermittlung und Informationsverarbeitung bestehend. Er unterlagert den betrieblichen Entscheidungs- und Managementprozess (Entscheidungsprozess) [sic]." (Gabler Wirtschaftslexikon, 2012, Stichwort Informationsprozess)[2]

Bauinformationsprozesse sind die Informationsprozesse des Bauwesens.

Die Erklärung des Informationsbegriffs erfolgt je nach Wissenschaftsrichtung unterschiedlich. Eine einheitliche und allgemein akzeptierte Theorie der Information existiert nicht. Im Wesentlichen ist jedoch zwischen einem naturwissenschaftlichen, vom Menschen unabhängigen, und informationswissenschaftlichen, vom Menschen abhängigen, Verständnis zu unterscheiden. In dieser Arbeit wird ein ingenieurtechnisches System zur Unterstützung von Bauinformationsprozessen entwickelt, welches letztlich zur Benutzung durch den Menschen bestimmt ist. Daher darf der Mensch Teil der Betrachtung bei der Begriffsbestimmung sein:

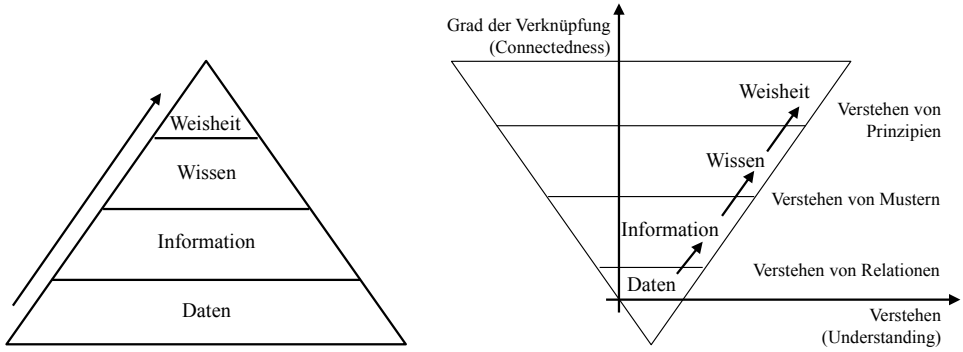
> „Information wird als Zusammenspiel von Form (Struktur), Inhalt (Bedeutung) und Wirkung (Bewertung) verstanden, aus der Sicht der Semiotik also als Triade von Syntax, Semantik und Pragmatik. Information ist dann das Ergebnis aus einem Prozeß der Interpretation der Zeichen (-struktur) durch fühlende, sich selbst organisierende lebende und soziale Systeme, in dem die durch die Interpretation gewonnene Bedeutung, durch ihre Wirkung als lebensdienlich oder nicht bewertet wird. Erst mit dieser durch die Wirkung erfolgte Bewertung ist die Information generiert." (Fuchs-Kittowski, 2002, S. 22)

Nach Voß & Gutenschwager (2000) sind Informationen zwar die Eingangsgrößen in Entscheidungsmodellen – die allgemeine Grundlage des Entscheidens ist jedoch Wissen. Abbildung 2.1a auf der nächsten Seite zeigt die Informationspyramide. Sie differenziert die Begriffe Daten, Information, Wissen und Weisheit. Demnach werden Informationen zunächst durch Daten abgebildet. Es gilt:

> „Daten: zum Zweck der Verarbeitung zusammengefasste Zeichen, die aufgrund bekannter oder unterstellter Abmachungen Informationen (d. h. Angaben über

[1] Brooks bezieht sich bei der Notwendigkeit des Informationsaustauschs zwar auf die Arbeitsteilung in IKT-Projekten; das Gesagte gilt jedoch auch für Bauplanungs- und -managementprozesse.

[2] http://wirtschaftslexikon.gabler.de/Archiv/9828/informationsprozess-v6.html



(a) Informationspyramide **(b)** Von Daten zur Weisheit – Lernen

Abbildung 2.1.: Information im Kontext des Entscheiders (beide Abbildungen aus Voß & Gutenschwager, 2000)

> Sachverhalte und Vorgänge) darstellen." (Gabler Wirtschaftslexikon, 2012, Stichwort Daten)[3]

Informationen können demnach aus der Interpretation von Daten gewonnen werden. Diese Interpretation kann jedoch nur durch eine Person erfolgen, da deren Wissen als Kontext benötigt wird (Leser & Naumann, 2007). Wissen wird wiederum aus Informationen gewonnen, indem diese zu anderen Informationen in Beziehung gesetzt (vernetzt), Informationen begründet (z. B. Aufzeigen von Ursache-Wirkungs-Beziehungen), Informationen konzeptualisiert (durch Selektion, Einordnung, Klassifizierung, Interpretation) oder komprimierte Erfahrung (Intuition) genutzt werden (Fuchs-Kittowski, 2002). Die Ebene *Weisheit* kann letztendlich als Wissen aufgefasst werden, welches durch die eigene Lebenserfahrung bewertet wurde.

Abbildung 2.1b stellt dar, wie durch Verstehen der Übergang zu einer höheren Ebene der Informationspyramide erreicht wird. Hier ist insbesondere der Übergang von Daten zu Informationen durch Verstehen von Relationen relevant. Im Sinne einer Entscheidungsunterstützungsfunktion ist dies nach Voß & Gutenschwager (2000) bereits dann schwierig, wenn die Informationssammlung (respektive der Informationsprozess) nicht vom Entscheider selbst durchgeführt wird. Es ist notwendig, Relationen zwischen Daten und Sachverhalten zu erkennen und zu systematisieren. Eine Transparenz der Daten ergibt sich unter Umständen erst durch Rekapitulation des Entscheidungsmodells, quasi des Kontextes.

Zusammenfassend sollen folgende komprimierte Begriffsdefinitionen nach Meyer zu Selhausen (1996) und Fuchs-Kittowski (2002) gelten:

Datum formalisierte Sachverhaltsaussage

Information im Kontext interpretierte, zweckbezogene Daten

Wissen begründete Information (konzeptualisierte, in Beziehung gesetzte Informationen)

Weisheit dem Wissen beigemessener Nutzwert (Witten et al., 2011, S. 35)

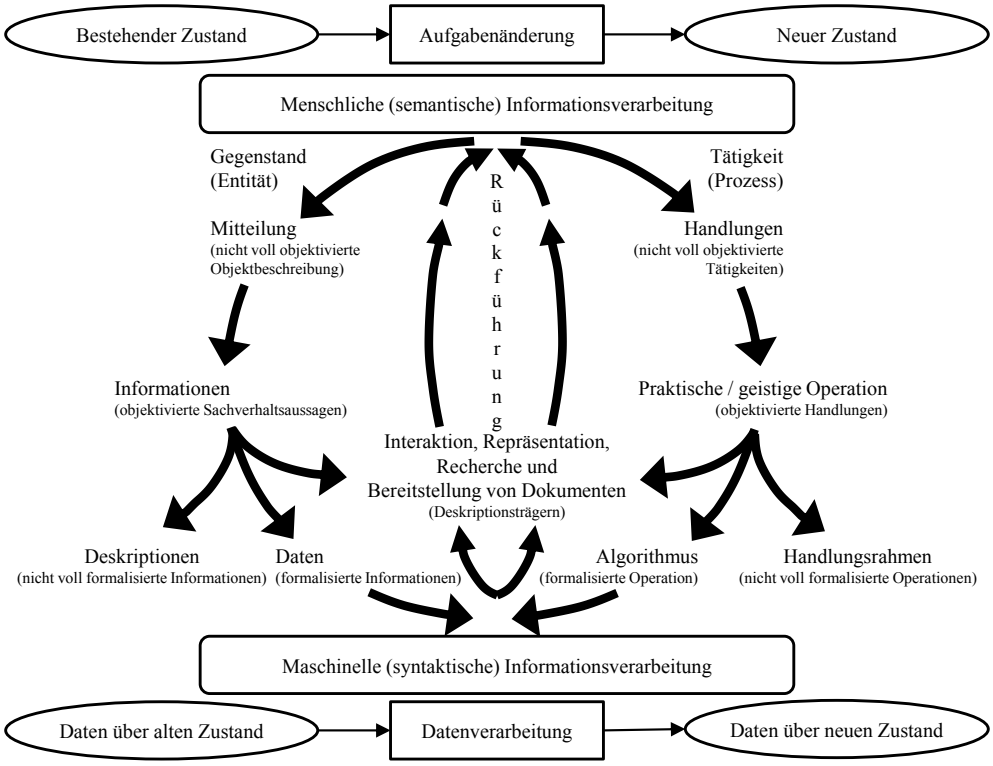


Abbildung 2.2.: Übergang von der menschlichen zur syntaktischen Informationsverarbeitung mit Rückführung der maschinellen Operationen in die Komplexität der menschlichen Aktivitäten (aus Fuchs-Kittowski, 2002)

2.1.2. Daten- und Informationsverarbeitung

Im Kontext einer prozessorientierten Arbeitsweise bilden Bauinformationsprozesse den Handlungsrahmen der Akteure. Es ist Aufgabe der Informations- und Kommunikationstechnologie (IKT) hierfür entscheidungsrelevante Informationen bereitzustellen und diese zu verarbeiten. Dabei ist zwischen menschlicher (semantischer) und maschineller (syntaktischer) Informationsverarbeitung zu unterscheiden. Abbildung 2.2 zeigt das Modell des Übergangs von semantischer zu maschineller Informationsverarbeitung nach Fuchs-Kittowski (2002). Die relevanten Konzepte sind:

1. Semantische Informationsverarbeitung im menschlichen Arbeitsprozess ist die Änderung von Zuständen durch die Erfüllung von Aufgaben (praktische oder geistige Tätigkeit des Menschen).
2. Maschinelle Informationsverarbeitung ist Datenverarbeitung: sie ändert die Daten über Zustände.
3. Das Wirksamwerden von Daten erfordert deren Rückführung über Umwandlungen in

[3] http://wirtschaftslexikon.gabler.de/Archiv/54483/daten-v5.html

Informationen und Wissen in die menschliche Tätigkeit.

Die menschliche Informationsverarbeitung kann damit als Kernelement der Wertschöpfung in Bauinformationsprozessen angesehen werden. Der Informationsverarbeitungs-Ansatz wird wie folgt erklärt:

> „Informationsverarbeitungs-Ansatz (Entscheidungstheorie): Ansatz zum Entscheidungsverhalten einer Einzelperson (Individualentscheidung). Der betrachtete Mensch wird v. a. als informationsverarbeitendes System gesehen (kognitiv-empirischer Ansatz). Charakterisierung:
> (1) Informationsbeschaffung und -verarbeitung sind wichtige Teile der Entscheidung;
> (2) situations- und kontextabhängige Sicht der Entscheidung;
> (3) Integration von Entscheidungsfällungsinstrumentarien und Eigenschaften des Menschen." (Gabler Wirtschaftslexikon, 2012, Stichwort Informationsverarbeitung)[4]

Die maschinelle Datenverarbeitung im Sinne der IKT wird als wesentliche Unterstützung der menschlichen Informationsverarbeitung – und damit von Bauinformationsprozessen – angesehen. Datenverarbeitung ist definiert als:

> „Datenverarbeitung / Data Processing: Erfassen, Übermitteln, Ordnen und Umformen von Daten zur Informationsgewinnung, i. Allg. mithilfe eines Computers." (Gabler Wirtschaftslexikon, 2012, Stichwort Datenverarbeitung)[5]

Die Mittel der Datenverarbeitung sind (Computer-)Programme:

> „Computerprogramm: 1. Allgemein: in der Informatik Darstellung eines Problemlösungsverfahrens in einer für den Computer verständlichen Form. Programme werden in einer Programmiersprache formuliert.
> 2. Speziell: bei Verwendung einer prozeduralen Programmiersprache maschinenverständliche Darstellung des Algorithmus und der Daten, die dieser bearbeitet." (Gabler Wirtschaftslexikon, 2012, Stichwort Programm)[6]

Letztendlich sind Algorithmen und deren Daten die Basisbausteine zur Bereitstellung von Informationen. Es ist Aufgabe der Bauinformatik, die zur Unterstützung von Bauinformationsprozessen notwendigen Daten und Algorithmen zu definieren (vgl. Arbeitskreis Bauinformatik, 2000).

2.1.3. Datenmodelle

Im allgemeinen Sprachgebrauch sind Modelle ein vereinfachtes Abbild der Realität. Bei näherer Betrachtung ist jedoch festzustellen, dass der Modellbegriff uneinheitlich verwendet wird. Im Wissenschaftsgebiet der Bauinformatik, als Querschnittsdisziplin zwischen Informatik und Bauingenieurwesen, ist die Vielfalt seiner Bedeutungen besonders groß. Beide Fachrichtungen zeichnen sich traditionell durch die Verwendung unterschiedlicher Modelle aus, beispielsweise (physische) Architekturmodelle, Rechenmodelle, Statische Modelle, Versuchsmodelle, Prozessmodelle, Simulationsmodelle oder Datenmodelle.

Analysen, Vergleiche und Literaturübersichten zu verschiedenen Modelltheorien sind bei Schlitt (2004) und Thomas (2005) zu finden. Davon ist u. a. die *Allgemeine Modelltheorie* nach

[4] http://wirtschaftslexikon.gabler.de/Archiv/10378/informationsverarbeitung-v7.html
[5] http://wirtschaftslexikon.gabler.de/Archiv/56458/datenverarbeitung-v7.html
[6] http://wirtschaftslexikon.gabler.de/Archiv/57129/programm-v7.html

Stachowiak (1973) weit verbreitet und akzeptiert. Gegenstand der vorliegenden Arbeit sind die in Bauinformationsprozessen auftretenden Datenmodelle. Diese können aus heutiger Sicht sehr gut mit der Allgemeinen Modelltheorie erklärt werden – sowohl in Hinblick auf ihren semantischen (baufachlichen), als auch ihren informationstechnischen Charakter.

Nach Stachowiak wird ein Modell durch drei wesentliche Merkmale konstituiert:

- Abbildung

  Modelle sind Abbild oder auch Vorbild eines Originals. Das Original kann natürlich oder künstlich, eine physische oder immaterielle Entität oder sogar selbst ein Modell sein.

- Verkürzung

  Modelle abstrahieren von ihrem Original. Dabei werden nur relevante Attribute des Originals erfasst.

- Pragmatismus

  Modelle adressieren einen Verwendungszweck in Bezug auf Operationen (Wozu ?), Subjekte (Für wen ?) und Zeiträume (Wann ?)

Ein Datenmodell ist ein Modell der Organisation bestimmter Daten. Es ist demnach nicht durch die potentiell oder tatsächlich vorhandenen Daten charakterisiert, sondern durch deren Strukturen zur maschinellen Speicherung und Verarbeitung. Entsprechend gilt:

> „**Datenmodell**: Modell der zu beschreibenden und verarbeitenden Daten eines Anwendungsbereichs (z. B. Daten des Produktionsbereichs, des Rechnungswesens oder die Gesamtheit der Unternehmensdaten) und ihrer Beziehungen zueinander." (Gabler Wirtschaftslexikon, 2012, Stichwort Datenmodell)[7]

Thomas (2002) präsentiert eine Taxonomie der Modelle der Informatik basierend auf einer empirischen Analyse. Ein Datenmodell gehört demnach zu den Entwurfsmodellen. Die bekanntesten Unterarten sind das Objektorientierte Modell (vgl. Rumbaugh et al., 1993) sowie Relationale Daten(bank)modelle, z. B. das Entitiy-Relationship-Modell (ER-Modell, Chen, 1976).

Einen Überblick über den Strukturierungsvorgang, die sogenannte Datenmodellierung, gibt u. a. Hay (2006). Aus semantischer Sicht müssen Datenmodelle den Informationsbedarf des Anwendungsbereichs möglichst vollständig und korrekt abbilden. Aus Sicht der Informationstechnik müssen Datenmodelle eine effiziente Implementierung ermöglichen. Diese Ziele können konträr sein. Datenmodellierung geschieht daher üblicherweise in folgenden Stufen:

1. Konzeptuelles Datenmodell

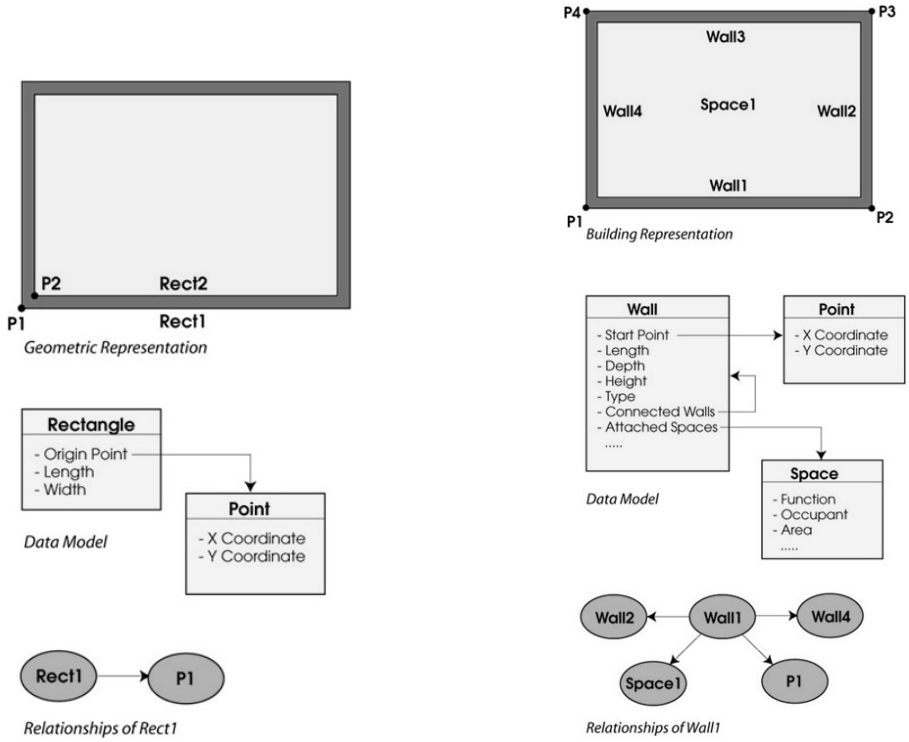
   Implementierungsunabhängig, modellierte Realweltobjekte: Entitäten sowie Beziehungen zwischen diesen, z. B. ER-Diagramme oder UML-Diagramme

2. Logisches Datenmodell

   Die Semantik wird auf Ebene von Datenmanipulationstechniken beschrieben, z. B. relationales Datenmodell für RDB, XML-Elemente oder objektorientierte Klassen.

3. Physisches Datenmodell

   Beschreibt physische Eigenschaften, wie Daten gespeichert oder verarbeitet werden, z. B. CPU-Spezifika, Tabellengrößen, Indizes usw.



(a) Geometrisches Datenmodell **(b)** Bauwerksmodell

Abbildung 2.3.: Beispiele für Datenmodelle von Wänden (beide Abbildungen aus Khemlani, 2004)

In dieser Arbeit sind vorrangig die logischen Datenmodelle des Bauwesens von Interesse, da sie austauschbare *strukturierte* Daten ermöglichen[8]. Als formales Modell, bspw. in Form von Schemas, ermöglichen sie die Validierung von Daten hinsichtlich der geforderten Datenstruktur. Eine Datenstruktur ist eine Spezifikation zur Zusammenfassung von Datenelementen zu größeren und komplexen Einheiten.

Je nach Anwendungsbereich können Informationen unterschiedlich abgebildet werden. Abbildung 2.3 zeigt exemplarisch zwei mögliche Datenmodelle für Wände. In 2.3a werden Wände durch die geometrischen Formen *Punkt* und *Linie* repräsentiert. Diese Darstellungsform wird bspw. in 2D/3D-CAD-Systemen zur Erstellung von technischen Zeichnungen eingesetzt. Abbildung 2.3b zeigt eine für Bauwerksmodelle typische Datenstruktur. Hier besitzen Wände nicht nur eine Geometrie sondern sind auch als Bauelement *Wand* wiedererkennbar, haben weitere Attribute wie Materialtyp und können in Verbindung mit Räumen sowie angrenzenden Wänden stehen. Aus diesen Daten lassen sich umfangreichere Informationen ableiten. Damit wird deutlich, dass Daten – egal welchen Fachbereichs – an ihre Modelle gebunden sind und dass jedes Datenmodell entsprechend seines Informationsangebots einen abgegrenzten Anwendungsbereich

[7] `http://wirtschaftslexikon.gabler.de/Archiv/74962/datenmodell-v6.html`

[8] In Abschnitt 3.4 wird gezeigt wie auch *semistrukturierte* Daten (z. B. schemalose XML-Daten) und *unstrukturierte* Daten (z. B. Dokumente) in das Multimodellkonzept integriert werden können.

besitzt.

Datenstrukturen begründen Datenformate – die Form wie Daten gespeichert und übertragen werden. Es können mehrere Datenformate für dieselbe Datenstruktur existieren, bspw. ein binäres Format mit geringem Speicherplatzbedarf und ein textbasiertes Format, welches besser durch Menschen lesbar ist.

Datenformate (auch: *Serialisierung*) sind Vorschriften zum Laden, Interpretieren und Speichern der Daten durch ein Programm.

2.1.4. Identität von Dateneinheiten

Ein Ziel der Datenmodellierung ist es, die Entitäten der realen Welt des Anwendungsbereichs auch innerhalb der Datenstruktur als Entitäten abzubilden. Beispielsweise existieren in Terminplan-Datenmodellen je nach Technologie Klassen oder Tabellen für Vorgänge, welche Attribute wie Start- und Endzeitpunkt besitzen. Diese Vorschriften bestimmen, wie die in Informationsprozessen real auftretenden Dateneinheiten abgebildet werden. Diese Dateneinheiten besitzen eine Identität, welche es zulässt, eine Dateneinheit von einer anderen zu unterscheiden. Dadurch wird eine eindeutige Referenzierung ermöglicht, die Grundlage für die Wiederverwendung von Dateneinheiten innerhalb und außerhalb des Datenmodells ist.

Diese Identität kann implizit im verarbeitenden System entstehen (bspw. bei Verwendung objektorientierter Programmiersprachen), notwendiger Teil der Datenmodellierung sein (bspw. Primärschlüssel im relationalen Datenmodell) oder explizit modelliert sein (bspw. Identifikator-Datenelement). Der Telecommunication Standardization Sector der International Telecommunication Union (ITU-T) definiert die Begriffe Identifikation (identification), Identifikator (identifier) sowie Identität (identity) wie folgt:

> „6.28 **identification**: The process of recognizing an entity by contextual characteristics.
> 6.29 **identifier**: One or more attributes used to identify an entity within a context.
> 6.30 **identity**: A representation of an entity in the form of one or more attributes that allow the entity or entities to be sufficiently distinguished within context. [...]"
> (ITU-T X.1252, 2010, S. 3 f.)

Im Kontext des Datenaustauschs in Bauinformationsprozessen ist insbesondere die Fragestellung nach expliziten Identifikatoren (IDs) in Datenmodellen relevant. Diese können als Stellvertreter (Proxy) für die eigentliche Dateneinheit angesehen werden und sind somit Grundlage für Informationen *über* diese Dateneinheit. Zum Beispiel werden dadurch ein konsistenter Zugriff auf Datenfragmente durch Drittsoftware ermöglicht oder das Nachverfolgen von Änderungen erlaubt[9].

Eine ID-basierte Identifizierung ist robust gegenüber strukturellen und inhaltlichen Änderungen (z. B. Dateneinheit verschieben, Unterelemente ändern). Sie ist jedoch fragil gegenüber validen (z. B. Element löschen) und invaliden Änderungen der ID, wie bspw. Element klonen (inklusive ID) oder ID-Wert ändern. Risiken für Bauinformationsprozesse im Zusammenhang mit Identifikatoren entstehen durch verwaiste Referenzen auf nicht existente IDs sowie durch mangelhafte ID-Qualität; d. h. die Wahrscheinlichkeit von Konflikten durch nicht einzigartige IDs innerhalb eines Fachmodells ist groß.

[9] Einen Ansatz zum Nachverfolgen von Änderungen *ohne* Identifikator beschreibt Weise (2006a)

2.2. Baufachmodelle

2.2.1. Zum Modellbegriff in der Bauinformatik

Der Einsatz von Modellen im Bauwesen hat eine lange Tradition. Mit deren Hilfe lassen sich wesentliche Fragen an Produkte und Prozesse schon vor ihrer Realisierung beantworten. Das ist besonders im Bausektor vorteilhaft, da Bauwerke und -prozesse komplex, teuer, einzigartig und langlebig sind. Nachträgliche Korrekturen sind nur schwer möglich. Andere, existierende Bauvorhaben können auch nur bedingt als Modell für ein konkretes Projekt herangezogen werden. Dementsprechend hat sich die Bauplanung historisch als eigenständige Disziplin des Bauwesens qualifiziert. Im Sinne der Modelltheorie ist das Ergebnis der Bauplanung ein *Vorgabemodell* eines Bauvorhabens (deskriptives Modell).

Die Verwendung von Modellen beschränkt sich nicht nur auf den Planungsprozess. Während und nach der Fertigstellung eines Bauwerks können Modelle auch als *Abbild* des Originals auftreten (präskriptives Modell). An ihnen werden Eigenschaftsanalysen durchgeführt, welche im Baukörper zu aufwändig oder unmöglich sind oder ihrer Natur nach nur mit *Modellen* funktionieren, z. B. weil sie eine Kopplung mit anderen Modellen erfordern.

Im Kontext von Entscheidungsprozessen ist relevant, *welche* Fragen durch das Modell beantwortbar sind. Bezogen auf Bauinformationsprozesse kann ein Modell so zur Informationsquelle werden. Kommt es zum Austausch dieser Informationen – nicht des originären Modells – ist es von Bedeutung, welche Fragen durch die Informationen *für den Empfänger* beantwortbar sind. Im Idealfall hätte das originäre Modell den gleichen Informationsgehalt wie die übertragenen Informationen. Im Allgemeinen kann gesagt werden, dass mit zunehmender Formalisierung der Modelle der Interpretationsspielraum zur Informationsgewinnung sinkt. Das bedeutet, mehr Menschen gewinnen übereinstimmende Informationen aus der Analyse desselben Modells. Eine Aussage über Umfang und Qualität der Informationen ist daraus jedoch nicht ableitbar.

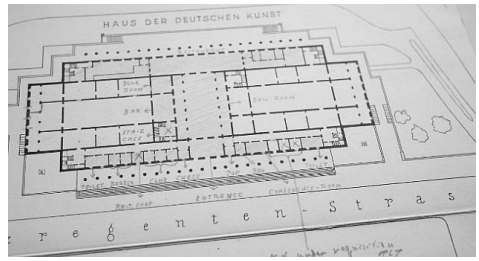
Abbildung 2.4 auf der nächsten Seite zeigt eine Auswahl von Modellen im Bauwesen. Ihnen liegen unterschiedliche Arten und Grade der Formalisierung zugrunde. Physische Architekturmodelle (2.4a) sind Anschauungsmodelle für räumliche Situation, Baukubatur oder für Material und Proportionen. Sie haben jedoch kaum formalen Charakter. So wird in der Regel der Modellersteller persönlich zur Interpretation des Modells benötigt. Auch sind Architekturmodelle nicht dazu gedacht, Erstellungsmaße abzunehmen. Aufgrund des vielfältigen Einsatzzwecks gibt es kaum Konventionen für Erstellung und Interpretation von Architekturmodellen.

Pläne in Form von Zeichnungen (2.4b) sind nach DIN 1356 (1995) genormt. Materialien können qualitativ, bspw. durch Schraffuren, beschrieben werden. Abmessungen sind durch Maßketten quantifiziert, fehlende Abmessungen sollen nachgemessen werden können. Der Interpretationsspielraum für die Informationen *Abmessung* und *Material* ist gegenüber dem Architekturmodell kleiner.

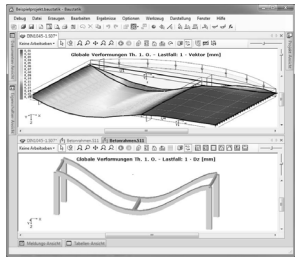
Abbildung 2.4c zeigt die Visualisierung eines statischen Modells. Der formale Charakter statischer Modelle liegt weniger in ihrer bildlichen Repräsentation als in der Normung zu Inhalt und Durchführung statischer Nachweise. Unter statischen Modellen versteht man auch Theorien zum Tragverhalten von Bauteilen, also mathematisch formalisierte Verhaltensmodelle.

Abbildung 2.4d zeigt eine mögliche Visualisierung eines virtuellen Gebäudemodells. Die Daten, aus denen dieses Modell besteht, sind streng formal – für sie gilt das genormte Datenmodell IFC (ISO 16739, 2005). Sie sind daher gut zum Austausch geeignet, da die Daten für unterschiedliche Softwareprodukte gleichbedeutend sind. Die Informationsinterpretation findet an

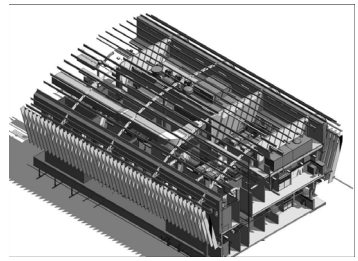
(a) Architekturmodell



(b) Zeichnung als Plan



(c) Statisches Modell



(d) Virtuelles Gebäudemodell

Abbildung 2.4.: Beispiele von Modellen im Bauwesen mit unterschiedlicher Formalisierung
Quellennachweis: (a) Bundesarchiv, Bild 183-27387-0001; (b) `http://www.hausderkunst.de/`; (c) `http://www.die.de`; (d) `http://www.aecbytes.com/`

der Schnittstelle Mensch-Computer statt. Hier ist es Aufgabe der Software, die Daten so zu präsentieren, dass der Interpretationsspielraum möglichst gering ist.

Im Bereich der IKT sind solche formalen Datenmodelle von besonderem Interesse. Sie werden dort oftmals auch nur *Modelle* genannt und sind in einer Modellierungssprache verfasst. Modelle sind eine Vorschrift zur Abbildung von Daten. Ein entsprechender Datensatz wird *Instanz* genannt. Formale Modelle ermöglichen es, ihre Instanzen auf Korrektheit zu überprüfen. Umgekehrt können Datenmodelle selbst als Daten aufgefasst werden und einer Bildungsvorschrift, dem so genannten *Meta-Modell*, unterliegen. Dessen Bildungsvorschrift wird *Meta-Meta-Modell* genannt. Da sich der Meta-Status immer nur relativ zu *einem* Modell in Beziehung bringen lässt, könnte diese Prozedur beliebig fortgeführt werden.

Die Object Management Group (OMG) definiert daher mit dem Standard Meta Object Facility (MOF, ISO 19502, 2005) eine Architektur aus vier Metaschichten (vgl. Abbildung 2.5 auf der nachfolgenden Seite, rechte Spalte). Das wesentliche Merkmal dieser Architektur ist die Selbstbeschreibung von M3 durch den Einsatz einer reflexiven Sprache und damit das Beenden der Metaebenen. Zwar konnte sich die Architektur aufgrund der Festlegung auf definierte Modellierungssprachen sowie fehlender Werkzeugunterstützung bis heute nicht in der Praxis durchsetzen, ihre wesentlichen Prinzipien finden aber Anwendung. Die Modellgetriebene Softwareentwicklung (Model Driven Software Development, MDSD) propagiert z. B. einen liberaleren Ansatz ohne Festlegung von Sprachen oder Anzahl der Metaebenen. Mit dem Ziel einer pragmatischen Arbeitsweise in der Softwareentwicklung simplifiziert Merks (2008) die Rolle der formalen Datenmodelle sogar durch das Streichen des Meta-Status:

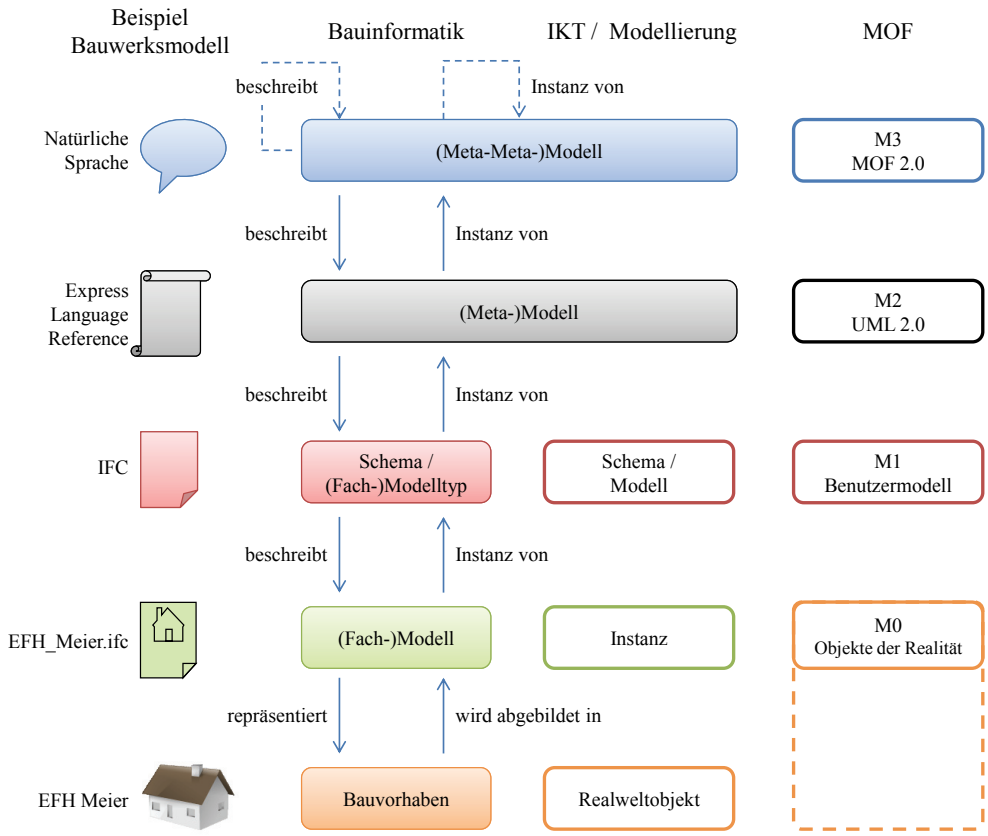


Abbildung 2.5.: Modellbegriff in Bauinformatik und IKT am Beispiel eines Einfamilienhauses. Die Verwendung des Begriffs *Modell* unterscheidet sich in Bauinformatik und IKT auf den Ebenen MOF M0 und M1.

„The model of a model is a ~~meta~~model" (Merks, 2008, S. 8)

Dadurch wird die heute gebräuchliche Verwendung des Begriffs *Modell* in der IKT verdeutlicht: Modelle sind formale Vorschriften für *Daten*. Für die Bauinformatik sind die formalen Datenmodelle des Bauwesens relevant. Hier wird der Begriff *Modell* jedoch abweichend verwendet. Abbildung 2.5 verdeutlicht dies am Beispiel eines Bauwerksmodells – der Datenrepräsentation einer konkreten Gebäudegeometrie. Auf der untersten Ebene stehen die Originale der Realwelt, in diesem Fall das Bauwerk *EFH Meier*. Eine Ebene höher werden Daten dargestellt. Die Datei `EFH_Meier.ifc` repräsentiert festgelegte Informationen über das Realweltobjekt EFH Meier, bspw. die Koordinaten der Gebäudeecken. Im Sinne einer baufachlichen Anwendung stellt die Datei *keine* Vorschrift für weitere Daten dar. In der IKT wird auf dieser Ebene deshalb der Begriff *Instanz* verwendet. Die Bauinformatik verwendet hier jedoch üblicherweise den Begriff

Modell, *Fachmodell* oder *Baufachmodell*. Die Ursachen dafür sind vermutlich:

- Die Modelle des Bauwesens bilden traditionell Teile von Bauvorhaben ab.
- Im Sinne der Bauinformatik sind Daten solche Modelle.
- Die softwaretechnischen Aspekte dieser Daten unterliegen der Verantwortlichkeit der IKT.

Die OMG unterscheidet nicht zwischen Realweltobjekt und Dateninstanz, sondern fasst beide zu M0 zusammen. Dies liegt möglicherweise am softwaretechnischen Bezug des Standards, bei dem nicht auf die fachliche Bedeutung der Daten eingegangen wird. Baufachmodelle werden somit wie folgt definiert:

Fachmodell (auch: *Baufachmodell* oder *Modell* im Sinne der Bauinformatik) ist eine MOF M0-äquivalente Dateninstanz des Bauwesens

Die Ebene OMG M1 ist die Ebene der Datenmodelle. Sie werden im IKT-Bereich *Modell* oder *Schema* genannt. Auch in der Bauinformatik ist der Begriff *Schema* üblich, oftmals wird aber auch *(Fach-)Modelltyp* verwendet um zu verdeutlichen, dass es sich um die Vorschrift des Baufachmodells handelt. Im Beispiel ist das die IFC-Spezifikation (ISO 16739, 2005). Auf dieser Ebene erfolgt die Datenmodellierung, welche festlegt, wie baufachliche Informationen in Daten abgebildet werden. Nicht alle Baufachmodelle besitzen jedoch ein *formales* Schema. Fachmodelltypen sind daher definiert als:

Fachmodelltyp (auch: *Baufachmodelltyp*) ist eine MOF M1-äquivalente Dateninstanz des Bauwesens (s. g. Schema, Datenschema) oder deren informale Entsprechung

Ab Ebene M2 findet in der Regel keine terminologische Trennung mehr zwischen Bauinformatik und IKT statt. Dies ist die Ebene der Definition von Modellierungssprachen, welche auf M1 verwendet werden. Die Modellierungssprache von IFC ist EXPRESS. EXPRESS ist ein informales Modell, definiert durch eine standardisierte Sprachspezifikation in Form eines Dokuments (ISO 10303-11, 2004). Das Metamodell von EXPRESS (Ebene M3) ist somit implizit die natürliche Sprache[10]. M3 ist die Ebene der Metamodelle der Modellierungssprachen, bspw. MOF für UML. Es muss jedoch nicht jedes Meta-Meta-Modell selbstbeschreibend sein, weitere Metaschichten sind möglich. Außerdem hat auch nicht jede Modellierungssprache ein separates Metamodell – so ist bspw. Ecore bereits auf Ebene M2 selbstbeschreibend (Steinberg et al., 2009).

2.2.2. Vielfalt der Datenformate im Bauwesen

Die Vielfalt der Datenformate im Bauwesen ist groß. In der Literatur werden ausgewählte Übersichten wiedergegeben (Karimi & Akinci, 2010; Eastman et al., 2011; Hansen & Zenobia, 2011). Eine vollständige Sammlung der Datenformate des AEC-Sektors ist jedoch nicht bekannt.

Tabelle 2.1 auf der nächsten Seite zeigt eine Auswahl relevanter Baufachmodelltypen mit ihren zugehörigen Domänen. Die Aufzählung ist nicht abschließend. Es wird jedoch deutlich, dass die Fachmodelltypen unterschiedlich spezialisiert sind (vertikale Ausprägung). Einige sind nur in *einer* oder wenigen Domänen einsetzbar, andere wie IFC wurden dafür ausgelegt, möglichst

[10] Mit der Spezifikation OMG Express 1.0 (2010) gibt es einen Ansatz, um EXPRESS genügend zu formalisieren, damit Transformationsprozesse von Express-Instanzen (Ebene M1) in andere M2-Formate – wie UML oder XML – automatisiert durchführbar sind.

viele Anwendungsgebiete abzudecken. Die horizontale Ausprägung zeigt auf, dass für einige Domänen wie Abwassertechnik nur wenige, für andere wie Baustelle oder Hochbaugeometrie hingegen mehrere Fachmodelltypen existieren. Dabei ist jedoch zu beachten, dass in den Schemas möglicherweise unterschiedliche Informationen abgebildet werden, da andere Aspekte der Domäne betrachtet werden.

Im unteren Teil der Tabelle wird wiedergegeben, auf welcher Technologie die Datenformate der Fachmodelltypen basieren. Die Mehrheit der hier aufgeführten Typen hat Datenformate auf Basis von XML oder ASCII. ASCII bedeutet hier, dass es sich um ein textbasiertes Format ohne formale Meta-Datenstruktur handelt. Daten der Baugeräteliste (BGL) sind über einen Internetservice verfügbar. Für einige Fachmodelltypen existieren sogar mehrere Datenformate auf Basis unterschiedlicher Technologien.

Die Ursachen der Vielfalt der Datenformate im Bauwesen sind:

Tabelle 2.1.: Beispiele für Fachmodelltypen des Bauwesens mit ihren Domänen und der Basis ihrer Datenformate

Datenformat / Domäne	BGL	CityGML	Datanorm	DXF	Energy Plus	GAEB D80-89	GAEB DA11	IFC	ISYBAU	KML	LandXML	LDAP / LDIF	Microsoft Project
Ablaufplanung								✓					✓
Abwassertechnik									✓				
Abrechnung							✓						
Baustelle	✓	✓								✓	✓		
Hochbaugeometrie		✓		✓				✓		✓			
Facility Management								✓					
Geodaten		✓								✓	✓		
HVAC			✓		✓			✓					
Leistungsbeschreibung			✓			✓		✓					
Mengen						✓	✓	✓					
Organisation								✓				✓	
Preise			✓			✓		✓					
Tiefbaugeometrie				✓							✓		
Datenformat / Basis													
XML (XSD)		✓				✓		✓	✓	✓	✓		✓
SPF (EXPRESS)								✓					
ASCII			✓	✓	✓	✓	✓					✓	
Service	✓											✓	

1. Hohe Anzahl traditioneller Baumodelle und historisch etablierter Bauprozesse

   Die Komplexität der Bauvorhaben führte schon lange vor dem Aufkommen der Rechentechnik zu einer auf traditionellen Modellen basierten Arbeitsweise. So sind technische Zeichnungen beispielsweise seit dem Mittelalter belegt. Probleme wurden gewöhnlich durch Zerlegung in Teilprobleme gelöst. Seit dem Aufkommen der Technischen Mechanik etwa im 17. Jahrhundert entstanden viele spezialisierte Verhaltensmodelle der Tragfähigkeit von Bauteilen. Auch die Rahmenbedingungen der geschäftlichen Abläufe sind zum Beispiel seit 1926 in der Vergabe- und Vertragsordnung für Bauleistungen (DIN VOB, 1926, 2012) dokumentiert.

2. Überführung traditioneller Modelle in Computermodelle

   Praxl (2010) beschreibt wie traditionelle Modelle mit Aufkommen der Rechentechnik in Computermodelle überführt wurden. Auch aufgrund mangelnder Rechenkapazitäten wurden die Baumodelle dabei nahezu unverändert auf Computermodelle übertragen – technische Zeichnungen wurden in CAD-Datenformaten abgebildet, statische Modelle z. B. in FEM-Modellen usw. Die Vielfalt blieb dabei erhalten. Auch die korrespondierenden Applikationen sind ursprünglich zur Lösung feingranularer Aufgaben gestaltet worden und nicht zur zentralen Integration (Fisher et al., 1997).

3. Hohe Anzahl orthogonaler Domänen

   Die große Anzahl an Fachgebieten innerhalb des Bauwesens führt zu entsprechend separierten Informationsbedarfen der Anwendungsbereiche. Es ist davon auszugehen, dass sich auch in Zukunft durch Wissenszuwachs und technischen Fortschritt die Belange im Bauwesen erhöhen und weitere Domänen hinzukommen werden. Dadurch wird sich auch die Zahl der Datenformate erhöhen.

4. Redundante, versionierte und konkurrierende Datenformate

   In Bezug auf das formale Datenmodell existieren von einigen Fachmodelltypen redundante Datenformate. So gibt es bspw. GAEB DA 1990 (ASCII-basierend; Gemeinsamer Ausschuss Elektronik im Bauwesen (GAEB), 1990) und GAEB-DA-XML (XML basierend; GAEB-DA-XML 3.2, 2013) sowie IFC (SPF-basierend) und IFC-XML (ISO 10303-28:2007, 2007). Die Ursache hierfür ist meist in der technischen Weiterentwicklung der Datenformate begründet. Inkompatible neue Formate lösen ältere ab. Die fachliche Pflege der Schemas erfordert oftmals Änderungen, welche sich in neuen Versionen eines Datenformats niederschlagen. Diese sind je nach Art der Änderung abwärts kompatibel zu älteren Versionen.

   Darüber hinaus existieren viele konkurrierende Datenformate, welche sich inhaltlich überschneiden oder an bestimmte Softwareprodukte gekoppelt sind. So wurden beispielsweise durch das Bundesministerium für Verkehr, Bau und Stadtentwicklung (BMVBS) (2012) die europäischen Datenformate für Ausschreibung und Vergabe verglichen. Dong et al. (2007) stellen IFC und gbXML gegenüber. Einen Marktüberblick über Terminplanungssoftware gibt Tulke (2010, S. 29). Für die Parallelentwicklung oder Alleinstellung der Formate sind meist organisatorische oder wirtschaftliche Gründe der Spezifizierungsgremien oder Softwarehersteller die Ursache. Zudem wird die Fragmentierung der Datenformate durch die Abwärtskompatibilität der Fachanwendungen sowie deren umfangreiche Im- und Exportfunktionen gefördert.

2.2.3. Filtern von Baufachmodellen

Das Filtern bestimmter Dateneinheiten aus einer großen Datenmenge ist eine der wichtigsten Aufgaben der IKT. Die zu verarbeitende Datenmenge in Bauinformationsprozessen ist abhängig von Arbeitsweise, Domäne und Fachmodelltyp. Generell kann jedoch gesagt werden, dass die Gesamtdatenmenge besonders bei fortgeschrittenem Projektstand ein so hohes Niveau erreicht, dass sie manuell nicht mehr sinnvoll erfassbar ist. Damit aus dem Datenbestand entscheidungsrelevante Informationen abgeleitet werden können, muss er entsprechend der Aufgabe reduziert werden.

Filtern wird jedoch nicht nur zum Vermindern der Datenmenge benötigt. Vielmehr kann es als elementare Operation zur Identifikation von Dateneinheiten angesehen werden. Filtern ist damit Bestandteil von Viewern, Konvertern und vielen weiteren Basiskomponenten moderner Baufachanwendungen. Im Kontext von Datenbanken beschreibt Conrad (1997, S. 61) Filteroperationen als Vorgang der Selektion eines Ausschnitts aus einem Schema, wobei keine Transformation in ein anderes Datenmodell erforderlich ist. Beim Filtern wird demnach eine Teilmenge der Daten gebildet, welche einem zuvor festgelegten Kriterium entspricht, bspw. alle Vorgänge im Monat Mai, alle Wände höher als 3 m oder alle Leistungspositionen für Betonarbeiten. Das Kriterium kann beliebig komplex werden, muss jedoch für eine automatische Ausführung berechenbar sein. Auch zunächst willkürlich erscheinende Kriterien wie eine manuelle Selektion von Dateneinheiten (z. B. Stützen in einem 3D-Viewer) können ein solches berechenbares Kriterium bilden. In der Regel kann die Identität der betroffenen Dateneinheiten ermittelt werden, wodurch ein Zielmengenkriterium entsteht.

Filtern ist die Selektion eines Ausschnitts eines Baufachmodells entsprechend eines Kriteriums.

Die Größe der Datenmenge ist für die Dauer des Filtervorgangs maßgebend. Der Aufwand, um einen Algorithmus zur Berechnung der Ergebnismenge eines Filters zu erstellen, wird von der Komplexität des Datenmodells sowie der Komplexität des Kriteriums bestimmt. Die Überprüfung, ob eine bestimmte Dateneinheit zur Ergebnismenge eines Filters gehört, kann letztendlich nur durch die Auswertung seiner Datenelemente erfolgen. Demzufolge muss das Kriterium mit Bedingungen für mögliche Datenelemente formulierbar sein, d. h. Filterkriterien müssen kompatibel zum Schema sein oder in eine kompatible Form überführt werden können.

Um über genügend Ausdrucksstärke zur Formulierung komplexer Kriterien zu verfügen, werden in der IKT vorrangig textuelle Abfragesprachen zum Filtern verwendet.

Auf Meta-Modellebene definierte Filtersprachen

Für einige Meta-Modelle (Ebene MOF M2) sind Abfragesprachen einschließlich Ablaufumgebungen verfügbar. Beispielsweise ist SQL eine Abfragesprache für Relationale Datenbanken und verwendet die strukturellen Konzepte Tabelle, Spalte und Zeile (ISO 9075-1, 2008). XPath und XQuery sind populäre Abfragesprachen für XML und arbeiten auf Elementen und Attributen (W3C XPath 1.0, 1999; W3C XQuery 1.0, 2010; W3C XML 1.0, 2008).

Das im Bauwesen wichtige Schema IFC wurde in EXPRESS modelliert (ISO 16739, 2005; ISO 10303-11, 2004). Koonce et al. (1998) schlagen mit der EXPRESS Query Language (EQL) eine SQL-artige Filtersprache für EXPRESS vor. Mit EXPRESS-X existiert eine textuelle Mappingsprache zur Datentransformation zwischen EXPRESS-Schemas (ISO 10303-14, 2005). Zwar wurde EXPRESS-X nicht als Filtersprache entworfen, trotzdem kann sie zu diesem Zweck verwendet werden (Weise, 2006b).

Mit dem Generalized Model Subset Definition Schema (GMSD) wurde eine weitere Abfragesprache für EXPRESS entwickelt. GMSD bietet neben dem Filtern von Fachmodellen die Möglichkeit, auf Ebene MOF M1 EXPRESS-basierte Schemas zu Subschemas zu reduzieren (Weise et al., 2003). Bei sehr komplexen, mehrere Domänen umfassenden Datenstrukturen wie IFC ist es notwendig, fachspezifische Sichten – so genannte Model Views – zu erzeugen[11]. Mittels GMSD werden dabei auch die Fachmodelle (MOF M0) derart gefiltert, dass sie zu den Subschemas konform sind. Eine weitere Abfragesprache zur Erstellung EXPRESS-basierter Teilmodelle ist die Partial Model Query Language (PMQL; Adachi, 2002).

Abfragesprachen auf Meta-Modellebene haben einen großen Anwendungsbereich, da sie für alle Fachmodelltypen einsetzbar sind, die mit dem Meta-Modell spezifiziert wurden. Das ist im Bauwesen aufgrund der Vielfalt der Datenmodelle wichtig. Die Definition der Sprachen auf den Datenstruktur-Konzepten dieser Meta-Modelle führt jedoch zu einem hohen semantischen Abstraktionsgrad. Um konkrete Anfragen zu formulieren, müssen Benutzer das Schema (Ebene MOF M1) verstehen und baufachliche Kriterien in Kriterien auf Datenstruktur-Ebene umwandeln können. Direkte Anwender solcher Sprachen sind daher überwiegend Softwareentwickler. Diese Abfragesprachen sind für Endanwender ungeeignet.

Mit den Mitteln der Softwareentwicklung kann jedes Fachmodell auch ohne Abfragesprache gefiltert werden. Im Zuge einer Eigenimplementierung muss es dazu geparst und entsprechend des Kriteriums programmatisch verarbeitet werden. Für einige Datenformate und Programmiersprachen existieren Parser-Bibliotheken, welche Teile dieser Aufgaben übernehmen. Mit dem Standard Data Access Interface (SDAI) sind solche Bibliotheken für EXPRESS sogar genormt (ISO 10303-22, 1998). Einige Programmiersprachen bieten interne Navigationshilfen für Objektnetze, bspw. Groovy mit GPath oder Microsofts .NET-Sprachen mit Language Integrated Query (LINQ; Staudemeyer, 2007; Pialorsi & Russo, 2010). Letztendlich wird jedoch immer auf den Konzepten von Datenschema und Programmiersprache gearbeitet. Mit turing-vollständigen Programmiersprachen lässt sich jedes berechenbare Problem von Fachmodellen lösen. Dieses Prinzip kommt in Fachapplikationen zur Anwendung. Allerdings kann der hierzu notwendige Implementierungsaufwand erheblich sein.

Semantische Filtersprachen auf Schemaebene

Auf Schemaebene (MOF M1) definierte Filtersprachen sind Fachsprachen – so genannte Domänenspezifische Sprachen (engl. Domain Specific Language, DSL; vgl. Fowler, 2010). Da sie nur für *ein* Datenmodell definiert sind, können sie dessen Datenstruktur vor dem Nutzer verbergen. DSLs erlauben eine höhere fachliche Ausdrucksstärke, was zu einfacheren Kriterien-Formulierungen führt. Nutzer müssen die interne Datenorganisation des Fachmodelltyps nicht mehr im Detail kennen.

Hartmann (2009) schlägt ein Konzept einer Filter-Fachsprache für IFC-Modelle vor. Mazairac & Beetz (2012, 2013) präsentieren die laufende Arbeit zur Abfragesprache BIMQL[12]. Die Sprache soll das Filtern und die Manipulation (Create, Read, Update, Delete – CRUD) von IFC-Daten innerhalb des Open Source Frameworks *bimserver.org* (vgl. Beetz et al., 2010) vereinfachen. Es werden Sprachkonstrukte vorgeschlagen, die sowohl tief verschachtelte als auch indirekte und

[11] vgl. Model View Definition (MVD) `http://www.buildingsmart-tech.org/specifications/mvd-overview/mvd-overview-summary`

[12] `http://bimql.org/`

implizite Relationen komfortabel navigierbar machen. IFC-Klassennamen sollen dynamisch durch natürlichsprachliche Vokabeln austauschbar sein.

Domänenspezifische Filtersprachen für Geodaten werden von Graeff (2003) beschrieben. Im weitesten Sinn handelt es sich bei LDAP-Suchanfragen (vgl. Carter, 2003) auch um eine Fachsprache für Verzeichnisdaten wie Personen und Kontaktdaten.

Auch wenn die inhärente Datenstruktur bei domänenspezifischen Filtersprachen nicht offengelegt werden muss, können solche Sprachen nur Daten liefern, welche *explizit* im Datenmodell vorgesehen sind. Das schließt die zumutbare Anwendung von Logik wie Datumskonvertierung oder arithmetische Operationen auf Attributen ein.

Semantische Filtersprachen auf Anwenderebene

Semantische Filtersprachen sind domänenspezifische Abfragesprachen für einen bestimmten Fachmodelltyp, welche einen besonderen fachlichen Aspekt adressieren. Dazu stellen sie dem Nutzer Sprachkonstrukte auf Ebene dieser speziellen Problemstellung zur Verfügung. Sie ermöglichen eine prägnante Formulierung von Filterkriterien, welche vollkommen von der zugrundeliegenden Datenstruktur entkoppelt sein dürfen. Damit können sie Informationen liefern, welche nur implizit im Datenmodell abgebildet sind. Die Umwandlung von Fachkriterien in Datenstrukturkriterien erfolgt innerhalb der Ablaufumgebung dieser Sprachen und erfordert in der Regel komplexe Logik.

Borrmann (2007) präsentiert eine räumliche Abfragesprache für IFC-Modelle. Sie ermöglicht eine ausdrucksstarke Beschreibung von geometrischen und topologischen Lagebeziehungen von Bauteilen als Filterkriterium. Dazu werden metrische (z. B. *distance, closerThan, fartherThan*), direktionale (z. B. *above, below, northOf*) und topologische (z. B. *within, contain, touch*) Operatoren eingeführt. Tulke (2010) stellt eine Verknüpfungssprache für Bauwerksinformationsmodelle vor und wendet diese in der modellbasierten Terminplanung an. Ein Aspekt der Sprache ist die Identifikation von Bauteilen. Mit *inZoneRelation* und *betweenAxisRelation* werden dafür Funktionen zur räumlichen Lageermittlung bereitgestellt.

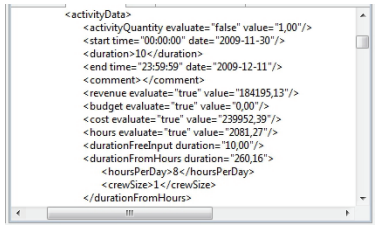
Weiterführende Literatur

Cerovsek (2008) gibt einen Überblick zu Techniken zur Anfrageformulierung im Bauwesen. Katranuschkov et al. (2010) beschreiben eine generelle Methodik zum Filtern komplexer Bauwerksmodelle. Windisch et al. (2012b) stellen ein Konzept zum einheitlichen Filtern von Fachmodellen unterschiedlicher Domänen vor.

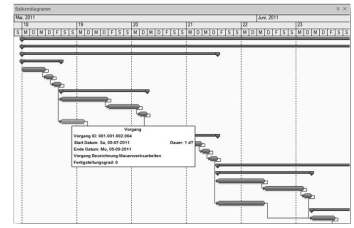
2.3. Domänenübergreifende Bauinformationsräume

2.3.1. Bereitstellung und Zugriff auf Daten

Informationen entstehen an der Schnittstelle Mensch-Computer durch automatisierte Aufbereitung vorhandener Daten und Interpretation durch einen Fachanwender. Die nutzbare Datenbasis sowie die Software zu ihrer Erschließung bestimmen somit den maximal möglichen Umfang



(a) Vorgangsdaten im Texteditor



(b) Vorgangsdaten in Terminplan-Software (Quelle: `http://www.rib-software.com/`)

Abbildung 2.6.: Unterschiedliche Informationsräume bei gleichen Daten

und Inhalt nutzbarer Informationen in Bauinformationsprozessen[13]. Dieses Potential wird hier durch den Begriff *Informationsraum* beschrieben. Allgemein wird der Begriff *Informationsraum* in ähnlichem Zusammenhang verwendet. Folgende Definitionen verdeutlichen dies:

> „Information space is a set of concepts and relations among them held by an information system. Information space is produced by a set of known procedures, and is changed through intentional manipulation of its content." (Newby, 1996, S. 1)

> „Informationsraum: Mehrdimensionale Struktur für Ordnung von, Navigation durch und Zugriff auf Informationen" (van Hoof, 2003, S. 17)[14]

> „Gemeinsame Informationsräume: unterstützen die implizite Kommunikation zwischen Teammitgliedern und dienen zur eher längerfristigen Informationsspeicherung. Zu dieser Klasse werden Bulletin–Board–Systeme, verteilte Hypertextsysteme, spezielle gemeinsam genutzte multiuserfähige Datenbanken wie Kalendersysteme gezählt." (Hauschild, 2004, S. 42)[15]

> „Der Informationsraum einer Web-Anwendung stellt die Fülle an Informationen und Inhalten dar, die einer Web-Anwendung zur Darstellung, Bereitstellung, Berechnung und Modifikation zur Verfügung stehen." (Nussbaumer, 2007, S. 7)

In dieser Arbeit steht die Vergrößerung des Informationspotentials vorhandener Daten im Vordergrund. Informationsräume werden daher wie folgt definiert:

Informationsraum: Das Potential ableitbarer Informationen aus, sowie anwendbarer Operationen auf gegebene Daten unter Inanspruchnahme verfügbarer Datenverarbeitung

Bauinformationsraum: Informationsraum des Bauwesens

Bauinformationsräume beschreiben den potentiellen Informationswert einer Technologie für einen Fachanwender. Die Technologie besteht dabei aus den Faktoren Daten und zugehörige Software. Abbildung 2.6 verdeutlicht diesen Zusammenhang. Für den Mensch als Adressat haben die identischen Vorgangsdaten in einem Texteditor weniger direkten Informationswert als

[13] Zwar kann auch Hardware, bspw. Cave Automatic Virtual Environment (CAVE), Rapid Prototyping oder Force Feedback, einen Beitrag zur Datenrepräsentation leisten; letztendlich handelt es sich jedoch auch nur um eine Darstellungsform der Daten, welche zu diesem Zweck durch eine Software aufbereitet wurden.

[14] Aus dem Forschungsprojekt PreBIS entnommen. `http://prebis.informatik.uni-leipzig.de/`

[15] Im Kontext von Computer Suported Cooperative Work in der Bauplanung.

in einer Terminplan-Software. Umgekehrt kann der Informationswert auch innerhalb derselben Software variieren – nämlich dann, wenn unterschiedliche Datenmodelle benutzt werden. So könnte ein Vorgangsdatenmodell Meilensteine erlauben, ein anderes jedoch nicht.

Zwar ist die Leistungsfähigkeit eines Informationsraumes auch von den möglichen Algorithmen abhängig, das Datenmodell hat bei allgemeinen theoretischen Betrachtungen jedoch den höheren Stellenwert. Dies ist u. a. dadurch begründet:

- Daten haben eine höhere Lebensdauer als Algorithmen
- Daten werden zwischen Anwendern ausgetauscht und gelten somit für alle kompatiblen Applikationen
- Es herrscht ein Gemeinsames Verständnis zur Interpretation von verbreiteten Datenmodellen
- Datenmodelle sind leichter zu beschreiben und zu standardisieren als Algorithmen
- Datenmodelle implizieren bereits offensichtliche Verarbeitungsmechanismen
- Algorithmen arbeiten auf dem Datenmodell und sind daher – bei zumutbarem Implementierungsaufwand – an dessen Informationspotential gebunden[16].

Kategorisierung

Bauinformationsräumen muss nicht zwangsläufig nur *ein* Datenmodell zugrunde liegen. Ebenso gibt es keine Beschränkung bei Domänen und Datenformaten. Tabelle 2.2 zeigt die Kategorisierung der Bauinformationsräume nach Format und Domäne ihrer Daten. Beide Aspekte werden dabei nach ihrer Gleichartigkeit differenziert. Wenn im Informationsraum nur *ein* Datenformat oder verschiedene Versionen desselben Datenformats verarbeitet werden können, handelt es sich dabei um ein homogenes Datenformat – anderenfalls um heterogene Datenformate. Wenn im Informationsraum nur Daten aus einem Fachgebiet verarbeitet werden können, handelt es sich dabei um eine homogene Domäne – anderenfalls um heterogene Domänen.

Traditionelle Fachanwendungen sind die Bauinformationsräume für homogene Domänen und homogene Datenformate. Beispiele sind 2D-CAD-Anwendungen, Terminplan-Software, Statikprogramme oder Applikationen zur Erstellung von Ausschreibungsunterlagen. Diese Art der Informationsräume ist gut erforscht und wird in dieser Arbeit nicht weiter untersucht. Heterogene Datenformate bei homogener Domäne werden in der Regel durch Import- und Export-Funktionen innerhalb traditioneller Fachanwendungen realisiert. Dabei kommen Transformationsfunktionen zum Einsatz. Diese werden in Abschnitt „Mapping" auf Seite 34 im

Tabelle 2.2.: Kategorisierung von Bauinformationsräumen

Datenformate \ Domänen	homogen	heterogen
homogen	Traditionelle Fachanwendungen	Produktdatenmodelle
heterogen	Import / Export Konverter	Integrierte Systeme / Multimodelle

[16] vgl. Abschnitt „Semantische Filtersprachen auf Schemaebene" auf Seite 24

Zusammenhang mit integrierten Systemen näher betrachtet. Produktdatenmodelle ermöglichen die Abbildung heterogener Domänen in einem homogenen Datenformat. Sie werden im folgenden Abschnitt beschrieben. Informationsräume mit heterogenen Domänen und heterogenen Datenformaten werden bislang nur durch integrierte Systeme unterstützt[17]. Mit dieser Arbeit wird ein neuer Ansatz in Form von Multimodellen vorgestellt.

2.3.2. Produktdatenmodelle

Produktdatenmodelle sind komplexe Datenmodelle zur Abbildung relevanter Daten eines Produktes. Bauwerksmodelle sind die Produktdatenmodelle des Bauwesens. Oftmals wird auch die Bezeichnung *Building Information Model* (BIM) verwendet. Aktuellere Publikationen definieren BIM jedoch als Methode (*Building Information Modeling*) und nicht als Bauwerksmodell. Die relevanten Bauwerksmodelle sind die Industry Foundation Classes (IFC)[18], Green Building XML (gbXML)[19] sowie CIMSteel Integration Standards (CIS/2)[20]. Trotz ihres Namens können Bauwerksmodelle neben der Bauwerksbeschreibung (bspw. Geometrie, Materialien, Einbauten, Umgebung) auch Daten über den Planungs- und Produktionsprozess (bspw. Vorgänge, Leistungsverzeichnisse, Organisationsdaten) abbilden. Abbildung 2.7 auf der nachfolgenden Seite zeigt im oberen Teil exemplarisch die Domänen von IFC. Produktdatenmodelle und ihre Softwareanwendungen bilden somit Informationsräume mit heterogenen Domänen und homogenem Datenformat. Eine prinzipielle Betrachtung zur Nutzung von Bauwerksmodellen als gemeinsamen Informationsraum liefert Hauschild (2004). Weise et al. (2009) demonstrieren die domänenübergreifende Nutzung von IFC am Beispiel modellbasierter Ablaufplanung, Tulke (2010) präsentiert hierfür eine Verknüpfungssprache. Tauscher (2011) zeigt, wie Terminpläne aus gegebenen Bauwerksdaten generiert werden können. Das domänenübergreifende Resultat kann in IFC hinterlegt werden.

In Bauinformationsprozessen entstehen große Datenmengen, welche in Produktdatenmodellen hinterlegt werden. So genannte *Model Server* werden deswegen als Technologien zur Bewältigung der großen Fachmodelle, zur Bildung und Integration von Teilmodellen sowie zur Unterstützung der verteilten Bearbeitung eingesetzt (bspw. EUROSTEP Model Server[21], EDMServer[22]) und weiterentwickelt (vgl. Kiviniemi et al., 2005a; Beetz et al., 2010).

Die Komplexität von Produktdatenmodellen ist durch die große Anzahl verwendeter Klassen sehr hoch (vgl. bspw. Abbildung 2.7). Darüber hinaus wird zum Beispiel bei IFC ein indirektes Modellierungskonzept verwendet, bei dem zusammengehörende Daten über weitläufige Assoziationen verbunden sind. Dies macht die manuelle Handhabung von Produktdatenmodellen nahezu unmöglich und erschwert die Implementierung von Anwendungen. Daher werden Hilfestellungen zum Zugriff auf solche Datenmodelle vorgeschlagen (Katranuschkov et al., 2003) oder Filter eingesetzt[23]. Weitere Forschungsgebiete sind u. a. die Änderung und Versionierung des Meta-Modells (Amor et al., 2002; Nour & Beucke, 2008) sowie die Verfolgung von Änderungen bzw. die Versionierung der Fachmodelldaten (Weise, 2006a; Koch, 2008; Nour & Beucke, 2010). Des Weiteren entstehen in verteilten Informationssystemen (concurrent engineering)

[17] vgl. Abschnitt 2.3.4 auf Seite 33
[18] http://www.buildingsmart.org/standards/ifc
[19] http://www.gbxml.org/
[20] http://www.aisc.org/
[21] http://www.eurostep.com/
[22] http://www.jotne.com/
[23] vgl. Abschnitt 2.2.3 auf Seite 23

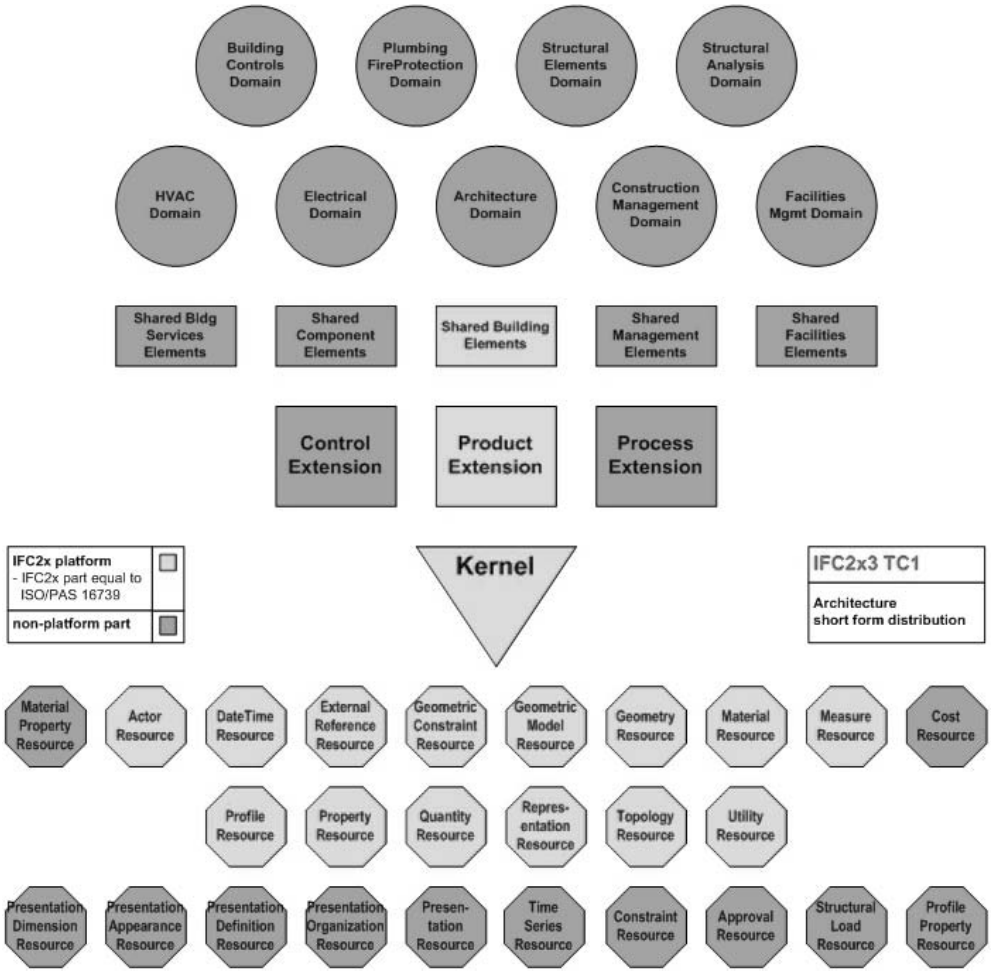


Abbildung 2.7.: Architektur des Produktdatenmodells IFC 2x3 (Quelle: `http://www.buildingsmart-tech.org/`)

Probleme der Daten- und Modellkonsistenz. Einen Überblick über die Herausforderungen und Lösungsmöglichkeiten auf diesem Gebiet geben Scherer & Katranuschkov (2006).

Die Komplexität des Produktdatenmodells IFC wird in empirischen Studien als *ein* Grund der bislang schlechten Übernahme in die industrielle Praxis genannt (Tse et al., 2005; Howard & Björk, 2008). Die Ursache der Datenmodell-Komplexität ist die Komplexität der Planungs- und Bauprozesse an sich sowie die historische Entwicklung der Bauwerksmodelle.

Das General AEC Reference Model (GARM; Gielingh, 1988) ist ein Meta-Modell für die Bereiche Architektur, Bauingenieurwesen, Anlagen- und Schiffsbau. Ein relevantes Merkmal ist die domänenübergreifende Abbildung der Lebenszyklusphasen von Produkten wie Bedarfsanalyse, Planung, Produktion, Änderung und Verwertung. Im RATAS-Projekt (Björk, 1994) wurden in den 1980er und 1990er Jahren bereits grundlegende Ideen heutiger Bauwerksmodelle wie

Abstraktion, Relationen und Komponentenhierarchie entwickelt. Björk (1995) schlägt daraufhin ein Bauwerksmodell bestehend aus einem *Building Kernel Model* zur Abdeckung von Anwendungsdaten aller Phasen und Domänen sowie weiteren Aspect Models zur Abdeckung von Anwendungsdaten spezieller Phasen und Domänen vor. In den Projekten ATLAS (Tolman & Poyet, 1994), COMBINE (Augenbroe, 1994) und VEGA (Junge et al., 1997) wurden weitere Ansätze und Vorschläge zur Modellierung domänenübergreifender Daten des Bauwesens erarbeitet.

Im Projekt COMBI wird eine Mappingsprache für die Integration von Modellen unterschiedlicher Domänen vorgestellt (Scherer, 1994b; Katranuschkov & Scherer, 1996). Steinmann (1997) schlägt Bauwerksmodelle mit dynamischen Strukturen zur Unterstützung des Entwurfsprozesses vor. Eastman (1999) präsentiert Architekturen zur Integration verteilter Bauwerksmodelle. Ibrahim et al. (2004) stellen die Ansätze zu zentralem Produktdatenmodell und zu verteilten Modellen gegenüber. Ein Überblick zu verteilten Produktdatenmodellen wird von Firmenich & Rank (2007) gegeben.

Im COMBI-Folgeprojekt ToCEE (Scherer, 1998) wurden die erarbeiteten Konzepte im Drei-Schichten-System der IFC abgebildet und dafür, sowie für Dokumente und Prozessmodelle, Serverstrukturen bereitgestellt und auf ihre Verlinkung hingewiesen. Im Projekt ISTforCE (Katranuschkov et al., 2001) wurden auf dieser Basis Infrastrukturen, Services und User Interfaces zur Projektkommunikation entwickelt. Die Fortführung erfolgte mit dem Projekt iCSS (Juli & Scherer, 2002). Dort wurde ein Übergang von der dokumentenbasierten Bearbeitung zu einer produktmodellbasierten Bearbeitung mit Produktdatenservern erarbeitet. Als Maßnahme zur Integration wurde gezeigt, wie Dokumente mit Produktdaten zu verknüpfen sind. Im Projekt InteliGrid wurde auf Basis von Beschreibungslogik und Ontologien ein mehrschichtiges Framework zur Unterstützung von verteiltem Datenzugriff und -verarbeitung in virtuellen Organisationen geschaffen (Gehre et al., 2005; Dolenc et al., 2007). Beetz (2009) zeigt, wie Bauwerksmodelle gänzlich in Beschreibungslogik und Ontologien modelliert oder in diese überführt werden können.

Eine Zusammenfassung über die Entwicklungsgeschichte und den Standardisierungsprozess von IFC geben Laakso & Kiviniemi (2012).

Bauwerksmodelle unterliegen in ihrer Konzeption als domänenübergreifender Informationsraum verschiedenen, sich gegenseitig bedingenden Faktoren. Für die Datenmodellierung besteht ein Zielkonflikt zwischen der allgemeingültigen, vereinheitlichten und wiederverwendbaren Abbildung von Daten einerseits sowie der flexiblen, individuellen Anwendbarkeit in möglichst vielen Domänen und Projekten andererseits. Datenmodelle und ihre erschließende Software bewegen sich im Spannungsfeld neu aufkommender oder konkurrierender IKT-Technologien wie STEP, UML, XML oder Ontologien. Die wachsende Leistungsfähigkeit von Computern sowie die mit Grid- und Cloud-Technologie verfügbaren Ressourcen erlauben einen Effizienzausgleich nicht optimaler Datenmodelle und Algorithmen durch Rechen- und Speicherkapazität.

Erweiterung von Datenmodellen

Die Anwendungsgrenzen von Baufachmodelltypen sind durch deren Domäne beschränkt. Dies gilt auch für Produktdatenmodelle mit mehreren Domänen. Es existieren jedoch domänenübergreifende Bauinformationsprozesse, deren Daten sich aufgrund solcher Anwendungsgrenzen nicht vollständig durch *ein* Datenmodell abbilden lassen. Dazu zählen bspw. die Kombinationen von

- Baustellen- mit Unfallschutzdaten (Kamardeen, 2010)
- Gebäude- mit Energiedaten (Katranuschkov et al., 2011)
- Baustellen- mit Vorgangsmodellen (Kim et al., 2011; Chavada et al., 2012)
- Bauwerks- mit Vorgangsmodellen (Kugler et al., 2011)
- Bauwerks- mit Bodenmodell (Hilfert & Hegemann, 2012)
- Baustellen- mit Geodaten (Irizarry & Karan, 2012)

Hartmann et al. (2008) zeigen weitere Fallstudien und Anwendungsbeispiele für 3D/4D-Modelle auf.

Ein Lösungsmöglichkeit ist die Ausdehnung der Anwendungsgrenzen eines Datenmodells durch Erweiterung des Schemas. Dabei werden neue Datenelemente für die Konzepte der zusätzlichen Domänen hinzugefügt. Solche Schemaerweiterungen sind bei den meisten Baufachmodelltypen technisch möglich. Für Produktdatenmodelle wie IFC ist eine solche Erweiterung sogar konzeptuell vorgesehen. Dementsprechend existieren auch Erweiterungs- und Ersatzvorschläge für beispielsweise:

- Tragwerksysteme (Weise et al., 2000)
- Brücken (Yabuki et al., 2006)
- Bodensysteme (Zobl & Marschallinger, 2008)
- Straßen, Brücken und Tunnel (Lee & Kim, 2011)
- Geodaten und Verkehrsinfrastruktur (Esfahani et al., 2012)
- Tunnelbohrmaschinen (Hegemann et al., 2012)

Obwohl Schemaerweiterungen technisch möglich sind, können die Interoperabilitätsprobleme domänenübergreifender Bauinformationsprozesse damit nicht generell gelöst werden. Dies hat folgende organisatorischen Gründe:

1. Bilaterale Schemaerweiterung

   Die oben genannten Beispiele verdeutlichen, dass sich Erweiterungsbestrebungen in der Regel bottom-up vollziehen. Das bedeutet, einzelne domänenübergreifende Problemstellungen der Praxis werden analysiert und durch eine spezielle Schemaerweiterung gelöst. Damit die Schemaerweiterung für reale Datenaustauschprozesse wirksam wird, muss betroffene Software entsprechend geändert oder neu implementiert werden. Dies ist normalerweise aufwändig. Daher kann sich eine spezielle Schemaerweiterung meist nur innerhalb eines geschlossenen Nutzerkreises etablieren. Meist entsteht sie im Zusammenhang mit einer prototypischen Neuentwicklung einer domänenübergreifenden Fachanwendungen oder geschieht für eine konkrete Weiterentwicklung einer Applikation.

2. Notwendige Akzeptanz für multilaterale Schemaerweiterung

   Um eine Schemaerweiterung dem offenen Nutzerkreis zugänglich zu machen, muss sie eine notwendige Akzeptanz erfahren. Dazu bedarf es einer Einigung der Anwender über Inhalt und Struktur der Schemaerweiterung. Eine solche Einigung wird vorrangig durch die Umsetzung in einem offenem Standard widergespiegelt. Die dafür notwendigen Standardisierungsverfahren sind langwierig und haben oftmals einen Kompromiss zum Ergebnis, bei dem der ursprüngliche Vorschlag geändert oder verallgemeinert zur Anwendung kommt.

3. Beherrschbarkeit eines Superdatenmodells

   Die fortwährende Erweiterung eines Datenmodells führt letztendlich zur Realisierung eines Superdatenmodells, welches die Daten aller möglichen Bauinformationsprozesse abbilden kann. Es ist zwar technisch möglich, ein endlich großes Superdatenmodell zu konstruieren – dieses ist aber nicht mehr durch den Menschen beherrschbar. Zum einen bedarf es der Koordination eines globalen, abschließenden und möglichst widerspruchsfreien Modellierungsprozesses, zum anderen müssten die erarbeiteten Konzepte durch die Anwender gefunden, verstanden und korrekt benutzt werden. Es ist nicht davon auszugehen, dass diese Komplexität und Harmonisierung organisatorisch und in einem vernünftigen Zeitraum zu bewältigen ist.

2.3.3. nD-Modelling

Trotz ihrer Unterstützung für unterschiedliche Domänen werden Bauwerksmodelle in der industriellen Praxis hauptsächlich zum Austausch von geometriebasierten CAD-Daten verwendet (Gu & London, 2010). *nD-Modelling* bezeichnet die Anwendung von Bauwerksmodellen zur Modellierung fachspezifischer Aspekte *zusätzlich* zur Bauwerksgeometrie. Dies umfasst sowohl Problemstellungen *einer* Domäne als auch domänenübergreifende Aufgabenstellungen. Beispiele möglicher Fachanwendungen (Domain Applications) sind im oberen Abschnitt der Abbildung 2.8 auf der nachfolgenden Seite wiedergegeben. *nD-Modelling* ist dabei nicht auf die im Bauwerksmodell möglichen Domänen beschränkt. Vielmehr handelt es sich um ein Konzept zum integrierten Einsatz von Fach- und Bauwerksdaten, auch aus unterschiedlichen Datenmodellen. Ziel ist die Erstellung eines multidimensionalen Computermodells zur Unterstützung des gesamten Planungs- und Bauprozesses. Der Begriff wurde durch das Projekt *3D to nD Modelling* geprägt, wobei das IFC-basierte 3D-Geometriemodell im Mittelpunkt stehen soll und die anderen Fachmodelle (4...nD) als dessen Erweiterung fungieren (Lee et al., 2003). Die multidimensionale, domänenübergreifende Repräsentation der Daten soll dabei primär in einer virtuellen 3D-Welt erfolgen (vgl. z. B. Fu et al., 2006). Als Ergebnis mehrerer *nD-Modelling*-Workshops schlagen Lee et al. (2007) das Konzept eines Technologie-Framework vor (vgl. Abbildung 2.8):

> „As a result of the five workshops, a new technology framework [...] emerged to support the nD modelling roadmap [...]. The technology framework that provides the architecture for different domain applications which are directed by business process is based on a service-oriented platform supported by various technologies such as visualisation, decision support and analysis:
> [...]
> The services include technology service and data service. The technology service provides common technologies, such as visualising the building data, multi-aspects decision support, data analysis for thermal, structure and so on and process-control mechanism. The data service provides the data access to two types of data sources, building data and domain data. Building data can be described as data definition and representation of the building model and it has to be supported by interoperable data standards such as IFCs. Domain data is specific data related to each domain such as regulations for building accessibility, weather data is for energy simulation and so on. Currently, these two data sources are often not linked. It is suggested that research has to be done to integrate the two data sources through developing common concepts (ontology/classification/dictionary)." (Lee et al., 2007, S. 346 ff.)

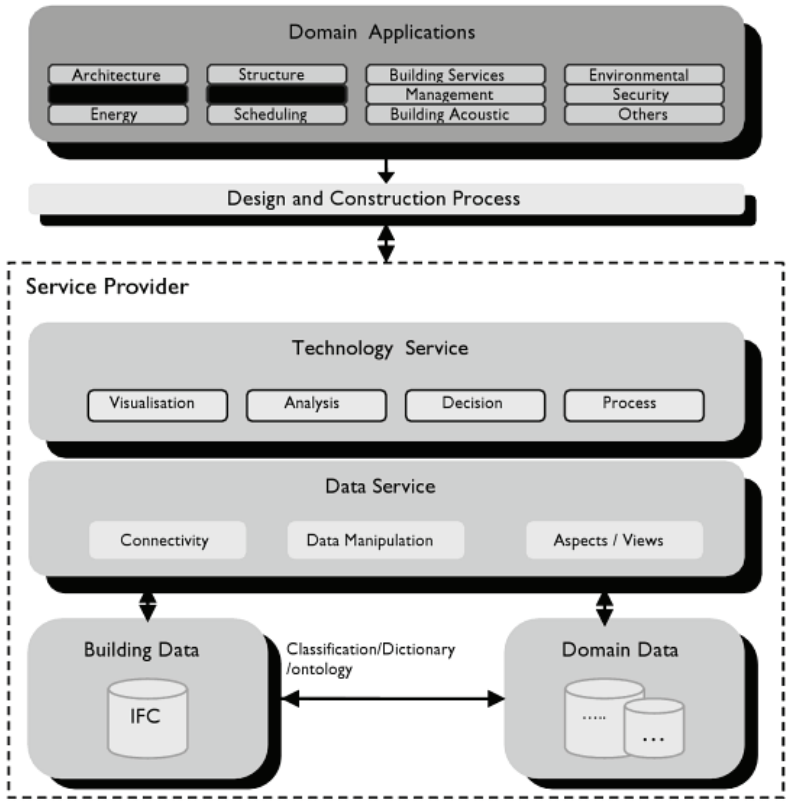


Abbildung 2.8.: nD modelling technology framework (aus Lee et al., 2007)

Eine wichtige Annahme ist dabei, dass die Nutzung der Fachanwendungen durch Geschäftsprozesse gesteuert wird. Im Kontext dieser Arbeit können diese mit Bauinformationsprozessen gleichgesetzt werden. Wesentliches Merkmal des Frameworks ist die Trennung des Bauwerksmodells von anderen Fachmodellen, da diese nicht verlinkt sind. Daraus ergibt sich die Anforderung zur Erforschung der Integration der getrennten Datenressourcen. Es wird die Nutzung einheitlicher Konzepte wie Ontologien, Klassifikationen oder Wörterbücher vorgeschlagen. Das bedeutet, dass die Datenmodelle nicht zwangsläufig zu einem allumfassenden gemeinsamen Datenformat, dem so genannten Super-Produktdatenmodell, zusammengeführt werden müssen. Vielmehr werden lose gekoppelte Bauinformationsräume mit heterogenen Domänen und heterogenen Datenformaten gefordert.

2.3.4. Integrierte Systeme

Die Realisierung von Bauinformationsräumen mit heterogenen Domänen und heterogenen Datenformaten findet heute durch integrierte Systeme statt. Dabei werden alle relevanten, ursprünglich heterogenen Daten in ein gemeinsames internes Zwischenformat überführt, auf welchem später alle Datenoperationen stattfinden (vgl. auch Abbildung 2.10 auf Seite 38).

Das Zwischenformat wird im Allgemeinen nach seiner Unterstützung für vorhandene Abfragesprachen oder Verknüpfungsmechanismen gewählt. So kommen beispielsweise relationale, objektorientierte oder NoSQL-Datenbanken sowie Ontologien als Zwischenformat in Frage. Integrierte Systeme sind Plattformlösungen, die eine enge Bindung von Daten und Zugriffsystem besitzen.

Der Standard ISO 10303-1 (STEP, 1994) beschreibt einen Ansatz zur Integration von Partialmodellen mit unterschiedlichen Datenformaten zur Produktbeschreibung. Die Integration geschieht durch die Nutzung von domänenspezifischen Anwendungsprotokollen, bspw. für Baustatik, sowie die Verwendung gemeinsamer Meta-Datenstrukturen (Jardim-Gonçalves & Steiger-Garção, 2002). Diaz & Petersen (2002) beschreiben ein Projektkommunikationssystem, bei dem vorrangig Dokumente mit einem Gebäudestrukturmodell verknüpft werden. Willenbacher (2002) schlägt eine linkbasierte Integrationsplattform vor, bei welcher die Daten in getrennten Partialmodellen hinterlegt und durch Links verbunden werden. Die Aktualisierung der Daten wird dabei durch die Plattform gewährleistet. Der Ansatz fokussiert die dynamische und semiautomatische Erstellung sowie das Management von Mappings zwischen den Partialmodellen, umgesetzt durch einen hybriden Modellansatz mit zentraler Komponente (Willenbacher & Hübler, 2004). Van Nederveen & Tolman (2001) schlussfolgern jedoch aus dem ATLAS-Projekt, dass sich solche Ansätze aufgrund der komplexen Vorschriften zur Verlinkung in der Praxis nicht umsetzen lassen. Rüppel et al. (2006) schlagen eine agentenbasierte Plattform zur Integration von Partialmodellen vor. Riedel & Wender (2007) präsentieren eine Zugriffsmethode für einen Partialmodellverbund. Im Bereich des Mobile Computing schlägt Menzel (2007) ein Datawarehouse für die mehrdimensionale Datenhaltung vor und Scherer & Reinhardt (2007) beschreiben den Zugriff auf diesen Informationsraum. Nour (2008) stellt eine GUI zur Handhabung von Partialmodellen vor. Jeong (2008) präsentiert einen Kollaborationsansatz, bei dem domänenspezifische Datenformate verteilt erhalten bleiben und Daten mithilfe von Mappings in ein neutrales Format umgewandelt werden. Ye (2009) zeigt eine Methode zum Zugriff auf Daten des Bauprojektmanagements durch den Einsatz eines zentralen Datenmodells und Data Warehouse-Techniken sowie fortgeschrittener Visualisierungstechnologien. Wender (2009) beschreibt ein Informationszugriffssystem, bei welchem Navigationslinks zur Erschließung des Informationsraumes zum Einsatz kommen. Die Datenhaltung geschieht dabei wie von Willenbacher beschrieben (Wender & Hübler, 2009). Baumgärtel et al. (2012) stellen ein virtuelles Labor für Energieeffizienz-Simulationen vor. Die heterogenen Fachmodelle werden in das RDF-Format (W3C RDF, 2004) konvertiert und in einer Ontologie integriert. Curry et al. (2012) nutzen einen vergleichbaren Ansatz[24] für Energieanalyse und Nachhaltigkeitsbetrachtungen. Redmond et al. (2012) schlagen aufgrund einer empirischen Studie einen plugin- und cloudbasierten Datenaustausch in Bauinformationsprozessen vor.

Einen generellen Überblick über Systemintegration im AEC-Sektor geben Shen et al. (2010).

Mapping

Die Datenintegration der Quellmodelle in das Zwischenformat ist ein Kernaspekt integrierter Systeme. Einen Überblick über die Methodik und den Stand der Forschung geben Conrad (1997) sowie Leser & Naumann (2007). Bei der Datenintegration müssen zuerst Korrespondenzen zwischen Quellschema und Zwischenformat automatisch oder manuell gefunden werden und

[24] vgl. auch Abschnitt „Linked Data" auf Seite 39

danach deren Daten so umgeformt werden, dass sie zum Zwischenformat konform sind. Dieser Vorgang wird Modelltransformation genannt.

Modelltransformation Bei einer Schema- bzw. Datenmodelltransformation werden alle Schemaelemente des Ausgangsschemas in semantisch äquivalente Konstrukte des Zielmodells übertragen (Conrad, 1997). Dabei können Transformationsfunktionen zum Einsatz kommen.

Transformationsfunktion Eine Transformationsfunktion wandelt die Ausgangsdaten eines Quellmodells um, sodass sie zum Zielmodell konform sind.

Mapping Der Begriff *Mapping* steht für eine weitere Art von Transformationsfunktionen. Unter Mapping versteht man zum einen eine Menge von Korrespondenzen zwischen Attributen unterschiedlicher Schemata. Diese werden Wertkorrespondenzen genannt. Zum anderen bezeichnet es einen komplexen Prozess, der, ausgehend von den Wertkorrespondenzen, komplexere Schemakorrespondenzen und schließlich Datentransformationsvorschriften ableitet (Leser & Naumann, 2007).

Das automatische Auffinden von Wertekorrespondenzen wird *Matching* genannt.

> „Mit Schema **Matching** werden Verfahren bezeichnet, die Korrespondenzen zwischen semantisch äquivalenten Elementen zweier Schemata automatisch zu erkennen versuchen“ (Leser & Naumann, 2007, S. 144)

Die Herausforderung für integrierte Systeme ist das *vollständige* Mapping der heterogenen Quellmodelle in das Zwischenformat. Dabei ist je nach Anwendungszweck des integrierten Systems *Vollständigkeit* wie folgt zu verstehen:

1. Spezieller domänenübergreifender Informationsraum

   Das integrierte System wurde zur Lösung einer fachspezifischen Aufgabenstellung, bspw. modellbasierte Mengenermittlung oder Vorbereitung, Durchführung und Auswertung von Logistiksimulationen, entworfen. Das Datenmodell im Zwischenformat enthält dabei nur die zur Aufgabenstellung notwendigen Daten. Bei der Erstellung der Mappings ist die vollständige Befüllung des Zwischenformats als Zielmodell maßgeblich. Es ist unerheblich, ob auch alle Daten der heterogenen Quellmodelle überführt werden. Aufgabenbezogene integrierte Systeme kommen daher mit einem kleineren Zwischenmodell und weniger Mappings aus. Sie besitzen jedoch einen eingeschränkten Anwendungsbereich, da nicht auf die vollständigen originalen Informationen zugegriffen werden kann.

2. Absoluter domänenübergreifender Informationsraum

   Das integrierte System wurde zur Datenintegration geschaffen. Das Zwischenformat muss die Vereinigungsmenge aller Konzepte der heterogenen Quellmodelle abbilden. Bei der Erstellung der Mappings ist die vollständige Überführung der heterogenen Quelldaten maßgeblich. Absolute integrierte Systeme besitzen daher ein größeres Zwischenmodell und eine hohe Anzahl Mappings. Dafür kann auf die vollständigen originalen Informationen zugegriffen werden, wodurch keine Einschränkung des Anwendungsbereiches entsteht.

Der Entwurf von Modelltransformationen ist komplex. Matching und Transformationen sind nicht immer eindeutig, sodass Qualitätsverluste möglich sind (vgl. Fröbel et al., 2011). Ursachen sind semantische Abweichungen zwischen Quell- und Zielmodell; bspw. eine unterschiedliche Datenmodellierung gleicher Realsachverhalte oder umgekehrt: eine ähnliche Modellierung verschiedener Realsachverhalte. Des Weiteren können, bspw. bei geometrischen Repräsentationen,

mathematische Überführungen nur näherungsweise geschehen. Die vollständige und korrekte Modelltransformation von einem komplexen Baufachmodell A in ein Modell B und zurück ($A \mapsto B \mapsto A$) bleibt in der Praxis ein Ideal.

Für die technische Umsetzung stehen allgemeine Transformationssprachen zur Verfügung, bspw. QVT (OMG QVT, 2011)[25] oder EXPRESS-X (ISO 10303-14, 2005). Katranuschkov (2000) gibt einen Überblick über Mappingverfahren im AEC-Bereich und schlägt eine spezifische Mappingsprache vor. Des Weiteren werden allgemein gültige Transformationsmuster vorgestellt. Transformationsmuster sind Beschreibungen von Modelltransformationen und Transformationsfunktionen, die dazu geschaffen sind, ein allgemeines Transformationsproblem in einem bestimmten Kontext zu lösen. Transformationsmuster können in Anlehnung an die Entwurfsmuster, bekannt aus dem Bereich des Software Engineering (Gamma et al., 2004), entwickelt werden. Spezifische Transformationssprachen sind notwendig, um die Charakteristika des Anwendungsbereichs zu unterstützen. Deren Implementierung wird zunehmend einfacher, beispielsweise durch Werkzeuge zur Entwicklung von DSLs (vgl. bspw. Diedrichsen, 2008).

2.3.5. Domänenübergreifende Links

Beziehungen zwischen Daten können durch Links ausgedrückt werden. *Link* ist dabei ein Überbegriff für einen Verweis auf Daten. Je nach eingesetzter Technologie müssen die Dateneinheiten, auf die verwiesen wird, explizit oder implizit identifizierbar sein[26].

Link Ein Link ist ein Verweis auf Daten. Links können selbst Daten sein.

Olbrich (1998) analysiert das Modellieren mit Relationen in der Bauinformatik und schlägt Entwurfsmuster dahingehend vor. Es werden vorrangig binäre Relationen untersucht. Die Betrachtungsweise ist unabhängig von bestehenden Baufachmodelltypen. Es kann geschlussfolgert werden, dass die Konzepte sowohl innerhalb *eines* als auch zwischen verschiedenen Datenmodellen angewendet werden können. Hanff (2003) beschreibt Abhängigkeiten in objektorientierten Systemen und stellt Methoden zur verzögerten Aktualisierung der Objekte vor. Fröbel et al. (2009) untersuchen allgemeine Muster zur Kopplung von Baufachanwendungen. Die Analyse geht über den Bereich der Datenstrukturen hinaus und betrachtet auch Verhaltens-, Verteilungs- und Übertragungsmuster.

Das grundlegende Konzept von Verweisen ist Bestandteil der meisten Modellierungssprachen und wird innerhalb vieler Baufachmodelltypen angewendet. Im Zusammenhang mit domänenübergreifenden Datenmodellen und heterogenen Datenformaten sind auch die Beziehungen über Fachmodellgrenzen hinweg zu betrachten.

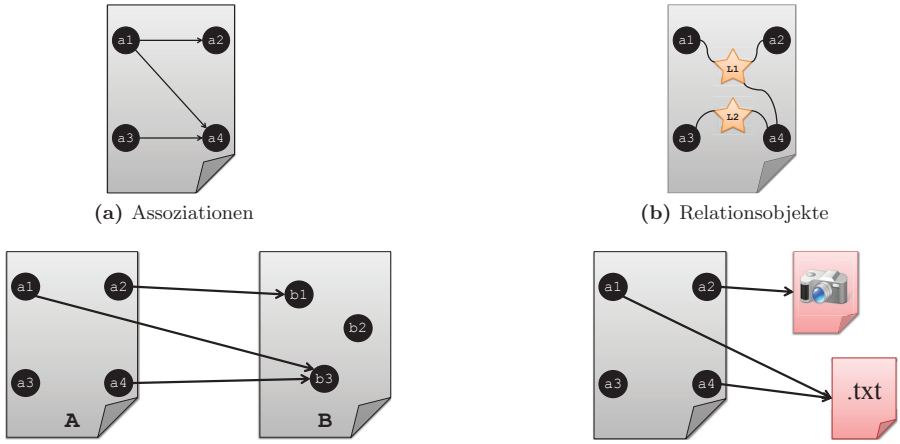
Linksysteme der Fachmodelle

1. Referenzen auf Objekte innerhalb desselben Fachmodells

   a) Assoziationen (vgl. Abbildung 2.9a)

      Assoziationen sind direkt im Datenmodell vorgesehene Referenzen auf andere Dateneinheiten derselben Modellinstanz. In XML-basierten Modellen werden diese

[25] QVT ist teilweise informal. Vorschläge zur Formalisierung sind jedoch vorhanden (bspw. von Garcia, 2008).
[26] vgl. auch Abschnitt 2.1.4 auf Seite 16



(a) Assoziationen **(b)** Relationsobjekte

(c) Externe Referenzen **(d)** Dokumentreferenzen

Abbildung 2.9.: Linksysteme der Fachmodelle

Verweise z. B. durch Verschachtelung (bei Containment-Referenzen) oder über ID-Referenzen, bei Relationalen Datenbanken über Fremdschlüssel hergestellt. Domänenübergreifende Informationen entstehen durch Assoziationen zwischen Dateneinheiten unterschiedlicher Fachrichtungen, z. B. innerhalb von Produktdatenmodellen.

b) Relationsobjekte (vgl. Abbildung 2.9b)

In Analogie zu Mappingtabellen von Datenbanken können Assoziationen auch in eigenständigen Datenobjekten definiert sein. Diese Art der Modellierung ermöglicht eine losere Bindung zwischen den Dateneinheiten, da diese ihre Beziehungen untereinander nicht mehr explizit kennen müssen. Dies erlaubt zum einen die übersichtlichere Datenmodellierung bei *vielen* vorgegebenen Relationen, zum anderen können auf Basis neutraler Beziehungsobjekte *dynamisch* durch den Nutzer beliebige Beziehungen in der Modellinstanz angelegt werden. Während Mappingtabellen in relationalen Datenbanken technisch notwendig sind um $n : m$-Beziehungen abzubilden, basiert bspw. bei IFC der Großteil der Referenzen auf dem Modellierungskonzept der Relationsobjekte (`IFCRelAssociates`), um beliebige, auch domänenübergreifende, Beziehungen zu ermöglichen.

2. Referenzen auf Objekte in anderen gleichartigen Fachmodellen (vgl. Abbildung 2.9c)

Besteht die Datenbasis aus mehreren Fachmodellen gleichen Typs, müssen Dateneinheiten über die Modellgrenzen hinweg referenzierbar sein, damit ein vereinigter Informationsraum gebildet werden kann. Einige Fachmodelltypen wie Microsoft Project unterstützen diese Funktion nativ (z. B. über Haupt- und Teilprojekte). Im Bereich der Produktdatenmodelle gibt es umfangreiche Forschungsarbeiten zu verteilten Produktdatenmodellen (siehe auch Firmenich & Rank, 2007). Kiviniemi et al. (2005b) schlagen in diesem Zusammenhang eine Erweiterung von IFC vor, welche die Referenzierung von externen Objekten in anderen IFC-Modellen ermöglicht. Allgemein ergibt sich durch die Verwendung von fachmodellbasierten, extern gerichteten Objektreferenzen immer mindestens *ein* führendes Modell, welches Eigentümer der Links ist. Damit ist das führende Modell von den

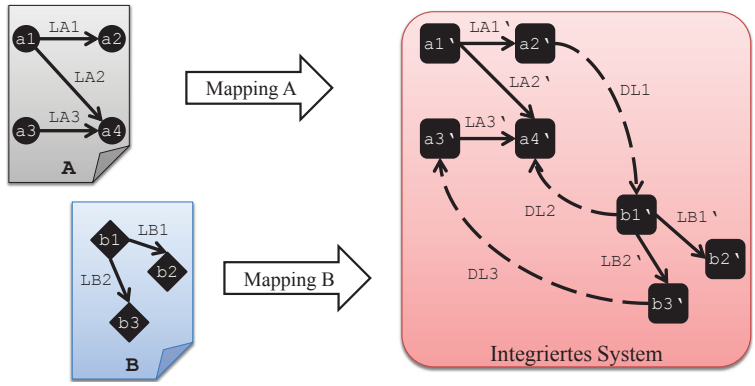


Abbildung 2.10.: Links in integrierten Systemen

referenzierten Fachmodellen abhängig und in seiner Konsistenz an deren Lebenszyklus gebunden.

3. Referenzen auf externe Dokumente (vgl. Abbildung 2.9d)

   Fachmodelle können Referenzen auf externe Dokumente enthalten und somit Informationen bereitstellen, die über das Anwendungsgebiet des Fachmodells hinausgehen. Dokumente können bspw. Texte, Bilder, Webseiten oder auch andere Fachmodelle sein. Der Verweis ist in der Regel eine URL oder ein Dateipfad und beinhaltet nur die Information über das Dokument an sich, nicht jedoch über Objekte innerhalb des Dokuments. Die Möglichkeit zur Referenzierung externer Dokumente muss im Schema vorgesehen sein, wie zum Beispiel bei Microsoft Project (Hyperlink-Referenzfelder), IFC (`IFCExternalReference`) oder CityGML (`ExternalReference`; OGC CityGML, 2012). Im ICCSS-Projekt (Juli & Scherer, 2002) wurde dazu alternativ auch eine Verlinkung vom Dokument zu den einzelnen Objekten des Objektmodells vorgeschlagen, da diese beim Konstruieren von 2D und 3D in einem CAD-System automatisch erfolgen kann.

Links in integrierten Systemen

Durch die normalisierte Abbildung aller Daten im einheitlichen Zwischenformat existiert in integrierten Systemen quasi nur *ein* homogenes Fachmodell. Somit können dort die gleichen Referenzierungsmethoden angewendet werden wie unter Abschnitt „Linksysteme der Fachmodelle" auf Seite 36 beschrieben.

1. Referenzen auf Objekte innerhalb des integrierten Systems

   In integrierten Systemen sind zum einen die Links der originalen Fachmodelle abzubilden (vgl. in Abbildung 2.10 $LA1' \ldots LA3', LB1' \ldots LB2'$). Zum anderen sind möglicherweise neue domänenübergreifende Links, zwischen Objekten aus ursprünglich unterschiedlichen Originalmodellen, hinzuzufügen ($DL1 \ldots DL3$).

2. Referenzen auf externe Dokumente

   Einige integrierte Systeme wie Document Management Systeme (DMS) sind nicht primär zur vollständigen Integration von Fachmodellen entworfen worden. Vielmehr werden

in einem Zwischenformat Metadaten über Dokumente verwaltet, mit dem Ziel diese zu archivieren und später wieder aufzufinden. Eine wesentliche Funktion solcher und verwandter integrierter Systeme ist die Referenzierung der Originaldokumente (Schmale et al., 2008).

Linked Data

Linked Data[27] ist ein Begriff aus dem Bereich des Semantic Web. Dabei werden Daten im WWW in einer normierten Weise veröffentlicht und verlinkt, sodass sie automatisiert auswertbar sind und ein Informationsmehrwert entsteht (Heath & Bizer, 2011). Zu diesem Zweck kommen standardisierte Protokolle, Formate und Verfahren des Semantic Web, wie URIs, HTTP, RDF, XML und SPARQL zum Einsatz.

Mit dem Ressource Description Framework (RDF; W3C RDF, 2004) existiert ein Datenmodell mit formaler Semantik für explizite Relationsobjekte. Links werden dabei als logische Aussagen formuliert. Dazu wird ein Tripel aus Subjekt, Prädikat und Objekt verwendet. Subjekt und Prädikat sind Ressourcen, das Objekt kann eine Ressource oder ein Wert in Form eines Literals sein. Die Bezeichner der Ressourcen sind global eindeutige URIs. Mit Abfragesprachen wie SPARQL (W3C SPARQL, 2008) kann der Informationsraum erschlossen werden.

Linked Data ist mit dem Ziel entworfen worden, die im *WWW* verfügbaren Daten zu verlinken und zu erschließen. Der Einsatz als domänenübergreifender Informationsraum für das *Bauwesen* erfordert daher Anpassungen. Im Wesentlichen müssen die Daten der Baufachmodelle per Mapping in RDF oder eine Ontologie (z. B. per Web Ontology Language, OWL; W3C OWL, 2004) überführt werden. Dieser Ansatz wurde bspw. für IFC von Beetz (2009) sowie mit heterogenen Fachmodellen bspw. von Baumgärtel et al. (2012), Curry et al. (2012) und Törmä et al. (2012) verfolgt. Auch wenn die transformierten Fachmodelle im Semantic Web-Format nur lose gekoppelt vorliegen, handelt es sich bei Linked Data-Ansätzen prinzipiell um integrierte Systeme. Link- und Abfragelogik sind auf Daten in einem normierten Format angewiesen.

2.4. Resümee

Bauprozesse sind auf kommunizierte Informationen zwischen den Akteuren angewiesen. Um solche Informationen automatisiert verarbeiten zu können, müssen diese als Daten abgebildet werden. Mit den Mitteln der Datenmodellierung erfolgt dabei eine Strukturierung. Im Ergebnis entstehen Datenaustauschformate, welche – wie eine natürliche Sprache – von allen Kommunikationsteilnehmern verstanden werden müssen. Die Bedeutung einer Nachricht ergibt sich durch die Interpretation der übermittelten Daten. Zum Informationsaustausch werden in der Baupraxis daher standardisierte oder anderweitig etablierte Datenformate, deren Semantik global definiert ist, verwendet. Anderenfalls kann die Semantik der Daten auch direkt zwischen den Kommunikationspartnern vereinbart werden. Für die flexible Unterstützung projektbezogener Kommunikation – wie sie im Bauwesen besonders ausgeprägt ist – ist es wichtig, die Wahlfreiheit der Datenformate nicht einzuschränken. Dementsprechend muss das Multimodellkonzept möglichst viele existierende Datenformate unterstützen (vgl. Kapitel 3).

Die in *einem* Datenmodell abbildbaren Informationen sind prinzipiell beschränkt. Üblicherweise beziehen sie sich auf eine konkrete Domäne wie Terminplanung oder Ausschreibung. Instanzen

[27] `http://www.w3.org/standards/semanticweb/data`

dieser Datenmodelle, s. g. Baufachmodelle, können dann mit zugehörigen Fachanwendungen verarbeitet werden. Somit ergibt sich für jedes Datenmodell ein abgeschlossener Informationsraum – das Informationspotential seiner Datenstruktur und der darauf operierenden Fachanwendungen. Viele Informationsprozesse des Bauwesens sind ihrer Natur nach aber interdisziplinär. Bezogen auf die verwendeten Datenmodelle sind sie *domänenübergreifend*, d. h. die auszutauschenden Informationen lassen sich nicht in einem einzigen Datenmodell abbilden. Vielmehr bestehen sie aus Daten, welche auf verschiedene, fachlich spezialisierte Datenformate aufgeteilt sind. Dabei entstehen getrennte, heterogene Informationsräume. Ziel des Multimodellkonzepts ist es daher, die getrennten heterogenen Informationsräume zusammenzuführen, indem die modellübergreifenden Beziehungen zwischen den Daten abbildbar und auswertbar gemacht werden.

Bisherige Ansätze können nicht beide Herausforderungen – heterogene Datenformate *und* auswertbare modellübergreifende Beziehungen – gleichzeitig lösen. Dies liegt im Wesentlichen daran, dass existierende Filter- und Abfragesprachen mit ihren entsprechenden Ablaufumgebungen zur Auswertung verwendet werden. Entsprechend ihrer Konzeption erfordern diese aber das Vorliegen aller Daten in einem korrespondierenden, homogenen Datenformat. Aus diesem Grund wird die Multimodell-Abfragesprache MMQL mit entsprechendem Interpreter vorgestellt (vgl. Kapitel 4 und 5). Statt des homogen Daten*formates* benötigt die MMQL lediglich einen homogen, generischen Daten*zugriff*.

Kapitel 3.

Das Multimodellkonzept

Zusammenkommen ist ein Beginn,
Zusammenbleiben ein Fortschritt,
Zusammenarbeiten ein Erfolg.

(Henry Ford)

Multimodelle erlauben die lose Kopplung von Baufachmodellen zur Bildung flexibler domänenübergreifender Bauinformationsräume. Durch eine externe Verlinkung und die Verwendung nahezu beliebiger Datenformate können die Fachmodelle auch in herkömmlichen Fachanwendungen weitergenutzt werden. Der Einsatz von Multimodell-Containern erlaubt den Datenaustausch solch eines Verbundes. Dieses Kapitel beschreibt die prinzipielle Arbeitsweise mit Multimodellen, deren innere Struktur sowie die daraus ableitbaren grundlegenden Operationen.

Abschnitt 3.1 zeigt die Schwierigkeiten vorhandener Integrationsansätze in domänenübergreifenden Bauinformationsprozessen und stellt den Multimodell-Ansatz als Lösungsmöglichkeit vor. In Abschnitt 3.2 wird die multimodellbasierte Arbeitsweise behandelt. Dabei wird der Anwendungsbereich des Konzepts sowie die Einbeziehung bestehender Fachanwendungen erläutert. Der strukturelle Aufbau von Multimodellen wird in Abschnitt 3.3 gezeigt und formalisiert. Abschnitt 3.4 beschreibt, welche Baufachmodelle am Multimodell-Verbund teilnehmen können. Die Spezialisierung des allgemeinen Multimodell-Ansatzes für spezifische, abgeschlossene Bauinformationsräume wird in Abschnitt 3.5 dargestellt. Abschließend werden in Abschnitt 3.6 strukturelle Operationen auf Multimodellen analysiert.

3.1. Das Multimodell-Paradigma

3.1.1. Übergang von bauwerksorientierter zu prozessorientierter Arbeitsweise

Die Informationsprozesse des Bauwesens sind nicht mit denen anderer Industriezweige vergleichbar. Zwar gibt es in anderen Branchen, bspw. im Schiffsbau, auch Produkte mit Unikatcharakter – aber im Bauwesen sind darüber hinaus auch die Baustelle, deren Infrastruktur sowie die Zusammensetzung aller Projektbeteiligten wie Bauherr, Nutzer, Planer, Baufirmen, Kreditgeber oder Baubehörden einzigartig (Scherer, 1998; Katranuschkov et al., 2001). Die Struktur der Bauindustrie, mit überwiegend kleinen Unternehmen, spiegelt diesen Umstand wieder (vgl. Bundesinstitut für Bau-, Stadt- und Raumforschung BBSR, 2012b). Da eine übergreifende Zusammenarbeit in der Regel nur für *ein* Bauprojekt besteht, können sich industrieweit keine langfristigen unternehmensübergreifenden Produktionsprozesse, wie bspw. in der Automobilindustrie, etablieren. Dadurch lassen sich Bauinformationsprozesse meist auch nicht mit Standardsoftware der Wirtschaftsinformatik abbilden. Bauinformationsprozesse sind entsprechend ihren zugrundeliegenden Bauplanungs- und -managementprozessen projektindividuell, nicht standardisiert, unregelmäßig, ad hoc und aufgabenbezogen.

Ein Forschungsziel der Bauinformatik ist daher die Schaffung einer *gemeinsam nutzbaren, integrierten Datenbasis*. Besonders für Planungsprozesse wurden mit Produktdatenmodellen dahingehend Fortschritte erzielt. Die relevanten Aspekte in Bezug auf domänenübergreifende Aufgabenstellungen wurden in Abschnitt 2.3 vorgestellt. Zusammenfassend wird deutlich, dass sich die aktuellen Integrationsbemühungen in diesem Bereich vorwiegend *BIM-zentriert* (im Sinne von Bauwerksmodellen) darstellen:

1. Bauwerksmodelle, insbesondere IFC, sollen möglichst viele Daten der Planungsprozesse abbilden können. Die Schemas sind dementsprechend umfangreich und komplex.
2. Zur Abbildung von Daten weiterer Domänen werden
   a) Schemaerweiterungen existierender Bauwerksmodelle angestrebt. Dies führt zu einer weiteren Vergrößerung von Umfang und Komplexität der Schemas.
   b) Ansätze mit loser Kopplung (bspw. nD-Modelling) vorgeschlagen, bei denen Bauwerksmodelle eine führende Rolle einnehmen.
   c) integrierte Systeme entwickelt, welche zwar domänenübergreifende Informationen erzeugen, solche Daten aber nicht handhabbar austauschen können.
3. Zur verbesserten Handhabung der komplexen Bauwerksmodelle werden Filtermethoden wie Subschemas oder Model Views sowie wissensbasierte Zugriffsschichten entwickelt.
4. Die zentrale Datenverwaltung mit Client-Server-Architekturen wird durch unterschiedliche Datenbanksysteme, Methoden für Mapping, Versionierung, Zugriffskontrolle und Clustering sowie durch zunehmende Verlagerung von Fachlogik und GUI vom Client auf den Server gestützt.

Eine Ursache der Dominanz BIM-zentrierter Datenintegration ist die zugrundeliegende *bauwerksorientierte Arbeitsweise* der Architekten und Ingenieure in Planungsprozessen. Diese ist produktorientiert und fokussiert das Projekt als Ganzes (Scherer & Schapke, 2010). Im Mittelpunkt steht die Erstellung eines detaillierten, fehlerarmen Vorgabemodells des Bauwerks unter Einhaltung aller Randbedingungen. Diese Arbeitsweise wird auch auf andere Phasen des

Bauwerkslebenszyklus wie Ausführung, Betrieb, Sanierung oder Abriss übertragen, da zu deren Bewältigung im Grunde die selben Akteure und Werkzeuge zum Einsatz kommen.

Die Idee einer projektweiten, zentralen, gemeinsam genutzten Datenbasis in Form eines singulären logischen[1] Bauwerksmodells konnte sich in der industriellen Praxis bisher jedoch nicht durchsetzen. Dies hat mehrere Gründe:

1. Technische Ursachen

    a) Aktuelle Bausoftware ist überwiegend gar nicht oder nur beschränkt, auf eine herstellergebundene, proprietäre Weise, zur parallelen kollaborativen Arbeit geeignet.

    b) Prototypisch entwickelte Integrationslösungen der Forschung, wie bspw. integrierte Systeme, finden kaum Einzug in Marktsoftware. Ihre technischen Konzepte basieren meist auf neuartigen Technologien und orientieren sich nicht an bestehenden Produkten.

    c) Keine Anwendersoftware beherrscht *alle* Domänenaspekte aktueller Bauwerksmodelle.

    d) Zwar sind Speziallösungen wie integrierte Systeme in der Lage *intern* domänenübergreifende Verknüpfungen und Datenverarbeitung (bspw. modellbasierte Mengenermittlung) durchzuführen – ein *externer*, neutraler Datenaustausch der Ergebnisse ist jedoch unmöglich, wenn nicht alle Daten innerhalb *eines* Fachmodelltyps abbildbar sind.

2. Organisatorische Ursachen

    a) Allein die Existenz von Produktdatenmodellen im Bauwesen führt nicht automatisch zur Verbesserung der Bauinformationsprozesse. Vielmehr müssen auch die Organisationen und Arbeitsprozesse angepasst werden. Entsprechend dieser Managementaufgabe wird der Begriff *BIM* daher nicht mehr als Bauwerksmodell, sondern als *Methode* (Building Information Modelling) aufgefasst (Eastman et al., 2011).

    b) Der Einsatz eines zentralen Datenmodells erfordert das Management und die Kontrolle einer einzelnen Organisation über dieses Datensystem. Die Übernahme einer solchen Strategie in Bauprojekte ist unwahrscheinlich, da die verschiedenen Interessenträger Teile der Projektinformation selbst kontrollieren möchten (Zamanian & Pittman, 1999).

    c) Besonders in der Planung ist die aktuelle Arbeitsweise durch Aufgabenteilung zwischen den Fachplanern geprägt. Ergebnis einer Arbeitsaufgabe ist in der Regel ein Dokument oder Fachmodell, bspw. Schal- und Bewehrungspläne, Massenschätzung, Energiebedarfsberechnung oder ein Leistungsverzeichnis. Für andere Lebenszyklusphasen existieren analoge Übergabeobjekte wie Kennzahlen, Pläne oder Berichte. Dadurch ist der Leistungsumfang einer Aufgabe abgegrenzt und das Arbeitsergebnis messbar. Zum Nachweis der Aufgabenerfüllung verbleiben die Dokumente der Zwischen- und Endergebnisse in lokaler Kopie beim Leistungserbringer. Diese Arbeitsweise lässt sich nicht mit einem singulären logischen Bauwerksmodell beibehalten, da dort individuelle Arbeitsergebnisse nicht mehr klar abgegrenzt werden können (Juli & Scherer, 2002).

[1] im Sinne von *nicht physisch* singulär; also auch in Form von Partialmodellen, welche nach außen hin als Gesamtmodell erscheinen

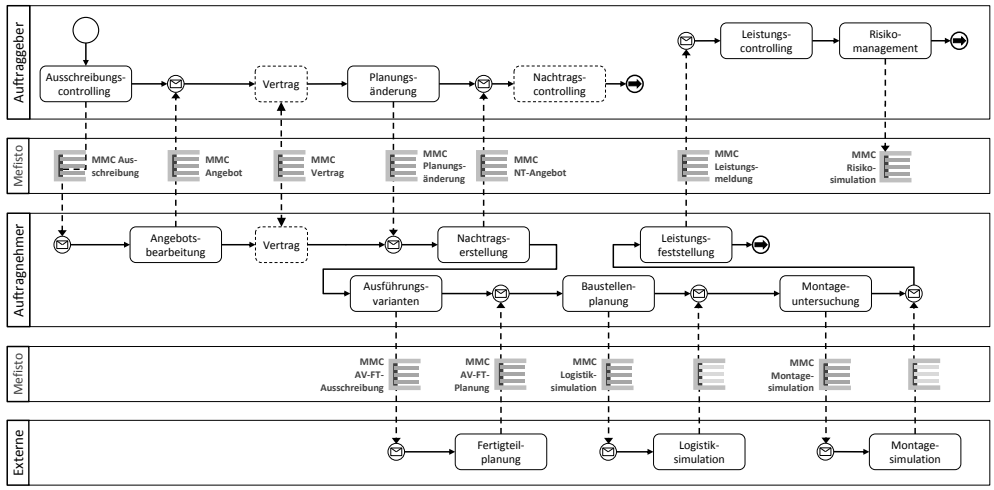


Abbildung 3.1.: Informationsaustausch durch Multimodell-Container (MMC) bei prozessorientierter Arbeitsweise. Beispiel eines Prozessausschnitts in BPMN-Notation aus dem Mefisto-Projekt (aus Schapke & Hilbert, 2012b).

d) Liebich et al. (2011) haben die Konsequenzen von Building Information Modelling für Verträge und Honorare von Planern untersucht. BIM ist demzufolge eine Grundsatzentscheidung für eine Planungsmethode und keine Zusatzleistung aktueller Planungsleistungen. Dabei müssen die Aufwandsverteilungen zwischen den Fachdisziplinen prinzipiell neu bewertet und der entstehende Mehrwert vergütet werden. Die Effizienz von BIM ist vom Umfang der vertraglichen Verankerung gemeinschaftlichen Handelns (gemeinsame Vorteile und Risiken) abhängig. BIM ist mit der aktuellen HOAI nicht abbildbar. Daher sind neue Vertrags- und Vergütungsregeln notwendig.

e) Die Einführung neuer Technologien wird durch die Besonderheiten des Bauwesens erschwert. Dazu zählen der Unikatcharakter von Produkt, Produktionsort und Zusammensetzung der Projektbeteiligten, die Struktur der Bauindustrie, der hohe Anteil der Entwurfskosten am Baupreis, der hohe Abstraktionsgrad der Modelle, die generellen Probleme der Kommunikation über Bauwerksentwurf und Koordination der Zusammenarbeit sowie die Umsetzung von Softwaresystemen für verteiltes kooperatives Arbeiten im Kontext temporärer und virtueller Organisationen (Hauschild, 2004; Weise, 2006a).

Mit dem Übergang von bauwerksorientierter zu prozessorientierter Arbeitsweise (Scherer & Schapke, 2011) können Arbeitsaufgaben und Bauinformationsprozesse besser aufeinander abgebildet werden. Indem man die Bauplanungs- und -managementprozesse als formale Prozesse betrachtet, können diese in Teilprozesse (auch Aktivitäten, hier: Arbeitsaufgaben) zerlegt werden, welche untereinander verkettet und mathematisch analysiert werden können. Input, Output oder Ressourcen solcher Arbeitsaufgaben sind Informationen in Form von Dokumenten oder Fachmodellen (Froese, 2003; Huhnt & Richter, 2010; Huang et al., 2012). Die Output-Information eines Teilprozesses kann notwendiger Input des Nachfolgeprozesses sein.

Die prozessorientierte Arbeitsweise bildet durch ihren organisatorischen Fokus auf Aufgabenbewältigung die *Vorgänge* der industriellen Praxis besser ab, als die ganzheitlich fokussierte

bauwerksorientierte Arbeitsweise. Auch die Unterstützung der zugehörigen Bauinformationsprozesse gestaltet sich einfacher, da anstelle einer *allumfassenden* Integrationslösung lediglich *aufgabenspezifisch* Informationsbedarfe sichergestellt werden müssen. Je mehr von diesen Bauinformationsprozessen ihre In- und Outputdaten übergabegerecht abbilden können, umso mehr durchgehende Informationsübergänge entstehen, welche schlussendlich zum Projektziel führen. Abbildung 3.1 auf der vorangegangenen Seite zeigt diesen durchgehenden prozessorientierten Informationsaustausch exemplarisch für die Anfangsphasen eines Bauprojektes.

Das Übergabeformat wird dabei maßgeblich von der Aufgabenstellung und der vorhandenen Software bestimmt. Zwar existieren für Standard-Bauinformationsprozesse etablierte Datenformate[2] – es sind jedoch keine flexibel anwendbaren Lösungen für den Datenaustausch in domänenübergreifenden Aufgabenstellungen verfügbar. Um der wachsenden Relevanz domänenübergreifender Bauinformationsprozesse in allen Lebenszyklusphasen gerecht zu werden, wird mit dieser Arbeit das Multimodell-Konzept vorgestellt.

3.1.2. Verband gleichgestellter Fachmodelle

Multimodelle bündeln Fachmodelle unterschiedlicher Domänen und erlauben die Verbindung von deren Elementen in externen Linkmodellen. Ziel ist es, die ursprünglich getrennten Informationsräume zusammenzuführen und zwischen den Projektbeteiligten austauschbar zu machen.

Beispielsweise können die Daten für eine Energiebedarfsberechnung in verschiedenen Fachmodellen hinterlegt sein (vgl. Tabelle 3.1): ein CityGML-Modell enthält die umgebende Bebauung, ein Energy Plus-Modell (Crawley et al., 2001) die Wetterdaten[3] und ein IFC-Modell die Bauwerksdaten. Durch Zusammenführung und Verlinkung können diese Daten neutral ausgetauscht und automatisiert ausgewertet werden.

Tabelle 3.1.: Beispiel kombinierter Fachmodelle zur Energiebedarfsberechnung

Domäne \ Datenformat	CityGML	Energy Plus	IFC
Hochbaugeometrie			✓
Geodaten	✓		
HVAC		✓	
Basis \ Datenformat			
XML (XSD)	✓		
SPF (EXPRESS)			✓
ASCII		✓	

[2] vgl. Abschnitt 2.2.2 auf Seite 20

[3] Wetterdaten im EnergyPlus-Format sind bspw. bei `http://apps1.eere.energy.gov/buildings/energyplus/weatherdata_about.cfm` verfügbar.

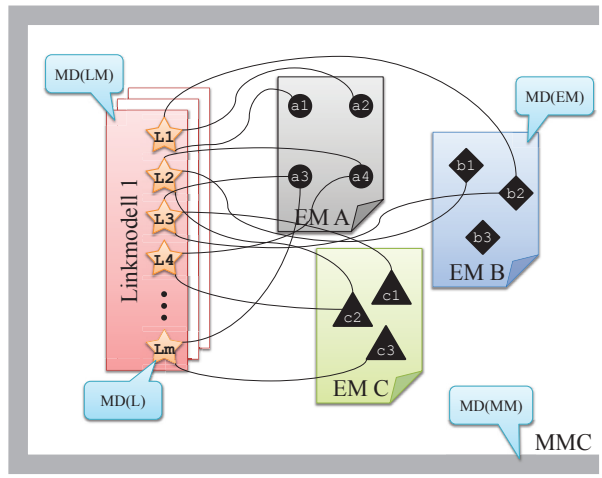


Abbildung 3.2.: Konzept der Multimodelle. Gleichgestellte, heterogene Elementarmodelle ($EM_A \dots EM_C$) werden gebündelt. Links zwischen den Datenelementen werden als Fachmodell-externe Relationsobjekte ($L_1 \dots L_m$) abgebildet und in Linkmodellen zusammengefasst. Metadaten können für das Multimodell ($MD(MM)$) sowie für die Elementarmodelle ($MD(EM)$), Linkmodelle ($MD(LM)$) und Links ($MD(L)$) vergeben werden. Alle Daten werden in Form eines Multimodell-Containers (MMC) ausgetauscht.

Das Multimodellkonzept stützt sich auf folgende Ausgangssituation und Annahmen:

1. Für jede relevante Domäne existieren unabhängige, etablierte und möglicherweise standardisierte Datenformate. Diese werden mit Fachanwendungen erstellt, ausgetauscht und verarbeitet. Dadurch ist in Bauprojekten eine Datenbasis aus überwiegend domänenspezifischen Fachmodellen vorhanden. Die Daten domänenübergreifender Bauinformationsprozesse lassen sich durch Aufteilung auf unterschiedliche domänenspezifische Fachmodelle repräsentieren. Bisher sind datenformatübergreifende Beziehungen jedoch nicht in austauschbarer Form abbildbar.

2. Bausoftware ist hochentwickelt, teuer und auf bestimmte fachliche Aufgabenstellungen spezialisiert. Marktübliche Fachanwendungen werden auch weiterhin zur Erstellung und Bearbeitung von Fachmodellen eingesetzt. Die getätigten Investitionen der Gemeinschaft in Datenstandards sowie der Unternehmen in Softwareprodukte sind schützenswert. Für domänenübergreifende Aufgabenstellungen ist die Kombination vorhandener Applikationen der Neuentwicklung eines allumfassenden Modellierungssystems vorzuziehen (Jongeling et al., 2005, S. 38).

3. In einzelnen Bauinformationsprozessen muss eine aufgabenspezifische Kombination von wenigen (zwei bis ca. zehn) Fachmodellen unterstützt werden. In keinem Fall soll die Datenbasis des *gesamten* Bauprojekts durch *ein* singuläres Multimodell repräsentiert werden.

Unter diesen Randbedingungen präsentieren sich Multimodelle als Behälter heterogener Fachmodelle mit explizit loser Kopplung zwischen deren Datenelementen. Die Kernkonzepte von Multimodellen sind (vgl. auch Abbildung 3.2):

1. Gleichgestellte Fachmodelle mit beliebigen Datenformaten

   Multimodelle bündeln Fachmodelle mit beliebigen Datenformaten. Dazu zählen auch Ressourcen wie Relationale Datenbanken oder Webservices. Ziel ist die Anwendung von etablierten Baufachmodelltypen wie GAEB oder IFC sowie prinzipiell jedem anderen Datenmodell – auch von außerhalb des Bauwesens – in domänenübergreifenden Bauinformationsprozessen. Die Fachmodelle eines Multimodells sind gleichgestellt. Es gibt kein integrierendes, führendes oder notwendiges Fachmodell. Durch den Austausch unveränderter Fachmodelle ist ein Zugriff auf die vollständigen Originaldaten möglich. Da keine Datenkonvertierung notwendig ist, werden mögliche Informationsverluste aus Transformationsprozessen vermieden.

   Ein Fachmodell, welches in Multimodellen zur Anwendung kommt, wird *Elementarmodell* genannt.

2. ID-basierte, externe Linkmodelle

   Elementarmodellübergreifende Beziehungen zwischen Datenelementen werden durch Relationsobjekte (*Links*) in expliziten Datenmodellen, den so genannten *Linkmodellen*, außerhalb der Fachmodelle abgebildet. Diese Links können mehrwertig sein. Da Links aufgabenspezifisch sind, steht jedes Linkmodell für eine andere Bedeutung der enthaltenen Links.

   Die Datenelemente der Elementarmodelle werden über Identifikatoren (IDs) referenziert. Auf diese Weise können die Elementarmodelle unverändert bleiben und es entsteht eine lose Kopplung. Wird ein Elementarmodell alleinstehend betrachtet, kann nicht entschieden werden, ob es Teil eines Multimodells ist. Somit sind die ursprünglichen Fachmodelle weiterhin direkt in herkömmlichen Baufachanwendungen nutzbar.

3. Metadaten

   Elementarmodelle, Linkmodelle, Links sowie das Multimodell als Behälter an sich können um beliebige Metadaten ergänzt werden. Die Metadaten dienen der Semantischen Beschreibung der Inhalte und tragen dazu bei, die Intention des Multimodell-Erstellers bei einem Datenaustausch zu erhalten.

4. Container

   Ein Multimodell kann als singuläre Informationsressource neutral übertragen werden. Dazu werden Elementar- und Linkmodelle sowie die Metadaten in einen *Multimodell-Container* (MMC) serialisiert. Zur Reduktion des Speicherbedarfs können Elementarmodelle auch per URI referenziert werden. Multimodell-Container sind die Grundlage für austauschbare domänenübergreifende Informationsräume.

3.1.3. Multimodell-Begriff in Bauinformatik und IKT

Der Begriff „Multimodell" wird bereits im Bereich der Bauinformatik und IKT benutzt. Im Bereich der Partialmodelle bezeichnen Kiviniemi et al. (2005b) ein Konzept zur Verlinkung homogener Bauwerksmodelle als „multi-model environment". In der technischen Mechanik wird der Begriff zur Identifikation statischer Systeme über Messwerte verwendet Smith & Saitta (2008); Smith (2009); Goulet et al. (2010). Durflinger et al. (1998) präsentieren eine Multimodell-Datenbank, deren Prinzip auf vereinheitlichtem Speicherformat mit multiplem Erscheinungsbild

nach außen basiert. Die Datenbanksoftware ArangoDB[4] beschreibt sich selbst als „Multi model database", wobei Kombinationen aus Schlüssel-Wert-Paaren, Dokumenten und Graphen verwendet werden. Im Bereich MDSD bilden so genannte Domänenmodelle Software-Artefakte. Denton et al. (2008) schlagen eine Plattform zur modellgetriebenen Softwareentwicklung mit unterschiedlichen, kombinierten Domänenmodellen vor. Hessellund (2009) beschreibt mit „Multimodeling" eine Methode zur Koordination verschiedener Domänenspezifischer Sprachen.

3.2. Multimodellbasierte Arbeitsweise

3.2.1. Basis-Linktypen und Anwendungsbereiche

Die möglichen Anwendungsbereiche des Multimodell-Konzepts ergeben sich durch die Analyse der zu verlinkenden Fachmodelle. Im Sinne der prozessorientierten Arbeitsweise kann aus einem solchen Anwendungsbereich geschlussfolgert werden, ob eine konkrete Aufgabenstellung mithilfe des Multimodell-Konzepts *prinzipiell* lösbar ist.

In Tabelle 2.2 auf Seite 27 wurden bereits Bauinformationsräume nach Datenformat und Domäne kategorisiert. Unterscheidet man die verlinkten Baufachmodelle innerhalb eines Multimodells

Tabelle 3.2.: Basis-Linktypen und mögliche Anwendungsbereiche von Multimodellen

Fachmodellverlinkung \ Inhaltsähnlichkeit der Fachmodelle			analog	divers
Basis-Linktyp	Datenformate	Domänen		
modell-übergreifend	homogen	homogen	Kennzeichnung von Redundanzen, Unterschieden, Detaillierung, Versionierung	Aufteilung von Informationen, Teilmodelle gleicher Systeme
	homogen	heterogen	*nicht möglich*	unterschiedliche Systeme in eigenständigen Fachmodellen
datenformat-übergreifend	heterogen	homogen	Matching, Informationsabgleich, Synchronisation	Teilmodelle in unterschiedlichen Datenformaten
domänen-übergreifend	heterogen	heterogen	*nicht möglich*	domänenübergreifende / interdisziplinäre Aufgabenstellungen
Dokumenten-links	heterogen	unbestimmt	*nicht möglich*	Verbindung von semantischen Fachdaten mit Mediendaten

[4] `http://www.arangodb.org/`

nach denselben Kriterien, so ergeben sich die in Tabelle 3.2 auf der vorherigen Seite dargestellten Basis-Linktypen. Diese beschreiben zunächst die Gleichartigkeit der verlinkten Fachmodelle bezogen auf ihr Datenformat und ihre Domäne.

Des Weiteren ist nach der inhaltlichen Ähnlichkeit der verlinkten Fachmodelle zu differenzieren. Die möglichen Anwendungsgebiete unterscheiden sich stark – je nachdem, ob die Modelle einen analogen oder diversen fachlichen Inhalt besitzen. Analoger Inhalt bedeutet, dass die verlinkten Fachmodelle jeweils dieselben Realweltobjekte abbilden, wobei beabsichtigte (z. B. Variantenuntersuchung oder Planungsfortschritt) und unbeabsichtigte (z. B. bei Modellbearbeitung mit verschiedener Software) Abweichungen zugelassen sind. Diverser Inhalt bedeutet, dass mindestens *ein* verlinktes Fachmodell ein zweites, anderes Realweltobjekt abbildet. Eine konkrete Entscheidung über Inhaltsähnlichkeit kann nicht immer eindeutig getroffen werden. In jedem Fall ist der Kontext der Aufgabenstellung zu berücksichtigen.

Eine Kombination von Baufachmodellen mit heterogenen Domänen aber analogem Inhalt ist nicht möglich. Es ist davon auszugehen, dass Informationen aus verschiedenen Fachgebieten auch unterschiedliche Realweltobjekte und -konzepte abbilden.

Modellübergreifende Verlinkung ist die Verlinkung von mindestens zwei Fachmodellen mit homogenem Datenformat, bspw. von zwei oder mehr IFC-Produktdatenmodellen im SPF-Format. Besitzen diese Modelle einen analogen Inhalt, so können mit dem Multimodell-Konzept Redundanzen oder Unterschiede der Datenelemente per Link kenntlich gemacht werden, z. B. bei unterschiedlichem Detaillierungsgrad oder neuem Versionsstand bei fortschreitender Modellbearbeitung.

Modelle mit diversem Inhalt aber homogener Domäne beschreiben eine Aufteilung gleichartiger Informationen in Teilmodelle, bspw. die Geometriedaten diskreter Bauabschnitte eines Flughafens.

Unterschiedlicher Inhalt und heterogene Domänen repräsentieren Informationen für unterschiedliche Systeme oder Subdomänen. Diese Konstellation ist besonders bei Produktdatenmodellen denkbar, da diese eine Vielzahl solcher Bauwerksaspekte aufnehmen können. Ein mögliches Szenario ist die Kopplung des Tragsystems mit der Bauteilrepräsentation per Multimodell-Link.

Datenformatübergreifende Verlinkung ist die Verlinkung von Fachmodellen mit heterogenem Datenformat und homogener Domäne. So ist es denkbar, ein Leistungsverzeichnis im Format GAEB-DA-XML mit einem Leistungsverzeichnis, welches in einer Tabellenkalkulation geführt wird, zu verlinken. Besitzen beide Modelle den analogen Inhalt so können die identischen Datenelemente – in diesem Fall LV-Positionen – mit einer Verlinkung gekennzeichnet werden. Dieses Matching ermöglicht bspw. einen Informationsabgleich bei unabhängigen Modelländerungen oder die Vorbereitung von Konvertierungen bzw. Mappings. Darüber hinaus gelten auch die Anwendungsgebiete entsprechend der modellübergreifenden Verlinkung von Modellen mit homogener Domäne und analogem Inhalt.

Besitzen die Fachmodelle diversen Inhalt, können Teilmodelle in unterschiedlichen Datenformaten gebildet werden. Bezogen auf das Beispiel der Leistungsverzeichnisse könnte eine Vergabeeinheit in GAEB-DA-XML beschrieben sein und eine weitere im Tabellenkalkulationsformat.

Domänenübergreifende Verlinkung ist die Verlinkung von Fachmodellen mit heterogenem Datenformat, heterogener Domäne und diversem Inhalt. Diese Konstellation ermöglicht

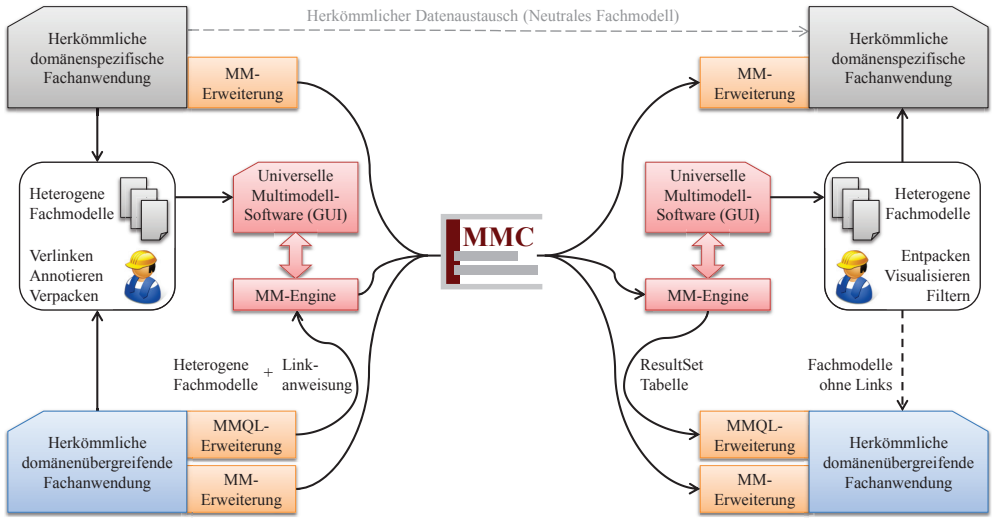


Abbildung 3.3.: Möglichkeiten der Integration vorhandener Fachanwendungen in multimodellbasierte Bauinformationsprozesse

die Bearbeitung der Klasse von domänenübergreifenden, möglicherweise interdisziplinären Aufgabenstellungen. Für das Multimodell-Konzept gilt dieser Basis-Linktyp als allgemeine Grundlage, da hier die größtmögliche Variation von Format und Inhalt der Fachmodelle gegeben ist. Methodisch stellen die vorgenannten Basis-Linktypen inkludierte Sonderfälle dar. Aus diesem Grund gelten für die domänenübergreifende Verlinkung auch die Anwendungsgebiete der modellübergreifenden Verlinkung von Modellen mit heterogener Domäne und diversem Inhalt.

Dokumentenlinks stellen die Sonderform der Verlinkung von Fachmodellen mit Dokumenten dar. Dokumente sind in diesem Zusammenhang unstrukturierte oder unbekannt strukturierte Daten wie Texte, Bilder oder Videos. Aus Sicht des Multimodells ist es nicht möglich, Datenelemente der Dokumente semantisch zu identifizieren und zu verlinken. Die Interpretation der Dokumentendaten geschieht ausschließlich durch den Menschen. Daher kann für Dokumente auch keine – im Multimodell korrespondierende – Domäne angegeben werden. Dennoch ist es in der Baupraxis wichtig, semantischen Fachmodellelementen mediale Inhalte zuzuordnen. Abschnitt 3.4.4 auf Seite 71 erläutert, wie Dokumente in die multimodellbasierte Arbeitsweise eingebunden werden können.

3.2.2. Softwareintegration und Datenaustausch

Die Wesensmerkmale multimodellbasierter Bauinformationsprozesse sind erschließbare domänenübergreifende Informationsräume sowie Datenaustausch mit Multimodell-Containern. Dazu sind eine oder mehrere Softwareanwendungen notwendig, welche zum einen die fachliche, domänenübergreifende Erstellung, Bearbeitung und Auswertung der Daten ermöglichen und zum anderen Multimodell-Container lesen oder schreiben können.

Fachanwendungen können mit diesen Eigenschaften grundlegend neu erstellt werden. Dies ist insbesondere für domänenübergreifende Aufgabenstellungen erforderlich, für welche bislang

keine Implementierung existiert. Prinzipiell soll jedoch vorhandene Bausoftware weitergenutzt werden. Abbildung 3.3 auf der vorherigen Seite zeigt, auf welchen Wegen herkömmliche Fachanwendungen in multimodellbasierte Bauinformationsprozesse integriert werden können:

1. Nutzung einer universellen Multimodell-Software

   Eine universelle Multimodell-Software kann Multimodelle mit beliebigen Elementarmodelltypen verarbeiten. Dazu besitzt sie einen modularen Aufbau und ist um Komponenten für neue Elementarmodelltypen erweiterbar. Typische Funktionen sind:

   - Einlesen und Visualisieren der Elementarmodelle
   - Erstellung, Analyse und Manipulation von Links
   - Erschließung des domänenübergreifenden Informationsraumes durch Multimodell-Filter und -Viewer
   - Annotation mit Metadaten
   - Ver- und Entpacken von Multimodell-Containern

   Solche Anwendungen können eine generische Kernkomponente – die Multimodell-Engine (MM-Engine) – nutzen, um die Grafische Oberfläche (GUI) von der Multimodell-Logik zu trennen. Auf diese Weise kann die MM-Engine in dritte Anwendungen per Programmierschnittstelle (API) eingebunden oder als Service, bspw. per Representational State Transfer (REST), zur Verfügung gestellt werden. Die prototypische Implementierung der universellen Mutlimodell-Software *M2A2* wird in Abschnitt 6.1 auf Seite 160 vorgestellt.

   Der Einsatz einer universellen Multimodell-Software ist eine Ergänzung der bisherigen Arbeitsweise. Es müssen keine Änderungen an herkömmlichen Fachapplikationen stattfinden. Dadurch können diese aber auch nur Fachmodelle entsprechend ihrer Domänen produzieren. In einem zusätzlichen Arbeitsschritt werden die heterogenen Fachmodelle in der universellen Multimodell-Applikation verlinkt, annotiert und zu Multimodell-Containern verpackt. Auf der Empfängerseite können mit derselben Software Container gelesen sowie Multimodelle visualisiert und domänenübergreifend gefiltert werden. Hierdurch wird ein informationeller Mehrwert gegenüber alleinstehenden Fachmodellen geschaffen. Die originalen Elementarmodelle können entnommen und den herkömmlichen Fachanwendungen zugeführt werden. Dies schließt jedoch die Nutzung der Multimodell-Links aus, da die Anwendungen diese nicht interpretieren können.

   Die Arbeit mit einer universellen Multimodell-Software erfordert menschlichen Eingriff. Für das Verlinken und das domänenübergreifende Filtern ist Wissen über die baufachliche Semantik der Daten sowie über die verwendeten Datenstrukturen notwendig. Aufgrund der Allgemeingültigkeit der Applikation kann dieses Wissen dort nicht fest verankert sein, sondern muss über Nutzeranweisungen zur Anwendung kommen. Die Abläufe können daher nur semiautomatisch erfolgen.

2. Erweiterung von Fachanwendungen um Multimodellfähigkeiten

   Soll eine herkömmliche Fachanwendung direkt Multimodell-Container austauschen können, damit bspw. kein Einsatz einer weiteren Software notwendig ist, muss diese um Multimodellfähigkeiten erweitert werden. Die im Folgenden beschriebenen Erweiterungsmöglichkeiten gelten analog auch für die Erstellung neuer Fachanwendungen.

   a) Multimodell-Erweiterung herkömmlicher domänenspezifischer Fachanwendungen

Domänenspezifische Fachanwendungen sind inhaltlich nicht multimodellfähig, wenn sie nur *einen* Fachmodelltyp unterstützen. Dennoch kann ein singuläres Elementarmodell in einem Multimodell-Container verpackt ausgetauscht werden, da dieses bereits ein gültiges Multimodell darstellt. Somit wird bspw. die Teilnahme an Bauinformationsprozessen ermöglicht, welche einen alleinigen Datenaustausch auf Basis von Multimodell-Containern fordern.

b) Erweiterung herkömmlicher domänenübergreifender Fachanwendungen

   i. Multimodell-Erweiterung

      Domänenübergreifende Fachanwendungen werden derart erweitert, dass sie ihre Daten in heterogenen Fachmodellen abbilden und mit den fachmodellübergreifenden Links als Multimodell-Container austauschen können. Diese Art der Erweiterung eignet sich besonders für spezialisierte Anwendungen mit wenigen Domänen, welche bereits über interne Logik zur Realisierung domänenübergreifender Informationsräume (bspw. integriertes Datenmodell oder Mechanismen für domänenübergreifende Links) verfügen.

   ii. MMQL-Erweiterung

      Einige domänenübergreifende Fachanwendungen verfügen nicht über die notwendigen Kapazitäten zum Parsen vielfältiger Datenformate oder zur Auswertung domänenübergreifender Links. Dazu zählen unter anderem Anwendungen

      - mit breitem, generischen Einsatzgebiet wie bspw. Simulationswerkzeuge
      - ohne vollständige fachliche Datenhaltung wie bspw. Baustatiksoftware
      - sowie vorwiegend lesend zugreifende Programme wie Visualisierungen.

      Diese Anwendungen können die Multimodell-Engine als Bibliothek oder Service einbinden, um multimodellbasierte Daten auszutauschen. Technisch ist dies mit der Anbindung an eine Relationale Datenbank vergleichbar. Die Kommunikation mit der Bibliothek erfolgt im Kern über die Multimodell-Abfragesprache MMQL[5]. Die Multimodell-Engine beinhaltet Logik, um den domänenübergreifenden Informationsraum von Multimodellen zu erschließen; das heißt sie nutzt Elementarmodellparser zum Zugriff auf Originaldaten und kann die Links zwischen Elementarmodellen auswerten. Anfrageergebnisse werden, analog zu relationalen Datenbanken, in Tabellenform zurückgegeben.

Das Multimodellkonzept gibt Fachmodellen einen hohen Stellenwert. So ist vorgesehen, dass fachliche Informationen ausschließlich in Elementarmodellen abzubilden sind. Die Verwendung von Metadaten oder Links zur Nutzdatenabbildung ist nicht zulässig. Existiert für eine Domäne kein Fachmodelltyp, so soll dafür ein neues Datenmodell entworfen werden. Ebenso sind auch Multimodell-Links für ein singuläres Elementarmodell unzulässig – dies stünde für eine fachliche Ergänzung des Datenmodells. Vorrangiges Ziel des Multimodellkonzepts ist die Bereitstellung domänenübergreifender Informationen durch Überwindung der Heterogenität der Datenstrukturen. Die jeweilige Neuentwicklung oder Änderung eines Datenformates für eine spezifische fachübergreifende Aufgabenstellung soll somit vermieden werden.

Gleichermaßen soll die Bearbeitungshoheit bei den Fachanwendungen verbleiben. Elementarmodelle sind in einem Multimodell als schreibgeschützt zu betrachten. Sind im einem Bauinformationsprozess Änderungen an einem Elementarmodell innerhalb eines Multimodellverbundes

[5] vgl. Kapitel 4 auf Seite 83

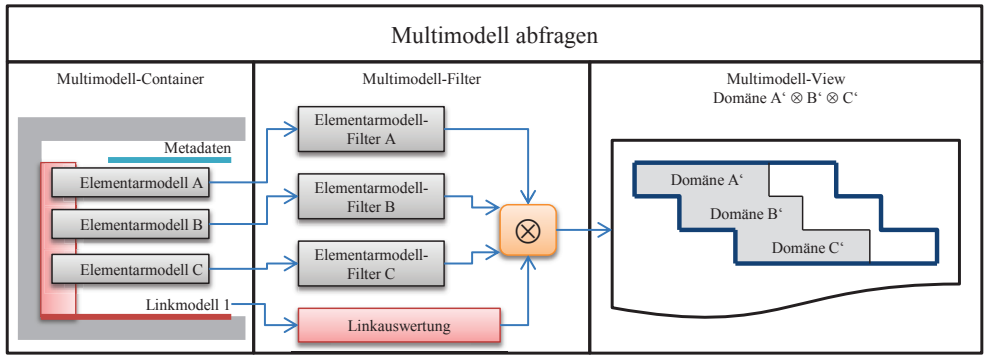


Abbildung 3.4.: Erzeugung von Multimodell-Views durch Multimodell-Filter

notwendig, bspw. durch verkettete Prozessschritte, so sind der betroffene Multimodell-Container zu entpacken, das Elementarmodell zu ändern, die Links anzupassen und ein neuer Multimodell-Container zu erstellen.

3.2.3. Multimodell-Views

Filter sind wesentliche Elemente zur Erschließung von Informationsräumen. Mit den korrespondierenden Fachanwendungen existieren bereits umfangreiche Möglichkeiten, die Informationen eines singulären Elementarmodells für Anwender zugänglich zu machen. Zwar existieren bereits Applikationen wie Navisworks[6], iTWO[7], Synchro[8] oder Vico Office Suite[9], welche unterschiedliche Datenformate lesen können – der erzielbare Informationsgewinn ist jedoch auf deren Anwendungsgebiet beschränkt. *Beliebige* domänenübergreifende Informationen können durch einen Experten bisher nur flüchtig und von Hand gewonnen werden. Dazu ist die parallele Nutzung domänenspezifischer Einzelanwendungen sowie die *manuelle*, gedankliche, implizite Zuordnung korrespondierender Realsachverhalte über die Einzelanwendungen hinweg erforderlich.

Mithilfe von Multimodellen können zusammengehörende Informationen über die persistenten Links *automatisch* ausgewertet werden. Somit erscheint das Multimodell gegenüber einem Benutzer wie ein einziger abgeschlossener Informationsraum. Dies ermöglicht neuartige Filtermethoden, bei denen Informationen eines Fachmodells durch Linkauswertung auch über Kriterien anderer Fachmodelle gefiltert werden können (vgl. Abbildung 3.4). Damit entstehen so genannte *Multimodell-Views*, welche eine reduzierte und über die ursprünglich separaten Domänen kombinierte Datenmenge des Multimodellverbundes wiedergeben.

3.2.4. Multimodell-Templates

Die Multimodell-Metadaten erlauben die semantische Beschreibung von Multimodell-Inhalten. So können beispielsweise für Elementarmodelle Informationen über deren Domäne, Datenformat,

[6] Autodesk® Navisworks®; `http://www.autodesk.de/navisworks-features`
[7] RIB iTWO®; `www.rib-software.com/itwo`
[8] Synchro Software Ltd; `http://www.synchroltd.com/`
[9] Vico Software, Inc., Vico Office Suite; `http://www.vicosoftware.com/products/Vico-Office/`

Erstellungszeitpunkt, Gültigkeitsbereich, Projektstatus usw. hinterlegt werden. Ebenso können auch Linkmodellen Metadaten für ihren fachlichen Verlinkungszweck zugeordnet werden[10]. Gemeinsam mit ihren Nutzdaten bilden Metadaten den fachlichen *Ist*-Zustand eines Multimodells ab. Werden die Metadaten erstellt, *bevor* die korrespondierenden Nutzdaten existieren, können sie als fachlicher *Soll*-Zustand eines Multimodells interpretiert werden. Dadurch können Vorgaben zur Erstellung von Multimodellen mit definiertem Inhalt formal spezifiziert werden.

Multimodell-Template Ein Multimodell-Container der nur Metadaten zum Zweck einer Multimodell-Vorgabe enthält, wird Multimodell-Template genannt (Hilbert & Scherer, 2012).

Über Multimodell-Templates (MMT) können Bauinformationsprozesse einen höheren Automatisierungsgrad erlangen. Die Formalisierung der Anforderungen an Elementar- und Linkmodelle erlaubt die Generierung, Verwaltung, Versendung und Analyse von Templates sowie den Vergleich von Multimodell-Vorgabe und -Ergebnis durch Computerprogramme. Einen ähnlichen Ansatz beschreibt Henckels (2005).

3.2.5. Schlussfolgerung

Die multimodellbasierte Arbeitsweise ist durch neue Teilprozesse im Informationsablauf gekennzeichnet. Diese können automatisiert und wissensbasiert in Fachanwendungen hinterlegt sein oder durch einen Experten unter Zuhilfenahme einer universellen Multimodell-Software manuell ausgeführt werden:

1. Elementarmodelle müssen statt bisher gedanklich, implizit und flüchtig, nun explizit und persistent verlinkt werden.
2. Elementarmodelle und Links müssen annotiert und in Multimodell-Container verpackt werden.
3. Links können ausgewertet werden. Dadurch sind Multimodell-Views zur Erschließung des domänenübergreifenden Informationsraums erzeugbar.
4. Die Nutzung von Metadaten erlaubt die Erzeugung von Multimodell-Templates als formale Anforderungsbeschreibung. Anforderung und Arbeitsergebnis können verglichen werden.

Abbildung 3.5 auf der nächsten Seite gibt einen Überblick über die im Mefisto-Projekt identifizierten Teilprozesse zur Erstellung und Nutzung von Multimodellen (Schapke & Fuchs, 2011). Einige Teilprozesse wie *Elementarmodelle generieren, Aufgabenerfüllung überprüfen* und *Multimodell-Container veröffentlichen* sind spezifisch und setzen weitere Softwarekomponenten voraus. Vorhandene Elementarmodelle (Bereich links) werden durch Verlinken und Annotieren zu Multimodellen innerhalb einer Applikation ($MM\,2$). Durch Verpacken und möglicherweise weiteres Annotieren können die Daten als Multimodell-Container (MMC) serialisiert und versendet werden. Der Empfänger kann die Informationen des Multimodells visualisieren, mit dem Inhalt anderer Multimodell-Container ($MMC\,1$) vergleichen oder gegen ein Multimodell-Template (MMT) validieren. Außerdem können auf Basis der Informationen aus $MM\,2$ weitere Multimodelle ($MM\,3a \ldots d$) erzeugt werden, bspw. durch Multimodell-Filtern, Umwandlung der Elementarmodelle in andere Datenformate (Mappen) oder Verringern bzw. Erhöhen des Detaillierungsgrades der Elementarmodelle (Verdichten / Expandieren). Dabei kann es notwendig werden, auch die Linkmodelle anzupassen.

[10] vgl. auch Abschnitte 3.5.3 und 3.5.4 ab Seite 74

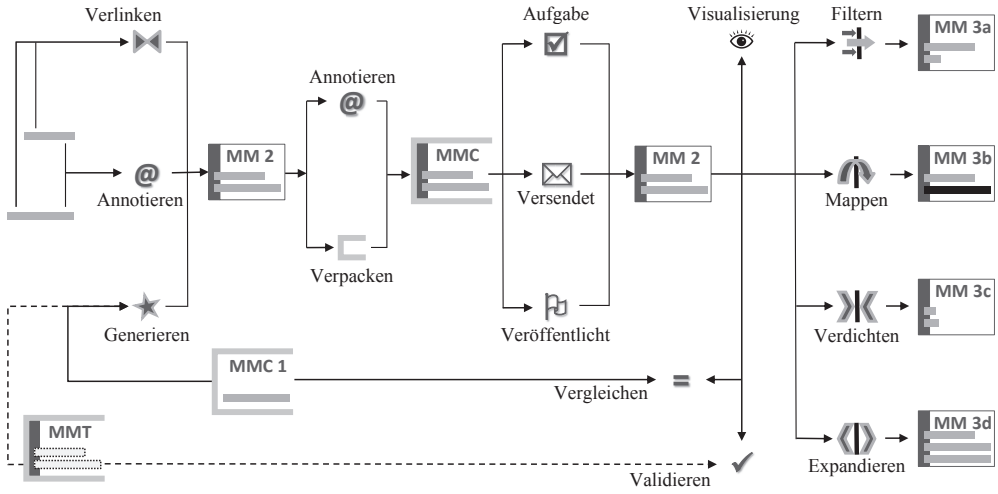


Abbildung 3.5.: Teilprozesse zur Multimodell-Erstellung und -Nutzung (aus Schapke & Fuchs, 2011). MM: Multimodell (innerhalb einer Applikation); MMC: Multimodell-Container; MMT: Multimodell-Template

3.3. Prinzip und Aufbau von Multimodellen

3.3.1. Schematischer Überblick

Um domänenübergreifende Informationen aus einem Multimodell gewinnen zu können, muss der Zugriff auf die zugrunde liegenden Daten geregelt werden. Dabei sind folgende drei Aspekte zu betrachten:

1. Das *Generische Multimodell*: die Datenstruktur des Multimodells zur Bündelung und Verlinkung von Fachmodellen
2. Das *Ideelle Elementarmodell*: die idealisierte Struktur der Elementarmodelle zum generischen Zugriff auf ihre Inhalte
3. *ID-basierte Links*: die lose Kopplung von Generischem Multimodell und Ideellem Elementarmodell.

Ein Multimodell wird von einem Nutzer als singuläre Informationseinheit behandelt. Daher wird mit dem Generischen Multimodell eine Datenstruktur geschaffen, die einen Einstiegspunkt zur Navigation von Elementarmodellen und Links unterstützt. Im Wesentlichen bietet sie die Möglichkeit, eine Menge von Elementarmodellen aufzunehmen, ohne deren Bedeutung oder innere Struktur zu kennen. Außerdem können Beziehungen zwischen einzelnen Elementen der Elementarmodelle als gruppierte Links in verschiedenen Linkmodellen aufgenommen werden. Aufgrund der unbekannten, nicht navigierbaren inneren Struktur der Elementarmodelle werden Links mithilfe eindeutiger Zeichenketten abgebildet. Diese repräsentieren die IDs der verlinkten Elemente. Die beschriebenen Komponenten können zusätzliche Metadaten in Form von Schlüssel-Wert-Paaren aufnehmen. Durch eine geeignete Serialisierung des Generischen Multimodells in Containern wird ein neutraler Austausch der Multimodell-Informationen gewährleistet.

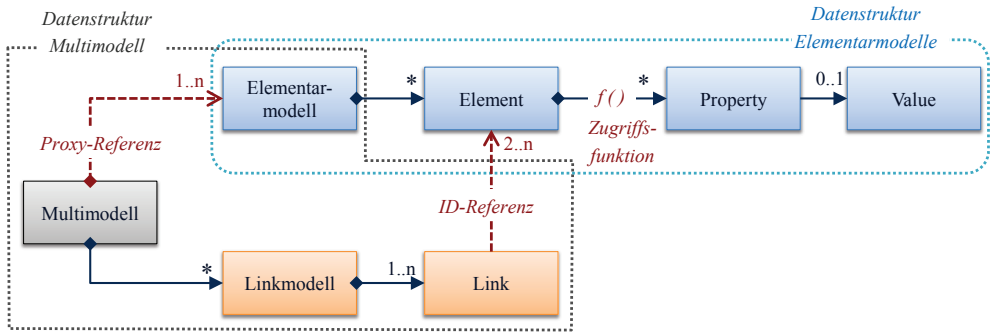


Abbildung 3.6.: Schematischer Aufbau von Multimodellen

Die Verarbeitung von Multimodell-Informationen erfordert darüber hinaus den Zugriff auf Daten innerhalb der Elementarmodelle. Dieser Zugriff muss in einem Softwaresystem generisch erfolgen; d. h. auf die gleiche Art und Weise – unabhängig von Datenformat und Semantik des einzelnen Elementarmodells. Dazu wird mit dem Ideellen Elementarmodell der innere Aufbau von Elementarmodellen verallgemeinert und idealisiert abgebildet. Erweiterungen des Softwaresystems bewerkstelligen die Übersetzung von Navigationszugriffen im Ideellen Elementarmodell auf die konkreten Elementarmodell-Datenstrukturen. Die Kernkonzepte des Ideellen Elementarmodells sind zum einen per ID auffindbare Elemente und zum anderen Properties als Behälter der eigentlichen Werte. Die Navigation von Element zu Property erfolgt über Zugriffsfunktionen um komplexe oder implizite Modellierungskonstrukte der zugrundeliegenden Elementarmodell-Datenstrukturen zu berücksichtigen.

Abbildung 3.6 zeigt den resultierenden schematischen Aufbau von Multimodellen. Der Bereich *Datenstruktur Multimodell* spiegelt Aspekt 1, das Generische Multimodell, wider. Der Bereich *Datenstruktur Elementarmodelle* repräsentiert das Ideelle Elementarmodell (Aspekt 2). *Proxy-Referenz* und *ID-Referenz* sind die Methoden zur losen Kopplung der beiden Datenstrukturen (Aspekt 3). Die genannten Konzepte werden im Folgenden detailliert beschrieben.

3.3.2. Das Generische Multimodell

Das Generische Multimodell ist die Definition des Datenschemas eines allgemeinen Multimodells zum Zweck der internen Repräsentation in einem Softwaresystem sowie zum neutralen Datenaustausch. Es soll das Multimodell-Konzept unabhängig von einem fachlichen Anwendungsgebiet unterstützen. Der Fokus liegt daher auf der Ermöglichung von Links zwischen Fachmodellen unterschiedlicher Domänen sowie der expliziten und neutralen Darstellung dieser Links zum Zweck des Datenaustauschs und ihrer Wiederverwendung. Außerdem soll das Generische Multimodell sowohl direkt nutzbar als auch individualisierbar sein[11]. Fuchs et al. leiten aus diesen Randbedingungen die folgenden Anforderungen an ein solches Datenschema ab (Fuchs et al., 2011, S. 207):

1. Das Schema muss *generisch* im Sinne von *domänenneutral* sein.
2. Das Schema muss *konkret* sein, d. h. es muss die Ad Hoc-Erzeugung valider Multimodell-Instanzen erlauben.

[11] vgl. Abschnitt 3.5 auf Seite 72

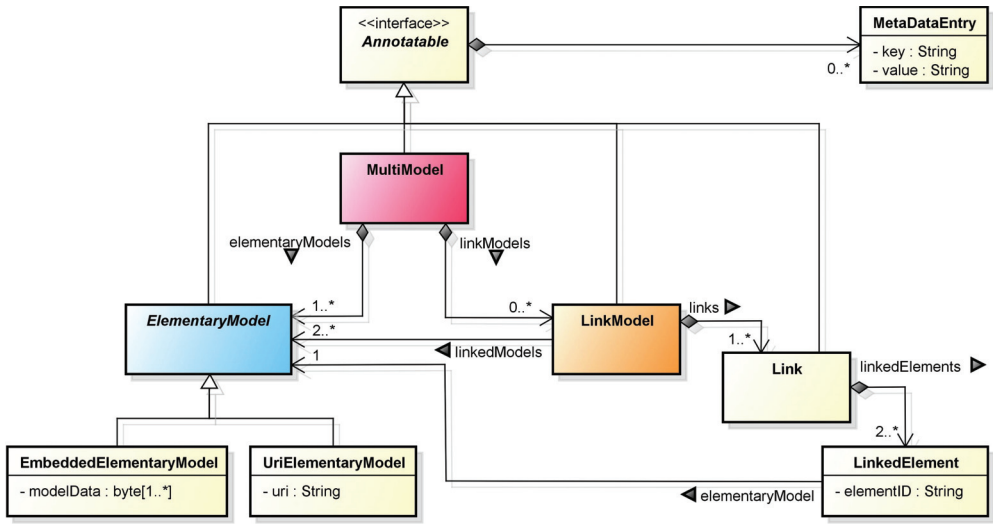


Abbildung 3.7.: UML-Klassendiagramm des Generischen Multimodells

3. Das Schema muss das *Speichern* (möglicherweise semantisch orthogonaler) Fachmodelle erlauben.
4. Diese Fachmodelle dürfen *nicht modifiziert* werden.
5. Referenzen zwischen Elementen unterschiedlicher Fachmodelle müssen persistent und wiederherstellbar sein.
6. Die Kardinalität dieser Referenzen ist ≥ 2.
7. Das Schema muss erweiterbar und adaptierbar sein, d. h. es muss möglich sein, Konverter für unterschiedliche Containerformate zu erstellen.
8. Es muss eine Referenzimplementierung möglich sein, welche die Instanziierung und Manipulation konformer Multimodell-Instanzen ermöglicht.
9. Es muss ein Standardformat zur Serialisierung von Multimodell-Instanzen geben.

Das Generische Multimodell ist als objektorientiertes Modell umgesetzt. Durch das objektorientierte Modellieren (Rumbaugh et al., 1993) können die verschiedenen Anforderungsaspekte erfüllt werden:

- Abbildung inhaltlich struktureller Anforderungen (bspw. Kardinalitäten)
- Anforderungen an prinzipielle Erweiterbarkeit (bspw. durch Erweiterung mittels Subklassen)
- Anforderungen an Ausführbarkeit (bspw. durch Implementierung in einer objektorientierten Programmiersprache)
- Anforderungen an den Datenaustausch (bspw. durch werkzeuggestützte automatische Ableitung eines korrespondierenden Serialisierungsformats).

Abbildung 3.7 zeigt das resultierende Klassendiagramm des Generischen Multimodells. Die Klasse `MultiModel` repräsentiert das Konzept der Multimodelle. Sie enthält $1..n$ Elementarmodelle (Klasse `ElementaryModel`) und $0 \ldots n$ Linkmodelle. Elementarmodelle können

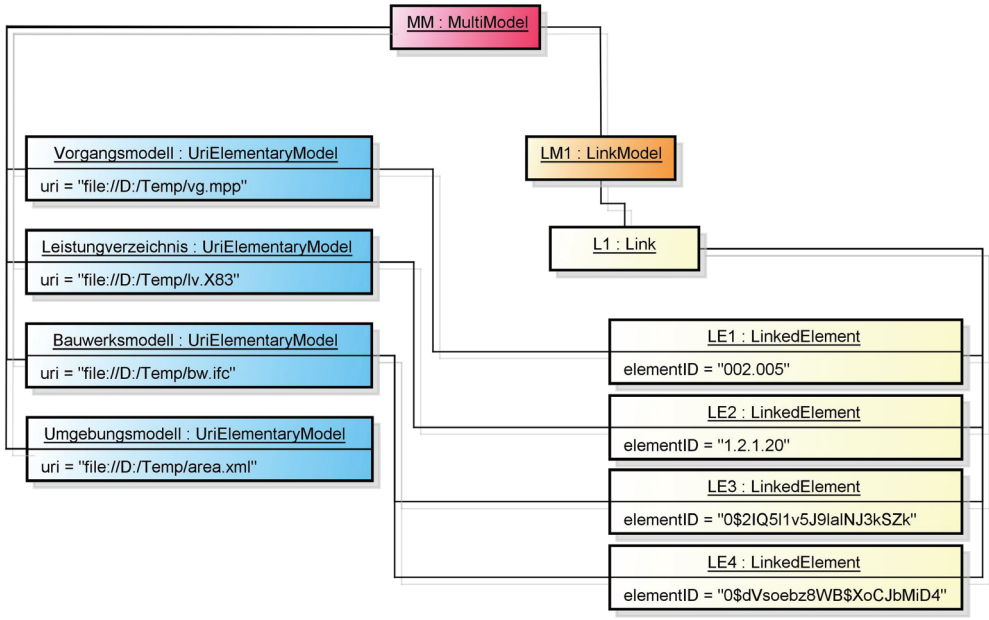


Abbildung 3.8.: UML-Objektdiagramm für einen Link eines exemplarischen Multimodells

entweder als Array von Bytes eingebettet werden oder per URI den physischen Speicherort des Fachmodells referenzieren. Durch die URI-Referenzierung lässt sich die zu übertragende Datenmenge beim Multimodell-Austausch wesentlich reduzieren – insbesondere Bauwerksmodelle wie IFC können mehrere 100 MByte groß sein. Im Sinne der losen Kopplung zwischen Multimodell und Elementarmodell sind die `ElementaryModel`-Klassen lediglich Stellvertreter für das eigentliche Fachmodell. Es handelt sich um eine Anwendung des Proxy-Entwurfsmusters (Gamma et al., 2004).

Die Klasse `LinkModel` repräsentiert das Konzept der aufgabenspezifischen Verlinkung. Jede Instanz entspricht einer individuellen Kombination von Elementen. Dazu enthält die Klasse $1..n$ Links. Außerdem enthält sie einen Verweis auf diejenigen Elementarmodelle, deren Elemente in dieser Instanz verlinkt sind. Damit wird einem Softwaresystem die inhaltliche Analyse vorliegender Linkmodelle vereinfacht. Es müssen nicht alle Links überprüft werden, um herauszufinden, ob das Linkmodell für eine konkrete Aufgabenstellung relevant ist.

Ein Link verbindet mindestens zwei Elemente der Elementarmodelle. Elemente sind jedoch nicht direkt referenzierbar. Zum einen ist dies technisch unmöglich, da aus der Sichtweise des Generischen Multimodells die interne Datenstruktur der Elementarmodelle unbekannt ist. Zum anderen ist eine explizite lose Kopplung gefordert, welche nicht durch Klassenassoziationen erreicht werden kann. Die Klasse `Link` besteht deswegen aus $2 \ldots n$ Instanzen der Klasse `LinkedElement`. Ein LinkedElement beschreibt stellvertretend ein verlinktes Element. Dafür kennt es seine ID (Attribut `elementID`) sowie das zugehörige Elementarmodell, da IDs nur innerhalb *eines* Elementarmodells einzigartig sein müssen.

Abbildung 3.8 zeigt einen Ausschnitt eines Objektdiagramms für ein exemplarisches Multimodell. Das Linkmodell `LM1` verbindet drei der vier im Multimodell vorhandenen Elementarmodelle.

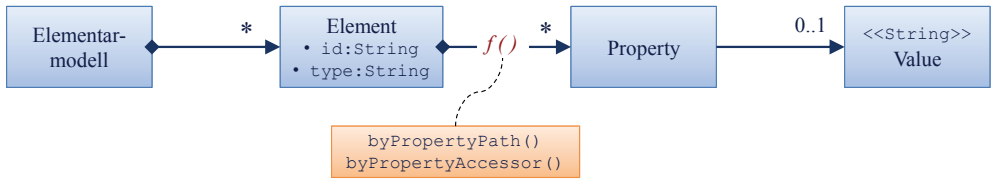


Abbildung 3.9.: Komponenten des Ideellen Elementarmodells mit den Zugriffsfunktionen für Properties

Der Link `L1` beschreibt eine Relation zwischen *einem* Element des Vorgangsmodells, *einem* Element des Leistungsverzeichnisses und zwei Elementen des Bauwerksmodells. Es handelt sich demzufolge um eine mehrwertige Relation mit vier Elementen aus drei Elementarmodellen. Weitere Links können analoge Relationen herstellen. Tabelle 3.3 zeigt `L1` und weitere Links in kompakter Darstellung. Die zu einem Link gehörenden `LinkedElement`-Instanzen werden nach ihren Elementarmodellen gruppiert; ihre IDs werden kommasepariert aufgeführt.

Alle Klassen außer LinkedElement implementieren das Interface `Annotatable`. Ihren Instanzen ist es dadurch möglich, beliebige Metadaten als Zeichenketten in Schlüssel-Wert-Paaren zu erhalten. Inhalt und Bedeutung der Metadaten werden nicht im Datenschema, sondern in externen Katalogen geregelt. Im Sinne des Multimodell-Konzepts soll es sich dabei um Informationen zur Unterstützung des Austauschs verlinkter Fachmodelle handeln. Die Klasse `LinkedElement` besitzt diese Funktionalität ausdrücklich nicht. Es soll vermieden werden, dass Multimodelle dazu benutzt werden, Fachinformationen auf Ebene der Elemente außerhalb der zuständigen Elementarmodelle abzubilden.

Das vollständige Datenschema des Generischen Multimodells ist in Anhang A.1 als äquivalente, automatisch generierte, XML-Schema Definition wiedergegeben. Dieses XSD ist die Definition eines neutralen Serialisierungsformats zum Austausch von Multimodellen.

3.3.3. Das Ideelle Elementarmodell

Das Ideelle Elementarmodell ist eine virtuelle Struktur zum generischen Datenzugriff auf die Inhalte von Elementarmodellen, unabhängig von deren Datenformat. Dazu wird eine verallgemeinerte, idealisierte Abbildung gängiger Datenformate geschaffen. Um eine möglichst große Anzahl von Fachmodellen repräsentieren zu können, muss diese Struktur genügend abstrakt und mit dem Generischen Multimodell harmonisiert sein. Abbildung 3.9 zeigt das Ideelle Elementarmodell.

Ein Elementarmodell besteht aus 0..n Elementen. Elemente sind die extern verlinkbaren

Tabelle 3.3.: Kompakte Darstellung einer Linkmodell-Instanz in tabellarischer Form

Link	Leistungsverzeichnis	Bauwerksmodell	Vorgangsmodell
L1	1.2.1.20	0$2IQ5l1v5J9lalNJ3kSZk, 0$dVsoebz8WB$XoCJbMiD4	002.005
L2	1.2.1.20	0fhU7jPFB9KwEQGZe595	
L3	1.2.1.50, 1.2.1.70		002.005, 002.006
⋮	⋮	⋮	⋮

Komponenten. Dafür müssen sie referenzierbar sein. Elemente besitzen deshalb eine ID – eine Zeichenkette (String), die innerhalb des Elementarmodells eindeutig ist. Im Ideellen Elementarmodell erlangen Elemente anhand ihrer ID eine Identität. ID und Elementinstanz sind demnach äquivalent. Eine weitere Eigenschaft von Elementen ist ihre Zugehörigkeit zu einem Typ (type). Die Struktur des Ideellen Elementarmodells orientiert sich an hierarchisch strukturierten Modellen. Die Verschachtelung von Elementen wie sie bspw. in XML möglich ist, wird hier durch eine Dekomposition aufgelöst. Alle Elemente befinden sich auf derselben Hierarchieebene. Der Zugriff auf Elemente erfolgt einzeln über ihre ID oder gruppiert über ihren Typ. Für jedes Elementarmodell muss deswegen mindestens *ein* Element-Typ existieren. Zwei Elemente gehören zum selben Typ, wenn sie gleiche Properties besitzen.

Ein Property ist eine von seinem Element aus zugreifbare Komponente hinter der sich ein Value, die eigentliche Information, befindet. Der Wert wird als String behandelt und darf unbesetzt sein. Elemente besitzen $0..n$ Properties. Der Zugriff auf die Properties erfolgt über Funktionen. Diese führen die eigentliche Navigation in der zugrundeliegenden realen Datenstruktur aus. Je nach Aufbau des konkreten Datenformats kann jedes Property eine eigene Zugriffsfunktion benötigen.

Die verschachtelte oder verkettete Anordnung von Informationen ist ein bewährtes Muster zur Bewältigung der Komplexität in der Datenmodellierung und ist in nahezu allen Baufachmodellen wiederzufinden. Katranuschkov (2000) führt daher im Kontext der Mappingsprache CSML das Telescope-Pattern ein. Leser & Naumann (2007) beschreiben unter anderem *Geschachtelt→Flach* als relevante Wertkorrespondenz beim Schema-Mapping. Im Ideellen Elementarmodell wird die Funktion `byPropertyPath()` verwendet, um zu einem Property in hierarchischen Substrukturen zu navigieren. Dazu wird der Funktion der relative Pfad zum Property übergeben, weswegen einem Aufrufer die materielle Datenstruktur des Elementarmodells bekannt sein muss. Pfade können trivial sein und beispielsweise den Zugriff von einem XML-Element auf sein Attribut realisieren. So liefert etwa `qty` das Mengenattribut des Elements Leistungsverzeichnisposition aus einem Elementarmodell vom Typ GAEB-DA-XML. Das Startdatum eines Vorgangs aus dem Mefisto-Vorgangsmodell ist in der dritten Ebene definiert und wird mit `activityData/start/date` angesprochen. Mehrwertig definierte Substrukturen können durch Angabe des nullbasierten Listenindex navigiert werden. Beispielsweise liefert `Addresses[2]/Country` das Länderattribut der dritten Adresse eines IfcOrganisation-Elements eines IFC-Modells.

Sind Properties eines Elements nicht über einen hierarchischen Pfad erreichbar, soll dieser vor einem Anwender verborgen werden oder ist zusätzliche Logik notwendig, können sogenannte Property-Accessors eingesetzt werden. Die Funktion `byPropertyAccessor()` ruft benannte Software-Bausteine auf, welche definierte Filteroperationen für einen speziellen Element-Typ ausführen. Die Bausteine können Parameter festlegen, über die ihr Verhalten gesteuert wird. Zum Beispiel sind in IFC-Modellen die Schlüssel-Wert-Eigenschaftswerte (IFC-Property) eines Elements nur über Relationsobjekte ermittelbar, nicht jedoch direkt vom Element aus navigierbar. Ein Property-Accessor mit dem Namen *IfcProperty* kann auf den IFC-Property-Wert eines Elements zugreifen, indem als Parameter der Schlüssel angegeben wird. `IfcProperty -FireRating` liefert demnach die Feuerwiderstandsklasse eines IfcWall-Elements. In GAEB-DA-XML dürfen bestimmte Texte HTML-ähnlich formatiert angegeben werden. `Text -Outline` gibt beispielsweise den unformatierten Kurztext einer Leistungsverzeichnisposition zurück. Property-Accessors können demnach auch semantisch hochwertige Filteroperationen ausführen und dadurch sogar Properties bereitstellen, die das zugrundeliegende Datenformat nicht direkt vorsieht. Denkbar wäre zum Beispiel die exakte geometrische

Volumenberechnung für IFC-Bauteilelemente.

Der Zugriff auf Properties wird über die beiden Funktionen teilweise an die Implementierungen für die jeweiligen Elementarmodell-Typen delegiert. In Konsequenz können auch die inhärenten Elementeigenschaften `id` und `type` als Property aufgefasst werden. Die korrespondierenden Zugriffsfunktionen sind `id()` und `type()`. Allgemein ermöglicht der Einsatz externer Logik, beliebige Modellierungskonstrukte der materiellen Datenstrukturen einzubeziehen und im Ideellen Elementarmodell anwendbar zu machen. Die Implementierungen können generische Vorlagen verwenden, beispielsweise die Filtersprache XPath für XML-basierte Datenstrukturen.

Das Ideelle Elementarmodell ist zur *Laufzeit* in Multimodell-Softwaresystemen relevant. Dabei ist seine Struktur nicht als Implementierungsanweisung zu verstehen, sondern als vereinheitlichte Kommunikationsschnittstelle für Datenzugriffe. Das Konzept ermöglicht es, Daten der Elementarmodelle auf Anforderung detailliert bereitzustellen, wodurch ein vorgelagertes und vollständiges Mapping aller am Multimodell beteiligten Datenstrukturen vermieden wird.

3.3.4. ID-basierte Links

Die lose Kopplung von Elementarmodellen im Multimodell beruht auf der Verwendung von ID-basierten Links. Dabei dienen Zeichenketten als Identifikator für Elemente. Zeichenketten sind leicht serialisierbar und behalten auch außerhalb des Elementarmodells ihre Bedeutung. Dies ermöglicht das externe Aufstellen, Speichern und Nachverfolgen von Elementreferenzen auf Datenebene. Dadurch können zum einen Informationseinheiten in Elementarmodellen verlinkt werden ohne dass diese geändert werden müssen. Zum anderen ist zum bloßen Datenzugriff auf ID-basierte Links prinzipiell kein korrespondierendes Softwaresystem notwendig, wie dies beispielsweise bei relationalen und einigen nicht-relationalen Datenbanken der Fall ist (Pernul & Unland, 2003; Edlich et al., 2010).

Der Multimodell-Ansatz fordert demnach die Existenz von ID-Attributen[12] in jedem Elementarmodell. Für den Fall, dass keine belegten ID-Attribute vorhanden sind, beispielsweise bei unzulänglicher Modellqualität oder weil das Datenschema überhaupt keine solchen Attribute vorsieht, können die folgenden Ausweichlösungen angewendet werden:

1. Nicht manipulative Ausweichlösungen

   Nicht manipulative Methoden lassen das Elementarmodell unberührt. Sie schließen aus den existierenden Daten auf die Identität eines Elements.

   a) In Relationalen Datenbanksystemen gespeicherte Elementarmodelle: Der Primärschlüssel des Elements kann direkt als ID verwendet werden. Zusammengesetzte Schlüssel können durch einen geeigneten Separator, bspw. Komma oder Semikolon, konkateniert werden. Im Multimodell sollen Metadaten-Einträge für das Elementarmodell dem Empfänger eines Multimodells den Zugriff auf diese Elemente vereinfachen. Die Schlüssel `cib.model.artificialID.rdb.pk.{db | schema | table | column}` können dafür benutzt werden, den korrespondierenden Ort der ID in der Datenbank, respektive Schema, Tabelle und Spalte, zu beschreiben. Primärschlüssel relationaler Datenbanken sind stabile IDs, da Datenbanken für eine permanente Identität ihrer Objekte konzipiert sind. Die Methode ist auf andere Persistenzsysteme übertragbar, wenn diese eine inhärente, dauerhafte und als Zeichenkette repräsentierbare Identität ihrer Objekte bereitstellen.

[12] vgl. Abschnitt 2.1.4 auf Seite 16

Die nachfolgenden Methoden verwenden deskriptive oder strukturelle Eigenschaften eines Elements um seine Identität zu beschreiben. Da sich diese Eigenschaften bei einer zukünftigen Bearbeitung des Elementarmodells ändern können, sind die so ermittelten IDs potentiell instabil. Die strukturelle Position eines Elements muss nach jeder Veränderung des Elementarmodells überprüft und die ID möglicherweise neu erzeugt und im Linkmodell geändert werden.

b) Feature-Path: Die Identität eines Elements wird über den Navigationsweg in der Datenstruktur des Elementarmodells ermittelt. Dabei werden die Features, bspw. Referenzen oder Attribute, ausgehend vom Wurzelknoten in der angegebenen Reihenfolge ausgewertet. In einem fiktiven Bauwerksmodell könnte der Ausdruck `Root→Building→Roof` auf das Dachelement eines Gebäudes verweisen. Ein Metadaten-Eintrag `cib.model.artificialID.method = FeaturePath` kennzeichnet die Anwendung dieser Methode. Für mehrwertige Features wie Listen hat diese Methode ein undefiniertes Verhalten.

c) Feature-Path mit Collection-Index: Die Methode Feature-Path wird hier über die optionale Angabe eines nullbasierten Collection-Index erweitert. Dadurch können mehrwertige Features berücksichtigt werden. Der Ausdruck `Root→Building(1)→Wall(17)` würde demnach die 18. Wand des 2. Gebäudes eines fiktiven Bauwerksmodells liefern. Ein nicht gesetzter Collection-Index liefert für mehrwertige Features das erste Element. Für diese Methode ist der Metadaten-Eintrag `cib.model.artificialID.method = CollectionIndex` vorgesehen. Es wird vorausgesetzt, dass mehrwertige Features im Elementarmodell eine Ordnung haben.

Die im Anschluss genannten Methoden beschreiben Elemente anhand der Wertausprägungen ihrer Attribute. Dadurch liefern sie nicht die Identität *eines* Elements, sondern eine Untermenge von Elementen gleichen Typs. Diese Untermenge kann potentiell leer sein, genau *ein* oder mehr als *ein* Element besitzen. Daher ist ein erhöhter Aufwand zur Erzeugung der IDs notwendig, wenn Wertausprägungen von Attributen als ID verwendet werden. Um ID-Konflikte mit anderen Elementen der gleichen Klasse zu vermeiden, müssen so viele Attribute zur Beschreibung eines Elements herangezogen werden, bis kein anderes Element die gleiche Wertausprägung besitzt. Dies ist in der Praxis jedoch nicht immer möglich, da es durchaus Elemente mit gleicher Attributbelegung geben kann. Eine erhöhte Wahrscheinlichkeit dafür besteht insbesondere bei stark objektifizierten Datenstrukturen wie IFC, bei denen einzelne Klassen relativ wenige Attribute besitzen und die meisten Eigenschaften wie Geometrie nur indirekt navigierbar sind. Die so entstehenden IDs sind demnach bei Modelländerungen in zweifacher Hinsicht potentiell instabil. Die in einer ID benutzten Attributwerte können sich zum einen im korrespondierenden Element ändern, zum anderen aber auch in allen anderen Elementen des gleichen Typs. In beiden Fällen muss eine solche ID möglicherweise neu erzeugt und im Linkmodell geändert werden. Darüber hinaus kann der Fall eintreten, dass es nicht mehr möglich ist, das Element eindeutig über seine Attributwerte zu beschreiben.

d) Attributwerte: Die Identität eines Elements wird, möglicherweise nicht eindeutig, über die Belegung einiger oder aller seiner Attributwerte beschrieben. Durch einen geeigneten Separator, bspw. Komma oder Semikolon, werden die einzelnen Ausdrücke getrennt. So könnte eine ID einer Wand eines fiktiven Bauwerksmodells beispielsweise `height=3.30, length=4.20, width=0.24` sein. Dabei soll gelten, dass alle Attributbelegungen übereinstimmen müssen – sie sind gleichsam durch ein logisches

UND verknüpft. Ein Metadaten-Eintrag `cib.model.artificialID.method = AttributeValues` kennzeichnet die Anwendung dieser Methode.

e) Elementarmodell-Filter: Die vorstehend beschriebene Methode wird dahingehend erweitert, dass beliebige logische Aussagen über ein Element gültig sind. In Konsequenz bedeutet das den Einsatz von Elementarmodell-Filtern. Elementarmodell-Filter sind in diesem Zusammenhang ausführbare Programme, die über einen Namen identifizierbar sind. Dementsprechend wird dieser in einem Metadaten-Eintrag `cib.model.artificialID.filter = %filterName%` hinterlegt. Die Konfiguration des Filters ist eine konkrete Anfrage, die als Zeichenkette abgebildet und als ID verwendet wird. Die Form des Abfrageausdruck ist allein vom Elementarmodell-Filter abhängig. Diese Verfahrensweise eignet sich nur bedingt als Identitätsbeschreibung, da sie potentiell mehrdeutige Ergebnisse liefert.

2. Manipulative Ausweichlösungen

   Manipulative Methoden generieren IDs und speichern diese im Elementarmodell. Damit verstoßen sie zwar gegen das Multimodellprinzip, Fachmodelle nicht zu verändern – sie sind jedoch eine Alternative für Fälle, in denen die Anwendung nicht manipulativer Methoden unmöglich ist. Manipulative Methoden erzeugen potentiell stabile IDs, da sie die uneingeschränkte Kontrolle über die ID-Generierung besitzen. Techniken zur ID-Generierung sind aus dem Umfeld der Relationalen Datenbanksysteme bekannt (Garmany et al., 2005; ISO 9834-8, 2004).

   a) Das Elementarmodell-Schema sieht ID-Attribute vor, sie sind in der Modell-Instanz jedoch nicht belegt: Fehlende IDs können nach der Direktive des Schemas erzeugt und an den dafür vorgesehenen Stellen gesetzt werden. Im Multimodell sollte das Elementarmodell einen Metadaten-Eintrag `cib.model.generatedID = true` erhalten.

   b) Das Elementarmodell-Schema sieht keine ID-Attribute vor: Falls das Datenformat Kommentare unterstützt, beispielsweise wie XML oder INI, können IDs generiert und dort gesetzt werden. Dafür sollte eine geeignete Zeichenkette der ID vorangestellt werden, um den speziellen Kommentar von regulären Anmerkungen zu unterscheiden. Der gewählte Präfix sollte im Multimodell im Metadaten-Eintrag des Elementarmodells propagiert werden: `cib.model.artificialID.comment = %prefix%`.

   c) Das Elementarmodell-Schema sieht keine ID-Attribute vor: ID-Attribute können den Elementen hinzugefügt werden, falls das Datenformat dies generell unterstützt, zum Beispiel XML oder JSON. Dazu muss ein einzigartiger und repräsentativer Attributname gewählt werden. Dieser sollte im Multimodell im Metadaten-Eintrag des Elementarmodells propagiert werden: `cib.model.artificialID.attribute = %attributeName%`. In den neuen Attributen können nun generierte IDs gesetzt werden. Das Hinzufügen neuer Attribute verletzt das Elementarmodell-Schema und kann zu Inkompatibilitäten beim Datenaustausch führen.

3.3.5. Formalisierung der strukturellen Multimodellkomponenten

In den vorangegangenen Abschnitten wurde der strukturelle Aufbau von Multimodellen erörtert. Die beschriebenen Konzepte ermöglichen die strukturierte Abbildung von Multimodelldaten.

Damit wird der Informationsraum domänenübergreifender Bauinformationsprozesse gebildet. Die strukturellen Kernkomponenten mit ihren Beziehungen wurden bereits in Abbildung 3.6 präsentiert. Sie werden im Folgenden formalisiert. Auf dieser Basis können dann Methoden zur gezielten Erschließung des komplexen Informationsraumes aufgestellt werden[13].

Value Ein Value v ist die kleinste Informationseinheit im Multimodell. Sie wird durch eine endliche Zeichenkette repräsentiert. Aufgrund des generischen Ansatzes des Multimodell-Paradigmas[14] ist keine strukturelle Transformation in andere Datentypen vorgesehen. Values haben immer einen Wert. Der Wert `null` steht dabei für den definierten Zustand, welcher das Fehlen eines Wertes symbolisiert.

$$\begin{aligned} \mathbb{T} &:= \text{Menge der in einem Automaten repräsentierbaren Zeichenketten} \\ V &:= \mathbb{T} \cup \{null\} \end{aligned} \tag{3.1}$$

Property Properties sind die kleinste Einheit zur Datenstrukturierung im Multimodell. Ein Property p ist eine Abbildung von einem Propertyschlüssel s zu einem zugehörigem Value v.

Propertyschlüssel Während sich die tatsächlich relevante Information in v befindet, kann der Propertyschlüssel s vereinfachend als eindeutiger Name des Properties innerhalb eines Elements verstanden werden. s ist eine hinreichende Bedingung zur Ermittlung der Identität des Properties. Aufgrund der hohen möglichen Komplexität der zu nutzenden Datenstrukturen werden mit Property-Zugriffsfunktionen jedoch flexiblere Mechanismen zur Ermittlung der Identität eines Properties benötigt. s wird daher als flüchtiges, unterscheidbares Objekt definiert, welches von solchen Funktionen erzeugt wird[15].

$$\begin{aligned} S &:= \text{Menge der aus Property-Zugriffsfunktionen erzeugbaren Schlüssel} \\ p &:= s \mapsto v \,|\, s \in S, v \in V \\ \forall s, p &: s \Rightarrow p \end{aligned} \tag{3.2}$$

Element Elemente sind die obere Einheit zur Datenstrukturierung im Multimodell. Zur genaueren Unterscheidung können sie auch *Multimodell-Elemente* genannt werden. Ein Element e besteht aus einer Menge von Properties p. Deren Schlüssel s müssen innerhalb des Elements eindeutig sein. Eineindeutigkeit ist nicht möglich, da ein identisches Property über unterschiedliche Zugriffsfunktionen erreichbar sein kann. Zusätzlich zu Gleichung 3.2 gilt somit: s ist nicht von p ableitbar.

$$\begin{aligned} e &:= \{p_1, \ldots, p_n\} \\ \forall p \in e &: p_i \neq p_j \,|\, i \neq j \\ \forall s, p &: s \not\Leftarrow p \end{aligned} \tag{3.3}$$

Identifikator (ID) Elemente haben eine hohe Relevanz für das Multimodell, da sie die verlinkbaren Informationseinheiten sind. Ihnen wird daher eine explizite Identität zugewiesen: der Identifikator (auch: die ID). Eine ID id ist eine endliche Zeichenkette, die innerhalb

[13] vgl. Kapitel 5 auf Seite 127
[14] vgl. Abschnitte 3.1 auf Seite 42 und 3.4 auf Seite 67
[15] vgl. Abschnitt 4.2.3 auf Seite 94

eines Elementarmodells eineindeutig ist. IDs und Elemente können somit als bedeutungsgleich betrachtet werden, da von dem Einen auf das Andere geschlossen werden kann. In Abschnitt 4.2.3 wird beschrieben, wie von einer ID auf das zugehörige Element zugegriffen werden kann. Umgekehrt ist jedem Element seine ID bekannt. Sie wird als inhärentes Property mit der Zugriffsfunktion $id()$ definiert.

$$\begin{aligned} id &\in \mathbb{T} \\ \forall id &: id \Leftrightarrow e \\ \forall id &: id \subseteq p \\ id(e) &:= e \mapsto id \end{aligned} \tag{3.4}$$

Elementarmodell Aus struktureller Sicht ist ein Elementarmodell em eine Menge von Elementen e. Jedes Element ist innerhalb eines Elementarmodells einzigartig, d. h. jede ID ist nur einmal vergeben. Aus funktionaler Sicht ist ein Elementarmodell eine übertragbare Instanz eines Datenmodells mit einer abgegrenzten Fachdomäne und einer vereinbarten Semantik. Das bedeutet, dass ein Elementarmodell nicht zwangsläufig ein korrespondierendes explizites Schema benötigt. Beispielsweise sind XML-Dokumente ohne XSD möglich. Lediglich die Bedeutung der Daten muss bekannt sein. Das Datenmodell muss sich in die ideelle Elementarmodell-Datenstruktur *Element→Property→Value* überführen lassen.

$$\begin{aligned} em &:= \{e_1, \ldots, e_n\} \\ \forall e \in em &: e_i \neq e_j \,|\, i \neq j \\ \forall id = id(e) &: id_i \neq id_j \,|\, i \neq j \end{aligned} \tag{3.5}$$

Element-Typ Element-Typen dienen der Klassifizierung von Elementen innerhalb eines Elementarmodells em. Ein Element-Typ et ist eine Menge von Elementen e mit gleichen Property-Schlüsseln $s(e)$[16]. Die Bildung von Element-Subtypen durch Erweiterung um zusätzliche Property-Schlüssel ist zulässig. Dabei soll das Liskovsche Substitutionsprinzip (Liskov & Wing, 1994) gelten, wodurch auch Elemente eines Subtyps mit den Property-Schlüsseln ihres Supertyps angesprochen werden können.

$$\begin{aligned} s(e) &:= e \mapsto s \,|\, s \in S, s \Rightarrow p, p \in e \\ et &:= \{e_1, \ldots, e_n \in em \,|\, s(e_1) = s(e_2) = \ldots = s(e_n)\} \end{aligned} \tag{3.6}$$

Elementarmodell-Typ Ein Elementarmodell-Typ emt ist diejenige Klasse von Elementarmodellen, welche durch das – möglicherweise nur implizit existierende – Metamodell ihrer Elementarmodelle gebildet wird. Besitzt ein Elementarmodell beispielsweise ein explizites Schema, so sind Elementarmodell-Typ und Schema äquivalent. Daraus folgt, dass Elementarmodelle vom selben Elementarmodell-Typ identische mögliche Element-Typen $ET(em)$ besitzen.

$$\begin{aligned} ET(em) &:= \{et_1, \ldots, et_n \,|\, e_{ij} \in em \wedge e_{ij} \in et_i\} \\ \forall em_1, em_2 &: emt(em_1) = emt(em_2) \Rightarrow ET(em_1) = ET(em_2) \end{aligned} \tag{3.7}$$

[16] Zum Beispiel ist eine IFC-Klasse ein Element-Typ. Die Attribute der Klasse sind die Properties und deren Attributnamen sind die Propertyschlüssel.

Link Links sind die strukturellen Komponenten zur Aufnahme expliziter Relationen zwischen n_e Elementen aus m_{em} Elementarmodellen. n_e und m_{em} sind grundlegende Kenngrößen der Multiplizität von Links. Die Anzahl der Relata n_e wird als **Kardinalität** eines Links bezeichnet. Die Anzahl der Elementarmodelle m_{em}, denen diese Elemente angehören, wird **Arität** eines Links genannt. Die Relationen sind mehrstellig (n-är, $n_{em_1} : n_{em_2} : \ldots : n_{em_m}$), mindestens jedoch zweistellig (binär, $1:1$). In Abschnitt 3.1 auf Seite 42 wurde erläutert, dass mindestens zwei Elementarmodelle verlinkt werden sollen. Daher gilt:

$$\begin{aligned} n_e &\geq 2 \\ m_{em} &\geq 2 \\ m_{em} &\leq n_e \\ Einwertiger\ Link &:= m_{em} = n_e \\ Mehrwertiger\ Link &:= m_{em} < n_e \end{aligned} \tag{3.8}$$

Verlinktes Element Die explizite Darstellung der Elemente in Links erfolgt über ihre IDs. Eine ID ist jedoch nur innerhalb *eines* Elementarmodells einzigartig. Zur Berücksichtigung gleicher IDs in unterschiedlichen Elementarmodellen wird das Konzept des *verlinkten Elements* (*linked element, le*) eingeführt. Ein verlinktes Element beschreibt die globale, multimodellweite Identität eines Elements, indem ein Schlüssel aus Element-ID und dem zugehörigen Elementarmodell zusammengesetzt wird. Strukturell ist ein verlinktes Element le ein geordnetes Paar aus einem Elementarmodell em und einer ID id eines Elements e aus em. Ein Link l ist somit eine Menge von mindestens zwei verlinkten Elementen le.

$$\begin{aligned} le &:= (em, id = id\,(e)) \mid e \in em \\ l &:= \{le_1, \ldots, le_n\} \mid n \geq 2 \end{aligned} \tag{3.9}$$

Linkmodell Ein Linkmodell lm besteht aus einer Menge von Links l. Zusätzlich werden alle Elementarmodelle em aufgeführt, von denen mindestens *ein* Element e Relatum von l ist. Diese Elementarmodelle werden *verlinkte Elementarmodelle* eines Linkmodells genannt. Diese Information ist redundant, da sie aus l abgeleitet werden kann. Die Redundanz wird jedoch durch den häufigen Gebrauch zur Inspektion von Linkmodellen sowie durch eine unkompliziertere Definition von Einschränkungen (Constraints) gerechtfertigt.

$$lm := (\{em_1, \ldots, em_m\}, \{l_1, \ldots, l_p\}) \mid m \geq 2, p \geq 1 \tag{3.10}$$

Multimodell Ein Multimodell mm besteht aus einer Menge von Elementarmodellen em und einer Menge von Linkmodellen lm. Dabei muss mindestens *ein* Elementarmodell vorhanden sein. Ein Multimodell ist auch ohne ein Linkmodell gültig. Multimodelle mit mindestens einem Linkmodell werden *echte Multimodelle* genannt. Aus Gleichung 3.10 folgt, dass echte Multimodelle aus mindestens zwei Elementarmodellen bestehen müssen.

$$\begin{aligned} mm &:= (\{em_1, \ldots, em_q\}, \{lm_1, \ldots, lm_r\}) \mid q \geq 1, r \geq 0 \\ \forall q, r &: \quad r \geq 1 \Rightarrow q \geq 2 \end{aligned} \tag{3.11}$$

Definition der Komponenten-Einschränkungen

Die nachfolgenden Regeln gelten für *ein* Multimodell mm:

Alle verlinkten Elementarmodelle $\widehat{em}$ eines Linkmodells lm müssen in mm enthalten sein.

$$\forall \widehat{em} \in lm \quad : \quad \widehat{em} \in mm \,|\, lm \in mm \tag{3.12}$$

Alle verlinkten Elementarmodelle $\widehat{em}$ eines Linkmodells lm müssen einzigartig sein.

$$\forall \widehat{em} \in lm \quad : \quad \widehat{em_i} \neq \widehat{em_j} \,|\, i \neq j \tag{3.13}$$

In den Links l des Linkmodells dürfen nur Elementarmodelle $e\acute{m}$ aus $\widehat{em}$ vorkommen.

$$\forall e\acute{m} \in le \quad : \quad e\acute{m} \in \widehat{em} \,|\, le \in l, l \in lm \tag{3.14}$$

Die verlinkten Elemente le eines Links l müssen einzigartig sein.

$$\begin{aligned} \forall le \quad &: \quad le_i \neq le_j \Leftrightarrow em_i \neq em_j \vee id_i \neq id_j \,|\, em \in le, id \in le, i \neq j \\ \forall le \in l \quad &: \quad le_i \neq le_j \,|\, le \in l, i \neq j \end{aligned} \tag{3.15}$$

3.4. Anwendbare Fachmodelle

3.4.1. Anwendbare Meta-Datenstrukturen

Aus der Definition des Elementarmodells kann nicht direkt geschlussfolgert werden, welche konkreten Baufachmodell-Typen in Multimodellen eingesetzt werden können. Die Zielsetzung des Multimodell-Ansatzes, möglichst viele Fachmodell-Typen beteiligen zu können, macht dahingehend keine Einschränkungen. Die erfolgreiche Beteiligung eines Baufachmodell-Typs ist ausschließlich von der effektiven Überführung in die Datenstruktur des Ideellen Elementarmodells abhängig. Dies setzt eine softwaretechnische Implementierung im Einzelfall voraus. Eine vollständige und abschließende Auflistung konkreter anwendbarer Fachmodell-Typen ist somit nicht möglich.

Um dennoch intuitiv entscheiden zu können, ob ein Fachmodell-Typ multimodellfähig ist, kann erörtert werden, ob er zu einer Gruppe überführbarer Typen gehört. Eine sinnvolle Gruppierung ist die Einordnung in Meta-Datenstrukturen, also die übergeordneten Vorschriften und Modellierungskonzepte mit denen der Fachmodell-Typ definiert wurde[17]. Können Regeln zur Überführung in die ideelle Elementarmodell-Datenstruktur auf Ebene einer Meta-Datenstruktur aufgestellt werden, dann sind alle Fachmodell-Typen, die sich von dieser Meta-Datenstruktur ableiten, multimodellfähig. Die Überprüfung jedes einzelnen Fachmodell-Typs wird dadurch überflüssig.

Tabelle 3.4 auf der nachfolgenden Seite zeigt mögliche Überführungen baufachlich relevanter Meta-Datenstrukturen in die Datenstruktur des Ideellen Elementarmodells. Für inhärent objekt- bzw. klassenorientierte Meta-Datenstrukturen (XML, Express, MOF) können Komplexeinheiten

[17] Einen generellen Überblick über übliche Daten-Serialisierungsformate und deren Modellierungskonzepte gibt die Wikipedia-Seite `http://en.wikipedia.org/wiki/Comparison_of_data_serialization_formats`.

(bspw. Klassen, Entitäten, Typen) auf Multimodell-Elemente abgebildet werden. Deren atomare Merkmale (bspw. Feld, Attribut) werden auf Properties abgebildet. Analog können auch Relationale Datenbanken behandelt werden.

Beim Format *Comma Separated Value* (CSV) können die Zellen als Elemente aufgefasst werden – ihre ID bestimmt sich dabei über Zeilen- und Spaltennummer. Da eine Zelle keine weiteren untergeordneten Merkmale hat, wird für ihren Inhalt ein stellvertretendes (Proxy) Property *Value* eingeführt. Auf diesem Weg wird die strukturelle Anforderung des Ideellen Elementarmodells nach der Existenz eines Properties erfüllt.

Somit sind alle Fachmodelle in Multimodellen nutzbar, die auf Basis abbildbarer Meta-Datenstrukturen definiert sind und deren Elemente per ID angesprochen werden können. Ausweichlösungen für Fachmodelle ohne explizite IDs wurden bereits in Abschnitt 3.3.4 auf Seite 61 vorgestellt.

Beispielsweise sind die Fachmodelle GAEB-DA-XML (Leistungsverzeichnisse), XPDL (Workflows) oder MS Project (Vorgänge) nutzbar, da sie XML basiert sind. IFC ist als Express-Instanz ebenfalls multimodellfähig sowie Dokumente von Tabellenkalkulationen im CSV-Format. Im Rahmen der prototypischen Umsetzung des universellen Multimodell-Softwaresystems *M2A2* wurden verschiedene Baufachmodell-Typen als Elementarmodelle implementiert; eine Auflistung ist in Anhang C.1 gegeben.

3.4.2. Programmatische Erweiterung

Die Überführung von Fachmodellen in Elementarmodelle ist Voraussetzung zur Teilnahme an Multimodell-Operationen. Sowohl die Überführung an sich, als auch die Multimodell-Operationen benötigen ein implementiertes Softwaresystem, in dem sie ausgeführt werden können. Soll das Multimodell-Softwaresystem universell – also nicht auf festgelegte Elementarmodell-Typen beschränkt sein – muss es über strukturierte Erweiterungsmöglichkeiten für zusätzliche Elementarmodell-Typen verfügen (Fuchs et al., 2010). Da einige Operationskomponenten wie Elementarmodell-Parser und -Viewer Kenntnisse über Struktur und Semantik der Elementarmodelle benötigen, sind auch für solche Bestandteile Erweiterungspunkte vorzusehen.

Abbildung 3.10 auf der nachfolgenden Seite zeigt die relevanten Erweiterungspunkte für ausbaubare Multimodell-Softwaresysteme[18]. Diese besitzen Abhängigkeiten untereinander, wodurch

Tabelle 3.4.: Mögliche Überführung relevanter Meta-Datenstrukturen in die ideelle Elementarmodell-Datenstruktur

Meta-Datenstruktur	Elementarmodell	Multimodell-Element	Property
RDB, ERM	Datenbank	Tabelle	Spalte
XML	Dokument	XML-Element	Textknoten, Attribut
Express	Dokument	Entität	Attribut
CSV	Dokument	Zelle	Proxy-Property *Value*
INI	Dokument	Sektion	Schlüssel
JSON	Dokument	Objekt	Eigenschaft
MOF (XMI, HUTN)	Modell	Klasse	Attribut

[18] Eine detaillierte Beschreibung der Erweiterungsfunktionalität wird in den Abschnitten 5.1.3 und 6.1.2 gegeben.

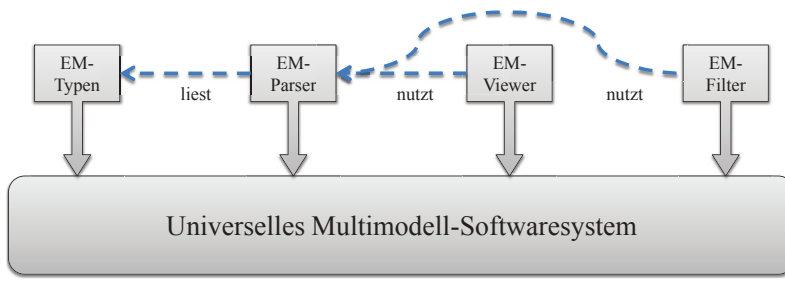


Abbildung 3.10.: Relevante Elementarmodell-Erweiterungspunkte universeller Multimodell-Softwaresysteme und ihre Abhängigkeiten untereinander

konkrete Erweiterungen für andere Komponenten wiederverwendbar sind. Die Pfeilrichtung gibt die Abhängigkeitsrichtung an. Das bedeutet, dass bspw. ein Elementarmodell-Filter seinen benötigten Parser kennen muss – umgekehrt weiß ein Parser nicht, welche Filter ihn verwenden. Alle Abhängigkeiten haben eine Kardinalität von $n : m$. So können unterschiedliche Parser für den selben Elementarmodell-Typ existieren, was beispielsweise die aktuelle Situation im Open Source-Bereich von IFC widerspiegelt[19]. Umgekehrt können mehrere Elementarmodell-Typen vom selben Parser gelesen werden, z. B. alle XML-basierten Fachmodelle von einem generischen XML-Parser.

Elementarmodell-Parser sind ein wichtiger Bestandteil erweiterbarer Multimodell-Softwaresysteme. Sie ermöglichen den Übergang von einer externalisierten Datenstruktur in genau *ein* korrespondierendes internes Datenmodell im Hauptspeicher. Dieses interne Datenmodell ist Voraussetzung für jegliche weiterführende Informationsverarbeitung wie Visualisierung oder Filtern. Die interne Bereitstellung eines geparsten Elementarmodells erlaubt die Ausführung beliebiger Fachlogik auf diesen Daten, da diese mit den Mitteln der implementierenden Programmiersprache konsumiert werden können. Eine hohe Qualität des internen Datenmodells sowie möglicherweise vorhandene Hilfsfunktionen für Zugriff und Integrität können die Umsetzung von Fachfunktionen erleichtern. Beispielsweise beschreiben Steinberg et al. (2009) die Anwendung des Eclipse Modeling Frameworks (EMF), einem Werkzeug zur Generierung von Datenmodell, Parser und Hilfsfunktionen für Java und andere Programmiersprachen auf Basis von Metamodellen.

Der Vorteil programmatischer Erweiterungen ist die hohe Flexibilität von universellen Programmiersprachen. Einem Multimodell-Softwaresystem können so nahezu uneingeschränkte semantische Fachfunktionalitäten hinzugefügt werden. Nachteile sind die dafür erforderlichen Programmierkenntnisse, unabhängig vom tatsächlichen Erweiterungsumfang. Außerdem muss die Erweiterung oder das gesamte Softwaresystem neu kompiliert und an die Benutzer verteilt werden. Die generelle programmatische Erweiterungsfähigkeit eines Multimodell-Softwaresystems um neue Elementarmodell-Typen ist im Wesentlichen von der Verfügbarkeit geeigneter Parser abhängig.

[19] Zur Zeit übliche IFC-Open Source-Parser für Java sind z. B. BIMServer (`http://www.bimserver.org`), Open IFC Tools (`http://www.openifctools.org`) sowie der Express-Parsergenerator JSDAI (`http://www.jsdai.net`).

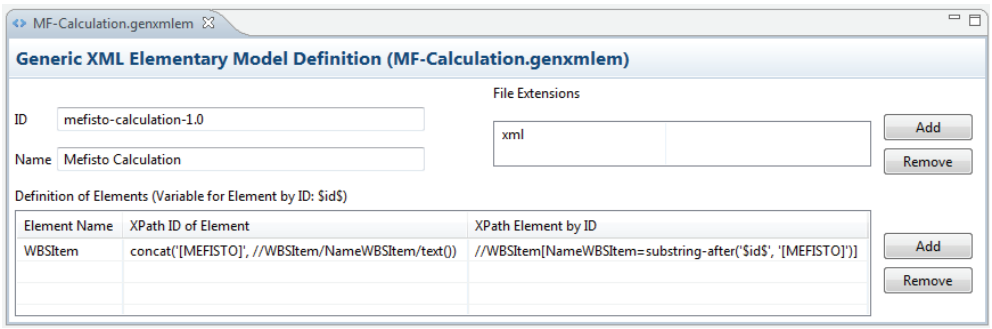


Abbildung 3.11.: Deklaration des Elementarmodell-Typs 'Mefisto Calculation'

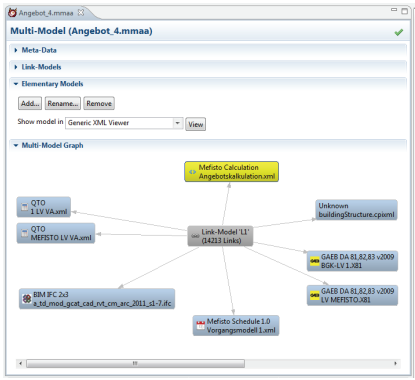
3.4.3. Deklarative, generische XML-Erweiterung

Das Hinzufügen neuer Elementarmodell-Typen soll auch auf Nutzerebene möglich sein – ohne Programmieraufwand und ohne Neukompilierung des universellen Multimodell-Softwaresystems. Dafür müssen zum einen die Informationen über den konkreten Elementarmodell-Typ zur Laufzeit verfügbar gemacht werden. Zum anderen muss es allgemeingültige Komponenten wie Parser oder Viewer geben, welche die korrespondierenden Elementarmodell-Instanzen verarbeiten können. Für XML-basierte Fachmodelle wurde diese Systematik exemplarisch umgesetzt. Die XML-Struktur ist formalisiert (W3C XML 1.0, 2008) und Softwarebibliotheken zum Parsen und Filtern sind für die meisten Programmiersprachen frei verfügbar. Ein Großteil der Fachmodelle im Bauwesen ist XML-basiert. Ihnen wird damit die potentielle Nutzung in Multimodellen ermöglicht. Die Systematik ist auch auf andere Meta-Datenstrukturen anwendbar, bspw. Express oder JSON.

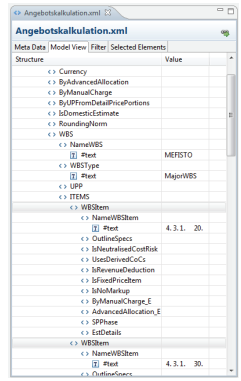
Abbildung 3.11 zeigt den Editor zum Deklarieren von XML-basierten Elementarmodell-Typen. In diesem Beispiel wird *Mefisto Calculation*, ein Modell für Kostenansätze und -kalkulation, bearbeitet. In der oberen Hälfte werden Angaben zu ID und Name des Elementarmodell-Typs sowie möglichen Endungen der Dateinamen eingegeben. In der unteren Hälfte werden die Element-Typen festgelegt. Nur diese können später verlinkt und abgefragt werden. Dazu benötigen die Elemente einen Namen (Spalte 1) sowie Angaben zum Finden der ID eines Elements (Spalte 2) und zum Finden eines Elements ausgehend von seiner ID (Spalte 3). Diese Filter werden mittels XPath-Ausdrücken (W3C XPath 1.0, 1999) formuliert.

Die vollständige Deklaration kann als Datei gespeichert und mit anderen Nutzern ausgetauscht werden. Durch Angabe dieser Datei in der Konfiguration der Multimodell-Anwendung werden zukünftig Elementarmodelle des deklarierten Typs erkannt (vgl. Abbildung 3.12a) und können in einem generischen XML-Viewer dargestellt werden (vgl. Abbildung 3.12b). Die Teilnahme an ID-basierten Operationen wie Verlinkung und Multimodell-Filtern wird durch die dynamische Auswertung der XPath-Ausdrücke ebenfalls gewährleistet.

Die deklarative, generische XML-Erweiterung ist besonders für selten genutzte oder prototypische Fachmodelle empfehlenswert. Die Visualisierung über den baumartigen XML-Viewer bietet keine semantisch aufbereiteten Fachinformationen. Dafür ist die Darstellungsmethode jedoch robust gegenüber strukturellen Änderungen im Fachmodell-Schema.



(a) Erkennen eines Elementarmodells vom Typ 'Mefisto Calculation' (Oberer Knoten)



(b) Anzeige des Modellinhalts im generischem XML-Viewer

Abbildung 3.12.: Nutzung instanziierter Elementarmodelle des Typs 'Mefisto Calculation'

3.4.4. Integration von Dokumenten

Das Multimodellkonzept unterstützt neben der Integration von Fachmodellen auch die Einbeziehung von beliebigen Dokumenten wie Bildern, Texten oder Binärdateien. Dokumente sind in diesem Zusammenhang unstrukturierte Daten, deren Informationsgehalt im Kontext der Multimodellanwendung unbekannt oder irrelevant ist. Im Sinne der Informationsverarbeitung handelt es sich bei Dokumenten um Deskriptionenträger, also nicht voll formalisierte Informationen (vgl. Abbildung 2.2 auf Seite 12). Aus Dokumenten lassen sich keine direkten Elementarmodelle bilden, da ihre Semantik nicht vereinbart wurde. Für die Multimodellapplikation stellen Dokumente die kleinste erschließbare Struktureinheit dar. Ein Anwender kann sich deren Information nur über eine korrespondierende dritte Softwareanwendung zugänglich machen.

Die konsequente Anwendung des Multimodellprinzips erlaubt lediglich eine Verlinkung von Elementen aus Elementarmodellen. Um Dokumenten die Teilnahme in Multimodellen zu ermöglichen, muss daher ein Elementarmodelltyp geschaffen werden, welcher Dokumente kapseln kann. In Anhang A.2 wird das Fachmodell *Dokumentencontainer* in Form eines XML-Schemas präsentiert. Dort wird ein Type `Document` deklariert, der eine ID, eine URI auf die eigentliche Dokumentenressource sowie den MIME-ContentType des Dokuments als Metadatum hält. Das Fachmodell kann eine Kollektion von `Document`-Elementen aufnehmen. Diese Container-Struktur ermöglicht die Gruppierung der Dokumentenobjekte in mehrere Fachmodelle. Die Überführung des Fachmodells in ein Elementarmodell erfolgt wie in Abschnitt 3.4.1 auf Seite 67 beschrieben. Dabei entspricht der `Document`-Type einem Multimodell-Element, seine Attribute MIME-Type und URI werden zu gleichnamigen Properties zugeordnet. Auch das ID-Attribut kann direkt als Element-ID verwendet werden und erlaubt somit die Verlinkung von Dokument-Elementen mit beliebigen anderen Fachmodell-Elementen. Abbildung 3.13 auf der nachfolgenden Seite zeigt eine Beispielinstanz des Dokumentencontainers im entsprechenden Elementarmodell-Viewer.

Der offensichtliche Anwendungsfall für die Verlinkung von Dokumenten mit Fachmodellelemen-

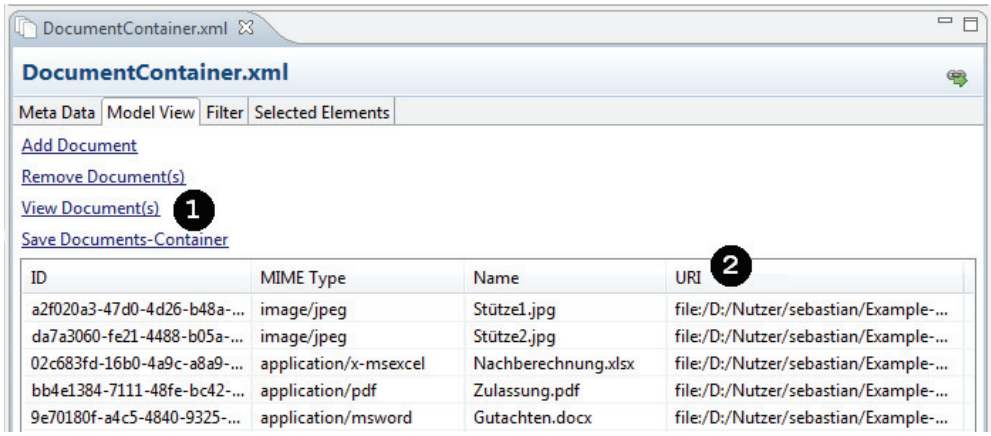


Abbildung 3.13.: Elemetarmodell-Viewer für Dokumentencontainer mit geöffneter Beispielinstanz. Das Kommando *View Document(s)* ① zeigt die selektierten Dokument-Elemente in ihrer zugeordneten Softwareanwendung an. Dazu wird die eigentliche Dokument-Ressource, beschrieben durch ihre URI ②, zum Öffnen an das Betriebssystem delegiert.

ten ist die Anreicherung des Fachmodells mit multimedialen Informationen zur Auswertung durch einen menschlichen Nutzer. Es sind aber auch darüber hinausgehende Szenarien denkbar, bei denen Dokumente die führende Dateneinheit darstellen – beispielsweise bei der Analyse von Textdokumenten (Schapke & Scherer, 2008).

Wann eine Informationsressource als Dokument oder als Fachmodell behandelt wird, ist vom Kontext und dem Einsatzgebiet des Multimodells abhängig. Die Vereinbarung einer Semantik ist nämlich nicht prinzipiell unmöglich. So kann eine PDF-Datei in einem Szenario ein Dokument sein – in einem anderen Fall können PDF-Dateien als Elementarmodelle behandelt werden, deren Elemente beispielsweise Kapitel, Absätze und Bilder sind.

3.5. Multimodell-Spezialisierung

3.5.1. Überblick

Das Generische Multimodell bildet die Systematik für allgemeine Multimodelle ohne Beschränkung auf ein Anwendungsgebiet. Die Fähigkeit zum Einsatz von Fachmodellen aus beliebigen Domänen ermöglicht eine universelle Verwendung von Multimodellen – auch außerhalb des AEC-Sektors. Für das Bauwesen eröffnet sich dadurch eine hohe Vielfalt potentieller Einsatzbereiche. Sogar bislang unbekannte interdisziplinäre Bauinformationsprozesse können durch diesen Ansatz unterstützt werden. Aufgrund seiner hohen Allgemeingültigkeit kann das Generische Multimodell jedoch nicht von selbst die fachliche Semantik der Anwendungsdomäne reflektieren. Zwar können den Multimodellkomponenten domänenspezifische Informationen in Form von Metadaten hinzugefügt werden, deren Inhalt und Bedeutung müssen jedoch – insbesondere zum Zweck des Datenaustauschs – definiert werden.

Der Austausch von Multimodellen in einem Multimodellprojekt erfordert demnach die Abgren-

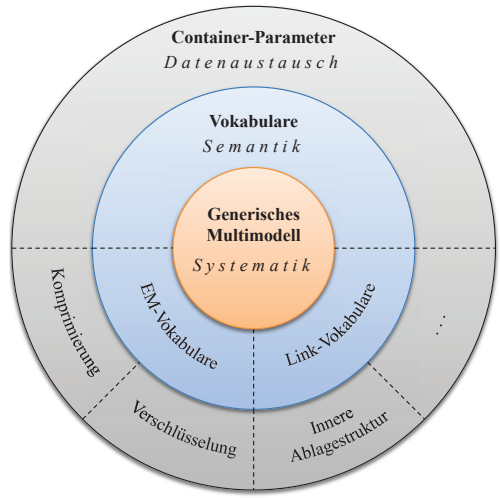


Abbildung 3.14.: Metaebenen der Multimodell-Spezialisierung

zung und Beschreibung des zugehörigen abgeschlossenen Informationsraumes. Dies geschieht durch den Einsatz kontrollierter Vokabulare der Multimodellprojekt-Domäne. In diesem Zusammenhang wird ein Multimodellprojekt durch diejenige Klasse definiert, zu der alle dort auftretenden Bauinformationsprozesse gehören. Außerdem müssen Festlegungen über das auszutauschende Datenformat – den so genannten Multimodell-Container – getroffen werden.

Der Vorgang zur Konkretisierung von Multimodellinhalt und -austausch wird Multimodell-Spezialisierung genannt. Die oben beschriebenen Konzepte werden in Abbildung 3.14 als übergeordneten Ebenen dargestellt. Dabei steht das Generische Multimodell mit seinen Kernkonzepten *Bündelung von Elementarmodellen* und *ID-basierte externe Links* als Systematik im Ausgangspunkt. Auf der nächsten Ebene erfolgt die Zuordnung der Semantik der Elementar- und Linkmodelle über jeweilige kontrollierte Vokabulare. Auf der letzten Ebene werden Spezifika des Datenaustauschformats definiert, beispielsweise die Speicherorte der Elementar- und Linkmodelle (Innere Ablagestruktur) und mögliche Datenkomprimierungs- oder -verschlüsselungsmechanismen. Es ist sogar möglich die Datenstruktur des Generischen Multimodells semantisch anzupassen oder zu erweitern.

Ein Beispiel einer Multimodell-Spezialisierung für die Domäne des Bauprojektmanagements wird in Abschnitt 6.2 auf Seite 165 beschrieben.

3.5.2. Eingrenzung von Informationsräumen

Aus formaler Sicht ist die Eingrenzung des Informationsraums IR die Bildung einer konkreten Teilmenge MM_{IR} aus allen möglichen Multimodellen.

$$\begin{aligned} \mathbb{MM} &:= \text{Menge aller möglichen Multimodelle} \\ MM_{\mathrm{IR}} &\subset \mathbb{MM} \end{aligned} \tag{3.16}$$

Termini t sind die Elemente eines kontrollierten Vokabulars. Ein kontrolliertes Vokabular $kv_i\,(x)$

für Link- oder Elementarmodelle x ist ein geordnetes Paar aus einem Terminus t_b als Bezeichner und einem Wertevorrat T_x mit eindeutigen Termini. Für jedes Modell können i kontrollierte Vokabulare angelegt werden.

$$\begin{aligned} t &:= \text{Terminus des Informationsraums } IR \\ t &\in \mathbb{T} \\ T_x &:= \{t_1, \ldots, t_n\} \mid t_i \neq t_j, i \neq j \\ kv_i(x) &:= (t_b, T_x) \end{aligned} \tag{3.17}$$

Im Sinne der Multimodell-Spezialisierung wird ein abgeschlossener Informationsraum beschrieben, indem die kontrollierten Vokabulare auf die Eigenschaften $pm_i(x)$ von Elementar- und Linkmodellen x als Vorschrift angewendet werden. Bei einer konkreten Multimodell-Instanz werden die Bezeichner-Termini $\widehat{t_{b_i}}$ durch Metadaten-Schlüssel und die zugehörigen Wertausprägungen $\widehat{t_{v_i}}$ durch den korrespondierenden Metadaten-Wert repräsentiert. Die Modell-Eigenschaften dienen dann als Kriterium zur Klassifikation. Gehört ein Multimodell zum betrachteten Informationsraum, müssen alle Eigenschaftsbelegungen seiner Elementar- und Linkmodelle mit den vorgeschriebenen kontrollierten Vokabularen harmonieren.

$$\begin{aligned} pm_i(x) &:= \left(\widehat{t_{b_i}}, \widehat{t_{v_i}}\right) \\ \forall mm, kv_i(x), x \in mm &: mm \in MM_{\text{IR}} \Leftrightarrow \widehat{t_{b_i}} = t_{b_i} \wedge \widehat{t_{v_i}} \in T_{x_i} \end{aligned} \tag{3.18}$$

3.5.3. Elementarmodell-Vokabulare

Elementarmodell-Vokabulare bestimmen die Klassen von Baufachmodellen die zu einem bestimmten Zweck in Multimodellen verwendet werden sollen. In Anhang B sind die Vokabulare wiedergegeben, welche Schapke & Hilbert (2012a,b) für die Fachmodelle des Informationsraums *Bauprojektmanagement* definieren. Dieser Einsatzzweck besitzt eine genügend hohe Allgemeingültigkeit um einen Großteil relevanter Informationsprozesse im Bauwesen abzubilden. Dementsprechend sind auch die Eigenschaftsaspekte universell:

- *Domain* beschreibt den Fachbereich des Elementarmodells
- *Phase* beschreibt die Projektphase, in der das Elementarmodell verwendet wird
- *Level of Detail* beschreibt den Detaillierungsgrad des Elementarmodells
- *Status* beschreibt den Bearbeitungsstand des Elementarmodells

Diese Elementarmodell-Vokabulare sind in einer Vielzahl *Bauprojektmanagement*-verwandter Multimodell-Anwendungsbereiche nutzbar. Die Wertevorräte können entsprechend des Ziel-Informationsraums angepasst werden. In der Regel sind dabei die Werte für die Domäne der Elementarmodelle zu erweitern; die Werte für Phase und Level-of-Detail jedoch einzuschränken.

Für nicht kompatible Anwendungsbereiche können Eigenschaftsaspekte ausgelassen oder hinzugefügt werden. So ist es denkbar, dass für Informationsräume, in denen Messdaten einen hohen Stellenwert besitzen, ein Vokabular für den Aspekt *Datenqualität* notwendig ist. Der Aspekt *Gültigkeitszeitraum* könnte in Szenarien mit Elementarmodellen für externe Tarife oder Preislisten relevant sein.

3.5.4. Linkmodell-Vokabulare

Die Semantik der Links lässt sich analog über Vokabulare ausdrücken. Dabei ist zu beachten, dass ein Linkmodell eine Komposition einzelner Links ist und dass eine Annotation von Eigenschaftswerten sowohl am Link, als auch am Linkmodell stattfinden kann. Dabei soll gelten, dass eine Eigenschaft des Linkmodells $pm_i(lm)$ für alle zugehörigen Links l gilt. Ist dieselbe Eigenschaft jedoch auch am Link deklariert, so hat der spezielle Eigenschaftswert des einzelnen Links eine höhere Priorität und überschreibt die Vorgabe des Linkmodells.

$$\begin{aligned} pm_i\,(lm) &:= \left(\widehat{t_{b_i}}, \widehat{t_{v_i}}\right) \\ pm_i\,(l) &:= \left(\widetilde{t_{b_i}}, \widetilde{t_{v_i}}\right) \\ \forall l \in lm \quad &: \quad \widetilde{t_{v_i}} = \begin{cases} \widetilde{t_{v_i}} & \text{für } \widetilde{t_{b_i}} \neq \emptyset \\ \widehat{t_{v_i}} & \text{für } \widehat{t_{b_i}} \neq \emptyset \\ \emptyset & \text{sonst} \end{cases} \end{aligned} \tag{3.19}$$

Multimodell-Links sind Relationen und können daher grundlegend auch mit deren Eigenschaften (bspw. Richtung, Symmetrie, Eindeutigkeit) beschrieben werden. Dabei wurden in Abschnitt 3.3.5 auf Seite 63 für Arität und Kardinalität sowie für einige strukturelle Beziehungen bereits globale Einschränkungen getroffen, welche in keinem Fall verletzt werden dürfen. Die Eigenschaften auf Relationenebene haben mathematischen Charakter und sind in ihrer Kombination zum Zweck einer Multimodell-Spezialisierung nur schwer formulierbar. Kadolsky schlägt daher eine Notation auf Basis der Backus-Naur-Form vor (Fuchs et al., 2011; ISO 14977, 1996). Dennoch eignen sich diese elementaren Eigenschaften eher zum Aufstellen konkreter Constraints als zur Bildung eines kontrollierten Vokabulars.

Linkmodell-Vokabulare können mithilfe der Definition von Prädikaten für die Linkbedeutung festgelegt werden. Es existieren zwei inhärente Prädikate: *Dokumentlink* und *Identitätslink*. Ein Dokumentlink weist einem Multimodell-Element ein oder mehrere Dokumente zu[20]. Ein *Identitätslink* verlinkt Multimodell-Elemente, welche identische Realweltobjekte repräsentieren; beispielsweise ein IfcBuilding und ein Building in CityGML. Weitere Prädikate sind abhängig vom Ziel-Informationsraum und können zum Beispiel sein: *führt aus*, *ist verantwortlich* oder *beliefert*. Prädikate können eine Richtung implizieren. Diese kann mittels Relationseigenschaften beschrieben werden, sofern sich die Leserichtung nicht aus dem Kontext der Elementarmodelle erkennen lässt. In der Regel ist eine Richtungsumkehr durch Umkehr der Diathese baufachlich korrekt: *wird ausgeführt*, *wird verantwortet*, *wird beliefert*. Daher sollte eine Relationsrichtung nur dann vorgeschrieben werden, wenn eine Richtungsumkehr fachlich unmöglich ist.

Scherer & Schapke (2011) beschreiben die Eigenschaften von multimodellbasierten Informationsräumen in Hinblick auf die wechselseitigen Abhängigkeiten der einzelnen Fachmodelle in Bauprojekten. Abbildung 3.15 auf der nachfolgenden Seite zeigt die drei identifizierten Dimensionen des Multimodell-Informationsraumes:

- Horizontale Interdependenzen für Fachmodelle unterschiedlicher Domänen
- Vertikale Interdependenzen für Fachmodelle mit unterschiedlichem Detaillierungsgrad
- Longitudinale Interdependenzen für Fachmodelle mit unterschiedlichen Versionen

[20] vgl. Abschnitt 3.4.4 auf Seite 71

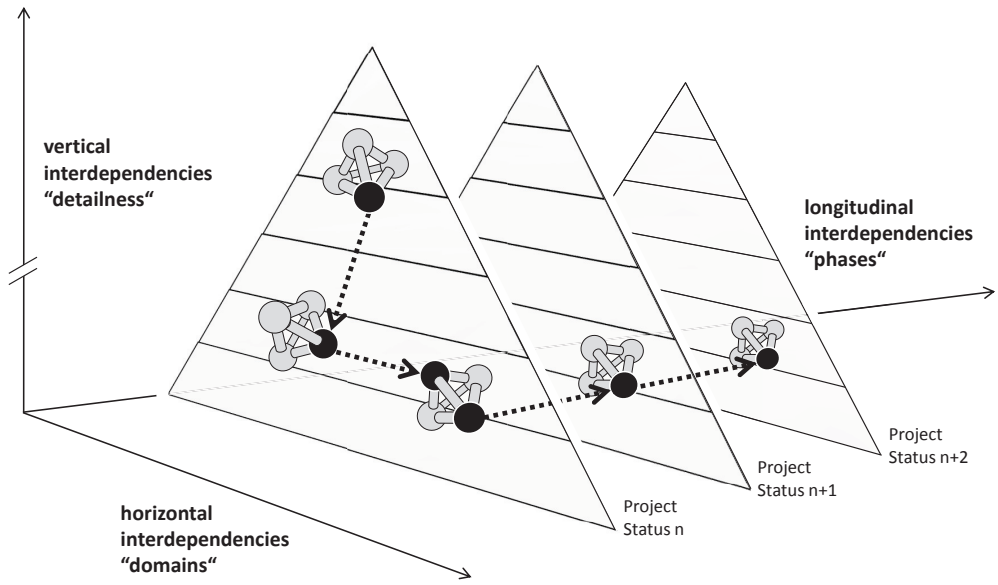


Abbildung 3.15.: Interdependenzen der Fachmodelle in Bauprojekten (aus Scherer & Schapke, 2011)

Die genannten Interdependenzen werden in Multimodellen durch Linkmodelle ausgedrückt. Dementsprechend kann ein korrespondierendes Vokabular für den Eigenschaftsaspekt *Interdependenz* angelegt werden. So drückt der Wert *Detail* beispielsweise aus, dass ein Link das gleiche Element in Elementarmodellen mit unterschiedlichem Level of Detail (LoD) verbindet. In diesem Fall handelt es sich gleichzeitig um einen Identitätslink. Die Richtung der Detaillierung (Expansion oder Verdichtung) kann aus dem LoD-Vokabular der beteiligten Elementarmodelle abgeleitet werden. Tabelle 3.5 fasst die Basisvokabulare für Linkmodelle und Links zusammen und nennt drei exemplarische Prädikate. Auf diese Basisvokabulare aufbauend, können in Multimodell-Spezialisierungen definierte Linktypen gebildet werden.

Tabelle 3.5.: Basisvokabulare für Linkmodelle und Links

Bezeichner	Wertevorrat
Intent	Document
	Identity
	Predicate: *führt aus*, *ist verantwortlich*, *beliefert*, ⋮
Interdependency	Domain
	Detail
	Version

Weiterführende Literatur

Das Aufstellen von Vokabularen und Constraints zur Multimodellspezialisierung ist vom Wissen über den Ziel-Informationsraum sowie den Techniken zur Bewältigung der komplexen Eigenschaftsbeschreibungen abhängig. Ein allgemeingültiger oder vollständiger Ansatz zur Vokabularerstellung kann an dieser Stelle nicht gegeben werden. Daher wird auf weiterführende Literatur in diesem Kontext verwiesen. Olbrich (1998) beschreibt die Eigenschaften von Relationen und schlägt allgemeine Modellierungsmuster für Relationen vor. Ontologien werden in der Bauinformatik häufig zur Wissensrepräsentation und -verarbeitung eingesetzt. Insbesondere für Relationen wurden von Wand et al. (1999) hierzu Grundlagen erarbeitet. Einen allgemeinen Ansatz im Kontext von Multimodellen stellen Schapke & Fuchs (2011) mit der Baukernontologie vor. Scherer et al. (2012) präsentieren eine Ontologie für den Informationsraum *Energiemanagement*.

Kosovac et al. (2000) präsentieren einen Thesaurus-Service zur Datenintegration. Bubner & Friedrich (2003) schlagen einen Ansatz zur Klassifizierung von Links im Bereich verteilter Bauwerksmodelle vor. Einen Überblick zur Datenklassifikation im Bauwesen gibt Wix (2007). Das International Framework for Dictionary (IFD)[21] ist eine Methode zur Erstellung mehrsprachiger Wörterbücher für den AEC-Sektor und wird innerhalb von buildingSMART International[22] auf Basis der ISO 12006-3 (2007) entwickelt (Bell et al., 2008). Beetz (2009, S. 45 ff.) führt die relevanten Wörterbücher, Klassifikationen und Taxonomien des Bauwesens auf.

3.5.5. Multimodell-Container

Ein Multimodell-Container ist das vereinbarte Datenformat zur Übertragung eines Multimodells in einem Multimodell-Projekt. Die beeinflussbaren Parameter wurden bereits in Abbildung 3.14 genannt. Zusätzlich muss eine Datenstruktur zur Aufnahme der im Generischen Multimodell abgelegten Daten vereinbart werden.

Das Generische Multimodell besitzt ein inhärentes XML-Serialisierungsformat[23] . Dieses kann unverändert in der Container-Struktur verwendet werden. Insbesondere in den frühen Phasen eines Multimodell-Projektes existiert bei allen Beteiligten nur ein geringfügiges Wissen über die benutzten Fachmodelle sowie über Semantik und Mehrwert der verlinkten Informationen. Zudem ist es notwendig, schnell Container-Prototypen zu erstellen, um mit den verlinkten Fachmodellen zu experimentieren. Hier unterstützt der generische, vorgefertigte Ansatz die dynamische Arbeitsweise, da keine weiteren Programmier- oder Konfigurationsarbeiten notwendig werden. Nachteilig ist, dass die Semantik des Informationsraums nur über den internen Metadatenmechanismus abgebildet werden kann.

Wird es im weiteren Projektverlauf erforderlich, diese semantischen Informationen explizit abzubilden, muss das Multimodell-Datenschema angepasst werden. Zum Beispiel können eigene Klassen für die verwendeten Elementarmodell-Typen angelegt, Vokabulare als Attribute abgebildet oder spezielle Metadaten wie Änderungshistorie eingeführt werden. Dazu kann das existierende Schema des Generischen Multimodells angepasst (Adaption) oder im Sinne der Objektorientierung erweitert (Extension) werden. Während Extension als schnelle und einfache Möglichkeit angesehen wird, um eine Multimodell-Spezialisierung vorzunehmen, bietet die

[21] http://www.ifd-library.org/
[22] http://www.buildingsmart-tech.org/
[23] vgl. Anhang A.1 auf Seite 185

Adaption durch den Entwurf eines eigenständigen Multimodell-Datenschemas eine erhöhte Flexibilität.

3.6. Multimodell-Operationen

3.6.1. Elementaroperationen

Der Mehrwert von Multimodell-Software gegenüber traditioneller Software ist die Fähigkeit zur Operation auf Multimodellen. Operationen ändern den Zustand, respektive den Datenbestand, des Multimodells. Dieser Zustand muss vor ($\mathfrak{pre}$) und nach ($\mathfrak{post}$) der Operation konsistent sein. Daher ist es notwendig zu betrachten, welche Operationen für Multimodelle möglich sind und wie diese untereinander in Beziehung stehen.

In Analogie zu Relationalen Datenbanken existieren für alle Dateneinheiten prinzipiell die vier Elementaroperation Erstellen (Create), Lesen (Read), Ändern (Update) und Löschen (Delete), kurz CRUD (Martin, 1983). Tabelle 3.6 zeigt die Elementaroperationen für die fünf Dateneinheiten des Multimodells. Bei Metadaten wird zusätzlich aufgeführt, für welche Dateneinheit diese bestimmt sind. Eine Multimodell-Software muss nicht alle Elementaroperationen beherrschen. Vorrangig kann zwischen nur lesender und schreibender Multimodell-Software unterschieden

Tabelle 3.6.: Notation der Multimodell-Elementaroperationen

Operation / Dateneinheit	Erstellen (Create)	Lesen (Read)	Ändern (Update)	Löschen (Delete)
Multimodell	$\mathfrak{C}(mm)$	$\mathfrak{R}(mm)$	$\mathfrak{U}(mm)$	$\mathfrak{D}(mm)$
Elementarmodell	$\mathfrak{C}(em)$	$\mathfrak{R}(em)$	$\mathfrak{U}(em)$	$\mathfrak{D}(em)$
Linkmodell	$\mathfrak{C}(lm)$	$\mathfrak{R}(lm)$	$\mathfrak{U}(lm)$	$\mathfrak{D}(lm)$
Link	$\mathfrak{C}(l)$	$\mathfrak{R}(l)$	$\mathfrak{U}(l)$	$\mathfrak{D}(l)$
Metadaten $x \in \{mm, em, lm, l\}$	$\mathfrak{C}(md(x))$	$\mathfrak{R}(md(x))$	$\mathfrak{U}(md(x))$	$\mathfrak{D}(md(x))$

werden, wobei davon ausgegangen wird, dass für Schreibfähigkeit auch Lesefähigkeit notwendig ist. Da Lesezugriffe keine Seiteneffekte aufweisen dürfen, sind die Leseoperationen $\mathfrak{R}(x)$ als trivial zu betrachten. Sie werden im Folgenden nicht explizit aufgeführt, auch wenn sie natürlich Bestandteil jener komplexen Operationen sind, welche schlussendlich die Funktionalität einer Multimodell-Software ausmachen.

Die durch die Operationen zu ändernden Dateneinheiten sind im Generischen Multimodell organisiert. Diese Datenstruktur sowie die in Abschnitt 3.3.5 auf Seite 63 beschriebenen Einschränkungen implizieren Abhängigkeiten zwischen den einzelnen Dateneinheiten, sodass die Elementaroperationen selbst nicht immer atomar sein können.

Das Erstellen von Metadaten erfordert die Existenz der zu annotierenden Dateneinheit:

$$\forall \mathfrak{C}\left(md(x)\right), x \in \{mm, em, lm, l\} \quad : \quad \begin{cases} \exists x & \mathfrak{pre} \\ md(x) \text{ ist Metadatum von } x & \mathfrak{post} \end{cases} \tag{3.20}$$

Aus der Kompositionsbeziehung ergibt sich, dass beim Löschen des Kompositums auch dessen Komponenten gelöscht werden müssen.

$$\begin{aligned} \mathfrak{D}(x) &\Rightarrow \mathfrak{D}\left(md_i(x)\right) \mid x \in \{mm, em, lm, l\} \\ \mathfrak{D}(mm) &\Rightarrow \mathfrak{D}(em_i) \wedge \mathfrak{D}(lm_j) \mid em_i \in mm, lm_j \in mm \\ \mathfrak{D}(lm) &\Rightarrow \mathfrak{D}(l_i) \mid l_i \in lm \end{aligned} \tag{3.21}$$

Aus Gleichung 3.10 auf Seite 66 folgt, dass beim Löschen eines Elementarmodells em auch diejenigen Links zu löschen sind, welche Elemente aus em beinhalten.

$$\mathfrak{D}(em) \Rightarrow \mathfrak{D}(l_i) \mid em \in le \wedge le \in l_i \tag{3.22}$$

Aus den Gleichungen 3.10 und 3.14 auf Seite 67 folgt, dass beim Löschen eines Links auch diejenigen verlinkten Elementarmodelle $\widehat{em_i}$ des Linkmodells entfernt werden müssen, welche in den übrigen Links nicht mehr benutzt werden. Dieses Entfernen ist eine Änderungsoperation des Linkmodells.

$$\mathfrak{D}(l) \Rightarrow \mathfrak{U}_{\widehat{EM}}(lm) \mid l \in lm \tag{3.23}$$

Aus Gleichung 3.11 auf Seite 66 folgt, dass beim Ändern eines Linkmodells (Entfernen eines verlinkten Elementarmodells $\widehat{em}$ nach Gleichung 3.23) das gesamte Linkmodell gelöscht werden muss, wenn weniger als zwei verlinkte Elementarmodelle verbleiben würden. Gleiches gilt für das Löschen von Links, falls weniger als ein Link im Linkmodell verbleiben würde.

$$\begin{aligned} \mathfrak{U}_{\widehat{EM}}(lm) &\Rightarrow \mathfrak{D}(lm) \mid \widehat{EM} \in lm \wedge \mathfrak{pre} : \left|\widehat{EM}\right| = 2 \\ \mathfrak{D}(l) &\Rightarrow \mathfrak{D}(lm) \mid l \in L \wedge L \in lm \wedge \mathfrak{pre} : |L| = 1 \end{aligned} \tag{3.24}$$

Änderungsoperationen von Kompositumeinheiten können durch Erstellungs-, Änderungs- oder

Löschoperationen einer Komponenteneinheit begründet sein.

$$\begin{aligned}
\mathfrak{U}(mm) &:= \mathfrak{C}(x) \vee \mathfrak{U}(x) \vee \mathfrak{D}(x) \mid x \in \{em, lm, l, md(mm)\} \\
\mathfrak{U}(lm) &:= \mathfrak{C}(x) \vee \mathfrak{U}(x) \vee \mathfrak{D}(x) \mid x \in \{l, md(lm)\} \\
\mathfrak{U}(em) &:= \mathfrak{C}(md(lm)) \vee \mathfrak{U}(md(lm)) \vee \mathfrak{D}(md(lm)) \vee \text{Änderung Elementarmodell} \\
\mathfrak{U}(l) &:= \mathfrak{C}(md(l)) \vee \mathfrak{U}(md(l)) \vee \mathfrak{D}(md(l)) \vee \text{Änderung Verlinktes Element} \\
\mathfrak{U}(md(x)) &:= \text{Änderung Metadatenschlüssel oder -wert}
\end{aligned} \tag{3.25}$$

3.6.2. Höhere Operationen

Elementaroperationen treten innerhalb einer Multimodell-Software in der Regel nicht alleinstehend, sondern zusammengefasst als höhere Operationen in Erscheinung. Während atomare und verkettete Elementaroperationen lediglich die strukturelle Konsistenz von Multimodellen sicherstellen müssen, gewährleisten höhere Operationen darüber hinaus auch die fachliche Konsistenz von Links. Dabei werden unter anderem die kontrollierten Vokabulare des Ziel-Informationsraums benutzt.

Elementarmodell ändern Ein vorhandenes Elementarmodell wird durch die Fachanwendung geändert. Vorhandene Links müssen gelöscht oder geändert und neue Links müssen hinzugefügt werden.

$$\mathfrak{Change}(em) \quad \mapsto \quad \mathfrak{U}(em) \wedge \mathfrak{U}(lm) \tag{3.26}$$

Elementarmodell fortschreiben Der Inhalt eines vorhandenes Elementarmodells wird durch die Fachanwendung geändert und in einem zusätzlichen Elementarmodell abgelegt. Es werden ein neues Linkmodell erzeugt (Interdependency = Version) und die Änderungen verlinkt.

$$\mathfrak{Adjust}(em_a) \quad \mapsto \quad \mathfrak{C}(em_b) \wedge \mathfrak{C}(lm) \tag{3.27}$$

Elementarmodell anlegen Ein neues Elementarmodell wird angelegt und dem Multimodell hinzugefügt. Links zu anderen Elementarmodellen werden erzeugt. Dazu werden vorhandene Linkmodelle erweitert und / oder ein neues Linkmodell angelegt.

$$\mathfrak{Create}(em) \quad \mapsto \quad \mathfrak{C}(em) \wedge (\mathfrak{U}(lm_a) \vee \mathfrak{C}(lm_b)) \tag{3.28}$$

Elementarmodell integrieren Die Inhalte von zwei vorhandenen Elementarmodellen gleichen Typs werden in ein gemeinsames neues Elementarmodell überführt. Vorhandene Links müssen angepasst werden.

$$\mathfrak{Integrate}(em_a, em_b) \quad \mapsto \quad \mathfrak{C}(em_c) \wedge \mathfrak{U}(lm) \tag{3.29}$$

Elementarmodell mappen Der Inhalt eines vorhandenen Elementarmodells wird in ein Elementarmodell anderen Formats überführt. Vorhandene Links müssen angepasst werden.

$$\mathfrak{Map}(em_a) \quad \mapsto \quad \mathfrak{C}(em_b) \wedge \mathfrak{U}(lm) \tag{3.30}$$

Elementarmodell verdichten Der Detaillierungsgrad eines vorhandenen Elementarmodells wird verringert und der neue Inhalt in einem zusätzlichen Elementarmodell abgelegt. Es werden ein neues Linkmodell erstellt (Interdependency = Detail) und korrespondierende Elemente verlinkt.

$$\mathfrak{Condense}\,(em_a) \quad \mapsto \quad \mathfrak{C}\,(em_b) \wedge \mathfrak{C}\,(lm) \tag{3.31}$$

Elementarmodell expandieren Der Detaillierungsgrad eines vorhandenen Elementarmodells wird erhöht und der neue Inhalt in einem zusätzlichen Elementarmodell abgelegt. Es werden ein neues Linkmodell erstellt (Interdependency = Detail) und korrespondierende Elemente verlinkt.

$$\mathfrak{Expand}\,(em_a) \quad \mapsto \quad \mathfrak{C}\,(em_b) \wedge \mathfrak{C}\,(lm) \tag{3.32}$$

3.6.3. Anwenderoperationen

Anwenderoperationen sind komplexe Operationen auf Multimodellen mit fachlicher Bedeutung. Sie sind die Module der Bauinformationsprozesse. Einem Endanwender erscheinen sie als Kernfunktionalitäten einer Multimodell-Fachanwendung. Dem Nutzer muss dabei nicht exponiert werden, dass es sich intern um Multimodell-Operationen handelt.

Multimodelle aus Vorlage erstellen Input des Bauinformationsprozesses ist ein Multimodell-Template. Durch Fachwissen des Ingenieurs und unter Hilfestellung der Fachanwendung(en) wird ein Multimodell nach Maßgabe der Anforderung erstellt. Je nach bereits vorhandenen Informationen in Form von Elementarmodellen kommen dabei potentiell alle höheren Operationen – auch in Kombination – zur Anwendung.

Multimodelle filtern Das Filtern verlinkter Elementarmodelle ist die substanzielle Funktionalität zur Erschließung domänenübergreifender Informationsräume. Durch die Auswertung der Links entstehen Multimodell-Views, welche die ursprünglich getrennten Informationen zusammenführen sowie den Informationsumfang auf ein notwendiges Maß reduzieren. Maßgebliche Multimodell-Operationen sind dabei die Leseoperationen $\mathfrak{R}\,(x)$.

Multimodelle visualisieren Die Visualisierung von Multimodellen ist eine wesentliche Funktionalität anwenderorientierter Multimodell-Software. Dabei werden nicht nur die einzelnen Elementarmodelle autonom präsentiert. Vielmehr werden die kombinierten Informationen *so* dargestellt, dass sie wesentlich für die Aufgabenstellung sind und dass die ursprüngliche Trennung der Elementarmodelle und ihrer Domänen nicht mehr wahrnehmbar ist. Grundlage der Multimodell-Visualisierung sind Multimodell-Views.

3.7. Resümee

Das Multimodellkonzept unterstützt den Übergang von der bauwerksorientierten zur prozessorientierten Arbeitsweise. Kernpunkt ist dabei der Paradigmenwechsel bei der Abbildung austauschbarer, interdisziplinärer Daten. Statt eines zentralen, führenden Bauwerksinformationsmodells, beinhaltet ein Multimodell *gleichgestellte* Elementarmodelle. Auf diese Weise können beliebige Bauinformationsprozesse unterstützt werden – auch solche, in denen überhaupt keine Bauwerksinformationsmodelle vorkommen.

Daher ist es auch notwendig, dass das Multimodell – und kein Fachmodell einer bestimmten Domäne – Eigentümer der Links zwischen den Datenelementen unterschiedlicher Elementarmodelle ist. So erlangen die Links eine Unabhängigkeit von jedweden domänenspezifischen Belangen. Das Multimodellkonzept ist daher neutral im Hinblick auf Fachlichkeit *und* Datenformat der Elementarmodelle. Prinzipiell sind Multimodelle deswegen auch außerhalb des Bauwesens anwendbar. Dies ist eine grundlegende Voraussetzung zur Realisierung zukünftiger interdisziplinärer Bauinformationsprozesse, bei denen zur Zeit nicht relevante oder unbekannte Domänen zum Einsatz kommen.

Der offene Anwendungsbereich und die fachliche Neutralität des Konzeptes führen dazu, dass Multimodelle *selbst* keinerlei baufachliche Semantik besitzen. Vielmehr sind sie als ein technisches System zur Informationsrepräsentation zu verstehen – wie relationale Datenbanken oder XML. In Analogie zu Tabellen und Spalten bzw. Elementen und Attributen sind die Hauptkonzepte der Multimodelle: Elementarmodelle im Originalformat, per ID identifizierbare Elemente sowie ID-basierte, mehrwertige Links. Die Notwendigkeit zur Einführung dieses neuartigen Systems ergibt sich aus den Anforderungen der domänenübergreifenden Projektkommunikation. Der baufachliche Nutzwert entsteht – wie bei anderen IKT-Technologien auch – erst durch Anwendung des Systems mit Daten, welche eine baufachliche Semantik besitzen. Dabei hat der Anwender die vollständige Kontrolle, aber auch die vollständige Verantwortung, über die baufachliche Aussage seiner Aktionen.

Die entsprechenden Methoden und Werkzeuge zur Nutzung von Multimodellen werden in dieser Arbeit als *Erschließung domänenübergreifender Informationsräume* diskutiert. Diese arbeiten auf den o. g. informationstechnischen Konzepten der Multimodelle. In Kapitel 4 wird die Multimodell-Abfragesprache MMQL vorgestellt, die es einem Anwender erlaubt, baufachliche Kriterien zur Erzeugung und Abfrage von Multimodellen zu formulieren. Dabei kommt ein generischer, reflexiver Datenzugriff nach Maßgabe des Ideellen Elementarmodells zur Anwendung. In Kapitel 5 wird ein Interpreter präsentiert, der MMQL-Anweisungen in elementare Multimodell-Operationen umsetzt.

Der Übergang vom informationstechnischen System zu konkreten, interdisziplinären Ingenieurlösungen erfolgt innerhalb von *spezifischen*, domänenübergreifenden Baufachanwendungen. Für deren Erstellung und Befähigung zum multimodellbasierten Datenaustausch werden wiederverwendbare Softwarekomponenten angeboten. Diese werden in Abschnitt 6.1 im Rahmen von M2A2 vorgestellt. Dabei handelt es sich um eine erweiterbare, *universelle* Multimodell-Software, die einen Ad-Hoc-Einsatz von Multimodellen in Bauinformationsprozessen ermöglicht.

Das Generische Multimodell erlaubt die Hinterlegung semantischer Informationen als Schlüssel-Wert-Paare in den Metadaten der Elementarmodelle und Links. Mit der Multimodell-Spezialisierung ist es darüber hinaus möglich, komplexe baufachliche Semantik direkt in der Container-Datenstruktur zu modellieren. Dadurch schränkt sich zwar der Anwendungsbereich ein, die baufachliche Bedeutung der Daten gewinnt dafür an Formalisierung. In Abschnitt 6.2 wird eine solche Multimodell-Spezialisierung für die Domäne *Bauprojektmanagement* vorgestellt.

Die durchgehende, baufachliche Anwendung der Multimodell-Methode wird in Abschnitt 6.3 am Beispiel eines Zahlungsplans präsentiert.

Kapitel 4.

Die Multimodell-Abfragesprache MMQL

Die Verschiedenheit der Sprachen ist nicht eine
Verschiedenheit an Schällen und Zeichen,
sondern eine Verschiedenheit der Weltansichten.

(Wilhelm von Humboldt)

Mit dem Generischen Multimodell wurde eine Möglichkeit zur Bildung von datenformat-, datenmodell- und domänenübergreifenden Informationsräumen geschaffen. Um ihren vollen Nutzwert zu entfalten, müssen diese Informationsräume mithilfe von Softwareanwendungen durch den Ingenieur erschlossen werden können. Hierfür wird in diesem Kapitel die textuelle Multimodell-Abfragesprache MMQL vorgestellt. Die Sprache erlaubt die ausdrucksstarke, prägnante Formulierung von Anweisungen für Multimodell-Filter und Linkmanipulationen. Durch ihren deklarativen Charakter entbindet die MMQL den Anwender von der Aufstellung komplexer Strategien zur Operation auf Multimodellen. Auf diese Weise können Fachanwendungen einfacher um Multimodellfähigkeiten erweitert werden.

Dieses Kapitel definiert Syntax und Semantik der MMQL. Abschnitt 4.1 erläutert den Anwendungsbereich und gibt einen kurzen Sprachüberblick. Es wird das Beispiel vorgestellt, an welchem die meisten Sprachfunktionen innerhalb des Kapitels verdeutlicht werden. Abschnitt 4.2 zeigt, wie durch Anwendung des Ideellen Elementarmodells einheitlich auf die heterogenen Originaldaten der Elementarmodelle zugegriffen wird. Die hier vorgestellten Anweisungen sind wichtige Teile des Multimodell-Filterns und der Linkmanipulationen, welche in den Abschnitten 4.3 und 4.4 erläutert werden.

4.1. Konzeption

4.1.1. Anwendungsbereich

In Kapitel 3 wurden das Generische Multimodell, das Ideelle Elementarmodell und ID-basierte Links als Basisstrukturen der Multimodelle beschrieben. Sie erlauben die Erschließung des domänenübergreifenden Informationsraumes auf eine *generische* Art und Weise, da sie selbst keine baufachlichen Konzepte besitzen. Dadurch werden die uneingeschränkte Erweiterbarkeit um neue Elementarmodelltypen sowie die Entwicklung allgemein einsetzbarer Erschließungsmethoden ermöglicht. Grundlegende Multimodell-Funktionalitäten können in einer wiederverwendbaren Kernkomponente, der Multimodell-Engine (MM-Engine), abgelegt werden. Die Erweiterung der MM-Engine um neue Elementarmodell-Komponenten erfolgt durch Erweiterungsentwickler, welche Experten sowohl für den spezifischen Fachmodelltyp, als auch für das Multimodell-Konzept sind. Die MM-Engine stellt u. a. folgende Dienste zur Verfügung:

- Lesen von Multimodell-Containern
- Zugriff auf Elementarmodell-Originaldaten
- Zugriff auf Links
- Multimodell-Elementaroperation[1]

Als Folge der Allgemeingültigkeit fällt jedoch die baufachliche Semantik der Links in den Verantwortungsbereich des Anwenders. Anwender können hier Ingenieure als ambitionierte Endnutzer einer expliziten Multimodell-Software sein – oder Hersteller von domänenübergreifender Fachsoftware, welche die MM-Engine zur technischen Realisierung der Multimodellfähigkeit verwenden (vgl. Abbildung 4.1 auf der nachfolgenden Seite).

Zwar kann die MM-Engine Basisoperationen bereitstellen – die baufachliche Erschließung des Multimodell-Informationsraumes, bspw. durch Auswertung der Links, erfordert jedoch deren Kombination zu Verbundoperationen. Die Formulierung solcher konkreten Operationenabfolgen ist potentiell komplex. Dies hat folgende Ursachen:

1. Auf Originaldaten muss über die beteiligten Elementarmodell-Datenstrukturen zugegriffen werden.
2. Links müssen in den Strukturen des Generischen Multimodells konsistent gesetzt und fachlich korrekt ausgewertet werden.
3. Fachliche Klauseln zum Filtern von Elementarmodellen sowie die Einschränkungen von Links können beliebig kompliziert sein.

Zur Vereinfachung der Steuerung der MM-Engine kommt daher eine speziell entwickelte Programmiersprache, die *Multi-Model Query Language* (MMQL) zum Einsatz. Mithilfe der MMQL können Multimodelle gefiltert und manipuliert werden. Die Semantik der Sprache ist auf der Ebene von Generischem Multimodell und Ideellem Elementarmodell festgelegt. Im Sinne der Kategorisierung von Abschnitt 2.2.3 ist die MMQL – bezogen auf das Multimodell – eine auf Meta-Modellebene definierte Sprache. Sie ist dadurch für alle gültigen Multimodelle universell anwendbar[2].

[1] vgl. Abschnitt 3.6.1 auf Seite 78

[2] Eine Multimodell-Abfragesprache auf Schemaebene wäre für eine konkrete Multimodell-Spezialisierung anwendbar und besäße Sprachkonstrukte und - semantik für die festgelegten Elementarmodell- und Linkmodell-Vokabulare. Eine semantische Multimodell-Filtersprache wäre für einen speziellen domänenübergreifenden Aufgabenbereich wie die Verlinkung von GAEB-Leistungsverzeichnissen mit IFC-Bauwerksmodellen geschaffen.

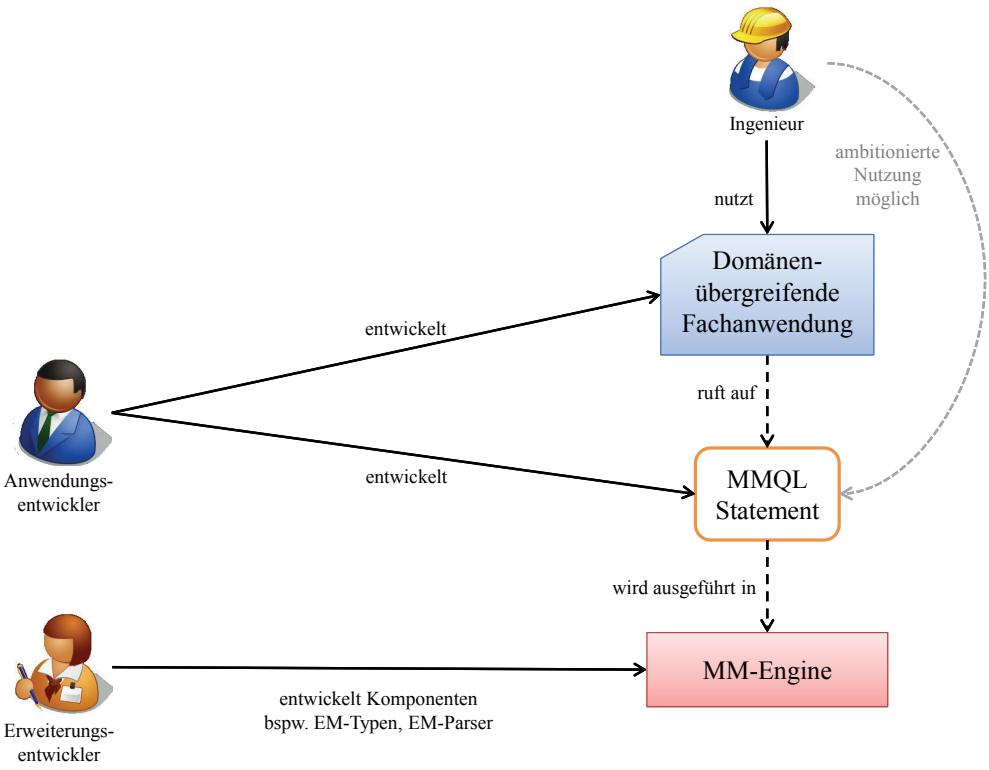


Abbildung 4.1.: Akteure einer MMQL-Anwendung

Das Sprachparadigma der MMQL ist mengenorientiert deklarativ. Ein Anwender beschreibt das gewünschte Resultat – der zugehörige Lösungsweg wird automatisch von der Ablaufumgebung ermittelt. Die Erstellung von Multimodell-Views oder Verlinkungszuständen wird somit vereinfacht; anstelle komplexer Logik brauchen nur noch die Kriterien für Linkbeziehungen und Elementarmodell-Filter definiert zu werden. Auf diese Weise kann die Aufstellung *ausführbarer* fachlicher Regeln für domänenübergreifende Aufgabenstellungen von technischen Sachverhalten getrennt werden.

Die MMQL besitzt eine textuelle Syntax, welche ausdrucksstärker als eine grafische Oberfläche, das sequentielle Aufrufen einzelner Programmfunktionen (API) oder eine grafische Syntax ist. Außerdem wird damit der direkte Einsatz als Server-Komponente vereinfacht. Syntax und Semantik der MMQL sind weitestgehend an SQL (ISO 9075-1, 2008) angelehnt, um eine geringe Einarbeitungszeit und hohe Nutzerakzeptanz zu erzielen.

Die Erzeugung von Multimodell-Views geschieht durch die MM-Engine auf Basis eines tabellarischen ResultSets. Damit wird der domänenübergreifende Informationsraum ähnlich wie eine Relationale Datenbank erschließbar. Tabellarische Daten lassen sich vielfältig aufbereiten, sodass sie nach Maßgabe der Fachanwendung z. B. in Simulationen oder zu einem neuen gefilterten Multimodell weiterverarbeitet werden können. Die Präsentation des ResultSets kann in der ursprünglichen Tabelle oder mithilfe weiterer Visualisierungen geschehen (vgl. Abbildung 4.2 auf der nächsten Seite). Zur Erzeugung von Multimodell-Views besitzt die

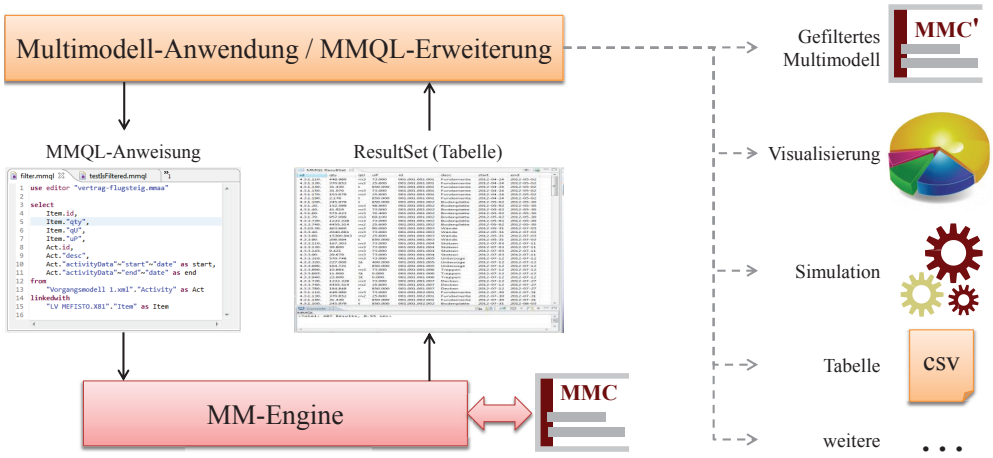


Abbildung 4.2.: Tabellarisches ResultSet als Multimodell-View. Erzeugung durch Multimodell-Filteranweisungen, weitere Darstellung und Verwendung nach Maßgabe der Fachanwendung.

MMQL Spracheigenschaften, welche an die Erfordernisse tabellarischer Strukturen angepasst sind[3].

4.1.2. Sprachüberblick

Die Aufgabe der MMQL ist die Formulierung von Anweisungen zur Erschließung des domänenübergreifenden Informationsraumes eines Multimodells. Dies beinhaltet die Definition von Multimodell-Filtern sowie Anweisungen zur Manipulation von Links und Linkmodellen. Die formale Beschreibung der vollständigen Syntax der MMQL ist in Anhang A.3 auf Seite 188 wiedergegeben. Die relevanten Sprachkonzepte werden in diesem Kapitel zur besseren Lesbarkeit in Syntaxdiagramme überführt. Die Namen der Produktionsregeln und Nonterminale bleiben dabei erhalten.

Elemente sind die signifikanten Einheiten im Multimodellkonzept. Ausschließlich *Elemente* sind als Träger einer ID verlinkbar. Daher zielen die MMQL-Anweisungen für Multimodell-Filter und Verlinkung im Kern auf die *Identifikation* der korrekten Untermenge von Multimodell-Elementen $\acute{E}$ für beabsichtigte Lese- oder Referenzierungsoperationen. Die Lesart solcher MMQL-Anweisungen ist deswegen – aus der Sichtweise eines einzelnes Elementarmodells em_1 – wie folgt:

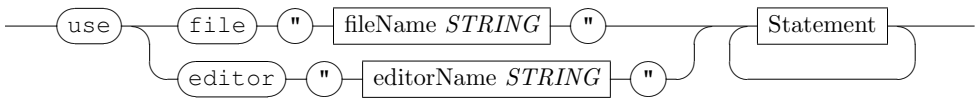
- Wähle diejenigen Elemente $\acute{E}_1$ des Typs et_1 aus allen Elementen E_1 des Elementarmodells em_1 aus, für die ein Kriterium k gilt. Verlinke sie mit anderen Elementen $E_2 \ldots E_n$ der Elementarmodelle $em_2 \ldots em_n$.
- Wähle diejenigen Elemente $\acute{E}_1$ des Typs et_1 aus allen Elementen E_1 des Elementarmodells em_1 aus, welche mit anderen Elementen $E_2 \ldots E_n$ der Elementarmodelle $em_2 \ldots em_n$ verlinkt sind und für die ein Kriterium k gilt. Präsentiere die Values der angegebenen Properties $\acute{P}$ jedes Elements $\acute{e} \in \acute{E}_1$ im Ergebnis.

[3] vgl. Abschnitt 4.3 auf Seite 102

- Das Kriterium k kann sowohl eine Einschränkung innerhalb *eines* Elementarmodells (Elementarmodell-Filter) als auch *elementarmodellübergreifend* sein – z. B. wenn die Dicke eines Bauelements (em_1) kleiner gleich der Dicke innerhalb der detaillierten Leistungsbeschreibung einer LV-Position (em_2) sein soll.

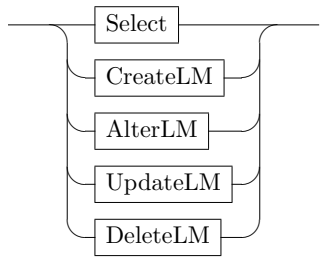
Ein MMQL-Anweisungsblock gilt nur für ein bestimmtes Multimodell. Operationen über mehrere Multimodelle hinweg werden nicht unterstützt. Ein MMQL-Anweisungsblock besteht demnach aus der Deklaration des zu benutzenden Multimodells (Befehl **use**) sowie weiteren Anweisungen (Statements). Ein Multimodell kann über seinen Dateinamen (Schlüsselwort `file`) oder – bspw. im Falle ungespeicherter Multimodelle in Anwendungen mit grafischer Oberfläche – über den Namen des geöffneten Editors (Schlüsselwort `editor`) deklariert werden:

Syntaxdiagramm 4.1: Mmql



Gültige weitere Anweisungen sind Abfrage-Statements (`Select`) sowie die Manipulationsanweisungen zum Erstellen (`CreateLM`), strukturellen Ändern (`AlterLM`) und Löschen (`DeleteLM`) von Linkmodellen sowie deren Links (`UpdateLM`):

Syntaxdiagramm 4.2: Statement



Beispiel 1

Aus einem gegebenen Multimodell sollen diejenigen Wände ermittelt werden, welche die Expositionsklasse *XC1* besitzen und vor dem 01.03.2013 fertiggestellt werden sollen. Außerdem sollen sie sich in der Menge der in einem 3D-Viewer selektierten Bauelemente befinden. Das Multimodell (`"ausschreibung_fr.mmaa"`) besteht aus drei Elementarmodellen (EM) und einem Linkmodell (LM):

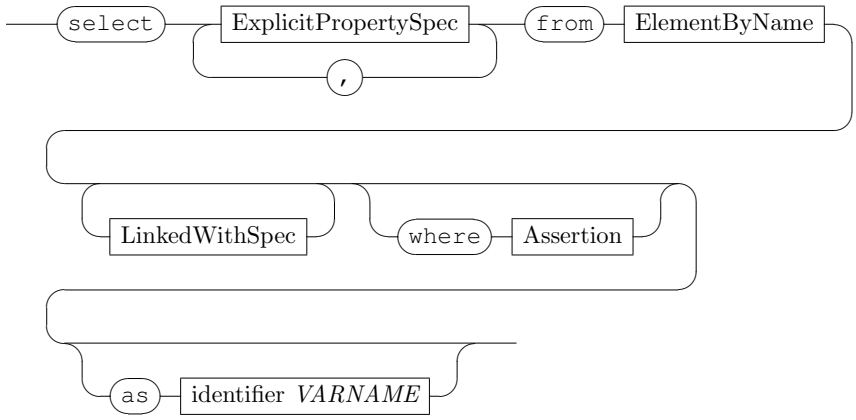
- $EM1$: ein Leistungsverzeichnis (LV) in GAEB-DA-XML (`"LV 1.X83"`); enthält die Informationen zur Expositionsklasse
- $EM2$: ein Bauwerksmodell in IFC (`"BW.ifc"`); enthält die Wände als Bauteile

- $EM3$: ein Vorgangsplan in einem XML-Format (`"Vorgangsmodell 1.xml "`); enthält die Endtermine der Bauvorgänge
- $LM1$: ein Linkmodell (`"LM1"`); verlinkt u. a. LV-Positionen aus $EM1$, Wände aus $EM2$ und Vorgänge aus $EM3$

Quelltext 4.1 auf der nachfolgenden Seite zeigt den MMQL-Anweisungsblock des Multimodell-Filters zur Lösung von Beispiel 1. In Zeile 1 wird durch den Befehl **use** festgelegt, dass das Multimodell mit dem Dateipfad `"C:\Container\ausschreibung_fr.mmaa"` verwendet werden soll.

Ab Zeile 3 folgt ein einzelnes **select**-Statement, für das folgende Syntaxregel gilt:

Syntaxdiagramm 4.3: Select



In Zeile 4 ff. werden die Properties spezifiziert, deren Werte in den Spalten des ResultSets erscheinen sollen (`ExplicitPropertySpec`): die ID der LV-Position, die ID der Wand sowie der Endtermin des Vorgangs. Diese Angabe entspricht der Projektion der relationalen Algebra. Zur einfacheren Wiederverwendung in den nachfolgenden Kriterien werden Alias vergeben (**as**-Klausel).

Die Zeilen 7–12 geben die Elemente an, deren Links ausgewertet werden sollen (Klauseln **from** und **linkedwith**). Dies sind Positionen aus dem Leistungsverzeichnis (`"LV 1.X83"."Item"`), Wände aus dem Bauwerksmodell (`"BW.ifc"."IfcWall"`) sowie Vorgänge aus dem Vorgangsplan (`"Vorgangsmodell 1.xml"."Activity"`). Zur einfacheren Wiederverwendung in den vorausgegangenen Property-Spezifikationen sowie den nachfolgenden Kriterien werden auch hier Alias vergeben.

Die Zeilen 13–18 geben die Filterbedingungen der Elementarmodelle an (**where**-Klausel). Der Detailtext der LV-Position soll die Zeichenkette *XC1* enthalten, der Endtermin des Vorgangs soll vor dem 01.03.2013 liegen und die Wände sollen in der aktuellen Selektion des 3D-Viewers enthalten sein (vgl. dazu auch Abbildung 4.3 auf Seite 110).

Quelltext 4.1: Beispiel eines Multimodell-Filters in MMQL

```
1  use file "C:\Container\ausschreibung.mmaa"
2
3  select
4      item.id as LVPos,
5      wall.id as WallID,
6      activity."activityData"~"end"~"date" as end
7  from
8      "LV 1.X83"."Item" as item
9  linkedwith
10     "BW.ifc"."IfcWall" as wall
11 linkedwith
12     "Vorgangsmodell 1.xml"."Activity" as activity
13 where
14     item ? ("text", "detail") like "XC1"
15     and
16     date: end < date: "2013-03-01"
17     and
18     wall in viewselect
19 // as selectionXC1
```

Gemeinsam schränken Linkauswertung und Elementarmodell-Filter die Anzahl der Ergebniszeilen ein. Sie entsprechen der Selektion der relationalen Algebra.

Kommentare werden zeilenweise mit doppelten Schrägstrichen eingeleitet (`//`). So wird durch das Auskommentieren in Zeile 19 auf die optionale Angabe eines Alias für das Resultat des **select**-Statements verzichtet.

Quelltext 4.2: Ergebnis des Multimodell-Filters im CSV-Format (Auszug)

```
369 1.2.1.10.,3yyYGfBhn3Cupra2Dwp0zv,2012-07-16
370 1.2.1.10.,3zHkGa4Sz0DAJiMStwk4Uq,2013-01-10
371 1.2.1.10.,3zPnrcWSX0SgV5yi6M$r10,2012-09-11
372 1.2.1.10.,3zoOSSFYj1CPGQGhIaBhKp,2013-01-10
373 1.2.1.10.,3zu9BbFJfAI9rh994z61sC,2013-01-10
374 1.2.1.10.,3zuFHk3hr2fxuc6V_uCO3u,2012-07-16
375 1.2.1.20.,1AF4okcbjFow1QW94nwkvX,2012-11-09
376 1.2.1.20.,1noCdZnmL95vUistrRFUdr,2012-11-09
377 1.2.1.20.,1qA_SnD$j0K8ET12IWRkGr,2013-01-10
378 1.2.1.20.,1u3qiTMiz8kPJ4krAtXLD1,2013-01-10
379 1.2.1.20.,24X0GxPmbCIvxD70Yf_3gi,2013-01-10
380 1.2.1.20.,26HmWdAEn8XhwDdu9OsspU,2012-11-09
381 1.2.1.20.,2ZUAmKQWvDJwvecU3aqxho,2012-11-09
```

Quelltext 4.2 zeigt einen Auszug aus dem sich ergebenden ResultSet im Format Comma Separated Values (CSV). Da eine LV-Position (erste Spalte) mit mehreren Wänden (zweite Spalte) verlinkt ist und einem Vorgang (dritte Spalte) auch mehrere Wände zugeordnet sind, erscheinen zeilenweise alle zutreffenden Kombinationen als Ergebnis.

Eine detaillierte Erläuterung der einzelnen Sprachspezifikationen zum Multimodell-Filtern und zu Linkmanipulationen geben die folgenden Abschnitte.

4.2. Zugriff auf Originaldaten

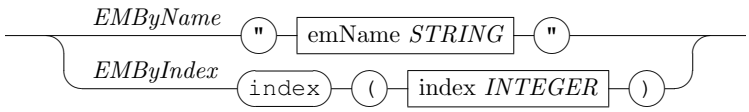
4.2.1. Zugriff auf Elemente und Properties

Durch den Verzicht auf ein integrierendes Datenschema – und damit verbundene Mappings – erlaubt das Multimodellkonzept den Zugriff auf die Originaldaten der heterogenen Elementarmodelle. Mit dem Ideellen Elementarmodell wurde hierfür eine virtuelle Struktur zur einheitlichen Angabe von Dateneinheiten geschaffen[4]. Der individuelle, technische Zugriff erfolgt durch Delegation an einen geeigneten Elementarmodell-Parser[5].

Die Sprachkonzepte der MMQL zum Zugriff auf Originaldaten folgen der Struktur des Ideellen Elementarmodells (vgl. Abbildung 3.9). Frei verwendbare Bezeichner, bspw. Namen von Modellen oder Element-Typen, werden in Anführungszeichen gesetzt, da diese vom Typ `STRING` sind und Leer- sowie Sonderzeichen enthalten können[6].

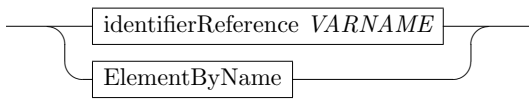
Ein Elementarmodell wird spezifiziert, indem dessen Name (als Metadatum mit dem Schlüssel `mmaa.model.name` hinterlegt) oder dessen Index (Positionsnummer im Multimodell, Schlüsselwort `index`) angegeben wird:

Syntaxdiagramm 4.4: ElementaryModelSpec



In Abschnitt 4.1.2 wurde erläutert, dass Element-Spezifikationen in MMQL als Menge gleichartiger Elemente aufzufassen sind. Elemente werden daher über den Namen ihres Element-Typs qualifiziert (`ElementByName`). Alternativ kann eine andere `ElementByName`-Spezifikation über den Alias (`identifierReference`) wiederverwendet werden (z. B. `item`, `wall` und `activity` in Quelltext 4.1, Zeile 4 ff.):

Syntaxdiagramm 4.5: ElementSpec

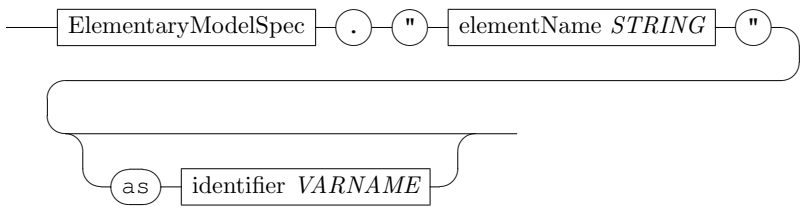


Eine `ElementByName`-Spezifikation besteht aus der Angabe des übergeordneten Elementarmodels, einem Punkt sowie aus dem Namens des Element-Typs[7]. In Quelltext 4.1 (Zeilen 8, 10, 12)

[4] vgl. Abschnitt 3.3.3 auf Seite 59
[5] vgl. Abschnitt 5.1.3 auf Seite 131
[6] vgl. Anhang A.3 auf Seite 188
[7] vgl. Abschnitt 3.3.5 auf Seite 63

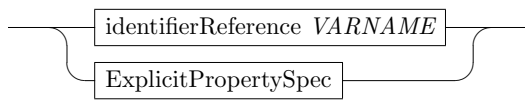
werden beispielsweise `"LV 1.X83"."Item"`, `"BW.ifc"."IfcWall"` und `"Vorgangsmodell 1.xml"."Activity"` deklariert. Optional kann ein Alias angeben werden, damit die Elementspezifikation an anderen Stellen wiederverwendet werden kann (hier: **as** `item`, **as** `wall` und **as** `activity`):

Syntaxdiagramm 4.6: ElementByName



Properties sind ihren Elementen zugeordnet. Eine Element-Spezifikation wird in MMQL als Menge aufgefasst, über die zum Auswertungszeitpunkt iteriert wird. Die Property-Spezifikation gilt daher als Angabe des Properties *desjenigen* Elements, welches in einem solchen Iterationsschritt betrachtet wird. Die Spezifikation eines Properties kann explizit (`ExplicitPropertySpec`) erfolgen oder es wird eine andere `ExplicitPropertySpec`-Spezifikation über den Alias (`identifierReference`) wiederverwendet (z. B. in Quelltext 4.1, Zeile 16):

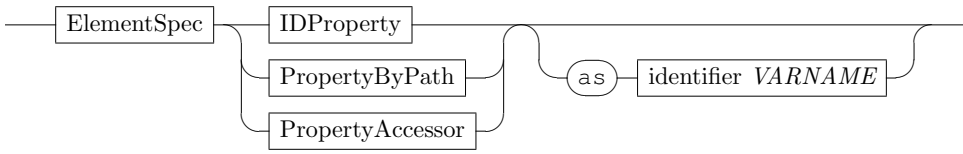
Syntaxdiagramm 4.7: PropertySpec



In Abschnitt 3.3.3 auf Seite 59 wurde erläutert, dass beim Ideellen Elementarmodell die Navigation von Element zu Property über Zugriffsfunktionen erfolgt, um die potentiell komplexen inneren Strukturen der zugrundeliegenden Datenformate abbilden zu können. Dafür sind dort zwei Konzepte vorgesehen, welche auch in der MMQL reflektiert werden: `PropertyByPath` und `PropertyAccessor`.

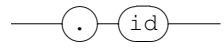
Die explizite Spezifikation von Properties beginnt mit der Angabe des übergeordneten Elements. Danach wird alternativ das inhärente Property *ID* oder eine der beiden Property-Zugriffsfunktionen angegeben. Diese werden in den folgenden Abschnitten näher erläutert. Optional kann ein Alias zur Wiederverwendung der Property-Spezifikation angeben werden (**as**-Klausel; z. B. in Quelltext 4.1, Zeile 4 ff.):

Syntaxdiagramm 4.8: ExplicitPropertySpec



Die Angabe des inhärenten ID-Properties erfolgt über einen Punkt und das Schlüsselwort `id` (z. B. in Quelltext 4.1, Zeile 4 f.):

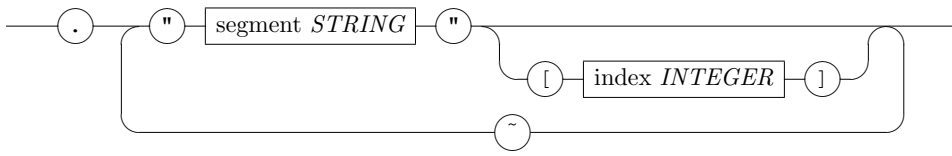
Syntaxdiagramm 4.9: IDProperty



4.2.2. Property-Zugriff auf Schemaebene

Die Property-Zugriffsfunktion `PropertyByPath` ermöglicht den schemakonformen Zugriff auf Properties. Dabei wird ausgehend von einem Element auf direkte oder verschachtelte bzw. verkettete Properties navigiert. Der entsprechende Navigationspfad wird auch *Feature-Path* genannt, da dessen Segmente Features bezeichnen – also sowohl Referenzen auf Komplexelemente (bspw. Typen oder XML-Elemente) als auch simple Attribute[8]. Der Feature-Path bildet die Property-Spezifikation. Deren Syntax ist wie folgt definiert:

Syntaxdiagramm 4.10: PropertyByPath



Als Trennsymbol zur vorausgegangenen Element-Spezifikation wird ein Punkt verwendet. Danach folgt der Feature-Path, wobei die Segmente in Anführungszeichen gesetzt und durch eine Tilde (`~`) getrennt werden. Hinter jedem Segment kann optional ein 0-basierter Collection-Index in eckigen Klammern stehen. Somit können mehrwertige Features wie Listen oder Arrays angesprochen werden. Ein nicht gesetzter Index ist äquivalent zu `index=0` und liefert für mehrwertige Features das erste Element, `index=1` das zweite usw.

[8] vgl. auch Abschnitt 3.3.4 auf Seite 61

Quelltext 4.3: Abbildung von Meilensteinen im XML-Vorgangsmodell (Auszug)

```
244 <schedule desc="Rahmenterminplan" ID="1">
245     <activity desc="Fertigstellung Rohbau E02" type="0" ID="002.005">
246         <activityData>
247             <start time="00:00:00" date="2013-01-10"/>
248             <duration>0</duration>
249             <end time="00:00:00" date="2013-01-10"/>
250             <comment></comment>
251             <durationDetType type="freeInput"/>
252         </activityData>
253     </activity>
254     <activity desc="Fertigstellung Rohbau E03" type="0" ID="002.006">
255         <activityData>
256             <start time="00:00:00" date="2013-03-28"/>
257             <duration>0</duration>
258             <end time="00:00:00" date="2013-03-28"/>
```

Beispiel 2

In Beispiel 1 wurde ein Vorgangsmodell im XML-Format verwendet. Der Element-Typ *Vorgang* dieses Elementarmodells wird durch das XML-Element `activity` abgebildet – als ID wird dessen XML-Attribut `ID` benutzt. Meilensteine sind als Vorgang mit identischem Start- und Endzeitpunkt definiert. Quelltext 4.3 zeigt einen solchen Meilenstein (Zeilen 245–253), ein weiterer ist angedeutet (ab Zeile 254).

Das Property *Beschreibung* ist im Schema als direktes XML-Attribut (`desc`) von `activity` modelliert. Die `PropertyByPath`-Spezifikation für die Vorgangsbeschreibung ist demnach:
... `"Activity"."desc"`.

Um auf den Endtermin eines Vorgangs zuzugreifen, muss von `activity` über die XML-Elemente `activityData` und `end` zum XML-Attribut `date` navigiert werden. Die `PropertyByPath`-Spezifikation des Vorgangsendes ist daher:
... `"Activity"."activityData"~"end"~"date"`

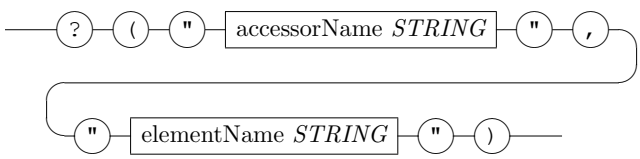
Ein weiterer Element-Typ des XML-Vorgangsmodells ist der übergeordnete Ablaufplan (XML-Element `schedule`, Zeile 1, ID ist XML-Attribut `ID`). Soll von einem `schedule`-Element aus das Startdatum des zweiten Vorgangs in MMQL angegeben werden, lautet die `PropertyByPath`-Spezifikation:
... `"Schedule"."activity"[1]~"activityData"~"start"~"date"`

Property-Zugriffe auf Schemaebene erfordern eine genaue Kenntnis des verwendeten Datenformats. Sie sind deswegen – besonders bei komplexen Datenmodellen – nur eingeschränkt für Endanwender geeignet. Sie sind auf technischer Ebene exakt nachvollziehbar, setzen jedoch die Möglichkeit zur *direkten* Navigation im Datenmodell voraus. Konzepte wie indirekte Relationsobjekte in IFC können auf diese Weise nicht angesprochen werden.

4.2.3. Semantischer Property-Zugriff

Um einen Zugriff von einem Multimodell-Element auf ein beliebiges Property zu gewährleisten, muss dafür individuelle Logik ausgeführt werden können. Mit dem Konzept der Property-Accessors werden benannte Erweiterungsmodule einem Element-Typ zugeordnet. Diese Programme können ein oder mehrere Properties für diesen Element-Typ zur Verfügung stellen. Dabei ist unerheblich, ob dieses Property im Datenschema direkt, indirekt, unter anderem Namen oder *überhaupt* modelliert ist. Auf diese Weise wird es möglich, sogar volatile Eigenschaften eines Multimodell-Elements zu erfassen. Fachlich bedeutet dies die Realisierung einer Filterfunktion auf Element-Ebene. Die fachliche Motivation verleiht einem solchen Property-Zugriff seine Semantik. Die MMQL-Syntax für einen `PropertyAccessor` ist wie folgt definiert:

Syntaxdiagramm 4.11: PropertyAccessor



Das Trennsymbol zur vorausgegangenen Element-Spezifikation ist ein Fragezeichen. Anschließend wird innerhalb geschweifter Klammern der Name des PropertyAccessors sowie der Name des gewünschten Properties angegeben.

Beispiel 3

In Beispiel 1 wurde ein Leistungsverzeichnis im Format GAEB-DA-XML verwendet. Für den Element-Typ LV-Position (`item`) sollte der Langtext auf das Vorkommen der Zeichenkette *XC1* untersucht werden. Wie Quelltext 4.4 zeigt, sind sowohl Langtext (Zeile 568 ff.) als auch der Kurztext (Zeile 576 f.) mit HTML-ähnlichen Formatierungsangaben durchsetzt.

Um den reinen Kurz- oder Langtext jeweils als Property eines `item`-Elements zur Verfügung zu stellen, muss über mehrere XML-Unterelemente navigiert sowie die Formatierungsangaben aus dem XML-Text entfernt werden. Diese Textoperationen werden in einem Property-Accessor mit dem Namen *text* für den Element-Typ LV-Position (`item`) implementiert. Sie stellen dort zwei Properties zur Verfügung: `outline` für Kurztext und `detail` für Langtext.

Die `PropertyAccessor`-Spezifikation für den Langtext einer LV-Position lautet demnach (vgl. Quelltext 4.1, Zeile 16, dort mit Alias-Referenz):

```
... "Item"?("text", "detail")
```

Beispiel 4

In Bauwerksmodellen im IFC-Format sind die Eigenschaften (`IfcProperty`) von

Quelltext 4.4: Beispiel einer LV-Position in GAEB-DA-XML (Auszug)

```
561 <Item ID="IBAGELNA" RNoPart="10">
562     <Qty>11946.149</Qty>
563     <QU>m3</QU>
564     <Description>
565         <CompleteText>
566             <DetailTxt>
567                 <Text>
568                     <p style="margin-left:14pt;text-align:left;margin-
                            top:0pt;margin-bottom:0pt;">
569                      <span style="font-family:Arial;font-size:10pt;
                            Color:rgb(0,0,0);">Ortbeton der Wand,<br/>aus
                            Beton DIN 1045,<br/>Wände in allen Bereichen,<
                            br/>auch von wandartigen Trägern,<br/>
                            Festigkeitsklasse: C 30/37<br/>
                            Expositionsklasse: XC1<br/>Wandstärke bis 30 cm
                            <br/>Wandhöhe entsprechend Ebene/Einbaubereich<
                            br/>gemäß Plänen der Anlage.<br/>Einbau in
                            Ebene U2 bis E04.</span>
570                     </p>
571                 </Text>
572             </DetailTxt>
573             <OutlineText>
574                 <OutlTxt>
575                     <TextOutlTxt>
576                         <p style="text-align:left;margin-top:0pt;margin-
                                bottom:0pt;">
577                          <span>Stahlbeton Wand bis d=30 cm</span>
578                         </p>
579                     </TextOutlTxt>
580                 </OutlTxt>
581             </OutlineText>
582         </CompleteText>
583     </Description>
584 </Item>
```

Bauteilen indirekt modelliert. Quelltext 4.5 zeigt eine Wand in IFC (Zeile 26). Deren Eigenschaft *Bewehrungsgrad* (Zeile 27) ist nicht direkt vom `IfcWall`-Objekt aus erreichbar. Vielmehr bildet das Objekt `IfcRelDefinesByProperties` (Zeile 25) den Einstiegspunkt zur Zuordnung von Eigenschaften zu Bauteilen.

Es ist nicht möglich, mittels PropertyByPath-Ausdrücken von IfcWall zum IfcProperty *Bewehrungsgrad* zu navigieren. Daher wird ein Property-Accessor mit dem Namen *property* (in Anlehnung an `IfcProperty`) für IFC-Objekte definiert, welcher diese Verbindung herstellt. Als Argument wird der gewünschte, in IFC frei wählbare, Name des IfcProperties angegeben. Die Spezifikation für *Bewehrungsgrad* lautet:

```
... "IfcWall"?("property", "Bewehrungsgrad")
```

Quelltext 4.5: Beispiel einer Wand in IFC (SPF-Format, Auszug)

```
#18=IFCPROPERTYSET('0e34DhIg54eRChSh6suuhf',#22,'Properties',$,(#29,#6,#
    46,#35,#63,#92,#98,#28,#45,#100));
#22=IFCOWNERHISTORY(#89,#20,.READWRITE.,.ADDED.,$,$,$,1346762258);
#23=IFCRELDEFINESBYPROPERTIES('00fepWKKHEsRK05Y228zkQ',#22,$,$,(#88),#18);
#88=IFCWALL('2EKEAwBorDjxO46qUthyqD',#22,'STB 25.0',$,'Wall',#85,#8,'Wall'
    );
#98=IFCPROPERTYSINGLEVALUE('Bewehrungsgrad',$,IFCLABEL('150 kg_m3'),$);
```

Die Anwendung von Property-Accessors ist auch für abstrakte Elemente möglich, bspw. könnte von Bauelementen (`IfcBuildingElement`) zur Nummer des zugehörigen Geschosses (`IfcStorey`) navigiert werden. Durch die freie Implementierung ermöglicht der semantische Property-Zugriff darüber hinaus auch Berechnungsoperationen. So kann bspw. die volatile Eigenschaft *Höhe* einer Wand in IFC durch einen Property-Accessor aus deren komplexen Geometrieinformationen kalkuliert werden.

4.2.4. Values

Values repräsentieren die atomaren Daten der Elementarmodelle. In der MM-Engine werden zum Zeitpunkt der MMQL-Auswertung Property-Spezifikationen durch die realen Werte aus dem Elementarmodell ersetzt. Diese werden in der MMQL zu zwei Zwecken benötigt:

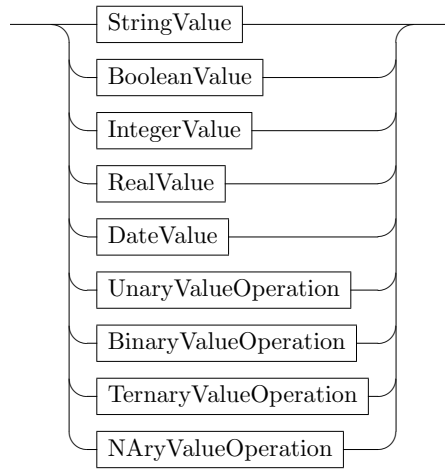
1. Zur Ausgabe als Ergebnis in den Zellen des ResultSets

2. Zur Auswertung von Kriterien

Values können daher als Property-Spezifikation angegeben werden. Darüber hinaus gibt es auch die Möglichkeit, Values in MMQL-Anweisungen als Literal darzustellen. Literale kommen zum Einsatz, wenn statische Werte unabhängig von den Elementarmodell-Daten benötigt werden, bspw. für Vergleiche mit festen Größen.

Aufgrund der Vielfalt der zugrunde liegenden Datenformate werden alle geparsten Values zunächst als Zeichenkette (String) interpretiert. Auch im ResultSet erscheinen sie als String. Lediglich wenn die MM-Engine die Bedeutung eines Values kennen muss – bspw. für Vergleiche oder mathematische Operationen – muss der Anwender den Datentyp des Values manuell deklarieren (Casting). Ohne Casting wird der Value weiterhin als String interpretiert und würde, bspw. bei mathematischer Verwendung, eine Fehlermeldung provozieren. Die Angabe eines Values kann daher aus einem der fünf Datentypen sowie aus einer ein-, zwei-, drei- oder n-wertigen Werteoperation bestehen:

Syntaxdiagramm 4.12: Value



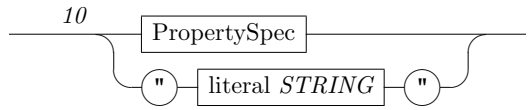
Datentypen

MMQL ist grammatikalisch und syntaktisch links assoziativ – bspw. bedeutet `a-b`, dass `b` von `a` subtrahiert wird und `a-b+c` evaluiert zu `(a-b)+c`. MMQL-Anweisungen mit verschiedenen, verketteten, grammatikalisch gleichgestellten Operatoren müssen in semantisch korrekter Reihenfolge interpretiert werden. Bspw. muss Punkt- vor Strichrechnung sowie UND vor ODER umgesetzt werden. Zu diesem Zweck erhalten betroffene Syntaxdefinitionen ein Gewicht. Dieses bezeichnet auf einer Skala von 1–10 den Vorrang gegenüber alternativen Operatoren. Die Auswertung von Values hat mit dem Gewicht 10 die höchste Priorität.

Als Datentypen sind Zeichenketten (String), Wahrheitswerte (Boolean), Ganzzahlen (Integer) und Fließkommazahlen mit doppelter Genauigkeit (Real) definiert. Ihr Wertebereich richtet sich nach der MOF-Spezifikation ISO 19502 (2005). Außerdem gibt es noch den Datentyp Datum (Date) mit Darstellung und Wertebereich in Anlehnung an ISO 8601:2004(E) (2004). Die Datentypen entsprechen den korrespondierenden primitiven Datentypen sowie den Klassen `java.lang.String` und `java.util.Date` in der Programmiersprache Java (vgl. Gosling et al., 2005, S. 35 f., 48 f.).

Der Wertebereich von Strings ist definiert durch die unendliche Menge aller endlichen Sequenzen von 16 bit-Zeichen, ausgenommen des Null-Zeichen-Werts (ISO 19502, 2005, S. 135), sowie `null`. Die Syntaxdefinition lautet:

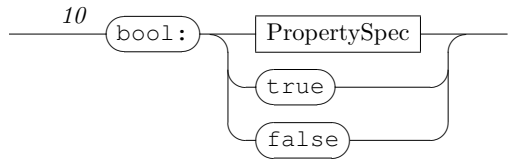
Syntaxdiagramm 4.13: StringValue



Wie alle Values können Strings als Property-Spezifikation oder als Literal angegeben werden. String-Literale werden in Anführungszeichen gesetzt.

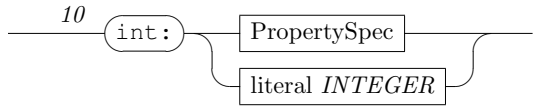
Wahrheitswerte können die Zustände **`true`** und **`false`** (ISO 19502, 2005, S. 135) sowie `null` einnehmen und werden mit dem Castingoperator **`bool:`** eingeleitet:

Syntaxdiagramm 4.14: BooleanValue



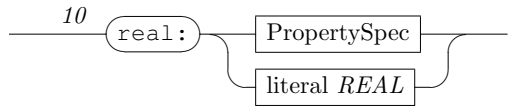
Integer sind die Teilmenge der ganzen Zahlen im Bereich von -2^{31} bis $+2^{31}-1$ (ISO 19502, 2005, S. 135) sowie `null`. Ihr Castingoperator ist **`int:`**:

Syntaxdiagramm 4.15: IntegerValue



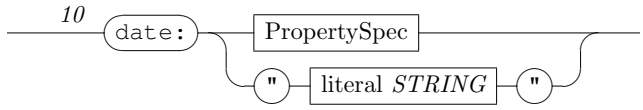
Fließkommazahlen sind die Teilmenge der rationalen Zahlen, die den Werten der IEEE double precision floating point numbers entspricht (ISO 19502, 2005, S. 135) sowie `null`. Ihr Datentyp wird mit **`real:`** deklariert:

Syntaxdiagramm 4.16: RealValue



Die Menge der Strings, die eine Datumsangabe nach ISO 8601:2004(E) (2004) formen, sowie `null` bilden den Datentyp Date. Dieser wird durch **`date:`** bestimmt:

Syntaxdiagramm 4.17: DateValue



Beispiel 5

In Quelltext 4.1 sind unterschiedliche Value-Angaben spezifiziert:

Nicht evaluierte Property-Spezifikationen In den Zeilen 4–6 sind die Values der IDs von LV-Positionen und Wänden sowie das Enddatum von Vorgängen angegeben. Sie dienen lediglich zur Ausgabe im ResultSet und werden zu diesem Zweck vom MMQL-Interpreter nicht logisch evaluiert. Es wird die originale Zeichenkette aus dem Elementarmodell ausgegeben. Der Value evaluiert daher in jedem Iterationsschritt zum Wert des Properties des betrachteten Multimodell-Elements. Dementsprechend stehen in der ersten Spalte des ResultSets (vgl. Quelltext 4.2) die ID-Werte der LV-Positionen (`1.2.1.10.`, `1.2.1.20.`, ...), in der zweiten Spalte die IDs der Wände (`3yyYGfBhn3Cupra2Dwp0zv`, `3zHkGa4Sz0DAJiMStwk4Uq`, ...) und in der dritten Spalte das Enddatum der Vorgänge (`2012-07-16`, `2013-01-10`, ...).

Evaluierte Property-Spezifikationen In den Zeilen 14 und 16 werden Values spezifiziert, die durch den MMQL-Interpreter ausgewertet und deswegen *verstanden* werden müssen. So evaluiert der Value des Property-Accessors `item ?("text", "detail")` zur (bereinigten) Zeichenkette des Langtextes der jeweiligen LV-Position des aktuellen Iterationsschritts, z. B. zu *„Ortbeton der Wand, aus Beton DIN 1045, Wände in allen Bereichen, auch von wandartigen Trägern, Festigkeitsklasse: C 30/37 Expositionsklasse: XC1 Wandstärke bis 30 cm Wandhöhe entsprechend Ebene/Einbaubereich gemäß Plänen der Anlage. Einbau in Ebene U2 bis E04."* (vgl. Quelltext 4.4). Der Value der Property-Spezifikation `end` (Alias-Referenz) evaluiert zum Enddatum des Vorgangs des aktuellen Iterationsschritts, z. B. *10. Januar 2013* oder *28. März 2013* (vgl. Quelltext 4.3).

Literale In Zeile 16 ist das Datumsliteral **`date:`**`"2013-03-01"` angegeben. Dieses evaluiert immer zu *1. März 2013*. Die Definition des Strings `"XC1"` in Zeile 14 evaluiert immer zu *„XC1"*.

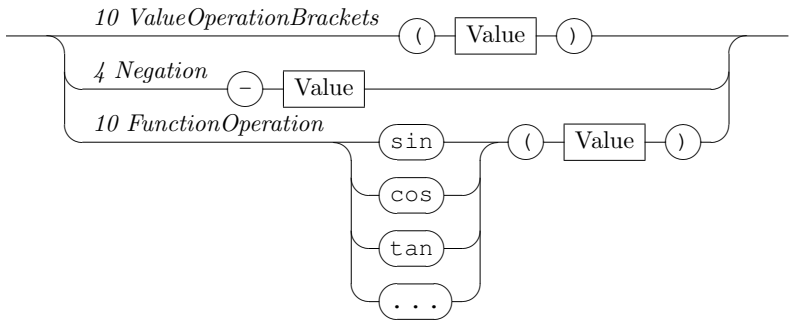
Value-Operationen

MMQL bietet die Möglichkeit, Operationen auf Values auszuführen. Formal handelt es sich um eine Transformation von *einem* (unäre Value-Operationen), zwei (binäre Value-Operationen), drei (ternäre Value-Operatoren) oder n (n-äre Value-Operatoren) Values in *einen* anderen Value. Die Ein- und Ausgangstypen der Values können verschieden sein.

Die hier angegebenen Operatoren gelten für die Datentypen Real und Integer, die Addition zusätzlich für Date und String. Des Weiteren existieren verschiedene Operationen zur Manipulation von Zeichenketten. Boolsche Operationen, sowie weitere Operationen, die zu Wahrheitswerten evaluieren, werden in Abschnitt 4.3.2 betrachtet. Die Anwendung einer Value-Operation auf einen Wert falschen Datentyps führt zu einer Fehlermeldung.

Unäre Value-Operationen sind wie folgt definiert:

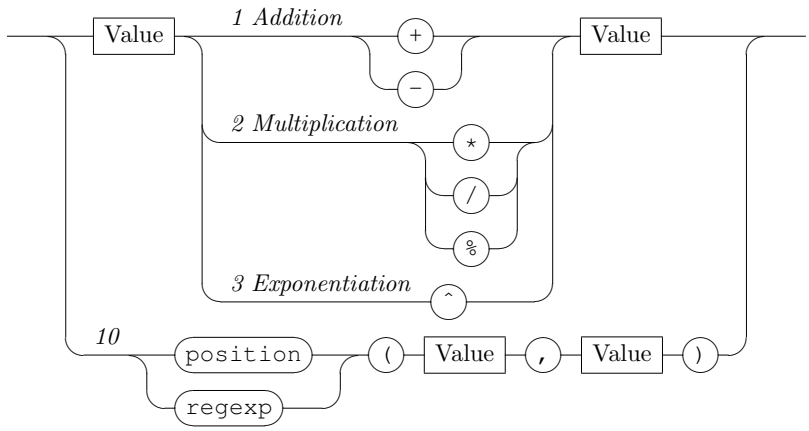
Syntaxdiagramm 4.18: UnaryValueOperation



Die Semantik der Klammeroperation ist *Vorrang* – die Operation hat daher das Gewicht 10. Die Negation multipliziert einen Zahlenwert mit −1. Des Weiteren werden verschiedene mathematische Operationen wie trigonometrische Funktionen oder Logarithmen sowie Funktionen zur Manipulation und Introspektion von Zeichenketten angeboten. Die vollständige Aufzählung unärer Value-Operationen ist in der Produktionsregel `FunctionOperation` in Quelltext A.3 auf Seite 188 (Zeile 43) wiedergegeben.

Binäre mathematische Operationen sind Addition (inkl. Subtraktion), Multiplikation (inkl. Division und Modulo) und Potenzierung. Außerdem existieren die Funktionen zur Bestimmung der Position von Zeichen innerhalb einer anderen Zeichenketten sowie zur Überprüfung eines Strings auf Übereinstimmung mit einem Regulären Ausdruck (Regexp, Pattern Matching):

Syntaxdiagramm 4.19: BinaryValueOperation



Da alle Value-Operationen wieder zu einem Value evaluieren, ist durch Substitution und unter Berücksichtigung der Operatorengewichte die Definition beliebig komplexer, verketteter Funktionen möglich.

Beispiel 6

Die Auslegerposition eines Turmdrehkrans ist im Baustellenmodell implizit in Polarkoordinaten angegeben. Sie wird jedoch in kartesischen Koordinaten benötigt. Dabei verweist der Alias `jibLength` auf das Property für die Länge des Auslegers, der Alias `jibAngle` auf das Property für den Drehwinkel des Turms in der Horizontalebene:

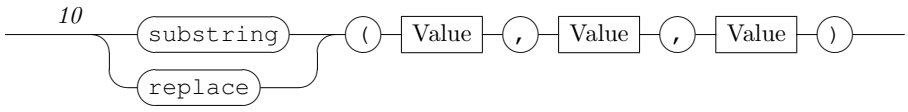
```
jibLength * cos(jibAngle)
jibLength * sin(jibAngle)
```

Für einen Vergleich mit dem Leistungsverzeichnis wird das Volumen eines quaderförmigen Fundaments benötigt:

```
footingLength * footingWidth * footingHeight
```

Ternäre Value-Operatoren sind für Funktionen zur Manipulation von Zeichenketten definiert:

Syntaxdiagramm 4.20: TernaryValueOperation

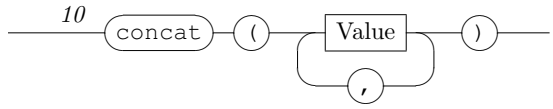


Während `replace` für die Ersetzung von Teilzeichenketten als Argumente nur Strings akzeptiert (Gesamtzeichenkette, alter Wert, neuer Wert), verwendet `substring` Argumente mit

gemischten Datentypen. Für die Rückgabe eines Teils einer Zeichenkette wird nämlich die originale Gesamtzeichenkette (String), sowie die 0-basierten Indizes (Integer) für Start und Ende des Substrings benötigt.

Die Funktion `concat` kann beliebig viele Zeichenketten aneinanderfügen. Damit ist sie ein n-ärer Value-Operator:

Syntaxdiagramm 4.21: NAryValueOperation



4.3. Multimodell-Filtern

4.3.1. Prinzip

Die Erschließung des Multimodell-Informationsraumes geschieht per MMQL durch Bildung eines geeigneten ResultSets. Das ResultSet des vollständigen Multimodells bestünde aus einer Tabelle, deren Spalten alle Properties aller Elemente aus allen Elementarmodellen wären und deren Zeilen die entsprechenden Values über die möglicherweise mehrwertig kombinierten Links aller Linkmodelle enthielten. Diese Datenmenge ist für einen Menschen nicht beherrschbar. Auch innerhalb des Computers wäre diese Darstellungsform bei realistischen Baufachmodellen nicht effizient handhabbar. Durch das Auskombinieren der mehrwertigen Links entstünde ein Vielfaches der ursprünglichen Datenmenge der Elementar- und Linkmodelle.

Um den Multimodell-Informationsraum nutzbringend zu erschließen, muss das Multimodell gefiltert werden. Dazu darf das ResultSet nur die für den Anwender relevanten Daten enthalten. Zu diesem Zweck wird die Anzahl der Spalten und Zeilen dieser Tabelle reduziert. Folgende Methoden bilden das Prinzip des Multimodell-Filterns:

- Projektion (Festlegung der Spalten)
- Selektion (Reduktion der Zeilen)
  - Filtern von Elementarmodellen
    - Property-Kriterien
    - Benannte Elementmengen
  - Modellübergreifendes Filtern
    - Linkauswertung
      - Elementkombination
      - Linkinterpretation
    - Elementarmodellübergreifende Property-Kriterien

Die Festlegung der Spalten des ResultSets (Projektion) geschieht durch Angabe von Property-Spezifikationen und wurde bereits in Abschnitt 4.1.2 auf Seite 86 dargestellt.

In den folgenden Abschnitten werden die Methoden der Selektion erläutert. Das gemeinsame Prinzip aller Selektionsmethoden ist die Ermittlung, ob ein Multimodell-Element in das ResultSet aufgenommen werden darf oder nicht. Zwar werden im ResultSet keine Elemente sondern Values von Properties dargestellt, implizit ist jedoch jedem Property sein korrespondierendes Element zugeordnet. Somit ist Multimodell-Filtern das Filtern von Elementen. Die Relevanz eines Multimodell-Elements ergibt sich aus:

1. der Existenz und Wirksamkeit zugeordneter Links
2. einem oder mehreren Kriterien für das Element
3. einer Kombination aus 1. und 2.

4.3.2. Filtern von Elementarmodellen

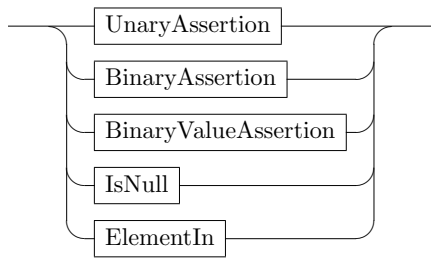
Das `select`-Statement der MMQL erlaubt die Anwendung aller Selektionsmethoden des Multimodell-Filterns. Die Spezifikation von Filtern für ein singuläres Elementarmodell ist ebenso zulässig wie komplexe elementarmodellübergreifende Anweisungen. In diesem Abschnitt werden die Methoden vorgestellt, welche sich auf die Daten *eines* Elementarmodells beziehen und bei denen Links sowie andere Elementarmodelle unberücksichtigt bleiben.

Im Grunde handelt es sich bei MMQL-Anweisungen zum reinen Elementarmodell-Filtern um `select`-Statements ohne `LinkedWith`-Klausel, wodurch keine Linkmodelle betrachtet werden. Des Weiteren dürfen keine weiteren Elementarmodelle, bspw. zur Property-Spezifikation, benutzt werden.

Property-Kriterien

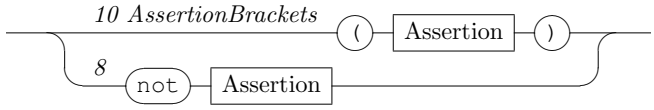
Property-Kriterien sind Filterkriterien, welche für die Wertausprägung (Value) eines Properties erfüllt sein müssen, damit das korrespondierende Multimodell-Element in das ResultSet aufgenommen wird. Dazu müssen die Kriterien zu Wahrheitswerten evaluieren. Sie werden deshalb auch Assertion (Aussage) genannt. In der MMQL sind folgende Assertion-Operatoren vorgesehen:

Syntaxdiagramm 4.22: Assertion



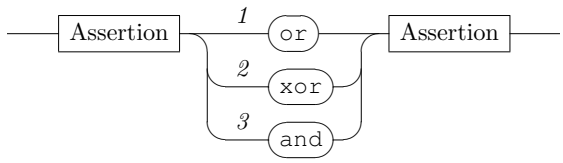
Zu den einstelligen Assertion-Operatoren gehören die Klammeroperation für Vorrang bei logischen Ausdrücken (Gewicht 10) sowie das logische NICHT (`not`):

Syntaxdiagramm 4.23: UnaryAssertion



Zweistellige Assertion-Operatoren sind – mit aufsteigender Priorität – das logische ODER (`or`), EXKLUSIV-ODER (`xor`) sowie UND (`and`).

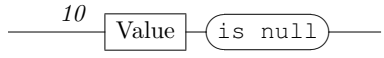
Syntaxdiagramm 4.24: BinaryAssertion



Das EXKLUSIV-ODER wurde zur vereinfachten Nutzung der Sprache aufgenommen. Es lässt sich auch durch Umformulierung ausdrücken und ist definiert als $a \oplus b \equiv \neg\,(a = b)$.

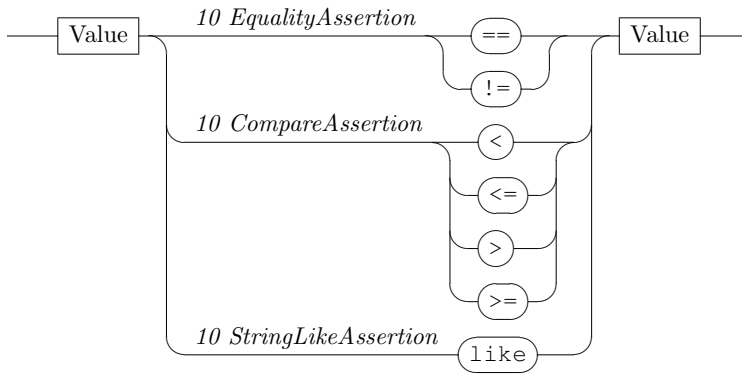
Analog zu Beispiel 6 auf Seite 101 lassen sich durch Substitution beliebig komplexe logische Ausdrücke bilden. Um die Originaldaten der Elementarmodelle in diese logischen Ausdrücke einbeziehen zu können, werden die folgenden Assertion-Operatoren benötigt. Sie sind selbst Assertions und stellen Aussagen über Values dar, welche zu Wahrheitswerten evaluieren. Die einzige unäre Value-Assertion ist die Prüfung eines Values auf Nichtbelegung (`null`):

Syntaxdiagramm 4.25: IsNull



Die folgenden binären Value-Assertions testen die Gleichheit bzw. Ungleichheit zweier Values (== bzw. !=), vergleichen die Größenverhältnisse zweier Werte (< ... >=) und prüfen ob die linke Zeichenkette die rechte enthält (`like`):

Syntaxdiagramm 4.26: BinaryValueAssertion



Quelltext 4.6: Beispiel eines Elementarmodell-Filters für ein Leistungsverzeichnis in MMQL

```
use file "C:\Container\ausschreibung.mmaa"

select
    item.id as LVPos,
    item ? ("text", "outline"),
    item."qty" as quantity
from
    "LV 1.X83"."Item" as item
where
    item ? ("text", "detail") like "stahl"
    and
    real: quantity > 500
```

Beispiel 7

Quelltext 4.6 zeigt einen Elementarmodell-Filter für das GAEB-DA-XML-Leistungsverzeichnis aus Quelltext 4.4. Er ermittelt die IDs, die Kurztexte und die Mengen aller LV-Positionen, welche die Zeichenkette *„stahl"* im Langtext enthalten und deren Menge größer als 500 ist. Das Filterergebnis ist in Quelltext 4.7 dargestellt.

Dreiwertige Logik

In Abschnitt 3.3.5 auf Seite 64 wurde festgelegt, dass Values zu `null` evaluieren wenn der tatsächliche Wert nicht belegt ist. Da `null` selbst ein Wert ist – mit der Bedeutung *undefiniert* – müssen alle logischen Werteoperationen diesen potentiellen Zustand reflektieren. Daher wird eine dreiwertige Logik für Assertion-Operatoren eingeführt. Tabelle 4.1 zeigt in Wahrheitstabellen

Quelltext 4.7: ResultSet des LV-Elementarmodell-Filters im Format CSV

```
3.30.,Stahlkonstruktion Decken,7050.069
3.10.,Stahlkonstruktion Träger,809.736
1.2.3.30.,Bewehrung FT-Nebenträger,3275.379
1.1.2.10.,Ortbeton Einzelfundamente,3162.428
1.2.2.30.,Bewehrung Hauptträger (250 bis 280 kg/m3),2229.572
1.2.6.150.,Bewehrung Fertigteilstützen,814.650
1.1.1.30.,Ortbeton Fundamentplatten bis 25 cm,1289.513
1.1.1.40.,Ortbeton Fundamentplatten bis 100 cm,6344.111
1.2.1.50.,Bewehrung Wände,1806.808
1.1.1.70.,Bewehrung Fundamentplatten,1211.374
1.2.5.40.,Bewehrung Filigran-Deckenplatten,4285.047
1.2.3.10.,Beton Nebenträger als Fertigteil,10926.083
```

das Verhalten der Junktoren in der MMQL. Dieses wurde in Analogie zu SQL festgelegt. Die Anwendung dreiwertiger Logik in der relationalen Algebra wird von Codd (1979) beschrieben. Sie ist auch im SQL-Standard (ISO 9075-1, 2008) definiert.

Durch Anwendung der dreiwertigen Logik können Ergebnisse entstehen, die nicht intuitiv vermutet werden – besonders dann, wenn bei tief geschachtelten Kriterien eine Aussage frühzeitig zu `null` evaluiert. Dies kann bei allen binären Value-Assertions eintreten, wenn einer der beiden Values in den Originaldaten nicht belegt ist. Das `null`-Ergebnis kann sich nach Tabelle 4.1 bis auf das Endresultat des Kriteriums auswirken.

Ein Kriterium kann allerdings durch `null`-Überprüfung so umformuliert werden, dass es immer einen zweiwertigen Wahrheitswert (`true` oder `false`) liefert. Beispielsweise kann der Ausdruck `a == true` potentiell zu `null` evaluieren; der Ausdruck `a == true and not a is null` jedoch nicht.

Welcher Ausdruck zur Anwendung kommt, ist von der Intention des Benutzers abhängig. In jedem Fall gilt ein Property-Kriterium, welches schlussendlich zu `null` evaluiert, ebenso wie `false`, als nicht erfüllt.

Tabelle 4.1.: Wahrheitstabellen der dreiwertigen Logik (W=wahr, F=falsch)

(a) Unärer Junktor

a	$\neg a$
W	F
F	W
null	null

(b) Binäre Junktoren

a	b	$a \vee b$	$a \wedge b$	$a \oplus b$	$a = b$
W	W	W	W	F	W
W	F	W	F	W	F
W	null	W	null	null	null
F	W	W	F	W	F
F	F	F	F	F	W
F	null	null	F	null	null
null	W	W	null	null	null
null	F	null	F	null	null
null	null	null	null	null	null

Quelltext 4.8: Beispiel eines Elementarmodell-Filters für ein Bauwerksmodell in MMQL

```
use file "C:\Container\ausschreibung.mmaa"

select
    wall.id,
    wall ? ("property", "FireRating") as firerating
from
    "BW.ifc"."IfcWall" as wall
```

Beispiel 8

Quelltext 4.8 zeigt einen Elementarmodell-Filter für das IFC-Bauwerksmodell aus Quelltext 4.5. Er ermittelt die IDs aller Wände und deren Belegung des IFC-Properties *FireRating*, welches die Feuerwiderstandsklasse repräsentiert. Das Filterergebnis ist in Quelltext 4.9 auszugsweise dargestellt. In einigen Zeilen fehlt die Feuerwiderstandsklasse, was im verwendeten CSV-Format den Wert `null` repräsentiert. Daran ist zu erkennen, dass für diese Wände das FireRating-Property in der ursprünglichen IFC-Datei nicht gesetzt wurde. Insgesamt hat das ResultSet 1784 Zeilen, was der Anzahl aller Wände entspricht.

Würde in Quelltext 4.8 das Kriterium
`where not firerating is null`
angefügt, kämen nur Wände in das ResultSet, deren FireRating-Property belegt ist. Das Filterergebnis enthielte 1460 Zeilen.

Würde man in Quelltext 4.8 stattdessen das Kriterium
`where firerating == "F90"`
anfügen, kämen nur Wände in das ResultSet, deren FireRating-Property belegt ist und *F90* beträgt. Wände mit *F60* oder *F120* würden nicht ins Ergebnis aufgenommen, weil der Gleichheitsoperator zu `false` evaluiert. Wände, deren FireRating-Property nicht belegt ist, würden deswegen nicht ins Ergebnis aufgenommen, weil der Gleichheitsoperator zu `null` evaluiert. Die Größe des ResultSets betrüge 943 Zeilen.

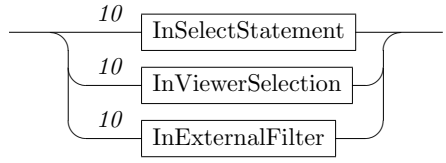
Benannte Elementmengen

Benannte Elementmengen bilden die zweite Methode zum Filtern von Elementarmodellen. Im Gegensatz zu Property-Kriterien liefern sie keine Values als Entscheidungskriterium über Elemente, sondern eine Menge von IDs, welche zulässige Elemente $\hat{E}$ repräsentieren. Ist ein betrachtetes Multimodell-Element e in dieser Liste enthalten ($e \in \hat{E}$), wird es in die Ergebnismenge aufgenommen. Das gemeinsame Schlüsselwort für benannte Elementmengen ist **in**. Folgende Methoden stehen als benannte Elementmengen zur Verfügung:

Quelltext 4.9: ResultSet des IFC-Elementarmodell-Filters im Format CSV (Auszug)

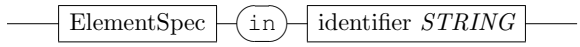
```
0zWdR7b7rC_OrFgjNh5umS,F90
2fbpLq0PH0tfWBrbSg3Rzx,F60
0Ijepo4RP1tRIw4WFwuViG,F60
11zL44e6n3yuvtvW$0VkiF,F120
27z_Y2OOv6Y874AZmjvVtW,F120
2WrV2XGDH4Q8DY$0OIQ8NS,F90
3YQexFb7b9qPdW5_8kSzzY,F120
2LB_49ioTCFwXGC51RQ4ee,
1XjKt_ubjEwwXL_sSNcFtX,F90
3ThwFySwv1VBnB8AKbFQzy,F60
3UK5DUnqrD6fsHn9mM4PI2,
1L4CLDp$52mxop$a_gvXqM,
3c70Dm_BDFfwG34zCgvffg,F90
3n0Xl_dafB69n8sE7bOGfn,F90
1uLXCMCzL3UBTTTzTJixua,F90
```

Syntaxdiagramm 4.27: ElementIn



Vorausgegangene Select-Ergebnismengen liefern die IDs aller Elemente eines ResultSets, welches in einer vorausgegangenen `select`-Anweisung ermittelt wurde. Dafür muss der Alias dieser `select`-Anweisung angegeben werden:

Syntaxdiagramm 4.28: InSelectStatement



Beispiel 9

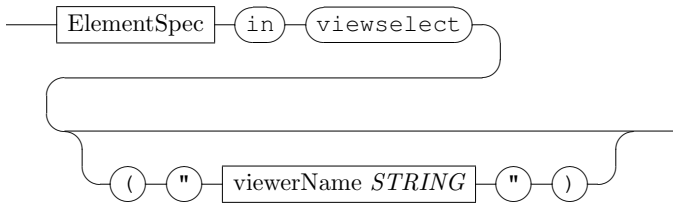
Quelltext 4.10 zeigt ein `select`-Statement, welchem der Quelltext 4.1 vorausgeht. Zusätzlich wurde dem dortigen `select`-Statement das Alias `selectionXC1` gegeben. Die nun nachfolgende Anweisung liefert alle LV-Positionen, deren Menge größer als 1000 ist und welche bereits im ResultSet der ersten Anweisung (vgl. Quelltext 4.2) enthalten sind. Das Ergebnis ist ein ResultSet mit *einem* Eintrag: `1.2.1.10.,11946.149`

Quelltext 4.10: Beispiel eines Elementarmodellfilters mit vorausgegangener Select-Ergebnismenge für ein LV in MMQL (Auszug)

```
21 select
22     item.id,
23     item."qty" as quantity
24 from
25     "LV 1.X83"."Item" as item
26 where
27     item in selectionXC1
28     and
29     real: quantity > 1000
```

Viewerselektionen liefern die IDs aller Elemente, welche in einem Elementarmodell-Viewer einer Multimodell-Anwendung ausgewählt sind:

Syntaxdiagramm 4.29: InViewerSelection



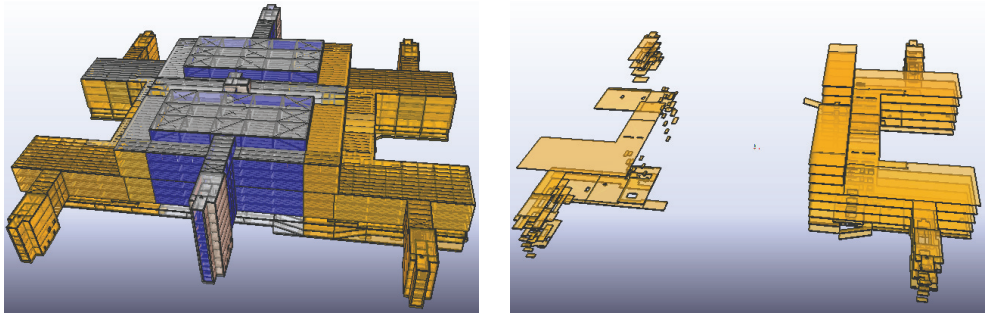
Dem Schlüsselwort **in** folgt der Modifikator **viewselect** sowie die optionale Angabe des Namens des Elementarmodell-Viewers in Klammern. Dies ist notwendig, wenn die Multimodell-Anwendung gleichzeitig mehrere Viewer für dasselbe Elementarmodell erlaubt.

Quelltext 4.11: Beispiel eines Elementarmodellfilters mit Viewerselektion für ein IFC-Modell in MMQL

```
1 use file "C:\Container\ausschreibung.mmaa"
2
3 select
4     slab.id
5 from
6     "BW.ifc"."IfcSlab" as slab
7 where
8     slab in viewselect
```

Beispiel 10

Quelltext 4.11 zeigt ein `select`-Statement, für ein IFC-Modell. Es liefert die IDs der Deckenplatten, welche sich in der Bauteilauswahl eines 3D-Viewers befinden



(a) Zum Zeitpunkt der Abfrage **(b)** Per MMQL ermittelte Deckenplatten

Abbildung 4.3.: Nutzerauswahl von Bauteilen (gelb) im IFC-3D-Viewer *Open-IFC-Tools* (`http://www.openifctools.org`)

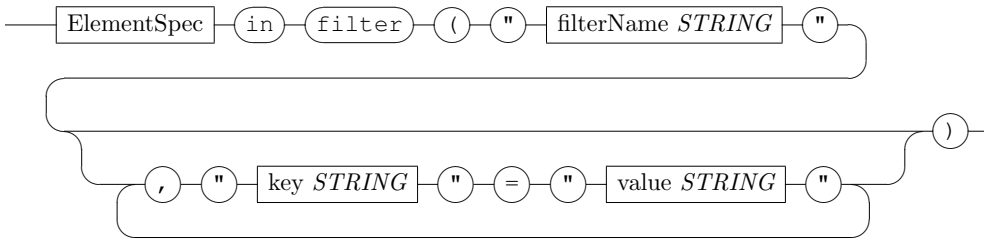
(vgl. Abbildung 4.3a). Das Resultat enthält 203 Deckenplatten. Zur Verdeutlichung wurden lediglich diese ermittelten Bauteile erneut ausgewählt (vgl. Abbildung 4.3b). Die Anzahl aller Deckenplatten im Gesamtmodell beträgt 365.

Externe Elementarmodellfilter sind Erweiterungsmodule, welche beliebige Logik ausführen können. Ihre Aufgaben sind:

- die Wiederverwendung existierender Filterservices
- die Realisierung domänenspezifischer, semantisch hochwertiger Fachfilter
- die Übernahme von Logik, welche in MMQL nicht effizient oder überhaupt nicht ausgedrückt werden kann.

Externe Elementarmodellfilter werden über den Modifikator **`filter`** und ihren Namen aufgerufen. Optional können Parameter über Schlüssel-Wert-Paare übergeben werden:

Syntaxdiagramm 4.30: InExternalFilter



Quelltext 4.12: Beispiel eines externen Elementarmodellfilters für ein IFC-Modell in MMQL

```
1 use file "C:\Container\ausschreibung.mmaa"
2
3 select
4     be.id
5 from
6     "BW.ifc"."IfcBuildingElement" as be
7 where
8     not be in filter("IFC Structural Filter")
```

Beispiel 11

Für eine Ausschreibung soll ein IFC-Modell nur Rohbauelemente enthalten. Um dies zu überprüfen, wird der externe IFC-Elementarmodellfilter `IFC Structural Filter` der *BIM Filter Toolbox* (BIMfit, Windisch et al., 2012a) verwendet. Quelltext 4.12 zeigt ein `select`-Statement, mit dem Aufruf des externen Tools in Zeile 8. Die Anfrage liefert alle Bauelemente, welche *nicht* Teil des `IFC Structural Filter`-Ergebnisses sind. Das Resultat enthält 0 Zeilen – das Bauwerksmodell enthält nur Rohbauelemente.

Quelltext 4.13: Mögliche Einbindung fremder textueller Filtersprachen in MMQL

```
1  use file "C:\Container\ausschreibung.mmaa"
2
3  select
4      column.id
5  from
6      "BW.ifc"."IfcColumn" as column
7  where
8      column in filter("Spatial Query Language", "query" =
9          "SELECT sc2
10          FROM struct_comp sc1, struct_comp sc2
11          WHERE sc1.id='58' AND sc2.above(VALUE(sc1))=1"
12     )
```

Durch die Nutzung externer Elementarfilter ist es sogar möglich, fremde textuelle Filtersprachen einzubinden. Quelltext 4.13 zeigt die hypothetische Nutzung der *Spatial Query Language*, vorgestellt von Borrmann (2007, Beispiel S. 224). Die Ermittlung der Lagebeziehung von Bauteilen kann per MMQL nicht sinnvoll durchgeführt werden – sie würde daher an den hierfür spezialisierten Filterservice delegiert.

4.3.3. Linkauswertung

Im vorangegangenen Abschnitt wurden Methoden zur Filterung von Elementarmodellen beschrieben. Diese Verfahrensweisen sind aus dem Bereich der singulären Fachmodelle bekannt.

Im Kontext von IFC demonstrieren Katranuschkov et al. (2010) die prinzipielle Idee, auch die Kriterien dritter, verlinkter Fachmodelle zur Filterung *eines* betrachteten Produktdatenmodells heranzuziehen. Das Multimodell-Konzept ermöglicht darüber hinaus das Filtern eines domänen-, modell- und datenformatübergreifenden Informationsraumes im Gesamten. Dabei kann, zusätzlich zu den Elementarmodellfiltern, die Einschränkung relevanter Elemente über die Auswertung der Linkmodelle erfolgen.

Strukturelle Linksemantik

Die Systematik der Linkauswertung folgt dabei der *strukturellen Linksemantik* – der prinzipiellen Deutungsmöglichkeit vorhandener und nicht vorhandener mehrwertiger Links auf Ebene des Generischen Multimodells. Die strukturelle Linksemantik gliedert sich in die unabhängigen Aspekte *Elementkombination* und *Linkinterpretation*.

Die Elementkombination legt fest, welche Elemente der Elementarmodelle kombiniert werden sollen und wie diese Kombination zu erfolgen hat. Diese Operation kann in Analogie zur relationalen Algebra wie eine `JOIN`-Anweisung betrachtet werden. Dabei entsprechen die Multimodell-Elemente den Relationen und die Linkmodelle den Mappingtabellen.

1. Der Modus *Natural* kombiniert demzufolge alle Elemente nach Maßgabe der vorhandenen Links in den Linkmodellen. Das bedeutet, dass nur diejenigen Elemente weiterverarbeitet werden, welche auch verlinkt sind (vgl. Beispiel 12 auf Seite 114).
2. Im Modus *Right Outer* werden alle Elemente des rechten Arguments weiterverarbeitet. Elemente, die nicht verlinkt sind, werden auf der linken Seite mit `null` aufgefüllt (vgl. Beispiel 13).
3. Der Modus *Full Cross Product* ignoriert vorhandene Links und bildet immer ein vollständiges Kreuzprodukt aller Elemente (vgl. Beispiel 14).

Die Anzahl weiterzuverarbeitender Elemente wächst potentiell von Modus 1 nach 3. Es wird vermutet, dass das Verhalten im Modus *Natural* der intuitiven Erwartungshaltung eines Nutzers entspricht – in der MMQL-Syntax wird daher auf einen speziellen Modifikator verzichtet.

Die Linkinterpretation bestimmt, welche Bedeutung ein gegebener Link für eine korrespondierende Menge von Multimodell-Elementen hat. Die Notwendigkeit dieser Betrachtung ergibt sich hauptsächlich aus der potentiellen Mehrwertigkeit der Links. Ein Link kann n_e Elemente aus m_{em} Elementarmodellen verknüpfen. Im Zuge einer Abfrage können jedoch einige dieser Elemente ungültig werden, z. B. durch Einschränkungen bei Elementkombinationen oder aufgrund von Elementarmodellfiltern.

1. Im Modus *Strict* gelten Links nur dann als existent, wenn alle ihre Elemente gültig sind. Es werden also entweder *alle* Elemente eines Links oder *keines* weiterverarbeitet (vgl. Beispiel 15).
2. Im Modus *Standard* gelten Links als dann existent, wenn mindestens zwei Elemente aus mindestens zwei Elementarmodellen des Links gültig sind. Es können also auch dann Elemente weiterverarbeitet werden, wenn Teile eines Links entfallen (vgl. ebenfalls Beispiel 12).

3. Im Modus *Transitive* werden Links transitiv ausgewertet. Ein Link wird dabei um all diejenigen Elemente erweitert, die von seinen Elementen aus über andere Links erreichbar sind. Für die Weiterverarbeitung von Elementen gelten die gleichen Regeln wie im Modus Standard. Dieser Modus liefert nur dann weitere Ergebnisse, wenn im Linkmodell mehr Element-Typen vorhanden sind als in der Anfrage benutzt werden (vgl. Beispiel 16).

Die Anzahl weiterzuverarbeitender Elemente wächst potentiell von Modus 1 nach 3. Für den Aspekt der Linkinterpretation wird vermutet, dass das Verhalten im Modus *Standard* der intuitiven Erwartungshaltung eines Nutzers entspricht, weswegen auch hier in der MMQL-Syntax auf einen Modifikator verzichtet wurde.

Tabelle 4.2 zeigt die möglichen Kombinationen der Modi *Elementkombination* und *Linkinterpretation* anhand der Syntax der Modifikatoren für den `linkedwith`-Operator – dem MMQL-Sprachelement zur Linkauswertung. Entsprechend sind `linkedwith`-Anweisungen wie folgt definiert:

Syntaxdiagramm 4.31: LinkedWithSpec

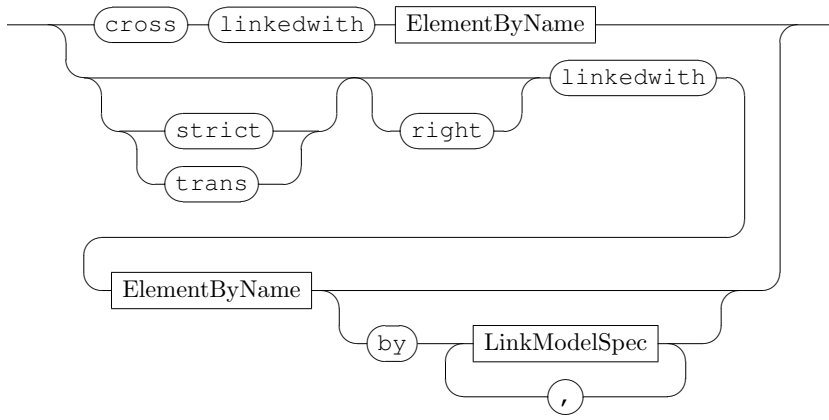
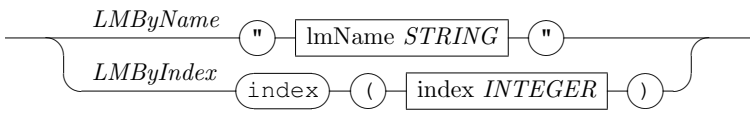


Tabelle 4.2.: Strukturelle Linksemantik: Modifikatoren der MMQL-linkedwith-Anweisung für mögliche Zusammenstellungen von Elementkombination und Linkinterpretation

Linkinterpretation / Elementkombination	Strict	Standard	Transitive
Natural	`strict linkedwith`	`linkedwith`	`trans linkedwith`
Right Outer	`strict right linkedwith`	`right linkedwith`	`trans right linkedwith`
Full Cross Product	`cross linkedwith`		

Die Elementkombinationen *Natural* und *Right Outer* operieren auf Linkmodellen. Dabei werden alle Linkmodelle des Multimodells herangezogen, welche mit der konkreten Anfrage korrespondieren. Alternativ kann über das Schlüsselwort `by` eine durch Komma separierte Liste von zu verwendenden Linkmodellen angegeben werden. Die Spezifikation erfolgt dabei über den Namen des Linkmodells (gegeben durch den Metadaten-Eintrag `mmaa.linkmodel.name`) oder über dessen Index (Positionsnummer im Multimodell):

Syntaxdiagramm 4.32: LinkModelSpec



Beispiele

Für die folgenden Beispiele wird ein simplifiziertes Multimodell in Anlehnung an Beispiel 1 betrachtet. Es besteht aus jeweils vier Elementen des Typs LV-Position (Item, $EM1$), Wand (IfcWall, $EM2$) und Vorgang (Activity, $EM3$) mit den folgenden, vereinfachten IDs:

Item ($EM1$)
item1
item2
item3
item4

IfcWall ($EM2$)
wall1
wall2
wall3
wall4

Activity ($EM3$)
activity1
activity2
activity3
activity4

Außerdem soll das Linkmodell $LM1$ folgende Links enthalten (in kompakter Darstellung):

Link	$EM1$	$EM2$	$EM3$
$L1$	item1	wall1	activity1
$L2$	item2	wall1, wall2	activity1
$L3$	item3	wall3	activity3, activity4
$L4$		wall4	activity4
$L5$	item4		activity4

Beispiel 12

Der folgende Ausschnitt einer MMQL-Abfrage zeigt die `linkedwith`-Operatoren zur Elementkombination im Modus *Natural* sowie zur Linkinterpretation im Modus *Standard* für die Elemente *Item* und *IfcWall*:

```
7  from
8      "LV 1.X83"."Item"
9  linkedwith
10     "BW.ifc"."IfcWall"
```

Das Ergebnis wertet alle Linkmodelle – hier das einzige Linkmodell $LM1$ – aus. Das Ergebnis enthält vier Zeilen mit den Kombinationen der Elemente vom Typ Item und IfcWall, welche in $LM1$ ein- oder mehrwertig verlinkt sind. Zum besseren Verständnis sind die ursächlichen Links für jede Ergebniszeile angegeben:

Item ($EM1$)	IfcWall ($EM2$)	(Link)
item1	wall1	($L1$)
item2	wall1	($L2$)
item2	wall2	($L2$)
item3	wall3	($L3$)

Beispiel 13

Es wird die gleiche Abfrage wie in Beispiel 12 gestellt. Allerdings sollen sämtliche Wände im Ergebnis enthalten sein:

```
7  from
8      "LV 1.X83"."Item"
9  right linkedwith
10     "BW.ifc"."IfcWall"
```

Dementsprechend enthält das Ergebnis nun zusätzlich `wall4`:

Item ($EM1$)	IfcWall ($EM2$)	(Link)
item1	wall1	($L1$)
item2	wall1	($L2$)
item2	wall2	($L2$)
item3	wall3	($L3$)
null	wall4	—

Beispiel 14

Dieser Ausschnitt einer MMQL-Abfrage zeigt die `linkedwith`-Operatoren zur Bildung eines Kreuzproduktes über alle Elemente:

```
7  from
8      "LV 1.X83"."Item"
9  cross linkedwith
10     "BW.ifc"."IfcWall"
11 cross linkedwith
12     "Vorgangsmodell 1.xml"."Activity"
```

Das Ergebnis ist unabhängig von der Existenz oder der Belegung irgendwelcher Linkmodelle. Das ResultSet enthält $4^3 = 64$ Zeilen. Jede Zeile enthält eine der möglichen Kombinationen der Multimodell-Elemente:

Item ($EM1$)	IfcWall ($EM2$)	Activity ($EM3$)
item1	wall1	activity1
item2	wall1	activity1
item3	wall1	activity1
item4	wall1	activity1
item1	wall1	activity2
item2	wall1	activity2
⋮	⋮	⋮
item3	wall4	activity4
item4	wall4	activity4

Beispiel 15

In diesem Beispiel werden die Elemente *IfcWall* und *Activity* abgefragt. Dabei sollen jedoch nur vollständige Links in das Ergebnis aufgenommen werden:

```
7  from
8      "BW.ifc"."IfcWall"
9  strict linkedwith
10     "Vorgangsmodell 1.xml"."Activity"
```

Das Ergebnis enthält lediglich *eine* Zeile, da nur in Link $L4$ ausschließlich Wände und Vorgänge verlinkt sind:

IfcWall ($EM2$)	Activity ($EM3$)	(Link)
wall4	activity4	($L4$)

Beispiel 16

Hier wird die gleiche Abfrage wie in Beispiel 12 gestellt. Allerdings sollen auch diejenigen Wände im Ergebnis enthalten sein, welche über andere Links und Elemente transitiv erreichbar sind:

```
7  from
8      "LV 1.X83"."Item"
9  trans linkedwith
10     "BW.ifc"."IfcWall"
```

Das Ergebnis enthält 4 zusätzliche Zeilen (mit ✓ markiert), welche durch die Interpretation der verlinkten Elemente aus $EM3$ entstehen. So ist nun bspw. die Wand `wall4` enthalten, da diese in $L4$ mit dem Element `activity4` verlinkt ist. Das Element `activity4` ist sowohl per $L3$, als auch per $L5$ erreichbar, weswegen schlussendlich `wall4` im Ergebnis berücksichtigt werden kann:

Item ($EM1$)	IfcWall ($EM2$)	(Neu)	(Links)
item1	wall1		$(L1)$, $(L1 \curvearrowright L2)$
item1	wall2	✓	$(L1 \curvearrowright L2)$
item2	wall1		$(L2)$, $(L2 \curvearrowright L1)$
item2	wall2		$(L2)$
item3	wall3		$(L3)$
item3	wall4	✓	$(L3 \curvearrowright L4)$
item4	wall3	✓	$(L5 \curvearrowright L3)$
item4	wall4	✓	$(L5 \curvearrowright L4)$

4.3.4. Elementarmodellübergreifende Property-Kriterien

Die bisher behandelten Property-Kriterien operierten als Teil des Elementarmodellfilterns jeweils auf *einem* Elementarmodell. Elementarmodellübergreifende Property-Kriterien operieren auf Werten, welche aus unterschiedlichen Elementarmodellen stammen. Die verschiedenen Elementarmodelle ergeben sich dabei aus `linkedwith`-Anweisungen – entweder durch Auswertung von Linkmodellen oder durch explizite Kombination mittels Kreuzprodukt. Elementarmodellübergreifende Property-Kriterien eignen sich zur Kontrolle vergleichbarer Größen in unterschiedlichen Modellen.

Quelltext 4.14: Beispiel für elementarmodellübergreifende Property-Kriterien in MMQL

```
1  use editor "ausschreibung_fr.mmaa"
2
3  select
4      wall.id,
5      wall ? ("property", "Betonklasse") as betonklasse,
6      item.id,
7      item ? ("text", "detail") as detailtext
8  from
9      "BW.ifc"."IfcWall" as wall
10 cross linkedwith
11     "LV 1.X83"."Item" as item
12 where
13     detailtext like "Wand"
14     and
15     detailtext like betonklasse
```

Beispiel 17

Quelltext 4.14 zeigt eine Abfrage für das Multimodell aus Beispiel 1. Die Festigkeitsklasse ist dabei im IFC-Modell für Wände aus Beton im IFC-Property *Betonklasse* hinterlegt. Es sollen alle Wände und alle LV-Positionen ausgegeben werden, welche im Langtext eine identische Beton-Festigkeitsklasse aufweisen (vgl. Zeile 15). Damit nur LV-Positionen aufgenommen werden, welche auch zu Wänden

korrespondieren, wird in Zeile 13 zusätzlich verlangt, dass im Langtext das Wort „Wand" vorkommt.

Quelltext 4.15 zeigt einen Auszug aus dem ResultSet. Ab Zeile 11 wurde der Langtext zur besseren Übersicht gekürzt. Insgesamt enthält das Ergebnis 3158 Einträge, welche jeweils eine mögliche Kombination einer Wand mit einer LV-Position gleicher Beton-Festigkeitsklasse repräsentieren. Im IFC-Modell gibt es 1784 Wände, im Leistungsverzeichnis 52 Positionen. Das Kreuzprodukt enthielte ohne die Angabe weiterer Kriterien $1784 \times 52 = 92.768$ Zeilen.

Quelltext 4.15: ResultSet für elementarmodellübergreifende Property-Kriterien im Format CSV (Auszug)

```
1  0zWdR7b7rC_OrFgjNh5umS,C 30/37,1.2.1.10.,"Ortbeton der Wand,
2  aus Beton DIN 1045,
3  Wände in allen Bereichen,
4  auch von wandartigen Trägern,
5  Festigkeitsklasse: C 30/37
6  Expositionsklasse: XC1
7  Wandstärke bis 30 cm
8  Wandhöhe entsprechend Ebene/Einbaubereich
9  gemäß Plänen der Anlage.
10 Einbau in Ebene U2 bis E04."
11 0Ijepo4RP1tRIw4WFwuViG,C 30/37,1.2.1.10.,"Ortbeton [...] C 30/37 [..]"
12 11zL44e6n3yuvtvW$0VkiF,C 30/37,1.2.1.10.,"Ortbeton [...] C 30/37 [..]"
13 27z_Y2OOv6Y874AZmjvVtW,C 30/37,1.2.1.10.,"Ortbeton [...] C 30/37 [..]"
14 2WrV2XGDH4Q8DY$0OIQ8NS,C 30/37,1.2.1.10.,"Ortbeton [...] C 30/37 [..]"
15 3YQexFb7b9qPdW5_8kSzzY,C 30/37,1.2.1.10.,"Ortbeton [...] C 30/37 [..]"
```

Aufgrund der Möglichkeit, Regeln zwischen unterschiedlichen Elementarmodellen auch ohne die Nutzung von Linkmodellen aufstellen zu können, sind elementarmodellübergreifende Property-Kriterien die Basis zur programmatischen Erzeugung von Links.

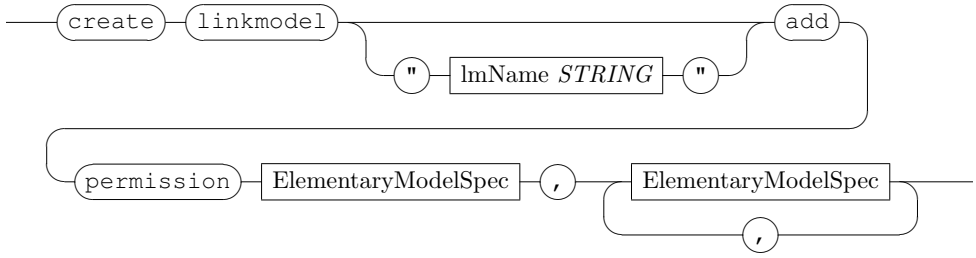
4.4. Linkmanipulation

4.4.1. Linkmodell-Operationen

Die Multi-Model Query Language soll domänenübergreifende Informationsräume nicht nur *abfragen*, sondern diese auch *erstellen* und *ändern* können. Da davon ausgegangen wird, dass Elementarmodelle durch externe Fachanwendungen bearbeitet werden, beschränkt sich die Anwendung manipulativer Operationen auf Linkmodelle und deren Links.

Die wesentliche strukturelle Eigenschaft von Linkmodellen ist die Festlegung der Elementarmodelle, deren Elemente im Linkmodell referenziert werden dürfen. Es handelt sich dabei sinngemäß um eine Erlaubnis (Permission) zur Teilnahme an Linkoperationen. Daher ist die Angabe der zulässigen Elementarmodelle Teil jeder administrativen Linkmodell-Anweisung. So ist die Anweisung zum Erstellen eines neuen Linkmodells wie folgt definiert:

Syntaxdiagramm 4.33: CreateLM



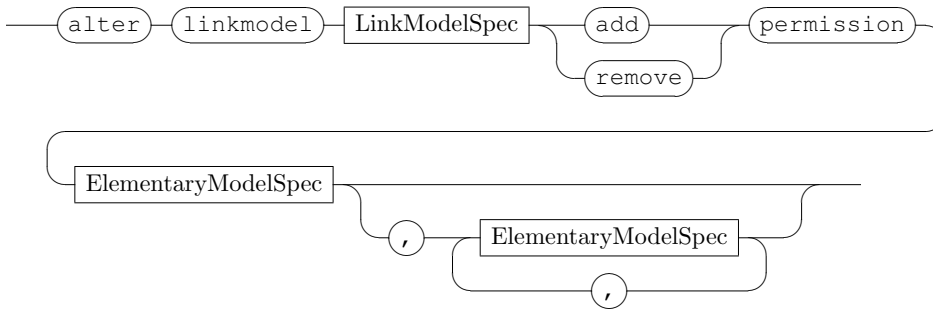
Nach den Schlüsselwörtern **create linkmodel** kann optional der Name des Linkmodells angegeben werden. Dieser wird im Meta-Datum mit dem Schlüssel `mmaa.linkmodel.name` hinterlegt. Anschließend müssen über die Schlüsselwörter **add permission** zwei oder mehr Elementarmodelle spezifiziert werden, deren Elemente im neuen Linkmodell teilnehmen dürfen. Beispielsweise erzeugt die Anweisung in Quelltext 4.16 das neue Linkmodell $LM2$, mit der Berechtigung für Elemente aus dem Leistungsverzeichnis und dem Bauwerksmodell, für das Multimodell aus Beispiel 1.

Quelltext 4.16: Beispiel für die Erzeugung eines neuen Linkmodells in MMQL

```
1 use file "C:\Container\ausschreibung.mmaa"
2
3 create linkmodel "LM2"
4     add permission "LV 1.X83", "BW.ifc"
```

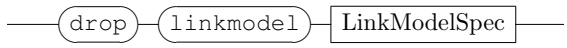
Die administrative Anweisung zum Hinzufügen (add) oder Entfernen (remove) berechtigter Elementarmodelle zu oder von einem bestehenden Linkmodell ist analog definiert:

Syntaxdiagramm 4.34: AlterLM



Die Anweisung **drop linkmodel** löscht das gesamte Linkmodell mit allen Links:

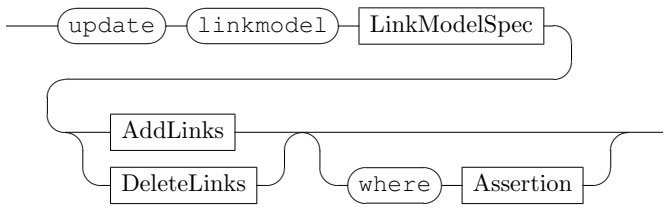
Syntaxdiagramm 4.35: DeleteLM



4.4.2. Link-Operationen

Link-Operationen sind Änderungsoperationen (Update) eines Linkmodells. Sie fügen Links hinzu (AddLinks) oder löschen Links (DeleteLinks) unter bestimmten Bedingungen. Anweisungen zum Ändern von Links sind folgendermaßen definiert:

Syntaxdiagramm 4.36: UpdateLM



Linkerzeugung

Die Erzeugung von Links geschieht für mindestens zwei Element-Typen aus mindestens zwei Elementarmodellen[9]. Dazu wird das Kreuzprodukt über alle Elemente der Typen gebildet und für jede Kombination das angegebene Kriterium ausgewertet. Evaluiert das Kriterium zu `true`, wird ein Link mit den Elementen der aktuellen Kombination erzeugt und in das Linkmodell eingetragen.

Auch die Systematik der Linkerzeugung folgt den Parametern der strukturellen Linksemantik. Die Erstellung der Links kann dabei über das *Zulassen von Duplizität*, *Interpretation der Arität* sowie über die *Zuweisung der Kardinalitäten* beeinflusst werden.

Das Zulassen von Duplizität entscheidet darüber, ob Links mit gleichem Inhalt in einem Linkmodell erlaubt sind. Dieser Fall kann insbesondere dann auftreten, wenn das Linkmodell sequentiell mit mehreren Anweisungen befüllt wird. Link-Duplikate können erwünscht sein, bspw. mit unterschiedlichen Metadaten. Die entsprechenden Modi sind *duplicate* (Duplikate zulassen) oder *distinct* (Duplikate verbieten).

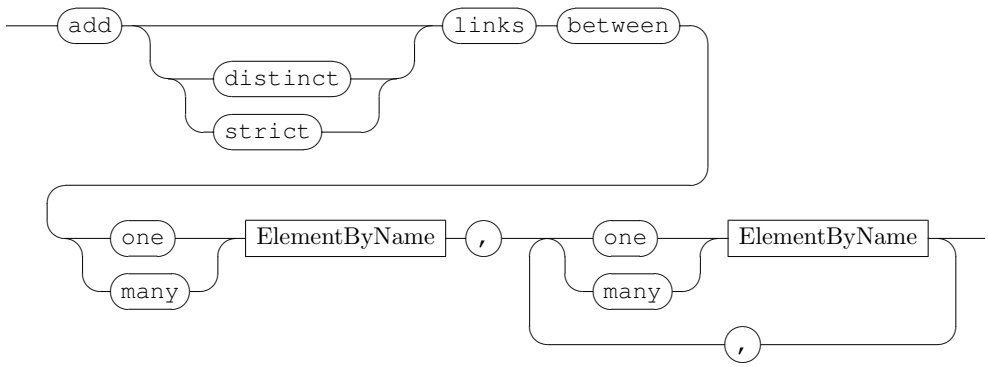
Die Interpretation der Arität legt fest, ob einem Linkmodell Links hinzugefügt werden dürfen, welche nicht für jedes zugelassene Elementarmodell auch mindestens *ein* belegtes Element besitzen. Die Modi sind *relax* (Arität wird als *zulässige* Größe interpretiert, unvollständige Links zulassen) und *strict* (Arität wird als *verbindliche* Größe interpretiert, unvollständige Links unterbinden). Der Modus *strict* soll den Modus *distinct* implizieren.

[9] vgl. auch Abschnitt 3.3.5 auf Seite 63

Die Zuweisung der Kardinalitäten bestimmt die individuelle Multiplizität (1 oder n) für jeden Element-Typ innerhalb eines Links. Durch die Kreuzproduktbildung entstehen zunächst Links der Kardinalität $1:1:\ldots:1$; d. h. pro Element-Typ gibt es maximal *ein* Element im Link. Je nach gewünschter Kardinalität können die Links schrittweise über bspw. $n:1:\ldots:1$ bis hin zu $n:n:\ldots:n$ zusammengefasst werden.

Unter Berücksichtigung dieser Parameter sind Anweisungen zur Linkerzeugung wie folgt definiert:

Syntaxdiagramm 4.37: AddLinks



Nach dem Schlüsselwort **`add`** folgt ein kombinierter Modifikator für Duplizität und Arität:

- die Modus-Kombination *duplicate* / *relax* erhält keinen Modifikator
- die Modus-Kombination *distinct* / *relax* erhält den Modifikator `distinct`
- die Modus-Kombination *distinct* / *strict* erhält den Modifikator `strict`

Nach den Schlüsselwörtern **`links between`** werden die zwei oder mehr zu verlinkenden Element-Typen durch Komma getrennt angegeben. Ihnen wird die Zuweisung der Kardinalität vorangestellt – `one` für *ein* Element je Link, `many` für *ein* oder mehr Elemente je Link.

Beispiel 18

Die Aufgabenstellung aus Beispiel 17 soll nun dazu verwendet werden, Links zwischen den identifizierten Elementen zu erstellen. Quelltext 4.17 zeigt die dazu notwendigen Anweisungen. Im Linkmodell dürfen Elemente aus allen drei Elementarmodellen enthalten sein. Es werden jedoch nur Links zwischen Item und IfcWall erzeugt. Daher muss die Interpretation der Arität im Modus *relax* erfolgen. Da das Linkmodell neu erstellt wurde (vgl. Zeile 3 f.) sind bisher keine Links vorhanden. Auf die Überprüfung von Dubletten kann daher verzichtet werden (Modus *duplicate*). Somit muss für die Linkanweisung kein Modifikator gesetzt werden (vgl. Zeile 7). In Zeile 11 erhält die Property-Accessor-Spezifikation den Alias `detailtext`, damit diese in Zeile 13 wiederverwendet werden kann.

Da die Links für beide Element-Typen einwertig (1 : 1) erstellt werden sollen, werden 3158 Links erzeugt. Dies entspricht der Anzahl der Ergebniszeilen aus Beispiel 17. Das neue Linkmodell *LM2* stellt sich demnach wie folgt dar (Auszug, zusätzliche Angabe der Linknummer):

$EM1$ (Item)	$EM2$ (IfcWall)	$EM3$	(Link)
1.2.1.10.	0$2IQ5l1v5J9lalNJ3kSZk	—	($L1$)
1.2.1.20.	0$2IQ5l1v5J9lalNJ3kSZk	—	($L2$)
1.2.1.10.	0$dVsoebz8WB$XoCJbMiD4	—	($L3$)
⋮	⋮	⋮	⋮
1.2.1.20.	3zuk6PAmX1HRpY4UDfRnAy	—	($L3158$)

Sollen in jedem Link mehrere LV-Positionen nach Wänden gruppiert erlaubt sein, so ist die Kardinalität für den Element-Typ *Item* auf `many` zu setzen (vgl. Zeile 8):

```
7   add links between
8       many "LV 1.X83"."Item" as item,
9       one "BW.ifc"."IfcWall" as wall
```

Entsprechend enthält das Linkmodell *LM2* 1579 Links, welche sich wie folgt ergeben (Auszug):

$EM1$ (Item)	$EM2$ (IfcWall)	$EM3$	(Link)
1.2.1.10., 1.2.1.20.	0$2IQ5l1v5J9lalNJ3kSZk	—	($L1$)
1.2.1.10., 1.2.1.20.	0$dVsoebz8WB$XoCJbMiD4	—	($L2$)
1.2.1.10., 1.2.1.20.	0fhU7jPFB9KwEQGZe595	—	($L3$)
⋮	⋮	⋮	⋮
1.2.1.10., 1.2.1.20.	3zuk6PAmX1HRpY4UDfRnAy	—	($L1579$)

Die Gruppierung aller Wände, welche zu einer LV-Position gehören, kann innerhalb eines Links durch Setzen der Kardinalität für den Element-Typ *IfcWall* auf `many` erreicht werden:

```
7   add links between
8       one "LV 1.X83"."Item" as item,
9       many "BW.ifc"."IfcWall" as wall
```

Das resultierende Linkmodell enthält dadurch 2 Links. Jeder LV-Position sind 1579 Wände zugeordnet:

$EM1$ (Item)	$EM2$ (IfcWall)	$EM3$	(Link)
1.2.1.10.	0$2IQ5l1v5J9lalNJ3kSZk, ... , 3zuk6PAmX1HRpY4UDfRnAy	—	($L1$)
1.2.1.20.	0$2IQ5l1v5J9lalNJ3kSZk, ... , 3zuk6PAmX1HRpY4UDfRnAy	—	($L2$)

Quelltext 4.17: Beispiel für die Erzeugung neuer Links in MMQL

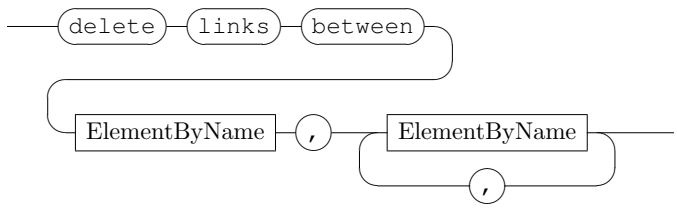
```
use file "C:\Container\ausschreibung.mmaa"

create linkmodel "LM2"
    add permission "LV 1.X83", "BW.ifc", "Vorgangsmodell 1.xml"

update linkmodel "LM2"
    add links between
        one "LV 1.X83"."Item" as item,
        one "BW.ifc"."IfcWall" as wall
    where
        item?("text", "detail") as detailtext like "Wand"
        and
        detailtext like wall?("property","Betonklasse")
```

Löschen von Links

Beim Löschen von Links werden sequentiell alle Links des Linkmodells untersucht. Wenn das angegebene Kriterium für die Elemente des untersuchten Links wahr ist, wird dieser gelöscht. Bei mehrwertigen Links – welche mehrere Elemente eines Typs beinhalten – ist es ausreichend, wenn das Kriterium für mindestens *eines* dieser Elemente wahr ist, um den gesamten Link zu löschen. Wird kein Kriterium angegeben, so werden alle Links gelöscht, welche mindestens *ein* Element der angegebenen Typen beinhalten. Es werden nur komplette Links gelöscht. Es ist nicht möglich, einzelne verlinkte Elemente aus einem Link zu entfernen.

Die Anweisung zum Löschen von Links ist daher durch die Schlüsselwörter **`delete links between`** sowie die Angabe von mindestens zwei Element-Typen definiert:

Syntaxdiagramm 4.38: DeleteLinks



Beispiel 19

Aus dem in Beispiel 18 neu erzeugten Linkmodell $LM2$ (Kardinalität $1:1$) sollen nun diejenigen Links entfernt werden, welche sich auf Wände mit der

Feuerwiderstandsklasse *F90* beziehen. Quelltext 4.18 zeigt die dazu notwendige Anweisung. Nach deren Ausführung enthält *LM2* nur noch 930 Links.

Quelltext 4.18: Beispiel für das Löschen von Links in MMQL

```
use file "C:\Container\ausschreibung.mmaa"

update linkmodel "LM2"
    delete links between
        "LV 1.X83"."Item",
        "BW.ifc"."IfcWall" as wall
    where
        wall?("property","FireRating") == "F90"
```

4.5. Resümee

Die Multi-Model Query Language MMQL ist eine Programmiersprache zur Erschließung von Multimodellen. Ihre Bedeutung entspricht derjenigen von SQL für relationale Datenbanken oder XPath für XML-Dokumente. Indem die Sprache die wesentlichen Konzepte von Multimodellen aufgreift, entbindet sie Anwender von der Aufstellung und Implementierung eigener Strategien zum Zugriff auf beliebige Multimodelle. Entsprechend dieser technologischen Abstraktionsebene gehören die Entwickler spezialisierter, interdisziplinärer Baufachanwendungen zur Anwender-Zielgruppe der Sprache.

Mit den Mitteln der MMQL ist es *nicht* möglich, den Inhalt von Elementarmodellen zu ändern. Dies bleibt alleinige Aufgabe der domänenspezifischen oder -übergreifenden Baufachanwendungen. Dedizierte Aufgabe von Multimodellen ist die Bereitstellung eines Datenaustauschformats zur Übermittlung strukturierter, interdisziplinärer Nachrichten auf Basis originaler Datenformate. In Verbindung mit einer korrespondierenden Ablaufumgebung (die s. g. Multimodell-Engine, vgl. auch Kapitel 5 und 6) stellt die MMQL ein Werkzeug zur Erzeugung und zur Auswertung solcher Nachrichten dar. Die wichtigsten Aufgaben der MMQL sind daher die Erstellung von Links sowie das Multimodell-Filtern. Dabei kommen die neuartigen Aspekte der *mehrwertigen Linkerzeugung* sowie der *strukturellen Linksemantik* – bestehend aus *Elementkombination* und *Linkinterpretation* – zur Anwendung.

Eine besondere Herausforderung bei der sprachlichen Konzeption stellt der Zugriff auf die Originaldaten der Elementarmodelle dar. Trotz der Verwendung potentiell beliebiger, und – aus Sichtweise der MMQL – unbekannter Datenformate, muss die entsprechende Syntax einheitlich und generisch sein. Diese Aufgabe wurde durch Einführung des *Ideellen Elementarmodells* gelöst. Das Ideelle Elementarmodell ist eine virtuelle Struktur, welche die Konzepte eines reflexiven Datenzugriffs abbildet. Dieser wird als geforderte Schnittstelle im jeweiligen Parser eines Datenformates realisiert. Auf diese Weise muss lediglich die Multimodell-Engine für neue Datenformate erweitert werden – nicht jedoch die MMQL als Sprache.

Um eine möglichst große Anzahl baufachlicher Problemstellungen mit dem Multimodell und der MMQL lösen zu können, muss bereits die Erschließung der jeweiligen Elementarmodelle exakt,

prägnant ausdrückbar und potentiell vollständig erfolgen können. Gleichzeitig erfolgt hierbei auch ein Übergang von baufachlichen, semantisch reichhaltigen zu technischen, strukturellen Belangen. Das Ideelle Elementarmodell bietet daher die Möglichkeit zur Implementierung individueller, semantischer Zugriffsfunktionen beim Übergang von Elementen zu Properties an (s. g. Property Accessors). In der MMQL können darüber hinaus beliebige, externe Elementarmodellfilter ausgeführt werden. Mit der Erweiterung der Multimodell-Engine um solche Komponenten, können letztendlich baufachliche Problemstellungen mit domänennäheren Ausdrücken formuliert werden.

Kapitel 5.

Interpretation von MMQL-Anweisungen

Wahrheit heißt Übereinstimmung des Begriffs mit seiner Wirklichkeit.

(Georg Wilhelm Friedrich Hegel)

Um multimodellbasierte Informationsräume mithilfe der MMQL praktisch erschließen zu können, muss die Sprache auf einem Computersystem für reale Daten ausführbar sein. Daher wird in diesem Kapitel die Arbeitsweise eines MMQL-Interpreters vorgestellt. Mit dessen Hilfe werden MMQL-Anweisungen in konkrete Multimodell-Operationen umgewandelt. Die dazu notwendigen Verfahrensschritte stellen auch eine ergänzende Beschreibung der einzelnen MMQL-Befehle dar – denn das Verhalten des Interpreters ist Teil der Semantik der deklarativen Sprache.

Abschnitt 5.1 klassifiziert die Sprache MMQL, beschreibt Entstehung und Aufbau des abstrakten Syntaxbaumes und erläutert die grundlegenden Erweiterungen der MMQL-Engine, welche notwendig sind, um die Sprache für konkrete Elementarmodell-Typen anzuwenden. Abschnitt 5.2 zeigt die einzelnen Schritte zur Ermittlung von Multimodell-Views. Dabei werden auch grundlegende Methoden wie das Auflösen von Elementen oder die Evaluation von Kriterien erläutert, welche zum Erstellen bzw. Löschen von Links in den Abschnitten 5.3 und 5.4 wiederverwendet werden. Eine Diskussion in Abschnitt 5.5 erörtert die Aspekte Effizienz und Korrektheit für die Methoden des MMQL-Interpreters.

5.1. Grundlagen der Ausführung der Sprache

5.1.1. Erhöhung der Formalisierung der Semantik

In Kapitel 4 wurde die Syntax der *Multi-Model Query Language* (MMQL) *formal* definiert[1]. Die Semantik der Sprache wurde lediglich *verbal* beschrieben. Für eine praxisnahe Anwendbarkeit der MMQL muss die Semantik jedoch einen höheren Grad der Formalisierung erlangen. Dies erfolgt auf zwei Wegen:

- durch eine Referenzimplementierung[2]
- durch die Funktionsbeschreibung des Interpreters

Tabelle 5.1 zeigt die Klassifikation der Programmiersprache MMQL. Zur Lösung eines Multimodell-Problems müssen nicht die dafür notwendigen Schritte angegeben werden – vielmehr wird das gewünschte Resultat beschrieben. Dazu werden auch Randbedingungen in Form von Kriterien verwendet. MMQL weist somit Eigenschaften der Sprachen der vierten und fünften Generation auf. Abstraktionsgrad und Programmierparadigma von MMQL sind deklarativ. Nach Henning et al. (2007) ist das deklarative Programmierparadigma dem funktionalen sowie dem logischen Paradigma zuzuordnen. Das Ausführungsschema der MMQL ist interpretierend. Erst zur Laufzeit wird durch den Interpreter aus den MMQL-Anweisungen und den zu nutzenden Daten (Multimodell) ein Lösungsweg erarbeitet und ausgeführt.

MMQL-Anweisungen zielen auf ein gewünschtes ResultSet oder einen neuen Zustand im Multimodell ab. Die Formulierung solcher Anweisungen orientiert sich somit auch an der Funk-

Tabelle 5.1.: Klassifikation der Programmiersprache MMQL (**Fettschrift**). Aspekte und Klassen in Anlehnung an Henning et al. (2007).

Generation	Programmier-paradigma	Abstraktionsgrad	Ausführungsschema
1. Binäre Maschinensprachen	Imperativ / Prozedural	Maschinensprache	Compilierend
2. Assemblersprachen	**Funktional / Applikativ**	Assemblersprache	**Interpretierend**
3. Problemorientierte, imperative / prozedurale Sprachen	**Logisch / Prädikativ**	Höhere Programmiersprache	Mischform
4. 4GL-Sprachen (anwendungs-orientiert)	Objektbasiert	Objektorientierte Programmiersprache	
5. Deklarative Programmier-sprachen	Objektorientiert	**Deklarative Sprache**	

[1] in Anhang A.3 auf Seite 188 ist zusätzlich die kompakte Syntaxdefinition der MMQL in Erweiterter Backus-Naur-Form (EBNF) wiedergegeben.
[2] vgl. Kapitel 6 auf Seite 159

tionsweise des Interpreters, welcher schlussendlich die Resultate produziert. Die Funktionsweise des Interpreters ist demnach Teil der Semantik der Sprache.

5.1.2. Der abstrakte Syntaxbaum

Die Abarbeitung von MMQL-Anweisungen durch den MMQL-Interpreter orientiert sich an Standardverfahren der Informatik. Einen Überblick zu Compiler- und Interpreterbau geben u. a. Mak (2009); Parr (2010); Cooper & Torczon (2011). Die Abarbeitung von SQL-Befehlen innerhalb relationaler Datenbank-Managementsysteme beschreiben bspw. Saake et al. (2005).

Im ersten Arbeitsschritt wandelt ein Parser den MMQL-Quelltext in einen abstrakten Syntaxbaum um. Diese Operation ist nur für valide Quelltexte möglich. Der abstrakte Syntaxbaum enthält in den Knoten Operatoren. Deren Kindelemente sind die zugehörigen Operanden und Modifikatoren. Kindelemente können selbst wieder Operatoren sein. Die Operatorentypen und ihre hierarchische Reihenfolge im Baum orientiert sich an den Produktionsregeln der Syntaxdefinition sowie den Vorrangregeln (Gewicht)[3]. Im Ergebnis entsteht eine baumartig navigierbare Abbildung des konkreten MMQL-Quelltexts. Abbildung 5.1 auf der nächsten Seite zeigt exemplarisch den abstrakten Syntaxbaum für den MMQL-Quelltext 4.6 auf Seite 105. Zur besseren Übersicht wurden die Knoten der Property-Spezifikationen und des Textvergleiches zusammengefasst.

Im zweiten Arbeitsschritt wird der abstrakte Syntaxbaum dem Interpreter übergeben. Dieser arbeitet den Baum knotenweise ab, beginnend beim Wurzelknoten MMQL. Je nach Operator werden Daten aus den Elementarmodellen gelesen, gespeichert oder verarbeitet. Komplexe und verschachtelte Operatoren delegieren dabei die Abarbeitung zunächst an ihre Kindelemente. Das übergeordnete Ziel ist dabei die Erzeugung und Ausführung einer geordneten Sequenz von Multimodell-Elementaroperationen[4], welche die Lösung der ursprünglichen MMQL-Anweisung sind.

Die formale Syntaxdefinition der MMQL erlaubt allgemeine Aussagen über MMQL-Anweisungen. So besteht jede mögliche MMQL-Anweisung `mmql`[5] aus *einer* Multimodell-Spezifikation `mms` und mindestens einem Statement `stmt` (vgl. EBNF Quelltext A.3 Zeile 1). Damit gilt:

$$\begin{aligned} \texttt{MMQL} &:= \text{Unendliche Menge valider MMQL-Anweisungen} \\ \texttt{STMT} &:= \text{Unendliche Menge valider Statement-Klauseln} \\ \texttt{mmql} \in \texttt{MMQL} &:= \{\texttt{mms}, \texttt{stmt}_1, \ldots, \texttt{stmt}_n \mid n \geq 1, \texttt{stmt}_\texttt{i} \in \texttt{STMT}\} \end{aligned} \tag{5.1}$$

Das zu verwendende Multimodell *mm* ergibt sich aus der Evaluation der Multimodell-Spezifikation `mms` durch den Interpreter. Dafür existiert die interne Funktion $\mathfrak{interpreteMMSpec}()$ als Bestandteil des Interpreters. Diese lädt im Falle von `MultiModelByFile` die entsprechende Datei und instanziiert das Multimodell oder übergibt im Falle von `MultiModelByEditor` das

[3] vgl. Erläuterung in Abschnitt 4.2.4 auf Seite 97 sowie Syntaxdiagramme 4.13 ff.
[4] vgl. Abschnitt 3.6.1 auf Seite 78
[5] Zur besseren Unterscheidung werden Mengen und Elemente von MMQL-Anweisungen in `Schreibmaschinenschrift` dargestellt.

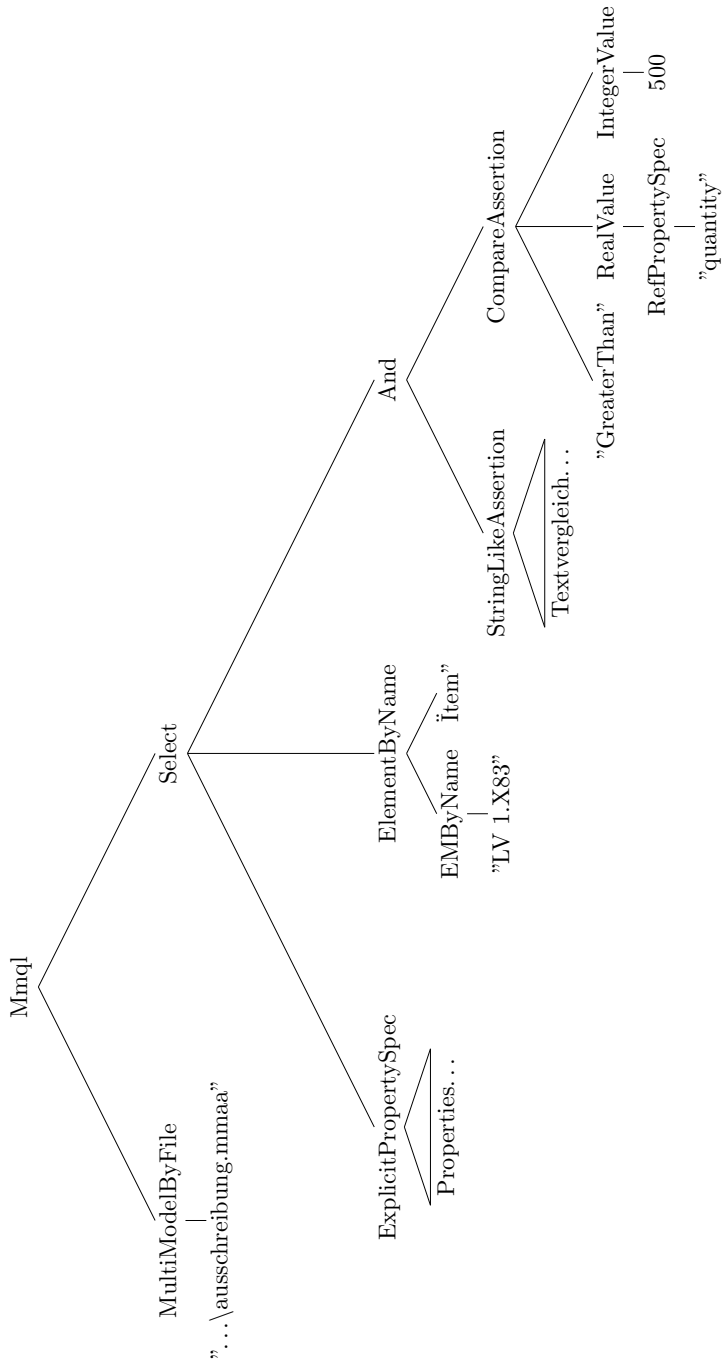


Abbildung 5.1.: Abstrakter Syntaxbaum des MMQL-Quelltexts 4.6 auf Seite 105

vorhandene Multimodell aus dem geöffneten Editor:

$$\begin{aligned} \texttt{MMBF} &:= \text{Unendliche Menge valider MultiModelByFile-Klauseln} \\ \texttt{MMBE} &:= \text{Unendliche Menge valider MultiModelByEditor-Klauseln} \\ \mathfrak{interpreteMMSpec}(\texttt{mms}) &= \begin{cases} \mathfrak{interpreteMMByFile}(\texttt{mms}) & \text{für } \texttt{mms} \in \texttt{MMBF} \\ \mathfrak{interpreteMMByEditor}(\texttt{mms}) & \text{für } \texttt{mms} \in \texttt{MMBE} \end{cases} \end{aligned} \tag{5.2}$$

Um solche Entscheidungen zu ermöglichen, können mithilfe des abstrakten Syntaxbaumes Zustände einer *konkreten* MMQL-Anweisung ermittelt werden. Da im Quelltext 4.6 (repräsentiert durch den Wurzelknoten `mmql`) das Multimodell per Datei angegeben wurde, gilt somit:

$$mm = \mathfrak{interpreteMMByFile}(\texttt{mms}) \,|\, \texttt{mms} \in \texttt{mmql} \tag{5.3}$$

Aus Sicht der Evaluation ist die Oberfunktion $\mathfrak{interpreteMMSpec}()$ definiert als Abbildung einer konkreten MMQL-Multimodell-Spezifikation auf die korrespondierende, reale Multimodell-Instanz. Das Gleiche gilt auch für die Unterfunktionen aus Gleichung 5.2:

$$\mathfrak{interpreteMMSpec}(\texttt{mms}) := \texttt{mms} \mapsto mm \tag{5.4}$$

Formal betrachtet ist der Interpreter eine Sammlung von Funktionen, welche Entscheidungsanweisungen und Werteoperationen vornehmen um schlussendlich Multimodell-Elementaroperationen zu erzeugen und auszuführen. Der abstrakte Syntaxbaum liefert die Ausführungsreihenfolge und die notwendigen Parameter dieser Funktionen.

5.1.3. Erweiterbarkeit der Multimodell-Engine

Die meisten internen Funktionen des MMQL-Interpreters benötigen den Zugriff auf die Originaldaten des Multimodells. Während mit dem Generischen Multimodell eine inhärente und allgemeingültige Implementierung für multimodellspezifische Datenstrukturen vorliegt, sind elementarmodellspezifische Zugriffsmechanismen individuell und müssen daher an Erweiterungskomponenten delegiert werden.

Bereits in Abschnitt 3.4.2 auf Seite 68 wurde dargelegt, wie Multimodell-Softwaresysteme erweiterbar sein müssen. Abbildung 5.2 auf der nächsten Seite zeigt hier die relevanten Erweiterungspunkte der MM-Engine, auf deren Details im Folgenden eingegangen wird. Ein wesentliches Merkmal der Erweiterungstechnik ist, dass der MM-Engine zur Laufzeit alle Erweiterungskomponenten bekannt sind. Dies wird im Diagramm durch entsprechende Registries sowie die Abhängigkeitsbeziehung *registers extension* des jeweiligen Ports verdeutlicht. Der MMQL-Interpreter kann somit notwendige Elementarmodell-Zugriffe auf Abstraktionsebene des Ideellen Elementarmodells abhandeln und die entsprechende Registry zur eigentlichen Ausführung beauftragen. Diese ermittelt anhand des Elemetarmodell-Typs den benötigten Elementarmodell-Parser, veranlasst diesen zum Einlesen des Elementarmodells in den Speicher und übergibt das Speicherobjekt einer EM-Filter-, Property-Accessor- oder Element-Query-Erweiterung.

Die Elementarmodell-Erweiterungskomponenten benötigen ebenso wie der vereinheitlichte Zugriff auf Originaldaten innerhalb der MM-Engine eine Umsetzung des Ideellen Elementarmodells in entsprechende Datenschnittstellen. Abbildung 5.3 auf Seite 133 zeigt die explizite

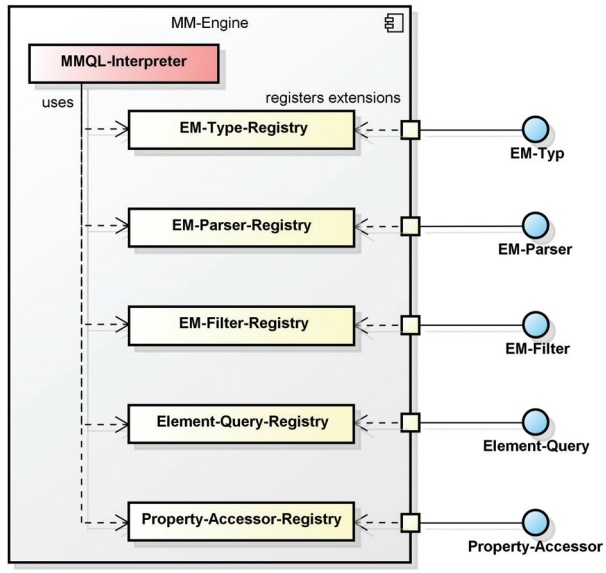


Abbildung 5.2.: UML-Komponentendiagramm der Erweiterungspunkte der Multimodell-Engine (MM-Engine)

Beschreibung der zuvor definierten Erweiterungspunkte sowie ihre Assoziationen untereinander. Die Schnittstellen bilden übergreifend folgende Grundprinzipien des Ideellen Elementarmodells ab:

- Unterscheidbare Typen von Elementarmodellen
- Per Zeichenkette benannte, unterscheidbare Element-Typen
- Per ID (Zeichenkette) repräsentierte Elemente
- Per Zugriffsfunktion erreichbare Properties
- Values als Zeichenkette

Die Schnittstelle *EM-Typ* repräsentiert die Deklaration eines beliebigen Elementarmodell-Typs. Über eine global eindeutige ID müssen alle Elementarmodell-Typen unterscheidbar sein. Dies gilt auch für verschiedene Versionen eines Typs. So könnte IFC 2x3 z. B. die ID `bim-ifc-2x3` erhalten, IFC 2x4 jedoch `bim-ifc-2x4`. Die Eigenschaft *name* liefert zusätzlich eine visuell lesbare Beschreibung des Elementarmodell-Typs.

Die Schnittstelle *EM-Parser* steht für eine Implementierung eines Elementarmodell-Parsers. Solche Erweiterungen können mittels der Methode *parse()* Elementarmodelle über deren URI in den Speicher laden – eingebettete Elementarmodelle müssen dazu temporär zwischengespeichert werden. Die Assoziation *is valid for* definiert, für welche Elementarmodell-Typen ein Parser funktioniert[6].

Die Schnittstellen *EM-Typ* und *EM-Parser* haben eine elementare Bedeutung für alle anderen Erweiterungen, da diese festlegen müssen, für welchen Elementarmodell-Typ sie gültig sind und für welche Parserimplementierung sie entwickelt wurden.

[6] vgl. hierzu auch Erläuterung und Beispiele in Abschnitt 3.4.2 auf den Seiten 68–69

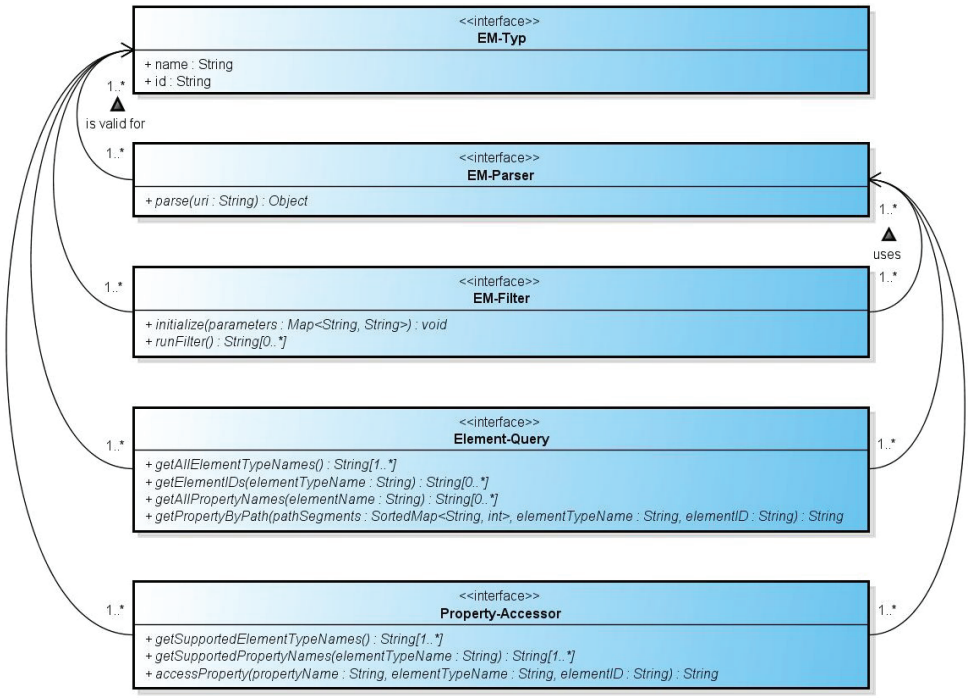


Abbildung 5.3.: UML-Klassendiagramm der Erweiterungspunkte der Multimodell-Engine

Die Bereitstellung von Multimodell-Elementen und der Zugriff auf Properties durch die Funktion *byPropertyPath* erfolgt über die Schnittstelle *Element-Query*. Deren Implementierung ist Voraussetzung zur Teilnahme an MMQL-Abfragen. Die Methode *getElementIDs()* liefert die IDs aller Elemente des angegebenen Element-Typs. Die Methode *getPropertyByPath()* ist die Umsetzung der entsprechenden Property-Zugriffsfunktion und liefert den Value eines bestimmten Properties. Dieses wird über seinen Feature-Path (in Form einer Sequenz von Segmentname und Collection-Index) sowie über das besitzende Element (in Form von Element-Typ und ID) spezifiziert.

Allein diese beiden Methoden ermöglichen die Erschließung des Multimodell-Informationsraumes mittels Basisfunktionalität. In Abschnitt 3.4.3 auf Seite 70 wurde dieses Prinzip bereits dazu verwendet, XML-basierte Elementarmodell-Typen auf Nutzerebene deklarativ zu beschreiben. Dabei wird die Methode *getElementIDs()* durch XPath-Ausdrücke abgebildet, *getPropertyByPath()* wird inhärent über die Navigation des DOM-Baumes (Document Object Model, W3C DOM Level 3, 2004) ermöglicht. Die Methoden *getAllElementTypeNames()* und *getAllPropertyNames()* sollen die Bedienung verbessern, indem sie helfen, Vorschläge zur Eingabe von MMQL-Code zu unterbreiten. Dazu werden alle Namen der Element-Typen geliefert, sowie die Namen aller Properties auf der ersten Ebene, ausgehend von einem Element-Typ.

Mit der Schnittstelle *Property-Accessor* wird die Implementierung von Zugriffsfunktionen des Typs *byPropertyAccessor* ermöglicht. Die Methode *accessProperty()* liefert dazu den Value desjenigen Properties, welches unter dem angegebenen Namen für das besitzende Element (spezifiziert über Element-Typ und ID) vereinbart ist. Wie bei der vorangegangenen Schnittstelle

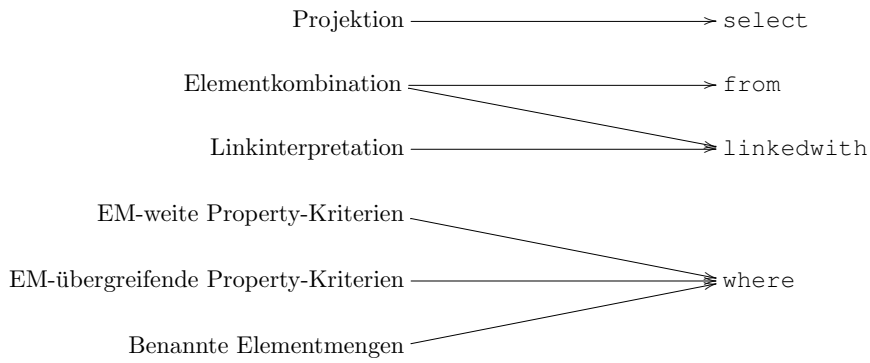


Abbildung 5.4.: Umsetzungsorte der Multimodell-Filtermethoden (linke Seite) in den Schlüsselworten der select-Anweisung (rechte Seite).

liefern die Methoden *getSupportedElementTypeNames()* und *getSupportedPropertyNames()* die Namen der von diesem Property-Accessor unterstützten Element-Typen und der dafür vereinbarten Property-Namen, um eine verbesserte Bedienung mit Vorschlägen zur Nutzereingabe zu erlangen.

Die Schnittstelle *EM-Filter* dient zur Implementierung externer Elementarmodell-Filter. In der Methode *initialize()* werden dem Filter die optionalen Parameter als Schlüssel-Wert-Paare übergeben. Anschließend wird mit *runFilter()* die spezielle Fachlogik ausgeführt. Diese ermittelt diejenige Menge von Elementen, welche die Filterbedingung erfüllen, und liefert deren IDs zurück.

Mit diesen fünf Schnittstellen kann der in Kapitel 4 vorgestellte Sprachumfang innerhalb einer erweiterbaren MM-Engine vollständig und mit den Originaldaten der Elementarmodelle zur Ausführung gebracht werden.

5.2. Ermittlung von Multimodell-Views

5.2.1. Überblick

Multimodell-View Ein Multimodell-View ist eine Teilmenge $\acute{V}$ aller Values V eines Multimodells. Die Values $\acute{V}$ sind in einem s. g. ResultSet $V_{i,j}$[7] tabellarisch angeordnet, um auch die Kombinationen der besitzenden Multimodell-Elemente zu repräsentieren. Unter Berücksichtigung der Links können diese Kombinationen elementarmodellübergreifend sein.

$$\begin{aligned} \acute{V} &\subseteq V \\ \acute{V} &\mapsto V_{i,j} \end{aligned} \tag{5.5}$$

Der Vorgang zum Erzeugen solcher Multimodell-Views – das s. g. Mutlimodell-Filtern – geschieht per MMQL durch die Anwendung der select-Klausel. Diese besteht aus einer oder

[7] vgl. Abschnitt 5.2.2 auf der nachfolgenden Seite; zur besseren Lesbarkeit wird das ResultSet nicht gestrichen als $\acute{V}_{i,j}$sondern lediglich als $V_{i,j}$ notiert

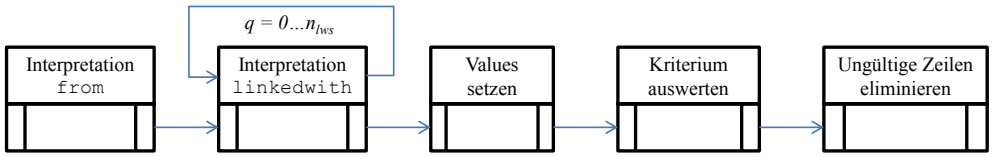


Abbildung 5.5.: Prinzip der Interpretation einer select-Anweisung

mehreren expliziten Property-Spezifikationen `prop`, welche die Projektion beschreiben, einer ElementByName-Spezifikation (from-Klausel) `from`, beliebig vielen LinkedWith-Spezifikationen `lws` sowie *einem* Kriterium (Assertion) `assrt`. Die Angabe des Alias wird hier vernachlässigt – Alias werden zur Laufzeit durch ihre definierenden Spezifikationen ersetzt.

$$\begin{aligned} \texttt{SLCT} &:= \text{Unendliche Menge valider Select-Klauseln} \\ \texttt{slct} \in \texttt{SLCT} &:= \{\texttt{prop}_1, \ldots, \texttt{prop}_{n_{prop}}, \texttt{from}, \texttt{lws}_1, \ldots, \texttt{lws}_{n_{lws}}, \texttt{assrt}\} \\ &\quad n_{prop} \geq 1, n_{lws} \geq 0 \end{aligned} \tag{5.6}$$

In Abschnitt 4.3.1 auf Seite 102 wurden die Methoden des Multimodell-Filterns analysiert. Für die Abarbeitung durch den MMQL-Interpreter ist vorrangig relevant, mit welchen Bestandteilen der select-Anweisung (Klauseln) diese Filter-Methoden umgesetzt werden. Abbildung 5.4 zeigt, an welchen Stellen einer select-Anweisung einzelne Multimodell-Filtermethoden umgesetzt sind. Durch diese Überführung kann der Interpreter Konstrukte einer Klausel schematisch gleich behandeln, unabhängig von der konkreten Filtermethode.

Die Vorgänge zur Abarbeitung einer kompletten select-Anweisung sind daher in Anlehnung an deren Klauseln gegliedert. Ihre Reihenfolge wird hauptsächlich durch die Anforderungen der elementarmodellübergreifenden Property-Kriterien bestimmt. Für deren Auswertung müssen alle Values bereits in Tabellenform vorliegen. Die Tabelle wird zuvor durch Linkauswertung erzeugt.

Abbildung 5.5 zeigt die Grobvorgänge zur Interpretation einer select-Anweisung. Im ersten Schritt werden die IDs der Elemente ermittelt, deren Element-Typ in der from-Klausel angegeben ist. Im zweiten Schritt werden die IDs der in den linkedwith-Klauseln angegebenen Element-Typen nach Maßgabe der vorhandenen Links hinzu kombiniert. Da die Anzahl der linkedwith-Klauseln beliebig $(0...n_{lws})$ sein kann, erfolgt dieser Vorgang sukzessiv. Das Zwischenergebnis ist eine interne Tabelle von IDs, welche im dritten Schritt durch die Values der gewünschten Properties (Projektion) ersetzt werden. Mit den Belegungen der konkreten Values können danach die Kriterien der where-Klausel evaluiert werden. Im letzten Schritt werden diejenigen Zeilen eliminiert, deren Kriteriumauswertung `false` oder `null` ergibt. Das Ergebnis ist das zur Anfrage konforme ResultSet.

5.2.2. Aufbau des ResultSets

Das ResultSet ist Träger der Daten des Multimodell-Views. Einem Benutzer gegenüber erscheint es als Tabelle von Werten.

ResultSet Das ResultSet ist eine Tabelle mit Values $V_{i,j}$. Die Spalten i entsprechen der Projektion und gruppieren die Values nach den Properties p_i. Die Anzahl der Spalten

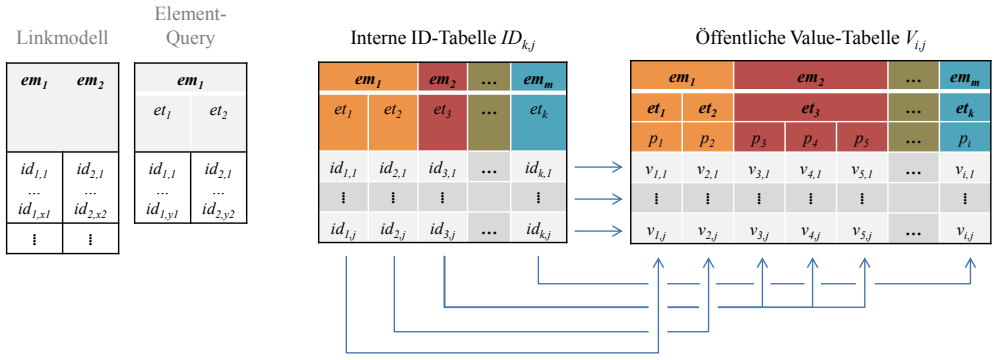


Abbildung 5.6.: Aufbau des ResultSets (rechts): Interne ID-Tabelle mit IDs, gruppiert nach Element-Typen (et_k) aus möglicherweise unterschiedlichen Elementarmodellen (em_m, hier exemplarisch farblich hervorgehoben); öffentliche Value-Tabelle, gruppiert nach Properties (p_i) entsprechend der angeforderten Projektion. Herkunft der IDs (links): Linkmodelle mit potentiell mehrwertigen Links (IDs lediglich nach Elementarmodellen gruppiert) und Elementarmodelle (IDs per Element-Query nach Element-Typen gruppiert).

entspricht der vom Nutzer angegebenen Anzahl Properties: $i = n_{prop}$. Die Zeilen j gruppieren die Values nach einzigartigen Kombinationen von Multimodell-Elementen $E_j = \{e_{1,j}, \ldots, e_{k,j}\}$. Die Anzahl k dieser Kombinationen ergibt sich aus der obligatorischen from-Klausel und der Anzahl der optionalen linkedwith-Klauseln: $k = 1 + n_{lws}$. Die Gesamtanzahl der Zeilen n_{row} lässt sich nicht allgemein ermitteln. Sie ist im Wesentlichen von den Daten der Elementarmodelle, den vorhandenen Links, den jeweils gewählten Modi zur strukturellen Linksemantik sowie den aufgestellten Kriterien abhängig: $n_{row} = 0 \ldots \prod_{x=1}^{k} |et_x|$.

Im vorangegangenen Abschnitt wurde beschrieben, dass das ResultSet schrittweise erstellt wird. Dabei werden bis zum Abschluss der Abarbeitung der letzten linkedwith-Klausel ausschließlich Zeilen – also Elementkombinationen E_j – ermittelt. Dies erfolgt durch Auswertung von Links und Elementen des Multimodells auf Basis ihrer IDs. Anschließend werden die Spalten i gemäß der geforderten Projektion erstellt und die einzelnen Values $v_{i,j}$ in den Zellen durch Auswertung des Properties p_i für das korrespondierende Element $e_{i,j} \in E_j$ gesetzt. Im letzten Schritt werden diejenigen Zeilen entfernt, für welche Property-Kriterien oder benannte Elementmengen zu `null` oder `false` evaluieren.

Die schrittweise Erstellung des ResultSets spiegelt sich in seinem internen Aufbau wider. Abbildung 5.6 zeigt seine beiden Bestandteile, welche sich maßgeblich aus der zweigliedrigen Erzeugung ergeben:

1. Die interne ID-Tabelle $ID_{k,j}$ für die ID-Kombinationen der Zeilen
2. Die öffentlich sichtbare Value-Tabelle $V_{i,j}$

Die zusätzliche interne ID-Tabelle ist für Nutzer nicht sichtbar. Ihre Aufgabe ist das Speichern der Ergebnis-Zeilen mit den kombinierten IDs $ID_j = \{id_{1,j}, \ldots, id_{k,j}\}$, resultierend aus Elementkombination und Linkinterpretation. Die interne ID-Tabelle kann als ein virtuelles Linkmodell aufgefasst werden, welches aus den kombinierten Elementen des Multimodell-Filterergebnis

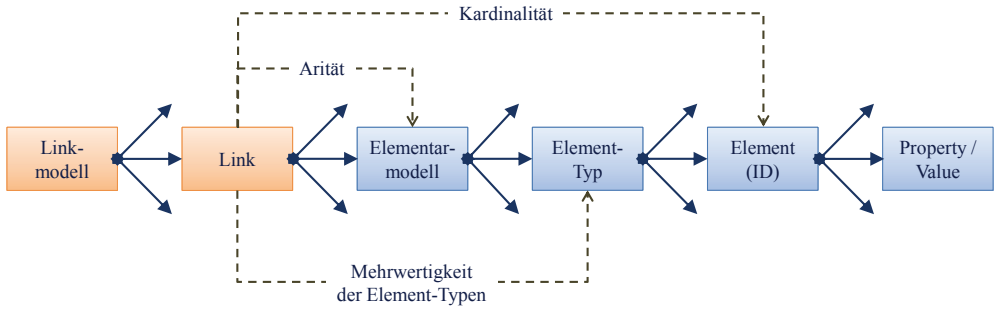


Abbildung 5.7.: Aufnahme des Element-Typs in die virtuelle Gruppierung der Multimodell-Komponenten während der Verarbeitung von MMQL-Statements. Relevante Kenngrößen der Mehrwertigkeit von Links.

besteht. Die ID-Tabelle wird durch Interpretation der from-Klausel und der linkedwith-Klauseln sukzessive spaltenweise aufgebaut[8]. Die Zeilen der ID-Tabelle sind äquivalent zu den Zeilen der späteren Value-Tabelle[9]:

$$ID_j = \{id_{1,j}, \ldots, id_{k,j}\} \quad \Leftrightarrow \quad E_j = \{e_{1,j}, \ldots, e_{k,j}\} \tag{5.7}$$

Im Linkmodell sind IDs lediglich ihren Elementarmodellen strukturell zugeordnet. Eine Aussage über irgendwelche Eigenschaften des korrespondierenden Multimodell-Elements ist somit nicht ohne weiteres möglich. Erst mithilfe der Element-Query-Erweiterungen[10] können Elemente zu Typen zu gruppiert werden. Dadurch erhöht sich die fachliche Semantik von Filterformulierungen. Anstelle der zuvor einzig möglichen Anfrage „alle Elemente des Elementarmodells" kann nun nach Typen (z. B. Klassen) von Elementen gesucht werden, bspw. „alle Wände des Bauwerksmodells". Abbildung 5.7 zeigt die sich somit ergebende virtuelle Gruppierung der Multimodell-Komponenten. Auch in Abbildung 5.6 ist zur Verdeutlichung des Ursprungs der IDs zusätzlich das besitzende Elementarmodell em_m des jeweiligen Element-Typs et_k angegeben.

Die MMQL-Anweisungen zur Elementkombination (from- und linkedwith-Klauseln) beschreiben ebenfalls Element-Typen. Werden vom Interpreter – im Falle von linkedwith per Oberfunktion $\mathfrak{interpreteLinkedWith}()$ – die realen Element-Typen et_k ermittelt, so formen diese die Spalten der internen ID-Tabelle, nach welchen die IDs gruppiert werden. Dabei stammt der erste Element-Typ aus der from-Klausel, alle weiteren aus den linkedwith-Klauseln:

$$et_k \quad = \quad \begin{cases} \mathfrak{interpreteFrom}(\texttt{from}) & \text{für } k = 1 \\ \mathfrak{interpreteLinkedWith}(\texttt{lws}_{k-1}) & \text{für } k = 2 \ldots n_{lws} + 1 \end{cases} \tag{5.8}$$

[8] vgl. Abbildung 5.5 auf Seite 135 sowie Abschnitt 5.2.3 auf der nachfolgenden Seite
[9] vgl. auch Gleichung 3.4 auf Seite 65
[10] vgl. Abschnitt 5.1.3 auf Seite 131

Dem Element-Typ et_k ist auch das zugehörige Elementarmodell em_m zugeordnet. Dieses ist in der MMQL Teil der Beschreibung jedes Element-Typs[11]:

$$\begin{aligned} \texttt{EMS} &= \text{Unendliche Menge valider ElementaryModelSpec-Angaben} \\ \texttt{ems}_k \in \texttt{EMS} &= \begin{cases} \texttt{from} \mapsto \texttt{ems}_k & \text{für } k = 1 \\ \texttt{lws}_{k-1} \mapsto \texttt{ems}_k & \text{für } k = 2 \dots n_{lws} + 1 \end{cases} \\ em_m &= \mathfrak{interpreteElementaryModelSpec}(\texttt{ems}_k) \end{aligned} \tag{5.9}$$

Nach Abarbeitung der letzten linkedwith-Klausel erfolgt die Projektion. Dabei werden u. a. die Spalten k der internen ID-Tabelle auf die Spalten i der öffentlichen Value-Tabelle abgebildet. Aus jeweils *einer* Spalte k (entsprechend *einem* Element-Typ et_k) wird *eine* oder mehrere Spalten i (entsprechend des oder der angeforderten Properties p_i aus et_k) erzeugt:

$$ID_k \mapsto V_i \mid id_{k,j} \equiv e_{k,j}, p_{i,j} \in e_{k,j}, p_{i,j} \mapsto v_{i,j} \tag{5.10}$$

Die interne ID-Tabelle und die öffentliche Value-Tabelle sind genau dann identisch, wenn durch den Nutzer in der Projektion jeweils nur das ID-Property jedes verwendeten Element-Typs angefordert wird. Diese Konstellation kann benutzt werden, um auf Anwenderebene das ResultSet als virtuelles, gefiltertes Linkmodell zu interpretieren:

$$p_i = id(e_x) \Rightarrow \; ID_k \equiv V_i \mid e_x \in et_k, i = k \tag{5.11}$$

Die Abfragemöglichkeit nach Element-Typen macht eine zusätzliche Umgruppierung der IDs aus den Linkmodellen notwendig. Außerdem müssen mehrwertige Links in einwertige Element-Kombinationen aufgebrochen werden, damit Property-Kriterien auf Basis herkömmlicher Logikoperatoren formuliert und ausgeführt werden können. Dies erhöht die Komplexität der Verfahren zur Erstellung des ResultSets. Die nachfolgenden Abschnitte bis einschließlich 5.2.6 beschreiben, wie die Belegung der internen ID-Tabelle im Detail ermittelt wird. In Abschnitt 5.2.7 auf Seite 147 wird genauer erläutert, wie per Projektion die befüllte ID-Tabelle in die öffentliche Value-Tabelle umgewandelt wird. Die Auswertung des Kriteriums und die daraus folgende Reduktion von ResultSet-Zeilen werden in Abschnitt 5.2.8 auf Seite 148 erklärt.

5.2.3. Sukzessive Linkauswertung

Die sukzessive Linkauswertung ist ein iteratives Verfahren zur Erstellung der internen ID-Tabelle des endgültigen ResultSets. Das grundlegende Merkmal ist die stufenweise Abarbeitung der linkedwith-Klauseln. Dabei wird in jedem Verarbeitungsschritt $q = 0 \dots n_{lws}$ eine vollwertige ID-Tabelle produziert, welche als Eingangsgröße des nächsten Iterationsschritts $q + 1$ benötigt wird. Jeder Element-Typ et_k bedingt somit einen solchen Verarbeitungsschritt. Eingangsgröße für den ersten Iterationsschritt $q = 0$ ist das ID-Set des Element-Typs et_1, welcher durch die from-Klausel spezifiziert ist[12]. Die sich aus der letzten linkedwith-Auswertung ergebende ID-Tabelle ist das Resultat des Verfahrens. Die Reihenfolge der angegebenen Element-Typen in den linkedwith-Klausel hat einen Einfluss auf das Endergebnis.

[11] vgl. `ElementaryModelSpec` in Quelltext A.3 auf Seite 188 Zeile 19
[12] vgl. Gleichung 5.13 auf Seite 140

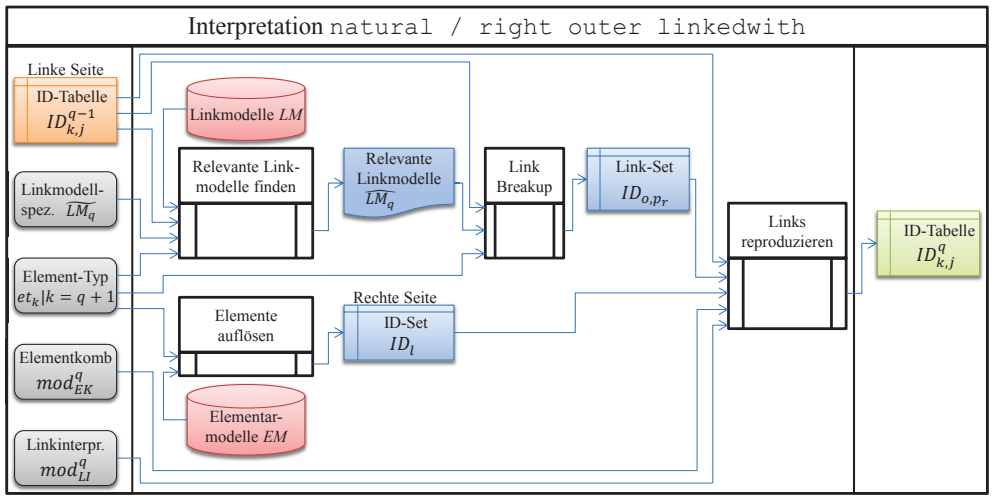


Abbildung 5.8.: Sukzessive Linkauswertung: Überblick zum Interpreterverfahren für linkedwith-Anweisungen bei Elementkombination *Natural* oder *Right Outer*

Die sukzessive Auswertung der linkedwith-Klauseln ermöglicht individuelle Modi der Linkinterpretation und Elementkombination für jeden angefragten Element-Typ[13]. Diese Angaben gelten für den korrespondierenden Verarbeitungsschritt und beeinflussen wie – ausgehend von der vorhandenen ID-Tabelle der vorangegangenen Abarbeitung – die neue ID-Tabelle gebildet wird. Auf diese Weise ermöglicht MMQL eine detaillierte Beeinflussung der Erschließung domänenübergreifender Informationsräume.

Durch ihren iterativen Charakter ist die sukzessive Linkauswertung für eine Implementierung als MMQL-Interpreterfunktion gut geeignet. Sie funktioniert für alle variablen Parameter eines Iterationsschritts q in sich geschlossen und in gleicher Weise. Die variablen Parameter der Auswertung einer einzelnen linkedwith-Anweisung $\texttt{lws}_q$ sind:

- die ID-Tabelle des vorangegangenen Iterationsschritts $ID_{k,j}^{q-1}$
- der hinzu zu kombinierende Element-Typ $et_k \,|\, k = q+1$
- die zu nutzenden Linkmodelle $\widetilde{LM_q} \subseteq LM$
- der Modus der Elementkombination $mod_{\text{EK}}^{q} \in \{\text{'natural'}, \text{'right'}, \text{'cross'}\}$
- der Modus der Linkinterpretation $mod_{\text{LI}}^{q} \in \{\text{'strict'}, \text{'standard'}, \text{'trans'}, \varnothing\}$

Des Weiteren gelten die folgenden Parameter, welche für alle linkedwith-Anweisungen aller select-Statements der MMQL-Anweisung global konstant sind:

- die vorhandenen Linkmodelle LM des benutzten Multimodells mm
- die vorhandenen Elementarmodelle EM des benutzten Multimodells mm

Abbildung 5.8 zeigt das Verfahren der sukzessiven Linkauswertung für die Elementkombinationen *Natural* und *Right Outer*[14]. Die variablen Parameter sind auf der linken Seite als

[13] vgl. Syntax-Definition `select` auf Seite 88 sowie `linkedwith` auf Seite 113, bzw. Quelltext A.3 Zeilen 3 und 27–29

[14] vgl. Absatz „Die Elementkombination" auf Seite 112

Eingangsgrößen aufgelistet. Die globalen Konstanten LM und EM sind als Datenquelle dargestellt.

Das Prinzip der Linkauswertung ist die Überprüfung der Reproduzierbarkeit vorhandener Links für die gegebenen Parameter. Dabei wird jede Zeile der vorhandenen ID-Tabelle ($ID_{k,j}^{q-1}$, linke Seite) mit jeder ID des Element-Typs (rechte Seite) kombiniert und anschließend getestet, ob diese Kombination einem relevanten, existenten Link entspricht. Alle Kombinationen, welche dieses Kriterium erfüllen, formen die resultierende ID-Tabelle $ID_{k,j}^{q}$. Das entsprechende Unterverfahren *Links reproduzieren* wird in Abschnitt 5.2.6 auf Seite 144 detailliert beschrieben. Das Unterverfahren zum Finden der relevanten Linkmodelle wird in Abschnitt 5.2.4 auf der nachfolgenden Seite erläutert. Das notwendige Umwandeln der n-ären Links in eine einwertige Form (Link Breakup) wird in Abschnitt 5.2.5 auf Seite 142 dargestellt.

Elemente auflösen und Interpretation der from-Klausel

Das Verfahren *Elemente auflösen* wandelt die Beschreibung eines Element-Typs et_k in eine Menge von IDs ID_l um. Dieses so genannte ID-Set repräsentiert alle l Multimodell-Elemente des Typs et_k im Elementarmodell $em_m \in EM$[15]. Dafür beauftragt der Interpreter die Element-Query-Registry[16], die Element-Query-Erweiterung für das Elementarmodell em_m zu finden und dort die Methode *getElementIDs()* mit dem Parameter et_k aufzurufen. Die Erweiterung ermittelt daraufhin die – möglicherweise leere – Menge von IDs ID_l:

$$ID_l = \mathfrak{getElementIDs}(et_k) \,|\, k = q + 1 \tag{5.12}$$

Die identische Vorgehensweise wird zur Interpretation der from-Klausel angewendet. Diese kommt in jeder select-Anweisung genau einmal vor und legt den ersten Element-Typ fest. Werden keine weiteren linkedwith-Klauseln spezifiziert, bleibt dies auch der einzige Element-Typ[17] dieser Abfrage. In jedem Fall bildet bei der Interpretation der from-Klausel das ermittelte ID-Set ID_l die erste interne ID-Tabelle $ID_{k,j}^{0}$ mit $k = 1$ und $j = l$:

$$ID_{1,l}^{0} = \mathfrak{getElementIDs}(et_1) \tag{5.13}$$

Full Cross Product

Eine linkedwith-Klausel kann neben den Elementkombinationen *Natural* und *Right Outer* auch den Modus *Full Cross Product* besitzen[18]. In diesem Fall erfolgt die Erstellung der resultierenden ID-Tabelle unabhängig von den vorhandenen Linkmodell-Daten durch Einbeziehung *aller* Multimodell-Elemente des spezifizierten Typs. In Abbildung 5.9 auf der nachfolgenden Seite ist zu erkennen, dass keine Eingangsgrößen oder Verarbeitungsschritte existieren, welche sich auf Linkmodell-Daten beziehen. Die Ermittlung der rechten Seite (ID-Set ID_l) erfolgt in gleicher Weise wie bei den anderen Modi zur Elementkombination.

[15] zur Ermittlung von em_m vgl. Gleichung 5.9 auf Seite 138
[16] vgl. Abschnitt 5.1.3 auf Seite 131
[17] Dies entspricht einem Elementarmodell-Filter; vgl. Abschnitt 4.3.2 auf Seite 103.
[18] vgl. Absatz „Die Elementkombination" auf Seite 112

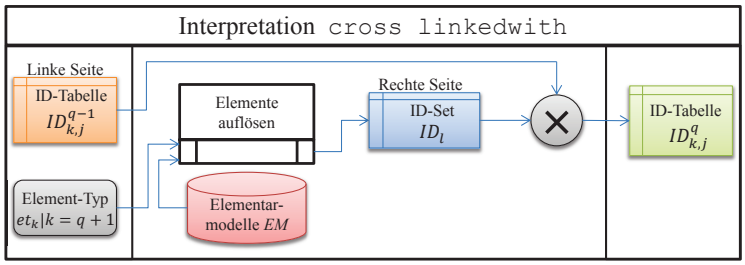


Abbildung 5.9.: Interpreterverfahren für `cross linkedwith`-Anweisungen

Die resultierende ID-Tabelle $ID^q_{k,j}$ ergibt sich aus dem Kreuzprodukt der Zeilen der linken Seite ID^{q-1}_j mit den IDs der rechten Seite, was eine vollständige Kombination bedeutet:

$$ID^q_{k,j} \quad = \quad ID^{q-1}_j \times ID_l \tag{5.14}$$

5.2.4. Relevante Linkmodelle finden

In jeder linkedwith-Klausel der Elementkombinationen *Natural* und *Right Outer* kann eine optionale Menge von Linkmodellen $\widetilde{LM_q} \subseteq LM$ angegeben werden, welche für die aktuelle Linkauswertung q herangezogen werden sollen[19]. Da Links aufgabenspezifisch – mglw. sogar mit gegensätzlicher Fachbedeutung – in Linkmodellen gruppiert werden, können Nutzer somit bestimmen, welche Links für den zu erstellenden Multimodell-View relevant sind. Werden keine Linkmodelle explizit spezifiziert, so werden alle Linkmodelle des Multimodells betrachtet: $\widetilde{LM_q} \equiv LM$.

Bei der Menge $\widetilde{LM_q}$ handelt es sich um eine Nutzerangabe, welche möglicherweise inexakte Angaben enthält. Beispielsweise könnte ein Linkmodell angegeben sein, welches kein Elementarmodell der aktuellen Linkauswertung q beinhaltet. $\widetilde{LM_q}$ darf daher lediglich als *potentielle* Grundmenge verwendbarer Linkmodelle aufgefasst werden. Das Interpreterverfahren *Relevante Linkmodelle finden* ermittelt für $\widetilde{LM_q}$ die relevanten Linkmodelle $\widehat{LM_q} \subseteq \widetilde{LM_q}$.

Da die IDs der Links in den Linkmodellen nach Elementarmodellen geordnet sind, müssen zunächst die zugehörigen Elementarmodelle $\widehat{EM_m}$ aus den Element-Typen et_k der aktuellen Linkauswertung ermittelt werden[20]. Die $\widehat{EM_m}$ ergeben sich aus den Elementarmodellen welche durch die Element-Typen der linken Seite bestimmt werden (entsprechend $\texttt{ems}_1 \ldots \texttt{ems}_q$), sowie dem Elementarmodell welches durch den Element-Typ der rechten Seite bestimmt wird ($\texttt{ems}_{q+1}$):

$$\widehat{EM_m} \quad = \quad \bigcup_{k=1}^{q+1} \mathfrak{interpreteElementaryModelSpec}(\texttt{ems}_k) \tag{5.15}$$

Die relevanten Linkmodelle $\widehat{LM_q}$ werden aus dem Vergleich der zugehörigen Elementarmodelle $\widehat{EM_m}$ mit den jeweiligen verlinkten Elementarmodellen $\breve{EM}$ jedes Linkmodells $\widetilde{lm} \in \widetilde{LM_q}$

[19] vgl. Syntax-Definition `linkedwith` auf Seite 113, bzw. Quelltext A.3 Zeilen 27–29

[20] zur Ermittlung von $em_m \in \widehat{EM_m}$ vgl. Gleichung 5.9 auf Seite 138

ermittelt[21]. Hierbei sind folgende Zustände möglich:

$$\begin{aligned} \textit{unrepräsentiert} &:= \widehat{EM_m} \cap \breve{EM} = \varnothing \\ \textit{unterrepräsentiert} &:= \widehat{EM_m} \supset \breve{EM} \\ \textit{exakt repräsentiert} &:= \widehat{EM_m} = \breve{EM} \\ \textit{überrepräsentiert} &:= \widehat{EM_m} \subset \breve{EM} \end{aligned} \tag{5.16}$$

Die relevanten Linkmodelle der aktuellen Linkauswertung q sind die exakt und überrepräsentierten Linkmodelle. Dabei ist es unerheblich, ob alle Links eines Linkmodells auch voll besetzt sind; d. h. dass sie entsprechend der verlinkten Elementarmodelle $\breve{EM}$ jeweils mindestens *eine* ID besitzen. Diese Überprüfung ist je nach Modus der Linkinterpretation relevant und wird im Unterverfahren *Links reproduzieren* durchgeführt[22].

$$\begin{aligned} \widehat{EM_m} \subseteq \breve{EM} &\Leftrightarrow \textit{exakt repräsentiert} \vee \textit{überrepräsentiert} \\ \widehat{LM_q} &= \bigcup_{n=1}^{\left|\widehat{LM_q}\right|} \widetilde{lm_n} \,|\, \widehat{EM_m} \subseteq \breve{EM}(\widetilde{lm_n}) \end{aligned} \tag{5.17}$$

5.2.5. Link Breakup

Das Verfahren des LinkBreakups bricht mehrwertige Links in einwertige Element-Kombinationen (bezogen auf Element-Typen) auf. Dies ist notwendig, um die Reproduzierbarkeit von Links unter Einsatz von Nutzerkriterien auf Basis herkömmlicher Logikoperatoren zu ermöglichen. Dazu müssen nämlich die *möglichen* Elementkombinationen, welche sich aus der Anfrage ergeben, und die in den relevanten Linkmodellen vorhandenen *tatsächlichen* Elementkombinationen in strukturell vergleichbarer Form vorliegen. Diese Voraussetzung ist jedoch aufgrund der folgenden beiden Sachverhalte nicht erfüllt:

1. Die verlinkten Elemente eines Links können inhärent nur nach Elementarmodellen gruppiert werden. Multimodell-Filteranfragen werden aber auf Ebene von Element-Typen formuliert, weswegen die Spalten der linken (ID-Tabelle $ID_{k,j}^{q-1}$) und rechten Seite (ID-Set ID_l) auch Element-Typen repräsentieren. Daher müssen die nach Elementarmodellen gruppierten, mehrwertigen Links der relevanten Linkmodelle nun mehrwertig nach Element-Typen umgruppiert werden.
2. Nach Gleichung 3.8 auf Seite 66 können Links mehr als *ein* Element je Elementarmodell enthalten (wenn $m_{em} < n_e$). Auch kann nicht ausgeschlossen werden, dass mehrere oder alle dieser Elemente zum selben Element-Typ gehören. Bezüglich eines Element-Typs sind Links also potentiell mehrwertig – die Zeilen der linken und rechten Seite enthalten diesbezüglich aber nur einwertige Einträge. Daher müssen die umgruppierten Linkmodelle aus Punkt 1 zusätzlich noch in einwertige Element-Kombinationen umgewandelt werden.

Das Verfahren des Link Breakups formt demnach die tatsächlichen Links der relevanten Linkmodelle $\widehat{LM_q}$ in Elementkombinationen (repräsentiert durch IDs) um, welche nach Element-Typen gruppiert und diesbezüglich einwertig sind. Diese Kombinationen werden im Link-Set ID_{o,p_r} kumuliert, wobei der Bezug zu den ursprünglichen Links erhalten bleibt, indem die

[21] vgl. auch Definition des Linkmodells in Gleichung 3.10 auf Seite 66
[22] vgl. Abschnitt 5.2.6 auf Seite 144

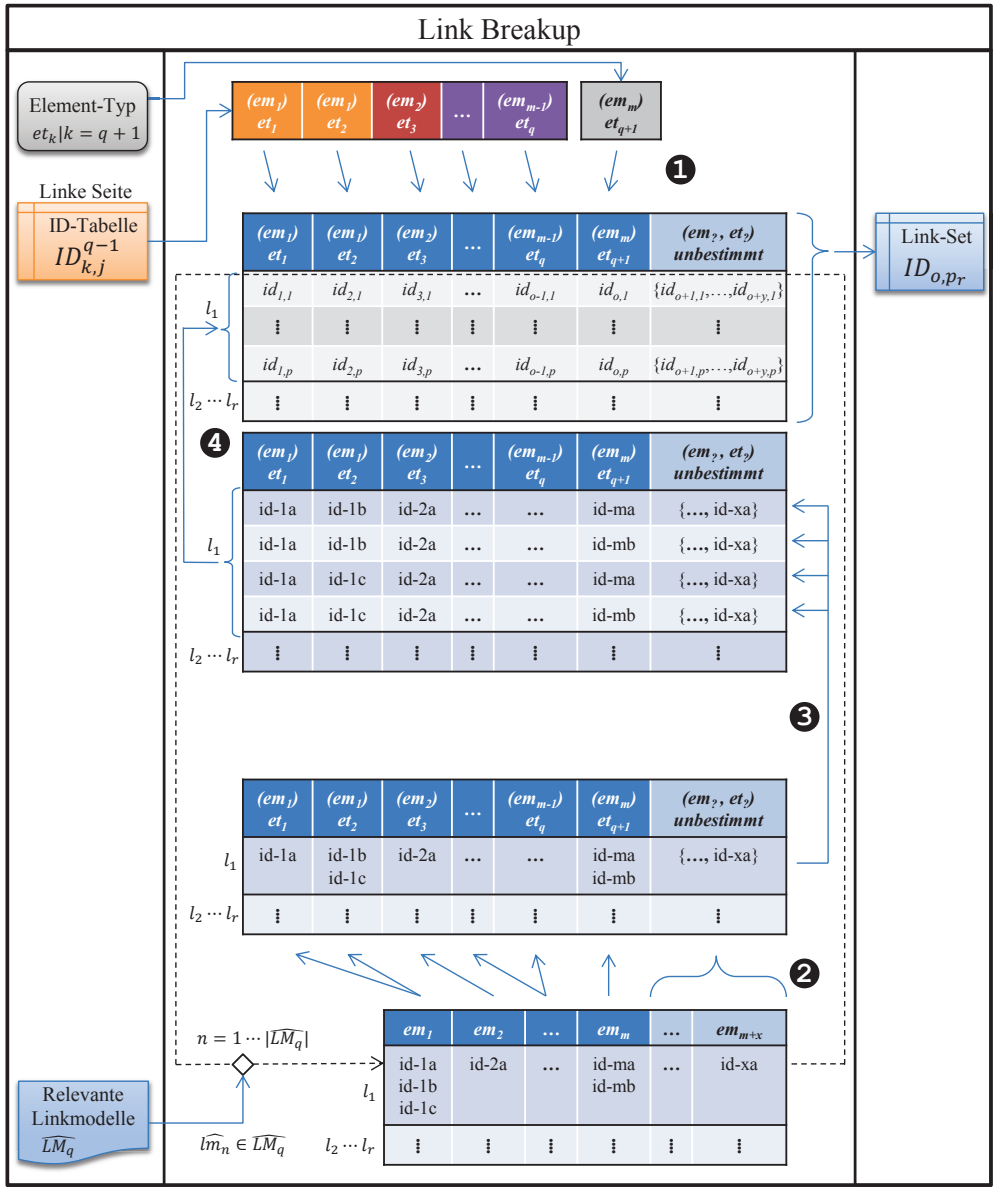


Abbildung 5.10.: Interpreterverfahren für Link Breakup. ① Festlegung der Spalten des Link-Sets durch die bestimmten Element-Typen; ② Umgruppierung der verlinkten Elemente nach vorhandenen Element-Typen, Aggregation der unbestimmten Element-Typen in *einer* Spalte; ③ Aufbrechen mehrwertiger Links in einwertige Elementkombinationen; ④ Kumulation im globalen Link-Set.

Zeilen p_r für jeden Link l_r eine logische Einheit bilden. Abbildung 5.10 auf der vorangegangenen Seite zeigt die dazu notwendigen Schritte.

Das Link-Set ist eine Tabelle von IDs mit o Spalten und $r \cdot p_r$ Zeilen. Eine oder mehrere Zeilen repräsentieren einen ursprünglich ein- oder mehrwertigen Link. Die Spalten repräsentieren die *bestimmten* Element-Typen. Diese ergeben sich aus den Spalten der linken Seite ($o = 1 \ldots q$) sowie dem Element-Typ der aktuellen linkedwith-Anweisung et_k ($o = q + 1$, vgl. Schritt ①). Zusätzlich existiert eine weitere Spalte ($o = q + 2$), welche alle IDs aus den Links aufnimmt, welche zu anderen Element-Typen gehören. Diese Element-Typen heißen *unbestimmt*.

In Schritt ② werden die verlinkten Elemente aller Links eines Linkmodells von Elementarmodellen auf Element-Typen umgruppiert. Ein Beispiel-Link l_1 illustriert dieses Vorgehen in Abbildung 5.10. Dabei bilden die im ersten Schritt ermittelten Element-Typen die Zielgruppen. Die Identifikation des Element-Typs einer einzelnen ID erfolgt durch Auflösen der Elemente für jeden Typ und Überprüfung, in welchem ID-Set die betrachtete ID enthalten ist[23]. Falls die ID in keinem Set vorkommt, so repräsentiert sie ein Multimodell-Element, welches für den aktuellen Iterationsschritt nicht relevant ist. Die ID ist dann den unbestimmten Element-Typen zuzuordnen. Bei der Umgruppierung einer ID id gelten demnach folgende Regeln zur Ermittlung der Zielspalte o:

$$o \quad = \quad \begin{cases} o & \text{für } id \in \texttt{getElementIDs}(et_o) \wedge o \leq q+1 \\ q+2 & \text{sonst} \end{cases} \tag{5.18}$$

Daraus ergibt sich, dass nach der Umgruppierung eines Links l_r jede Zelle $ID_{o,r}$ $0 \ldots n_e - 1$ IDs beinhaltet. Diese potentielle Mehrwertigkeit der Links wird in Schritt ③ in einwertige Elementkombinationen umgewandelt. Dazu wird spaltenweise das Kreuzprodukt über alle IDs $ID_{o,r}$ gebildet, wobei die leere Menge (keine ID in dieser Spalte vorhanden) erhalten bleiben soll. Genauso werden die IDs der unbestimmten Element-Typen ($o = q + 2$) als *ein* Objekt behandelt und in jeder Zeile übernommen:

$$ID^n_{o,p_r} = ID_{1,r} \times ID_{2,r} \times \ldots \times ID_{q+2,r} \tag{5.19}$$

Das lokale Link-Set ID^n_{o,p_r} enthält nun alle Links des Linkmodells $\widehat{lm_n} \in \widehat{LM_q}$ nach den zur Anfrage korrespondierenden Element-Typen gruppiert und je Link in einwertige Elementkombinationen aufgebrochen. Das lokale Link-Set wird in Schritt ④ an das globale Link-Set ID_{o,p_r} angefügt. Die Schritte ② bis ④ werden n-mal für jedes relevante Linkmodell wiederholt. Am Ende dieser Schleife ist das Link-Set ID_{o,p_r} vollständig erstellt worden.

5.2.6. Linkreproduktion

Die Linkreproduktion ist der Kern der sukzessiven Linkauswertung. Hier wird die resultierende ID-Tabelle des aktuellen Iterationsschritts q erstellt. In den Abschnitten 5.2.4 und 5.2.5 wurden die zur Linkreproduktion notwendigen Vorbereitungen beschrieben.

[23] Das Auflösen der Elemente nach ihrem Typ ist in Abschnitt „Elemente auflösen und Interpretation der from-Klausel" auf Seite 140 beschrieben. Interpreter-Implementierungen können an dieser Stelle Rechenzeiten verringern, indem sie bspw. zwischengespeicherte ID-Sets der vorangegangenen Iterationsschritte $0 \ldots q - 1$ wiederverwenden, sortieren und eine binäre Suche anwenden.

Abbildung 5.11 auf der nachfolgenden Seite zeigt das Prinzip der Linkreproduktion. In Schritt ① wird jede Zeile ID_j^{q-1} der ID-Tabelle des vorangegangenen Iterationsschritts (linke Seite) mit dem ID-Set des aktuellen Element-Typs (rechte Seite) mittels der beiden inneren Schleifen über j und l kombiniert:

$$\widetilde{ID_{k,j}^q} \quad = \quad ID_j^{q-1} \times ID_l \tag{5.20}$$

Jede Zeile $\widetilde{ID_j^q} \subseteq \widetilde{ID_{k,j}^q}$ dieser Kombination wird dabei mit jeder Zeile $ID_{r \cdot p_r}$ des Link-Sets ID_{o,p_r} verglichen (äußere Schleife über $r \cdot p_r$). Beide Zeilen sind gleich, wenn ihre IDs spaltenweise übereinstimmen. Die Spalte der unbestimmten Element-Typen bleibt dabei zunächst unberücksichtigt. Im Falle zweier unbelegter IDs soll Gleichheit gelten:

$$\begin{aligned} \varnothing \quad &= \quad \varnothing \\ \widetilde{id_{k,j}^q} = id_{o,p_r} \,|\, k = o, k = 1 \ldots o \quad &\Rightarrow \quad \widetilde{ID_j^q} = ID_{r \cdot p_r} \end{aligned} \tag{5.21}$$

Nach Maßgabe der Regeln für den Modus der aktuellen Linkinterpretation mod_{LI}^q ist zu entscheiden, ob bei Gleichheit die Zeile $\widetilde{ID_j^q}$ einen Teil eines relevanten Links l_r repräsentiert und demzufolge in die resultierende ID-Tabelle $ID_{k,j}^q$ aufgenommen wird[24].

$$istResultat(\widetilde{ID_j^q}) \quad := \quad \widetilde{ID_j^q} \in ID_{k,j}^q \tag{5.22}$$

Im Modus *standard* genügt dafür die Übereinstimmung der Zeile, da Teile eines Links entfallen dürfen.

$$mod_{\mathrm{LI}}^q = \text{'standard'} \wedge \widetilde{ID_j^q} = ID_{r \cdot p_r} \quad \Rightarrow \quad istResultat(\widetilde{ID_j^q}) \tag{5.23}$$

Auch bei transitiver Linkinterpretation dürfen Teile eines Links entfallen. Daher genügt entweder die Übereinstimmung der gesamten Zeile oder die Übereinstimmung der Spalten der linken Seite und mindestens *eine* transitive Erreichbarkeit der ID id_l der rechten Seite im Link-Set ID_{o,p_r}[25].

$$\begin{aligned} \widetilde{id_{k,j}^q} = id_{o,p_r} \,|\, k = o, k = 1 \ldots o - 1 \quad &\Rightarrow \quad linkeSeiteGleich(\widetilde{ID_j^q}) \\ mod_{\mathrm{LI}}^q = \text{'trans'} \wedge (\widetilde{ID_j^q} = ID_{r \cdot p_r} & \\ \vee linkeSeiteGleich(\widetilde{ID_j^q}) \wedge erreichbar(id_l)) \quad &\Rightarrow \quad istResultat(\widetilde{ID_j^q}) \end{aligned} \tag{5.24}$$

Dagegen muss im Modus *strict* ein Link *vollständig* reproduzierbar sein. D. h. zusätzlich zur Übereinstimmung der Zeile muss die Spalte der unbestimmten Element-Typen leer sein. Außerdem müssen alle Zeilen p_r aus dem Link-Set ID_{o,p_r} reproduzierbar sein und im Resultat erscheinen, da diese logisch zusammenhängen, um den potentiell mehrwertigen Link l_r

[24] vgl. auch Definition der Modi im Absatz „Die Linkinterpretation" auf Seite 112
[25] vgl. auch Beispiel 16 auf Seite 116

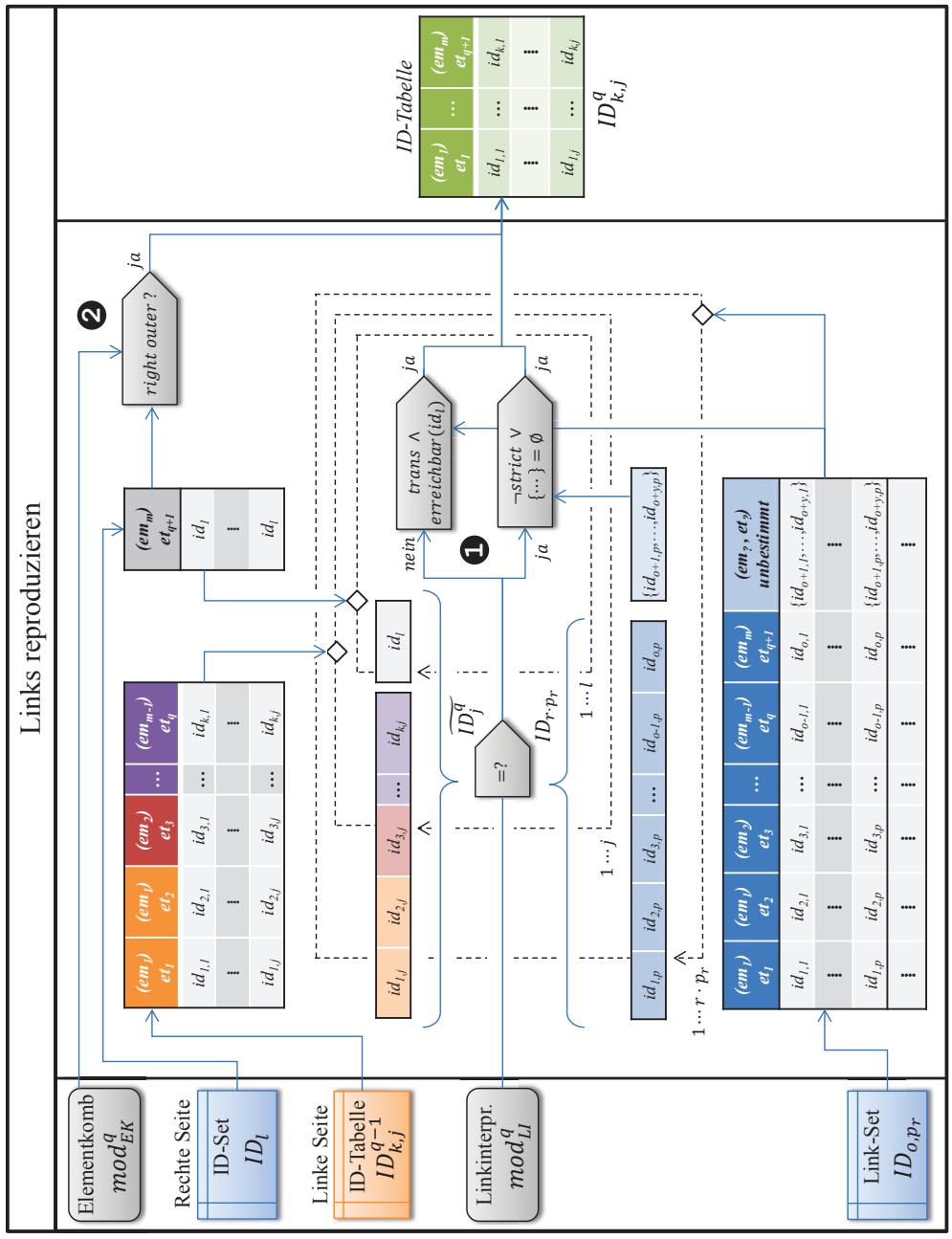


Abbildung 5.11.: Interpreterverfahren zum Reproduzieren von Links. ① Hinzufügen von Zeilen nach Maßgabe der Linkinterpretation; ② Auffüllen nicht vorhandener IDs der rechten Seite bei Elementkombination *right outer*.

abzubilden.

$$\begin{aligned} mod^q_{\mathrm{LI}} = \text{'strict'} \wedge \widetilde{ID^q_j} = ID_{r \cdot p_r} \\ \wedge \{id_{o+1,p}, \ldots, id_{o+y,p}\} = \varnothing \\ \wedge istResultat(ID_{1_r}) \wedge \ldots \wedge istResultat(ID_{p_r}) \quad \Rightarrow \quad istResultat(\widetilde{ID^q_j}) \end{aligned} \tag{5.25}$$

Die o. g. Aussagen zur Linkinterpretation sind in Abbildung 5.11 vereinfacht als Verzweigungsbedingungen eines Datenflusses dargestellt.

In Schritt ② erfolgt die Vervollständigung der resultierenden ID-Tabelle $ID^q_{k,j}$ für den Modus der Elementkombination *right outer*. Dazu wird überprüft, ob jede ID der rechten Seite ID_l mindestens einmal im Resultat vorhanden ist. Fehlende IDs werden zeilenweise in der letzten Spalte angefügt, die übrigen Spalten dieser Zeilen werden mit $\varnothing$ aufgefüllt[26].

Im Modus *natural* sind keine weiteren Bearbeitungsschritte mehr notwendig. Der Modus *full cross product* ist unabhängig von einer Linkinterpretation und wird separat behandelt (vgl. Abschnitt „Full Cross Product" auf Seite 140).

5.2.7. Projektion

Die Projektion ist die Auswahl der Spalten des ResultSets. Diese wird in einem select-Statement durch die Angabe der Properties $\texttt{prop}_1, \ldots, \texttt{prop}_{n_{prop}}$ getroffen. Zur Befüllung der Zellen mit den korrespondierenden Values müssen zunächst die Zeilen mit den Multimodell-Elementkombinationen ermittelt werden (interne ID-Tabelle $ID_{k,j}$). Die dazu notwendigen Schritte wurden in den vorangegangenen Abschnitten erläutert.

Die Erstellung der öffentlichen Value-Tabelle $V_{i,j}$ erfolgt durch eine Abbildung der IDs auf Values[27]. Dazu wird – je nach Spezifikationsform des jeweiligen Properties `prop` – die Methode *getPropertyByPath()* der Element-Query-Erweiterung oder *accessProperty()* der Property-Accessor-Erweiterung aufgerufen. Als Argument wird die ID des entsprechenden Multimodell-Elements nach Maßgabe der internen ID-Tabelle $ID_{k,j}$ übergeben. Diese ermittelt sich aus der gleichen Zeile j wie der betrachtete Value und besitzt den gleichen Element-Typ et_k [28] wie das zugeordnete Property p_i:

$$\begin{aligned} getValue(id_{k,j}) \quad &:= \quad \begin{cases} getPropertyByPath(id_{k,j}) & \text{für PropertyByPath-Ausdruck} \\ accessProperty(id_{k,j}) & \text{für PropertyAccessor-Ausdruck} \end{cases} \\ id_{k,j} \mapsto v_{i,j} \quad &:= \quad getValue(id_{k,j}) \,|\, et_k = et(p_i) \end{aligned} \tag{5.26}$$

Aus Gleichung 5.26 folgt, dass nicht jede ID der internen ID-Tabelle zur Ermittlung eines Values herangezogen werden muss. Der Grund ist, dass in einem select-Statement die Angabe der Projektion unabhängig von der Elementkombination erfolgt, d. h. für das ResultSet müssen nicht zwangsläufig Properties aus *jedem* kombinierten Element-Typ ausgewählt werden.

[26] vgl. auch Beispiel 13 auf Seite 115
[27] vgl. auch Abschnitt 5.2.2 auf Seite 135
[28] vgl. auch Abbildung 5.6 auf Seite 136

5.2.8. Evaluation der where-Klausel

Die Selektion – also die Auswahl der Zeilen des ResultSets – ergibt sich beim Multimodell-Filtern aus der oben beschriebenen Linkauswertung sowie aus der Evaluation der where-Klausel. Diese ist aus der Technologie relationaler Datenbanken bekannt und wird im MMQL-Interpreter analog angewendet.

Das Kriterium der where-Klausel setzt sich aus beliebig vielen Unterkriterien zusammen. Die Reihenfolge ihrer Abarbeitung ergibt sich aus der syntaktischen Definition der Value-Operatoren und Property-Kriterien, einschließlich ihrer Vorrangspezifikation. Dadurch kann der Parser den abstrakten Syntaxbaum auch in Bezug auf die Unterkriterien der where-Klausel sortiert anlegen[29]. Der MMQL-Interpreter kann die einzelnen Unterkriterien danach knotenweise abarbeiten, wobei die Bearbeitung zunächst an den Kindknoten delegiert wird, wenn dieser kein evaluierter Wahrheitswert sondern wiederum ein Unterkriterium ist.

Im Kern besteht das Kriterium aus der logischen Verknüpfung von Aussagen (Assertions) über Values. Diese werden in MMQL-Anweisungen – wie bei der Projektion – durch ihre Properties benannt. Zum Zeitpunkt der Evaluation werden diese durch ihre realen Values ersetzt, wodurch sich konkrete Wahrheitswerte ergeben. So wird für jede Zeile j der öffentlichen Value-Tabelle $V_{i,j}$ des ResultSets das Kriterium evaluiert, indem die Values dieser Zeile für die entsprechenden Assertions eingesetzt werden. Evaluiert das Kriterium insgesamt zu `false` oder `null`[30], wird die Zeile j aus dem ResultSet entfernt. Das endgültige ResultSet besteht demnach nur noch aus denjenigen Zeilen, für welche das Kriterium zu `true` evaluiert.

5.3. Links erstellen

5.3.1. Einwertige Elementkombination

Eine Anweisung zum Erstellen von Links (AddLinks-Klausel, `addl`) wird in MMQL als Teil einer Änderungsanweisung für ein Linkmodell (UpdateLM-Klausel) formuliert. Sie besteht somit aus der Angabe des zu ändernden Linkmodells `lm`, den Angaben zum Modus der Duplizität ($\texttt{mod}_{\texttt{DUP}}$) und Arität ($\texttt{mod}_{\texttt{ARI}}$), s zu verlinkenden Element-Typen `et` mit ihrem jeweiligen Modus der Kardinalität ($\texttt{mod}_{\texttt{KARD}}$) sowie dem Kriterium, welches die Bedingungen zur Erzeugung eines individuellen Links beschreibt (Assertion, `assrt`)[31]:

$$\begin{aligned} \texttt{ADDL} &:= \text{Unendliche Menge valider UpdateLM-AddLinks-Klauseln} \\ \texttt{addl} \in \texttt{ADDL} &:= \{\texttt{lm}, \texttt{mod}_{\texttt{DUP}}, \texttt{mod}_{\texttt{ARI}}, \{\texttt{mod}_{\texttt{KARD}}, \texttt{et}\}_1, \ldots, \{\texttt{mod}_{\texttt{KARD}}, \texttt{et}\}_s, \texttt{assrt}\} \end{aligned} \quad (5.27)$$

Abbildung 5.12 auf der nächsten Seite zeigt das Interpreterverfahren zum Erstellen von Links. Die o. g. Parameter bilden dabei die Eingangsgrößen. In einem ersten Schritt ① wird eine mögliche unzulässige Arität überprüft. Im Modus *strict* müssen die zu verlinkenden Element-Typen $et_1 \ldots et_s$ aus allen verlinkten Elementarmodellen $\breve{EM}$ des Linkmodells lm stammen, d. h. die in der Anweisung verwendeten Elementarmodelle $\widehat{EM}$ müssen mit $\breve{EM}$ gleich sein. Jede Element-Spezifikationsklausel $\texttt{et}_\texttt{s}$ beinhaltet inhärent eine Elementarmodell-Spezifikationsklausel $\texttt{ems}_\texttt{s}$,

[29] vgl. auch Abschnitte „Value-Operationen" auf Seite 100 und 4.3.2 auf Seite 103 sowie Abbildung 5.1 auf Seite 130
[30] vgl. auch Abschnitt „Dreiwertige Logik" auf Seite 105
[31] vgl. auch Abschnitt „Linkerzeugung" auf Seite 120, sowie Quelltext A.3 auf Seite 188 Zeilen 8, 9 und 33

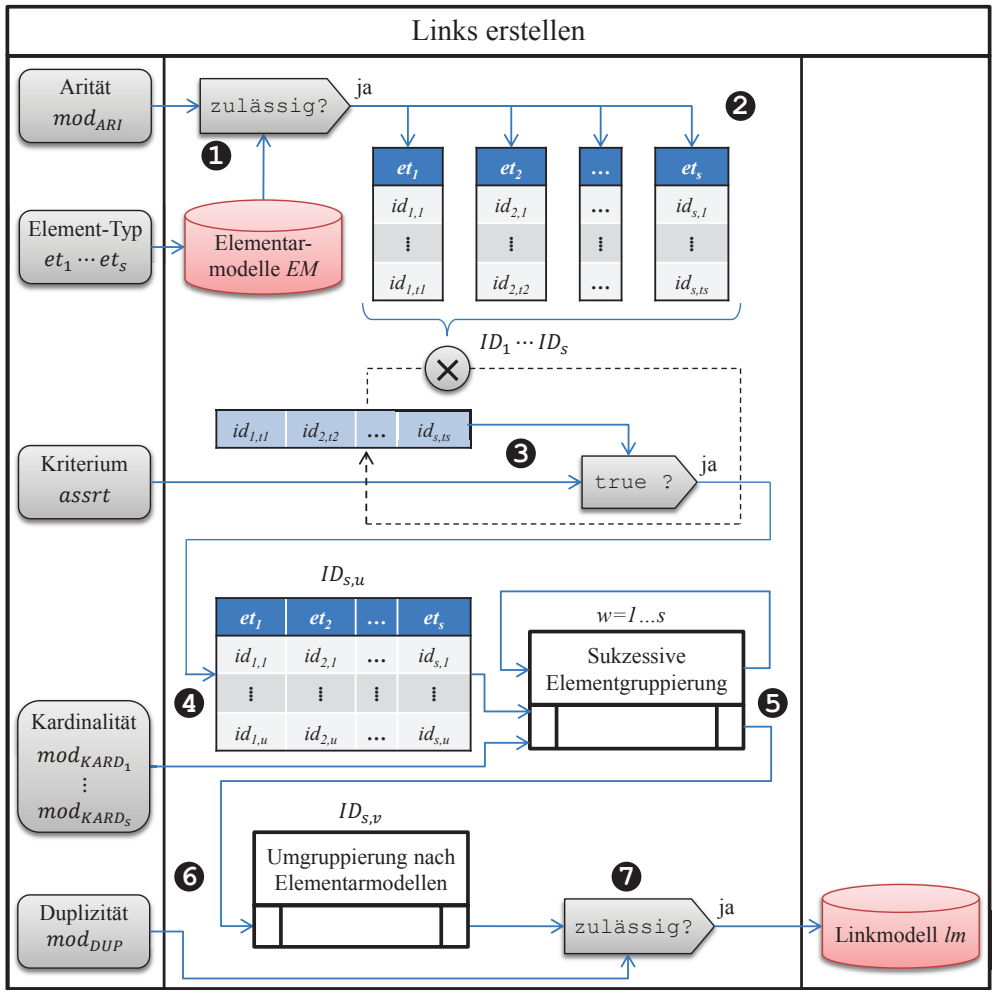


Abbildung 5.12.: Interpreterverfahren zum Erstellen von Links. ① Überprüfung auf zulässige Arität; ② Auflösen der Elemente; ③ Kombination der IDs und zeilenweise Evaluation des Kriteriums; ④ Zwischenspeichern der einwertigen Links; ⑤ Umwandlung in mehrwertige Links; ⑥ Umgruppierung nach Elementarmodellen; ⑦ Überprüfung zulässiger Duplizität.

wodurch das besitzende Elementarmodell eines Element-Typs immer angegeben ist:

$$\begin{aligned} \mathtt{et_s} &\Rightarrow \mathtt{ems_s} \\ \widehat{EM} &= \bigcup_{x=1}^{s} interpreteElementaryModelSpec(\mathtt{ems}_x) \\ \mathtt{mod_{ARI}} = \text{'strict'} &\Rightarrow \widehat{EM} = \breve{EM} \end{aligned} \tag{5.28}$$

In jedem Fall müssen in einer Anweisung zum Erstellen von Links Element-Typen aus mindestens zwei Elementarmodellen benutzt werden, damit die Anforderung aus Gleichung auf Seite 66 erfüllt ist:

$$\forall \widehat{EM} \quad : \quad \left|\widehat{EM}\right| \geq 2 \tag{5.29}$$

Im zweiten Schritt ② werden alle IDs der in den Elementarmodellen EM vorhandenen Elemente der spezifizierten Typen $et_1 \ldots et_s$ ermittelt. Dieser Vorgang erfolgt nach der in Abschnitt „Elemente auflösen und Interpretation der from-Klausel" auf Seite 140 beschriebenen Methode. Es entstehen s Mengen $ID_1 \ldots ID_s$ mit einer spezifischen Anzahl von t_s IDs. Dabei soll gelten, dass für jeden Element-Typ mindestens ein Element im entsprechenden Elementarmodell vorhanden sein soll. Eine explizite Verlinkung von Element-Typen, welche im gegebenen Elementarmodell nicht belegt sind, würde ein undefiniertes Verhalten darstellen und möglicherweise inkonsistente Links erzeugen:

$$\forall ID_x \,|\, x = 1 \ldots s \quad : \quad t_x \geq 1 \tag{5.30}$$

Das Prinzip der Erstellung von Links ist die Überprüfung jeder möglichen Element-Kombination auf Erfüllung des angegebenen Kriteriums. Dazu wird das Kreuzprodukt der ID-Mengen gebildet. Die Anzahl aller Kombinationen n_{komb} ergibt sich aus dem Produkt der spezifischen Anzahl Elemente je Typ:

$$\begin{aligned} ID_1 \times ID_2 \times \quad \ldots \quad \times ID_s \\ n_{\text{komb}} \quad = \quad \prod_{x=1}^{s} t_x \end{aligned} \tag{5.31}$$

Methodisch besteht jedoch keine Notwendigkeit, die vollständige Element-Kombination im Arbeitsspeicher des Interpreters vorzuhalten. Vielmehr ist es ausreichend, jede einzigartige Kombination flüchtig zu erstellen und dafür das Kriterium zu evaluieren (Schritt ③). Die Funktionsweise der Auswertung der where-Klausel wurde bereits in Abschnitt 5.2.8 auf Seite 148 beschrieben. Diejenigen Kombinationen, bei denen das Kriterium zu `true` evaluiert, werden zeilenweise in eine Tabelle von IDs $ID_{s,u}$ überführt. Diese enthält nach Überprüfung aller Kombinationen in jeder Zeile $1 \ldots u$ *einen* neuen Link (Schritt ④). Ein solcher Link ist einwertig – d. h. er ist eine einzigartige Kombination von s nach Typen gruppierten Multimodell-Elementen mit einer Kardinalität von $1:1:\ldots:1$. Keine der IDs aus $ID_{s,u}$ ist unbelegt ($\varnothing$, `null`), da die zu kombinierenden Grundmengen nach Gleichung 5.30 nicht leer sind. Ein Link mit solchen Eigenschaften ist der Grundtyp aller Multimodell-Links innerhalb der automatisierten Linkverarbeitung.

Falls der Nutzer für einen oder mehrere Element-Typen eine Kardinalität von n spezifiziert hat,

werden in einem Unterverfahren ⑤ die einwertigen Links zu mehrwertigen Links (bezogen auf den Element-Typ) umformuliert. Das dazu notwendige sukzessive Gruppieren von Elementen wird in Abschnitt 5.3.2 erläutert. Dabei reduziert sich potentiell die Anzahl der Links von u auf v, wobei gilt: $u \geq v$.

Die nun vorliegenden mehrwertigen Links $ID_{s,v}$ sind immer noch nach Element-Typen et gruppiert. Zur adäquaten Ablage im Linkmodell lm müssen diese in Schritt ⑥ nach Elementarmodellen em umgruppiert werden. Dazu werden alle IDs der Element-Typen des gleichen Elementarmodells zusammengefasst. Das Verfahren entspricht der Umkehrung von Schritt ② des Link Breakup in Abbildung 5.10 auf Seite 143.

Schlussendlich wird in Schritt ⑦ jeder Link einzeln dem Linkmodell lm hinzugefügt. Dabei wird im Modus *distinct* zuvor überprüft, ob ein gleicher Link nicht bereits in lm vorhanden ist. Dieser Modus der Duplizität gilt automatisch auch für den Arität-Modus *strict*.

5.3.2. Sukzessive Elementgruppierung

Das Unterverfahren der sukzessiven Elementgruppierung dient der Umwandlung von einwertigen Links (bezogen auf den Element-Typ) in mehrwertige Links. Dabei erfolgt eine Bündelung von Links schrittweise und nacheinander für alle Element-Typen $et_1 \ldots et_s$. Die Reihenfolge der angegebenen Element-Typen in der MMQL-Klausel hat einen Einfluss auf das Endergebnis.

Abbildung 5.13 auf der nächsten Seite zeigt einen solchen Schritt am Beispiel des zweiten Element-Typs ($w = 2 \Rightarrow et_2$) einer fiktiven Anweisung. Die Eingangsgrößen sind der Modus der Kardinalität des im momentanen Durchlauf zu betrachtenden Element-Typs $\texttt{mod}_{\texttt{KARD}_w}$ sowie die ID-Tabelle mit potentiell mehrwertigen Links des vorangegangenen Bearbeitungsschritts $ID_{s,v}^{w-1}$. Im ersten Bearbeitungsschritt ist dies die Tabelle der einwertigen Links $ID_{s,u}$. Bei jedem weiteren Schritt ist die Ausgangsgröße eine Tabelle $ID_{s,v}^{w}$ potentiell mehrwertiger Links, bei welcher – sofern möglich – die Linkanzahl u reduziert wurde, indem Links nach dem zu betrachtenden Element-Typ et_w mehrwertig zusammengefasst wurden.

Wurde vom Anwender die gewünschte Kardinalität des Element-Typs et_w mit 'one' angegeben, so dürfen keine Links zu mehrwertigen zusammengefasst werden und die Ergebnistabelle dieses Bearbeitungsschritts ist gleich der Eingangstabelle:

$$\texttt{mod}_{\texttt{KARD}_w} = \text{'one'} \quad \Rightarrow \quad ID_{s,v}^{w} = ID_{s,v}^{w-1} \tag{5.32}$$

Im Falle von 'many' sind zeilenweise jeweils diejenigen Links ID_{v}^{w-1} nach et_w zusammenzufassen, welche gleiche IDs bei den jeweils anderen Element-Typen besitzen. Dazu müssen alle Links (Zeilen x, y) miteinander verglichen werden:

$$ID_{w,v}^{w} \quad = \quad \bigcup_{x,y=1}^{v_{w-1}} ID_{w,x}^{w-1} \mid ID_{s,x}^{w-1} = ID_{s,y}^{w-1}, s \neq w \tag{5.33}$$

Links, welche bei anderen Element-Typen keine gleichen IDs besitzen, bleiben auch immer einwertig (bezogen auf et_w) erhalten. Daher ist in o. g. Gleichung $x = y$ zulässig und beschreibt den Fall, dass eine Zeile mit sich selbst verglichen wird. Alle einwertigen Links, welche auch in anderen mehrwertigen Links vorkommen, werden im Modus *strict* in Schritt ⑦ (Abbildung 5.12, Duplizität überprüfen) ausgesondert.

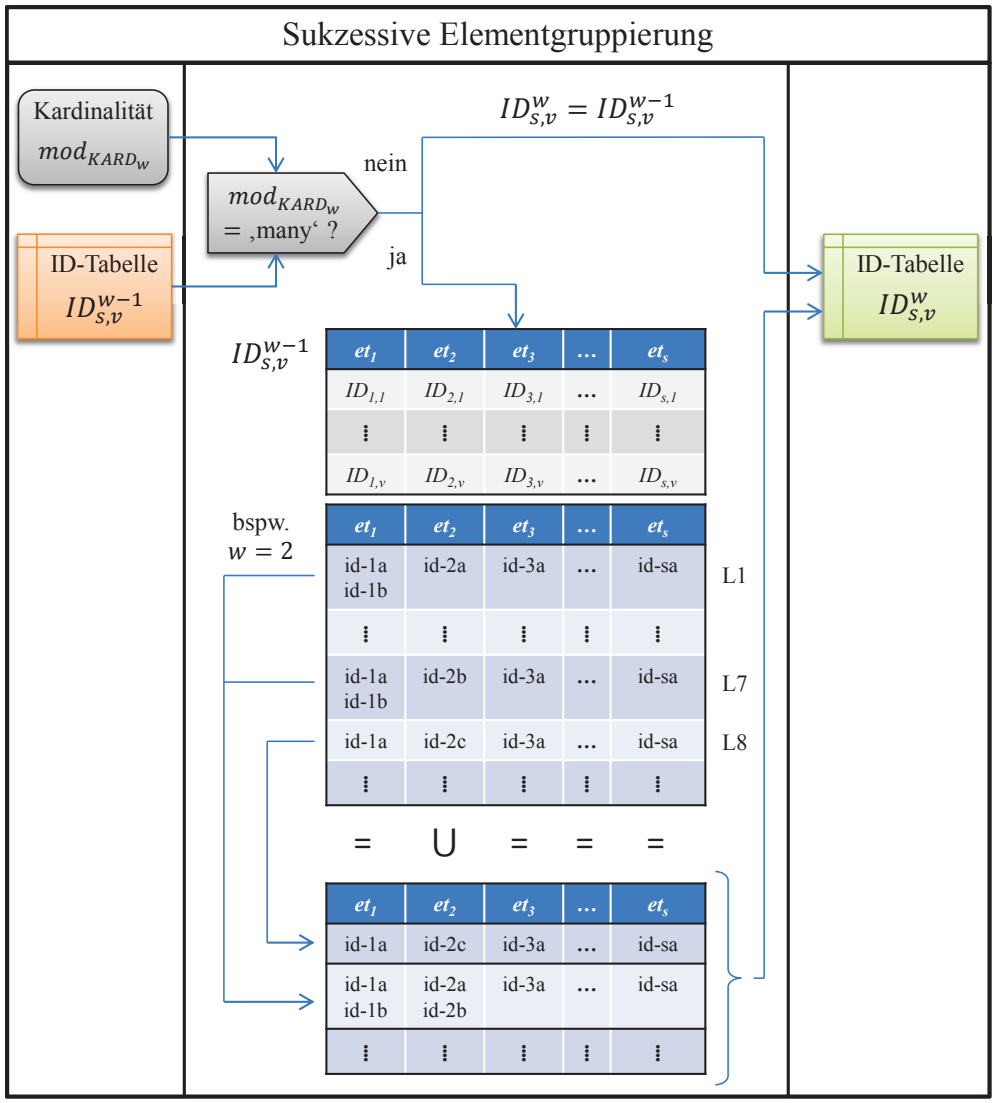


Abbildung 5.13.: Interpreterverfahren der sukzessiven Elementgruppierung; dargestellt am Beispiel $w = 2$.

Im Beispiel $w = 2$ der Abbildung 5.13 werden demnach die beiden ursprünglichen Links *L1* und *L7* mit $\{\texttt{id} - \texttt{2a}, \texttt{id} - \texttt{2b}\}$ für et_2 zusammengefasst, weil die ID-Mengen der anderen Element-Typen gleich sind:

- $\{\texttt{id} - \texttt{1a}, \texttt{id} - \texttt{1b}\}$ für et_1
- $\{\texttt{id} - \texttt{3a}\}$ für et_3
- ...
- $\{\texttt{id} - \texttt{sa}\}$ für et_s

Link *L8* bleibt dagegen für et_2 einwertig erhalten, da keine anderen Links für $et_1, et_3, \ldots, et_s$ gleiche ID-Mengen besitzen.

5.4. Links löschen

Ebenso wie das Erstellen von Links, wird die Anweisung zum Löschen von Links (DeleteLinks-Klausel) als Teil einer Änderungsanweisung für ein Linkmodell (UpdateLM-Klausel) formuliert. Dabei werden das zu ändernde Linkmodell, mindesten zwei Element-Typen sowie das Kriterium zum Löschen eines konkreten Links angegeben.

Während der Abarbeitung werden alle Links des Linkmodells daraufhin überprüft, ob

1. der Link mindestens *ein* Element jedes angegeben Element-Typs beinhaltet
2. das angegebene Kriterium für die Elemente des Links zu `true` evaluiert.

Wenn beide Kriterien zutreffen, wird der betrachtete Link vollständig aus dem Linkmodell entfernt.

5.5. Diskussion und Resümee

Die Multimodell-Abfragesprache MMQL und der zugehörige Interpreter bilden die Kernkomponenten zur Erschließung multimodellbasierter Informationsräume. Werden die hier beschriebenen Verfahren in eine entsprechende Softwarekomponente umgesetzt, so muss das Resultat u. a. den Anforderungen *Effizienz, Korrektheit* und *Robustheit* gerecht werden. Während Robustheit – also vernünftiges Verhalten trotz unerwarteter Ereignisse – hauptsächlich zum Implementierungszeitpunkt mit den Mitteln der Softwaretechnik realisiert wird, müssen die Aspekte *Effizienz* und *Korrektheit* bereits für die zugrunde liegenden Methoden analysiert werden.

5.5.1. Effizienz

Zur Erzeugung von Links sowie zur Ermittlung von Multimodell-Views ist letztendlich ein polynomialer Rechenaufwand in Abhängigkeit der Datenmenge $n \to \infty$ notwendig. Die Komplexität wird durch die Anzahl der vorhandenen Multimodell-Elemente je verwendetem Element-Typ bestimmt:

$$O\left(n_{et_1} \cdot n_{et_2} \cdot \ldots \cdot n_{et_m}\right) \tag{5.34}$$

Zwar ist die Anzahl der Elemente unterschiedlichen Typs in den Elementarmodellen prinzipiell eine unabhängige Größe – zur Abschätzung des größten Aufwands muss jedoch davon ausgegangen werden, dass in der Praxis

1. innerhalb desselben Elementarmodells eine steigende Anzahl Elemente vom Typ et_1 auch eine steigende Anzahl Elemente vom Typ et_2 nach sich zieht. Bspw. werden für mehr LV-Positionen i. d. R. auch mehr LV-Gruppen angelegt und mehr Wände implizieren auch mehr Decken, Fenster, Türen und Räume.
2. Elementarmodelle mit einer großen Anzahl von Elementen wiederum mit entsprechend großen und detaillierten dritten Elementarmodellen verlinkt werden.

Im Extremfall kann vereinfachend davon ausgegangen werden, dass die Anzahl der vorhandenen Elemente aller beteiligter Typen gleich wächst, wodurch der folgende polynomiale Aufwand entsteht:

$$O\left(n^m\right) \tag{5.35}$$

Nach Saake & Sattler (2006, S. 199 ff.) sind Aufgabenstellungen mit polynomialen Aufwand zwar noch effizient lösbar, dennoch werden – besonders bei großen m – die Grenzen der praktischen Berechenbarkeit schnell erreicht. Für die effiziente Anwendung der vorgestellten Methoden ergeben sich daher folgende Schlussfolgerungen:

1. Verringerung der Anzahl der Element-Typen m

   Die Verringerung der Anzahl der Element-Typen in einer MMQL-Anweisung kann nur auf Nutzerebene geschehen. Sie bedeutet einen Eingriff in die Semantik der Anweisung und wirkt sich direkt auf das Ergebnis aus. Wie auch bei anderen Programmiersprachen muss ein Nutzer bei MMQL-Anweisungen die Auswirkungen auf die Laufzeiteffizienz beachten. So sollten bspw. nur wirklich benötigte Element-Typen abgefragt werden.

   Des Weiteren kann es sinnvoll sein, *eine* komplexe Abfrage mit vielen Element-Typen in mehrere kleine mit wenigen Element-Typen aufzuteilen (bspw. auch mit vorausgegangenen Select-Ergebnismengen[32]) und die ResultSets manuell zusammenzuführen.

2. Verringerung der Anzahl der Elemente n

   a) Auf Multimodell-Ebene

      Multimodelle sollten auf Nutzerebene mit Rücksicht auf die spätere Laufzeiteffizienz erstellt werden. So ist für Elementarmodelle mit geringem geforderten Detaillierungsgrad dieser auch möglichst einzuhalten. Auch eine hohe Anzahl von Links beeinflusst die spätere Rechenzeit negativ. Es sollten nur relevante und – wenn dies zum Erstellungszeitpunkt bereits bekannt ist – wirklich in Abfragen benutzte Links erstellt werden. Die Gruppierung von Links erfolgt nach ihrem Zweck in Linkmodellen. Dabei sind mehrere Linkmodelle mit jeweils weniger Links wenigen Linkmodellen mit jeweils vielen Links vorzuziehen.

   b) Zum Zeitpunkt der Interpretation

      Implementierungen können Optimierungen dahingehend vornehmen, dass nur so wenig wie möglich Elemente n_{et} miteinander kombiniert werden. In den vorgeschlagenen Methoden werden beispielsweise zunächst alle n_{et} kombiniert, um später die Kriterien der where-Klausel für alle Kombinationen zu evaluieren. Diese Reihenfolge ergibt sich aus dem *potentiellen* Vorhandensein elementarmodellübergreifender Property-Kriterien, deren Auswertung nur für kombinierte Datenelemente realisierbar ist. *Tatsächlich* ist es jedoch möglich, sämtliche andere Kriterien bereits *vor* der

[32] vgl. Absatz „Vorausgegangene Select-Ergebnismengen" auf Seite 108

Kombination auf die Grundmenge n_{et} anzuwenden, um diese dadurch zu reduzieren. Auf diese Weise verringert sich die Anzahl an Kombinationen um ein Vielfaches.

3. Verringerung der Rechenzeit je Rechenschritt und n

   Weitere generelle Optimierungen können mit den Mitteln der Softwaretechnik realisiert werden. Dies betrifft sämtliche Rechenoperationen. Beispielsweise lässt sich sequentielles Suchen in ID-Listen durch vorheriges Sortieren in binäres Suchen umwandeln, Teilergebnisse können indiziert und zwischengespeichert werden und Prozesse können, soweit möglich, parallelisiert werden. Weiterhin können MMQL-Anfragen vor ihrer Ausführung speziell optimiert werden. Beispielsweise werden dabei die Kriterien der where-Klausel vereinfacht, die Reihenfolge der Kombination geändert oder langsame gegen schnelle Anweisungen ausgetauscht. Einen Überblick über Anfrageoptimierungen bei SQL-Datenbanken geben Saake et al. (2005, Kap. 7.), welche als Anregungen bei der Implementierung eines MMQL-Interpreters genutzt werden können.

5.5.2. Korrektheit

Reihenfolge von Anweisungen

Sowohl die Vorgänge der sukzessiven Elementgruppierung (Links erstellen) als auch der sukzessiven Linkauswertung (Ermittlung von Multimodell-Views) produzieren Ergebnisse, welche von der *Reihenfolge* der linkedwith- bzw. add links between-Anweisungen abhängig sind.

So ergeben sich für die Linkauswertung

```
7 from "LV 1.X83"."Item"
8 cross linkedwith "BW.ifc"."IfcWall"
9 right linkedwith "Vorgangsmodell 1.xml"."Activity"
```

potentiell andere Ergebnisse als für

```
7 from "LV 1.X83"."Item"
8 right linkedwith "Vorgangsmodell 1.xml"."Activity"
9 cross linkedwith "BW.ifc"."IfcWall"
```

Im ersten Fall wird zuerst ein Kreuzprodukt zwischen allen LV-Positionen und allen IFC-Wänden hergestellt und danach alle Vorgänge dem ResultSet hinzugefügt. Dabei sind Positionen *und* Wände mit `null` aufzufüllen, falls für einen Vorgang kein Link mit einer entsprechenden Position-Wand-Kombination existiert.

Im zweiten Fall wird zunächst die Kombination von LV-Positionen und allen Vorgängen hergestellt. Dabei werden *nur* LV-Positionen mit `null` aufgefüllt, falls ein Vorgang keinen Link zu einer Position besitzt. anschließend wird für all diese Position-Vorgang-Kombinationen das Kreuzprodukt mit allen IFC-Wänden gebildet.

Ebenso geschieht beim Erstellen von Links das Zusammenfassen zu mehrwertigen Links in Abhängigkeit der Reihenfolge der spezifizierten Element-Typen. Die folgende Anweisung

```
7  update linkmodel "LM2"
8      add links between
9          one "LV 1.X83"."Item",
10         many "BW.ifc"."IfcWall",
11         many "Vorgangsmodell 1.xml"."Activity"
```

fasst Links zuerst nach IFC-Wänden und dann nach Vorgängen zusammen, während mittels

```
update linkmodel "LM2"
    add links between
        one "LV 1.X83"."Item",
        many "Vorgangsmodell 1.xml"."Activity",
        many "BW.ifc"."IfcWall"
```

Links zunächst nach Vorgängen und später nach IFC-Wänden gruppiert werden. Dadurch entstehen möglicherweise andere mehrwertige Links als im ersten Fall.

Das reihenfolgeabhängige Verhalten des Interpreters gilt als korrekt, da es Teil der Semantik von MMQL ist. Die Sprache bietet über diesen Mechanismus weitere Steuerungsmöglichkeiten zur fein abstimmbaren Erschließung domänenübergreifender Informationsräume. Das reihenfolgeabhängige Verhalten kommt außerdem nicht bei Abfragen mit Standard-Modi oder bei der rein einwertigen Linkerzeugung zum Tragen. Die Erstellung komplexerer MMQL-Anweisungen kann im Einzelfall jedoch komplizierter werden. Um ein bestimmtes Ergebnis zu erzielen, sollte ein Nutzer daher solche Anfragen iterativ erstellen, Zwischenergebnisse anhand von Testdaten nachvollziehen und schrittweise weitere Statements hinzufügen. Mit *M2A2* wird in Abschnitt 6.1 auf Seite 160 ein Werkzeug für eine solche heuristische Vorgehensweise vorgestellt.

Korrelation von Linkinterpretation und Kardinalität

Die Ergebnisse einer Linkauswertung sind stark abhängig vom Aufbau der vorhandenen Links der Datenbasis. Dabei ist insbesondere der Zusammenhang zwischen dem Modus der Linkinterpretation und der vorhandenen Kardinalität der Links von Bedeutung. Die Kardinalität eines Links bezieht sich hier nicht nur auf die Anzahl der verlinkten Elemente. Da Linkerzeugung und -abfrage in MMQL auf Basis von Element-Typen erfolgen, ist es vielmehr relevant, ob in einem Link *ein* oder *mehrere* Elemente des gleichen Typs auftreten[33].

Tabelle 5.2 zeigt die Korrelation von Linkinterpretation und Kardinalität. Der Modus *transitive* wurde hier nicht aufgeführt, da dieser sich gewissermaßen über die real vorhandene Link-Datenbasis hinwegsetzt und auch sonst keinen direkten Bezug zur Kardinalität aufweist. Für einwertig (in Bezug auf jeden benutzten Element-Typ) erstellte Links ist es für das Abfrageergebnis unerheblich, welcher Modus der Linkinterpretation angewendet wird. Das Resultat ist identisch. Lediglich aus Gründen der Performance ist der Modus *standard* vorzuziehen, da dort keine Überprüfung auf vollständige (mehrwertige) Reproduzierbarkeit eines Links erfolgen muss.

Links, welche mindestens für einen Element-Typ mehrwertig sind, sind im Modus *strict* abzufragen, falls diese Mehrwertigkeit ausdrücklich im Ergebnis widergespiegelt werden soll.

Tabelle 5.2.: Korrelation von Linkinterpretation und Kardinalität

Linkinterpretation / Kardinalität	Strict	Standard
$1:1:\ldots:1$	+	++
$n:x_1:x_2:\ldots:x_i \mid x \in \{1,n\}$	++	+−

[33] vgl. dazu auch Abschnitte 5.2 auf Seite 134 sowie 5.3 auf Seite 148

Dies ist dann notwendig, wenn die kleinste Informationseinheit aus mehreren typgleichen Datenelementen besteht – bspw. bei der Zuordnung von einem Arbeitsvorgang *Aushub* zu den beiden Baugeräten *Bagger* und *LKW*. Der Modus *strict* sorgt dafür, dass dieser Link immer vollständig verarbeitet wird. Sollen jedoch auch *Teile* eines mehrwertigen Links fachlich bedeutsam sein – bspw. die Beziehung von *Aushub* nur zu *Bagger* – so sind solche Links im Modus *standard* abzufragen. Dieser Modus liefert potentiell mehr Resultate, da die Randbedingung der mehrwertigen Vollständigkeit entfällt. Für explizit mehrwertige Links ist *standard* jedoch nicht der bevorzugte Abfragemodus – die Links hätten ansonsten einwertig erstellt werden sollen.

Die in Tabelle 5.2 dargestellten Beziehungen stellen somit zugeordnete Intentionen zwischen Linkerzeugung und -abfrage dar. Bspw. beabsichtigt ein Nutzer, welcher Links mehrwertig statt einwertig erstellt, damit eine spätere Abfrage im Modus *strict*. Die Möglichkeit in MMQL zur freien Moduswahl zum Abfragezeitpunkt erlaubt es Dritten, dieser Intention – sofern sie bekannt ist – nachzukommen oder den Informationsraum bewusst auf eine andere Art und Weise zu erschließen.

5.5.3. Resümee

Die Interpretation von MMQL-Anweisungen ist Voraussetzung zur Anwendung der Multimodell-Methode auf realen Daten. Ziel ist die Änderung des internen Zustands von Multimodellen (Erzeugung, Änderung oder Löschen von Linkmodellen und Links) sowie die Erstellung von ResultSets als Repräsentationsform von Multimodell-Views.

Durch die Delegation des Zugriffs auf heterogene Elementarmodell-Daten an die jeweilige Parser-Erweiterung, kann die Abarbeitung von MMQL-Statements in einer generischen Art und Weise erfolgen. In diesen Erweiterungspunkten findet der Übergang von den Multimodell-Konzepten zum individuellen Datenformat statt; bspw. indem Parser die IDs aller Elemente mit dem angegebenen Element-Typ-Namen liefern. Ebenso können alle Links sowie benannte Elementmengen (z. B. externe Filter) als Mengen von IDs aufgefasst werden. Der Interpreter kann somit entscheiden, welche Datenelemente für die weitere Abarbeitung relevant sind, indem er lediglich auf solchen ID-Mengen operiert.

Beim Multimodell-Filtern wird versucht, vorhandene Links für die angegebenen Parameter stufenweise zu reproduzieren (*Sukzessive Linkauswertung*). Ein ResultSet repräsentiert somit ein *virtuelles, gefiltertes Linkmodell*: seine interne ID-Tabelle stellt die einwertigen Elementkombinationen entsprechend des Filters dar. Beim Erzeugen von Links werden hingegen alle möglichen Elementkombinationen überprüft, ob sie dem vorgegebenen Kriterium entsprechen.

Obwohl diese grundlegenden Strategien prinzipiell unkompliziert sind, ist ihre tatsächliche Umsetzung komplex. Dies liegt im Wesentlichen daran, dass folgende Aspekte zu integrieren sind:

1. Links sind potentiell mehrwertig
2. IDs sind nur innerhalb *eines* Elementarmodells eindeutig
3. IDs sind innerhalb von Links nur nach Elementarmodellen gruppiert, MMQL-Statements basieren jedoch auf Element-Typen

Im Resultat ist der Interpreter Teil der Semantik von MMQL – bspw. dadurch, dass die Reihenfolge der Anweisungen Einfluss auf das Bearbeitungsergebnis besitzt. Daher ist es wichtig, Anwendern eine Testumgebung zur Verfügung zu stellen, mit der MMQL-Statements

ausgeführt und das Ergebnis analysiert werden kann. In Abschnitt 6.1 wird deswegen die M2A2-Plattform vorgestellt. Diese besitzt einen spezialisierten MMQL-Editor, eine Multimodell-Engine sowie grafische Viewer zur Analyse von MMQL-Statements, ResultSets sowie Link- und Elementarmodellen.

Kapitel 6.

Implementierung und Anwendung

> Auch in Wissenschaften kann man eigentlich nichts wissen. Es will immer getan sein.
>
> *(Johann Wolfgang von Goethe)*

In den vorangegangenen Kapiteln wurden neuartige Strukturen und Methoden zur Aufstellung und Erschließung domänenübergreifender Informationsräume entworfen. Im Folgenden werden Realisierbarkeit und Praktikabilität dieser Theorien überprüft.

Abschnitt 6.1 beschreibt die prototypische Implementierung einer universellen Multimodell-Software. Dabei stehen die Umsetzung der substanziell notwendigen Erweiterbarkeit und Modularisierung sowie die effiziente Entwicklung entsprechender Erweiterungen im Vordergrund. Beides sind wichtige Voraussetzungen zur Integration der multimodellbasierten Arbeitsweise in existierende Baufachanwendungen.

Der Multimodell-Container des Mefisto-Projekts ist ein Beispiel für die aufgabengerechte Spezialisierung des Multimodell-Konzepts. Seine Eigenschaften werden in Abschnitt 6.2 ebenso beschrieben wie die Zusammenarbeit mit dem generischen Konzept.

In Abschnitt 6.3 wird die baupraktische Aufgabenstellung der Ermittlung eines Zahlungsplans durch automatisierte Multimodellbildung und -filterung gelöst. Dieses Beispiel soll die prinzipielle Eignung der vorgestellten Methoden zur Bewältigung domänenübergreifender Aufgabenstellungen im Bauwesen bestätigen.

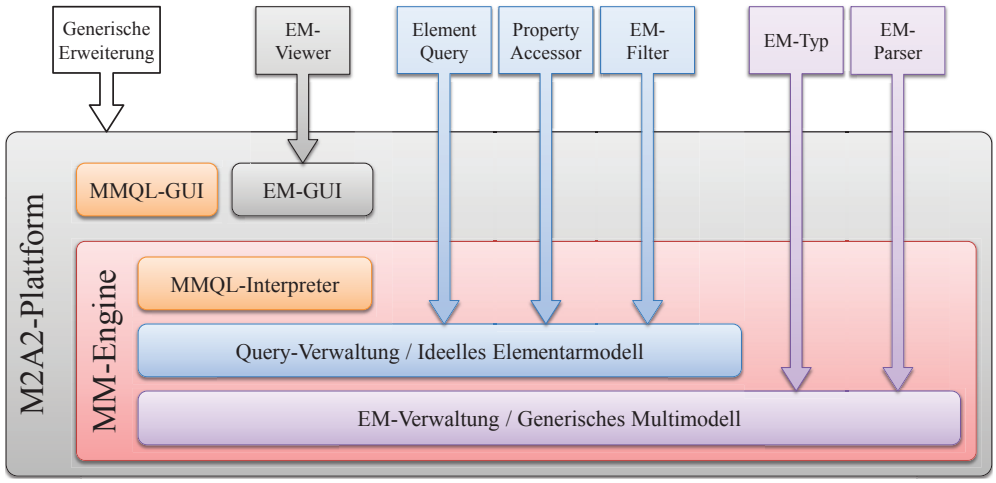


Abbildung 6.1.: Architektur der universellen Multimodell-Software M2A2

6.1. Universelle Multimodell-Software M2A2

6.1.1. Softwarearchitektur

In den vorangegangenen Kapiteln wurden Multimodelle als Struktur domänenübergreifender Informationsräume und die Programmiersprache MMQL als Erschließungsmethode dafür vorgestellt. Mit der *Multi-Model Assembly and Analyzing Platform* (M2A2) erfolgte eine prototypische softwaretechnische Realisierung dieser Konzeption. Aufgabe der Implementierung war die Überprüfung von Konsistenz und Anwendbarkeit der aufgestellten Theorien sowie die Erlangung eines Erfahrungsrückflusses zur Verbesserung der Konzepte.

Die M2A2-Plattform ist modular aufgebaut. Je nach Komposition der Module kann sie im Sinne der multimodellbasierten Arbeitsweise[1] wie folgt eingesetzt werden:

- als universelle MM-Software zur Erstellung, Manipulation und Analyse von Multimodellen
- als MM-Engine zur Erweiterung existierender Fachanwendungen um Multimodellfähigkeiten
- als Framework und Bibliothek zur Neuentwicklung spezialisierter Multimodellanwendungen
- als interaktives Entwicklungswerkzeug für MMQL-Anweisungen (MMQL-Editor)
- als Testumgebung für bekannte (bspw. Parser, Filter oder Viewer) sowie zukünftige (bspw. Mapper oder Validatoren) Elementar- und Multimodellkomponenten

Abbildung 6.1 zeigt die Softwarearchitektur der M2A2-Plattform in Hinblick auf die relevanten Module für die o. g. Einsatzgebiete. Im Wesentlichen besteht die Anwendung aus einer entkoppelten Multimodell-Engine (MM-Engine) und einer Plattform mit grafischer Oberfläche.

[1] vgl. Abschnitt 3.2 auf Seite 48

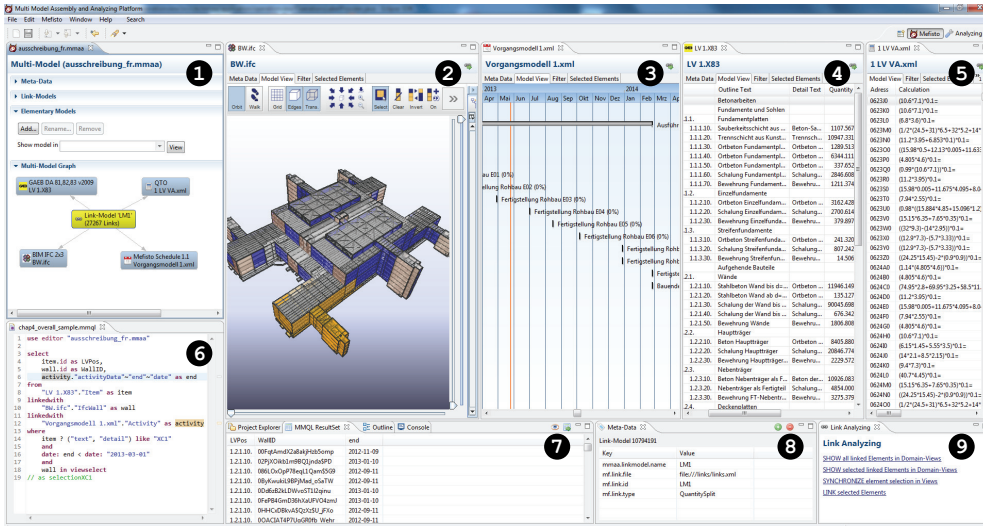


Abbildung 6.2.: Grafische Oberfläche der M2A2-Plattform; geöffnetes Multimodell analog zu Beispiel 1 auf Seite 87 ①; Elementarmodell-Viewer für das Bauwerksmodell ②, den Vorgangsplan ③, das Leistungsverzeichnis ④ und das Modell der Mengenansätze ⑤; ein Editor mit der MMQL-Abfrage aus Quelltext 4.1 ⑥; das zugehörige ResultSet ⑦; Sicht zur Bearbeitung von Multimodell-Metadaten ⑧; Sicht mit multimodellspezifischen Befehlen, z. B. manuelle Verlinkung selektierter Elemente ⑨.

Die MM-Engine – welche auch alleinstehend als Bibliothek oder Service nutzbar ist – übernimmt die wesentlichen Aufgaben zur Abbildung und Erschließung von multimodellbasierten Informationsräumen:

- Lesen und Schreiben von Multimodellen
- Verwaltung registrierter Elementarmodell-Typen und das Parsen von Elementarmodellen
- Verwaltung und Aufruf von registrierten Erweiterungen, welche den inhaltlichen Zugriff auf Elementarmodelle per Ideellem Elementarmodell bewerkstelligen
- Service zur Interpretation von MMQL-Anweisungen und Rückgabe des ResultSets

Die Plattform mit grafischer Oberfläche nutzt die MM-Engine und stellt darüber hinaus Editoren für Multimodelle und MMQL-Abfragen sowie die Verwaltung und Integration von Elementarmodell-Viewern zur Verfügung. Außerdem können Oberfläche und Logik der Plattform über eine generische Erweiterungsschnittstelle nahezu beliebig ausgebaut werden. Abbildung 6.2 zeigt die grafische Oberfläche der M2A2-Plattform. Die einzelnen Sichten (Fenster) können durch den Benutzer flexibel angeordnet werden.

6.1.2. Erweiterbare Ablaufumgebung

Als Ablaufumgebung der M2A2-Plattform wurde die Eclipse Rich Client Platform[2] (RCP; Daum, 2007) gewählt. Eclipse RCP basiert auf Equinox, einer Implementierung des OSGi-

[2] `http://www.eclipse.org/home/categories/rcp.php`

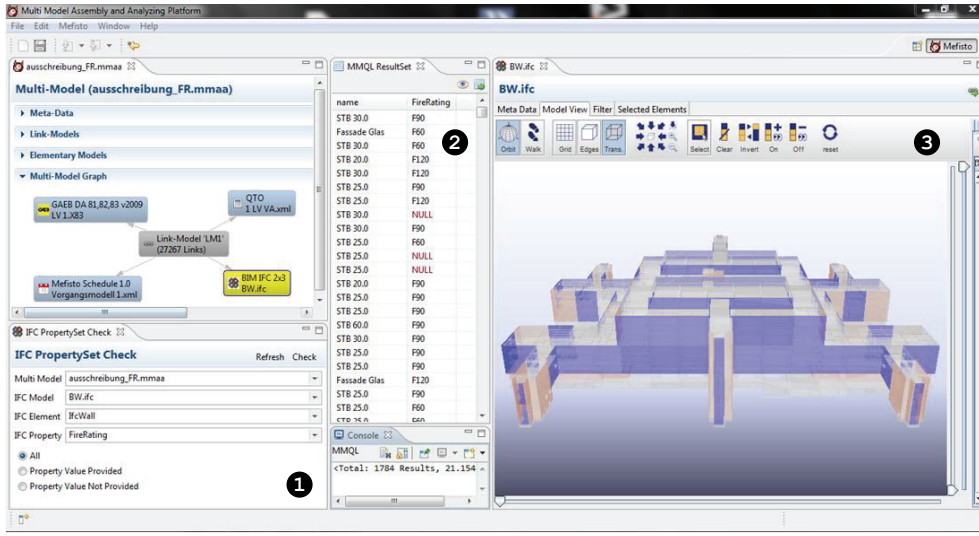


Abbildung 6.3.: Beispiel einer generischen M2A2-Erweiterung: Überprüfung von Eigenschaften gemäß IFC Property Set Definitions; formularbasierte Eingabe ①; tabellarisches Filterergebnis (MMQL-ResultSet) ②; Visualisierung des Filterergebnisses im IFC-3D-Viewer ③

Standards[3] (McAffer et al., 2010). OSGi bietet dynamische Modularität zur Laufzeit innerhalb einer Java Virtual Machine[4] (JVM). Software-Komponenten, sogenannte OSGi-Bundles, können gestartet, gestoppt und aktualisiert werden, ohne dass die JVM angehalten werden muss. OSGi-Bundles beschreiben ihre Interdependenzen in einer Manifest-Datei, wodurch eine feinkörnige Komposition der Laufzeitartefakte ermöglicht wird – eine Voraussetzung zum Aufbau von unterteilbaren Frameworks zur gezielten Erweiterung mit Drittmodulen (D'Anjou et al., 2004). Somit ist es möglich, das Multimodell-Framework, notwendige Bibliotheken und eine zunächst unbekannte Anzahl von Client-Komponenten in einfacher Weise zusammenarbeiten zu lassen und dennoch getrennt ausliefern zu können.

Darüber hinaus folgt Equinox einem Plugin-Konzept, indem es einen generischen Erweiterungsmechanismus via XML-Deskriptoren bereitstellt. Plugins können über diese so genannten Extension Points sowohl erweitert werden, als auch durch Registrierung an anderen Extension Points selbst zusätzliche Funktionalität anbieten. So kann durch die Nutzung der Eclipse RCP eine aufwändige Eigenimplementierung eines proprietären Erweiterungsmechanismus vermieden werden. Das Erstellen eigener Extension Points erlaubt die Konzeption einer spezifischen, kontrollierten Erweiterungsstrategie, wie sie von der Architektur der M2A2-Plattform gefordert wird (vgl. Abbildung 6.1). Folglich wurden für die in den Abschnitten 5.1.3 auf Seite 131 und 6.1.1 auf Seite 160 vorgestellten Multimodell-Erweiterungen entsprechende Extension Points angelegt. Anschließend konnten dafür Multimodell-Erweiterungen wie EM-Parser, EM-Viewer oder Property-Accessors implementiert werden. Ein Auszug umgesetzter Erweiterungen und deren Abhängigkeiten untereinander ist in Anhang C.2 auf Seite 198 illustriert.

Mit den Multimodell-Extension Points wurden Registries verbunden, welche die in einer

[3] http://www.osgi.org
[4] http://docs.oracle.com/javase/specs

konkreten Softwarekonfiguration vorhandenen Erweiterungen zur Laufzeit leichtgewichtig verwalten. Durch die Verwendung der XML-Deskriptoren kann Eclipse nämlich die Aktivierung von Plugins bis zu dem Zeitpunkt verschieben, an dem Java-Klassen des Plugins geladen werden müssen. Dieses sogenannte Lazy Loading ermöglicht einen schnellen Programmstart und einen kleineren Speicherbedarf, welcher bei der Arbeit mit großen Baufachmodellen wie IFC Vorteile bietet. Somit können der Multimodell-Plattform alle vorhandenen Erweiterungen bekannt gemacht werden, ohne dass diese geladen sein müssen.

Über die in der Eclipse RCP bereits vorhandenen Extension Points, kann die M2A2-Plattform nativ, außerhalb der vorgesehenen Multimodell-Erweiterungsstrategie, ausgebaut werden. So zeigt Abbildung 6.3 auf der vorangegangenen Seite eine neue Sicht ① zur Überprüfung von Properties in IFC-Modellen auf Konformität mit ausgewählten IFC Property Set Definitions[5]. Die Abarbeitung geschieht über eine interne MMQL-Abfrage – die Ergebnisdarstellung erfolgt u. a. über ein ResultSet ② und den per EM-Viewer-Erweiterung vorhandenen IFC-3D-Viewer ③. Eine Übersicht, wie die Eclipse RCP generell als Plattform für Anwendungen oder Frameworks verwendet und erweitert werden kann, geben Gamma & Beck (2004).

6.1.3. Implementierung von Datenmodellen

Die Implementierung von Datenmodellen erfolgte mithilfe geeigneter Bibliotheken[6]. So wurde bspw. Java CSV[7] zum Parsen von CSV-Daten verwendet sowie JSDAI[8] und Open IFC Tools[9] zum Parsen von IFC-Daten im SPF-Format.

XML-basierte Datenmodelle wie GAEB-DA-XML wurden mit dem Eclipse Modeling Framework (EMF[10]) umgesetzt. Mit EMF kann aus einem Ecore-Modell[11] Java-Code generiert werden, womit Instanzen dieses Modells u. a. erzeugt, manipuliert und serialisiert werden können. Außerdem propagieren die generierten Java-Klassen innere Zustandsänderungen, wodurch die Implementierung korrespondierender Viewer und Editoren nach dem Model View Controller-Pattern (MVC; Gamma et al., 2004) erleichtert wird. Die notwendigen Ecore-Modelle wurden durch Konvertierung aus den baufachlichen XML-Schemas gewonnen (vgl. dazu Steinberg et al., 2009). EMF ermöglicht darüber hinaus Introspektion und die generische Verarbeitung von Ecore-Modellen und -Instanzen. Auf dieser Basis konnten die entsprechenden Element-Query-Erweiterungen generalisiert werden, wodurch sich der Programmierbedarf bei neuen Elementarmodell-Typen wesentlich reduzieren lässt. Im Regelfall müssen somit für einen spezifischen Elementarmodell-Typ nur noch die Klassen der Multimodell-Elemente sowie der jeweilige Pfad zum ID-Attribut angegeben werden[12].

Das Generische Multimodell wurde ebenfalls mittels EMF realisiert.

---

[5] `http://www.buildingsmart-tech.org/specifications/pset-releases`
[6] Eine Übersicht der in M2A2 umgesetzten Baufachmodelle ist in Anhang C.1 auf Seite 197 zu finden.
[7] `http://www.csvreader.com/java_csv.php`
[8] `http://www.jsdai.net`
[9] `http://www.openifctools.org`
[10] `http://www.eclipse.org/modeling/emf`
[11] Ecore ist eine Implementierung des Standards *Essential Meta Object Facility* (EMOF; ISO 19502, 2005)
[12] vgl. hierzu auch die analoge Vorgehensweise bei der deklarativen XML-Erweiterung in Abschnitt 3.4.3 auf Seite 70

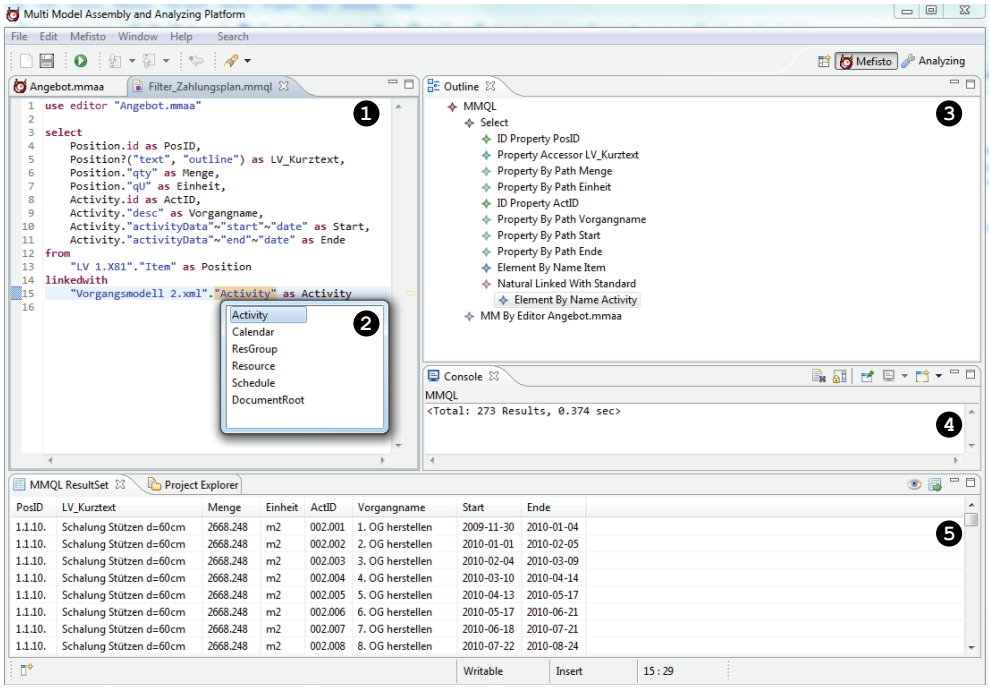


Abbildung 6.4.: MMQL-Editor in M2A2; Quelltexteditor mit Syntax-Hervorhebung, Hervorhebung von Variablen-Nutzung u. a. ①; Vorschläge zur Code-Vervollständigung ②; Abstrakter Syntaxbaum des zu bearbeitenden MMQL-Statements ③, Konsolenausgabe des Interpreters ④; tabellarisches MMQL-ResultSet ⑤

6.1.4. Implementierung der Sprache

MMQL wurde mit EMFText[13] (Heidenreich et al., 2009), einem Framework zur Definition domänenspezifischer Sprachen (DSLs), implementiert. Einen generellen Überblick über DSLs geben bspw. Kleppe (2008) und Fowler (2010).

EMFText erlaubt die Definition einer textuellen Syntax für ein Ecore-Metamodell – den abstrakten Syntaxbaum. Aus dieser Definition können anschließend Parser[14], Interpreter-Gerüst und ein sprachspezifischer Editor generiert werden. Dieser bietet u. a. Syntax-Hervorhebung, eine Gliederungssicht und Code Completion – Vorschläge zur Vervollständigung des Quelltextes bei Betätigung einer Tastenkombination.

Abbildung 6.4 zeigt den MMQL-Editor ① mit aktivem Code Completion ②, die Gliederungssicht für den aktuellen MMQL-Quelltext ③ sowie die Konsolenausgabe ④ und das MMQL-ResultSet nach Ausführung des MMQL-Befehls ⑤. Diese Ergebnisse werden vom MMQL-Interpreter produziert, welcher auf Basis des von EMFText erzeugten Gerüsts und nach der Methodik von Kapitel 5 programmiert wurde.

[13] `http://www.emftext.org`
[14] das zugrundeliegende Parser-Framework ist ANTLR (`http://www.antlr.org`; Parr, 2007)

6.2. Multimodell-Spezialisierung für das Bauprojektmanagement

Im Mefisto-Projekt[15] wurde ein Managementführungssystem für die partnerschaftliche, prozessgesteuerte und risikokontrollierte Abwicklung von Bauprojekten entwickelt. Dabei wurde auch der domänenübergreifende Datenaustausch zwischen Auftraggeber, Auftragnehmer und weiteren Subunternehmern analysiert. Im Ergebnis entstand eine Multimodell-Spezialisierung[16] für die Domäne *Bauprojektmanagement*. Dazu wurde der relevante Informationsraum abgegrenzt, indem die Domänen der zulässigen Fachmodelle festgelegt wurden[17] (Scherer & Schapke, 2011; Schapke & Fuchs, 2011):

- Bauwerksmodelle
- Baustellenmodelle
- Organisationsmodelle
- Leistungsmodelle
- Kostenmodelle
- Terminmodelle
- Risikomodelle

Des Weiteren wurden Zustände für die folgenden Fachmodelleigenschaften spezifiziert[18]:

- Phase (Level of Development)
- Detailliertheit (Level of Detail)
- Status

Diese und andere Metadaten wie Informationen über Ersteller, Multimodell-Software und Änderungszeitpunkt wurden in einem eigenen Multimodell-Datenschema abgebildet. Im Sinne der Erweiterungssystematik des Generischen Multimodells handelt es sich dabei um eine Adaption. Anhang C.3 auf Seite 199 zeigt die relevanten Bestandteile dieses s. g. Mefisto-Container-Schemas. Dessen strukturelle Eigenschaften sind:

- Die o. g. semantischen Eigenschaften eines Elementarmodells werden explizit modelliert (`metaType`).
- Ein Elementarmodell (`modelType`) darf aus mehreren Partialmodellen (`contentType`) bestehen.
- Ein Elementar-Partialmodell darf parallel in mehreren Datenformaten (`contentType/format`) vorliegen.
- Elementar-Partialmodelle werden im Container als eingebettete Datei übertragen (`contentType/file`).
- Modellierung des Linkmodells in einem eigenen Teilschema (hier zusammengefasst dargestellt).

[15] `http://www.mefisto-bau.de`; Laufzeit: 01.04.2009–30.09.2012; Förderkennzeichen des Bundesministeriums für Bildung und Forschung (BMBF): 01LA09001; Scherer et al., 2011
[16] vgl. Abschnitt 3.5 auf Seite 72
[17] das korrespondierende Vokabular ist in Anhang B.1 auf Seite 191 wiedergegeben
[18] vgl. Vokabulare in den Anhängen B.2, B.3 und B.4 ab Seite 192

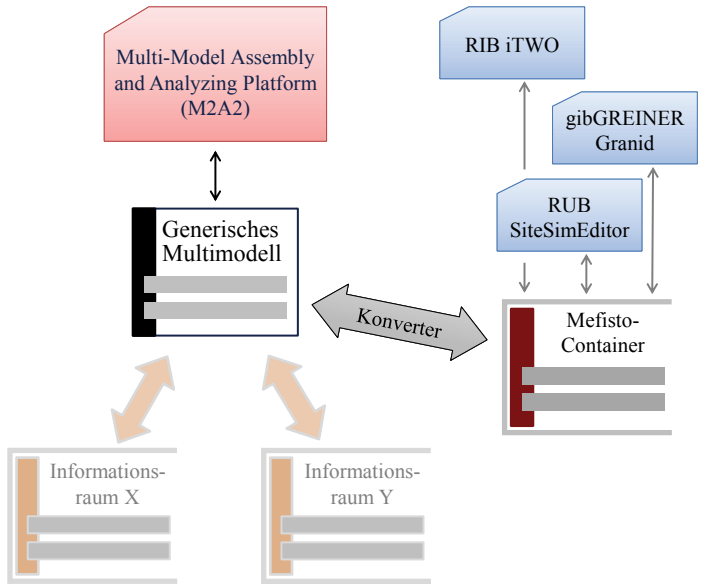


Abbildung 6.5.: Übergang vom Generischen Multimodell zu abgeschlossenen Informationsräumen mittels Konvertern (in Anlehnung an Schapke & Fuchs, 2011)

- Verlinkte Elemente beziehen sich auf ein Elementar-Partialmodell in einem bestimmten Format (`linkType/m` referenziert `modelType/id` und `linkType/c` referenziert `contentType/id`).
- Ein Link kann einen Faktor (`relationType/factor`) beinhalten, der den Anteil eines Bauelements an der Menge einer Leistungsposition repräsentiert (Mengensplit).
- Werden gefilterte Teil-Fachmodelle (Subsets) verwendet, kann der angewandte Filter angegeben werden. Dies geschieht über eine ID (`subsetType/subsetCode`) aus einem Filterkatalog oder über eine anwendungsspezifische Filtersyntax (`subsetFilterType/application`), z. B. ein MMQL-Statement.
- Werden lediglich die Metadaten zu Elementar-Partialmodellen, Subsets und Linkmodellen angegeben, so handelt es sich bei dem Mefisto-Container um ein Multimodell-Template. Dieses beschreibt den erwarteten Inhalt eines noch zu erstellenden Multimodells.

Darüber hinaus wird festgelegt, dass der Mefisto-Container als komprimierte Datei- und Verzeichnisstruktur übertragen wird. Eine Instanz des Mefisto-Container-Schemas, die Datei `container.xml` im Wurzelverzeichnis, bildet dabei den Einstiegspunkt zur Navigation. Dort sind die weiteren Speicherorte der Elementar- und Linkmodelle hinterlegt (Muntzinger & Fuchs, 2010).

Die M2A2-Plattform stellt die Verbindung zu abgeschlossenen Informationsräumen über Konverter her (vgl. Abbildung 6.5). Spezifisch modellierte Konstrukte wie Subset-Informationen werden dabei in Metadaten des Generischen Multimodells umgewandelt und umgekehrt. Dadurch müssen die universellen Multimodell-Operationen der M2A2-Plattform wie EM-Parser, EM-Filter oder MMQL nicht für jeden Informationsraum neu adaptiert werden. Im Kontext des Mefisto-Containers verhält sich die M2A2-Plattform somit nahezu wie eine spezifische

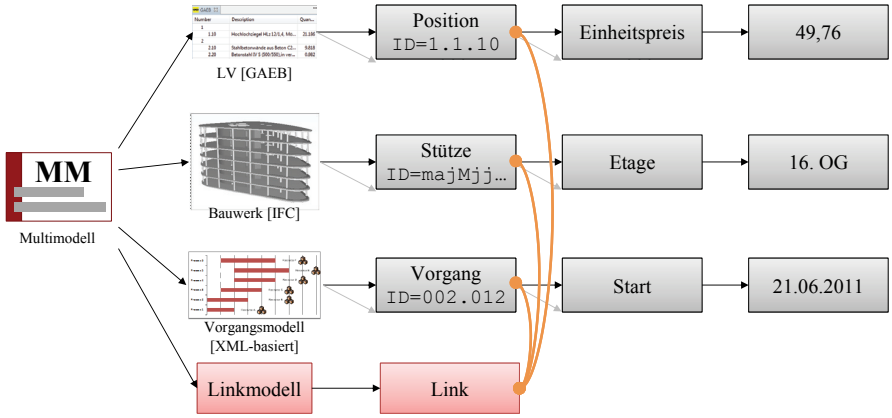


Abbildung 6.6.: Schematische Darstellung des Multimodells zur Ermittlung des Zahlungsplans.

Multimodell-Spezialanwendung, bspw. die Anwendungen für das Bauprojektmanagement durch Auftraggeber (gibGREINER Granid[19]) und Auftragnehmer (RIB iTWO®[20]) oder ein Werkzeug zur Vorbereitung von Baulogistiksimulationen (SiteSimEditor der Ruhr-Universität Bochum; König & Marx, 2011).

6.3. Multimodellbasierte Ermittlung von Zahlungsplänen

6.3.1. Beispielszenario

Im Folgenden werden das Multimodellkonzept, die Abfragesprache MMQL sowie die M2A2-Implementierung exemplarisch an einer praktischen, domänenübergreifenden Aufgabenstellung getestet. Dazu soll aus Sicht des Auftraggebers ein Zahlungsplan automatisiert ermittelt werden. Gegenstand der Untersuchung sind die Rohbaumaßnahmen für den turmartigen Teil (5.–22. OG) eines Hochhauses zum Zeitpunkt der Genehmigungsplanung.

Ein Zahlungsplan ist eine Funktion von Kosten über die Zeit. Für die Aufstellung des Zahlungsplans werden daher diskrete Kosten-Zeit-Abhängigkeiten benötigt. Diese müssen aus den vorliegenden Fachmodellen ermittelt werden. Zu diesem Zweck wurden die folgenden Fachmodelle als Elementarmodelle im Multimodell `hochaus.mmaa` gebündelt (vgl. auch Abbildung 6.6):

1. Leistungsverzeichnis in GAEB-DA-XML 3.1

   Das Leistungsverzeichnis enthält die vertraglich zu vereinbarenden Leistungen. Diese werden in Positionen beschrieben, welche eine gleichartige Leistung qualitativ (Kurztext, Langtext), quantitativ (Menge, Einheit) und monetär (Einheitspreis, Gesamtbetrag inkl. Nachlass) spezifizieren. Diese Positionen gelten jedoch für das gesamte Bauvorhaben und erlauben weder Rückschlüsse auf die Zusammensetzung der kumulierten Teilleistungen, noch auf Ort und Zeit ihrer Erbringung.

[19] `http://gibgreiner.net/de/GRANID`
[20] `www.rib-software.com/itwo`

Tabelle 6.1.: Ergebnisse der MMQL-Voranalysen der Elementarmodelle

(a) Leistungsverzeichnis (17 Zeilen)

id	Kurztext	Menge	Einheit	Einheits-preis	Gesamt-betrag
1.1.10.	Schalung Stützen d=60cm	2668,248	m2	49,76	132772,02
1.1.20.	Bewehrung Stützen d=60cm	109,35	t	767,78	83956,74
1.1.30.	Beton Stützen d=60cm	397,458	m3	181,01	71943,87
1.1.40.	Schalung Stützen d=70cm	131,274	m2	49,76	6532,19
1.1.50.	Bewehrung Stützen d=70cm	6,246	t	767,78	4795,55
1.1.60.	Beton Stützen d=70cm	22,734	m3	181,01	4115,08
1.1.70.	Schalung Stützen d=75cm	138,996	m2	49,76	6916,44
1.1.80.	Bewehrung Stützen d=75cm	7,164	t	767,78	5500,38
1.1.90.	Beton Stützen 75cm	26,064	m3	181,01	4717,84
1.1.100.	Zuschlag für Blitzschutz	0,0	m	42,65	0,00
1.1.110.	Schalung Innenwände	18927,052	m2	23,77	449896,03
1.1.120.	Bewehrung Innenwände	304,937	t	687,01	209494,77
1.1.130.	Beton Innenwände	4065,221	m3	123,43	501770,23
1.1.140.	Zuschlag Kernwände SB 3	0,0	m2	42,65	0,00
1.1.150.	Schalung Decken	26269,857	m2	30,67	805696,51
1.1.160.	Bewehrung Decken	1149,313	t	691,27	794485,60
1.1.170.	Beton Decken	9194,459	m3	114,90	1056443,34

(b) Vorgangsmodell (26 Zeilen, Auszug)

id	Vorgangsname	Start	Ende
001	Rahmentermine	2009-11-30	2011-09-02
002	Rohbau	2009-11-30	2011-07-27
002.001	5.OG herstellen	2009-11-30	2010-01-04
002.002	6.OG herstellen	2010-01-01	2010-02-05
⋮	⋮	⋮	⋮
002.018	22.OG herstellen	2011-06-23	2011-07-27
003	Baustelleneinrichtung	2009-11-16	2011-09-15

(c) Bauwerksmodell (1136 Zeilen, Auszug)

id	name	Revit-Level-Name	Volumen
majMjjrcfk6izGL_wAkvPA	- Stütze rund ø60cm C30/37	16.OG	0,9330
SiY8nV23GE6xZu0e$diwSQ	STB-Wand 35cm C30/37	22.OG	12,0092
JogixXns_UC4A5ll$5K0xw	Decke Ortbeton 35cm C30/37	16.OG	204,6308
xg6MQsKq9k_qt83557T9wQ	- Stütze rund ø70cm C90/105	18.OG	1,2699
gF88A6dDy0WDEJ08QxZ4Sw	- Stütze rund ø75cm C90/105	8.OG	1,4578
7lLAKvHuwEGMckFOIoRPnA	STB-Wand 50cm C30/37	12.OG	1,1336
⋮	⋮	⋮	⋮

Mit der folgenden MMQL-Abfrage wurde eine Voranalyse des Leistungsverzeichnisses `LV` durchgeführt:

Quelltext 6.1: MMQL zur Voranalyse des Leistungsverzeichnisses

```
use editor "hochhaus.mmaa"

select
    Position.id,
    Position ? ("text", "outline") as Kurztext,
    Position."qty" as Menge,
    Position."qU" as Einheit,
    Position."uP" as Einheitspreis,
    Position."iT" as Gesamtbetrag
from
    "LV"."Item" as Position
```

Das Ergebnis der Abfrage ist in Tabelle 6.1a auf der vorherigen Seite abgebildet. Relevante Bauteile sind Stützen, Decken und Wände. Jede der 17 LV-Positionen verfügt über Belegungen für die o. g. Properties. Die Positionen 1.1.100. und 1.1.140. sind offensichtlich Bedarfspositionen.

2. Vorgangsmodell in MF Schedule (XML-basiert)[21]

   Das Vorgangsmodell enthält einen Rahmenterminplan mit Sammel- und Einzelvorgängen. Die Einzelvorgänge beschreiben die Start- und Endtermine der geplanten Erstellung der einzelnen Geschosse. Dazu enthalten sie die Nummer des jeweiligen OG im Namen. Aus dem Vorgangsmodell sind jedoch keine Aussagen über die zur Fertigstellung notwendigen Bauteile und Leistungen ableitbar. Die folgende Abfrage erzeugt das in Tabelle 6.1b dargestellte Ergebnis:

Quelltext 6.2: MMQL zur Voranalyse des Vorgangsmodells

```
use editor "hochhaus.mmaa"

select
    Vorgang.id,
    Vorgang."desc" as Vorgangname,
    Vorgang."activityData"~"start"~"date" as Start,
    Vorgang."activityData"~"end"~"date" as Ende
from
    "Vorgangsmodell"."Activity" as Vorgang
```

3. Bauwerksmodell in IFC2x3 (Datenformat SPF)

   Das Bauwerksmodell enthält u. a. die zu erstellenden Bauteile einschließlich Lage und Geometrie. Jedes Bauelement beschreibt sein geometrisches Volumen sowie das zugeordnete OG in IFC-Properties. Stützen besitzen einen kreisförmigen Querschnitt. Ihr Durchmesser ist im Bauteilnamen enthalten. Das Bauwerksmodell enthält keine Informationen über notwendige Leistungen oder Bauzeiten der Bauteile. Das Abfrageergebnis in Tabelle 6.1c wird von folgendem Quelltext erzeugt:

[21] vgl. Anhang C.1 auf Seite 197

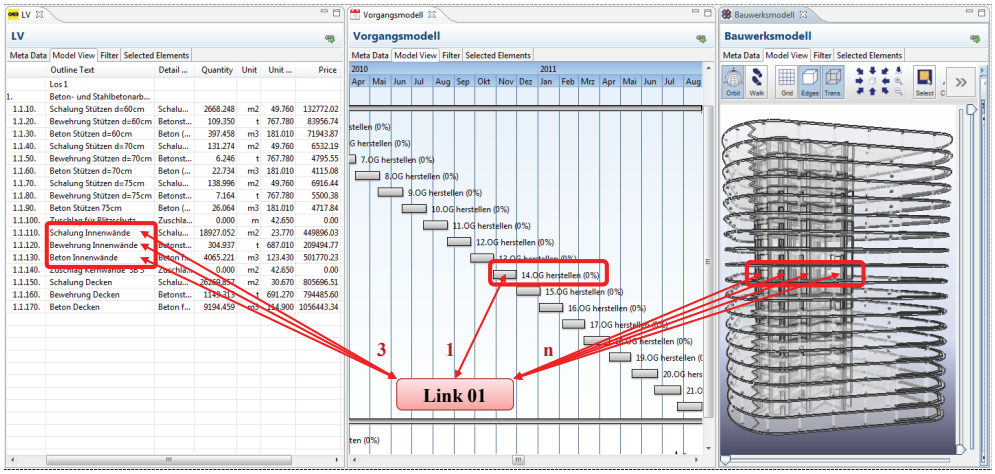


Abbildung 6.7.: Visualisierung der Elementarmodelle des Beispielszenarios in der M2A2-Plattform sowie die gewünschte Verlinkung der Elemente am Beispiel eines Links.

Quelltext 6.3: MMQL zur Voranalyse des Bauwerksmodells

```
use editor "hochhaus.mmaa"

select
    BE.id,
    BE."name",
    BE ? ("property", "RevitLevelName") as RevitLevelName,
    BE ? ("property", "Volumen") as Volumen
from
    "Bauwerksmodell"."IfcBuildingElement" as BE
```

6.3.2. Erstellen der Links

Bisher liegen die Informationen zu Kosten und Terminen getrennt in den Elementarmodellen LV und Vorgangsmodell vor. Eine Verlinkung von LV-Positionen mit Vorgängen würde Aussagen darüber ermöglichen, welche Leistungen in welchen Obergeschossen erbracht werden. Allerdings ist diese Zuordnung in einigen Fällen nur mit Kenntnis der Lage von Bauteilen möglich. So ist durch visuelle Analyse (vgl. Abbildung 6.7) zwar zu erkennen, dass in jedem Geschoss Schal-, Bewehrungs- und Betonierarbeiten für Decken und Wände notwendig sind – aber wie sich die Stützen mit unterschiedlichem Durchmesser über die Geschosse verteilen, kann mit dieser manuellen Methode nicht mehr effizient herausgefunden werden. Um später fundierte, diskrete Kosten-Zeit-Abhängigkeiten erstellen zu können, müssen daher auch die an einem Vorgang und einer Leistung beteiligten Bauelemente verlinkt werden.

Somit liefert ein Vorgang die Daten zu Start und Ende der Errichtung eines Obergeschosses und die Bauelemente dieses Obergeschosses bestimmen darüber, welche Leistungen erbracht werden[22]. Für eine Gruppe von Stützen, Wänden oder Decken gelten dabei die jeweils entsprechenden Schal-, Bewehrungs- und Betonierleistungen. Ein Link besteht somit aus 1 Vorgang,

[22] Der Anteil der je Bauteil erbrachten Menge an der Gesamtmenge der LV-Position wird in der nachgelagerten

n Bauelementen gleichen Typs und 3 LV-Positionen. Abbildung 6.7 zeigt die Struktur der aufzustellenden Links an einem Beispiel. Die Bedeutung der Links ist: *Bauteile und ihre Leistungsanteile an einem Vorgang*. Quelltext 6.4 auf der nachfolgenden Seite zeigt die notwendigen MMQL-Statements um die Multimodell-Elemente wie beschrieben zu verlinken.

Die Links für Wände, Decken und Stützen werden sukzessive erstellt. Auf diese Weise können die Bauelemente typsicher angegeben werden, sodass das Matching mit LV-Positionen nur noch statisch über den korrespondierenden Bauteilnamen als Teil des Kurztextes zu erfolgen braucht (Zeilen 10 & 12, 25 & 27, 36 & 38). Die Bedarfsposition *Zuschlag Kernwände SB3* soll jedoch in keinem Fall einer Gruppe von Wänden zugeordnet werden (Zeile 14). Zusätzlich müssen bei Stützen die Durchmesser des Querschnitts unterschieden werden. Hier erfolgt das Matching über die zwei Ziffern, welche sich im Bauteilnamen und im Kurztext jeweils direkt vor der Zeichenkette „`cm`" befinden (Zeilen 40–43).

Die Zuordnung von Bauteilen zu Vorgängen erfolgt über ein Matching der OG-Nummern im IFC-Property `RevitLevelName` und innerhalb des Vorgangsnamen. Für eine exakte Übereinstimmung müssen dazu beim Vorgangsnamen die Zeichenkette „`herstellen`" sowie verbleibende Leerräume entfernt werden (Zeilen 16–19, 29 f., 45 f.).

Es existieren 18 Vorgänge zum Erstellen von Geschossen und fünf differenzierte Gruppen von Leistungen (für Wände, für Decken und für Stützen mit drei verschiedenen Durchmessern). Im Ergebnis entstehen somit $18 \cdot 5 = 90$ Links, welche automatisch im Linkmodell *LM1* hinterlegt werden. Das Resultat kann anschließend mit der M2A2-Plattform visuell verifiziert werden.

In einem optionalen Schritt kann das Multimodell nun per Container versendet werden.

6.3.3. Multimodell-Filter und Berechnung des Zahlungsplans

Der Zahlungsplan soll eine Aufstellung der bereitzuhaltenden Geldmittel für jeden Kalendermonat beinhalten. Dazu müssen für jeden Monat die diskreten anfallenden Kosten berechnet werden. Die Datengrundlage für die Berechnung dieser Kosten-Zeit-Abhängigkeiten ist der domänenübergreifende Informationsraum des Multimodells. Auf Basis der erstellten Links kann nun abgefragt werden, an welchen Vorgängen jede LV-Position beteiligt ist[23]:

Die Ausführung von Quelltext 6.5 hat das in Tabelle 6.2 auf Seite 173 dargestellte ResultSet zum Ergebnis. Jede Zeile repräsentiert genau *eine*, aus dem Linkmodell ableitbare, Kombination von LV-Positionen und Vorgängen. Aus den 15 relevanten LV-Positionen und den 18 Vorgängen ergeben sich somit $15 \cdot 18 = 270$ Zeilen. Mit den Mitteln der MMQL können die gewonnenen Daten an dieser Stelle jedoch nicht weiter verarbeitet werden. Die Berechnung des Zahlungsplans erfolgt daher nachgelagert in einer Tabellenkalkulationssoftware. Dazu wird das ResultSet dort per CSV-Schnittstelle importiert.

Die Ermittlung anteiliger Kosten für einen Kalendermonat geschieht unter der Annahme eines linearen Zusammenhangs von Bauzeit und anrechenbaren Kosten. Das bedeutet, dass nach Verstreichen von 30 % der aufsummierten Gesamtdauer aller Vorgänge, die einer LV-Position zugeordnet sind, auch 30 % des Gesamtbetrages dieser Position liquidiert werden können. Wird für jede Zeile der monatliche Zeitanteil eines Vorgangs an der Gesamtdauer zur Erbringung der

Berechnung abgeschätzt. Für eine genaue Ermittlung wären zugeordnete Mengensplits erforderlich. Diese sind in den Ausgangsdaten jedoch nicht vorhanden.

[23] Das Bauwerksmodell wird in der Multimodell-Abfrage nicht benötigt. Durch seine Einbeziehung beim Verlinken ist die bauteilabhängige Beziehung zwischen LV-Positionen und Vorgängen bereits in den Links repräsentiert.

Quelltext 6.4: MMQL zur Erstellung der Verlinkung

```
use editor "hochhaus.mmaa"

create linkmodel "LM1"
    add permission "Bauwerksmodell", "Vorgangsmodell", "LV"

update linkmodel "LM1"
    add links between
        one "Vorgangsmodell"."Activity" as Vorgang,
        many "LV"."Item" as LVPosition,
        many "Bauwerksmodell"."IfcWall" as Wand
    where
        LVPosition ? ("text", "outline") as Kurztext like "wände"
        and
        not Kurztext like "Zuschlag"
        and
        // Nr. des OG im Vorgangsname
        trim(replace(Vorgang."desc", "herstellen", ""))
            // Nr. des OG im Bauelement-Property
            == trim(Wand ? ("property", "RevitLevelName"))

update linkmodel "LM1"
    add links between
        one "Vorgangsmodell"."Activity" as Vorgang,
        many "LV"."Item" as LVPosition,
        many "Bauwerksmodell"."IfcSlab" as Decke
    where
        LVPosition ? ("text", "outline") as Kurztext like "Decken"
        and
        trim(replace(Vorgang."desc", "herstellen", ""))
            == trim(Decke ? ("property", "RevitLevelName"))

update linkmodel "LM1"
    add links between
        one "Vorgangsmodell"."Activity" as Vorgang,
        many "LV"."Item" as LVPosition,
        many "Bauwerksmodell"."IfcColumn" as Stuetze
    where
        LVPosition ? ("text", "outline") as Kurztext like "Stützen"
        and
        // Stützendicke aus IfcColumn.name
        substring(Stuetze."name", position(Stuetze."name", "cm") - 2,
            position(Stuetze."name", "cm"))
            // Stützendicke aus LV-Kurztext
            == substring(Kurztext, position(Kurztext, "cm") - 2, position(
                Kurztext, "cm"))
        and
        trim(replace(Vorgang."desc", "herstellen", ""))
            == trim(Stuetze ? ("property", "RevitLevelName"))
```

Quelltext 6.5: MMQL zur Filterung des Multimodells

```
use editor "hochhaus.mmaa"

select
    LVPosition.id as LVPosID,
    LVPosition."iT" as Gesamtbetrag,
    Vorgang.id as VorgID,
    Vorgang."activityData"~"start"~"date" as Start,
    Vorgang."activityData"~"end"~"date" as Ende
from
    "LV"."Item" as LVPosition
linkedwith
    "Vorgangsmodell"."Activity" as Vorgang
```

gesamten LV-Position berechnet, kann auf diese Weise auch der korrespondierende monetäre Betrag angegeben werden. Die für einen Kalendermonat insgesamt anfallende Summe ergibt sich durch Aufsummierung der monatlichen Beträge aller Zeilen. Abbildung 6.8 auf der nachfolgenden Seite zeigt den so resultierenden Zahlungsplan.

Der Zahlungsplan kann nun für weitere Analysen verwendet werden, z. B. zur Risikosimulation (vgl. Flemming & Fuchs, 2012). Ändern sich Daten in den zugrundeliegenden Fachmodellen, kann der Zahlungsplan mit der beschriebenen Methode automatisch neu aufgestellt werden. Auch sind andere Ansätze zur Ermittlung der anteiligen Kosten je Zeitintervall anwendbar, sofern diese bekannt sind. Beispielsweise erlauben die vorhandenen Daten auch eine Berechnung über die im untersuchten Zeitintervall anteilig produzierten Volumina der Bauteile. Dazu müssen diese nach Bauteiltyp gruppiert werden, da sich die kubikmeterbezogenen Herstellungsdauern für Decken, Wände und Stützen verfahrensbedingt unterscheiden.

Tabelle 6.2.: ResultSet des Multimodell-Filters für den Zahlungsplan (270 Zeilen, Auszug)

Zeile	LVPosID	Gesamtbetrag	VorgID	Start	Ende
⋮	⋮	⋮	⋮	⋮	⋮
160	1.1.90.	4717,84	002.016	2011-04-18	2011-05-23
161	1.1.90.	4717,84	002.017	2011-05-20	2011-06-27
162	1.1.90.	4717,84	002.018	2011-06-23	2011-07-27
163	1.1.110.	449896,03	002.001	2009-11-30	2010-01-04
164	1.1.110.	449896,03	002.002	2010-01-01	2010-02-05
165	1.1.110.	449896,03	002.003	2010-02-04	2010-03-09
⋮	⋮	⋮	⋮	⋮	⋮

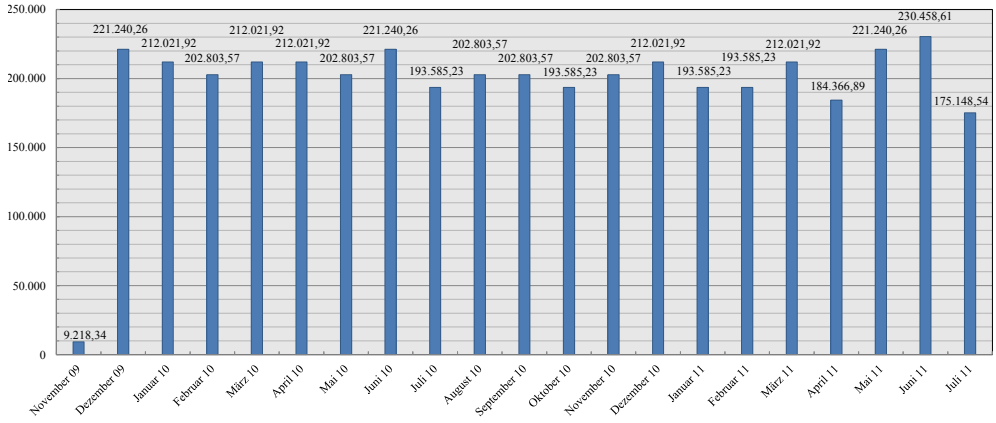


Abbildung 6.8.: Balkendiagramm des ermittelten Zahlungsplans [€]

6.4. Bewertung und Resümee

6.4.1. Bewertung der Ausführungsdauern

Eine stichprobenhafte, manuelle Aufstellung der für den Zahlungsplan benötigten Daten[24], nahm 3,5 h in Anspruch. Zwar begünstigte die etagengleiche, turmartige Struktur der Bauelemente die Bearbeitungsdauer – dafür musste ein Großteil der Zeit für Überprüfung und Fehlerkorrektur aufgewendet werden.

Tabelle 6.3 zeigt die reinen Ausführungszeiten der MMQL-Anweisungen zur Verlinkung und zum Multimodell-Filtern, wie sie zur automatisierten Datenerzeugung verwendet wurden. Der Aufwand zur Erstellung der Quelltexte betrug darüber hinaus 1,0 h – er soll jedoch vernachlässigt werden, da er für diese Klasse von Aufgabenstellungen nur ein einziges Mal erbracht werden muss. D. h. die MMQL-Anweisungen können wiederverwendet werden, solange sich die Fachmodelle nicht grundlegend strukturell ändern.

Der quantitative Geschwindigkeitsgewinn der multimodellbasierten gegenüber der manuellen Datengewinnung beträgt demnach zwischen Faktor 214,6 und Faktor 482,8. Dies bedeutet einen deutlichen Zeitvorteil, besonders bei wiederholter Ausführung auf geänderten Fachmodellen. Unabhängig davon bietet die automatisierte Datengewinnung den qualitativen Vorteil der

Tabelle 6.3.: Durchschnittliche Ausführungszeiten der MMQL-Quelltexte auf einem Intel® Core™ i5-3337U-Prozessor mit 1,8 GHz und einer 64-bit JVM (Version 1.7.0_40) unter Windows 8.

∅ Ausführungszeit [s] / MMQL-Quelltext	bei geparsten Modellen	inkl. Parsen der Modelle
6.4 – Verlinkung	25,8	52,7
6.5 – MM-Filter	0,3	6,0

[24] vgl. Tabelle 6.2 auf der vorherigen Seite

prinzipbedingten Fehlerfreiheit: MMQL-Anweisungen können auf denselben Multimodell-Daten beliebig oft ausgeführt werden und ermitteln immer identische Resultate. Aufgrund ihrer Fehleranfälligkeit ist dies durch die manuelle Datengewinnung nicht gewährleistet.

6.4.2. Resümee

Für die Anwendung der Multimodell-Methode in realen Bauinformationsprozessen werden Implementierungen von Multimodell-Containern, Fachanwendungen und der Multimodell-Engine benötigt. Mit der M2A2-Plattform wurde eine universelle Multimodell-Software präsentiert, die aus wiederverwertbaren Komponenten für diese und weitere Aspekte aufgebaut ist. Als generische, erweiterbare Plattform ist sie für allgemeine Multimodell-Aufgaben wie Linkanalyse und MMQL-Programmierung konzipiert. Anwenderoperationen für spezifische, baufachliche Problemstellungen können jedoch über Plugins hinzugefügt werden.

Am Beispiel der Ermittlung eines Zahlungsplanes konnte gezeigt werden, dass bereits mit der vorhanden Funktionalität von M2A2 domänenübergreifende Aufgabenstellungen des Bauwesens effizient lösbar sind. Dabei kommt die Software zur Erzeugung von Multimodellen und zur Bereitstellung von Multimodell-Views zum Einsatz. Für weitere, baufachliche Anwenderoperationen können externe, spezialisierte Fachanwendungen benutzt werden.

Die Multimodell-Methode und die bereitgestellten Software-Komponenten sollen die Entwicklung spezifischer Baufachanwendungen für konkrete interdisziplinäre Aufgabenstellungen vereinfachen. Für abgeschlossene Anwendungsbereiche, wie Ausschreibung und Vergabe, kann es dabei sinnvoll sein, den Multimodell-Container für dieses Fachgebiet zu spezialisieren. Mit dem Mefisto-Container wurde dies für die Domäne *Bauprojektmanagement* demonstriert.

Kapitel 7.

Fazit

> Die Wissenschaft fängt eigentlich erst da an, interessant zu werden, wo sie aufhört.
>
> *(Justus von Liebig)*

7.1. Zusammenfassung

Der datenmodell-, datenformat- und domänenübergreifenden Informationsunterstützung kommt im Bauwesen ein hoher Stellenwert zu. Multimodelle gestatten die Aufstellung, den Austausch und die Erschließung domänenübergreifender Informationsräume auf neuartige Weise. Sie verwenden unveränderte, heterogene Fachmodelle als Datenbasis und unterstützen damit die prozessorientierte Arbeitsweise. Sie erlauben sowohl eine Weiterverwendung existierender Baufachsoftware als auch die Neu- und Weiterentwicklung interdisziplinärer Applikationen. Durch ihre fachliche Neutralität und die Anwendbarkeit nahezu beliebiger Datenformate eignen sie sich für eine Vielzahl aktueller und zukünftiger fachübergreifender Aufgabenstellungen im Bauwesen.

Das Generische Multimodell definiert die Datenstruktur zur Aufnahme der originalen Fachmodelle (den s. g. Elementarmodellen) und der externen Linkmodelle mit den expliziten, ID-basierten, mehrwertigen Links zur Abbildung modellübergreifender Relationen zwischen den Datenelementen. Metadaten erleichtern den Datenaustausch per Container durch die Speicherung beschreibender Informationen. Über Einschränkungen durch die dabei verwendeten Vokabulare ist es möglich, Multimodelle für abgegrenzte Informationsräume zu spezialisieren.

Der Einsatz heterogener Datenformate erfordert einen *generischen* Zugriff auf die Fachdaten der Elementarmodelle. Mit dem Ideellen Elementarmodell wurde dazu eine virtuelle Struktur geschaffen, welche eine verallgemeinerte, idealisierte Abbildung gängiger Datenformate repräsentiert. Innerhalb von Multimodell-Softwaresystemen stellt sich das Ideelle Elementarmodell als Schnittstelle zu den konkreten Parsern der physischen Datenformate dar. Durch diese Verallgemeinerung lassen sich diese Applikationen um neue Elementarmodell-Typen erweitern. Mit der *Multi-Model Assembly and Analyzing Platform* (M2A2) erfolgte eine prototypische Implementierung eines solchen *universellen* Multimodell-Softwaresystems.

Aus den Strukturen von Generischem Multimodell und Ideellem Elementarmodell lassen sich elementare Operationen auf Multimodellen ableiten. Um multimodellbasierte Informationsräume nutzbringend erschließen zu können, müssen diese Elementaroperationen zu komplexen Verbundoperationen zusammengesetzt werden. Dabei sind die besonderen Eigenschaften zu berücksichtigen, welche für multimodellbasierte Informationsräume identifiziert wurden: die

Modi zu Elementkombination und Linkinterpretation beim Multimodell-Filtern sowie die Modi zu Duplizität, Arität und Kardinalität beim Erstellen von Links.

Zur Vereinfachung dieser komplizierten Tätigkeit wurde die deklarative Programmiersprache *Multi-Model Query Language* (MMQL) entwickelt. Diese bietet die Möglichkeit, Aufgabenstellungen des Multimodell-Filterns und der Linkmanipulation prägnant zu formulieren. Syntax und Semantik wurden dabei weitestgehend an die bekannte Datenbanksprache SQL angelehnt. Die Ausführung der MMQL-Anweisungen erfolgt durch einen entsprechenden Interpreter. Dieser ermittelt die Lösung für die beschriebene Aufgabenstellung. Sprache und Interpreter wurden ebenfalls prototypisch implementiert und in M2A2 integriert. Auf diese Weise konnten die entwickelten Methoden zur Erschließung multimodellbasierter, domänenübergreifender Informationsräume am praktischen Beispiel eines Zahlungsplans überprüft werden.

7.2. Ergebnisdiskussion

Die Ergebnisse der Arbeit wurden bereits im vorangegangenen Abschnitt zusammengefasst. An dieser Stelle sollen die geleisteten wissenschaftlichen Beiträge herausgestellt und bewertet werden.

Lose gekoppelte, heterogene Fachmodelle

In der Forschungsgemeinschaft herrscht Übereinstimmung darüber, dass die Abbildung aller Daten eines Bauvorhabens nicht in einem einzelnen Fachmodell geschieht, sondern über mehrere Fachmodelle verteilt erfolgt. Die damit verbundenen Problematiken werden unter den Themengebieten der Partialmodelle, des nD-Modelling und Linked Data untersucht. Bisherige Lösungsansätze gehen davon aus, dass für eine modellübergreifende Erschließung alle Daten in einem einheitlichen Format (bspw. eine Datenbank, Ontologie / RDF oder IFC) vorliegen müssen, damit formatspezifische Verknüpfungs- und Filtermethoden (z. B. SQL oder SPARQL) angewendet werden können. Fachmodelle mit abweichendem Originalformat müssen dazu konvertiert werden. Eine Methode wie das Multimodell, welche gekoppelte, originale, explizit heterogene Fachmodelle als direkte Datenbasis für Bauinformationsprozesse benutzt, war bisher unbekannt.

Der Austausch verlinkter Fachmodelle war mit den bisherigen Ansätzen entweder überhaupt nicht, nur im homogenen (möglicherweise nicht originalen) Datenformat oder nur unter Nutzung eines führenden Fachmodells – in welchem auch die Links hinterlegt waren – möglich. Im Generischen Multimodell sind die Fachmodelle gleichgestellt und heterogen. Damit ermöglicht das Konzept erstmals austauschbare Partialmodelle, deren Alleinstellungsmerkmal ihre Neutralität im Hinblick auf Fachlichkeit *und* Datenformat ist.

Generischer Zugriff auf Originaldaten

Die in einem Multimodell teilnehmenden Fachmodelle werden Elementarmodelle genannt. Trotz ihrer ursprünglichen Trennung und ihrer unterschiedlichen Formate werden sie als gemeinsamer Informationsraum bereitgestellt. Dazu müssen ihre Daten auch in einem gemeinsamen System verarbeitbar sein. Das Konzept des Ideellen Elementarmodells erlaubt den vereinheitlichten Zugriff auf Daten beliebiger Formate. Dazu wird eine virtuelle Struktur definiert, welche

ein Grundkonzept der Datenmodellierung aufgreift: die Aggregation simpler Dateneinheiten zu komplexen Elementen. Das Ideelle Elementarmodell stellt sich als *Schnittstelle* zu den individuellen Parsern der Datenformate dar. Es ist für einen generischen, reflexiven Zugriff auf Datenstrukturen konzipiert, wobei der Übergang von Multimodell-Elementen (als komplexen) zu Properties (als simplen Dateneinheiten) mit beliebiger Logik erfolgen kann.

Durch den Verzicht auf ein integrierendes Datenschema werden keine implementierten Mappings benötigt, bei welchen zum Entwurfszeitpunkt Quell- und Zielformat sowohl technisch als auch semantisch vollständig verstanden und verlustfrei ineinander überführbar sein müssen. Des Weiteren müsste ein integrierendes Datenschema ständig für neue Anwendungszwecke erweitert werden und unterläge somit dem Dilemma eines potentiellen Superdatenmodells. Beim Ideellen Elementarmodell wird der Zugriff auf die Originaldaten an die jeweiligen Implementierungen delegiert. Anstelle eines semantischen, vollständigen Mappings müssen diese lediglich einen technischen Zugriff per Reflexion nach Maßgabe der Schnittstelle bereitstellen.

Gegenüber Mappings ermöglicht das Ideelle Elementarmodell die unbegrenzte Erweiterbarkeit um neue Elementarmodell-Typen, sowie einen geringeren Aufwand bei der Integration eines solchen Typs. Das Fehlen eines integrierenden, semantischen Datenmodells verlagert die Herstellung struktureller Fachsemantik jedoch auf den Zeitpunkt der Erschließung des gemeinsamen Informationsraums. Das bedeutet, dass in Abfragen die Struktur der originalen Elementarmodelle benutzt werden muss. Außerdem ist der Zugriff abhängig von der jeweiligen Parser-Implementierung, weswegen durch das Konzept ein vollständiger Zugriff auf alle Daten eines originalen Elementarmodells nicht *prinzipiell* gewährleistet werden kann.

ID-basierte Links

Die Identifikation und Verlinkung von Dateneinheiten über eindeutige Identifikatoren (IDs) ist ein bekanntes und bspw. in Datenbanken angewandtes Prinzip der Informatik. IDs erlauben die Entkopplung des Linkobjekts von den eigentlichen Dateneinheiten und bilden damit die Grundlage der Nutzung unveränderter Elementarmodelle im Multimodell. Allerdings können ID-basierte Links brechen, d. h. dass ein durch eine ID beschriebenes Element nicht existiert. Solche Inkonsistenzen können jedoch durch automatisierte Überprüfung aufgedeckt und behoben werden. Schwerwiegender ist dagegen eine generelle Abwesenheit von IDs in einem Elementarmodell. Zwar wurden insgesamt acht mögliche Ausweichlösungen für diesen Fall genannt, der individuelle Aufwand zur Ableitung künstlicher IDs kann in Ausnahmefällen jedoch sehr hoch sein. Manipulative Ausweichlösungen ergänzen fehlende IDs, widersprechen damit aber dem Grundkonzept des Multimodells, Elementarmodelle unverändert zu lassen.

Mehrwertige Links

Die Links des Multimodells dürfen mehrwertig sein; d. h. jeder Link kann eine Relationen von mehr als zwei Elementen beschreiben. Solche Relationen sind im Bauwesen nicht unüblich, beispielsweise bei der Zuordnung eines Krans zu zwei Ladevorgängen. Mehrwertige Links sind datentechnisch schwerer zu beherrschen als binäre Links. Strukturell können mehrwertige Links auch in mehrere binäre Links umgewandelt werden. Dabei ist jedoch zu überprüfen, ob die fachliche Bedeutung erhalten bleibt. Im Wesentlichen ist die Frage zu beantworten, ob die so entstandenen binären Links nur gemeinsam existieren dürfen. Im Beispiel des Krans wäre dies

der Fall, wenn die beiden Ladevorgänge einen untrennbaren Umladeprozess, bestehend aus Ent- und Beladung, beschreiben.

Die potentielle Mehrwertigkeit von Links führt zu einer gesteigerten Komplexität der Multimodell-Methode. Da die Bedeutung mehrwertiger Links fach- und problemspezifisch ist, muss die Art der Interpretation durch den Nutzer entscheidbar sein. Entsprechend wird durch die Multimodell-Methode die Mehrwertigkeit der Links – und damit auch die einhergehende Komplexität – dem Nutzer exponiert. Ob sich eine binäre oder eine anderweitige bevorzugte Nutzungsweise in der Praxis etablieren wird, bleibt an dieser Stelle offen. Aus methodischer Sicht bilden mehrwertige Links die allgemeine Lösung.

Multimodell-Filtern – Strukturelle Linksemantik

Die erarbeitete Methode zum Multimodell-Filtern setzt sich aus dem Elementarmodell-Filtern und dem neuartigen Aspekt der *Linkauswertung* zusammen. Zwar ist das Prinzip des Heranziehens vorhandener Relationen zum Filtern bereits bekannt, jedoch waren bei bisherigen Ansätzen die Relationen Teil des Fachmodells – und damit technisch, strukturell motiviert sowie überwiegend semantisch eindeutig. Dagegen werden beim Multimodell-Filtern externe und aus Sichtweise eines Elementarmodells semantisch undefinierte, mehrwertige Links verarbeitet. Die Systematik der Linkauswertung folgt daher der *strukturellen Linksemantik* – der prinzipiellen Deutungsmöglichkeit vorhandener und nicht vorhandener mehrwertiger Links durch den Nutzer. Die *Linkinterpretation* bestimmt dabei, welche Bedeutung ein gegebener Link für eine korrespondierende Menge von Multimodell-Elementen hat. Die *Elementkombination* legt fest, welche Elemente der Elementarmodelle kombiniert werden sollen und ob unverlinkte Elemente im Filterergebnis erscheinen dürfen.

Mehrwertige Linkerzeugung

Der generische Zugriff auf die Elementarmodell-Originaldaten ermöglicht die Definition *elementarmodellübergreifender Kriterien für Properties*, z. B. zum Vergleichen von Werten aus unterschiedlichen Modellen. Dies erlaubt zum einen die Definition modellübergreifender Kriterien als weiteren Aspekt des Multimodell-Filterns und bildet zum anderen die Basis der Linkerzeugung. Dort wird für jede mögliche Elementkombination entschieden, ob diese als Link im Linkmodell aufzunehmen ist. Dafür sind – neben Elementarmodell-Filtern – elementarmodellübergreifende Property-Kriterien von wesentlicher Bedeutung.

Ein weiterer Aspekt der vorgestellten Methode zur automatisierten Linkerzeugung ist die Fähigkeit zur indirekten Erstellung mehrwertiger Links. Dies erfolgt durch *sukzessive Elementgruppierung* – die schrittweise Umwandlung mehrerer einwertiger Links in *einen* mehrwertigen (bezogen auf einen Element-Typ). Legt ein Nutzer die Mehrwertigkeit für einen oder mehrere Element-Typen fest, wird damit eine fachliche Äquivalenz der einwertigen zu dem umgewandelten, mehrwertigen Link ausgedrückt.

Eine direkte automatisierte Erstellung mehrwertiger Links ist mit dieser Methode nicht möglich. Dazu müssten nicht nur alle Elemente typweise einzeln kombiniert werden, sondern zusätzlich auch innerhalb jedes Typs mehrwertig untereinander. Bei m Element-Typen würde die Anzahl der zu überprüfenden Kombinationen dabei von

$$n_1 \cdot n_2 \cdot \ldots \cdot n_m$$

auf

$$(2^{n_1} - 1) \cdot (2^{n_2} - 1) \cdot \ldots \cdot (2^{n_m} - 1)$$

steigen. Diese Menge wäre einerseits für praktische Probleme nicht in vernünftiger Zeit berechenbar. Andererseits würde auch die Aufstellung entsprechender Auswahlkriterien komplexer, da die Möglichkeit zur unterscheidbaren Adressierung mehrerer typgleicher Elemente eingeräumt werden müsste. Das manuelle Anlegen eines beliebigen Links ist dagegen immer möglich.

Textuelle, deklarative Programmiersprache MMQL

Mit der *Multi-Model Query Language* (MMQL) wurde eine Programmiersprache sowie ein zugehöriger Interpreter entwickelt. Damit wird die Aufstellung und Ausführung von Multimodell-Filtern und Anweisungen zur Linkerzeugung – im Verhältnis zu einer programmatischen Nutzung elementarer Operationen – stark vereinfacht. Im Gegensatz zu einer Bibliothek vordefinierter Anwenderoperationen bietet MMQL als *textuelle Sprache* ein fast uneingeschränktes Anwendungsspektrum. Besonders die Definition komplexer Kriterien ist weder mit Bibliotheken noch mit grafischen Sprachen derart kompakt und prägnant möglich. MMQL besitzt somit die Mächtigkeit zur nahezu vollständigen Erschließung multimodellbasierter Informationsräume. Dies wird auch durch die Möglichkeit zur Integration beliebiger parametrisierbarer, externer Elementarmodell-Filter unterstützt.

Die Auslegung von MMQL als *deklarative Sprache* erleichtert zudem das Erschließen des domänenübergreifenden Informationsraumes. Es muss nur die zu lösende Aufgabenstellung beschrieben werden. Der entsprechende Lösungsweg wird anschließend vom MMQL-Interpreter ermittelt und ausgeführt. Dies entbindet den Nutzer von der Aufstellung und Implementierung komplexer Lösungsstrategien. Zu diesem Zweck sind die o. g. methodischen Ansätze zum Multimodell-Filtern und der mehrwertigen Verlinkung als sprachliche Konzepte abgebildet und im Interpreter für reale Daten ausführbar realisiert. Die benutzten originalen Datenstrukturen müssen jedoch vom Anwender beherrscht und in die Formulierung der MMQL-Ausdrücke eingebracht werden.

Anwendbarkeit

Das Multimodell-Konzept ist für nahezu beliebige Fachmodelle anwendbar. Eine abschließende Aufzählung geeigneter Datenformate ist jedoch nicht möglich. Die Bedingung zur Teilnahme ist die individuelle Implementierbarkeit des Ideellen Elementarmodells als Schnittstelle zum konkreten Parser. Allerdings ermöglicht der reflexive Zugriffscharakter eine grundlegende Implementierung auf abstrakter Ebene. So wird bei mindestens sieben relevanten Meta-Datenstrukturen – darunter XML, EXPRESS und Relationale Datenbanken – von einer prinzipiellen Anwendbarkeit ausgegangen. Es ist zu vermuten, dass nur vereinzelte Datenformate existieren, auf welche nicht per Ideellem Elementarmodell zugegriffen werden kann. Somit ist die Multimodell-Methode auch außerhalb des Bauwesens anwendbar. Für das Bauwesen bedeutet dies eine Unterstützung für beliebige interdisziplinäre Bauinformationsprozesse.

Für die Teilnahme eines konkreten Elementarmodell-Typs ist eine korrespondierende Erweiterung der Multimodell-Software notwendig. Diese muss einmalig implementiert werden. Für einige Elementarmodell-Typen wie IFC und GAEB wurden entsprechende Erweiterungen bereits im Rahmen dieser Arbeit angefertigt. So bietet bspw. die Erweiterung für CSV die

Möglichkeit, Daten aus Tabellenkalkulationsprogrammen als vollwertiges Elementarmodell zu behandeln. Dokumenten wie Bilder oder Videodaten, welche vorrangig nur durch den Menschen ausgewertet werden sollen, verhilft ein Elementarmodell-Wrapper zur Teilnahme an Multimodellen. Für XML-basierte Elementarmodell-Typen konnte eine generische Erweiterung geschaffen werden. Diese erlaubt das Hinzufügen solcher Fachmodell-Typen per deklarativer Angabe von Element-Typen und XPath-Ausdrücken auf Nutzerebene – ohne Programmieraufwand und Kompilieren.

Die Umsetzung und Bereitstellung der Konzepte als IKT-Komponenten auf verschiedenen Ebenen – von der Datenstruktur über Bibliotheken und Services bis hin zur alleinstehenden, universellen Multimodell-Software M2A2 – erlaubt die sofortige, direkte Anwendung der Multimodell-Methode in der Praxis. Für abgrenzbare Szenarien ist es per Multimodell-Spezialisierung darüber hinaus möglich, auf jeder dieser Ebenen die bereitgestellten IKT-Komponenten wiederzuverwenden, zu erweitern oder völlig neu zu entwerfen, wobei die Erkenntnisse dieser Arbeit das methodische Gerüst bilden.

Es wurde gezeigt, wie sich Multimodelle in bestehende Fachanwendungen integrieren lassen. Dabei kann MMQL von einem Endnutzer oder einem Computersystem zur Kommunikation mit den bereitgestellten Werkzeugen benutzt werden. Die Verwandtschaft zur bekannten Datenbanksprache SQL und die Rückgabe der Ergebnisse als tabellarisches ResultSet sollen bewährte Konzepte aufgreifen und eine möglichst schnelle Einarbeitung erlauben.

Qualitäts- und Zeitvorteil gegenüber manueller Datenermittlung

Die traditionelle, manuelle Arbeitsweise erlaubt keine präzise und zugleich effiziente Lösung von umfangreichen domänenübergreifenden Aufgabenstellungen. Dagegen ermöglicht die multimodellbasierte Arbeitsweise durch ihre Automatisierung eine prinzipbedingte Fehlerfreiheit; eine MMQL-Anweisung wird auf denselben Ausgangsdaten immer identische Resultate produzieren. Für das Anwendungsbeispiel wurde ein nomineller Geschwindigkeitsvorteil zwischen Faktor 214,6 und Faktor 482,8 gegenüber einer manuellen Lösung festgestellt.

Der Aufwand zur Einbindung eines neuen Elementarmodell-Typs ist abhängig von der Verfügbarkeit unterstützender Software-Bibliotheken sowie der Erfahrung des Erweiterungsentwicklers. Aus architektonischer Sicht entsteht ein Vorteil, wenn in der Plattform Basisklassen für das zugrundeliegende Meta-Datendatenmodell wiederverwendet werden können. Zum Beispiel wird für XML-basierte Elementarmodelle eine solche Komponente auf Basis von EMF zur Verfügung gestellt. Dadurch ist es möglich, ein mittelgroßes Datenmodell (20…50 relevante Element-Typen) mit einem geschätzten Aufwand von 0,5 Personentagen in die Plattform zu integrieren. Hinzu kommen die Aufwendungen zur Erstellung von Elementarmodell-Viewern und -Filtern. Die Erstellung eines Tree-Viewers für ein solches Datenmodell wird – je nach gewünschtem Funktionsumfang und der Struktur des Datenmodells – mit weiteren 0,5…2,0 Personentagen abgeschätzt.

Zielerfüllung der Arbeit

Ziel der Arbeit war die Entwicklung einer Methode zur Erzeugung, Übertragung und Erschließung datenformat-, datenmodell- und domänenübergreifender Informationsräume.

Die zuvor beschriebenen Ergebnisse erfüllen gemeinsam die gestellte Zielsetzung. Da die Übertragung von Multimodellen sich nicht von der Übertragung bisheriger Fachmodelle

unterscheidet, wurde in der Arbeit auf diesen Aspekt nicht näher eingegangen. Zur Übertragung können bislang etablierte Mittel wie bspw. E-Mail, FTP oder Cloud-Speicher zum Einsatz kommen.

Hypothese 1 wird durch ID-basierte Links gestützt.

Forschungsbemühungen zu Mapping, Partialmodellen, domänenübergreifenden Produktdatenmodellen, nD-Modelling, Linked-Data sowie die Vorschläge zur Erweiterung bestehender Datenmodelle um neue Domänen untermauern den Bedarf nach Strukturen und Methoden für eine interdisziplinäre Datenabbildung und -analyse. Im Falle des Multimodells werden diese Strukturen durch ID-basierte Links realisiert. Jeder Link stellt dabei eine neue Informationseinheit dar, deren modellübergreifender Verknüpfungswert deutlich höher ist als der ursprüngliche Informationswert der zuvor unverlinkten Elemente.

Hypothese 2 wird durch externe Links gestützt.

Durch die Verwendung externer Links bleiben die Elementarmodelle unverändert. Dadurch ist ihre Datenintegrität gewährleistet. Informationen können sowohl getrennt für jede Domäne als auch als domänenübergreifender Informationsraum ausgewertet werden.

Dass Elementarmodelle nach einer Teilnahme in einem Multimodell weiterhin in existierenden Fachanwendungen benutzt werden können, ist experimentell überprüfbar, z. B. durch byteweisen Vergleich, die Anwendung einer Hash-Funktion oder durch eine manuelle Überprüfung mit der realen Fachanwendung. Lediglich durch den Einsatz manipulativer Ausweichlösungen bei fehlenden IDs kann die Datenintegrität nicht gewährleistet werden.

Hypothese 3 wird durch die prototypische Implementierung von MMQL, Interpreter und M2A2 gestützt.

In der Arbeit wurden die Methoden zum strukturellen Aufbau und der Erschließung des multimodellbasierten Informationsraumes dargelegt. Diese wurden so entworfen, dass eine modulare Implementierung möglich ist. Anhand der prototypischen Realisierung konnte dies überprüft werden.

7.3. Ausblick

Die Ergebnisse dieser Arbeit bieten eine methodische und technische Hilfestellung für alle Forschungs- und Entwicklungsvorhaben, welche sich mit interdisziplinären Aufgabenstellungen des Bauwesens befassen. Die Bereitstellung von Werkzeugen zur Erschließung domänenübergreifender Informationsräume entbindet Wissenschaftler und Programmierer von der technischen Entwicklungsarbeit bezüglich der Datenformate und des Datenaustauschs. Vielmehr ist es nun möglich, sich bei solchen Projekten auf die semantischen, baufachlichen Problemstellungen zu konzentrieren. Als mögliches Anwendungsgebiet seien hier exemplarisch Cyber Physical Systems genannt, bei denen die Verlinkung physischer Objekte mit virtuellen Modellen neue Anwendungen in Simulation und Steuerung von Bauprozessen ermöglicht (Sørensen et al., 2008; Wetter, 2011).

Die Umsetzung und Anwendung von Multimodellen in der industriellen Praxis erfordert die Standardisierung von Datenformaten und Methoden. Aus diesem Grund wurde am 8. Oktober 2013 das buildingSMART-Projekt "Multimodelle" gegründet[1]. Dort soll auf Basis des Mefisto-

[1] http://www.buildingsmart.de/kos/WNetz?art=News.show&id=174

Projektes ein Multimodell-Container für die modellbasierte Ausschreibung und Vergabe ausgearbeitet und zur Standardisierung bei buildingSMART und beim VDI vorgeschlagen werden.

Die vorgestellte Methode des Multimodell-Filterns liefert Ergebnisse in Form eines allgemeingültigen, tabellarischen ResultSets. Diese Repräsentation eignet sich gut für eine technische Weiterverarbeitung, ist aber als Visualisierungskonzept für einen Nutzer ungeeignet. Daher müssen Methoden entwickelt werden, welche die unkomplizierte Erstellung von Multimodell-Viewern erlauben. Multimodell-Viewer visualisieren die Daten der verschiedenen Domänen eines gemeinsamen Informationsraumes nutzbringend. Vorschläge und Ansätze zu dieser Thematik finden sich bei z. B. bei Bretthauer et al. (2001); Karlshøj (2007); Tauscher & Scherer (2012). Die Relevanz von Multimodell-Viewern steigt mit der fortschreitenden Einführung des Building Information Modeling in realen Bauprojekten. Dort könnte der erhöhte Bedarf an Informationsmanagement sogar zur Einführung neuer Rollen wie Project Information Officer (PIO; Froese, 2007) oder BIM-Manager (Liebich et al., 2011) führen.

Des Weiteren wäre es notwendig, Multimodell-Filterergebnisse wiederum als valides Multimodell abzubilden. Dabei würde sich der Inhalt der Elementar- und Linkmodelle reduzieren. Eine Herausforderung besteht in der Erstellung valider Elementarmodelle, welche aus Konformität zum Schema möglicherweise mehr Daten als das eigentliche Filterergebnis enthalten könnten. Solche reduzierten Multimodelle würden einen kontinuierlichen Informationsfluss in multimodellbasierten Bauinformationsprozessen gewährleisten.

Die in den Multimodellen verwendeten Metadaten erlauben die Einordnung eines Multimodells in einen größeren, projektweiten oder -übergreifenden Kontext. So können innerhalb einer virtuellen Organisation zum einen Berechtigungen in Abhängigkeit des Multimodell-Inhalts vergeben werden. Zum anderen ist es möglich, verkettete Bauinformationsprozesse auf Basis der Metadaten zu definieren, Informationsbedarfe konkret zu spezifizieren sowie Schlussfolgerungen über erledigte und noch notwendige Arbeitsschritte zu ziehen. Den aktuellen Stand zu Forschungsergebnissen dieser Thematik präsentieren Hilbert & Scherer (2012); Schapke & Hilbert (2012b).

Gleichfalls müssten die Metadaten als Kriterium für MMQL-Abfragen innerhalb des Multimodells herangezogen werden können. Zur Zeit hat ein Nutzer Elementar- und Linkmodelle für Abfragen explizit zu benennen. Über Metadaten-Kriterien ließen sich MMQL-Anfragen verallgemeinern, da Elementar- und Linkmodelle dann anhand ihrer Annotationen im Multimodell bestimmt werden könnten.

Genauso sind in MMQL-Abfragen die Namen der Element-Typen und Properties abhängig vom jeweiligen Datenformat und der Implementierung der Element-Query-Erweiterung. Um insgesamt eine höhere Unabhängigkeit der Multimodell-Abfragen vom genutzten Datenformat zu erlangen, ist ein neutrales, möglichst globales Vokabular notwendig. Generelle Ansätze für entsprechende Klassifikationen im Bauwesen sind OmniClass[2], IFD[3] und in gewisser Hinsicht auch die IFC Property Set Definitions[4]. Spezielle Ansätze gibt es darüber hinaus z. B. für den Bereich Bauprojektmanagement (Scherer & Schapke, 2011) oder ganzheitliches Energiemanagement (Scherer et al., 2012). Für den Einsatz in Multimodellen müssen Vokabulare jedoch nicht nur einen fachmodellübergreifenden Konsens darstellen, sondern auch baufremde Domänen beschreiben können.

[2] `http://www.omniclass.org/`
[3] International Framework for Dictionaries; `http://www.ifd-library.org/`
[4] `http://www.buildingsmart-tech.org/specifications/pset-releases`

Anhang A.

Datenmodelle und Spezifikationen

A.1. Das Generische Multimodell

Quelltext A.1: MultimodelXMI.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns:multimodel="http://cib.tu-dresden.de/multimodel"
    xmlns:xmi="http://www.omg.org/XMI" xmlns:xsd="http://www.w3.org/2001/
    XMLSchema" targetNamespace="http://cib.tu-dresden.de/multimodel">
  <xsd:import namespace="http://www.omg.org/XMI" schemaLocation="platform:
      /plugin/org.eclipse.emf.ecore/model/XMI.xsd"/>
  <xsd:complexType name="MultiModel">
    <xsd:complexContent>
      <xsd:extension base="multimodel:Annotatable">
        <xsd:choice maxOccurs="unbounded" minOccurs="0">
          <xsd:element name="elementaryModels" type="
              multimodel:ElementaryModel"/>
          <xsd:element name="linkModels" type="multimodel:LinkModel"/>
        </xsd:choice>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="MultiModel" type="multimodel:MultiModel"/>
  <xsd:complexType abstract="true" name="ElementaryModel">
    <xsd:complexContent>
      <xsd:extension base="multimodel:Annotatable"/>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="ElementaryModel" type="multimodel:ElementaryModel"/>
  <xsd:complexType name="UriElementaryModel">
    <xsd:complexContent>
      <xsd:extension base="multimodel:ElementaryModel">
        <xsd:attribute name="uri" type="xsd:string"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="UriElementaryModel" type="
      multimodel:UriElementaryModel"/>
  <xsd:complexType name="EmbeddedElementaryModel">
    <xsd:complexContent>
      <xsd:extension base="multimodel:ElementaryModel">
        <xsd:attribute name="data" type="xsd:string"/>
```

```
33        </xsd:extension>
34      </xsd:complexContent>
35    </xsd:complexType>
36    <xsd:element name="EmbeddedElementaryModel" type="
          multimodel:EmbeddedElementaryModel"/>
37    <xsd:complexType name="LinkModel">
38      <xsd:complexContent>
39        <xsd:extension base="multimodel:Annotatable">
40          <xsd:choice maxOccurs="unbounded" minOccurs="0">
41            <xsd:element name="linkedModels" type="
                  multimodel:ElementaryModel"/>
42            <xsd:element name="links" type="multimodel:Link"/>
43          </xsd:choice>
44          <xsd:attribute name="linkedModels" type="xsd:string"/>
45        </xsd:extension>
46      </xsd:complexContent>
47    </xsd:complexType>
48    <xsd:element name="LinkModel" type="multimodel:LinkModel"/>
49    <xsd:complexType name="Link">
50      <xsd:complexContent>
51        <xsd:extension base="multimodel:Annotatable">
52          <xsd:choice maxOccurs="unbounded" minOccurs="0">
53            <xsd:element name="linkedElements" type="
                  multimodel:LinkedElement"/>
54          </xsd:choice>
55        </xsd:extension>
56      </xsd:complexContent>
57    </xsd:complexType>
58    <xsd:element name="Link" type="multimodel:Link"/>
59    <xsd:complexType name="LinkedElement">
60      <xsd:choice maxOccurs="unbounded" minOccurs="0">
61        <xsd:element name="elementaryModel" type="multimodel:ElementaryModel
              "/>
62        <xsd:element ref="xmi:Extension"/>
63      </xsd:choice>
64      <xsd:attribute ref="xmi:id"/>
65      <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
66      <xsd:attribute name="elementID" type="xsd:string" use="required"/>
67      <xsd:attribute name="elementaryModel" type="xsd:string"/>
68    </xsd:complexType>
69    <xsd:element name="LinkedElement" type="multimodel:LinkedElement"/>
70    <xsd:complexType abstract="true" name="Annotatable">
71      <xsd:choice maxOccurs="unbounded" minOccurs="0">
72        <xsd:element name="metaDataEntries" type="multimodel:MetaDataEntry"/
              >
73        <xsd:element ref="xmi:Extension"/>
74      </xsd:choice>
75      <xsd:attribute ref="xmi:id"/>
76      <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
77    </xsd:complexType>
78    <xsd:element name="Annotatable" type="multimodel:Annotatable"/>
79    <xsd:complexType name="MetaDataEntry">
80      <xsd:choice maxOccurs="unbounded" minOccurs="0">
81        <xsd:element ref="xmi:Extension"/>
```

```
82      </xsd:choice>
83      <xsd:attribute ref="xmi:id"/>
84      <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
85      <xsd:attribute name="key" type="xsd:string" use="required"/>
86      <xsd:attribute name="value" type="xsd:string" use="required"/>
87    </xsd:complexType>
88    <xsd:element name="MetaDataEntry" type="multimodel:MetaDataEntry"/>
89  </xsd:schema>
```

A.2. Fachmodell Dokumentencontainer

Quelltext A.2: DocumentXML.xsd

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <xsd:schema xmlns:document="http://tu-dresden.de/cib/mmaa/document"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="http://tu
       -dresden.de/cib/mmaa/document">
3    <xsd:complexType name="Documents">
4      <xsd:sequence>
5        <xsd:element maxOccurs="unbounded" minOccurs="0" name="documents"
             type="document:Document"/>
6      </xsd:sequence>
7    </xsd:complexType>
8    <xsd:element name="Documents" type="document:Documents"/>
9    <xsd:complexType name="Document">
10     <xsd:attribute name="id" type="xsd:string"/>
11     <xsd:attribute name="mimeType" type="xsd:string"/>
12     <xsd:attribute name="uri" type="xsd:string"/>
13   </xsd:complexType>
14   <xsd:element name="Document" type="document:Document"/>
15 </xsd:schema>
```

A.3. MMQL: Formale Sprachbeschreibung

Im Folgenden wird die Syntaxdefinition der Multi-Model Query Language (MMQL) in der erweiterten Backus-Naur-Form (EBNF; ISO 14977, 1996) wiedergegeben.

Quelltext A.3: Formale Syntaxdefinition der MMQL in Erweiterter Backus-Naur-Form (EBNF)

```
1  Mmql                ::= 'use' MultiModelSpec Statement+
2  Statement           ::= Select | MMManipulation
3  Select              ::= 'select' ExplicitPropertySpec ( ','
       ExplicitPropertySpec )* 'from' ElementByName LinkedWithSpec* ( 'where'
       Assertion )? ( 'as' VARNAME )?
4  MMManipulation      ::= CreateLM | AlterLM | UpdateLM | DeleteLM
5  CreateLM            ::= 'create' 'linkmodel' ( '"' STRING '"' )? 'add' '
       permission' ElementaryModelSpec ( ',' ElementaryModelSpec )+
6  AlterLM             ::= 'alter' 'linkmodel' LinkModelSpec ( 'add' | '
       remove' ) 'permission' ElementaryModelSpec ( ',' ElementaryModelSpec )*
7  DeleteLM            ::= 'drop' 'linkmodel' LinkModelSpec
8  UpdateLM            ::= 'update' 'linkmodel' LinkModelSpec ( AddLinks |
       DeleteLinks ) ( 'where' Assertion )?
9  AddLinks            ::= 'add' ( 'distinct' | 'strict' )? 'links' 'between'
        LinkBetweenSpec ( ',' LinkBetweenSpec )+
10 DeleteLinks         ::= 'delete' 'links' 'between' ElementByName ( ','
       ElementByName )+
11 MultiModelSpec      ::= MultiModelByFile | MultiModelByEditor
12 MultiModelByFile    ::= 'file' '"' STRING '"'
13 MultiModelByEditor  ::= 'editor' '"' STRING '"'
14 ElementaryModelSpec ::= EMByIndex | EMByName
15 EMByIndex           ::= 'index' '(' INTEGER ')'
16 EMByName            ::= '"' STRING '"'
17 ElementSpec         ::= RefElementSpec | ElementByName
18 RefElementSpec      ::= VARNAME
19 ElementByName       ::= ElementaryModelSpec '.' '"' STRING '"' ( 'as'
       VARNAME )?
20 PropertySpec        ::= RefPropertySpec | ExplicitPropertySpec
21 RefPropertySpec     ::= VARNAME
22 ExplicitPropertySpec ::= ElementSpec ( PropertyByPath | PropertyAccessor |
        IDProperty ) ( 'as' VARNAME )?
23 PropertyByPath      ::= '.' PathSegment ( '~' PathSegment )*
24 PathSegment         ::= '"' STRING '"' ( '[' INTEGER ']' )?
25 IDProperty          ::= '.' 'id'
26 PropertyAccessor    ::= '?' '(' '"' STRING '"' ',' '"' STRING '"' ')'
27 LinkedWithSpec      ::= CrossLinkedWith | LinkedWithByLM
28 CrossLinkedWith     ::= 'cross' 'linkedwith' ElementByName
29 LinkedWithByLM      ::= ( 'strict' | 'trans' )? 'right'? 'linkedwith'
       ElementByName ( 'by' LinkModelSpec ( ',' LinkModelSpec )* )?
30 LinkModelSpec       ::= LMByName | LMByIndex
31 LMByName            ::= '"' STRING '"'
32 LMByIndex           ::= 'index' '(' INTEGER ')'
33 LinkBetweenSpec     ::= ( 'one' | 'many' ) ElementByName
34 Value               ::= BooleanValue | StringValue | IntegerValue |
       RealValue | DateValue | UnaryValueOperation | BinaryValueOperation |
       TernaryValueOperation | NAryValueOperation
35 BooleanValue        ::= 'bool:' ( PropertySpec | ( 'true' | 'false' ) )
```

```
36 StringValue          ::= PropertySpec | '"' STRING '"'
37 IntegerValue         ::= 'int:' ( PropertySpec | INTEGER )
38 RealValue            ::= 'real:' ( PropertySpec | REAL )
39 DateValue            ::= 'date:' ( PropertySpec | '"' STRING '"' )
40 UnaryValueOperation ::= ValueOperationBrackets | Negation |
      FunctionOperation
41 ValueOperationBrackets ::= '(' Value ')'
42 Negation             ::= '-' Value
43 FunctionOperation    ::= ( 'sin' | 'cos' | 'tan' | 'sinh' | 'cosh' | 'tanh'
       | 'deg' | 'rad' | 'asin' | 'acos' | 'atan' | 'exp' | 'log' | 'log10' |
       'round' | 'length' | 'lower' | 'upper' | 'trim' | 'reverse' ) '('
      Value ')'
44 BinaryValueOperation ::= Addition | Multiplication| Exponentiation |
      StringPosition | StringRegexp
45 Addition             ::= Value ( '+' | '-' ) Value
46 Multiplication       ::= Value ( '*' | '/' | '%' ) Value
47 Exponentiation       ::= Value '^' Value
48 StringPosition       ::= 'position' '(' Value ',' Value ')'
49 StringRegexp         ::= 'regexp' '(' Value ',' Value ')'
50 TernaryValueOperation ::= ('substring' | 'replace') '(' Value ',' Value ',
      ' Value ')'
51 NAryValueOperation   ::= 'concat' '(' Value (',' Value)* ')'
52 Assertion            ::= UnaryAssertion | BinaryAssertion | IsNull |
      ElementIn | BinaryValueAssertion
53 UnaryAssertion       ::= AssertionBrackets | Not
54 AssertionBrackets    ::= '(' Assertion ')'
55 Not                  ::= 'not' Assertion
56 BinaryAssertion      ::= And | Or | Xor
57 And                  ::= Assertion 'and' Assertion
58 Or                   ::= Assertion 'or' Assertion
59 Xor                  ::= Assertion 'xor' Assertion
60 IsNull               ::= Value 'is null'
61 ElementIn            ::= InViewerSelection | InExternalFilter |
      InSelectStatement
62 InViewerSelection    ::= ElementSpec 'in' 'viewselect' ( '(' '"' STRING '"'
       ')' )?
63 InExternalFilter     ::= ElementSpec 'in' 'filter' '(' '"' STRING '"' ( ','
       FilterOption )* ')'
64 FilterOption         ::= '"' STRING '"' '=' '"' STRING '"'
65 InSelectStatement    ::= ElementSpec 'in' VARNAME
66 BinaryValueAssertion ::= EqualityAssertion | CompareAssertion |
      StringLikeAssertion
67 EqualityAssertion    ::= Value ( '==' | '!=' ) Value
68 CompareAssertion     ::= Value ( '<' | '<=' | '>' | '>=' ) Value
69 StringLikeAssertion ::= Value 'like' Value
70 VARNAME              ::= ( [A-Z] | [a-z] ) ( [A-Z] | [a-z] | [0-9] | '_' )*
71 STRING               ::= [!-~]*
72 INTEGER              ::= '0' | [1-9] [0-9]*
73 REAL                 ::= ( '0' | [1-9] [0-9]* ) '.' [0-9]+ ( ( 'e' | 'E' )
      ( '+' | '-' )? [0-9]+ )?
```

Anhang B.

Elementarmodell-Vokabulare

Im Folgenden sind die Elementarmodell-Vokabulare des Mefisto-Projektes nach Schapke & Hilbert (2012a,b) aufgelistet. Hierarchische Strukturen wurden in flache Listen von Wertevorräten überführt und nach ihrem Bezeichner gruppiert.

B.1. Domain

Tabelle B.1.: Wertevorrat für Domain

Symbol	Terminus
BIM	Building Information Model
BIM.BSS	Building Spatial System
BIM.BSS.BPS	Building Spatial Zoning Structure
BIM.BSS.BUS	Building Spatial Room Structure
BIM.BEM	Building Element Model
BIM.BEM.BES	Building Element Structure
BIM.BEM.BDS	Building Distribution System
BIM.BEM.BFS	Building Furnishing System
BIM.OTH	Other Building Information Model
SPM	Specification Model
SPM.PRG	Functional Spatial Program
SPM.BOQ	Bills Of Quantity
SPM.QTO	Quantity Takeoff
SPM.QTO.MCQ	Model Based Calculated Quantities
SPM.QTO.MBQ	Model Basic Quantities
SPM.OTH	Other Specification Model
TSM	Time Schedule Model
TSM.MSS	Mile Stone Schedule
TSM.PCS	Project Controlling Schedule
TSM.CES	Construction Execution Schedule
TSM.OTH	Other Time Schedule Model
COM	Cost Model
COM.DEC	Design Estimate Costs
COM.BEC	Bid Estimate Costs
COM.WEC	Work Estimate Costs
COM.OTH	Other Cost Model

Fortsetzung auf der nächsten Seite ...

... Fortsetzung von der vorherigen Seite

Symbol	Terminus
RIM	Risk Model
RIM.CAT	Risk Catalog
RIM.PRL	Project Risk List
CSM	Construction Site Model
CSM.CSL	Construction Site Layout

B.2. Phase

Tabelle B.2.: Wertevorrat für Phase

Symbol	Terminus
P	Projekt
P.1	Konzept
P.1.1	Grundlagenermittlung
P.1.2	Vorplanung
P.2	Planung
P.2.1	Entwurf
P.2.1.1	Genehmigungsplanung
P.2.1.2	Ausführungsplanung
P.3	Vergabe und Vertrag
P.3.1	Ausschreibung
P.3.1.1	Vergabeunterlagen erstellen
P.3.1.1.1	Vorbereitung der Ausschreibung zur Koordination der Vergaben
P.3.1.1.1.1	Aufbereitung der Planung
P.3.1.1.1.2	Anforderungen an das Bauwerk aktualisieren
P.3.1.1.1.3	Vergabeeinheiten festlegen
P.3.1.1.1.4	Steuerungsterminplan erstellen/fortschreiben
P.3.1.1.1.5	Risikoliste erstellen
P.3.1.1.2	Projektstrukturierung und Modellaufbereitung
P.3.1.1.3	Leistungsverzeichnis erstellen
P.3.1.1.4	Baubeschreibung erstellen
P.3.1.1.5	Risikoliste / Maßnahmen für Ausschreibung aktualisieren
P.3.1.1.6	Qualitätssicherung Ausschreibung und Vergabeunterlagen zusammenstellen
P.3.1.2	Bekanntmachung und Vergabeunterlagen bereitstellen
P.3.1.2.1	Bekanntmachung
P.3.1.2.2	Vergabeunterlagen bereitstellen
P.3.2	Angebotserstellung
P.3.2.1	Überprüfen des Entwurfs
P.3.2.1.1	Modellprüfung und Erstellung Koordinationsmodelle zur Angebotserarbeitung (inkl. Mengenermittlung)

Fortsetzung auf der nächsten Seite ...

... Fortsetzung von der vorherigen Seite

Symbol	Terminus
P.3.2.1.2	Beurteilung Tragwerk
P.3.2.1.3	Beurteilung Bodenverh. / Überprüfung Gründungskonzept / Baugruben
P.3.2.1.4	Überprüfung Haustechnische Anlagen
P.3.2.1.5	Überprüfung Nutzungskonzept
P.3.2.1.6	Überprüfung Ausstattungskonzept
P.3.2.1.7	Überprüfung Energiekonzept
P.3.2.1.8	Prüfung Fassadenkonstruktion
P.3.2.1.9	Kaufmännische Vertragsprüfung
P.3.2.1.10	Rechtliche Vertragsprüfung
P.3.2.1.11	Konsolidierung der Fachplanerprüfungen
P.3.2.2	Bauarten / -methoden & Ausstattung
P.3.2.2.1	Konsolidierung Entwurfsprüfung (inkl. Mengenprüfung)
P.3.2.2.2	Festlegen der Baumethoden
P.3.2.3	Vorbereitung Terminplan
P.3.2.3.1	Überprüfen der Bauzeiten
P.3.2.4	Baustelleneinrichtung
P.3.2.4.1	Überprüfen und Detaillierung der Baustelleneinrichtung, Erstellung der Baugeräteliste
P.3.2.4.2	Überprüfen der Infrastruktur Baustelle / Versorgungssicherheit
P.3.2.5	Kalkulation
P.3.2.5.1	Überprüfung der Leistungsbeschreibung
P.3.2.5.2	Preisanfragen Nachunternehmerleistung
P.3.2.5.3	Preisanfragen Baustoffe, Bauhilfsstoffe, Geräte
P.3.2.5.4	Preisbildung
P.3.2.5.5	Risikobewertung
P.3.2.5.6	Formulierung Angebotsbedingungen / Preise / Risiko
P.3.2.5.7	Endformulierung / Verabschiedung Angebot
P.3.2.6	Erstellen der Nebenangebote
P.3.2.7	Angebotsunterlagen zusammenstellen und abgeben
P.3.2.7.1	Darstellung des Angebotes, Gesamtbewertung, Technische Präsentation
P.3.2.7.2	Angebotsunterlagen zusammenstellen und veröffentlichen
P.3.3	Vergabe- und Vertragsverfahren
P.3.3.1	Vergabe mit nichtoffenem Verfahren bzw. Verhandlungsverfahren
P.3.3.1.1	Teilnahmeanträge von Vergabeplattform übernehmen
P.3.3.1.2	Auswahl Bewerber
P.3.3.2	Vergabe mit offenem Verfahren
P.3.3.2.1	Angebote von Vergabeplattform übernehmen und prüfen
P.3.3.2.2	Angebote durch AG bewerten
P.3.3.2.3	Aufklärungsgespräch
P.3.3.3	Vertrag
P.3.3.3.1	Vertragsabschluss AG
P.4	Erstellung
P.4.1	Arbeitsvorbereitung
P.4.1.1	Konsolidierung der Ausführungsunterlagen und -pläne

Fortsetzung auf der nächsten Seite ...

... Fortsetzung von der vorherigen Seite

Symbol	Terminus
P.4.1.1.1	Strukturierung der Fachmodelle Modelle für die Arbeitsvorbereitung
P.4.1.2	Ausführungsplanung / Freigabeverwaltung
P.4.1.2.1	Modellprüfung und Erstellung Koordinationsmodelle zur Arbeitsvorbereitung / Ausführungsplanung / Freigabenverwaltung
P.4.1.2.2	Erstellung prüffähiger Statik
P.4.1.2.3	Erstellung prüffähiger Gründungsstatik / Ausführungsstatik Verbau
P.4.1.2.4	Auslegung / Planung Haustechnische Anlagen
P.4.1.2.5	Montageplanung
P.4.1.2.6	Entwurfsplanung Architektur
P.4.1.2.7	Werkplanung Architektur
P.4.1.2.8	Zuarbeit Ausstattungsplanung zur Werkplanung
P.4.1.2.9	Detaillierte Energieberechnung / Zuarbeit zur Werkplanung
P.4.1.2.10	Werksplanung Fertigteile
P.4.1.2.11	Werksplanung Stahlbau
P.4.1.2.12	Werksplanung Holzbau
P.4.1.2.13	Werkplanung Fassade
P.4.1.2.14	Werksplanung Fahraufzüge / Rolltreppen
P.4.1.2.15	Konsolidierung der Fachplanerprüfungen
P.4.1.3	Detaillierter Terminplan
P.4.1.3.1	Detaillierten Terminplan aufstellen
P.4.1.3.2	Finanzterminplan als Unterplan Masterplan
P.4.1.4	Detaillierte Baustelleneinrichtung
P.4.1.4.1	Detaillierung der Baustelleneinrichtung
P.4.1.4.2	Planung Hilfsmaßnahmen (temporäre Hilfsmaßnahmen) Rüstungen / Gerüste / Verbau
P.4.1.4.3	Schalungsplanung / Abschnittsplanung Ausbau (temporäre direkte Maßnahmen / Zwischenlager)
P.4.1.5	Logistiksimulation Varianten
P.4.1.6	Montagesimulation Varianten
P.4.1.7	Budgetierung
P.4.1.7.1	Budget aufstellen und verwalten
P.4.1.7.2	Kontenpläne aufstellen (Kostenstellen), Zuordnung LV
P.4.1.7.3	Freigabe Finanzplanung
P.4.1.8	Baustellenanweisungen
P.4.2	Bauausführung AG
P.4.2.1	Konsolidierung Ausführungsunterlagen AG
P.4.2.2	Leistungsänderungen AG
P.4.2.2.1	Nachtrag (NA1) durch: Mengenänderungen / zusätzliche Leistungen / Änderung des Bauablaufs
P.4.2.2.2	Nachtrag (NA2) durch: Änderung des Bauentwurfs
P.4.2.3	Baufortschrittskontrolle AG
P.4.2.4	Rechnungslegung
P.4.2.4.1	Leistungen feststellen
P.4.2.4.2	Leistungen abrechnen

Fortsetzung auf der nächsten Seite ...

... Fortsetzung von der vorherigen Seite

Symbol	Terminus
P.4.2.5	Kostenkontrolle und -prognose
P.4.2.5.1	Kostenkontrolle Vertrag
P.4.2.5.2	Kostenstand Projekt
P.4.2.5.3	Kostenprognose Projekt
P.4.2.5.4	Mittelabflusskontrolle
P.4.2.5.5	Steuerungsmaßnahmen
P.4.2.6	Qualitätssicherung der Bauausführung
P.4.2.6.1	Qualitätssicherung Produktion, Personal, Baustelleneinrichtung, Baustoffe, Prozesse
P.4.2.6.2	Abnahme der Bauleistung
P.4.3	Bauausführung AN
P.4.3.1	Konsolidierung Ausführungsunterlagen AN
P.4.3.2	Planverwaltung AN
P.4.3.2.1	Vorbereitung der Planverwaltung
P.4.3.2.2	Planverwaltung
P.4.3.3	Beschaffung
P.4.3.3.1	Vertrag von Subunternehmerleistung
P.4.3.3.2	Vertrag von Zulieferleistung
P.4.3.4	Qualitätssicherung AN
P.4.3.4.1	QM-Plan fortschreiben
P.4.3.4.2	Grundwasserstandkontrolle
P.4.3.4.3	Material- / Produktkontrollen (Dichtigkeit, Energie, Schallschutz, Brandschutz, Beton)
P.4.3.4.4	Geometriekontrolle
P.4.3.4.5	Reporting Qualität
P.4.3.5	Logistik und Einsatzplanung
P.4.3.6	Baufortschrittskontrolle AN
P.4.3.6.1	Ist / Soll Erhebung Termine
P.4.3.6.2	Baufortschritt prüfen und an AG melden
P.4.3.6.3	Anpassung Terminplan, Umplanung
P.4.3.7	Kosten- und Leistungskontrolle AN
P.4.3.7.1	Ist / Soll Erhebung Kosten & Leistungen (Mengen)
P.4.3.7.2	Ist Leistungen prüfen und an AG melden
P.4.3.7.3	Rechnungsprüfung AG / Inkassoprüfung AN
P.4.3.8	Leistungsänderungen AN
P.5	Nutzung

B.3. Level of Detail

Der Wertevorrat für den Aspekt *Level of Detail* wurde in Anlehnung an den Standard CityGML(OGC CityGML, 2012) definiert:

Tabelle B.3.: Wertevorrat für Level of Detail

Symbol	Terminus
0	Regional Landscape
1	City & Region
2	Property, City District & Projects
3	Building Exterior & Sections
4	Building Elements & Interior
5	Building Element Components

B.4. Status

Tabelle B.4.: Wertevorrat für Status

Symbol	Terminus
α	Model Requests and Templates
β	Model Draft
γ	Accepted / Released Models
ϵ	Deprecated / Rejected Models

Anhang C.

Implementierungsdetails

C.1. Liste der in M2A2 implementierten Baufachmodelle

Tabelle C.1.: In M2A2 implementierte Baufachmodelle; Art der Erweiterung: Programmatisch (P), Deklarativ (D)

Meta-Datenstruktur	Fachmodell-Typ	Beschreibung	Art
XML	TSM–Mefisto	Vorgangsmodell des Mefisto-Projekts	P
	RIM–Mefisto	Risikomodell des Mefisto-Projekts	P
	SPM.QTO–Mefisto	Modell für REEB-konforme Mengenermittlung des Mefisto-Projekts	P
	COM.Calculation–Mefisto	Modell für Kalkulationsansätze des Mefisto-Projekts	D
	TSM.Process–Mefisto	IFC-konformes, XML-basiertes Prozessmodell	P
	SPM.BOQ–GAEB	Leistungsverzeichnisse nach GAEB-DA-XML	P
MOF (XMI)	Sensor Data–Hesmos	Vereinfachtes Sensordaten-Modell des Hesmos-Projekts	P
	Wind Data–Sara	Vereinfachtes Windlastprofil-Modell des Sara-Projekts	P
CSV	Multi–CSV	Comma Separated Values, Fachsemantik ergibt sich auf Nutzerebene	P
Express	BIM–IFC	IFC 2x3 in SPF	P
—	TSM–iCal	Terminmodell im iCalendar-Format (RFC 5545, 2009)	P

C.2. Implementierte Erweiterungen der M2A2-Plattform

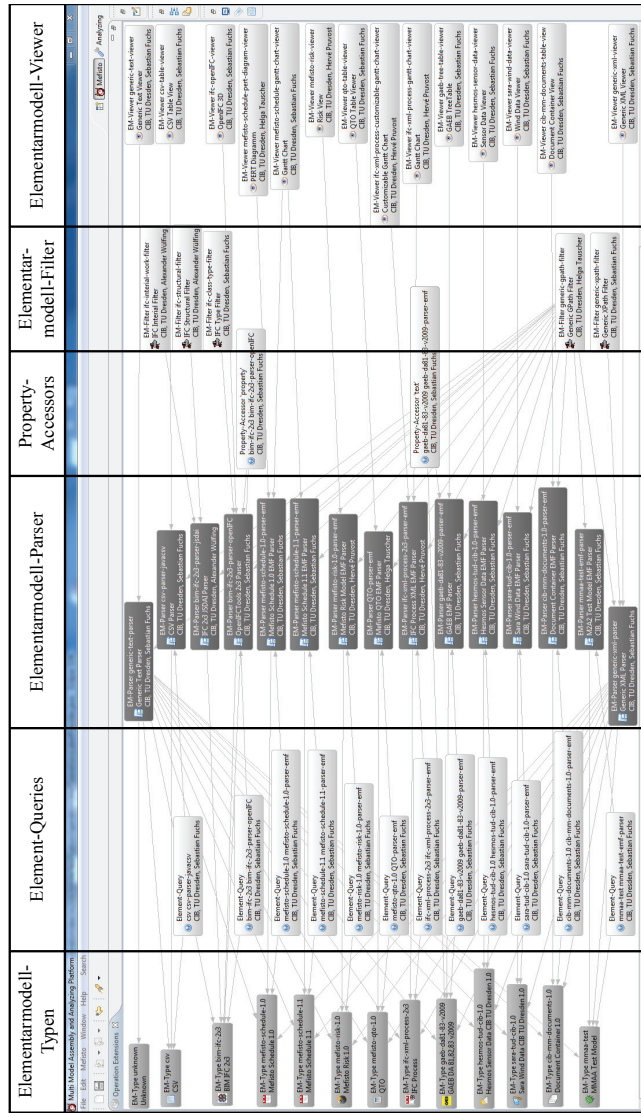


Abbildung C.1.: Bildschirmaufnahme der in M2A2 implementierten Fachmodule und deren Abhängigkeiten untereinander. Jegliche baufachliche Funktionalität wird in M2A2 über Erweiterungen (Plugins) bereitgestellt. Damit wird die fachliche Neutralität des Multimodell-Kerns gesichert. Erweiterungen können von anderen Plugins wiederverwendet werden, wodurch sich direkte Abhängigkeiten ergeben.

C.3. XML-Schema des Mefisto-Multimodell-Containers

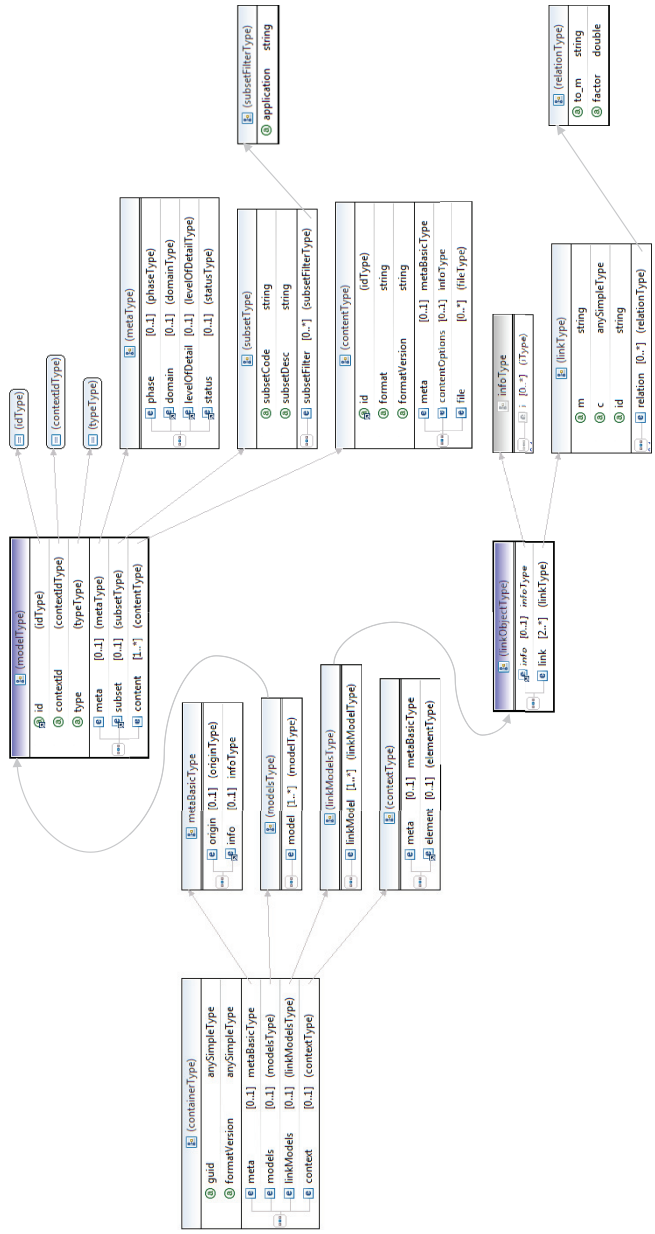


Abbildung C.2.: XML-Schema des Mefisto-Multimodell-Containers (Auszug; in Anlehnung an Muntzinger & Fuchs, 2010; Demharter & Pflug, 2011)

Literaturverzeichnis

A

Adachi, Y. (2002). *Overview of Partial Model Query Language*. Technical report, VTT Building and Transport / SECOM Co. Ltd., Intelligent Systems Lab. VTT Report VTT-TEC-ADA-12.

Amor, W., A., & Ge, C. W. (2002). Mapping IFC versions. In Ž. Turk & R. J. Scherer (Eds.), *eWork and eBusiness in Architecture, Engineering and Construction: Proceedings of the Fourth European Conference on Product and Process Modelling in the Building and Related Industries, Portorož, Slovenia, 9–11 September 2002* (pp. 373–377).: Taylor & Francis.

Aouad, G., Lee, A., & Wu, S., Eds. (2006). *Constructing the Future: nD Modelling*. Routledge.

Arbeitskreis Bauinformatik (2000). Bauinformatik – Fachgebiet des Bauingenieurwesens an den deutschsprachigen Universitäten. `http://www.ak-bauinformatik.tu-berlin.de/menue/denkschrift/`.

Augenbroe, G. (1994). An overview of the COMBINE project. In Scherer (1994a), (pp. 547–554).

B

Bargstädt, H.-J. & Ailland, K., Eds. (2011). *Proceedings of the 11th International Conference on Construction Applications of Virtual Reality CONVR 2011*, number 21 in Schriftenreihe der Professur Baubetrieb und Bauverfahren. Verlag der Bauhaus-Universität Weimar.

Baumgärtel, K., Katranuschkov, P., & Scherer, R. J. (2012). Design and software architecture of a cloud-based virtual energy laboratory for energy-efficient design and life cycle simulation. In Gudnason & Scherer (2012), (pp. 9–16).

Beetz, J. (2009). *Facilitating distributed collaboration in the AEC/FM sector using Semantic Web Technologies*. PhD thesis, Technische Universiteit Eindhoven.

Beetz, J., van Berlo, L., de Laat, R., & van den Helm, P. (2010). Bimserver.org – An Open Source IFC Model Server. In Thabet (2010).

Bell, H., Bjørkhaug, L., Bjaaland, A., & Grant, R. (2008). *IFD Library White Paper*. Technical report, buildingSMART.

Björk, B. C. (1994). RATAS project – developing an infrastructure for computer-integrated construction. *Journal of Computing in Civil Engineering*, 8(4), 401–419.

Björk, B. C. (1995). *Requirements and Information Structures for Building Product Data Models*. PhD thesis, Helsinki University of Technology.

Borrmann, A. (2007). *Computerunterstützung verteilt-kooperativer Bauplanung durch Integration interaktiver Simulationen und räumlicher Datenbanken*. PhD thesis, Technische Universität München.

Bretthauer, G., Dietze, S., Häfele, K.-H., Isele, J., & Jäkel, J. (2001). *Nachhaltiges Planen, Bauen und Wohnen im Informationszeitalter*. Number 6626 in Wissenschaftliche Berichte, Forschungszentrum Karlsruhe, Technik und Umwelt FZKA. Selbstverlag, Online-Ressource.

Brooks, F. P. (2003). *Vom Mythos des Mann-Monats*. mitp-Verlag.

Bubner, A. & Friedrich, T. (2003). Klassifikation und Defintion baufachlicher Verknüpfungstypen als Basis zur Modellüberführung im konstruktiven Ingenieurbau. In *Internationales Kolloquium über Anwendungen der Informatik und Mathematik in Architektur und Bauwesen*, number 16 in IKM (pp. 137–140).: Bauhaus-Universität Weimar.

Bundesinstitut für Bau-, Stadt- und Raumforschung BBSR (2012b). Bauwirtschaft in den Regionen – Die bauwirtschaftliche Entwicklung in Deutschland auf regionaler Ebene. Online-Publikation BBSR-Analysen KOMPAKT 3/2012.

Bundesinstitut für Bau-, Stadt- und Raumforschung BBSR (2012a). Bericht zur Lage und Perspektive der Bauwirtschaft 2012. Online-Publikation BBSR-Analysen KOMPAKT 13/2012.

Bundesministerium für Verkehr, Bau und Stadtentwicklung (BMVBS) (2012). GAEB – Europäischer Vergleich – Wissenschaftliche Untersuchung der verschiedenen europäischen Datenaustauschformate für Ausschreibung und Vergabe im Vergleich zum GAEB-Datenaustauschformat. BMVBS-Online-Publikation, Nr. 17/2012.

C

Carter, G. (2003). *LDAP System Administration*. O'Reilly Media, 1st edition.

Cerovsek, T. (2008). On AEC query formulation techniques. In Zarli & Scherer (2008), (pp. 269–277).

Chavada, R., Dawood, N., & Kassem, M. (2012). Construction workspace management: the development and application of a novel nD planning approach and tool. *ITcon*, 17, 213–236.

Chen, P. P.-S. (1976). The entity-relationship model – toward a unified view of data. *ACM Trans. Database Syst.*, 1(1), 9–36.

Codd, E. F. (1979). Extending the database relational model to capture more meaning. *ACM Trans. Database Syst.*, 4(4), 397–434.

Conrad, S. (1997). *Föderierte Datenbanksysteme: Konzepte Der Datenintegration*. Springer, illustrated edition.

Cooper, K. & Torczon, L. (2011). *Engineering a Compiler, Second Edition*. Morgan Kaufmann, 2nd edition.

Crawley, D. B., Lawrie, L. K., Winkelmann, F. C., Buhl, W. F., Huang, Y., Pedersen, C. O., Strand, R. K., Liesen, R. J., Fisher, D. E., Witte, M. J., & Glazer, J. (2001). EnergyPlus: creating a new-generation building energy simulation program. *Energy and Buildings*, 33(4), 319–331. Special Issue: BUILDING SIMULATION'99.

Curry, E., O'Donnell, J., Corry, E., Hasan, S., Keane, M., & O'Riain, S. (2012). Linking building data in the cloud: Integrating cross-domain building data using linked data. *Advanced Engineering Informatics*, 27(2), 206–219.

D

D'Anjou, J., Fairbrother, S., Kehn, D., Kellerman, J., & McCarthy, P. (2004). *The Java Developer's Guide to Eclipse*. Addison-Wesley Professional, 2nd edition.

Daum, B. (2007). *Rich-Client-Entwicklung mit Eclipse 3.3. – Anwendungen, Plugins und Rich Clients*. dpunkt.verlag, 3., überarb., erw. edition.

Demharter, J. & Pflug, C. (2011). Einsatz von Multimodellen zur Projektabwicklung beim Auftragnehmer. In Scherer et al. (2011), (pp. 43–57).

Denton, T., Jones, E., Srinivasan, S., Owens, K., & Buskens, R. W. (2008). NAOMI – An Experimental Platform for Multi-modeling. In *Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems*, MoDELS '08 (pp. 143–157). Berlin, Heidelberg: Springer-Verlag.

Diaz, J. & Petersen, M. (2002). Integrierte Gebäudeplanung auf Basis eines modellbasierten Projektkommunikationssystems. In *VDI-Jahrbuch Bautechnik 2002*. VDI - Verlag GmbH, Düsseldorf.

Diedrichsen, B. (2008). Development of a domain-specific language for model transformations and its application to a model migration problem. In Friese et al. (2008).

DIN 1302 (1999). Deutsches Institut für Normung: DIN 1302:1999-12 Allgemeine mathematische Zeichen und Begriffe. Standard.

DIN 1356 (1995). Deutsches Institut für Normung: DIN 1356-1:1995-02 Bauzeichnungen – Teil 1: Arten, Inhalte und Grundregeln der Darstellung. Standard.

DIN 5473 (1992). Deutsches Institut für Normung: DIN 5473:1992-07 Logik und Mengenlehre; Zeichen und Begriffe. Standard.

DIN VOB (1926). Deutsches Institut für Normung: VOB – Verdingungsordnung für Bauleistungen. Standard.

DIN VOB (2012). Deutsches Institut für Normung: VOB – Vergabe- und Vertragsordnung für Bauleistungen. Standard.

Dolenc, M., Katranuschkov, P., Gehre, A., Kurowski, K., & Turk, Z. (2007). The InteliGrid platform for virtual organisations interoperability. *ITcon*, 12, 459–477.

Dong, B., Lam, K. P., Huang, Y. C., & Dobbs, G. M. (2007). A comparative study of the IFC and gbXML informational infrastructures for data exchange in computational design support environments. In International Building Performance Simulation Association (IBPSA) (Ed.), *Proceedings of Building Simulation 2007* (pp. 1530–1537).

Durflinger, K., Reid, J. D., & Logan, K. M. (1998). Multi-model database management system engine for database having complex data models. US Patent 5713014.

E

Eastman, C., Teicholz, P., Sacks, R., & Liston, K. (2011). *BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors*. John Wiley & Sons, 2nd edition.

Eastman, C. M. (1999). *Building Product Models: Computer Environments, Supporting Design and Construction*. CRC Press, 1st edition.

Edlich, S., Friedland, A., Hampe, J., & Brauer, B. (2010). *NoSQL: Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken*. Hanser Fachbuchverlag.

Esfahani, N., Balder, R., & Scherer, R. J. (2012). Traffic infrastructure design and geo-information systems, a case of interoperability. In Gudnason & Scherer (2012), (pp. 597–602).

F

Firmenich, B. & Rank, E. (2007). Überblick zum Themenbereich Verteilte Produktmodelle. In *Vernetzt-kooperative Planungsprozesse im Konstruktiven Ingenieurbau* (pp. 121–132). Springer.

Fisher, N., Barlow, R., Garnett, N., Finch, E., & Newcombe, R. (1997). *Project Modelling in Construction: Seeing is Believing*. Thomas Telford Ltd, 1st edition.

Flemming, C. & Fuchs, S. (2012). Verbesserung der Prognose von Zahlungsplänen durch Multimodell-Filterung und 4-Stufen-Risikosimulation. In R. J. Scherer, G. Faschingbauer, & H. Pruvost (Eds.), *Bauinformatik – Baupraxis 2012: Industrieforschungsprojekte in der Bau-IT*, number 6 in Veranstaltungen des Instituts für Bauinformatik (pp. 85–97).

Fowler, M. (2010). *Domain-Specific Languages (Addison-Wesley Signature Series (Fowler))*. Addison-Wesley Professional, 1st edition.

Friese, P., Zambrowski, S., & Zimmermann, F., Eds. (2008). *Proceedings of the Second Workshop on MDSD Today 2008*, Berichte aus der Softwaretechnik. Shaker Verlag.

Fröbel, T., Firmenich, B., & Koch, C. (2009). Coupling Patterns In Civil Engineering Applications. In K. Gürlebeck & C. Könke (Eds.), *Proceedings of the 18th International Conference on the Applications of Computer Science and Mathematics in Architecture and Civil Engineering (IKM), 2009, Weimar, Germany* (pp. 1–15).

Fröbel, T., Firmenich, B., & Koch, C. (2011). Quality assessment of coupled civil engineering applications. *Advanced Engineering Informatics*, 25(4), 625–639.

Froese, T. (2003). Future directions for IFC-based interoperability. *ITcon*, 8, 231–246.

Froese, T. (2007). Information management in nD. In Aouad et al. (2006), chapter 13, (pp. 245–259).

Fu, C., Aouad, G., Lee, A., Mashall-Ponting, A., & Wu, S. (2006). IFC model viewer to support nD model application. *Automation in Construction*, 15(2), 178–185.

Fuchs, S., Kadolsky, M., & Scherer, R. J. (2011). Formal Description of a Generic Multi-Model. In *Proceedings of the 2011 IEEE 20th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, WETICE '11 (pp. 205–210). Washington, DC, USA: IEEE Computer Society.

Fuchs, S., Katranuschkov, P., & Scherer, R. J. (2010). A Framework for Multi-Model Collaboration and Visualisation. In K. Menzel & R. J. Scherer (Eds.), *eWork and eBusiness in Architecture, Engineering and Construction: ECPPM 2010* (pp. 115–120).: CRC Press / Balkema.

Fuchs-Kittowski, K. (2002). Wissens-Ko-Produktion: Verarbeitung, Verteilung und Entstehung von Informationen in kreativ-lernenden Organisationen. In C. Floyd, C. Fuchs, & W. Hofkirchner (Eds.), *Stufen zur Informationsgesellschaft. Festschrift zum 65. Geburtstag von Klaus Fuchs-Kittowski* (pp. 59–125). Frankfurt a. M.: Peter Lang-Verlag.

G

Gabler Wirtschaftslexikon (2012). Gabler Wirtschaftslexikon. http://wirtschaftslexikon.gabler.de/ Online Publikation mit Permanent-URL für Stichwörter.

GAEB-DA-XML 3.2 (2013). Gemeinsamer Ausschuss Elektronik im Bauwesen (GAEB) – Organisation des Austauschs von Informationen über die Durchführung von Baumaßnahmen: GAEB-Datenaustausch XML. Standard.

Gamma, E. & Beck, E. G. K. (2004). *Eclipse erweitern*. Open source library. Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (2004). *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software (Programmer's Choice)*. Addison-Wesley Verlag, 1st edition.

Garcia, M. (2008). Formalization of QVT-Relations: OCL-based Static Semantics and Alloy-based Validation. In Friese et al. (2008).

Garmany, J., Walker, J., & Clark, T. (2005). *Logical Database Design Principles*. Auerbach Publications Taylor & Francis Group.

Gehre, A., Katranuschkov, P., Stankovski, V., & Scherer, R. J. (2005). Towards semantic interoperability in virtual organisations. In Scherer (2005), (pp. 18–21).

Gemeinsamer Ausschuss Elektronik im Bauwesen (GAEB) (1990). *Regelungen für den Datenaustausch Leistungsverzeichnis*. Beuth Verlag GmbH, 2nd edition.

Gielingh, W. (1988). General AEC reference model (GARM) an aid for the integration of application specific product definition models. In P. Christiansson & H. Karlsson (Eds.), *CIB W078 + W074 Seminar on Conceptual Modelling of Buildings* (pp. 165–178).

Gosling, J., Joy, B., Steele, G., & Bracha, G. (2005). *Java™ Language Specification*. Addison Wesley, 3rd edition.

Goulet, J.-A., Kripakaran, P., & Smith, I. F. C. (2010). Multimodel Structural Performance Monitoring. *Journal of Structural Engineering*, 136(10), 1309–1318.

Graeff, B. (2003). Abfragesprachen für rasterbasierte Geo-Informationssysteme. *Geomatik Schweiz*, 5, 224–228.

Gu, N. & London, K. (2010). Understanding and facilitating BIM adoption in the AEC industry. *Automation in construction*, 19(8), 988–999.

Gudnason, G. & Scherer, R. J., Eds. (2012). *eWork and eBusiness in Architecture, Engineering and Construction: ECPPM 2012*. Taylor & Francis.

H

Hanff, J. (2003). *Abhängigkeiten zwischen Objekten in ingenieurwissenschaftlichen Anwendungen*. PhD thesis, Technische Universität Berlin.

Hansen, K. & Zenobia, K. (2011). *Civil Engineer's Handbook of Professional Practice*. John Wiley & Sons.

Hartmann, T., Gao, J., & Fischer, M. (2008). Areas of application for 3D and 4D models on construction projects. *Journal of Construction Engineering and management*, 134(10), 776–785.

Hartmann, U. (2009). Ein deklarativer Ansatz zur domänen-übergreifenden Modellanalyse – Schnittstellen für den Einsatz domänen-spezifischer Sprachen. In P. von Both & V. Koch (Eds.), *Forum Bauinformatik 2009: 23. bis 25. September 2009, Universität Karlsruhe (TH)* 179–198: KIT Scientific Publishing.

Hauschild, T. (2004). *Computer Supported Cooperative Work-Applikationen in der Bauwerksplanung auf Basis einer integrierten Bauwerksmodellverwaltung.* PhD thesis, Bauhaus Universität Weimar.

Hay, D. C. (2006). *Data Model Patterns: A Metadata Map.* The Morgan Kaufmann series in data management systems. Elsevier Morgan Kaufmann.

Heath, T. & Bizer, C. (2011). *Linked Data (Synthesis Lectures on the Semantic Web: Theory and Technology).* Morgan & Claypool Publishers, 1st edition.

Hegemann, F., Lehner, K., & König, M. (2012). IFC-based product modeling for tunnel boring machines. In Gudnason & Scherer (2012), (pp. 289–296).

Heidenreich, F., Johannes, J., Karol, S., Seifert, M., & Wende, C. (2009). Derivation and Refinement of Textual Syntax for Models. In *Proceedings of the 5th European Conference on Model Driven Architecture – Foundations and Applications*, ECMDA-FA '09 (pp. 114–129). Berlin, Heidelberg: Springer-Verlag.

Henckels, D. (2005). *Fehldatenmodellierung – Ein intuitives Informationsaustauschmodell für den rechnergestützten Bauwerkslebenszyklus.* PhD thesis, Universität Karlsruhe (Technische Hochschule).

Henning, P. A., Hoffmann, D. W., & Vogelsang, H. (2007). Grundlagen der Programmiersprachen. In P. A. Henning & H. Vogelsang (Eds.), *Handbuch Programmiersprachen* chapter 1, (pp. 9–57). Carl Hanser Verlag.

Hessellund, A. (2009). *Domain-specific multimodeling.* PhD thesis, IT University of Copenhagen.

Hilbert, F. & Scherer, R. J. (2012). Context-specific Multi-model-template Retrieval. In *Proceedings of the Working Conference on Virtual Enterprises, PRO-VE 2012.*

Hilfert, T. & Hegemann, F. (2012). Dynamische Speicherung von Bodendaten: Implementierung eines hybriden Bodendatenmodells. In F. Hegemann, C. Kropp, T. Rahm, & K. Szczesny (Eds.), *Forum Bauinformatik 2012: Ruhr-Universität Bochum 26.–28. September 2012*, number 4 in Informationstechnologie (pp. 63–70).: Europäischer Universitätsverlag, Bochumer Universitätsverlag.

Howard, R. & Björk, B. C. (2008). Building information modelling–Experts' views on standardisation and industry deployment. *Advanced Engineering Informatics*, 22(2), 271–280.

Huang, L., Häubi, F., Breit, M., & Borrmann, A. (2012). BIM-based Information Exchange for Functional AEC Design and Ordering Processes. In V. Telichenko, A. Volkov, & I. Bilchuk (Eds.), *Proceedings of the 14th International Conference on Computing in Civil and Building Engineering (14th ICCCBE), Moscow 27–29 June, Moscow State University of Civil Engineering (National Research University)*: Publishing House ASV.

Huhnt, W. & Richter, S. (2010). Visualization of construction process models. In Tizani (2010).

I

Ibrahim, M., Krawczyk, R., & Schipporeit, G. (2004). Two approaches to BIM: a comparative study. In B. Rüdiger, B. Tournay, & H. Orback (Eds.), *Conference Proceedings – Architecture in the Network Society: 22nd eCAADe Conference Proceedings*, volume 22 (pp. 610–616).: Copenhagen, Denmark: Royal Danish Academy of Fine Arts.

Irizarry, J. & Karan, E. P. (2012). Optimizing location of tower cranes on construction sites through GIS and BIM integration. *ITcon*, 17, 351–366.

ISO 10303-1 (1994). International Organization for Standardization: Industrial automation systems and integration – Product data representation and exchange – Part 1: Overview and fundamental principles. Standard.

ISO 10303-11 (2004). International Organization for Standardization: Industrial automation systems and integration – Product data representation and exchange – Part 11: Description methods: The EXPRESS language reference manual. Standard.

ISO 10303-14 (2005). International Organization for Standardization: Industrial automation systems and integration – Product data representation and exchange – Part 14: Description methods: The EXPRESS-X language reference manual. Standard.

ISO 10303-22 (1998). International Organization for Standardization: Industrial automation systems and integration – Product data representation and exchange – Part 22: Implementation methods: Standard data access interface. Standard.

ISO 10303-28:2007 (2007). International Organization for Standardization: Industrial automation systems and integration – Product data representation and exchange – Part 28: Implementation methods: XML representations of EXPRESS schemas and data, using XML schemas. Standard.

ISO 12006-3 (2007). International Organization for Standardization: ISO 12006-3:2007 Building construction – Organization of information about construction works – Part 3: Framework for object-oriented information. Standard.

ISO 14977 (1996). International Organization for Standardization: ISO/IEC 14977:1996(E) Information technology – Syntactic metalanguage – Extended BNF. Standard.

ISO 16739 (2005). International Organization for Standardization: ISO/PAS 16739:2005 Industry Foundation Classes, Release 2x, Platform Specification (IFC2x Platform). Standard.

ISO 19502 (2005). Object Management Group: ISO/IEC 19502 Meta Object Facility (MOF) Specification 1.4.1. Standard.

ISO 8601:2004(E) (2004). International Organization for Standardization: Industrial automation systems and integration – Data elements and interchange formats – Information interchange – Representation of dates and times. Standard.

ISO 9075-1 (2008). International Committee for Information Technology Standards: INCITS/ISO/IEC 9075-1-2008 SQL/Framework, Part 1. Standard.

ISO 9834-8 (2004). ITU Telecommunication Standardization Sector: ISO/IEC 9834-8, ITU-T Recommendation X.667 Information technology – Open Systems Interconnection – Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 object identifier components. Standard.

ITU-T X.1252 (2010). ITU Telecommunication Standardization Sector: Cyberspace security – Identity management Baseline identity management terms and definitions. Standard.

J

Jardim-Gonçalves, R. & Steiger-Garção, A. (2002). Implicit multilevel modeling in flexible business environments. *Commun. ACM*, 45(10), 53–57.

Jeong, Y. (2008). *Mediating Semantics for Multidisciplinary Collaborative Design*. PhD thesis, University of California, Berkley.

Jongeling, R., Emborg, M., & Olofsson, T. (2005). nD modelling in the development of cast in place concrete structures. *ITcon*, 10, 27–41. Special Issue From 3D to nD modelling.

Juli, R. & Scherer, R. J. (2002). iCSS – Ein integriertes Client-Server-System für das virtuelle Planungsteam. In *VDI-Berichte*, number 1668. VDI Verlag, Düsseldorf.

Junge, R., Koethe, M., Schulz, K., Zarli, A., & Bakkeren, W. (1997). The VEGA Platform–IT for the virtual enterprise. In *CAAD futures*, volume 97 (pp. 591–616).

K

Kamardeen, I. (2010). 8D BIM modelling tool for accident prevention through design. In *Proceedings of the 26th Annual ARCOM Conference, Leeds, September, Association of Researchers in Construction Management, Reading*, volume 1 (pp. 281–289).

Karimi, H. A. & Akinci, B. (2010). *CAD and GIS Integration*. CRC Press, illustrated edition.

Karlshøj, J. (2007). Data visualization – A Danish example. In Aouad et al. (2006), chapter 14, (pp. 260–275).

Katranuschkov, P. (2000). *A Mapping Language for Concurrent Engineering Processes*. PhD thesis, Technische Universität Dresden, Fakultät Bauingenieurwesen.

Katranuschkov, P., Gehre, A., & Scherer, R. J. (2003). An ontology framework to access IFC model data. *ITcon*, 8, 413–437. Special Issue eWork and eBusiness.

Katranuschkov, P., Guruz, R., Liebich, T., & Bort, B. (2011). Requirements And Gap Analysis For Bim Extension To An Energy Efficient Bim Framework. In *ICT for a Low Carbon Economy – EEBuilding Data Models – Proceedings of the 2nd Workshop organised by the EEB Data Models Community CIB Conference W078-W012, 26-28 October 2011 Sophia Antipolis – France*.

Katranuschkov, P. & Scherer, R. J. (1996). Schema mapping and object matching: a STEP-based approach to engineering data management in open integration environments. In *Proceedings of the CIB W78-96 Workshop Construction on the Information Highway*.

Katranuschkov, P., Scherer, R. J., & Turk, Z. (2001). Intelligent services and tools for concurrent engineering – An approach towards the next generation of collaboration platforms. *ITcon*, 6, 111–128. Special Issue Information and Communication Technology Advances in the European Construction Industry.

Katranuschkov, P., Weise, M., Windisch, R., Fuchs, S., & Scherer, R. (2010). BIM-based generation of multi-model views. In Thabet (2010).

Khemlani, L. (2004). The IFC Building Model: A Look Under the Hood. `http://www.aecbytes.com/feature/2004/IFCmodel.html`.

Kim, H., Chun, S., Kim, J., & Kim, Y. (2011). Developing a web-based 5d system connecting cost, schedule and 3d model. In Bargstädt & Ailland (2011), (pp. 65–71).

Kiviniemi, A., Fischer, M., & Bazjanac, V. (2005a). Integration of multiple product models: Ifc model servers as a potential solution. In Scherer (2005).

Kiviniemi, A., Fischer, M., & Bazjanac, V. (2005b). Multi-model environment: links between objects in different building models. In Scherer (2005), (pp. 277–284).

Kleppe, A. (2008). *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*. Addison-Wesley Professional, 1st edition.

Koch, C. (2008). *Bauwerksmodellierung im kooperativen Planungsprozess: Mit der Objektorientierung zur Verarbeitungsorientierung*. PhD thesis, Bauhaus Universität Weimar.

König, M. & Marx, A. (2011). Aufbereitung von Multimodellen für Simulationsstudien zur Ausführungsplanung. In Scherer et al. (2011), (pp. 93–106).

Koonce, D., Huang, L., & Judd, R. (1998). EQL an Express Query Language. *Computers & Industrial Engineering*, 35(1-2), 271–274.

Kosovac, B., Froese, T., & Vanier, D. (2000). Integrating heterogeneous data representations in model-based AEC/FM systems. In G. Gudnason (Ed.), *Proceedings of CIT*, volume 2 (pp. 556–567).: Icelandic Building Research Institute.

Kugler, M., Kordi, B., Franz, V., & Samkari, K. (2011). Linking Product And Process Data In The Modeling Environment 'CiSmo'. In Bargstädt & Ailland (2011), (pp. 171–181).

L

Laakso, M. & Kiviniemi, A. (2012). The IFC Standard – A Review of History, Development and Standardization. *ITcon*, 17, 134–161.

Lee, A., Marshall-Ponting, A. J., Aouad, G., Wu, S., Koh, I., Fu, C., Cooper, R., Betts, M., Kagioglou, M., & Fischer, M. (2003). *Developing a vision of nD-enabled construction*. Construct IT Centre of Excellence, University of Salford.

Lee, A., Wu, S., & Aouad, G. (2007). nD modelling – Where next ? In Aouad et al. (2006), chapter 20, (pp. 341–349).

Lee, S. H. & Kim, B. G. (2011). IFC Extension for Road Structures and Digital Modeling. *Procedia Engineering*, 14, 1037–1042.

Leitbild Bau (2009). Leitbild Bau – Zur Zukunft des Planens und Bauens in Deutschland – eine gemeinsame Initiative der deutschen Bauwirtschaft. Online, Bundesministerium für Verkehr, Bau und Stadtentwicklung (BMVBS).

Leser, U. & Naumann, F. (2007). *Informationsintegration – Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen*. dpunkt.verlag.

Liebich, T., Schweer, C.-S., & Wernik, S. (2011). *Die Auswirkungen von Building Information Modeling (BIM) auf die Leistungsbilder und Vergütungsstruktur für Architekten und Ingenieure sowie auf die Vertragsgestaltung*. Technical report, buildingSMART International.

Liskov, B. & Wing, J. M. (1994). *Family Values: A Behavioral Notion of Subtyping*. Technical report, ACM Transactions on Programming Languages and Systems.

Lochmann, D. (2008). *Information – was es ist und was es nicht ist: Eine kritische Diskussion über Irrtümer und Wahrheiten*. Books on Demand.

M

Mak, R. (2009). *Writing Compilers and Interpreters: A Software Engineering Approach*. Wiley, 3rd edition.

Martin, J. (1983). *Managing the data-base environment*. Prentice-Hall.

Mazairac, W. & Beetz, J. (2012). Towards a Framework for a Domain Specific Open Query Language for Building Information Models. In A. Borrmann, P. Geyer, P. de Wilde, & Y. Rafiq (Eds.), *Proceedings of the 2012 eg-ice Workshop*.

Mazairac, W. & Beetz, J. (2013). BIMQL – An open query language for building information models. *Advanced Engineering Informatics*, 27(4), 444–456.

McAffer, J., VanderLei, P., & Archer, S. (2010). *OSGi and Equinox: Creating Highly Modular Java Systems*. Addison-Wesley Professional, 1st edition.

Menzel, K. (2007). Mehrdimensionale Informationsmodelle. In Scherer et al. (2007), (pp. 127–136).

Merks, E. (2008). The Unbearable Stupidity of Modeling. `http://mdsd08.techjava.de/stupidmodeling.pdf`.

Meyer zu Selhausen, H. (1996). Informationsflußmanagement in der Bank. In Informationsring Kreditwirtschaft e.V. (Ed.), *Neue Wege des Informationsmanagements in Banken – Chancen und Risiken von Kommunikationsnetzen – Internes Knowlege-Management. Neuntes Symposium des Informationsrings Kreditwirtschaft e.V.*: iK report, Zürich.

Mislin, M. (1988). *Geschichte der Baukonstruktion und Bautechnik: von der Antike bis zur Neuzeit: eine Einführung*. Werner.

Muntzinger, D. & Fuchs, S. (2010). Softwareanwendungen für die Integration und Auswertung kombinierbarer Fachmodelle. In Scherer & Schapke (2010), (pp. 117–128).

N

Newby, G. B. (1996). Metric Multidimensional Information Space. In D. Harman (Ed.), *Proceedings of TREC-5. Gaithersburg, MD: The National Institute of Science and Technology*.

Nour, M. (2008). A Graphical User Interface for handling IFC Partial Model Exchange. In *Proceedings of The 12. International Conference on Computing in Civil and Building Engineering (ICCCBE)*.

Nour, M. & Beucke, K. (2008). An open platform for processing IFC model versions. *Tsinghua Science & Technology*, 13, 126–131.

Nour, M. & Beucke, K. (2010). Object versioning as a basis for design change management within a BIM context. In Tizani (2010), (pp. 147–152).

Nussbaumer, M. (2007). *Entwicklung und Evolution dienstorientierter Anwendungen im Web-Engineering*. PhD thesis, Karlsruhe Institute of Technology.

O

OGC CityGML (2012). Open Geospatial Consortium: OGC City Geography Markup Language (CityGML) Encoding Standard 2.0.0. Standard.

Olbrich, M. (1998). *Relationenorientiertes Modellieren mit Objekten in der Bauinformatik*. PhD thesis, Universität Hannover.

OMG Express 1.0 (2010). Object Management Group: Reference Metamodel for the EXPRESS Information Modeling Language Specification 1.0. Standard.

OMG QVT (2011). Object Management Group: Meta Object Facility (MOF) 2.0 Query/View/Transformation, v1.1. Standard.

P

Parr, T. (2007). *The Definitive Antlr Reference: Building Domain-Specific Languages (Pragmatic Programmers)*. Pragmatic Bookshelf, 1st edition.

Parr, T. (2010). *Language Implementation Patterns: Create Your Own Domain-Specific and General Programming Languages (Pragmatic Programmers)*. Pragmatic Bookshelf, 1st edition.

Pernul, G. & Unland, R. (2003). *Datenbanken im Unternehmen: Analyse, Modellbildung und Einsatz*. Lehrbücher Wirtschaftsinformatik. Oldenbourg.

Pialorsi, P. & Russo, M. (2010). *Programming Microsoft LINQ in Microsoft .NET Framework 4*. Microsoft Press, 1st edition.

Praxl, O. (2010). *Das Berufsbild des Bauingenieurs: Was der Bauingenieur wissen und können muss*. GRIN Verlag.

R

Redmond, A., Hore, A., Alshawi, M., & West, R. (2012). Exploring how information exchanges can be enhanced through Cloud BIM. *Automation in Construction*, 24, 175–183.

RFC 5545 (2009). The Internet Engineering Task Force (IETF) – Internet Calendaring and Scheduling Core Object Specification (iCalendar). Standard.

Riedel, T. & Wender, K. (2007). Realisierung von Anfragefunktionalität für einen dynamischen Partialmodellverbund. In A. Merkel, R. Schütz, & T. Wießflecker (Eds.), *Tagungsband zum 19. Forum Bauinformatik* (pp. 303–310).

Rivard, H., Cheung, M. M. S., Melhem, H. G., Miresco, E. T., Amor, R., & Ribeiro, F. L., Eds. (2006). *Building on IT – Proceedings of the Joint International Conference on Computing and Decision Making in Civil and Building Engineering, Montreal, Canada, June 14–16, 2006. Joint publication by ICCCBE-XI, ICCC-ASCE 2006, DMUCE-5, CIB-W78, CIB-W102*.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorensen, W. (1993). *Objektorientiertes Modellieren und Entwerfen*. Carl Hanser Verlag, Prentice-Hall International Inc.

Rüppel, U., Meißner, U. F., & Lange, M. (2006). An agent-based cooperation platform for fire protection planning. In Rivard et al. (2006), (pp. 3246–3255).

S

Saake, G., Heuer, A., & Sattler, K.-U. (2005). *Datenbanken: Implementierungstechniken*. mitp-Verlag, 2nd edition.

Saake, G. & Sattler, K.-U. (2006). *Algorithmen und Datenstrukturen: Eine Einführung mit Java*. Dpunkt Verlag, 3., überarb. a. edition.

Schapke, S.-E. & Fuchs, S. (2011). Mefisto – Eine multimodellbasierte Plattform für das Bauprojektmanagement. In Scherer et al. (2011), (pp. 11–42).

Schapke, S.-E. & Hilbert, F. (2012a). Mefisto – Vokabulare zu Modelldomänen, Projektphasen, Detailstufen und Status. `http://www.mefisto-bau.de/resources/resources_models.html`. Datensätze im XML-Format.

Schapke, S.-E. & Hilbert, F. (2012b). Prozessgesteuertes Bauprojektmanagement mit Multimodellen – Vortrag zum 3. Mefisto-Kongress. `http://www.mefisto-bau.de/pdf/k3_01.pdf`.

Schapke, S.-E. & Scherer, R. J. (2008). Semantic annotation and sharing of text information in AEC/FM. In Zarli & Scherer (2008), (pp. 279–287).

Scherer, R. J., Ed. (1994a). *Product and Process Modelling in the Building Industry – Proceedings of the first European Conference, Dresden, 5–7 October 1994*. Balkema.

Scherer, R. J. (1994b). The EU Project COMBI – Objectives and Overview. In Scherer (1994a), (pp. 503–510).

Scherer, R. J. (1998). A Framework for the Concurrent Engineering Environment. In R. Amor (Ed.), *Product and Process Modelling in the Building Industry: Proceedings of ECPPM'98 – the Second European Conference on Product and Process Modelling in the Building Industry, BRE, UK. 19–21 October 1998* (pp. 449–458).: Building Research Establishment.

Scherer, R. J., Ed. (2005). *Proceedings of the 22nd Conference on Information Technology in Construction, July 19–21 2005 in Dresden, Germany*.

Scherer, R. J. & Katranuschkov, P. (2006). From Data to Model Consistency in Shared Engineering Environments. In I. F. Smith (Ed.), *Intelligent Computing in Engineering and Architecture*, volume 4200 of *Lecture Notes in Computer Science* (pp. 615–626). Springer Berlin Heidelberg.

Scherer, R. J., Katranuschkov, P., Kadolsky, M., & Laine, T. (2012). Ontology-based integration of building and related information models for integrated lifecycle energy management. In Gudnason & Scherer (2012), (pp. 951–956).

Scherer, R. J., Menzel, K., & Schach, R., Eds. (2007). *Mobile Computing im Bauwesen*. Expert-Verlag GmbH.

Scherer, R. J. & Reinhardt, J. (2007). Navigation in Informationsräumen. In Scherer et al. (2007), (pp. 137–152).

Scherer, R. J. & Schapke, S.-E., Eds. (2010). *MEFISTO: Management - Führung - Information - Simulation im Bauwesen – Tagungsband 1. Kongress*, number 2 in Veranstaltungen des Instituts für Bauinformatik.

Scherer, R. J. & Schapke, S.-E. (2011). A distributed multi-model-based Management Information System for simulation and decision-making on construction projects. *Advanced*

Engineering Informatics, 25(4), 582–599. Special Section: Advances and Challenges in Computing in Civil and Building Engineering.

Scherer, R. J., Tauscher, H., & Schapke, S.-E., Eds. (2011). *MEFISTO: Management - Führung - Information - Simulation im Bauwesen – Tagungsband 2. Kongress*, number 4 in Veranstaltungen des Instituts für Bauinformatik.

Schlitt, M. (2004). *Grundlagen und Methoden für Interpretation und Konstruktion von Informationssystemmodellen*. PhD thesis, Universität Bamberg.

Schmale, R., Maier, B., Komke, T., & Götzer, K. (2008). *Dokumenten-Management*. Dpunkt.Verlag, 4., vollst. überarb. u. erw. edition.

Shen, W., Hao, Q., Mak, H., Neelamkavil, J., Xie, H., Dickinson, J., Thomas, R., Pardasani, A., & Xue, H. (2010). Systems integration and collaboration in architecture, engineering, construction, and facilities management: A review. *Advanced Engineering Informatics*, 24(2), 196–207.

Smith, I. F. (2009). Multiple-Model Structural Identification. In C. Boller, F. Chang, & Y. Fujino (Eds.), *Encyclopedia of Structural Health Monitoring* (pp. 2199–2208). Chichester, UK: Wiley.

Smith, I. F. C. & Saitta, S. (2008). Improving Knowledge of Structural System Behavior through Multiple Models. *Journal of Structural Engineering*, 134(4), 553–561.

Sørensen, K. B., Christiansson, P., Svidt, K., Jacobsen, K., & Simoni, T. (2008). Towards Linking Virtual Models with Physical Objects in Construction using RFID-Review of Ontologies. In D. Rischmoller (Ed.), *Proceedings of the CIB W78 2008: 25th International Conference – Santiago, Chile, 15–17 July*.

Stachowiak, H. (1973). *Allgemeine Modelltheorie*. Wien, New York: Springer-Verlag.

Staudemeyer, J. (2007). *Groovy für Java-Entwickler*. O'Reilly.

Steinberg, D., Budinsky, F., Paternostro, M., & Merks, E. (2009). *EMF: Eclipse Modeling Framework*. Boston, MA: Addison-Wesley, 2nd edition.

Steinmann, F. (1997). *Modellbildung und computergestütztes Modellieren in frühen Phasen des architektonischen Entwurfs*. PhD thesis, Bauhaus Universität Weimar.

T

Tauscher, E. (2011). *Vom Bauwerksinformationsmodell zur Terminplanung – Ein Modell zur Generierung von Bauablaufplänen*. PhD thesis, Bauhaus-Universität Weimar.

Tauscher, H. & Scherer, R. J. (2012). Towards a configurable nD-viewer for building information models: A generic model for the description of visualization methods. In Gudnason & Scherer (2012), (pp. 271–277).

Thabet, W., Ed. (2010). *Proceedings of the CIB W78 2010: 27th International Conference – Cairo, Egypt, 16–18 November*.

Thomas, M. (2002). *Informatische Modellbildung – Modellieren von Modellen als ein zentrales Element der Informatik für den allgemeinbildenden Schulunterricht*. PhD thesis, Universität Potsdam.

Thomas, O. (2005). *Das Modellverständnis in der Wirtschaftsinformatik: Historie, Literaturanalyse und Begriffsexplikation*. Number 184 in Veröffentlichungen des Instituts für Wirtschaftsinformatik. Institut für Wirtschaftsinformatik (IWi) im Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI).

Tizani, W., Ed. (2010). *Computing in Civil and Building Engineering, Proceedings of the International Conference (ICCCBE 2010), Nottingham 30 June–2 July*. Nottingham University Press.

Tolman, F. P. & Poyet, P. (1994). The ATLAS models. In Scherer (1994a), (pp. 473–478).

Törmä, S., Oraskari, J., & Hoang, N. V. (2012). Distributed Transactional Building Information Management. In P. Pauwels (Ed.), *Workshop Report – First International Workshop on Linked Data in Architecture and Construction* (pp. 9–11). `http://multimedialab.elis.ugent.be/LDAC2012/`.

Tse, T. K., Wong, K. A., & Wong, K. F. (2005). The utilisation of building information models in nD modelling: A study of data interfacing and adoption barriers. *ITcon*, 10, 85–110. Special Issue From 3D to nD modelling.

Tulke, J. (2010). *Kollaborative Terminplanung auf Basis von Bauwerksinformationsmodellen*. PhD thesis, Bauhaus-Universität Weimar.

V

van Hoof, A. J. M. (2003). Rollen- und Aufgabenangepasste Informationsversorgung. `http://ais.informatik.uni-leipzig.de/download/2003s_s_cwm/`.

van Nederveen, S. & Tolman, F. (2001). Neutral Object Tree Support For Inter-Discipline Communication In Large-Scale Construction. *ITcon*, 6, 35–44.

Voß, S. & Gutenschwager, K. (2000). *Informationsmanagement (German Edition)*. Springer, 1st edition.

W

W3C DOM Level 3 (2004). World Wide Web Consortium: Document Object Model (DOM) Level 3 Core Specification Version 1.0. Standard.

W3C OWL (2004). World Wide Web Consortium: OWL Web Ontology Language Overview. Standard.

W3C RDF (2004). World Wide Web Consortium: Resource Description Framework (RDF): Concepts and Abstract Syntax. Standard.

W3C SPARQL (2008). World Wide Web Consortium: SPARQL Query Language for RDF. Standard.

W3C XML 1.0 (2008). World Wide Web Consortium: Extensible Markup Language (XML) 1.0 (Fifth Edition). Standard.

W3C XPath 1.0 (1999). World Wide Web Consortium: XML Path Language (XPath) Version 1.0. Standard.

W3C XQuery 1.0 (2010). World Wide Web Consortium: XQuery 1.0: An XML Query Language (Second Edition). Standard.

Wand, Y., Storey, V. C., & Weber, R. (1999). An ontological analysis of the relationship construct in conceptual modeling. *ACM Trans. Database Syst.*, 24(4), 494–528.

Weise, M. (2006a). *Ein Ansatz zur Abbildung von Änderungen in der modell-basierten Objektplanung*. PhD thesis, Technische Universität Dresden.

Weise, M. (2006b). *Formalisierung von Berechnungsvorschriften mit EXPRESS bzw. EXPRESS-X*. Technical report, Institut für Bauinformatik, TU Dresden. Bericht an die Arbeitsgruppe Modellbasierte Mengen bei der IAI.

Weise, M., Katranuschkov, P., & Scherer, R. J. (2000). A proposed extension of the IFC project model for structural systems. In R. Gonçalves, A. Steiger-Garçao, & R. J. Scherer (Eds.), *Product and Process Modelling in Building and Construction: Proceedings of the Third European Conference on Product and Process Modelling in the Building and Related Industries, Lisbon, Portugal, 25–27 September 2000* (pp. 229–238).: Balkema.

Weise, M., Katranuschkov, P., & Scherer, R. J. (2003). Generalised Model Subset Definition Schema. In R. Amor (Ed.), *Construction IT bridging the distance: Proceedings of the CIB W78's 20th International conference on information technology for construction, Waiheke Island, New Zealand, 23–25 April 2003*: University of Auckland.

Weise, M., Liebich, T., Tulke, J., & Bonsma, P. (2009). IFC support for model-based scheduling. In A. Dikbas & F. H. Giritli (Eds.), *Proceedings of the 26th International Conference on IT in Construction & 1st International Conference on Managing Construction for Tomorrow, Istanbul, Turkey, 1–3 October 2009*.

Wender, K. (2009). *Das virtuelle Bauwerk als Informationsumgebung für die Planung im Bestand zur Organisation und Strukturierung einer digitalen Bauwerksdokumentation*. PhD thesis, Bauhaus-Universität Weimar.

Wender, K. & Hübler, R. (2009). Enabling more human-oriented exploration of complex building information models. *Journal of Computing in Civil Engineering*, 23(2), 84–90.

Wetter, M. (2011). A View on Future Building System Modeling and Simulation. In J. L. M. Hensen & R. Lamberts (Eds.), *Building Performance Simulation for Design and Operation*. Routledge, UK.

Willenbacher, H. (2002). *Interaktive verknüpfungsbasierte Bauwerksmodellierung als Integrationsplattform für den Bauwerkslebenszyklus*. PhD thesis, Bauhaus-Universität Weimar.

Willenbacher, H. & Hübler, R. (2004). Intelligent Link-Management for the Support of Integration in Building Life Cycle. In *Proc. of International Conference on Computing in Civil and Building Engineering 2004*.

Windisch, R., Katranuschkov, P., & Scherer, R. J. (2012a). A Generic Filter Framework for Consistent Generation of BIM-based Model Views. In A. Borrmann, P. Geyer, P. de Wilde, & Y. Rafiq (Eds.), *Proceedings of the 2012 eg-ice Workshop*.

Windisch, R., Wülfing, A., & Scherer, R. J. (2012b). A generic filter concept for the generation of BIM-based domain-and system-oriented model views. In Gudnason & Scherer (2012), (pp. 311–319).

Witten, I. H., Frank, E., & Hall, M. A. (2011). *Data Mining: Practical Machine Learning Tools and Techniques*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 3rd edition.

Wix, J. (2007). Data classification. In Aouad et al. (2006), chapter 11, (pp. 209–225).

Y

Yabuki, N., Lebegue, E., Gual, J., Shitani, T., & Zhantao, L. (2006). International collaboration for developing the bridge product model IFC-Bridge. In Rivard et al. (2006), (pp. 1927–1936).

Ye, J. (2009). *Integrating data models, analysis and multidimensional visualizations : a unified construction project management arena*. PhD thesis, University of British Columbia.

Z

Zamanian, M. K. & Pittman, J. H. (1999). A software industry perspective on AEC information models for distributed collaboration. *Automation in Construction*, 8(3), 237–248.

Zarli, A. & Scherer, R. J., Eds. (2008). *eWork and eBusiness in Architecture, Engineering and Construction: ECPPM 2008*. CRC Press / Balkema.

Zobl, F. & Marschallinger, R. (2008). GeoBIM–Subsurface Geo Building Information Modelling. *Geoinformatics*, 11(8), 40–43.

Stichwortverzeichnis

In `Schreibmaschinenschrift` gedruckte Begriffe beziehen sich auf Konzepte in Modellierungs- oder Programmiersprachen. **Fett** gedruckte Seitenzahlen geben den Ort der Begriffsdefinition an.

Bisher erschienene Dissertationen, Habilitationen und Hefte des Instituts für Bauinformatik[1]

Markus Hauser	Eine kognitive Architektur für die wissensbasierte Unterstützung der frühen Phasen des Entwurfs von Tragwerken	logos Verlag Berlin, 06/1998
Martin Zsohar	Stochastische Größen der Resonanzfrequenzen und der Verstärkung seismischer Wellen im horizontal geschichteten zufälligem Medium	Shaker Verlag Aachen, 1998
Christian Steurer	Modellierung und Berechnung des Ermüdungsfortschritts mit stochastischen Differentialgleichungen	Selbstverlag, 1999
Peter Katranuschkov	A Mapping Language for Concurrent Engineering Processes	Heft 1, 2001
Karsten Menzel	Methodik zur nachhaltigen, rechnergestützten Ressourcenverwaltung im Bauwesen	Heft 2, 2003
Michael Eisfeld	Assistance in Conceptual Design of Concrete Structures by a Description Logic Planner	Heft 3, 2004
Matthias Weise	Ein Ansatz zur Abbildung von Änderungen in der modellbasierten Objektplanung	Heft 4, 2006
Jörg Bretschneider	Ein wellenbasiertes stochastisches Modell zur Vorhersage der Erdbebenlast	Heft 5, 2006
Martin Keller	Informationstechnisch unterstützte Kooperation bei Bauprojekten	Heft 6, 2007
Kamil Umut Gökçe	IT Supported Construction Project Management Methodology Based on Product Model and Quality Management	Heft 7, 2008
Gerald Faschingbauer	Simulationsbasierte Systemidentifikation im Rahmen der baubegleitenden geotechnischen Überwachung	Heft 8, 2011
Wael Sharmak	Dynamic Network Planning in Construction Projects using Configurable Reference Process Models	Heft 9, 2011
Amin Zahedi Khameneh	Wave-type based Real-Time Prediction of Strong Ground Motion	Heft 10, 2012

[1] Bis September 2003 Lehrstuhl für Computeranwendung im Bauwesen